Fatec São Paulo
CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Processamento de Dados

FACULDADE DE TECNOLOGIA DE SÃO PAULO

JEFFERSON MENDES SILVA

Chatterbots podem ser úteis?

São Paulo – São Paulo – Brasil

2012

Fatec São Paulo
CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Processamento de Dados

FACULDADE DE TECNOLOGIA DE SÃO PAULO

JEFFERSON MENDES SILVA

Chatterbots podem ser úteis?

Monografia submetida como exigência parcial
para a obtenção do Grau de Tecnólogo em
Processamento de Dados

Orientador: Prof. Dr. Maurício Amaral de Almeida

São Paulo – São Paulo – Brasil

2012

DEDICATÓRIA

Dedico este trabalho primeiramente a Deus, por permitir minha existência, das pessoas ao meu redor e ao nosso mundo existir.

Agradeço também à minha esposa, apoiando-me em todos os momentos com um amor incondicional, ternura e aconchego.

Aos meus dois filhos, com sua inocência e alegria me dando um motivo para evoluir e aprimorar-me, com o foco de prover-lhes um melhor futuro.

Agradeço aos meus pais, por permitirem minha existência e me oferecerem a educação, ética e seu amor incondicional, essenciais para meu sucesso.

Também aos meus irmãos e irmãs, sogra e sogro e minha cunhada, por direta ou indiretamente pela torcida e apoio nos diversos momentos de minha vida.

Aos meus amigos e colegas de faculdade, aos colegas e amigos do trabalho, demais parentes e outras pessoas do meu círculo social, em sua maioria torcendo e esperando a conclusão de meus estudos com sucesso e futuro desenvolvimento.

Também aos meus amigos e parentes falecidos, por estarem com certeza me observando e me apoiando de seu próprio modo, este sendo o mais misterioso e incerto, mas com o mesmo amor desempenhado por eles em vida.

Agradeço aos meus professores, de toda a vida, ensinando-me quase tudo que sei, fornecendo a cultura e os conhecimentos utilizados hoje em dia nas diversas fases e necessidades na minha vida.

Pode parecer inusitado, mas agradeço também as dificuldades e as adversidades, pois sem as mesmas não haveria superação, a necessidade, o desafio, e, sem estes, não seria possível a superação e a evolução.

AGRADECIMENTO

Agradeço ao meu professor orientador por ter me ensinado os preceitos da linguagem AIML e incentivar o interesse em inteligência artificial com suas magníficas aulas.

Agradeço aos demais professores do curso por ensinarem-me as demais matérias necessárias para a confecção deste trabalho, também por seu apoio e os demais “empurrõezinhos” para que me formasse e concluísse este trabalho.

Em especial ao professor Arima, por ter me ensinado a parte de monografias e a me direcionar ante este projeto final, e pela descoberta do mundo acadêmico.

Também à professora Elisabete, por ter criado um monstrinho em desenvolvimento web, de tanto estímulo durante suas aulas de desenvolvimento web, estas essenciais às necessidades ante a entrega deste trabalho.

RESUMO

Este trabalho consiste no desenvolvimento ou adaptação de um programa de chatterbot a ser utilizado e aprimorado por alunos pela Fatec São Paulo. A principal necessidade do desenvolvimento deste é apresentar a tecnologia e os preceitos da linguagem AIML, demonstrando como utilizá-la e desvendando sobre a sua real utilidade.

Como o foco deste programa será a utilização em aplicações web com multiusuários, será utilizado o PHP associado ao MySQL.

Já existem dois programas desenvolvidos para trabalhar-se com AIML, o Program-O e o Program-E, ambos com suas particularidades, prós e contras, dentre estas duas opções, foi escolhido o Program-O, por ser mais atual e estar mais conciso.

Sendo o AIML é uma linguagem baseada em XML, também será rapidamente abordado um pouco sobre a linguagem XML, para o entendimento e perfeita codificação da linguagem.

A parte de banco de dados também será abordada, em especial sobre o funcionamento do MySQL e um suprassumo de sua teoria, para poder programar o software desejado.

O resultado alcançado foi um sucesso parcial. Foi possível criar um chatterbot capaz de coisas além de o simples falar, mas o desenvolvimento deste consome muito tempo, e necessita de aprimoramentos contínuos. Constatou-se que um chatterbot pode sim ser muito útil e até viável comercialmente, desde que tomados os devidos cuidados com sua implantação e manutenção.

Este trabalho pode até ser utilizado como um manual em português sobre AIML e o Programa-O.

ABSTRACT

This work consists in developing or adapting chatterbot software to be used and enhanced by FATEC's Sao Paulo students. The main target of it is to introduce its technology and the principal concepts of the AIML style, showing how to use it and analyzing about your real utility.

The focus is to use a program in multi-user web applications, so the choice is to use the PHP language with MySQL Database.

There is already two programs to work with AIML using PHP and MySQL, they're Program-E and Program-O, with their pros and cons and its particularities, was chosen the Program-O cause it's the newer and more stable.

AIML is an XML based language style, so will be explained a little about XML, to ease the understanding and to perfect coding the language.

Database concepts will also be shown, in special about MySQL working and a little bit of database theory, required for implement the desired software.

The result achieved was a partial success. It was possible to make a chatterbot do more than only talking, but a chatterbot developing takes too much time and need a continuous updating. However, was seen that a chatterbot can really be very useful and even commercial viable, taking care of with his implementation and maintenance.

This work can even be used as an AIML and Program-O manual.

SUMÁRIO

INTRODUÇÃO

AIML (*Artificial Intelligence Markup Language)* é uma linguagem que desafia os paradigmas da ciência da computação. Enquanto pensa-se ser extremamente complexo para uma máquina conversar com um humano sobre assuntos diversos, o Dr. Richard S. Wallace demonstra que não é bem assim...

Seres humanos gastam a maioria do seu tempo com palavras banais. Mesmo estas sendo de assuntos diversos, abrangendo várias áreas de interesse, com níveis de cultura, tanto de assuntos como de interlocutores muito variados, resume-se que a quantidade de palavras utilizadas nestas conversações é pequena, havendo muita repetição.

Sempre quando em uma conversa convencional, formula-se uma pergunta com respostas óbvias, possuindo a maioria das respostas para estas perguntas possuem as respostas genéricas como: sim, não sei, não, talvez sim ou talvez não.

O mesmo acontece com assuntos diversos. Ao estabelecer uma conversa padrão sempre é citado sobre assuntos corriqueiros, como o clima, algum evento circulando nas mídias ou até fofocas sobre alguma pessoa.

Tirando proveito destas circunstâncias, Dr. Richard Wallace e seus companheiros da *ALICE (Artificial Linguistic Internet Computer Entity) software foundation* desenvolveram o AIML, uma linguagem baseada em XML (Extensible Markup Language) utilizada para o fim de estimular e facilitar a programação de *chatterbots*.

É necessário um entendimento prévio de XML para facilitar a programação em AIML, pois a mesma necessita que as cadeias XML estejam bem formatadas para o AIML funcionar corretamente.

Um *chatterbot,* ou *chatbot*, é um programa de computador que assume o lugar de um interlocutor, dialogando com pessoas comuns utilizando-se textos pré-programados. Em inglês *bot* é a abreviação de *robot*, significando robô enquanto *chatter* significa conversador.

Da mesma maneira dos humanos, o chatterbot assume uma personalidade, possuindo também suas próprias opiniões sobre um determinado assunto. São capazes de aprender coisas novas, como respostas para uma nova pergunta ou até mesmo sobre abordar um assunto de maneiras completamente diferentes.

Sempre ao se estabelecer um diálogo entre dois interlocutores, na maioria das vezes, esta funciona como uma comunicação simplex. Na comunicação simplex existe apenas um canal de comunicação, onde um interlocutor faz a função de emissor e outro de receptor, havendo a inversão desta sempre após o término de uma oração. Ou seja, mesmo dispondo de duas bocas e dois pares de ouvidos, geralmente um humano fala e outro ouve, e raramente os dois falam e ouvem exatamente ao mesmo tempo. Diz-se o ditado: “Quando um burro fala, o outro abaixa a orelha”. Este tipo de comunicação é a predominante em chats baseados em texto.

Um *chatterbot* funciona com o mesmo princípio, ele espera uma entrada do usuário, processa a mesma e retorna um resultado. Neste caso não há a possibilidade de o usuário interromper a comunicação.

Os estudos do Dr. Richard Wallace baseiam-se principalmente na criação do chatterbot *ALICE*, utilizando-se da língua inglesa, sendo seu maior foco servir de psicóloga.

A *ALICE* é o maior chatterbot baseado em texto já criado, pois conta com mais de 40.000 categorias. Uma categoria em *AIML* define-se em um conjunto de uma pergunta e uma ou mais respostas pré-programadas, podendo a mesma ser organizadas em assuntos diferentes ou dependerem da resposta anterior do chatterbot ou do usuário.

```
<aiml>
<category>*</category>
<template>Você digitou <star/></template>
</aiml>
```

Um exemplo simples de AIML. O que for digitado pelo usuário será repetido pelo bot.

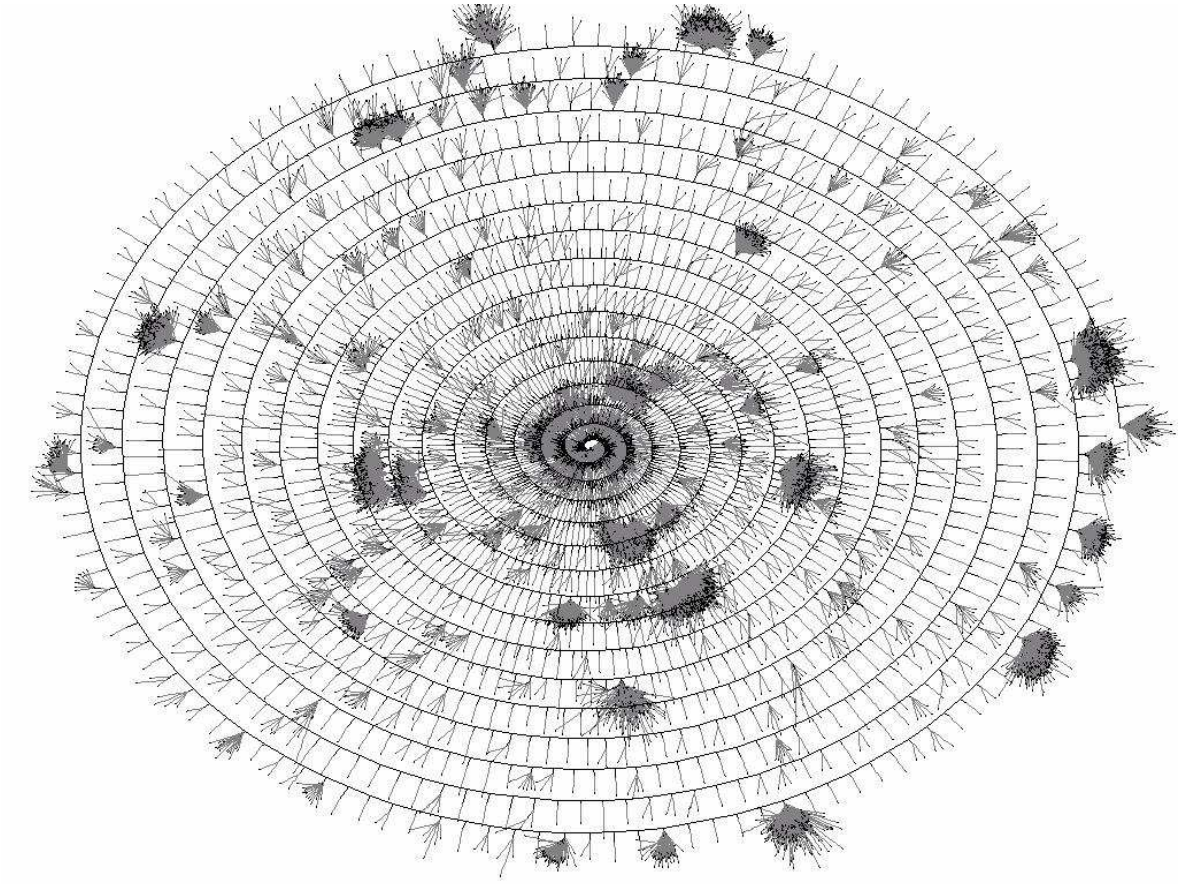

Figura 1 – O cérebro de ALICE. Cada linha é uma pergunta associada a uma resposta ou conjunto de respostas. Aqui estão carregadas 24,637 categorias das quase 40.000 programadas. A linha da espiral representa a raiz, e as bifurcações os tópicos e thats.

Assim como humanos, chatterbots também possuem personalidade, desejo comum em qualquer meta-humano. Existem técnicas para simular esta personalidade e maneiras de expressá-las. Também será exposto sobre isso no momento oportuno.

Há diferenças entre um *chatterbot* e um mecanismo de perguntas e respostas padrão, como o "*Ask Jeeves"*, por exemplo. No primeiro estabelece-se um assunto e conversa-se sobre ele até que um novo assunto seja estabelecido ou cessar a conversação, enquanto no *Ask* responde-se apenas uma pergunta por vez, sem associação entre uma resposta e outra anteriormente dita.

Chatterbots podem ser utilizados de diversas maneiras, e os teóricos de Inteligência Artificial definem em súmula que uma das funções mais importantes de um meta-humano seria a capacidade de o mesmo se comunicar de maneira independente.

Qualquer máquina ou programa necessita de uma programação anterior, mas a inteligência artificial é definida por vir a encontrar soluções para necessidades sem resposta previamente conhecidas. Numa conversa entre um humano e um *chatterbot*, nunca se sabe ao certo o rumo desta conversação.

Como se percebe, a criação de um *chatterbot* não é exclusivamente tecnológica, mas também teórica e linguística. Se durante uma conversação há inúmeras repetições de palavras, como estabelecer um modelo para a mesma?

Este mesmo modelo linguístico já existe e foi estudado pelo professor George Kingsley Zipf, que é conhecido como Lei de Zipf.

Cita-se a tradução de um trecho falando sobre esta lei: "*Pegue todas as palavras em um bloco de texto, como o exemplar de hoje do The New York Times, e conte quantas vezes uma palavra aparece. Se o histograma resultante for classificado por maior ocorrência de cada palavra e assim por diante ("um", "uma", "o", "para", "de", "e") então a forma da curva é uma "curva Zipf" para este texto. Se esta curva fosse desenhada em uma forma logarítmica, ela pareceria uma linha estreita com curvatura de -1".* (WALLACE, 2003).

Utilizando-se desta teoria, possibilita-se muito bem reduzir toda a infinidade de frases e assuntos a serem discutidos em um número muito menor de frases pré-programadas. Mesmo assim, esta é uma tarefa extremamente árdua e cansativa, parecendo até impossível, mas prova-se por este caminho a existência de uma opção viável e útil computacionalmente.

Uma coisa é utilizar esta tecnologia de forma acadêmica ou para fins de entretenimento, mas será possível a utilização destes para algo realmente útil? Será possível programar um *chatterbot* para que este se torne uma secretária, um vendedor, ou até um garoto propaganda valendo a pena o investimento?

É nisto que consiste este trabalho, será discutida entre as diversas possibilidades a criação de um *chatterbot* realmente útil em algo, realizando algum trabalho para os humanos, começando das coisas mais simples até as mais complexas.

Para tanto, este deverá ser acessível a uma grande quantidade de pessoas, sendo possível a distinção de cada usuário utilizador deste *chatterbot* tão como a memorização de seus dados e preferências.

Será utilizada a linguagem de programação PHP (Personal Home Page ou Hypertext Preprocessor), aliada ao Servidor Gerenciador de Banco de Dados MySQL. Como a linguagem PHP executa em ambiente servidor, sendo transparente aos browsers existentes no mercado, ele poderá ser utilizado por qualquer usuário pela www (*World Wide Web*), desde que este possua os privilégios necessários para utilizar o site.

Como serão utilizados dados preferenciais de usuários, utilizam-se configurações de autenticação e de segurança de usuários, assim como definidos por WELLING (2005) e por CONVERSE (2004). Será analisado neste trabalho como utilizá-lo em conjunto com o AIML.

Como todo o conteúdo da linguagem AIML e os dados de usuários serão armazenados em um banco de dados, conhecimento prévio de um SGBD e modelagem de banco de dados serão necessários.

Já existem programas pré-fabricados para exercer a função de chatterbot em PHP conjuntamente com MySQL, tão como em diversas outras linguagens, mas será que os mesmos suprirão estas necessidades?

Citou-se no início do texto sobre inteligência artificial, mas o que é inteligência artificial? O que a inteligência artificial diz sobre o processamento da linguagem necessária para a programação de um *chatterbot*?

Existe também um enorme vocabulário abreviado utilizado durante as conversações em chats convencionais, que deverão ser conhecidas. Como o AIML lidará com estes fatores e estas diferenças?

