

**UNIVERSIDADE LUSÓFONA DE HUMANIDADES E TECNOLOGIAS**  
**Licenciatura em Informática**  
**Projecto Final de Curso**

**Construção de uma aplicação de *software*  
para geração de manuais de referência**

**Docente** Prof. António Frazão

**Discentes** Alexandre Mesquita  
Hugo Santos

Lisboa, 16 de Novembro de 2007

*À Susana e à Zélia*

Alexandre Mesquita

*À Susana e à Talika*

Hugo Santos

## **Agradecimentos**

Os discentes gostariam de agradecer ao seu coordenador, Prof. António Frazão, e às seguintes pessoas que contribuíram para a realização deste Projecto Final de Curso:

Adelaide Trabuco, Carlos António, Prof. Ernesto Morgado, Fernando Martinho, João Pavão, Prof. João Pavão Martins, João Távora, Rui Aurélio, Tiago Loureiro, Tiago Maduro Dias e Tom Weissman.

## Resumo

A geração de manuais de referência é grande importância para a SISCOG, uma empresa de desenvolvimento de *software*. Desde cedo que a documentação interna dos seus produtos é produzida com o propósito de obter manuais de referência de forma automática, por forma a disponibilizar uma *interface* clara aos técnicos utilizadores da sua tecnologia. Para guiar a extracção de dados e a sua publicação num formato de fácil leitura, a documentação interna está escrita de acordo com um conjunto de regras pré-definido. O objectivo deste trabalho é a construção de um programa que realize a geração dessa documentação de referência, no âmbito de um Projecto Final de Curso na área de Informática. A abordagem para solucionar este problema consiste em estudar as tecnologias existentes e escolher a que se ajusta melhor às necessidades do programa. Para efeito de aprendizagem é consultada bibliografia de referência e são utilizados alguns pacotes de software *open source* que possam dar suporte à resolução do problema. O resultado do trabalho efectuado é um programa com dois módulos que comunicam entre si. Um módulo para extrair informação da documentação interna e assinalar erros, caso existam. Um outro módulo para traduzir para um formato adequado a informação processada pelo primeiro módulo. Para o primeiro módulo são utilizados os conceitos da teoria de processadores de linguagens e para o segundo módulo são utilizadas as tecnologias XML, XSL, DOM e OLE. O programa construído está em condições de poder ser introduzido no processo de desenvolvimento de *software* da SISCOG. Embora concebido para um conjunto de requisitos em particular, pode adaptar-se facilmente às crescentes necessidades da empresa.

## **Abstract**

The generation of reference manuals is of high importance for SISCOG, a software development company. Since early days the internal documentation of its products is produced with the intention of obtaining reference manuals in an automatic way, in order to provide a simple interface to the users of its technology. The internal documentation is written according to a set of predefined rules that can be use as a guide to extract the relevant data and publish it in an easy to read format. The goal of this work, which is performed in the scope of a final year project of a university degree in the area of informatics, is to build a program that acts as a generator of that reference documentation. The approach to solve this problem is to study existing technologies and choose the one that fits best the program's needs. For learning purposes it is consulted the literature of reference and it is used some software packages in order to support the solving of the problem. The result of the work done is a program with two modules communicating with each other: a module for extracting the internal documentation and signalling any existing errors, and another module to translate to a proper format the information that was processed by the first module. The first module uses the principles and techniques of compiling. The second module uses the following technologies: XML, XSL, DOM and OLE. The program is able to fulfil its purpose and therefore can be integrated into SISCOG's software development process. Although it was designed to be in compliance to a particular set of requirements it can be easily adapted to the meet SISCOG's growing needs.

## Índice

1	Introdução .....	7
2	Desenvolvimento .....	8
2.1	Módulo de Extracção de dados.....	8
2.1.1	Enquadramento Teórico.....	9
2.1.1.1	Processadores de Linguagens.....	9
2.1.1.2	A Linguagem Fonte.....	10
2.1.1.3	A Análise do Texto Fonte .....	10
2.1.2	Concepção.....	12
2.1.2.1	Definição do Texto Fonte.....	12
2.1.2.2	Definição da Linguagem Fonte .....	14
2.1.2.3	Definição do Processador.....	22
2.1.2.4	Definição do Texto Alvo .....	24
2.1.2.5	Conceito de Execução do Módulo.....	25
2.2	Módulo de Exportação de Dados .....	27
2.2.1	Tecnologias .....	28
2.2.1.1	Document Object Model .....	28
2.2.1.2	Extensible Markup Language (XML) .....	28
2.2.1.3	XML Schema (XSD).....	29
2.2.1.4	Extensible Stylesheet Language Family (XSL) .....	29
2.2.1.5	Object Linking and Embedding (OLE) .....	29
2.2.2	Concepção.....	29
2.2.2.1	Definição do Módulo.....	29
2.2.2.2	Conceito de Execução .....	30
3	Conclusão.....	32
	Referências bibliográficas.....	33
	Anexo 1 – Especificação de Requisitos de Software.....	34
	Anexo 2 – Caso de Teste .....	50

# 1 Introdução

A SISCOG – Sistemas Cognitivos, Lda, é uma empresa de desenvolvimento de *software* que, no âmbito da sua actividade, realiza documentação técnica relativa ao código fonte dos seus sistemas. Esta documentação interna é redigida pelo programador com o apoio de uma ferramenta de apoio à documentação existente no ambiente de desenvolvimento da empresa, que produz texto de forma automática com base em *templates*. A documentação que é produzida, tanto pelos *templates* como a que é adicionada manualmente pelo programador, deve estar estruturada de acordo com um conjunto de regras<sup>1</sup>, que têm como objectivo a extracção da documentação para posterior geração de manuais de referência.

Assim, o tema escolhido para este trabalho resultou da necessidade de *a)* extrair documentação interna existente nos ficheiros de código fonte e *b)* publicá-la num formato de documentação de fácil consulta e leitura.

A construção de um programa de *software* para satisfazer esta necessidade, o *gerador de manuais de referência*, deve pois usar as actuais regras de estruturação de escrita para saber de que forma irá extrair informação útil para o manual de referência. Se existir documentação que não esteja em conformidade com essas regras então o programa não irá extrair a informação correcta. Nestas condições o programa deve ser capaz de cumprir outro dos seus objectivos que é o de *c)* assinalar se existem erros em determinada documentação interna.

Das três principais tarefas que o programa deve executar, decorre dos requisitos de *software* que o programa deve ser dividido em dois módulos que comunicam entre si. Um módulo para extrair informação da documentação interna e assinalar erros, caso existam. Um outro módulo para traduzir para um formato adequado a informação processada pelo primeiro módulo (ver Figura 1).

---

<sup>1</sup> Ver secção 1.3. Regras do Anexo I – Especificação de Requisitos de Software.

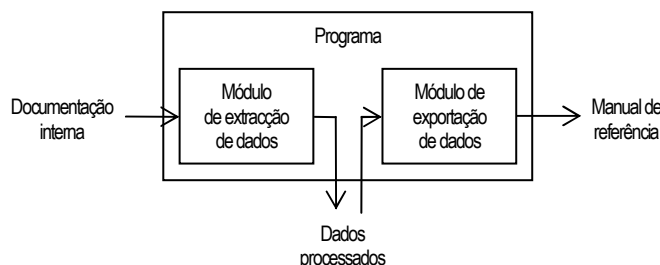


Figura 1: Principais módulos do programa

O principal desafio que se colocou para a construção deste programa foi a definição da arquitectura de cada um dos módulos. Para tal, tomou-se a decisão de fazer primeiramente um trabalho de investigação sobre tecnologias existentes, e escolher depois a que melhor se ajustasse às necessidades do programa.

A documentação produzida na realização deste Trabalho Final de Curso está dividida em quatro partes: um relatório (o actual documento), que descreve o trabalho efectuado pelos discentes, dois anexos, um para a especificação de requisitos e um outro para a descrição de um caso de teste utilizado para testar o programa e, finalmente, um resumo do trabalho em Português e em Inglês.

O relatório introduz o problema proposto, descreve a metodologia e as tecnologias usadas para solucionar o problema e termina com uma conclusão sobre os resultados obtidos.

O anexo *Especificação de Requisitos de Software* lista os requisitos entregues à Coordenação do Trabalho Final de Curso aquando do início do projecto, mais os requisitos que resultaram da análise às regras de escrita da documentação interna da SISCOG.

