UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO PARA SUMARIZAÇÃO AUTOMÁTICA DE TEXTOS ESCRITOS EM LÍNGUA PORTUGUESA

ALEXANDRE BUSARELLO

BLUMENAU
2013

2013/1-03

ALEXANDRE BUSARELLO

PROTÓTIPO PARA SUMARIZAÇÃO AUTOMÁTICA DE TEXTOS ESCRITOS EM LÍNGUA PORTUGUESA

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Profa. Joyce Martins, Mestre - Orientadora

BLUMENAU
2013

2013/1-03

PROTÓTIPO PARA SUMARIZAÇÃO AUTOMÁTICA DE TEXTOS ESCRITOS EM LÍNGUA PORTUGUESA

Por

ALEXANDRE BUSARELLO

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

Presidente:	Profa. Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB
Membro:	Prof. Roberto Heinzle, Doutor – FURB

Blumenau, 08 de julho de 2013

Dedico este trabalho à minha esposa, filha, família e a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

Primeiramente, a Deus, que é a fonte de tudo.

À minha esposa e filha, que são as pessoas que mais amo neste mundo.

Ao meu Pai e à minha Mãe, que me apoiaram desde o início.

À minha orientadora, Joyce Martins, pelo apoio e dedicação em me ajudar a concluir este trabalho.

Para conhecermos os amigos é necessário passar pelo sucesso e pela desgraça. No sucesso, verificamos a quantidade e, na desgraça, a qualidade.

Confúcio

RESUMO

Este trabalho apresenta a especificação e a implementação de um protótipo de ferramenta para realizar a sumarização (extrato) de textos escritos em língua portuguesa. Compõe-se de três etapas: análise, transformação e síntese. Na primeira etapa é usado o analisador morfológico Palavras. A etapa seguinte utiliza o conceito de abordagem superficial com a técnica de sumarização por palavra-chave. Através do algoritmo de Extração de Palavras-Chave baseado em frequência de Padrões (EPC-P), foi possível extrair as palavras-chave de um texto e classificar as sentenças pelo número de ocorrências de palavras-chave. Na etapa de síntese, as sentenças classificadas são ordenadas e é gerado o sumário.

Palavras-chave: Sumarização automática. Processamento de linguagem natural. EPC-P.

ABSTRACT

This paper presents the specification and implementation of a prototype tool to perform the summarization (extract) of texts written in Portuguese language. It consists of three stages: analysis, transformation and synthesis. In the first step the morphological analyzer Palavras is used. The next step uses the concept of superficial approach to the technique of summarization by keyword. Through extraction algorithm based Keyword Frequency Standards (EPC-P), it was possible to extract the keywords from a text and rank sentences by the number of occurrences of keywords. In synthesis step, the sentences are ordered and classified.

Key-words: Automatic summarization. Natural language processing. EPC-P.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

EA – *Enterprise Architect*

EDU – *Elementary Discourse Unit*

EPC-P – Extração de Palavras-Chave baseado em frequência de Padrões

HTML – *HyperText Markup Language*

IDE – *Integrated Development Environment*

PLN – Processamento de Linguagem Natural

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

RST – *Rhetorical Structure Theory*

UML – *Unified Modeling Language*

VISL - *Visual Interactive Syntax Learning*

SUMÁRIO

1 INTRODUÇÃO

Sumarização é uma palavra originada do verbo latino *sumere* que significa reduzir, diminuir, sintetizar (HEERDT, 1997). Desta forma, é possível dizer que o processo de sumarizar um texto consiste em resumi-lo eliminando o que é irrelevante sem alterar o foco central. No cotidiano as pessoas fazem uso da sumarização muitas vezes sem perceber. Por exemplo, ao narrar um evento a uma pessoa, realiza-se uma síntese do que aconteceu, ao invés de descrever de forma exata o ocorrido (MARTINS et al., 2001, p. 3). Assim sendo, é feita uma sumarização. A sumarização automática é a sintetização de um texto de forma automatizada por programas de computador. Cada pessoa possui uma forma única de organizar as informações relevantes, refletindo assim no sumário manual gerado. O mesmo ocorre com programas de computadores que reproduzem a sumarização automática: programas diferentes geram sumários diferentes.

A sumarização automática é composta de três etapas macro que são: análise, transformação e síntese. Na etapa de análise é feito o processamento de um ou mais textos fontes, gerando como saída a representação interna de todo conteúdo analisado. A etapa de transformação executa o processo de sumarização com base na representação interna gerada pela análise tendo como saída a reprodução interna do sumário. Por fim, a etapa de síntese transforma o conteúdo da reprodução interna do sumário em forma de língua natural (PARDO, 2008, p. 6).

Diante do exposto, propõe-se o desenvolvimento de uma ferramenta para efetuar a sumarização automática de textos, aplicando as etapas descritas anteriormente. A ferramenta tem como saída resumos gerados a partir de textos escritos em língua portuguesa, contendo apenas informações consideradas relevantes. São usados como entrada textos de notícias.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo para resumir textos, visando auxiliar no processo de sumarização automática.

Os objetivos específicos do trabalho são:

a) sumarizar textos de artigos de notícias;
b) processar textos escritos de acordo com as normas gramaticais da língua portuguesa;
c) utilizar uma técnica da abordagem superficial para sumarização.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão. O próximo capítulo apresenta os aspectos teóricos estudados para o desenvolvimento do trabalho. São relatados temas como processamento de linguagem natural, abordagens da sumarização automática, etapas da sumarização e o método de sumarização por palavra-chave utilizado no desenvolvimento do protótipo. Também são relacionados alguns trabalhos correlatos. O capítulo 3 aborda o desenvolvimento propriamente dito, detalhando os requisitos, a especificação e a implementação, além de trazer resultados e discussões. Por fim, no capítulo 4 são apresentadas as conclusões, bem como sugestões para possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

As seções foram divididas de acordo com os conteúdos estudados para o desenvolvimento do protótipo. A seção 2.1 descreve algumas das etapas do processamento de linguagem natural. Em seguida, a seção 2.2 apresenta as duas principais abordagens da sumarização automática. Na seção 2.3 estão relacionadas as etapas para execução da sumarização automática. A seção 2.4 traz o método de sumarização por palavra-chave e o algoritmo Extração de Palavras-Chave baseado em frequência de Padrões (EPC-P). Por fim, na última seção são apresentados os trabalhos correlatos.

2.1 PROCESSAMENTO DA LINGUAGEM NATURAL

Desde a introdução dos computadores digitais, no início dos anos 40, buscaram-se formas de fazer essas máquinas entenderem as instruções das pessoas para execução de tarefas. A primeira alternativa criada para atender essa comunicação humano-computador foi a linguagem de programação, conhecida também como linguagem de máquina. Com o passar dos anos, várias novas linguagens surgiram e se aproximaram cada vez mais da linguagem humana. Porém, por mais inteligível que as linguagens atuais possam ser, as instruções continuam tendo que ser digitadas exatamente da forma prescrita, caso contrário, obtém-se como resposta do compilador um erro de sintaxe. Outra opção cuja aplicação sem dúvida é muito mais complexa continua sendo um desafio: criar programas capazes de se comunicar com a língua natural dos humanos (SILVA et al., 2007, p. 5). O estudo do Processamento de Linguagem Natural (PLN) surgiu tendo como objetivo a resolução deste desafio.

O PLN tem inúmeras aplicações e entre elas está a análise textual. A etapa inicial da análise textual é o pré-processamento do mesmo. Nesta etapa é feito o reconhecimento e a classificação das entidades do texto através das análises léxica e morfológica, sintática e semântica. A eficiência da extração das entidades que compõem um texto em linguagem natural está diretamente ligada à complexidade do algoritmo.

> Em geral, a eficiência da extração das entidades de um texto se comporta de forma exponencial com a complexidade algorítmica. Com heurísticas simples atingimos facilmente um acerto de 80%. A dificuldade está em atingir valores acima de 90%, sabendo que 100% é praticamente impossível. (ARANHA, 2007, p. 60).

Neste trabalho, a utilização do PLN limitou-se às analises léxica e morfológica, onde são extraídos os *tokens* (palavra), assim como determinada a classe gramatical das palavras do texto.

A análise léxica consiste na extração e classificação dos *tokens*. Um *token* é uma cadeia de caracteres que representa uma parte de uma estrutura, sendo que em linguagem natural ele representa uma palavra ou símbolo. Os *tokens* devem ser classificados de acordo com o seu tipo, como por exemplo os *tokens* que correspondem às palavras “gato”, “cachorro” e “flor” podem ser categorizados como substantivo. Nas linguagens de programação de computadores, o *token* tem um sentido único e nunca será expresso por mais de uma palavra. Já na língua portuguesa pode-se ter várias palavras com um mesmo significado, ou palavras iguais com significados diferentes, dependendo do contexto. Isto é, dependendo do contexto da oração, as palavras que são substantivo podem, por exemplo, virar adjetivo ou verbo. Esse tipo de situação irá aparecer na análise morfológica da palavra de acordo com o contexto do texto.

A análise morfológica na língua portuguesa classifica as palavras de acordo com sua categoria gramatical. No PLN sua função é exatamente a mesma, porém o que será classificado é o *token* correspondente às palavras. Esta classificação também é conhecida como etiquetagem, que consiste na identificação das classes morfológicas criando uma “etiqueta” para cada classe. Por exemplo, na frase “Eu tropecei na pedra”, a etiquetagem pode ser feita da seguinte forma: (eu, PPE), (tropecei, VP), (na, PAF), (pedra, SSF) (MÜLLER, 2003, p. 4). No Quadro 1 tem-se o significado de cada etiqueta.

Quadro 1 – Etiquetagem da frase “Eu tropecei na pedra”

etiqueta	**descrição**	**palavra**
PPE	pronome pessoal	eu
VP	verbo no passado	tropecei
PAF	preposição + artigo feminino	na
SSF	substantivo singular feminino	pedra

Fonte: adaptado de Müller (2003, p. 5).

As análises léxica e morfológica podem ser tratadas como uma única etapa, pois ambas dependem uma da outra.

2.2 ABORDAGENS DA SUMARIZAÇÃO AUTOMÁTICA

Existem três abordagens conhecidas para sumarização automática que são: superficial, profunda ou híbrida (mescla as duas técnicas anteriores).

A abordagem superficial abrange técnicas estatísticas para criar a sumarização textual. Por esse motivo é tida como a abordagem mais simples visto que não abrange a parte mais complexa do processamento de linguagem natural que é a compreensão e interpretação do texto para geração do sumário. Os sumários gerados pela abordagem superficial podem ser chamados de extratos, que são resumos que utilizam recortes das ideias consideradas principais no próprio texto fonte (BALAGE FILHO; PARDO; NUNES, 2007, p. 4). Entre as várias abordagens superficiais, destacam-se: palavras-chave, localização e relacional. O método de palavras-chave é abordado na seção 2.4.

Conforme Coracini (2009, p. 53), o método de localização considera que as informações principais de um texto estão na primeira e na última sentença de um texto e estas devem estar inclusas no sumário.

> Baxendale mostrou que, em 85% de uma amostra de 200 parágrafos, a sentença topical era a primeira e, em 7%, a última. Nos 8% restantes, as informações relevantes encontravam-se entre a primeira e a última de um mesmo parágrafo. Apesar do resultado aparentemente pouco significativo de 7% associado à última sentença, o autor considerou que esta também é importante para o sumário, pois constitui o elo de ligação com o parágrafo seguinte e, portanto, com a primeira sentença deste parágrafo, de alta topicalidade no contexto textual. (MARTINS et al., 2001, p. 10).

O método relacional ou adaptativo pode ser considerado também, segundo Black e Johnson (1988 apud MARTINS et al., 2001, p. 11), um método baseado em conhecimento. Neste método são utilizadas representações gráficas do texto, nas quais são moldadas as relações entre sentenças dependendo de suas relações semânticas. As sentenças que forem mais relacionadas com outras cujo entendimento do texto seja de maior dificuldade, recebem um peso alto e, portanto, será mais provável sua utilização no sumário gerado (MARTINS et al., 2001, p. 11). Um exemplo desta relação semântica poderia ser a ocorrência de substantivos comuns entre as sentenças.

