SIM: Um Gerador Semi-Automático de Documentos

Carmen Lúcia Lodi Maidantchik

Tese submetida ao Corpo Docente da Coordenação dos Programas de Pós-Graduação em Engenharia da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências em Engenharia de Sistemas e Computação.

Aprovada por:

Profa. Ana Regina Cavalcanti da Rocha, D.Sc.
COPPE/UFRJ
(Presidente)

Prof. Jano Moreira de Souza, Ph.D.
COPPE/UFRJ

Prof. Zieli Dutra Thomé Filho, D.Sc.
COPPE/UFRJ

Prof. Marcos da Silva Borges, Ph.D.
NCE/UFRJ

Rio de Janeiro, R.J. - Brasil
Março de 1992

MAIDANTCHIK, Carmen Lúcia Lodi

SIM: Um Gerador Semi-Automático de Documentos

[Rio de Janeiro] 1992

xii, 137 pg., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1992)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Ambientes de Desenvolvimento de Software

I. COPPE/UFRJ II. Título (série)

aos amigos Giuliana, Maria e Xexéo

Agradecimentos

Ao Prof. Zichichi que acreditou e apoiou este projeto.

A Luisa Cifarelli que acompanhou o meu trabalho com interesse e dedicação.

A orientadora e, acima de tudo, amiga Ana Regina pelo carinho e dedicação.

Ao Jano que contribuiu na definição da tese e incentivou minha vinda ao CERN.

Ao Marcos Borges pelo incentivo à minha formação.

Ao Zieli pelo esforço dedicado à Colaboração entre a UFRJ o o CERN.

Ao Prof. Maculan pelo apoio à Colaboração entre a UFRJ c o CERN e por ser um exemplo de dedicação à pesquisa.

Ao Xeséo pelas suas contribuições profissionais, filosóficas e pessoais.

Ao Giuseppe que apoiou tecnica c moralmentc o desenvolvimento da. tese.

A COPPE-Oceânica e principalmente, Protásio. Carl. Mauricio c Antônio pela ajuda nas questões burocráticas c por serem amigos.

Ao Pedro pelo apoio moral c ajuda nas questões burocráticas.

A Paolo Palazzi, Bob van Eijk, Rene Brun, Fons Rademakers e Federico Carminati pelas sugestões e contribuições ao projeto.

Ao Seixas, Frank e Koki por não me deixarem esquecer o português e o modo de vida brasileiro.

Aos meus amigos do Brasil que apesar dos mais de 10.000 km de distância me fizeram sentir muito próxima de casa.

Aos meus amigos do CERN. alemães. americanos, argentinos. bolivianos, brasileiros, chineses, escoceses, espanhóis. franceses. holandeses. indianos, ingleses, italianos, iranianos, noruegueses, paquistaneses, peruanos, portugueses, suíços e turcos que me deram a oportunidade de conhecer o mundo em apenas uma cidade.

Ao World Laboratory, Projeto LAA e Grupo de Simulação de Monte Carlo que permitiram a realização desta tese.

Ao CNPq pelo apoio financeiro.

Finalmente, à minha mãe que contribuiu de forma decisiva para minha formação em todos os aspectos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

SIM: Um Gerador Semi-Automático de Documentos

Carmen Maidantchik

Março de 1992

Orientadora: Ana Regina Cavalcanti da Rocha, D.Sc.

Programa: Engenharia de Sistemas e Computação

Manuais de produtos de software são, geralmente, elaborados na fase final de projeto, após a implementação do sistema ou, mesmo, durante a vida útil do software. Assim sendo, esses manuais apresentam, muitas vezes, inconsistências pois modificações no código implicam em atualizações dos documentos, tarefa que é frequentemente esquecida. Em ambientes científicos, aonde são numerosas as mudanças em um software, esse problema se torna ainda mais crítico.

Este trabalho apresenta a ferramenta SIM, que gerencia as informações de produtos de software de forma a produzir seus respectivos manuais de utilização, de uma forma semi-automática.

A construção de SIM envolveu pesquisas sobre tipos de documentação, produção de documentos e desenvolvimento de software científico. Diversas técnicas e idéias para geração de documentos foram reunidas definindo uma metodologia, que focaliza a introdução da qualidade na documentação, implementada na ferramenta SIM.

SIM trata o problema da existência de inconsistências entre o código e sua documentação extraindo informações diretamente do arquivo-fonte e integrando as atividades de codificação e elaboração da documentação do software em um mesmo ambiente.

A ferramenta oferece roteiros para documentação, os quais podem ser adaptados à aplicação em desenvolvimento, e permite uma abstração sobre o formato dos manuais, desde que o conteúdo é manipulado separadamente do layout do documento. Através desse esquema de roteiros, SIM torna possível a produção de documentos em uma forma padrão.

SIM foi implementado no CERN[1], em Genebra, no contexto do Projeto de Colaboração UFRJ/CERN. Tinha-se, portanto, um ambiente científico e foram consideradas suas características particulares no que diz respeito ao desenvolvimento de software, tais como: carência de especificações e projeto, desenvolvimento incremental, manipulação de grande quantidade de dados e tendência a sofrer várias modificações e adaptações.

[1] Organização Européia para Pesquisa Nuclear (a abreviação provém da designação original francesa: Conseil Européen pour la Recherche Nucléaire).

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Semi-Automatic Generation of Documents with SIM

Carmen Maidantchik

March, 1992

Thesis Supervisor: Ana Regina Cavalcanti da Rocha, D.Sc.

Department: Computer Systems Engineering

Software manuals are usually produced at the end of the design phase, after the system implementation or even during the operational life. Because of that, most of the documents have inconsistencies since code modification implies the updating of the documents, what is frequently forgotten. In a scientific environment, where computing processes need numerous changes, the task of providing written information is even more critical.

This work introduces the SIM software, which manages software information in order to produce the users manuals, in a semi-automatic way.

The implementation involved research about documentation types, document production and scientific software development. Several techniques and ideas about the document generation were brought together, defining a methodology, supported by SIM, focusing the production of documentation with quality.

SIM deals with the problem of inconsistencies between code and documentation by extracting information directly from the source code, and by integrating the code implementation and the document generation activities in an unique environment.

This system provides guidelines for building up software documentation that can

be customized to the application requirements. Within this system the user has an abstraction over typesetting details of the final documents, since the content and the format of the manuals are handled separately. Through a guideline approach, SIM makes possible the documents production in a standard way.

SIM was implemented at CERN[2], in Geneva, within the Collaboration Project between UFRJ and CERN. There was, therefore, a scientific environment and its characteristics concerning software development were taken into account, such as: lack of specifications and project definition, incremental development, manipulation of large amounts of data and a tendency of suffering several modifications and adaptations.

[2] European Organization for Nuclear Research (the acronym comes from the original french name: **Conseil Europèen pour la Recherche Nucléaire).**

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo I

Motivação

Este capitulo descreve os fatores que motivaram o desenvolvimento desta tese, define o objetivo da tese e apresenta sua organização.

I.1 O Projeto de Colaboração UFRJ/CERN

A investigação realizada no CERN estuda as partículas fundamentais com as quais toda a matéria é constituída. O estudo de ínfimos constituintes da matéria juntamente com a exploração das imensas regiões do espaço procura compreender a estrutura do Universo. Embora essas duas áreas da pesquisa - uma estudando o muito pequeno e a outra, o muito grande - sejam, à primeira vista, diferentes, a compreensão da estrutura microscópica da matéria contribui para esclarecer os problemas da astrofísica e o mistério da origem do Universo.

Para se observar objetos verdadeiramente pequenos utilizam-se microscópios. A potência de um microscópio depende do comprimento de onda da radiação por ele emitida. Quanto menor o comprimento de onda mais detalhes podem ser observados. Reduzir o comprimento de onda significa aumentar a energia.

A área de estudos no CERN denomina-se Física de Altas Energias devido às elevadas energias fornecidas por máquinas que aceleram partículas estáveis (prótons, por exemplo), pela aplicação de forças elétricas e magnéticas, provocando colisões de modo a revelar seus constituíntes. Essas máquinas são chamadas *aceleradores.* O principal acelerador do CERN, o LEP[1], entrou em operação em fins de 1989 e trata-se de um acelerador circular de 27 km de comprimento, passando por território francês e suíço, a uma profundidade de 150 m.

Para se observar o que acontece quando as partículas interagem, os físicos precisam de detectores. Há mais de 50 anos atrás, Ernest Rutherford usava vídeos

[1] Large Electron-Positron collider.

fluorescentes para contar a chegada de partículas - a olho nu - e anotava os resultados em um caderno. Nas máquinas de hoje, milhões de partículas colidem a cada segundo. Logicamente, vídeos fluorescentes, olhos humanos e um caderno de anotações não são suficientes. Sofisticados equipamentos eletrônicos registram e monitoram os dados, que são, posteriormente, processados por computadores de alta velocidade.

O Projeto de Colaboração da UFRJ no CERN teve início em 1988 [1][2]. Pesquisadores brasileiros do Programa de Engenharia de Sistemas e Computação da COPPE participaram no desenvolvimento de sistemas em linha (conectados ao detector de partículas) e fora de linha (desconectados do cletector) na experiência DELPH1[2], nas áreas de Engenharia de Software e Banco de Dados. Atualmente, o grupo participa do Projeto LAA.

O Projeto LAA destina-se a pesquisar detectores para os aceleradores da próxima geração e a desenvolver diversos sistemas computacionais para simulações, via método de Monte Carlo. Seus projetos utilizam técnicas inovadoras e contêm requisitos mais exigentes, devido a grande quantidade de energia que será utilizada nesses super-aceleradores (SSC, LHC, etc.).

I.2 Objetivo da Tese

O desenvolvimento de sistemas computacionais se depara com diversos problemas, dentre os quais, dificuldades em entender, modificar e adaptar o software.

A Engenharia de Software tem como objetivo aumentar a qualidade de produtos de software e a produtividade de suas equipes de desenvolvimento. Esse objetivo, entretanto, somente é atingido através de uma série de procedimentos seguidos durante o processo de desenvolvimento do software.

Qualidade de Software diz respeito a todos os aspectos do software envolvendo o código propriamente dito e sua documentação.

[2]DELPHI juntamente com OPAL, L3 e ALEPH correspondem aos quatro maiores detectores do CERN.

Com o crescente desenvolvimento de sistemas computacionais, torna-se cada vez mais necessária a geração de documentos, em todos os níveis. Esses documentos envolvem desde a descrição do hardware e software (criação e implementação) até que procedimentos seguir para utilizá-lo [3].

Geralmente, a documentação é escrita pelos próprios desenvolvedores do software e não por documentadores profissionais [4]. Um erro comum é acreditar que um especialista em determinado assunto pode perfeitamente explicá-lo aos outros. Muitas vezes, esse tipo de pensamento resulta em uma documentação tão inadequada que leva a usuários insatisfeitos mesmo se o software for de alta qualidade.

A geração de documentação de boa qualidade é um problema de grande interesse e importância para a Engenharia de Software. E uma parte vital de qualquer projeto de sistemas computacionais e, quando negligenciada, pode trazer erros desastrosos e perda de tempo e dinheiro na produção e manutenção do software. A existência de uma documentação adequada, sem dúvida, reduz esforços e custos durante o desenvolvimento do software.

Atualmente, não há um esquema geral para produzir documentações de software. Entretanto, a importância desse problema é bastante reconhecida e novas técnicas começam a aparecer.

Os métodos e técnicas utilizados durante o desenvolvimento do software, orientam a equipe de trabalho a lidar com a complexidade do sistema em construção e, portanto, influenciam a qualidade do produto final [5].

A complexidade dos experimentos em Física de Altas Energias cresce continuamente. Os físicos trabalham com estudos de simulação que requerem cada vez mais pesquisas. A quantidade de dados a serem armazenados, interpretados, gerenciados e controlados requerem sistemas especializados para realizar essas tarefas de modo eficiente e amigável.

Este trabalho define uma ferramenta de apoio à documentação de Software Científico, considerando suas peculiaridades: a ferramenta SIM (Software Information Manager).

SIM faz parte de um único ambiente, onde o usuário pode construir a docu-

mentação relativa ao programa, ao mesmo tempo em que o escreve. A geração dos documentos (Manual do Usuário, de Referência e de Manutenção do Programa) é auxiliada por roteiros fornecidos pelo ambiente, compostos por palavras-chave (templates), e que podem ser configurados pelo próprio usuário. Aos documentos são integrados comandos do formatador de textos LaTeX [6]. SIM permite uma abstração sobre o formato final dos manuais já que as informações lógicas são separadas das informações físicas.

SIM é integrado ao CMZ [7] (Code Management System with Zebra), um software desenvolvido no CERN, para gerência de código (Fortran ou C) e que inclui diversas facilidades, tais como: edição, compilação, gerência de versões e verificação de variáveis.

I.3 Organização da Tese

Os capítulos II, III e IV apresentam as pesquisas realizadas que levaram à escolha de uma metodologia para produção de documentos do software em um ambiente científico. Em detalhes:

- O segundo capítulo apresenta um estudo sobre as informações do software, baseado na identificação dos tipos de documentação e na classificação de sistemas para produção de documentos;
- No capítulo III são apresentadas as características do software científico. Descreve-se o contexto do CERN, no que diz respeito ao desenvolvimento de software e às ferramentas de apoio à construção de programas HEP[3]. Ao final, apresenta-se um esquema sobre a produção de documentos nesse centro de pesquisas;
- O capítulo IV reúne uma série de idéias e técnicas para geração, com qualidade, dos manuais de utilização de um programa. São, portanto, apresentados o que se entende por qualidade de uma documentação, os princípios básicos

[3] High Energy Physics.

para gerar uma boa documentação, uma organização no processo de produção de documentos (documentação estruturada), a descrição de um ambiente integrando as atividades de codificação e documentação e, uma proposta para o conteúdo dos manuais de utilização do software e a descrição de um ambiente integrando as atividades de codificação e documentação.

O capítulo V apresenta a ferramenta SIM, cuja implementação se baseia na metodologia escolhida após a pesquisa realizada. São identificadas as informações manipuladas por SIM e apresentados a arquitetura do ambiente implementado e o processo de produção de documentos. Explica-se como o usuário define o conteúdo e o formato do documento. Ao final, descreve-se alguns detalhes de sua implementação.

O último capítulo corresponde à conclusão do trabalho. São apresentadas observações quanto à importância e a aplicabilidade do trabalho e sugestões para futuras pesquisas.

O apêndice A exemplifica como utilizar a ferramenta SIM para produção de manuais. O apêndice B apresenta um exemplo de um documento gerado automaticamente pela ferramenta, contendo todos os ítens para documentação de programas propostos por SIM e, seguindo o formato padrão oferecido pela ferramenta. O apêndice C corresponde ao manual da aplicação PHOTOS, cuja elaboração foi auxiliada por SIM.

I.3.1 Notação Utilizada

Neste documento, o negrito é utilizado para introduzir conceitos e termos:

comunicabilidade: o texto deve ser escrito...

"Typewritter" simula a saída de um vídeo de computador, trechos de programas:

```
PROGRAM {name}
```

Itálico enfatiza informações:

A Engenharia de Software tem como objetivo aumentar...