O anexo *Caso de Teste* descreve o conjunto de dados inicial, e os resultados previstos e observados do teste efectuado.

## 2 Desenvolvimento

### 2.1 Módulo de Extracção de dados

Para o módulo de extracção de dados da documentação interna, cuja estrutura é ilustrada na Figura 2, a pesquisa centrou-se na teoria de Processadores de Linguagens.



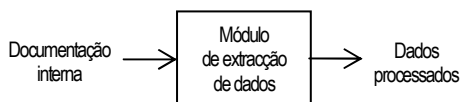


Figura 2: Módulo de extracção de dados

Os discentes adquiriram conhecimento sobre esta teoria através da consulta de livros universitários e através da utilização de programas *open source* especialmente concebidos para o processamento de linguagens. Os livros e os programas complementaram-se, tendo fornecido uma excelente forma de exercício de aprendizagem dos conceitos. A obra que serviu de inspiração e que teve sem dúvida o maior peso neste processo foi a de Aho, Sethi e Ullman (1988), conhecida no meio universitário como o “livro do dragão”. Tanto esta obra como a de Crespo (2001) foram seguidas para definir os conceitos e a estrutura da actual secção.

Durante o desenvolvimento deste módulo, alguns dos programas usados para a aprendizagem acabaram por ser integrados no mesmo: os pacotes CL-YACC, LEXER, REGEX e MK-defsystem, e um protótipo desenvolvido anteriormente na SISCOG. Para todos eles está disponível o código fonte, sendo o Lisp a linguagem na qual estão codificados.

## 2.1.1 Enquadramento Teórico

Esta secção introduz a teoria de processamento de linguagens, que será usada na secção 2.1.2 *Concepção* para descrever o desenvolvimento do módulo de extracção de dados.

### 2.1.1.1 Processadores de Linguagens

Um processador de uma linguagem é um programa que lê um texto escrito numa linguagem – a linguagem *fonte* – e a traduz para um texto equivalente numa outra linguagem – a linguagem *alvo* (ver Figura 3). No processo de tradução, o processador deve relatar ao seu utilizador a presença de eventuais erros no texto fonte.

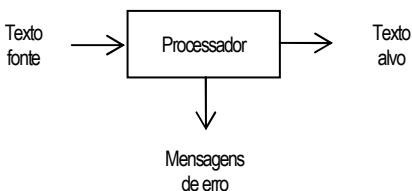


Figura 3: Um processador de uma linguagem

Existem diversos tipos de processadores de linguagens, tais como os interpretadores, compiladores, *assemblers*, tradutores em geral, processadores de documentos, entre outros. Como o objectivo do programa a desenvolver é o de extrair documentação interna de ficheiros de código fonte, fazendo também o reconhecimento de marcas de formatação e de organização de texto, como os parágrafos, as listas ordenadas, o texto indentado, as referências cruzadas, etc., o processador a usar neste módulo é categorizado como um tipo de *processador de documentos*.

Dependendo do tipo de processador, encontramos uma grande variedade de linguagens fonte. No processador utilizado neste módulo a linguagem fonte é definida a partir das regras de escrita de documentação interna.

### **2.1.1.2 A Linguagem Fonte**

A linguagem na qual o texto fonte está escrito constitui uma *linguagem sobre um vocabulário* e define-se como um conjunto de frases em que cada frase é uma sequência finita de *símbolos terminais (tokens)* pertencentes a esse vocabulário.

Para especificar uma linguagem é pois necessário definir quais são os símbolos do vocabulário, indicar as regras que estabelecem as combinações possíveis de símbolos, isto é, as combinações que dão origem a frases válidas e, por fim, verificar se essas frases têm um significado que faça sentido.

Os símbolos de um vocabulário, em número finito, são definidos através da enumeração dos mesmos.

A estrutura das frases é definida por meio de um conjunto de *regras sintáticas*. As frases de uma linguagem são definidas através de uma propriedade comum e não por enumeração explícita.

As *regras semânticas* definem em que condições as frases sintacticamente correctas devem poder ser interpretadas.

A linguagem fonte usada no módulo de extracção de dados será descrita na secção 2.1.2.2 *Definição da Linguagem Fonte*.

### **2.1.1.3 A Análise do Texto Fonte**

Os processadores de linguagens, em particular os compiladores, são usualmente caracterizados por dois grandes componentes: um de *análise* e outro de *síntese*. O componente de análise faz o

*reconhecimento* do significado do texto fonte, dividindo-o em partes mais pequenas e criando uma representação intermédia do mesmo. O componente de síntese *reage* ao significado identificado na representação intermédia, produzindo o texto alvo pretendido.

O processador utilizado neste módulo é mais simplificado e não utiliza um componente de síntese. Assim, o texto alvo deste módulo corresponde a uma representação intermédia do texto fonte, isto é corresponde ao *output* do componente de análise (ver Figura 4). Uma vez validado o significado do texto fonte o módulo de análise pode então “reagir” e gerar a representação intermédia do texto fonte.

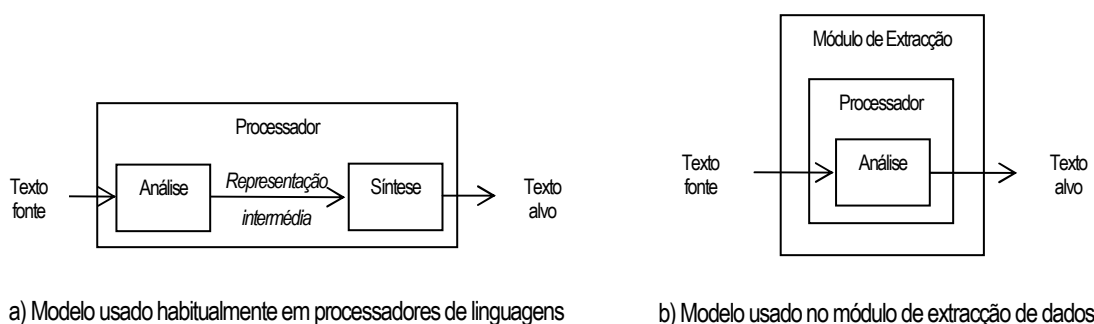


Figura 4: O Modelo Análise-Síntese de Processamento

A análise realizada pelo processador procura reconhecer se o texto fonte, que neste contexto é designado por *frase de entrada*, pertence ou não à linguagem, isto é, verifica se corresponde ou não a uma frase da linguagem.

No módulo de extração de dados a análise é dividida em quatro sub-fases: *análise léxica*, *análise sintáctica*, *análise semântica* e *geração de texto intermédio* (ver Figura 5).

Na análise léxica ou *scanning*, o fluxo de caracteres da frase de entrada é lido da esquerda para a direita e agrupado em símbolos terminais da linguagem fonte. Se não for possível agrupar os caracteres em símbolos terminais então estaremos na presença de um *erro léxico*.

Na análise sintáctica ou *parsing*, os símbolos terminais identificados na análise léxica são agrupados hierarquicamente em símbolos terminais e *símbolos não terminais* da linguagem fonte. Se a sequência de símbolos fornecida não estiver de acordo com as regras sintácticas definidoras da estrutura hierárquica então deve ser assinalado um *erro sintáctico*.

Na análise semântica, a frase de entrada que foi reconhecida na análise sintáctica como sendo uma frase pertencente à linguagem fonte, é analisada com o objectivo de saber se tem significado

no contexto da linguagem. No caso de a frase não fazer sentido, deve ser assinalado um *erro semântico*. Se frase tiver um significado correcto o gerador de texto intermédio poderá então fazer uma *tradução* para o texto alvo.

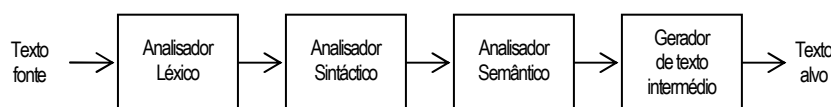


Figura 5: Sub-tarefas do componente de análise usado no módulo de extracção de dados

O tratamento dos erros efectuado nas três primeiras fases é normalmente dividido em *detecção do erro* e *recuperação do erro*. No caso de ser detectado um erro, o processamento é retomado se o processador tiver uma estratégia para recuperar do erro, caso contrário o processamento é interrompido e, conseqüentemente, o texto alvo não será produzido.