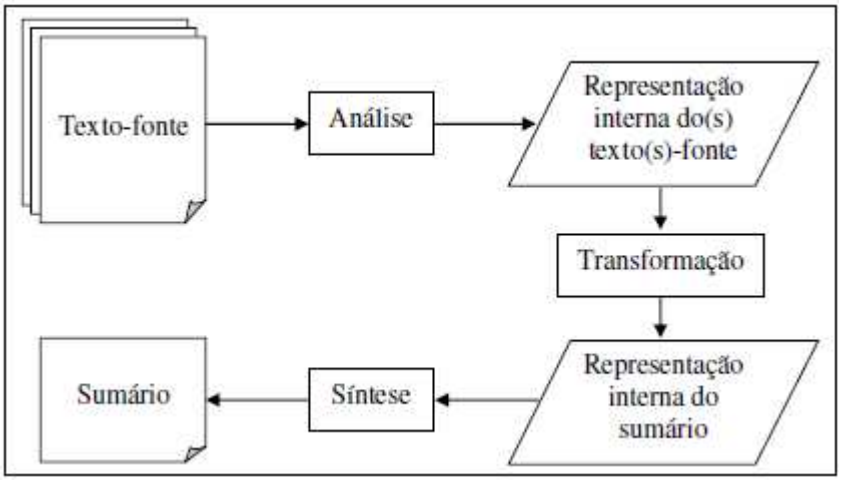
Na abordagem profunda o estudo é focado em algoritmos e técnicas que visam a interpretação. A abordagem profunda, diferentemente da abordagem superficial, utiliza métodos que tentam fazer a máquina interpretar o texto de forma a conseguir gerar um resumo similar a de um humano, ou seja, não realiza apenas recortes do texto baseado em estatísticas. Na abordagem profunda leva-se em conta todo o processamento da linguagem natural,

incluindo processos cognitivos para geração do sumário (CORACINI, 2009, p. 51). Por esse motivo, pode-se dizer que a abordagem profunda é mais complexa que a abordagem superficial. Couture (1985 apud MARTINS et al., 2001, p. 20) descreveu a ideia da abordagem profunda em três itens: identificar qual a proposição central do texto fonte; identificar quais as informações complementares podem ser úteis para colaborar com a transmissão da mensagem e preservação da ideia central; permitir estruturar os sumários com base nos itens anteriores de forma a gerar um novo texto coerente e coeso. Pela complexidade de criar um sumarizador que utilize da abordagem profunda de forma eficiente, a maioria dos trabalhos na área foca na abordagem superficial.

2.3 ETAPAS DA SUMARIZAÇÃO AUTOMÁTICA

Segundo Jones (1998 apud CARDOSO; PARDO; NUNES, 2011, p. 60), os sistemas de sumarização automática possuem uma arquitetura genérica dividida em: análise, transformação e síntese (Figura 1).

Figura 1 – Etapas da sumarização automática



Fonte: Pardo (2008, p. 6).

A entrada de um sistema de sumarização automática consiste em um ou mais textos fontes. A fase de análise irá interpretar os textos-fonte através dos analisadores léxico, sintático e semântico, a fim de extrair uma representação formal que possa ser processada automaticamente.

A transformação pode ser considerada a etapa mais importante da sumarização automática, pois é nela que será gerada a representação interna do sumário ou o sumário

propriamente dito, com base na representação de texto-fonte fornecida pela etapa anterior (análise). Nesta fase podem ser utilizados os métodos de seleção de conteúdo, agregação e substituição para realizar a compactação das informações contida nos textos-fonte. No fim desta etapa a representação interna ou final do sumário será gerada.

A síntese, que é a última fase, irá gerar o sumário em linguagem natural através da representação interna criada na etapa anterior (CARDOSO; PARDO; NUNES, 2011, p. 60). Nos casos da abordagem superficial, onde é gerado um resumo extrativo, esta etapa acaba por não existir pois a própria etapa de transformação já pode gerar o sumário final.

2.4 MÉTODO DE SUMARIZAÇÃO POR PALAVRA-CHAVE

Segundo Black e Johnson (1988 apud MARTINS et al., 2001, p. 9), as ideias vão sendo desenvolvidas no texto e, à medida que o texto vai evoluindo, os termos chaves vão aparecendo com maior frequência. Assim, o objetivo do método de sumarização por palavras-chave é determinar qual a distribuição estatística destes termos chaves do texto. A partir dessa frequência, devem ser extraídas as sentenças que as contenham, agrupando-as de forma a construir um sumário na mesma ordem em que apareceram originalmente.

Para extração das palavras-chave em um texto, pode-se usar o algoritmo EPC-P. Na concepção de Pereira, Souza e Nunes (2002), o algoritmo é baseado na análise da frequência de palavras pertencentes a padrões pré-determinados. O algoritmo EPC-P trabalha com seis padrões de palavras, que são:

a) nome[1];
b) nome + preposição + nome;
c) nome + adjetivo;
d) nome + adjetivo + adjetivo;
e) nome + adjetivo + preposição + nome;
f) nome + preposição + nome + adjetivo.

Para definir os padrões, deve-se determinar as classes gramaticais das palavras. Portanto, o texto-fonte precisa ser pré-processado, sendo efetuada a etiquetagem das palavras. Uma vez etiquetado, o texto deve ser percorrido na sua totalidade para criar seis listas em

[1] “nome” pode ser um substantivo comum ou nome próprio.

ordem alfabética, uma para cada um dos seis padrões de palavras. Deve-se também armazenar o número de ocorrências de cada palavra no texto. Juntamente com a criação destas seis listas, são criadas outras seis listas, uma para cada um dos seis padrões de palavras, contendo apenas os radicais[2] das palavras. Tendo as listas de radicais, estas devem ser ordenadas em ordem decrescente em relação ao número de ocorrências das palavras no texto. As palavras das seis primeiras listas de padrões devem permanecer em ordem alfabética.

Finalizada a construção das doze listas, deve-se iniciar a construção da lista de palavras-chave. Inicialmente deve-se criar uma lista chamada de `lista1`, que deve ser preenchida da seguinte forma:

a) mantem-se ponteiros apontando para o início de cada uma das seis listas de radicais. Para cada um dos seis elementos apontados, deve-se encontrar a frequência relativa, que é o número de ocorrências referente ao radical da palavra analisada dividido pela quantidade total de ocorrências do padrão onde a palavra encontra-se;
b) insere-se na `lista1` o radical de maior frequência relativa que ocorrer pelo menos mais de uma vez no texto, juntamente com um marcador que indica o padrão ao qual o radical pertence;
c) incrementa-se o ponteiro referente à lista de radicais onde a palavra adicionada na `lista1` encontra-se;
d) repete-se o processo até que a `lista1` tenha cinquenta elementos ou até que todos os radicais nas seis listas de radicais tenham sido analisados.

Um exemplo de construção da `lista1` pode ser visto no Apêndice A. Após criada a `lista1`, inicia-se a construção da lista final de palavras-chave (`lista2`). A `lista2` é preenchida da seguinte forma:

a) obtém-se o primeiro radical da lista que armazena os radicais do padrão “nome”;
b) toma-se a primeira ocorrência deste radical na `lista1`, se ele ocorrer na mesma;
c) insere-se na `lista2` o radical encontrado, se este já não existir nessa lista;
d) repete-se o processo para o próximo elemento da lista de radicais de “nomes” até que a `lista2` tenha trinta elementos ou até que todos os radicais na lista do padrão “nome” tenham sido analisados.

[2] Existem alguns algoritmos para radicalização de palavras da língua portuguesa que podem ser utilizados. Um deles é o radicalizador de Porter adaptado para a língua portuguesa da biblioteca PTStemmer (OLIVEIRA, 2010).

No final da etapa que cria a `lista2`, tem-se uma lista composta de radicais. Necessita-se retornar para cada radical da `lista2` a melhor palavra da lista de padrão de onde saiu o radical. Para isto precisa-se apenas percorrer a lista original de palavras do padrão pertencente ao radical, retornando a melhor ocorrência (PEREIRA; SOUZA; NUNES, 2002). Um exemplo da construção da `lista2` pode ser visto no Apêndice B.

O algoritmo EPC-P serve para extrair as palavras-chave de um texto. Após obtê-las, precisa-se aplicar algum algoritmo para gerar o resumo extrativo. O método mais simples e conhecido para gerar extratos através de palavra-chave é o que, “dado um texto e seu respectivo conjunto de palavras-chave, qualquer sentença pertencente ao texto que contenha pelo menos uma das palavras-chave fará parte do sumário gerado” (MARGARIDO; PARDO; ALUÍSIO, 2008, p. 1). Porém existe adaptações deste método que geram resultados mais expressivos. Pode-se, por exemplo, classificar as sentenças de acordo com o número de ocorrências das palavras-chave no texto.

2.5 TRABALHOS CORRELATOS

Existem trabalhos acadêmicos que exploram a sumarização automática de textos, alguns dos quais produziram efetivamente sistemas para criação de resumos. Dentre eles, destacam-se: GIST SUMMarizer (PARDO, 2002) e RHeSumaRST (SENO, 2005).

2.5.1 GIST SUMMarizer

GIST SUMMarizer foi um dos primeiros sumarizadores de textos voltados para a língua portuguesa. Tenta simular a sumarização feita por uma pessoa, buscando a ideia principal do texto-fonte para então complementá-la com informações adicionais relevantes (BALAGE FILHO; PARDO; NUNES, 2007, p. 5).

Na sua primeira versão o sistema sumarizador contemplava os seguintes itens:

a) realização de sumário de apenas um texto fonte como entrada;
b) realização de sumários em forma de extrato (seleção de sentenças inteiras do texto);

c) produção de sumários genéricos não destinados a um público específico.

Já na sua segunda versão, algumas funcionalidades foram adicionadas ao sistema:

a) realização de sumário com mais de um texto fonte como entrada;
b) elaboração de sumário focado em interesse da audiência de acordo com um tópico especificado pelo usuário;
c) realização da sumarização no interior das sentenças.

Basicamente o processo de sumarização deste sistema faz uso de algumas das diversas técnicas da abordagem superficial (BALAGE FILHO; PARDO; NUNES, 2007, p. 5).

2.5.2 RHeSumaRST

O objetivo deste trabalho resume-se na poda das estruturas *Rhetorical Structure Theory* (RST) de textos e não de textos escritos em linguagem natural.

> A RST fundamenta-se no princípio de que um texto tem uma estrutura retórica subjacente e que, através dessa estrutura, é possível recuperar o objetivo comunicativo que o escritor do texto pretendeu atingir ao escrevê-lo. Essa estrutura é composta por unidades elementares do discurso (Elementary Discourse Unit ou EDUs, no inglês), inter-relacionadas por meio de relações retóricas. (SENO, 2005, p. 8).

Esta poda RST é realizada através de algumas heurísticas como:

a) identificar informações de menor relevância para exclusão da estrutura RST;
b) verificar o relacionamento de termos anafóricos e seus antecedentes visando garantir a preservação dos mesmos quando forem inclusos na estrutura do sumário.

O sistema RHeSumaRST é composto por três módulos de processamento que são: delimitação de veias, classificação de saliência e poda. O primeiro módulo é alimentado com a estrutura RST do texto fonte gerada pelo *Annotation Tool*. Nela é aplicado o algoritmo de delimitação de veia. No segundo módulo as EDUs da estrutura são classificadas de acordo com a saliência. No último módulo é executada a poda da estrutura RST e a geração do sumário no formato de texto (SENO, 2005, p. 50).

3 DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo apresenta as etapas do desenvolvimento do protótipo e sua utilização, sendo que nas seções seguintes tem-se:

a) os Requisitos Funcionais (RF) e os Requisitos Não-Funcionais (RNF);
b) a especificação do protótipo, incluindo os diagramas de casos de uso e de classes;
c) a implementação, onde são detalhadas as técnicas e ferramentas utilizadas, a extração de etiquetas, a implementação do algoritmo EPC-P, a geração do sumário propriamente dito e a operacionalidade do protótipo;
d) as dificuldades encontradas, os resultados obtidos e uma discussão a respeito dos mesmos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo da ferramenta proposta deverá:

a) disponibilizar uma interface para entrada de texto em português (RF);
b) permitir escolher o nível de compactação do sumário a ser gerado (RF);
c) mostrar os *tokens* (palavras) e as etiquetas extraídas no processamento léxico e morfológico do texto (RF);
d) mostrar as listas utilizadas no algoritmo EPC-P, assim como as palavras-chave extraídas do texto-fonte (RF);
e) mostrar as palavras-chave e a classificação (*rank*) de cada sentença (RF);
f) disponibilizar o extrato gerado na forma de texto escrito em língua portuguesa (RF);
g) alertar o usuário caso sejam detectados erros durante o processo a sumarização (RF);
h) ser desenvolvido com a linguagem de programação C# no ambiente *Visual Studio* (RNF);
i) ser transparente durante o processo de sumarização, exigindo a intervenção do usuário apenas em caso de erro (RNF).