Termos em inglês são apresentados em "sans serif":

os sistemas WYSIWYG - what **you** see is what **you get**, que...

Capitulo II

Informações do Software

Neste segundo capitulo é apresentado um estudo sobre as informações do software, baseado na identificação dos tipos de documentação e na classificação de sistemas para produção de documentos.

II.1 Tipos de Documentação

O termo *software* refere-se a um programa e a todas as informações associadas e materiais necessários para auxiliar sua instalação, operação, reparos e evoluções [8]. Portanto, o termo software não se refere somente ao código-fonte mas também a sua documentação.

Documentação não é, necessariamente, apenas um conjunto de informações escritas. A documentação de um software inclui:

- relatórios que descrevem os produtos gerados durante seu desenvolvimento;
- comentários no código-fonte;
- manuais de utilização do sistema;
- informações **on-line.**

Assim sendo, todas as informações utilizadas para comunicação com usuários finais ou outros profissionais envolvidos no processo de construção e manutenção do sistema devem ser consideradas como parte da documentação do software.

Uma documentação adequada e de boa qualidade contém descrições desde a especificação do produto até à utilização do sistema computacional.

II.1.1 Relatórios do ciclo de desenvolvimento do Software

Independente do método de desenvolvimento de software escolhido, pode-se definir cinco tipos de documentação de software:

- da Especificação: descreve os objetivos e uma definição geral dos requisitos do sofware a ser construído;
- do Projeto: descreve os conceitos do sistema a ser construído, seus objetivos, requisitos e as necessidades dos usuários;
- da Construção: se refere aos detalhes da construção, incluindo a descrição do ambiente operacional, métodos e técnicas utilizados na implementação;
- da Manutenção: contém a descrição do sistema (parte da documentação do Projeto e da Construção), incluindo alguns dados, tais como: erros conhecidos, limitações e os procedimentos a serem seguidos quando o sistema precisar ser extendido ou modificado.

O conjunto dessas informações serve para [9]:

- permitir que os gerentes verifiquem se os requisitos foram atingidos;
- registrar dados técnicos permitindo coordenar futuros desenvolvimentos e modificações do software;
- facilitar o entendimento entre os integrantes da equipe de desenvolvimento e manutenção;
- informar aos usuários a funcionalidade e as capacidades do software para que eles possam verificar se o programa serve às suas necessidades.

II.1.2 Comentários no código-fonte

Geralmente, programas são lidos e manipulados por diversas pessoas, seja durante sua construção seja durante sua manutenção. Muitas vezes, os responsáveis

pela atualização, adaptação ou evolução do código não são aqueles que participaram de seu desenvolvimento. Devido a esse fator, deve-se escrever o código-fonte de forma a que possa ser entendido com relativa facilidade por pessoas que não participaram de seu desenvolvimento.

Existem diretrizes que orientam a geração de comentários em uma forma padrão. Essas incluem, por exemplo, a indicação de que cada rotina deve conter um cabeçalho com as seguintes informações: funcionalidade, parâmetros de entrada e saída, autor, data inicial e da última atualização, etc. Outra diretriz corresponde à descrição de cada ação do programa, implementada em um bloco de instruções, através de um comentário em linguagem natural, de modo que, o conjunto desses comentários explique o funcionamento do programa.

II.1.3 Manuais de utilização do sistema

O ideal seria que não existisse nenhuma informação escrita sobre o software, ou seja, que o sistema computacional fosse suficientemente fácil de ser entendido, utilizado e modificado sem necessidade de explicações adicionais [10]. Entretanto, como essa é uma meta, atualmente inatingível, faz-se necessário documentar os sistemas computacionais. Baseado em [11], [12], [10], [9] e [13] os manuais de utilização do sistema podem ser classificados em:

- Manual do Usuário: descreve as funções realizadas pelo software, através de uma terminologia adequada, ou seja, de acordo com o ambiente para o qual o sistema foi desenvolvido. Deve descrever a aplicabilidade do sistema, seu propósito, dados de entrada, resultados, arquivos acessados *e* exemplos;

- Guia do Operador: descreve o ambiente operacional do sofware e se destina ao grupo de suporte operacional. Fornece informações para a instalação do sistema e que dispositivos são acessados pelo sistema;

- Manual de Manutenção do Programa: descreve a implementação de cada parte do sistema, suas variáveis e arquivos manipulados. Esse manual deve conter

detalhes das rotinas para auxiliar futuras modificações e melhorias, e servir como uma referência na detecção de erros;

– Manual de Referência: contém informações básicas sobre o sistema. Geralmente, é destinado a usuários com alguma experiência em aplicações da mesma área do sistema.

II.1.4 Informações on-line

Esse tipo de informações distingue-se das classes anteriores devido a suas capacidades de interação com o usuário. São desenvolvidas de acordo com as características do hasdware e sistema operacional onde o sistema é executado.

Podem ser vistas como um método para reduzir a quantidade de material impresso e contribuem para a implementação de um ambiente consistente, dado que o próprio sistema orienta o usuário em seu trabalho.

Podem ser classificadas em quatro tipos [10]:

– Sistemas de mensagens e erros: oferecem informações sobre o estado do processamento e orientam o usuário na identificação de erros;

– Guias de referência (menus ou Help): quando o usuário necessita saber como realizar uma tarefa específica;

– Tutorial: introduz um novo usuário na operação do sistema;

– Entrada e recuperação de dados: o sistema oferece campos para a entrada e visualização dos dados manipulados pelo programa.

Um recente desenvolvimento na área de informações on-line é a utilização de interfaces gráficas. Uma grande vantagem é que, ao substituir palavras por imagens gráficas (ícones), a interface se torna independente de idiomas. Essas representações podem descrever com simplicidade operações detalhadas e/ou complexas.

11.2 Produção de Documentos

A produção de documentos é composta por duas etapas [14]:

- definição do conteúdo e da estrutura da documentação;
- geração do documento a partir da especificação de sua aparência.

A primeira etapa é tipicamente chamada *edição* enquanto a segunda é conhecida como *formatação*. Mais precisamente, forinatação se refere ao lay-out dos objetos que compõe um documento apresentado em vídeo ou papel.

Um sistema para a produção de clocumentos envolve quatro tarefas:

- Edição: corresponde a criação e modificação de objetos abstratos[1];
- Importação e Exportação: corresponde à importação e exportação de objetos elaborados em outros sistemas;
- Formatação: corresponde a transformação de objetos abstratos em objetos concretos a partir de sua definição física;
- Visualização: de objetos concretos através de dispositivos de visualização ou de impressão.

A figura II.1 mostra a arquitetura de um sistema para produção de documentos.

II.2.1 Sistemas para produção de documentos

Os sistemas para produção de documentos começaram a ser desenvolvidos nos anos 60. Os primeiros sistemas eram simples evoluções de programas de edição que organizavam as palavras em linhas de mesma extensão e essas, em páginas. Poucos comandos podiam ser incluídos no texto para controlar o formato do documento, tais como, inserção de espaços e alimentação de linhas, devido à limitação das impressoras da época.

[1] um objeto é definido como *abstrato* quando não são considerados os componentes que especificam sua aparência física e, *concreto*, caso contrário.

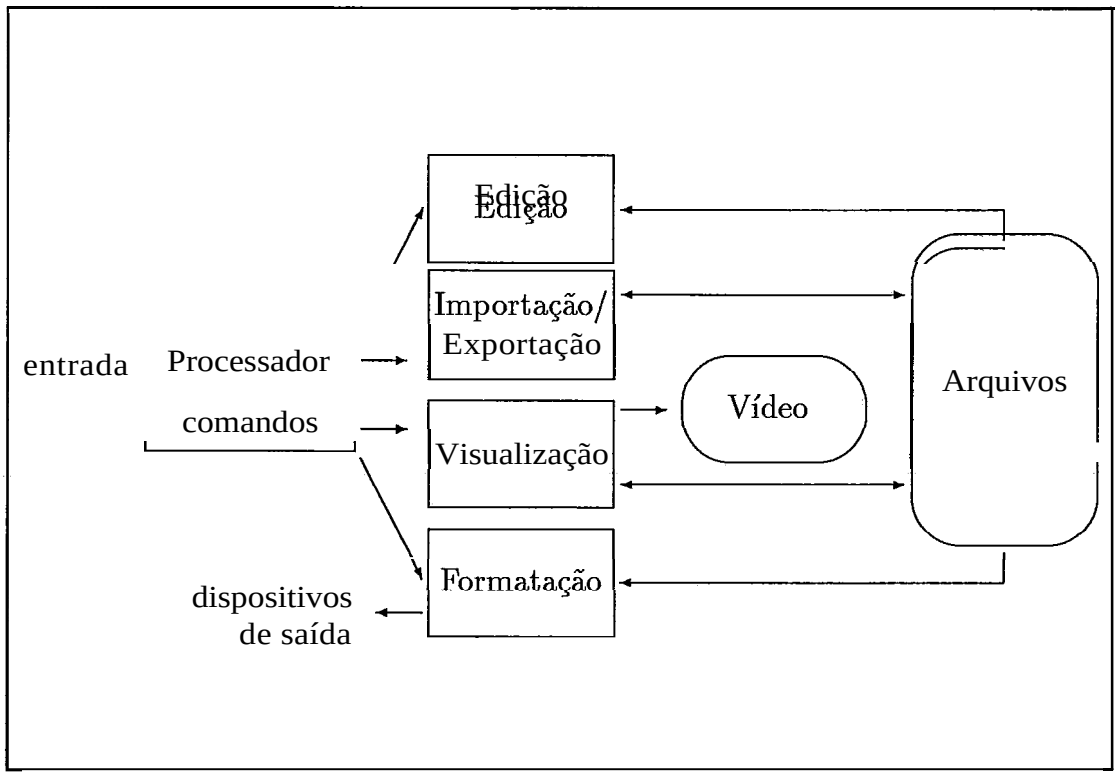

Figura 11.1: Arquitetura de um sistema para produção de documentos

A linguagem, na qual os comandos eram expressos, era de baixo nível, semelhante ao Assembler. Com o desenvolvimento das linguagens de programação, novas instruções (macros) foram criadas oferecendo comandos de mais alto-nível. Esses comandos também puderam ser aprimorados devido às novas impressoras.

Os programas foram desenvolvidos com o objetivo de manipular outros elementos que não fossem textos, tais como: tabelas, fórmulas matemáticas e gráficos. Surgiram, também, outras ferramentas, por exemplo, para manipular referências bibliográficas ou detectar erros ortográficos. Nesse contexto, Donald Knuth definiu TEX [15][16][17], cujo objetivo era obter tipografia de alta qualidade.

Além dessas ferramentas, começaram a aparecer sistemas profissionais para geração de documentos para computadores como, por exemplo, o SGML[2] (Standard Generalized Markup Language) da IBM, que é uma linguagem para formatar textos e descrever figuras. Nos fins dos anos 70, o aparecimento das impressoras a laser permitiram o desenvolvimento de formatadores com uma melhor qualidade de

[2]SGML (padrão ISO 8879).

tipografia.

O último estágio desses tipos de sistemas apareceu com a utilização de vídeos gráficos onde era possível a visualização da mesma imagem que é produzida pela impressora. São os sistemas WYSIWYG - **what you see is what you get,** que utilizam toda a potencialidade do hardware, através de vários tamanhos e tipos de letras e da incorporação de gráficos, oferecendo uma interface de alto-nível.

Classificação dos Sistemas

Segundo André et. al. [18], as ferramentas computacionais que auxiliam o processo de elaboração de documentos, no que se refere a facilidades para a edição e definição do formato final, podem ser classificadas em:

- editores e processadores de textos;
- formatadores;
- editores dirigidos por sintaxe;
- ferramentas especializadas.

a) Editores e processadores de textos

Editores, de propósito geral, somente produzem documentos de forma rudimentar os quais ainda devem ser submetidos a formatadores.

Os sistemas processadores de texto, originariamente derivados dos editores, apresentam funções para elaboração do formato do documento [19]. Eles permitem centralizações e justificações, escolha dos tipos de letra, palavras sublinhadas e, modificação de determinados parâmetros como espaçamento entre linhas e definição das margens. Mas, ainda cabe ao usuário numerar capítulos, seções e figuras, gerenciar as referências e construir índices, listas de figuras, tabelas e o sumário. Além disso, qualquer mudança implica em uma atualização manual das informações. A disposição de parágrafos em páginas, evitando, por exemplo, parágrafos começando na última linha de uma página, também é realizada manualmente.

Para documentos mais estruturados, outros processadores de texto podem ser utilizados, classificados como outlining systems, MicroSoft Word, por exemplo. Alguns oferecem uma geração automática do sumário através da hierarquização de seções, apresentadas na tela por identações. A manipulação direta de seções (reordenação, deleção, inclusão, etc.) podem ser obtidas por simples operações. O usuário pode ter uma visão geral do documento requerendo somente os cabeçalhos.

Os outlining systems solucionam alguns problemas dos processadores de texto normais mas, não oferecem todas as facilidades de formatação. Finalmente, dentre esses sistemas interativos encontram-se os específicos para geração de determinadas classes de documentos (técnicos, por exemplo).

b) Formatadores

Os sistemas da sub-seção anterior são interativos: o usuário visualiza na tela a imagem do documento que está manipulando e cada modificação aparece imediatamente. Assim sendo, ele pode reagir, rapidamente, se o resultado obtido não é o esperado. Os formatadores não são ferramentas interativas: os comandos são submetidos e o resultado só é visualizado depois de um certo tempo. O usuário não pode intervir no processo de execução. Caso o resultado não seja satisfatório, ele deve usar um editor para modificar o arquivo descrevendo o que deseja.

Apesar desse ponto negativo, há uma compensação: a maioria dos formatadores gerenciam automaticamente referências, compilam o sumário e índices. Esses programas também possuem uma melhor qualidade de tipografia e podem manipular grandes documentos, o que geralmente é difícil com programas interativos.

Novas ferramentas, tais como XDVI - previewer, estão sendo desenvolvidas para solucionar o problema da falta de interatividade dos formatadores. Através desses visualizadores obtêm-se na tela gráfica do computador a mesma representação das páginas quando impressas, economizando-se assim, tempo e papel. O usuário pode identificar erros no vídeo embora não possa corrigí-los através dessa ferramenta. Ele deve editar o arquivo original e achar o comando ou a parte do texto que está incorreta.

c) Editores dirigidos por sintaxe

Geralmente, essa classe de editores corresponde a sistemas interativos cujo objetivo é gerenciar a elaboração de programas.

Ao contrário dos editores descritos anteriormente, os editores dirigidos por sintaxe "conhecem" a linguagem na qual o programa, por eles processado, é escrito. Por exemplo, pela extensão do arquivo a ser editado, o VAX Language-Sensitive Editor (VAXLSE) reconhece qual a linguagem do programa (.FOR, Fortran; .PLI, PL/I ou .PAS, PASCAL).