## 2.1.2 Concepção

Esta secção descreve o desenvolvimento efectuado para a produção do módulo de extracção de dados. Os conceitos introduzidos na secção anterior são ilustrados com exemplos e, em simultâneo, são definidos os principais elementos do módulo: o texto fonte que constitui a documentação interna existente nos ficheiros de código fonte, uma linguagem para o processamento desse texto fonte, os vários componentes constituintes do processador, o texto alvo produzido pelo processador e, finalmente, a ordem de execução das principais operações realizadas pelo módulo.

### 2.1.2.1 Definição do Texto Fonte

Na SISCOG, um ficheiro de código fonte é composto por código Lisp e por documentação relacionada com esse código fonte (ver exemplo na Figura 6). Um ficheiro de código fonte é iniciado com a documentação sobre o ficheiro, delimitada por marcas de comentário Lisp, e é designada por cabeçalho de ficheiro. É seguida por zero ou mais elementos de código fonte com a respectiva documentação, também delimitada por marcas de comentário Lisp, colocada imediatamente antes do respectivo elemento. No exemplo indicado na Figura 6, o ficheiro de código fonte contém um cabeçalho de ficheiro, seguido de uma instrução Lisp, seguido do

cabeçalho do elemento de código fonte «fn1», seguido da definição do elemento de código fonte «fn1».

```

;;;-----
;;;
;;;      Copyright (C) 1995, SISCOG - Sistemas Cognitivos Lda.
;;;      All rights reserved
;;;
;;;-----
;;;Description
;;;      This file contains the definition of some functions.
;;;
;;;      It is used as an example for some \emph{explanations}.
;;;
;;;History
;;;      Date          Author          Description
;;;      98/04/01      Silva           Created
;;;-----

(in-package :test-package)

;;;-----
;;;FN1
;;;Description
;;;      Calculate the value of \emph{x+2y}.
;;;
;;;      \visibility
;;;      SI
;;;
;;;      \args
;;;      \arg{x} is a \emph{number}.
;;;
;;;      \arg{y} is a \emph{number}.
;;;
;;;      \return-types
;;;      A \emph{number}.
;;;
;;;History
;;;      Date          Author          Description
;;;      99/05/10      Helena          Created
;;;-----
(defun fn1 (x y)
  (+ x (* 2 y)))

```

Figura 6: Exemplo do conteúdo de um ficheiro de código fonte

Qual é então o texto fonte que constitui o *input* para o processador? Para responder a esta pergunta é necessário recordar o objectivo principal do programa: a geração de um manual de referência. Este manual de referência deve conter a informação sobre cada ficheiro e sobre cada elemento de código fonte, informação essa que é classificada em atributos<sup>2</sup>. Existem atributos que podem ser obtidos a partir da documentação registada explicitamente nos cabeçalhos, aos quais atribuiu-se a designação de *atributos directos*, e existem os chamados *atributos derivados*, que não são obtidos a partir do texto dos cabeçalhos. Tome-se como exemplo o elemento de código fonte «fn1» indicado na Figura 6. Os valores dos seus atributos directos estão registados no seu cabeçalho: o atributo «Nome» tem o valor «FN1», o atributo «Description» tem o valor

<sup>2</sup> Ver secção 1.3. Regras do Anexo I – Especificação de Requisitos de Software.

«Calculate the value of  $\text{\emph{x+2y}}$ », o atributo «Visibility» tem o valor «SI», etc. Dois dos seus atributos derivados são o «Source» e o «Package». O primeiro corresponde ao nome do ficheiro e à sua localização no sistema de ficheiros. O segundo corresponde ao valor «:test-package» na expressão «(in-package :test-package)».

O texto fonte que deve ser processado é, portanto, todo aquele que permita identificar os valores dos atributos de cada elemento ou ficheiro de código fonte<sup>3</sup>. Note-se que este «texto fonte» deve ser entendido num sentido mais lato pois pode incluir alguma informação que não esteja escrita explicitamente num ficheiro de código fonte como é o caso dos valores para alguns atributos derivados.

### 2.1.2.2 Definição da Linguagem Fonte

Conforme foi referido na secção 1 *Introdução*, a caracterização dos ficheiros e dos elementos de código fonte e o preenchimento dos seus cabeçalhos fonte estão sujeitos a um conjunto de regras definidas pela SISCOG.

Quanto aos ficheiros de código fonte, são caracterizados por atributos. Os atributos têm um nome, um valor, obrigatório ou não, e uma palavra-chave que os identifica no contexto do cabeçalho do ficheiro. Ainda sobre o valor, a maior parte dos atributos directos corresponde a texto livre, que pode e deve ser formatado (de acordo com outro conjunto de regras bem definido) quer para aumentar a sua legibilidade, quer para adicionar significado ao mesmo. A estrutura de um cabeçalho obedece também a um conjunto de regras de preenchimento. O texto deve ser escrito usando determinadas palavras-chave, numa determinada sequência, etc.

Quanto aos elementos de código fonte, as regras assemelham-se às dos ficheiros de código fonte. As principais diferenças localizam-se nos tipos de atributos, que são em maior número, e ao preenchimento dos respectivos cabeçalhos. Relativamente às regras de formatação para os valores dos atributos, aplicam-se as mesmas dos ficheiros de código fonte.

No âmbito do *design* do módulo decidiu-se agrupar as particularidades de cada uma destas realidades e dar-lhes um nome por forma a poder-se fazer uma fácil correspondência com uma determinada linguagem fonte. Assim, foram definidos um *modelo de caracterização do ficheiro*

---

<sup>3</sup> A secção 2.1.2.2 *Definição da Linguagem Fonte* mostra um exemplo de um texto fonte no contexto da definição de uma linguagem fonte.

de código fonte (MC-F), um modelo de caracterização do elemento de código fonte (MC-E), e um modelo de formatação de texto (MF-T).

Este processo deu origem à definição de três linguagens fonte. O MC-F foi relacionado com a linguagem de documentação interna do ficheiro de código fonte (LDI-F). Com o MC-E foi feita a correspondência com a linguagem de documentação interna do elemento de código fonte (LDI-E). O MF-T foi relacionado a linguagem de formatação interna (LFI).

O exemplo da Figura 7 ilustra a correspondência entre o texto existente num ficheiro de código fonte e a linguagem relativamente à qual seria feito o processamento desse texto.

```

;;;-----
;;;
;;;      Copyright (C) 1995, SISCOG - Sistemas Cognitivos Lda.
;;;      All rights reserved
;;;
;;;-----
;;;Description
;;;      This file contains the definition of some functions.
;;;      It is used as an example for some \emph{explanations}.
;;;
;;;History
;;;      Date      Author      Description
;;;      98/04/01   Silva      Created
;;;
;;;-----
(in-package :test-package)

;;;-----
;;;FN1
;;;Description
;;;      Calculate the value of \emph{x+2y}.
;;;      \visibility
;;;      SI
;;;      \args
;;;      \arg{x} is a \emph{number}.
;;;      \arg{y} is a \emph{number}.
;;;      \return-types
;;;      A \emph{number}.
;;;History
;;;      Date      Author      Description
;;;      99/05/10   Helena      Created
;;;
;;;-----
(defun fn1 (x y)
  (+ x (* 2 y)))

```

Figura 7: Exemplo do conteúdo de um ficheiro de código fonte

A existência de três linguagens distintas tem como consequência imediata a necessidade de definir três processadores, um para cada linguagem, para serem executados pelo módulo de

extracção de dados<sup>4</sup>, sendo que a ordem de execução de cada um depende apenas do texto fonte que esteja a ser considerado para processamento num dado momento.

