Resumo

A falta de documentação de projetos de software tem se mostrado um sério problema dentro de empresas de desenvolvimento. Este acontecimento pode acarretar outros problemas graves como a perda de manutenibilidade, impossibilidade de reutilização de códigos e até mesmo o abandono de certos projetos.

Em certos casos, acredita-se que, como a documentação não incrementa diretamente a rentabilidade da empresa, não se deve lhe dar tanta prioridade como, por exemplo, a construção de um *framework*. A boa documentação, além de facilitar a manutenção, pode evitar que todo conhecimento de um determinado projeto fique detido nas mãos de um desenvolvedor ou um grupo deles. Com isso a empresa torna-se menos dependente dos membros da sua equipe.

Infelizmente a documentação não é um trabalho simples de se fazer e também não deixa de ser uma tarefa custosa, quando consideramos os fatores tempo e mão de obra.

O C-Doc foi idealizado para auxiliar na construção da documentação de projetos escritos em diversas linguagens de programação que dão suporte a blocos de comentários - de onde são extraídas as informações necessárias para a construção da documentação. O C-Doc ainda fornece a capacidade de receber novos módulos (*plug-ins*), a fim de produzir novos formatos de documentação de acordo com o desejo dos produtores de software, e dá suporte nativo aos construtores da linguagem de especificação formal CML (*C Modeling Language*).

Abstract

The lack of documentation in software projects has showed itself a serious problem among software development organizations. This event can cause many problems, like difficulties on maintenance, impossibility of software reuse and even the abandon of certain projects.
In some cases, the belief is that as documentation does not straightly increase profits of the company, it shouldn't receive as much priority as, for instance, the construction of a framework. Good documentation, beyond making maintenance easier, can avoid that the knowledge of a determined project remains in the hands of a developer, or a group of developers. This way, a company becomes less dependent of the members of its team.
Unfortunately, documenting software is not a simple work to do, and it is also a costly task, when considering time and resources needed.
C-Doc was idealized to help in the construction of documentation in projects developed in programming languages that offer blocks of comments - from where it is extracted the information needed for building the documentation. C-Doc also supplies the ability to receive new modules (plugins), looking to produce new documentation formats, according to the wish of software producers, and it provides native support to the constructors of the formal specification language CML.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

CML – *C Modeling Language*

JFlex – *Faster Scanner Generator for Java*

CUP – *Constructor of useful parsers*

HTML – *Hypertext Markup Language*

Manpages – *Manual Pages*

XML – *Extensible Markup Language*

TXT – *Text* (Texto)

RTF – *Rich Text Format*

API – *Application Programming Interface*

FLEX - *The Fast Scanner Analyzer*

Yacc – *Yet Another Compiler-Compiler*

JML – Java *Modeling Language*

GUI – *Graphical User Interface*

AST – *Abstract Syntactic Tree* (Árvore Sintática Abstrata)

EBNF – *Extended Backus-Naur-Form*

Agradecimentos

Gostaria de agradecer a minha família, minhas irmãs Mércia e Débora, e especialmente aos meus pais Paulo e Célia, que sempre lutaram, independente das dificuldades, pela ótima formação educacional dos filhos. Agradeço especialmente à minha namorada Danusa que sempre me apoiou nos momentos difíceis, inclusive em todo processo de construção deste trabalho.

Gostaria de agradecer aos meus amigos dos tempos do colégio: Danillo, Rodrigo e Sérgio, que, apesar da distância e dos esporádicos encontros nestes últimos cinco anos, sei que estarão sempre ao meu lado.

Agradeço também a todos os amigos que me conquistaram durante estes cinco anos no meio acadêmico e profissional, especialmente aos integrantes d'Os Caras, aos integrantes do CEUL no Serpro, ao pessoal da Policentro, da Cemicro Informática e da secretaria de Pós-graduação da POLI.

Por último e não menos importantes gostaria de agradecer ao corpo docente do Departamento de Sistemas Computacionais, especialmente aos professores Fernando Buarque e Carlos Alexandre. Se hoje acredito no meu futuro profissional, devo isto a vocês.

Obrigado a todos.

Capítulo 1

Introdução

Com o passar do tempo, as empresas de desenvolvimento de software e os desenvolvedores independentes atentaram para a importância da documentação para a qualidade e manutenção dos sistemas desenvolvidos.

A falta de documentação de um projeto de médio ou grande porte pode acarretar problemas gravíssimos como a dificuldade de manutenção e até mesmo a depreciação completa do sistema. A situação de empresas piora ainda mais quando estas costumam manter projetos sob responsabilidade de apenas um funcionário.

Além de ajudar na manutenção do sistema, uma boa documentação pode interferir em vários outros aspectos do sistema, como o reuso de código [1]. Com uma boa documentação economiza-se também no suporte ao usuário. Já que um sistema bem documentado possui um incremento de usabilidade, fazendo com que caia a quantidade de requisições de suporte por partes dos usuários [2].

De acordo com o autor Dennis M. Ahern [3], processos bem documentados podem reduzir ou até mesmo eliminar a confusão sobre o que deve ser feito em fases de manutenção, de modo que as atividades sejam executadas mais consistentemente.

O grande problema está nas dificuldades de se documentar um projeto, principalmente pelo fato de a documentação não gerar renda diretamente, fazendo com que lhe seja atribuída uma prioridade muito baixa.

Este trabalho foi idealizado para auxiliar o processo de documentação de projetos desenvolvidos em diversas linguagens. A idéia é que com a ferramenta C-Doc possa-se construir a documentação do projeto com informações extraídas de anotação adicionadas dentro dos blocos de comentários.

O C-Doc é uma ferramenta com um alto nível de customização, pois se pode utilizá-lo em projetos desenvolvidos em diversas linguagens, bastando ajustar alguns parâmetros de entrada do sistema. Além disso, o C-Doc pode também gerar documentações em diversos formatos, já que

apresenta uma arquitetura de *plug-ins*, e assim, possibilitando o acoplamento de novos módulos de saída além dos que já acompanham a ferramenta.

1.1 Objetivos

Os principais objetivos deste trabalho de conclusão de curso são:

- Desenvolver uma ferramenta (o C-Doc) que ajude no processo de documentação, baseada em informações passadas pelo usuário através de comentários no código fonte da aplicação desenvolvida.
- Criar módulos de saída (*plug-ins*) distintos capazes de apresentar modelos de documentação diversificados.
- Geração de um completo e detalhado manual de utilização do sistema e de criação de novos módulos acopláveis ao sistema

1.2 Estrutura do Trabalho

Esta monografia está organizada em Capítulos e Apêndices. A seguir será detalhado o conteúdo de cada parte:

O capítulo 2 apresenta alguns trabalhos relacionados a este e às ferramentas que auxiliaram na construção do C-Doc. Dentre os trabalhos relacionados estão o Javadoc e o Doxygen. Dentre as ferramentas utilizadas estão os geradores automáticos de *scanners* e *parsers*, JFlex e Cup e a linguagem CML.

O Capítulo 3 apresenta a ferramenta C-Doc propriamente dita, como se deu a construção do seu compilador e também dos módulos de saída.

O Capítulo 4 apresenta o resultado apresentado pelo C-Doc. Documentos gerados pelos módulos de saída.

O Capítulo 5 conclui este trabalho e no Apêndice A é apresentado um manual de utilização do C-Doc.

Capítulo 2

Trabalhos Relacionados e Fundamentação Teórica

Atualmente, existem diversas ferramentas de auxílio à construção de documentação de projetos disponíveis aos desenvolvedores. Estas ferramentas geralmente utilizam técnicas de extração de informações contidas em comentários anotados nos códigos-fonte, em que são utilizadas *tags* pré-definidas na construção da ferramenta. Como um bom exemplo de ferramenta que possui estas características podemos citar o Javadoc [12], ferramenta esta desenvolvida pelos laboratórios da *Sun Microsystems* [15], com o objetivo de auxiliar no processo de documentação de projetos desenvolvidos utilizando-se da linguagem *Java* [13]. Um outro trabalho relacionado a este é a linguagem CML [4] que, apesar de não ser uma ferramenta de auxílio à documentação, extrai informações sobre requisitos não funcionais do sistema, contidas em comentários, para verificar se o código respeita o que foi especificado.

Neste capítulo serão abordados com mais detalhes alguns trabalhos relacionados, como a extração de informações de blocos de comentários, as ferramentas de auxílio à documentação Javadoc e Doxygen [14]. Por fim serão apresentadas as ferramentas utilizadas para a construção da ferramenta C-Doc, como o JFlex [16], o CUP [17] e a linguagem CML.