3.2 ESPECIFICAÇÃO

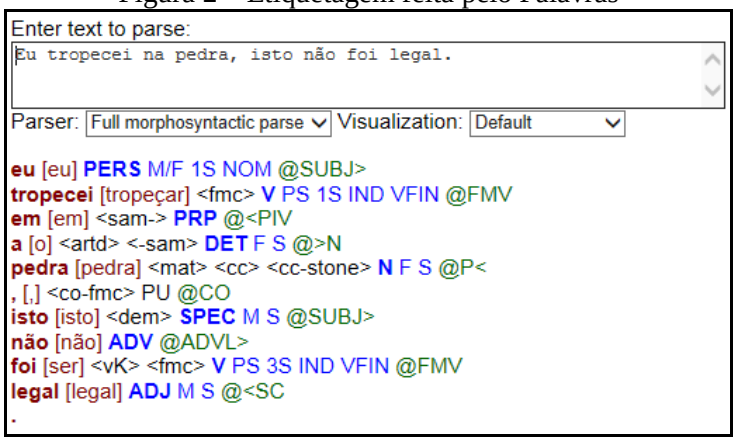
Nesta seção é apresentada a especificação do protótipo, incluindo as etapas do processo de sumarização automática, bem como descritos os diagramas de casos de uso e de classes da *Unified Modeling Language* (UML). O diagrama de classes foi criado pela ferramenta Microsoft Visual Studio 2010. O diagrama de casos de uso foi especificado utilizando a ferramenta *Enterprise Architect* (EA).

3.2.1 Etapas da sumarização

Conforme descrito na seção 2.3, o processo de sumarização automática compõe-se de três etapas: análise, transformação e síntese.

Neste protótipo a etapa de análise efetua apenas as análises léxica e morfológica do texto-fonte gerando sua representação interna, isto é, os *tokens* e suas etiquetas. Deve-se identificar substantivos simples, nomes próprios, adjetivos, preposições, entre outras classes gramaticais das palavras. Assim, na etapa de análise foi utilizado o analisador mofológico Palavras (BICK et al., 2000), componente do projeto *Visual Interactive Syntax Learning* (VISL), disponível na web. O analisador morfológico tem como entrada o texto a ser analisado e como saída os *tokens* reconhecidos e as respectivas etiquetas. A Figura 2 mostra o resultado da análise da frase “Eu tropecei na pedra, isto não foi legal.”, feito pelo Palavras.

Figura 2 – Etiquetagem feita pelo Palavras



Já no Quadro 2 tem-se as etiquetas utilizadas pelo protótipo, a partir da etiquetagem resultante do analisador morfológico Palavras.

Quadro 2 – Etiquetagem usada no protótipo

etiqueta	descrição	palavra
PERS	pronome pessoal	eu
V	verbo	tropecei
PRP	preposição	em
DET	artigo ou quantificador atributivo	a
N	substantivo	pedra
SPEC	pronome indefinido ou quantificador nominal	isto
ADV	advérbio	não
V	verbo	foi
ADJ	adjetivo	legal

Observa-se no Quadro 2 que o analisador morfológico Palavras gera diversas etiquetas. O protótipo especificado extrai todas as etiquetas retornadas, além das necessárias para efetuar a sumarização. Como todas as etiquetas são extraídas, torna-se possível substituir futuramente, se necessário, o algoritmo utilizado para efetuar a sumarização, caso dependa de outras classes gramaticais que não são utilizadas hoje pelo algoritmo atual. No Quadro 2 estão as etiquetas necessárias para o algoritmo de sumarização, exceto pela etiqueta PROP, não visualizada no quadro, que correspondente a nome próprios.

A próxima etapa do processo de sumarização automática é a transformação do *tokens* e suas etiquetas na representação interna do sumário. É utilizada uma abordagem superficial através do método de sumarização por palavra-chave, executando-se o algoritmo EPC-P. Uma vez extraídas as palavras-chave, como descrito na seção 2.4, usa-se um algoritmo para gerar o resumo extrativo. O algoritmo em questão classifica as sentenças do texto-fonte de acordo com o número de ocorrências (peso) das palavras-chave. Utiliza-se este número pois o mesmo já foi identificado na etapa de extração das palavras-chave através do algoritmo EPC-P.

Inicialmente adiciona-se em uma lista cada sentença[3] do texto. Em seguida, a lista deve ser percorrida em sua totalidade e, para cada sentença, deve-se verificar quais as palavras-chave existentes e somar o peso de cada palavra-chave para determinar a pontuação da sentença. Tendo a pontuação de cada sentença, deve-se selecionar todas as sentenças que obtiveram a pontuação maior que 0. Essas farão parte do resumo extrativo.

O protótipo permite também considerar um percentual de compressão do extrato gerado. Este percentual define um ponto de corte através da maior pontuação existente. O ponto de corte é utilizado para seleção das sentenças. Para isto são obedecidas as seguintes regras:

[3] Entende-se por sentença tudo aquilo que termina em ponto, ponto de exclamação ou ponto de interrogação e que não esteja entre parênteses.

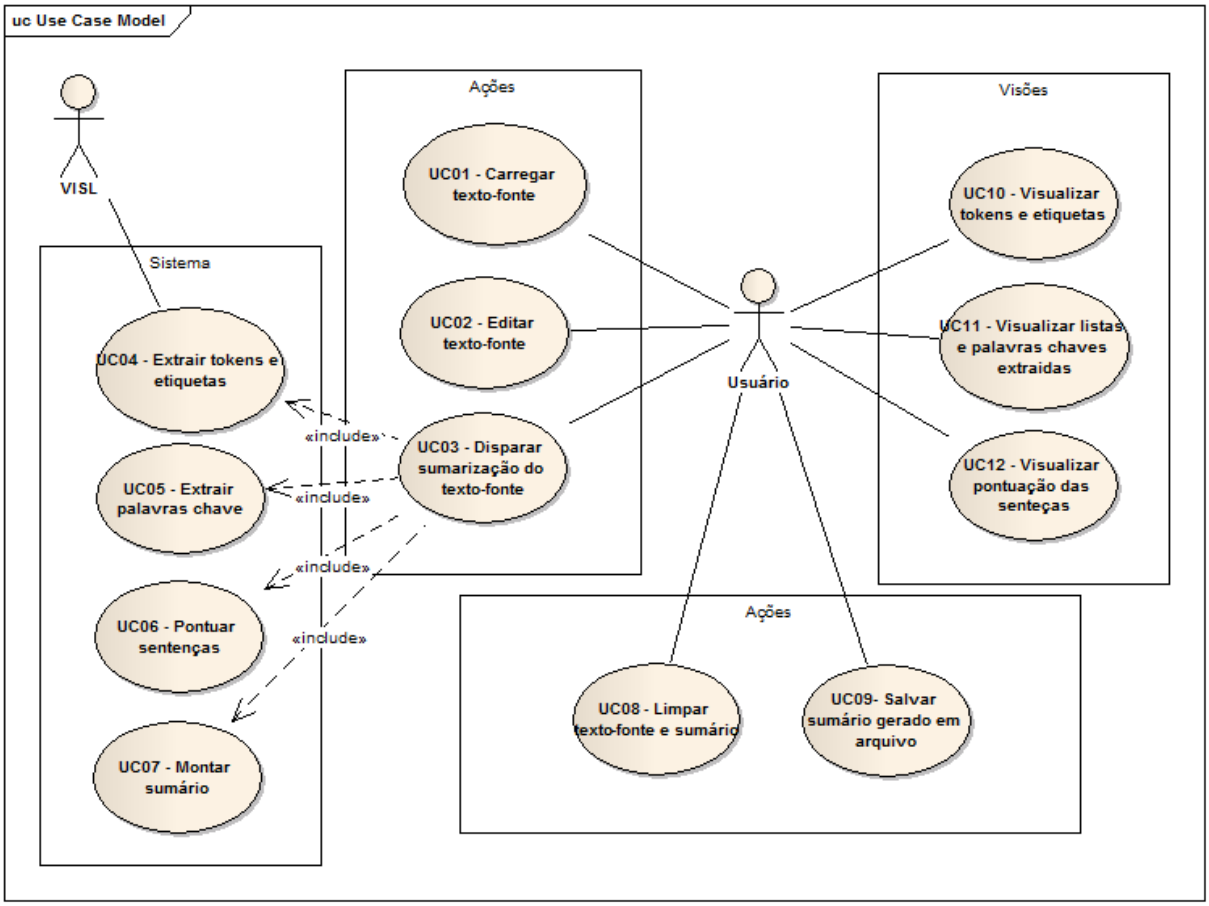
a) quando o percentual de compressão for 100%, o ponto de corte é a maior pontuação menos 1;
b) para percentual de compreensão menor que 100%, aplica-se o percentual à maior pontuação menos 1, obtendo-se o ponto de corte.

Como o protótipo já gera um resumo extrativo do texto-fonte, não foi explicitamente especificada a fase de síntese.

3.2.2 Diagrama de casos de uso

Os casos de uso do protótipo estão divididos em ações e visões, como pode ser observado na Figura 3. Entre as ações estão os casos de uso onde o usuário pode fornecer as informações e executar o processo de sumarização. Entre as visões estão os casos de uso onde é possível visualizar o que foi gerado pelo processo de sumarização.

Figura 3 – Casos de uso



Os casos de uso `Carregar texto-fonte` ou `Editar texto-fonte`, detalhados nos Quadro 3 e Quadro 4, respectivamente, representam a ação inicial do usuário no protótipo. Para

que seja possível efetuar o processo de sumarização, é necessário primeiramente ter um texto-fonte.

Quadro 3 – Caso de uso: `Carregar texto-fonte`

UC01 – Carregar texto-fonte	
Descrição	Permite selecionar um arquivo com texto-fonte.
Atores	Usuário
Cenário principal	1. O usuário pressiona o botão para carregar arquivo. 2. O usuário informa o local e o nome do arquivo a ser carregado. 3. O protótipo carrega o texto-fonte. 4. O protótipo disponibiliza o texto-fonte para o usuário.
Pré-condições	O arquivo do texto-fonte deve estar no formato `txt`.
Pós-condições	Texto-fonte carregado no protótipo.

Quadro 4 – Caso de uso: `Editar texto-fonte`

UC02 – Editar texto-fonte	
Descrição	Permite editar o texto-fonte carregado no protótipo.
Atores	Usuário
Cenário principal	1. O usuário clica na área do texto-fonte. 2. O usuário edita o texto conforme a necessidade.
Pré-condições	O texto-fonte deve estar carregado no protótipo.
Pós-condições	Texto-fonte alterado.

Após carregar ou editar um texto-fonte, é possível disparar o processo de sumarização e obter resultados através do caso de uso `Disparar sumarização do texto-fonte` (Quadro 5).

Quadro 5 – Caso de uso: `Disparar sumarização do texto-fonte`

UC03 – Disparar sumarização do texto-fonte	
Descrição	Permite realizar o processo de sumarização.
Atores	Usuário
Cenário principal	1. O usuário informa o percentual de compressão do sumário a ser gerado. 2. O usuário pressiona o botão para executar a sumarização. 3. O protótipo extrai os *tokens* e respectivas etiquetas do texto-fonte (`UC04 – Extrair tokens e etiquetas`). 4. O protótipo extrai as palavras-chave (`UC05 – Extrair palavras-chave`). 5. O protótipo pontua as sentenças de acordo com as palavras-chave (`UC06 – Pontuar sentenças`). 6. O protótipo monta o sumário de acordo com a pontuação das sentenças (`UC07 – Montar sumário`).
Exceções	No passo 3: erro de comunicação com o analisador morfológico Palavras, na página do projeto VISL. No passo 3: erro na extração de etiquetas de algum *token*. No passo 4: erro na identificação das listas padrões para o algoritmo EPC-P. No passo 4: erro na execução do algoritmo EPC-P. No passo 5: erro na extração das sentenças. No passo 5: erro no cálculo da pontuação. No passo 6: erro na seleção das sentenças. Caso ocorra um erro, o protótipo termina a operação e apresenta uma mensagem de erro.
Pré-condições	O texto-fonte deve estar carregado no protótipo.
Pós-condições	Sumário apresentado ao usuário.

Ao disparar a sumarização do texto-fonte, o protótipo executa os casos de uso necessários para obter o sumário, quais sejam: `Extrair tokens e etiquetas` (Quadro 6), `Extrair palavras-chave` (Quadro 7), `Pontuar sentenças` (Quadro 8) e `Montar sumário` (Quadro 9).