Quanto ao *bot* em si, este fará o trabalho de simular um humano em uma conversação normal. Para tanto, o mesmo deverá possuir um nível de cultura e uma personalidade definida. PLANTEK demonstra alguns passos e dicas de como construir um humano virtual, não só na parte textual, mas também na parte de

criação de imagens e movimentos, além da automatização de processos e de funções. Quais seriam a personalidade e o nível de conhecimento necessário para o nosso *chatterbot*?

Este trabalho será com a análise da viabilidade do desenvolvimento de um *chatterbot* para esta finalidade, e a possibilidade de facilitar sua criação, utilização e adaptação por outros programadores.

OBJETIVOS GERAIS

Este trabalho tem por definição desenvolver um sistema de *chatterbot* para ser utilizado e aprimorado para e pela Fatec-SP (Faculdade de Tecnologia de São Paulo). Este *chatterbot* não será apenas um conversador, mas também um armazenador de informações com autenticação. A ideia futura é agregar serviços e funções a esse *chatterbot*, fazendo com que o mesmo realize algumas tarefas específicas, como armazenar e abrir sites favoritos, guardar fotos e vídeos e também reproduzir diversos tipos de mídia, por exemplo.

OBJETIVOS ESPECÍFICOS

Serão abordados os seguintes tópicos:

1. **Linguagem XML** – Será abordado neste tópico o que é o XML, e os elementos necessários para codificar XML, utilizando-se de cadeias bem-formatadas, essencial para a programação em AIML;

2. **Conceitos Básicos de Inteligência Artificial** – O que é a inteligência Artificial e como uma máquina pode conversar com humanos?

3. **Linguagem AIML –** O que é o chatterbot e como programá-lo?

4. **Conceitos básicos de Banco de Dados –** O que é necessário saber sobre Banco de Dados para o armazenamento de um chatterbot e dados relevantes para o programa a ser desenvolvido?

5. **Autenticação, segurança e gestão de SGBD com PHP –** Como fazer com o chatterbot ser útil para armazenar informações com segurança em ambiente web multiusuário utilizando-se do PHP?

6. **AIML utilizando o PHP –** Os principais passos para o AIML executar rotinas PHP. Quais os cuidados e truques relativos ao programa a ser utilizado?

Além destes tópicos, também será analisado externamente a esta monografia os programas já existentes de AIML utilizando a linguagem PHP aliado ao MySQL,

sendo também realizadas adaptações e/ou complementos destes com um início da programação do nosso *chatterbot*.

METODOLOGIA

Será utilizada pesquisa bibliográfica para consulta dos temas envolvidos na criação de um chatterbot. Também será utilizado o desenvolvimento de programa, ou a adaptação de um pré-existente a fim de obter-se o *chatterbot* desejado, com a consequente apresentação do mesmo.

JUSTIFICATIVA

Chatterbot é uma ferramenta muito divulgada e utilizada em vários outros países, principalmente na Coréia do Sul para o ensino e treinamento da língua inglesa. Esta tecnologia é pouco usada no Brasil, tendo apenas uma única empresa especializada em seu desenvolvimento e poucos em execução.

Por oferecer um potencial enorme, poder ser utilizada em várias plataformas de maneira independente e até automatizar algumas tarefas, sendo uma tecnologia que merece divulgação e aprofundamento.

Algumas empresas, entre elas a Petrobrás, já utilizam um chatterbot como garoto propaganda, substituindo um garoto propaganda comum ou um atendente on-line.

DESENVOLVIMENTO

O AIML é uma linguagem baseada em XML. Mas o que é o XML e qual sua utilidade? O que é necessário saber de XML para desenvolver-se AIML?

LINGUAGEM XML

A linguagem XML ("*Extensible Markup Language*") é uma linguagem de marcação. Para entender-se o que é uma linguagem de marcação será usado à analogia de um supermercado.

Uma rede de supermercados, sendo o Grupo Pão de Mel um exemplo fictício, possui diversos nomes fantasia para seus mercados, como o Próprio Pão de Mel, o Noite% entre outros. Cada um destes mercados possuem diversas lojas, e cada loja possui um corredor que armazena uma determinada classe de produtos e cada corredor com seus respectivos tipos de produtos, subdivididos por marca e assim por diante.

Uma linguagem de marcação é uma linguagem hierárquica. Ou seja, esta pode possuir pais ou subordinados. O mercado Pão de Mel loja 106 em seu corredor de laticínios e afins possui um achocolatado da marca Toddy de 500 gramas. Em outra loja da mesma rede, pode haver o mesmo achocolatado, mas com datas de validade e preços diferentes.

Representando esta situação em uma lista hierárquica, usa-se esta topologia:

1. Grupo Pão de Mel
    a. Mercado Pão de Mel
        i. 106
            1. Laticínios
                a. Achocolatados
                    i. Toddy
                        1. 500g
                        2. R$ 4,45
                        3. Sabor tradicional

Para este exemplo, fica simples organizar este item, mas ao se inserir diversos itens para desconhecedores de supermercados, nesta topologia seria mais difícil de visualizar. Para um software, ter-se-ia o mesmo problema, pois tanto um humano como um software demorariam a procurar uma informação específica,.

Ter-se-ia então a solução de colocar-se um identificador do que esta sendo falado antes da informação, não havendo então a necessidade de ficar tentando adivinhar o sentido da informação ali escrita. Insere-se então, uma marcação nesta informação:

1. Empresa: Grupo Pão de Mel
    a. Afiliada: Mercado Pão de Mel
        i. Loja: 106
            1. Setor: Laticínios
                a. Tipo: Achocolatados
                    i. Marca: Toddy
                        1. Atributos: 500g, R$ 4,45.

Analisando desta maneira, com todos os dados inclusos, ela parecerá uma árvore, Onde há uma raiz, chamada de topo e suas subdivisões. Outro exemplo abaixo ilustra esta situação:

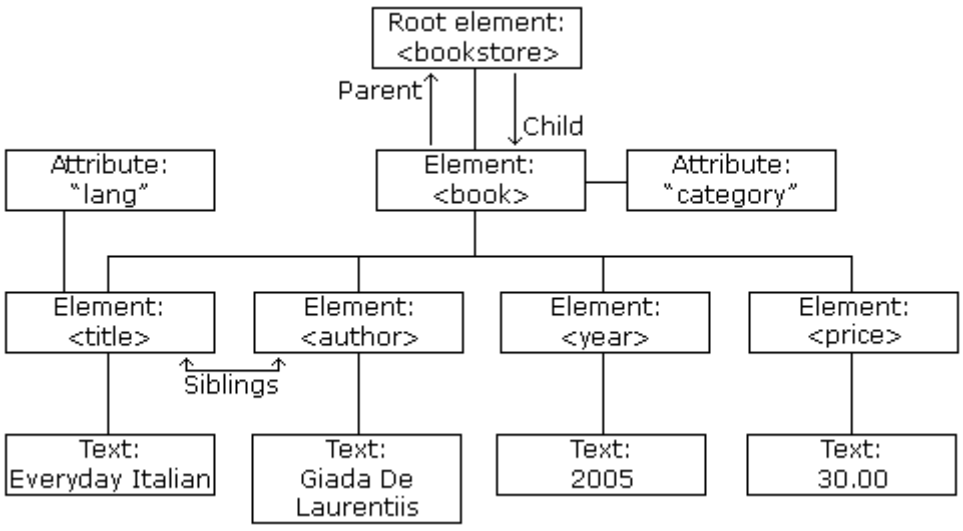

Figura 2 – Um exemplo da topologia XML – Fonte: PITTS-MOULTIS(1998)

O XML é uma linguagem de marcação feita para armazenar e transportar dados em forma de texto simples.

A *w3schools* diz que no futuro todos os bancos de dados, processadores de texto e planilhas deveriam poder interpretar e armazenar nativamente a linguagem XML, pois a mesma é clara, portável e facilitaria em muito a conversão da informação em diversos sistemas e softwares diferentes.

Como nota-se na figura 1, existem palavras entre os sinais < e >. Esta é denominada de *tag.* Em qualquer linguagem, a tag funciona como uma palavra-chave, ao qual se agrega uma função específica.

Uma tag em XML serve para organizar-se o conteúdo dos dados, evitando deixa-los soltos ao vento. Para tanto se poderia definir a raiz de nosso primeiro exemplo como <Empresas>. Neste caso, <Empresas> é denominado de elemento, como <Empresas> está no topo do documento, ele será o elemento raiz, de onde todos os demais se ramificam.

Esta mesma empresa poderia possuir um valor, denominado de atributo com o valor comércio varejista. Um atributo provê informações sobre o item armazenando valores referentes ao item em questão. Na Linguagem XML escrever-se-ia assim:

<Empresas tipo: ”Comércio Varejista”>Grupo Pão de Mel</Empresas>

Percebe-se a existência de uma tag de fechamento </Empresas> informando o término das informações referentes à empresa. Como a empresa possui diversas informações, a mesma possuirá diversas *tags*, organizando os dados de maneira clara e concisa.

Um elemento pode não possuir nenhuma informação relevante, pois apenas seu nome já bastaria como informação, não havendo nenhum filho ou ramificação e sem textos ou atributos. Pode-se incluir em um produto o elemento <Promoção/>, indicando que o produto, seu elemento pai, está em promoção.

Podem ser inseridos comentários em um texto XML. Da mesma maneira que em um documento HTML, eles começam com <!-- e terminam com -->.

A tabela abaixo é um exemplo de documento XML bem formatado. O mesmo modelo abaixo será necessário para utilizar-se da linguagem AIML:

```
<?XML version ”1.0”?>
<Empresas tipo: ”Comércio Varejista”>
    <Grupo Pão de Mel>
        <Lojas>
            <número =”106”>
                <Setores>
                    <Laticínios>
                        <Achocolatados>
                            <Toddy>
                                <Peso>500g</Peso>
                                <Sabor>Chocolate ao Leite</Sabor>
                                <Preço>R$ 4,45</Preço>
                                <Promoção/>
                            </Toddy>
                        </Achocolatados>
                    </Laticínios>
                    </Setores>
            </Número>
        </Lojas>
    </Grupo Pão de Mel>
</Empresas>
```

Para ser bem formatada, uma cadeia XML deve seguir algumas regras simples:

1. Toda *tag* deve possuir uma respectiva *tag* de fechamento, salvo caso esta seja uma *tag* simples como <promoção/>:
2. As *tags* filho devem ser fechadas antes de seus pais:
3. Não deve haver repetição de *tags* de mesmo nome no mesmo nível, evitando desentendimentos e conflito de informações:

Estas informações sobre XML são mais que suficientes para desenvolver um chatterbot. Existem muito mais informações a respeito de XML na web e em várias publicações, podendo as mesmas ser consultadas posteriormente.

Existem várias versões da linguagem XML, com suas maneiras de se programar e suas especificações. A primeira *tag* é a <?XML Version=”XXX”?>, indicando a versão em uso no documento atual.

O QUE É INTELIGÊNCIA ARTIFICIAL?

Este tópico visa expor poucos comentários sobre a inteligência artificial. Este se faz necessário para incluir o chatterbot na sua devida área de atuação na IA (Inteligência Artificial), expondo suas limitações frente algumas das teorias comentadas neste capítulo. Não será um assunto tão aprofundado para evitar-se perda de foco.

Introdução

A Inteligência Artificial é considerada como uma nova ciência, cujo principal objetivo, definido por vários teóricos e obras, é tornar um sistema ou um computador inteligente, semelhante aos humanos. Para tanto, é necessário definir o que é inteligência e por que os humanos são inteligentes.

A inteligência humana é definida de várias maneiras diferentes. Há alguns anos a inteligência era definida apenas pela capacidade de um ser humano resolver problemas lógicos, existindo até alguns testes como os de QI (quociente de inteligência), servindo para calcular esta inteligência.

Atualmente é considerado também o conceito de inteligências múltiplas, onde cada indivíduo pode possuir um elevado nível em algumas delas e não tão satisfatórios em outros.

Uma maioria dos grandes atletas, como Pelé, possuem uma inteligência cinética exuberante, por preverem movimentos tanto de seus corpos como de objetos a serem manipulados por eles ou outras pessoas e executá-los com perfeição, mas podem não possuir a inteligência lógica como um Einstein, sendo o inverso também verdadeiro.

Em súmula, define-se a inteligência humana como a capacidade de executarem-se tarefas e resolver problemas de maneira eficiente, e de também possuir o dom da cognição, ou seja, o ato de conhecer e acumular novos conhecimentos.

Esta cognição está intimamente relacionada à maioria das funções do ser humano, como a de pensar, visualizar, entre outras, estando fortemente atrelada à psicologia e comportamento humanos, possuindo até mesmo um ramo da psicologia que a estuda.

Nenhum humano nasce inteligente, esta inteligência é direcionada e acumulada ao longo da vida do indivíduo, de acordo com seus ensinamentos e experiências vividas.

A inteligência artificial visa reproduzir esta inteligência, ou pelo menos parte dela, em sistemas computacionais. Assim como as diversas áreas de inteligência humana, existem várias vertentes sobre a inteligência artificial, sendo este um campo extremamente amplo da ciência.

A enorme maioria dos sistemas computacionais tem como finalidade receber uma entrada de informação e devolvê-la da maneira adequada após um tipo específico de tratamento. O mesmo ocorre com nós humanos ao perceber-se algo em nosso exterior e executar alguma reação.

Uma reação simples pode ser descrita como acender-se uma lâmpada forte em nosso rosto e ter-se automaticamente a reação instintiva de proteger nossos olhos. Esta é mais uma questão de reflexos que de inteligência. Mas encontrar a fonte de tal iluminação, se analisar sobre a possibilidade e as consequências de tentar eliminá-la ou reduzi-la conforme a necessidade ou momento, ou decidir sobre outras atitudes diferenciadas, cabe a uma resolução mais cognitiva.

Esta é, talvez, a grande diferença entre sistemas convencionais e sistemas utilizadores de inteligência artificial, aproximar uma resolução de problemas ou assumir comportamentos mais próximos dos humanos em determinadas situações.

Um humano completamente artificial só será possível após solucionarem-se todos os itens inerentes ao comportamento e ações humanas em separado, e após unirem-se todos eles em um único sistema. Entretanto, os aspectos relacionados à mente humana não foram desvendados na sua totalidade, e talvez nem possível sejam.

Outra questão interessantíssima foi exposta por RUSSEL (2003) em sua obra, mesmo o computador podendo resolver um problema considerado possível apenas por humanos exigindo certo grau de raciocínio, como a totalidade dos testes de Turing, por exemplo, ele talvez não possa ser considerado inteligente, pois o mesmo pode não saber como chegou a esta conclusão e não possa simular as minúcias do pensamento humano.

O mesmo ocorre com o nosso robô tagarela, ele pode responder devidamente as perguntas inseridas por um usuário, mas com certeza absoluta, ele não entende absolutamente nada sobre as minúcias da pergunta do usuário e nem mesmo de sua própria resposta, ou seja, não possui razão ao responder, a não ser à razão imposta pelo seu mestre. Alternadamente, o mesmo pode processar expressões numéricas e guardar devidos tipos de informações com um desempenho impossível a nós humanos. Um simples exemplo é a criptografia de uma senha, um humano sem utilizar nenhum recurso tecnológico poderia nunca conseguir realizá-la apenas com sua mente durante toda sua existência, mas um computador pode fazê-la em frações de segundo.

Os ramos de estudo da IA

Para facilitar o estudo e o aprimoramento da IA, é interessante dividi-la em partes. Cada grupo de pesquisadores e autores dividiu-a em ramos do conhecimento, sendo não apropriável defini-la na sua totalidade com uma única frase ou estuda-la seguindo apenas uma única linha de pensamento, devido à sua amplitude e dificuldade.

RUSSEL (2003) definiu estas quatro formas de acordo com diversos autores que definiram o termo de IA de conforme seu ramo de pesquisa, organizando-as em uma tabela:

Sistemas que pensam como seres humanos	Sistemas que pensam racionalmente
"O novo e interessante esforço para fazer os computadores pensarem, [...] máquinas com mentes, no sentido total e literal." (Haugeland, 1985)	"O estudo das faculdades mentais pelo uso de modelos computacionais." (Charminak e McDermott, 1985)
"[Automatização de] atividades que associamos ao pensamento humano, atividades como a tomada de decisões, a resolução de problemas, o aprendizado..." (Bellman. 1978)	"O estudo das computações que tornam possível perceber, raciocinar e agir." (Winston, 1992)
Sistemas que atuam como seres humanos	**Sistemas que atuam racionalmente**
"A arte de criar máquinas que executam funções que exigem inteligência quando executadas por pessoas." (Kurzweill, 1990)	"A inteligência Computacional é o estudo do projeto de agentes inteligentes." (Poole et al., 1998)

Figura 3 – Definições de inteligência artificial agrupadas em quatro categorias distintas. (Fonte: RUSSEL (2003) pag. 5)

A maneira na qual está organizada esta tabela está citada abaixo:

"Em linhas gerais, as que estão na parte superior da tabela se relacionam a *processos de pensamento e raciocínio*, enquanto as definições da parte inferior se referem ao *comportamento.* As definições do lado esquerdo medem o sucesso e, termos de fidelidade do desempenho *humano*, enquanto as definições do lado direito medem o sucesso comparando-o a um conceito *ideal* de inteligência, que chamaremos de **racionalidade.** Um sistema é racional se "faz tudo certo", com os dados que tem". (RUSSEL, 2003)

SILVA define inteligência artificial como apenas duas das expostas por RUSSEL (2003): utilizadores e acumuladores de conhecimentos e os simuladores de comportamentos humanos, sendo respectivamente, os sistemas da parte superior e da parte inferior da figura 3, sendo sua função primordial a de simular-se conhecimento.

