UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SISTEMA PARA AVALIAÇÃO DE MONOGRAFIAS PARA TRABALHOS DE CONCLUSÃO DE CURSO

MARLON FERNANDO MANTAU ESPINDOLA

BLUMENAU
2012

2012/2-21

MARLON FERNANDO MANTAU ESPINDOLA

SISTEMA PARA AVALIAÇÃO DE MONOGRAFIAS PARA TRABALHOS DE CONCLUSÃO DE CURSO

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.

Prof. Antônio Carlos Tavares, Mestre – Orientador

BLUMENAU
2012

2012/2-21

SISTEMA PARA AVALIAÇÃO DE MONOGRAFIAS PARA TRABALHOS DE CONCLUSÃO DE CURSO

Por

MARLON FERNANDO MANTAU ESPINDOLA

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

Presidente:	Prof. Antônio Carlos Tavares, Mestre – Orientador, FURB
Membro:	Prof. Roberto Heinzle, Mestre – FURB
Membro:	Prof. José Roque Voltolini da Silva – FURB

Blumenau, 11 de dezembro de 2012

Dedico este trabalho a todos os amigos e a toda a minha família que me deu o apoio necessário para seguir em frente com esta idéia.

AGRADECIMENTOS

À Deus, pela força para superar todos os obstáculos.

À minha família que sempre me incentivou e apoiou, especialmente aos meus pais que sempre estiveram presentes para me auxiliar nesta jornada.

À minha namorada Kelly, pela compreensão em função de todo o tempo dedicado a elaboração deste trabalho.

Ao meu orientador Antônio Carlos Tavares, pela ajuda e por ter acreditado na conclusão deste trabalho.

Aos professores que direta ou indiretamente contribuíram para a minha formação.

RESUMO

No presente trabalho são descritas a especificação e a implementação de um analisador de monografias de Trabalho de Conclusão de Curso (TCC) do curso de Ciência da Computação. O protótipo lê um arquivo no formato PDF informado pelo usuário e faz uma checagem verificando se todas as seções necessárias para a construção de um TCC estão presentes. É checado também as listas de ilustrações, tabelas, símbolos, siglas e o sumário, verificando suas páginas e a citação de cada item no texto. Conceitos de análise léxica e sintática para fazer a análise da construção correta de cada item são utilizados. Também é feita uma checagem das referências descritas e das citações no texto inteiro.

Palavras-chave: Ciência da computação. TCC. Análise de texto. Extração de texto. Análise sintática. Análise léxica.

ABSTRACT

On this work is described the specification and the implementation of a monograph analyzer for a course conclusion work for computer science course. The prototype reads an archive in the PDF format set by the user and does checking to see if all the sections that are necessary in the construction of this course conclusion work are in it. It is checked the illustrations lists, tables, symbols, acronym and the summary, checking their pages and the quotation of each item in the text. Concepts like lexical analysis and syntactic analysis to do the analysis of each item and their construction are used. Also is done a checking for the bibliographic reference and their quotation in the text.

Key-words: Computer science. TCC. Text analysis. Text extraction. Lexical analysis. Syntactic analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

BNF – Backus-Naur *Form*

EBNF – *Extended* Backus-Naur *Form*

GALS – Gerador de Analisador Léxico e Sintático

HTML – *HyperText Markup Language*

ISO – *International Organization for Standardization*

PDF – *Portable Document Format*

SGBD – Sistema de Gerenciamento de Banco de Dados

TCC – Trabalho de Conclusão de Curso

UML – *Unified Modeling Language*

VCL – *Visual Component Library*

XML - *eXtensible Markup Language*

LISTA DE SÍMBOLOS

ε – épsilon

$ - cifrão

- sustenido

* - asterisco

% - por cento

@ - arroba

α – alfa

β – beta

γ – gama

SUMÁRIO

1 INTRODUÇÃO

Em algumas instituições de ensino o aluno de graduação ao chegar ao final de seu curso deverá apresentar um Trabalho de Conclusão de Curso (TCC) em formato de uma monografia. Segundo Drumond (2004), a monografia é uma pesquisa sobre conhecimentos existentes, publicados ou não, utilizando-se de atividades integradas, junto a uma metodologia, visando alcançar objetivos previamente definidos. A monografia é o resultado de uma investigação científica, apresentando uma contribuição relevante, original e pessoal para a ciência, (SALOMON, 1979, p. 121). Para Lakatos e Marconi (1991, p. 154), a monografia deve ter um valor representativo e obedecer uma metodologia rigorosa.

As monografias, em geral, apresentam a mesma estrutura: introdução, desenvolvimento e conclusão (LAKATOS; MARCONI, 1991, p. 155). Essas devem conter uma formatação e conteúdo formalmente estabelecido, descrita na NBR 14724 (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2005) para que seja devidamente aprovada como uma monografia de um TCC. Esta monografia deve ser construída usando-se diversas regras que servem para melhorar o tratamento e entendimento da idéia estudada e conferir a estas monografias uma padronização na sua construção (MONOGRAFIA, 2012).

Para o armazenamento em formato digital de uma monografia, hoje em dia é muito comum o uso do formato *Portable Document Format* (PDF), visto que este formato é amplamente utilizado no meio acadêmico e pelas organizações em geral para os mais diversos fins. O software é distribuído pela Adobe Systems gratuitamente. O formato PDF foi criado pela Adobe Systems e vem sendo aperfeiçoado nos últimos 17 anos. Este formato já tornou-se um padrão para a captura e revisão de informações em quase todos aplicativos ou sistemas operacionais (ADOBE, 2012). Atualmente o formato PDF é especificado segundo a norma ISO 32000 mantida pela Organização Internacional de Normalização (*International Organization for Standardization*) (ISO, 2012), possibilitando a criação de arquivos em PDF mais acessíveis.

Considerando-se os aspectos citados acima, esta proposta tem por objetivo construir uma ferramenta de software para analisar relatórios de TCC do curso de Ciência da Computação da Universidade Regional de Blumenau, no formato PDF. Depois desta leitura fazer a verificação se o mesmo apresenta as características esperadas para um documento desta natureza, conforme as normas estabelecidas pelo curso. Esta ferramenta será útil para

alunos em processo de elaboração de seu TCC como também para professores que terão que avaliar este documento após a sua conclusão pois ela aponta falhas básicas na construção dessa monografia que está sendo analisada, como a falta de uma referência descrita na referência bibliográfica e outros problemas que são descritos nos objetivos deste trabalho.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal é desenvolver uma ferramenta para ler arquivos no formato PDF de TCCs e verificar se os mesmos estão de acordo com as especificações de formatação adotadas pelo curso de Ciências da Computação da Universidade Regional de Blumenau.

Os objetivos específicos do trabalho são:

a) verificar os espaçamentos de margem e de parágrafos do documento;
b) verificar a existência de todas as seções de uma monografia (pré-textuais: capa, folha de rosto, folha de aprovação, dedicatória (opcional), agradecimentos (opcional), epígrafe (opcional), resumo, resumo na língua inglesa, lista de ilustrações (opcional), lista de tabelas (opcional), lista de siglas (opcional), lista de símbolos (opcional) e sumário; textuais: introdução, fundamentação teórica do trabalho, desenvolvimento e conclusão; pós-textuais: referências, apêndices (opcional) e anexos (opcional));
c) verificar a relação entre citações e referências bibliográficas;
d) verificar a existência de todos os itens descritos nas listas de ilustrações, listas de tabelas, listas de siglas e listas de símbolos, verificando sua respectiva página e a existência de pelo menos uma menção de cada item no texto;
e) verificar o sumário, examinando os capítulos e suas respectivas páginas.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos. O capítulo 2 apresenta as tecnologias envolvidas e os aspectos teóricos estudados para o desenvolvimento deste

trabalho. São relatados temas como análise léxica, análise sintática, autômatos finitos, estrutura de um PDF, uma introdução ao PDFtoolkit e o funcionamento da ferramenta Gerador de Analisadores Léxico e Sintático (GALS).

No capítulo 3 é abordado o desenvolvimento da ferramenta de software, detalhando a especificação e implementação.

O capítulo 4 apresenta as conclusões deste trabalho, bem como as sugestões para possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é dada uma introdução a interpretadores. Na Seção 2.2 são explanados alguns conceitos básico sobre análise léxica, expressões regulares e autômatos finitos. Na seção 2.3 são apresentados alguns conceitos da análise sintática. Na seção 2.4 é explicado um pouco sobre a estrutura do arquivo PDF. Na seção 2.5 é dada uma pequena introdução à biblioteca PDFtoolkit. Na seção 2.6 é demonstrado o uso da ferramenta GALS e seus conceitos. Na seção 2.7 são apresentados trabalhos correlatos ao tema em questão.

2.1 INTERPRETADORES

O interpretador é basicamente um compilador, mas ao contrário do compilador ele não gera código-objeto e sim executa o programa-fonte imediatamente. “Interpretadores são usados freqüentemente também em situações educacionais e de desenvolvimento de software, quando os programas são traduzidos e re-traduzidos muitas vezes” (LOUDEN, 2004, p. 4).

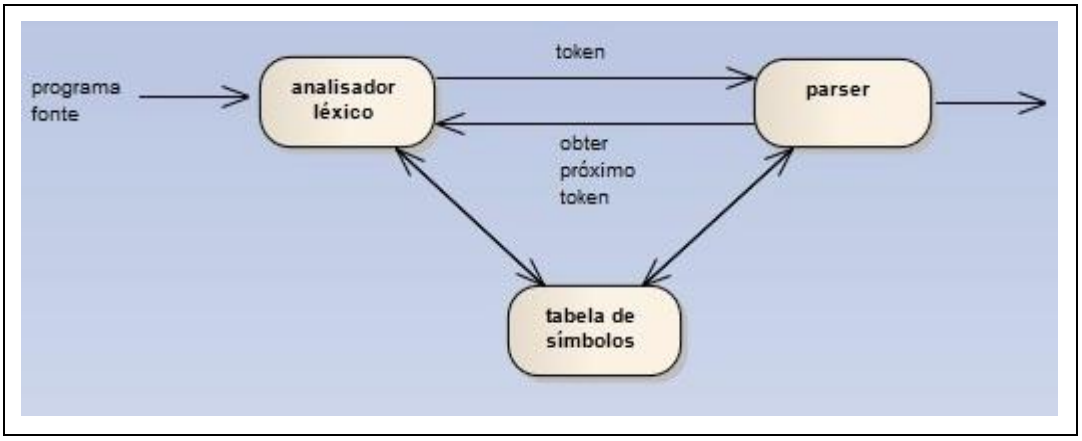
Dependendo da implementação, um interpretador possui várias fases, como a varredura inicial (análise léxica), análise sintática, análise semântica, otimização de código, geração de código e outras, sendo que as fases de análise léxica e análise sintática sempre se encontram presentes neste processo.

2.2 ANÁLISE LÉXICA

Uma das primeiras fases de um interpretador é a fase da análise léxica. Esta é a fase em que o interpretador lê o programa-fonte como um arquivo de caracteres e seleciona várias marcas identificando seus tipos e valores (LOUDEN, 2004, p. 31). O analisador léxico tem como tarefa passar esta seqüência de marcas para o *parser* do analisador sintático, para que ele possa fazer a sua análise (AHO; SETHI; ULMAN, 1995, p. 38).

Esta relação entre o analisador léxico e o *parser* pode ser vista na figura 1, onde exemplifica o analisador léxico sendo usado como uma sub-rotina ou uma co-rotina do

parser.



Fonte: adaptado de Aho, Sethi e Ullman (1995, p. 39).

Figura 1 - Interação do analisador léxico com o *parser*

As marcas também são conhecidas como *tokens*. Segundo Hiebert (2003, p. 14), os *tokens* são a unidade básica do texto de entrada, que são separados pelo analisador léxico e podem ser classificados como palavras reservadas, identificadores, símbolos especiais, constantes de tipos básicos (inteiro, real, literal, etc.) entre outras categorias dependendo da definição do analisador.

"Em geral, existe um conjunto de cadeias de entrada para as quais o mesmo *token* é produzido como saída. Esse conjunto de cadeias é descrito por uma regra chamada de um padrão associado ao *token* de entrada. O padrão é dito reconhecer cada cadeia do conjunto." (AHO; SETHI; ULMAN, 1995, p. 39). O lexema é um conjunto de caracteres que foi reconhecido por um padrão do *token*.

Algumas das características de um *token*, os padrões e lexemas, podem ser vistos no Quadro 1.

TOKEN	LEXEMAS EXEMPLO	DESCRIÇÃO INFORMAL DO PADRÃO
Const	**const**	**Const**
IF	IF	IF
Relação	<, <=, =, <>, >, >=	< ou <= ou = ou <> ou > ou >=
Id	pi, contador, D2	letra seguida por letra e/ou dígitos
Num	3.1416, 0, 6.02E23	qualquer constante numérica
Literal	"conteúdo da memória"	quaisquer caracteres entre aspas, exceto aspa

Fonte: adaptado de Aho, Sethi e Ullman (1995, p. 39).

Quadro 1 - Exemplos de *tokens*

Em geral os analisadores léxicos descartam os espaços que existem nos programas fonte, pois normalmente eles não são relevantes para a análise sintática, mas é importante guardá-los a fim de mostrar mensagens de erros mais amigáveis para o usuário (GRUNE, 2001, p. 53).

Para representar as regras para a formação e classificação do *token* são usadas as

expressões regulares, descritos a seguir.

2.2.1 EXPRESSÕES REGULARES

Segundo Grune (2001, p. 54), uma expressão regular é uma fórmula que descreve um conjunto de *strings* possivelmente infinito. Estas expressões podem ser consideradas como uma receita para gerar *strings* ou como um padrão para combiná-los. Alguns componentes básicos de expressões regulares podem ser observados no Quadro 2. Estes componentes são usados para construir as expressões que definem os padrões do *tokens*.

Padrões básicos:	Descrição:
X	O caractere x
.	Qualquer caractere, em geral com exceção de nova linha
[xyz...]	Qualquer dos caracteres x, y, z, ...
Operadores de repetição:	
R?	Um R ou nada (= opcionalmente um R)
R*	Zero ou mais ocorrências de R
R+	Uma ou mais ocorrências de R
Operadores de composição:	
R1 R2	Um R1 seguido por um R2
R1 | R2	Um R1 ou um R2
Agrupamento:	
(R)	O próprio R

Fonte: adaptado de Grune (2001, p. 54).

Quadro 2 - Componentes de expressões regulares

Uma outra opção para a representação destes padrões é o uso da descrição regular, já que as expressões regulares as vezes se tornam complexas para o entendimento (GRUNE, 2001, p. 55). Um exemplo de descrição regular pode ser visto no Quadro 3. Para este exemplo é usado a definição de um identificador em uma linguagem qualquer, onde ele é composto por uma sequência de letras ou dígitos, começados por uma letra. Não são permitidos dois sublinhados consecutivos e nem podem terminar com um sublinhado.

```
letra → [a-zA-z]
dígito → [0-9]
sublinha → _
letra_ou_dígito → letra | dígito
final_sublinhado → sublinha letra_ou_dígito+
identificador → letra letra_ou_dígito* final_sublinhado
```

Fonte: adaptado de Grune (2001, p. 55).

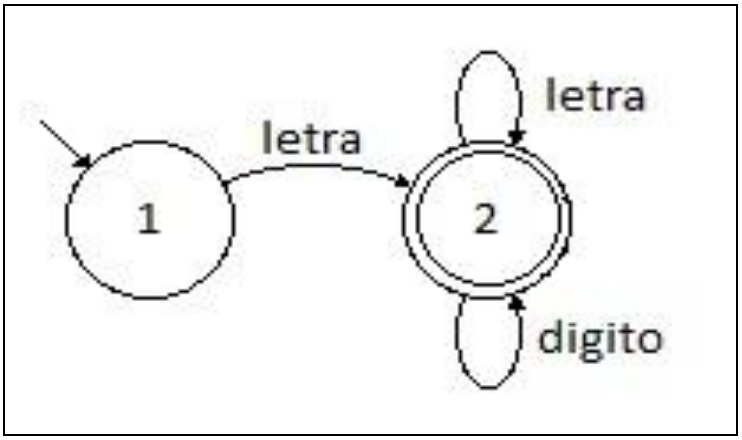
Quadro 3 - Exemplo de descrição regular

Através de autômatos finitos é possível representar de maneira mais prática as expressões regulares vistas nesta seção.

2.2.2 AUTÔMATOS FINITOS

Segundo Louden (2004, p. 47), autômatos finitos são uma forma matemática de escrever tipos particulares de algoritmos. Em particular para construção de sistemas de varredura, onde são usados para descrever um processo de reconhecimento de padrões em uma cadeia de caracteres.

O padrão de identificadores é normalmente definido pela seguinte definição regular: `identificador = letra (letra | dígito)*` (considerando que letra e dígito já tenham sido previamente definidos). (Figura 2).



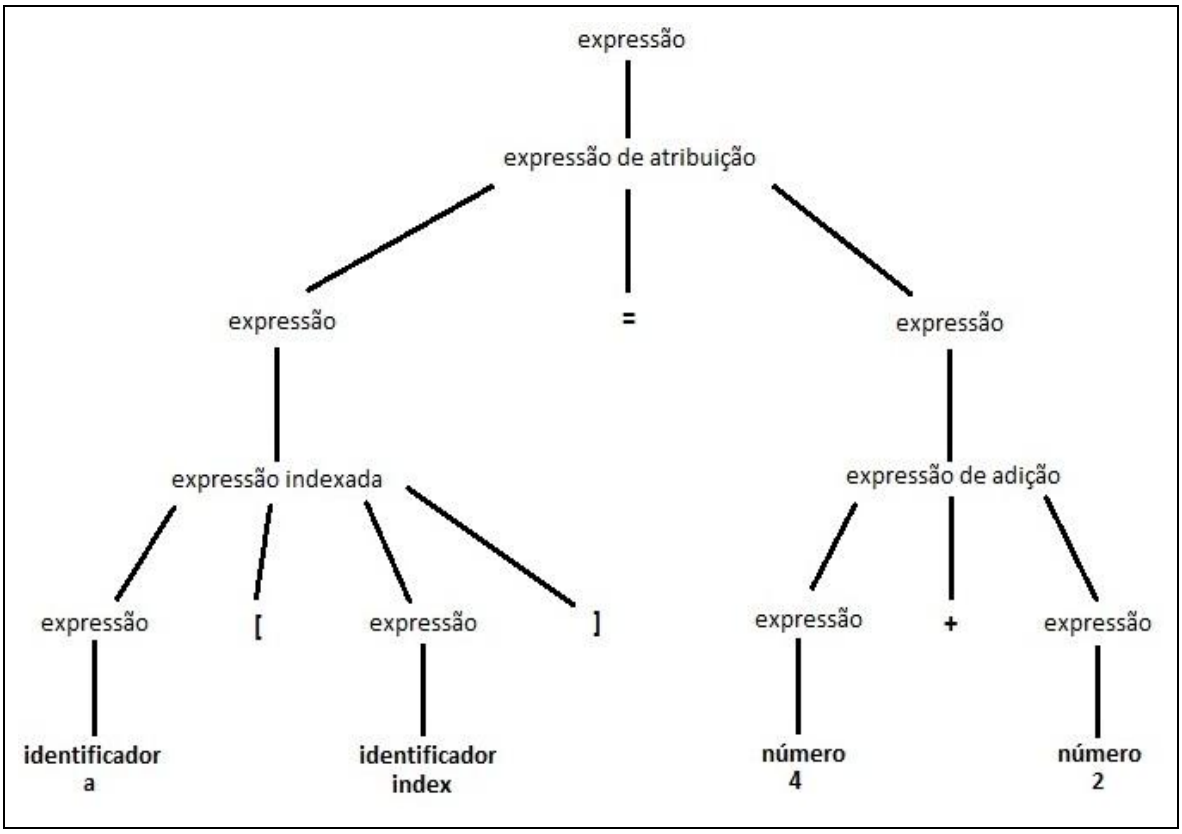
Fonte: adaptado de Louden (2004, p. 47).

Figura 2 - Um autômato finito para identificadores

Segundo Hiebert (2003, p. 15), o autômato parte de um estado inicial, analisando símbolo a símbolo até encontrar um símbolo que indique o final do *token*, e para que este *token* seja considerado válido, ele deve encontrar-se em um estado final. Caso contrário o *token* é considerado inválido.

2.3 ANÁLISE SINTÁTICA

Na fase da análise sintática, o analisador recebe os *tokens* do analisador léxico e verifica se esta pode ser gerada pela gramática da linguagem-fonte (AHO; SETHI; ULMAN, 1995, p. 72). Esta gramática é representada através das gramáticas livres de contexto. O resultado desta análise sintática é geralmente no formato de uma árvore sintática. Na Figura 3 é apresentada uma árvore sintática resultado de uma análise da linha de código `a[index]=4+2`, implementada em C.



Fonte: adaptado de Louden (2004, p. 9).

Figura 3 - Árvore sintática e o resultado da análise sintática

Existem vários métodos de análise sintática quem podem ser utilizados para fazer esta análise, mas os métodos mais utilizados são os *top-down* e os *bottom-up*. Como os nomes já dizem, os analisadores sintáticos *top-down* geram a árvore sintática de cima para baixo, enquanto os analisadores sintáticos *bottom-up* geram a árvore sintática de baixo para cima (AHO; SETHI; ULMAN, 1995, p. 72).

2.3.1 GRAMÁTICA LIVRE DE CONTEXTO

Segundo Louden (2004, p. 97), uma gramática livre de contexto é uma especificação de estrutura sintática da linguagem de programação, que é muito similar a especificação da estrutura léxica e que se utiliza de regras recursivas para a especificação das estruturas. “Uma gramática consiste em um conjunto de regras de produção e um símbolo de partida.” (GRUNE, 2001, p. 31).

Ao falar sobre gramáticas é comum o uso de algumas nomenclaturas para facilitar a sua identificação, sendo elas:

a) símbolos não terminais: são determinados pelo uso de letras maiúsculas, na maioria

A, B, C e N;

b) símbolos terminais: são determinados por letras minúsculas próximas do fim do alfabeto, como x, y e z;
c) sequências de símbolos gramaticais: são determinadas por letras gregas próximas ao início do alfabeto, como alfa (“α”), beta (“β”) e gama (“γ”);
d) letras minúsculas próximas ao início do alfabeto representam elas mesmas como terminais (a, b, c etc);
e) a sequência vazia é representada por épsilon “ε”.

Sendo assim, uma regra de gramática livre de contexto representada na forma de Backus-Naur *Form* (BNF) ficaria da seguinte maneira, como demonstra o Quadro 4.

```
exp → exp op exp | ( exp ) | número
op → + | - | *
```

Fonte: adaptado de Louden (2004, p. 97).
Quadro 4 - Exemplo de uma gramática livre de contexto em BNF

O primeiro símbolo (Quadro 4) é o nome da estrutura e o próximo símbolo é o meta-símbolo “→”. Depois deste símbolo aparecem uma cadeia de símbolos, um símbolo do alfabeto ou o meta-símbolo “|”.

A primeira regra no Quadro 4, define uma regra de expressão, que é composta por uma expressão, seguida de um operador e depois mais uma expressão, ou por um abre parênteses (“(”), uma expressão e depois por um fecha parênteses (“)”), ou então um número. A segunda regra define o operador, que pode ser um dos símbolos +, - e *.

Existem outras alternativas comuns para representar a gramática acima. Uma delas, muito comumente utilizada, é mostrada no Quadro 5.

```
<exp> ::= <exp> <op> <exp> | ( <exp> ) | NÚMERO
<op> ::= + | - | *
```

Fonte: adaptado de Louden (2004, p. 99).
Quadro 5 - Outro exemplo de especificação de gramática livre de contexto

2.3.2 ANÁLISE DESCENDENTE (*TOP-DOWN*)

Segundo Grune (2001, p. 104), o algoritmo da análise sintática descendente percorre a árvore de análise sintática do topo (raiz) até o fundo (folhas), em pré-ordem, ou seja, da esquerda para a direita.

Os tipos de analisadores sintáticos mais comuns de serem usados são os analisadores

recursivos com retrocesso e os analisadores recursivos preditivos.

Nos analisadores sintáticos recursivos com retrocesso, cada símbolo não terminal é implementado como um procedimento, que tentará o reconhecimento da regra. Se existir mais de uma regra de produção para este símbolo não terminal, o analisador tentará regra por regra até conseguir encontrar um sucesso ou sua falha em todos (PRICE; TOSCANI, 2001, p. 38).

Nos analisadores sintáticos recursivos preditivos o analisador tenta prever a próxima entrada com base em um ou mais *tokens* de verificação à frente do *token* atual (LOUDEN, 2004, p. 143). Mas, segundo Aho, Sethi e Ulman (1995), a gramática deve estar livre de recursão à esquerda e também fatorada a esquerda para que seja possível o uso deste tipo de analisador preditivo.

2.3.3 ANÁLISE REDUTIVA (*BOTTOM-UP*)

Este tipo de análise é uma tentativa de construir uma árvore de derivação a partir das folhas (fundo) até a raiz (topo), tentando derivar sempre mais a direita. É chamada redutiva pois a sentença de entrada vai sendo reduzida até o ponto em que atinge o símbolo inicial da gramática.

Estes analisadores redutivos geralmente são implementados por autômatos de pilha, e na sua fita de entrada está a sentença a ser analisada seguida de um marcador de fim ($).

> O processo de reconhecimento consiste em transferir símbolos da fita de entrada para a pilha até que tenha na pilha um lado direito de produção. Quando isso ocorre esse lado direito é substituído (reduzido) pelo símbolo do lado esquerdo da produção. O processo segue adiante com esses movimentos (empilhamento e redução) até que a sentença de entrada seja completamente lida, e a pilha fique reduzida ao símbolo inicial da gramática. (PRICE; TOSCANI, 2001, p. 54).

2.4 *PORTABLE DOCUMENT FORMAT* (PDF)

O objetivo dos arquivos formato PDF é disponibilizar para seus usuários uma maneira fácil e segura para visualizar e trocar entre si os mesmos. Independente do ambiente em que o usuário se encontra.

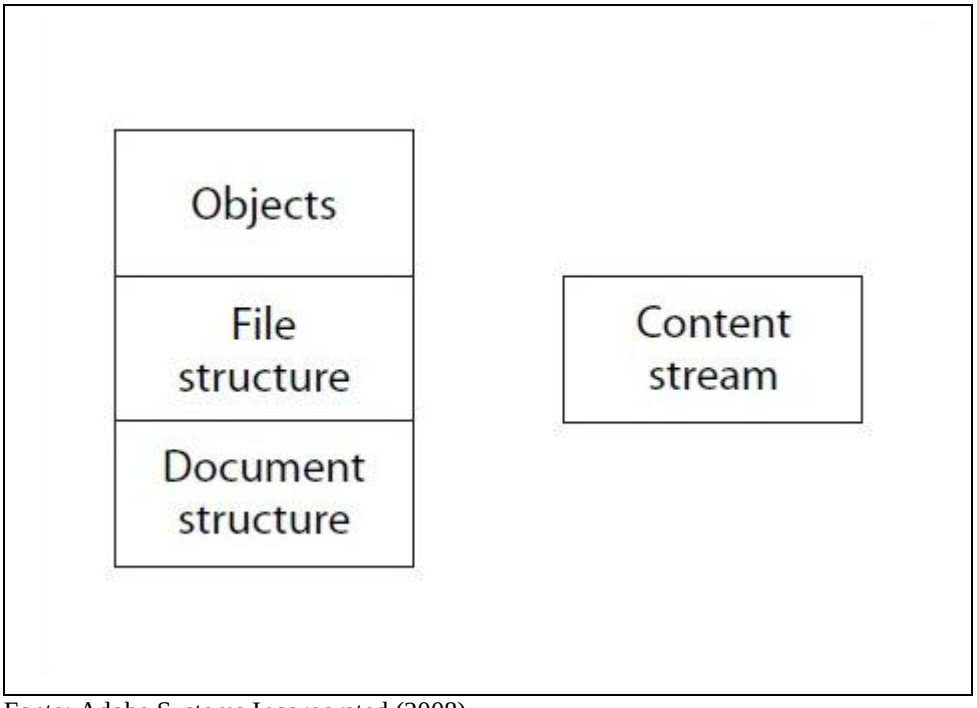
A ISO 32000 especifica uma maneira digital de representar documentos, chamados *Portable Document Form*. O PDF foi desenvolvido pela Adobe Systems Incorporated desde

1993 e em 2007 esta ISO foi editada (ADOBE SYSTEMS INCORPORATED, 2008).

2.4.1 ESTRUTURA DO PDF

Segundo Adobe Systems Incorporated (2008), na norma ISO 32000 a estrutura de um arquivo PDF é definida da seguinte maneira:

a) *objects* (objetos): um documento PDF é uma estrutura de dados composta por uma pequena gama de objetos básicos. Os objetos são do tipo booleano, inteiro, real, alfa-numéricos, nomes, matrizes, dicionários, *streams* e o objeto nulo;
b) *file structure* (estrutura do arquivo): esta estrutura determina como os objetos são guardados, como são acessados e como são atualizados;
c) *document structure* (estrutura do documento): nesta estrutura é especificado como os objetos básicos são usados para representar os elementos de um documento PDF: páginas, fontes, anotações e outros;
d) *content stream* (fluxo de conteúdo): aqui é guardada uma sequência de instruções descrevendo a aparência de uma página ou de outra entidade gráfica existente no documento.



Fonte: Adobe Systems Incorporated (2008).

Figura 4 - Componentes do PDF

Mais detalhes sobre a estrutura e funcionamento dos arquivos tipo PDF podem ser encontrados em Adobe Systems Incorporated (2008).

2.5 A BIBLIOTECA PDF *TOOLKIT* VCL

A ferramenta fornecida pela Gnostice, o PDFtoolkit, é uma biblioteca bastante abrangente e versátil que implementa varias funções para o desenvolvedor de sistemas. Esta biblioteca está disponível para Delphi e para C++ Builder. A biblioteca PDFtoolkit ajuda o desenvolvedor mascarando a complexidade de manipulação de arquivos no formato PDF, possibilitando a criação de softwares que criam, alteram, apresentam, asseguram, assinam digitalmente e outras funções com o arquivo PDF (GNOSTICE, 2012b).

2.6 GALS (GERADOR DE ANALISADORES LÉXICOS E SINTÁTICOS)

O GALS é uma ferramenta que ajuda a construir analisadores léxico e sintático para fazer a análise em uma sentença informada utilizando uma BNF (GESSER, 2003). O GALS é um software livre e *open source*.