Esses editores orientam a construção de programas, oferecendo um esqueleto (figura 11.2) do programa onde o usuário pode completar com os comandos da linguagem.

```
        [options_stmt]
        PROGRAM {name}

C[header_comments]
        [implicit_stmt]
        COMMON {[/{common_blk_name}/] {common_elm}...}...
        DATA {{data-elm)  ... /{[{unsign_int_cons}*]{cons}}...}/
        1  [{data_elm}... /{[{unsign_int_cons}*]{cons}}...]...
        [specification_statement]...

[CDEC$_DIR]...
[CPAR$_DIR]...

        CALL {program_name}[({arg}...)]
        [executable_statement]...
        END

        [program_unit]...
```

Figura 11.2: Esqueleto de um programa Fortran no LSE

Alguns editores dirigidos por sintaxe são capazes de fazer a análise do programa.

Esses tipos de editores fazem parte de uma série de ferramentas pertencentes a um ambiente de programação ou de desenvolvimento de software e, normalmente, existem outros programas para produzir documentos. As ferramentas para produção

de documentos podem ser derivadas diretamente do próprio editor dirigido por sintaxe, como no caso do software *Mentor* [20].

d) Ferramentas especializadas

A maioria dos sistemas descritos acima são destinados à manipulação de textos. Entretanto, muitas vezes, diagramas, figuras, tabelas e fórmulas matemáticas ou químicas devem ser incluídas nos documentos. Consequentemente, há dois problemas: como incluir esses elementos no corpo do documento e como produzí-los.

Dependendo do tipo de sistema, o elemento a ser incluído pode ser criado e manipulado pelo próprio sistema, que pode ser tanto um formatador quanto um processador de texto. No caso dos formatadores, a sua linguagem inclui comandos para descrição de fórmulas, tabelas ou gráficos (TEX, LATEX, por exemplo). No caso de sistemas interativos, os programas geralmente incluem um ou vários editores especializados para manipular um tipo específico de elemento.

Um elemento pode ser construído por uma ferramenta especializada (sistemas CAD, por exemplo) e incorporados ao documento através de comandos especiais que permitam o seu redimensionamento. Deve-se, entretanto, ter a representação do elemento através de descrições padronizadas, tais como, *PosScript*. Mas, para realizar qualquer modificação deve-se chamar novamente a ferramenta especializada, gerar a representação e então, importar o elemento no documento.

Os formatadores manipulam os elementos de um documento de modo estruturado e com comandos de alto-nível. As tabelas não são descritas pela posição e comprimento de cada linha ou de cada palavra mas, sim através da organização das colunas e linhas e do seu conteúdo. Da mesma forma, fórmulas são descritas pela estrutura matemática e não pelo posicionamento de cada caracter.

11.3 Considerações

A identificação dos tipos de documentação permitiu classificar as diversas informações dos produtos de software, verificando que os comentários no código-fonte

ou algumas informações **on-line** também devem estar presentes nos manuais de utilização do sistema. Através da definição de cada tipo de documentação, a tarefa de se produzir um manual de utilização, por exemplo, é simplificada pois o objetivo desse documento foi definido previamente.

A classificação de sistemas para produção de documentos possibilitou conhecer o histórico desses programas e a sua evolução. Essa pesquisa auxiliou a escolha de um sistema adequado para produzir os manuais de utilização do software.

Capítulo III
Software Científico

Este capítulo apresenta as características do software científico. Descreve-se o ambiente CERN, onde foi implementada a ferramenta SIM, no que diz respeito ao desenvolvimento de software. São apresentadas as ferramentas de apoio à construção de programas HEP e um esquema sobre a produção de documentos nesse centro de pesquisas.

III.1 Características

Engenheiros e cientistas, que escrevem suas próprias aplicações, utilizam as mais elementares ferramentas de software (editores e compiladores) para desenvolver sistemas complexos [11]. Isso porque as ferramentas de apoio ao desenvolvimento de software são inspiradas em problemas de ambientes comerciais e portanto, não são adequadas ao ambiente científico.

Para os engenheiros e cientistas que não são especialistas em ciência da computação solucionar o problema do ponto de vista de seu domínio é o primeiro passo, bastante anterior ao de aprender novos métodos ou linguagens de programação mais avançadas para melhor implementar a solução. Em sua maioria, esses cientistas aprenderam a programar em Fortran para desenvolver aplicações grandes e complexas, em um domínio de problema no qual são especialistas.

Esses profissionais utilizam, em geral, um método próprio baseado em desenvolvimento incremental e focalizado na produção do código: uma versão incompleta do programa é testada a procura de erros, decodificada, incrementada e assim sucessivamente. Com o objetivo de alcançar rapidamente soluções, esse método desconsidera conceitos e métodos sobre especificação dos requisitos, projeto ou documentação.

O resultado é frequentemente um único programa, difícil de ser modificado, reutilizado, transportado e decodificado, e que somente o próprio autor consegue entendê-

lo. Ou seja, o produto final é de baixa qualidade, do ponto de vista da Engenharia de Software, o que só se torna aparente quando começam a aparecer **bugs** de "forma misteriosa".

A qualidade de um software está diretamenta associada à qualidade de seu processo de desenvolvimento, ou seja, como foi organizado esse processo e quais foram os métodos, técnicas e ferramentas utilizadas durante o seu desenvolvimento.

Para promover a qualidade do software, que a comunidade científica escreve e utiliza, são necessárias ferramentas de desenvolvimento que considerem tanto sua resistência a mudanças quanto o uso de uma metodologia informal e limitada.

III.1.1 Complexidade

A complexidade de um produto de software dificulta sua produção e envolve consideráveis esforços durante o seu desenvolvimento.

A complexidade do software científico é derivada de diversos fatores [21], tais como:

- Estrutura, relativa à complexidade de seus subsistemas, módulos, etc;
- Comportamento, definido através de:
  - funções, ou seja, das transformações dos componentes de um estado para outro;
  - processos ou fluxo de controles;
  - reações aos eventos aos quais o sistema deve responder;
  - tempo de resposta.
- Aplicação, que corresponde aos requisitos que o sistema deve respeitar, tais como:
  - recuperação de informações;

    segurança dos dados;

- proteção contra eventos externos;
- interfaces com o usuário e com outros sistemas;
- operações que mantém o sistema.

– Ambiente de desenvolvimento, relativo à dificuldade das atividades de:

- projeto e implementação;
- documentação e treinamento;
- certificação e testes.

111.2 O CERN e seu contexto no desenvolvimento de software científico

Os experimentos que estão sendo construídos no CERN são cada vez maiores e mais complexos, envolvendo a coleta de inúmeros eventos [22]. As colaborações com o CERN são numerosas e envolvem instituições e indivíduos de diferentes nações.

Os métodos computacionais constituem uma ferramenta indispensável nas pesquisas em Física de Altas Energias. Os sistemas auxiliam a coleta e seleção dos dados, simulam experimentos e novos mecanismos a serem construídos, gerenciam bibliotecas de programas e armazenam dados que podem ser manipulados por diversos grupos [23][24].

O *software científico*, em geral, é desenvolvido pelos próprios pesquisadores (no caso do CERN, os físicos) e engloba características peculiares, tais como:

– carência de documentação, principalmente resultante das fases de especificação e projeto;

– tendência a sofrer diversas modificações e adaptações, de acordo com as necessidades que surgem com sua utilização;

– manipulação de uma grande quantidade de dados;

– baixa qualidade no que se refere à interface com o usuário;

– manuais de utilização do software desatualizados e incompletos.

Esses problemas se agravam durante a vida útil do software, pois os sistemas não são desenvolvidos utilizando-se métodos adequados. Isso torna a tarefa de produzir documentos eficientes bastante crítica. Por outro lado, a comunidade científica já reconhece a necessidade de melhorar a qualidade do software utilizado bem como aumentar a produtividade dos físicos e programadores.

III.2.1 Controle e Avaliação da Qualidade de Programas

O CERN, no que diz respeito ao desenvolvimento de software científico, não foge às características de outros centros de pesquisa, focalizando a construção do código, que pode ser dividida em duas etapas: concepção e revisão sintática. No primeiro passo, o problema é formulado e a solução é implementada em uma linguagem de programação. No segundo passo, revisa-se o código com o objetivo de escrevê-lo de forma correta.

Baseado nesse "método", ferramentas foram desenvolvidas para auxiliar a construção de programas HEP e a comunicação entre os diversos membros das equipes, provenientes de diferentes institutos e países. Algumas dessas ferramentas são apresentadas a seguir.

Comentários no Código-fonte

Um estudo sobre avaliação da qualidade dos programas do experimento DELPHI foi realizado dentro da colaboração entre a UFRJ e o CERN.

Palermo [25][26] propõe diversos critérios para avaliar a qualidade do software produzido no contexto da colaboração DELPHI. Entre os atributos identificados destaca-se, por sua relação com este trabalho, a auto-descrição, ou seja, existência de documentação no código-fonte.

O objetivo de se ter um programa auto-documentado é obter-se clareza e, consequentemente, facilitar futuras modificações no software. Porpõe-se, então, que as

seguintes informações, estejam presentes nos programas do DELPHI:

- título;
- função;
- autor;
- data inicial da codificação;
- data da última atualização;
- descrição do método;
- referências;
- descrição dos argumentos;
- entrada de dados;
- saída de dados;
- mecanismos (somente aqueles que não sejam padrões).

Obedecendo o seguinte formato: as informações devem estar incluídas entre linhas de asteriscos ('*'), e cada linha, contendo a documentação, deve iniciar (na coluna 1) e terminar (na coluna 72) com um asterisco.

Os ítens *descrição do método, referências* e *mecanismos* podem ser omitidos em casos aonde não aplicáveis. Existem mais dois níveis de informações:

- explicações quanto à lógica do programa;
  (identificadas por asteriscos da coluna 1 à 3)
- explicações detalhadas do código.
  (identificadas por um asterisco na coluna 1)

A figura III.1 apresenta um exemplo de como deve ser a documentação dos programas do DELPHI.

```
*************************************************************
*  Programa:                                                *
*  Titulo:                                                  *
*  Autor:                                                   *
*  Data inicial:                                            *
*                                                           *
*                                                           *
*************************************************************
***  Logica do Programa:
***
***
*  Descricao Detalhada do Codigo:
*
*

      PROGRAM TESTE
```

Figura 111.1: Comentários no código dos programas do DELPHI

Verificação das Convenções do Código

O programa FLOPPY [27] corresponde a uma ferramenta de software que verifica se um programa em Fortran segue as convenções de codificação pré-definidas e gera um novo arquivo com *DO-loops* e *IF-blocks* identados.

A lista de convenções contém 44 ítens, listados na tabela 111.1, onde aqueles precedidos por asterisco correspondem às convenções padrões.

O programa FLOPPY ainda oferece uma opção para gerar um arquivo para o programa FLOW, o qual produz árvores de chamadas a subrotinas, permitindo uma exploração interativa dessas estruturas, e tabelas dos *COMMON blocks*.

Manutenção e Atualização de Programas

O programa PATCHY [28] auxilia a manutenção e o transporte do código-fonte entre diversos computadores. Em um mesmo arquivo, são definidas linhas de código em diferentes versões e o programador pode selecionar ou modificá-las através de simples comandos. Ou seja, o código-fonte além de conter linhas de comandos também inclui instruções para modificar o programa.

Convenções de Codificação	
* I	evitar linhas de comentários ao final do módulo
* 2	finalizar todos os módulos com **END**
* 3	os **COMMON** blocks declarados devem ser usados no módulo
* 4	variáveis **COMPLEX** e **DOUBLE PRECISION** devem estar no final dos **COMMON** blocks
* 5	as definições em **COMMON** blocks não devem ser alteradas
* 6	nomes de variáveis devem ter no máximo **6** caracteres
7	variáveis em **COMMON** blocks devem ter **6** caracteres
8	variáveis fora de **COMMON** blocks devem ter menos de **6** caracteres
* 9	variáveis inteiras devem iniciar com letras entre **I** e **N**
* 10	variáveis não podem ser iguais às palavras-chave em **FORTRAN**
* li	evitar linhas de comentários após a declaração do módulo
* i2	nome do módulo não pode ser igual a funções internas do **FORTRAN**
* 13	a primeira declaração do módulo deve ser seu nome
* 14	o módulo deve começar com pelo menos **3** linhas de comentários
15	linhas de comentários devem começar com **C**
* 16	não devem haver comentários entre linhas de continuação
* 17	evitar tipos de variáveis que não **sejem** padrão **(e.g., INTEGER*2)**
* 18	evitar definições **múltiplas** de **COMMON por** linha
* 19	não dimensionar variáveis do **COMMON** fora desse
* 20	evitar espaços em branco nos nomes das variáveis
* 21	evitar espaços em branco nas entidades sintáticas
* 22	evitar o uso do comando **PRINT** (usar **WRITE)**
23	o comando **END** não deve ter um label
* 24	evitar a construção **WRITE(* ...**
25	evitar **o** uso do comando **WRITE** em funções
* 26	evitar o uso do comando **PAUSE**
* 27	labels não devem ser iniciados na primeira coluna
* 28	o comando **STOP** deve ser precedido por um **WRITE** descriptivo
* 29	evitar o uso de **ENTRY** em funções
* 30	evitar entrada e **saída** de dados em funções
31	evitar o uso alternativo do comando **RETURN**
* 32	o nome de um **COMMON** não pode ser igual a nomes de variáveis
* 33	evitar o uso de rotinas obsoletas da biblioteca de programas do **CERN**
34	evitar nomes de funções iguais às internas
* 35	funções locais devem ser declaradas como **EXTERNAL**
* 36	todos os nomes dos módulos devem ser diferentes
* 37	evitar expressões com variáveis de tipos diversos **(e.g., A=B/I)**
* 38	a dimensão de parâmetros do tipo **CHARACTER** deve ser *
* 39	a ordem das instruções deve estar correta
* 40	separar declaração de funções com comentários
* 41	não usar nomes definidos nas funções em outros lugares
42	usar **LLT, LFT para comparar** variáveis do **tipo CHARACTER**
43	variáveis fora de **COMMON** e não parâmetros dever ter menos de 6 caracteres
* 44	a dimensão de parâmetros deve ser *

Tabela 111.1: Lista de convenções do código verificados por FLOPPY

A figura 111.2 apresenta um exemplo de um código em Fortran com comandos PATCHY:

```
      SUBROUTINE CCTOC
C
+SEQ,CMDATA,CMLUN.
C
      CHARACTER*256 CLINE
      CHARACTER*80  KLINE, LINE(6), OLDDIR, CARD, LEAF
C
+SELF,IF=-IBM.
   40 READ(LUNSCR,10000,END=150) CLINE
+SELF,IF=IBM.
   40 CLINE  = ' '
      READ(LUNSCR,NUM=NBYT,END=150) CLINE
+SELF.
      NREC   = NREC+1
      LENGTH = LENOCC(CLINE)
        .
        .
```

Figura 111.2: Exemplo de um programa Fortran com comandos PATCHY

O comando *+SEQ* substitui CMDATA e CMLUN pelas associadas sequências de linhas de código definidas previamente (figura 111.3):

```
+KEEP,CMDATA.
      PARAMETER(KDMAX=100000)
      COMMON/CMDATA/IDATA(KDMAX),NDATA
C
+KEEP,CMLUN.
      PARAMETER   (MAXLUN=50)
      COMMON/CMLUN/NLUN,ILUN,LUNEDT,LUNSCR,LUNC1,LUNC2,
     +             LUNLOG,LUNBAT,LUNEXC,LUNTMP
      COMMON/CMLUN1/CHTOP(MAXLUN)
      CHARACTER*64  CHTOP
        .
        .
```

Figura 111.3: Exemplo de um arquivo contendo comandos PATCHY

Dessa forma, para modificar o COMMON CMLUN, por exemplo, somente é

necessário editar um arquivo aonde todas as sequências estão definidas.