Uma análise mais profunda encaixa os chatterbots nos sistemas que pensam como seres humanos, pois atualmente os mesmos não obedecem às filosóficas "leis do pensamento", (não possui uma análise racional da saída do robô ao se responder uma pergunta).

O teste de Turing

Alan Turing foi um dos primeiros teóricos a se preocupar com a inteligência artificial comportamental. O mesmo, ao invés de questionar se as máquinas poderiam ser realmente inteligentes, propôs um teste de inteligência comportamental. Este tem como base tentar enganar um humano, em uma conversa on-line de cinco minutos, por pelo menos 30% do tempo, fazendo-se passar por uma pessoa.

O programa mais bem sucedido, em parte das vezes em uma versão apenas textual deste teste, foi a ALICE. Mas a mesma também apresentou dificuldades a ser examinada por julgadores treinados, não passando no teste, mas enganou alguns dos destreinados pelos cinco minutos, inclusive, segundo RUSSEL (2003), até um juiz na competição Loebner Prize.

O teste de Turing na íntegra consiste também de movimentos e imagens, portanto, está ainda longe de ser alcançado com os recursos e técnicas disponíveis hoje, além de não ser dada a devida importância a passar neste teste hoje em dia.

Entretanto, para criar-se um humano virtual, PLANTEK demonstrou como utilizar-se de diversos recursos de multimídia, indicando alguns softwares possíveis de serem utilizados para este fim. Uma das técnicas utilizadas foi de criar-se um modelo facial tridimensional, fazendo este parecer ainda mais com um ser humano.

O reconhecimento e processamento de voz também foram citados e seriam muito interessantes, prendendo ainda mais o usuário em seu local. O chatterbot ALICE possui também um sistema de texto-para-áudio e uma face que se movimenta ao se falar, assim como o robô Kirk, mas estes não simulam comportamentos faciais. Novas tecnologias, como o “Microsoft Kinect,” possibilitam a detecção de movimentos do usuário, e a mesma poderia muito ser também utilizada.

Em uma utopia para a época de sua publicação, PLANTEK imaginou um diálogo onde um robô poderia acender e regular a luminosidade das lâmpadas de seu quarto, e isto é perfeitamente possível hoje se utilizando o AIML integrando-o a algum dispositivo ou software de controle, como os utilizado hoje em dia em hotelaria.

O AIML versus processamento de linguagem natural

O significado de linguagem natural já está implícito em seu próprio nome. Ela diz respeito ao processo de comunicação de linguagem sem grandes esforços mentais, ou seja, uma comunicação comum do dia-a-dia.

Esta comunicação consiste no processo de formularem-se frases, sendo estas elaboradas quase de maneira automática na mente humana para expressar seus sentimentos. Ao expor uma frase, não há a preocupação com toda a análise sintática e/ou morfológica inclusa nesta, saindo esta automaticamente da maneira correta para a comunicação no meio do qual se encontra, não necessariamente sendo plenamente culta.

Para tanto, um processador de linguagem natural necessita de realizar as análises sintática e morfológica da frase a ser formulada. Estas duas etapas da formulação da frase são as mais simples de se fazer na linguagem natural, pelo motivo de as mesmas obedecerem a regras definidas na norma culta.

Além de determinar-se uma frase está correta do ponto de vista da língua portuguesa, há outro problema: uma frase pode estar correta nestes dois aspectos, mas pode conter falhas no aspecto semântico. A frase: “Eu irei comer uma torta holandesa”, pode referir-se tanto ao tipo da torta quanto ao seu local de fabricação, sendo as duas distintas e apresentando uma ambiguidade. Ou seja, a análise semântica mexe com o significado da frase. Um PLN (processador de linguagem natural) sem esta função pode gerar frases do tipo “Eu vou comer um café”.

A última, e mais difícil etapa a se solucionar é a pragmática. Além de verificar toda a parte gramatical e o sentido da frase, a mesma verifica a coerência desta. Uma frase pode satisfazer os três etapas, mas a frase: “A Alemanha é tricampeã da Copa América” é totalmente fora de nexo, pois um país europeu nunca pode disputar uma Copa América. Aliando-se este a um contexto, exponencia-se este problema.

O AIML não é um processador de linguagem natural, por não formular ele próprio as suas frases completamente. As frases do AIML já vêm pré-moldadas, sofrendo

poucas adaptações no decorrer de sua utilização, enquanto raramente um PLN repetirá uma frase já dita.

A vantagem de utilizar-se o AIML é de que a análise pragmática da frase é feita automaticamente pelo Botmaster durante sua programação, mas exige um esforço humano intenso para ensinar tudo o que o robô necessita falar. Entretanto, um PLN maduro pode responder uma infinidade de frases, mas o pode desviar o foco da conversação ou expor uma frase indesejável, como “Não gosto de meu mestre”. Fica provada a diferença de direções destas duas tecnologias: uma é focada na geração de frases, e outra em manter uma conversação.

A satisfação na resolução do teste de Turing só pode ser alcançada com uma união destas duas vertentes de resolução da linguagem, pois caso um chatterbot AIML não possua a devida resposta, o mesmo pode recorrer a um PLN para resolvê-la.

O que é o AIML em Inteligência Artificial?

O AIML trata-se de um agente. “Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores” RUSSEL (2003).

No caso de um agente inteligente, como um programa de computador, o ambiente pode ser especificado pelas tarefas a serem desempenhadas pelo programa, pois todo programa possui uma função principal. Os sensores de um programa são definidos basicamente pelo seu hardware, no caso de um chatterbot baseado em texto, um teclado ou outro dispositivo de entrada e um dispositivo de saída, como vídeo ou outros.

Existem diversos tipos de agentes inteligentes. O chatterbot pode ser enquadrado como um agente reativo, pois o mesmo executa uma operação de acordo com a percepção de seus sensores. Um agente reativo pode também conter sistemas de aprendizagem, como a tag <learn> no AIML.

O sistema estudado é enquadrado, na maneira mais apropriada, aos agentes baseados na utilidade, dentre os agentes citados por RUSSEL (2003). Evidencia-se

este fato pela satisfação e interesse do usuário ao conversar com um robô, pois, se o mesmo não estiver “feliz” com a conversa, o mesmo abandonará a conversa e o robô causará uma má impressão, afugentando os usuários ao invés de atraí-los.

A LINGUAGEM AIML

Este tópico estará organizado em três partes: a parte teórica, a parte técnica e a parte prática. A parte teórica cercará sobre o estudo e as partes teóricas de um chatterbot, enquanto a parte técnica será direcionada especificamente para os comandos a serem utilizados no desenvolvimento e manutenção de um chatterbot e como aperfeiçoar-se sua codificação. A parte prática deste trabalho será o conteúdo AIML do Programa.

Esta é a principal parte deste trabalho, onde será dada maior ênfase. Todos os assuntos já englobados anteriormente serão essenciais para o seu entendimento.

Parte teórica

Elisa foi o primeiro chatterbot criado com sucesso. Ele possuía um artifício de linguagem arcaico, utilizando processamento de texto plano e puro, sendo muito trabalhosa sua organização. Elisa tem como principal função de ser uma psicóloga, utilizando-se de perguntas, fazendo seu paciente buscar uma autorreflexão e interiorização para solução de seus problemas. O estilo de psicologia deste robô é similar à psicologia de Carl Rogers. Seu criador foi Joseph Weizenbaun em 1976, “através de um conjunto reduzido de expressões em aproximadamente 204 linhas, Eliza conseguia estimular uma investigação sobre o assunto abordado” (Silva).

A partir deste trabalho, o Dr. Richard Wallace e seus parceiros da ALICE Software Foundation desenvolveram uma nova técnica para escrever-se uma linguagem de um chatterbot, chamada de linguagem AIML. AIML nada mais é do que uma linguagem de marcação, baseada na SGML e XML para a escrita dos roteiros de conversação de um chatterbot. É simples e fácil de programar, em relação ao modelo da ELISA, podendo ser estruturada por pessoas não programadoras, com auxílio de softwares adicionais, como o *pandorabots*, por exemplo.

Uma grande vantagem desta linguagem é seu alto índice de reaproveitamento. O chatterbot ALICE, desenvolvido pela equipe de Wallace, é reutilizado na maioria dos *bots* existentes na linguagem inglesa para a criação e reaproveitamento em outros *bots* comerciais. Uma grande maioria dos bots programados hoje em dia tem como

base outro bot já mais amadurecido, modificado para atender às demandas do seu botmaster.

Como já dito na introdução do trabalho, a realização deste método de linguagem foi elaborado a partir dos estudos de Zipf, através da percepção de um padrão nos textos utilizados em uma conversação, dizendo que em grande parte das palavras utilizadas seria desnecessária para o entendimento da conversa, e também que várias palavras são repetidas diversas vezes em uma conversação, como afirmações positivas e negativas, artigos e entre outras coisas.

Um paralelo a este estudo pode ser associado às técnicas de leitura dinâmica, nesta técnica, o cérebro humano é treinado a ignorar algumas palavras na leitura de um texto, e também lê-lo não apenas em uma direção ou sequência, diminuindo em muito o tempo de leitura de um livro ou revista redigido de uma maneira simples e com palavras utilizadas no dia-a-dia.

O AIML utiliza desta mesma técnica. Ele procura em uma pergunta algumas palavras-chave inseridas pelo Botmaster, podendo ser atreladas a um assunto ou a resposta anterior, facilitando em muito o direcionamento da conversação. O Botmaster é o programador do chatterbot, ele é o responsável por ensinar o robô a responder perguntas.

É importantíssimo antes de exemplificarem-se comandos ou técnicas da criação de um chatterbot, falar-se sobre as teorias envolvidas, tornando muito mais simples na etapa de codificação obter um melhor aproveitamento da linguagem reduzindo seu tempo de codificação e quantidade de categorias.

Quanto aos Botmasters e à comunidade de desenvolvedores da ALICE, Dr Wallace define dois tipos principais: os reducionistas e os experimentalistas. Estes dois tipos acabaram sendo criados ante a revolução criada no campo de Inteligência Artificial a partir do AIML, e tornaram-se mais até vertentes de um modo de direcionar o futuro e aprimoramento da linguagem do que simplesmente perfis de programadores. Parecem-se até como partidos políticos.

Os reducionistas estão seguem a lógica aristotélica e matemática, e seu principal foco é de simplificar e manter a linguagem mais simples possível. Estes podem ser comparados a utilizadores de leitura dinâmica, quanto mais simples e menos palavras forem utilizadas, melhor.

Os experimentalistas estão, por sua vez, menos atrelados a teoria lógica da linguagem AIML, estando mais preocupados com a utilização desta linguagem como a base de um novo e revolucionário sistema, podendo ser atrelada a vários outros sistemas, integrando-a a novas aplicações e tecnologias, como reconhecimento de voz e até o de movimentos. SILVA cita em sua obra a existência de pesquisas tentado desenvolver o iAIML. Este seria o AIML associado a intenções, onde se armazena também as intenções do usuário ou do bot, auxiliando a escrever roteiros mais simples e refinar-se ainda mais a eficiência do chatterbot.

Para o sucesso de um chatterbot é evidente a necessidade de unirem-se as duas propostas acima citadas. Todo programa de computador, por mais banal ou sofisticado que seja, tem a necessidade de tornar-se eficiente e eficaz, reduzindo-se seu tamanho, número de instruções, complexidade de leitura e interpretação do código por outros programadores. Também é necessária, no ambiente integrado com a web, e de posse de inúmeros recursos tecnológicos existentes, a interoperabilidade entre sistemas e estes recursos. Reforçando, um bom Botmaster deve utilizar as duas técnicas alinhadas e bem dosadas.

A principal causa do funcionamento do chatterbot está na capacidade de oferecer as respostas-estímulo. Assim como no caso dos animais citados por SILVA no capítulo sobre inteligência artificial, os chatterbots também possuem respostas e executam ações de uma maneira quase instintiva, sendo estes agentes reativos. Como eles não possuem a capacidade natural de possuírem o dom da razão, respondem a estímulos externos assim como um cardume de peixe age em conjunto ao serem atacados por predadores.

Neste caso, como a conversação a partir de chatterbots funcionam numa comunicação half-duplex, são causados respostas-estímulo tanto do lado do programa como do lado do usuário.

Um chatterbot também possui a capacidade de aprender coisas novas. Ou seja, a partir de uma conversação com um usuário, o Botmaster pode identificar conversas ou frases mal respondidas e programar o chatterbot para respondê-la devidamente. Ou permitir que o próprio usuário o ensine.

O sucesso no desenvolvimento do AIML está na capacidade de realizar-se a execução de um roteiro de conversação. O Botmaster define um roteiro básico de uma possível conversa entre o usuário e o chatterbot, e a partir deste roteiro, vai se refinando as habilidades do chatterbot.

Em sua obra, SILVA dá dicas importantíssimas de como um iniciante pode definir um roteiro da linguagem AIML. Tal processo será refinado e demonstrado no momento oportuno. É muito fácil perder-se no desenvolvimento de um chatterbot, pois como um chatterbot necessita de várias categorias, cerca de 10000 no mínimo para ser realmente eficiente, a partir das 5000, segundo SILVA, um iniciante já começa a se perder, travando e paralisando o projeto devido à alta complexidade.

Um botmaster deve ser organizado, possuir disciplina e métodos para controlar-se e direcionar a programação do chatterbot. Ao se programar um chatterbot, deve-se ter em mente de que ele possuirá inúmeras categorias, e não somente as que se têm em mente para programar.

Parte técnica

Tipos de arquivos AIML

Existem algumas versões de arquivos AIML. Assim como arquivos HTML ou XML, este deverá conter juntamente na tag de abertura do Arquivo um atributo contendo a sua versão. A falta deste atributo não implicará no mau funcionamento do programa, mas poderá causar confusões quando de sua manutenção ou atualização por terceiros.

<aiml version="1.0.1">

Cada versão possui seu próprio conjunto de tags com seu respectivo funcionamento, podendo ou não ser alterada de versão em versão. Atualmente existem as versões 0.9 1.0 e 1.0.1 dos arquivos AIML. Neste trabalho será utilizada a versão 1.0.1, por esta estar mais concisa e atualizada. Além destes também deverá ser inserida uma tag de abertura de XML, pelo mesmo motivo da de AIML. Neste Caso será feito o uso da versão 1.0 codificação UTF-8.

Todos os nomes de arquivos AIML devem preceder da extensão "aiml", para este ser identificado facilmente dentre outros arquivos.

Categoria

Uma categoria pode ser definida como a mínima parte de uma conversação verbal entre dois interlocutores. Esta mínima parte consiste de um envio de informação, podendo ser uma pergunta ou afirmação de um dos interlocutores e uma consequente resposta ou reação através de uma nova informação pelo outro interlocutor. Como será trabalhado com texto, esta ação e reação serão escritas como texto. O site dictionary.com define o termo inglês *categories* como duas possibilidades para este caso de uso: uma delas é de uma divisão de classes, e outra é de um jogo de palavras, popularizado no Brasil com o nome de "stop", onde se escolhe um tema, como carros, novelas e outros, em seguida os participantes escrevem várias palavras a esta categoria relacionada. Como uma categoria precisa de no mínimo duas informações, ela deverá possuir, no mínimo, duas outras tags,

sendo estas a *pattern* e a *template*. Esta tag é aberta com <*category*> e fechada com </*category*>.

Padrões

Os padrões são as informações inseridas em uma categoria, sendo estas referidas ao que o usuário poderá inserir em uma afirmação ou resposta. Como uma informação expressa pelo usuário pode não ser exatamente como estarem programadas ou como previstas pelo *botmaster*, elas pelo menos seguem ou serão pensadas como um determinado padrão de entrada. Ao se programar um chatterbot, é importante pensar sobre as diferentes possibilidades de um usuário dizer uma mesma coisa. Um padrão pode servir como fonte de entrada para diversas frases pensadas por um humano. Para utilizar-se de um padrão, segue o exemplo abaixo:

<pattern>Ola</pattern>

Onde “Ola” é uma informação podendo ser expressa pelo usuário e <pattern> é a tag do padrão.

Um padrão também poderá fazer uso de coringas. Os coringas facilitam em muito a simplificação de padrões, fazendo com que várias frases diferentes se enquadrem no mesmo padrão. O mais utilizado deles é o “*”, onde ele substitui uma cadeia de caracteres quaisquer que pode ser inserida pelo usuário.

Neste caso <pattern>Bom dia, *</pattern> serve para as frases: Bom dia. Bom dia, tudo bem? Bom dia, como você está hoje? E infinitas outras.

O AIML sempre procura do específico para o genérico, ou seja, primeiro localiza a frase mais esmiuçada, como *<pattern>Bom dia meu nome é *</pattern>,* e depois *<pattern>bom dia *</pattern>,* mais abrangente.