Esta ferramenta permite a criação de analisadores léxicos e analisadores sintáticos, utilizando uma linguagem semelhante a notação *Extended* Backus-Naur *Form* (EBNF), com algumas diferenças para o seu funcionamento e também permite que no meio da construção da BNF sejam apontadas algumas ações semânticas, que são úteis para recuperação de *tokens* e também para fazer outros tipos de checagem no meio do processo da análise sintática (GESSER, 2003).

O GALS também possui uma ferramenta que permite fazer testes léxicos e sintáticos em suas BNF construídas (Figura 5), onde do lado esquerdo é a entrada do texto que vai ser analisado e o resultado desta análise é demonstrado do lado direito, em forma de árvore.

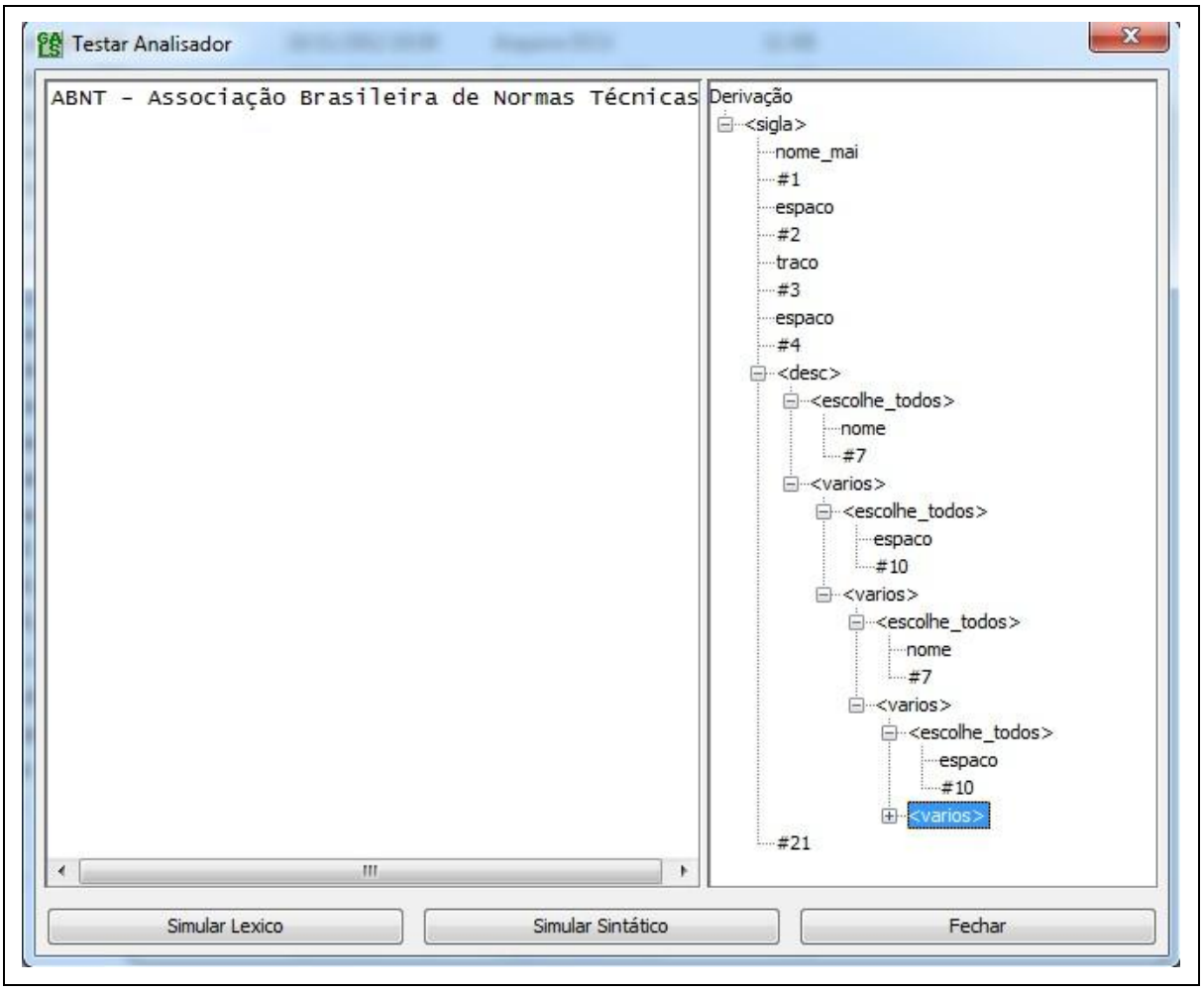


Figura 5 - Ferramenta de teste sintático do GALS

As ações semânticas são definidas no meio da BNF, definindo exatamente o local onde se espera uma ação ser tomada. Ela é definida por meio do caractere "#" seguido de um número que define o número daquela ação semântica. A Figura 6 demonstra um exemplo de uma BNF montada no ambiente do GALS e suas ações semânticas, onde podem ser vistas as definições regulares e os *tokens* definidos para esta BNF. Depois em "Não Terminais" são definidos as produções que serão utilizadas nesta análise e na seção "Gramática" é feito a implementação de cada uma dessas produções definidas anteriormente.

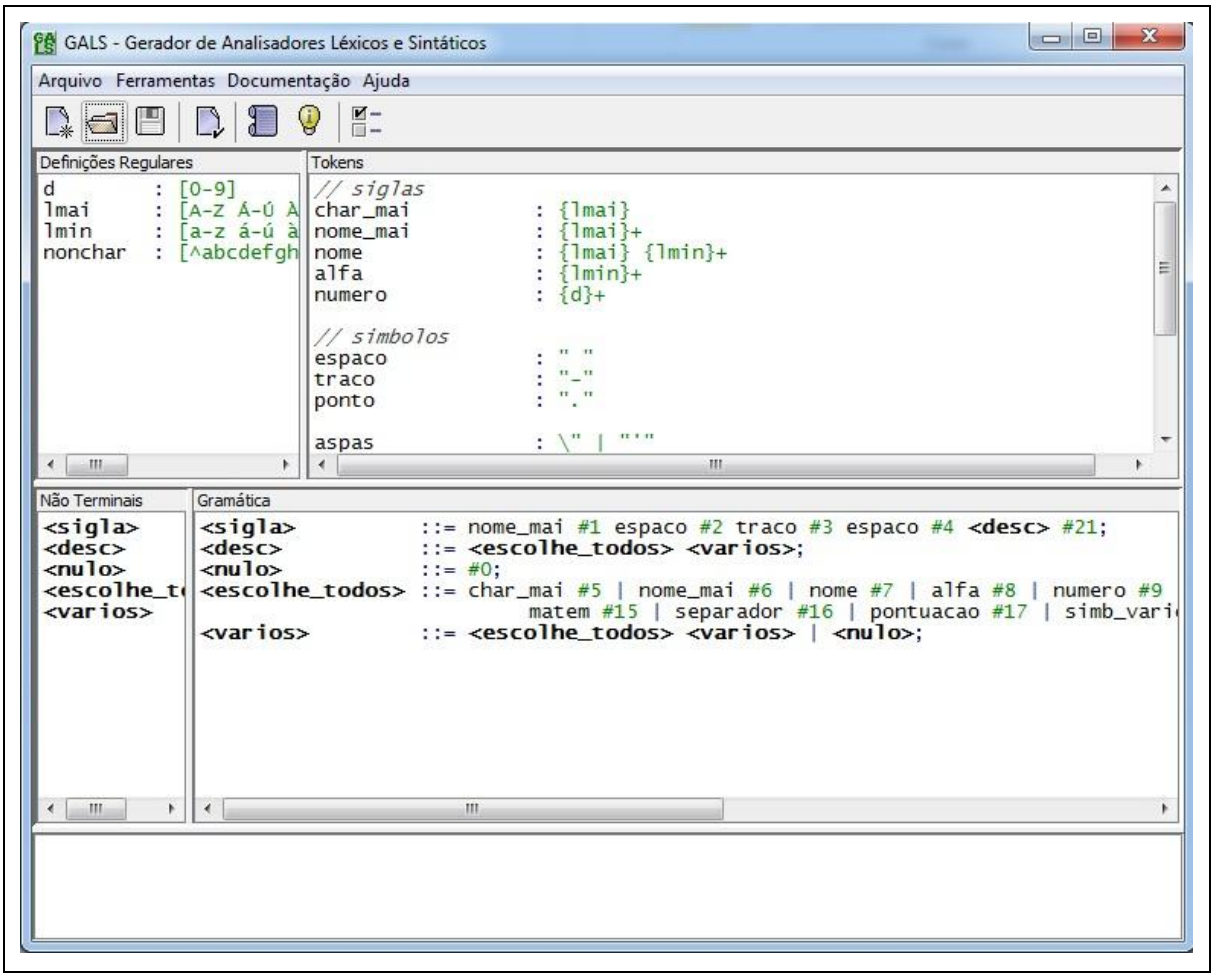


Figura 6 - Exemplo de construção de uma BNF no GALS

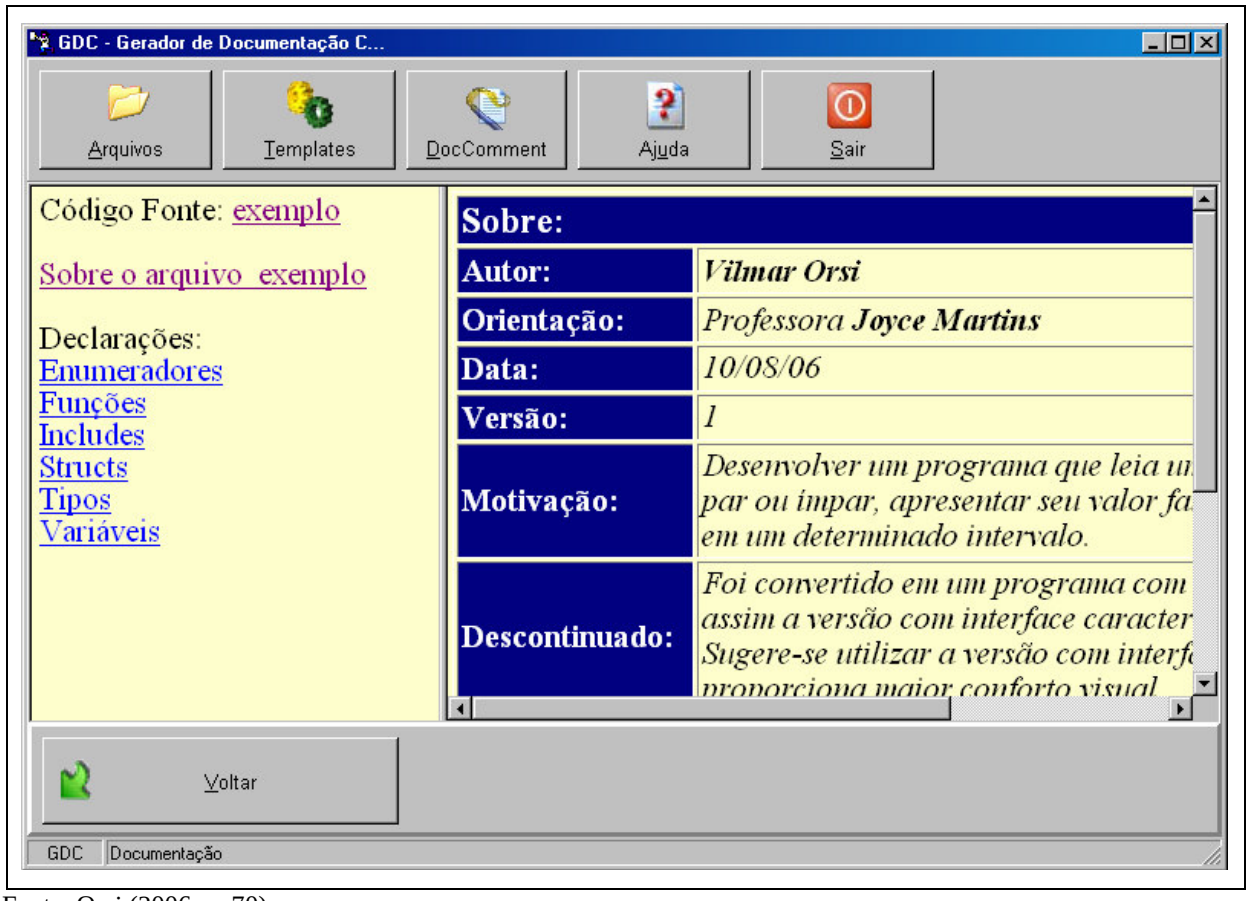
2.7 TRABALHOS CORRELATOS

Os trabalhos correlatos pesquisados para o tema proposto são o “Gerador de Documentação para Linguagem C, utilizando Templates” (ORSI, 2006), “Processamento de Texto Escrito em Linguagem Natural para um Sistema Conversor Texto-Fala” (OECHSLER, 2009) e o “Protótipo de um Compilador para a Linguagem PL/SQL” (HIEBERT, 2003). A seguir são apresentadas breves descrições destes trabalhos.

2.7.1 Gerador de Documentação para Linguagem C, utilizando Templates

No seu projeto, Orsi (2006) define um protótipo que se preocupa em ler códigos fontes

gerados na linguagem de programação C e gerar uma documentação do fonte. Este protótipo identifica comentários previamente escritos no código e também faz a documentação das linhas de código escritas. Para a geração desta documentação, o código é analisado léxica e sintaticamente pelo protótipo. Para a definição da BNF para as análises léxica e sintática foi utilizada a ferramenta GALS. Depois da análise feita no código fonte, a documentação é mostrada para o usuário em arquivos *HyperText Markup Language* (HTML) e formatada por um *template* selecionado pelo usuário, como mostra na Figura 7. O próprio protótipo disponibiliza uma ferramenta de construção de *templates* para serem usados na demonstração da documentação gerada.



Fonte: Orsi (2006, p. 70).

Figura 7 - Interface de documentação

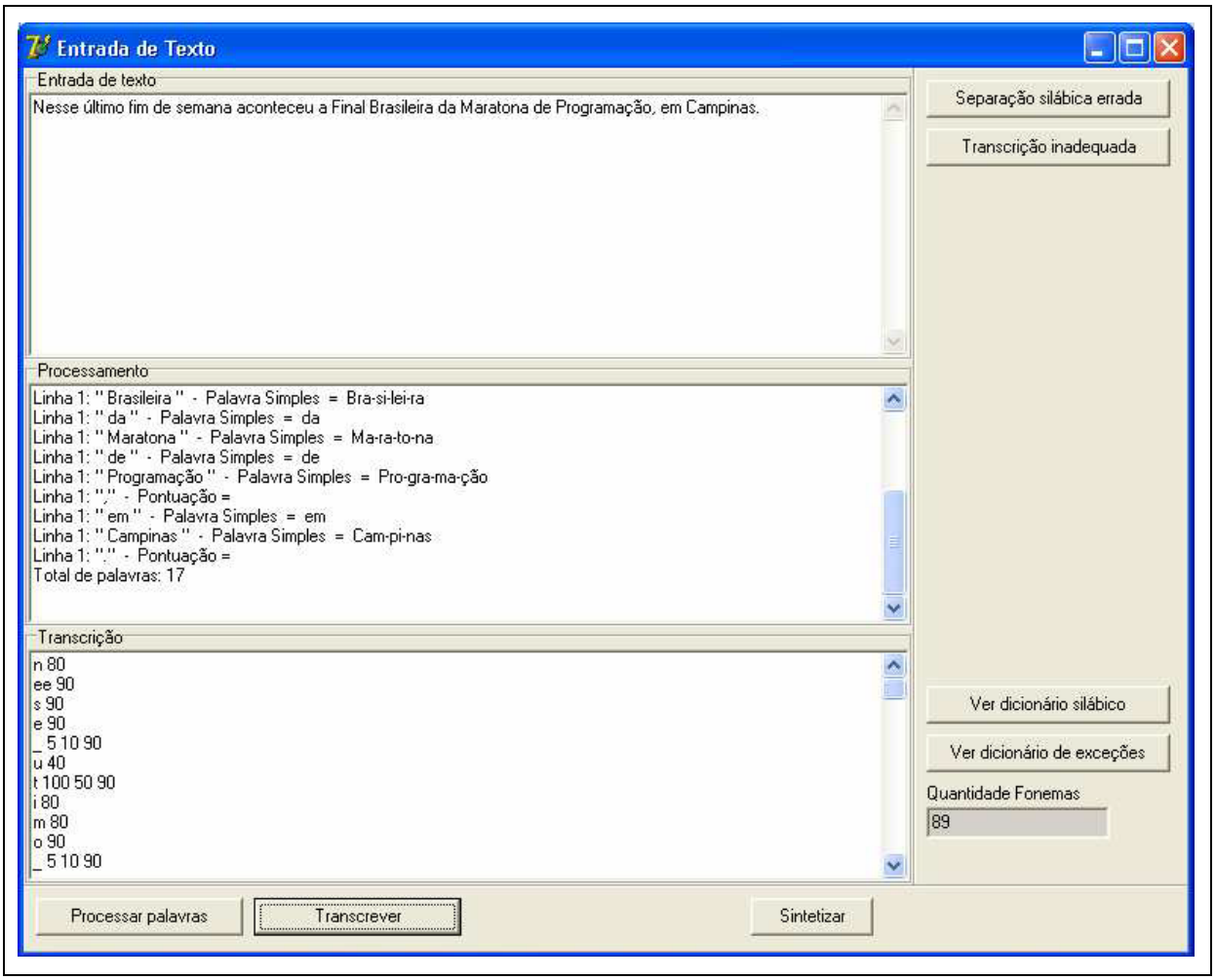
2.7.2 Processamento de Texto Escrito em Linguagem Natural para um Sistema Conversor Texto-Fala

No seu protótipo, Oechsler (2009) estudou uma maneira de fazer a leitura de um texto de entrada, escrito em português, e fez uma transcrição deste texto para um código

intermediário no formato para que o sintetizador MBROLA possa fazer seu reconhecimento corretamente.

Depois da entrada do texto pelo usuário no protótipo é feita uma separação silábica. Esta separação é demonstrada para o usuário, junto com a classificação das palavras separadas. O protótipo utiliza de análise léxica e sintática para validar e classificar o texto de entrada digitado, utilizando-se da ferramenta GALS para implementar estes analisadores.

No final, o protótipo gera um arquivo de saída com uma lista de fonemas que devem ser sintetizados, obedecendo o formato de entrada do sintetizador MBROLA (OECHSLER, 2009, p. 38) utilizado no protótipo, como pode ser visto na Figura 8.



Fonte: Oechsler (2009, p. 57).

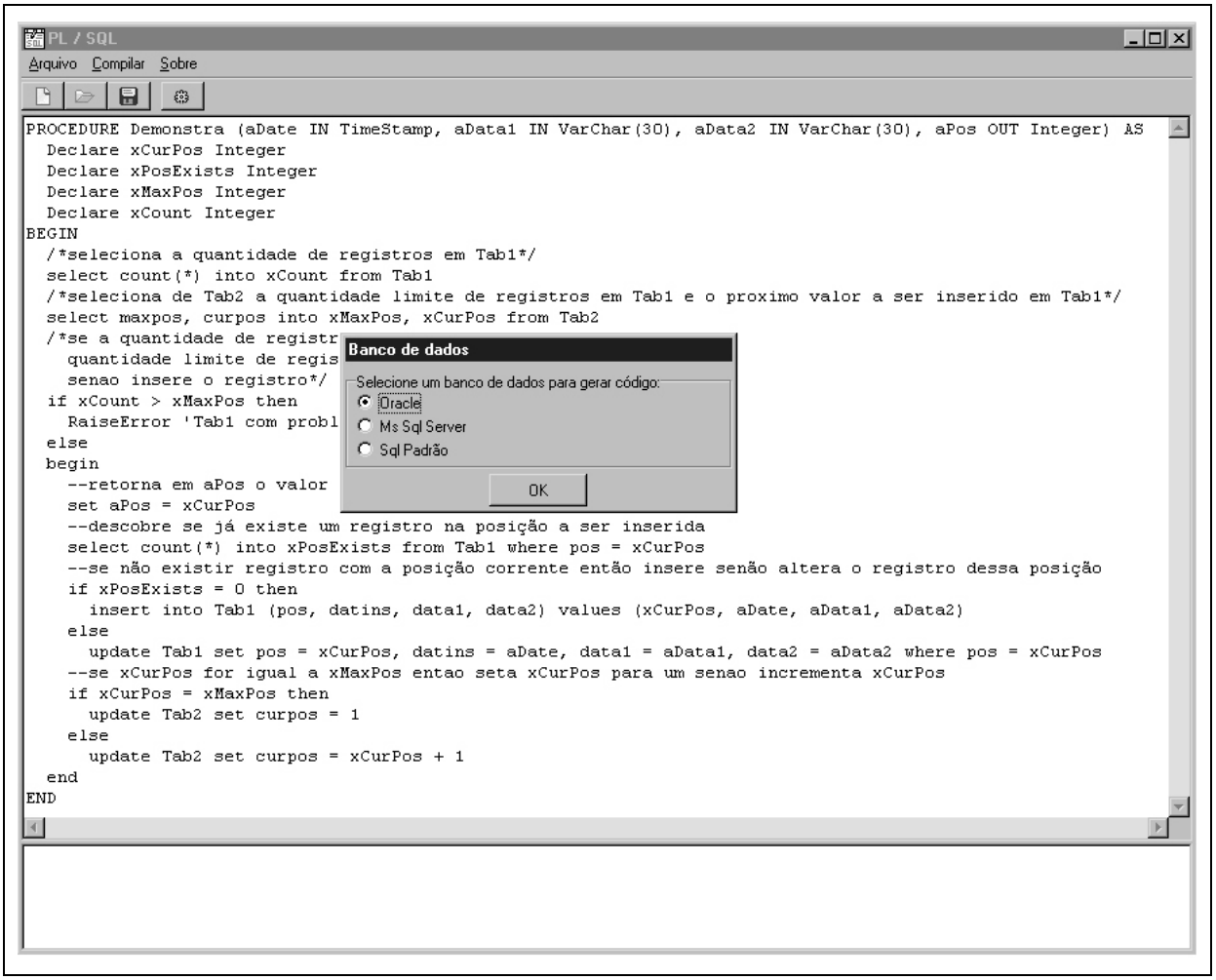
Figura 8 - Transcrição do texto e processamento das palavras

2.7.3 Protótipo de um Compilador para a Linguagem PL/SQL

Neste trabalho, Hiebert (2003) implementa um protótipo em que um conjunto de

comandos é digitado para o protótipo e com esses comandos são feitas verificações de checagem de erros e depois a geração de código específico para os Sistemas Gerenciadores de Banco de Dados (SGBD) Oracle e MS SQL Server. Para fazer esta definição da gramática dos comandos a serem digitados foi utilizada a notação BNF.

A Figura 9 mostra a tela principal do protótipo, com um código compilado e a escolha da geração do código para um SGBD.



Fonte: Hiebert (2003, p. 42).

Figura 9 - Tela para a escolha do SGBD

3 DESENVOLVIMENTO

Este capítulo detalha as etapas do desenvolvimento do protótipo. São apresentados os requisitos, a especificação e a implementação do mesmo, mencionando as técnicas e ferramentas utilizadas. Por fim, são indicados os resultados obtidos com este trabalho.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos do sistema encontram-se classificados em Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF).

a) o sistema deverá ser implementado na linguagem *Object* Pascal, utilizando o ambiente de programação Borland Delphi 7 (RNF);
b) o protótipo deverá fazer a leitura de um TCC em formato PDF para que possa fazer as análises (RF);
c) o protótipo deverá fazer a checagem das margem esquerda e dos parágrafos das páginas de texto do TCC (RF);
d) o protótipo deverá reconhecer a existência de todas as seções da monografia (pré-textuais: capa, folha de rosto, folha de aprovação, dedicatória (opcional), agradecimentos (opcional), epígrafe (opcional), resumo, resumo na língua inglesa, lista de ilustrações (opcional), lista de tabelas (opcional), lista de siglas (opcional), lista de símbolos (opcional) e sumário; textuais: introdução, fundamentação teórica do trabalho, desenvolvimento e conclusão; pós-textuais: referências, apêndices (opcional) e anexos (opcional)). Um erro deve ser apontado caso não encontre alguma destas seções (RF);
e) o protótipo deverá verificar se todas as referências descritas na seção de referências bibliográficas estão devidamente citadas no texto do TCC (RF);
f) o protótipo deverá verificar a existência de todos os itens descritos nas lista de ilustrações, lista de tabelas, lista de siglas e lista de símbolos, verificando se a página está corretamente apontada e pelo menos uma menção deste item no texto do TCC (RF);

g) o protótipo deverá verificar o sumário, examinando todos os seus capítulos e suas sub seções com suas respectivas páginas (RF).

3.2 ESPECIFICAÇÃO

Na seqüência é apresentada a especificação do protótipo, que foi modelado na ferramenta Entreprise Architect. O sistema foi desenvolvido seguindo a análise orientada a objetos, utilizando a notação *Unified Modeling Language* (UML) para a criação dos diagramas de caso de uso, classes e de seqüência.

3.2.1 Diagrama de casos de uso

O diagrama apresentado na Figura 10 apresenta a ação que pode ser tomada pelo usuário no protótipo.

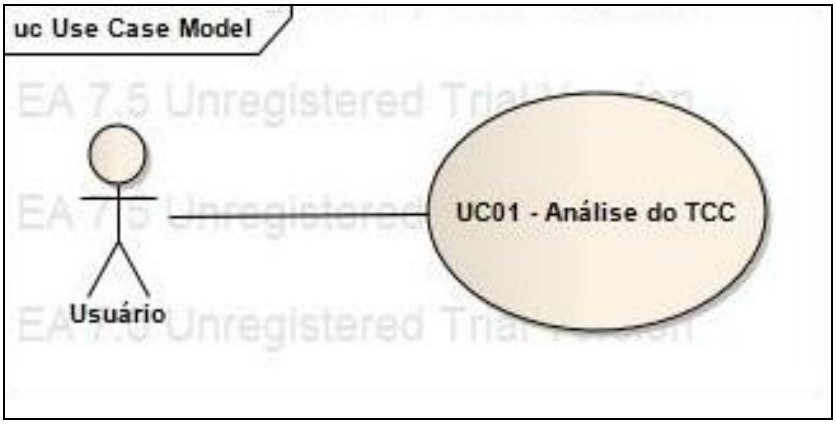


Figura 10 - Diagrama de casos de uso do protótipo

No caso de uso UC01, o usuário entra com algumas configurações básicas para que a análise possa ser feita corretamente. Depois disso o usuário escolhe o arquivo a ser analisado e manda o protótipo executar. Em seguida o protótipo irá mostrar para o usuário as mensagens de erro, caso tenha encontrado algum.

3.2.2 Diagrama de classes

Este diagrama de classes mostra uma visão de como as classes estão estruturadas e relacionadas entre si. Para facilitar a visualização este diagrama foi dividido em diagrama de classes do documento PDF, analisadores, analisador de referencias, analisador de siglas, analisador de símbolos, analisador de sumário e analisador de tabelas/ilustrações.

3.2.2.1 Pacote `DocumentoPDF`

O diagrama de classes do pacote `DocumentoPDF` (Figura 11) apresenta as classes criadas para manipular os dados retirados do documento originalmente selecionado pelo usuário, permitindo assim uma classificação de suas páginas e uma manipulação mais rápida e eficiente de toda a estrutura do documento (páginas e linhas).

A classe `TDocumentoPDF` é onde são guardadas todas as informações necessárias para a análise do documento. Esta classe é carregada logo no início da análise, com as informações retiradas diretamente do documento original selecionado pelo usuário.

A classe `TPaginasDocumento` é uma lista das páginas que o documento possui. É guardada em forma de lista em memória para maior velocidade de acesso e é indexada pelo número da página.

`TPagina` é uma classe que implementa as características de uma página com número da página e tipo de página. O atributo `fTipoPagina` é um tipo enumerado que classifica as páginas do documento para facilitar as análises do protótipo posteriormente. Esse atributo pode assumir os seguintes valores: `tpTexto`, `tpCapa`, `tpRosto`, `tpAprovacao`, `tpDedicatoria`, `tpAgradecimentos`, `tpEpigrafe`, `tpResumo`, `tpAbstract`, `tpIlustracoes`, `tpTabelas`, `tpSiglas`, `tpSimbolos`, `tpSumario`, `tpReferencias` e `tpOutras`.

A classe `TLinhasDocumento` é uma lista das linhas que cada página contém. Esta classe é guardada em forma de lista também para melhorar a performance e ainda possui alguns métodos de acesso rápido aos valores da linha, como o texto da linha.

A classe `TLinha` é basicamente a linha. Cada linha possui características como sua posição na página e seu valor texto.

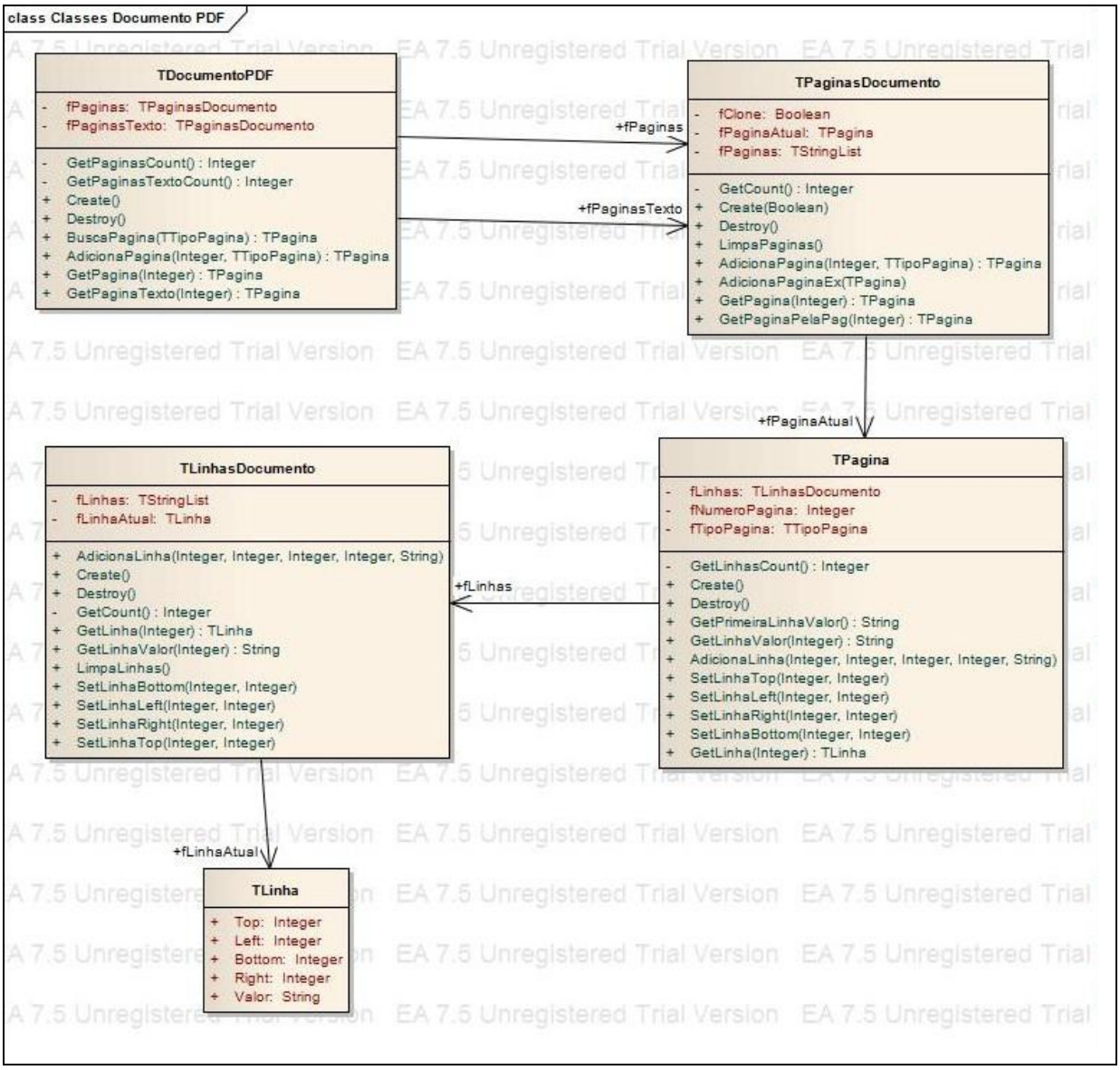


Figura 11 - Diagrama da classe do pacote `DocumentoPDF`

3.2.2.2 Pacote `Analisadores`

O diagrama de classe apresentado na Figura 12 representa as classes do pacote `Analisadores`, sua estrutura e relacionamento com outras classes.