O comando *+SELF* gerará um programa com a linha:

40 READ(LUNSCR,10000,END=150) CLINE, caso a implementação não seja em uma máquina IBM ou, com as linhas:

40 CLINE = ' ' e

READ(LUNSCR,NUM=NBYT,END=150) CLINE, caso contrário.

Desenvolvimento e Manutenção do código a nível de rotinas

Para apoio às atividades de desenvolvimento e manutenção do código, está disponível no CERN, a ferramenta CMZ [7] (Code Management System with Zebra [29][30][31][32]), que apoia a construção e manutenção de código em Fortran ou C. Para a fase de desenvolvimento, CMZ oferece:

- acesso direto a qualquer rotina através do editor local (o qual pode ser escolhido pelo usuário);
- verificação de variáveis indefinidas;
- identação automática de rotinas;
- renumeração de labels;
- re-ordenação de rotinas em ordem alfabética;
- um único comando para compilar uma rotina e atualizar a biblioteca-objeto;
- importação e exportação de arquivos do sistema operacional para o ambiente CMZ.

Para a fase de manutenção, CMZ oferece:

- mecanismo de controle de versões;
- comparação de rotinas com diferentes versões;
- operações diretas com a biblioteca-objeto.

CMZ interpreta instruções do PATCHY, oferece uma interface padrão (ver próxima seção) e é executado em diversas plataformas (IBM, VAX, DEC, Cray, Sun, etc.).

Padronização da Interface

KUIP (Kit for a User Interface Package) [33] é um programa gerenciador de interface para qualquer aplicação baseada em entrada interativa de comandos. KUIP é integrado ao programa como uma subrotina que faz uma verificação preliminar da sintaxe do comando e respectivos parâmetros.

Em todos os programas que utilizam KUIP são integrados ao help da aplicação mais dois níveis de comandos, ilustrados na figura 111.4.

```
  ...   help

From  /...

 1:   KUIP          Command Processor commands.
 2:   MACRO         Macro Processor commands.
 3:   "programa"    Help para os comandos do "programa".

Enter a number ('\'=one level back, 'Q'=command mode):
```

Figura 111.4: Help gerado por KUIP para a aplicação *"programa"*

O primeiro nível (1: KUIP) oferece comandos para chamar o editor local, listar a sintaxe de comandos, redefinir comandos (por exemplo, o comando `ls` pode ser definido como sendo `dir`) e gerar um manual, correspondente ao help, em SGML, TEX ou LATEX.

O segundo nível (2: **MACRO**) oferece comandos para a definição de macros, ou seja, palavras-chave associadas aos comandos da aplicação (exemplo: o macro *COMP* é definido como um comando para compilar rotinas, independente do ambiente onde é executado).

Através desses mecanismos, KUIP oferece uma interface padrão aos usuários com todos os programas que utilizam essa ferramenta.

III.2.2 Produção de Documentos

O ambiente de processamento de documentos no CERN [34][35][36], à semelhança de outros contextos, tem se desenvolvido muito nos últimos anos. Tempos atrás, as secretárias utilizavam diversos sistemas processadores de texto incompatíveis e dependentes de máquinas, por exemplo: Norsk Data, Wang, AES, Philips, Olivetti e Nixdorf. Ao mesmo tempo, grande parte da comunidade dos chamados usuários científicos utilizava formatadores de texto como Waterloo *Script* e TEX em computadores de grande porte [37] (IBM e VAX).

Atualmente, os usuários do CERN[1] utilizam uma série de ferramentas integradas ao ambiente computacional. A tabela 111.2 mostra que processadores de texto se encontram disponíveis no CERN.

111.3 Considerações

O estudo e o conhecimento do ambiente para o qual e no qual uma ferramenta será desenvolvida permite que:

- na construção do programa sejam utilizadas as facilidades que o ambiente oferece;
- o sistema resultante seja adequado ao ambiente, considerando os hábitos de seus usuários.

Um outro fator importante é conhecer que ferramentas computacionais são utilizadas no ambiente para se construir um programa compatível. A compatibilidade ou integração entre sistemas novos e antigos evita que o usuário tenha o esforço de modificar seu método de trabalho para poder utilizar um novo programa *e* estimula a utilização das novas técnicas implementadas pois o aprendizado é quase imediato se a interface do novo sistema for parecida com a dos sistemas utilizados.

'consiste em 500 funcionários administrativos e mais de 4000 físicos, engenheiros, técnicos e visitantes.

Sistema	Programa	Principal Utilização	Sistema de Impressão	Visualizador
PC	MS-Word	administrativa	parte do sistema	WYSIWYG
	LaTeX (PCTeX)	científica	comando "DVIPS"	comando VIEW
Macintosh	MS-Word	administrativa	parte do sistema	WYSIWYG
	LaTeX (TeXtures)	científica	parte do sistema	parte do sistema
VM/CMS	SGMLDCF BookMaster	elaboração de manuais e livros	comando "XPRINT"	não é aplicável
	LaTeX	científica	comando "DVITOPS"	não é aplicável
VAX VMS	VAX Document	administrativa, elaboração de manuais e livros	comando "XPRINT"	não é aplicável
	LaTeX	científica	comando "DVIPS"	comando "XDVI" (somente em VAXStations)
WORK-STATIONS	LaTeX	científica	comando "DVIPS"	"XDVI", "GS" (PostScript)
	Interleaf	elaboração de manuais e livros	parte do sistema	WYSIWYG
	FrameMaker	científica	parte do sistema	WYSIWYG
	The Puslisher	científica	parte do sistema	WYSIWYG
	nroff/troff	científica	lpr	não é aplicável

Tabela **111.2:** Visão esquemática dos processadores de texto disponíveis e utilizados no CERN

Capitulo IV

Geração dos Manuais do Software

Este quarto capítulo reúne uma série de idéias e técnicas para geração, com qualidade, dos manuais de utilização de um programa. São apresentados um conceito geral sobre o que se entende por qualidade de documentação, os atributos e os princípios básicos de uma boa documentação, uma organização no processo de produção de documentos (documentação estruturada) e uma proposta para o conteúdo dos manuais de utilização do software. Ao final, descreve-se um ambiente que integre as atividades de codificação e documentação.

IV.1 Qualidade da Documentação

Qualidade de um software corresponde ao conjunto de propriedades que caracterizam como o sistema pode satisfazer às necessidades de seus usuários. Um produto de software, com qualidade inclui, entre outras características, uma documentação útil, atualizada e consistente, garantindo sua eficiência desde que os usuários entendam o sistema com o qual trabalham.

A geração de informações escritas sobre um produto de software corresponde a um dos aspectos mais críticos durante a construção de um sistema computacional eficiente e também um dos mais negligenciados. Não há algoritmos para produzir uma boa documentação, mas existem algumas diretrizes. Qualquer que seja o tipo de aplicação, sua documentação deve incorporar os três princípios seguintes:

- estar atualizada;
- estar completa, do ponto de vista das informações que contém;
- ser fácil de manipular de forma que a tarefa de procurar informações seja rápida e simples.

Mesmo quando consideráveis esforços são dedicados à produção e manutenção de documentos, esses podem falhar e não serem úteis se o seu estilo e formato forem inapropriados a sua proposta ou público.

Para se avaliar a qualidade de um documento deve-se, primeiro, definir o que entendemos por *qualidade de uma documentação*. Qualidade é um conceito multi-dimensional e, portanto, é alcançada através de diversos atributos. A seguir define-se o conjunto de atributos identificados e que contribuem para a qualidade da documentação de um produto de software.

IV.1.1 Atributos de Qualidade de uma Documentação

Uma documentação de qualidade é aquela que traz eficiência na utilização do sistema pois através dela os usuários entendem o produto com o qual estão trabalhando. Uma boa documentação deve ser fácil de ser manipulada, independentemente se o leitor é novato ou com experiência e deve incorporar os seguintes fatores de qualidade baseados em [25], [38], [39] e [40]:

- usabilidade, ou seja, o documento deve servir ao seu propósito. Relacionados a esse fator, inclui-se o sub-fator:

  comunicabilidade: o texto deve ser escrito em uma linguagem de fácil entendimento ao usuário e ao seu nível de conhecimento.

- legibilidade, para que o documento seja facilmente entendido. Para tanto, o texto deve ser:

  - claro: ter apenas um significado;
  - correto: obedecer a regras gramaticais, sintáticas e de pontuação;
  - conciso: expressar uma informação utilizando um mínimo de palavras e sentenças;
  - consistente: não conter ambiguidades e conflitos;
  - modular: as informações relativas a um assunto devem estar agrupadas;

- uniforme quanto a terminologia;
- uniforme quanto ao nível de abstração.

– manipulabilidade, para que o documento seja facilmente acessado e manipulado. Para tanto, o texto deve ser:

- estruturado: as partes que compõe o documento devem formar uma organização hierárquica;
- disponível: se o documento é atualizado e pronto para ser manipulado;
- rastreável: deve ser fácil a ação de se localizar a informação necessária.

– manutenibilidade, o documento deve pertimir mudanças (relativas a correções e adaptações) e evoluções durante sua utilização. Portanto, deve ser:

- modificável: o documento pode ser modificado facilmente;
- expansivel: o documento pode ser expandido (evoluído) facilmente, considerando as informações presentes e sua organização.

– fidedignidade, é um atributo que a documentação deve ter para corresponder, com fidelidade, ao que foi especificado e projetado. Relacionados a esse fator, define-se um documento:

preciso: não deve haver qualquer tipo de incorreção no conteúdo do documento;

- completo: isto é, que descreve todas as funções do produto que são necessárias a diferentes tipos de usuário.

IV.2 Princípios Básicos para gerar documentos

A maioria dos desenvolvedores de software trata a documentação como um produto a parte do sistema computacional ao invés de integrá-la ao programa.

Retting [4] considera a documentação como um processo de engenharia envolvido no desenvolvimento do software. As informações que devem estar presentes em um documento devem ser projetadas de forma a satisfazer as necessidades do usuário, submetidas a testes e constantemente atualizadas.

Poston [41] responde a pergunta O *que é uma boa documentação?* através de três regras básicas:

> **Regra 1:** *criar documentos somente para uma audiência específica com uma determinada necessidade.*

Um documento auxilia a tarefa do usuário de interagir com o sistema computacional. Consequentemente, ele aprova a documentação se satisfizer suas necessidades. Seguindo a regra 1, os documentos devem ser sucintos e direcionados a uma tarefa específica.

> **Regra 2:** *organizar e estruturar uma documentação de modo a facilitar tanto sua criação quanto sua utiliizaação.*

Uma maneira de minimizar as dificuldades encontradas durante o processo de produção de documentos é fornecer previamente uma organização e uma estrutura para criação do documento.

A regra 2 possibilita a geração de documentos consistentes quanto ao estilo, formato e notações seguindo padrões pré-definidos. Um estilo padrão define regras de identação e a apresentação das informações no documento. Um formato padrão facilita localizar dados. Notações padrão reforçam a definição dos dados.

Os padrões devem ser entendidos tanto pelos usuários quanto pelos escritores. Dessa forma, todo o esforço se concentra no trabalho com o sistema e não com o entendimento do documento.

> **Regra 3:** *incluir apenas as informações manipuladas pelo software e necessárias ao usuário, registrando como são geradas.*

Essa regra restringe a quantidade de trabalho empregada durante a criação de um documento e facilita o entendimento do documento dado que apenas as informações essenciais serão incluídas.

Relacionados às regras anteriores, pode-se definir os princípios básicos para a geração de um documento de qualidade [3]:

– quanto ao conteúdo:

  · uniformidade de tratamento, ou seja, um tópico nunca deve ser mais detalhado que outros de igual importância;

    todo o material relevante deve ser incluído;

    redundâncias desnecessárias e excesso de detalhes podem dificultar o entendimento do usuário;

  · um volume excessivo de explicações pode confundir em vez de ajudar no entendimento. Apresentar somente exemplos essenciais, diretos e simples, dentre os quais alguns tutoriais, que incentivam a utilização e aprendizado do sistema;

  · vocabulário simples e sentenças diretas facilitam o entendimento.

– quanto à organização:

  · índices e cabeçalhos ajudam o leitor a se localizar;

  · deve-se utilizar uma estrutura **top-down** fornecendo uma visão geral antes de entrar em detalhes;

  · introduções e sumários auxiliam o leitor a procurar o que precisa;

    apresentar o conteúdo do documento permitindo que o próprio leitor decida que capítulos o interessam.

– quanto ao formato:

    em geral, figuras e tabelas podem explicar mais do que palavras e por isso devem ser utilizadas;

  · lembrar-se que o leitor, geralmente, folheia um documento à procura de informações e, para facilitá-lo nessa tarefa, utilizar diferentes estilos de letras (negrito, itálico, etc.) bem como espaçamentos para realçar dados importantes.

IV.3 Documentaçãoestruturada

Um documento pode ser descrito como uma coleção de objetos onde os de alto-nível são formados por objetos primitivos. Por exemplo, um livro é sub-dividido em capítulos, cada capítulo em seções, sub-seções, parágrafos, etc. Essa organização é conhecida como *documentação estruturada* [18].

