

**UNIVERSIDADE REGIONAL DE BLUMENAU**  
**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**  
**CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO**

**CHATTERBOT PARA AUXILIAR O USUÁRIO NO**  
**ATENDIMENTO AO PROCESSO DE ATUALIZAÇÃO DE**  
**SOFTWARE**

**CARLOS EDUARDO TRENTIN**

**BLUMENAU**  
**2016**

**CARLOS EDUARDO TRENTIN**

**CHATTERBOT PARA AUXILIAR O USUÁRIO NO  
ATENDIMENTO AO PROCESSO DE ATUALIZAÇÃO DE  
SOFTWARE**

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Alexander Roberto Valdameri, Mestre - Orientador

**BLUMENAU  
2016**

# **CHATTERBOT PARA AUXILIAR O USUÁRIO NO ATENDIMENTO AO PROCESSO DE ATUALIZAÇÃO DE SOFTWARE**

Por

**CARLOS EDUARDO TRENTIN**

Trabalho de Conclusão de Curso aprovado  
para obtenção dos créditos na disciplina de  
Trabalho de Conclusão de Curso II pela banca  
examinadora formada por:

Presidente:	<hr/> Prof. Alexander Roberto Valdameri, Mestrado – Orientador, FURB
Membro:	<hr/> Prof. Roberto Heinzle, Doutor – FURB
Membro:	<hr/> Prof. Matheus Carvalho Viana, Doutor – FURB

Blumenau, 7 de Julho de 2016

## **AGRADECIMENTOS**

Agradeço a meus pais, Eneide Trentin e Antônio Carlos Trentin, que sempre se dedicaram ao máximo para proporcionar o melhor estudo para a sua família.

Ao meu orientador Alexander Roberto Valdameri pela dedicação e empenho em proporcionar seu tempo para me auxiliar a desenvolver este trabalho.

## **RESUMO**

A pesquisa descrita neste trabalho apresenta o desenvolvimento de um chatterbot para auxiliar o usuário no atendimento ao processo de atualização de versão de software. O usuário pode interagir com o chatterbot e sanar suas dúvidas. A partir de uma página web. Visando facilitar a interação, caso o usuário pergunte alguma informação que o chatterbot não consiga responder, é iniciado um diálogo com hiperlinks. Para a criação de hiperlinks, o interpretador responsável consulta a base de conhecimentos e faz uso de tags específicas para montar as respostas. Todas as possibilidades de respostas que o chatterbot pode fornecer está na base de conhecimentos desenvolvida utilizando AIML. Devido a não existir uma ferramenta que auxilie na criação de base de conhecimentos AIML, a criação teve que ser feita manualmente, necessitando de um grande tempo e esforço para a mesma. Também, foi perceptível a dificuldade de respostas para erros de digitação, necessitando a criação de vários padrões de domínios semelhantes para as respostas. Apesar disso, os resultados apresentaram indicativos de que os objetivos foram alcançados, uma vez que foi possível a interação do usuário com o chatterbot para auxiliar o atendimento ao processo de atualização de versão de software.

Palavras-chave: Chatterbot. Hiperlink. AIML.

## **ABSTRACT**

The research described in this paper presents the development of a chatterbot to assist the user in the care process software version upgrade. The user can interact with the chatterbot and answer your questions. From a web page. To facilitate the interaction, if you ask some information that chatterbot can not answer, it starts a dialogue with hyperlinks. For creating hyperlinks, the official interpreter query the knowledge base and makes use of specific tags to assemble the answers. All possible answers that chatterbot can provide is the knowledge base developed using AIML. Because there is a tool to assist in creating knowledge base AIML, the creation had to be done manually, requiring a great effort and time to it. Also, the difficulty of answers to typos was noticeable, requiring the creation of various patterns of similar domains to the answers. Nevertheless, the results presented indicate that the objectives were achieved, since it was possible user interaction with chatterbot to assist the care process software version upgrade.

Key-words: Chatterbot. Hiperlinks. AIML.

## LISTA DE FIGURAS

Figura 3 - <i>Chatterbot</i> Andrew .....	18
Figura 4 - Arquitetura Andrew .....	19
Figura 5 - Interação com o usuário .....	20
Figura 6 - Variáveis iAIML.....	21
Figura 7 - Arquitetura da proposta apresentada por Corrêa (2010).....	22
Figura 8 - Diagrama de casos de uso .....	26
Figura 9 - Resultado da <i>Tag</i> <hyperlink>, <option> e <header> na apresentação ao usuário.....	28
Figura 10 - Resultado da <i>tag</i> <query> na apresentação ao usuário.....	29
Figura 11 - Resultado da <i>query</i> <img> ao usuário .....	30
Figura 12 - Diagrama de pacotes .....	31
Figura 13 - Diagrama de classes pacote abextensions .....	31
Figura 14 - Diagrama de classe do pacote controller.....	32
Figura 15 - Diagrama de classes do pacote dao .....	33
Figura 16 - Diagrama de classes do pacote model .....	34
Figura 17 - Diagrama de classe do pacote session .....	35
Figura 18 - Diagrama de classe pacote utils.....	35
Figura 19 - Diagrama de atividades.....	37
Figura 20 - Arquitetura da solução .....	38
Figura 21 - Diálogo entre o <i>chatterbot</i> e o usuário.....	46
Figura 22 - Diálogo entre o <i>chatterbot</i> e o usuário utilizando hiperlinks.....	47
Figura 23 – Diálogo usuário responsável pelo suporte a atualização de versão de software ...	58
Figura 24 - Diálogo gestora da área que participa do suporte a atualização de versão .....	59

## LISTA DE QUADROS

Quadro 1 - Classificação evolutiva seguindo tecnologias aplicadas .....	14
Quadro 2 - Base de uma categoria em AIML.....	15
Quadro 3 – Utilizando variáveis e recursividade.....	17
Quadro 4 - Características dos trabalhos .....	22
Quadro 5 - Requisitos funcionais .....	24
Quadro 6 - Requisitos não funcionais .....	25
Quadro 7 - Exemplo de utilização da <i>tag</i> <header> .....	27
Quadro 8 - Exemplo de utilização da <i>tag</i> <option>.....	27
Quadro 9 - Exemplo de utilização da <i>tag</i> <hiperlink>.....	28
Quadro 10 - Exemplo de utilização da <i>tag</i> <query> .....	29
Quadro 11 - Exemplo de utilização da <i>tag</i> <img> .....	30
Quadro 12 - Padrão de reconhecimento para a falta da criação de um campo durante o processo de atualização de versão .....	40
Quadro 13 - Padrão de reconhecimento para orientação sobre a mensagem <i>socket</i> .....	40
Quadro 14 - Padrão para montagem de menu com possibilidades de diálogo atualização de versão Delphi .....	41
Quadro 15 - Base de conhecimento para iniciar o diálogo com o usuário .....	41
Quadro 16 - Classe <i>HiperlinkExtension</i> método <i>hiperlink</i> .....	42
Quadro 17 - Classe <i>QueryExtension</i> método <i>query</i> .....	43
Quadro 18 - Método <i>executeQuery</i> da classe <i>QueryDao</i> .....	43
Quadro 19 - Método <i>executeSQL</i> da classe <i>Dao</i> .....	44
Quadro 20 - Método <i>getConnection</i> da classe <i>ConnectionBase</i> .....	45
Quadro 21 - Quadro comparativo entre a pesquisa e os trabalhos correlatos.....	49
Quadro 22 - Caso de uso: Dialogar com o <i>chatterbot</i> através de perguntas .....	55
Quadro 23 - Caso de uso: Consultar o histórico da conversa com o <i>chatterbot</i> .....	55
Quadro 24 - Caso de uso: Dialogar com o <i>chatterbot</i> através de hiperlinks .....	55
Quadro 25 - Bases de conhecimentos e seu propósito.....	56
Quadro 26 - Perguntas realizadas aos usuários .....	57
Quadro 27 - Respostas usuário responsável pelo suporte a atualização de versão.....	57
Quadro 28 - Resposta gestora da área que participa do suporte a atualização de versão .....	58



## LISTA DE ABREVIATURAS E SIGLAS

AIML - *Artificial Intelligence Markup Language*

ALICE - *Artificial Linguistic Internet Computer Entity*

FAQ - *Frequently Ask Questions*

FURB - Universidade Regional de Blumenau

JSP – *Java Server Pages*

RF – Requisitos Funcionais

RNF – Requisitos não funcionais

SGML - *Standard Generalized Markup Language*

SO – Sistema Operacional

WEB - *World Wide Web*

XML - *eXtensible Markup Language*

# SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>11</b>
1.1 OBJETIVOS.....	12
1.2 ESTRUTURA.....	12
<b>2 FUNDAMENTAÇÃO TEÓRICA .....</b>	<b>13</b>
2.1 ATUALIZAÇÃO DE SOFTWARE.....	13
2.2 CHATTERBOT.....	14
2.3 ARTIFICIAL INTELLIGENCE MARKUP LANGUAGE.....	15
2.4 TRABALHOS CORRELATOS .....	17
2.4.1 Chatterbot para esclarecimento de dúvidas sobre as formas de ingresso em cursos da FURB .....	18
2.4.2 Integrando chatterbot e agente animado de interface em um ambiente virtual de aprendizagem .....	19
2.4.3 Robô de conversação aplicado a educação a distância como tutor inteligente .....	20
2.4.4 Quadro comparativo entre os trabalhos correlatos .....	22
<b>3 DESENVOLVIMENTO.....</b>	<b>24</b>
3.1 REQUISITOS PRINCIPAIS DA SOLUÇÃO .....	24
3.2 ESPECIFICAÇÃO .....	25
3.2.1 Casos de uso.....	25
3.2.2 Especificação de <i>tags</i> AIML para manipulação de dados .....	26
3.2.3 Diagrama de pacotes e de classes.....	30
3.2.4 Diagrama de atividades .....	36
3.2.5 Arquitetura proposta.....	37
3.3 IMPLEMENTAÇÃO .....	38
3.3.1 Técnicas e ferramentas utilizadas.....	38
3.3.2 Base de conhecimentos AIML .....	39
3.3.3 <i>Tag</i> para criação de <i>hiperlink</i> a partir da base de conhecimentos AIML.....	41
3.3.4 <i>Tag</i> para execução de consultas a base de dados a partir da base de conhecimentos AIML .....	42
3.3.5 Executando consultas a base de dados .....	43
3.3.6 Operacionalidade da implementação .....	45
3.4 RESULTADOS E DISCUSSÕES.....	47

<b>4 CONCLUSÕES.....</b>	<b>50</b>
4.1 EXTENSÕES .....	51

## 1 INTRODUÇÃO

A evolução é uma característica da espécie humana, seja ela pessoal, corporativa, ou até mesmo, em situações inusitadas. Deste modo, o ser humano evolui e desenvolve novas formas de pensar ou realizar determinada ação. Mas para essa evolução, sempre foi necessária uma pesquisa, um embasamento, algo em que fosse possível acreditar. Na maioria dos casos, isso é feito com informações geradas ou adquiridas a partir de variadas fontes.

O surgimento da *World Wide Web* (Web) no início dos anos 90, ficou marcado como um divisor de águas no pensamento sobre a informação. Segundo Branski (2003), estão disponíveis na Web milhares de páginas cobrindo os mais variados assuntos e interesses. Estima-se que, em outubro de 2013, eram 105 milhões de usuários ativos na Internet (ASSENSIO, 2013). No ano passado, a soma chegava a 110 milhões de usuários ativos (GIACOMELI, 2015). Esse número de usuários ativos gera inúmeros documentos. Mas, diferentemente de bibliotecas onde existe uma organização para cada assunto e interesse, a Web é um oceano de informações, sem classificação, sem seguir um padrão.

Com a evolução e expansão da Web, começou-se a utilizar a mesma como forma de sanar dúvidas sobre produtos. Como por exemplo, o *Frequently Ask Questions* (FAQ), onde são realizados cadastros de perguntas e respostas das dúvidas mais frequentes dos usuários. Mas, segundo Teixeira (2005), o FAQ só funciona bem quando a dúvida do usuário é básica e se refere a um domínio do conhecimento pequeno. Além disso, o FAQ é estático, restrito e requer leitura de todas as perguntas e respostas até encontrar a resposta que interessa, sendo que, às vezes, a resposta desejada não é encontrada.

Uma alternativa que pode ser inserida na modalidade de suporte são os *chatbots*. Segundo Teixeira e Menezes (2003), *chatbot* é um programa de computador que tenta simular um ser humano na conversação com as pessoas. O objetivo é responder às perguntas de tal forma que os usuários tenham a impressão de estar conversando com outra pessoa e não com um programa de computador. Deste modo visa sanar as dúvidas de forma rápida e assertiva, ampliando assim o horizonte, até então, restrito ao FAQ para suporte ao usuário.

Um software para gestão hospitalar pode fornecer a possibilidade de gestão de um hospital ou operadora. Se tratando de um software que está diretamente ligado a seres humanos, cada segundo de indisponibilidade pode afetar uma vida. O processo de atualização do software pode ocasionar esta indisponibilidade, caso durante o processo seja apresentada alguma inconsistência ou ação errada por parte do usuário.

Diante do exposto, o *chatterbot* desenvolvido visa auxiliar o usuário no atendimento ao processo de atualização de versão de software. A partir disso, busca-se minimizar uma possível indisponibilidade do software neste processo.

## 1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um *chatterbot* para dialogar com usuários acerca de dúvidas sobre o processo de atualização de versão de software.

Tem-se ainda como objetivos específicos:

- a) possibilitar que a interação ocorra através da língua portuguesa – brasileira;
- b) disponibilizar uma interface web para interação entre o usuário e o *chatterbot*;
- c) possibilitar o incremento da base de conhecimento em AIML.

## 1.2 ESTRUTURA

Esta monografia está organizada em capítulos. No primeiro capítulo é apresentada a introdução ao assunto da pesquisa e descritos os objetivos do trabalho. O segundo capítulo descreve a fundamentação teórica das tecnologias utilizadas para desenvolvimento da solução. É apresentado um conceito sobre *chatterbot* e a tecnologia utilizada para criação da base de conhecimentos. O terceiro capítulo descreve o desenvolvimento da solução proposta, que inclui os requisitos propostos, as ferramentas e técnicas utilizadas, assim como a operacionalidade do *chatterbot*. Por fim, o quarto capítulo descreve as conclusões sobre a pesquisa e apresentada alternativas para desenvolvimento de trabalhos futuros relacionados a este.

## 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos e técnicas envolvidos na pesquisa. A Seção 2.1 apresenta o processo de atualização de softwares. A Seção 2.2 apresenta o conceito sobre *chatbot* e suas características na visão de Franklin e Graesser (1996). Na Seção 2.3 é apresentado o conceito da linguagem AIML. Por fim, na Seção 2.4 são apresentados trabalhos correlatos ao assunto abordado nesta pesquisa.

### 2.1 ATUALIZAÇÃO DE SOFTWARE

Atualização de versão ou *upgrade*, segundo 7Graus (2011), significa em computação, ir para uma versão mais recente de um determinado produto. *Upgrade* é muito utilizado na área da informática e de equipamentos eletrônicos e consiste na troca de um hardware, um software ou um firmware, por uma versão melhor e mais recente. Com isso, disponibilizando mais funcionalidades ou melhorando os sistemas já existentes.

As atualizações mais conhecidas no mercado e mais frequentes realizadas por usuários, são em celulares ou em Sistemas Operacionais (SO). Como exemplo, destaca-se a empresa Apple, onde segundo Simões (2014), possui 90% de atualização de seu SO iOS após disponibilização. Já, seu concorrente, o SO desenvolvido pela Google de código aberto, Android, conta apenas com 10% de atualização após uma disponibilização de atualização do SO. Isto identifica que, devido à forma que é desenvolvido o software, seja para qualquer finalidade, a cada nova alteração se torna mais difícil sua atualização e os usuários acabam deixando de atualizar seus softwares e hardwares.

Recomenda-se sempre manter os softwares atualizados. Um software desatualizado pode conter vulnerabilidades, permitindo assim, a invasão por usuários maliciosos. Essa invasão coloca em risco seus dados, sua privacidade e até vidas de pessoas. Como exemplo, a utilização do SO Windows XP, que mesmo após a empresa responsável pelo software não prestar mais suporte sobre o mesmo, ainda é utilizado nas máquinas de usuários. Um reflexo da vulnerabilidade que isso ocasiona é o ataque sofrido em um hospital na Austrália, onde alguns programas foram afetados por um vírus devido a vulnerabilidades do SO Windows XP. Setores do hospital tiveram que deixar de utilizar o software. Devido ao ataque sofrido (MELBOURNE, 2016).