Uma explicação mais aprofundada sobre coringas será abordada no tópico oportuno.

Modelos

Os modelos são o contrário dos padrões, servindo para indicar o modelo de frase respondida por um chatterbot após receber a entrada do usuário satisfazendo seu

respectivo padrão. Nota-se a necessidade de no mínimo uma união de uma categoria, um padrão e um modelo para a ocorrência de um diálogo.

Na falta de um destes itens, a cadeia AIML não estará bem formatada e não será possível analisar uma categoria e, em alguns casos, nenhuma de um arquivo.

Um modelo sempre virá logo após um padrão. Este modelo poderá conter apenas um texto simples ou ser incluído conjuntamente com vários outros recursos do AIML, estes explicados mais adiante.

Para utilizar-se um modelo, deverá ser utilizada a tag <template>:

```
<category>
<pattern>Olá</pattern>
<template>Olá, já nos conhecemos?</template>
</category>
```

Para um usuário não ficar solto ao vento após inserir um texto não especificado com um padrão distinto, deverá ser inserido um padrão genérico, podendo ser um padrão com apenas um *, respondendo algo como, “não entendi” para o usuário receber pelo menos uma resposta, ao invés de uma linha em branco ou um erro.

Uma técnica bastante utilizada por botmasters, também é utilizada por humanos com grande habilidade em conversação, é sugerir de uma forma sutil outro assunto a ser abordado. Ao invés de responder-se um grosseiro “não sei”, poderia se perguntar-se ao usuário se ele gosta de carros, por exemplo. Ao invés de deixar-se o usuário insatisfeito com o desempenho do chatterbot, pode-se direcionar a sua conversa de uma maneira prática e elegante.

Coringas

Foi falado anteriormente de utilizarem-se coringas como recurso para simplificar-se a utilização dos padrões, mas não se citou de como utilizar-se o conteúdo inserido pelo usuário para outros fins.

Toda inserção do usuário em uma conversação AIML é armazenado pelo programa interpretador em variáveis. No nosso caso, ele armazenará os valores de uma maneira possível de ser recuperada utilizando-se XML.

É possível então, após definir um padrão utilizando-se curingas, resolver o que fazer com este dado armazenado em uma variável, utilizando-se da tag <star/>.

Segue como exemplo a seguinte frase:

Usuário: Olá, meu nome é Jefferson.
Bot: Olá Jefferson, sobre o que deseja conversar?

Neste caso, codificara-se uma categoria da seguinte forma:

<category>
<pattern>Olá, meu nome é *</pattern>
<template>Olá <star/>, sobre o que deseja conversar?</template>
</category>

Na versão 0.9 da AIML, utilizava-se apenas a tag <star/>. Esta tag ficou obsoleta da versão 1.0 em diante, sendo substituída por <star index="X"/>. Esta modificação possibilitou ao chatterbot distinguir entre diversos coringas, possibilitando uma maior robustez e utilização. Neste caso pode-se utilizar a seguinte frase:

Usuário: Quanto é 3 mais 4.
Bot: É igual a 4 mais 3.

Para codificar esta sentença, digita-se:

<category>
<pattern>Quanto é * mais * </pattern>
<template> É igual a <star index="2"/> mais <star index="1"/></template>
</category>

Onde a numeração do índice é uma numeração inteira sequencial começando do 1, sendo que seu limite varia de cada interpretador. Caso seja inserida uma tag star sem definição de índice, nos moldes do AIML 0.9, este será automaticamente referenciado como índice um.

Existe outro coringa além do "*" que é o "_". A diferença entre os dois é que o "_" possui uma maior prioridade que o "*", sendo este processado primeiro. Era muitíssimo importante na versão 0.9, mas com o advento do índice, acabou até perdendo sua principal utilidade.

Recursão

O maior sucesso da linguagem AIML, conjuntamente com as tags de definição de contexto, é a tag <srai>. A principal função é de servir de referenciar-se a um padrão anterior, como uma função recursiva, diminuindo em muito a quantidade de padrões e modelos escritos economizando trabalho. Existem pelo menos sete significados ou funções para esta tag, separados desta maneira por Wallace (2003):

1. *Redução simbólica;*
2. *Divisão;*
3. *Correção gramatical;*
4. *Sinônimos;*
5. *Detecção de palavras chave;*
6. *Condicionais;*
7. *Qualquer combinação das seis anteriores.*

Redução simbólica

Consiste em pegar-se um padrão e simplificá-lo. Pode ser utilizada para pegar um padrão enorme e simplificar por um menor já escrito, utilizando o mesmo modelo para responder várias perguntas: “Vai chover?”, “Vai chover hoje?”, “Será que vai chover?” ou “Será que vai chover hoje?”.

Neste caso codifica-se desta maneira:

```
<category>
<pattern>Vai chover?</pattern>
</template>Não sei, não consigo olhar para o céu</template>
</category>
<category>
<pattern>vai chover *?</pattern>
<template><srai>vai chover?</srai></template>
</category>
<category>
<pattern>Sera que *</pattern>
<template><srai><star/></srai></template>
</category>
```

Nota-se que o conteúdo dentro da tag <srai> é a mesma do padrão a ser referenciado. Como foi colocado uma <star> na última codificação, este padrão servirá para quaisquer padrões que perguntem "será que...", localizando o conteúdo em qualquer outra pergunta.

Divisão

Outra função da tag srai é de dividir uma sentença em menores. Facilita-se em muito o trabalho caso o usuário apresentar mais de uma informação em uma mesma entrada. Um caso citado por Wallace (2003) é o caso do *Yes*. Pois o mesmo pode indicar uma resposta afirmativa para uma pergunta, e, logo após a pontuação, possibilitando inserir outra pergunta ou afirmação, ou até mesmo um novo assunto.

Para tanto, basta responder uma pergunta com um texto normalmente, adicionando-se uma tag <srai> aonde for conveniente para exibir-se a resposta de outro padrão.

Correção Gramatical e Sinônimos

Assim como na redução simbólica, a tag srai também é usada para correção gramatical. Assim, ao usuário usar gírias utilizadas comumente em chats, pode-se substituí-las para o termo correto, verificando a sua existência dentre várias outras frases sem precisar reescrevê-las. A mesma técnica serve também para correções gramaticais na frase, possibilitando também a aprendizagem pelo usuário da grafia correta em alguns casos. Alguns chatterbots são especialistas nisso.

Os sinônimos funcionam da mesma maneira, existem várias palavras diferentes podendo significar uma mesma coisa em um determinado contexto.

```
<category><pattern>Você está feliz hoje?</pattern>
<template>Fico sempre feliz quando converso com você!</template></category>
<category><pattern>vc *</pattern>
<template><srai>você <star/></srai></template></category>
<category><pattern>Bom dia * </pattern>
<template>Olá, tudo bem?</template></category>
<category><pattern> Boa tarde *</pattern>
<template><srai>bom dia <star/></srai></template></category>
```

Detecção de palavras chave

Outra grande utilização da tag srai é de utilizar as palavras-chave. Resume-se uma frase grande e complexa exprimindo uma ideia através de uma palavra simples. Exemplifica-se a criação de um algoritmo complexo para autenticação de usuário ou armazenamento de informações em um banco de dados através de um padrão AUTENTICARUSUARIO, onde seu modelo poderia ser um código qualquer misturado com um texto a ser retornado ao usuário. Assim, ao utilizar-se qualquer combinação de texto para referenciar-se ao desejo de autenticar-se no sistema, pode-se redirecionar o padrão para o padrão apropriado:

```
<category><pattern>MEMORIZARDADO: * = *</pattern>
<template><javascript>/*Esta tag será explicada mais adiante!*/</javascript>
</template></category>
<category><pattern>Lembre para mim: * = * </pattern>
<template><srai>MEMORIZARDADO: <star index="1"/> = <star index="2"/>
</template></category>
```

Variáveis em AIML

Antes de falar-se sobre condicionais, deve-se explicar como o AIML armazena valores de variáveis simples. Assim como qualquer linguagem de programação, variáveis possuem um nome podendo ser variáveis de ambiente, vindo pré-definidas no sistema operacional ou na própria linguagem, ou variáveis de programa, sendo estas definidas na própria codificação do programa.

No caso do AIML, o interpretador possui algumas variáveis de ambiente servindo para armazenar informações específicas, como o nome do Bot, o nome do usuário dentre outros aspectos da linguagem.

Outras variáveis podem ser definidas, sendo necessário apenas definir-se o nome deste. Também, como nas linguagens de programação, estas podem ser inicializadas. O valor da variável não inicializada depende da linguagem e do interpretador utilizado.

Para definir-se a variável ou atribuir-lhe um valor, basta utilizar-se da tag set. Para tanto se usa o nome da variável de atributo e expressar-se o seu valor:

```
<category><pattern>Meu nome é *</pattern>
<template>Prazer em conhecê-lo <set name="nome"><star/></set> </template></category>
```

Quanto a utilizar-se o valor armazenado, basta utilizar-se da tag <get>, juntamente com o atributo do nome da variável, assim

<category><pattern>Qual é meu nome?</pattern>

<template>Seu nome é <get name="nome"/></template></category>

Condicionais

Para usar condicionais simples com AIML, basta atrelar-se uma variável. Caso a variável possua um valor atribuído, pode testar-se este valor. Pode-se tanto atribuir um valor como falso ou verdadeiro como um valor literal, sendo este a ser testado:

<category><pattern>Eu gosto de carros</pattern>

<template>Que bom que vc gosta de carros<set name="gostacarro">gostosim</set> </template></category>

<category><pattern>Não quero ter carro</pattern>

<template><srai/>naoquerocarro <get name="gostacarro/></srai></template> </category>

<category><pattern>naoquerocarro gostosim</pattern>

<template>Por não quer ter um carro se você gosta de carros? </template> </category>

Frases anteriores

Outra técnica para utilizar-se o AIML de maneira mais eficiente é de saber-se o que foi dito antes. Para tanto, referimo-nos os modelos anteriores com a tag that. Assim como a tag star, este também pode possuir índices, dependendo do interpretador, para referenciar-se a diversas respostas anteriores.

Não se referencia aos padrões por estes serem mais difíceis de identificar, pois existe a recursão para facilitar-nos na parte do modelo.

Basta simplesmente usar-se então a tag <*that*> com o índice antes do padrão ou sem o mesmo, para evidenciar-se o último modelo que possivelmente foi dito.

Sempre quando em um modelo for inserida uma tag *that*, o padrão só será validado se o modelo anterior for também validado juntamente com o conteúdo de <that>, caso contrário não haverá a resposta desejada. Através deste artifício, possibilita-se produzir duas respostas diferentes para a mesma pergunta, simplesmente analisando-se a resposta anterior. Uma categoria sem a tag that, pode ser assumir o valor simbólico de * ou nulo em uma tag that.

<category><pattern>Eu sou *</pattern>

<template>Prazer <set name="person"><star/></set></template></category>

<category><that>Prazer <set name="person"><star/></set></that> <pattern> Eu sou *</pattern>

<template>Você já se apresentou a mim, <get name="person"/>! </template> </category>

Este exemplo poderia ser facilmente escrito somente com a utilização de recursão e as tags <set>/<get>. Outro exemplo interessantíssimo foi escrito por Silva (p. 35) sobre o desarmamento:

<category><pattern>faça uma pergunta</pattern>

<template>você é favorável ao desarmamento?</template></category>

<category><pattern>sim</pattern>

<that>você é favorável ao desarmamento?</that>

<template>eu também acho arma coisa perigosa</template></category>

Caso não haja um "sim" qualquer respondido pelo usuário, sem que a resposta anterior seja "você é favorável ao desarmamento?", a resposta do bot será qualquer outra programada, diferentemente desta.

Tópicos

Talvez uma das tags mais importantes seja as tags de tópicos. Como nas conversas normais, situa-se primeiramente sobre um assunto específico e conversa-se até a exaustão sobre este assunto, até mudar-se este assunto para outro e assim sucessivamente, até o final de uma conversação.

Os humanos tem a facilidade de identificar quando outra pessoa está em um assunto ou muda para outro, às vezes simplesmente pela feição do outro usuário ou simplesmente quando um dos interlocutores utiliza palavras-chave como "mudando de assunto...", por exemplo.

O mesmo pode acontecer com os bots, define-se um assunto inicial, sendo tratado como * ou assunto raiz, e daí podem programar-se categorias agrupadas em diferentes assuntos, como futebol, filmes, games, tecnologia entre outros.

Assim pode-se programar um chatterbot muito flexível e bastante interessante, pois com mais assuntos possibilita-se prender mais o usuário e mantê-lo interessado em continuar na conversação. Os humanos geralmente gostam de conversar com quem

fala sobre assuntos interessantes a si ou com pessoas mais cultas, conversando sobre vários assuntos diferentes, aumentando o seu leque de conhecimento.

Na parte prática, será falado em como concentrar assuntos diferentes em arquivos separados, facilitando em muito sua manutenção. Também será falado sobre roteiros, melhorando em muito sua implantação.

Para tanto, agrupa-se várias categorias em uma tag chamada de ***<topic>***.

Esta tag receberá um atributo *name* contendo o tópico a ser referenciado por este conjunto de categorias. Para utilizar-se este tópico, dever-se utilizar de uma tag topic, devendo englobar as categorias pertencentes a este tópico:

<set name="topic">nome do tópico</set> em um modelo.
<category><pattern>Vamos falar sobre carros</pattern>
<template>Ok, vamos falar sobre <set name="topic">carros</set></template> </category>
<topic name ="carros>
<category><pattern>De qual voce gosta</pattern>
<template>Gosto dos carros do Need for Speed</template></category>
<category><pattern>Qual seu * favorito</pattern>
<template>Gosto dos lamborghini</template></category>
</topic>

Para retornar ao tópico raiz, basta declarar o nome do tópico com "*" ou nulo, dependendo o interpretador utilizado. Caso o interpretador não localizar nenhuma ocorrência no tópico atual, ele automaticamente tentará localizar alguma no tópico raiz, dependendo da linguagem a ser utilizada.

Através deste advento, pode-se então separar os roteiros de maneira inteligente, executando-se a programação de um-a-um, simplificando e organizando ainda mais a programação. Ao falar-se na parte prática da linguagem AIML, este assunto será abordado com maior ênfase.

Segundo SILVA (p 40), utilizar-se de tópicos diminui em muito o tempo de procura do interpretador nas respectivas tags. Como o bot ALICE possui mais de 40.000 categorias, este deveria percorrer praticamente sua maioria para encontrar uma resposta, sem contar com as chamadas de recursão, para retornar esta resposta.

Utilizando-se tópicos, este número cairá drasticamente, representando uma melhoria muito favorável ao desempenho.

Verdadeiros condicionais

Em recursão foi falado sobre como condicionais podem ser utilizados através da tag <srai>. Mas, esta tag serve apenas para fazer-se condicionais simples. Outra forma de fazerem-se condicionais é utilizar-se do bloco ***<condition>***.

Este bloco funciona da mesma maneira que o bloco *case* das linguagens baseadas em *C*. Define-se primeiramente uma condição, uma resposta padrão e várias outras apontando para respostas diferentes. WALLACE (2003, p. 8) diz que a linguagem AIML deve ser mais simples possível, então, simplesmente com esta tag pode-se alcançar níveis muito altos de complexidade utilizando somente as tags já existentes em AIML, deixando para as linguagens hospedeiras a tarefa de executar outras operações, satisfazendo assim os experimentalistas.

Assim como em outras linguagens, define-se a variável a ser testada, define-se o valor padrão a ser retornado, caso não haja uma correspondência, e o valor a ser retornado de cada correspondência programada.

Segundo ANUPAMA (p 7-10) existem várias maneiras de tratar condicionais, mas todas elas recaem sobre a maneira como descrita acima, enquanto SILVA demonstra apenas uma, sendo utilizada com todos os tipos.

Para critério de desempate, WALLACE (2003) em seu manual de referência cita e demonstra exemplos dos três tipos de condicionais definidos: os de testes múltiplos, os de lista e o único.

Os de testes múltiplos funcionam como se fossem feitos várias afirmações, testando todas as informações e retornando todas com resultado verdadeiro. Fazendo-se um paralelo com outras linguagens convencionais, funciona como se houvessem vários condicionais “if” em sequencia, sendo que em cada ocorrência verdadeira executará o conteúdo do seu corpo.

Uma condição múltipla pode ser referenciada por várias tags <*condition*> contidas no mesmo modelo. Para tanto se utiliza a seguinte sintaxe:

<condition name="nome da variável" value="valor a ser testado"> RESULTADO SE VERDEDEIRO </condition>

Neste caso, pode-se retornar mais de uma resposta, possuindo pelo menos uma verdadeira.

Já nas de condições de lista múltipla e lista simples, serão testadas todas até a primeira ocorrência de verdadeiro. A de listas múltiplas testam várias variáveis e vários valores, parando seu processamento na primeira ocorrência de verdadeiro, enquanto o de lista única testa apenas uma variável e vários valores a serem atribuídos a ela. Neste caso, será retornada apenas uma mensagem de resposta.

```
<category><pattern>avaliaveiculo</pattern>
<template><condition name="veiculo">
<li value="bicicleta">Bicicleta é um veículo saudável e ecológico</li>
<li value="carro">Carro é um veículo muito útil</li>
<li value="metro">Metrô é um veículo muito lotado</li>
<li>Veículo interessante...</li>
</template></category>
```

O último item da lista aceitará qualquer valor, valendo como o valor padrão da condição. Este valor padrão pode ser incluso em qualquer tipo de condicional.