A seguir (figura IV.1), são apresentadas as representações lógica e física de uma documentação estruturada:

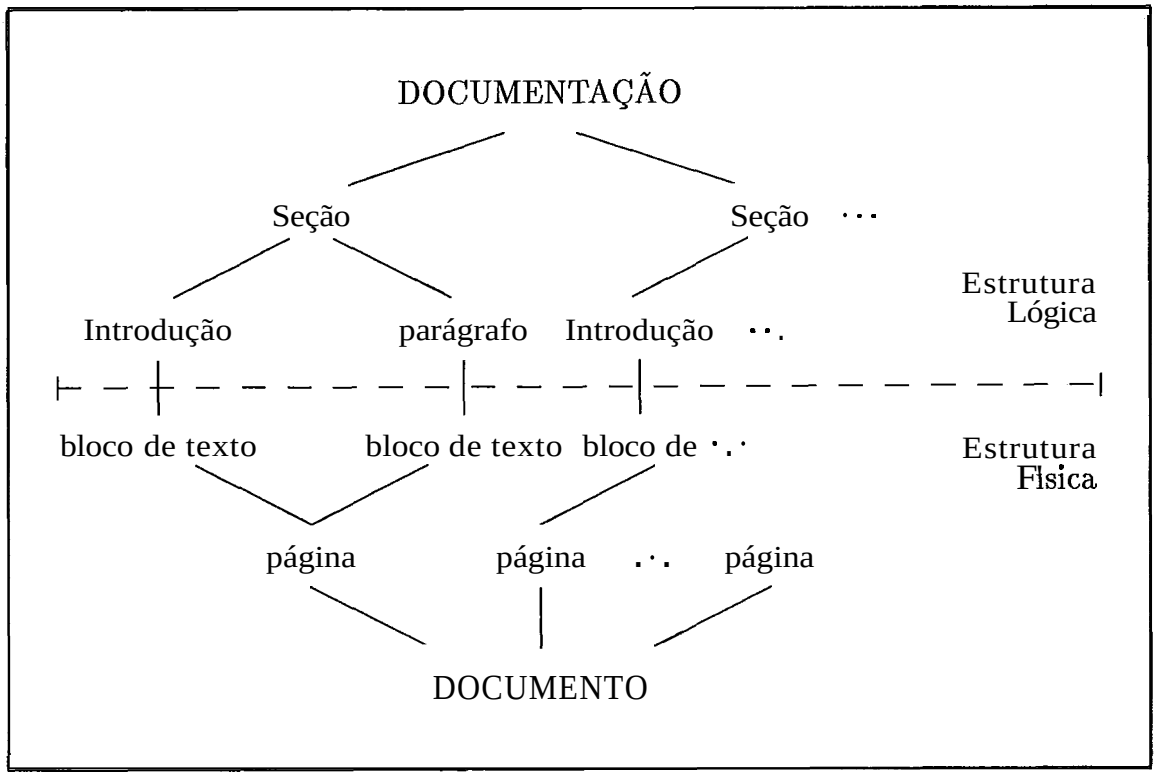

Figura IV.1: Estruturas lógica e física de uma documentação estruturada

A estrutura lógica combina elementos em sucessivos elementos de alto-nível representando o conteúdo do documento (parágrafos, seções e a documentação). A estrutura física combina elementos em elementos de alto-nível representando a visualização do documento (blocos, páginas e o documento)'

'algumas palavras como *parágrafo* ou *documento* expressam tanto um sentido lógico como físico.

IV.3.1 Formato X Conteúdo

No início da construção de um documento, o escritor está interessado na informação que irá manipular e não como essa informação será representada [42]. Torna-se necessário estabelecer abstrações apropriadas para manipular com os dados propriamente ditos e com o seu formato de modo independente.

A aparência dos elementos que compõem um documento pode ser descrita separadamente em um arquivo contendo as informações de seu estilo [43]. Dessa forma, separa-se a forma do conteúdo. Ao invés de modelar um documento como uma sequência de caracteres e suas diversas propriedades pode-se modelá-lo como uma estrutura hierárquica de unidades contendo sub-unidades. Uma estrutura comum consiste em capítulos, seções e parágrafos. Propriedades visuais, tais como identações, fontes e tamanhos de letras, são associadas implicitamente às unidades. Portanto, o título de um capítulo aparece com um fonte de tamanho grande e com espaçamentos não porque essa informação é envolvida por comandos explícitos mas porque é o *título de um capítulo.*

Esse método retira os dados sobre o estilo do texto e se aproxima da maneira como os autores naturalmente pensam sobre documentação, o que permite as tarefas da criação do texto (informações lógicas) e da definição do estilo (informações físicas) serem realizadas separadamente e, consequentemente, de modo mais fácil. Uma outra vantagem desse método é que, uma vez que as informações lógicas se encontram separadas das físicas, a produção dos documentos é indenpendente do sistema de formatação utilizado.

IV.4 Integrando o código e sua documentação

> *Ambientes de Engenharia de Software oferecem aos seus usuários ferramentas necessárias para o desenvolvimento sistemático de software e para a sua manutenção.*
>
> Houghton, R. C. e Wallace, D. R. [12]

Os Ambientes de Engenharia de Software podem ser caracterizados por diversos

fatores, tais como: seu relacionamento com o ciclo de vida, suas capacidades e limitações, usuários e nível de apoio.

Uma proposta para geração de documentos durante o processo de desenvolvimento do produto é apresentada pelo **Federal Information Processing Standards Publications [9]:**

- Fase de Especificação de Requisitos: a documentação deve conter:
  - definição geral dos requisitos;
  - estudo da viabilidade;
  - plano de desenvolvimento;
  - análise de custos e benefícios.
- Fase de Definição: a documentação deve conter:
  - definição dos requisitos funcionais;
  - definição dos requisitos de dados;
  - plano de verificação e testes.
- Fase de Projeto: a documentação deve conter:
  - especificação dos sistemas e sub-sistemas;
  - plano de verificação e testes.
- Fase de Projeto Detalhado: a documentação deve conter:
  - especificação do programa;

    especificação do banco de dados;
  - plano de testes.
- Fase da Programação: a documentação deve conter:
  - manual do usuário;

- manual de operação;
- manual de manutenção do programa;
- relatório da análise de testes.

– Fase de Operação e Manutenção: a documentação deve conter:

  modelo do requerimento formal para solicitação de mudanças;
- procedimentos para atualização dos documentos.

Observando-se o estado da arte em Engenharia de Software verifica-se, através das ferramentas já desenvolvidas e disponíveis para o uso, que ainda é difícil, se não impossível, encontrar-se um ambiente de desenvolvimento que forneça apoio completo a todas as atividades do processo de desenvolvimento.

Pode-se definir um ambiente de programação como um ambiente de engenharia de software limitado a uma fase: a da construção do código e, segundo o Federal Information Processing Standards, elaboração dos manuais.

Uma visão antiga em desenvolvimento de software é que primeiro constrói-se o produto e depois, escreve-se a documentação [44]. Essencialmente, o problema era que a documentação era vista como um trabalho a parte e de menor importância que o software. A visão atual é desenvolver o sistema e seus documentos ao mesmo tempo, ou seja, considerar a documentação como parte integrante do produto sob desenvolvimento. Procedendo-se assim, os manuais devem estar concluídos no final da fase de construção do programa.

. Um ambiente que integra o código e sua documentação deve considerar que muitas das informações presentes no primeiro, tais como, comentários ou até mesmo partes do código podem ser necessárias no documento final. Nesse caso, programas que extraem dados do código-fonte, denominados filtros, tornam-se necessários pois as ferramentas automáticas garantem que todas as informações necessárias sejam extraídas e que essas correspondam ao arquivo-fonte.

IV.5 Conteúdo dos Manuais

O Federal Information Processing Standards, propõe os seguintes roteiros para Manual do Usuário, Manual do Operador e Manual de Manutenção do Programa:

Manual do Usuário:

1. Informações gerais

   (a) Apresentação: introduz a aplicação e suas funções gerais.

   (b) Ambiente: identifica o ambiente onde o sistema é instalado.

   (c) Referências, tais como:

      i. autorização para se obter o software;

      ii. documentos anteriormente publicados sobre o projeto;

      iii. documentação relativa a outros projetos relacionados a esse.

2. Aplicação

   (a) Descrição geral: quando e como utilizar o software. Deve incluir:

      i. proposta do software;

      ii. funções realizadas;

      iii. capacidades operacionais.

   (b) Operação: mostra o relacionamento entre as funções do sistema e suas entrada e saída de dados. Descreve aspectos de segurança e privacidade do sistema.

   (c) Equipamento: descreve o equipamento no qual o sistema é executado.

   (d) Estrutura: apresenta a estrutura do software e descreve a função de cada componente.

   (e) Execução: descreve as capacidades de execução do software, identificando:

i. dados quantitativos sobre entradas, saídas, tempo de resposta, tempo de processamento e erros;

ii. dados qualitativos sobre flexibilidade e confiabilidade.

(f) Banco de Dados: descreve todos os arquivos de dados manipulados pelo sistema.

(g) Processos e entrada e saída de dados: descreve as entradas, o fluxo de dados e resultados bem como os relacionamentos entre entrada e saída de dados.

3. Procedimentos e requisitos

(a) Inicialização: descreve passo a passo os procedimentos para se iniciar o processamento.

(b) Entrada: define os requisitos para preparar a entrada de dados. Inclui:

i. formatos;

ii. exemplos.

(c) Resultados: define os requisitos para cada saída de dados. Inclui:

i. formatos;

ii. exemplos.

(d) Erros e recuperação: lista os possíveis erros e ações que devem ser tomadas para corrigí-los.

Manual do Operador:

1. Informações gerais

(a) Apresentação: introduz as funções gerais do software.

(b) Ambiente: identifica o ambiente operacional onde o sistema deve ser instalado.

(c) Referências, tais como:

i. autorização para se obter o software;

ii. documentos anteriormente publicados sobre o projeto;

iii. documentação relativa a outros projetos relacionados a esse.

2. Visão geral

(a) Organização do software: fornece um esquema geral com as entradas, saídas, arquivos de dados e sequências de suas operações.

(b) Criação do programa: identifica cada programa por título e número de referência.

(c) Criação dos arquivos: identifica cada arquivo permanente que é referenciado, criado ou atualizado pelo sistema por título, número de referência e informa seu tamanho para armazenamento.

3. Descrição das execuções

(a) Identificação: organiza as informações de cada execução de forma útil ao centro de operações, identificando:

i. controle de entrada de dados;

ii. dados operacionais (tais como: comandos e mensagens para o operador, tempo estimado de execução, pessoal de contacto caso houver problemas, etc.)

iii. arquivos criados ou atualizados durante a execução;

iv. procedimentos para recuperar ou reiniciar a execução.

4. Procedimentos emergenciais
(exemplo: executar o sistema backup)

5. Operações remotas
(como executar o programa através de um terminal remoto)

Manual de Manutenção do Programa:

1. Informações gerais

   (a) Apresentação: introduz a aplicação do software a ser mantido.

   (b) Ambiente: identifica o ambiente onde o sistema foi implementado.

   (c) Referências, tais como:

      i. autorização para se obter o software;

      ii. documentos anteriormente publicados sobre o projeto;

      iii. documentação relativa a outros projetos relacionados a esse.

2. Descrição do Programa
   (ou programas que compõem o sistema)

   (a) Descrição/Identificação do Programa:

      i. problema e método utilizado para solucioná-lo;

      ii. entrada de dados (descrição completa);

      iii. processamento: lógica, variáveis, fórmulas, tratamento de erros, etc;

      iv. resultados (descrição completa);

      v. interfaces com outros software, descrevendo: formatos, mensagens, parâmetros, etc;

      vi. descrição da execução, informando carregamento na memória, manipulação de erros e outros dados operacionais.

3. Ambiente Operacional

   (a) Hardware: identifica o equipamento computacional necessário, descrevendo:

      i. processador;

      ii. dispositivos de entrada e saída;

      iii. dispositivos de transmissão de dados.

(b) Software de suporte: identifica os sistemas necessários para cada programa do sistema, tais como:

i. sistemas operacionais;

ii. compiladores;

iii. outros como, por exemplo, sistemas gerenciadores de dados, geradores de relatórios, etc.

(c) Banco de Dados (descrição completa).

4. Procedimentos para a Manutenção

(a) convenções utilizadas.

(b) procedimentos para verificação e controle das funções.

(c) procedimentos para correção de erros.

(d) atualizações periódicas do banco de dados, arquivos, etc.

(e) diagrama de fluxo de dados entre os programas.

A preparação de um documento deve ser vista como uma tarefa contínua, iniciada com esboços preliminares, seguida de mudanças e revisões até se chegar numa documentação adequada para sua utilização.

Cada tipo de manual é escrito para uma audiência particular, que pode ser tanto um único usuário quanto um grupo desses. Os usuários da documentação a utilizam como suporte para a realização de funções específicas, tais como: utilização, instalação ou manutenção de um software. Portanto, o conteúdo de cada manual deve considerar a terminologia e o nível de detalhes apropriado aos seus usuários.

Esses três tipos de roteiros para os manuais apresentam redundâncias aparentes. Por exemplo, a parte introdutória, que fornece ao leitor uma apresentação do software, está presente em cada documento para evitar referências a outros documentos. A descrição das entradas de dados, resultados e equipamentos também se encontram nos diferentes manuais. Evidentemente, essas informações aparentemente redundantes devem ser diferentes em contexto e, talvez, em relação à terminologia e nível

de detalhamento desde que cada documento se refere a diferentes audiências e atividades.

A utilização dos roteiros oferece certa flexibilidade, alcançada através da organização de seu conteúdo e, durante a construção dos documentos deve-se considerar os seguintes fatores:

- tamanho e complexidade do software, que influenciará o nível de detalhamento;
- combinação dos documentos, dado que em algumas aplicações, faz-se necessário agrupar todas as informações dentro de uma mesma capa, deve-se, nesse caso, identificar as diferentes partes dos documentos (Parte **I** - Usuários, Parte *II* - Operações, etc.);
- a sequência do conteúdo deve seguir, a princípio, a ordem apresentada embora não seja essencial;
- os *títulos* devem ser os mesmos apresentados anteriormente mas, podem ser modificados para refletir uma terminologia específica do sistema apresentado;
- as seções apresentadas nos roteiros podem ser expandidas ou divididas para melhor expressar as informações.

IV.6 Considerações

Para se obter um documento útil e de boa qualidade não basta apenas desejá-lo. Deve-se fazer um esforço, nessa direção, desde o início de sua elaboração [45]. Definir, previamente, a organização do documento, seus objetivos, conteúdo e os atributos de qualidade mais importantes que o manual deve atender facilita a geração de documentos que atendam a suas necessidades.

Capítulo V
SIM: Um Gerenciador de Informações do Software

Este capítulo apresenta a ferramenta SIM *que apoia a produção de documentos de produtos de software.* SIM *foi desenvolvida no CERN (Centro Europeu de Física de Partículas) em Genebra, Suíça. Sua implementação considerou as características do desenvolvimento de software nesse centro de pesquisas e foi baseada no estudo, apresentado nos capz'tulos anteriores, sobre documentação de software. São apresentadas uma visão geral da ferramenta, sua arquitetura, o processo de produção de documentos e detalhes sobre sua implementação.*

V.l Objetivos e visão geral da ferramenta SIM

A proposta apresentada nesta tese para resolusão do problema de se produzir uma documentação de software com qualidade prevê um único ambiente, provido de ferramentas computacionais adequadas, apoiando a construção, gerência e manutenção do código-fonte e de seus respectivos documentos. Um ambiente com essas características tem as seguintes vantagens:

- favorece a geração da documentação desde as fases iniciais do desenvolvimento;
- permite que a documentação de cada módulo do software seja feita concomitantemente com a sua construção;
- por melhorar a qualidade da documentação, atua como suporte a futuras manutenções.

A ferramenta SIM, desenvolvida nesta tese, constitui parte desse ambiente integrado para manipular código-fonte e sua documentação. Seus objetivos são: evitar

redundância de informações e garantir a consistência entre o código e documentos.

SIM possui um alto grau de flexibilidade, extraindo informações diretamente do código, em Fortran, oferecendo roteiros para documentação, que podem ser adaptados a diferentes aplicações e fornecendo ao usuário uma abstração sobre o formato do documento final, dado que seu conteúdo e descrição física são especificados separadamente.

No ambiente SIM, podem ser classificadas três tipos de informações:

- as referentes à implementação de uma rotina;
- linhas do programa identificadas por comentários especiais, denominados tags;
- dados gerais, totalmente manipulados pelo usuário, descritos através de um processador de textos a sua escolha.