## 2.2 CHATTERBOT

A concepção dos *chatbot* teve seu início pelo matemático Turing (1950), que levantou a seguinte questão: as máquinas podem pensar? A partir desta pergunta, se deu início a busca por tarefas que uma máquina poderia realizar.

Segundo Comarella e Café (2008), *chatbots* são programas que simulam uma conversa similar à de seres humanos. São utilizados para os mais diversos propósitos, desde simular uma conversa com um amigo, até para uso comercial. O objetivo é responder às perguntas de tal forma que as pessoas tenham a impressão de estar conversando com outra pessoa e não com um programa de computador. Após o envio de perguntas em linguagem natural, o programa consulta uma base de conhecimento e em seguida fornece uma resposta que tenta imitar o comportamento humano.

Segundo Franklin e Graesser (1996), agentes autônomos podem ser categorizados como:

- a) reatividade: consegue se adaptar a mudanças do ambiente;
- b) autônomo: tem controle sobre suas próprias ações;
- c) orientado a objetivo: não se limita a seus conhecimento, consegue evoluir e exercer ações além do proposto;
- d) continuamente temporário: exerce um tempo contínuo de funcionamento;
- e) comunicativo: se comunica com outros agentes, incluindo pessoas;
- f) aprendizagem: possibilita mudança de comportamento proveniente de experiências passadas;
- g) móvel: capaz de se mover para outra máquina;
- h) flexível: capaz de exercer ações que não estão pré-definidas;
- i) personagem: possui caráter emocional e personalidade.

Um *chatbot* não precisa ter todas as categorias citadas para ser considerado um agente. Moura (2003) propõe uma classificação seguindo a evolução das tecnologias utilizadas. O Quadro 1 apresenta as gerações e sua característica.

Quadro 1 - Classificação evolutiva seguindo tecnologias aplicadas

Geração	Característica
1ª Geração	Casamento de padrão e uso de regras gramaticais
2ª Geração	Técnicas de Inteligência Artificial
3ª Geração	Linguagens de Marcação

Fonte: Moura (2003).

Moura (2003) gerou o Quadro 1 comparando épocas e *chatbots*. Como exemplo de *chatbot* da primeira geração, tem-se o Eliza, criado por Weizenbaum (1966), com pouco

mais de 200 linhas. Eliza recebe um dado digitado pelo usuário, compara com os dados em uma lista de padrões, onde é verificado o padrão mais semelhante ao digitado, analisa a sentença com base em regras gramaticais de decomposição e devolve respostas com regras de construção de frases associadas às regras de decomposição.

Na segunda geração, pode-se citar o *chatterbot* Julia, desenvolvido por Mauldin (1994). Julia foi desenvolvido para auxiliar o usuário durante um jogo, onde a mesma é um personagem. Internamente, Julia possui um modelo interno do mundo que pode ser atualizado no momento em que a conversa está se desenvolvendo. Deste modo, é capaz de contextualizar o diálogo, gerando um realismo maior nos seus diálogos.

Na terceira geração, Wallace (2001) apresenta o *Chatterbot Artificial Linguistic Internet Computer Entity* (ALICE), que utiliza linguagem natural composta por módulos de conversação, dividido por idade, gênero, localização geográfica e profissão. ALICE também armazena o nome do usuário e o tópico da conversa, fazendo com que não mude de assunto aleatoriamente e consiga ser mais clara e direta nas respostas sobre o assunto.

### 2.3 ARTIFICIAL INTELLIGENCE MARKUP LANGUAGE

A linguagem *Artificial Intelligence Markup Language* (AIML) é uma linguagem baseada em módulos ou diálogos básicos que são chamados de categorias. Cada categoria possui um padrão de entrada que é a sentença digitada pelo usuário e, um padrão de resposta que será usado para montar a sentença a ser retornada ao usuário. Este padrão de resposta utiliza *tags* de marcação para a sua construção (MOURA, 2003).

AIML é uma linguagem de marcação baseada em *eXtensible Markup Language* (XML) para a escrita de *chatterbots*. Como o XML é uma forma restrita de *Standard Generalized Markup Language* (SGML), os objetos AIML também estão de acordo com documentos SGML (WALLACE, 2001). O Quadro 2 ilustra uma representação de AIML.

Quadro 2 - Base de uma categoria em AIML

1	<code>&lt;aiml&gt;</code>
2	<code>&lt;category&gt;</code>
3	<code>&lt;pattern&gt; padrão de entrada &lt;/pattern&gt;</code>
4	<code>&lt;template&gt; modelo para respostas ao usuário &lt;/template&gt;</code>
5	<code>&lt;/category&gt;</code>
6	<code>&lt;/aiml&gt;</code>

Observa-se no Quadro 2 um exemplo de codificação para a *tag* *category*, em que:

- nas linhas 1 e 6 tem-se a tag `<aiml>`, utilizada para determinar o início e o fim do documento AIML;
- nas linhas 2 até 6 é definida uma categoria, na qual `<pattern>` é o padrão de



entrada que pode vir digitado o usuário e `<template>` é a saída enviada para o usuário.

AIML também dispõe de *tags* para utilizar o valor digitado pelo usuário, permitir a criação de variáveis e para utilização de recursividade. Essa recursividade é diferente da conhecida em linguagens de programação, uma vez que a mesma é utilizada quando é encontrada a *tag* `<srai>`. Deste modo, o interpretador não retorna o *template* de saída diretamente. Ele entende o *template* como uma nova entrada, assim, repetindo o processo até não encontrar mais a *tag* e retornar o *template* de saída para o usuário (MARIETTO, 2013).

Observa-se no Quadro 3 um exemplo de codificação das *tags* `^`, `*`, `<srai>`, `<set>` e `<get>` que:

- a) nas linhas 3, 8 e 13 é utilizado o símbolo `^` (circunflexo) que indica que após o Oi, pode haver 0 ou mais palavras;
- b) nas linhas 4, 9 e 20 é utilizado a *tag* `<srai>` que, conforme mencionado no quadro 3, é utilizada para realizar uma recursividade dentro da base de conhecimento, deste modo, retornando apenas o *template* resultante quando não encontrar mais a *tag*;
- c) nas linhas 19 e 25 é utilizada a *tag* `<set>`, a qual é responsável pela criação de variáveis, onde o padrão a ser utilizado é `<set name="valor"> </set>`, sendo “valor” o dado armazenado;
- d) na linha 26 é utilizada a *tag* `<get>`, esta *tag* é utilizada para buscar valores de variáveis. A nomenclatura seguida para esta *tag* é `<get name="xxx"/>`, aonde “xxx” é o nome da variável.

Quadro 3 – Utilizando variáveis e recursividade

1	<aiml>
2	<category>
3	<pattern> Oi ^</pattern>
4	<template> <srai>Olá</srai> </template>
5	</category>
6	<category>
7	<pattern> Oe ^</pattern>
8	<template> <srai>Olá</srai> </template>
9	</category>
10	<category>
11	<pattern> Olá ^</pattern>
12	<template> Olá! Qual é seu nome? </template>
13	</category>
14	<category>
15	<pattern> Me chamo * </pattern>
16	<template> <set name="nome">*</set> </template>
17	<template> <srai>Meu nome é</srai> </template>
18	</category>
19	<category>
20	<pattern> Meu nome é * </pattern>
21	<template> <set name="nome">*</set> </template>
22	<template> Muito prazer Sr(a) <get name="nome"/>!</template>
23	</category>
24	</aiml>

Segundo Leonhardt (2005), quando o usuário digita uma pergunta, o interpretador AIML aplica um processo de normalização na sentença escrita. Este processo é composto de três etapas distintas: substituição, separação de sentenças e ajuste de padrão. A primeira etapa consiste em substituir algumas palavras ou caracteres de forma que seja possível reter informações que podem ser perdidas em outras etapas da normalização. A segunda etapa divide a sentença de entrada em tantas quanto forem necessárias, de acordo com a pontuação. A terceira etapa, de ajuste de padrão, remove a pontuação das sentenças e converte todas as letras para maiúsculo. Após a entrada ser normalizada, o interpretador busca a resposta na base de conhecimento. Teixeira (2005) relata que a base de conhecimentos em AIML guardam os <patterns> em uma estrutura de árvore gerenciada a partir de um algoritmo. Deste modo, os valores ficam armazenados em memória possibilitando uma rápida e eficiente busca de <patterns>.

## 2.4 TRABALHOS CORRELATOS

Esta seção tem como objetivo relatar trabalhos relacionados ao tema proposto. Entre os trabalhos identificados estão o trabalho desenvolvido por Oliveira (2015), de Machado (2005) e o robô de conversação de Corrêa (2010).

#### 2.4.1 Chatterbot para esclarecimento de dúvidas sobre as formas de ingresso em cursos da FURB

Oliveira (2015) propôs a implementação de um *chatterbot* que responde dúvidas sobre as formas de ingresso (para calouros) em cursos da Universidade Regional de Blumenau (FURB). Esse *chatterbot* possui uma área administrativa, onde é possível realizar o cadastro das informações utilizadas para auxílio nas respostas ao usuário. Para interação com o usuário foi desenvolvida uma página *web*. A cada pergunta realizada pelo usuário, o interpretador faz consulta a base de conhecimento desenvolvida utilizando AIML. O retorno da base de conhecimento pode possuir *tags* específicas, que foram desenvolvidas especificamente pelo autor, estas podem identificar a necessidade de uma consulta a ontologia para ajuda na construção da resposta ao usuário. A Figura 3 ilustra a página de interação do usuário com o *chatterbot*.

Figura 1 - Chatterbot Andrew

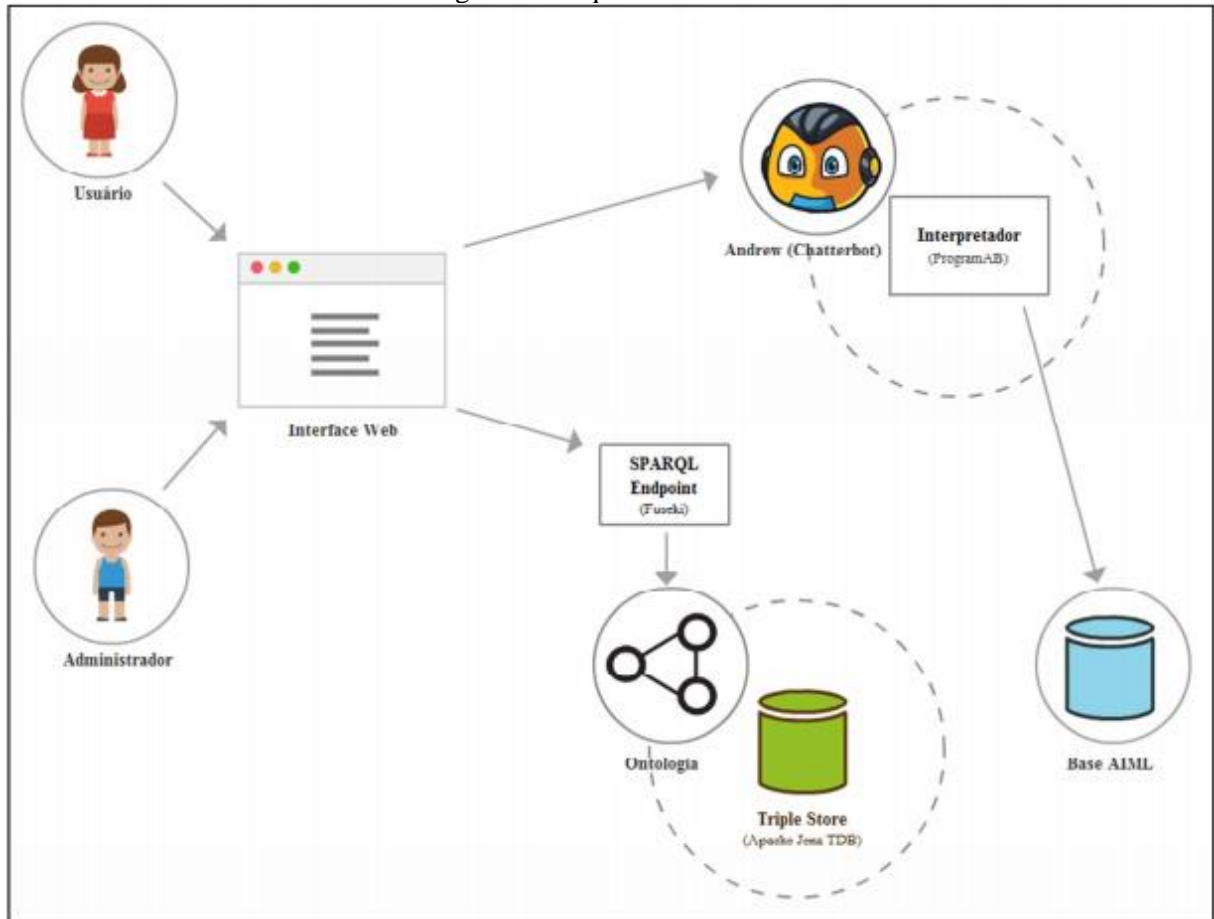


Fonte: Oliveira (2015).

Para desenvolvimento do *backend* da aplicação, foi utilizada a linguagem de programação Java. Para a base de conhecimento foi utilizada linguagem AIML, o interpretador utilizado para comunicação entre usuário e base de conhecimento foi o Program AB e uma Ontologia OWL armazenada em um *endpoint* SPARQL. O autor menciona que os objetivos do trabalho foram alcançados, apesar de mencionar dificuldade em realizar os testes devido a não existir uma ferramenta de automatização de testes. Também foi mencionada a dificuldade de criar a base de conhecimentos em AIML. Como não existe uma ferramenta

para criar a mesma, toda a criação foi feita manualmente. A Figura 4 exemplifica a arquitetura do *chatbot* Andrew.

Figura 2 - Arquitetura Andrew

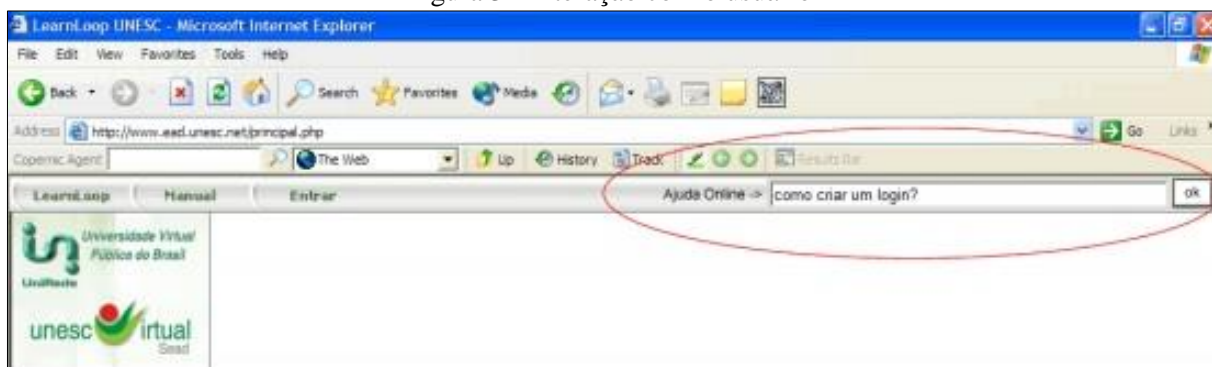


Fonte: Oliveira (2015).

#### 2.4.2 Integrando chatbot e agente animado de interface em um ambiente virtual de aprendizagem

Machado (2005) traz uma integração com o ambiente virtual de aprendizagem *Learnloop*. Nesta pesquisa, foi desenvolvido um *chatbot* para esclarecer dúvidas do usuário quanto à utilização das ferramentas de ambiente por meio de diálogos em linguagem natural. A partir de um campo criado na ferramenta *Learnloop* com o nome Ajuda Online, permite ao usuário realizar questionamentos. Para adição deste campo a ferramenta *Learnloop*, foi utilizada a linguagem de programação php. A Figura 5 ilustra o local onde é feita a interação do usuário com o *chatbot*.