Se tomarmos como exemplo a LDI-F, o texto fonte corresponde à parte do cabeçalho de um ficheiro de código fonte que é relevante para o processamento. Considerando novamente o texto mostrado na Figura 6, o cabeçalho do ficheiro corresponde ao primeiro conjunto de linhas, consecutivas, que estão limitadas à esquerda por marcas de comentário Lisp<sup>5</sup> (ver Figura 8). Desse texto, apenas uma parte constitui a frase de entrada para o processador, a qual é indicada na Figura 9.

```
;;;-----  
;;;  
;;;      Copyright (C) 1995, SISCOG - Sistemas Cognitivos Lda.  
;;;      All rights reserved  
;;;  
;;;-----  
;;;Description  
;;;      This file contains the definition of some functions.  
;;;  
;;;      It is used as an example for some \emph{explanations}.  
;;;  
;;;History  
;;;      Date      Author      Description  
;;;      98/04/01  Silva      Created  
;;;-----
```

Figura 8: Exemplo de um cabeçalho de ficheiro de código fonte

```
;;;Description  
;;;      This file contains the definition of some functions.  
;;;  
;;;      It is used as an example for some \emph{explanations}.  
;;;  
;;;History  
;;;      Date      Author      Description  
;;;      98/04/01  Silva      Created
```

Figura 9: Exemplo de um texto fonte para o processador da LDI-F

Embora não seja visível na Figura 8, o texto contém caracteres de mudança de linha, no final de cada linha, e caracteres de tabulação entre algumas palavras. O texto da Figura 9 já não contém caracteres de quebra de linha pois são suprimidos pelo processador durante a leitura do texto. No entanto, é mantido o mesmo efeito visual na figura para não se perder a legibilidade do texto.

## Vocabulário

Nesta secção regressamos à especificação de um dos componentes necessários à definição de uma linguagem fonte: o seu vocabulário. Por forma a manter a simplicidade da descrição, e pelo facto

<sup>4</sup> A ordem de execução de cada processador é ilustrada na secção 2.1.2.5 *Conceito de Execução* do Módulo.

<sup>5</sup> Ver *requisito de software 11* no Anexo 1 – *Especificação de Requisitos de Software*.



de ter um processo de definição análogo ao das outras duas linguagens, usaremos apenas a LDI-F como referencial para desenvolver o tema da definição de uma linguagem.

A escolha sobre quais devem ser os símbolos terminais da LDI-F depende de vários factores. Enumeram-se alguns: por um lado, pode influenciar ou ser influenciada pela definição da sintaxe da linguagem; por outro lado, depende do modo como processador irá fazer o reconhecimento desses símbolos no texto fonte<sup>6</sup>.

No que diz respeito ao texto fonte que é lido pelo processador, observa-se que nem todos os caracteres têm utilidade para se conseguir extrair o valor de determinado atributo do ficheiro fonte. É o caso das marcas de comentário Lisp (o ponto e vírgula) ou a eventual presença de tabulações ou espaços em branco duplicados. Torna-se pois necessário introduzir um nível de abstracção para o reconhecimento dos símbolos terminais, em que os caracteres supérfluos são suprimidos.

A Tabela 1 lista os símbolos terminais da LDI-F que podem ser encontrados no exemplo da Figura 9 e a sua correspondência com as sequências de caracteres que compõem a frase de entrada para o processador. Recorde-se que, no texto indicado na Figura 9, já não existem caracteres de mudança de linha.

Símbolo terminal	Sequência de caracteres do texto fonte
:DESCRIPTION	;;;Description
:DOC-TEXT	;;; This file contains the definition of some functions. ;;; ;;; It is used as an example for some \emph{explanations}. ;;;
:HISTORY	;;;History
:HISTORY-HEADER	;;; Date Author Description
:DOC-TEXT	;;; 98/04/01 Silva Created

Tabela 1: Exemplo de símbolos terminais da LDI-F

O texto fonte indicado na Figura 9 será então representado no contexto da LDI-F conforme se indica na Figura 10, depois de reconhecidos os símbolos terminais.

<sup>6</sup> O reconhecimento dos símbolos é realizado pelo analisador léxico do processador, que será apresentado na secção *Analisador Léxico*.

:DESCRIPTION :DOC-TEXT :HISTORY :HISTORY-HEADER :DOC-TEXT

Figura 10: Exemplo de uma frase de entrada para o processador da LDI-F após reconhecidos os símbolos terminais

Na Tabela 1 observa-se que o símbolo terminal `:DESCRIPTION` da LDI-F corresponde, no texto fonte, à cadeia de caracteres `;;;Description`. No entanto, o que aconteceria se, no texto fonte, existisse a cadeia de caracteres `;;; Description` ou até mesmo `;;;Description`”? Há pois a necessidade de se poder alargar o tipo de correspondência a uma sequência padrão que inclua este género de situações. Tal padrão poderia ser descrito da seguinte forma: «Três caracteres “;”, seguidos de zero ou um carácter “espaço” ou “tabulação”, seguidos de “Description”, seguido de zero ou mais caracteres “espaço” ou “tabulação”». A notação escolhida para descrever tais padrões no vocabulário da LDI-F é a *expressão regular* (ver terceira coluna da Tabela 2), um formalismo normalmente utilizado para descrever *linguagens regulares*.

Símbolo terminal	Sequência de caracteres do texto fonte	Expressão regular
:DESCRIPTION	<code>;;;Description</code>	<code>:::[[:space:]]?Description[[:space:]]*</code>
:DOC-TEXT	<code>;;; This file contains the definition of some functions. ;;; ;;; It is used as an example for some \emph{explanations}. ;;;</code>	Zero ou mais instâncias de qualquer um dos seguintes padrões:  <code>:::[[:space:]]*</code> <code>:::[[:space:]]*.*</code>
:HISTORY	<code>;;;History</code>	<code>:::[[:space:]]?History[[:space:]]*</code>
:HISTORY-HEADER	<code>;;; Date Author Description</code>	<code>:::[[:space:]]+Date[[:space:]]+Author[[:space:]]+Description[[:space:]]*</code>
:DOC-TEXT	<code>;;; 98/04/01 Silva Created</code>	Zero ou mais instâncias de qualquer um dos seguintes padrões:  <code>:::[[:space:]]*</code> <code>:::[[:space:]]*.*</code>

Tabela 2: Exemplo de símbolos terminais da LDI-F

Relativamente ao texto fonte mostrado na Figura 10, foi referido que este resulta de uma abstracção do texto fonte original. Note-se, no entanto, que foram “suprimidos” detalhes que constituem informação útil para o manual de referência. É o caso do valor do atributo “Description” de um ficheiro de código fonte, que é extraído, como veremos na secção *Semântica*, da sequência que sucede a “;;;Description” no texto fonte (a sequência em causa corresponde ao valor indicado na segunda coluna da Tabela 1, na linha correspondente à primeira instância de :DOC-TEXT). Se esse texto é substituído pelo símbolo :DOC-TEXT no processo de abstracção, como guardar o valor do atributo “Description” do ficheiro de código fonte por forma a ser mostrado posteriormente no manual de referência?

No processamento de linguagens, é comum coleccionar a informação associada a símbolos terminais em atributos ou *propriedades*<sup>7</sup> desses símbolos. Para os símbolos das linguagens utilizadas neste módulo foram definidas três propriedades: o *símbolo* propriamente dito, o *valor*, e o *lexema*. Destas, apenas as duas primeiras são usadas no processamento.

A propriedade “valor” armazena o extracto do texto fonte que é útil para o processamento. Relativamente à propriedade “lexema”, corresponde à sequência de caracteres do texto fonte que é coincidente com o padrão associado ao símbolo. A segunda e terceira colunas da Tabela 3 indicam, respectivamente, o conteúdo das propriedades “valor” e “lexema” de cada símbolo terminal identificado no texto fonte da Figura 9.

Símbolo	Valor	Lexema
:DESCRIPTION	:DESCRIPTION	;;;Description
:DOC-TEXT	“This file contains the definition of some functions.” :EMPTY-LINE “It is used as an example for some \emph{explanations}.” :EMPTY-LINE	;;; This file contains the definition of some functions. ;;; It is used as an example for some \emph{explanations}. ;;;
:HISTORY	:HISTORY	;;;History
:HISTORY-HEADER	:HISTORY-HEADER	;;; Date Author Description
:DOC-TEXT	“98/04/01 Silva Created”	;;; 98/04/01 Silva Created

Tabela 3: Propriedades dos símbolos terminais da LDI-F

<sup>7</sup> Para manter a simplicidade da explicação, neste documento será usado o termo “propriedade” quando se estiver a fazer referência ao atributo de um símbolo terminal, e será usado o termo “atributo” quando se estiver a fazer referência ao atributo de um ficheiro de código fonte ou de um elemento de código fonte.

Em resumo, a especificação do vocabulário da LDI-F inclui a enumeração dos seus símbolos terminais, sendo feita por meio de uma definição orientada à correspondência com um determinado padrão.