Transformações

Um recurso muito interessante são as transformações. É muito simples programar-se um chatterbot possuindo por referência um único gênero, número e grau, mas, e como é necessário satisfazer a todos?

Quando programa-se um chatterbot, sempre o usuário será tratado, por padrão, como uma terceira pessoa do singular com o sexo indefinido. Acontece que, conforme a conversação fica mais profunda, pode-se tratar esta pessoa como uma do sexo masculino ou do feminino, por exemplo. Sendo assim, as programações das categorias podem não ser muito eficientes quando da mudança de gênero.

Para satisfazer estas necessidades, utiliza-se das tags transformadoras ***<gender/>***, ***<person/>*** e ***<person2/>***, onde as mesmas detectam palavras-chave como artigos e adjetivos e as substitui pelo gênero, número e/ou grau desejados. Para tanto, é necessário à programação de um arquivo onde deverão ser contidas estas transformações, podendo varias de cada interpretador, mas possuindo sempre o mesmo modelo.

Quando um usuário introduz um padrão, o interpretador, caso não encontre nenhuma ocorrência, poderá automaticamente utilizar-se do arquivo de substituições para procurar a ocorrência parecida, e então retorná-la.

Assim, deve-se ter em mãos todos os artigos, pronomes e até adjetivos mais utilizados e montar-se uma matriz para satisfazerem-se todas estas substituições.

Na língua portuguesa, há um problema muito maior: a conjugação verbal. Diferentemente da língua inglesa, nossas transformações são muito mais complexas, e não seguem um padrão tão definido como a da língua inglesa, representando um desafio muito maior para os programadores utilizarem.

Recomenda-se no início pensar-se apenas em programar na terceira pessoa do singular, com o pronome “você”, e utilizar-se das tags transformadoras o mínimo possível, para só depois de definir-se uma sequência de roteiro mais sólida, fazerem-se as definições das substituições necessárias.

Os transformadores também podem ser utilizados em conjunto com condicionais. Ao invés de programarem-se milhares de categorias para satisfazerem-se ambos os gêneros, define-se apenas metade delas, utilizando-se os condicionais para direcionarem-se os modelos:

```
<category><pattern>Tenho os olhos azuis</pattern>
<template>Você parece muito
<condition name="gender" value="female">simpatica</condition>
<condition name="gender" value="male">lindo</condition>
</template></category>
```

Como Miracle é do sexo feminino, ela acha um homem lindo e uma mulher apenas simpática. O exemplo acima foi citado por Wallace (2003) na linguagem inglesa.

Outra maneira de programar-se esta mesma categoria seria apenas de utilizar-se de substituições alinhadas a recursões:

```
<category><pattern>tenho os olhos azuis</pattern>
<template>você parece muito <srai>elogiagenero</srai></template></category>
<category><pattern>elogiagenero</pattern>
<template><condition name="gender">
        <li value="female">simpatica</li>
        <li value="male">lindo</li>
        <li>elegante</li></template></category>
```

Ocorre o mesmo resultado do anterior, mas neste caso, a recursão pode ser utilizada para várias outras categorias podendo elogiar um gênero específico. Caso não haja gênero especificado, ele terá como padrão elogiar o usuário como elegante.

As tags abordadas neste capítulo são as essências para efetuar-se uma boa programação de um bot. Existem inúmeras outras a serem utilizadas, mas não farão parte deste trabalho.

Parte prática

Silva foi o autor mais interessado em como refinar a etapa de programação efetiva de um chatterbot. Ele define nove dicas básicas de como se organizar um chatterbot passo a passo. As dicas e técnicas mais interessantes, expostas neste estudo serão abordadas neste capítulo.

Separação de arquivos

Todas as categorias contidas em um chatterbot devem estar armazenadas em um local seguro. Além de estas estarem contidas nos respectivos arquivos ou banco de dados utilizados pelo interpretador, também devem possuir seu backup. A maneira mais simples de guarda-las é em seu próprio arquivo de extensão “aiml”.

O backup não deve ser efetuado apenas para contingenciamento do programa em si, mas também para facilitar sua alteração futura e organização. Os arquivos de categorias do chatterbot Alice e de diversos outros estão organizados por assunto.

Diferentemente dos tópicos, estas podem ser agrupadas indefinidamente sem possuir um tema específico. Um arquivo cumprimentos pode ser criado contendo apenas as categorias utilizadas ao se reverenciar alguém.

Comentários e referências de autor

Qualquer arquivo baseado em XML pode conter comentários. Um comentário, assim como em HTML, começa com <-- e termina com --!. Usar e abusar dos comentários auxilia qualquer programador, até o botmaster criador do bot a localizar-se e entender o que foi programado.

Referências sobre o autor e autoria, como datas, e-mail de contato e descrição sucinta do funcionamento também auxiliam em muito a vida de quem irá reutilizar o código. Um bot pode e deve ser programado tendo em vista sua alteração por qualquer usuário da comunidade livre e de novos pesquisadores.

Criação de roteiros

Ao invés de apenas sair programando várias categorias a fio sem nenhum objetivo, é preferível possuir um ou vários roteiros de conversação de um chatterbot. Um grande facilitador é ter bem definida a personalidade do chatterbot a ser desenvolvido, sendo mais fácil prever-se uma resposta a uma pergunta de cunho pessoal ao chatterbot.

Da mesma maneira, é necessário saber-se do público alvo do chatterbot para fazê-lo ser atrativo o suficiente, senão ninguém irá acessá-lo, ou outro público poderá acessá-lo, causando um desastre caso este chatterbot tenha um foco publicitário ou comercial.

Substituições

A maioria dos interpretadores de AIML possuem arquivos de substituições, funcionando como um dicionário de sinônimos, usando-os devidamente, economiza-se em muito a quantidade de categorias a serem utilizadas.

A maioria dos interpretadores já vem com as categorias de ALICE carregadas e já possuem seu arquivo de substituições programado, possuindo este arquivo, fica muito mais fácil adaptá-lo à língua portuguesa.

Como a conversação assemelha-se a um chat, é muito comum à utilização de abreviações neste diálogo. Então, as mesmas também deverão estar contidas neste arquivo.

Caso o interpretador utilizado não possua um arquivo de substituições, as mesmas deverão ser programadas através de categorias utilizando recursão.

Personalidade do chatterbot

Assim como nos humanos, um chatterbot pode ter seus gostos e preferências. Estas devem estar devidamente definidas no roteiro preliminar do bot. Alguns interpretadores possuem um conjunto de variáveis associadas a preferências, como filme preferido e etc. Estas preferências devem estar alinhadas ao foco do chatterbot, potencializando seu desenvolvimento. Como o foco deste trabalho é

desenvolver um chatterbot para estudo em uma faculdade, é preferível a utilização de uma linguagem formal, ser fã da faculdade e conhecer várias particularidades desta. Também deve possuir carisma, neste caso, é preferível um robô do sexo feminino.

Escritores de AIML e plataformas de teste

Uma boa pedida de um escritor de AIML off-line é o Simple AIML Editor da RIOT software. Com ele é possível escrever vários arquivos de AIML simultaneamente. Não é tão simples de se visualizar, mas é preferível a editar o arquivo aiml diretamente como XML para os iniciantes.

Um ambiente de teste magnífico, e também escritor on-line, é o site do pandorabots. Com ele é possível criar um chatterbot do zero, testá-lo e até divulga-lo para outras pessoas poderem utilizá-lo. Uma boa maneira de se programar, caso não possua muita experiência, é escrever-se um diálogo comum e utilizar-se seu programa codificador, para transformar o diálogo em AIML.

Uma desvantagem óbvia é deste não utilizar os mesmos métodos ou funções de um programa executado localmente, mas, possuindo-se o foco na conversação, é a arma mais poderosa a se utilizar.

Assim como em outros softwares, é extremamente recomendável testar quaisquer modificações no software em um ambiente de teste antes de colocá-lo em produção, evitando-se assim exibir falhas ao usuário.

Um botmaster experiente pode ter maior facilidade escrevendo diretamente um arquivo aiml utilizando um editor de texto. Neste caso é recomendado utilizar-se o notepad++, selecionando o tipo de arquivo como xml para melhorar-se a visualização. Esta foi a opção escolhida neste trabalho.

Monitore as conversas e visitas

A maioria esmagadora dos interpretadores atuais possui a facilidade de armazenar log. Isso facilita em muito o aprimoramento do chatterbot, pois as perguntas não

respondidas ou respondidas indevidamente pelo chatterbot podem ser identificadas, e futuramente corrigidas pelo botmaster.

A monotonia é o maior adversário de um botmaster, assim como em um site comum, afasta o usuário causando uma queda significativa de acessos pelo desinteresse. Um chatterbot deve ser capaz de entreter e informar o usuário da maneira adequada para seu público alvo.

Integração entre sistemas

Caso o chatterbot seja utilizado com alguma finalidade ou facilidades específicas demandando a utilização de outros sistemas, como acionar dispositivos ou comunicar-se com alguma outra linguagem, estas devem estar muito bem definidas antes de sua codificação.

Se forem programadas todas as categorias de um chatterbot, e após as funcionalidades do sistema a ser integrado sem um projeto anterior, pode ser necessário refazer-se várias categorias novamente, visando suprir as necessidades deste sistema. Caso deseje antecipar a programação das categorias, deixe a parte a servir de integração comentada e bem definida, visando sua mais fácil modificação futuramente.

Diagramação

Nenhuma das obras consultadas apresentou foi da utilização de diagramação. No começo deste trabalho foi exposto e comentado sobre o cérebro de ALICE, sendo este em forma de espiral onde a linha de espiral representa o tópico raiz e os demais são alinhados.

Além deste desenho, não há nenhuma outra plotagem, nova tecnologia ou software criadora de uma diagramação. Apesar de desejar manter a programação de AIML mais simples possível, uma diagramação auxiliaria em muito na hora de se escrever tags alinhadas com a tag *that*. Além do mais, uma plotagem efetuada de uma maneira dinâmica seria muito mais simples de visualizar e alterar-se caso necessário um chatterbot mais amplo.

Uma ideia neste projeto foi de, após definir-se o roteiro básico, rascunhar-se um diagrama de fluxo qualquer apenas para facilitar a programação, ainda mais se este chatterbot for integrado com outro sistema. Não poderá ser feito com todas as categorias, por demandar um tempo muito longo e talvez até ser feito em vão, mas pelo menos nas mais importantes pode auxiliar em muito a manutenção futura.

CONCEITOS BÁSICOS DE BANCO DE DADOS

Será ilustrado neste capítulo de como utilizar o banco de dados necessário ao nosso programa. Será falado superficialmente sobre as teorias, apenas para facilitar a utilização e adaptação do programa escolhido.

Introdução

Uma questão sempre levantada em qualquer curso básico de banco de dados pelo instrutor aos alunos é se estes sabem a diferença entre dado e informação. Um dado é apenas uma sequência qualquer de caracteres armazenada em algum lugar, enquanto informação é um dado no qual alguém sabe de sua utilidade.

Nos sistemas de computação, os dados podem ser armazenados em memória permanente ou em memória volátil. Os dados salvos em uma memória permanente ficam armazenados até sua remoção ou modificação, enquanto os dados da memória volátil se perdem após um desligamento do sistema ou do término da execução de um programa. Os dados podem ser armazenados em arquivos simples ou em um sistema gerenciador de banco de dados, apresentando este maior segurança, confiabilidade, velocidade dentre outras várias vantagens.

Todas as categorias serão armazenadas em um banco de dados através de nosso interpretador, entretanto o mesmo deve ser preparado previamente para tanto. Novos esquemas deverão ser desenvolvidos à parte para utilizarem-se outras funções necessárias ao nosso programa, caso queira-se armazenar dados.

Definições

Um banco de dados (BD) é gerenciado por um sistema gerenciador de banco de dados (SGBD). Existem vários sistemas diferentes cada um com suas particularidades, seguindo estes as mesmas definições básicas.

Um SGBD pode conter inúmeros bancos de dados diferentes. Este banco de dados pode ser denominado como esquema. Um banco de dados possui várias tabelas, também chamados como relação ou entidade. Esta tabela por sua vez possui várias

colunas, chamadas de atributos, e também possuem várias linhas, recebendo o nome de tupla.

As tabelas de um banco de dados recebem um nome identificador da tabela, e também podem possuir um relacionamento. Um relacionamento é uma estrutura que diz que algumas linhas de uma relação podem ter correspondência com outras de outras relações.

Uma maneira de representar um banco de dados pode ser feito também por diagramas, estes diagramas podem ser definidos como modelos de entidade-relacionamento (MER), pois neste são desenhados várias entidades e linhas interligando-as demonstrando os relacionamentos. Existem várias notações deste diagrama, com suas particularidades. Como o foco deste trabalho é de utilizar-se chatterbots, não serão abordados a fundo.

Tipos de atributos

Um atributo é nada mais de que um tipo de valores. Um atributo pode possuir vários tipos de dados diferentes, sendo numéricos de várias grandezas, booleanas ou alfanuméricas, dentre outras.

Um atributo possui força: um atributo pode ser forte ou fraco dependendo de seu possível conteúdo. Um atributo forte define-se por ausentar-se de repetição, ou seja, em nenhuma das tuplas de uma tabela este atributo estará repetido, havendo unicidade. A este atributo pode ser nomeado de chave candidata, pois a mesma é capaz, por esta força, de representar a tabela e suas linhas.

Pode haver em uma tabela mais de uma chave candidata, mas apenas uma pode ser a eleita a representante, e esta é denominada de chave primária.

Da mesma maneira, um atributo pode ser utilizado para se referenciar um relacionamento entre sua entidade e outra. E esta coluna é denominada e chave estrangeira, pois seus dados vêm de outra tabela.

É extremamente recomendável retalhar-se um banco de dados: quanto mais separada e organizadas estiverem suas entidades, muito mais simples e rápido

modificar-se o banco de dados, além de facilitar em muito na hora de utilizarem-se os dados com o programa.

Relacionamentos

Os relacionamentos fazem a junção lógica de duas tabelas distintas. Um dos grandes motivos de utilizar-se banco de dados é de diminuir-se o espaço de armazenamento dos dados. A forma mais precisa de se alcançar esta diminuição é evitar-se a redundância. A tabela a seguir mostra um exemplo clássico de desperdício:

CPF	NOME	Telefone	Endereço	CEP
322.123.123-12	Feliciano Augusto dos Anjos	1122223344	Rua Verde, 12	08333-234
322.123.123-12	Feliciano Augusto dos Anjos	1133334445	Rua Azul, 55	08435-344
322.123.123-12	Feliciano Augusto dos Anjos	1155556666	Rua Rosa, 56	08243-180

Figura 4 - Tabela Confusa

Este cliente possui três endereços distintos. Como seu nome está repetido várias vezes, estão ocupando espaço valioso, quase metade dos caracteres deste é desperdiçada.

Uma maneira de se evitar este desperdício é desmembrar esta tabela em duas: uma de endereços e uma de clientes. Pode-se fazer o CPF, sendo único neste caso, tornar-se nossa chave primária:

CPF	NOME
322.123.123-12	Feliciano Augusto dos Anjos

Figura 5 - Tabela clientes

Agora se define uma tabela de endereços, sendo o CPF seja uma chave estrangeira desta tabela, não havendo a necessidade de repetir todo o nome do cliente:

CPF	Telefone	Endereço	CEP
322.123.123-12	1122223344	Rua Verde, 12	08333-234
322.123.123-12	1133334445	Rua Azul, 55	08435-344
322.123.123-12	1155556666	Rua Rosa, 56	08243-180

Figura 6 - Tabela Endereços

Esta tabela não está plenamente normalizada, mas já é possível ter-se uma ideia de como reduzir-se espaço e iniciar-se a normalização.

Normalização

Todos os dados em um banco de dados devem estar organizados de maneira clara e eficiente, visando garantir melhor desempenho e economia de espaço, evitando-se a redundância, como demonstrado utilizando as chaves estrangeiras, relacionamentos entre outras técnicas. A este tipo de organização dá-se o nome de normalização.

Esta normalização está dividida em cinco etapas, denominadas de formas normais (FN). Para normalizar-se um banco de dados completamente, devem ser normalizadas na sequencia da 1ª FN (forma normal) até a 5ª FN. Portanto, para normalizar-se um banco de dados na 2ª FN, esta já deve estar na 1ª FN.

Primeira forma normal

Para a primeira forma normal, removem-se os atributos multivalorados. Um exemplo de atributo multivalorado é o presente na tabela confusa (Figura 4).

Na *tabela confusa*, as tuplas são distintas, pois representam valores diferenciados, porém, nas três primeiras colunas, os valores estão repetidos, portanto, não obedecem à primeira FN.