Quadro 6 – Caso de uso: `Extrair tokens e etiquetas`

UC04 – Extrair tokens e etiquetas	
Descrição	Extrai os *tokens* e as etiquetas correspondentes do texto-fonte.
Atores	Usuário e VISL
Cenário principal	1. Ao disparar a sumarização do texto-fonte, o protótipo envia o texto-fonte através de uma requisição `get` para o analisador morfológico Palavras, na página do projeto VISL. 2. O protótipo recebe um HTML com os *tokens* e as etiquetas. 3. O protótipo extrai os *tokens* e as etiquetas.
Exceções	No passo 1: erro de comunicação com o analisador morfológico Palavras, na página do projeto VISL. No passo 3: erro na extração de etiquetas de algum *token*. Caso ocorra um erro, o protótipo termina a operação e apresenta uma mensagem de erro.
Pré-condições	O texto-fonte deve estar carregado no protótipo. O computador deve estar conectado a internet.
Pós-condições	Representação interna do texto-fonte (*tokens* e respectivas etiquetas) disponível.

Quadro 7 – Caso de uso: `Extrair palavras-chave`

UC05 – Extrair palavras-chave	
Descrição	Extrai a lista de palavras-chave mais relevantes no texto-fonte.
Atores	Usuário
Cenário principal	1. O protótipo aplica o algoritmo EPC-P para a extração das palavras-chave.
Exceções	No passo 1: erro na identificação das listas padrões para o algoritmo EPC-P ou erro na execução do algoritmo EPC-P. Caso ocorra um erro, o protótipo termina a operação e apresenta uma mensagem de erro.
Pré-condições	A representação interna do texto-fonte deve estar disponível.
Pós-condições	Lista de palavras-chave extraída do texto-fonte.

Quadro 8 – Caso de uso: `Pontuar sentenças`

UC06 – Pontuar sentenças	
Descrição	Pontua as sentenças de acordo com as palavras-chave presentes na mesma.
Atores	Usuário
Cenário principal	1. O protótipo extrai cada sentença do texto-fonte. 2. O protótipo determina a pontuação da sentença pelo número de ocorrências de cada palavra-chave presente na sentença.
Exceções	No passo 1: erro na extração das sentenças. No passo 2: erro no cálculo da pontuação. Caso ocorra um erro, o protótipo termina a operação e apresenta uma mensagem de erro.
Pré-condições	A lista de palavras-chave deve ter sido extraída do texto-fonte.
Pós-condições	Lista de sentenças com suas pontuações disponível.

Quadro 9 – Caso de uso: `Montar sumário`

UC07 – Montar sumário	
Descrição	Monta o sumário através da seleção das sentenças pontuadas.
Atores	Usuário
Cenário principal	1. O protótipo seleciona as sentenças mais relevantes de acordo com a pontuação de corte, calculada através do percentual de compressão informado pelo usuário (no caso de uso `UC03 – Disparar sumarização do texto-fonte`).
Exceções	No passo 1: erro na seleção das sentenças. Caso ocorra um erro, o protótipo termina a operação e apresenta uma mensagem de erro.
Pré-condições	A lista de sentenças com suas pontuações deve estar disponível.
Pós-condições	Texto sumarizado disponível.

O usuário pode salvar o sumário gerado em arquivo texto (Quadro 10). Pode também visualizar: os *tokens* e as etiquetas extraídas (Quadro 11), as palavras-chave obtidas (Quadro 12) e as pontuações das sentenças (Quadro 13). Estas visões existem para que usuário tenha uma percepção do que foi feito no processo de sumarização, podendo ver quais sentenças foram selecionadas e quais não foram.

Quadro 10 – Caso de uso: `Salvar sumário gerado em arquivo`

UC09 – Salvar sumário gerado em arquivo	
Descrição	Permite salvar o sumário gerado em um arquivo texto.
Atores	Usuário
Cenário principal	1. O usuário pressiona o botão para salvar o sumário gerado. 2. O usuário informa o local e o nome do arquivo a ser salvo. 3. O protótipo salva o arquivo com o nome e no local indicados.
Exceções	No passo 1: erro ao salvar arquivo no local indicado. Caso ocorra um erro, o protótipo termina a operação e apresenta uma mensagem de erro.
Pré-condições	O sumário deve ter sido gerado.
Pós-condições	Texto sumarizado salvo em arquivo texto.

Quadro 11 – Caso de uso: `Visualizar tokens e etiquetas`

UC10 – Visualizar tokens e etiquetas	
Descrição	Permite visualizar os *tokens* e respectivas etiquetas extraídos pelo analisador morfológico Palavras.
Atores	Usuário
Cenário principal	1. O usuário pressiona o botão para visualizar *tokens* e etiquetas. 2. O protótipo exibe uma tabela com os *tokens* extraídos do texto-fonte. 3. O usuário seleciona um *token*. 4. O protótipo exibe as etiquetas do *token* selecionado.
Pré-condições	O sumário deve ter sido gerado.
Pós-condições	*Tokens* e etiquetas exibidos.

Quadro 12 – Caso de uso: `Visualizar listas e palavras-chave extraídas`

UC11 – Visualizar listas e palavras-chave extraídas	
Descrição	Permite visualizar as listas utilizadas e as palavras-chave encontradas pelo algoritmo EPC-P.
Atores	Usuário
Cenário principal	1. O usuário pressiona o botão para visualizar palavras-chave. 2. O protótipo exibe as listas utilizadas no formato de texto. 3. O protótipo exibe uma tabela com as palavras-chave encontradas.
Exceções	Não há exceções previstas.
Pré-condições	O sumário deve ter sido gerado.
Pós-condições	Listas utilizadas e palavras-chave encontradas exibidas.

Quadro 13 – Caso de uso: `Visualizar pontuação das sentenças`

UC12 – Visualizar pontuação das sentenças	
Descrição	Permite visualizar as sentenças e suas pontuações, assim como o ponto de corte calculado pelo percentual de compressão.
Atores	Usuário
Cenário principal	1. O usuário pressiona o botão para visualizar a pontuação das sentenças. 2. O protótipo exibe uma tabela com as sentenças, a sequência de cada uma no texto-fonte e a respectiva pontuação. 3. O usuário seleciona uma sentença. 4. O protótipo exibe quais as palavras-chave que a sentença selecionada contém.
Exceções	Não há exceções previstas.
Pré-condições	O sumário deve ter sido gerado.
Pós-condições	Sentenças e pontuações exibidas.

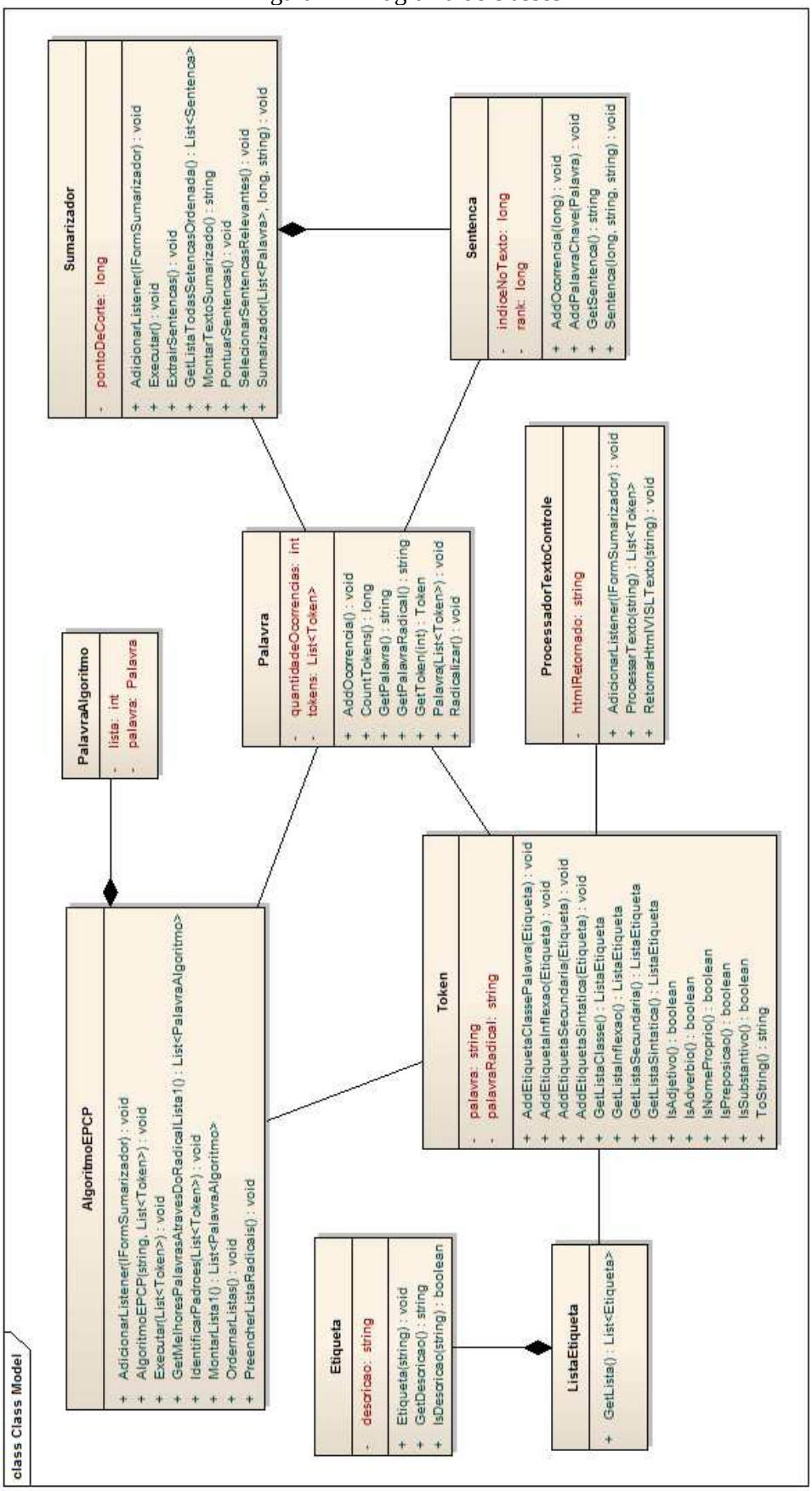
3.2.3 Diagrama de classes

A Figura 4 apresenta o diagrama de classes do protótipo. As classes `ProcessadorTextoControle`, `AlgoritmoEPCP` e `Sumarizador` são as classes de controle responsáveis por realizar a sumarização do texto-fonte. As demais classes são as de modelo, utilizadas pelas classes de controle.

A classe `Token` representa os *tokens* extraídos e retornados pelo analisador morfológico Palavras, sendo que um *token* é composto por uma palavra e suas etiquetas. É através desta classe que é possível saber se determinada palavra é, por exemplo, um substantivo. A classe `Palavra` representa uma palavra do texto processado. Nela tem-se a quantidade de ocorrências da palavra no texto, calculada pelo o `AlgoritmoEPCP`, e o radical da palavra, determinado pelo método `Radicalizar`. A classe `Sentenca` representa uma sentença do texto-fonte. Possui como atributos a pontuação da sentença, calculada pela própria classe, e um índice, para indicar a ordem da sentença no texto-fonte. A classe `ListaEtiqueta` tem como única finalidade armazenar uma lista de `Etiquetas`.

A classe `ProcessadorTextoControle` efetua as requisições necessárias ao analisador morfológico Palavras e processa o HTML retornado, obtendo assim os *tokens* e suas etiquetas. A lista de etiquetas obtidas pela classe `ProcessadorTextoControle` é utilizada ao instanciar a classe `AlgoritmoEPCP`. Na classe `AlgoritmoEPCP` o método `Executar` aplica os passos do algoritmo EPC-P, que são: identificar lista de padrões a partir do texto etiquetado, criar as listas de radicais para cada padrão, ordenar as listas, calcular a ocorrência de cada padrão e montar as listas `lista1` e `lista2` através das regras do algoritmo. Após a execução do método `Executar`, obtém-se a lista de palavras-chave. Com a lista de palavras-chave é possível criar o sumário através da classe `Sumarizador`. Para instanciar a classe `Sumarizador`, é necessário informar o texto-fonte, o percentual de compressão e a lista de palavras-chave. Através do método `Executar`, a classe calcula a pontuação e faz a seleção das sentenças. São selecionadas apenas as sentenças com pontuação igual ou superior ao ponto de corte calculado a partir do percentual de compressão.

Figura 4 – Diagrama de classes



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas na implementação do protótipo, como é feita a extração das etiquetas geradas pelo analisador morfológico Palavras, a implementação do algoritmo EPC-P, a geração do sumário e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

As técnicas e ferramentas utilizadas para o desenvolvimento do protótipo proposto foram as seguintes:

a) a *Integrated Development Environment* (IDE) Microsoft Visual Studio 2010, utilizada para a codificação do protótipo;
b) o *framework* Windows Forms presente no Microsoft Visual Studio 2010, para criação das telas do protótipo no padrão Windows;
c) a linguagem C#, para codificar o protótipo;
d) a biblioteca PTStemmer (OLIVEIRA, 2010), para obter os radicais das palavras contidas em um texto-fonte, para a implementação do algoritmo EPC-P.

