PROTÓTIPO PARA SUMARIZAÇÃO AUTOMÁTICA DE TEXTOS ESCRITOS EM LÍNGUA PORTUGUESA

ALEXANDRE BUSARELLO
JOYCE MARTINS

Roteiro

- Introdução
- Objetivos
- Fundamentação Teórica
- Especificação
- Implementação
- Operacionalidade
- Resultados e discussão
- Conclusão
- Extensões

Introdução

- Sumarização automática consiste em gerar resumos ou extratos de um texto, eliminando o que é irrelevante e mantendo o que for considerado importante
- Etapas principais:
  - Análise
  - Transformação
  - Síntese (Opcional)

Objetivos

- Desenvolver um protótipo para resumir textos, visando auxiliar no processo de sumarização automática
  - Sumarizar textos de artigos de notícias
  - Processar textos escritos de acordo com as normas gramaticais da língua portuguesa
  - Utilizar uma técnica da abordagem superficial para sumarização

FUNDAMENTAÇÃO TEÓRICA

Abordagem superficial

- Abrange técnicas de sumarização baseadas em estatística
- Não abrange a compreensão e a interpretação do texto
- Exemplo:
  - Método de palavra-chave

Etapas da sumarização automática: análise (PLN)

- Análise léxica e morfológica

Exemplo: "A casa caiu"

Palavra	Etiqueta	Descrição
A	ART	Artigo
casa	SUB	Substantivo
caiu	VP	Verbo no passado

Exemplo: "Alexandre casa hoje"

Palavra	Etiqueta	Descrição
Alexandre	NOP	Nome próprio
casa	SUB ou V?	Verbo
hoje	SUB	Substantivo

Etapas da sumarização automática: transformação

- Tem como entrada o texto analisado
- Seleção, agregação e/ou substituição de conteúdo
- Retorna o sumário intermediário ou final

Método de sumarização por palavra-chave

- Baseado na frequência das palavras
- A medida que o texto evolui, termos chaves aparecem com mais frequência
- Algoritmo EPC-P

Algoritmo EPC-P

- Identificação e criação das seis listas de padrões de palavras, que são:
  - Lista de nome
  - Lista de nome + preposição + nome
  - Lista de nome + adjetivo
  - Lista de nome + adjetivo + adjetivo
  - Lista de nome + adjetivo + preposição + nome
  - Lista de nome + preposição + nome + adjetivo

Algoritmo EPC-P

- Para identificar e criar estas listas, utiliza-se como base a lista de *tokens* etiquetados
- Exemplo: “Família de Tatcher”
    - “Família” - entra na lista de nome
    - “Família de” - não se encaixa em nenhum padrão
    - “Família de Tatcher” - entra na lista de nome + preposição + nome
    - “Tatcher” - entra na lista de nome

Algoritmo EPC-P

- Para cada palavra adicionada a uma das listas, deve-se adicionar junto quantas vezes está aparece no texto
- Após concluída a fase de construção das seis listas de padrões, deve-se criar mais seis listas somente com os radicais das palavras contidas nas seis listas de padrões
- As listas de radicais devem ser ordenadas de forma decrescente em relação as ocorrências das palavras

Algoritmo EPC-P

- Exemplo de montagem da `lista1`:

Índice/ Listas	1	2	3
1 N=1->2	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}	null	null
3 NA=1	mandados consecutivos {[O = 3; PO = 12] = 0,25}	null	null
4	null	null	null
5	null	null	null
6	null	null	null

Algoritmo EPC-P

- Após concluída a montagem da `lista1`, executa-se a criação da `lista2` com as palavras chave finais
- Para cada palavra na lista do padrão de radical nome, verifica-se existe um mesmo radical na `lista1`
- Caso exista, adiciona a primeira ocorrência encontrada na `lista1` na `lista2`
- Apenas adiciona na `lista2` se o radical da palavra ainda não existir na `lista2`

Algoritmo EPC-P

- Exemplo:

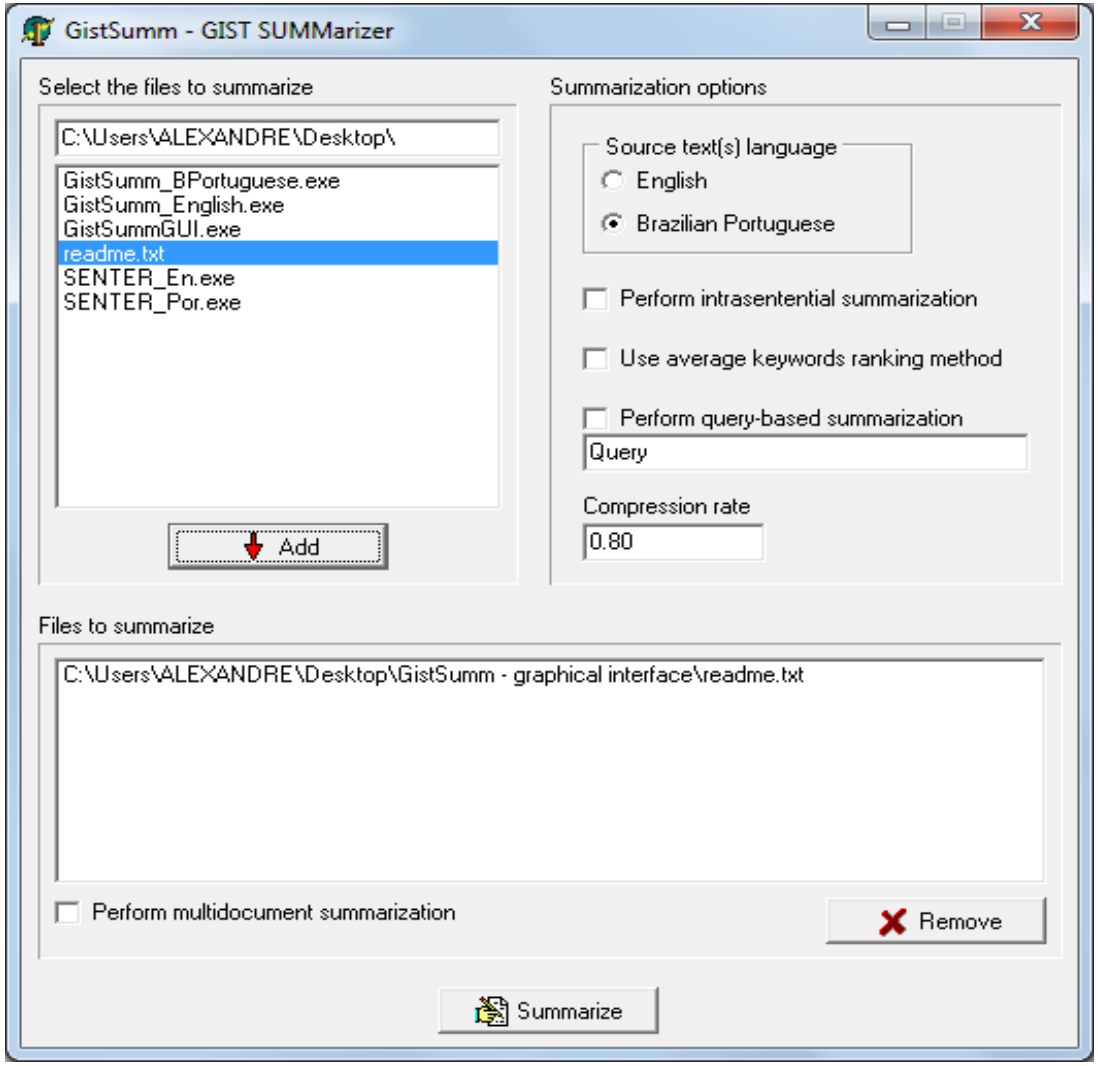
Índice/ Listas	1	2	3	4
Lista de radical nome	familiar | familia	Tatcher | tatch	Ministra | ministr	...
Lista1	Tatcher | tatch	governo | govern	família | familia	null

No exemplo acima, entra na lista2 as palavras: família e Tatcher

Algoritmo EPC-P

- Após obter a lista de palavras-chave final, estas são utilizadas para gerar o extrato
- Todas as sentenças que conter pelo menos uma ocorrência de alguma palavra-chave da `lista2`, será adicionada ao extrato

Trabalhos Correlatos: GIST SUMMarizer



ESPECIFICAÇÃO

Principais Requisitos Funcionais

- Disponibilizar interface para entrada de texto em português
- Permitir escolher o nível de compactação do sumário a ser gerado
- Mostrar os *tokens* e as etiquetas extraídas no processamento léxico e morfológico do texto

Principais Requisitos Funcionais

- Mostrar as listas utilizadas no algoritmo EPC-P, assim como as palavras-chave extraídas do texto-fonte
- Mostrar as palavras-chave e a classificação (*rank*) de cada sentença
- Alertar o usuário caso sejam detectados erros durante a sumarização