2.1 Trabalhos Relacionados

2.1.1 Doxygen

O Doxygen é uma ferramenta de auxílio à documentação gratuita sob licença GPL desenvolvida desde 1997 sobre a responsabilidade de Dimitri van Heesch. Esta ferramenta extrai informações específicas contidas em códigos-fonte de diversas linguagens como C++, C, Java, Objective-C, Python, entre outras, e é capaz de gerar documentações navegáveis, formatadas e em formato para impressão, em HTML ou em outros formatos como o XML e TXT. Além destas, o Doxygen possui outras características que merecem destaque. Elas são[9]:

- Portabilidade;
- compatibilidade com o Javadoc (apresentado adiante) e outras ferramentas similares;
- reconhecimento automático de referências cruzadas;
- conversão de documentos em HTML para formatos como L A T E X, RTF ou *manpages*;
- geração automática de diagrama de classes.

```
/**
 * Method documentation.
 * @param x The parameter.
 * @return The return value.
 * @see anotherFunction()
 */

/** Single-line documentation. */
/// Single-line documentation.
```

Figura 2-1. Exemplos de blocos de comentários suportados pelo Doxygen

O Doxygen suporta diversas formas de iniciar e finalizar um bloco de comentário que terá suas informações extraídas para se construir a documentação. Algumas delas são demonstradas na **Figura 2-1**. Pode-se observar que o Doxygen utiliza os blocos de comentários usuais da linguagem, adicionando a estes um caractere adicional que marcará o bloco, para que o compilador extraia as informações lá contidas. Nos casos apresentados, foram utilizados os caracteres * (2x) e / como delimitadores.

2.1.2 Javadoc

O Javadoc é uma ferramenta desenvolvida pela Sun microsystems com o objetivo de gerar a documentação de projetos escritos na linguagem Java. Esta ferramenta possui um funcionamento similar ao da ferramenta Doxygen, porém com algumas características importantes que os diferenciam. Dentre estas características podemos citar:

- trabalha apenas com a linguagem Java;
- produz apenas documentos no formato HTML;
- forte integração com os ambientes de desenvolvimento disponíveis;
- provê uma API para criação de Doclets [10] e Taglets [11] que permitem a análise da estrutura de uma aplicação Java.

A forma como o desenvolvedor deve proceder para escrever os comentários nos códigos fonte, permitindo que o Javadoc possa extrair as informações necessárias com a finalidade de construir a documentação, serviu de inspiração para diversas ferramentas relacionadas. Temos como exemplos o Doxygen e o próprio C-Doc. Podemos observar na **Figura 2-2** um exemplo de como devem ser especificados os comentários.

```
/**
 * Validata um movimento de xadrez.
 *
 * @param aColunaDe   Coluna atual da peça a ser movida
 * @param aLinhaDe    Linha atual da peça a ser movida
 * @param aColunaPara Coluna destino da peça a ser movida
 * @param aLinhaPara  Linha destino da peça a ser movida
 * @return            verdadeiro se o movimento é válido ou falso se inválido
 * @author            José da Silva
 */
boolean validaMovimento(int aColunaDe, int aLinhaDe, int aColunaPara, int aLinhaPara)
{
    ...
}
```

Figura 2-2. Exemplo de comentário suportado pelo Javadoc

O Javadoc se utiliza de *tags* que podem ser usadas opcionalmente para extrair informações mais específicas referentes a um determinado método ou classe. Estas *tags* podem ser observadas na **Figura 2-2** e, na **Tabela 2-1**, pode-se observar a lista das *tags* mais utilizadas pelos desenvolvedores Java.

Tabela 2-1. Alguns exemplos de *tags* suportadas pelo Javadoc.

Tag	Descrição
@author	Nome do desenvolvedor.
@deprecated	Marca o método como *deprecated*. Algumas IDEs exibirão um alerta de compilação se o método for chamado.
@exception	Documenta uma exceção lançada por um método — veja também @throws.
@param	Define um parâmetro do método. Requerido para cada parâmetro.
@return	Documenta o valor de retorno. Essa tag não deve ser usada para construtores ou métodos definidos com o tipo de retorno *void*.
@see	Documenta uma associação a outro método ou classe.
@since	Documenta quando o método foi adicionado à classe.
@throws	Documenta uma exceção lançada por um método. É um sinônimo para a @exception introduzida no Javadoc 1.2.
@version	Exibe o número da versão de uma classe ou um método.

O resultado apresentado pela documentação gerada pelo Javadoc é altamente navegável, já que o mesmo reconhece a ligação entre os métodos e classes envolvidas na documentação. Um exemplo de documentação gerada pelo Javadoc pode ser observado na **Figura 2-3**.

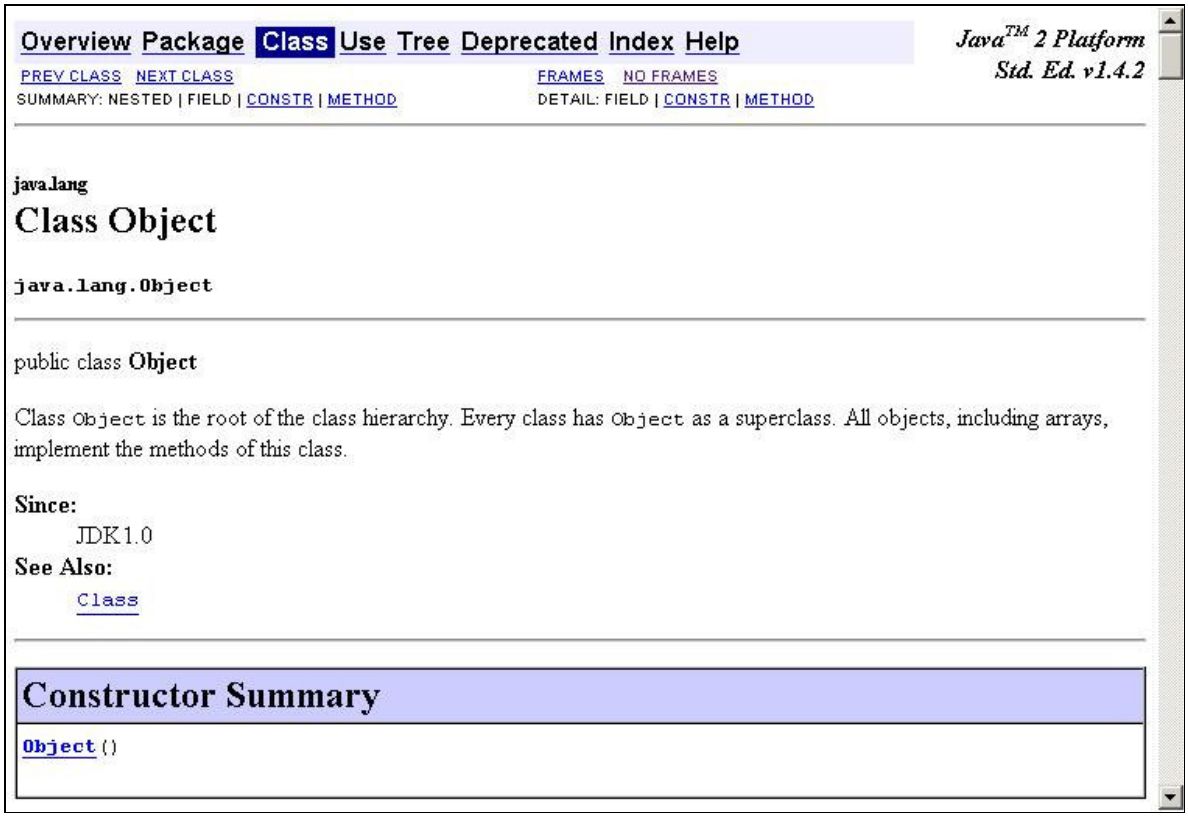

Figura 2-3. Exemplo de documentação gerada pelo Javadoc.

2.2 Ferramentas Utilizadas

2.2.1 JFlex

O JFlex [16] é um gerador de analisadores léxicos para Java escrito em Java. Foi desenvolvido por Elliot Berk da Universidade de Princeton com base no gerador Flex [20], escrito em linguagem C. As principais características do JFlex são listadas a seguir:

- suporte completo à codificação Unicode;
- *scanners* rápidos, de alto desempenho;
- geração rápida de *Scanners;*
- sintaxe de especificação conveniente;
- independência de plataforma;
- compatível com o JLex.

A utilização de ferramentas como o JFlex é largamente encorajada, já que a construção de compiladores sem a utilização de ferramentas similares torna-se uma tarefa bastante árdua e suscetível a erros. Isto não ocorre quando utilizamos ferramentas como o JFlex, já que estas ferramentas, quando se utilizam expressões regulares bem especificadas, geram uma máquina de estados ótima e sem falhas.

Na **Figura 2-4** podemos observar um exemplo de especificação utilizando o JFlex.

```
//sessão de código do usuário
package foo
%%

//sessão de declarações e especificações
%class FooBar

whitespace = [ \t\n\r\f]
letters = [a-zA-Z_]
numbers = [0-9]

%%

//sessão de regras léxicas
{letters}       {return new Symbol(sym.LETTER)}
{numbers}       {return new Symbol(sym.NUMBER)}
{whitespace}    { /* ignore */ }
```

Figura 2-4. Exemplo de especificação de um *scanner* hipotética.