A biblioteca PTStemmer implementa três algoritmos para radicalização de palavras da língua portuguesa, sendo estes: o Orengo, o Porter e o Savoy. A biblioteca é de código livre e atualmente possui suporte para Java, Python e C#. O Quadro 14 mostra como utilizar os algoritmos Orengo e Porter na linguagem C#.

Quadro 14 – PTStemmer: `Orengo e Porter`

```
1  Stemmer stemmer = new OrengoStemmer();
2  s.ignore(new string[] { "de", "na" });
3  Console.WriteLine(stemmer.getWordStem("extremamente"));
4
5  Stemmer stemmer = new PorterStemmer();
6  s.ignore(new string[] { "de", "na" });
7  Console.WriteLine(stemmer.getWordStem("extremamente"));
```

Nas linhas 1 e 5 são instanciados objetos do tipo `Stemmer` das classes `OrengoStemmer` e `PorterStemmer`, respectivamente, ambas com a implementação do respectivo algoritmo de radicalização. Nas linhas 2 e 6 é informado que não deve ser efetuada a radicalização das palavras “de” e “na”. Por fim, nas linhas 3 e 7, a palavra da qual se deseja extrair o radical é

passada como parâmetro para o método `getWordStem`, sendo o resultado apresentado no console.

3.3.2 Extração das etiquetas

Para fazer a etiquetagem de um texto-fonte utiliza-se o analisador morfológico Palavras, disponível na página do projeto VISL, exigindo, desta forma, uma conexão com a internet para o correto funcionamento do protótipo. Envia-se uma requisição `get` com o texto-fonte e obtém-se os *tokens* e as etiquetas dos mesmos através do HTML retornado. Este HTML obedece um padrão de cores e símbolos, sendo possível extrair os *tokens* e as etiquetas através de processamento de texto. No Quadro 15 tem-se o HTML retornado pelo VISL correspondente à frase “a família de Tatcher”.

Quadro 15 – HTML retornado pelo VISL

```
<dl>
  <dt>
    <b><font color="maroon">a</font></b>
    <font color="maroon">[o]</font>  &lt;artd&gt;
    <font color="blue"><b>DET</b> F S </font>
    <font color="darkgreen">@&gt;N</font>
  <dt>
    <b><font color="maroon">família</font></b>
    <font color="maroon">[família]</font>  &lt;HH&gt;
    <font color="blue"><b>N</b> F S </font>
    <font color="darkgreen">@NPHR</font>
  <dt>
    <b><font color="maroon">de</font></b>
    <font color="maroon">[de]</font>
    <font color="blue"><b>PRP</b> </font>
    <font color="darkgreen">@N&lt;</font>
  <dt><b>
    <font color="maroon">Tatcher</font></b>
    <font color="maroon">[Tatcher]</font>  &lt;hum&gt;
    <font color="blue"><b>PROP</b> M/F S </font>
    <font color="darkgreen">@P&lt;</font>
  <dt>
    <b><font color="maroon">.</font></b>
</dl>
```

A classe responsável pelo processamento do HTML é a `ProcessadorTextoControle`. Para tanto, envia uma requisição `get` com o texto-fonte escrito em língua portuguesa e retorna uma lista de objetos da classe `Token`. Um `Token` está entre as *tags* `<dt>` ou entre uma *tag* `<dt>` até a *tag* de finalização `</dl>`. O texto que está entre as *tags* `<b><font color="maroon">` e `</font></b>` é a descrição do `Token` e na sequência tem-se as etiquetas do `Token`.

A classe `Token` é composta por quatro listas de etiquetas, que são: de classe de palavra, de inflexão, sintáticas e secundárias. As etiquetas de classe de palavra representam a classe gramatical das palavras, como por exemplo advérbio, adjetivo, artigo ou substantivo. Já as etiquetas de inflexão correspondem às variações de gênero e de número. As etiquetas sintáticas, como o nome sugere, correspondem à sintaxe, como por exemplo objeto direto ou indireto. E, por fim, as etiquetas secundárias representam todas as etiquetas restantes e são usadas para remover a ambiguidade de uma palavra. Um artigo, por exemplo, é classificado com a etiqueta de classe de palavra `DET`. Mas um artigo pode ser definido ou indefinido. Nesse caso, é necessária uma etiqueta secundária para indicar se o artigo é definido ou indefinido, removendo a ambiguidade. A classe `ProcessadorTextoControle` identifica em qual das quatro listas deve adicionar a etiqueta através da propriedade `color` da *tag* `<font>`. Por exemplo, `<font color="blue">` indica que a etiqueta é de classe de palavra.

No Quadro 16 tem-se o código que envia a requisição com o texto-fonte e armazena o HTML retornado.

Quadro 16 – Método: `RetornarHtmlVISLTexto`

```
 1  private void RetornarHtmlVISLTexto(string texto) {
 2    try {
 3      string url =
 4      "http://beta.visl.sdu.dk/visl/pt/parsing/automatic/parse.php?text=" +
 5      texto + "&parser=parse&visual=niceline";
 6
 7      HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
 8      HttpWebResponse response = (HttpWebResponse)request.GetResponse();
 9      Stream resStream = response.GetResponseStream();
10      StreamReader oReader = new StreamReader(resStream, Encoding.UTF8);
11
12      this._htmlRetornado = oReader.ReadToEnd();
13      oReader.Close();
14      resStream.Close();
15    }
16    catch (Exception excecao) {
17      throw new Exception
18      ("[Comunicar VISL] - Erro ao processar texto no VISL.\n " +
19      excecao.Message);
20    }
21  }
```

Existe uma limitação do analisador morfológico Palavras: o texto-fonte a ser etiquetado deve ter apenas 5430 caracteres. Para contornar esta limitação, envia-se o texto a ser etiquetado em várias partes, cada uma com no máximo 5430 caracteres, caso exceda o limite permitido, ou seja, o método `RetornarHtmlVISLTexto` é chamado várias vezes. Após receber o HTML de cada parte, o protótipo processa e acumula os resultados em memória.

3.3.3 Algoritmo EPC-P

A classe responsável por encapsular o algoritmo EPC-P é a `AlgoritmoEPCP`. A primeira etapa do algoritmo é identificar as palavras e adicionar na sua respectiva lista de padrão, conforme explicado na seção 2.4. Para isto, faz-se uso da lista de objetos da classe `Token`. Essa lista é percorrida na sua totalidade e são identificados os padrões através dos métodos `IsSubstantivo`, `IsNomeProprio`, `IsAdjetivo` e `IsPreposicao`, mostrados nos Quadros 17, 18, 19 e 20, respectivamente. Neste momento, também é determinado o número de ocorrências de cada palavra identificada.

Quadro 17 – Método: `IsSubstantivo`

```
1  public bool IsSubstantivo() {
2    return _etiquetasClassePalavra.GetLista().Exists(w => w.IsDescricao("N"));
3  }
```

Quadro 18 – Método: `IsNomeProprio`

```
1  public bool IsNomeProprio() {
2    return
3    _etiquetasClassePalavra.GetLista().Exists(w => w.IsDescricao("PROP"));
4  }
```

Quadro 19 – Método: `IsAdjetivo`

```
1  public bool IsAdjetivo() {
2    return _etiquetasClassePalavra.GetLista().Exists(w =w.IsDescricao("ADJ"));
3  }
```

Quadro 20 – Método: `IsPreposicao`

```
1  public bool IsPreposicao() {
2    return
3    _etiquetasClassePalavra.GetLista().Exists(w => w.IsDescricao("PRP"));
4  }
```

Após identificar as listas dos padrões, é necessário criar mais seis listas de radicais, uma para cada padrão, conforme descrito no algoritmo. No Quadro 21 pode-se verificar o método `Radicalizar` da classe `Palavra`. Todas as listas originais dos padrões são percorridas, e através do método `Radicalizar` são geradas novas seis listas com os radicais.

Quadro 21 – Método: `Radicalizar`

```
1  public void Radicalizar(){
2    Stemmer s = Stemmer.StemmerFactory(Stemmer.StemmerType.PORTER);
3    // Ignorar preposições
4    s.ignore(new string[] { "de", "da", "do", "na", "no",
5                            "em", "para", "por", "a", "até",
6                            "desde", "entre", "com", "contra",
7                            "sem", "sob", "sobre", "trás"});
8    foreach (Token t in _tokens) {
9      t.PalavraRadical = s.getWordStem(t.Palavra);
10   }
11 }
```

Estas listas de radicais possuem as mesmas instâncias dos objetos da classe `Palavra` pertencentes às listas de padrões, porém com os radicais gerados. Em seguida, as listas são

ordenadas, sendo que as de padrões por ordem alfabética e as de radicais por ordem decrescente em relação ao número de ocorrências das palavras. Ainda antes de montar a `lista1` do algoritmo, é necessário obter o total de ocorrências de cada padrão. Para isto, basta somar a ocorrência de cada palavra nas listas de padrões e armazenar o resultado. Por fim, pode-se iniciar o processo de criação da `lista1`, que é a primeira lista de palavras-chave do algoritmo. Para criar essa lista, precisa-se calcular a frequência relativa de cada radical a ser comparado. O cálculo da frequência relativa de um radical é mostrado no Quadro 22.

Quadro 22 – Cálculo da frequência relativa de um radical

```
1  float frequenciaRelativaRadical1 = 0.0f;
2  if (radical1 != null) {
3    frequenciaRelativaRadical1 =
4    (float)radical1.Ocorrencia / (float)ocorrenciasNome;
5  }
```

Após calcular a frequência relativa, inicia-se com zero um ponteiro para cada lista de radicais. Em seguida, obtém-se o radical de cada lista através do ponteiro criado. Verifica-se qual o radical com a maior frequência relativa que ocorre pelo menos uma vez ou mais no texto. Este radical será incluido na `lista1`. Apenas o ponteiro da lista de padrões pertencente ao radical selecionado é incrementado. Este processo repete-se até que a `lista1` possua cinquenta elementos ou que as listas de radicais tenham sido percorridas nas suas totalidades.

Concluída a criação da `lista1`, precisa-se criar a `lista2`, que é a lista de palavras-chave final. Para isto, percorre-se a lista de radicais do padrão “nome” e a `lista1`. Obtém-se o primeiro elemento da `lista1` que se encaixe com radical que está sendo percorrido na lista de radicais de nome. Adiciona-se na `lista2` o radical obtido, caso este ainda não exista na lista. Este procresso repete-se até que a `lista2` possua trinta elementos ou que a lista de radicais do padrão “nome” tenha sido percorrida na sua totalidade. O Quadro 23 mostra a etapa de criação da `lista2`.

Quadro 23 – Criação da lista final de palavras-chave do protótipo

```
 1  foreach (Palavra p2 in nome_Radical) {
 2    if (listaMelhoresPalavras.Count() == 30)
 3      break;
 4    if (lista1.Exists(w =>
 5       (w.palavra.GetPalavraRadical().IndexOf(p2.GetPalavraRadical())>-1))) {
 6      foreach (PalavraAlgoritmo p3 in lista1)  {
 7        if (p3.palavra.GetPalavraRadical().IndexOf(p2.GetPalavraRadical())>-1)
 8        {
 9          if (!listaMelhoresPalavras.Exists(w=>(w.palavra.GetPalavraRadical()
10               == p3.palavra.GetPalavraRadical()))) {
11            //lista2
12            listaMelhoresPalavras.Add(p3);
13            break;
14          }
15        }
16      }
17    }
18  }
```

3.3.4 Geração do sumário

A classe `Sumarizador` é responsável por efetuar o sumário através de uma lista de palavras-chave. São três os passos para geração do sumário: extrair as sentenças do texto-fonte, pontuar as sentenças de acordo com as palavras-chave e selecionar as sentenças relevantes através do nível de compressão e da pontuação.

Para extrair as sentenças utiliza-se expressão regular. No Quadro 24 pode-se ver o método que utiliza a expressão regular para extrair as sentenças.

Quadro 24 – Método: `ExtrairSentencas`

```
 1  private void ExtrairSentencas() {
 2    String expressaoRegular =
 3    @"([^\\.\\!\\?] (\( [^\\)]* \))?)+ (\\. | \\? | \\! | \\.\\.\\.)";
 4    long indiceSentencaNoTexto = 0;
 5    foreach (Match match in Regex.Matches(_textoOriginal.Trim(),
 6             expressaoRegular, RegexOptions.IgnorePatternWhitespace)) {
 7      String ponto =
 8      _textoOriginal.Substring(_textoOriginal.IndexOf(match.Value.Trim()) +
 9      match.Value.Trim().Length, 1);
10      Sentenca s =
11      new Sentenca(match.Value.Trim(), ponto, indiceSentencaNoTexto);
12      _sentencasTodas.Add(s);
13      indiceSentencaNoTexto++;
14    }
15  }
```

A expressão regular usa como delimitadores os símbolos “.”, “!” e “?”, exceto quando esses símbolos são encontrados dentro de parentêses. O resultado do método são as sentenças reconhecidas juntamente com a localização delas no texto-fonte.