As informações referentes à implementação de uma rotina são automaticamente extraídas do código. Correspondem ao nome da rotina, seus parâmetros e respectivos tipos, variáveis manipuladas e respectivos tipos, rotinas chamadas, arquivos acessados, e common **blocks.** Esses dados são extraídos do código através de uma análise sintática parcial da linguagem, identificando suas palavras-chave, tais como: SUBROUTINE, FUNCTION, REAL, INTEGER, CALL, OPEN, COMMON, etc.

A figura V.1 apresenta parte de uma rotina em Fortran e a figura V.2 apresenta a sua documentação gerada por SIM.

O segundo tipo de informação manipulada por SIM corresponde às linhas do programa, propriamente ditas. SIM permite que o usuário defina comentários especiais (SIM tags), os quais são incluídos no programa, identificando partes do código que devem estar presentes na documentação. Essa facilidade evita que o usuário tenha que escrever novamente na documentação informações que já se encontram no código.

As figuras V.3 e V.4 ilustram um exemplo da utilização das SIM tags. O usuário define que, por exemplo, a tag *ERR identifica as mensagens de erros de seu programa e a tag *COM, os comentários que devem ser incluídos na documentação.

```
      SUBROUTINE PHOTOS(IPPAR)
C
      DOUBLE PRECISION BET(3) ,PB
      REAL CHAPOI,PHEP,PCHARG(5)
      INTEGER NCHARG,NMXHEP,IP,IPPAR
      COMMON/HEPEVT/PHEP(5,NMXHEP),CHAPOI(PDMMAX)
C--
C--   Order charged particles according to decreasing mass
      IF (NCHARG.GT.1) CALL PHOOMA(1,NCHARG,CHAPOI)
C--
C--   Boost charged daughter to rest frame of parent...
      DO 10 J=1,IPPAR
         BET(J)=-PHEP(J,IP)/PHEP(5,IP)
         PB=BET(1)*PHEP(1,CHAPOI(NCHARG) )+BET(2)
   10 CONTINUE
      DO 20 J=1,IPPAR
         PCHARG(J)=PHEP(J,CHAPOI(NCHARG))+BET(J)
   20 CONTINUE
```

Figura V.l: Exemplo de um trecho de programa em Fortran

```
\Routine name SUBROUTINE PHOTOS

\Input specification
 \Parameter name IPPAR
 \Type INTEGER

\Variables definition
Type DOUBLE PRECISION, variables: BET(3), PB
Type REAL, variables: CHAPOI, PHEP, PCHARG(5)
Type INTEGER, variables: NCHARG, NMXHEP, IP

\Calling routines
CALL PHOOMA, variables: 1, NCHARG, CHAPOI

\Common blocks
Common HEPEVT, variables: PHEP(5, NMXHEP), CHAPOI(PDMMAX)
       .
       .
```

Figura V.2: Exemplo da documentação de um trecho de programa em Fortran

```
      IDABS=ABS(IDHEP(I))
      IF ((IDABS.GT.9).AND.(IDABS.NE.21)) THEN
         DATA=NCHARG
         WRITE(PHLUN,900) TEXT,INT(SDATA)
C*ERR
 900     FORMAT(1H,A,': Too many charged Particles, NCHARG =',I6)
C*
      ENDIF
C*COM
C--   Order charged particles according to decreasing mass, this
C--   is to increase efficiency(smallest  mass is treated first).
C*
      IF(NCHARG. GT.I) CALL PHOOMA(1,NCHARG,CHAPOI)
C*COM
C--   Check whether parent is in its rest frame...
C*
 30   IF(ABS (PHEP(4,IP))/PHEP(5, IP).GT.I.E-8) THEN
```

Figura **V.3:** Exemplo de um trecho de programa em Fortran com SIM tags

```
                .

\funcionalidade da rotina
\*COM
C--   Order charged particles according to decreasing mass, this
C--   is to increase efficiency (smallest mass is treated first).

C-    Check whether parent is in its rest frame...

                .
                .
                .

\mensagens de erro
\*ERR
 900     FORMAT(1H,A,': Too many charged Particles, NCHARG =',I6)
```

Figura **V.4:** Exemplo da documentação resultante com SIM tags

SIM inclui as linhas do programa, identificadas pelas tags, em suas respectivas posições na documentação. A tag `*COM` define os comentários que descrevem a funcionalidade da rotina e a tag `*ERR`, as mensagens de erro do programa.

A consistência entre o programa e sua documentação é garantida pois esses dois tipos de informações são extraídos diretamente do código para o documento.

O terceiro tipo oferece liberdade ao usuário quando ao descrever, por exemplo, fórmulas matemáticas ou figuras incluídas no programa. Um exemplo para descrever uma fórmula matemática, usando o formatador de textos TEX, é mostrado a seguir:

```
$ \int\!\!\!\int_D f(x,y)\,dx\,dy $
```

e que se apresenta, no texto:

$\iint_D f(x,y)\,dx\,dy$

V.2 Arquitetura de um ambiente integrando *código* e sua documentação

Um ambiente que integre as atividades de construção de programas e produção de seus documentos deve reunir ferramentas que apoiem as seguintes atividades:

- construção do código-fonte;
- geração do código-executável (através da compilação e link-edição de rotinas);
- elaboração da documentação, no que diz respeito às informações que o documento deve conter, que pode ser auxiliada por roteiros;
- produção do documento, ou seja, definição de seu formato.

O usuário escreve o programa através de um editor de textos. As rotinas construídas podem ser selecionadas, compiladas e link-editadas por simples comandos.

O ambiente extrai informações diretamente de uma ou mais rotinas criando arquivos contendo sua documentação. Esses arquivos podem ser modificados, inserindo informações adicionais.

Para facilitar a manipulação da documentação, o formato físico é gerenciado separadamente. Dessa forma, cada informação está associada à descrição de seu **layout**, que se encontra em um outro arquivo. No momento da produção dos documentos, as informações são integradas à descrição de seu formato.

SIM foi integrado ao CMZ (apresentado no capítulo 111), uma ferramenta já disponível no CERN, para gerenciamento de código Fortran ou C. A figura V.5 mostra a integração dessas duas ferramentas.

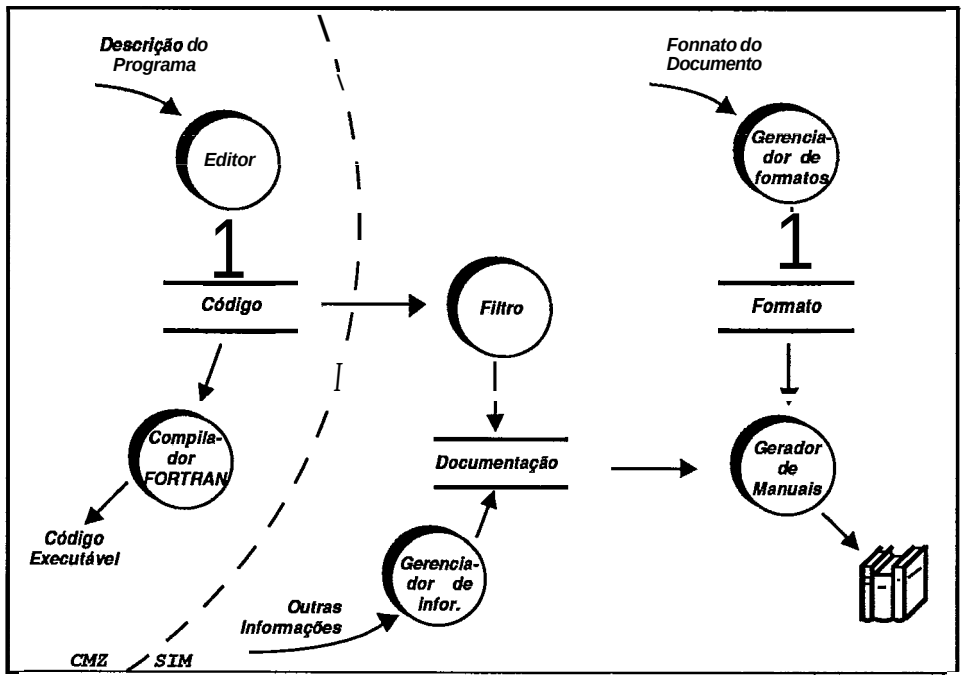

Figura V.5: Visão geral da arquitetura da integração das ferramentas SIM e CMZ

As atividades de ***edição*** e de ***compilação,*** dentre outras não apresentadas neste esquema, são apoiadas por CMZ. As ferramentas CMZ e SIM estão integradas de forma que CMZ oferece mecanismos para o desenvolvimento e manutenção do código

e SIM, para a geração e manutenção dos documentos do software.

A ferramenta SIM é composta por:

- um Filtro, que extrai os dados para a documentação diretamente do código-fonte;
- um Gerenciador de Informações, que permite a inclusão de informações adicionais às já extraídas pelo filtro;
- um Gerenciador de Formatos, que permite a definição do **layout** de cada informação contida na documentação;
- um Gerador de Manuais, que associa as informações lógicas às suas descrições físicas, definindo o formato do documento.

V.2.1 Integração das ferramentas CMZ e SIM

A ferramenta CMZ considera os hábitos de trabalho dos cientistas, em particular os do CERN, e possibilita:

- manipular a geração de código produzido de forma incremental;
- desenvolver programas de forma modular e uniforme quanto ao estilo;
- encaminhar o usuário a adquirir gradualmente melhores hábitos durante o desenvolvimento de software.

Na tabela V.l são mostrados os principais comandos do CMZ com uma breve descrição de sua função.

CMZ reúne os seguintes mecanismos para apoio às fases de desenvolvimento e manutenção:

- gerência de versões: data da criação do arquivo, data das atualizações, número da versão associada a cada arquivo, nome do autor, etc;
- compilação e link-edição:

Comando	Descrição
DECKLIST	Permite a seleção de um ou mais decks. Muitos comandos do CMZ aceitam DECKLIST como parâmetro
AUTHOR	Define o nome do autor (informação que é inserida em cada deck)
LMODS	Lista todos os decks modificados pelo autor AUTHOR durante o período especificado
MAKE	Cria um novo arquivo CMZ
FILE	Conecta um arquivo CMZ já existente
RELEASE	Fecha e desconecta um arquivo CMZ
CLIST	cria um arquivo no sistema operacional com o conteúdo de um arquivo CMZ ou um diretório CMZ
YTOC	Converte um arquivo PATCHY em arquivo CMZ
CTOY	Converte um arquivo CMZ em arquivo PATCHY
EDIT	Edita um ou mais decks através do editor local
READ	Lista um ou mais decks através do brouser local
FTOC	Importa para o CMZ um arquivo Fortran no sistema operacional
CTOF	Transcreve um ou mais decks para o arquivo de compilação
YTOF	Converte um arquivo PATCHY diretamente para Fortran
TTOC	Importa para o CMZ um arquivo texto no sistema operacional
CTOT	Exporta um ou mais decks contento texto para um arquivo no sistema operacional
CFORTRAN	Transcreve um ou mais decks para o arquivo de compilação e compila
CFLIB	Transcreve um ou mais decks para o arquivo de compilação, compila e arquiva
LIBRARY	Inclui ou substitui os módulos em código-objeto na biblioteca objeto corrente
MKALL	Recompila os decks que estão desatualizados em relação à biblioteca default
YCORR	Gera um deck, no diretório $CORR, contendo as diferenças entre dois decks, diretórios ou arquivos
DIFFERENCES	Informa as diferenças entre dois decks através do utilitário local (Apollo: cmf, Unix: diff, VAX: DIFF)
VERSION	Marca os decks de maior cíclo como pertencentes a versão x.yy/zz (por exemplo: 1.01/02 ou 13.03/11)
ALPHA_ORDER	Ordena todos os decks em ordem alfabética
INDENT	Faz automaticamente a identação de um ou mais decks que contém código em Fortran
LABEL	Organiza automaticamente, em um ou mais decks, os labels
UNDEFINED	Encontra, em uma lista de decks, as variáveis não definidas
TREE	Gera a sequência de chamadas a rotinas

Tabela V.l: Principais comandos do CMZ

- o usuário pode selecionar os *decks*[1], como por exemplo, os pertencentes à versão de número N ou ao autor X ou ao diretório D;
- definir o nome da object library, do código-executável a ser gerado e do compilador;

  Com apenas um comando, o código é compilado e link-editado, de acordo com esses parâmetros pré-definidos.

– correção automática do código quanto a:

- verificação de variáveis não definidas;

  identação automática;

  renumeração de labels.

– edição dos arquivos, podendo especificar o editor de textos.

Como já observado, SIM está integrada à ferramenta CMZ, de forma que o ambiente CMZ+SIM apoia o desenvolvimento e manutenção do código (através de CMZ) *e* a geração e manutenção de sua documentação (através de SIM).

A figura V.6 apresenta um exemplo da documentação de uma rotina Fortran no ambiente integrado CMZ+SIM.

As primeiras linhas dos arquivos são gerenciadas pelo CMZ. Na janela à esquerda da figura, encontram-se as seguintes informações:

– `+PATCH,//PHOTOS/CODE` informa o nome do arquivo CMZ (arquivo `//PHOTOS`) e do diretório (diretório `/CODE`) aos quais o deck, que está sendo editado, está associado;

– `+DECK,PHOTOS.` informa o nome do deck (deck PHOTOS) que está sendo editado;

– `*--` Author : B. van **Eijk** and Z. Was **,**

`*CMZ : 1.00/00` 11/11/91 18.40.34 by Bob van **Eijk** e

[1] no ambiente CMZ, *deck* corresponde a um arquivo, contendo código (uma rotina) ou texto.

*CMZ : 20/12/91 10.20.55 by Z. Was fazem parte do mecanismo histórico do CMZ.

A primeira linha somente é apresentada quando um novo *deck* é criado, identificando, assim o autor (definido pelo comando AUTHOR - ver tabela V.l) original da rotina. A cada modificação, CMZ apresenta automaticamente uma linha contendo a data, hora e autor de cada modificação; o usuário pode acrescentar outras linhas de comentários, (* determination of kinematical limit e * charged particles included, como no exemplo apresentado) descrevendo a modificação realizada.