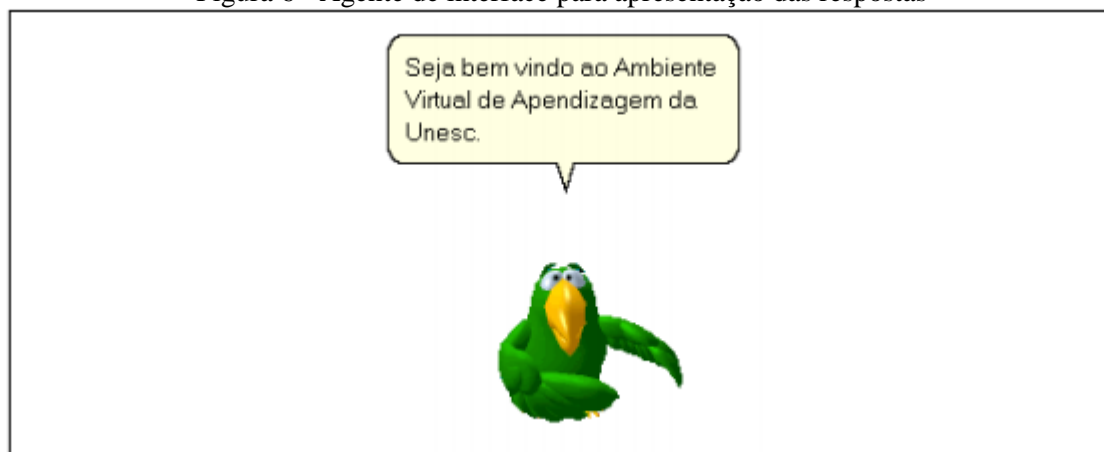
Figura 3 - Interação com o usuário



Fonte: Machado (2005).

A partir destes questionamentos, um interpretador desenvolvido em Java pelo autor se comunica com o interpretador Program D que, por sua vez, consulta a base de conhecimento desenvolvida em AIML. Para execução do *backend* da aplicação, foi utilizado o container Jetty. Para maior interação nas respostas com o usuário, um agente de interface da Microsoft exibe a resposta ao usuários. Deste modo, a partir de scripts, é possível fazer com que o agente executasse movimentos de acordo com a resposta. O agente utilizado no diálogo está ilustrado na Figura 6. Toda a interação entre usuário e *chatbot* é armazenada em logs para XML para posterior consulta e aprimoramento da base de conhecimento.

Figura 6 - Agente de interface para apresentação das respostas



Fonte: Machado (2005).

#### 2.4.3 Robô de conversação aplicado a educação a distância como tutor inteligente

Corrêa (2010) propôs a criação de um *chatbot* para aplicação na área da educação, com a possibilidade de servir como um tutor virtual inteligente de determinada disciplina. O *chatbot* está sempre disponível ao aluno, sendo que a aplicação seria para uma instituição de ensino a distância.

A arquitetura proposta por Corrêa (2010) consiste em: uma página desenvolvida em HTML para interação com o usuário; uma base de conhecimento em AIML para responder as sentenças; e uma segunda base de conhecimento para armazenar os históricos de interação entre o usuário e o visitante e posteriormente incrementar a base de conhecimento. O interpretador utilizado foi PyAIML desenvolvido para ser utilizado pela linguagem de programação Python. O autor fez a utilização da linguagem iAIML, que é a linguagem AIML com tratamento de intenção a cada categoria para obter o correto diálogo com o usuário. Aplicado o tratamento da tag *<condition>*, permitiu-se ao *chatterbot* identificar o momento da conversa com o usuário. Este momento pode ser: a abertura, em que o *chatterbot* está saudando o usuário; desenvolvimento, quando o *chatterbot* está desenvolvendo o diálogo com o usuário; ou, o fechamento, quando o *chatterbot* está em processo de conclusão do diálogo com o usuário. A Figura 6 apresenta três variáveis que foram utilizadas para o tratamento de intenção, estas são testadas a cada interação, afim de saber qual a intenção da conversa. A linguagem iAIML foi utilizada para desenvolvimento da base de conhecimento para aplicação da Teoria da Análise de Conversação que consiste em uma interação verbal entre um ou mais interlocutores, havendo aceitação ao tema pelos mesmos e um mínimo de conhecimento sobre o assunto a ser tratado, não sendo necessária a interação face a face (Marcuschi, 1991, p. 15).

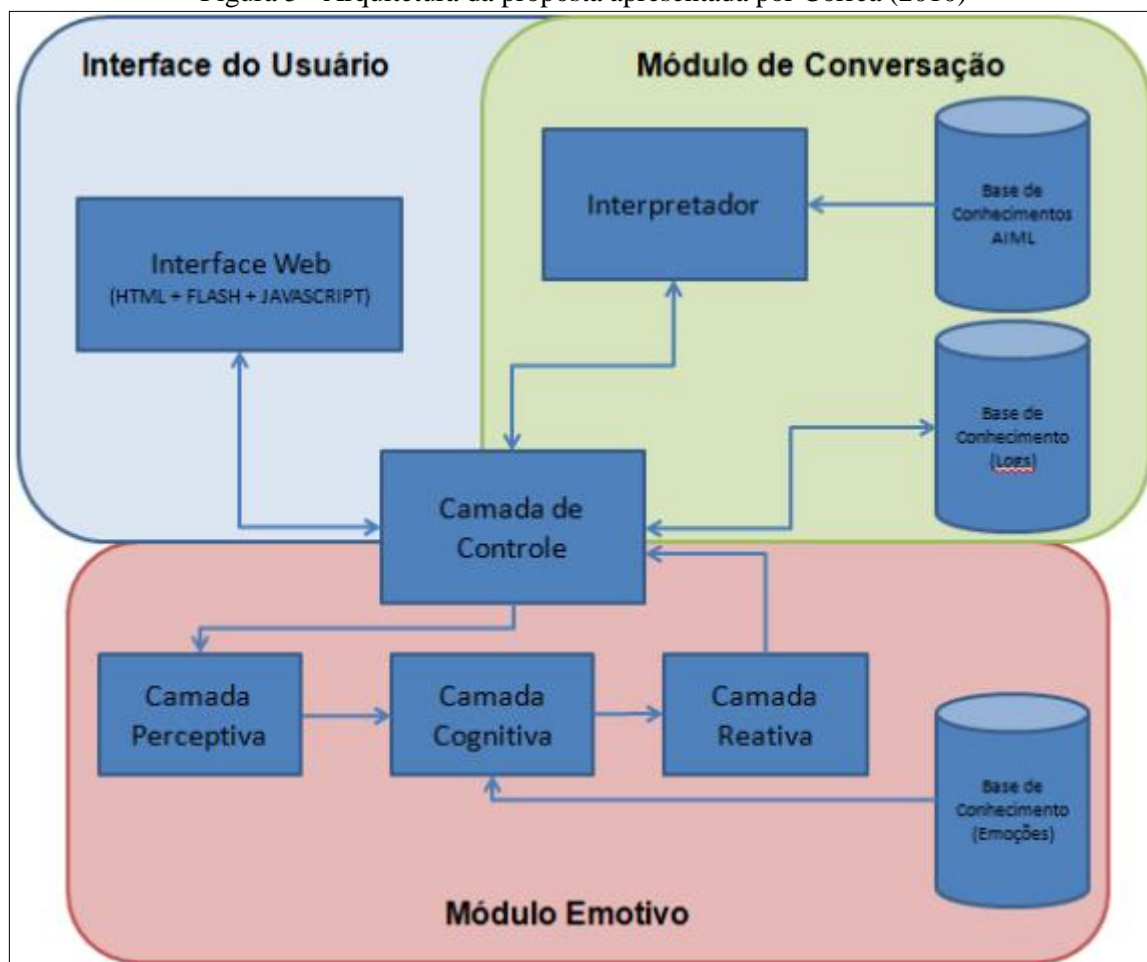
Figura 4 - Variáveis iAIML

```
<set name="session">abertura</set>
<set name="user_intention">saudar</set>
<set name="bot_intention">perguntar como esta se sentindo</set>
```

Fonte: Corrêa (2010).

Ainda, foi proposta uma terceira base de conhecimento para aplicação de uma Teoria da Análise de Conversação, em que o autor visa deixar a conversa mais humana. Para o tratamento de emoções foi utilizado três camadas. Camada Perceptiva, Cognitiva e a Reativa. A camada perceptiva tem a intuição de interagir com a base de conhecimento afim de compreender a emoção que o aluno quer passar com a frase. A camada cognitiva faz consultar a base afetiva para obter a emoção a ser passada, levando em consideração a emoção adquirida pela camada perceptiva. A camada reativa recebe a emoção da resposta obtida pela camada cognitiva e retorna a resposta com a emoção interpretada pelas três camadas para a camada de controle. A Figura 7 mostra a arquitetura proposta por Corrêa (2010).

Figura 5 - Arquitetura da proposta apresentada por Corrêa (2010)



Fonte: Corrêa (2010).

#### 2.4.4 Quadro comparativo entre os trabalhos correlatos

A partir das informações obtidas com os trabalhos apresentados nas Seções 2.6.1, 2.6.2 e 2.6.3, elaborou-se o Quadro 4 com as principais características de cada trabalho.

Quadro 4 - Características dos trabalhos

características / trabalhos relacionados	Oliveira (2015)	Machado (2005)	Corrêa (2010)
linguagem da base de conhecimento	AIML	AIML	AIML/iAIML
uso de ontologia	sim	não	não
linguagem de programação para página web	Java Web	PHP	Phyton
área administrativa	sim	sim	não
interpretador AIML	Program AB	Program D	PyAIML

A partir do Quadro 4 é possível identificar que todos os trabalhos utilizam a linguagem AIML como linguagem de desenvolvimento da base de conhecimento. Somente Corrêa (2010) utiliza iAIML para criação da base de conhecimento para tratamento de emoções.

Percebe-se que dois dos trabalhos possuem uma área administrativa para alteração de informações e cadastro para posterior consulta. A linguagem utilizada para o *front end* difere entre os trabalhos. Por fim, cada trabalho utiliza um interpretador diferente para comunicação com a base de conhecimento.



### 3 DESENVOLVIMENTO

Este capítulo tem como objetivo descrever os processos de desenvolvimento utilizados na pesquisa. A Seção 3.1 apresenta os requisitos funcionais e os não-funcionais. A Seção 3.2 apresenta a especificação da solução. A Seção 3.3 apresenta a implementação da solução, apresentando trechos do códigos e exemplos de diálogos do usuário com o *chatterbot*. Por fim, a Seção 3.4 apresenta os resultados e discussões da implementação.

#### 3.1 REQUISITOS PRINCIPAIS DA SOLUÇÃO

Os Requisitos Funcionais (RF) e os Requisitos Não Funcionais (RNF) propostos para a solução são apresentados nos Quadros 5 e 6. Para os RFs foi estabelecida a relação entre o requisito e o caso de uso correspondente.

Quadro 5 - Requisitos funcionais

Requisitos funcionais (RF)	Casos de Uso (UC)
RF 01: a página <i>web</i> de interação com o usuário deve permitir digitar e receber mensagens de respostas do <i>chatterbot</i>	UC 01
RF 02: permitir que o <i>chatterbot</i> responda perguntas sobre o processo de atualização de versão de software	UC 01
RF 03: a página <i>web</i> deve exibir o histórico da conversa para consulta durante o diálogo com o <i>chatterbot</i>	UC 02
RF 04: o <i>chatterbot</i> deve responder na língua portuguesa	UC 01
RF 05: permitir a interação com o usuário a partir de <i>hiperlinks</i>	UC 03
RF 06: o início da interação com o usuário a partir de <i>hiperlinks</i> deve ser a partir do momento que o <i>chatterbot</i> não possuir resposta para um diálogo	UC 03

Quadro 6 - Requisitos não funcionais

Requisitos não funcionais (RNF)
RNF 01: Utilizar a linguagem HTML para desenvolvimento da página <i>web</i>
RNF 02: Utilizar a linguagem de programação <i>JSP</i> para desenvolvimento da página <i>web</i>
RNF 03: Utilizar o padrão de desenvolvimento Orientado à Objetos
RNF 04: Utilizar a linguagem de programação Java para desenvolvimento do <i>backend</i>
RNF 05: Utilizar o banco de dados MySQL para simulação de uma base de dados de software
RNF 06: Utilizar o interpretador Program AB para interpretar a base de conhecimento AIML
RNF 07: Possibilitar que a interação entre o usuário e o <i>chatterbot</i> seja em língua portuguesa
RNF 08: Criar uma base de conhecimento utilizando a linguagem AIML
RNF 09: permitir que o usuário interaja com o <i>chatterbot</i> através de uma página <i>web</i>
RNF 10: realizar consultas em uma base dados para auxiliar na construção das respostas ao usuário
RNF 11: Utilizar CSS na página <i>web</i> para permitir uma interface legível e adequada

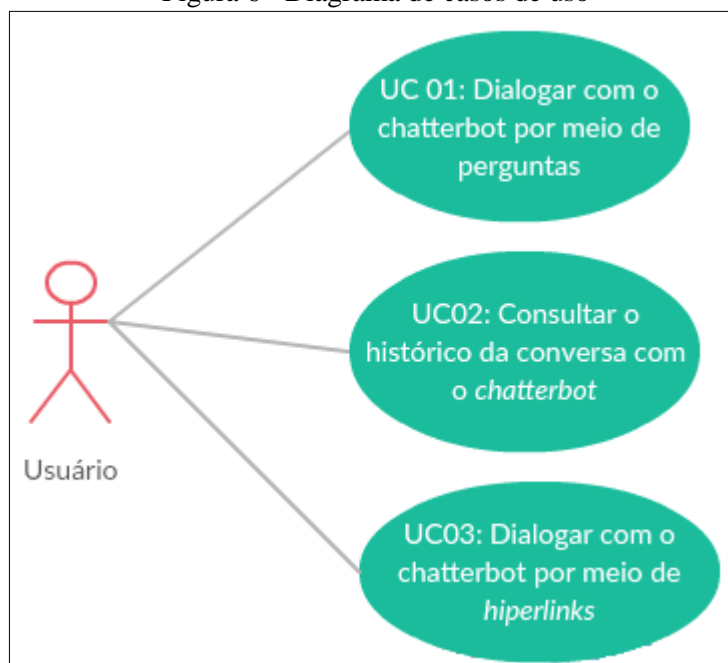
### 3.2 ESPECIFICAÇÃO

Nesta seção é apresentado a especificação da solução do *chatterbot*. Na Seção 3.2.1 são apresentados os casos de uso. Na Seção 3.2.2 são especificadas as *tags* criadas para manipulação de dados, imagens e *hiperlinks*. Na Seção 3.2.3 são apresentados os diagramas de pacotes e de classes. Na Seção 3.2.4 é apresentado o diagrama de atividades. Por fim, na Seção 3.2.5 é apresentada a arquitetura desenvolvida nesta solução.

#### 3.2.1 Casos de uso

Para melhor entendimento das possibilidades de interação do usuário com o *chatterbot*, foi desenvolvido um diagrama de casos de uso. Esses, apresentado na Figura 8. No Apêndice A encontra-se a descrição detalhada dos casos de uso apresentados na Figura 8.

Figura 6 - Diagrama de casos de uso



### 3.2.2 Especificação de *tags* AIML para manipulação de dados

Afim de complementar a resposta ao usuário e reduzir a possibilidade de dúvida do mesmo, foram desenvolvidas *tags* adicionais. Nesta seção são apresentadas as *tags* adicionais que executam uma consulta a uma base de dados, adicionam uma imagem a resposta ao usuário, criam *hiperlinks* no diálogo com o usuário para auxiliá-lo durante a interação.

#### 3.2.2.1 Tag <header>

A *tag* <header> é utilizada em conjunto com as *tags* <option> e <hiperlink>. A união das três *tags* é responsável pela criação do *hiperlink* na interação com o usuário, afim de apresentar o diálogo com o *links* para auxílio na dúvida. *Tag* <header> ilustra o texto que é apresentado na opção para clique do usuário. Um exemplo de utilização desta *tag* está no Quadro 7 em destaque.