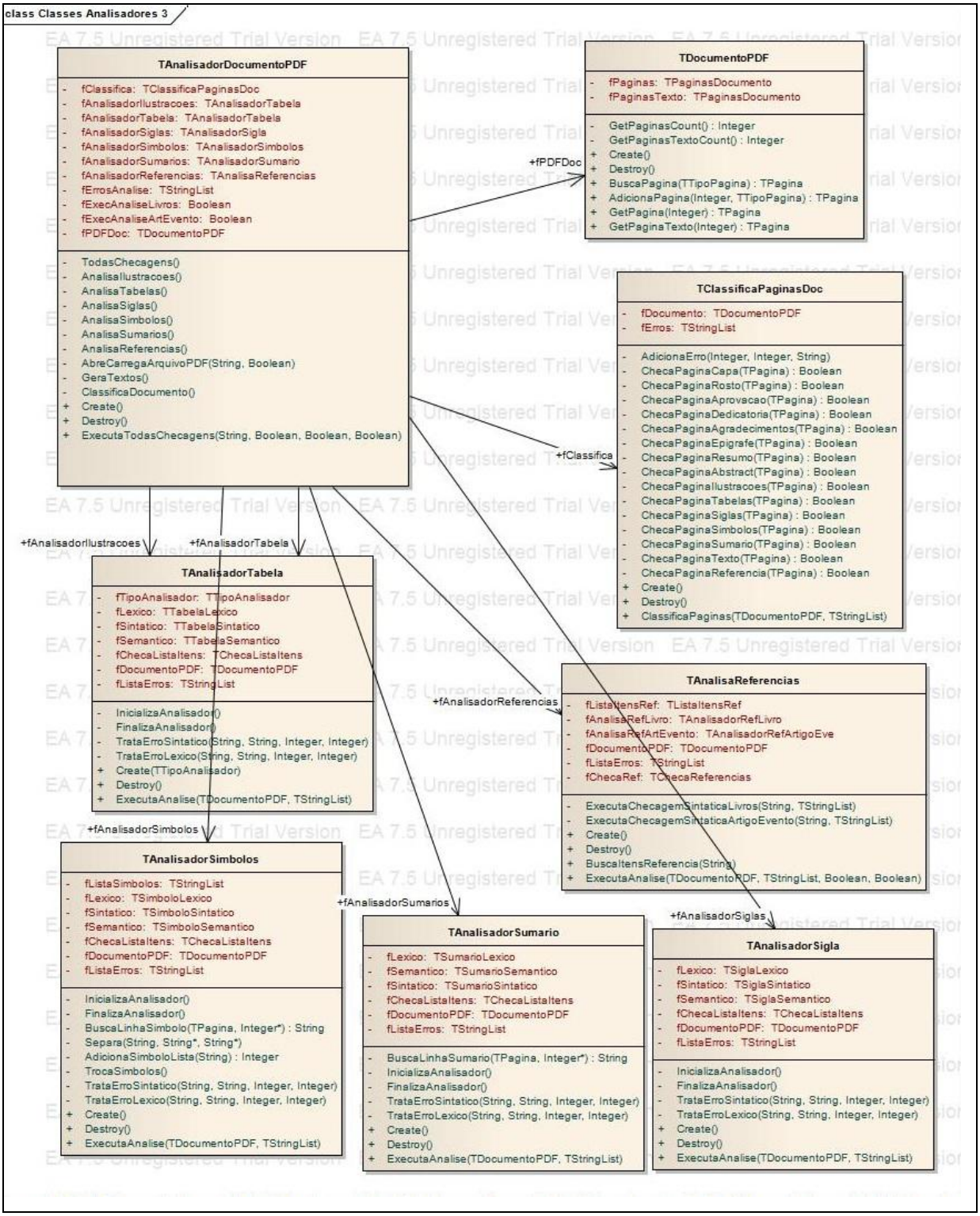


Figura 12 - Diagrama da classe do pacote `Analisadores`

A classe `TAnalisadorDocumentoPDF` é a classe principal do protótipo. É ela quem faz a conexão entre ele e o usuário. É nela que são acumuladas as mensagens de erro que os analisadores encontraram durante as análises. É ela que faz a carga do documento original PDF para a propriedade `fPDFDoc`, sendo esta propriedade do tipo classe `TDocumentoPDF`, que fica armazenado nesta classe até o final da execução das análises.

A classe `TClassificaPaginasDoc` é responsável por classificar as páginas deste documento e verificar a estrutura básica do documento, checando a existência das seções que compõem um TCC.

`TAnalisadorTabela` é responsável por realizar a análise no documento carregado para analisar posteriormente e relatar os erros com relação a lista de ilustrações e/ou lista de tabelas.

A classe `TAnalisadorSigla` é responsável pelo analisador documento carregado, para detectar os erros de construção da lista de siglas.

A classe `TAnalisadorSimbolos` faz a análise e relata os erros encontrados no documento carregado com relação a lista de símbolos.

`TAnalisadorSumario` é uma classe que analisa os erros encontrados no sumário do documento previamente carregado.

A classe `TAnalisaReferencias` é responsável pela análise e relatório de erros encontrados nas referências bibliográficas do documento carregado.

3.2.2.3 Pacote `AnalisadorReferencias`

Na Figura 13 é apresentado o diagrama de classes do pacote `AnalisadorReferencias`, mostrando sua estrutura e seus relacionamentos com outras classes.

A classe `TAnalisaReferencias` é responsável por identificar as informações de autores e ano de publicação de cada referência que for encontrada na seção de referências bibliográficas. Depois de encontrar estas informações a analisador abastece a lista de itens de referência.

A classe `TListaItensRef` é uma lista de itens de referências que foi encontrada e abastecida na classe acima especificada. Ela é guardada em forma de lista em memória.

`TRecItemRef` é uma estrutura com as características principais de uma referência. Algumas delas são os autores e o ano da publicação de sua obra.

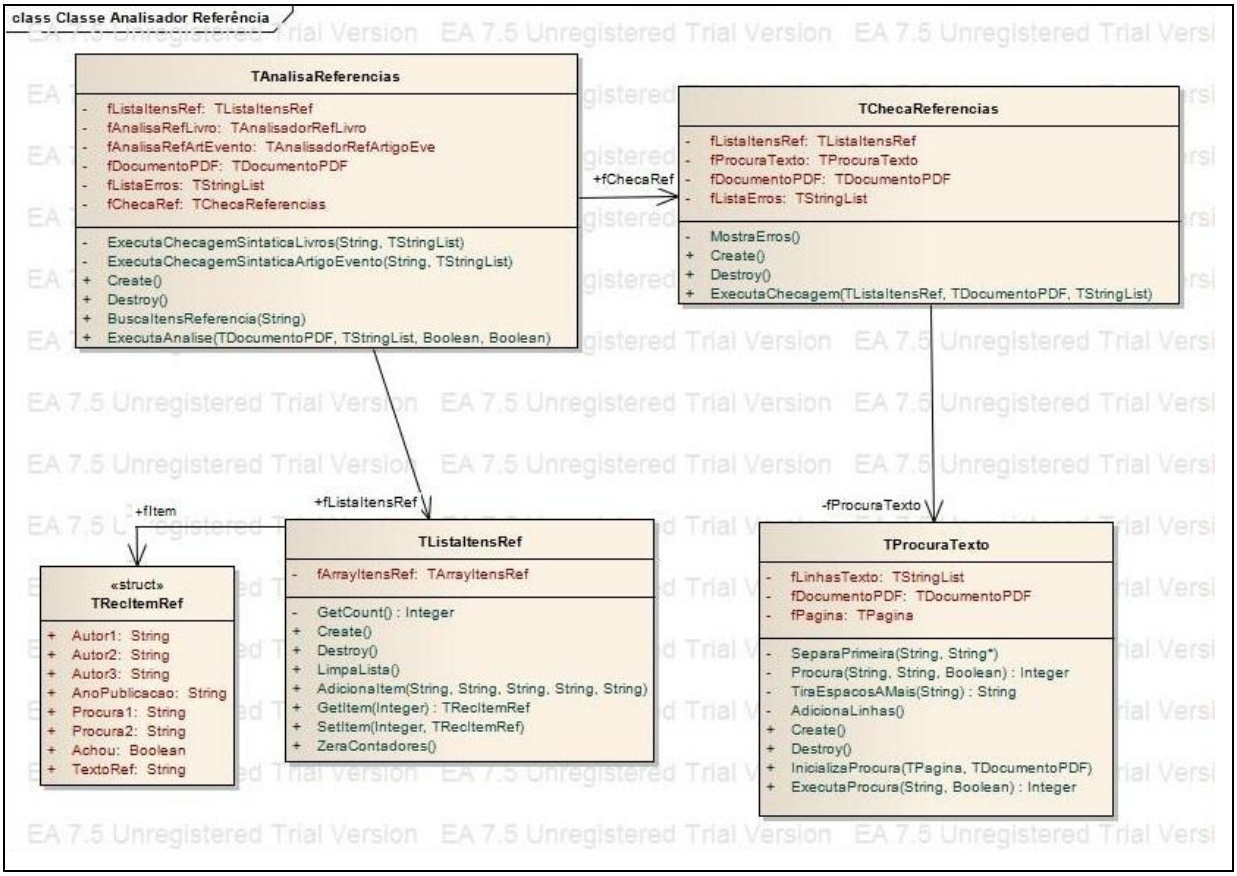


Figura 13 - Diagrama da classe do pacote `AnalisadorReferencias`

A classe `TChecaReferencias` é responsável por pegar a lista de itens de referências e fazer a varredura no texto, procurando pelas citações e caso não ache alguma reportar os erros.

A classe `TProcuraTexto` é responsável pela procura do item no texto, sendo que ele procura pelo item na página especificada e retorna se o item foi encontrado na página correta ou quantas vezes foi encontrado no texto inteiro.

3.2.2.4 Pacote `AnalisadorSiglas`

A Figura 14 demonstra as classes do pacote `AnalisadorSiglas` mostrando suas estruturas e seus relacionamentos com outras classes.

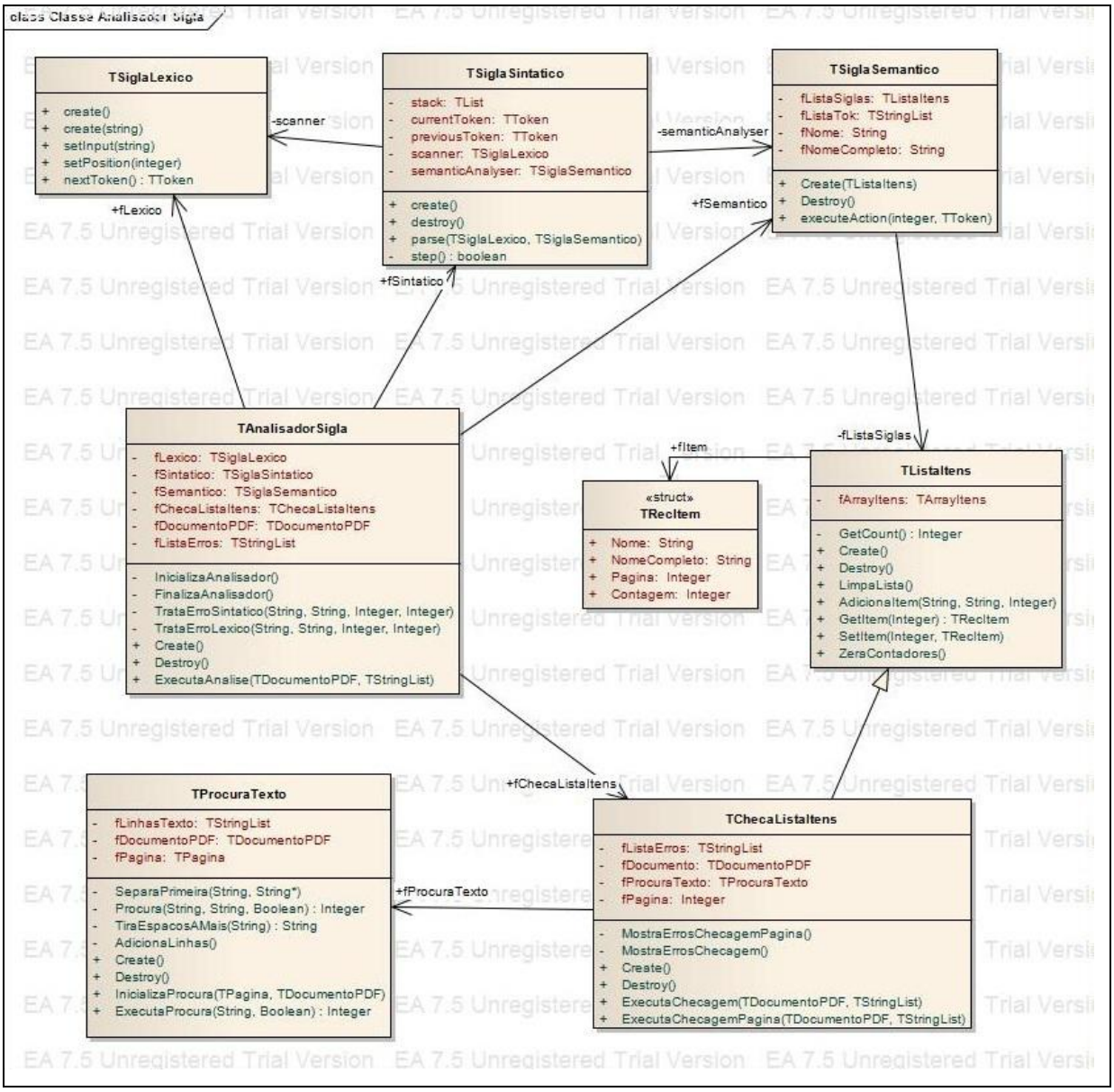


Figura 14 - Diagrama da classe do pacote `AnalisadorSiglas`

A classe `TAnalisadorSigla` faz a análise sintática de cada item de uma lista de siglas, analisando para verificar se cada item está escrito corretamente como especificado em Silva (2012a). Caso esta definição não esteja correta, os métodos `TrataErroSintatico()` e `TrataErroLexico()` relatam o erro para o usuário.

`TSiglaSintatico` é uma classe gerada automaticamente pelo GALS. Para a geração desta classe, o GALS usa uma BNF construída para que se possa fazer a análise de um item da lista de siglas. Esta BNF está descrita no Quadro 20, no apêndice A. Esta classe faz a análise sintática, retornando erro caso a sentença analisada não seja reconhecida pela BNF.

A classe `TSiglaLexico` também é gerada pelo GALS e auxilia o analisador sintático fazendo a análise dos *tokens* e separando-os e classificando-os para o uso.

`TSiglaSemantico` é mais uma classe gerada automaticamente pelo GALS, e parte da

BNF do Quadro 20. Esta classe contém os métodos que executam as ações semânticas definidas na BNF do GALS. Neste caso, as ações semânticas nada mais são do que remover cada item da lista de siglas e adicioná-los em uma lista em memória para o uso posterior.

A classe `TListaItens` é uma lista de itens que é guardada em memória objetivando um acesso mais rápido. Nela é guardado cada item que é analisado com sucesso na análise sintática da lista de siglas.

`TRecItem` é um registro que possui as características para a procura deste item no texto. Possui as características como nome, página e a contagem de aparições no texto.

A classe `TChecaListaItens` é uma classe herdada de `TlistaItens`, portanto possui a lista de itens já removida da lista de siglas. Esta classe é responsável por pegar cada item da lista de itens e procurar no texto e depois reportar os erros caso não as encontre.

A classe `TProcuraTexto` já foi devidamente especificada no item 3.2.2.3.

3.2.2.5 Pacote `AnalisadorSimbolos`

A Figura 15 mostra as classes do pacote `AnalisadorSimbolos` com a sua estrutura e seus relacionamentos.

A classe `TAnalisadorSimbolos` faz a análise sintática de cada item encontrado na seção lista de símbolos, analisando sintaticamente e reportando os erros caso eles existam.

`TSimboloSintatico` é uma classe gerada automaticamente pelo GALS e esta classe é gerada automaticamente baseando-se na especificação de uma BNF construída especificamente para analisar um item de uma lista de símbolos. Esta BNF pode ser vista no Quadro 22, no apêndice A. Esta classe retorna erro caso não consiga completar a análise sintática da sentença.

`TSimboloLexico` é outra classe gerada pelo GALS e ela auxilia o analisador sintático a fazer a sua análise, separando e classificando os *tokens* da sentença de entrada.

A classe `TSimboloSemantico` também é gerada pelo GALS e sua função é executar as ações semânticas definidas na BNF do GALS, que neste caso é recuperar e adicionar os itens analisados com sucesso em uma lista para um posterior uso.

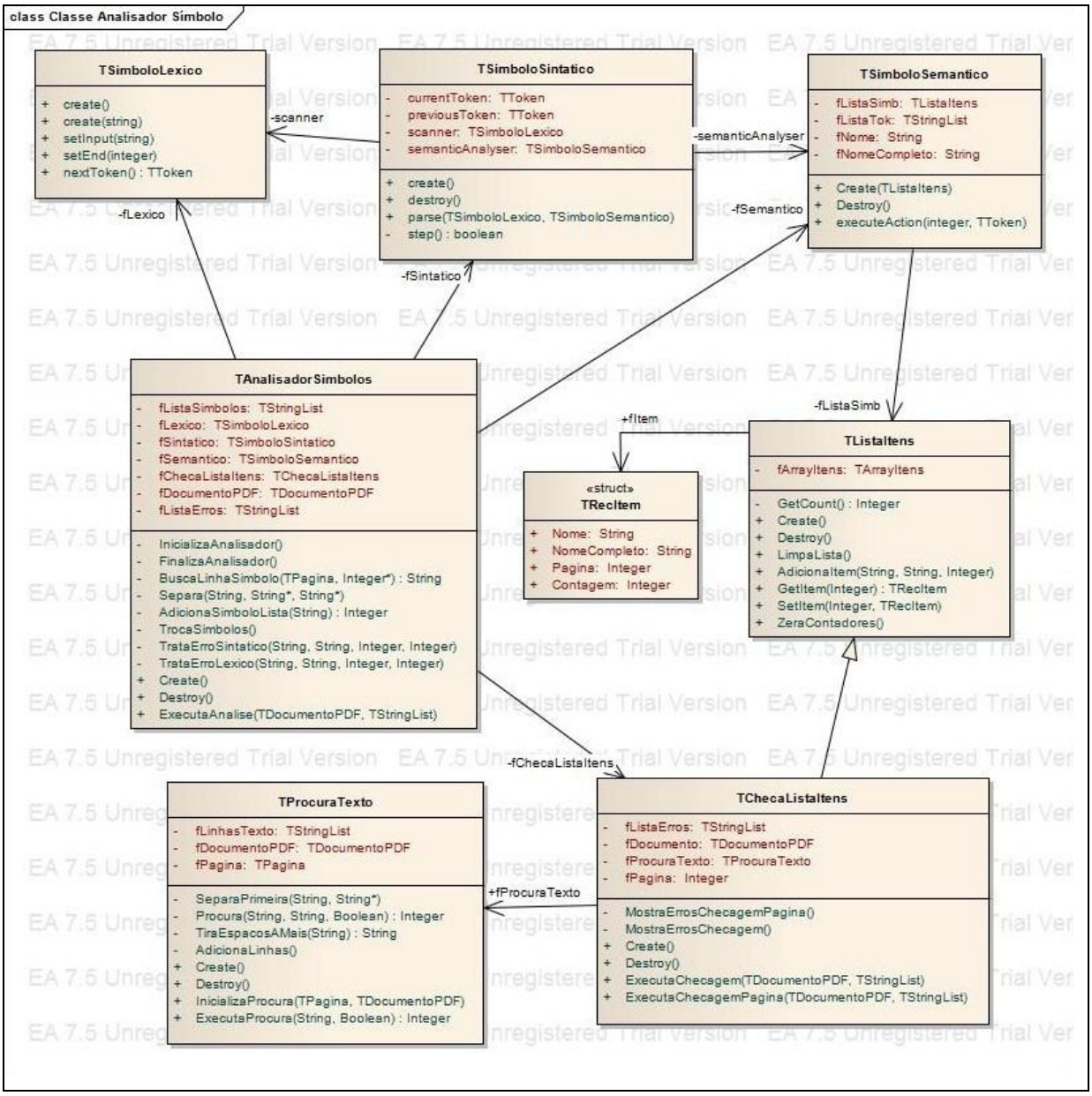


Figura 15 - Diagrama da classe do pacote `AnalisadorSimbolos`

As classes `TListaItens`, `TRecItem` e `TChecaListaItens` já foi devidamente especificada no item 3.2.2.4.

A classe `TProcuraTexto` já foi devidamente especificada no item 3.2.2.3.

3.2.2.6 Pacote `AnalisadorSumario`

A classe `TAnalisadorSumario` é a classe que faz a análise sintática de cada item encontrado na seção de sumário. Separa-os em uma lista para depois fazer a procura nas suas respectivas páginas.

`TSumarioSintatico` é uma classe gerada automaticamente pelo GALS. Esta classe é gerada sobre a construção de uma BNF específica para a análise de um item de sumário. A BNF usada para gerar essas classes pode ser vista no Quadro 24, apêndice A. Caso o analisador sintático não consiga reconhecer a sentença descrita pelo item ele retorna erro para o usuário.

`TSumarioLexico` é mais uma classe gerada automaticamente que é usada pelo analisador sintático para auxiliar na separação de *tokens* e sua classificação.

A classe `TSumarioSemantico` é outra classe gerada automaticamente pelo GALS e nela são executadas as funções semânticas definidas na BNF do GALS, que no caso desta BNF é identificar o item lido e adicioná-lo a uma lista junto da sua página, para fazer a procura no texto mais tarde.

As classes `TListaItens`, `TRecItem` e `TChecaListaItens` já foi devidamente especificada no item 3.2.2.4.

A classe `TProcuraTexto` já foi devidamente especificada no item 3.2.2.3.

A Figura 16 descreve as classes do pacote `AnalisadorSumarios` e sua estrutura.

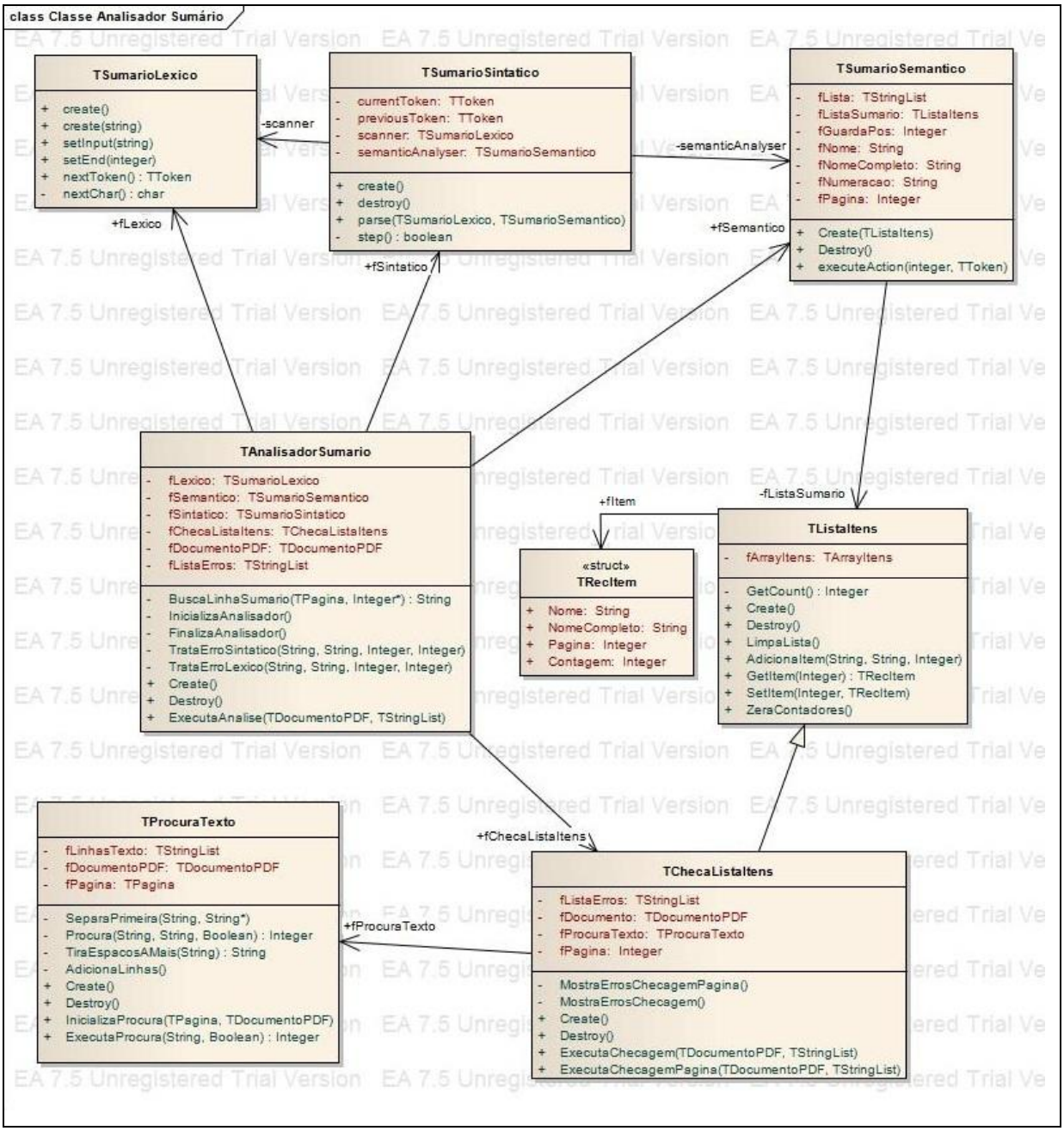


Figura 16 - Diagrama da classe do pacote `AnalisadorSumario`

3.2.2.7 Pacote `AnalisadorTabelas`

Neste diagrama de classes (Figura 17) estão representadas as classes do pacote `AnalisadorTabelas` que fazem a análise da lista de tabelas e da lista de ilustrações.

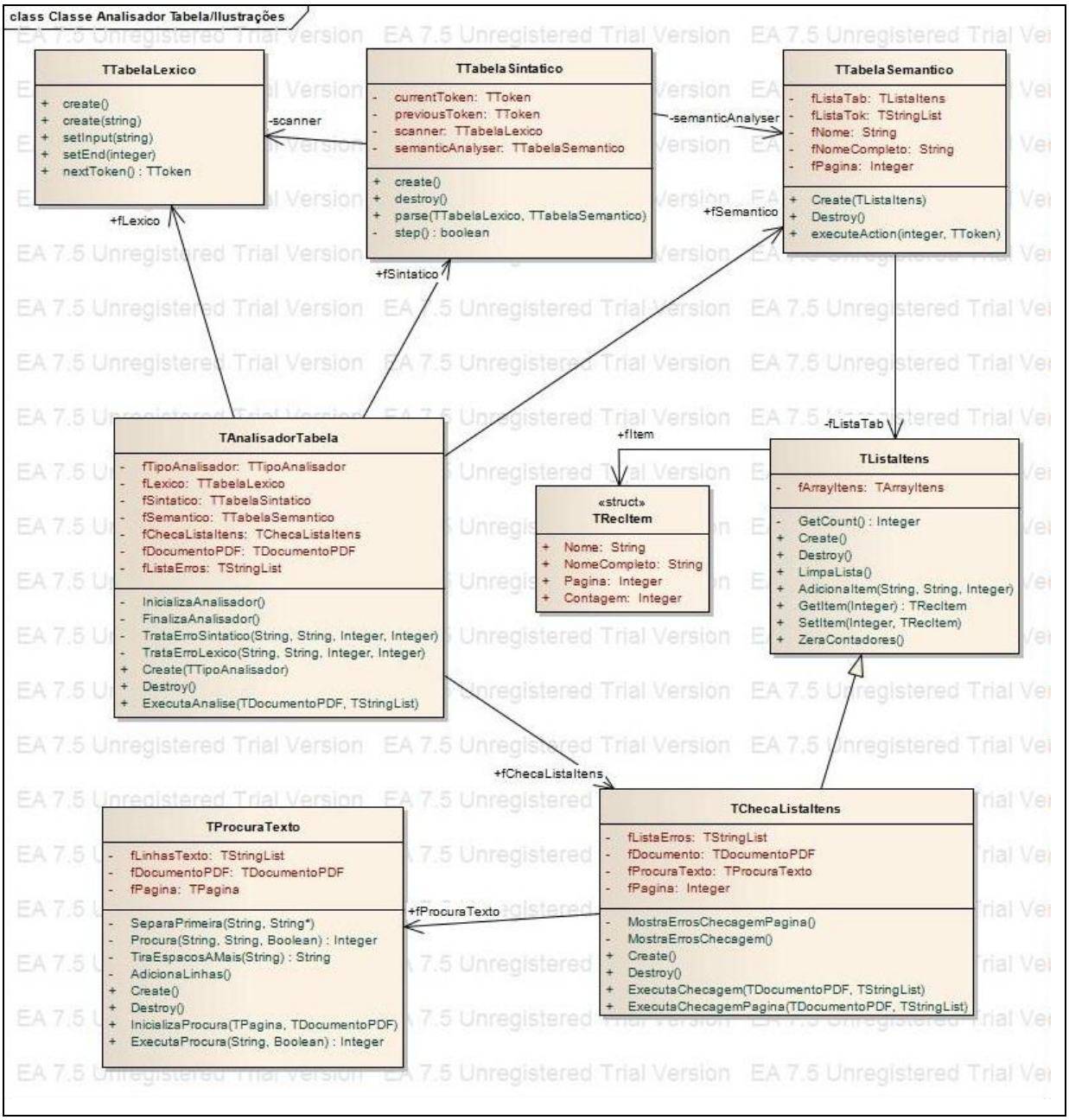


Figura 17 - Diagrama de classe do pacote `AnalisadorTabelas`

A classe `TAnalisadorTabela` é usada para fazer as análises da lista de tabelas e também é usada para analisar a lista de ilustrações. Esta classe de analisador é responsável por fazer uma análise sintática de cada item da lista de tabelas e da lista de ilustrações, analisando a construção deste item da lista. As regras para construção deste item podem ser vistas no documento de Silva (2012a). Depois da análise, o analisador separa cada item para fazer a checagem de páginas no texto.