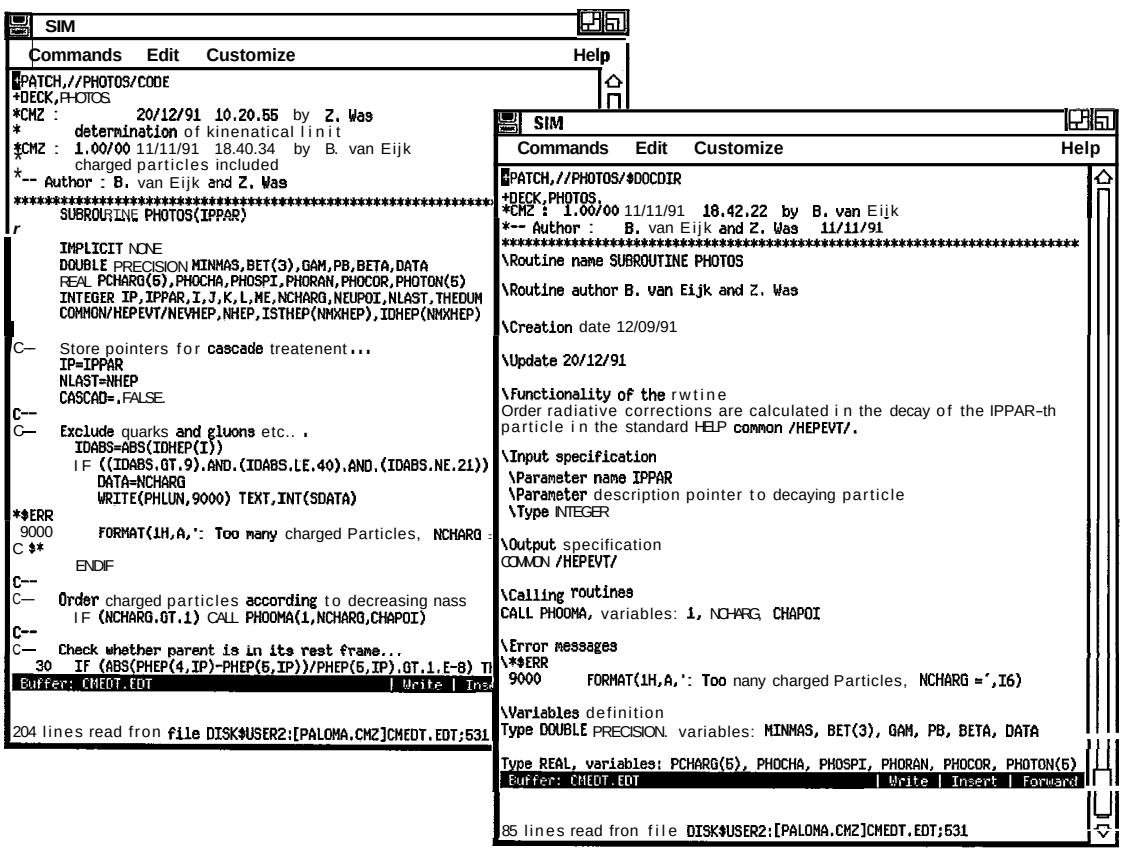

Figura V.6: Documentação de uma rotina

Após a elaboração da rotina, o usuário utiliza os comandos do CMZ para compilação e gerenciamento do código e os comandos do SIM para geração e gerencia-

mento da documentação.

As informações *Routine name*, *Input Specification* e ítens *Parameter name* e *Type, Calling routines, Variables definition* e *Error messages* foram extraídas diretamente do código. Essa última foi identificada no código através da tag *$ERR e as anteriores, pelo reconhecimento das palavras-chave da linguagem Fortran: *SUBROUTINE(IIPAR)*, *DOUBLE PRECISION*, *INTEGER* e *CALL PHOOMA(...)*. *Routine author, Creation date* e *Update* foram obtidas através do mecanismo histórico do *CMZ* e as demais informações *(Functionality of the routine, Parameter description* e *Output specification*) foram incluídas pelo próprio usuário.

A tabela V.2 apresenta os comandos do *SIM*, com a descrição de suas funções.

Comando	Descrição
DOCUMENT	Extrai as informações (1º e 2º tipos) de um ou mais *decks*, contendo código, e armazena a documentação correspondente em novos *decks*, com o mesmo nome, no diretório de documentação
SETDOCDIR	Permite a modificação do nome default ($DOCDIR) do diretório de documentação
DTOT	Gera um arquivo em formato LaTeX, contendo a documentação relativas aos *decks*, especificados pelo parâmetro DECKLIST no formato descrito no *deck* $STY_GUI

Tabela V.2: Comandos do *SIM*

Interface com o Usuário no ambiente CMZ+SIM

SIM utiliza o programa *KUIP* para sua interface com o usuário e sistema de **help on-line**. A figura V.7 ilustra o 1º nível de help. Os ítens 1 e 2 do help correspondem aos comandos oferecidos por *KUIP* (apresentados anteriormente no capítulo III); o ítem **3** corresponde aos comandos do *CMZ* e o ítem 4, aos do *SIM*.

As figuras V.8, V.9 e V.10 ilustram os *menus* dos comandos *DOCUMENT*, *SET_DOCDIR* e *DTOT* da ferramenta *SIM*.

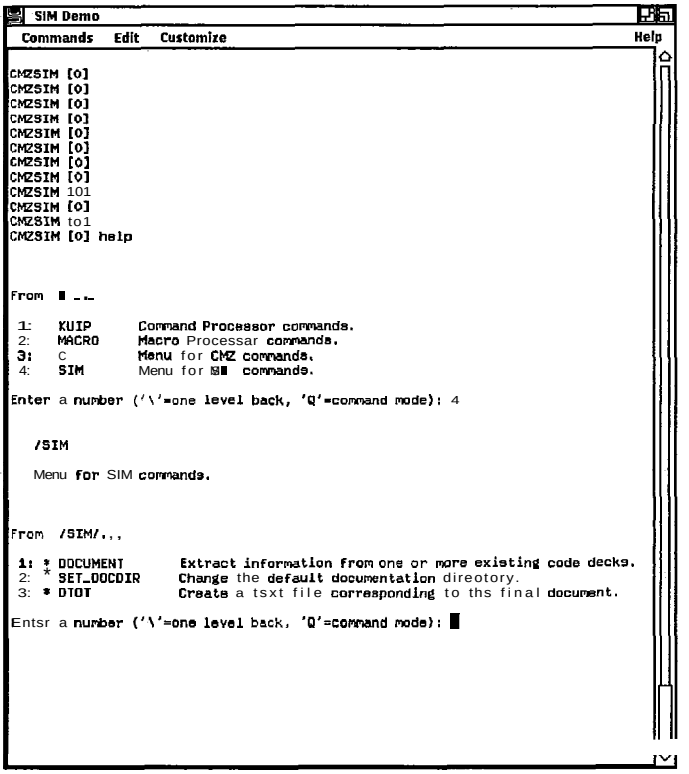

Figura V.7: Sistema de Help de CMZ+SIM - 1º nível

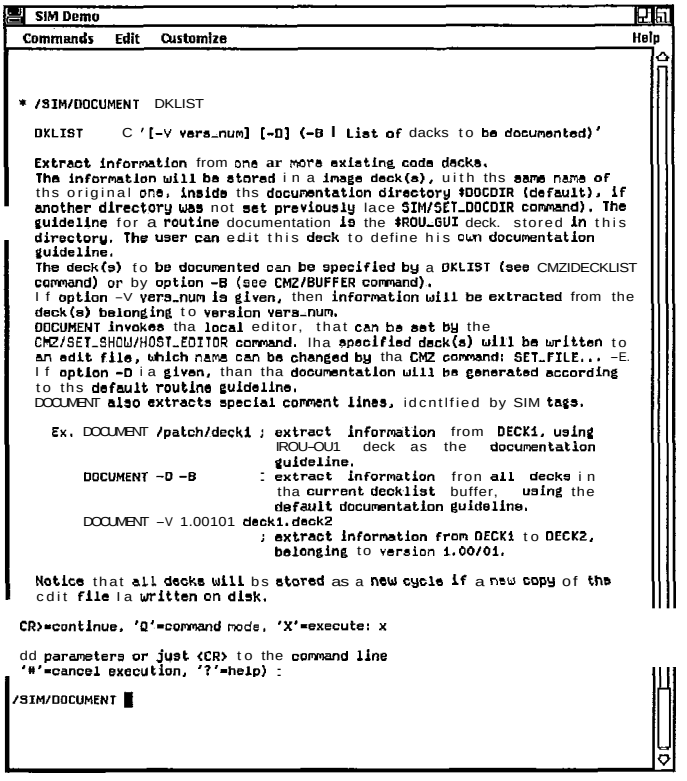

Figura V.8: Sistema de Help do SIM - comando DOCUMENT

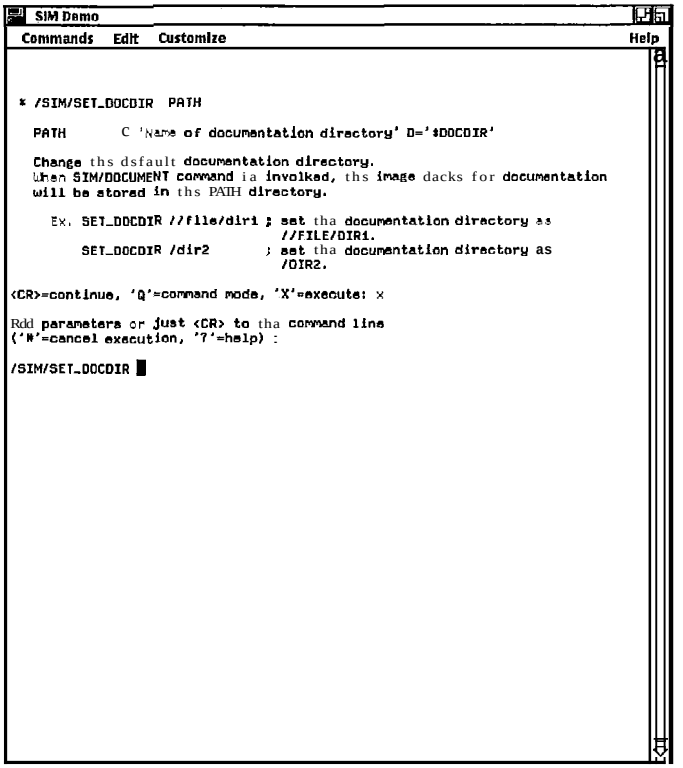

Figura V.9: Sistema de Help do SIM - comando *SET_DOCDIR*

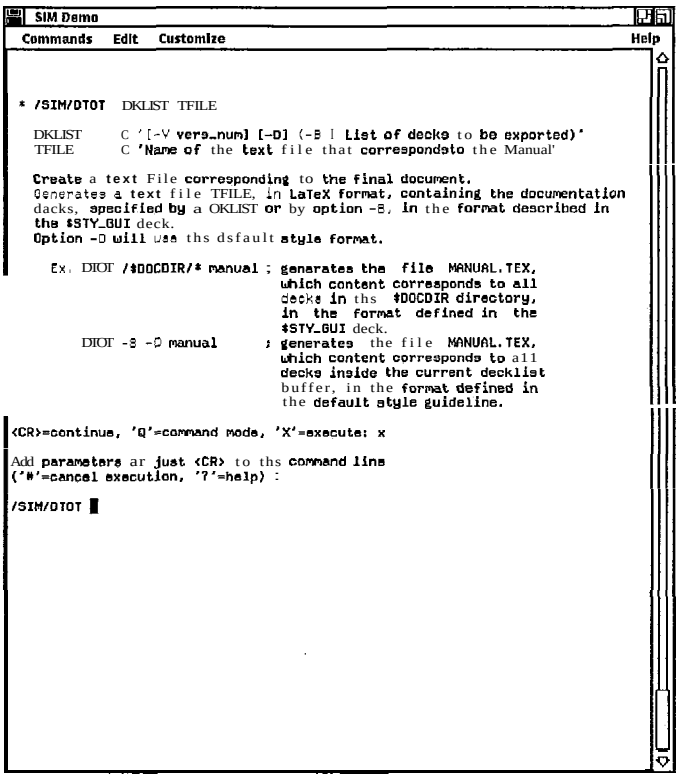

Figura V.lO: Sistema de Help do SIM - comando *DTOT*

KUIP é integrado ao programa como uma subrotina que faz uma verificação preliminar da sintaxe do comando e respectivos parâmetros. Na figura V.11, os erros provenientes das instruções de número 2 e 8 foram detectados pelo KUIP.

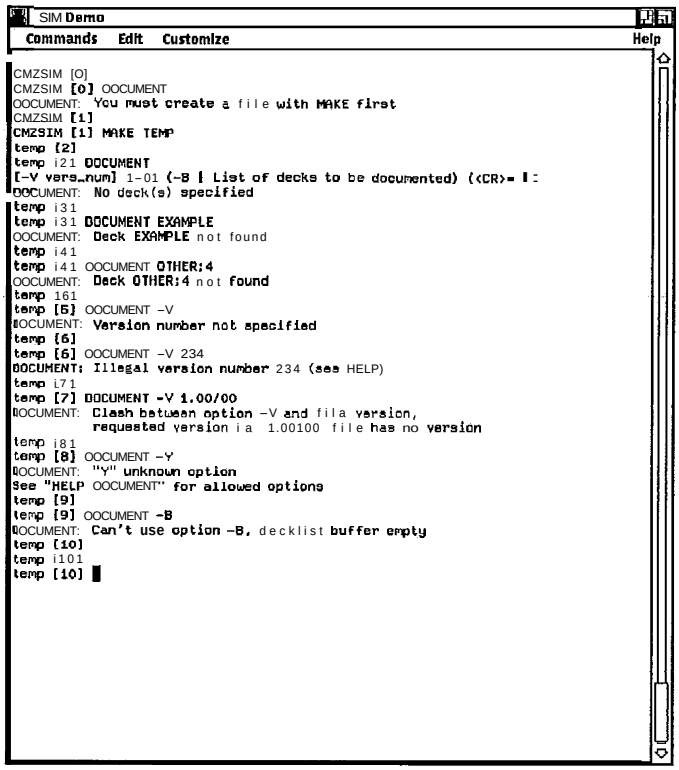

Figura V.11: Verificação da entrada de dados do comando DOCUMENT

V.3 Produção de documentos com SIM

Dentro do ambiente CMZ+SIM, a documentação é fisicamente separada do código, com o intuito de atingir um alto grau de modularidade e eficiência. A cada rotina (deck) documentada é criado um novo arquivo, contendo a documentação, com mesmo nome mas armazenada em um diretório especial. O diretório de documentação é denominado, por default, `$DOCDIR` e contém os seguintes roteiros de apoio à documentação:

- `$ROU_GUI`: contém as informações necessárias à documentação de uma rotina;
- `$MAN_GUI`: contém as informações necessárias à elaboração do manual (refere-se à parte introdutória do documento);

– `$STY_GUI`: define o formato do documento final.

V.3.1 Roteiros para auxílio à documentação

Para definir o conteúdo dos roteiros para auxílio à documentação foi necessário pesquisar o ambiente CERN, no que se refere às ferramentas para produção de documentos encontradas nesse laboratório [46][47][48][49] e aos manuais dos sistemas desenvolvidos no próprio CERN. Também foi realizada uma pesquisa bibliográfica, já apresentada nos capítulos anteriores, no sentido de identificar os diversos tipos de informações que um manual deve conter e os atributos de qualidade de uma boa documentação.

Com o apoio dos roteiros, a tarefa de documentação torna-se simples e padronizada. Simples, pois o usuário não necessita preocupar-se com que tipo de informações um documento deve conter. Padronizada, pois baseando-se nos roteiros pré-definidos documenta-se todas as rotinas de um programa e produz-se manuais para diferentes aplicações, de uma mesma forma.

Com o objetivo de oferecer uma maior flexibilidade, os roteiros encontram-se em arquivos, que podem ser modificados de acordo com as diferentes aplicações. O usuário pode incluir as tags nos roteiros e SIM substituirá as informações encontradas no código, obedecendo a ordem dos dados.