## Sintaxe

Conforme foi referido na secção 2.1.1.2 *A Linguagem Fonte*, as *regras sintáticas* de uma linguagem definem a estrutura a que devem obedecer as frases dessa linguagem.

Para definir a sintaxe da LDI-F é usado um formalismo designado *gramática independente de contexto*, e que é constituída por quatro componentes:

- um conjunto finito, não vazio, de símbolos terminais;
- um conjunto finito, não vazio, de *símbolos não-terminais*;
- um símbolo não terminal designado por *símbolo inicial* da gramática;
- um conjunto de regras sintáticas em que cada regra sintáctica, designada por *produção*, consiste num símbolo não terminal, chamado *lado esquerdo da produção*, seguido da sequência de caracteres “:=”, seguida de uma sequência de símbolos terminais ou não terminais, chamada *lado direito da produção*.

A Figura 11 mostra a especificação da gramática da LDI-F em notação BNF. É apresentada a listagem completa das suas produções, com a produção para o símbolo inicial, <file-header>, a encabeçar a lista. O símbolo  $\epsilon$  denota uma frase sem quaisquer símbolos terminais e tem o nome de *frase vazia*.

```
<file-header> ::= <description> <example> <remarks> <refs> <implem-notes> <unable-to-document> <history>
<description> ::= :DESCRIPTION :DOC-TEXT
<example> ::=  $\epsilon$  | :EXAMPLE :DOC-TEXT
<remarks> ::=  $\epsilon$  | :REMARKS :DOC-TEXT
<refs> ::=  $\epsilon$  | :REFS :DOC-TEXT
<implem-notes> ::=  $\epsilon$  | :IMPLEM-NOTES :DOC-TEXT
<unable-to-document> ::=  $\epsilon$  | :UNABLE-TO-DOCUMENT :DOC-TEXT
<history> ::= :HISTORY :HISTORY-HEADER <history-entries>
<history-entries> ::=  $\epsilon$  | <history-entries> :DOC-TEXT
```

Figura 11: Regras sintáticas da LDI-F

Diz-se que a sequência de símbolos terminais de uma frase de entrada pertence à linguagem LDI-F definida pela gramática indicada na Figura 11 se, começando pelo símbolo inicial e substituir-se repetidamente um símbolo não terminal pelo lado direito de uma produção para esse símbolo terminal, se obtém a sequência de símbolos terminais da frase de entrada. A este processo de substituição dá-se nome de *derivação*.

A derivação de uma frase é usualmente representada por meio de uma *árvore de derivação* (*parse tree*). A raiz da árvore contém o símbolo inicial, as folhas da árvore contêm os símbolos terminais, e os nós interiores contêm os restantes símbolos não terminais. Tomando como exemplo a frase da Figura 10, a sua árvore de derivação é conforme se indica na Figura 12.

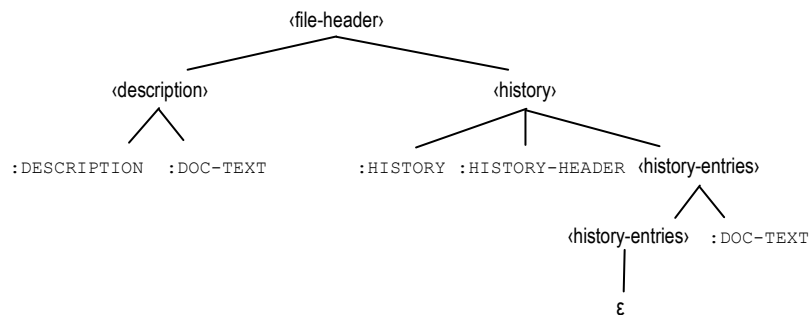


Figura 12: Árvore de derivação de uma frase de entrada

As frases de entrada que podem ser derivadas a partir do símbolo inicial formam a linguagem definida pela gramática.

## Semântica

Uma vez especificados o vocabulário e a sintaxe da LDI-F, torna-se possível estabelecer as regras que definem qual o significado das estruturas gramaticais sintacticamente correctas. Tomando como exemplo a produção para o símbolo <description>, sabe-se que o valor do atributo “Description” de um ficheiro de código fonte corresponde ao valor do símbolo terminal :DOC-TEXT que está no lado direito desta produção. É fácil verificar que esta regra semântica é sempre cumprida, pois trata-se de uma simples operação de atribuição de valores: a atribuição do valor do símbolo :DOC-TEXT ao atributo “Description” do ficheiro de código fonte.

Considere-se outro exemplo. A produção para o símbolo <history-entries>, na sua alternativa <history-entries> :DOC-TEXT, significa que, se a frase de entrada estiver de acordo com esta produção, o valor do atributo “Date” pode ser obtido dessa frase através da identificação da

última data (com o formato “aa/mm/dd”) existente no valor de :DOC-TEXT<sup>8</sup>. Neste caso, a regra semântica já impõe a condição de ter de existir no valor de :DOC-TEXT pelo menos uma data com o formato “aa/mm/dd”.

A Tabela 4 apresenta as duas regras consideradas.

Produção	Regra Semântica
<code>&lt;description&gt; ::= :DESCRIPTION :DOC-TEXT</code>	O valor do atributo “Description” do ficheiro de código fonte é dado pelo valor de :DOC-TEXT.
<code>&lt;history-entries&gt; ::= &lt;history-entries&gt; :DOC-TEXT</code>	Se o valor de :DOC-TEXT contém pelo menos uma data escrita com o formato “aa/mm/dd”, então o valor do atributo “Date” do ficheiro de código fonte é dado pela última data, com esse formato, existente no valor de :DOC-TEXT .

Tabela 4: Regras semânticas para algumas produções da LDI-F

### 2.1.2.3 Definição do Processador

Recorde-se, da secção 2.1.2.2 *Definição da Linguagem Fonte*, que foi tomada a decisão de utilizar três linguagens fonte e que, em termos de processamento, isso implica que são usados três processadores, um para cada linguagem. Todos os conceitos introduzidos até à presente secção são agora usados para se descrever a estrutura comum aos três processadores. É seguida a abordagem utilizada anteriormente, pelo que será referido apenas um dos processadores como exemplo, o processador para a LDI-F.

### Analizador Léxico

Em cada processador usado no módulo de extracção de dados, o respectivo analisador léxico é chamado pelo analisador sintáctico, que lhe pede o próximo símbolo terminal da frase de entrada. Após o pedido ser recebido, o analisador léxico lê os caracteres da frase de entrada até identificar um lexema para um símbolo terminal. Uma vez identificado, devolve ao analisador sintáctico o respectivo símbolo e valor desse símbolo terminal.

Nesta identificação de símbolos realizada pelo analisador léxico está incluída a remoção de espaços em branco ou de outros caracteres sem utilidade para o manual de referência como é o

<sup>8</sup> O valor de :DOC-TEXT é objecto de processamento no contexto da LFI, antes de ser atribuído ao valor do atributo “Description” do ficheiro de código fonte.

caso das marcas de comentário Lisp. Quanto um símbolo é identificado, a sua propriedade “valor” é obtida a partir da propriedade “lexema” através da remoção dos caracteres em causa.

## Analizador Sintáctico

A validação da frase de entrada pelo analisador sintáctico é feita usando uma estratégia conhecida por *análise sintáctica ascendente*. O seu objectivo é a construção da árvore de derivação da frase de entrada, mas no sentido contrário, começando nas folhas da árvore e terminando na raiz. Os símbolos terminais da frase nas folhas da árvore são lidos da esquerda para a direita (operação conhecida por *deslocamento* ou *shift*) e, quando é detectada uma sequência de símbolos que corresponde ao lado direito de uma produção da gramática, essa sequência é substituída pelo símbolo não terminal que está no lado esquerdo dessa produção (operação designada por *redução* ou *reduce*).

Considere-se a gramática da LDI-F e a frase de entrada cuja árvore de derivação está desenhada na Figura 12. O analisador sintáctico efectua a redução da frase “:DESCRIPTION :DOC-TEXT :HISTORY :HISTORY-HEADER :DOC-TEXT” para <file-header> pela ordem indicada na Figura 13.