`TTabelaSintatico` é uma classe gerada automaticamente pela ferramenta GALS. Esta classe é gerada baseando-se em uma BNF, construída para fazer a checagem dos itens desta lista de tabelas e da lista de ilustrações. Esta BNF pode ser vista no Quadro 18, no apêndice

A. Essa classe faz a checagem sintática e retorna erro caso o item da lista não esteja corretamente definido para satisfazer a BNF especificada.

`TTabelaLexico` é outra classe gerada pela ferramenta GALS, que ajuda o analisador sintático a realizar a sua análise, separando os *tokens* para o uso do analisador sintático.

`TTabelaSemantico` é gerada pelo GALS e é usado pelo analisador sintático para executar as ações semânticas definidas na BNF construída no GALS, e que para este protótipo serão apenas usadas para remover cada item da lista de ilustrações e lista de tabelas separadamente e adicionados em uma lista.

As classes `TListaItens`, `TRecItem` e `TChecaListaItens` já foi devidamente especificada no item 3.2.2.4.

A classe `TProcuraTexto` já foi devidamente especificada no item 3.2.2.3.

3.2.3 Diagrama de seqüência

A Figura 18 apresenta um diagrama de seqüência do caso de uso `UC01 – analisa documento`.

O usuário inicia o processo de análise do documento configurando algumas informações básicas para o funcionamento correto do analisador. Depois o mesmo usuário seleciona o arquivo a ser analisado e com a rotina `ExecutaTodasChecagens(NomeArq)` ele inicia a análise do seu documento no `AnalisadorDocumentoPDF`.

O `AnalisadorDocumentoPDF` recebe o nome do arquivo junto com a rotina anterior e faz a carga do documento na rotina `AbreCarregaArquivoPDF(NomeArq)`.

Depois da carga o mesmo `AnalisadorDocumentoPDF` começa a chamar as rotinas de classificação e análise do documento. Primeiro mandando o documento sem classificação para a rotina `ClassificaDocumento(Documento, ListaErros)` de `ClassificaPaginasDoc`. Nesta rotina o `AnalisadorDocumentoPDF` manda o documento carregado em memória e uma lista de erros, caso existirem erros. Estes erros podem ser adicionados nesta lista de erros.

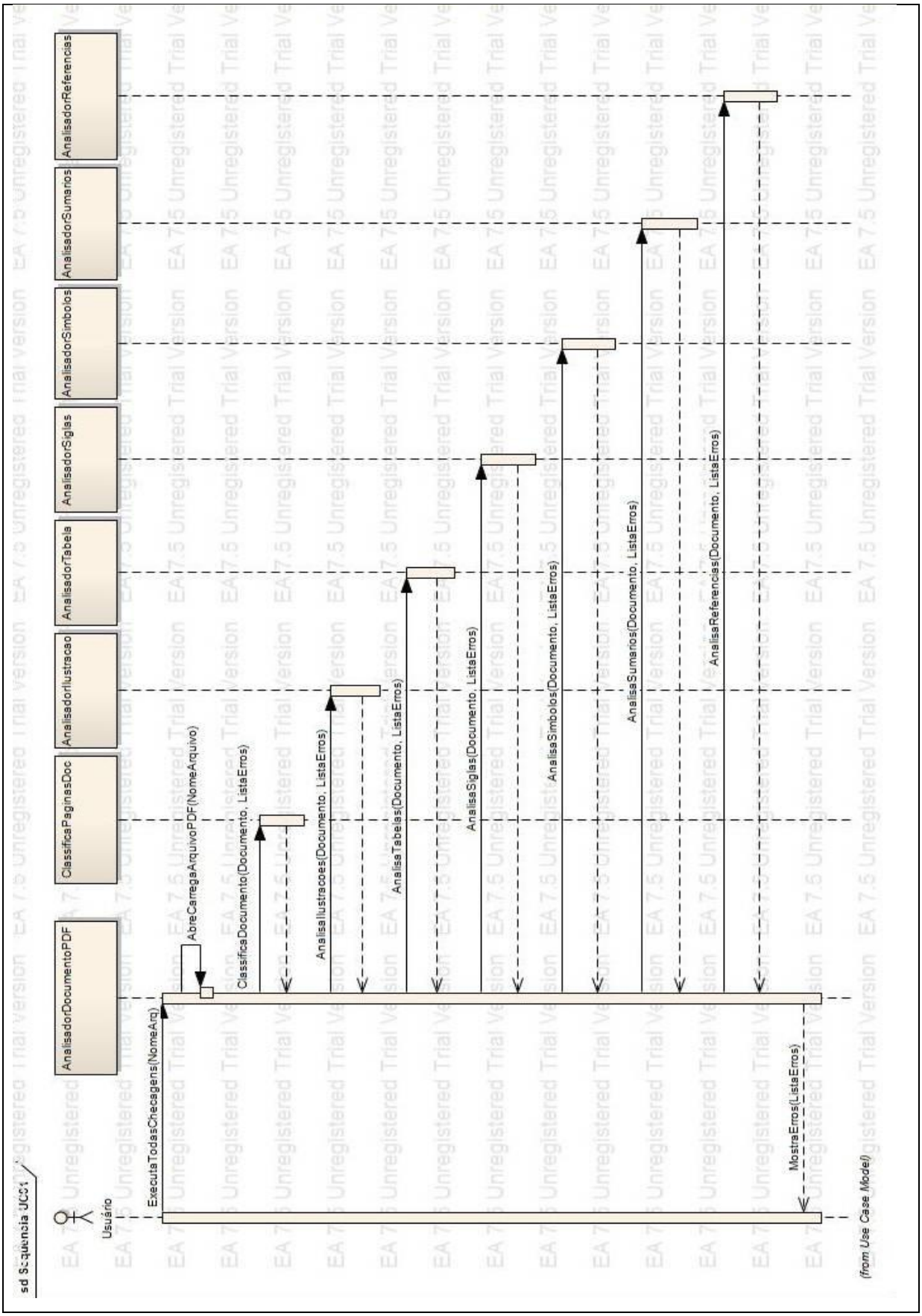


Figura 18 - Diagrama de seqüência `UC01 – analisa documento`

Acabada a rotina de classificação, e com o documento com suas páginas classificadas,

o AnalisadorDocumentoPDF manda para AnalisadorIlustracao na rotina AnalisaIlustracoes(Documento, ListaErros), onde ele vai procurar as páginas classificadas, como lista de ilustrações, e irá fazer a análise item por item e depois fazer a procura no documento pelos itens encontrados na lista de ilustrações. Os erros são adicionados na lista de erros que depois volta para o AnalisadorDocumentoPDF.

Depois de efetuada a análise das ilustrações, o AnalisadorDocumentoPDF manda para AnalisadorTabela na rotina AnalisaTabelas(Documento, ListaErros), onde este analisador procurar a página classificada como lista de tabelas e vai fazer a análise de todos os itens desta lista e vai depois procurar no documento para verificar se a tabela está na página correta, e adicionando os erros na lista de erros.

Quando acabada a análise das tabelas, o AnalisadorDocumentoPDF manda para o AnalisadorSiglas o documento e a lista de erros, utilizando a rotina AnalisaSiglas(Documento, ListaErros) para que o AnalisadorSiglas faça a análise e a checagem das siglas no documento, verificando se cada item da lista de siglas foi pelo menos mencionado uma vez no documento inteiro, e adicionando os erros na lista de erros.

Depois AnalisadorDocumentoPDF manda para o AnalisadorSimbolos usando a rotina AnalisaSimbolos(Documento, ListaErros) para que o mesmo faça a análise dos símbolos do documento, pegando e analisando os símbolos da lista de símbolos e depois faz a procura pelo símbolo no documento inteiro. Este símbolo deve ser citado pelo menos uma vez no documento, caso não tenha nenhuma menção é adicionado um erro na lista de erros.

Depois disso, o AnalisadorDocumentoPDF vai mandar para o AnalisadorSumarios, chamando a rotina AnalisaSumarios(Documento, ListaErros), para que o analisador possa fazer a checagem dos itens de sumário. Verifica se os itens estão nas páginas especificadas no sumário. Qualquer erro vai ser adicionado na lista de erros.

Ao término da análise do sumário, o AnalisadorDocumentoPDF chama a última análise que é o AnalisadorReferencias, usando a rotina AnalisaReferencias(Documento, ListaErros), onde ele vai fazer a análise das referências do documento, verificando a existência das citações com as referências no documento. Os erros serão adicionados na lista de erros.

Por último, quando todas as análises foram feitas, o AnalisadorDocumentoPDF mostra os erros que aconteceram para o usuário, caso existam. Caso os erros não existam no documento analisado o protótipo mostra apenas quais foram as análises feitas por ele neste documento.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas informações sobre as técnicas e ferramentas utilizadas para a implementação do protótipo, bem como o processo de implementação, com trechos do código fonte para um melhor entendimento.

3.3.1 Técnicas e ferramentas utilizadas

O protótipo foi desenvolvido na linguagem *Object* Pascal, utilizando-se da ferramenta Borland Delphi 7. Para consultas e dúvidas com a ferramenta utilizada foi usado o livro de Cantu (2003).

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

a) PDFtoolkit VCL v4 *Trial*: esta biblioteca foi obtida a partir do *site* da Gnostice (2012a) e utilizada para fazer a leitura do arquivo em formato PDF e para extrair os dados necessários para fazer as checagens no protótipo;
b) GALS: esta ferramenta foi utilizada para a geração das classes que serão responsáveis em fazer a análise léxica e sintática dos itens que vão ser analisados neste protótipo, usando-se das BNFs construídas para gerar estas classes.

Para facilitar o entendimento da implementação deste protótipo, será descrito aqui o processo de implementação dividido em partes. Uma parte para cada tipo de análise que é feita pelo protótipo para no final mostrar os resultados ao usuário.

3.3.1.1 Carga do documento PDF

Para conseguir fazer a carga do documento PDF original selecionado pelo usuário, fez-se necessário a procura de um componente para que faça esta leitura do documento em formato PDF e extraia as informações necessárias para o funcionamento do protótipo, tendo em vista que a linguagem de programação escolhida para a implementação deste protótipo não possui uma biblioteca própria para fazer a extração de dados de um arquivo PDF.

Primeiro foi feito um estudo para converter o arquivo PDF selecionado para um

formato de arquivo de mais fácil acesso que seria os arquivos *eXtensible Markup Language* (XML). Mas na procura, não foi encontrado nenhum tipo de software que consiga converter um arquivo PDF para XML de maneira a manter todas as informações necessárias para que o protótipo funcione. Então esta opção foi descartada.

Já que a opção de converter o arquivo PDF em um arquivo XML foi descartada foi feita uma procura por uma biblioteca para a linguagem *Object* Pascal que tivesse os recursos para extrair os dados necessários. Foram encontradas e testadas as seguintes opções:

a) *PowerPDF*: esta ferramenta é *open source* mas não atendeu as expectativas, pois é uma ferramenta para criação e visualização de arquivos PDF;
b) Debenu *Quick PDF Library*: é uma ferramenta paga, que extrai imagens, texto e outras características do documento PDF, mas pelo fato de usar uma versão *trial* desta biblioteca, na extração do texto do documento, ela trazia o caractere "*" espalhado randomicamente no meio do texto, impossibilitando sua utilização para a implementação do protótipo;
c) PDFtoolkit VCL v4: esta ferramenta, também paga, também foi testada a sua versão *trial* de 30 dias e dentre todas as opções vistas é única que não coloca nenhum tipo de restrição em suas funções. A função que extrai o texto do documento funciona de maneira satisfatória para a implementação deste protótipo, por isso ela foi a biblioteca escolhida para o uso neste trabalho.

Depois de escolher uma biblioteca para extrair os dados do documento PDF, agora é preciso carregar este arquivo em PDF para uma estrutura criada em memória para que o protótipo possa acessar rapidamente e facilmente estes dados sem que tenha que ficar lendo do arquivo PDF no disco várias vezes.

A rotina `AbreCarregaArquivoPDF(csArquivo, cbGeraArquivoTexto)` está implementada na classe `TAnalisadorDocumentoPDF` e tem esta função de abrir o documento PDF, extrair o texto de cada página e separar o estas informações em páginas e linhas dentro da classe `TDocumentoPDF`, que foi previamente demonstrada na Figura 11.

Na Figura 19 pode ser vista a rotina que faz a carga do arquivo PDF utilizando-se da rotina `LoadFromFile()` e a separação das páginas e linhas, junto com a utilização da rotina `ExtractTextFormatted()` que trata de extrair o texto das páginas.

```
begin
  // cria a classe de carga do documento PDF
  loPDF := TgtPDFDocument.Create(nil);
  try
    // carrega arquivo selecionado pelo usuário
    loPDF.LoadFromFile(csArquivo);
    // verifica se a carga está OK
    if   loPDF.IsLoaded  then
         begin
           // seta a unidade de medida das páginas para milímetros
           loPDF.MeasurementUnit := muMM;
           // pega o total de paginas do documento
           liTotalPages          := loPDF.PageCount;

           for   liInd := 0  to  liTotalPages-1  do
                 begin

                 // mostra o progresso da carga para o usuário
                 if   Assigned(fOnMostraProgressoCargaArq)  then
                      fOnMostraProgressoCargaArq('Carregando páginas arquivo PDF... ', liInd+1, liTotalPages);

                   // cria nova página no objeto em memória documento,
                   // tipo de todas as páginas na carga é igual a tpOutras
                   loPagina := fPDFDoc.AdicionaPagina(liInd, tpOutras);

                   // extrai o texto da página
                   loListW := loPDF.ExtractTextFormatted(liInd+1);
                   for   liIndLines := 7  to  loListW.Count-1  do
                         begin
                           lsLinhaAdd := Substitui(loListW.Strings[liIndLines]);
                           // adiciona linhas na página corrente do documento em memória
                           loPagina.AdicionaLinha(0, 0, 0, 0, lsLinhaAdd);
                         end;

                   // calcula posições das linhas nas páginas
                   Posicoes(loPDF, liInd);
                 end;

           // gera arquivos no formato TXT
           if   cbGeraArquivoTexto  then
                GeraTextos;
         end;

  finally
    // libera a classe de carga do documento PDF
    FreeAndNil(loPDF);
  end;
end;
```

Figura 19 - Rotina `AbreCarregaArquivoPDF`

Na carga do arquivo PDF para o documento em memória que foi utilizado pelo protótipo para fazer as análises, a rotina de carga carrega todas as páginas com o tipo da página `tpOutras`. Junto a rotina de abertura e carga do documento, existe a opção de gerar o arquivo carregado em formato texto para que se possa ver como a biblioteca PDFtoolkit conseguiu extrair o texto do arquivo em PDF selecionado.

3.3.1.2 Classificação do documento

Depois de carregado, o documento em memória possui todas as suas páginas com a propriedade `fTipoPagina` com o valor `tpOutras`, pois ainda não está classificado. A rotina de classificação do documento vai fazer a classificação destas páginas.

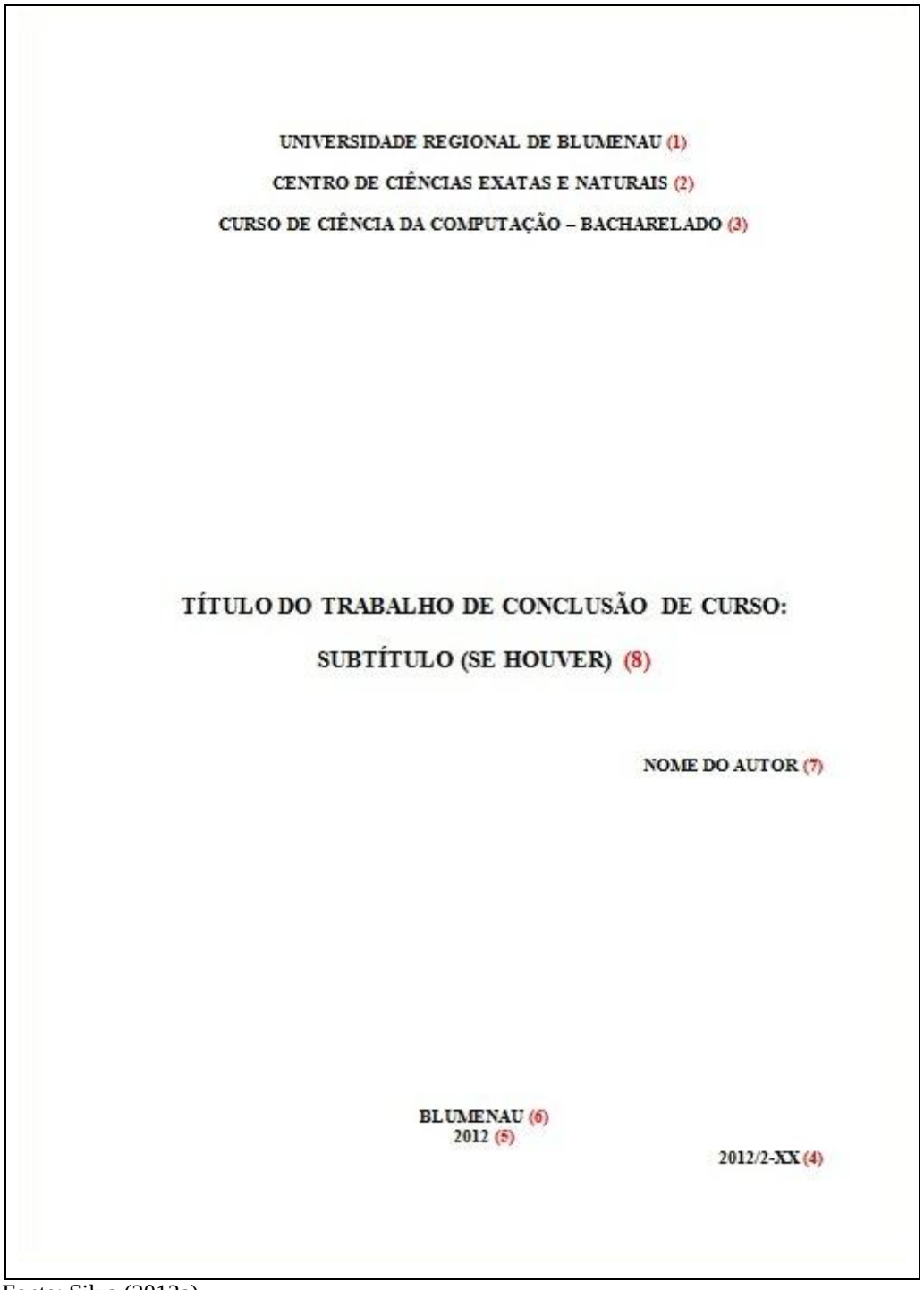
O Quadro 6 abaixo demonstra os elementos da monografia, segundo Associação Brasileira de Normas Técnicas (2005), e também demonstra o tipo de página atribuído, respectivamente, para cada elemento.

ESTRUTURA	**ELEMENTO**	**TTIPOPAGINA**
Pré-textuais	Capa (obrigatório)	tpCapa
	Lombada (obrigatório)	(ignorado)
	Folha de rosto (obrigatório)	tpRosto
	Folha de aprovação (obrigatório)	tpAprovacao
	Dedicatória(s) (opcional)	tpDedicatoria
	Agradecimento(s) (opcional)	tpAgradecimentos
	Epígrafe (opcional)	tpEpigrafe
	Resumo na língua vernácula (obrigatório)	tpResumo
	Resumo em língua inglesa (obrigatório)	tpAbstract
	Lista de ilustrações (opcional)	tpIlustracoes
	Lista de tabelas (opcional)	tpTabelas
	Lista de abreviaturas (opcional)	tpSiglas
	Lista de símbolos (opcional)	tpSimbolos
	Sumário (obrigatório)	tpSumario
Textuais	Introdução	tpTexto
	Fundamentação Teórica do Trabalho	tpTexto
	Desenvolvimento	tpTexto
	Conclusão	tpTexto
Pós-Textuais	Referências (obrigatório)	tpReferencias
	Glossário (opcional)	tpTexto
	Apêndice(s) (opcional)	tpTexto
	Anexo(s) (opcional)	tpTexto
	Índice(s) (opcional)	tpTexto

Quadro 6 - Elementos da monografia e seus respectivos tipos

A classificação das páginas deste documento é feita através da análise da ordem em que as páginas são encontradas e também fazendo uma comparação de padrões utilizando como modelo o Silva (2012a). O elemento lombada, encontrado no Quadro 6 é ignorado pelo classificador de documento, pois no documento descrito em Silva (2012a) não consta esta página.

A Figura 20 demonstra uma página de capa de um TCC, com os seus componentes principais, conforme especificado em Silva (2012a). Esta página tem que ser a primeira página do documento.



Fonte: Silva (2012a).

Figura 20 - Estrutura da página de capa

Para fazer a classificação da página da capa são identificados os seguintes componentes:

a) o nome da universidade em que o TCC pertence (Figura 20(1)). Este nome é informado pelo usuário nas configurações iniciais do analisador, junto com o nome do centro e o nome do curso;
b) o centro do curso em que o aluno está (Figura 20 (2));

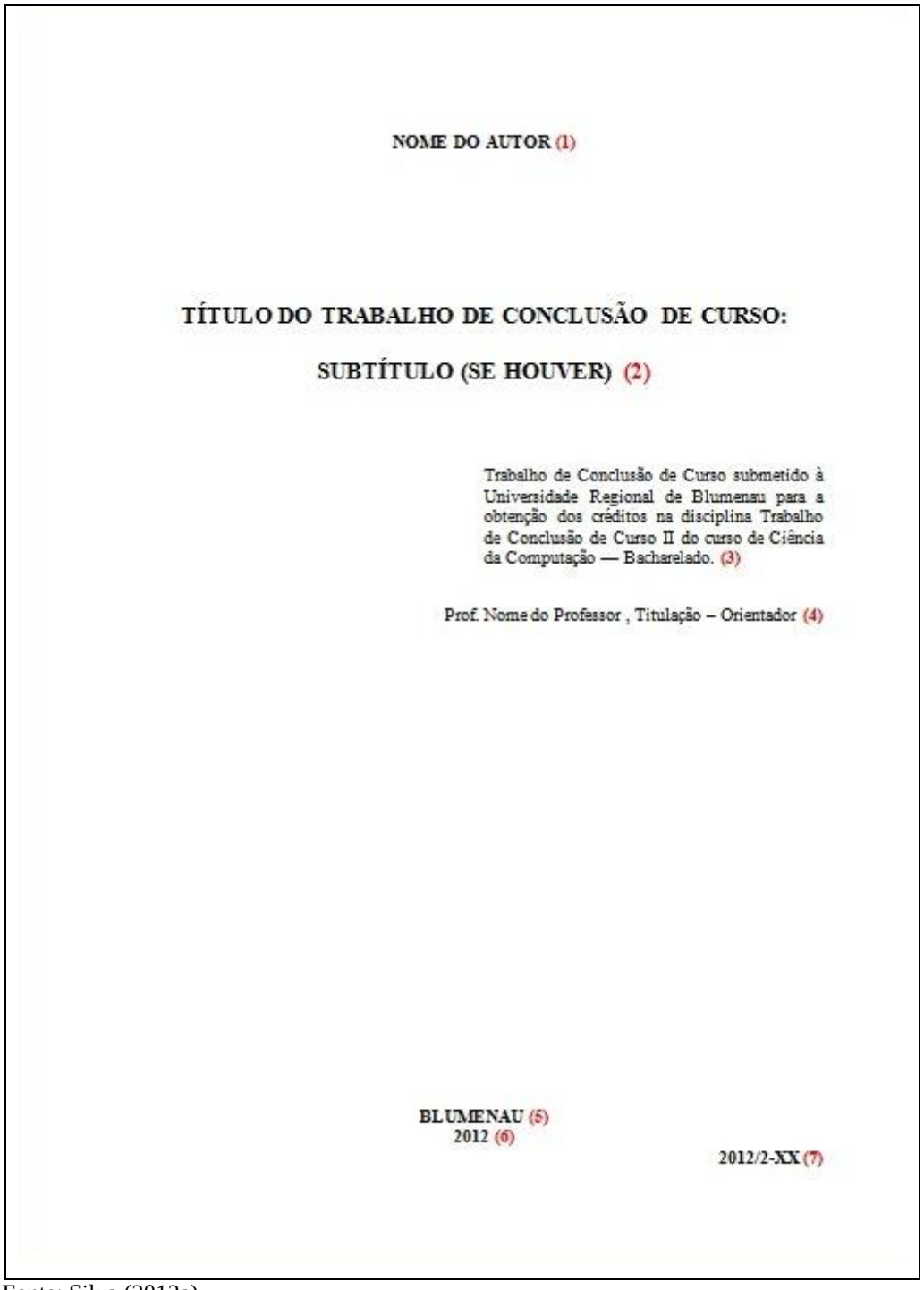
c) nome do curso em que o aluno está (Figura 20 (3));
d) é identificado no final da página o ano e o número do aluno (Figura 20 (4)) no formato em que se apresenta na Figura 20;
e) é identificado novamente o ano (Figura 20 (5));
f) é identificado o nome da cidade (Figura 20 (6));
g) depois de identificar a cidade é identificado o nome do autor (Figura 20 (7));
h) por último, tudo o que fica entre o nome do autor e o curso do autor, é considerado o título da obra (Figura 20 (8)).

Somente depois de encontrar todos estes padrões na primeira página, é que a página é classificada como `tpCapa`. Caso alguma destas características não seja corretamente identificada, o classificador retorna erro para o usuário, pois a capa é uma página obrigatória no TCC.

Na Figura 21 é demonstrada uma página de folha de rosto. Nela estão destacadas as suas principais características. A página de folha de rosto é a segunda folha do TCC. Nela é preciso a identificação dos seguintes padrões:

a) nome do autor (Figura 21(1)), que tem que ser o mesmo descrito na página da capa;
b) título da obra (Figura 21(2)), que também tem que ser o mesmo da página de capa;
c) texto da folha de rosto (Figura 21(3));
d) nome do professor (Figura 21(4)) que orientou o aluno;
e) nome da cidade (Figura 21(5)), ano (Figura 21(6)) e o ano mais a numeração do aluno (Figura 21(7)). Esses últimos tem que estar coerentes com a página de capa.

Depois de identificar todas estas características na segunda folha do TCC ela é classificada como `tpRosto`. A folha de rosto também é uma página obrigatória, portanto, caso não seja corretamente classificada, retorna erro para o usuário.



NOME DO AUTOR (1)

TÍTULO DO TRABALHO DE CONCLUSÃO DE CURSO:

SUBTÍTULO (SE HOUVER) (2)

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. (3)

Prof. Nome do Professor , Titulação – Orientador (4)

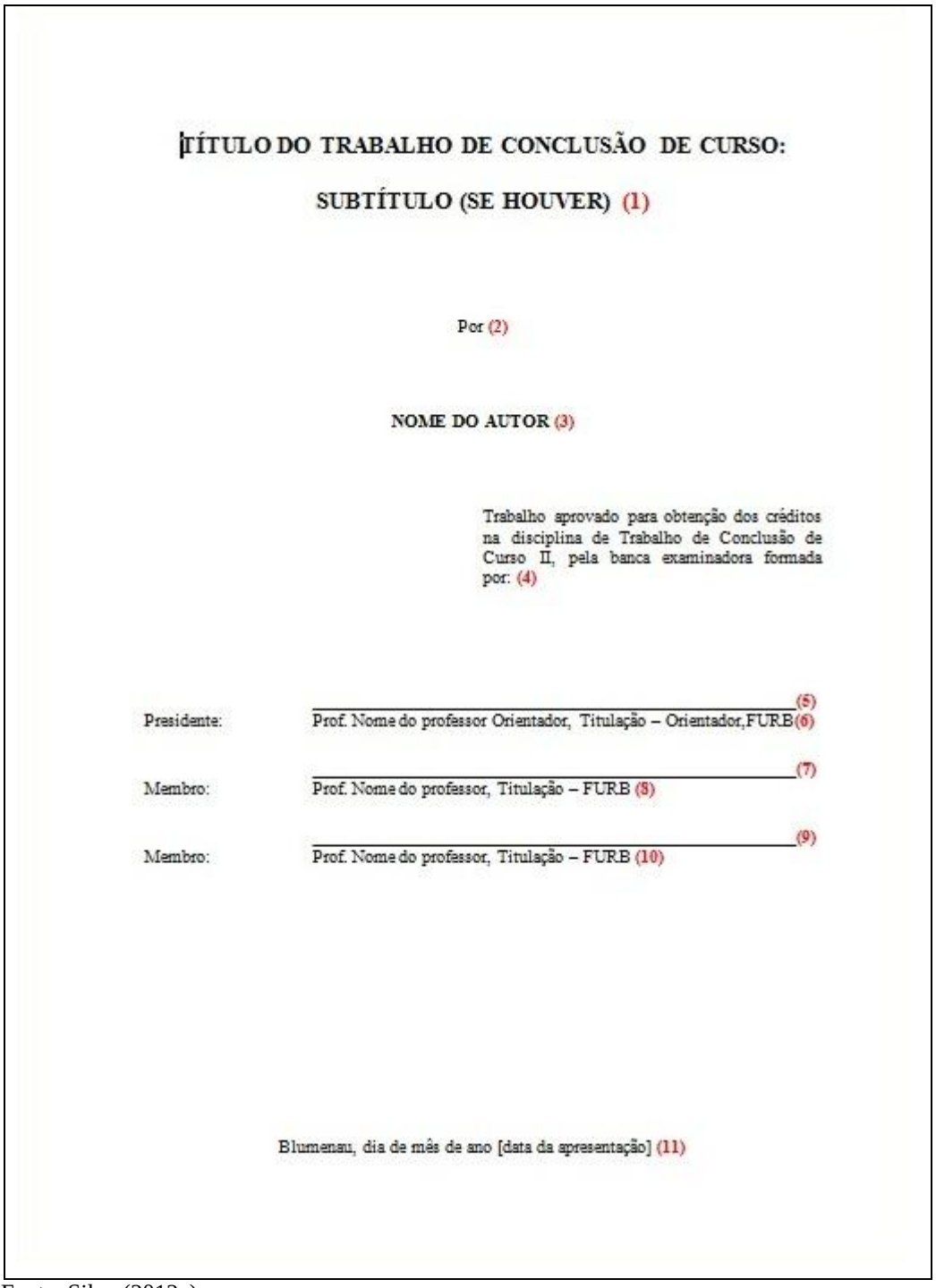
BLUMENAU (5)
2012 (6)

2012/2-XX (7)

Fonte: Silva (2012a).