Para normalizá-la na 1ª FN, proceder ao processo de dividi-la em duas e utilizar-se de relacionamentos, como mostrado nas figuras 5 e 6.

Segunda forma normal

A segunda forma normal diz respeito às chaves primárias e seus atributos dependentes. Para uma tabela estar na segunda FN, esta deve ter todos os seus atributos não chave dependentes na totalidade de sua chave primária, ou seja, se não houver este atributo de chave primária, a tupla perderá totalmente o seu sentido.

Esta dependência é chamada de dependência funcional, pois o funcionamento da tabela sem esta dependência é comprometida.

Terceira forma normal

A 3ª FN fala sobre dependência transitiva e as chaves candidatas. Uma dependência transitiva funciona quando um atributo A depende de um atributo B, e este atributo B depende de outro atributo C em uma mesma tabela, mas continua dependendo de A por estar na mesma relação. Para normalizar-se esta tabela, remove-se esta dependência transitiva, fazendo-se a dependência ser de A para B e em outra tabela de B para C, assim C não dependerá transitivamente de A.

Um exemplo pode ser dado na seguinte tabela:

CodPed	CodCli	NomeCli	NomeCidCli
1	1	Fulano	São Paulo

Figura 7 - Tabela Não Normalizada

Neste caso, o código da cidade do cliente e o nome da cidade do cliente dependem transitivamente do código do pedido. Para eliminar-se esta dependência transitiva, desmembra-se esta tabela em três partes distintas:

CodPed	CodCli
1	1

Figura 8 – Tabela Pedidos

CodCli	NomeCli	CodCidCli
1	Fulano	1

Figura 9 - Tabela Clientes

CodCidCli	NomeCidCli
1	São Paulo

Figura 10 - Tabela Cidades-Clientes

Caso execute-se apenas a segunda forma normal, haveria apenas duas tabelas, sendo que a cidade do cliente ainda seria dependente do código do pedido. Neste caso, normaliza-se a terceira forma normal. Caso desejar encontrar a cidade do cliente do pedido 1, faz-se através do cliente, mas sem a necessidade de manter a relação não normalizada.

As tabelas demonstradas como exemplo ainda podem ser completamente normalizadas e melhor organizadas podendo subdividi-la ainda mais e fazerem-se várias adaptações. Não será aprofundado tanto neste trabalho a ponto de efetuá-la, mas se frisa esta necessidade caso queira deixa-las totalmente organizadas.

Linguagem SQL

A linguagem SQL (*Structured Query Language*), significando linguagem estruturada de consultas, foi desenvolvida pela IBM em meados dos anos 70, tendo como função viabilizar a implantação de um modelo de banco de dados relacional.

Esta é composta por vários comandos em modo de texto, sendo separado cada um por ponto-e-vírgula.

Os comandos na linguagem SQL são separados em cinco subgrupos de linguagem, sendo alguns estes expostos neste capítulo. Será dada maior ênfase apenas no conteúdo necessário para programar nosso robô tagarela.

A linguagem SQL pode ser interpretada por praticamente todos os SGBDs existentes no mercado. Também a linguagem não é sensível ao caso, ou seja, não faz distinção entre maiúsculas e minúsculas. Assim como o COBOL (*Common Business Oriented Language*), ela é muito similar ao inglês, sabendo-se inglês, fica muito fácil programar-se, pois os comandos são muito simples de escreverem-se.

DML – Linguagem de Manipulação de dados

A Linguagem de manipulação de dados é responsável por todo o processo de manipular os dados de uma tabela. Através dela é possível criar novas tuplas, alterá-las, apaga-las ou selecioná-las. Os comandos da DML não alteram nada da estrutura do banco de dados ou da tabela, alterando apenas o seu conteúdo.

O comando Insert

O Comando *Insert* é responsável por inserir novos dados em uma tabela já existente. Para tanto, é necessário conhecer-se a tabela a inserir os dados. Há duas maneiras de inserirem-se os dados: diretamente na tabela ou especificando os campos a serem inseridos. O comando INSERT possui a seguinte sintaxe:

INSERT INTO nome-da-tabela (<colunas>) VALUES <valores>

Tendo-se como exemplo uma tabela Alunos em um BD escola com três campos: Código, Nome do Aluno e Turma, pode-se inserir os dados diretamente da seguinte maneira:

INSERT INTO Escola.Alunos VALUES ('1','Jefferson Mendes','1');

Caso não se recorde da ordem dos campos, insere-se diretamente da seguinte maneira:

INSERT INTO Escola.Alunos ('Código','Nome','Turma') VALUES ('1','Jefferson Mendes','1');

Caso haja algum erro, o SGBD retornará uma falha, e não inserirá esta linha na tabela, sendo obrigado o programador a corrigi-la. Isto funciona para qualquer comando a ser executado em SQL.

O comando Delete

O comando *delete* é responsável por apagar-se linhas de uma tabela. Sendo necessário especificar qual o critério que o comando deverá obedecer utilizando-se da cláusula WHERE. Esta cláusula será detalhada em momento mais oportuno.

O comando "*delete"* pode apagar desde apenas uma única linha como os dados de uma tabela inteira, portanto, deve-se tomar cuidado quando de sua utilização. Caso não seja inserido o critério, todas as linhas da referida tabela serão apagadas. Alguns SGBDs impedem esta ação de apagar todos os dados da tabela através de uma configuração de segurança.

A sintaxe do comande "*delete"* é dada da seguinte maneira:

DELETE FROM nome-da-tabela WHERE <critérios>;

Este comando apaga apenas os dados da tabela, mesmo apagando-se todos os dados, a tabela ainda continuará existindo com todas suas colunas, mas apenas sem dados.

Atualização de dados

Para ser realizada a atualização de dados, utiliza-se o comando UPDATE, este comando atualiza uma linha ou sequência de linhas a serem modificadas.

Da mesma maneira que uma sequencia de dados deve ser selecionada antes de ser apagada, um dado a ser atualizado também precisa ser selecionado. Neste caso, a cláusula where deve ser adicionada a este comando, contendo o critério para a sua atualização.

No exemplo da tabela endereços, simplesmente pode-se modificar um número de uma rua, para tanto se solicita um número de CPF e um CEP para identificá-la.

A sintaxe deste comando é como se segue:

UPDATE nome-da-tabela SET coluna1 = novovalor1, coluna2 = novovalor2,... WHERE critérios;

No nosso caso faz-se o seguinte:

UPDATE Endereços SET endereço = Rua Verde, 15 WHERE CEP = '08333-234' AND CPF = '322.129.038-12'

Apagar todos os dados de uma tabela

Caso deseja-se apagar todos os dados de uma tabela, mas desejando-se continuar com ela em branco inserida ainda no banco de dados, utilizar o comando ***truncate table.***

Este comando apaga todas as linhas da tabela, mas mantém todos os atributos dos campos, sendo possível inserir os dados novamente. Este comando será utilizado na manutenção do chatterbot para a parte de aiml.

TRUNCATE TABLE endereços;

DDL – Linguagem de Definição de Dados

A Linguagem de Definição de Dados é necessária para a manipulação da estrutura das tabelas em um banco de dados.

Com ela é possível criar-se, alterar e apagar tabelas. É necessário existir um banco de dados e as tabelas a serem relacionadas antes de sua criação ou modificação.

Criação de banco de dados

Para criar-se um banco de dados, utiliza-se o comando ***CREATE SCHEMA***. A partir desta criação, tabelas podem ser criadas, alternadamente, o comando ***DROP SCHEMA*** remove um banco de dados, com todas as suas tabelas e seus respectivos dados.

Por razão de segurança, recomenda-se habilitar o modo seguro do banco de dados num ambiente de produção, evitando a execução deste comando de exclusão.

CREATE SCHEMA <nome do banco de dados>;

Criação de tabelas

Para as tabelas serem criadas através do SQL, devem-se ter previamente as seguintes informações:

- Nome do banco de dados no qual a tabela será inserida;

- Nome da tabela desejada;
- Os nomes dos campos (colunas);
- O tipo de dados dos campos (string, inteiro, real, booleano, etc.);

De posse destas informações, utiliza-se o comando ***CREATE TABLE*** para criar uma tabela em um campo de dados. Observar que cada SGBD pode possuir os tipos de campos diferenciados, alguns exigindo o tamanho do campo e outros não, e outros opcionais em determinados tipos. NOTA: os nomes não devem conter espaços e é preferível não utilizar-se acentos.

A sintaxe do comando ***CREATE TABLE*** é como se segue:

CREATE TABLE <nome do banco>.<nome da tabela>(
<nome da coluna1> <tipo do campo1>(tamanho do campo1),
<nome da coluna2> <tipo do campo2>(tamanho do campo2),
...);

Assim, cria-se a tabela de usuários de aiml, por exemplo:

CREATE TABLE miracle.aiml_users (
user_login varchar(50) PRIMARY KEY,
user_name varchar(80));

Parâmetros adicionais também podem ser inseridos logo após o tamanho do campo para cada coluna, como o PRIMARY KEY, referenciando o campo como chave primária da tabela e os CONSTRAINT e REFERENCES para atribuir-se chave estrangeira a um campo. Lembre-se que cada SGBD possui suas particularidades e estas deverão ser consultadas anteriormente.

Alteração de tabelas

Os campos das tabelas podem ser alterados, mesmo depois de criados. Podem ocorrer erros, exclusões ou transformações de dados de campos em algumas situações, portanto, ao alterar-se o tipo de dados de um campo, devem analisar-se as consequências. O mesmo ocorre com a alteração de nome de campos, pois se o mesmo for referenciado como chave primária ou estrangeira, pode haver falhas na consistência do banco.

A excluir-se um campo, excluem-se também os dados respectivos, às vezes sem opção de recuperação.

Uma tabela pode ser alterada através do comando ALTER TABLE.

Exclusão de Tabelas

Uma tabela pode ser totalmente excluída através do comando DROP TABLE. Como ao excluir-se um banco de dados, os dados desta tabela também são excluídos.

CONCEITOS BÁSICOS DE AUTENTICAÇÃO, SEGURANÇA E UTILIZAÇÃO DE BANCO DE DADOS COM PHP.

Este tópico é destinado a exemplificar como utilizar o PHP conjuntamente com banco de dados, tendo como principal foco a utilização dos dois para criar-se um sistema de autenticação de usuários e como repositório de dados pessoais.

A princípio, distingue-se um usuário de outro, e a partir daí possuir a certeza se este usuário é quem diz realmente ser.

Para tanto, precisa-se criar uma tabela contendo os dados destes usuários. Esta tabela deve possuir como identificador principal, ou chave primária, uma informação única do usuário, podendo ser um número de CPF ou um nome de usuário, por exemplo. Neste caso, lidando-se com chatterbots, opta-se por selecionar o nome de usuário como chave primária desta tabela.

Adicionalmente, necessita-se de adicionar um campo de senha em nossa tabela, pois o mesmo usuário deverá ser validado, entende-se como validação uma confirmação de que o usuário é que realmente diz ser.

Não se armazena a senha em sua forma pura, pois qualquer pessoa ou programa mal intencionado pode capturá-la e utilizá-la sem grandes problemas.

Utiliza-se, então uma técnica chamada de criptografia. A criptografia consiste em pegar-se uma sequência de caracteres quaisquer e transformá-la em outra praticamente indecifrável. Para isso utiliza-se uma chave de criptografia. A este texto modificado dá-se o nome de texto cifrado.

Existem dois tipos de chaves de criptografia: a chave pública e a chave privada.

Uma chave pública é um algoritmo de criptografia destinada a distribuir-se a vários usuários, para que os mesmos entrem com os dados a serem criptografados e que obtenham a saída deste texto cifrado.

Uma chave privada realiza o inverso: ela recebe um texto cifrado e retorna o dado original. Esta chave é restrita a uma pequena quantidade de usuários e a aplicações

específicas. A principal finalidade é de cifrar-se um texto a ser transmitido por via pública, como na internet, sendo este texto recebido e decifrado no seu destino para ser posteriormente utilizado. Segredos de estado pode ser um bom exemplo disso.

Abaixo é ilustrado um exemplo deste processo:

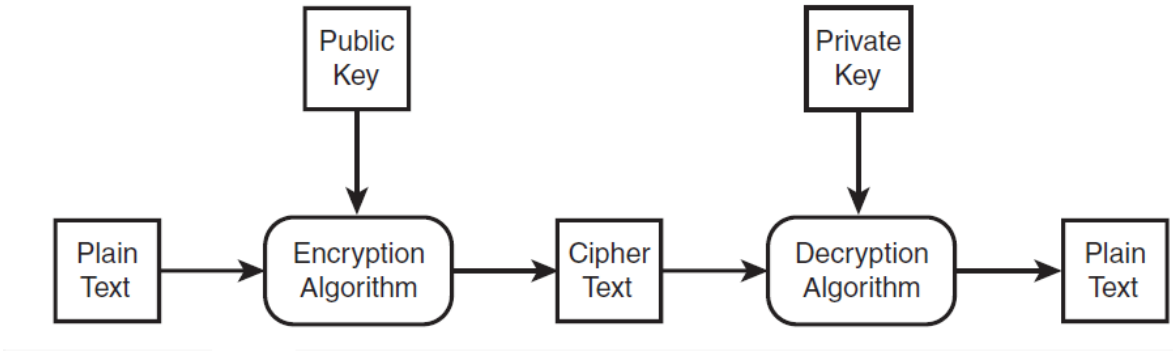

Figura 11. Fonte: WELLING (2005) – pag. 350

Entretanto, a nossa finalidade é de criar-se um ambiente de autenticação seguro. Portanto, é interessante utilizar-se de um algoritmo de criptografia que não possua um algoritmo de decriptação, ou seja, não possibilitando através de um texto já cifrado obter-se o texto original, pois este processo poderia ser utilizado por hackers para obtê-lo, facilitando o trabalho deles.

O PHP já possui um algoritmo concebido nativamente para esta finalidade, sendo o SHA1 (Secure Hash Algorithim 1).

Este é um algoritmo de tamanho fixo, ou seja, não importa o tamanho do texto puro a ser encriptado, ele sempre retornará um texto do mesmo tamanho com o texto cifrado, complicando ainda mais a sua decifragem. É um método tão eficiente que nem seu próprio criador consegue reverter o processo.

Como fazer então para utilizar-se deste método para armazenamento de senhas?

- Pegar a senha do usuário e cifrá-la;
- Armazenar-se este texto cifrado no banco de dados juntamente com o nome do usuário em outro campo;

- Ao autenticar-se o usuário, cifra-se a senha fornecida e compara-se com a já cadastrada no banco de dados. O algoritmo escolhido sempre retornará o mesmo texto cifrado para o mesmo texto plano, sem falhas. Se ambas forem iguais, a senha estará correta!

A função que no PHP faz o processo de criptografia do SHA1 é a função **sha1()**;

Para tanto, fazer-se o seguinte:

$textocifrado = sha1($textooriginal);

Então a variável do PHP $textocifrado é o texto cifrado do texto plano $textooriginal após o processamento desta linha de código.

Primeiramente, é necessário conectar-se ao SGBD desejado. Como será utilizado o MySQL, utiliza-se a função mysql_connect().

O comando mysql_connect() possui a seguinte sintaxe:

$link = mysql_connect("nome do servidor", "nome do usuário", "senha");

A variável $link serve para armazenarem-se os dados da conexão. Caso ela seja mal sucedida, ela retornará falso. Esta variável pode ser utilizada em outros comandos ou para depuração.

Sempre que uma conexão for aberta, o PHP a utilizará até o final do processamento do script, encerrando-a ao seu final automaticamente ou quando esta for alterada.

Há duas tarefas principais para o PHP nesta parte de autenticação de usuário: criar-se um usuário e efetuar seu *login*.

As consultas do PHP para o MySQL, tanto para a maioria de outros SGBD no PHP, são feitas através de SQL. Utiliza-se então a consulta SQL do capítulo anterior par criar-se o usuário no PHP utilizando a função mysql_query().

Esta função possui a seguinte sintaxe: $resultado = mysql_query ($sql, $link);

Onde *$sql* é a consulta SQL a ser utilizada e o $link é o link da conexão ao banco de dados, sendo opcional caso já foi feita a conexão anteriormente, mas necessária se

forem feitas mais de uma conexão no mesmo script. $result é o resultado da consulta, podendo ser verdadeiro, falso ou um dado específico.

Usa-se então a seguinte sequência para cadastrar-se um usuário:

```
$link = $mysql_connect("localhost","usuario","senha");
$sehnhacripto = sha1($senha);
$resultado = $mysql_query("INSERT INTO aiml_book.book_users (login, nome, senha) VALUES ('$login','$nome','$senhacripto');");
```

Para efetuar-se o login, utiliza-se a mesma função mysql_query(), mas usando uma consulta SELECT ao invés de uma INSERT:

```
$resultado = mysql_query("SELECT senha FROM aiml_book.book_users where (book_users.login = '$login'));
$dados = mysql_fetch_array($resultado);
If (sha1($senhafornecida) == $dados[0]) echo ("Senha correta! Você está autenticado!);
else echo ("Senha incorreta! Tente novamente ou até a próxima!");
```

Nota-se que o PHP reconhece qualquer variável e retorna o valor da mesma dentro da sequência de caracteres da consulta, e também da necessidade de cifrar-se a senha fornecida antes de compará-la a armazenada no banco de dados.