Processo de Sumarização: análise (VISL)

- Efetua análise léxica e morfológica do texto-fonte
- Retorna para o protótipo um HTML padronizado e formatado com os *tokens* e as etiquetas morfológicas
- O HTML é retornado através de uma requisição `GET`
- O protótipo extrai as etiquetas deste HTML padronizado

Processo de Sumarização: análise (VISL)

Enter text to parse:

o porta-voz da família de Thatcher informou que ela morreu em Londres

Go! Reset

Parser: Full morphosyntactic parse Visualization: Default

o [o] <artd> **DET** M S @>N
porta-voz [porta-voz] <Hprof> <tool> **N** M S @SUBJ>
de [de] <sam-> **PRP** @N<
a [o] <artd> <-sam> **DET** F S @>N
família [família] <HH> **N** F S @P<
de [de] **PRP** @N<
Thatcher [Thatcher] <hum> **PROP** F S @P<
informou [informar] <fmc> **V** PS 3S IND VFIN @FMV
que [que] **KS** @SUB **@#FS-<ACC**
ela [ela] **PERS** F 3S NOM @SUBJ>
morreu [morrer] <ve> **V** PS 3S IND VFIN @FMV
em [em] **PRP** @<ADVL
Londres [Londres] <civ> **PROP** F S @P<
.

Processo de Sumarização: transformação

- Após obter a lista de palavras-chave:
  - Cria-se uma lista com cada sentença do texto
  - Percorre-se a lista e para cada elemento verifica-se quantas palavras-chave existem na sentença:
    - Para cada palavra-chave existente, adiciona-se para a sentença, a quantidade de ocorrências no texto de cada palavra-chave
  - Calcula-se o ponto de corte para selecionar apenas as sentenças de acordo com a porcentagem de compressão

Processo de Sumarização: transformação

Valor da sentença com maior pontuação:

20

Percentual de compressão selecionado:

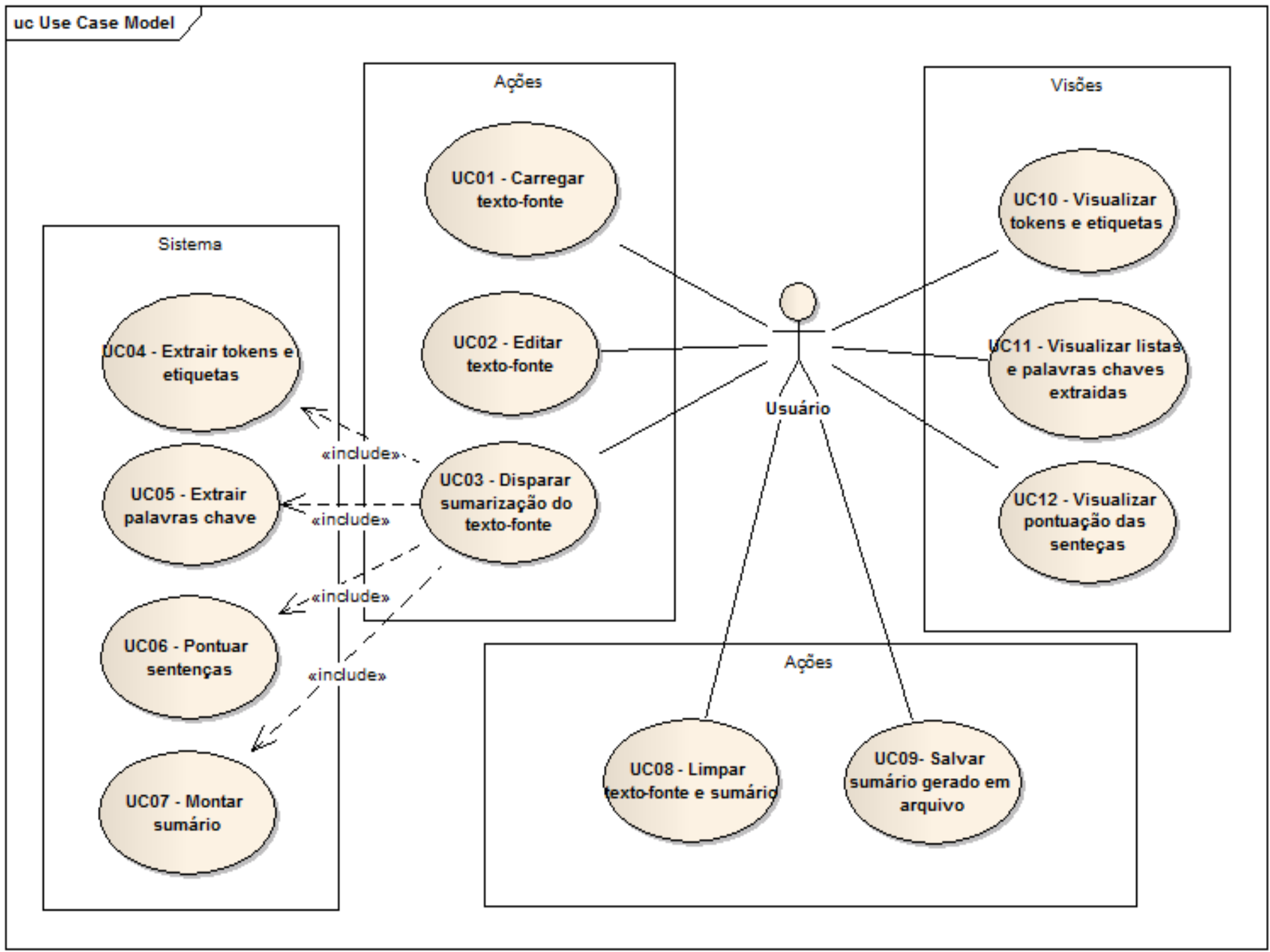
50%

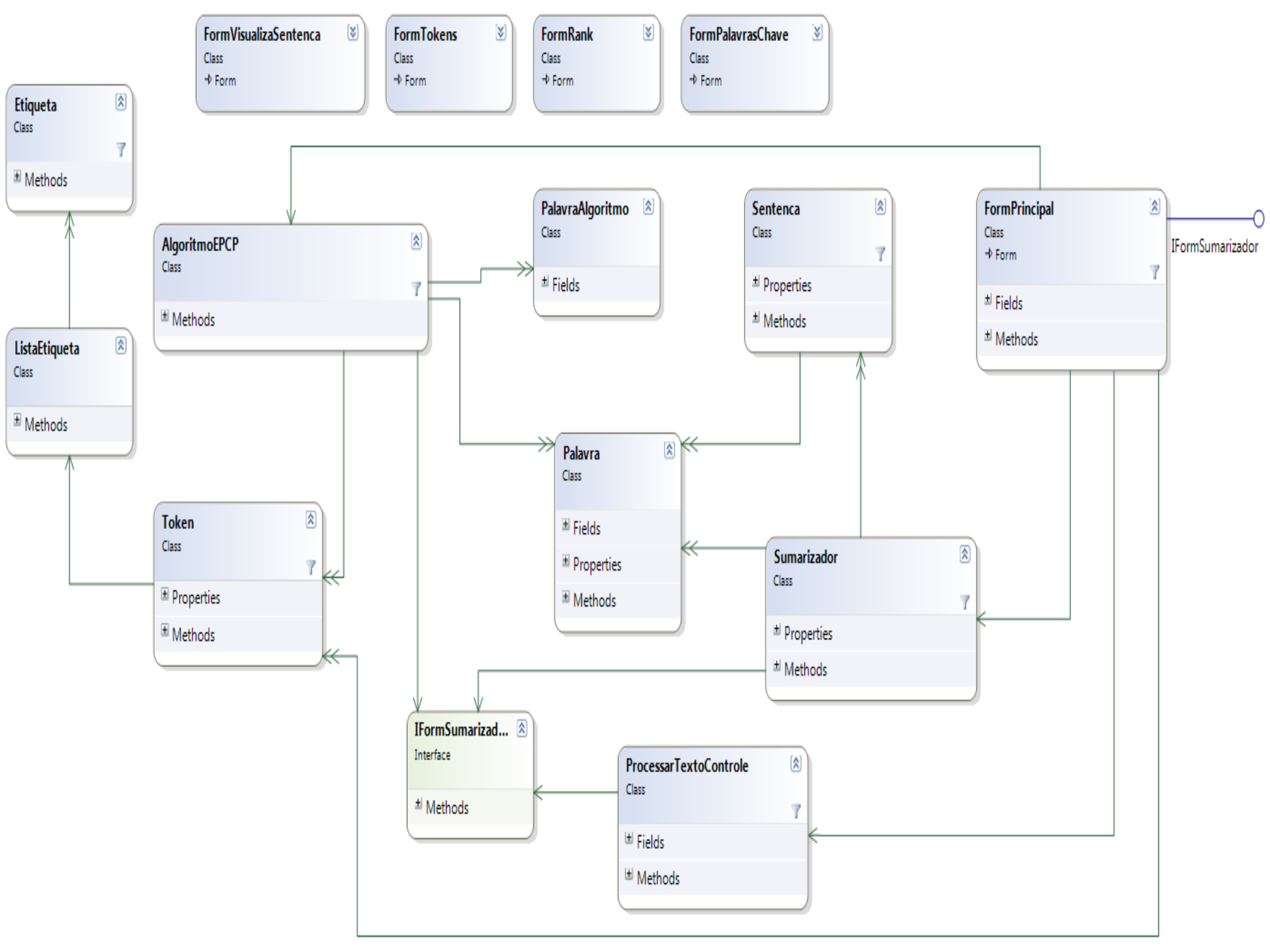
Cálculo:

(20 – 1) – 50% = 9,5

Trunc(9,5) = 9

9 = ponto de corte


uc Use Case Model
Ações
UC01 - Carregar texto-fonte
UC02 - Editar texto-fonte
UC03 - Disparar sumarização do texto-fonte
Visões
UC10 - Visualizar tokens e etiquetas
UC11 - Visualizar listas e palavras chaves extraidas
UC12 - Visualizar pontuação das senteças
Sistema
UC04 - Extrair tokens e etiquetas
UC05 - Extrair palavras chave
UC06 - Pontuar sentenças
UC07 - Montar sumário
«include»
«include»
«include»
«include»
Usuário
Ações
UC08 - Limpar texto-fonte e sumário
UC09- Salvar sumário gerado em arquivo


FormVisualizaSentenca
Class
Form
FormTokens
Class
Form
FormRank
Class
Form
FormPalavrasChave
Class
Form
Etiqueta
Class
Methods
AlgoritmoEPCP
Class
Methods
PalavraAlgoritmo
Class
Fields
Sentenca
Class
Properties
Methods
FormPrincipal
Class
Form
Fields
Methods
IFormSumarizador
ListaEtiqueta
Class
Methods
Palavra
Class
Fields
Properties
Methods
Token
Class
Properties
Methods
Sumarizador
Class
Properties
Methods
IFormSumarizad...
Interface
Methods
ProcessarTextoControle
Class
Fields
Methods

IMPLEMENTAÇÃO

Técnicas e Ferramentas

- Visual Studio com C#
- Windows Forms
- PTStemmer

PTStemmer

- Implementa os algoritmos de Orengo, Porter e Savoy

```
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"));
```