Figura 21 - Estrutura da página de folha de rosto

Depois da página da folha de rosto tem que vir a página de folha de aprovação. A Figura 22 demonstra a página de folha de aprovação e seus componentes.



TÍTULO DO TRABALHO DE CONCLUSÃO DE CURSO:

SUBTÍTULO (SE HOUVER) (1)

Por (2)

NOME DO AUTOR (3)

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por: (4)

Presidente: _____(5)
Prof. Nome do professor Orientador, Titulação – Orientador,FURB(6)

Membro: _____(7)
Prof. Nome do professor, Titulação – FURB (8)

Membro: _____(9)
Prof. Nome do professor, Titulação – FURB (10)

Blumenau, dia de mês de ano [data da apresentação] (11)

Fonte: Silva (2012a).

Figura 22 - Estrutura da página de folha de aprovação

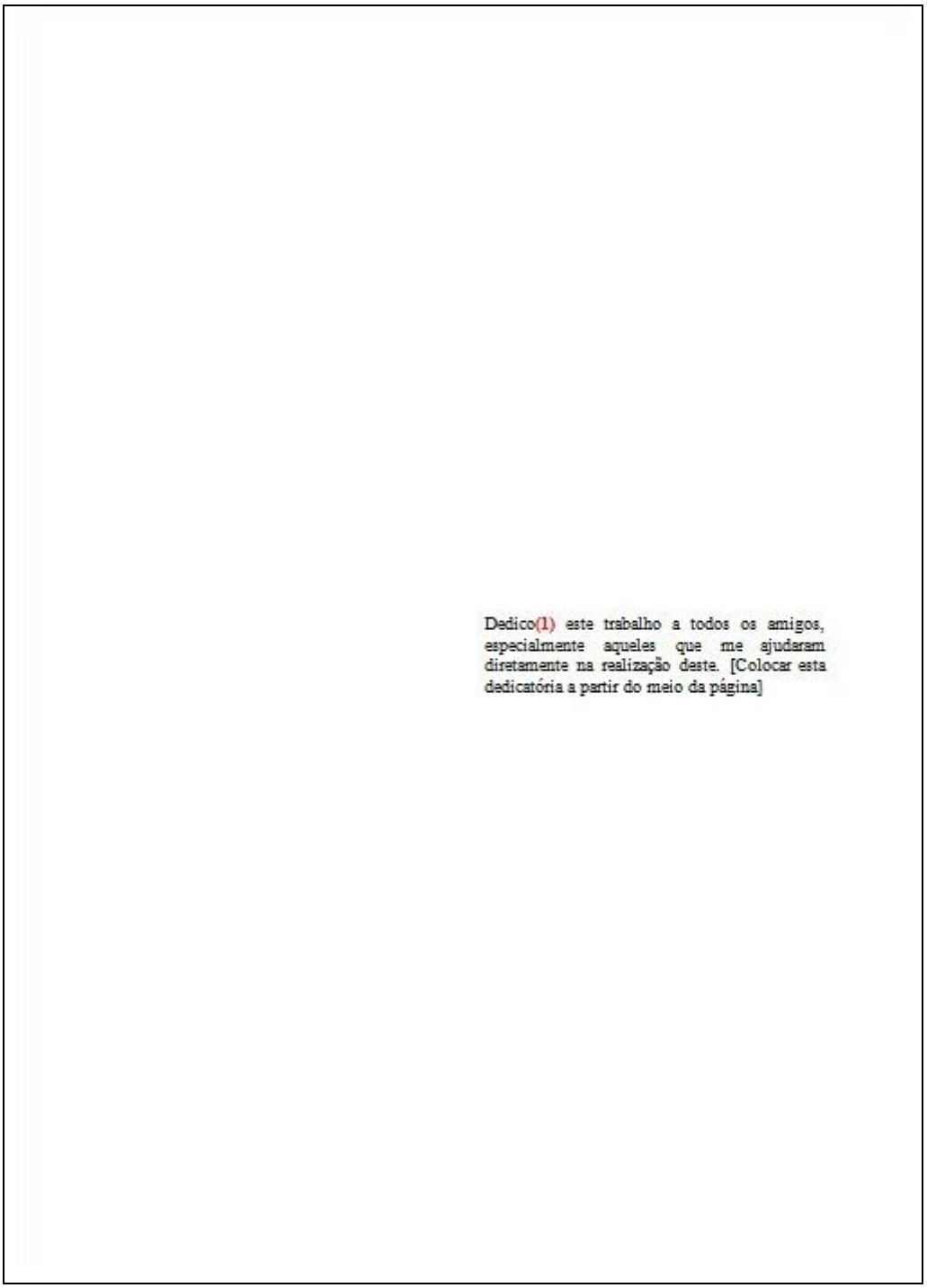
Para que a classificação da folha de aprovação seja positiva é preciso identificar as seguintes características:

a) o título da obra (Figura 22(1)), que deve estar coerente com a capa;
b) a palavra “Por” (Figura 22(2));
c) o nome do autor (Figura 22(3)), que tem de estar coerente com a capa;
d) o texto da folha de aprovação (Figura 22(4));

e) um conjunto de "_" formando uma linha (Figura 22(5)), como demonstra a Figura 22;
f) logo abaixo da linha, identifica-se a palavra "Presidente:", seguido pelo nome do professor (Figura 22(6));
g) um conjunto de "_" formando outra linha (Figura 22(7));
h) abaixo desta linha, é identificado a palavra "Membro:", seguido do nome do professor (Figura 22(8));
i) um conjunto de "_" formando a última linha desta folha (Figura 22(9));
j) abaixo, é identificado a palavra "Membro:", seguido do nome do professor (Figura 22(10));
k) por último tem a data (Figura 22(11)), no formato demonstrado na Figura 22.

Quando para esta página pode-se identificar todos estes padrões, ela é classificada como `tpRosto`. Caso não seja possível classificar a página como `tpRosto`, o classificador retorna um erro para o usuário, pois a página de folha de aprovação é obrigatória.

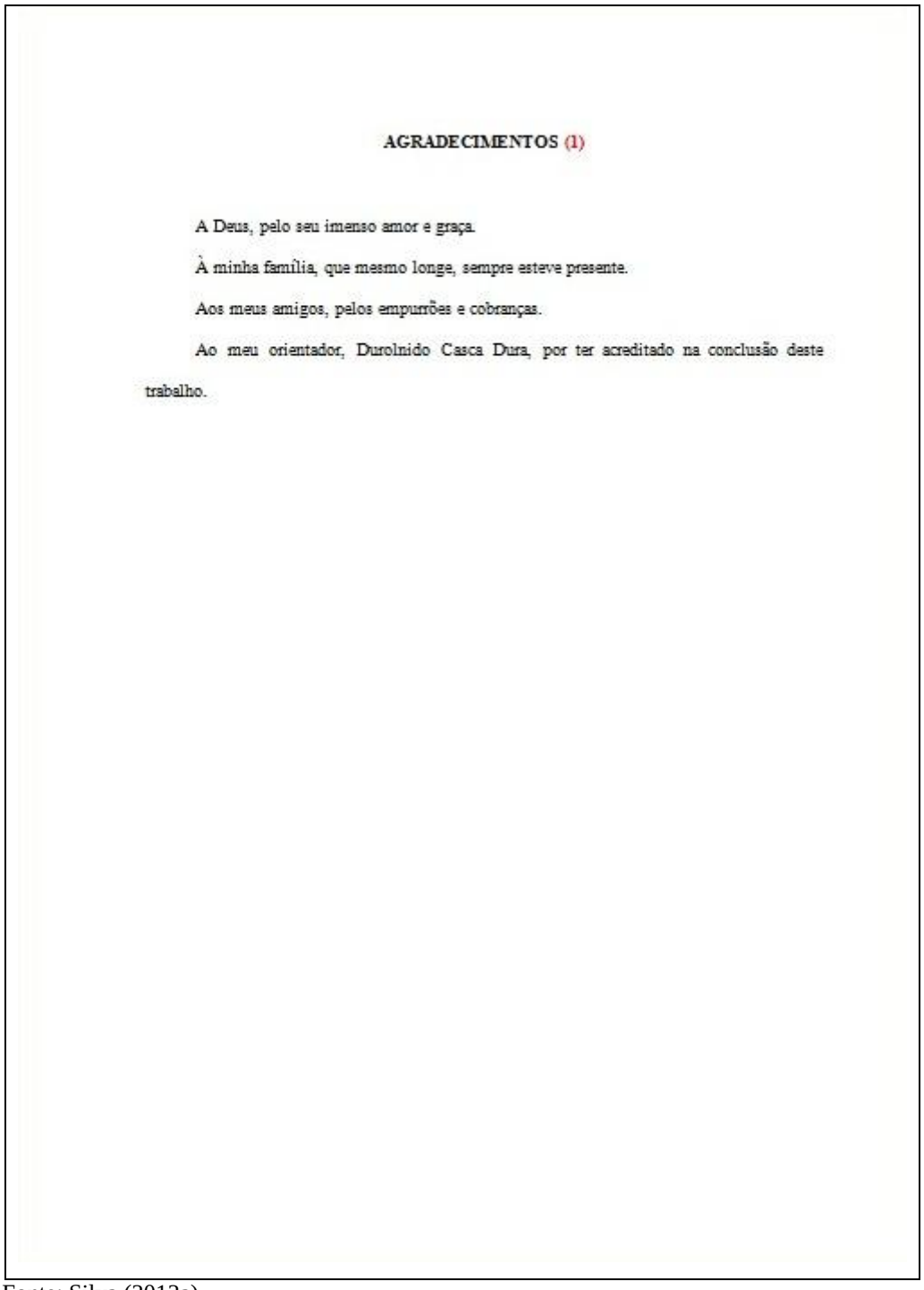
Depois da página de aprovação, pode surgir a página de dedicatória. Ela é uma página opcional, então nenhum erro é mostrado caso esta página não apareça no TCC. Esta página tem que aparecer depois da folha de rosto, se for o caso, e o único padrão de identificação desta página é a identificação da palavra "Dedico" (Figura 23(1)) no começo da frase que está escrita na página. Caso este padrão seja identificado, a página é classificada como `tpDedicatoria`. A Figura 23 mostra um exemplo de uma página de dedicatória.



Fonte: Silva (2012a).

Figura 23 - Estrutura da página de folha de dedicatória

A folha de agradecimentos é a próxima página a ser encontrada, caso ela esteja contida no TCC, pois é uma página opcional. Caso não apareça, não é mostrado nenhuma mensagem de erro. A Figura 24 mostra um exemplo de página de agradecimentos. O único padrão que tem para ser identificado nesta página é o aparecimento da palavra "AGRADECIMENTOS" (Figura 24(1)) no começo da página. Caso isso aconteça, esta página é classificada como `tpAgradecimento`.



AGRADECIMENTOS (1)

A Deus, pelo seu imenso amor e graça.

À minha família, que mesmo longe, sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

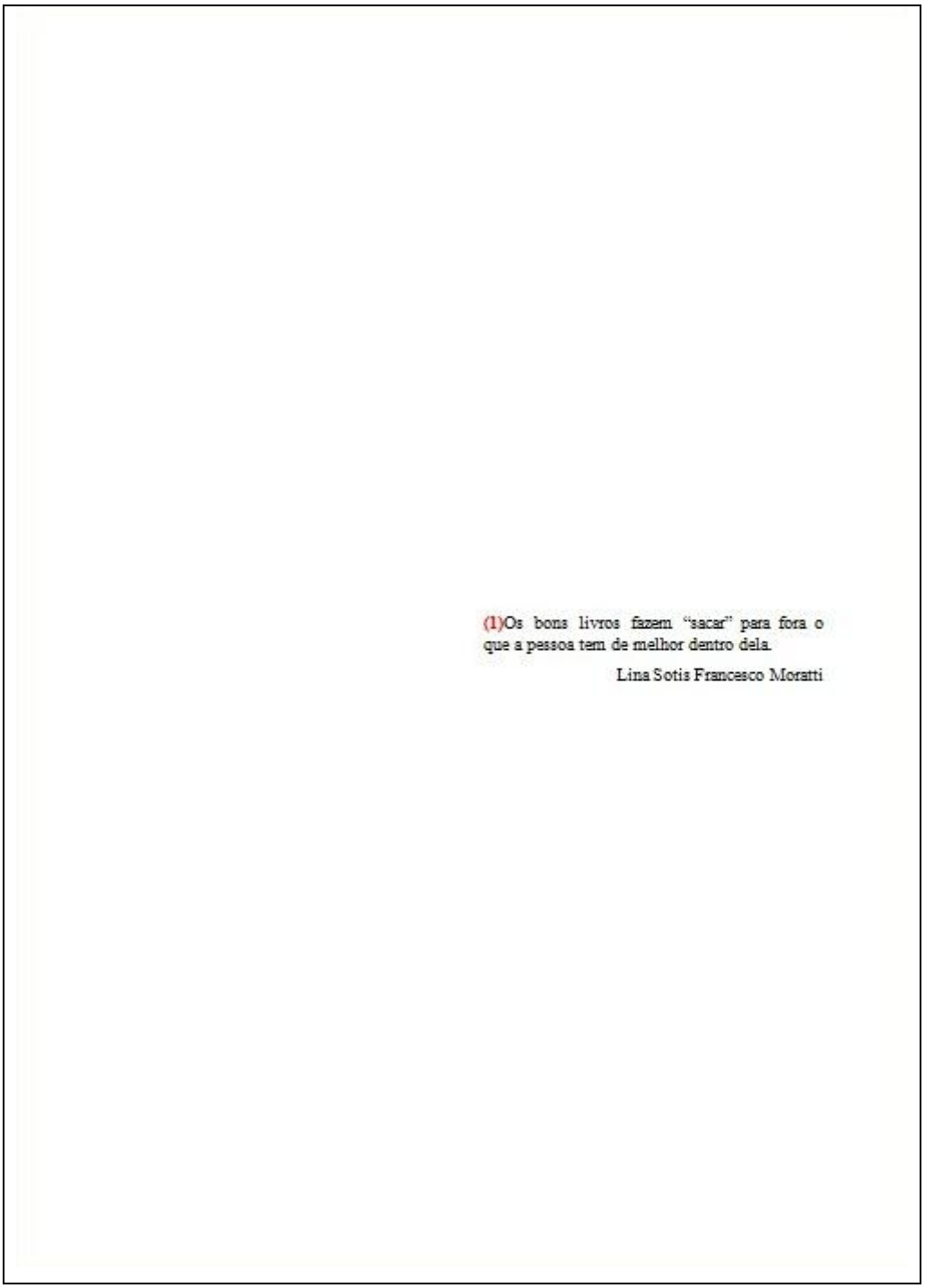
Ao meu orientador, Durolnido Casca Dura, por ter acreditado na conclusão deste trabalho.

Fonte: Silva (2012a).

Figura 24 - Estrutura da página de folha de agradecimentos

A Figura 25 demonstra uma página de epigrafe. Ela deve aparecer antes do sumário. Ela é uma página opcional também, por isso, caso não aparecer, nenhuma mensagem de erro é apontada. A página de epígrafe não é composta por nenhum tipo de palavra-chave. Esta página é composta apenas por uma frase ou pensamento, e o seu autor. Somente pode-se identificar a página de epigrafe se ela vem antes do sumário e depois da página de agradecimentos. Se a página conseguir contemplar estes requisitos, ela é classificada como

`tpEpigrafe`.



Fonte: Silva (2012a).

Figura 25 - Estrutura da página de folha de epígrafe

Na Figura 26 é mostrada um exemplo de uma página de resumo. São componentes da página de resumo:

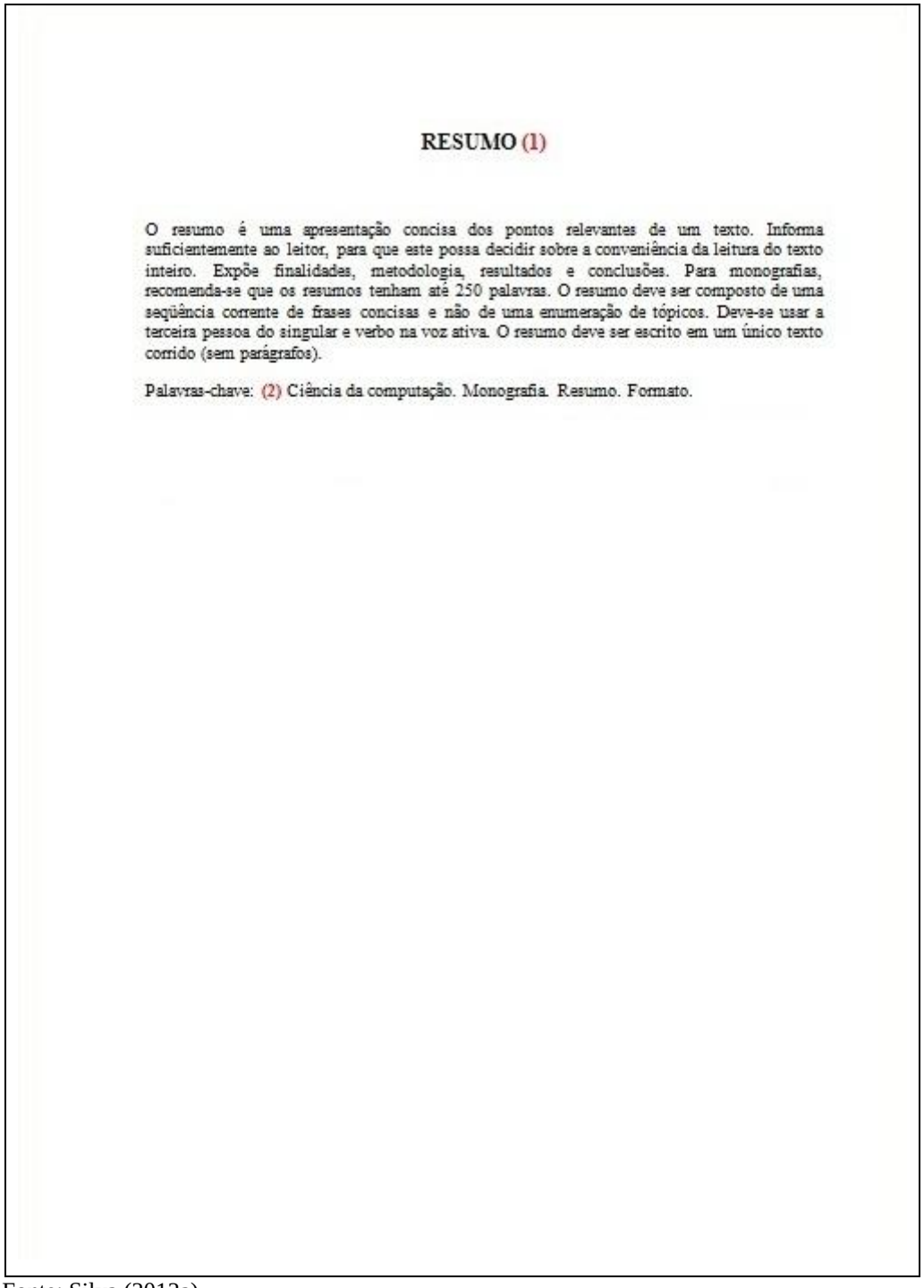
a) a palavra "RESUMO" (Figura 26(1));

b) o resumo propriamente dito, que é composto por algumas frases;

c) depois do resumo escrito propriamente dito, deve ser localizada as palavras

"Palavras-chave:" (Figura 26(2)) e depois algumas palavras definindo as palavras chaves.

Identificando todos estes padrões, a página é classificada como `tpResumo`. Esta página é obrigatória e se ela não for encontrada no TCC o classificador de documento mostra uma mensagem de erro para o usuário.



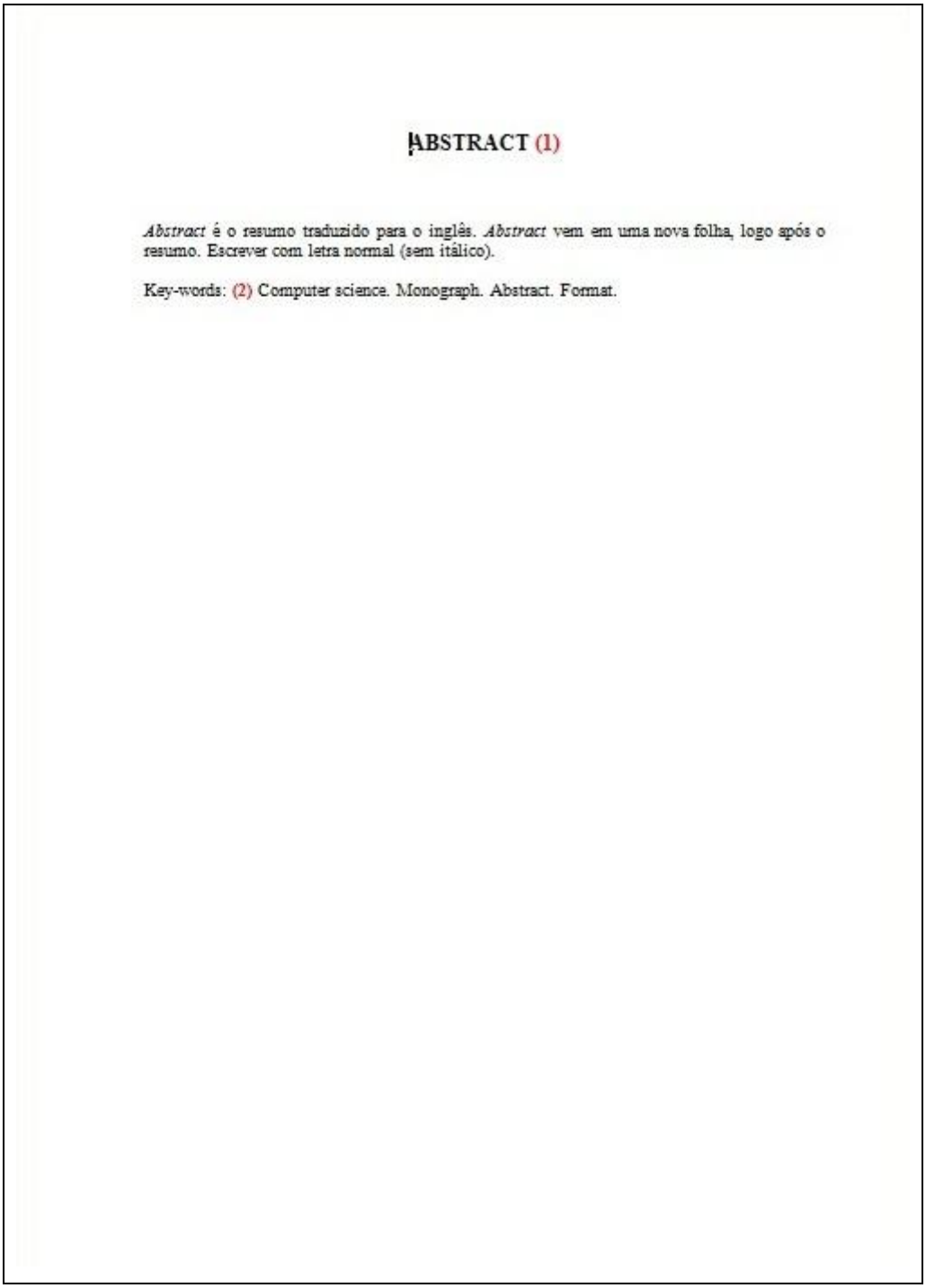
Fonte: Silva (2012a).

Figura 26 - Estrutura da página de folha de resumo

A Figura 27 mostra a estrutura de uma página de resumo em língua estrangeira que no

caso, esta página é escrita em inglês. Para que a página seja corretamente classificada ela deve conter os seguintes componentes:

a) a palavra “ABSTRACT” (Figura 27(1)) no começo da página;
b) a tradução da página de resumo, escrita agora em inglês;
c) depois do resumo escrito em inglês, deve-se encontrar as palavras “Key-words:” (Figura 27(2)) e depois algumas palavras para definição das *key-words*.



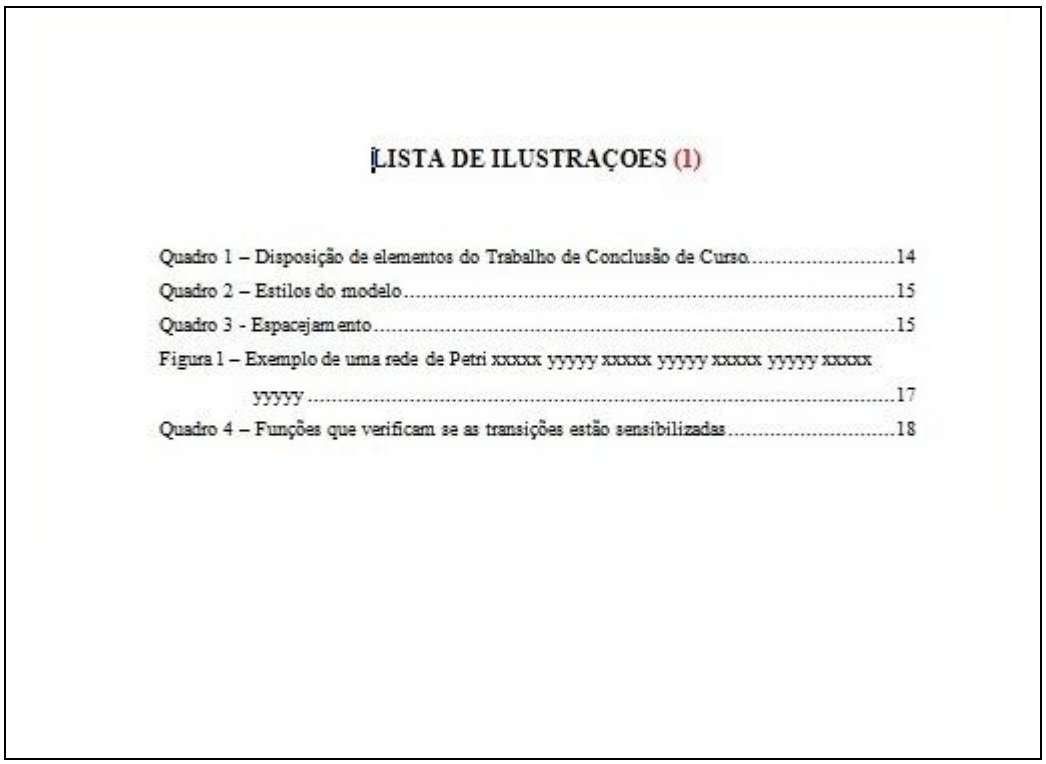
Fonte: Silva (2012a).

Figura 27 - Estrutura da página de folha de resumo em inglês

A Figura 28 mostra um exemplo de uma página de lista de ilustrações e seus componentes. Para que uma página de lista de ilustrações seja corretamente classificada deve-se identificar no começo da página a frase “LISTA DE ILUSTRAÇÕES” (Figura 28(1)).

Ao achar a frase acima descrita, o classificador classificará a página como `tpIlustrações`.

Os itens da lista de ilustrações serão analisados posteriormente pelo analisador de ilustrações.

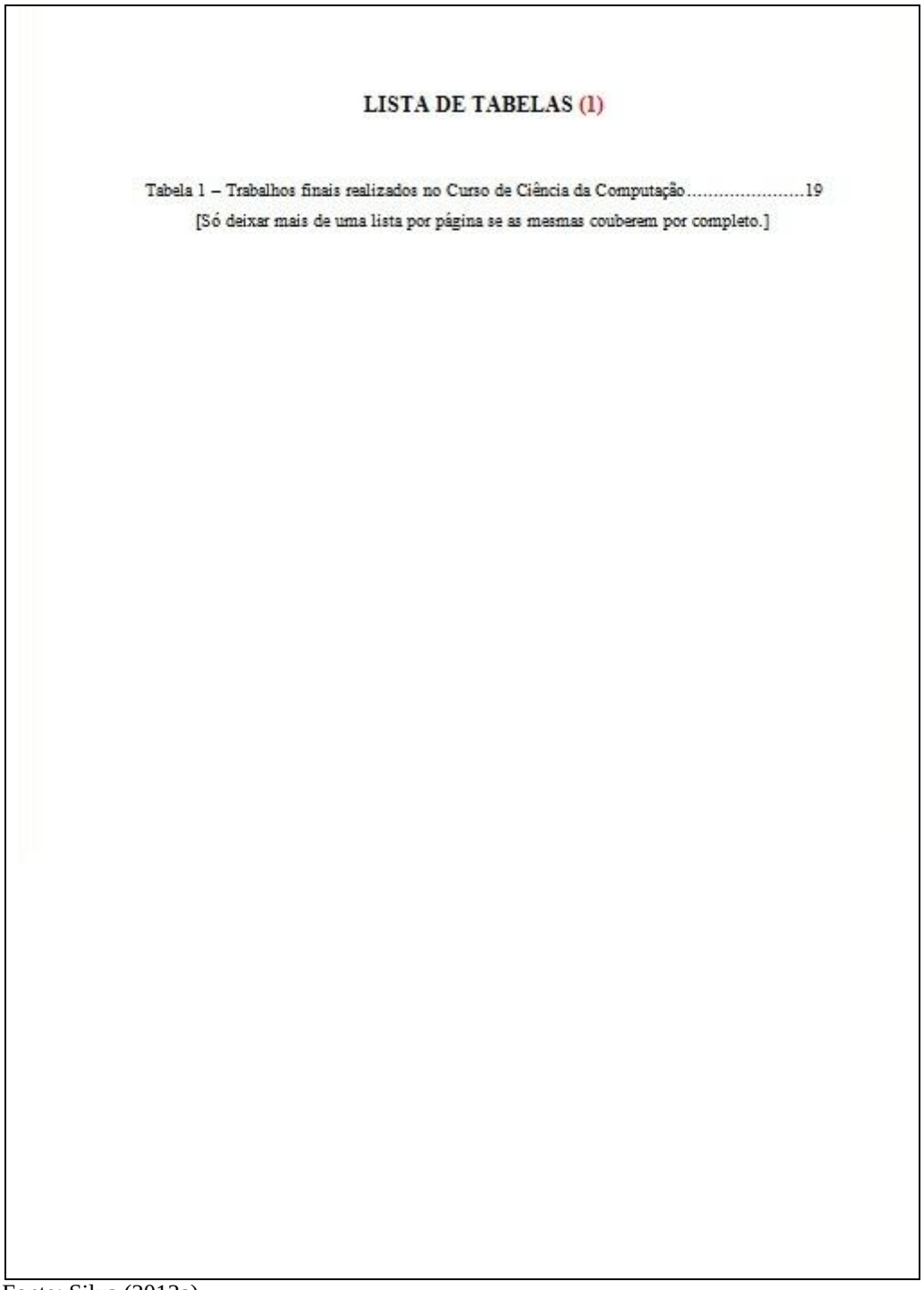


Fonte: Silva (2012a).