Quadro 7 - Exemplo de utilização da tag &lt;header&gt;

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <aiml>
4
5      <category>
6          <pattern>ATUALIZACAOVERSAO</pattern>
7          <template>Selecione a aplicação:
8          <hiperlink><header>- Delphi. </header><option>atualizaVersaoDelphi</option></hiperlink>
9          <hiperlink><header>- Java. </header><option>atualizaVersaoJava</option></hiperlink>
10         <srai>SELECIONAOPCAO</srai></template>
11     </category>
12
13 </aiml>

```

### 3.2.2.2 Tag <option>

A tag <option> é utilizada em conjunto com as tags <hiperlink> e <header>, esta tag <option> é responsável por armazenar a informação que será enviada para a base de conhecimento para consulta, caso o usuário realize a escolha da opção no diálogo utilizando *hiperlinks*. Um exemplo de utilização da tag <option> no Quadro 8.

Quadro 8 - Exemplo de utilização da tag &lt;option&gt;

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <aiml>
4
5      <category>
6          <pattern>ATUALIZACAOVERSAO</pattern>
7          <template>Selecione a aplicação:
8          <hiperlink><header>- Delphi. </header><option>atualizaVersaoDelphi</option></hiperlink>
9          <hiperlink><header>- Java. </header><option>atualizaVersaoJava</option></hiperlink>
10         <srai>SELECIONAOPCAO</srai></template>
11     </category>
12
13 </aiml>

```

### 3.2.2.3 Tag <hiperlink>

A tag <hiperlink> é a tag pai das tags (<header> e <option>). A união das três tags é responsável pela criação do *hiperlink* na interação com o usuário, afim de apresentar o diálogo com o *links* para auxílio na dúvida. O Quadro 9 exemplifica a utilização da tag <hiperlink>. A Figura 9 exemplifica o resultado do tratamento da tag pelo *chatterbot* e a apresentação da informação ao usuário.

Quadro 9 - Exemplo de utilização da tag &lt;hiperlink&gt;

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <aiml>
4
5      <category>
6          <pattern>ATUALIZACAOVERSAO</pattern>
7          <template>Selecione a aplicação:
8              <hiperlink><header>- Delphi. </header><option>atualizaVersaoDelphi</option></hiperlink>
9              <hiperlink><header>- Java. </header><option>atualizaVersaoJava</option></hiperlink>
10             <srai>SELECIONAOPCAO</srai></template>
11          </category>
12
13 </aiml>

```

Figura 7 - Resultado da Tag &lt;hiperlink&gt;, &lt;option&gt; e &lt;header&gt; na apresentação ao usuário

**Bot**

---

Bot é um robô desenvolvido para responder todas as suas dúvidas sobre o processo de Atualização de Versão de Software.

**Bot** > Olá tudo bem?

**Você** > sim

**Bot** > Legal! Você deseja fazer alguma pergunta para mim?

**Você** > ATUALIZACAOVERSAO

**Bot** > Selecione a aplicação:

- Delphi.

- Java.

Por favor, selecione uma opção.

Enviar

NOTE: Você deve preencher o campo acima e clicar em Enviar.

### 3.2.2.4 Tag <query>

A tag <query> é utilizada para auxiliar na criação de uma resposta ao usuário com a busca da informação a uma base de dados a partir de uma *query*. O Quadro 10 exemplifica como utilizar a tag e a Figura 10 exemplifica o resultado na apresentação ao usuário.

Quadro 10 - Exemplo de utilização da tag &lt;query&gt;

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <aiml>
4
5      <!-- INICIO - tempo de liberação de versão -->
6
7      <category>
8          <pattern>^ TEMPO ^ LIBERADA ^ VERSÃO</pattern>
9          <template> A versão do Software é liberada a cada
10             <query>select desc_inf from inf_tcc2 where inf = 'dias versão'</query> dias.
11          </template>
12      </category>
13
14 </aiml>

```

Figura 8 - Resultado da tag &lt;query&gt; na apresentação ao usuário

**Bot**

---

Bot é um robô desenvolvido para responder todas as suas dúvidas sobre o processo de Atualização de Versão de Software.

Bot > Olá tudo bem?

Você > tudo

Bot > Show! Você deseja fazer alguma pergunta para mim?

Você > de quanto em quanto tempo é liberada versão?

Bot > A versão do Software é liberada a cada 15 dias.

NOTE: Você deve preencher o campo acima e clicar em Enviar.

### 3.2.2.5 Tag <img>

A tag <img> é utilizada para auxiliar na criação de uma resposta ao usuário com a adição de uma imagem. No Quadro 11 é apresentado o modo de utilização desta tag e a Figura 11 apresenta o resultado do tratamento da tag ao usuário.

Quadro 11 - Exemplo de utilização da tag &lt;img&gt;