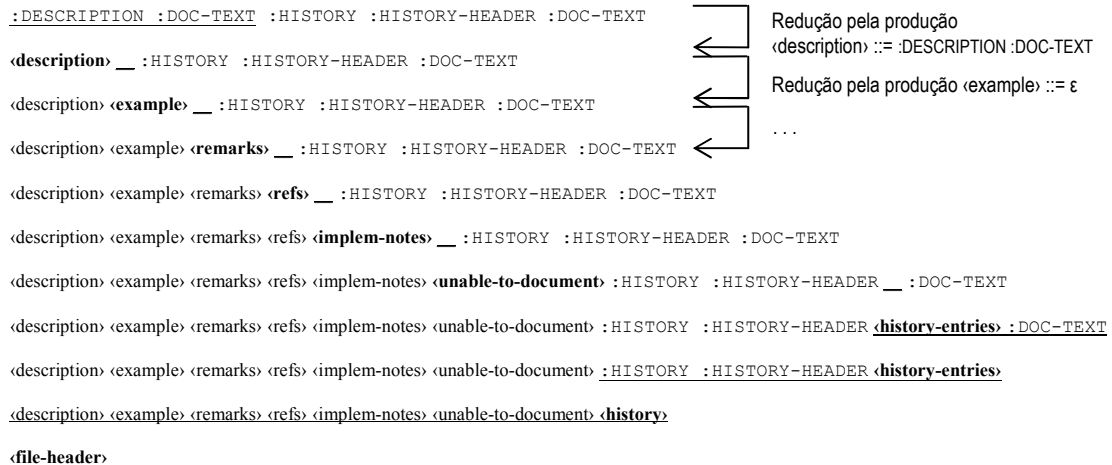


Figura 13: Redução de uma frase de entrada até ao símbolo inicial da gramática da LDI-F

Repare-se que cada redução efectuada pelo analisador sintáctico corresponde na realidade à operação oposta de uma derivação do símbolo não-terminal que está mais à direita no lado direito de cada produção. Esta técnica é conhecida por análise sintáctica ascendente LR. A leitura da frase de entrada é feita da esquerda para direita (*Left scan*) e a redução corresponde ao contrário de uma derivação do símbolo não terminal mais à direita (*Rightmost derivation*).

## Analizador Semântico e Gerador de Texto Intermédio

A análise sintáctica é o processo de determinar se uma frase de entrada pode ser gerada por uma gramática. Se for reconhecida como sendo uma frase pertencente à respectiva linguagem, pode então feita uma avaliação da sua semântica. Se a frase fizer sentido no contexto da linguagem ficam reunidas as condições para ser traduzida para o texto alvo.

Por outras palavras, a tradução do texto fonte para o texto alvo é orientada pela conformidade da sequência de símbolos terminais da frase de entrada com a gramática que define a sintaxe linguagem fonte.

A validação do significado de uma frase é feita pelo analisador semântico utilizando a informação que foi sendo colecionada nas propriedades dos seus símbolos terminais. Antes de iniciar o processo de validação, é acrescentada a propriedade “valor” a cada símbolo não terminal que está no lado esquerdo de uma produção da gramática e que corresponde à concatenação dos valores dos símbolos que estão no lado direito dessa produção. Tomando como exemplo a produção `<description> ::= :DESCRIPTION :DOC-TEXT`, o valor de `<description>` corresponde à concatenação dos valores de `:DESCRIPTION` e de `:DOC-TEXT`.

Relativamente ao excerto da frase de entrada que está em conformidade com esta produção, já vimos que está também de acordo com a regra semântica. Como tal, pode ser feita a geração de texto intermédio que, neste caso, corresponde à escrita do valor de `<description>` (ver Tabela 5).

Produção	Regra Semântica	Acção de conformidade com a regra
<code>&lt;description&gt; ::= :DESCRIPTION :DOC-TEXT</code>	O valor do atributo “Description” do ficheiro de código fonte é dado pelo valor de <code>:DOC-TEXT</code> .	Efectua a tradução para o texto alvo escrevendo o valor do símbolo não terminal <code>&lt;description&gt;</code>

Tabela 5: Exemplo de uma tradução para o texto alvo

### 2.1.2.4 Definição do Texto Alvo

Para representar uma frase processada pelo módulo de extracção de dados no contexto de uma linguagem, foi escolhida uma estrutura de fácil interpretação pelo Lisp com o objectivo de ser usada pelo módulo de exportação de dados. Uma frase em formato intermédio é representada por meio de uma lista de elementos, em que cada elemento contém informação sobre um atributo de um ficheiro de código fonte ou elemento de código fonte. Quando aplicável, um dos elementos da lista pode conter exclusivamente informação sobre erros detectados durante o processamento.



Retomando o exemplo da frase de entrada para o processador da LDI-F que está representada na Figura 10 e na Tabela 3, o *output* do módulo de extracção de dados seria representado conforme se indica na Figura 14. Do ficheiro de código fonte considerado seriam extraídos os valores dos atributos “Description”, “Date”<sup>9</sup>, e “Pathname”.

```
((:DESCRIPTION ("This file contains the definition of some functions." :EMPTY-LINE
               "It is used as an example for some (:EMPH "explanations")."
               :EMPTY-LINE))
 (:HISTORY "98/04/01      Silva      Created")
 (:PATHNAME "z:\\temp\\example.lisp"))
```

Figura 14: Representação intermédia de uma frase processada no contexto da LDI-F

### 2.1.2.5 Conceito de Execução do Módulo

As principais operações realizadas pelo módulo de extracção de dados são agrupadas em duas fases de execução. A primeira fase compreende a compilação do código fonte do módulo, tendo como efeito tanto a geração de funções que irão permitir instanciar os analisadores léxicos que são usados em primeiro lugar para cada linguagem, bem como a construção automática das tabelas de acções (*actions*) e de saltos (*gotos*) que são usadas para a análise sintáctica de cada linguagem (ver Figura 15).

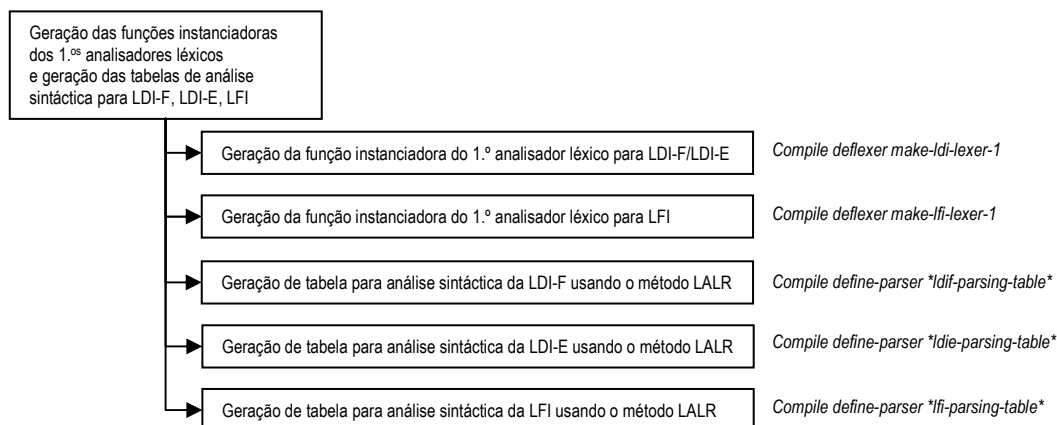


Figura 15: Actividades realizadas durante a compilação do código fonte do módulo de extracção de dados

A segunda fase corresponde à execução do módulo, cujas principais actividades estão representadas esquematicamente na Figura 16.

<sup>9</sup> Na realidade, a actual implementação do módulo de extracção de dados suporta apenas parcialmente a extracção do atributo “Date”. O *output* (:HISTORY "98/04/01 Silva Created") indicado na figura deverá ser alterado no futuro pela SISCOG para (:DATE "98/04/01"), e que irá corresponder à representação do atributo “Date” do ficheiro de código fonte em questão.