Figura 28 - Estrutura da página de folha de lista de ilustrações

A Figura 29 mostra uma página de lista de tabelas. Para a página ser classificada corretamente ela deve conter a seguinte frase “LISTA DE TABELAS” (Figura 29(1)) no inicio da página. Se a página for classificada com sucesso, ela é classificada como `tpTabelas`.

Os itens da lista de tabelas serão analisados mais tarde, pelo analisador de tabelas.

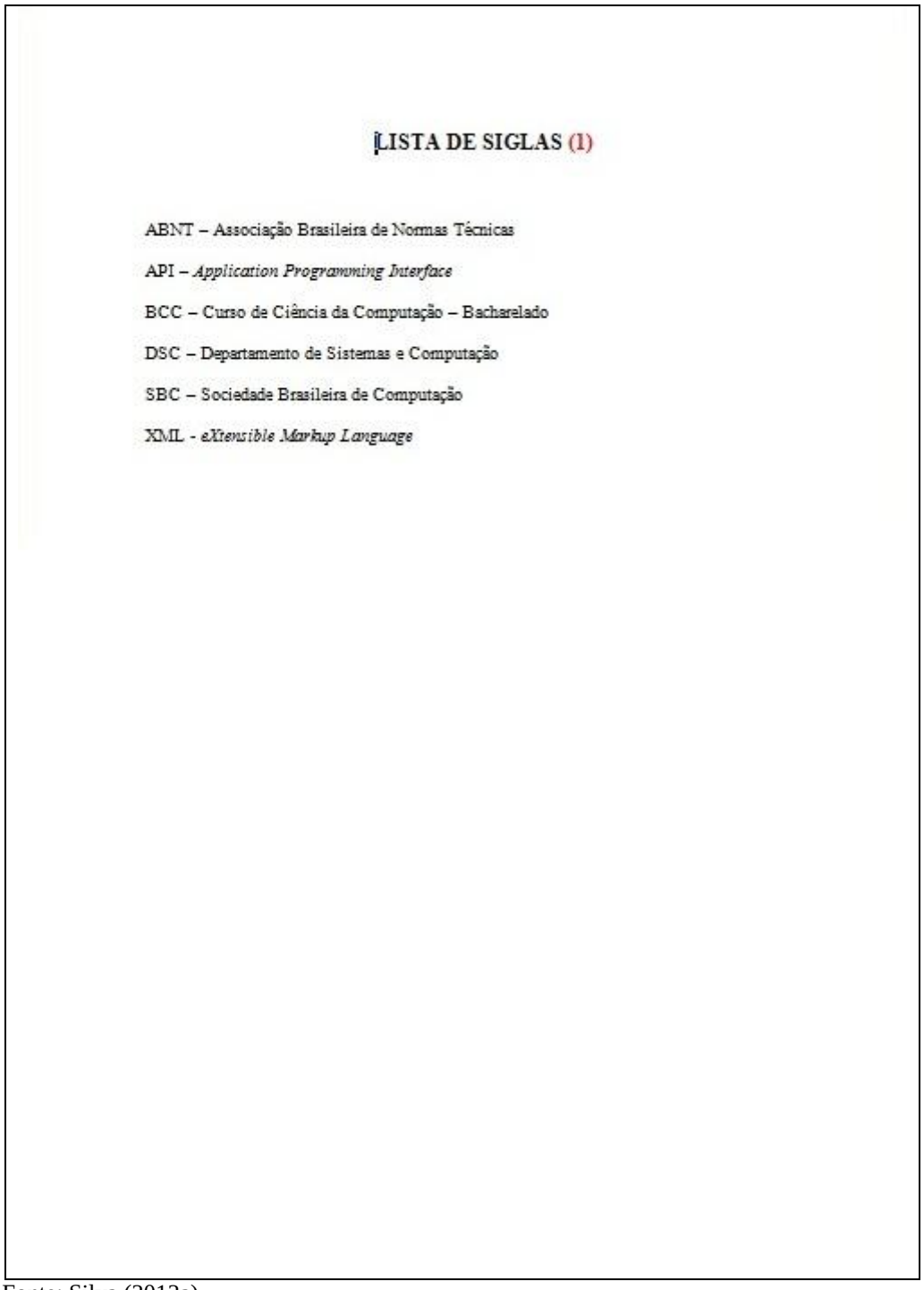


Fonte: Silva (2012a).

Figura 29 - Estrutura da página de folha de lista de tabelas

A Figura 30 mostra um exemplo de página de lista de siglas. Para que a página de lista de siglas seja identificada como uma página `tpSiglas` o classificador deve identificar a frase "LISTA DE SIGLAS" (Figura 30(1)) na primeira linha da página. Esta página não é uma página obrigatória.

Os itens encontrados na lista de siglas serão analisados posteriormente, pelo analisador de siglas.

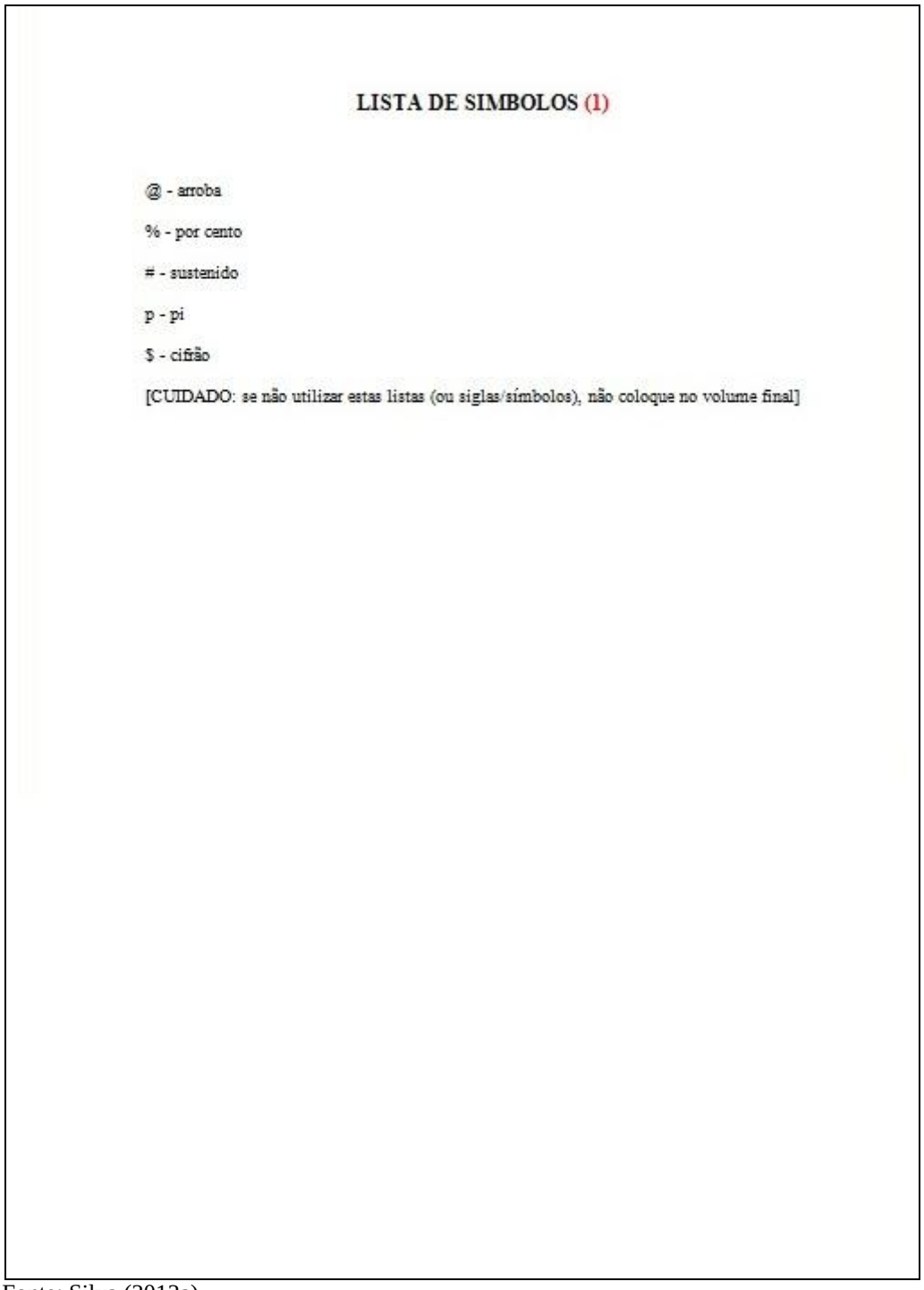


Fonte: Silva (2012a).

Figura 30 - Estrutura da página de folha de lista de siglas

Na Figura 31 é demonstrado um exemplo de página de lista de símbolos. Para esta página ser classificada como `tpSimbolos` o classificador deve encontrar a frase "LISTA DE SÍMBOLOS" (Figura 31(1)) na primeira linha da página. Esta página é uma página opcional para o TCC.

Os itens descritos na lista de símbolos serão analisados posteriormente pelo analisador de símbolos.



LISTA DE SIMBOLOS (1)

@ - arroba

% - por cento

- sustenido

p - pi

$ - cifrão

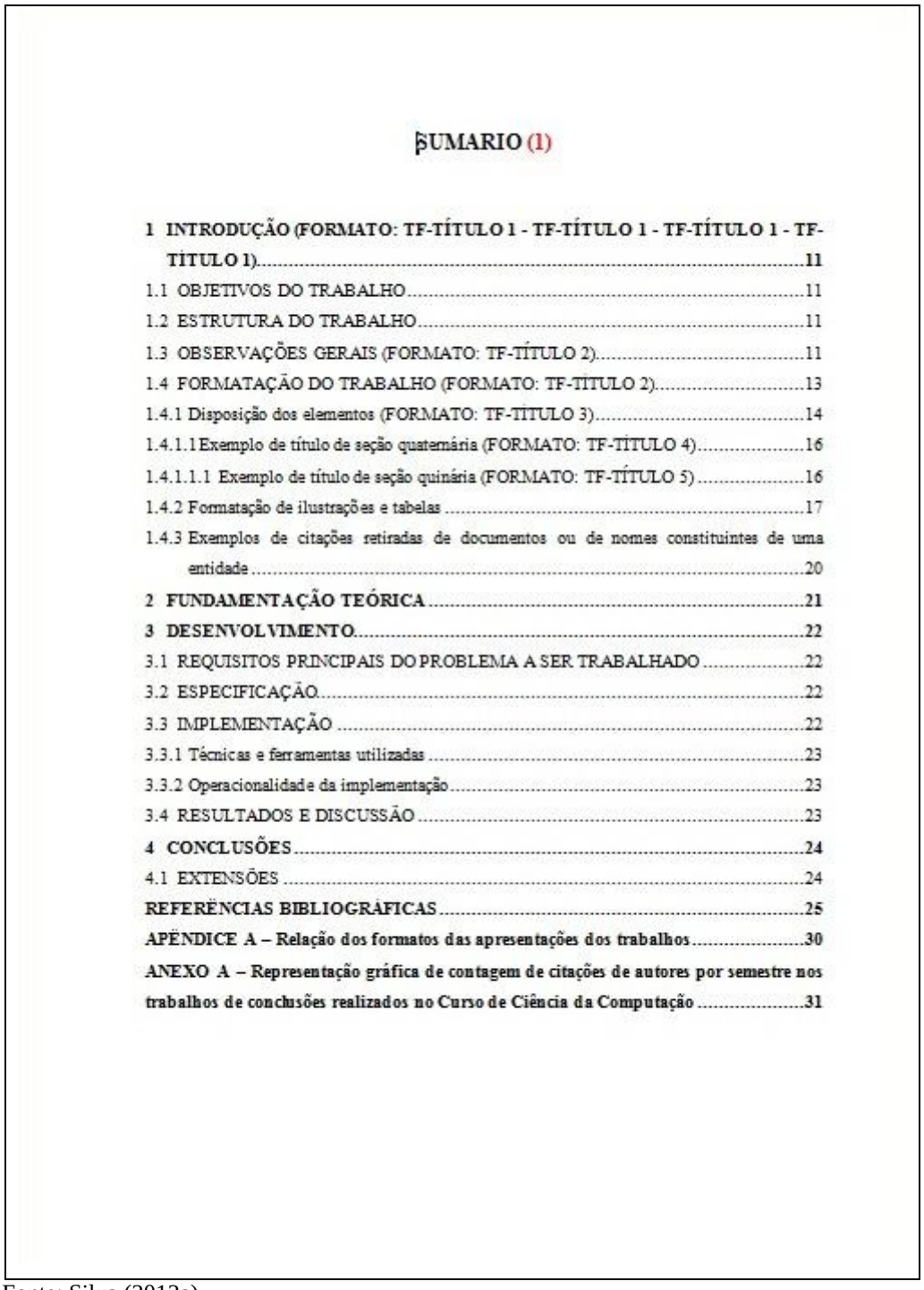
[CUIDADO: se não utilizar estas listas (ou siglas/símbolos), não coloque no volume final]

Fonte: Silva (2012a).

Figura 31 - Estrutura da página de folha de lista de símbolos

Na Figura 32 pode ser visto um exemplo de página de sumário. O classificador de documento classifica esta página como `tpSumario` caso encontre, na primeira linha da página a frase “SUMÁRIO” (Figura 32(1)). A página de sumário é uma página obrigatória para o documento, sendo que se não for encontrada, o classificador retorna erro para o usuário.

Cada item do sumário será analisado mais tarde pelo analisador de sumário.



SUMARIO (1)

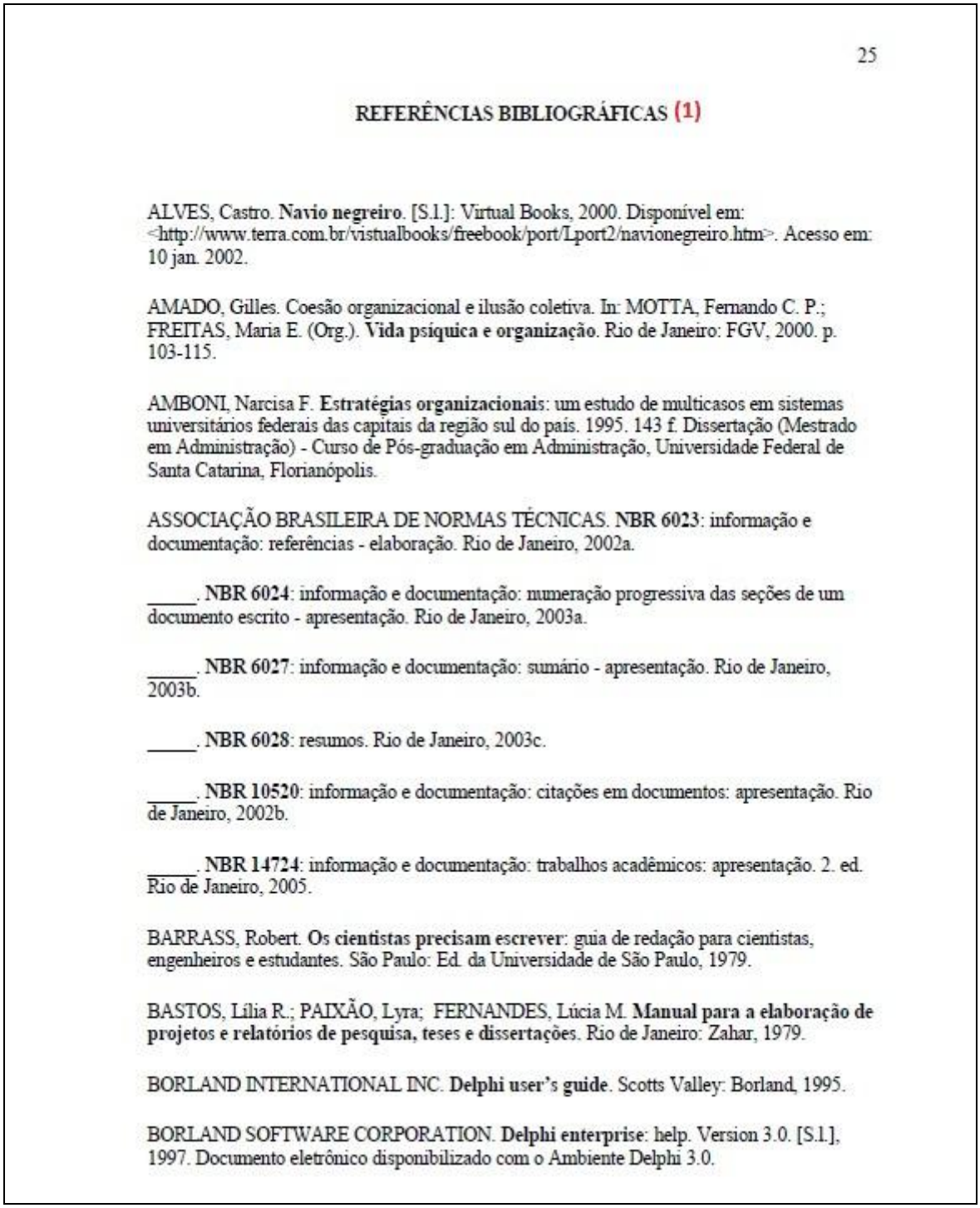
Fonte: Silva (2012a).

Figura 32 - Estrutura da página de folha de sumário

Depois de encontrar a página de sumário do TCC, o classificador começa a classificar as páginas de texto do documento. São classificadas como páginas de texto todas as páginas que são encontradas depois do sumário até encontrar a página de referências bibliográficas. Estas páginas recebem a classificação `tpTexto`.

Depois de identificar as páginas de texto, é o momento de classificar as páginas de referências bibliográficas. Na Figura 33, é mostrado um exemplo de página de referência

bibliográfica. Para que a página possa ser classificada como `tpReferencias`, o classificador deve identificar a frase "REFERÊNCIAS BIBLIOGRÁFICAS" (Figura 33(1)) no início da página. A página de referências é uma página obrigatória, sendo assim, caso o classificador não a encontre, deverá retornar erro para o usuário.



25

REFERÊNCIAS BIBLIOGRÁFICAS (1)

ALVES, Castro. **Navio negreiro**. [S.l.]: Virtual Books, 2000. Disponível em: <http://www.terra.com.br/virtualbooks/freebook/port/Lport2/navionegreiro.htm>. Acesso em: 10 jan. 2002.

AMADO, Gilles. Coesão organizacional e ilusão coletiva. In: MOTTA, Fernando C. P.; FREITAS, Maria E. (Org.). **Vida psíquica e organização**. Rio de Janeiro: FGV, 2000. p. 103-115.

AMBONI, Narcisa F. **Estratégias organizacionais**: um estudo de multicasos em sistemas universitários federais das capitais da região sul do país. 1995. 143 f. Dissertação (Mestrado em Administração) - Curso de Pós-graduação em Administração, Universidade Federal de Santa Catarina, Florianópolis.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: informação e documentação: referências - elaboração. Rio de Janeiro, 2002a.

_____. **NBR 6024**: informação e documentação: numeração progressiva das seções de um documento escrito - apresentação. Rio de Janeiro, 2003a.

_____. **NBR 6027**: informação e documentação: sumário - apresentação. Rio de Janeiro, 2003b.

_____. **NBR 6028**: resumos. Rio de Janeiro, 2003c.

_____. **NBR 10520**: informação e documentação: citações em documentos: apresentação. Rio de Janeiro, 2002b.

_____. **NBR 14724**: informação e documentação: trabalhos acadêmicos: apresentação. 2. ed. Rio de Janeiro, 2005.

BARRASS, Robert. **Os cientistas precisam escrever**: guia de redação para cientistas, engenheiros e estudantes. São Paulo: Ed. da Universidade de São Paulo, 1979.

BASTOS, Lília R.; PAIXÃO, Lyra; FERNANDES, Lúcia M. **Manual para a elaboração de projetos e relatórios de pesquisa, teses e dissertações**. Rio de Janeiro: Zahar, 1979.

BORLAND INTERNATIONAL INC. **Delphi user's guide**. Scotts Valley: Borland, 1995.

BORLAND SOFTWARE CORPORATION. **Delphi enterprise**: help. Version 3.0. [S.l.], 1997. Documento eletrônico disponibilizado com o Ambiente Delphi 3.0.

Fonte: Silva (2012a).

Figura 33 - Estrutura da página de folha de referência bibliográfica

As páginas que são encontradas depois das páginas de referência bibliográfica até o final das páginas do documento serão também classificadas como texto e receberão a

classificação `tpTexto`.

No final da classificação do documento, o classificador retorna todos os erros encontrados para o usuário.

3.3.1.3 Analisar lista de ilustrações e lista de tabelas

Devido ao fato de serem muito parecidas, as listas de ilustrações e listas de tabelas utilizam a mesma classe `TAnalisadorTabela` para fazer suas análises. No momento da criação da classe é passado um parâmetro `TTipoAnalisador` para identificar qual o tipo de lista que está sendo usada durante o processo.

Com o documento em memória devidamente classificado, o analisador do documento vai fazer a análise das listas de ilustrações desta monografia. Logo em seguida ele irá fazer a análise das listas de tabelas.

A primeira coisa a se fazer é buscar as páginas que representam a lista de ilustrações ou a lista de tabelas para que seja possível começar a análise. Para isso é utilizada a rotina `BuscaPagina()` da classe `TDocumentoPDF` passando como parâmetro `tpIlustracoes` ou `tpTabelas`. Podem existir zero ou mais de uma página de ilustrações, já que esta seção da monografia é opcional não será gerado erro caso não encontre nenhuma página. O mesmo acontece com a lista de tabelas.

Com a página da lista disponível, a classe `TAnalisadorTabela` faz análise sintática de cada item da lista, apontando os erros de construção caso existam. Se não houver erro, o item recém analisado vai para uma lista de itens a serem checados no texto posteriormente.

A análise sintática é executada pela classe `TTabelaSintatico` junto com as classes `TTabelaLexico` e `TTabelaSemantico`.

Chamando a rotina `parse()` de `TTabelaSintatico` é feita a análise e retornado o erro caso encontrado. Se não houver erros, através das ações semânticas definidas nesta mesma BNF, é possível implementar uma rotina na classe `TTabelaSemantico`, rotinas para recuperar o item que acaba de ser analisado e assim ele é adicionado na lista.

Na Figura 34 é apresentado a implementação da rotina de recuperação do item recém analisado.

```
procedure TTabelaSemantico.executeAction(action : integer; const token : TToken);
begin
  if   action > 0  then
       fListaTok.Add(token.getLexeme);

  // açoes semanticas definidas na BNF do GALS
  case   action  of
         // pega no nome e numero da Tabela/Figura/Quadro
         24 : fNome := Trim(PegaTokens);

         // pega o nome completo da Tabela/Figura/Quadro
         19,
         22 : fNomeCompleto := Trim(PegaTokensMenosUltimo);

         // final da análise
         29 : begin
                // pega a pagina do item
                fPagina := PegaPag;

                // Adiciona na lista de itens
                fListaTab.AdicionaItem(fNome, fNomeCompleto, fPagina);
              end;
  end;
end;
```

Figura 34 - Recuperação de itens na classe `TTabelaSemantico`

O Quadro 18 que se encontra no apêndice A mostra a BNF utilizada para fazer a análise sintática na lista de ilustrações e na lista de tabelas.

Como a lista de tabelas e a lista de ilustrações são listas onde é definida a página do item, agora o analisador irá procurar nas páginas definidas no item para ver se o mesmo realmente está definido na sua página correta. Caso não seja encontrado na sua página correta, um erro retornará para o usuário.

Por fim, o analisador vai fazer uma procura nas páginas de texto para ver se encontra pelo menos uma menção de cada item encontrado na lista de ilustrações e lista de tabelas. Caso não encontre nenhuma menção no texto, é gerado um erro para o usuário.

3.3.1.4 Analisar a lista de siglas

Depois de feitas as análises na lista de ilustrações e na lista de tabelas, é a vez de fazer a análise da lista de siglas.

A Figura 35, demonstra um exemplo de lista de siglas, que é diferente das listas de ilustração e de tabelas, pois não é necessário especificar a página em que aparecem as siglas, pelo simples fato de que a sigla pode aparecer em vários lugares do texto.

Figura 35 - Demonstração de uma lista de siglas

No resto, esta análise da lista de siglas é muito parecida com a análise da lista de ilustrações. Primeiro o analisador busca as páginas da lista de siglas, que como é uma página opcional, podem não aparecer no documento.

Com as páginas da lista de siglas, começa a ser feita uma análise sintática em cima de cada item desta lista. Esta análise é feita pela classe `TSiglaSintatico` ao chamar a rotina `parse()`. Esta classe junto com as outras classes `TSiglaLexico` e `TSiglaSemantico` foram geradas pelo GALS que se baseia em uma BNF construída para fazer a análise item a item na lista de siglas. Encontrando algum erro de construção, o analisador retorna um erro para o usuário.

Ao final desta análise sintática, se ela foi bem sucedida, a classe `TSiglaSemantico` adiciona o item da lista de siglas em uma lista de itens em memória, para que depois possa ser feito a checagem no texto. A Figura 36 mostra a implementação da rotina de adição do item para esta lista em memória, que é feito na classe `TSiglaSemantico`, que se baseia nas ações semânticas definidas na BNF que foi construída para essa análise.

```
procedure TSiglaSemantico.executeAction(action : integer; const token : TToken);
begin
  if   action > 0  then
       fListaTok.Add(token.getLexeme);

  // açoes semanticas definidas na BNF do GALS
  case   action  of
         // pega a sigla
         1  : fNome := token.getLexeme;

         // fim item lista sigla
         21 : begin
                fNomeCompleto := PegaCompleto;

                // adiciona item na lista de itens em memória
                fListaSiglas.AdicionaItem(fNome, fNomeCompleto, 0);
              end;
  end;
end;
```

Figura 36 - Implementação do `TSiglaSemantico`

A BNF para a análise da construção de um item da lista de siglas pode ser vista no apêndice A no Quadro 20, junto com a definição das ações semânticas no Quadro 21.

Com a lista de itens em memória povoada, agora o analisador faz uma checagem no texto do documento, verificando a existência de pelo menos uma menção para cada sigla. Caso não encontre nenhuma menção, não faz sentido a especificação desta sigla na lista de siglas, então o analisador retorna um erro para o usuário, como demonstrado a Figura 37.

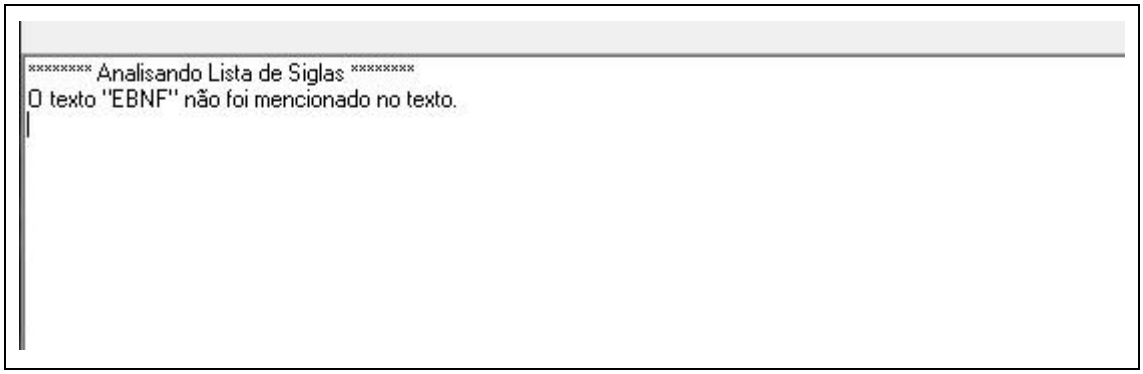


Figura 37 - Erro de sigla não encontrada

3.3.1.5 Analisar a lista de símbolos

Depois da análise da lista de siglas, o analisador do documento faz a análise da lista de símbolos. Na Figura 38 um exemplo da construção de uma lista de símbolos é mostrado.

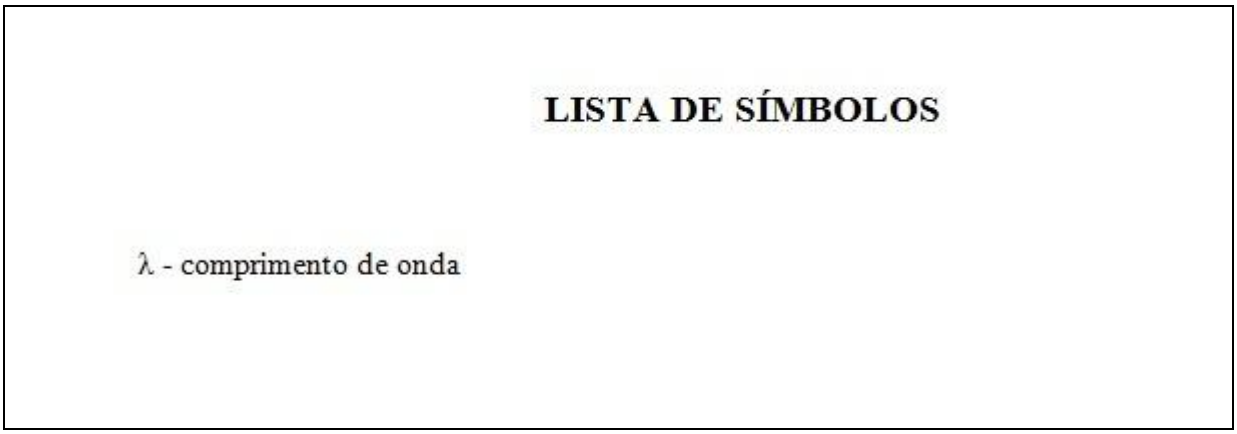


Figura 38 - Exemplo de lista de símbolos

O primeiro passo é fazer a análise sintática e depois adicionar o item analisado na lista de itens para fazer a checagem no texto depois. Para que se possa fazer a análise sintática, primeiro é preciso buscar as páginas da lista de símbolo do documento, usando a rotina `BuscaPagina()` e passando como parâmetro o tipo da página `tpSimbolos`.