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <aiml>
4
5      <category>
6          <pattern>RECYCLEBIN</pattern>
7          <template>Quando o parâmetro RECYCLEBIN está habilitado,
8          <img>recyclebin.jpg</img>
9
10         Solução:
11         - Alterar o parâmetro de RECYCLEBIN para "OFF";
12         - Realizar a limpeza da lixeira (PURGE RECYCLEBIN).
13         </template>
14     </category>
15
16 </aiml>

```

Figura 9 - Resultado da query &lt;img&gt; ao usuário

**Bot**

---

Bot é um robô desenvolvido para responder todas as suas dúvidas sobre o processo de Atualização de Versão de Software.


**Bot >** Olá tudo bem?

**Você >** tudo

**Bot >** Show! Você deseja fazer alguma pergunta para mim?

**Você >** RECYCLEBIN

**Bot >** Quando o parâmetro RECYCLEBIN está habilitado, não é possível prosseguir com a atualização de versão, pois será apresentada a seguinte mensagem:



Solução:

- Alterar o parâmetro de RECYCLEBIN para "OFF";
- Realizar a limpeza da lixeira (PURGE RECYCLEBIN).

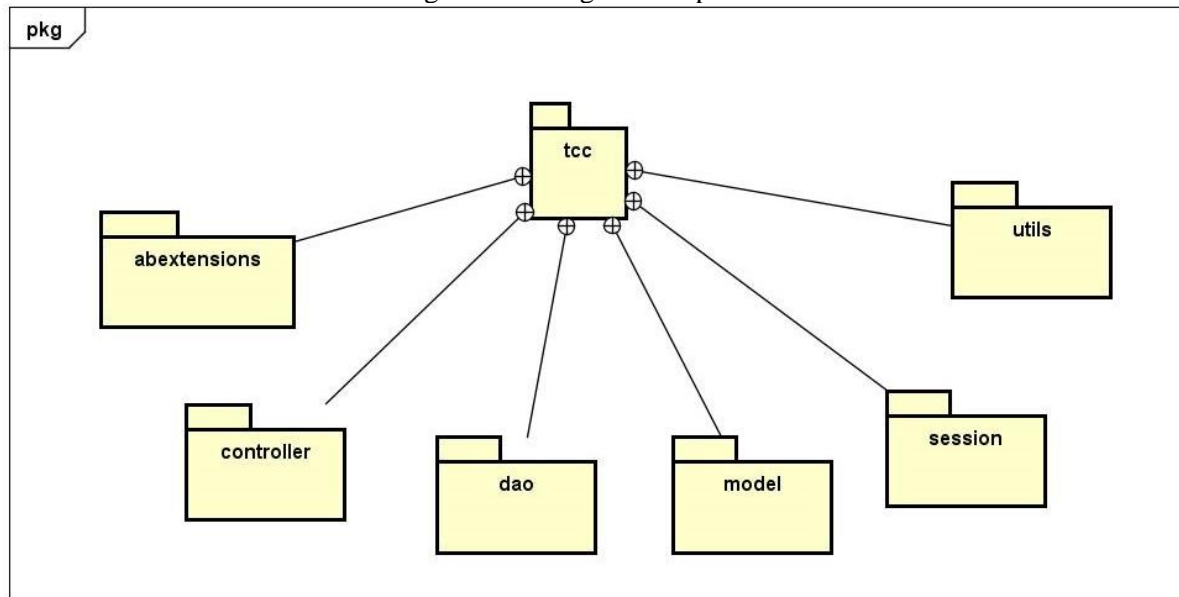
Enviar

NOTE: Você deve preencher o campo acima e clicar em Enviar.

### 3.2.3 Diagrama de pacotes e de classes

O objetivo desta seção é apresentar o diagrama de pacotes e de classes implementados para o desenvolvimento do *chatterbot*. O projeto foi separado em pacotes, com o objetivo de organizar as classes por solução. O nome do pacote já fornece um pré-entendimento das classes que estão contidas dentro do mesmo. A Figura 12 apresenta um diagrama de pacotes da solução.

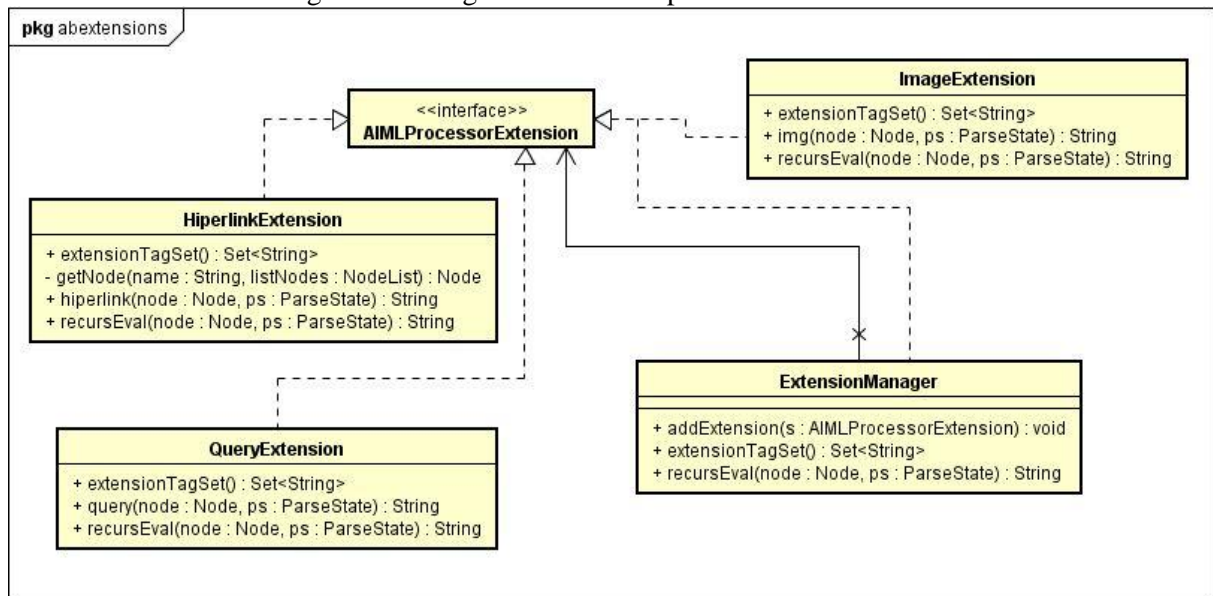
Figura 10 - Diagrama de pacotes



### 3.2.3.1 Pacote abextensions

As classes do pacote `abextensions` são apresentadas na Figura 13. Para auxiliar na criação das respostas aos usuários foram criadas *tags* AIML adicionais para esta atividade.

Figura 11 - Diagrama de classes pacote abextensions



Este pacote contém as classes que implementam o tratamento das *tags* AIML adicionais criadas: `HiperlinkExtension`, `QueryExtension`, `ImageExtension` e `ExtensionManager`. Essas classes implementam a interface `AIMLProcessosExtension` que obriga a implementação dos métodos `extensionTagSet` e `recursEval`. O método `extensionTagSet` é utilizado para retornar os nomes das tags que são implementados pela classe. O método `recursEval` é utilizado para interpretação da tag. A classe `ImageExtension` é responsável

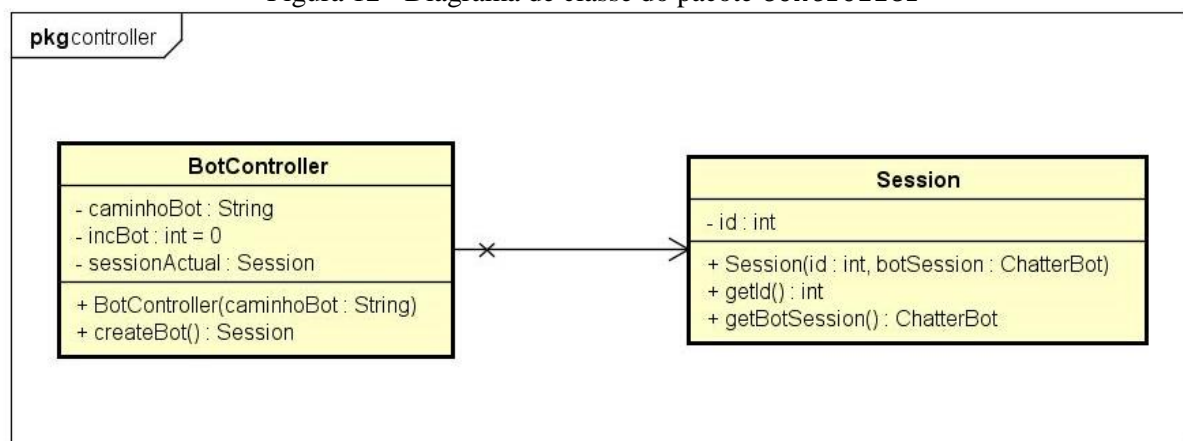


por interpretar a *tag* `<img>`, responsável pela adição de imagens para auxílio ao usuário na construção da resposta. O método responsável pela interpretação do conteúdo na classe é o `img`. A classe `HiperlinkExtension` é responsável por interpretar as *tags* `<hiperlink>` e suas *tags* filhas `<header>` e `<option>` responsáveis pela adição de links na interação com o usuário. O método responsável pela tratamento destas *tags* é o `hiperlink`. O método `getNode` é chamado dentro do método `hiperlink` e tem como objetivo buscar o conteúdo das *tags* filhas `<header>` e `<option>` para tratamento pelo método `hiperlink`. A classe `QueryExtension` é responsável pela interpretação da *tag* AIML `<query>` responsável por executar uma *query* na base de dados para obtenção de informações para auxiliar na montagem da resposta do *chatbot* ao usuário. O tratamento desta *tag* é feito pelo método `query`. A última classe do pacote `abextensions` é a classe `ExtensionManager`, esta classe tem como finalidade agrupar as *tags* apresentadas anteriormente. Isto se faz necessário, devido ao interpretador aceitar apenas a adição de uma extensão.

### 3.2.3.2 Pacote `controller`

As classes contidas no pacote `controller` disponibilizam os métodos responsáveis pela criação de novos *chatbots* para a sessão do usuário, quando solicitado pela interface *web*. O pacote contém a classe `BotController` conforme descrito na Figura 14.

Figura 12 - Diagrama de classe do pacote `controller`

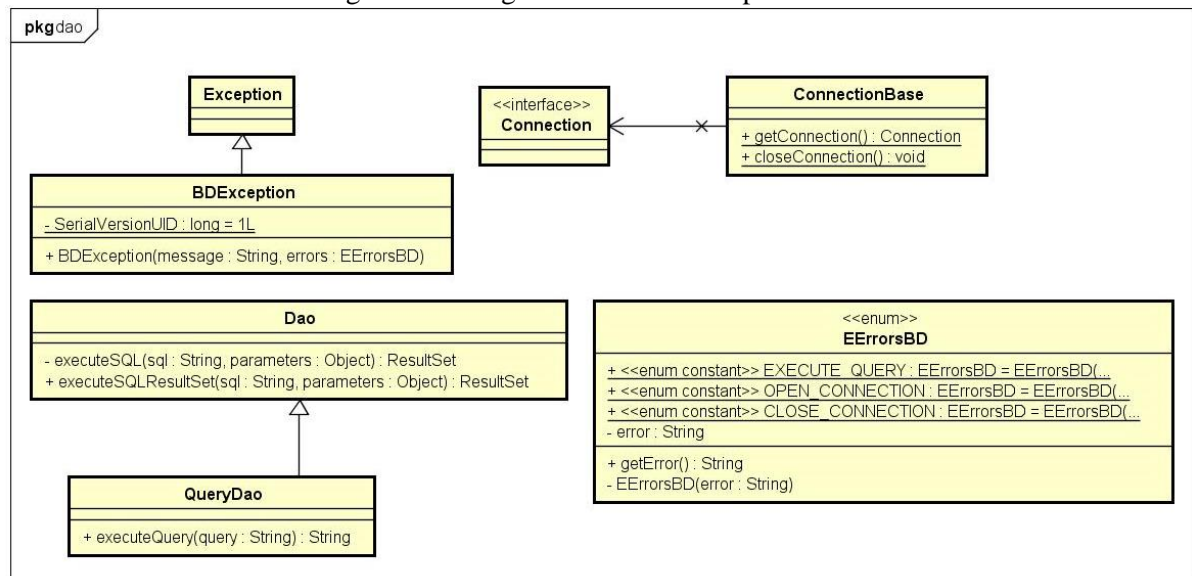


A classe `BotController` é responsável pela criação de novos *chatbots*. Por meio do método `createBot`, chamado a partir da interface *web*, é criada uma nova instância do *chatbot*, ainda, neste método `createBot` é referenciada as classes de tratamento das *tags* AIML adicionais criadas.

### 3.2.3.3 Pacote dao

As classes contidas no pacote `dao` são responsáveis por realizar consultas a base de dados, desenvolvida para conter informações com o intuito de auxiliar na construção de respostas do *chatterbot*. As classes contidas no pacote `dao` são as classes `BDEException`, `ConnectionBase`, `Dao`, `EErrorsBD` e `QueryDao`, conforme mostrado na Figura 15.

Figura 13 - Diagrama de classes do pacote `dao`



O enumerado `EErrorsBD` é responsável por conter mensagens padrões de erros que são utilizadas nas classes `Dao` e `QueryDao` nas exceções dos métodos que executam alguma iteração com o banco de dados. Por meio do método `getError` é possível retornar a mensagem de erro conforme definido na exceção do método. As constantes literais do enumerado são `EXECUTE_QUERY`, `OPEN_CONNECTION` e `CLOSE_CONNECTION`.

A classe `BDEException` estende da classe `Exception` para a criação de uma exceção de tratamento específica que utiliza as mensagens do enumerado `EErrorsBD`. Este tratamento de exceção é utilizado em todas as ações com o banco de dados. Deste modo, facilitando o entendimento da exceção e sua localização dentro do programa. A classe contém apenas um método, o `BDEException`, na qual faz referência ao método de criação da classe estendida `Exception`, adicionando na mensagem a informação do enumerado `EErrorsBD` citado anteriormente.

A classe `ConnectionBase` é responsável pela criação de uma conexão com o banco de dados. Conforme citado anteriormente, a consulta ao banco de dados se faz necessária para auxiliar a criação de respostas do *chatterbot*. Esta classe utiliza do método `getConnection` para definir a conexão com a base de dados, a partir de um driver de conexão. O método

`closeConnection` é responsável por fechar a conexão com o banco de dados criada pelo método `getConnection`. Este método é de suma importância, uma vez que, se muitas conexões permanecerem abertas com o banco de dados, pode ocorrer erros na tentativa de novas conexões devido ao banco de dados ter chegado ao seu limite.

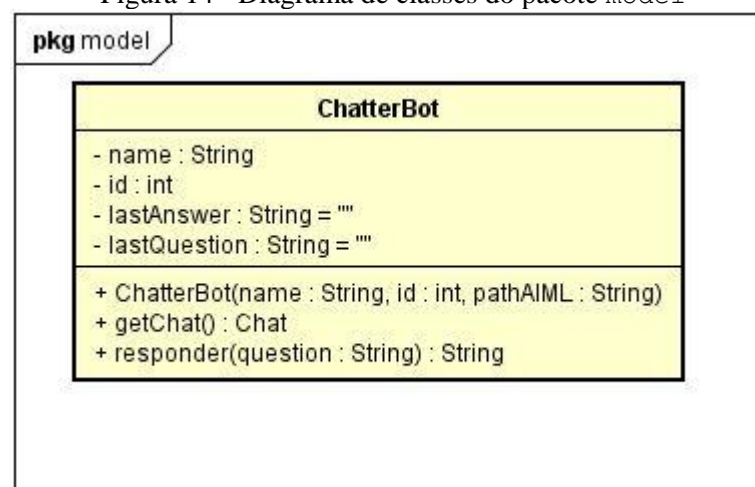
A classe `Dao` é responsável pela execução de consultas na base de dados. Por meio do método privado `executeSQL` é obtida uma conexão a partir da classe `ConnectionBase`. Também é criada uma nova declaração para a realização de uma consulta a base de dados. Os parâmetros são passados a partir da variável `parameters` e estes recebem um tratamento específico dentro do método para controle por tipo de atributo. Existe ainda, o método `executeSQLResultSet` que é público e fica responsável pela chamada do método privado para retornar o valor da execução da query.

A classe `QueryDao` que estende da classe `Dao` é responsável por receber o comando de consulta descrito na tag AIML `<query>`. Por meio do método `executeQuery` é realizada a chamada ao método `executeSQLResultSet` da classe `Dao` para execução da *query*. Após o retorno da execução da query, se faz necessário o desenvolvimento de uma lógica para obter o retorno da *query*. Em seguida, a informação é retornada para complementar a criação da resposta do *chatterbot* ao usuário.

#### 3.2.3.4 Pacote `model`

A classe `Chatterbot` do pacote `model` é responsável pela criação de novos *chatterbots* e executar perguntas ao *chatterbot*. Essas perguntas são as dúvidas informadas pelos usuários na interação com a interface *web*. A Figura 16 ilustra o pacote `model`.

Figura 14 - Diagrama de classes do pacote `model`



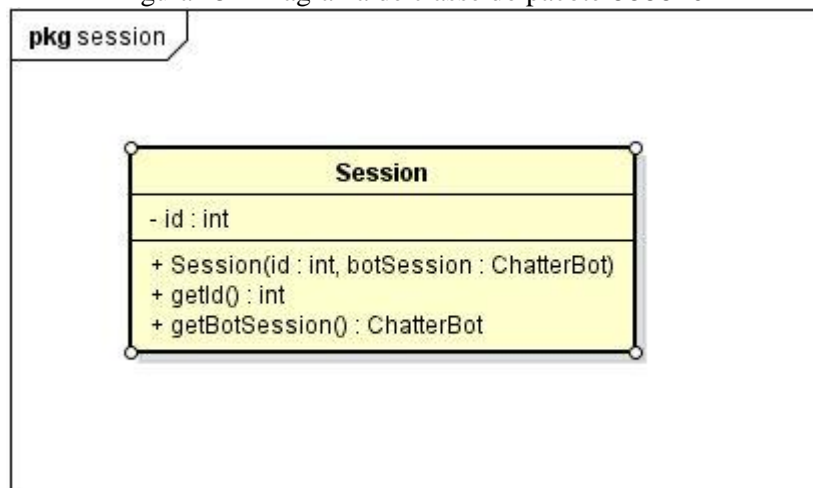
O método `Chatterbot` é responsável pela criação de novos *chatterbots*. Já o método `responder` é responsável por realizar os questionamentos ao *chatterbot*. Neste método

existem tratamentos para formatação da mensagem a partir do pacote `utils`, bem como, faz tratamento para caso o *chatterbot* não tenha resposta para a dúvida do usuário, iniciar o diálogo com o usuário através de *hiperlinks*.

#### 3.2.3.5 Pacote `session`

A classe `Session`, contida no pacote de mesmo nome é responsável por armazenar uma instância da classe `Chatterbot` citada anteriormente e um `<id>` para identificação, a fim de manter a rastreabilidade do *chatterbot* com a sessão de interação *web*. No pacote `session` está contida a classe `Session` conforme apresentado na Figura 17.

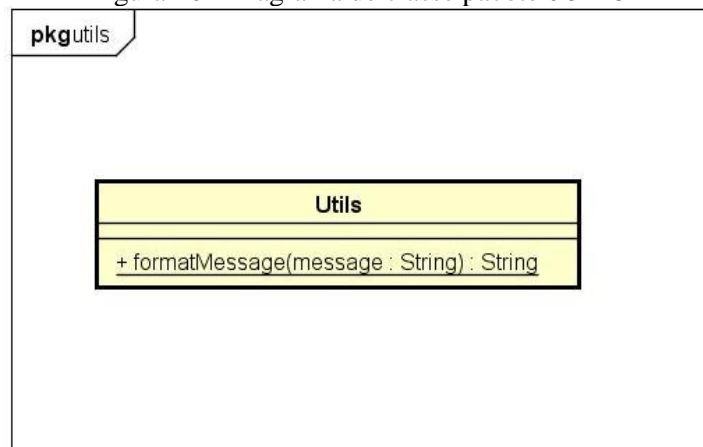
Figura 15 - Diagrama de classe do pacote `session`



#### 3.2.3.6 Pacote `utils`

O pacote `utils` é responsável por armazenar a classe de formatação da mensagens antes do envio para a interface *web*. No pacote `session` está contida a classe `Utils`, conforme apresentado na Figura 18.

Figura 16 - Diagrama de classe pacote `utils`

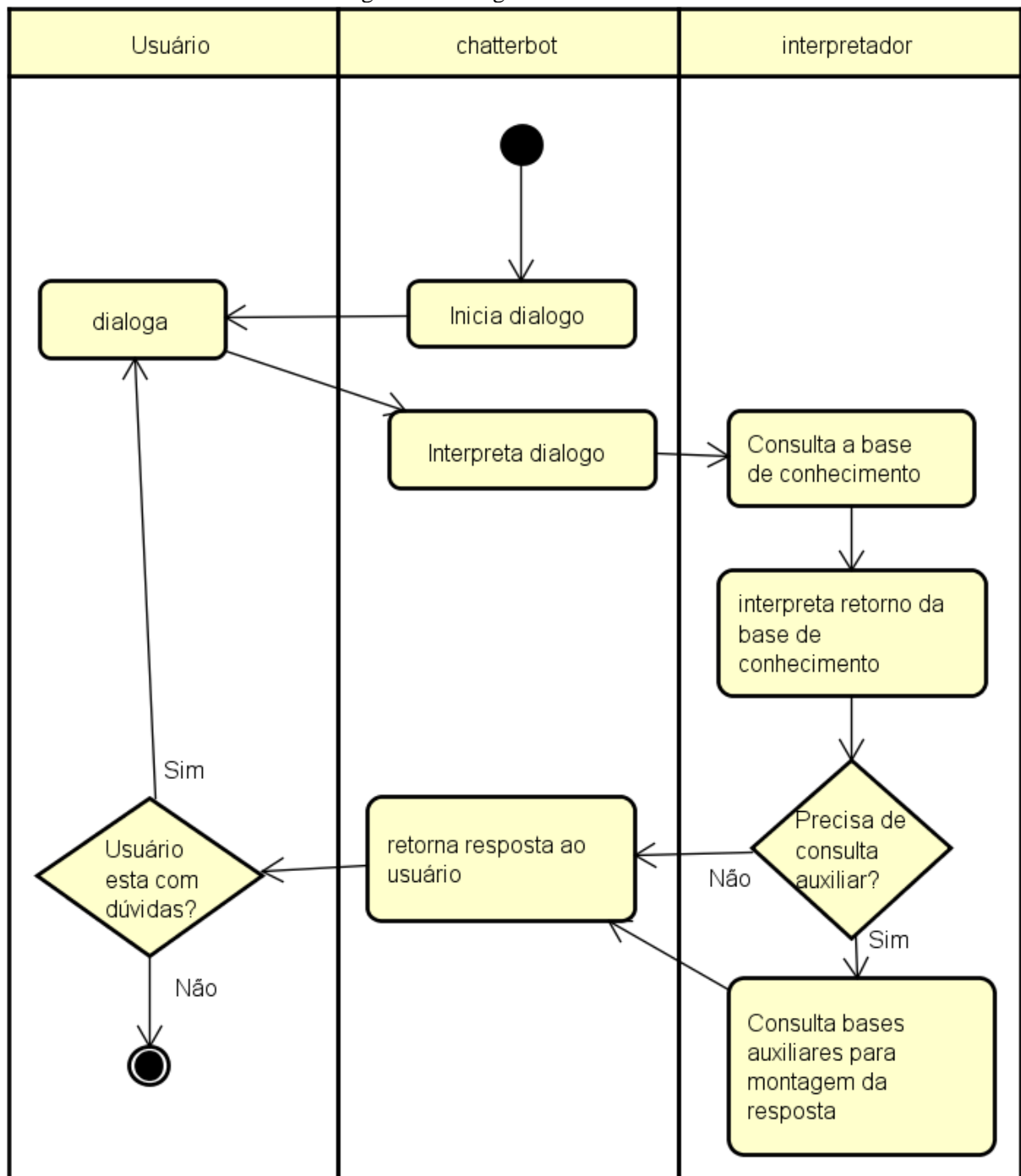


A formatação das mensagens são realizadas utilizando o método `formatMessage`. Este faz a substituição da mensagem adicionando a quebra de página caso tenha sido feita na *tag* `<template>` na base de conhecimento AIML. O método ainda adiciona a *tag* `<span>`, que é responsável por destacar mensagem Bot > em vermelho, informando a *class* `bot` a *tag* `<span>` para posterior tratamento pelo CSS na interface *web*.

#### 3.2.4 Diagrama de atividades

Nesta seção será apresentada o diagrama de atividades da solução proposta. A Figura 19 demonstra o processo de interação entre o usuário, o *chatterbot* e o interpretador. O usuário realiza a pergunta, o *chatterbot* processa a pergunta e envia ao interpretador que, por sua vez realiza a consulta na base de conhecimento. Se necessário, o interpretador realiza a consulta a base de dados, ou adiciona uma imagem, ou ainda, cria um *hiperlink*. Após feito isso, a mensagem de retorno é enviada ao usuário, caso o mesmo deseje realizar outro diálogo, esse ciclo se repete até que o usuário esteja com todas as dúvidas sanadas, onde se encerra o diálogo.

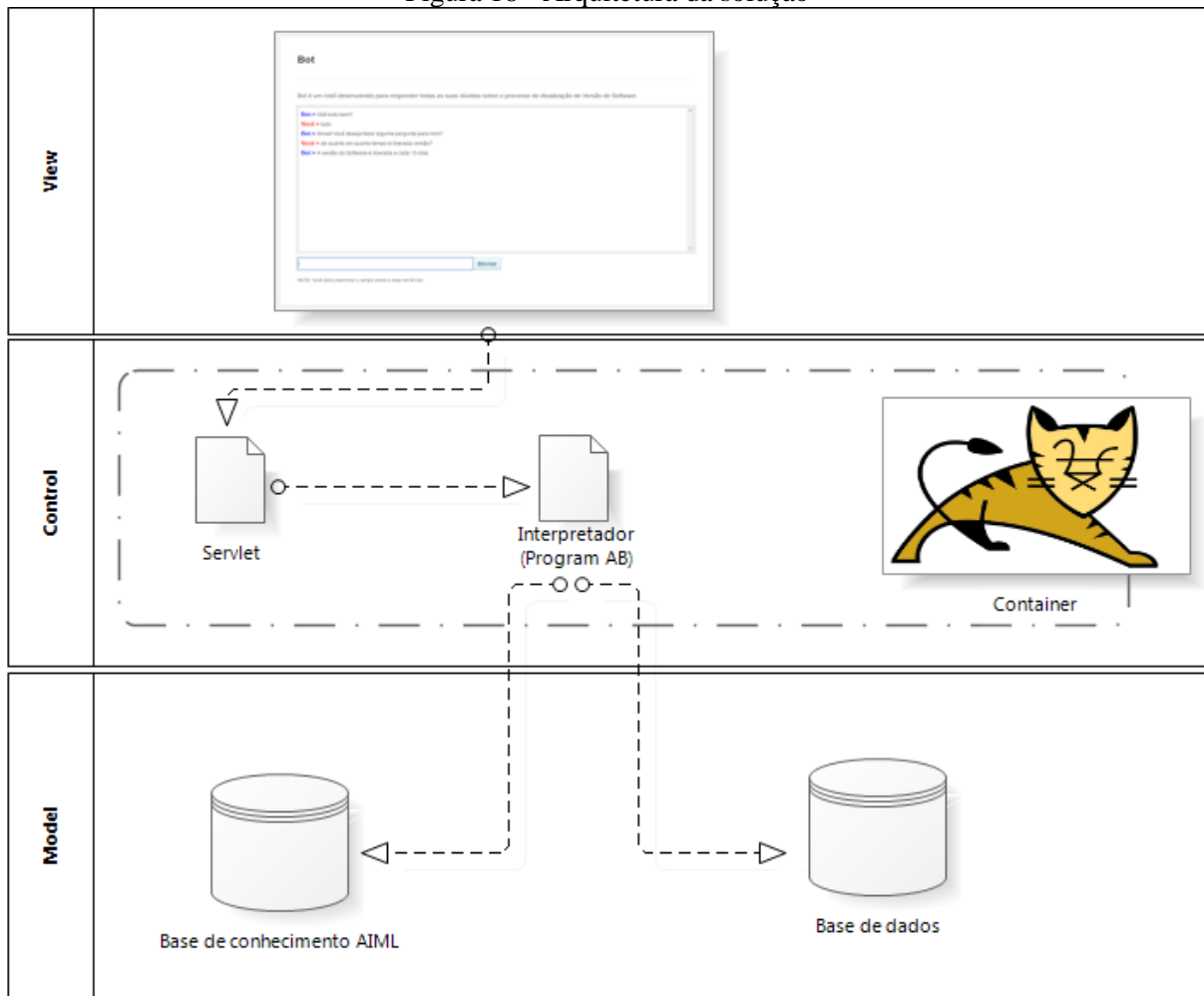
Figura 17 - Diagrama de atividades



### 3.2.5 Arquitetura proposta

Na Figura 20 é ilustrada a arquitetura com 3 camadas, a camada *view* onde é apresentada a página *web* para interação com o usuário, a camada *control*, onde está implementado o interpretador e a camada *model* contendo o base de conhecimento AIML e o banco de dados MySQL.

Figura 18 - Arquitetura da solução



### 3.3 IMPLEMENTAÇÃO

Nesta seção serão apresentadas as ferramentas utilizadas para o desenvolvimento do projeto, a implementação para comunicação com a base de conhecimento, a implementação para comunicação com a base de dados, a implementação para tratamento das novas *tags* AIML que foram criadas e a base de conhecimentos AIML que foi criada para diálogo com o usuário.

#### 3.3.1 Técnicas e ferramentas utilizadas

Nesta seção são abordadas as ferramentas e técnicas utilizadas para desenvolvimento do *chatterbot*. As ferramentas e técnicas são:

- Netbeans IDE 8: IDE utilizada para desenvolvimento da aplicação. Netbeans é uma IDE *open source* mantida pela Sun Microsystems;
- Java 1.8 build 74: linguagem de programação utilizada para o desenvolvimento da parte do servidor;

- c) Apache Tomcat 8.0.9: *web* container de código aberto desenvolvido pela Apache Software Foundation. Suporta diversas especificações do Java EE, como, *Java servlet*, *Java Server Pages* (JSP), etc;
- d) *Java Sever Pages* (JSP): tecnologia utilizada para desenvolvimento de páginas *web* dinâmicas. Esta linguagem possibilita a escrita de código Java em uma página HTML. Para visualizar a mesma se faz necessário a utilização de um servidor de aplicação, como o Tomcat. Estes *containers* possuem compiladores JSP que fazem a compilação do código em tempo de execução (GOMES, 2012);
- e) JavaScript: linguagem utilizada para validações na página *web* e chamadas para os métodos JSP. Foi utilizada a biblioteca jQuery v2.2.4. JavaScript é uma linguagem interpretada e com possibilidade de execução de scripts do lado cliente, facilitando a escrita de métodos sem a necessidade do servidor de aplicação para executá-los;
- f) Program AB: interpretador utilizado para reconhecimento e iteração com a base de conhecimentos AIML desenvolvida;
- g) MySQL versão 5.7.13: banco de dados utilizado para simulação do banco de dados de um software para gestão hospitalar. MySQL é um banco de dados *open source*, mas que também se apresenta na versão Enterprise que é comercializada pela Oracle Corporation.

### 3.3.2 Base de conhecimentos AIML

A base de conhecimentos AIML foi desenvolvida para que o *chatbot* dialogue com o usuário. Esta base de conhecimentos foi criada para dialogar sobre o processo de atualização de versão de software. Os padrões criados na base de conhecimentos foram definidos a partir de sentenças que podem ser digitadas pelo usuário quando for dialogar sobre o processo de atualização de versão de software.

Para contemplar o diálogo com o usuário, foram criados mais de dez arquivos `.aiml` divididos por assunto do diálogo (Apêndice B). Como o processo em ambas as aplicações é similar, as bases podem conter informações similares, alterando apenas a aplicação. Ainda, existem base de conhecimentos para introdução do diálogo com o usuário e, para iniciar a utilização de *hiperlinks* no diálogo, caso, o *chatbot* não tenha uma resposta para algum diálogo com o usuário.

A base de conhecimentos de cada aplicação estão divididas por tópicos de assuntos relevantes sobre a atualização de versão, como por exemplo, problemas em determinada fase do processo de atualização. Para exemplificar, tem-se como exemplo a base de conhecimento



*problens\_upgrade\_delphi.aiml*. Nesta base de conhecimento, estão contidas informações sobre o processo de atualização de versão da aplicação desenvolvida em Delphi, bem como, situações que podem ocorrer antes de iniciar o processo de atualização ou depois de finalizar o processo. O Quadro 12 ilustra um pedaço da base de conhecimento com a finalidade de orientar o cliente sobre a mensagem de que alguns campos não foram criados durante o processo de atualização de versão.

Quadro 12 - Padrão de reconhecimento para a falta da criação de um campo durante o processo de atualização de versão