2.2.2 Cup

O Cup [17] é um sistema de geração de *parsers* do tipo LALR – *Look Ahead Left Recursive* a partir de uma especificação simples de uma gramática. Este possui as mesmas funções da difundida ferramenta de geração de *parsers,* O *Yacc.* Entretanto, o Cup é totalmente escrito na linguagem Java, as especificações são compatíveis com a linguagem Java e os *parsers* são gerados na linguagem Java.

A especificação de uma gramática hipotética pode ser observada na **Figura 2-5**. Nela podemos notar que a especificação é dividida entre as sessões que listamos e descrevemos abaixo.

- Declarações iniciais – contém as declarações necessárias para o correto funcionamento do Cup durante a geração do parser.
- Terminais – contém os tokens que serão retornados pelo scanner que foi gerado pelo JFlex.
- Não Terminais – contém os símbolos que serão usados como auxiliares na construção da gramática.

- Precedências – contém a precedência das informações a serem reconhecidas pelo parser.
- Gramática – contém a definição da gramática propriamente dita.

```
import java_cup.runtime.*;

/* Declarações preliminares  */
init with {: scanner.init();              :};
scan with {: return scanner.next_token(); :};

/* Definição dos Terminais (tokens retornados pelo scanner). */
terminal            SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal            UMINUS, LPAREN, RPAREN;
terminal Integer    NUMBER;

/* Não Terminais */
non terminal            expr_list, expr_part;
non terminal Integer    expr, term, factor;

/* Ordem de precedência */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS;

/* A gramática */
expr_list ::= expr_list expr_part |
              expr_part;
expr_part ::= expr SEMI;
expr      ::= expr PLUS expr
            | expr MINUS expr
            | expr TIMES expr
            | expr DIVIDE expr
            | expr MOD expr
            | MINUS expr %prec UMINUS
            | LPAREN expr RPAREN
            | NUMBER
            ;
```

Figura 2-5. Exemplo hipotético da definição de uma gramática usando o Cup.

2.2.3 CML

Um aplicativo desenvolvido na linguagem Java pode ser anotado para incluir restrições adicionais ao modelo. Para isto existe a linguagem JML (Java Modeling Language) [6] que é uma linguagem de especificação formal baseada em técnicas de Projeto por Contrato (*Design by Contract*) [5].

Assim como JML, CML (C Modeling Language) [4] é uma linguagem de especificação formal também baseada em técnicas de Projeto por Contrato, utilizada para especificar características não funcionais (tamanho de código, fatores relacionados ao tempo de execução, etc) de funções e módulos desenvolvidos na linguagem C.

Quando a linguagem CML é utilizada para especificar o comportamento de aplicativos, anotações são inseridas no próprio código, ou seja, não existe a necessidade de armazenar as especificações em arquivos de metadados que são geralmente apresentados no formato XML. Esta abordagem já é muito utilizada em projetos orientados a objetos em que classes e interfaces são anotadas com o intuito de se construir uma documentação baseada nessas anotações.

Além de auxiliar na especificação de um aplicativo desenvolvido na linguagem C, o CML extrai as informações contidas nas anotações do código fonte, gerando um arquivo com formato específico que possa ser lido por ferramentas de análise. Por fim, estas ferramentas de análise apresentam o resultado ao usuário, explicitando se o programa respeita ou não o que foi especificado. Na **Figura 2-6** podemos observar, de forma gráfica, o funcionamento do CML.

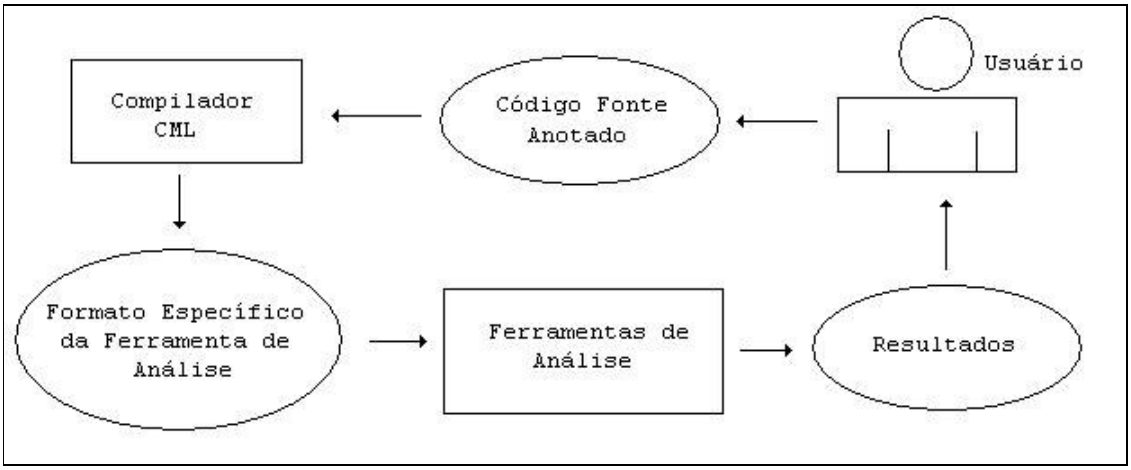

Figura 2-6. Esquema de funcionamento do CML.

As anotações aceitas pelo CML são bastante semelhantes às anotações suportadas pelo Javadoc. Um exemplo é apresentado na **Figura 2-7**. Na **Tabela 2-2** podemos observar algumas das principais *tags* suportadas pelo CML (uma tabela completa pode ser encontrada no Apêndice B).

```
/**
* @type T
* @name T1
* @scheduling NP
* @processor P1
* @release 1
* @phase 1
* @period 9
* @deadline 9
* @computing 1
*/
void T1 () {//implementação}
```

Figura 2-7. Exemplo de anotações suportadas pelo CML

Tabela 2-2. Principais anotações suportadas pelo CML.

Anotação	Descrição
@type	Indica o tipo da tarefa – pode ser T (tarefa comum) ou M (mensagem).
@name	Nome que identifica a tarefa.
@scheduling	Método de escalonamento – pode ser P (Preemptivo) ou NP (Não-preemptivo).
@processor/@bus	Nome do processador/canal de comunicação, onde a tarefa será alocada.
@release	Tempo de *release.*
@period	Tempo de período
@phase	Tempo de fase
@deadline	*Deadline* da tarefa
@computing / @communication	Tempo de computação da tarefa, ou veiculação da mensagem.
@preceds	Relação de precedência entre uma relação de tarefas.
@excludes	Relação de exclusão com uma lista de tarefas.

Cada uma destas *tags*, quando utilizadas, exige algum tipo de parâmetro. As exigência relacionadas a tais formalismos também são apresentadas no Apêndice B.

Capítulo 3

O C-Doc

A ferramenta C-Doc tem como objetivo principal gerar a documentação de projetos de software baseada em informações anotadas nos códigos fonte das aplicações. Apesar de ser uma ferramenta projetada para trabalhar com praticamente qualquer linguagem de programação que dê suporte a blocos de comentários, o foco deste trabalho está voltado para as linguagens C e CML.

Assim como o JML, o Javadoc e o próprio CML, o C-Doc é baseado em anotações embutidas no código fonte da aplicação que está sendo documentada. Para que não haja interferência na compilação do código fonte do aplicativo, as informações inerentes à documentação são inseridas em blocos de comentários.

O C-Doc foi construído com base em uma arquitetura de *plug-ins*. Esta arquitetura permite que novos *plug-ins* possam ser construídos e acoplados ao sistema para que se possa gerar novos formatos de documentos. Neste trabalho estes *plug-ins* são denominados módulos de saída.

Durante a concepção deste trabalho, foram construídos dois módulos de saída com o objetivo de exemplificar a construção destes. Os formatos de saída escolhidos foram o HTML, pela boa flexibilidade em termos de apresentação de conteúdos, e o formato *Manpages*, que é uma das formas mais difundidas de apresentação de documentações e manuais no mundo do Software Livre.

Neste capítulo serão abordadas as etapas do processo de implementação da ferramenta C-Doc e também a criação dos chamados módulos de saída, que são os módulos responsáveis pela geração da documentação final no formato de arquivo desejado pelo usuário. A primeira seção é dedicada à fase de modelagem da ferramenta. Na segunda seção apresenta-se a construção do compilador da ferramenta. Por fim, na terceira seção, será apresentada a forma de construção dos módulos de saída, já citados anteriormente.

3.1 Modelagem

O primeiro passo para a concepção da ferramenta é a especificação dos parâmetros da aplicação, das funcionalidades a serem implementadas e do escopo do projeto. Para este trabalho aconteceu uma redução do projeto em relação ao projeto inicial do C-Doc, já que, no projeto original, a ferramenta C-Doc dá suporte a diversas linguagens de programação, enquanto neste trabalho o suporte é dado apenas para as linguagens C e CML.