A pontuação das sentenças é feita percorrendo a lista de sentenças. Para cada sentença da lista, verifica-se quantas palavras-chave estão presentes e, para cada palavra-chave

existente na sentença, soma-se o número de ocorrências da palavra-chave na pontuação da sentença. O número de ocorrências de cada palavra-chave é obtido no algoritmo EPC-P. O Quadro 25 traz o método responsável por definir a pontuação de cada sentença.

Quadro 25 – Método: `PontuarSentencas`

```
 1  private void PontuarSentencas() {
 2    foreach (Sentenca s in _sentencasTodas) {
 3      string[] tokensSentenca = s.GetSentenca().Split(' ');
 4      for (int i = 0; i < tokensSentenca.Count(); i++) {
 5        // A palavra da sentenca está na lista de palavras chaves
 6        if (_palavrasChaves.Exists(w => w.GetPalavra().ToUpper()
 7            .Equals(tokensSentenca[i].ToUpper()))) {
 8          // Retorna a instancia da palavra
 9          Palavra p = _palavrasChaves.Find(w => w.GetPalavra()
10                      .ToUpper().Equals(tokensSentenca[i].ToUpper()));
11          s.AddOcorrencia(p.Ocorrencia);
12          s.AddPalavraChave(p);
13        }
14      }
15    }
16  }
```

A seleção das sentenças é feita através da pontuação de cada uma. Para isto deve-se calcular o ponto de corte. Este ponto irá indicar as sentenças que farão ou não parte do sumário. Para calcular o ponto de corte, verifica-se qual a sentença com maior pontuação e subtrai-se 1 desse valor. O resultado corresponde a 100% de compressão. Caso o percentual de compressão seja de 50%, aplica-se este percentual em cima do resultado obtido chegando assim ao ponto de corte. Considerando, por exemplo, que a maior pontuação seja 50 e que o percentual de compressão seja 100%, tem-se um ponto de corte igual a 49. Neste mesmo exemplo, porém com uma taxa de compressão de 50%, o ponto de corte seria 24 (50% de 49). Com o ponto de corte calculado, são selecionadas apenas as sentenças com pontuação maior ou igual ao ponto de corte. O Quadro 26 mostra o método que preenche a lista das sentenças selecionadas.

Quadro 26 – Método: `SelecionarSentencasRelevantes`

```
 1  private void SelecionarSentencasRelevantes() {
 2    _sentencasTodas = _sentencasTodas.OrderByDescending(w => w.Rank).ToList();
 3    long cemPorCento = _sentencasTodas.First().Rank - 1;
 4    _pontoDeCorte = (_taxaCompressao * cemPorCento) / 100;
 5
 6    if (listener != null) {
 7      listener.NotificarMinimoMaximo(0, _sentencasTodas.Count);
 8    }
 9    foreach (Sentenca s in _sentencasTodas) {
10      if (listener != null) {
11        listener.NotificarProgresso("Sumarizando [Selecionando Sentenças]",1);
12      }
13      if (s.Rank >= _pontoDeCorte) {
14        _sentencasSelecionadas.Add(s);
15      }
16    }
17    _sentencasSelecionadas = _sentencasSelecionadas.OrderBy
18                                                (w => w.IndiceNoTexto).ToList();
19  }
```

3.3.5 Operacionalidade da ferramenta

A operacionalidade da ferramenta é bastante simples. A interface principal do protótipo pode ser vista na Figura 5.

O protótipo necessita de uma conexão com a internet para efetuar a sumarização de textos, pois o processo de etiquetagem é efetuado pelo Palavras (BICK et al., 2000). Para utilizar o protótipo, o primeiro passo é entrar com um texto-fonte. Tem-se duas opções: digitar um texto na area superior do protótipo ou abrir um arquivo pressionando o botão correspondente (botão 6 da Figura 5). Após selecionado o arquivo, o sistema irá mostrar o texto conforme a Figura 6. Basta então definir o percentual de compressão (campo `Compressão %`) e pressionar o botão para executar a sumarização (botão 1 da Figura 5). Todas as etapas de sumarização são efetuadas de forma totalmente transparente para o usuário, sendo que o progresso do processo pode ser acompanhado através da barra de `Progresso`.

Figura 5 – Interface do protótipo

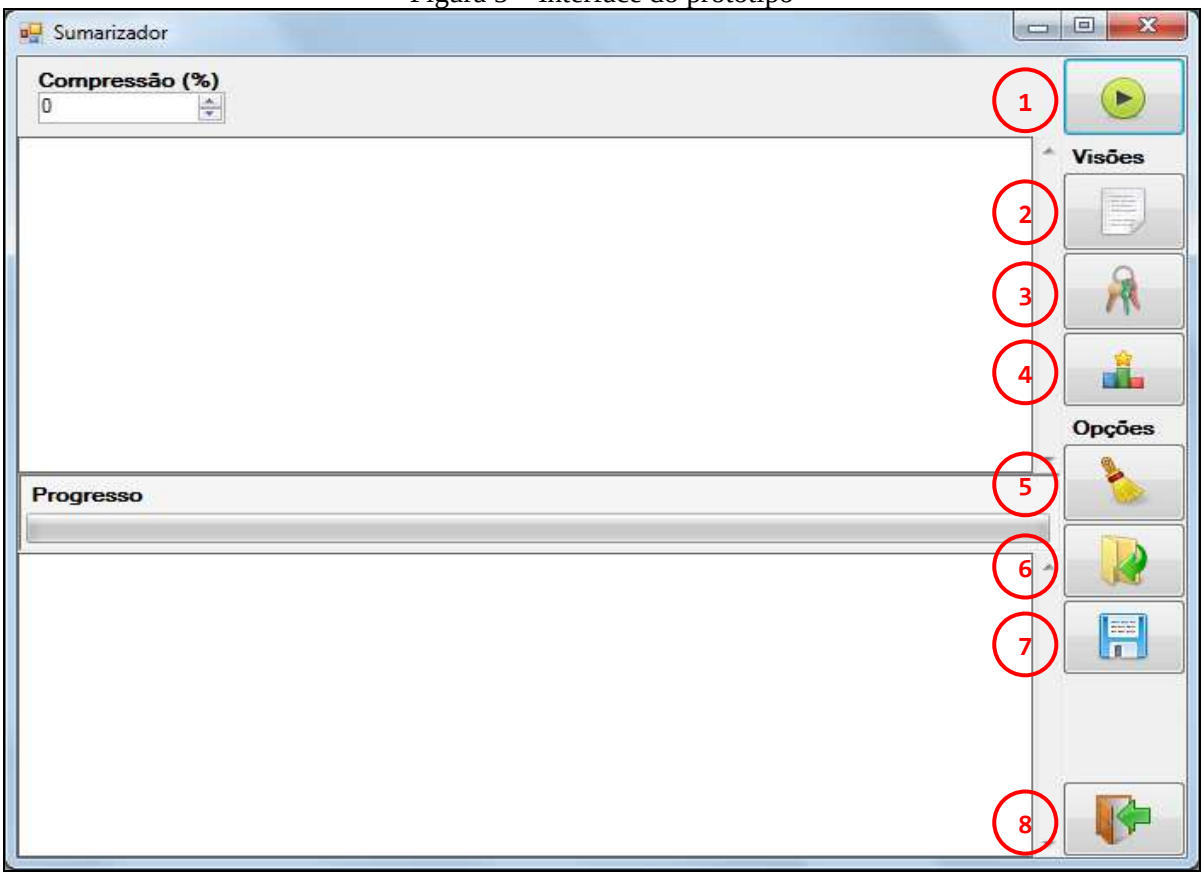
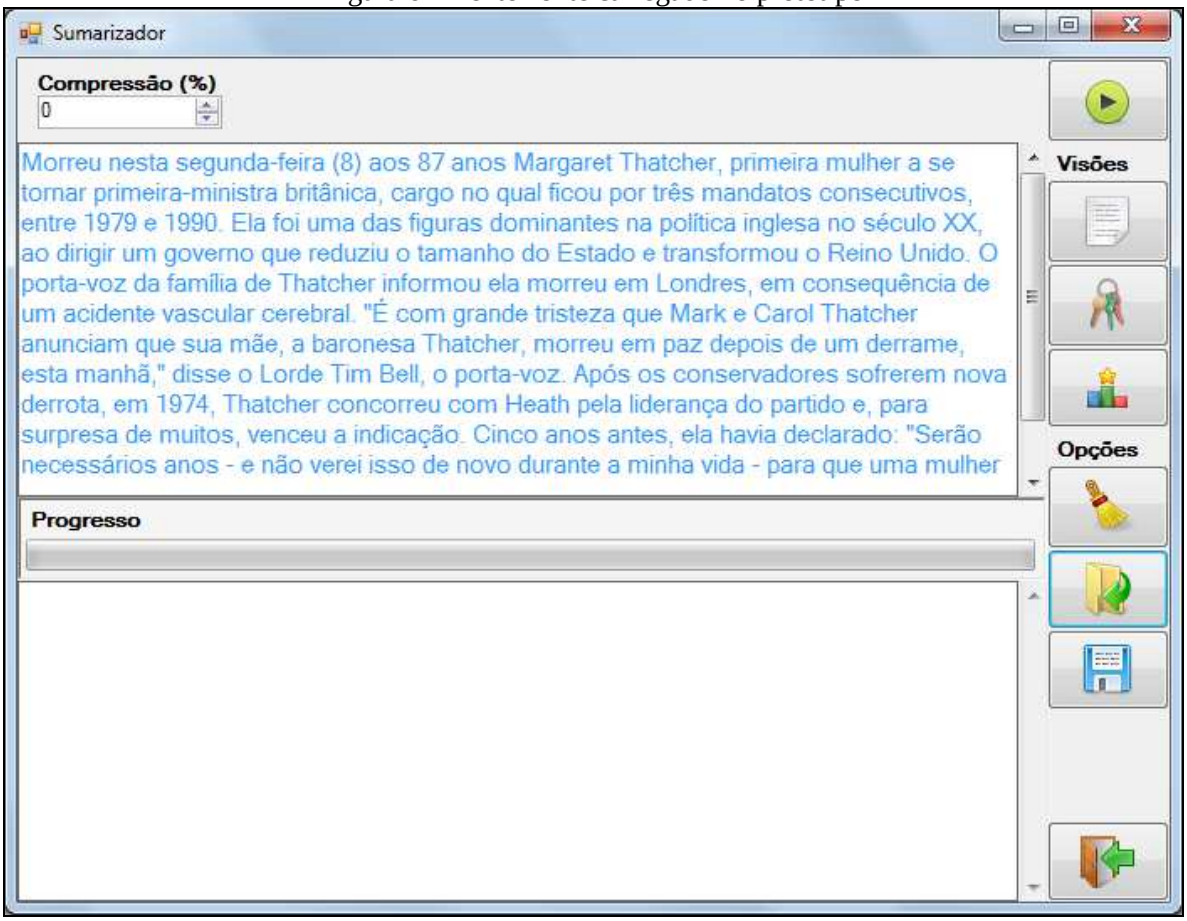
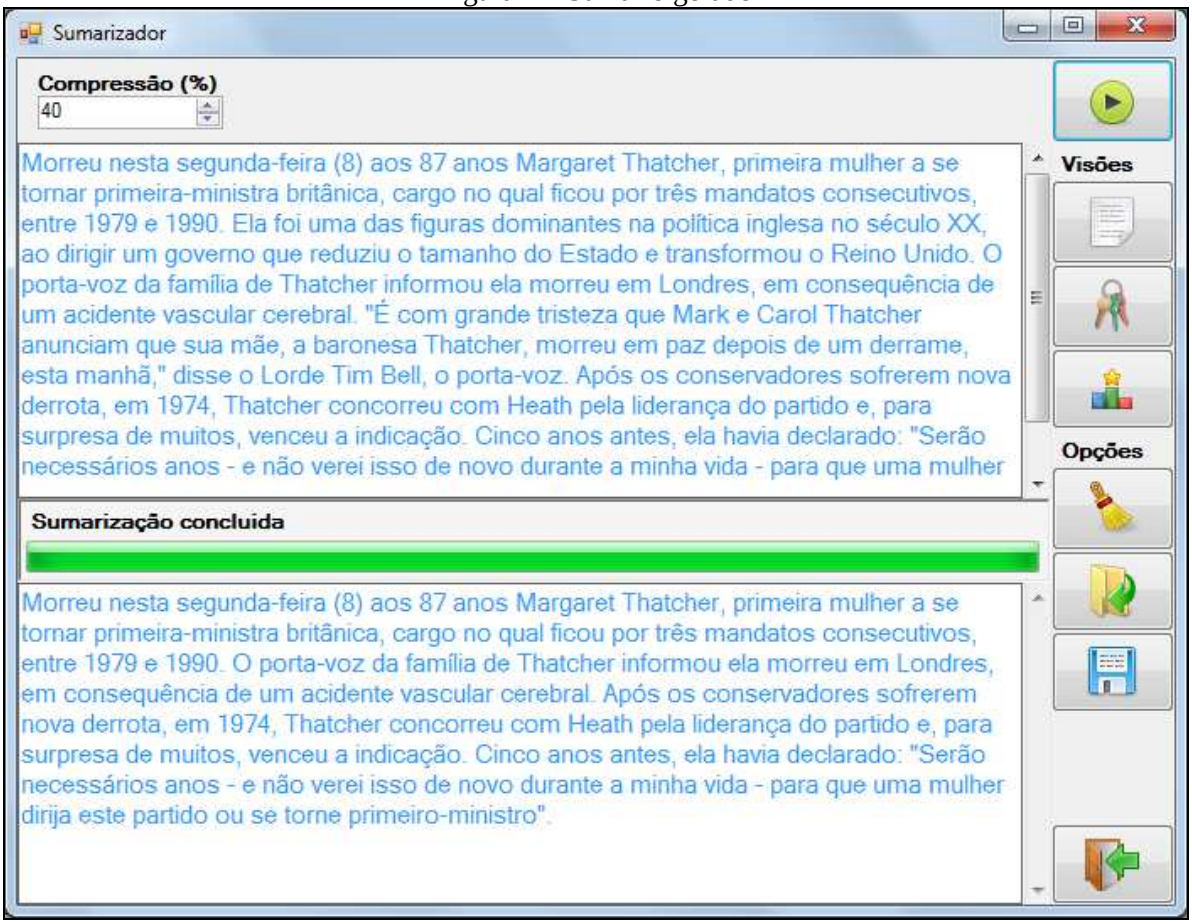