Para completar todas as necessidades do programa, funções para substituição de senha, nome e exclusão de usuário também devem ser previstas. Tais códigos não serão exemplificados neste tópico por não apresentarem nada de muito diferente. Para alterar-se ou excluir-se uma tupla da tabela, basta usar-se dos comandos DELETE e UPDATE do SQL demonstrados no tópico sobre banco de dados na consulta utilizando o PHP.

Outro recurso pode ser utilizado com alguma segurança chamam-se cookies. Os cookies são dados armazenados na máquina do usuário para uso posterior. Existe um tipo particular de cookie armazenado na máquina do usuário chamado de cookie de sessão. Segundo a Microsoft, um cookie de seção é armazenado na memória enquanto o site é visitado, e é destruído quando a conexão ao servidor web é desfeita.

Este é o melhor para utilizar-se em autenticação, pois ao invés de ser efetuado o login toda vez que o usuário necessitar de acessar algum recurso protegido, pode-se

armazenar em um cookie de seção o login do usuário já autenticado consultando-o sempre quando necessário. Como as páginas web perdem a informação armazenada nas variáveis e campos ao recarregarem-se as páginas, este recurso pode muito bem ser utilizado.

Outro tipo de cookie é chamado de cookie persistente. Um cookie persistente armazena as informações da página no disco rígido de maneira “permanente”. Portanto, sempre pode ser utilizado, enquanto estiver gravado na máquina do usuário, para guardar informações para uma visita posterior à página solicitada. No caso deste projeto, ele será utilizado para armazenar dados do usuário, como nome, data de nascimento e outros, sem a necessidade constante de acessar-se o banco de dados.

No PHP é muito simples utilizar-se cookies de seção, basta utilizar-se da seguinte sintaxe:

$_SESSION[‘nome_do_cookie’] = <valor>;

Para tanto a seção necessita ser inicializada para os cookies de seção poderem ser armazenados com a função session_start();

Há apenas um inconveniente de utilizar-se cookies, a própria segurança!

Como cookies armazenam texto puro, podem muito bem ser interceptados por hackers ou lidos localmente na máquina do usuário. Portanto, não é recomendado guardar dados sigilosos neles, como número de cartão de crédito ou senhas bancárias.

Tendo ciência deste aviso, muitos usuários tem medo de os cookies poderem armazenar vírus, sendo isso impossível, pois o cookie não executa código. Um alerta feito por CONVERSE (2004) diz que todo o site deve prever que um usuário recuse-se a receber cookies. Neste caso, ou força-se o usuário a receber cookies ou minimiza-se o desempenho de seu acesso, efetuando-se mais consultas no banco de dados.

No caso de utilizar-se os cookies persistentes, deve-se codificar a função setcookie();com a seguinte sintaxe:

setcookie('nome','valor','validade','caminho','domínio','seguro');

- Nome: É o nome da variável do cookie a ser salvo;
- Valor: É o valor a ser guardado do cookie;
- Validade: É a validade do cookie em segundos. Deve ser usado o formato do mktime() ou preferencialmente ser utilizada mktime() para programar a data, deve ser inteiro e representa a quantidade de segundos desde 0:00 de 01/01/1970;
- Caminho: é o caminho relativo de onde deve ser armazenado o cookie. Ex: '/pasta/';
- Domínio: é o domínio a qual será atrelado o cookie. Ex: 'localhost.miracle';
- Seguro: 0 para não seguro e 1 para seguro. Requer HTTPS ou SSL para conexões seguras. As mesmas precisam ser inicializadas anteriormente.

A função mktime possui a seguinte sintaxe:

$valor_em_segundos = mktime (horas, minutos, segundos, MÊS, DIA, ano);

AIML COM PHP, COMO UTILIZAR?

Nesta parte do trabalho, serão expostas as particularidades do “Program-O”, tanto na parte desenvolvida previamente por Elisabeth Pearrau como na parte modificada para possibilitar o andamento de nosso projeto, pois sem as mesmas, não existiria esta possibilidade. Os números das linhas em relação ao programa original podem estar diferentes, devido à constante alteração.

Dicas para instalação

Primeiramente deverão ser configurados um ambiente para desenvolvimento web em PHP e também um SGBD MySQL. A maneira mais simples de consegui-los é instalar o programa “EasyPHP” disponível no “Sourceforge”. Existem inúmeros tutorias ensinando sua utilização e instalação.

Após baixar o “Program-O” original ou outras versões modificadas, como a deste trabalho, e publicá-lo no servidor PHP e MySQL, será necessário criar-se o banco de dados a ser utilizado e os usuários do banco de dados do robô e do botmaster, utilizando qualquer uma das ferramentas preferidas e/ou disponíveis pelo instalador.

Após ser efetuado o procedimento acima, será necessário modificar-se os arquivos de configurações do “bot” com estas informações em dois arquivos: um do robô e um do administrador.

O caminho do arquivo do robô é “/bot/config.php”. Acessando-o, preencher seus dados nas linhas de 26 a 29 neste arquivo, com o nome de host (geralmente “localhost”), do banco de dados, do usuário do bot e a senha deste usuário respectivamente.

O mesmo deverá ser feito para a interface do administrador no arquivo (“/admin/funcs/config.php”) nas linhas de 12 a 15 com os dados do botmaster.

Assim, é possível instalar o programa através do link (“http://<host>/install_programo.php”). Caso não haja falhas, a instalação será completada. Havendo, corrija o problema e refaça a instalação, dependendo do passo, uma correção e posterior recarga da página já resolvem o problema.

Quanto à interface do administrador, apenas clique em “install my program-o admin area” ao final da instalação do Program-O para que este seja instalado automaticamente. Neste passo ele pedirá um nome de usuário e uma senha, devendo estes ser memorizados para se acessar a interface do administrador.

Dependendo da versão instalada, o mesmo já possuirá todas as categorias AIML padrão inseridas no BD, sendo que a versão oficial possui o bot ALICE e a versão deste trabalho possui o bot Miracle. Não desejando nenhuma delas, a tabela aiml deverá ser apagada através de um comando SQL “truncate table” e depois de inserido o novo AIML pela interface de administrador do Program-O.

Falhas do programa original

Ao instalar-se o Program-O original em um ambiente executando o Xdebug, algumas falhas e avisos são expostos no browser do usuário. O Xdebug é um programa que faz o mapeamento de erros do PHP, também dando algumas dicas sobre a má utilização de funções e variáveis. É interessante saná-las antes de publicar-se o site. Aqui se faz estas correções. Tais falhas já foram corrigidas no robô Miracle.

Ao iniciar-se o Program-O ele apresenta uma falha na inicialização de sessão. Esta falha consiste em iniciar uma sessão com uma já iniciada por outro processo. Para sanar este erro, basta utilizar-se do curinga supressor de erro “@” no início da linha apresentadora da falha. Existem outras maneiras mais elegantes de saná-la, utilizando um bloco if, por exemplo, não havendo necessidade neste caso, pois este erro não provoca nenhum efeito colateral no funcionamento do programa, além de prejudicar sua estrutura.

Este erro específico ocorre no arquivo “bot/response_handler.php” na linha 2, devendo ficar da seguinte maneira:

@session_start();

Outra falha acontece ao se executar a interface do administrador. Ao executá-la, ele faz a leitura de uma variável que não existe, e este gera um aviso do Xdebug, acabando com a formatação do site, para saná-lo, basta inicializar a variável com

um valor nulo. Então no arquivo “admin/pages/inc/header_nav_class.php” basta inicializar a variável $bclass com “” na linha 14 assim:

$bclass = “”;

Funcionamento do Programa-O

Inicialização

O Programa-O vem com um arquivo inicial chamado de “index.php”, este inicializando o nosso robô. Esse arquivo nada mais é que um arquivo html padrão incluindo o arquivo “bot/chat.php”, e ecoa o valor da string “$res”, sendo esta a interface de diálogo, um pulo de linha (
) e a entrada do usuário e a “$formchat”, esta contendo o formulário principal do chatterbot. Este arquivo pode ter seu nome alterado sem nenhuma consequência. Além destes, foi inclusa uma variável “$figura”, com a função de mostrar-se a face de nosso robô.

Qualquer arquivo ou página pode conter invocar este arquivo e, por conseguinte, exibir estes formulários, com o cuidado de não haver conflito de variáveis e de funções.

Há também a possibilidade de incluir-se um “Iframe” e apontá-lo para a página principal do robô, diminuindo assim o problema de conflito.

Alterar o formulário do chatterbot

O programa-o original possui seu formulário em inglês, visando à tradução do programa para português, este deve ser modificado. Basta então abrir o arquivo “bot/response_handler.php” e editar-se a função “formchat()” a partir da linha 550. O robô Miracle já se encontra em português. Outras modificações também foram feitas, mas serão explicadas mais adiante.

Alterar a interface de diálogo

Assim como no robô Miracle, é possível alterar também a interface de diálogo do robô. Este foi modificado em alguns trechos nas linhas 135 e 136 do arquivo “bot/chat.php”.

Descrição dos arquivos principais

Foi descrito acima as funções dos arquivos config.php, tanto no bot como no administrador. Outros arquivos essenciais também serão explicados. A maioria destes estão contidas no diretório “bot/’.

Bot/chat.php

Este arquivo é o inicializador do bot. Ele invoca todo o processamento do chatterbot e também é responsável por fazê-lo visível ao usuário.

Como o PHP é um programa volátil, não ficando residente na memória do servidor, todas suas variáveis são destruídas logo após a página ser totalmente carregada. Então, para passarem-se os valores de suas variáveis de uma página para outra, podem ser utilizadas das variáveis de sessão ou de passagem, através dos métodos “POST”, “GET” ou “SESSION”.

Todas as variáveis de conversação do robô são passadas por um vetor multidimensional indexado por string denominado de response_Array. Este vetor é passado a cada novo envio de dados ao robô e novo recarregamento de página através do método “POST”.

Este arquivo também possui a interface de diálogo a ser exibida no site. No final do arquivo, contém algumas instruções exibidoras de conteúdo das variáveis de conversação do robô, caso estas sejam descomentadas.

Bot/response_handler.php

Este arquivo é responsável para tratar a entrada do usuário, processar e entregar a resposta ao arquivo “chat.php”. Ele formata a entrada do usuário, chama as funções de tratamento de tags AIML e consulta o banco de dados para localizá-las.

Também expõe o formulário de resposta do bot.

Bot/check_aiml_part.php

Este arquivo é o responsável por verificar todas as tags inseridas no aiml e mandar executá-las caso encontre alguma correspondência com as programadas.

Caso deseja-se inserir novas tags no robô, esta deve ser inclusa neste arquivo antes de ser programada no arquivo chamador.

Bot/tag_functions.php

Após as tags serem identificadas, estas deverão possuir funções para que as mesmas sejam executadas. Neste arquivo está a maioria destas funções. A inicialização da função e término são muito parecidos e podem até ser copiadas, alterando-se poucas palavras para fazer-se novas tags.

Bot/getsetvars.php

Existem também as variáveis globais do AIML set e get, sendo estas gerenciadas por este arquivo. Outras funções essenciais também são definidas neste arquivo, como gênero e star.

Bot/debugging.php

Este arquivo é responsável pelo controle de erro do bot. Todos os erros podem ser ecoados ou omitidos na sua execução através deste arquivo. Para tanto basta comentar-se e descomentar-se as linhas 14 e 15 para obter o efeito desejado.

Modificações no programa

Os arquivos acima descritos precisaram ser modificados para que a Miracle pudesse realizar as funções desejadas. Neste capítulo estas serão demostradas e comentadas.

A tag system

A tag system original serve apenas para efetuar cálculos matemáticos simples e retornar seu resultado ao usuário. Como foi necessário fazer os modelos executarem funções do php, sendo esta a utilidade padrão desta função no padrão AIML, esta foi reprogramada no arquivo “tag_functions.php”.

Utilizando-se a tag <system>, executará quase qualquer função php sem a necessidade de utilizarem-se as tags de início e término padrão (<?php e ?>).

Esta faz a separação de todos os comandos e executa-os individualmente através de vários comandos “eval”. Como os comandos precisam necessariamente ser inclusos no banco de dados pelo botmaster antes de serem executados, dificulta-se o a trabalho de usuários mal intencionados.

Uma observação deve ser feita a trabalhar-se utilizando orientação a objetos ou ao utilizarem-se classes. Para referir-se a métodos ou propriedades de um objeto instanciado no PHP, não sendo este estático, utiliza-se o operador “->”.

O grande problema é a interpretação pelo Program-O para orientação a objetos. Ao inserir-se o operador de objetos “->”, ocorrerá uma grande falha no programa, pois ele trata o sinal de maior como fechamento de tag. Para contorna-lo, utilize ao invés deste operador o operador “-:”, onde os dois-pontos serão substituídos pelo sinal de maior ao executar-se a tag system.

A classe autenticar, responsável por cadastrar e autenticar usuários foi propositalmente programada desta maneira para servir de exemplo.

O mesmo ocorre com o operador de índice de vetores “=>”, o mesmo deve também ser substituído por “=:”.

Manipulando as respostas pela tag system

Para manipularem-se as respostas da tag system, será necessário saber previamente a utilidade do resultado.

Caso deseja utilizar-se na resposta do robô, como parte da tag template, deverá ser utilizado o comando “*return”*. O motivo de utilizar-se “*return”* deve-se ao fato do comando “*eval”* ser uma função, e este valor só será retornado à função chamadora após a execução deste comando.

Contudo, se a necessidade for de utilizar-se o resultado para fora do formulário de resposta do robô, como utilizar-se de “*javascript”* ou outros recursos, como digitar-se um texto na página, utilize-se do comando “*echo”* no seu lugar. Ecoando outros valores fará com que o texto seja digitado antes da interface de diálogo do robô.

Não utilizando nenhum destes dois comandos, o conteúdo da tag system será executado, mas nada será escrito nem no vídeo e nem no formulário de resposta do robô.

A tag solve

É interessante manter o funcionamento da tag system original, pois o usuário pode simplesmente perguntar ao bot “quanto é dois mais dois?”, para solucionar este problema, foi criada uma nova tag “solve”, realizando esta função.

A mesma suporta resolver equações aritméticas de adição, subtração, multiplicação e divisão com suporte a parênteses, também transforma o texto das operações em sinais. Como os números podem ser escritos por extenso, os mesmos serão substituídos antes.

Não será dada tanta ênfase para efetuar operações complexas ou para acertar sempre neste momento, pois o foco do programa é a conversação. Caso o robô não conseguir responder a esta pergunta, ela utilizará de uma resposta padrão. Esta tag ainda será amadurecida para apresentar cem por cento de acerto e mais funções.

Limpeza dos padrões

O programa original efetua a limpeza dos padrões antes que estes sejam processados pelo aiml. Isto significa que alguns caracteres podem ser omitidos ou substituídos por outros. O mesmo acontece com algumas palavras.

Os caracteres a serem limpos estão no arquivo “response_handler.php”, onde o mesmo apresenta várias palavras e sequencias de caracteres a serem substituídos.

Estes são feitos na sua maioria por dois comandos: o “*str_replace”* e o “*preg_replace”*. O primeiro procura uma sequencia de caracteres em uma *string* e a substitui pelo valor inserido no comando. O segundo procura por um padrão de expressão regular e a substitui, na maioria dos casos as excluindo. Uma expressão regular serve para se localizar vários caracteres de uma só vez, como alfabéticos e numéricos, por exemplo.

Todo o padrão inserido pelo usuário do robô é transferido para a variável $tmp neste comando e em seguida são feitas as substituições. Caso deseje alterá-las, basta modificar, excluir ou incluir novas linhas, utilizando a função apropriada e as substituições desejadas.

As alterações para a autenticação através do AIML

Além da criação da classe para autenticações e de bookmarks, outras alterações no programa fonte do robô também foram necessárias torna-lo possível. Aqui serão expostas estas alterações.

Alterações na tag system

A tag system foi previamente alterada para que os modelos executassem PHP, além disso, ela também deve poder procurar as classes necessárias e outras funções essenciais. Para tanto é necessária declarar, nesta função, a inclusão do arquivo onde estas funções residem através da diretiva “include”.

Recomenda-se incluir apenas um arquivo include, sendo que estes também incluam outros diversos arquivos a serem carregados.

Alterações no manipulador de diálogo e no formulário do robô

Ao implantar-se a autenticação, notaram-se três problemas:

1. Mesmo após definir o nome do usuário, o campo do diálogo continuava mostrando o nome do usuário “você”;
2. Ao inserir-se qualquer senha, durante sua digitação, a mesma era mostrada no campo de texto do formulário, possibilitando de outras pessoas a verem.
3. Após digitar esta senha, a mesma é mostrada integralmente na parte de diálogo do usuário.

Estes problemas foram solucionados como se segue:

1. Quanto ao problema número um, bastou-se atribuir uma variável com o nome do usuário, definida como “$nomeusuario” com o valor padrão de “Você”. Também foi definida uma variável de sessão nome de usuário, se esta for modificada modificará também o nome do usuário mostrado no diálogo.