A ferramenta C-Doc pode ser separada nas seguintes partes:

- Compilador – Parte do sistema responsável pela extração dos dados contidos nos blocos de comentários dos arquivos fonte da aplicação em que se deseja gerar a documentação.
- Módulo de armazenamento – Módulo do sistema responsável por gerar arquivos no formato XML contendo todas as informações extraídas do código fonte.
- Módulo de construção do formato final – Nesta parte, são extraídas as informações dos arquivos XML gerados na fase de armazenamento e é construída a documentação no formato desejado.

A **Figura 3-1** ilustra bem a interação entre as diversas partes do sistema apresentadas anteriormente.

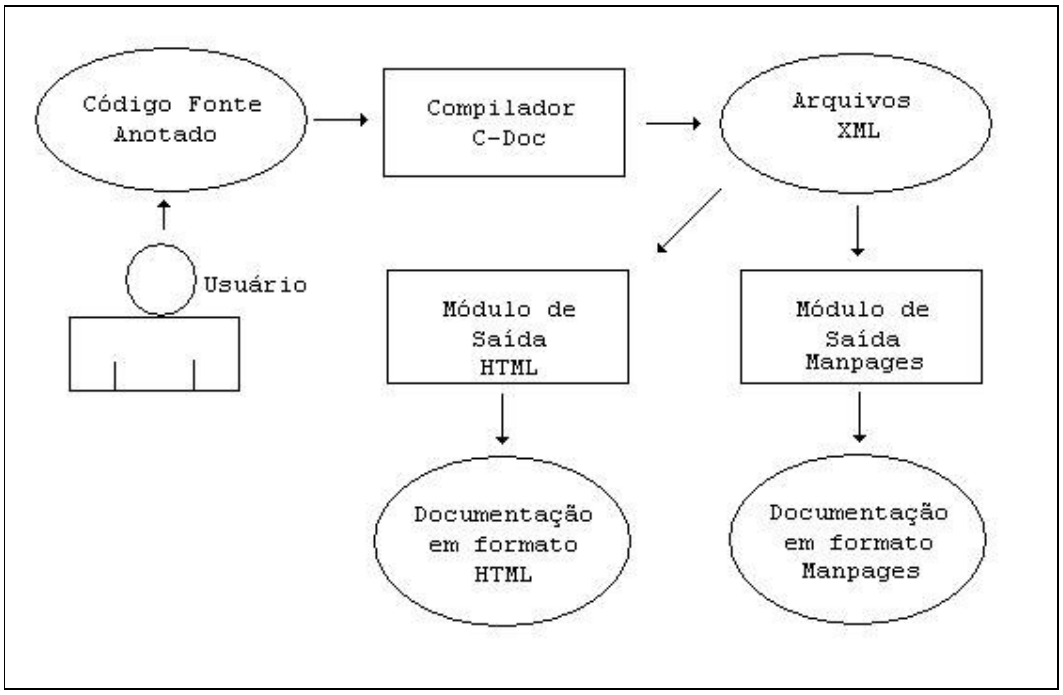

Figura 3-1. Fluxo de dados entre as partes do C-Doc

3.2 Compilador C-Doc

O Compilador C-Doc é a parte do sistema responsável pela extração dos dados anotados em blocos de comentários incluídos nos códigos fonte do sistema. Para que o processo de compilação desses dados seja bem sucedido, é preciso que as regras de construção das anotações sejam respeitadas. Cada uma das tags implementadas exige a presença de pelo menos um parâmetro associado, assim como a linguagem CML e o Javadoc. Na **Figura 3-2**, é apresentado um exemplo hipotético de anotações válidas para o C-Doc.

```
/*@
@usedmemory 6
@codesize 7
@cachehit 1, 2
@cachemiss 3, 4
@pagefault 9
@power 1.23
@pesaccess c
@optaccess { a, b, c, d}
@*/
int main (void) {
        printf("Hello World!");
        return 0;
}
/*@
@function outraFuncaoQualquer
@task teste
@*/
int outraFuncaoQualquer(void) {
        printf("Função qualquer");
}
```

Figura 3-2. Exemplo de codificação C-Doc.

3.2.1 Processo de compilação

O processo de compilação de uma determinada linguagem pode ser subdividido em duas etapas: a fase de análise e a fase de síntese [18]. A etapa de análise subdivide-se em análise léxica e análise sintática. Na fase de síntese, é tratada a geração de código, concluindo assim o processo de compilação. Nos próximos parágrafos será apresentada cada fase do processo de compilação mais detalhadamente.

1. **Análise Léxica:** Esta fase tem como objetivo reconhecer as palavras contidas num documento de entrada. Estas palavras são denominadas *tokens* e são passadas para o analisador sintático. Ou seja, o analisador léxico tem como objetivo identificar o grupo de uma determinada palavra (inteiro, identificador, palavra-chave, etc) e passar o *token* que representa a palavra para o analisador sintático.
2. **Análise Sintática:** Esta fase é responsável pelo reconhecimento da lógica do programa. O analisador sintático recebe os *tokens* gerados pelo analisador léxico e tenta reconhecer

uma determinada lógica à medida que os *tokens* são consumidos. Em casos em que o analisador sintático não reconhece uma entrada como correspondente à sua gramática definida anteriormente, este emite um sinal de erro e geralmente aborta.

3. **Síntese:** Após a fase de análise, o compilador utiliza a Árvore Sintática Abstrata (do inglês, AST – *Abstract Syntactic Tree*) para sintetizar a saída do compilador.

A construção de um compilador é uma tarefa bastante complexa e demanda muito tempo. Atualmente, a maioria dos desenvolvedores e mantenedores de compiladores é auxiliada por geradores automáticos de *scanners* e *parser*.

Para a construção do compilador do C-Doc, utilizaram-se as ferramentas JFlex e Cup já citadas no capítulo anterior.

3.2.2 Scanner

Como demonstrado na **Figura 3-3**, um arquivo de especificação léxica é dividido em três partes separadas pela seqüência de símbolos %%. Em todas as três partes da especificação é possível adicionar comentários nos mesmos padrões da linguagem Java.

```
//Seção de Códigos do Usuário
/*
 * Todo Código adicionado nesta seção
 * será copiado para o início do scanner
 * gerado.
 */

%%

//Seção de Opções e Declarações
/*
 * Nesta seção são inseridas opções de
 * customização do scanner a ser gerado.
 */

%%

//Seção de regras léxicas
/*
 * Nesta seção é especificado o scanner
 * propriamente dito.
 */
```

Figura 3-3. Divisão das seções de um arquivo de especificação de um *scanner*.

Para o caso da ferramenta C-Doc, foram adicionadas às primeira e segunda seções informações como o pacote em que o fonte do *scanner* será salvo, o nome da classe, a codificação utilizada, bem como todas as palavras chave utilizadas pelo C-Doc. A **Figura 3-4** apresenta a transcrição da primeira seção e parte da segunda seção da especificação do scanner. A

Tabela 3-1 apresenta todas as palavras chave do C-Doc. É importante observar que as palavras chave utilizadas pelo C-Doc são, em sua maioria, aproveitadas da ferramenta Javadoc e da linguagem CML.

```
package br.upe.dsc.pcsbj.cdoc.core;
import java_cup.runtime.Symbol;

%%

%class CdocCompiler
%unicode
%cup
%line
%column
```

Figura 3-4. Primeira e parte da segunda seções da especificação do *scanner*.

A terceira e última seção do arquivo de especificação do *scanner* apresenta as regras léxicas. É a partir destas regras será construído o autômato equivalente à especificação dada. Estas regras são bastante simples: um símbolo específico é retornado para o *parser* sempre que um *token* for reconhecido. Caso algo não especificado nas regras seja reconhecido, uma mensagem de erro é apresentada e a geração do *scanner* é abortada.