Como a lista de símbolos é uma seção opcional para este tipo de documento, a rotina `BuscaPagina()` pode voltar de zero a mais de uma página. Caso não existam páginas de lista de símbolo, o analisador passa para o próximo item a ser analisado. Caso existam páginas de lista de símbolos, começa então a análise sintática item a item desta lista. Mas ao contrário das outras análises feitas até agora, esta análise teve um problema, que era o símbolo da lista em si, que deve aparecer antes do traço.

Dependendo do símbolo, o analisador léxico `TSimboloLexico` apresenta um erro de "Caractere não esperado" e aborta a análise. Este erro acontece pelo fato de o GALS não conseguir identificar este tipo de símbolo épsilon ("ε"). Assim como este símbolo, podem vir outros símbolos e o analisador vai continuar a gerar o erro. Símbolos como "@#$%" e outros do tipo não apresentam problemas para o analisador léxico.

Identificado este problema, ao invés de mandar para a análise sintática exatamente o item que aparece na lista de símbolos, é feito uma troca. O analisador de símbolos pega este símbolo especificado antes do traço ("-") e o guarda numa lista interna de símbolos, substituindo por um número, que o identifica nesta lista interna de símbolos. O Quadro 7 mostra como acontece esta substituição.

ORIGINAL	**MODIFICADO PARA ANÁLISE**
ε - épsilon	1 - épsilon

Quadro 7 - Substituição do símbolo

Esta substituição acontece apenas para que a classe `TSimboloSintatico` possa fazer a análise sintática. A classe `TSimboloSintatico`, `TSimboloLexico` e `TSimboloSemantico` são

geradas automaticamente pelo GALS que para gerar estas classes baseia-se em uma BNF construída para fazer a checagem de cada item desta lista de símbolos. A BNF usada para fazer esta análise pode ser vista no apêndice A, no Quadro 22 e no Quadro 23 são definidas as ações semânticas que a classe `TSimboloSemantico` irá executar para que seja feita a adição de cada item analisado em uma lista de itens em memória, para depois fazer a checagem no texto. A Figura 39 mostra a implementação da rotina `executeAction()` na classe `TSimboloSemantico`.

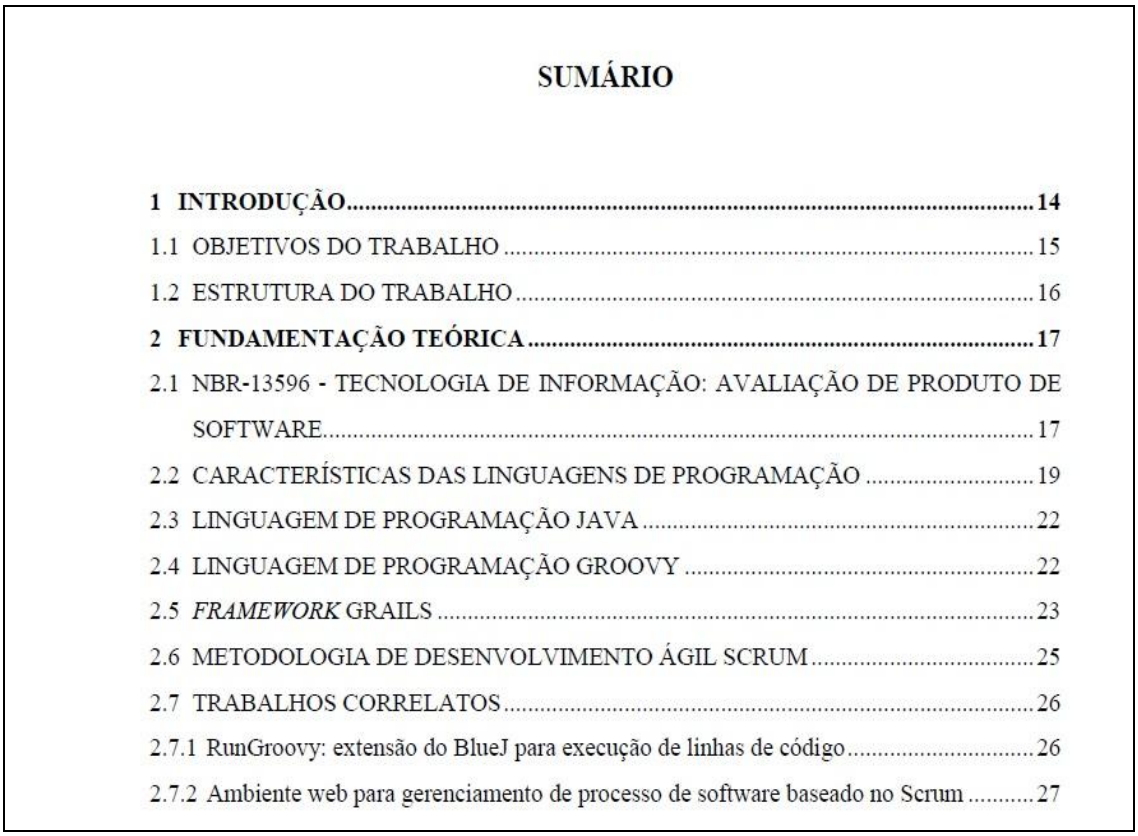
```
procedure TSimboloSemantico.executeAction(action : integer; const token : TToken);
begin
  if   action > 0  then
       fListaTok.Add(token.getLexeme);

  // açoes semanticas definidas na BNF do GALS
  case   action  of
         // pega o numero do símbolo
         1  : fNome := token.getLexeme;

         // fim item lista de símbolo
         21 : begin
                fNomeCompleto := PegaCompleto;

                // adiciona item na lista de itens em meória
                fListaSimb.AdicionaItem(fNome, fNomeCompleto, 0);
              end;
  end;
end;
```

Figura 39 - Implementação da rotina `executeAction()` na classe `TSimboloSemantico`

Depois da análise sintática e de adicionar os itens na lista de itens é feita uma procura pelos símbolos no texto do documento. Deve ser encontrada pelo menos uma menção do símbolo no texto todo, ou então não faz sentido especificar este símbolo na lista de símbolos. Caso não ache nenhuma menção é retornado erro para o usuário.

3.3.1.6 Analisar o sumário

A análise do sumário acontece depois que já foram analisadas todas as listas opcionais do documento, como as listas de ilustrações, tabelas, símbolos e siglas. A Figura 40 mostra um exemplo de construção do sumário de um documento.



SUMÁRIO

Figura 40 - Exemplo de sumário

Como nas outras análises, esta também passa por uma análise sintática antes de fazer a checagem nas páginas de texto do documento.

Para fazer esta análise sintática, antes é preciso procurar e pegar as páginas de sumário do documento. Como o sumário não é uma seção opcional, a rotina `BuscaPagina()` é usada passando-se o parâmetro do tipo da página `tpSumario`, e esta função tem que retornar no mínimo uma página. Caso não retorne página nenhuma, é retornado erro para o usuário.

No caso de encontrar uma ou mais páginas de sumário, começa então a análise sintática feita em cada item do sumário. Esta análise é feita pela classe `TSumarioSintatico` quando é chamada a rotina `parse()` desta mesma classe. Esta classe conta com a classe de análise léxica `TSumarioLexico` e com a classe de análise semântica `TSumarioSemantico`. Estas três classes são geradas automaticamente pelo GALS que por sua vez gera estas classes baseando-se em cima de uma BNF construída para fazer a análise sintática de um item de sumário. A BNF usada para fazer a análise dos itens de sumário pode ser vista no Quadro 24, que consta no apêndice A. No Quadro 25, no apêndice A, são mostradas as ações semânticas usadas pela classe `TSumarioSemantico` para adicionar os itens de sumário à uma lista de itens em memória que irá ser usada depois na checagem do texto.

Na Figura 41 é mostrado a implementação da classe `TSumarioSemantico` e a rotina usada para adicionar os itens na lista em memória.

```
  // açoes semanticas definidas na BNF do GALS
  case   action  of
         // Achou a numeração do item de sumário
         4  : fNumeracao := Trim(PegaCompleto);

         // terminou a descrição do item
         20,
         23 : fNome := Trim(PegaCompletoMenosUltimo);

         // fim item de sumário
         25 : begin
                fNomeCompleto := PegaCompleto;
                fPagina       := PegaIntUltima;

                // adiciona item na lista de memória
                fListaSumario.AdicionaItem(fNome, fNomeCompleto, fPagina);
              end;

         // fim item de sumário (referência bibliográfica)
         26 : begin
                fNome         := 'REFERÊNCIAS BIBLIOGRÁFICAS';
                fNomeCompleto := PegaCompleto;
                fPagina       := PegaIntUltima;

                // adiciona item na lista de memória
                fListaSumario.AdicionaItem(fNome, fNomeCompleto, fPagina);
              end;

         // fim item sumário (anexo)
         34 : begin
                fNomeCompleto := PegaCompleto;
                fPagina       := PegaIntUltima;

                // adiciona item na lista de memória
                fListaSumario.AdicionaItem(fNome, fNomeCompleto, fPagina);
              end;

         // fim item sumário (apêndice)
         42 : begin
                fNomeCompleto := PegaCompleto;
                fPagina       := PegaIntUltima;

                // adiciona item na lista de memória
                fListaSumario.AdicionaItem(fNome, fNomeCompleto, fPagina);
              end;
  end;
end;
```

Figura 41 - Implementação da classe `TSumarioSemantico`

Depois da análise de todos os itens, o analisador vai fazer a checagem da página do item de sumário. Cada item do sumário possui uma página, e agora é checado para ver se o item em questão está corretamente na página especificada ou não. Caso não esteja correto, o analisador retorna um erro para o usuário.

3.3.1.7 Analisar a referência bibliográfica

A análise da referência bibliográfica é a última parte da análise do documento. Segundo Silva (2012b), existem vários tipos de referências a serem estudados. Na Figura 42 são mostrados dois exemplos de referências de livros.

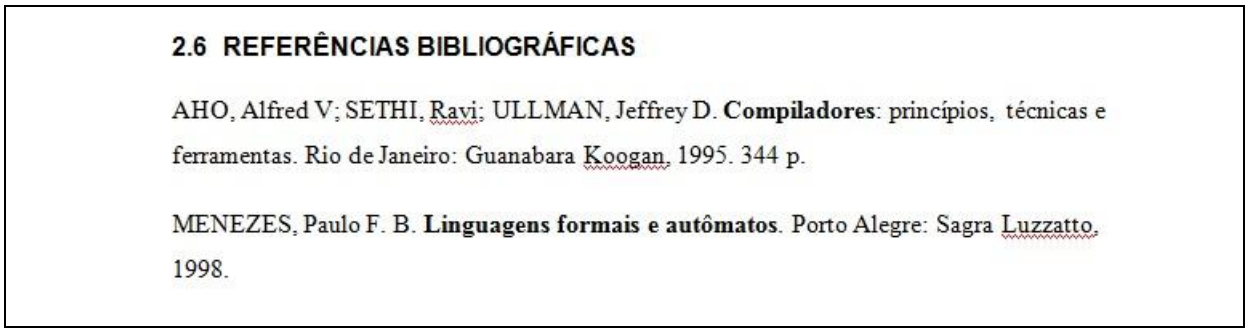


Figura 42 - Exemplos de referência bibliográfica

Na referência bibliográfica não é feita a análise sintática em cada item da referência. Mas mesmo não analisando sintaticamente os itens, deve-se conseguir identificar em cada referência os autores e o ano de publicação da obra para que possa ser feita a checagem por citações mais tarde.

A referência bibliográfica também é uma seção que não é opcional neste documento. Portanto quando é chamada a rotina `BuscaPagina()`, passando o parâmetro de tipo de página `tpReferencias`, essa rotina tem que retornar no mínimo uma página de referência ou então o analisador retorna um erro para o usuário.

Se a rotina retorna uma ou mais páginas de referência bibliográfica, então é feita a análise em cada item de referência para tentar retirar as informações de autores e ano de publicação da obra. Em seguida estas informações são adicionadas a uma lista de itens de referência em memória para que depois possa ser feita a procura no texto por citações.

As citações podem acontecer como mostra o Quadro 8. Sabendo-se do formato das citações no texto, são montadas então duas chaves de procura da citação no texto. Uma para cada modo de citação (Quadro 8).

MODO 1	**MODO 2**
(SOBRENOME, ano ...	Sobrenome (ano, ...

Quadro 8 - Modos de citação de referência bibliográfica

Usando os exemplos da Figura 42, o Quadro 9 mostra os resultados que a rotina `BuscaInfoReferencia()` vai encontrar e as chaves que irão ser montadas para a procura no texto.

	EXEMPLO 1	EXEMPLO 2
AUTOR 1	AHO	MENEZES
AUTOR 2	SETHI	
AUTOR 3	ULLMAN	
ANO	1995	1998
CHAVE 1	(AHO, SETHI e ULLMAN, 1995	(MENEZES, 1998
CHAVE 2	Aho, Sethi e Ullman (1995	Menezes (1998

Quadro 9 - Resultados da procura de autores e ano de publicação

Depois de adicionar os itens na lista em memória é feito então a procura nas páginas de texto do documento. Primeiro é feita a procura pela chave 1 montada e depois é feita a procura pela chave 2 montada. Se algum item está definido nas referências bibliográficas e não é achado nenhum tipo de citação para ele no texto, o analisador retorna uma mensagem de erro para o usuário.

No final desta checagem é feita então uma checagem onde o analisador de referências tenta identificar as possíveis citações no texto.

Para poder tentar identificar uma possível citação, primeiro é preciso achar um componente básico de uma citação, que é o caractere abre parênteses (“(”). O caractere parênteses sempre está presente quando é feito uma citação. Então quando o analisador encontra este caractere ele começa a fazer a identificação da citação, como demonstrado nas Figura 43 e Figura 44.

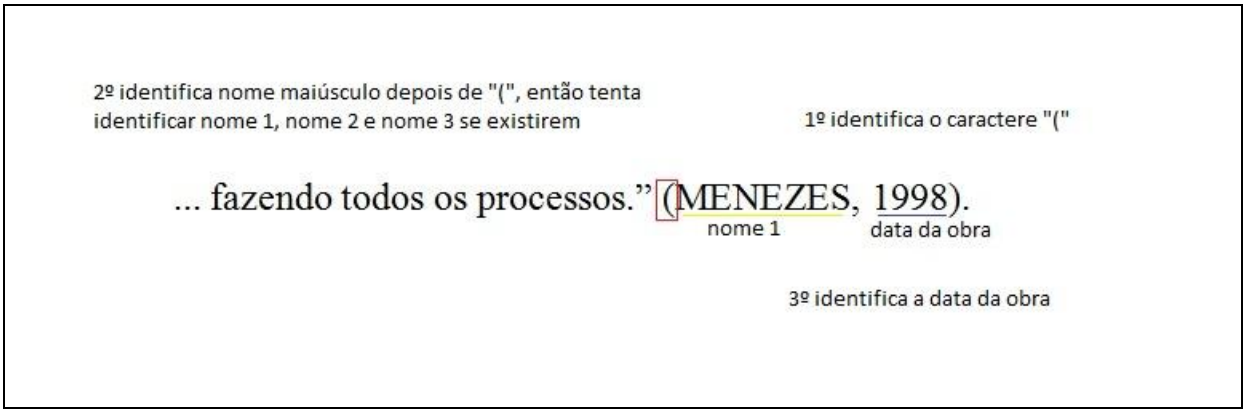


Figura 43 - Identificação de uma possível citação do modo 1

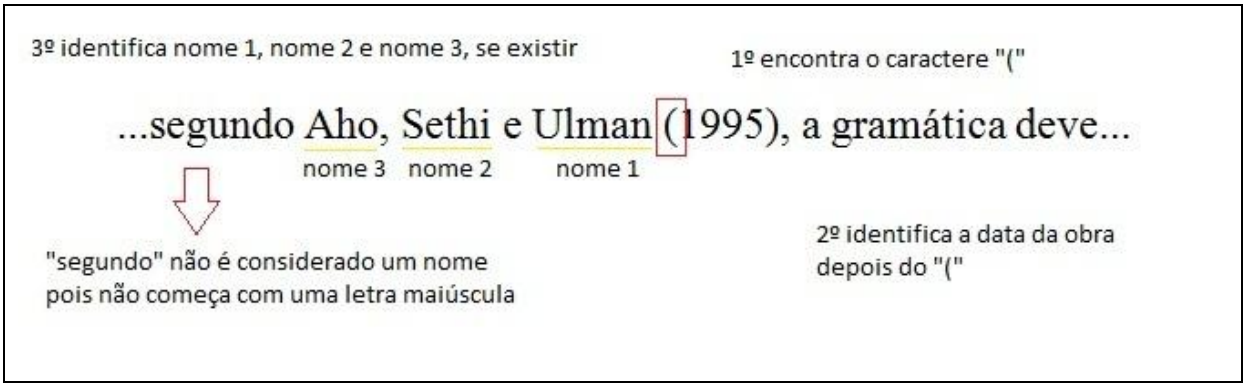


Figura 44 - Identificação de uma possível citação do modo 2

Quando se fala em identificação é usada sempre a palavra possível junto, pois nem sempre tudo o que tem parênteses e segue o padrão especificado na Figura 43 e na Figura 44 são necessariamente uma citação.

Depois de identificada esta possível citação no texto, é feito uma procura na lista de itens de referências para ver se a citação está devidamente referenciada na seção de referência bibliográfica. Caso não esteja referenciada, uma mensagem sugerindo um problema de referência será apresentada ao usuário.

3.3.1.8 Procura no texto

As funções de procura nas páginas de texto são utilizadas por todos os analisadores. Todas estas procuras são feitas pela classe `TProcuraTexto`. Esta classe tem basicamente uma função apenas, a função `ExecutaProcura()` que é passado a mensagem que se está procurando e ela retorna quantas vezes foi encontrado a mensagem no texto.

A rotina de procura funciona da seguinte maneira:

a) primeiro é pego a mensagem de procura e é retirado a primeira palavra desta mensagem;
b) depois é feita uma procura nas linhas, procurando esta primeira palavra que foi separada da mensagem;
c) se for achado alguma incidência desta primeira palavra no texto, é feito então uma comparação com a mensagem de procura inteira, se o texto encontrado for igual a mensagem inteira, então obteve um sucesso na procura, e continua a procura até o fim do texto.

O Quadro 10 abaixo exemplifica um caso.

Mensagem de procura = ‘Esta é a mensagem de procura’
Palavra usada para a primeira procura = ‘Esta’
O resto da mensagem para comparação final = ‘Esta é a mensagem de procura’

Quadro 10 - Exemplo de procura de uma mensagem no texto

Durante os testes do protótipo deparou-se com algumas situações onde ocorriam problemas na procura das mensagens nos textos. O problema acontece quando o componente PDFtoolkit extrai o texto do documento original, estava extraindo algumas linhas com mais de um espaço entre as palavras do texto original. Não foi conseguido levantar o motivo certo

do porquê deste acontecimento, mas uma hipótese levantada é que como o texto digitado hoje tem que ser justificado, as vezes o editor de texto em que o documento original foi digitado dá uma “esticada” nas palavras para que o texto fique corretamente justificado, aumentando assim visualmente o espaço entre as palavras, mas sem alterar o texto em si. Acredita-se que quando o componente vai fazer a leitura do arquivo original em PDF, ele identifique a linha como se ela tivesse dois espaços entre as palavras e não um. Extraindo o texto de forma errada e não corretamente como no original.

Sabendo deste problema, então quando é adicionado o texto na rotina de procura para se fazer a procura nos textos, é feito uma pequena correção com a rotina `TiraEspacosAMais()`, onde a rotina tira, caso existam, os espaços a mais que estão no texto, deixando o texto correto para efetuar uma procura.

A classe `TChecaListaItens` é usada em conjunto com a `TProcuraTexto` para que sejam feitas os dois tipos de procura existentes no protótipo. Uma é a `ExecutaChecagemPagina()` que faz a checagem da página correta de todos os itens adicionados nas lista de memória. A outra é a `ExecutaChecagem()`, que faz a procura em todo o texto para cada item adicionado nas listas de memória.

3.3.2 Operacionalidade da implementação

Para facilitar o entendimento da implementação deste protótipo, será descrito aqui o processo de implementação dividido em partes. Uma parte para cada tipo de análise que é feita pelo protótipo para no final mostrar os resultados ao usuário.

3.3.2.1 Configuração inicial e escolha do arquivo

Antes do usuário iniciar a análise do seu documento, ele precisa configurar algumas informações iniciais e essenciais para o funcionamento correto do protótipo.

A Figura 45 mostra a tela inicial e os campos que devem ser informados antes para que a análise do documento possa ser feita corretamente.

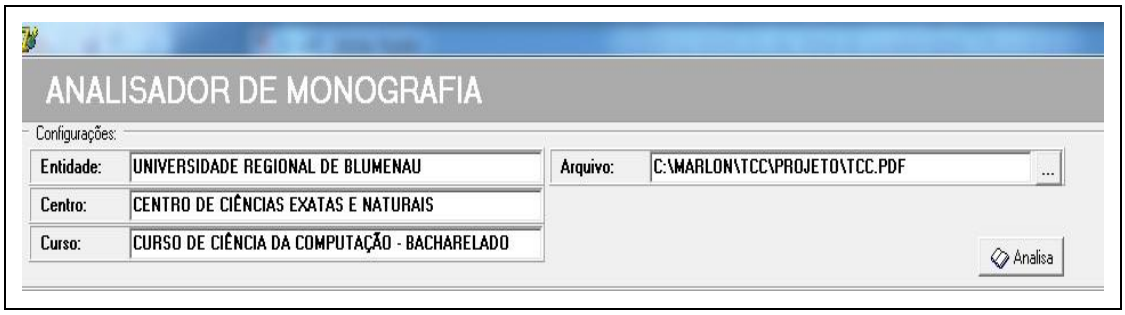


Figura 45 - Campos de configuração do analisador

A entidade, o centro e o curso são campos em que o usuário coloca a informação do nome do seu curso, o centro a qual ele pertence e a qual entidade o TCC vai ser submetido.

No campo arquivo, o usuário seleciona o seu TCC em formato PDF, podendo-se digitar o caminho ou apertando o botão ao lado do campo onde ele abre uma tela para a escolha do arquivo para o usuário.

Com estas informações acima devidamente preenchidas, clicando no botão `Analisa`, o protótipo começa a fazer a análise do documento selecionado. Na Figura 46 é apresentado o resultado da análise do documento, que mostra os problemas encontrados na análise em forma de texto para o usuário verificar.

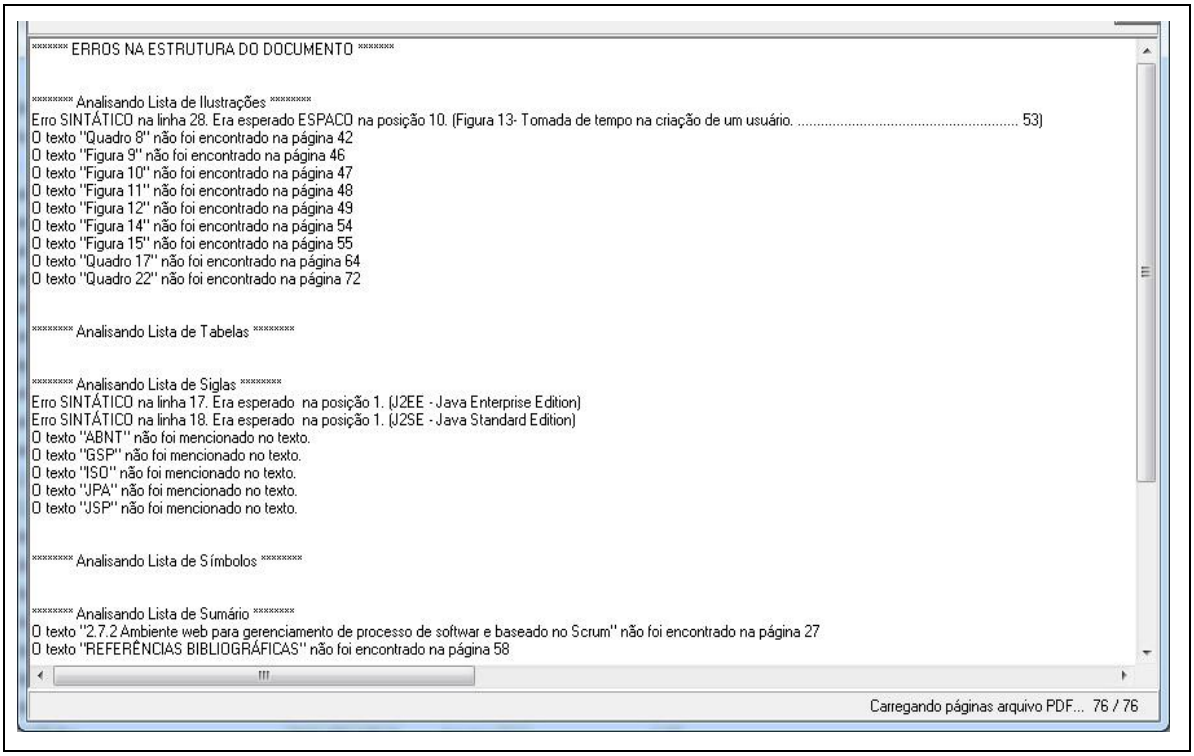


Figura 46 - Resultado da análise do documento selecionado

3.4 RESULTADOS E DISCUSSÃO

Para fazer os testes no protótipo implementado no presente trabalho foram utilizados 3 TCCs. Dois destes TCCs foram entregues em anos anteriores e também foi usado este próprio TCC para fazer as validações no protótipo e suas funções.

Com relação a fazer a verificação do espaçamento da margem e dos parágrafos do documento, foi encontrada uma dificuldade, pois a ferramenta utilizada para a extração de dados, PDFtoolkit, tem uma limitação ao fazer a extração do texto de uma página quando nesta página ela possui um quadro ou tabela. Ao extrair o texto de uma página que contenha um quadro ou tabela, ele não consegue diferenciar aquele texto, e identificar que aquele texto faz parte de um quadro ou tabela na página. Como a verificação das margens e dos parágrafos iriam ser feitos em cima da posição, em milímetros, da primeira palavra da linha em relação à margem esquerda, no caso de um texto dentro de um quadro ou tabela, o protótipo iria encontrar todos os textos do quadro ou tabela como estando fora da margem ou parágrafo, uma vez que geralmente um quadro ou tabela não obedece a configuração de margens e parágrafos do editor de texto, mostrando assim muitas linhas de "erro", onde efetivamente não seriam erros e sim textos dentro de quadros ou tabelas.

Por este motivo não foi dado continuidade a este item do protótipo.

No Quadro 11 são apresentados os resultados da classificação das páginas do documento selecionado pelo usuário. O resultado volta vazio e sem mensagens de erro, pois o classificador foi capaz de identificar e classificar todas as seções obrigatórias ou não existentes no TCC.

```
1  ******* ERROS NA ESTRUTURA DO DOCUMENTO *******
2
```

Quadro 11 - Resultado da classificação do documento

Os resultados da análise podem ser vistos no Quadro 12, onde os dois primeiros erros são erros sintáticos de construção do item da lista de ilustrações, onde no primeiro depois do número "13" era esperado um espaço, mas o analisador encontrou "-".

No segundo erro sintático, depois do número 55 era esperado o fim da linha, mas foi encontrado espaços, gerando os erros para o usuário. Neste caso, a BNF para análise da construção do item de ilustração poderia ser melhorada para ignorar estes espaços no final da linha, já que estes espaços não influenciam no resultado final da análise.

Os outros erros apresentados são erros de localização de página, onde a "Figura 9" foi

colocada na lista de ilustrações e indicada que estaria na página 46 e o analisador não encontrou esta figura na página 46, gerando uma mensagem de erro.

1	******** Analisando Lista de Ilustrações ********
2 3	Erro SINTÁTICO na linha 28. Era esperado ESPACO na posição 10. (Figura 13- Tomada de tempo na criação de um usuário. ..... 53)
4 5	Erro SINTÁTICO na linha 36. Era esperado FIM DA LINHA na posição 115. (Figura 15 - Gráfico de memória consumida pelo Java. ..... 55)
6	O texto "Figura 9" não foi encontrado na página 46
7	O texto "Figura 10" não foi encontrado na página 47
8	O texto "Figura 11" não foi encontrado na página 48
9	O texto "Figura 12" não foi encontrado na página 49
10	O texto "Figura 14" não foi encontrado na página 54
11	O texto "Quadro 17" não foi encontrado na página 64
12	O texto "Quadro 22" não foi encontrado na página 72

Quadro 12 - Resultado da análise da lista de ilustrações do documento

No Quadro 13 abaixo são apresentados os resultados da análise da lista de tabelas. O analisador de tabelas faz basicamente a mesma análise que é feita na análise de ilustrações. No caso deste documento, ele não possui uma lista de tabelas para ser analisada, sendo por essa razão o resultado da análise vazio.

1	******** Analisando Lista de Tabelas ********
2	

Quadro 13 - Resultado da análise da lista de tabelas do documento

A análise da lista de siglas é um pouco diferente da análise da lista de ilustração e de tabelas. O resultado pode ser visto no Quadro 14 e mostra que não houveram erros sintáticos na construção dos itens da lista de siglas. Foram encontrados alguns problemas porquê as siglas “GSP”, “ISO”, “JPA” e “JSP” foram definidas na lista de siglas mas não foram mencionadas no texto nenhuma vez, gerando assim uma mensagem de erro para o usuário.

1	******** Analisando Lista de Siglas ********
2	O texto "GSP" não foi mencionado no texto.
3	O texto "ISO" não foi mencionado no texto.
4	O texto "JPA" não foi mencionado no texto.