2. Já no segundo problema, o formulário de conversação do robô no programa original, está definido estaticamente como um tipo de caixa de texto. Para tanto, basta-se modificar o tipo de um input para *type=”$tipotexto”*, por exemplo. Também foi definida uma variável de sessão denominada tipotexto que a altera e mantém o estado da variável $tipotexto. Mudando o valor da variável para “password”, o problema está resolvido.

3. O terceiro problema foi o mais difícil de solucionar, pois não se sabia de onde vinham os valores expostos na saída do usuário por falta de documentação. Analisando as variáveis, notou-se que a maioria das variáveis do robô eram passadas através de um vetor multidimensional denominado de $response_array, mas além deste, há outro vetor multidimensional $_POST[‘response_array’] e vários outros valores fora deste vetor, apresentando ligeira diferença quanto aos seus índices.

   Percebeu-se então, a existência de um vetor “input”, responsável por armazenar as entradas de texto pura do usuário, sem qualquer tipo de tratamento. Alterando o seu valor para “*****” através da tag system, o mesmo esconde a senha digitada do usuário. Esta descoberta também abre diversas possibilidades de alterações futuras conforme novas necessidades.

Para alterar-se qualquer uma destas variáveis, basta utilizar a tag system, atribuindo o valor desejado à respectiva variável. O valor desta variável só será ecoado na resposta caso seja incluso um comando “return”.

O banco de dados do chatterbot

Como toda aplicação web sofisticada, o programa-o possui um banco de dados relacional.

Contudo, este banco de dados não está normalizado, principalmente na tabela de “aiml”, pois possibilita a inserção de dois ou mais modelos exatamente iguais, por possuir na mesma linha tanto o padrão como o modelo, não satisfazendo nem mesmo a primeira forma normal. Por este motivo, as demais formas nem mesmo serão analisadas e expostas neste trabalho.

Este acontecimento é infelizmente comum de ser encontrado em aplicações web, pois uma grande maioria de programadores ou não possuem o conhecimento técnico necessário para aperfeiçoar-se um banco de dados ou nem sequer se preocupam com ele.

Além do mais, não foi utilizada chave estrangeira em nenhuma das tabelas originais do programa-o, o que facilitaria em muito o impedimento da repetição de valores e aceleraria em muito o seu desempenho. Uma aplicação útil seria utilizar-se a tag “srai” como um localizador de chaves estrangeiras. Caso queira-se alterar todos os padrões que apontam para uma mesma sintaxe de modelo, bastaria alterar-se um único.

Por conseguinte, Elisabeth Pearrau fez o programa da maneira mais simples possível, possibilitando não especialistas a modificarem seu programa fonte e seu aprimoramento. Além do mais, caso fosse necessário alterar-se alguma categoria se a mesma fosse normalizada, haveria dificuldades, ainda mais se duas categorias tivessem modelos iguais e posteriormente houvesse a necessidade de diferenciá-las. A leitura das mesmas também seria trabalhosa.

Falsa normalização através de recursão

A falha de normalização acima pode ser minimizada através da tag srai. Como dito na parte técnica da linguagem aiml, usando-se da detecção de palavras-chave.

Definem-se então várias categorias contendo padrões de referência, utilizando-se de palavras-chave, fazendo estes a função de tabela normalizadora. Então se declaram várias outras categorias, apontando para estes padrões de referência fazendo esta falsa normalização.

Este procedimento pode aumentar a quantidade de linhas no banco de dados ao invés de diminuir, mas faz a função de facilitar a manutenção.

Uma das aplicações desta normalização foi feita no robô Miracle através da palavra chave “OQUEE”.

Assim sendo, as frases “o que é”, “o que significa”, “o que pode ser”, “qual o significado”, entre outras, pontam para a categoria “OQUEE”, normalizando, assim, parte da procura. Uma frase programada com “OQUEE chatterbot” pode responder várias outras perguntas similares.

Correção gramatical

A correção gramatical pode ser efetuada tanto através da tag srai como automaticamente pelo programa. Automaticamente é menos trabalhoso e mais eficiente, para tanto, basta incluir ou modificar as linhas na tabela spellcheck.

Os dados desta tabela serão modificados exatamente na maneira que forem inseridas, ou seja, ao colocar-se o pronome “tu” para substituir para “você”, ao usuário digitar “Tatuapé”, o bot entenderá “Tavoceapé”, pela falta dos espaços no início ou o final da palavra, evitar ao máximo utilizar lexemas sem os devidos espaços! Este problema é minimizado pela utilização da tag srai, mas a mesma aumentará, em muito, a quantidade de categorias da programação do robô.

Inserção automática de categorias durante a instalação

Para facilitar a instalação do robô em um novo ambiente, o mesmo deve conter todas as suas categorias em um arquivo SQL. Um arquivo SQL contém todos os comandos SQL separados pelo delimitador “;”, possibilitando assim executar vários comandos através de um único arquivo.

Para fazer com que este seja compatível na hora da instalação do robô, deve-se pegar uma versão já operacional e instalada do conteúdo do banco de dados relacionado aos arquivos AIML.

Assim sendo, podem ser utilizadas diversas ferramentas para a extração de seu conteúdo. Neste trabalho, por exemplo, foi utilizado o programa MySQL Workbench.

Todo o conteúdo das categorias fica armazenado dentro do banco de dados selecionado para o robô na tabela “aiml”, sendo que esta contém o texto total da categoria, o padrão e modelos após o processamento do programa, o conteúdo de “that” e também o tópico. Além disso, contém também o nome do arquivo no qual foram processadas as categorias.

Os únicos detalhes são que este arquivo não deve possuir comentários, deve conter um único comando por linha e sem separação de um comando em diversas linhas.

Quanto ao nome da tabela a ser inserido nos padrões, insira somente o nome da tabela sem o nome do banco de dados, pois o programa se encarregará de inseri-lo no devido momento.

Estes arquivos podem ser divididos em partes para facilitar a instalação do robô e sua manutenção. Todos os arquivos devem ser instalados na pasta aiml_sql. É recomendável manter-se somente os arquivos com extensão “aiml” nesta pasta, sem os respectivos arquivos de “backup”, pois o programa processará todos os arquivos desta pasta durante a instalação, sem distinção da extensão ou do conteúdo do arquivo. Repetições ou arquivos não SQL ocasionarão uma má instalação.

Caso deseje inserir dados para outras tabelas, dentro do mesmo banco de dados, pode ser utilizada a mesma pasta com arquivos SQL diferentes. Este procedimento foi utilizado para a inserção da tabela de correção gramatical “spellcheck”.

Caso deseje criar outras tabelas ou efetuar operações que não sejam para a inserção de dados, deverá ser editado o arquivo “install_programo.php” e inserido o seu conteúdo conforme o exemplo das outras tabelas inseridas no arquivo. Este processo é um pouco mais trabalhoso, mas possibilita uma maior segurança e maior robustez na instalação. O processo descrito foi efetuado com as tabelas do cadastro de usuários (“book_users”) e de links (“book_links”).

Arquivos AIML

Na parte de XML, foi demonstrado que todos os arquivos XML devem conter um cabeçalho contendo o tipo do esquema utilizado e sua codificação de caracteres.

A codificação de caracteres é extremamente importante por tratar-se de dizer ao programa-o como proceder quanto aos acentos e caracteres especiais. Uma orientação de codificação errada pode, com toda a certeza, arruinar completamente um chatterbot, pelo mesmo não reconhecer o que o usuário digita ou que o mesmo observe uma resposta estranha.

Para tanto, ao utilizarem-se editores de texto como o Notepad++, que é talvez um dos melhores para esta finalidade, é imprescindível salvar o arquivo com a codificação “UTF-8”, correspondente ao formato UNICODE. Quanto ao conjunto de caracteres, pode-se utilizar qualquer um, pois o SGBD MySQL se encarregará de convertê-lo para o formato correto, sendo este o formato Europeu ISO 8859-1.

Ao criarem-se as categorias, não utilize acentos nos padrões, pois caso o usuário não os utilizar ao formular-se a entrada, o mesmo não será encontrado. Removendo-se os acentos através das substituições na entrada de dados, satisfazem-se as duas situações. Como os modelos serão apenas ecoados no vídeo para o usuário, dentro da interface de diálogo do robô na esmagadora maioria dos casos, os acentos preferencialmente devem ser utilizados.

CONCLUSÃO

Foram estudadas e testadas todas as possibilidades sobre como fazer um chatterbot exercer funções além de apenas conversar com um usuário. Foi concluído que esta tarefa é perfeitamente possível atualmente com os recursos e programas já disponíveis, necessitando apenas de uma bela preparação e planejamento, e também de algum esforço e empenho.

A linguagem AIML é perfeita para esta união de funções, por possibilitar a chamada de rotinas através da tag <system> executando as funções desejadas.

O exemplo utilizado nesse trabalho realiza a simples tarefa de cadastrarem-se usuários, guardarem-se bookmarks, abri-los para o usuário e também responder perguntas de pesquisa através da Wikipédia.

O programa-o da maneira que foi concebido não possibilitaria a realização destas funções, sendo necessário modifica-lo drasticamente em algumas partes, e também de adapta-lo à língua portuguesa.

O mesmo ainda possui algumas falhas a serem sanadas, por exemplo, ao se inserir a senha, a mesma não aparece para o usuário, mas aparece quando o administrador pede para ver o log da última conversa.

Além do mais, seria necessária uma revisão em todos os quesitos de segurança da informação, pois o mesmo ainda não foi devidamente testado.

Quanto ao modelo textual da Miracle, este também precisa ser aprimorado, tanto para corrigir eventuais falhas do programa como para incrementar a sua capacidade de conversação. Esta talvez seja a tarefa mais trabalhosa, por demandar um maior tempo e uma sucessão de testes.

O publicar-se na web uma versão deste programa, ficou muito mais fácil identificarem-se as falhas ou ausências de categorias, pelo motivo de a língua portuguesa permitir uma vasta flexibilidade de uso da linguagem e também de cada usuário possuir diferente nível de cultura, idade e ideologias.

Foi indefinido também o tema principal e a personalidade do robô, deixando-a para os futuros programadores. A maior preocupação deste trabalho não foi de fazer-se um robô que conversasse por horas com um usuário, mas sim abrir caminho para que outros o façam e que possuam ferramentas para utilizar algo além do simples conversar.

Seria muito interessante fazer um programa novo utilizando completamente orientação a objetos, pois o Programa-O original não é, possibilitando assim uma integração muito maior com gerenciadores de conteúdo e outros programas já existentes, como o Joomla e o Wordpress, por exemplo.

As definições teóricas sobre os outros assuntos foram apenas superficiais, mas já podem expor aos leitores sobre as preocupações ao se projetar um robô.

REFERÊNCIAS

1. WALLACE, Dr Richard S. - **The Elements of AIML Style** – 28 de março de 2003 – PDF em inglês;

2. SILVA, Jeferson Luís da **- Chatterbots – Simuladores de Diálogo** – 2ª revisão – Bookess Editora – Impresso e em PDF-;

3. RUSSEL, Stuart e Peter Norvig - **Inteligência Artificial** – Tradução da Segunda Edição – Editora Campus – 2004;

4. WELLING, Luke & Thompson, Laura - **PHP & MySQL Desenvolvimento Web** – Tradução da Terceira Edição – Luke Welling & Laura Thomson– Editora Campus – 2005 – Impresso e PDF em inglês;

5. WALLACE, Dr Richard S. - **AIML 1.0 Tag Sets** – Disponível em: www.alicebot.org/committees/architecture/resolutions/aiml10.html/ - HTML em inglês;

6. WALLACE, Dr Richard S. - **AIML 1.0.1 tag set** – Disponível em: http://www.alicebot.org/documentation/aiml101.html - HTML em inglês;

7. PLANTEK, Peter M. & Ray Kurzwell- **Virtual Humans: A Build-It-Yourself Kit, Complete with Software and Step-by-Step Instructions** – Editora Amacon – Impresso em inglês;

8. SESHAGIRI, Anupama **Basic AIML User Manual -**– Disponível em: http://pt.scribd.com/doc/46003752/Anupama-AIML-User-Manual - PDF em inglês;

9. CONVERSE, Tim & Joyce Park & Clark Morgan– **PHP5 and MySQL Bible** - and Wiley Publishing – Abril 2004 – PDF em inglês e impresso em português;

10. **Manual do PHP** – Disponível em www.php.net/download-docs.php/ em inglês e português;

11. PITTS-MOULTIS Natanya & Cheryl Kirk - **XML Black Book** – The Coriolis Group – 11 de janeiro de 1998 – CD-ROM HTML em inglês;

12. **XML Tutorial** – Disponível em http://www.w3schools.com/xml/default.asp com respectivos links de roteiro – HTML em inglês;

13. **Descrição de cookies persistentes e por seção no Internet Explorer** – Disponível em: http://support.microsoft.com/kb/223799/PT em português.

PROGRAMAS UTILIZADOS

Easy-PHP 5.3.5.0 – Servidor Web Apache, SGBD MySQL, Interpretador PHP integrados - Disponível em www.easyphp.org/;

MySQL Workbench 5.2 CE – Ferramenta de administração, modelagem e query de Banco de dados MySQL – Disponível em http://wb.mysql.com/;

Aptana Studio 3 – IDE para desenvolvimento PHP – Disponível em www.aptana.com/studio/;

Notepad++ - Editor de textos utilizado para programação em diversas linguagens – Disponível em notepad-plus-plus.org/;

Program-O – um interpretador de linguagem AIML para PHP e MySQL – Disponível em http://www.program-o.com/;

Simple AIML Editor – um editor de arquivos AIML, simples, mas muito útil – Disponível em http://riotsw.com/sae.html.

ANEXOS

AIML TAG REFERENCE TABLE

AIML 0.9	AIML 1.0	Tag Type	Note
<alice>	<aiml>	AIML block delimeter	[Closing tags not shown]
<name/>	<bot name="name"/>	Built-in bot parameter	may appear in pattern
(see Note 2.)	<bot name="XXX"/>	Custom bot parameter	<srai>BOT XXX</srai>
<justbeforethat/>	<that index="2,1"/>	Built-in predicate	See Note 4.
<that/>	<that index="nx,ny"/>	Built-in predicate	default "that"
<that>	<that>	AIML that pattern	contains AIML pattern
<category>	<category>	AIML category	
<justthat/>	<input index="2"/>	Built-in predicate	
<beforethat/>	<input index="3"/>	Built-in predicate	
<condition name="X" value="Y">	<condition name="X" value="Y">	Conditional branch	
<condition>	<condition>	Conditional branch	
<gender>	<gender>	Gender substitution	Exchange "he" and "she"
<date/>	<date/>	Built-in predicate	date and time
<get_ip/>	<id/>	Built-in predicate	default "localhost"
<getname/>	<get name="xxx"/>	Built-in predicate	default "X-person"
<getsize/>	<size/>	Built-in predicate	# of categories loaded
<star/>	<star index="n"/>	Built-in predicate	binding of *
<thatstar/>	<thatstar index="n"/>	Built-in predicate	binding of * in that
<gettopic/>	<get name="topic"/>	Built-in predicate	default "you"
<topicstar/>	<topicstar index="n"/>	Built-in predicate	binding of * in topic
<getversion/>	<version/>	Built-in predicate	AIML program version
<get_xxx/>	<get name="xxx"/>	Custom predicate	Botmaster defined XXX, default (3)
<gossip>	<gossip src="X"/>	Append to file	
<load filename="X"/>	<learn>X</learn>	AIML loading	
<li name="X" value="Y">	<li name="X" value="Y">	Conditional branch item	used by <condition>
<li value="Y">	<li value="Y">	Conditional branch item	used by <condition name="X">
		General list item	used by <random>, <condition>
<pattern>	<pattern>	AIML Pattern	contains AIML pattern
<person/>	<person/>	Prounoun transform macro	<person><get_star/></person>
<person2>	<person2>	Prounoun transform	swap 1st & 2nd person
<person2/>	<person2/>	Prounoun transform macro	<person2><get_star/></person2>
<person>	<person>	Prounoun transform	swap 1st & 3rd person
<random>	<random>	Random selection	Random uniform selection
<setname>	<set name="name">	Built-in predicate	returns contents
<settopic>	<set name="topic">	Built-in predicate	returns contents
<set_XXX>	<set name="XXX">	Custom predicate	See Note 3.
<sr/>	<sr/>	Recursion macro	<srai><get_star/></srai>

AIML 0.9	AIML 1.0	Tag Type	Note
<srai>	<srai>	Recursion	
<system>	<system>	Execute OS shell	platform-dependent
<template>	<template>	AIML template	
<think>	<think>	Nullify output	
<topic name="X">	<topic name="X">	AIML topic group	X is AIML pattern
	<uppercase>	Text manipulation	convert all text to Uppercase
	<lowercase>	Text manipulation	convert all text to Lowercase
	<sentence>	Text manipulation	capitalize the first word
	<formal>	Text manipulation	capitalize every word
	<if name="X" value=Y">	Conditional branch	
	<else>	Conditional branch	
	<javascript>	AIMLScript	Javascript

Fonte: 4 - AIML 1.0 Tag Sets