```

5      <category>
6          <pattern>^ ALGUNS CAMPOS NAO ^ CRIADOS</pattern>
7          <that>Me fale mais sobre o problema na aplicação delphi</that>
8          <template>Caso ao final da Atualização da Versão apresente a mensagem:
9              "Alguns campos não foram criados", deve-se seguir os passos abaixo.
10         - Executar o comando conectado na base de dados select * from consiste_tabela_A
11         Este comando retornará o comando para que o campos seja criado na base de dados
12         Caso seja executado o comando e apresente um erro. Se faz necessário a abertura
13         O erro em anexo para que seja analisada a causa.
14         </template>
15     </category>

```

Conforme mencionado anteriormente, foi desenvolvido uma base de conhecimentos para orientações sobre o processo de atualização de versão em Java. Como exemplo, utiliza-se a base de conhecimento *problens\_upgrade\_java.aiml*. Nesta base de conhecimento, estão contidas as informações sobre o processo de atualização de versão da aplicação desenvolvida em Java. O Quadro 13 ilustra uma orientação ao usuário, sobre como proceder, caso seja apresentada uma mensagem contendo a palavra *socket*.

Quadro 13 - Padrão de reconhecimento para orientação sobre a mensagem *socket*

```

5      <category>
6          <pattern>^ MENSAGEM ^ SOCKET ^</pattern>
7          <that>Me fale mais sobre o problema na aplicacao java</that>
8          <template>Caso esteja sendo apresentada a mensagem:
9              "Não foi possivel obter conexão com o Socket".
10             É necessário reiniciar o servidor de aplicação.
11             Esta ação se faz necessária devido a perda do pool de conexões do servidor de aplicação.
12             Após reiniciado o servidor de aplicação. Favor reiniciar o processo de Atualização de Versão.
13         </template>
14     </category>
15

```

Tem-se também a base de conhecimentos para dialogar com o usuário utilizando *hiperlinks*, que são ativados a partir do momento que o *chatterbot* não possui uma resposta para uma sentença do usuário, visando assim, orientar o mesmo sobre as dúvidas que o *chatterbot* pode dialogar. Para exemplificar, o Quadro 14 ilustra uma categoria da base de conhecimento *upgradeVersionOptionsDelphi.aiml* que apresenta um menu ao usuário com as possibilidades de diálogos na atualização de versão da aplicação Delphi.

Quadro 14 - Padrão para montagem de menu com possibilidades de diálogo atualização de versão Delphi

```

5  <category>
6  <pattern>ATUALIZAVERSAODELPHI</pattern>
7  <template> Sobre Atualização de Versão de software. Estou programado para lhe ajudar com os itens abaixo:
8  <hiperlink><header>- Mais informações sobre a atualização de versão de software.</header> <option>MAISINFOATUALIZACAOVERSAO</option></hiperlink>
9  <hiperlink><header>- Pré-Requisitos para atualização da versão Delphi.</header><option>preReqAtuaDelphi</option></hiperlink>
10 <hiperlink><header>- Arquivos necessários para a atualização da versão Delphi.</header> <option>arquivosAtualizacaoDelphi</option></hiperlink>
11 <hiperlink><header>- Esta apresentando uma mensagem durante a execução da Fase 1 da atualização de versão.</header> <option></option>
12 <hiperlink><header>- Esta apresentando uma mensagem durante a execução da Fase 2 da atualização de versão.</header> <option></option>
13 <hiperlink><header>- Objetos inválidos ao final da atualização de versão.</header> <option>objetoInvalido</option></hiperlink>
14 <hiperlink><header>- Mensagem "alguns campos não foram criados!" ao final da atualização de versão!</header> <option>camposN</option></hiperlink>
15 <srai>SELECIONAOPCAO</srai></template>
16 </category>

```

Por fim, tem-se a base de conhecimentos para iniciar o diálogo com o usuário, cumprimentando-o. O Quadro 15 ilustra parte dos padrões desenvolvidos na base de conhecimento *introduction.aiml* para iniciar um diálogo com o usuário.

Quadro 15 - Base de conhecimento para iniciar o diálogo com o usuário

```

4  <category>
5  <pattern>OI ^</pattern>
6  <template> <srai>OLÁ</srai> </template>
7  </category>
8
9  <category>
10 <pattern>OIE ^</pattern>
11 <template> <srai>OLÁ</srai> </template>
12 </category>
13
14 <category>
15 <pattern>OE ^</pattern>
16 <template> <srai>OLÁ</srai> </template>
17 </category>
18
19 <category>
20 <pattern>OLA ^</pattern>
21 <template> <srai>OLÁ</srai> </template>
22 </category>
23
24 <category>
25 <pattern>OLÁ ^</pattern>
26 <template> Olá! você esta bem?
27 </template>
28 </category>

```

### 3.3.3 Tag para criação de *hiperlink* a partir da base de conhecimentos AIML

Para a criação de *hiperlinks* de interação com o usuário, a partir do momento que o *chatbot* não possuir resposta ao diálogo, foi desenvolvida a classe *HiperlinkExtension*, que implementa a classe *AIMLProcessorExtension* do pacote do interpretador Program AB. Deste modo, criou-se a tag *<hiperlink>*, cuja manipulação é feita pelo método *hiperlink* ilustrado no Quadro 16.

Quadro 16 - Classe *HiperlinkExtension* método *hiperlink*

```

1 public String hiperlink(Node node, ParseState ps) {
2     String hiperlinkText = "";
3
4     NodeList childs = node.getChildNodes();
5     Node headerNode = this.getNode("header", childs);
6     Node optionNode = this.getNode("option", childs);
7
8     if (headerNode != null
9         && optionNode != null) {
10
11         String header = AIMLProcessor.evalTagContent(headerNode, ps, null);
12         String option = AIMLProcessor.evalTagContent(optionNode, ps, null);
13
14         if (header != null
15             && option != null) {
16
17             hiperlinkText = "<a href=\"#" onclick=\"getAnswers('" +
18                 option + "', '" + header + "'); return false\">" + header + "</a>";
19         }
20     }
21
22     return hiperlinkText;
23 }

```

Observa-se que na linha 4 é criada a variável `childs` com a referência das *tags* filhas da *tag* `<hiperlink>`. Nas linhas 5 e 6, são realizadas as chamadas do método `getNode` para buscar a referência das *tags* `<header>` e `<option>`. Em seguida, nas linhas 8 e 9 são testados os conteúdos das variáveis `headerNode` e `optionNode`, afim de identificar se existiam valores. Caso sim, são obtidos os valores das *tags* nas linhas 11 e 12. Por fim, caso possua conteúdo, nas linhas 17 e 18, é feita a criação da *tag* `<a>` para criação do *hiperlink* na página *web*.

### 3.3.4 Tag para execução de consultas a base de dados a partir da base de conhecimentos AIML

Para a execução de consultas à base de dados para complementar a criação da resposta ao diálogo do usuário. Foi criada a *tag* `<query>`. Para o tratamento de *tag*, foi criada a classe *QueryExtension*, que funciona como uma extensão do interpretador Program AB. Conforme no Quadro 17, o método `query` é responsável por fazer a manipulação da *tag*.

Quadro 17 - Classe *QueryExtension* método *query*

```

1 public String query(Node node, ParseState ps) throws BDEException {
2
3     String sqlResult = "";
4     String sql = AIMLProcessor.evalTagContent(node, ps, null);
5
6     if (!sql.isEmpty()) {
7         sqlResult = new QueryDao().executeQuery(sql);
8     }
9
10    return sqlResult;
11 }

```

Observa-se que na linha 4 é atribuído o conteúdo da *tag* à variável *sql*. Caso tenha valor (linha 6), o método *executeQuery* que é chamado para executar a consulta ao banco de dados. Após execução da consulta ao banco de dados, a variável *sqlResult* é responsável por retornar o valor da consulta. Deste modo, a *tag* `<query>`, recebe o resultado da consulta ao banco de dados.

### 3.3.5 Executando consultas a base de dados

Para consultar a base de dados a partir do conteúdo enviado pela *tag* `<query>`, foi implementada a classe *QueryDao*. O método *executeQuery* dessa classe é ilustrado no Quadro 18.

Quadro 18 - Método *executeQuery* da classe *QueryDao*

```

1 public String executeQuery(String query) throws BDEException {
2
3     try {
4         String queryResult = "";
5
6         ResultSet rs = executeSQLResultSet(query);
7         if (rs.next())
8             queryResult = rs.getString(1);
9
10        return queryResult;
11
12    } catch (Exception e) {
13        throw new BDEException(e.getMessage(),
14                                EErrorsBD.EXECUTE_QUERY);
15    } finally {
16        ConnectionBase.closeConnection();
17    }
18
19 }

```

Na linha 6 do método mostrado no Quadro 18 é executado o método *executaSQLResultSet*, responsável por retornar o conteúdo da consulta na instância do *ResultSet*. Na linha 7 é verificado se a query retornou valor da consulta. Caso sim, na linha 8 é adicionado o valor a variável *queryResult*. Na linha 13 e 14 é realizado o tratamento para

execução de uma exceção caso aconteça alguma falha de comunicação durante a consulta e, por fim, a linha 16 fecha a conexão estabelecida com o banco de dados para realização da consulta.

Observa-se no Quadro 18 que o método *executeQuery* da classe *QueryDao* realiza a chamada do método *executaSQLResultSet*. Este método, é um método público da classe *Dao*. Esta classe foi criada para realizar a consulta ao banco de dados. O método responsável por realizar a consulta ao banco de dados, é o método privado *executeSQL*, conforme ilustrado no Quadro 19.

Quadro 19 - Método *executeSQL* da classe *Dao*

```

1 private ResultSet executeSQL (String sql, Object... parameters) throws BDEException {
2     try {
3         ResultSet resultSet;
4         PreparedStatement prepStatement
5             = ConnectionBase.getConnection().prepareStatement(sql,
6                 ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
7
8         int index = 1;
9         if (parameters != null) {
10             for (Object obj : parameters) {
11                 if (obj instanceof ArrayList) {
12                     ArrayList auxObj = (ArrayList) obj;
13
14                     for (Object aux : auxObj) {
15                         if (aux != null) {
16                             prepStatement.setObject(index++, aux);
17                         } else {
18                             prepStatement.setObject((index++), "");
19                         }
20                     }
21                 } else {
22                     prepStatement.setObject((index++), obj == null ? "" : obj);
23                 }
24             }
25         }
26         resultSet = prepStatement.executeQuery();
27         return resultSet;
28     } catch (Exception e) {
29         throw new BDEException(e.getMessage(),
30             EErrorsBD.EXECUTE_QUERY);
31     }
32 }
33 }

```

Na linha 4,5 e 6 é criada a conexão com o bando de dados passando como parâmetro a variável *sql*, na qual contém a consulta da *tag <query>*. Entre as linhas 9 a 26 é realizado o tratamento para passagem de parâmetro, caso tenha sido passado parâmetros para a consulta. Na linha 27 é executada a consulta e, na linha 28, é realizado seu retorno. Ainda, entre as linhas 29 a 32 é realizado o tratamento de exceção, caso aconteça uma falha de comunicação com o banco de dados.

Por fim, na linha 5 do Quadro 19 é executado o método *getConnection* da classe *ConnectionBase*. Esta classe foi criada para centralização da conexão com o banco de dados,

deste modo, tem-se apenas uma classe abrindo conexão com o banco de dados. O método *getConnection* está ilustrado no Quadro 20.

Quadro 20 - Método *getConnection* da classe *ConnectionBase*