Figura 6 – Texto-fonte carregado no protótipo



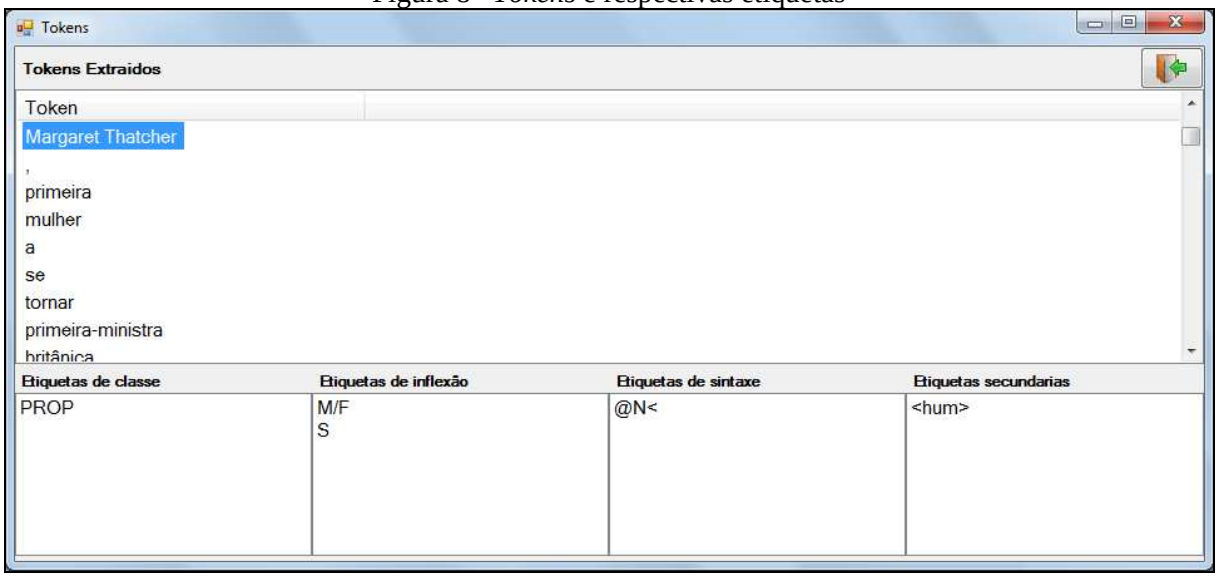
Na Figura 7 pode-se visualizar o sumário gerado (na parte inferior da tela) a partir do texto-fonte (na parte superior da tela).

Figura 7 – Sumário gerado



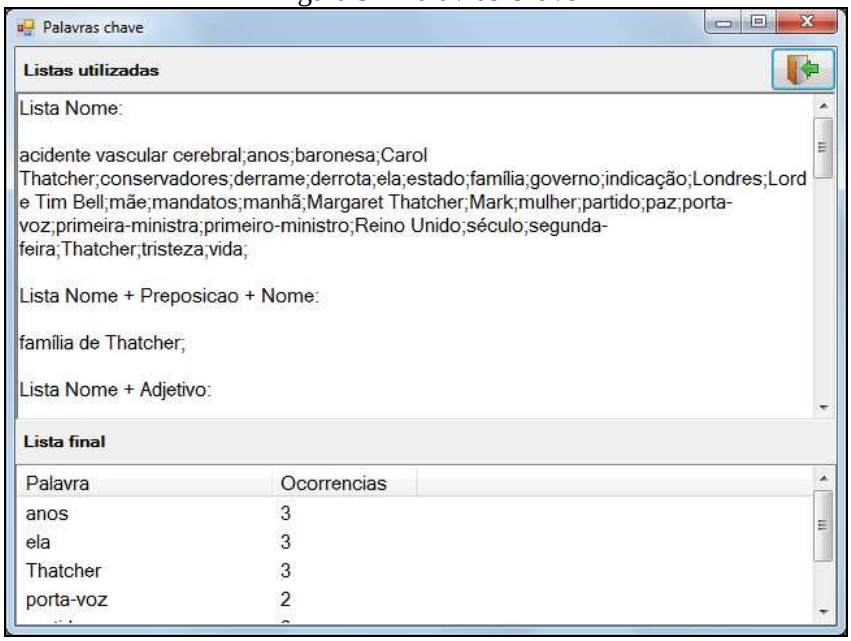
Com o sumário gerado, é possível visualizar alguns detalhes do processo realizado. Ao pressionar o botão 2 da Figura 5, pode-se ver todos os *tokens* extraídos do texto-fonte, assim como as respectivas etiquetas (Figura 8). Deve-se selecionar o *token* desejado. Ao selecionar o *token* "Margaret Thatcher", por exemplo, são exibidas as suas etiquetas em quatro listas distintas. A primeira lista (`Etiquetas de classe`) possui a etiqueta `PROP` que corresponde a nome próprio. A segunda lista (`Etiquetas de inflexão`) possui duas etiquetas, a primeira etiqueta indica o gênero, no exemplo o *token* selecionado é uma palavra masculina e feminina, enquanto a segunda etiqueta indica o número, no caso do *token* selecionado é uma palavra no singular. A etiqueta exibida na terceira lista (`Etiquetas de sintaxe`) não possui uma definição clara na documentação do analisador morfológico Palavras, portanto não é descrita. A quarta lista (`Etiquetas secundárias`) apresenta uma etiqueta indicando que o *token* trata-se do nome de um humano (`<hum>`).

Figura 8 –*Tokens* e respectivas etiquetas



É possível também, ao pressionar o botão 3 da Figura 5, mostrar todas as listas utilizadas no algoritmo EPC-P e a lista final de palavras-chave (Figura 9).

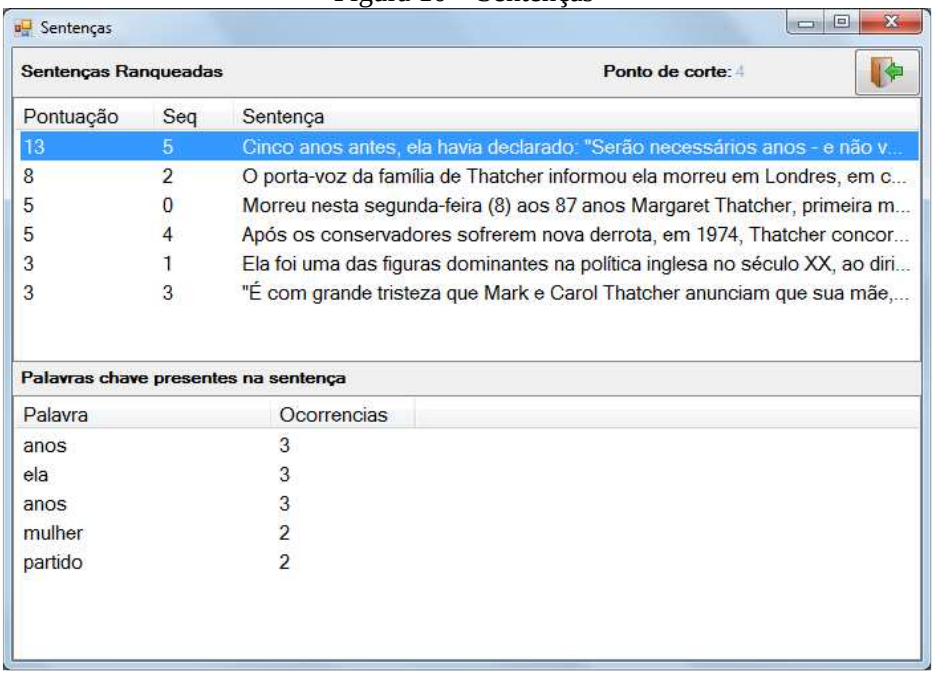
Figura 9 – Palavras-chave



Existe também a opção, através botão 4 da Figura 5, de observar todas as sentenças extraídas do texto-fonte, assim como a pontuação e as palavras-chave presentes em cada uma. Na Figura 10 é possível verificar o ponto de corte (`Ponte de corte: 4`) e a pontuação de cada sentença, ambas as informações utilizadas para a seleção das sentenças que compõem o sumário final. Tendo como exemplo a sentença selecionada na Figura 10, pode-se verificar que a soma dos valores da coluna `Ocorrências` das palavras-chave presentes na sentença resulta em 13, pontuação da sentença. Já o ponto de corte é obtido efetuando o seguinte cálculo: `((MP – 1) / 100) * TC`, onde `MP` corresponde à maior pontuação das sentenças e

`TC` ao percentual de compressão. Aplicando esta fórmula ao exemplo, que possui uma taxa de compressão de 40% (conforme Figura 7), chega-se ao seguinde resultado: `((13 - 1) / 100) * 40 = 4,8`. Esse valor é truncado, sendo o ponto de corte igual a 4. Com o ponto de corte definido, são selecionadas todas as sentenças com pontuação igual ou superior a 4. Assim sendo, o sumário apresentado na Figura 7 é composto pelas sentenças (`Seq`) 0, 2, 4 e 5, ou seja, a primeira, a terceira, a quinta e a sexta sentenças do texto-fonte original.

Figura 10 – Sentenças



Além das funcionalidades descritas, existem também botões com ações para: limpar o sumário gerado e deixar o protótipo pronto para uma nova sumarização (botão 5 da Figura 5); salvar o sumário gerado (na parte inferior da tela) em um arquivo texto (botão 7 da Figura 5); encerrar a execução e fechar o protótipo (botão 8 da Figura 5).

3.4 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou o desenvolvimento de um protótipo para sumarização automática de textos, através do método de palavras-chave. Fez-se uso do algoritmo EPC-P para a extração das palavras-chave utilizadas no processo de sumarização.

Os resultados obtidos mostraram-se satisfatórios, visto que as notícias testadas no protótipo não perderam o sentido e mantiveram o tema central. Foram testadas dez notícias

diferentes. Para todas os resultados gerados foram considerados bons, alguns melhores que outros, porém sempre atendendo ao objetivo proposto. Foi possível também sumarizar textos de artigos científicos. Porém, nesses casos, por se tratarem de textos mais complexos e extensos, o resultado não se mostrou tão bom quanto a sumarização de notícias.

A maior restrição do trabalho fica por conta da utilização do analisador morfológico Palavras, que obriga a utilização da internet e a disponibilidade do serviço para execução do processo de sumarização. Outro restrição é a presença de caracteres especiais no texto a ser sumarizado. Nos testes realizados, foi possível notar falha no processo de extração de etiquetas quando são usados alguns caracteres especiais como o símbolo sustenido (#). O analisador morfológico não retorna uma resposta válida ou esperada, gerando erro no processo de extração das etiquetas.