Tabela 3-1. Palavras chave utilizadas pelo C-Doc

@author	@cachehit	@cachemiss	@code	@codesize
@deadline	@deprecated	@docroot	@excludes	@function
@link	@optaccess	@param	@pagefault	@period
@pesaccess	@phase	@power	@precedes	@processor
@release	@return	@scheduling	@see	@sends
@since	@task	@usedmemory	@version	@wcet

3.2.3 Parser

Apesar de não ter uma divisão tão bem definida como na especificação do *scanner,* o arquivo de especificação do *parser* também pode ser dividido em três partes: seção de códigos do usuário, seção de declaração de símbolos terminais e não terminais e a seção da gramática. A **Figura 3-5** apresenta as duas primeiras seções do arquivo de especificação do *parser* e nos próximos parágrafos será apresentada, em detalhes, a especificação da gramática do C-Doc.

```
package br.upe.dsc.pcsbj.cdoc.core;
import java_cup.runtime.*;

terminal WORD, INTEGER, COMMA, LCHAVE, RCHAVE, REAL, NPORP, BEGIN, END;
terminal ATTASK, ATPROCESSOR, ATPESACCESS;
terminal ATPHASE, ATRELEASE, ATWCET, ATDEADLINE, ATPERIOD, ATUSEDMEMORY, ATCODESIZE, ATPAGEFAULT;
terminal ATSCHEDULING, ATPRECEDES, ATEXCLUDES, ATSENDS, ATCACHEHIT, ATCACHEMISS, ATPOWER, ATOPTACCESS;
terminal ATFUNCTION;
terminal ATAUTHOR, ATCODE, ATDOCROOT, ATDEPRECATED, ATLINK, ATPARAM, ATRETURN, ATSEE, ATSINCE, ATVERSION;

non terminal cml_integer, cml_string, cml_expr, cml_text, cml_cache,cml_power, cml_scheduling;
non terminal cml_preexopt, lista_strings, javadoc, phrase, cml_sends, comentario, c_doc;
```

Figura 3-5. Sessões de códigos do usuário e definição de símbolos.

O Cup é compatível com o padrão EBNF – *Extended Backus-Naur-Form* para a especificação da gramática que gerará o parser. Na **Figura 3-6** pode-se observar um exemplo hipotético de uma gramática utilizando o padrão EBNF. Nesta gramática apresentada, podemos observar que as operações aritméticas (soma, subtração, multiplicação, divisão, módulo e negação) entre um ou dois `'expr'` ou números podem ser reduzidos ao elemento mais abstrato `'expr'`.

```
expr ::=    expr '+' expr | expr '-' expr | expr '*' expr
          | expr '/' expr | expr '%' expr | '(' expr ')'
          | '-' expr | number
```

Figura 3-6. Exemplo hipotético de gramática especificada no padrão EDNF.

No compilador do C-Doc, o elemento mais abstrato, ou seja, a raiz da árvore, é um bloco de comentário contendo atribuições em seu interior. A **Figura 3-7** apresenta como este elemento é especificado na sintaxe do Cup.

```
comentario ::=   BEGIN cml_text END
               | comentario BEGIN cml_text END
               ;
```

Figura 3-7. Definição do elemento mais abstrato do compilador do C-Doc.

A **Figura 3-8** apresenta a definição de mais dois elementos abstratos da gramática do C-Doc. O elemento cml_text, em sua abstração máxima, representa tudo que estiver contido dentro do bloco de comentário.

```
cml_text ::=  cml_expr
            | cml_text cml_expr
            ;

cml_expr ::=  cml_integer
            | cml_string
            | cml_cache
            | cml_preexopt
            | cml_sends
            | cml_power
            | cml_scheduling
            | javadoc
            | c_doc
            ;
```

Figura 3-8. Definição de elementos abstratos da gramática.

Na **Figura 3-9** são apresentadas as especificações das *tags* exportadas da ferramenta CML separadas por tipo de parâmetro recebido por cada *tag*. Pode-se observar, na **Tabela 3-2**, um relacionamento entre o elemento gramatical e as *tags* do CML, bem com um exemplo de expressão que seria corretamente avaliada para cada relacionamento.

Tabela 3-2. Relacionamento entre elementos abstratos e tags CML

Elemento	*Tags* Relacionadas	Exemplo
`cml_string`	@task @pesaccess @processor	@task MyClass
`cml_preexopt`	@precedes @excludes @optaccess	@precedes {MyClass, YourClass, FirstClass}
`cml_cache`	@cachemiss @cachehit	@cachemiss 3,10
`cml_sends`	@sends	@sends message_name, bus_name, time, receiver
`cml_integer`	@phase @release @wcet @deadline @period @usedmemory	@phase 10
`cml_power`	@power	@power 4.2
`cml_scheduling`	@scheduling	@scheduling NP

```
cml_string ::=  ATTASK phrase
              | ATPESACCESS phrase
              | ATPROCESSOR phrase
               ;

cml_integer ::=  ATPHASE INTEGER
               | ATRELEASE INTEGER
               | ATWCET INTEGER
               | ATDEADLINE INTEGER
               | ATPERIOD INTEGER
               | ATUSEDMEMORY INTEGER
               | ATCODESIZE INTEGER
               | ATPAGEFAULT INTEGER
               ;

cml_cache ::=  ATCACHEMISS INTEGER COMMA INTEGER
             | ATCACHEHIT INTEGER COMMA INTEGER
             ;

cml_preexopt ::=  ATPRECEDES LCHAVE lista_strings RCHAVE
                | ATEXCLUDES LCHAVE lista_strings RCHAVE
                | ATOPTACCESS LCHAVE lista_strings RCHAVE
                ;

cml_sends ::=  ATSENDS WORD COMMA WORD COMMA WORD COMMA WORD;

cml_power ::=  ATPOWER REAL;

cml_scheduling  ::=  ATSCHEDULING NPORP;
```

Figura 3-9. Elementos abstratos que representam as *tags* exportadas do CML.

Na **Figura 3-10**, é apresentado o restante dos elementos da gramática do C-Doc. Estes consistem nas *tags* exportadas do Javadoc e nas do próprio C-Doc. As definições destes elementos são bastante semelhantes às apresentadas anteriormente e todas as *tags* recebem como parâmetro um conjunto de palavras (uma frase).

```
c_doc ::=  ATFUNCTION WORD;

lista_strings ::=  WORD
                 | WORD COMMA lista_strings
                 ;

phrase ::=  WORD phrase
          | WORD
          ;

javadoc ::=  ATAUTHOR phrase
           | ATCODE phrase
           | ATDOCROOT phrase
           | ATDEPRECATED phrase
           | ATLINK phrase
           | ATPARAM phrase
           | ATRETURN phrase
           | ATSEE phrase
           | ATSINCE phrase
           | ATVERSION phrase
           ;
```

Figura 3-10. Elementos gramaticais referentes às *tags* do Javadoc e do próprio C-Doc.

3.3 Módulos de Saída

Até este ponto foram apresentados os detalhes da extração das informações contidas nos comentários dos arquivos fonte de um projeto. Nesta subseção é apresentada a parte da ferramenta responsável pela construção da documentação propriamente dita. Parte esta que funciona como *plug-in*, dando possibilidade de se criarem novos módulos para que se possa gerar documentações em outros formatos.

Para título de exemplos de criação de módulos de saída, os desenvolvedores do C-Doc fornecem dois formatos diferentes de apresentação: HTML e Manpages. Nesta subseção todos os exemplos mostrados serão referentes ao formato HTML.

Os módulos de saída devem implementar a interface Adaptador.java, contida no pacote br.upe.dsc.pcsbj.cdoc.adaptadores. Esta interface é apresentada na **Figura 3-11**.

```
package br.upe.dsc.pcsbj.cdoc.adaptadores;

import java.io.File;
import java.io.IOException;

import br.upe.dsc.pcsbj.cdoc.xml.Tags;

public interface Adaptador {

        public Tags lerXML(File pFile)throws IOException, ClassNotFoundException;

}
```

Figura 3-11. Interface a ser implementada por todos os módulos de saída

Além da obrigatoriedade de implementação da interface Adaptador.java, é aconselhável que se crie uma classe para definir um objeto que conterá as informações extraídas dos arquivos XML e as constantes utilizadas para a construção da documentação, como por exemplo as definições de cabeçalho e rodapé dos arquivos finais.

A extração dos dados dos arquivos XML pode ser feita da maneira que melhor convier para o desenvolvedor, mas é fortemente aconselhável que se utilizem bibliotecas de tratamento de arquivos XML disponíveis na internet. Um bom exemplo é a biblioteca XStream [19], que possui as seguintes características:

- open source;
- fácil utilização;
- gera arquivos XML geralmente legíveis;
- não exige que objetos sigam uma regra particular, por exemplo, não se exige que os objetos sejam serializáveis ou que só possuam atributos primitivos;
- efetua avaliações prévias nos arquivos XML em busca de inconsistências.

O formato do arquivo XML de onde se deve extrair as informações para a construção da documentação segue um padrão semelhante ao apresentado na **Figura 3-12**.

```
<object-stream>
    <list>
        <br.upe.dsc.pcsbj.cdoc.xml.Tags>
            <function>main</function>
            <autor>Joao</autor>
            <code>7</code>
            <cacheHit>1,2</cacheHit>
            <cacheMiss>3,4</cacheMiss>
            <pageFault>9</pageFault>
        </br.upe.dsc.pcsbj.cdoc.xml.Tags>
    </list>
</object-stream>
```

Figura 3-12. Exemplo de arquivo XML

Por fim, para se ter um módulo de saída, basta tratar as informações extraídas dos arquivos XML, colocá-las no formato desejado e gravar em arquivos.