```
1 public static Connection getConnection() throws BDException {  
2  
3     try {  
4         Class.forName("com.mysql.jdbc.Driver");  
5  
6         connection = DriverManager.getConnection(  
7             "jdbc:mysql://localhost:3306/tcc2", "root", "root");  
8  
9     } catch (Exception e) {  
10        throw new BDException(e.getMessage(),  
11            EErrorsBD.OPEN_CONNECTION);  
12    }  
13  
14    return connection;  
15 }
```

Observa-se que, na linha 4 é obtido o *driver* de conexão com o bando de dados. Na linha 6 é iniciada a conexão com o banco de dados. Nas linhas 9 a 12 é realizado o tratamento de exceção caso aconteça alguma falha na solicitação e conexão com o banco de dados.

### 3.3.6 Operacionalidade da implementação

O projeto consiste em fazer a interação do usuário com um *chatterbot* e, para isso, foi criada uma página *web*. Esta página contém um campo texto, onde é registrado o histórico do diálogo e as respostas do *chatterbot*. Um campo texto para o usuário digitar seu diálogo para o *chatterbot* responder e, um botão enviar que, envia o diálogo para o *chatterbot* responder. O diálogo enviado pelo usuário ao *chatterbot* não deve conter erro ortográfico ou erro de concordância verbal. A Figura 21 ilustra um diálogo entre o *chatterbot* e o usuário.

Figura 19 - Diálogo entre o *chatterbot* e o usuário

**Bot**

Bot é um robô desenvolvido para responder todas as suas dúvidas sobre o processo de Atualização de Versão de Software.

Bot > Olá! tudo bem?

Você > sim e com você

Bot > Estou muito bem! Você deseja fazer alguma pergunta para mim?

Você > sim

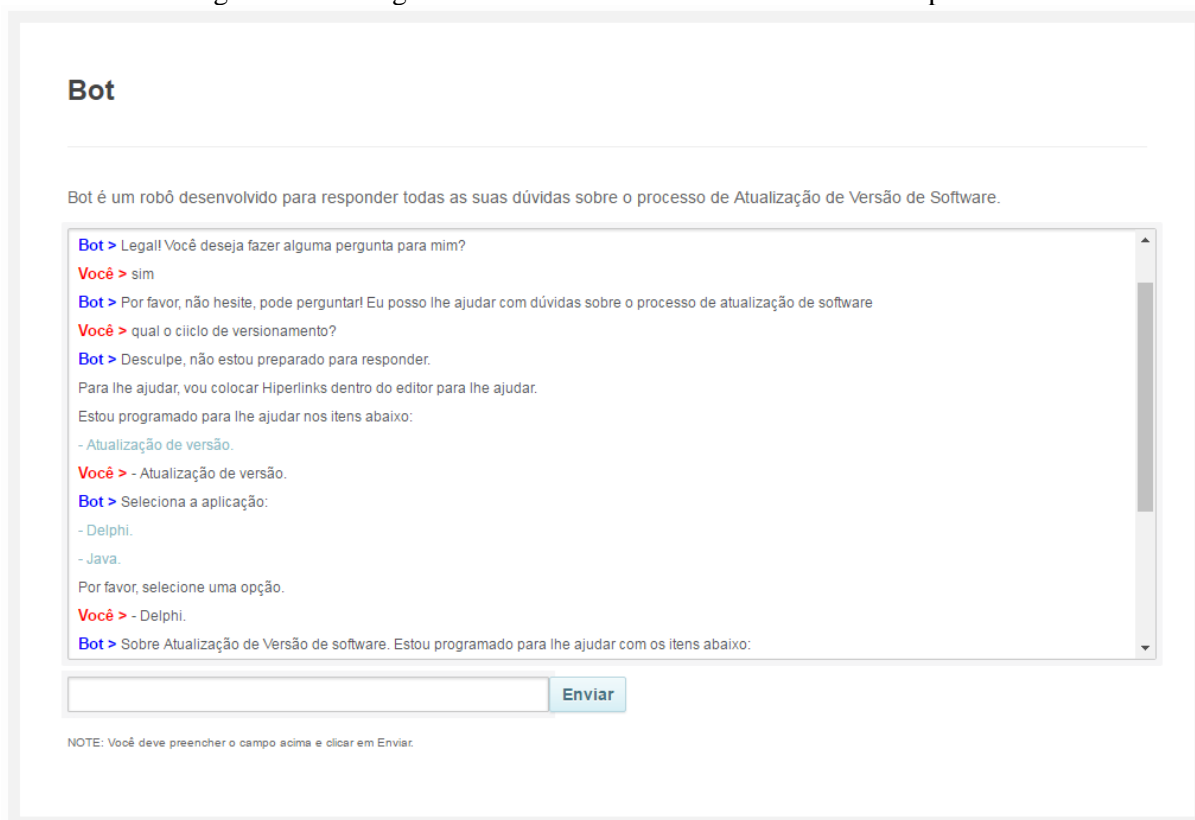
Bot > Por favor, não hesite, pode perguntar! Posso lhe responder sobre o processo de atualização de versão de Software. Posso lhe dar informações de arquivos necessários, possíveis correções para erros, pré-requisitos para atualização da versão.

Você > qual é o ciclo de versionamento?

Bot > A versão do Software é liberada a cada 15 dias.

NOTE: Você deve preencher o campo acima e clicar em Enviar.

Caso o usuário faça uma pergunta que o *chatterbot* não consiga responder, é iniciado um diálogo com o usuário utilizando *hiperlinks*. Os *hiperlinks* são adicionados no campo texto que mantém o histórico do diálogo com o usuário. Para melhor entendimento, a Figura 22 ilustra um diálogo com o usuário, em que, devido a erros de português na pergunta formulada pelo usuário, o *chatterbot* não conseguiu interpretar o diálogo. Assim, iniciou um diálogo através de *hiperlinks* como forma de auxiliar o usuário na conversa. Conforme o usuário clica sobre os *hiperlinks*, o diálogo vai se desenvolvendo.

Figura 20 - Diálogo entre o *chatterbot* e o usuário utilizando hiperlinks

Pode-se observar na Figura 22 que o usuário pode consultar o histórico da conversa a qualquer momento, para isso, é necessário rolar a barra de rolagem do campo texto que armazena o histórico do diálogo com o *chatterbot*.

### 3.4 RESULTADOS E DISCUSSÕES

A solução desenvolvida conseguiu apresentar um *chatterbot* para interação com o usuário auxiliando-o no atendimento de suporte ao processo de atualização de versão de software. Além disso, foram criados *hiperlinks* para auxiliar o usuário no diálogo com o *chatterbot*.

Em testes simulando um diálogo com o *chatterbot* foi possível sanar dúvidas sobre o processo de atualização de software. Também foram realizados testes com dois usuários que participam do processo de suporte a atualização de versão de software. Apesar da amostra não ser expressiva, os resultados obtidos foram positivos, haja vista que os usuários conseguiram dialogar com o *chatterbot*. Mais detalhes sobre os testes realizados com os usuários está apresentado no apêndice C. Ainda, visando auxiliar o usuário na interação com o *chatterbot*, caso seja enviada uma sentença que o *chatterbot* não consiga obter respostas ao consultar a base de conhecimentos, são adicionados *hiperlinks* ao diálogo para auxiliar o usuário na resolução da dúvida ao processo. Durante os testes com os usuários e desenvolvimento, foram



identificados alguns problemas e dificuldades na criação da base de conhecimento AIML para interação com o usuário a partir de sentenças do usuário, devido as formas de comunicações serem específicas de cada usuário.

O primeiro a ser destacado foi a criação da base de conhecimento AIML. A base de conhecimentos AIML foi desenvolvida de forma manual. Esta atividade consumiu muito tempo devido a separação por assuntos. Foi necessário manter vários arquivos `.aiml` das bases AIML abertos para realizar a interação entre as sentenças. Esse fato, torna o processo repetitivo e com grande possibilidade de erro, uma vez que não há um analisador das sentenças já criadas, afim de verificar se alguma categoria está similar a outra, podendo causar uma sobrescrita de categorias, afetando assim, a resposta na interação com o usuário, ou até, retornando uma orientação incorreta.

Outro problema é a gestão da base de conhecimentos. Como são arquivos de extensões `.aiml` contendo inúmeras *tags*, é necessário aplicar uma gestão de simples assimilação, por nomes, ou fazer uma organização dos arquivos de forma que no futuro, com o aumento da base de conhecimentos, cada arquivo `.aiml` contenha apenas sentenças de um determinado interesse. Caso isso não seja feito, a manutenção da base de conhecimentos AIML pode se tornar inviável, devido à má documentação ou organização das sentenças dentro da base de conhecimentos AIML.

Durante o desenvolvimento e testes, outra dificuldade encontrada foi a identificação de um padrão de categoria para as sentenças digitadas pelos usuários com erros ortográficos. Em virtude do sistema de identificação de padrões AIML ser necessário o casamento de um padrão de reconhecimento, o erro ortográfico pode causar o não casamento com uma categoria. Por exemplo, o usuário digitou a sentença “esto com problema na atualização”, devido ao erro na digitação da palavra “estou”, por “esto”, não é encontrado uma categoria para resposta a sentença do usuário.

A acentuação também foi uma problemática enfrentada durante todo o desenvolvimento e testes. A maioria das frase com acentuação, foram repetidas sem os acentos para garantia de casamento de padrão. Com isso, a base de conhecimento fica repetitiva, com muitas recursões entre a base de conhecimento AIML para casamento com o padrão principal, que seria o padrão correto, caso a sentença do usuário fosse digitada corretamente. Um exemplo da situação mencionada é a frase, “Quando será liberada a próxima versão”. Esta frase possui duas palavras com acento, permitindo que o usuário a digite de duas formas diferentes. Para esse tipo de situação em específico, foi criado dois

padrões de casamento: um com acento na palavra “próximo”, e outro sem acento na palavra “próximo”, criando assim, uma resposta caso o usuário não coloque acentuação na frase.

O Quadro 21 apresenta de forma comparativa algumas características em relação aos trabalhos correlatos e ao trabalho apresentado nesta pesquisa.

Quadro 21 - Quadro comparativo entre a pesquisa e os trabalhos correlatos

	bot	Oliveira (2015)	Machado (2005)	Corrêa (2010)
linguagem da base de conhecimento	AIML	AIML	AIML	AIML/iAIML
uso de ontologia	não	sim	não	não
linguagem de programação para página web	Java Web	Java Web	PHP	Phyton
área administrativa	não	sim	sim	não
criação <i>tags</i> adicionais	sim	sim	não	não
faz uso de banco de dados	sim	não	não	não
<i>hyperlinks</i> para auxílio a comunicação	sim	não	não	não
interpretador AIML	Program AB	Program AB	Program D	PyAIML

Dentre os trabalhos correlatos exemplificados e a solução desenvolvida, não se identifica um projeto que tenha o mesmo objetivo do proposto. Todavia, apresentam situações semelhantes em determinados aspectos entre todos, como: interação com o usuário utilizando o idioma português e todos possuem uma página web para interação com o usuário. Alguns requisitos são específicos do aqui apresentado e ao de Oliveira (2015), como, o interpretador utilizado, a criação de *tags* adicionais e o uso da linguagem para a página *web*. Mas a solução aqui proposta, possui características únicas, como, uso de banco de dados para auxílio na montagem das respostas ao usuário e a comunicação com a utilização de *hyperlinks* para auxílio a comunicação com o usuário.

## 4 CONCLUSÕES

O principal objetivo da solução era a disponibilização de uma *chatbot* para interação com o usuário para auxílio ao processo de atualização de versão de software. Este objetivo foi alcançado, disponibilizando um *chatbot* para diálogo. Além disso, foram disponibilizados também uma interação com banco de dados para auxílio na construção de respostas ao usuário e, diálogo a partir de *hiperlinks* com o usuário para auxílio. Estes dois objetivos não estavam relacionados inicialmente para a solução.

A base de conhecimento AIML foi a utilizada para desenvolvimento da base de conhecimentos para auxílio sobre o processo de atualização de software. Como a base de conhecimento AIML trabalha com casamento de padrões para respostas ao usuário, o objetivo de dialogar no idioma português com o usuário foi atingido, devido à base de conhecimento ser toda desenvolvida no mesmo idioma. Caso se queira utilizar outro idioma, é necessário que toda a base de conhecimento seja reescrita no idioma desejado.

Para interação com o usuário foi desenvolvida uma página *web*, em que é possível enviar sentenças ao *chatbot* e consultar o históricos das sentenças. A página *web* foi desenvolvida para tornar o diálogo com o *chatbot* simples e objetivo. A partir disso, o objetivo da solução, na qual visa a disponibilização de uma página *web* para interação com o usuário foi alcançado.

A base de conhecimentos desenvolvida foi dividida por assuntos, possibilitando ao usuário gerenciador do *chatbot*, o incremento da base de conhecimentos. A base de conhecimentos pode ser incrementada a partir de novas situações vividas pelos usuários do suporte de software. O incremento da base de conhecimentos deve ser feito de forma manual através da edição de arquivos AIML (.aiml), isto visa segurança nas informações contidas na base de conhecimentos e, garante que somente os usuários autorizados poderão incrementar a base de conhecimentos com novos padrões.

Devido à base de conhecimentos AIML ser desenvolvida manualmente foi necessário um grande esforço para a criação, uma vez que foi criado padrão a padrão e, em outras situações, quando já se está desenvolvendo um diálogo com várias etapas e se faz preciso adicionar apenas um detalhe entre este diálogo, é necessário testar o diálogo completo, afim de verificar se outros padrões foram afetados. A não existência da possibilidade de criação de cenários de testes para os padrões, tornou o desenvolvimento da base de conhecimentos trabalhoso. Isto se deve em virtude de cada novo padrão criado ou alterado ser necessário

testar todos os cenários novamente de forma manual, necessitando de um grande esforço que, poderia ser automatizado.

A utilização de um banco de dados para simulação da busca de informações ao banco de dados de um software para gestão hospitalar, auxiliou na criação de um cenário real do suporte ao processo de atualização de software, em que a consulta auxilia o *chatterbot* na criação de respostas dinâmicas. Sendo assim, a cada versionamento do software, o *chatterbot* apenas consulta a base de dados para obter informações sobre o versionamento do software. Desta forma, não se faz necessário a manutenção a base de conhecimento AIML para adição das informações após a geração da nova versão.

Também, vale destacar a criação de *hyperlinks* para diálogo com o *chatterbot*. Este recurso é ativado a partir do momento que o *chatterbot* não possui resposta para uma sentença do usuário, visando sanar a dúvida de forma mais objetiva. A forma de escrita também pode influenciar no não casamento com os padrões desenvolvidos, o que pode ser considerado uma dificuldade da pesquisa, uma vez que, utilizando a linguagem AIML é necessário criar vários casamentos de padrões, afim de abranger a maior possibilidade de sentenças dos usuários.

Dentre os objetivos alcançados, vale destacar ainda que, com a possibilidade do usuário poder dialogar com o *chatterbot* sobre o processo de atualização de versão de software, há ganho de tempo na resolução de seu problema e não se faz necessário o acionamento do suporte da empresa. Dessa forma, o usuário ganhará tempo e, a empresa não necessitará disponibilizar um colaborador para resolução da situação.

Por fim, o *chatterbot* apresentou algumas dificuldades devido a limitações da linguagem AIML. Contudo essas limitações podem ser superadas com o desenvolvimento de trabalhos futuros.

#### 4.1 EXTENSÕES

São sugeridas as seguintes extensões para a continuidade do *chatterbot*:

- a) representar as respostas do *chatterbot* através de um agente virtual;
- b) permitir a interação com o *chatterbot* através de voz;
- c) tradução da base de conhecimentos para outros idiomas de forma automatizada;
- d) criação de ferramenta para criação e manutenção da base de conhecimentos AIML;
- e) criação de ferramenta para automatização do incremento da base de conhecimentos AIML a partir de situações vividas pelos usuários do software;
- f) criação de ferramenta para análise de textos e geração de sentenças de modo automatizado;

- g) criação de algoritmo para identificação de palavras por som, visando a compreensão do *chatterbot* a erros ortográficos na construção da sentença do usuário;
- h) abrir atendimento com o histórico do diálogo caso o *chatterbot* não consiga resolver;
- i) armazenamento das iterações com os usuários para posterior análise e criação de novos padrões.

## REFERÊNCIAS

- 7GRAUS. **Significado upgrade**. [S.l.], 2011. Disponível em: <<http://www.significados.com.br/upgrade/>>. Acesso em: 23 jan. 2016.
- ASSENSIO, Claudia. **Ibope aponta que acesso à internet cresce 3% no 2º trimestre**. [S.l.], 2013. Disponível em: <<http://exame.abril.com.br/tecnologia/noticias/ibope-aponta-que-acesso-a-internet-cresce-3-no-2o-tri>>. Acesso em: 24 jan. 2016.
- BRANSKI, Regina M. **Recuperação de informações na web**. [S.l.], 2003. Disponível em: <[http://www.brapci.inf.br/\\_repositorio/2010/11/pdf\\_7b0e618ad3\\_0012984.pdf](http://www.brapci.inf.br/_repositorio/2010/11/pdf_7b0e618ad3_0012984.pdf)>. Acesso em: 14 fev. 2016.
- COMARELLA, Rafaela L.; CAFÉ, Ligia M. A. Chatterbot: conceito, características, tipologia e construção. **Informação & Sociedade: estudos**, João Pessoa, n. 2, p. 55-67, 2008. Disponível em: <<http://www.ies.ufpb.br/ojs/index.php/ies/article/view/1758>>. Acesso em: 23 fev. 2016.