Quanto aos trabalhos correlatos descritos, o que mais se assemelha ao presente protótipo para efeito de comparação é o GIST SUMMarizer (BALAGE FILHO; PARDO; NUNES, 2007). O Quadro 27 apresenta uma breve comparação entre eles.

Quadro 27 – Comparativo entre ferramentas

característica	**protótipo desenvolvido**	**GIST SUMMarizer**
plataforma	*desktop*	*desktop*
método de sumarização	palavras-chave	*gist*
abordagem	superficial	superficial
taxa de compressão	sim	sim
suporte a múltiplos arquivos	não	sim
visualização de detalhes da execução do processo	sim	não

4 CONCLUSÕES

Cada vez mais pesquisadores investem no estudo da sumarização automática de texto. Alguns com o intuito de colaborar com a área de Linguística Computacional e outros por necessitarem ferramentas para auxiliar na simplificação do acesso à informação. Assim, a sumarização automática pode ser aplicada como uma forma de resumir as informações disponíveis e facilitar a vida de quem busca por elas. Para desenvolver uma ferramenta para sumarização automática de texto, faz-se necessário o estudo do processamento de linguagem natural, a partir do qual é possível implementar as etapas do processo de sumarização usando uma abordagem de sumarização automática superficial ou profunda.

Durante o desenvolvimento do trabalho ficou evidente a dificuldade em formalizar a linguagem natural, pois, diferente das linguagens artificiais (linguagens de programação), é uma linguagem complexa e ambígua. Em função dessa dificuldade, optou-se por usar uma ferramenta para fazer as análises léxica e morfológica das palavras, visando focar o desenvolvimento no processo de sumarização, que é o objetivo principal do trabalho. Porém isto gerou algumas limitações no protótipo, tais como funcionar apenas com acesso a internet e, consequentemente, depender da disponibilidade da página web para o correto funcionamento do protótipo, além de depender do correto funcionamento do analisador morfológico no que diz respeito à etapa de etiquetagem das palavras.

Contudo, o presente trabalho atingiu os objetivos propostos, apesar das restrições citadas. Foi desenvolvida uma ferramenta que a partir de um texto-fonte gera um sumário extrativo de bom nível conforme os testes realizados. Além disto, o protótipo não limitou-se apenas a notícias como havia sido proposto, podendo também sumarizar textos de artigos científicos. Observa-se porém que os sumários gerados são mais adequados para notícias devido ao algoritmo utilizado.

As tecnologias e ferramentas utilizadas mostraram-se eficazes e tornaram mais ágil o processo de desenvolvimento do protótipo. O analisador morfológico Palavras, do projeto VISL, norteou a classificação das palavras e poupou um tempo considerável de desenvolvimento, mesmo sendo necessário extrair a informação retornada de um HTML. A linguagem C# facilitou bastante o desenvolvimento com o recurso de expressão regular nativa e funções prontas. A biblioteca PTStemmer também foi indispensável no desenvolvimento do

protótipo, pois a implementação de um algoritmo para obter radicais de palavras é algo complexo e que impactaria diretamente no tempo de desenvolvimento.

4.1 EXTENSÕES

Existem pontos que podem ser melhorados e incrementados no protótipo desenvolvido, sendo eles:

a) desenvolver analisadores léxico e morfológico próprios para identificar as classes gramaticais das palavras, removendo assim diversas limitações como a dependência do analisador Palavras, disponível na web;
b) adicionar outras técnicas de sumarização superficial, profunda ou híbrida ao protótipo, aumentando assim a possibilidade de gerar sumários diferentes;
c) permitir a sumarização de múltiplos arquivos de texto em paralelo;
d) permitir gerar sumários abstratos e não apenas extrativos;
e) permitir que os textos estejam em formatos diversos e não apenas em `txt`;
f) aceitar caracteres especiais no texto a ser sumarizado.

REFERÊNCIAS BIBLIOGRÁFICAS

ARANHA, Christian N. **Uma abordagem de pré-processamento automático para mineração de textos em português:** sob o enfoque da inteligência computacional. 2007. 144 f. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/10081/10081_1.pdf>. Acesso em: 09 abr. 2012.

BALAGE FILHO, Pedro P.; PARDO, Thiago A. S.; NUNES, Maria G. V. **Sumarização automática de textos científicos:** estudo de caso com o sistema GistSumm. São Carlos, 2007. Disponível em: <http://www.icmc.usp.br/~taspardo/NILCTR0711-BalageEtAl.pdf>. Acesso em: 08 abr. 2012.

BICK, Eckhard et al. **The parsing system:** Palavras. [S.l.], [2000]. Disponível em: <http://beta.visl.sdu.dk/visl/pt/parsing/automatic/parse.php>. Acesso em: 04 jun. 2013.

CARDOSO, Paula C. F.; PARDO, Thiago A. S.; NUNES, Maria G. V. Métodos para sumarização automática multidocumento usando modelo semântico-discursivos. In: WORKSHOP A RST E OS ESTUDOS DO TEXTO, 3., 2011, Cuibá. **Anais...** Cuibá: Sociedade Brasileira de Computação, 2011. p. 59-74. Disponível em: <http://www.icmc.usp.br/~taspardo/RST2011-CardosoEtAl2.pdf >. Acesso em: 09 abr. 2012.

CORACINI, Sandra R. **O resumo como parâmetro de avaliação da compreensão leitora**. 2009. f. 171. Dissertação (Mestrado em Letras) - Departamento de Letras, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/13951/13951_5.PDF>. Acesso em: 09 abr. 2012.

HEERDT, Paulo. **Sumarização do processo e do procedimento.** Porto Alegre, 1997. Disponível em: <http://www.heerdt.adv.br/site/artigos/sumarizacao_do_processo_e_do_procedimento.doc>. Acesso em: 13 mar. 2012.

MARGARIDO, Paulo R. A.; PARDO, Thiago A. S.; ALUÍSIO, Sandra M. **Sumarização automática para simplificação de textos:** experimentos e lições aprendidas. São Carlos, 2008. Disponível em: <www.icmc.usp.br/~taspardo/IHC-UAI2008-MargaridoEtAl.pdf>. Acesso em: 04 jun. 2013.

MARTINS, Camilla B. et al. **Introdução à sumarização automática**. São Carlos, 2001. Disponível em: <http://www.icmc.usp.br/~taspardo/RTDC00201-CMartinsEtAl.pdf>. Acesso em: 11 mar. 2012.

MÜLLER, Daniel N. **Processamento de linguagem natural**. Porto Alegre, 2003. Disponível em: <http://www.inf.ufrgs.br/~danielnm/docs/pln.pdf>. Acesso em: 04 abr. 2012.

OLIVEIRA, Pedro. **PTStemmer**: a stemming toolkit for the portuguese language. [S.l.], [2010]. Disponível em: <http://code.google.com/p/ptstemmer/>. Acesso em: 04 jun. 2013.

PARDO, Thiago A. S. **GistSumm:** um sumarizador automático baseado na idéia principal de textos. São Carlos, 2002. Disponível em: <http://www.icmc.usp.br/~taspardo/NILCTR0213-Pardo.pdf >. Acesso em: 10 abr. 2012.

_____. **Sumarização automática:** principais conceitos e sistemas para o português brasileiro. São Carlos, 2008. Disponível em: <http://www.icmc.usp.br/~taspardo/NILCTR0804-Pardo.pdf>. Acesso em: 15 mar. 2012.

PEREIRA, Marcel B.; SOUZA, Carolina F. R.; NUNES, Maria G. V. Implementação, avaliação e validação de algoritmos de extração de palavras-chave de textos científicos em português. **Revista Eletrônica de Iniciação Científica**, [S.l.], v. 2, n. 1, não paginado, 2002. Disponível em: <http://143.54.31.10/reic/edicoes/2002e1/cientificos/AlgoritmosDeExtracaodePalavrasChave.pdf>. Acesso em: 04 jun. 2013.

SENO, Eloize R. M. **RHeSumaRST:** um sumarizador automático de estruturas RST. 2005. 81 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de São Carlos, São Carlos. Disponível em: <http://www.bdtd.ufscar.br/htdocs/tedeSimplificado/tde_arquivos/3/TDE-2006-03-13T09:00:58Z-887/Publico/DissERMS.pdf>. Acesso em: 10 abr. 2012.

SILVA, Bento C. D. et al. **Introdução ao processamento das línguas naturais e algumas aplicações.** São Carlos, 2007. Disponível em: <http://www.icmc.usp.br/~taspardo/NILCTR0710-DiasDaSilvaEtAl.pdf>. Acesso em: 11 mar. 2012.

APÊNDICE A – Algoritmo EPC-P: preenchimento da `lista1`

A seguir é exemplifica a forma como é preenchida a `lista1` no algoritmo EPC-P. No Quadro 28 tem-se uma tabela onde as linhas representam cada uma das seis listas de padrões. A primeira coluna possui um ponteiro, representado por uma letra, para os elementos da lista. As demais colunas são as palavras pertencentes à lista. Cada elemento da lista é uma palavra com a quantidade de ocorrências da palavra no texto (letra `O`) e a quantidade total de ocorrências do padrão (`PO`) . No exemplo, a lista número 1 tem o ponteiro igual a (`[N = 1]`). Além disso, o primeiro elemento da lista (`Tatcher`) tem quantidade de ocorências igual a 7 (`O = 7`) e quantidade total de ocorrências do padrão igual a 10 (`PO = 10`), isto é 7 ocorrências da palavra "Tatcher" mais 3 ocorrências da palavra "família".

Quadro 28 – Listas de padrões

lista / índice	1	2	3
1 [N = 1]	Tatcher [O = 7; PO = 10] = 0,7	família [O = 3; PO = 10] = 0,3	null
2 [NPN = 1]	família de Tatcher [O = 5; PO = 9] = 0,55	...	...
3 [NA = 1]	mandados consecutivos [O = 3; PO = 12] = 0,25	...	...
4 [NAA = 1]	null	null	null
5 [NAPN = 1]	null	null	null
6 [NPNA = 1]	null	null	null

Inicialmente, todos os ponteiro estão fazendo referência ao primeiro elemento de cada lista. Todos os elementos apontados são analisados. Assim, tendo como base o Quadro 28, as palavras "Tatcher", "família de Tatcher" e "mandados consecutivos" são comparadas e é selecionada a que possuir a maior frequência relativa. A frequência relativa de uma palavra é igual a quantidade de ocorrências da palavra (`O = 7`) dividida pela quantidade de total de ocorrências do padrão (`PO = 10`). Para a palavra "Tatcher", a frequência relativa é igual a 0,7. A palavra adicionada a `lista1` na primeira análise é "Tatcher". Após a primeira análise, incrementa-se o ponteiro `N`. Na próxima análise, as palavras "família", "família de Tatcher" e "mandados consecutivos" são analisadas. Neste caso, a palavra "família de Tatcher" entra na `lista1` por ter a maior frequência relativa. O ponteiro `NPN` é incrementado. Até esse ponto, tem-se na `lista1` as palavras na seguinte ordem de inserção: "Tatcher" e "família de Tatcher". Ao final do processo, a `lista1` contém as palavras que são mais relevantes, por ordem de relevância, até o limite de 30 elementos.

APÊNDICE B – Algoritmo EPC-P: preenchimento da `lista2`

A seguir é exemplificada a forma como é preenchida a `lista2` no algoritmo EPC-P, que é a lista final de palavras-chave. No Quadro 29 tem-se na primeira linha os radicais da lista do padrão nome e na segunda linha a `lista1`. Ambas as listas possuem três elementos cada. Cada elemento das listas tem em sua descrição a palavra seguido do seu radical.

Quadro 29 – Lista de radicais de nomes e `lista1`

lista / índice	1	2	3
lista de radical	familiar | familia	Tatcher | tatch	ministra | ministr
`lista1`	Tatcher | tatch	governo | govern	família | familia

Todos os elementos da lista de radical nome são analisados. Então inicia-se o processo pela primeira palavra. Deve-se verificar se a palavra “familiar” possui o mesmo radical da palavra “Tatcher”. Como não possui, passa-se para a palavra “governo” e, por fim, chega-se na palavra “família”, que possui o mesmo radical da palavra “familiar”. Neste caso adiciona-se a palavra “família” na `lista2` e passa-se para a próxima palavra da lista de radical nome. Este processo é repetido até que todos os elementos da lista de radical nome tenham sido analisados. O resultado, nesse caso, é que na `lista2` tem-se as palavras “família” e “Tatcher”.