Capítulo 4

Documentos Gerados

Este capítulo tem como objetivo apresentar os resultados obtidos pelo C-Doc, além de validar a proposta apresentada neste trabalho. Este se divide em três subseções. A primeira seção mostra os resultados apresentados pelo módulo HTML. A segunda seção detalha os resultados apresentados pelo módulo Manpages. A terceira e última seção apresenta a interface gráfica com o usuário.

4.1 Módulo HTML

O formato HTML foi escolhido para ser utilizado para demonstrar o funcionamento, devido à universalidade do formato e pela sua grande flexibilidade. As possibilidades de apresentação de uma documentação utilizando o HTML são muito grandes. Isto torna o HTML um dos formatos mais utilizados atualmente para apresentação de documentos.

Para este módulo, para cada função contida em cada arquivo fonte no projeto, é gerado um diretório com o nome desta função. Neste diretório são gerados três arquivos HTML: (i) um arquivo principal que separa a tela em três frames, (ii) um que apresenta as *tags* exportadas da linguagem CML e (iii) um que mostra as *tags* restantes.

A **Figura 4-1** exemplifica o resultado final de todo o processo de extração e construção da documentação feito pelo C-Doc. Na parte lateral esquerda pode-se observar um menu de acesso a todas as funções do projeto. O código deste menu pode ser encontrado na raiz do diretório onde estão depositados os diretórios de cada função. Nos *frames* centrais são adicionadas as informações extraídas do comentário da função selecionada.

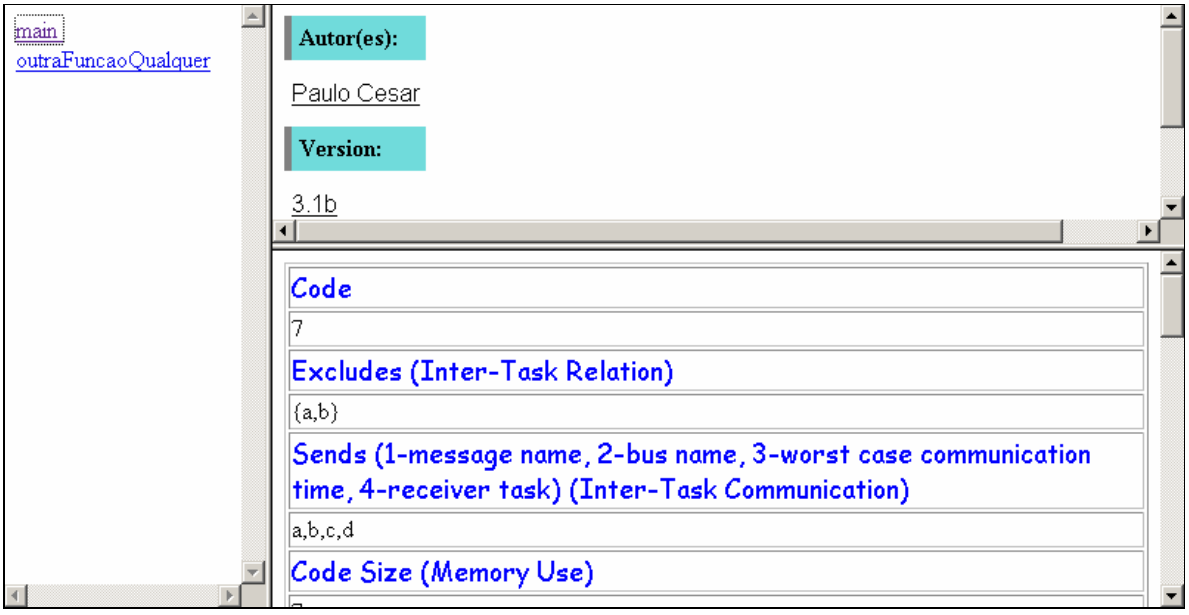

Figura 4-1. Saída do módulo HTML.

4.2 Módulo Manpages

O formato Manpages foi escolhido para apresentar os resultados do C-Doc, devido à sua difusão no mundo do Software Livre. A ferramenta capaz de ler e exibir informações contidas em Manpages pode ser encontrada por padrão em qualquer distribuição Unix *Like* (Linux, BSDs, Minix, Unix) acessada através do comando *man*.

```
.TH main(C) - C-Doc by nois

.SH NAME
main

.SH AUTHOR
Paulo

================================================================================

CML Tags

.SH EXCLUDES
{a,b}

.SH SENDS
a,b,c,d

.SH CODE SIZE
7

.SH CACHE HIT
1,2
```

Figura 4-2. Trecho de código Manpage gerado pelo C-Doc.

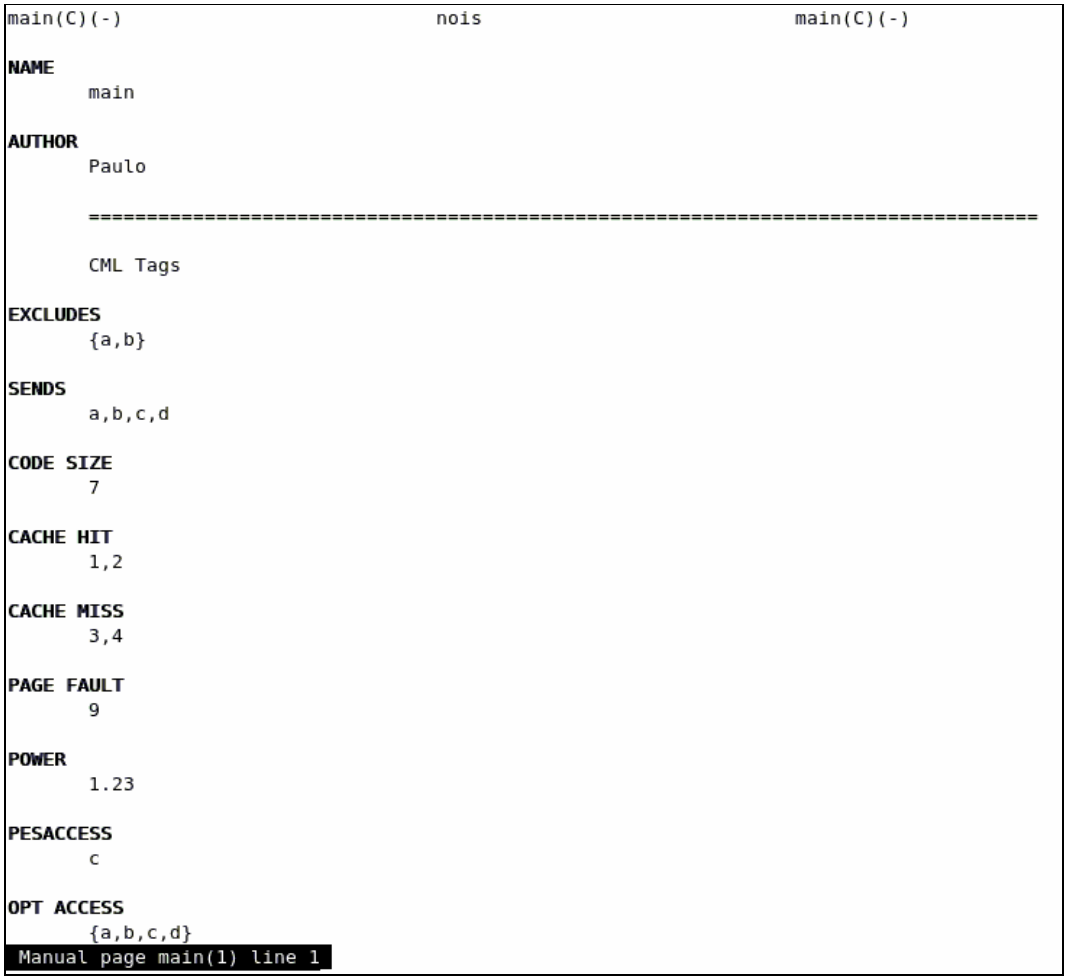

Figura 4-3. Exibição da Manpage gerada pelo C-Doc.

Os arquivos de Manpages foram idealizados para serem exibidos em ambientes *Shell* (sem interface gráfica), já que a maioria dos servidores não possui meios de exibição de formatos gráficos. Na **Figura 4-2** observa-se um exemplo de código de uma Manpage gerada pelo C-Doc. A exibição deste código utilizando a ferramenta *man* pode ser observada na **Figura 4-3**.

Capítulo 5

Conclusões e Trabalhos Futuros

A documentação de software é uma preocupação constante de equipes de desenvolvimento, principalmente para os gerentes de projetos que procuram tornar seu projeto o mais independente possível de seus desenvolvedores. Entretanto é muito difícil fazer com que todos os integrantes da equipe entendam a real importância da documentação de um projeto. Um dos motivos para que isso ocorra é o fato de a documentação não ter um retorno financeiro direto para a empresa, por isso, muitas vezes deixa a desejar ou é completamente abandonada.