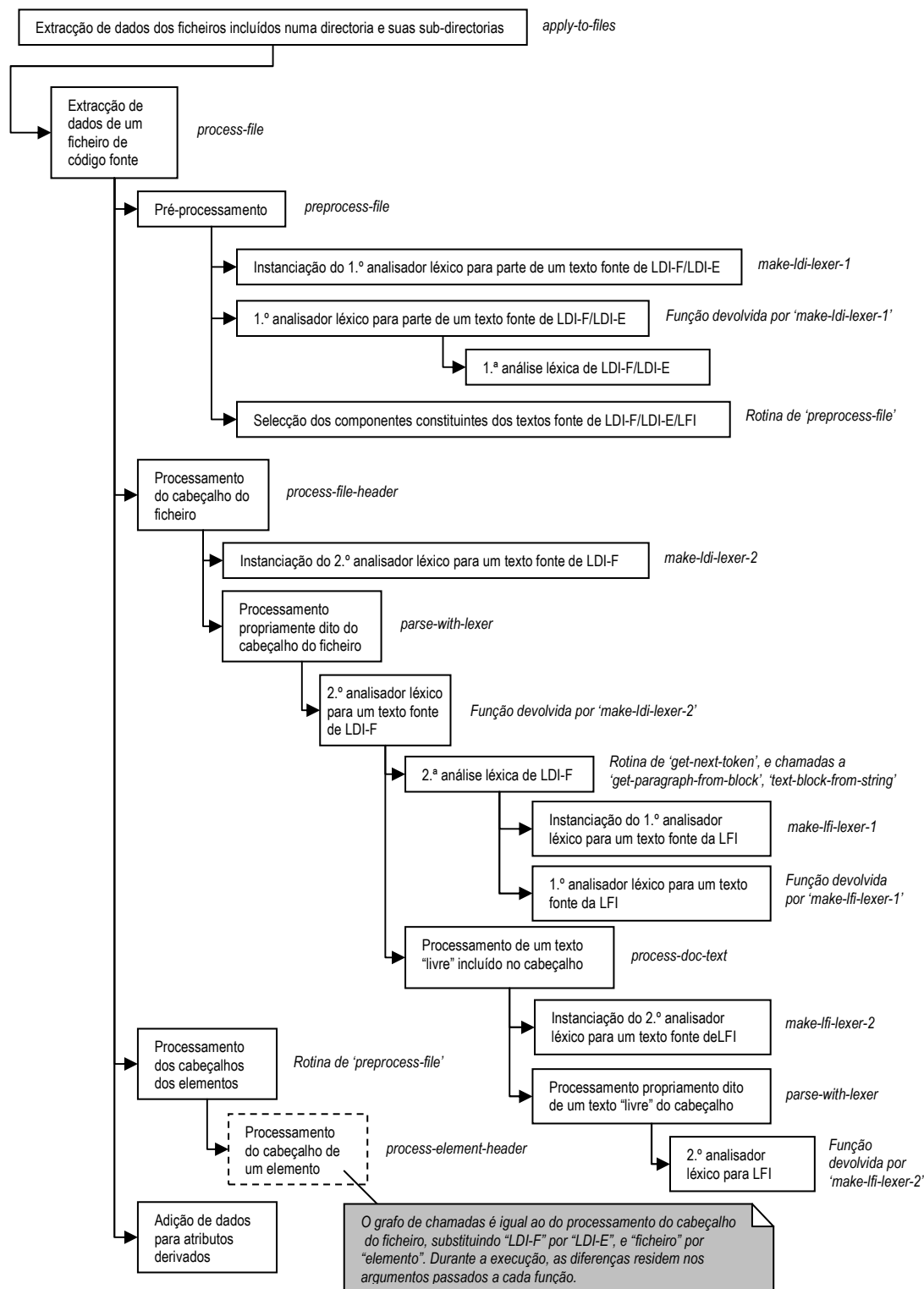


Figura 16: Grafo de chamadas das principais operações realizadas pelo módulo de extracção de dados

Nestas operações estão envolvidos os pacotes de *software* que foram anunciados na secção 1 *Introdução*. As seguintes secções apresentam uma descrição sumária destes componentes.

## LEXER

Este componente realiza a análise léxica inicial a uma frase de entrada, para cada uma das três linguagens. Inclui um gerador de uma função que permite instanciar um analisador léxico para uma frase de entrada em particular.

Para a geração da função com o papel instanciador, recebe um vocabulário de uma linguagem especificado sob a forma de um conjunto expressões regulares e de acções de transposição para símbolos terminais dessa linguagem. Neste processo, o suporte à utilização de expressões regulares é dado pelo pacote REGEX.

O analisador léxico instanciado para uma frase de entrada contém essa mesma frase. Assim, cada vez que é chamado pelo analisador sintáctico, o analisador léxico devolve o próximo símbolo terminal e respectivo valor que forem identificados nessa frase.

## CL-YACC

Este componente é responsável por realizar a análise sintáctica, a análise semântica e a geração de texto intermédio de frases pertencentes a cada uma das linguagens referidas. Inclui um gerador LALR (*lookahead LR*) de tabelas de acções e de saltos usadas para a análise sintáctica de uma determinada frase de entrada.

Para a geração de uma tabela de acções e de saltos recebe uma gramática independente de contexto em notação BNF com acções semânticas associadas a cada produção da gramática.

Para a execução do algoritmo LR recebe a tabela de acções e de saltos gerada previamente e o segundo analisador léxico instanciado para uma frase de entrada, e devolve a frase processada.

## MK:DEFSYSTEM

Este utilitário permite especificar a definição de um sistema em linguagem Lisp.

## 2.2 Módulo de Exportação de Dados

O módulo de exportação de dados é responsável por transformar os dados contidos num ficheiro de formato intermédio em dados que podem ser lidos pelo utilizador. O texto fonte deste módulo são os dados processados pelo módulo de extracção de dados, o texto alvo deste módulo

corresponde a uma representação do texto fonte na forma de um documento HTML (ver Figura 17).

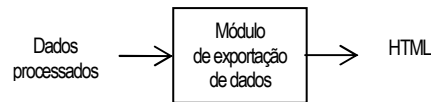


Figura 17: Módulo de exportação de dados

Na fase de *design* deste módulo decidiu-se que seria gerado um documento intermédio no formato XML, sendo posteriormente convertido num documento HTML<sup>10</sup>. Esta decisão teve por base a facilidade de conversão de dados do formato XML em vários outros formatos, sem haver a necessidade de alterar o *output* do módulo de exportação.

## 2.2.1 Tecnologias

Esta secção descreve as tecnologias utilizadas no módulo de exportação de dados.

### 2.2.1.1 Document Object Model

O *Document Object Model* (DOM) é uma plataforma (e uma linguagem) de interface neutra que permite que programas acessem dinamicamente e actualizem o conteúdo, a estrutura e o estilo de documentos. Este modelo permite processar um documento e integrar o resultado desse processamento no próprio documento.

No módulo de exportação de dados é utilizado o DOM de nível 1 disponibilizado no *Allegro Common Lisp* comercializado pela Franz, que respeita o *standard* definido pelo W3C.

### 2.2.1.2 Extensible Markup Language (XML)

O XML é uma linguagem amplamente conhecida, razão pela qual não será abordada aqui em detalhe. No entanto a sua especificação pode ser consultada no seguinte endereço:

<http://www.w3.org/TR/xml/>

<sup>10</sup> Ver secção 2.2.2.1 *Definição do Módulo*.

### **2.2.1.3 XML Schema (XSD)**

O esquema XSD é um conjunto de regras que permite validar a estrutura, o conteúdo e a semântica de um documento XML.

### **2.2.1.4 Extensible Stylesheet Language Family (XSL)**

O XSL é um conjunto de recomendações sobre como transformar e apresentar documentos XML, sendo composto pelos seguintes elementos:

- XSL-FO: sendo o XSL uma linguagem utilizada para definir folhas de estilo, uma folha de estilo XSL-FO descreve a forma como um documento XML de um determinado tipo deve ser mostrado;
- XSLT: é uma linguagem para transformação de documentos XML, que permite transformar os dados de um documento XML noutro documento XML ou ainda em documentos que possam ser lidos por humanos. De referir que o documento original não é alterado;
- XPath: é uma linguagem utilizada para aceder a partes de um documento XML, ou para realizar operações sobre valores contidos num documento XML.

### **2.2.1.5 Object Linking and Embedding (OLE)**

A tecnologia OLE desenvolvida pela Microsoft permite embeber e relacionar objectos e documentos.

## **2.2.2 Concepção**

Esta secção descreve o desenvolvimento efectuado para a produção do módulo de exportação de dados.