- Utilizado o algoritmo de Porter no protótipo

Processar retorno do VISL

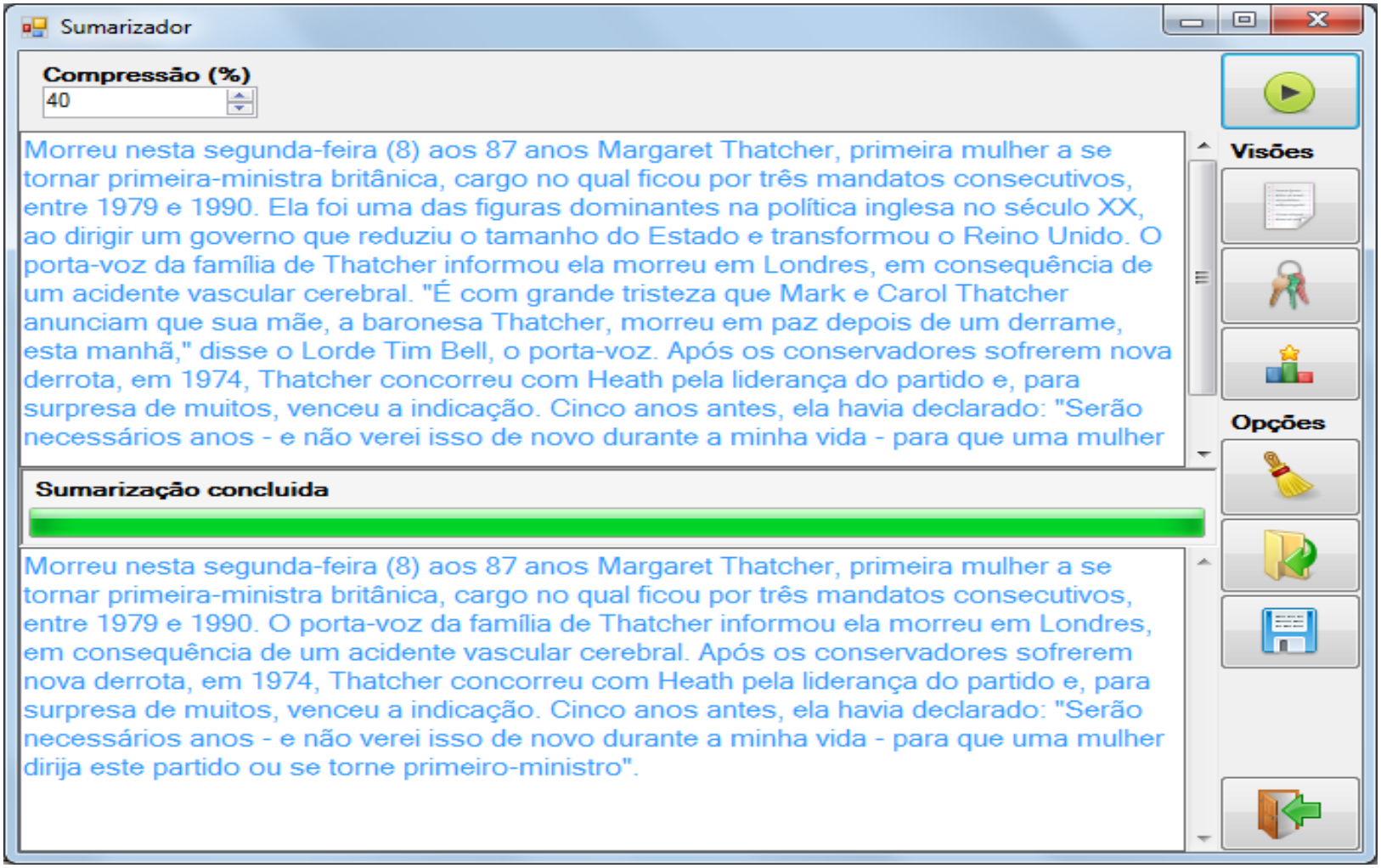
- Classe `ProcessarTextoControle`
- O VISL permite apenas analisar texto com no máximo 5430 caracteres
- Esta limitação foi contornada pelo protótipo enviando textos maiores que 5430 em várias partes

Algoritmo EPC-P

- Classe `AlgoritmoEPCP`
- Através desta classe obtém-se uma lista de `Palavra` com as respectivas palavras-chave identificadas pelo algoritmo
- A classe `Token` possui os métodos: `IsSubstantivo`, `IsNomeProprio`, `IsAdjetivo` e `IsPreposicao`, que auxiliam na identificação das palavras da lista de padrões
- A classe `Palavra` possui o método `Radicalizar`, que auxilia na criação das seis listas de radicais

OPERACIONALIDADE

Operacionalidade da Implementação



Resultados e Discussão

Caracteristica	Protótipo	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

Resultados e Discussão

- O protótipo mostrou melhores resultados com notícias, porém é possível efetuar sumários de textos de artigos científicos de forma menos eficiente
- Limitação do VISL para caracteres especiais, por exemplo o “#”

Conclusão

- Dificuldade em formalizar a linguagem natural:
  - Por este motivo, optou-se por usar o VISL para fazer a etapa de análise léxica e morfológica
    - Focou-se apenas na sumarização automática que foi o proposto no trabalho
    - Em contra partida foi criada uma dependência do protótipo com a disponibilidade do serviço do VISL

Conclusão

- Conseguiu-se atingir o objetivo:
  - A ferramenta foi capaz de gerar sumários a partir de um texto-fonte, porém podem ocorrer problemas referente a retornos inesperados do VISL.

Extensões

- Desenvolver analisadores léxico e morfológico próprios para identificar as classes gramaticais das palavras, removendo assim diversas limitações como a dependência do analisador VISL, disponível apenas na web
- Adicionar outras técnicas de sumarização superficial ou profunda ao protótipo
- Permitir a sumarização de múltiplos arquivos de texto

Extensões

- Permitir que os textos estejam em formatos diversos e não apenas em txt
- Aceitar caracteres especiais no texto a ser sumarizado
- Implementar um algoritmo próprio para radicalização de palavras

APRESENTAÇÃO DO APLICATIVO