- CORRÊA, Abel. **Robô de conversação aplicado a educação a distância como tutor inteligente**. 2010. 68 f. Monografia (Especialização) - Curso de Especialização em Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <<https://www.lume.ufrgs.br/bitstream/handle/10183/79662/000895318.pdf?sequence=1>>. Acesso em: 25 jan. 2016.
- GOMES, Fabio. **Introdução ao Java Server Pages - JSP**. [S.l.], 2012. Disponível em: <<http://www.devmedia.com.br/introducao-ao-java-server-pages-jsp/25602>>. Acesso em: 15 fev. 2016.
- FRANKLIN, Stan; GRAESSER, In: THIRD INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, 3., 1996, Memphis. **Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents**. Memphis: Springer-verlag, 1996. p. 1 - 11. Disponível em: <<http://www.inf.ufrgs.br/~alvares/CMP124SMA/IsItAnAgentOrJustAProgram.pdf>>. Acesso em: 8 mar. 2016.
- GIACOMELI, Suelen. **Relatório/resumo: a internet no Brasil em 2015**. [S.l.], 2015. Disponível em: <<http://blog.pmweb.com.br/a-internet-no-brasil-em-2015/>>. Acesso em: 15 fev. 2016.
- LEONHARTDT, Michelle D. **Doroty: um chatterbot para treinamento de profissionais atuantes no gerenciamento de redes de computadores**. 2005. 110 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <<http://www.lume.ufrgs.br/bitstream/handle/10183/5659/000473673.pdf?sequence=1>>. Acesso em: 15 fev. 2016.
- MACHADO, Fabrizio C. **Integrando chatterbot e agente animado de interface em um ambiente virtual de aprendizagem**. 2005. 68 f. Monografia (Especialização) - Curso de Pós-graduação Especialização em MBA e Banco de Dados, Universidade do Extremo Sul Catarinense-UNESC, Criciúma. Disponível em: <<http://www.bib.unesc.net/biblioteca/sumario/000028/00002807.pdf>>. Acesso em: 24 jan. 2016.
- MARCUSCHI, Luiz Antônio. **Análise da Conversação**. 1991. Editora Ática. 2º Edição.

MARIETTO, Maria G. B. et al. Artificial intelligence markup language: a brief tutorial. **International Journal of Computer Science & Engineering Survey**, China, v. 4, n. 3, p. 1-19, 2013. Disponível em: <<http://www.airccse.org/journal/ijcses/papers/4313ijcses01.pdf>>. Acesso em: 7 mar. 2016.

MAULDIN, MICHAEL L., 12., 1994, Pittsburgh. CHATTERBOTS, TINYMUDS, and the Turing Test Entering the Loebner Prize Competition. Pittsburgh: Aai-94 Proceedings, 1994. 5 p. Disponível em: <<http://aaai.org/Papers/AAAI/1994/AAAI94-003.pdf>>. Acesso em: 23 fev. 2016.

MELBOURNE, The Royal. **Update regarding Melbourne Health computer virus**. [S.l.], 2016. Disponível em: <<https://www.thermh.org.au/news/update-regarding-melbourne-health-computer-virus/>>. Acesso em: 23 fev. 2016.

MOURA, Thiago J. M. **Um chatterbot para aquisição automática de perfil do usuário**. 2003. 127 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós Graduação em Ciência da Computação, Universidade Federal de Pernambuco, Recife. Disponível em: <<http://www.liber.ufpe.br/teses/arquivo/20050228150106.pdf>>. Acesso em: 10 fev. 2016.

OLIVEIRA, Bruno de. **Chatterbot para esclarecimento de dúvidas sobre as formas de ingresso em cursos da FURB**. 2015. 66 f. TCC (Graduação) - Curso de Ciência da Computação, Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau, Blumenau, 2015. Disponível em: <[http://dsc.inf.furb.br/arquivos/tccs/monografias/2015\\_1\\_bruno-de-oliveira\\_monografia.pdf](http://dsc.inf.furb.br/arquivos/tccs/monografias/2015_1_bruno-de-oliveira_monografia.pdf)>. Acesso em: 24 jan. 2016.

SIMÕES, Pedro. **SOs móveis: iOS 7 com 90% de adoção, KitKat apenas com 10%**. [S.l.], 2014. Disponível em: <<http://pplware.sapo.pt/informacao/sos-moveis-ios-7-com-90-de-adopcao-kitkat- apenas-com-10/>>. Acesso em: 23 fev. 2016.

TEIXEIRA, Sérgio. **Chatterbots: uma proposta para a construção de bases de conhecimento**. 2005. 100 f. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória. Disponível em: <<http://www.multicast.com.br/sergio/tuxbot-dissertacao-mestrado-sergio-teixeira.pdf>>. Acesso em: 12 fev. 2016.

TEIXEIRA, Sérgio; MENEZES, Crediné S. de. **Facilitando o uso de ambientes virtuais através de agentes de conversação**. [S.l.], 2003. Disponível em: <<http://www.nce.ufrj.br/sbie2003/publicacoes/paper48.pdf>>. Acesso em: 24 jan. 2016.

TURING, Alan M. **Computing machinery and intelligence**. [S.l.], 1950. Disponível em: <<http://www.abelard.org/turpap/turpap.php>>. Acesso em: 24 jan. 2016.

WALLACE, Richard. **Artificial Intelligence Markup Language (AIML)** - version 1.0.1. [S.l.], 2001. Disponível em: <<http://www.alicebot.org/TR/2001/WD-aiml/>>. Acesso em: 11 fev. 2016.

WEIZENBAUM, Joseph; (Ed.). ELIZA: a computer program for the study of natural communications between man and machine: A. G. OETTINGER. **Communications Of The Acm**, Cambridge, v. 9, n. 1, p.36-45, jan. 1966. Disponível em: <<http://web.stanford.edu/class/linguist238/p36-weizenbaum.pdf>>. Acesso em: 10 fev. 2016.

## APÊNDICE A – Detalhamentos dos casos de uso

Nesta seção são apresentados os detalhamentos dos casos de uso, com descrição, ator, pré-condição e cenário. O quadro 22 apresenta o caso de uso 01 (Dialogar com o chatterbot através de perguntas), o quadro 23 apresenta o caso de uso 02 (Consultar o histórico da conversa com o chatterbot) e o quadro 24 apresenta o caso de uso 03 (Dialogar com o chatterbot através de hiperlinks).

Quadro 22 - Caso de uso: Dialogar com o *chatterbot* através de perguntas

Número	UC01
Caso de Uso	Dialogar com o <i>chatterbot</i> através de perguntas
Ator	Usuário
Pré-condição	Nenhuma
Cenário Principal	<ol style="list-style-type: none"> <li>1. O usuário entra na página <i>web</i>.</li> <li>2. O usuário digita uma sentença.</li> <li>3. O usuário envia a sentença ao <i>chatterbot</i>;</li> <li>4. A resposta do <i>chatterbot</i> é adicionado ao campo de interação com o usuário.</li> </ol>

Quadro 23 - Caso de uso: Consultar o histórico da conversa com o *chatterbot*

Número	UC02
Caso de Uso	Consultar o histórico da conversa com o <i>chatterbot</i>
Ator	Usuário
Pré-condição	Nenhuma
Cenário Principal	<ol style="list-style-type: none"> <li>1. O usuário entra na página <i>web</i>.</li> <li>2. O usuário digita uma sentença.</li> <li>3. O usuário envia a sentença ao <i>chatterbot</i>;</li> <li>4. A resposta do <i>chatterbot</i> é adicionado ao campo de interação com o usuário.</li> <li>5. O usuário consulta o campo texto que ficam armazenados o diálogo com o <i>chatterbot</i>. Para consultar históricos que não aparecem mais no campo texto, é possível utilizar a barra de rolagem para movimentações do campo texto.</li> </ol>

Quadro 24 - Caso de uso: Dialogar com o *chatterbot* através de hiperlinks

Número	UC03
Caso de Uso	Dialogar com o <i>chatterbot</i> através de hiperlinks
Ator	Usuário
Pré-condição	Nenhuma
Cenário Principal	<ol style="list-style-type: none"> <li>1. O usuário entra na página <i>web</i>.</li> <li>2. O usuário digita uma sentença.</li> <li>3. Caso o <i>chatterbot</i> não possua resposta a sentença digitada pelo usuário. Iniciasse o dialogo a partir de hiperlinks.</li> <li>4. <i>Chatterbot</i> inicia o dialogo a partir de hiperlinks.</li> <li>5. Usuário clica no campo texto onde são apresentados os hiperlinks para selecionar a opção.</li> </ol>



## APÊNDICE B – Detalhamento da base de conhecimentos AIML utilizadas pelo *chatterbot*

No quadro 25 é apresentado o nome das bases de conhecimentos AIML e a descrição do seu propósito.

Quadro 25 - Bases de conhecimentos e seu propósito

introduction	Base de conhecimento responsável por fazer a saudação inicial ao usuário
mainoptions	Base de conhecimento responsável por apresentar o menu principal de respostas a partir do diálogo com hiperlinks
problens_upgrade	Base de conhecimento que contempla informações gerais sobre a atualização de versão de ambas aplicações e, sobre inicia o diálogo sobre problemas na atualização de versão de software
problens_upgrade_delphi	Base de conhecimento responsável por problemas na atualização de versão da aplicação desenvolvida em Delphi
problens_upgrade_java	Base de conhecimento responsável por problemas na atualização de versão da aplicação desenvolvida em Java
start	Base de conhecimento responsável por solicitar o estado do usuário
startHiperlink	Base de conhecimento responsável por iniciar o diálogo com hiperlinks com o usuário
upgradeVersionDelphHelpPharse1	Base de conhecimento responsável por conter auxílios ao usuário referente a fase 1 da atualização de versão de software desenvolvida em Delphi
upgradeVersionOptions	Base de conhecimento responsável por selecionar a versão de interesse no dialogo através de hiperlinks
upgradeVersionOptionsDelphi	Base de conhecimento responsável pela montagem do menu de opções de auxílios ao usuário sobre a atualização de versão de software desenvolvida em Delphi
upgradeVersionOptionsJava	Base de conhecimento responsável pela montagem do menu de opções de auxílios ao usuário sobre a atualização de versão de software desenvolvida em Java
upgradeVersionReturnDelphi	Base de conhecimento responsável por respostas sobre o processo de atualização de versão de software desenvolvido em Delphi. Para o diálogo através de hiperlinks
upgradeVersionReturnJava	Base de conhecimento responsável por respostas sobre o processo de atualização de versão de software desenvolvido em Java. Para o diálogo através de hiperlinks
utils	Base de conhecimento responsável por manter padrões para auxilio a montagem dos menus no diálogo através de hiperlinks

## APÊNDICE C – Testes do *chatterbot* por usuários que participam do processo de atualização de versão de software

Esta seção tem como objetivo apresentar os resultados dos testes realizados com dois usuários que participam do processo de atualização de versão de Software. Foi disponibilizada uma máquina para que os dois usuários realizassem a interação com o *chatterbot*. Ao final, foram realizadas as perguntas que estão descritas no Quadro 26 afim de obter a experiência no diálogo com o *chatterbot*. Nos Quadros 27 e 28 estão descritas as respostas dos usuários as perguntas e, nas Figuras 23 e 24 uma imagem do diálogo realizado com o *chatterbot* durante os testes.

Quadro 26 - Perguntas realizadas aos usuários

- Você vê relevância neste trabalho?
- Você conseguiu dialogar com o Chatterbot?
- Qual foi sua dúvida?
- Você acredita que poderia ajudar o cliente sem necessidade de contato com nosso suporte?
- Você acredita que este processo possa auxiliar nosso usuário do suporte ao auxílio com o cliente durante a atualização de versão?
- Acredita que este processo possa agilizar o atendimento ao cliente?

Quadro 27 - Respostas usuário responsável pelo suporte a atualização de versão

Teste 1: Usuário que participa do suporte a atualização de versão de software
- Você vê relevância neste trabalho? R: Tem, porque mesmo que tenha muita coisa em manual. O cliente não deseja realizar a leitura e com esta forma dinâmica, eles poderiam interagir com o <i>chatterbot</i> .
- Você conseguiu dialogar com o Chatterbot? R: Sim.
- Qual foi sua dúvida? R: Quando será liberada a versão 1666.
- Você acredita que poderia ajudar o cliente sem necessidade de contato com nosso suporte? R: Sim, desde que as frases possam ser montadas de forma completa. O modo <i>hyperlink</i> é uma boa orientação automática.
- Você acredita que este processo possa auxiliar nosso usuário do suporte ao auxílio com o cliente durante a atualização de versão? R: Sim, principalmente com o pessoal novo. Onde não seria preciso parar alguém para treinar. Fica mais intuitivo para aprendizagem.
- Acredita que este processo possa agilizar o atendimento ao cliente? R: Pode, porque se o usuário(cliente) fez todos os passos que o <i>chatterbot</i> orientou. O nosso suporte pode conectar direto e saber que X passos (padrões) já foram realizados pelo cliente quando o <i>chatterbot</i> orientou.

Figura 21 – Diálogo usuário responsável pelo suporte a atualização de versão de software

**Bot**

Bot é um robô desenvolvido para responder todas as suas dúvidas sobre o processo de Atualização de Versão de Software.

Bot > Olá! tudo bem?

Você > não

Bot > Espero que fique bem logo. Posso ajudar com alguma coisa?

Você > sim

Bot > Por favor, não hesite, pode perguntar! Eu posso lhe ajudar com dúvidas sobre o processo de atualização de software

Você > quando será liberada a versão 1666?

Bot > A versão 1666 será liberada no dia 18/10/2016

Enviar

NOTE: Você deve preencher o campo acima e clicar em Enviar.

Quadro 28 - Resposta gestora da área que participa do suporte a atualização de versão

Teste 2: Gestora da área responsável pelo suporte a atualização de versão de software
- Você vê relevância neste trabalho? R: Sim. Acredito que poderia ser utilizado para uma pré-triagem antes de vir ao atendente, caso o <i>chatterbot</i> não consiga resolver. Poderíamos ter uma diminuição na demanda caso o <i>chatterbot</i> consiga resolver. Ao fim do dia, bem desenvolvido o <i>chatterbot</i> , poderia diminuir o custo sobre o processo. Utilizar deste recurso para novos serviços.
- Você conseguiu dialogar com o Chatterbot? R: Consegui.
- Qual foi sua dúvida? R: Objetos inválidos.
- Você acredita que poderia ajudar o cliente sem necessidade de contato com nosso suporte? R: Sim, poderia.
- Você acredita que este processo possa auxiliar nosso usuário do suporte ao auxílio com o cliente durante a atualização de versão? R: Sim, como se utilizasse o FAQ.
- Acredita que este processo possa agilizar o atendimento ao cliente? R: Sim, nos casos que ele pode esperar. Hoje se perde muito tempo com informações básicas do cliente, como por exemplo um <i>print</i> do erro, mensagens adicionais padrões. Com este recurso o <i>chatterbot</i> me enviaria estas informações, me auxiliando e ganhando tempo no atendimento.

Figura 22 - Diálogo gestora da área que participa do suporte a atualização de versão

**Bot**

---

Bot é um robô desenvolvido para responder todas as suas dúvidas sobre o processo de Atualização de Versão de Software.

**Bot >** Olá! tudo bem?

**Você >** tudo e com você?

**Bot >** Estou muito bem! Você deseja fazer alguma pergunta para mim?

**Você >** sim

**Bot >** Por favor, não hesite, pode perguntar! Eu posso lhe ajudar com dúvidas sobre o processo de atualização de software

**Você >** estou com objetos inválidos

**Bot >** Caso esteja com objetos inválidos na base de dados. Para correção dos mesmos pode ser seguido os passos:

- Executar o comando `Select count(1) from inv_v;` (saber a quantidade de objetos inválidos), exec `valida_objetos_sistema;` (executar a validação dos objetos). Caso após a execução do `VALIDA_OBJETOS_SISTEMA`, os objetos permanecerem inválidos, é necessário seguir a próxima dica;
- Executar o comando `select * from inv_v3;` (irá identificar o objeto raiz que esta deixando os demais objetos inválidos). O retorno deste `Select` e o arquivos de log da atualização de versão que estão na pasta `DIR_VERSAO` devem ser enviados anexados a uma Ordem de Serviço para analisar a causa.

NOTE: Você deve preencher o campo acima e clicar em Enviar.