### **2.2.2.1 Definição do Módulo**

A exportação de dados divide-se em duas fases principais: a geração de um documento de formato XML, um formato intermédio, e a geração de um documento de formato HTML, o texto alvo deste módulo (ver Figura 18). Estas fases são descritas nas secções seguintes.

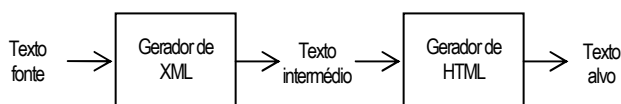


Figura 18: Sub-tarefas do módulo de exportação de dados

## Gerador de XML

O algoritmo que implementa o gerador de XML tem por base o *Document Object Model* ou DOM. Este algoritmo começa por criar um documento XML, neste documento é indicado o esquema XSD que valida o documento XML.

O passo seguinte consiste em percorrer os dados contidos no ficheiro de formato intermédio e colocá-los no documento XML. De referir que são adicionados ao documento XML apenas os elementos cujo valor do atributo `:VISIBILITY` é o pretendido para o documento que está a ser gerado, e ainda, que os dados não são colocados no documento XML de forma aleatória: primeiro o algoritmo procura determinados símbolos no ficheiro de formato intermédio, quando os encontra abre uma etiqueta (*tag*) correspondente ao símbolo que encontrou e coloca dentro dessa *tag* o valor correspondente ao símbolo. Depois de todos os dados contidos no ficheiro de formato intermédio terem sido colocados no documento XML este é guardado de forma persistente.

## Gerador de HTML

O algoritmo que implementa o gerador de HTML, tem por base a tecnologia *Object Linking and Embedding (OLE)*. O algoritmo aplica um ficheiro XSL<sup>11</sup> ao documento XML, gerando desta forma o HTML de acordo com as regras definidas no ficheiro XSL.

### 2.2.2.2 Conceito de Execução

As principais operações realizadas pelo módulo de exportação de dados são agrupadas em três fases de execução (ver Figura 19). A primeira fase corresponde à instanciação de um documento XML. A segunda fase corresponde ao processamento dos dados contidos no ficheiro de formato intermédio e à sua instanciação no documento XML. A terceira fase corresponde à transformação do documento XML num documento HTML.

<sup>11</sup> Ver secção Extensible Stylesheet Language Family (XSL).

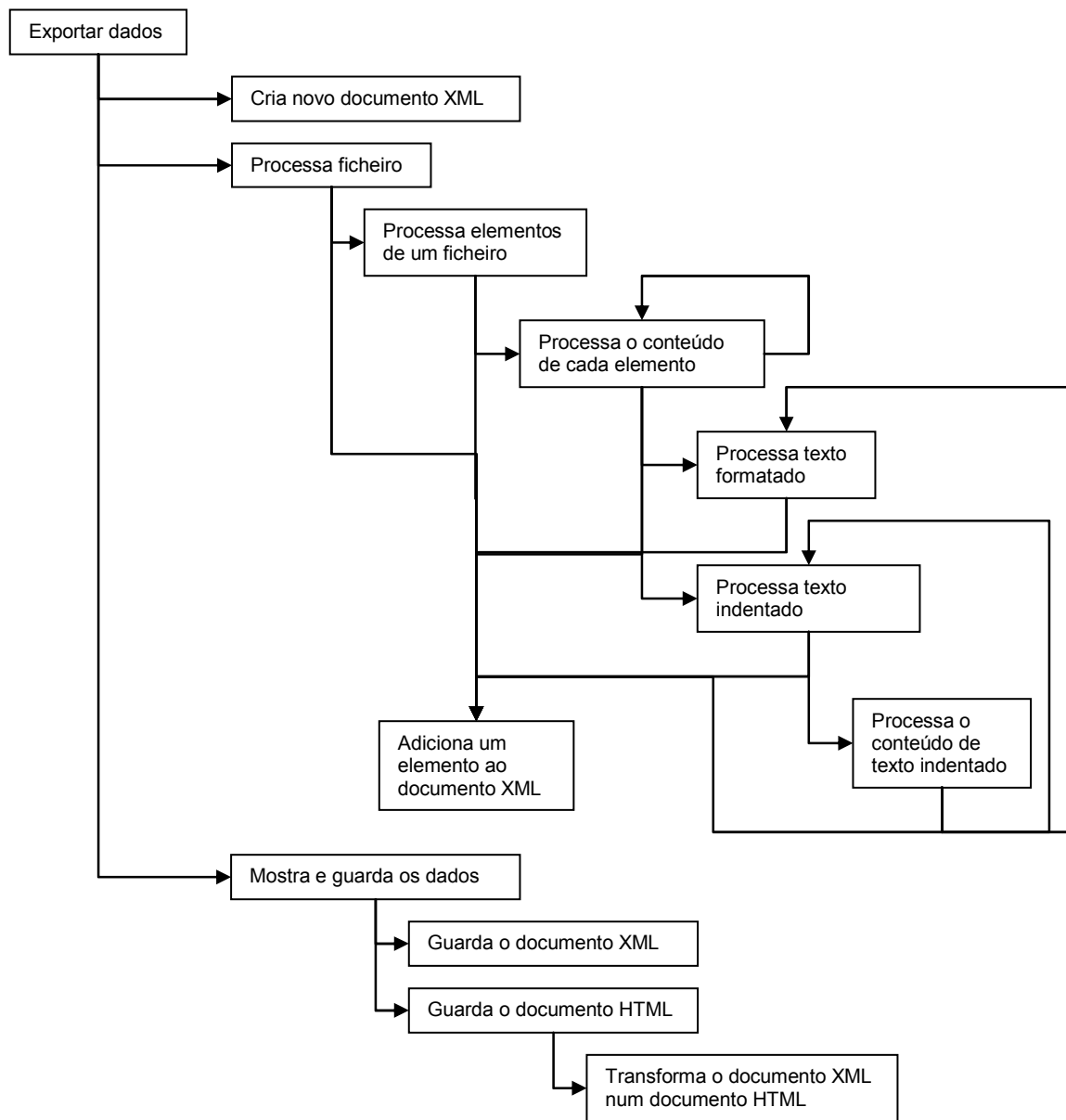


Figura 19: Grafo de chamadas das principais operações realizadas pelo módulo de exportação de dados.

### 3 Conclusão

A construção de um programa de *software* para extrair documentação interna existente em ficheiros de código fonte, com base em regras de estruturação de escrita, e depois publicá-la num formato de documentação de fácil consulta e leitura, foi realizada com sucesso. O programa que foi construído está neste momento em condições de poder ser usado internamente e em exclusivo pela SISCOG. Existem melhoramentos a fazer, necessariamente, mas já serão baseados num sistema de geração de manuais estabelecido. O anexo *Caso de Teste* salienta dois pontos de melhoria.

A pesquisa realizada pelos discentes revelou-se positiva para a construção do programa. A maior parte do tempo foi dispendido nesta actividade, no entanto, conclui-se que foi fundamental para atingir os objectivos propostos. Além disso, a investigação sobre técnicas e tecnologias existentes contribuiu para a aquisição de conhecimentos sobre os processadores de linguagens e sobre o XML e tecnologias associadas que, no início do projecto, constituíam temas novos para os discentes.

A geração de manuais de referência é de grande importância, tendo sido definida há muito pela SISCOG. Desde cedo que a documentação interna é produzida com o propósito de obter manuais de referência de forma automática. O gerador de documentação de referência produzido no âmbito deste projecto final de curso pode agora ser introduzido no processo de desenvolvimento de *software* da SISCOG.



## Referências bibliográficas

- [1] AHO, Alfred V., SETHI, Ravi and ULLMAN, Jeffrey D. (1988), *Compilers – Principles, Techniques and Tools*. Reading, Massachusetts: Addison-Wesley.
- [2] CRESPO, Rui Gustavo (2001), *Processadores de Linguagens – Da Concepção à Implementação*, Lisboa: IST Press.
- [3] HOPCROFT, John E., MOTWANI, Rajeev and ULLMAN, Jeffrey D. (2003), *Introduction to Automata Theory, Languages, and Computation* (2<sup>nd</sup> ed.), Upper Saddle River, N.J.: Pearson.
- [4] AZEVEDO, Mário (2004), *Teses, Relatórios e Trabalhos Escolares – Sugestões para Estruturação e Escrita* (4.<sup>a</sup> ed), Lisboa: Universidade Católica Editora.