5	O texto "JSP" não foi mencionado no texto.

Quadro 14 - Resultado da análise da lista de siglas do documento

No Quadro 15 são apresentados os resultados da análise da lista de símbolos. Neste exemplo o documento não tem a seção de lista de símbolos. A análise da lista de símbolos é muito parecida com a análise da lista de siglas. Ela somente analisa a construção correta dos seus itens e depois procura por uma menção do item no texto.

1 2	******** Analisando Lista de Símbolos ********

Quadro 15 - Resultado da análise da lista de símbolos do documento

No Quadro 16 são apresentados os resultados da análise do sumário do documento. Nesta análise são feitas a análise da construção de cada item do sumário e a checagem da página de cada item.

O Quadro 16 mostra nenhum erro na construção dos itens e mostra um erro na checagem de página do item “2.7.2”. Pode-se observar que na definição do item no sumário (Figura 47) e a sua menção na página 27 (Figura 48) estão idênticas. Este erro somente foi encontrado porque ferramenta que é usada para a extração do texto, PDFtoolkit, comete um erro ao extrair o texto do sumário. Extraindo o texto da maneira como mostra o Quadro 16, onde ele separa a palavra “software” em “softwar e”. Desta maneira, o analisador de sumário faz a procura pela frase inteira com a palavra separada, e não a acha na página 27, quando ela está corretamente na página 27, gerando a mensagem de erro para o usuário.

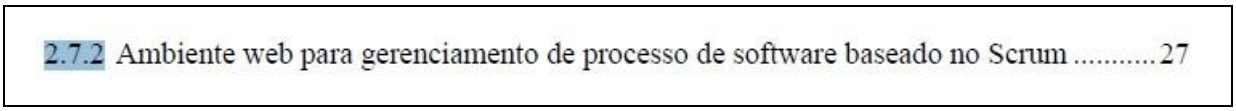


Figura 47 - Construção do item 2.7.2 no sumário

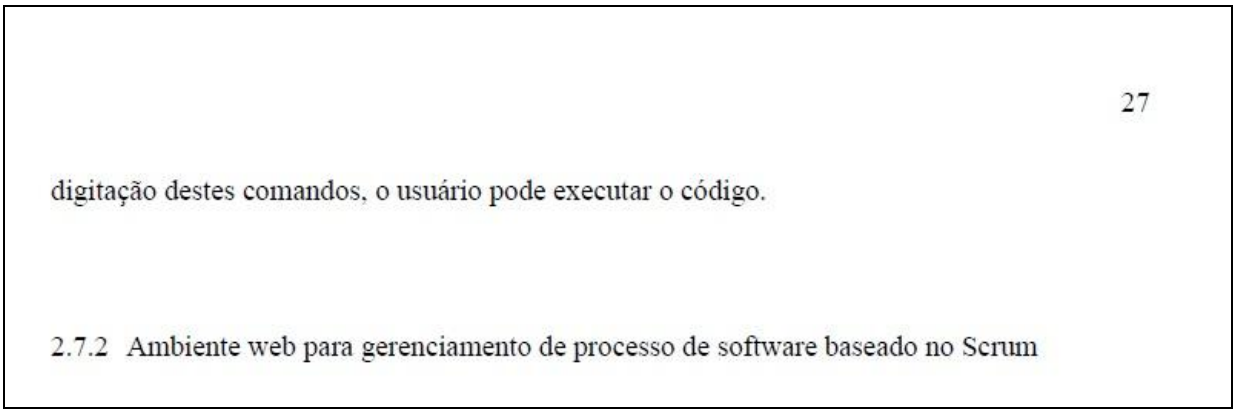


Figura 48 - Item 2.7.2 descrito na página 27

Tirando este “erro” encontrado pelo analisador de sumário, não existem mais erros na definição do sumário do documento.

1	******** Analisando Lista de Sumário ********
2	
3	O texto "2.7.2 Ambiente web para gerenciamento de processo de softwar e baseado no Scrum" não foi encontrado na página 27
4	

Quadro 16 - Resultado da análise do sumário do documento

No Quadro 17 são apresentados os resultados da análise das referências bibliográficas. São 6 erros para este documentos, todos mostrando referências que foram colocadas nas referências bibliográficas mas não foram corretamente referenciadas no texto.

Nas citações que se encontram nas linhas 3 e 17, as citações existem no texto, mas foram feitas com acentuação e por este motivo o analisador de referências não os encontrou, gerando mensagem de erro.

Na citação da linha 14 ela foi feita no texto sem a acentuação, também gerando mensagem de erro por não ter encontrado a citação.

As citações das linhas 8 e 11 não foram em nenhum momento citados no texto, gerando a mensagem de erro.

Na última mensagem de erro apresentada, na linha 23, “NOOMAN” foi escrito de maneira errada “NOONAN”, fazendo com que o analisador de referência não encontre a citação no texto e gerando a mensagem de erro para o usuário.

1	******** Analisando Lista de Referências bibliográficas ********
2	
3	Não foi encontrada nenhuma citação para a referência "ASSOCIACAO BRASILEIRA DE
4	NORMAS TECNICAS. NBR-13596: tecnologia de informação - avaliação de produto de
5	software - características de qualidade e diretrizes para o seu uso. Rio de Janeiro: ABNT, 1996. 10
6	p.".
7	
8	Não foi encontrada nenhuma citação para a referência "CONTROLCHAOS. What is Scrum.
9	[S.l.], 2005. Disponível em: <http://www.controlchaos.com>. Acesso em: 25 maio 2010. ".
10	
11	Não foi encontrada nenhuma citação para a referência "DOEDORLEIN, Osvaldo P. Aprendendo
12	Groovy. Java Magazine, São Paulo, ano IV, n. 32, p. 30-44, jan. 2006.".
13	
14	Não foi encontrada nenhuma citação para a referência "KÖNIG, Dierk. Groovy em ação. Rio de
15	Janeiro: Alta Books, 2007.".
16	
17	Não foi encontrada nenhuma citação para a referência "MULLER, Henrique M. RunGroovy:
18	extensão do bluej para execução de linhas de código. 2007. 67 f. Trabalho de Conclusão de Curso
19	(Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade
20	Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2007-
21	II/TCC2007-2-18-VF HenriqueMMuller.pdf>. Acesso em: 1 jun. 2010.".
22	
23	Não foi encontrada nenhuma citação para a referência "TUCKER, Allen B.; NOOMAN, Robert
24	E. Linguagens de programação: princípios e paradigmas. 2. ed. São Paulo: McGraw-Hill, 2008.".

Quadro 17 - Resultado da análise das referências bibliográficas do documento

4 CONCLUSÕES

Este trabalho se propôs a implementar um protótipo de um analisador de monografias voltada para o curso de Ciência da Computação da Universidade Regional de Blumenau, verificando algumas características importantes para que um TCC possa ser considerado correto para avaliação dos professores.

Para a implementação do protótipo foram utilizadas as ferramentas PDFtoolkit, para fazer a leitura e extração de dados em um arquivo PDF, e o GALS para construir, testar e gerar as classes de análise léxica e sintática nos itens estudados no trabalho.

No que diz respeito a verificação dos espaçamentos de margem esquerda e parágrafos do documento, houveram alguns problemas com relação a ferramenta PDFtoolkit, que dificultaram a extração da posição do texto normal e do texto contido em quadros ou tabelas. Já que a ferramenta não consegue distinguir o que é uma linha de texto normal e o que é uma linha de texto dentro de um quadro ou tabela, fica difícil criticar os problemas de erros de margens e parágrafos. Este objetivo não foi concluído, pela dificuldade apresentada pela ferramenta.

Os outros objetivos propostos neste trabalho puderam ser analisados, estudados e implementados no protótipo com sucesso, retornando assim resultados satisfatórios para a conclusão dos mesmos, como podem ser vistos na seção anterior de resultados e discussões.

Este protótipo pode ser utilizado tanto por acadêmicos como para os professores que corrigem as monografias na universidade. Pelos acadêmicos pode ser usado como um reforço para fazer um levantamento dos problemas existentes na monografia para que possam ser acertados antes de entregar ao professor orientador, acelerando assim o processo do acerto da monografia.

Pelos professores também pode ser utilizado afim de procurar os problemas e apontá-los para o acadêmico que fez a monografia, eliminando algumas etapas da correção rapidamente.

Como limitação deste protótipo, pode ser apontada a falta de análise sintática e léxica nas referências bibliográficas. É interessante uma análise a este ponto pois muitos dos erros dos acadêmicos acontecem ao escrever essa seção do TCC. A análise sintática da construção de um item somente é realizada nas listas de ilustrações, lista de tabelas, lista de siglas, lista de símbolos e no sumário. Devido a complexidade das construções de uma referência e seus diversos tipos de referências, não houve tempo hábil para a construção e testes de uma BNF

satisfatória para fazer a análise sintática e léxica dos itens da seção de referências bibliográficas.

4.1 EXTENSÕES

Como sugestão para continuação deste trabalho e melhoria do sistema, pode-se citar:

a) fazer a análise sintática e léxica da seção de referências bibliográficas;
b) trabalhar de maneira diferenciada as tabelas e quadros descritos no TCC;
c) fazer a checagem das margens e parágrafos do texto;
d) extrair as imagens do texto e fazer uma análise diferenciada para elas no documento;
e) verificar e tratar os tipos de fontes e tamanhos de fontes na checagem do documento;
f) construir um leitor de arquivos PDFs que possa extrair todos os dados necessários para fazer a análise no documento.

REFERÊNCIAS BIBLIOGRÁFICAS

ADOBE. **História do PDF Adobe.** [S.l.], 2012. Disponível em: <http://www.adobe.com/br/products/acrobat/adobepdf.html>. Acesso em: 17 ago. 2012.

ADOBE SYSTEMS INCORPORATED. **Document management – portable document format – part 1:** PDF 1.7. [S.l.], 2008. Disponível em: <http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf>. Acesso em: 31 ago. 2012.

AHO, Alfred V; SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores**: princípios, técnicas e ferramentas. Rio de Janeiro: Guanabara Koogan, 1995. 344 p.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724**: informação e documentação: trabalhos acadêmicos apresentação. 2. ed. Rio de Janeiro, 2005.

CANTU, Marco. **Dominando o Delphi 7**: a bíblia. São Paulo: Pearson Education do Brasil, 2003. 801 p.

DRUMOND, Regina C. **Monografia:** como fazer. [S.l.], 2004. Disponível em: <http://www.feis.unesp.br/extensao/teia-saber/teia2004/Downloads/Como%20Fazer%20Monografia.pdf.>. Acesso em: 29 ago. 2012.

GESSER, Carlos E. **GALS**: documentação. Versão 2003.10.03. [S.l.], [2003]. Documento eletrônico disponibilizado com o Ambiente GALS.

GNOSTICE. **Downloads**. [S.l.], 2012a. Disponível em: <http://www.gnostice.com/PDFtoolkit_VCL.asp>. Acesso em: 17 ago. 2012.

_____. **Overview**. [S.l.], 2012b. Disponível em: <http://www.gnostice.com/PDFtoolkit_VCL.asp>. Acesso em: 31 ago. 2012.

GRUNE, Dick. **Projeto moderno de compiladores**: implementação e aplicações. Rio de Janeiro: Campus, 2001. 671 p.

HIEBERT, Dennis. **Protótipo de um compilador para a linguagem PL-SQL**. 2003. 52 f, il. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2003.

ISO. **ISO 32000-1:2008 - document management - portable document format - part 1**: PDF 1.7. [S.l.], 2012. Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51502>. Acesso em: 30 ago. 2012.

LAKATOS, Eva M.; MARCONI, Marina de A. **Metodologia do trabalho científico:** procedimentos básicos; pesquisa bibliográfica, projeto e relatório; publicações e trabalhos científicos. 3. ed. São Paulo: Atlas, 1991. 214 p.

LOUDEN, Kenneth C. **Compiladores**: princípios e práticas. São Paulo: Thomson Pioneira, 2004. 569 p.

MONOGRAFIA. In: Wikipédia, a enciclopédia livre. [S. l.]: Wikimedia Foundation, 2012. Disponível em: <http://pt.wikipedia.org/wiki/Monografia>. Acesso em: 20 ago. 2012.

OECHSLER, Thiago Minhaqui. **Processamento de texto escrito em linguagem natural para um sistema conversor texto-fala**. 2009. 72 f, il. Trabalho de Conclusão de Curso - (Graduação em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2009.

ORSI, Vilmar. **Gerador de documentação para Linguagem C, utilizando templates**. 2006. 98 f. Trabalho de Conclusão de Curso - (Graduação em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2006.

PRICE, Ana M. A; TOSCANI, Simão Sirineo. **Implementação de linguagens de programação**: compiladores. 2. ed. Porto Alegre: UFRGS/Instituto de Informática: Sagra Luzzatto, 2001. 195 p.

SALOMON, Delcio V. **Como fazer uma monografia:** elementos de metodologia do trabalho científico. 6. ed. Belo Horizonte: Interlivros, 1979. 317 p.

SILVA, José R. V. **Modelo do texto final.** Blumenau, 2012a. Notas de Aula. Disciplina Trabalho de Conclusão de Curso II, Ciência da Computação da Universidade Regional de Blumenau. Disponível em: <http://www.inf.furb.br/~roque/tcc/compactados/VolumeFinal-v8-3-14.7z>. Acesso em: 22 ago. 2012.

_____. **Como fazer referências bibliográficas.** Blumenau, 2012b. Notas de Aula. Disciplina Trabalho de Conclusão de Curso II, Ciência da Computação da Universidade Regional de Blumenau. Disponível em: <http://www.inf.furb.br/~roque/tcc/como_fazer_referencias_bibliograficas.pdf>. Acesso em: 22 set. 2012.

APÊNDICE A – Gramáticas utilizadas na análise dos itens do documento

Neste apêndice são apresentadas as gramáticas que foram utilizadas no protótipo para fazer as análises dos itens, juntamente com suas respectivas semânticas.

No Quadro 18 é representada a gramática utilizada para fazer as análises do itens de uma lista de tabelas e ilustrações. Esta gramática foi construída na ferramenta GALS e nela também são vistas as ações semânticas desta gramática.

```
# Definições Regulares
d        : [0-9]
lmai     : [A-Z Á-Ú À-Ù Â-Û Ã-Õ Ä-Ü Ç]
lmin     : [a-z á-ú à-ù â-û ã-õ ä-ü ç]
nonchar: [^abcdefghijklmnopqrstuvwyxzABCDEFGHIJKLMNOPQRSTUVWYXZ]

# Tokens
// tabelas, figuras e quadros          // simbolos
fig          : "Figura"                 espaco          : " "
tab          : "Tabela"                 traco           : "-"
quad         : "Quadro"                 ponto           : "."
                                        varios_pontos : "." "."+
//                                      aspas           : \" | "'"
char_mai     : {lmai}                   simbolo         : "@" | "#" | "$" | "%" | "&"
nome_mai     : {lmai}+                  conchete        :  "(" | ")" | "{" | "}" | "[" | "]" | "<" | ">"
nome         : {lmai} {lmin}+           matem           : \+ | "*" | "=" | "/"
alfa         : {lmin}+                  separador       : "_" | "\"
numero       : {d}+                     pontuacao       : "!" | "?" | ";" | ":" | ","

simb_varios  : "¹" | "²" | "³" | "£" | "¢" | "¬" | "§" | "º" | "ª"
nono         : {nonchar}

# Gramática
<tabelas_ou_quadros_ou_figuras>  ::= <tab_ou_quad_ou_fig> espaco #23 numero #24
                                     espaco #25 traco #26 espaco #27 <desc>
                                     espaco #28 numero #29;
<tab_ou_quad_ou_fig>             ::= fig #1 | tab #2 | quad #3;
<desc>                           ::= <escolhe_todos> <varios>;
<escolhe_todos>                  ::= char_mai #4 | nome_mai #5
                                   | nome #6 | alfa #7
                                   | numero #8 | espaco #9
                                   | traco #10 | aspas #11
                                   | simbolo #12 | conchete #13
                                   | matem #14 | separador #15
                                   | pontuacao #16 | simb_varios #17
                                   | nono #18;
<varios>                         ::= ponto #19 espaco #20 varios_pontos #21
                                   | <escolhe_todos> <varios>
                                   | varios_pontos #22;
```

Quadro 18 – BNF - análise de itens da lista de tabelas e ilustrações e as ações semânticas

O Quadro 19 mostra as ações semânticas e suas respectivas ações tomadas pelo protótipo durante a análise da lista de tabelas e ilustrações.

Número	Ação
#1	Guarda o token fig na lista de tokens
#2	Guarda o token tab na lista de tokens
#3	Guarda o token quad na lista de tokens
#4	Guarda o token char_mai na lista de tokens
#5	Guarda o token nome_mai na lista de tokens
#6	Guarda o token nome na lista de tokens
#7	Guarda o token alfa na lista de tokens
#8	Guarda o token numero na lista de tokens
#9	Guarda o token espaco na lista de tokens
#10	Guarda o token traco na lista de tokens
#11	Guarda o token aspas na lista de tokens
#12	Guarda o token simbolo na lista de tokens
#13	Guarda o token conchete na lista de tokens
#14	Guarda o token matem na lista de tokens
#15	Guarda o token separador na lista de tokens
#16	Guarda o token pontuacao na lista de tokens
#17	Guarda o token simb_varios na lista de tokens
#18	Guarda o token nono na lista de tokens
#19	Guarda o token ponto na lista de tokens e extrai da lista de tokens o nome completo do item que está sendo analisado
#20	Guarda o token espaco na lista de tokens
#21	Guarda o token vários_pontos na lista de tokens
#22	Guarda o token vários_pontos na lista de tokens e extrai da lista de tokens o nome completo do item que está sendo analisado
#23	Guarda o token espaco na lista de tokens
#24	Guarda o token numero na lista de tokens e extrai da lista de tokens o nome do item que está sendo analisado
#25	Guarda o token espaco na lista de tokens
#26	Guarda o token traco na lista de tokens
#27	Guarda o token espaco na lista de tokens
#28	Guarda o token espaco na lista de tokens
#29	Guarda o token numero na lista de tokens e extrai da lista de tokens o numero da pagina do item que está sendo analisado e adiciona o item na lista em memória

Quadro 19 - Ações semânticas realizadas na análise de itens de tabela e ilustrações

No Quadro 20 é demonstrada uma gramática construída na ferramenta GALS. Esta gramática foi utilizada para fazer a análise léxica e sintática de um item de lista de siglas. Estão representadas também as ações semânticas no quadro 20.

```
# Definições Regulares
d         : [0-9]
lmai      : [A-Z Á-Ú À-Ù Â-Û Ã-Õ Ä-Ü Ç]
lmin      : [a-z á-ú à-ù â-û ã-õ ä-ü ç]
nonchar: [^abcdefghijklmnopqrstuvwyxzABCDEFGHIJKLMNOPQRSTUVWYXZ]
# Tokens
// siglas
char_mai        : {lmai}
nome_mai        : {lmai}+
nome            : {lmai} {lmin}+
alfa            : {lmin}+
numero          : {d}+

// simbolos
espaco          : " "
traco           : "-"
ponto           : "."
aspas           : \" | "'"
simbolo         : "@" | "#" | "$" | "%" | "&"
conchete        : "(" | ")" | "{" | "}" | "[" | "]" | "<" | ">"
matem           : \+ | "*" | "=" | "/"
separador       : "_" | "\"
pontuacao       : "!" | "?" | ";" | ":" | ","
simb_varios     : "¹" | "²" | "³" | "£" | "¢" | "¬" | "§" | "º" | "ª"
nono            : {nonchar}
# Gramática
<sigla>                 ::= nome_mai #1 espaco #2 traco #3 espaco #4 <desc> #21;
<desc>                  ::= <escolhe_todos> <varios>;
<escolhe_todos>         ::= char_mai #5 | nome_mai #6
                            | nome #7 | alfa #8
                            | numero #9 | espaco #10
                            | traco #11 | aspas #12
                            | simbolo #13 | conchete #14
                            | matem #15 | separador #16
                            | pontuacao #17 | simb_varios #18
                            | nono #19 | ponto #20;
<varios>                ::= <escolhe_todos> <varios> |  ε ;
```

Quadro 20 – BNF – análise de itens da lista de siglas e as ações semânticas

O Quadro 21 mostra as ações semânticas e suas respectivas ações tomadas pelo protótipo durante a análise da lista de siglas.

Número	Ação
#1	Guarda o token nome_mai na lista de tokens e extrai da lista de tokens a sigla do item que está sendo analisado
#2	Guarda o token espaco na lista de tokens
#3	Guarda o token traco na lista de tokens
#4	Guarda o token espaco_mai na lista de tokens
#5	Guarda o token char_mai na lista de tokens
#6	Guarda o token nome_mai na lista de tokens
#7	Guarda o token nome na lista de tokens
#8	Guarda o token alfa na lista de tokens
#9	Guarda o token numero na lista de tokens
#10	Guarda o token espaco na lista de tokens
#11	Guarda o token traco na lista de tokens
#12	Guarda o token aspas na lista de tokens
#13	Guarda o token simbolo na lista de tokens
#14	Guarda o token conchete na lista de tokens
#15	Guarda o token matem na lista de tokens
#16	Guarda o token separador na lista de tokens
#17	Guarda o token pontuacao na lista de tokens
#18	Guarda o token simb_varios na lista de tokens
#19	Guarda o token nono na lista de tokens
#20	Guarda o token ponto na lista de tokens
#21	Extrai da lista de tokens o nome completo do item que está sendo analisado e adiciona o item na lista de siglas em memória

Quadro 21 - Ações semânticas realizadas na análise de itens de siglas

No Quadro 22 é demonstrada uma gramática construída na ferramenta GALS para fazer a análise léxica e sintática nos itens de uma lista de símbolos. Estão representadas também as ações semânticas desta gramática.

```
# Definições Regulares
d        : [0-9]
lmai     : [A-Z Á-Ú À-Ù Â-Û Ã-Õ Ä-Ü Ç]
lmin     : [a-z á-ú à-ù â-û ã-õ ä-ü ç]
nonchar: [^abcdefghijklmnopqrstuvwyxzABCDEFGHIJKLMNOPQRSTUVWYXZ]
```

```
# Tokens
// simbolos
char_mai        : {lmai}
nome_mai        : {lmai}+
nome            : {lmai} {lmin}+
alfa            : {lmin}+
numero : {d}+

// simbolos
espaco          : " "
traco           : "-"
ponto           : "."
aspas           : \" | "'"
simbolo: "@" | "#" | "$" | "%" | "&"
conchete        : "(" | ")" | "{" | "}" | "[" | "]" | "<" | ">"
matem           : \+ | "*" | "=" | "/"
separador       : "_" | "\"
pontuacao       : "!" | "?" | ";" | ":" | ","
simb_varios     : "¹" | "²" | "³" | "£" | "¢" | "¬" | "§" | "º" | "ª"
nono            : {nonchar}
```

```
# Gramática
<simbolo>           ::= <desc_simb> espaco #2 traco #3 espaco #4 <desc> #21;
<desc_simb>         ::= numero #1;
<desc>              ::= <escolhe_todos> <varios>;
<escolhe_todos>     ::= char_mai #5 | nome_mai #6
                        | nome #7 | alfa #8
                        | numero #9 | espaco #10
                        | traco #11 | aspas #12
                        | simbolo #13 | conchete #14
                        | matem #15 | separador #16
                        | pontuacao #17 | simb_varios #18
                        | nono #19 | ponto #20;
<varios>            ::= <escolhe_todos> <varios> | ε ;
```

Quadro 22 – BNF - análise de itens da lista de símbolos e as ações semânticas

O Quadro 23 mostra as ações semânticas e suas respectivas ações tomadas pelo protótipo durante a análise da lista de símbolos.

Número	Ação
#1	Guarda o token numero na lista de tokens e extrai da lista o numero do símbolo do item que está sendo analisado
#2	Guarda o token espaco na lista de tokens
#3	Guarda o token traco na lista de tokens
#4	Guarda o token espaco_mai na lista de tokens
#5	Guarda o token char_mai na lista de tokens
#6	Guarda o token nome_mai na lista de tokens
#7	Guarda o token nome na lista de tokens
#8	Guarda o token alfa na lista de tokens
#9	Guarda o token numero na lista de tokens
#10	Guarda o token espaco na lista de tokens
#11	Guarda o token traco na lista de tokens
#12	Guarda o token aspas na lista de tokens
#13	Guarda o token simbolo na lista de tokens
#14	Guarda o token conchete na lista de tokens
#15	Guarda o token matem na lista de tokens
#16	Guarda o token separador na lista de tokens
#17	Guarda o token pontuacao na lista de tokens
#18	Guarda o token simb_varios na lista de tokens
#19	Guarda o token nono na lista de tokens
#20	Guarda o token ponto na lista de tokens
#21	Extrai da lista o nome completo do item que está sendo analisado e adiciona o item na lista de símbolos em memória

Quadro 23 - Ações semânticas realizadas na análise de itens de símbolos

No Quadro 24 está representado a gramática construída na ferramenta GALS utilizada para fazer a análise léxica e sintática dos itens de um sumário. Juntos estão representados as ações semânticas da gramática.

```
# Definições Regulares
d         : [0-9]
lmai      : [A-Z Á-Ú À-Ù Â-Û Ã-Õ Ä-Ü Ç]
lmin      : [a-z á-ú à-ù â-û ã-õ ä-ü ç]
nonchar: [^abcdefghijklmnopqrstuvwyxzABCDEFGHIJKLMNOPQRSTUVWYXZ]
```

```
# Tokens
// sumario
ref               : "REFERÊNCIAS BIBLIOGRÁFICAS"
ap                : "APÊNDICE"
an                : "ANEXO"
                                          // simbolos
char_mai          : {lmai}                espaco          : " "
nome_mai          : {lmai}+               traco           : "-"
nome              : {lmai} {lmin}+        ponto           : "."
alfa              : {lmin}+               varios_pontos   : "." "."+
numero            : {d}+                  aspas           : \" | "'"

simbolo: "@" | "#" | "$" | "%" | "&"
conchete          : "(" | ")" | "{" | "}" | "[" | "]" | "<" | ">"
matem             : \+ | "*" | "=" | "/"
separador         : "_" | "\"
pontuacao         : "!" | "?" | ";" | ":" | ","
simb_varios       : "¹" | "²" | "³" | "£" | "¢" | "¬" | "§" | "º" | "ª"
nono              : {nonchar}
```

```
# Gramática
<sumario>                   ::= <numeracao_sum> <desc> espaco #24 numero #25
                                | <referencia>
                                | <anexo>
                                | <apendice>;
<numeracao_sum>             ::= numero #1 <ponto_ou_espaco>;
<ponto_ou_espaco>           ::= ponto #2 numero #3 <ponto_ou_espaco>
                                | espaco #4 ;
<desc>                      ::= <escolhe_todos> <varios>;
<escolhe_todos>             ::= char_mai #5 | nome_mai #6
                                | nome #7 | alfa #8
                                | numero #9 | espaco #10
                                | traco #11 | aspas #12
                                | simbolo #13 | conchete #14
                                | matem #15 | separador #16
                                | pontuacao #17 | simb_varios #18
                                | nono #19;
<varios>                    ::= ponto #20 espaco #21 varios_pontos #22
                                | <escolhe_todos> <varios>
                                | varios_pontos #23;
<referencia>                ::= ref <escolhe_ref> espaco numero #26 ;
<escolhe_ref>               ::= espaco varios_pontos
                                | ponto espaco varios_pontos
                                | varios_pontos;
<anexo>                     ::= an #27 espaco #28 char_mai #29 espaco #30 traco #31
                                espaco #32 <desc> espaco #33 numero #34 ;
<apendice>                  ::= ap #35 espaco #36 char_mai #37 espaco #38 traco #39
                                espaco #40 <desc> espaco #41 numero #42 ;
```

Quadro 24 – BNF - análise de itens do sumário e as ações semânticas

O Quadro 25 mostra as ações semânticas e suas respectivas ações tomadas pelo

protótipo durante a análise da lista de símbolos.

Número	Ação
#1	Guarda o token numero na lista de tokens
#2	Guarda o token ponto na lista de tokens
#3	Guarda o token numero na lista de tokens
#4	Guarda o token espaco na lista de tokens e extrai da lista de tokens a numeração do item que está sendo analisado
#5	Guarda o token char_mai na lista de tokens
#6	Guarda o token nome_mai na lista de tokens
#7	Guarda o token nome na lista de tokens
#8	Guarda o token alfa na lista de tokens
#9	Guarda o token numero na lista de tokens
#10	Guarda o token espaco na lista de tokens
#11	Guarda o token traco na lista de tokens
#12	Guarda o token aspas na lista de tokens
#13	Guarda o token simbolo na lista de tokens
#14	Guarda o token conchete na lista de tokens
#15	Guarda o token matem na lista de tokens
#16	Guarda o token separador na lista de tokens
#17	Guarda o token pontuacao na lista de tokens
#18	Guarda o token simb_varios na lista de tokens
#19	Guarda o token nono na lista de tokens
#20	Guarda o token ponto na lista de tokens e extrai da lista de tokens o nome do item que está sendo analisado
#21	Guarda o token espaco na lista de tokens
#22	Guarda o token vários_pontos na lista de tokens
#23	Guarda o token vários_pontos na lista de tokens e extrai da lista de tokens o nome do item que está sendo analisado
#24	Guarda o token espaco na lista de tokens
#25	Guarda o token numero na lista de tokens e extrai da lista de tokens o nome completo e a pagina do item que está sendo analisado e adiciona este item na lista de sumários em memória
#26	Guarda o token numero na lista de tokens e extrai da lista de tokens o nome completo e a página do item que está sendo analisado (REFERÊNCIA) e adiciona este item na lista de sumários em memória Guarda o token an na lista de tokens Guarda o token espaco na lista de tokens
#27	Guarda o token char_mai na lista de tokens
#28	Guarda o token espaco na lista de tokens
#29	Guarda o token traco na lista de tokens
#30	Guarda o token espaco na lista de tokens
#31	Guarda o token espaco na lista de tokens
#32	Guarda o token numero na lista de tokens tokens e extrai da lista de tokens o nome completo e a
#33	página do item que está sendo analisado (ANEXO) e adiciona este item na lista de sumários em
#34	memória Guarda o token ap na lista de tokens Guarda o token espaco na lista de tokens
#35	Guarda o token char_mai na lista de tokens
#36	Guarda o token espaco na lista de tokens
#37	Guarda o token traco na lista de tokens
#38	Guarda o token espaco na lista de tokens
#39	Guarda o token espaco na lista de tokens
#40	Guarda o token numero na lista de tokens tokens e extrai da lista de tokens o nome completo e a
#41	página do item que está sendo analisado (APÊNDICE) e adiciona este item na lista de sumários em
#42	memória

Quadro 25 - Ações semânticas realizadas na análise de itens de sumário