Com o objetivo de facilitar o trabalho dos desenvolvedores durante a documentação e de reduzir custos nas fábricas de desenvolvimento de software, foi desenvolvida uma ferramenta denominada C-Doc, com função de extrair dados encontrados nos comentários adicionados pelos desenvolvedores nos códigos-fonte desenvolvidos e gerar a documentação do sistema a partir destes.

Além da ferramenta, este trabalho propôs um modelo de armazenamento de dados extraídos de comentários em códigos-fonte utilizando o XML, objetivando uma padronização no armazenamento e transporte destes dados.

Este capítulo apresenta as principais contribuições dadas por este trabalho, além dos trabalhos futuros.

5.1 Contribuições

Este trabalho teve como objetivo principal apresentar uma ferramenta genérica de auxílio na geração de documentação a partir de códigos-fonte gerados pelos desenvolvedores de software. Além desta, outras contribuições deste trabalho são detalhadas a seguir:

Ferramenta C-Doc

A ferramenta desenvolvida durante este trabalho procura automatizar o processo de documentação, reduzindo custos da produção de software e incentivando técnicas de reuso de código, técnica na qual uma boa documentação dos códigos é fundamental.

O C-Doc apresenta total compatibilidade com a linguagem CML, apresentada por Oliveira [4].

Modelo de Armazenamento e transporte

Este trabalho apresentou uma proposta de modelo XML para armazenamento e transporte de dados extraídos de códigos-fonte. A vantagem de se ter um modelo de armazenamento é a de serem criadas aplicações relacionadas compatíveis entre si.

Extensibilidade

O C-Doc foi concebido de forma que suas funcionalidades pudessem ser estendidas para diversas linguagens de programação utilizando o conceito de *plug-ins*, procurando atingir um maior número de desenvolvedores possível, e dando também a possibilidade de um certo desenvolvedor ou um grupo deles implementarem sua própria extensão para o C-Doc.

O C-Doc também é extensível quanto ao formato da documentação que poderá ser gerada a partir dele. Neste caso, a utilização do conceito de *plug-ins* torna possível a criação de novos formatos de saída além dos formatos de páginas *web* (HTML) e páginas de manuais do Linux (*manpages*) já apresentados anteriormente.

5.2 Trabalhos Futuros

Como foi apresentada anteriormente neste texto, a ferramenta C-Doc extrai somente informações contidas dentro dos blocos de comentários. Caso o C-Doc pudesse extrair informações contidas no código-fonte em si, teríamos uma maior integridade das informações da documentação gerada. Assim poder-se-ia associar um bloco de informações extraídas de um comentário a uma função ou método declarado imediatamente abaixo do bloco de comentário sem que se precise informar isto explicitamente.

Os módulos de saída existentes até o presente momento (HTML e Manpages) foram criados de forma experimental e com objetivo principal de mostrar a viabilidade da utilização da arquitetura modular do sistema. Estes módulos, apesar de serem totalmente funcionais, não apresentam uma saída visualmente amigável, já que este não era objetivo deste trabalho. Novos módulos de saída que apresentassem formatos mais amigáveis tornariam a ferramenta C-Doc mais profissional.

O processo de criação de módulos pode ser bastante trabalhoso para uma pessoa leiga em relação ao desenvolvimento de aplicações ou à linguagem Java. Seria de grande valia se uma ferramenta de auxílio para a criação de módulos fosse desenvolvida. Esta ferramenta possibilitaria a criação de novos módulos acopláveis ao sistema sem que se tenha conhecimento da linguagem Java. As informações necessárias à criação de um módulo seriam apresentadas ao sistema através de uma interface gráfica.

Bibliografia

[1] SOURCE CODE COMMENTS. Link: <http://www.literateprogramming.com/quotes_sc.html> Último Acesso: 14 de setembro de 2007.

[2] The Cranky User: The importance of documentation. Link: <http://www.ibm.com/developerworks/web/library/wa-cranky34.html>. Acessado em: 12de setembro de 2007.

[3] AHERN, Dennis M.; ARMSTRONG, Jim; CLOUSE, Aaron; FERGUSON, Jack R.; HAYES, Will; NIDIFFE, Kenneth E.. CMMI® SCAMPISM Distilled: Appraisals for Process Improvement. Ed. Addison Wesley Professional, 2005. 240 p.

[4] OLIVEIRA JR. F., LIMA R. M. F., CORNÉLIO M. L., SOARES S. C. B., MACIEL P., BARRETO R., OLIVEIRA JR. M., TAVARES E. *CML: The C modeling Language*. XI Brazilian Symposium on programming language, pp. 5-18, Natal – RN, 2007.

[5] MEYER, B. et al. Design by Contract: The Lessons of Ariane. IEEE Computer, 30(2):129-130, Janeiro 1997.

[6] LEAVENS, G. e CLIFTON, C. Lessons from the JML Project. Verified Software: Theories, Tools, Experiments. Department of Computer Science, Iowa State University, TR #05-12a, Abril de 2005.

[7] DBC for C (Design by Contract for C). Disponível em <http://www.onlamp.com/pub/a/onlamp/2004/10/28/design_by_contract_in_c.html>. Último acesso em 01/09/2007.

[8] PETERCHEN. *10 minutes to document your code.* The Code Project, 20 de janeiro de 2003. Link: http://www.codeproject.com/tips/doxysetup.asp. Acessado em 12 de novembro de 2007.

[9] IMAMURA M., *Using Doxygen.* Linux User Group at Georgia Tech, 30 de maio de 2002. Link: http://www.lugatgt.org/articles/doxygen/Acessado em 12 de novembro de 2007.

[10] SUN Microsystems, *Docklet Overview*. Link: http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/overview.html. Acessado em 13 de novembro de 2007.

[11] SUN Microsystems, *Taglet Overview.* Link: http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/taglet/overview.html. Acessado em 13 de novembro de 2007.

[12] Javadoc Tool. Disponível em <http://java.sun.com/j2se/javadoc/>. Acesso em 14 de outubro de 2007.

[13] Java Language. Disponível em < http://www.java.com/en/ >. Acessado em 12 de outubro de 2007.

[14] Doxygen Tool. Disponível em < http://www.stack.nl/~dimitri/doxygen/>. Acessado em 1° de novembro de 2007.

[15] Sun Microsystems. Disponível em <http://www.sun.com/>. Acessado em 03 de novembro de 2007.

[16] JFlex – *The Fast Scanner Generator for Java*. Disponível em <http://jflex.de>. Acessado em 04 de setembro de 2007.

[17] CUP – *Constructor of Useful Parsers.* Disponível em < http://www2.cs.tum.edu/projects/cup/>. Acessado em 15 de setembro de 2007.

[18] GRUNE, D. et al. Projeto Moderno De Compiladores. Ed. Campus. Rio de Janeiro. 2001.

[19] XStream. Disponível em <http://xstream.codehaus.org/>. Acessado em 23 de novembro de 2007.

[20] Flex – *The Fast Scanner Analyzer*. Disponível em <http://www.gnu.org/software/flex/>. Acessado em 26 de novembro de 2007.

[21] Yacc – *Yet Another Compiler-Compiler.* Disponível em < http://dinosaur.compilertools.net/yacc/index.html> Acessado em 26 de novembro de 2007.

Apêndice A

Manual de Utilização e Construção de novos Módulos

1. Introdução

Este manual mostra como deve ser utilizado o C-Doc para a geração de documentação para arquivos fonte em linguagem C, incluindo também informações fundamentais para a construção de um adaptador para o mesmo.

2. Visão Geral

Nesta seção, apresentaremos como se deve utilizar o C-Doc. Para isso utilizaremos o adaptador para HTML como exemplo, mas, a princípio, qualquer adaptador poderia ser utilizado.

2.1. A interface gráfica com o usuário

A interface gráfica do C-Doc é forma mais fácil de se beneficiar das funcionalidades do C-Doc e também o método aconselhável aos usuários leigos e que não se sentem à vontade com a linguagem Java.

A interface do C-Doc é bastante simples e de fácil utilização. Todos os seus componentes serão explicados, e para um melhor entendimento, podemos acompanhar tais explicações acompanhando a figura 1.

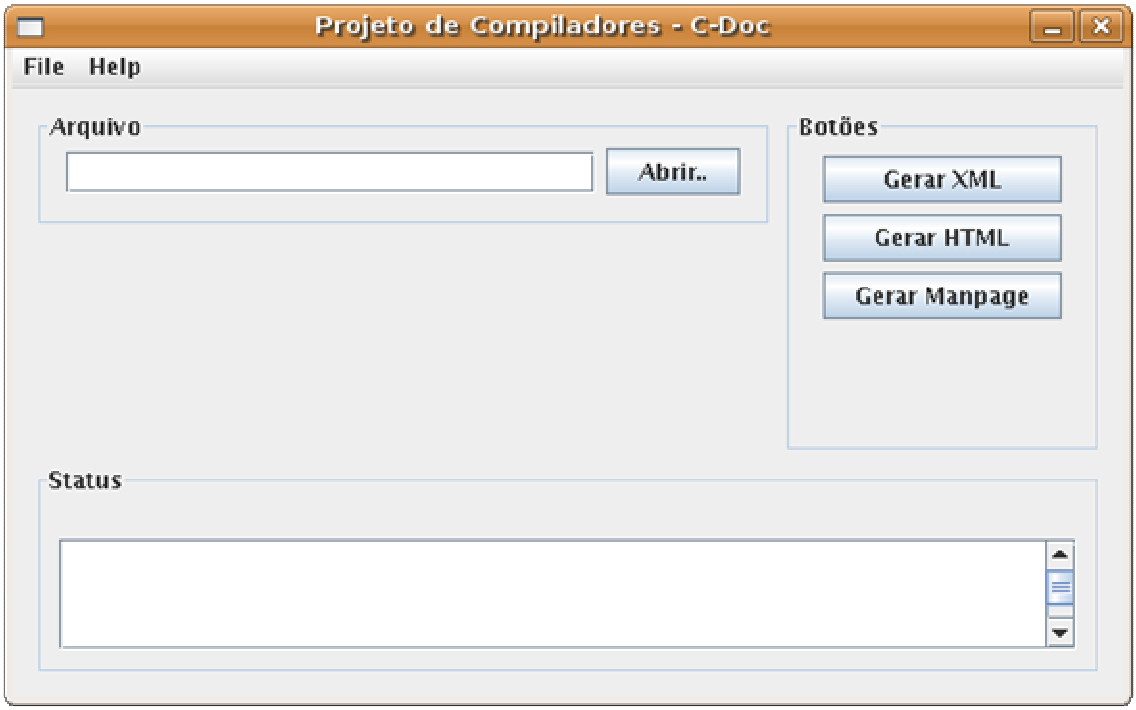

Ilustração 1 - Interface gráfica do C-Doc

O primeiro passo que devemos seguir é indicar qual o arquivo fonte que será utilizado para a extração das informações referentes aos comentários. Para isso basta indicar a localização completa do arquivo no campo "Arquivo". O botão "Abrir.." pode ajudar nesta tarefa, exibindo uma janela de navegação no sistema de arquivos. O próximo passo a ser seguido é a geração dos arquivos XML. Para isso deve ser clicado o botão "Gerar XML".

As informações contidas nos comentários do arquivo fonte serão extraídas e armazenadas em arquivos XML, um para cada bloco de comentário. A localização dos arquivos está indicada no arquivo Constants.java na *String* constante "*DIR_XML*".

Na interface original, estão disponíveis para utilização dois adaptadores: HTML e Manpages. Para gerar o formato de arquivo desejado, basta clicar no botão correspondente. Os arquivos serão armazenados no diretório indicado no arquivo Constants.java nas *Strings* constantes "*DIR_HTML*" e "*DIR_MANPAGE*".

Na interface gráfica é possível observar também uma janela de log, que informa as informações de status e de erro da aplicação.

3. Construindo um novo adaptador

Quando se deseja que uma documentação gerada pela C-Doc esteja em um formato diferente dos disponíveis junto com a ferramenta, é necessário que um adaptador seja construído. Para se construir um adaptador para o C-Doc é necessário um nível de conhecimento intermediário da linguagem de programação Java.

Para demonstrar a construção de um adaptador utilizaremos o adaptador HTML disponível com o C-Doc. É interessante lembrar que o usuário está livre para construir um novo adaptador HTML ou até mesmo alterar os adaptadores disponibilizados junto com o C-Doc.

Para a construção do adaptador HTML foi criado na raiz do projeto C-Doc (chamada javadoc) um pacote chamado adapHtml, onde serão armazenadas todas as classes referentes a tal adaptador. Neste pacote foram criadas duas classes, *Html.java* e *XmlToHtml.java.*

3.1. A classe *Html.java*

A classe *Html.java* contém as definições das *Strings* de construção do arquivo final. Estas *Strings s*ão padronizadas e invariantes, geralmente são tags de abertura e fechamento, cabeçalhos dos arquivos, componentes obrigatórios, etc.

Além da definição destas constantes são também criados métodos que retornam os valores destas constantes (métodos *get).* São estes métodos que serão utilizados pela classe *XmlToHtml.java* para acessar os valores das constantes geradas.

3.2. A classe *XmlToHtml.java*

A classe *XmlToHtml.java* é a classe responsável pela construção em si dos arquivos fde documentação finais. É nela que é imbutida a lógica necessária para a construção de arquivos concisos. Tal classe implementa a interface *Adaptadores.java* que exige que o método *lerXML* seja implementado. Este método geralmente é igual para todos os adaptadores e possui a implementação apresentada a seguir.

```
public Tags lerXML(File pFile) throws IOException, ClassNotFoundException {

        XStream stream_xml = new XStream();
        FileReader leitor_de_arquivo = new FileReader(pFile);

        // obtem o stream
        ObjectInputStream entrada =
stream_xml.createObjectInputStream(leitor_de_arquivo);

        // transforma o stream em List
        List lista = (List) entrada.readObject();
        // transforma list para Tags
```

Este Método lê um arquivo XML e, utilizando a biblioteca *Xstream*, extrai as informações do mesmo e constrói um objeto *Tags* que contém todas as informações pertinentes aos comentários.

O próximo passo é criar o método principal, que fará a construção do arquivo em si. Este método deve iniciar com a abertura do arquivo que será utilizado para guardar as informações dos comentários, como mostrado no bloco a seguir.

```
Tags pTags = this.lerXML(pFile);

        File arquivo = new File(Constantes.DIR_MANPAGE + "//" +
pFile.getName().replace(".xml", ""));

        FileWriter escreve_arq = null;
        PrintWriter escreve_format = null;

        arquivo.delete();

        if (arquivo.createNewFile()) {

                escreve_arq = new FileWriter(arquivo, true);
                escreve_format = new PrintWriter(escreve_arq);
```

O método segue com a construção das *Strings* que serão armazenadas no arquivo. O trecho de código a seguir exemplifica a forma de construção da *String* referente a *Tag @author*. Ao final da construção de todas as *Strings* necessárias, basta concatená-las e armazená-las no arquivo.

```
autores = pTags.getAutor();
if (!autores.equals("")) {

        htmlAutores += Html.getInicioHtmlAutores() + autores
                + Html.getFimHtmlAutores();
}
```

4. Boas práticas

Como boa prática de programação de adaptadores, é extremamente aconselhável que a classe *Constants.java* seja utilizada para armazenar os nomes de diretórios e arquivos. Desta forma teremos uma maior manutenibilidade e centralização de informações.

Sempre que um novo adaptador deva ser construído é aconselhável que se utilizem, como referência, as classes de adaptadores fornecidas junto com o C-Doc.

Apêndice B

Tabela de Construtores da linguagem CML

Construtor	Descrição	Formato	Programa/ Função	Requisitos não funcionais
@task	nome da Tarefa	String	F	-
@processor	processador alocado para a tarefa	String	P e F	alocação de recursos
@scheduling	estratégia de agendamento	NP (Não Preemptivo) ou P (Preemptivo)	F	agendamento
@phase	tempo de fase	Inteiro	F	restrição temporal
@release	tempo de liberação	Inteiro	F	restrição temporal
@wcet	pior caso de tempo de execução	Inteiro	P e F	restrição temporal
@deadline	tempo de deadline	Inteiro	F	restrição temporal
@period	tempo de período	Inteiro	F	restrição temporal
@precedes	tarefas que precedem esta tarefa	Lista de tarefas entre { e } separados por vírgula	F	Relação entre tarefas
@excludes	tarefas que excluem esta tarefa	Lista de tarefas entre { e } separados por vírgula	F	Relação entre tarefas
@sends	mensagem enviada	1-nome da mensagem, 2-nome do canal, 3-Pior caso de tempo de comunicação, 4-Tarefa recebedora	F	comunicação entre tarefas
@usedmemory	quantidade máxima de memória	Inteiro (Kb)	P e F	uso de memória
@codesize	tamanho do código binário	Inteiro (Kb)	P e F	uso de memória
@cachehit	número máximo de acesso a cache.	L1:Inteiro,L2:Inteiro	P e F	performance
@cachemiss	número máximo de falhas ao acessar a cache	L1:Inteiro,L2:Inteiro	P e F	performance
@pagefault	número máximo de page faults	Inteiro	P e F	performance
@power	Energia máxima	Real (Joules)	P e F	potência

	consumida			
@pesaccess	Acesso concorrente não é permitido	String (var)	F	consistência
@optaccess	Alguns acessos concorrentes são permitidos.	Lista de variáveis entre { e } separadas por vírgula.	F	consistência