A seguir, são apresentados os roteiros para documentação oferecidos por SIM:

Roteiro para Documentação de Rotinas

– nome da rotina

– nome do(s) autor(es)

– data de criação

– data da última atualização

– funcionalidade da rotina

– histórico das modificações

- especificação de entrada ***e*** de saída:
  - nome do parâmetro
  - descrição
  - tipo
  - conjunto de valores possíveis
  - valor default
- definição das variáveis locais
- chamada por (rotina que chama)
- chama (lista de rotinas chamadas)
- funções chamadas
- arquivos acessados
- common blocks (Fortran)
- bibliotecas acessadas
- mensagens de erro
- exemplo

Roteiro para a Elaboração do Manual

- capa de rosto
  - nome da aplicação

    nome do manual
  - nome do(s) autor(es)
  - data
  - número da versão

- número de referência

– informações adicionais

- nota (informações gerais, tais como: referência à outras versões ou publicações)
- endereço do(s) autor(es)
- direitos autorais

– agradecimentos

– prefácio

– sumário

– lista de figuras

– lista de tabelas

– introdução

  visão geral da aplicação

- objetivo do manual

  descrição do conteúdo

- terminologia / notações
- dependências do sistema operacional
- instalação
- histórico das revisões
- conceitos básicos

– descrição da aplicação

- funcionalidade do sistema
- arquitetura do sistema

inicialização

- exemplo de uma execução

descrição de cada comando (documentação das rotinas)

A partir dos roteiros oferecidos pelo sistema, pode-se gerar, a princípio, o Manual de Manutenção do Programa e o Manual de Referência.

O **Manual de Manutenção do Programa** descreve todas as rotinas do programa de forma detalhada (variáveis, parâmetros, etc.). O **Manual de Referência** descreve as rotinas que o usuário final irá acessas, apresentando apenas as informações relevantes para a utilização do programa (não são citados os detalhes da implementação).

O **Manual do Usuário** pode ser produzido a partir do Manual de Referência adicionando-se outras informações. Esse manual deve considerar que, em muitos casos, os usuários finais não são da área de computação e apenas fazem uso do sistema, não necessariamente tendo que conhecer detalhes de sua construção. Pode ter conteúdos diversos, dependendo da aplicação.

V.3.2 Definição do formato do documento

Após a definição do conteúdo do manual, SIM oferece um terceiro roteiro para descrever o formato do mesmo. Cada informação que deve estar presente no manual é identificada por palavras-chave (`introduction`, `overview`, `commands`, `parameter`, etc.) e a cada uma delas são associadas macros, implementadas em TEX, que descrevem seu layout.

TEX foi escolhido para definir o formato dos manuais produzidos por SIM devido às suas capacidades como formatador de textos e a sua portabilidade.

D. Knuth da Universidade de Standford desenvolveu o TEX no fim dos anos 70 para geração de documentos contendo expressões matemáticas com alta qualidade de tipografia [14]. Novos algoritmos foram desenvolvidos para quebra de parágrafos em linhas, agrupamento de linhas em páginas e para hifenização.

Sua linguagem de especificação permite a descrição de objetos complexos (além

de expressões matemáticas [50]), tais como:

- tabelas;
- diagramas;
- referências cruzadas, por exemplo:
  - bibliografia;
  - índice remissivo;
  - sumário.
- gráficos (por definição de linhas e objetos geométricos).

Baseando-se nesse poderoso formatador de textos, outros programas foram desenvolvidos:

- LaTeX corresponde a um pacote de macros em TeX, escrito por Leslie Lamport, que adiciona ao último uma coleção de comandos em nível mais abstrato;
- DVITOPS transforma a saída do TeX ou LaTeX (formato *.DVI*) para formato PostScript [51] e também permite a importação de figuras (em formato PostScript) elaboradas em editores gráficos ou outros programas de formatação;
- XDVI permite a visualização do documento através de um vídeo gráfico.

Através da utilização das macros SIM o arquivo resultante contendo a documentação torna-se simples de ser entendido, pois a definição do **layout** das informações está presente em outro arquivo. A figura V.12 mostra como exemplo a documentação de uma rotina contendo macros SIM.

No exemplo apresentado, o formato das informações *SUBROUTINE* PHOTOS, Routine author, Creation date, Update, Input *specification*, Parameter *name*, *Parameter description*, Type, etc é definido separadamente no roteiro `$STY_GUI`.

```
                .

\routname{SUBROUTINE PHOTOS)

\routauthor{Routine author} B. van Eijk and Z. Was

\creation{Creation date) 26/11/90

\update{Update}  26/03/90

\inputpar{Input specification)
 \parname{Parameter name} IPPAR
 \pardescr{Parameter description} pointer to decaying particle
 \type{Type} INTEGER

\outputpar{Output specification} COMMON /HEPEVT/
                .
                .
```

Figura V.12: Exemplo de uma documentação contendo macros SIM

Às macros do SIM podem ser integrados comandos do LaTeX oferecendo ao usuário a possibilidade de produzir documentos ainda mais sofisticados.

Na ferramenta SIM, um documento é composto por partes:

- chapter;
- section;
- subsection;
- subsubsection;
- paragraph;
- subparagraph;
- cover, que corresponde à capa do manual;
- `tableofcontents`, `listoffigures` e `listoftables`, sumário, lista de figuras e lista de tabelas, respectivamente.

O usuário pode descrever o estilo da página de cada parte do documento através da macro `\make@pagestyle` e seus parâmetros:

– **parte:** *chapter, tableofcontents, section,* etc.

– **estilo:**

  · *empty,* sem cabeçalho e sem rodapé;

    plain, com o número da página no rodapé.

– **numeração:**

  · *roman,* numeração em números romanos: **i,ii,...;**

    arabic, numeração em números arábicos: **1,2,....**

– **página:** numeração da parte a partir do número **n.**

– **nova página:** define se começa ou não outra página.

Os estilos de página de *section, subsection,* `subsubsection`, *paragraph* e *subparagraph* seguem o estilo de *chapter*.

O estilo do cabeçalho de cada parte é definido através do macro `\make@head` e seguintes parâmetros:

– **parte:** *chapter, tableofcontents, section,* etc.

– **tamanho da fonte:**

  · *tiny*, como em tamanho;

  · *scriptsize*, como em **tamanho;**

  · *footnotesize*, como em tamanho;

    small, como em tamanho;

  · *normalsize*, como em tamanho;

  · *large*, como em *tamanho;*

Large, como em tamanho;

. LARGE, como em TAMANHO;

. *Huge*, como em **tamanho**.

– **tipo do estilo:**

roman, como em estilo;

· italic, como em estilo;

bold type, como em **estilo;**

· sans *serif*, como em estilo;

· slanted, como em estilo;

small caps, como em ESTILO;

· typewriter, como em `estilo`.

– **modo:** se centralizado, a direita ou a esquerda.

– **identação:** em centímetros.

– **régua:** com uma linha abaixo do cabeçalho.

– **distância anterior:** em relação à informação anterior (em centímetros).

– **distância posterior:** em relação à informação posterior (em centímetros).

As informações presentes no manual podem ser associadas a uma das partes do documento através da macro `\set`, como por exemplo: Visão Geral da Aplicação corresponde a section, cujo formato já foi definido pelo comando:

```
\make@head{section}...
```

ou, o formato informação pode ser definido diretamente pelo comando:

```
\make@head{overview}{Large}{sf}{center}{0cm}{rule}{3cm}{1cm}
```

gerando (figura V.13):

Figura **V.13:** Exemplo da apresentação física de uma informação

SIM oferece um formato padrão para o documento e, a partir de arquivos (ou *decks*, dentro do ambiente CMZ) contendo as informações do manual, baseando-se nos roteiros, o sistema gera um arquivo em LaTeX. O usuário apenas altera o roteiro contendo a definição do formato do documento caso deseje fazer alguma modificação.

Outra vantagem do sistema SIM é a possibilidade do usuário inserir diretamente comandos em LaTeX.

Roteiro para descrever o Formato do Manual

A definição, proposta por SIM, para o formato do manual se apresenta da seguinte forma:

a) Estilo do documento: corresponde a um comando do LaTeX que define o documento como do tipo report e seguindo o estilo SIM:

```
\documentstyle[sim]{report}
```

b) Folha de Rosto: define o estilo de página da folha de rosto e como suas informações devem ser apresentadas.

```
\make@pagestyle{cover}{empty}{}{}{}
\make@head{application}{LARGE}{bf}{center}{0cm}{}{2cm}{1cm}
\make@head{manual}{Large}{bf}{center}{0cm}{}{1cm}{1cm}
\make@head{author}{large}{rm}{center}{0cm}{}{1cm}{1cm}
\make@head{date}{large}{rm}{center}{0cm}{}{1cm}{1cm}
\make@head{version}{normalsize}{rm}{center}{0cm}{}{2.5cm}{1cm}
\make@head{ref}{normalsize}{rm}{center}{0cm}{}{0.2cm}{1cm}
```

Obs.: se o usuário não escrever uma data, LaTeX gerará a data atual.

c) Informações Adicionais

```
\make@pagestyle{chapter*}{plain}{roman}{1}{newpage}
\make@head{chapter*}{Huge}{bf}{left}{parindent}{rule}{2cm}{1cm}
\set{intrremarks}{chapter*}
\make@head{note}{normalsize}{rm}{left}{parindent}{}{3cm}{-0.2cm}
\make@head{address}{normalsize}{rm}{left}{parindent}{}{2cm}{-0.2cm}
\make@head{copy}{normalsize}{rm}{left}{parindent}{}{2cm}{-0.2cm}
```

d) Agradecimentos e Prefácio

```
\set{acknow}{chapter*}
\set{preface}{chapter*}
```

e) Sumário e listas

```
\make@pagestyle{tableofcontents}{}{}{}{newpage}
\make@pagestyle{listoffigures}{}{}{}{newpage}
\make@pagestyle{listoftables}{}{}{}{newpage}
\make@head{tableofcontents}{Huge}{bf}{right}{0cm}{rule}{2cm}{1cm}
\make@head{listoffigures}{Huge}{bf}{right}{0cm}{rule}{2cm}{1cm}
\make@head{listoftables}{Huge}{bf}{right}{0cm}{rule}{2cm}{1cm}
```

f) Partes do documento

```
\make@pagestyle{chapter}{plain}{arabic}{1}{newpage}
\make@head{chapter}{Huge}{bf}{left}{0cm}{rule}{2cm}{1cm}
\make@head{section}{Large}{bf}{left}{0cm}{rule}{1cm}{1cm}
\make@head{section*}{Large}{bf}{left}{parindent}{rule}{1cm}{0.5cm}
\make@head{subsection}{large}{bf}{left}{0cm}{rule}{1cm}{0.5cm}
\make@head{subsection*}{large}{bf}{left}{parindent}{rule}{1cm}{0.5cm}
\make@head{subsubsection}{normalsize}{bf}{left}{parindent}{rule}{1cm}{0.5cm}
\make@head{subsubsection*}{normalsize}{bf}{left}{parindent}{rule}{1cm}{0.5cm}
\make@head{paragraph}{normalsize}{bf}{left}{parindent}{}{0.5cm}{-0.2cm}
\make@head{paragraph*}{normalsize}{bf}{left}{parindent}{}{0.5cm}{0.2cm}
```

```
\make@head{subparagraph}{normalsize}{bf}{left}{1.0cm}{}{0.5cm}{-0.2cm}
\make@head{subparagraph*}{normalsize}{bf}{left}{1.0cm}{}{0.5cm}{0.2cm}
```

g) A **Introdução:** onde as informações introduction, overview of the application, purpose of this manual, description of its contents, terminology, operating systems dependencies, installation, revision history e basic concepts são definidas como partes do manual:

```
\set{introduction}{chapter}
\set{overview}{section}
\set{purpose}{section}
\set{contents}{section}
\set{terminology}{subsection}
\set{opsystem}{section}
\set{installation}{subsection}
\set{revhistory}{section}
\set{concepts}{section}
```

h) Descrição da Aplicação: onde as informações description of the application, functionality of the system, architecture of the system, initialization e execution são definidas como partes do manual:

```
\set{description}{chapter}
\set{functionality}{section}
\set{architecture}{section}
\set{initialization}{section}
\set{execution}{section}
```

i) Descrição de cada comando: as informações description of each command, routine name, routine author, creation date, update, functionality of the routine, modification history, input specification, output specification, parameter name, parameter description, type, range and/or set of values, default value, variables definition, called

by, calling routines, calling functions, accessed files (I/O), **common blocks, included libraries, error messages** e **example** são definidas como partes do manual:

```
\set{commands}{chapter}
\set{routname}{section}
\set{routauthor}{paragraph}
\set{creation}{paragraph}
\set{update}{paragraph}
\set{functionrout}{paragraph}
\set{history}{paragraph}
\set{inputpar}{paragraph}
\set{outputpar}{paragraph}
\set{parname}{subparagraph}
\set{pardescr}{subparagraph}
\set{type}{subparagraph}
\set{range}{subparagraph}
\set{defaultvalue}{subparagraph}
\set{variables}{paragraph*}
\set{called}{paragraph}
\set{routines}{paragraph*}
\set{functions}{paragraph*}
\set{files}{paragraph}
\set{common}{paragraph*}
\set{libraries}{paragraph}
\set{error}{paragraph}
\set{example}{paragraph}
```

V.3.3 Geração da Documentação

Através do comando *SET_DOCDIR*, o usuário define o nome do diretório de documentação. Caso não seja modificado, SIM armazenará toda a documentação no diretório default (`$DOC_DIR`).

O usuário define ao sistema que rotinas deseja documentar através do comando DOCUMENT, que extrai informações de um ou mais arquivos contendo código-fonte, tais como:

- nome da rotina
- nome do(s) autor(es) (identificado pelo comando AUTHOR do CMZ)
- data de criação (obtida através do mecanismo histórico do CMZ)
- data da última atualização (mecanismo histórico do CMZ)
- histórico das modificações (mecanismo histórico do CMZ)
- especificação de entrada e saída:
  - nome do parâmetro
  - tipo (inteiro, real, lógico, etc.)
- variáveis internas e respectivo tipo
- subrotinas chamadas
- arquivos acessados
- **common blocks** (código Fortran)

e todas as tags definidas pelo usuário. Portanto, o comando DOCUMENT extrai todas as informações que estiverem identificadas no roteiro para documentação de rotinas (`$ROU_GUI`) e chama o editor local. O usuário pode, então, acrescentar dados, gerando uma documentação completa da(s) rotina(s). O arquivo contendo a documentação é armazenado no diretório de documentação com o mesmo nome da rotina contendo o código-fonte.

Para a elaboração da parte introdutória do manual, o usuário pode editar o roteiro para elaboração do manual (`$MAN_GUI`), completando as informações e modificando-o de acordo com a aplicação.

Sem se preocupar com o formato do documento final, o usuário chama o comando *DTOT* com os seguintes argumentos: nome do arquivo a ser gerado, nome do *deck* contendo a parte introdutória do manual e nome dos *decks* contendo a documentação das rotinas. O comando *DTOT* gera um arquivo correspondente ao documento final, contendo comandos LaTeX, no sistema operacional. Esse documento segue o formato padrão do SIM mas, se não for apropriado, o usuário pode modificar os parâmetros que definem o **layout** das informações, descritos no roteiro `$STY_GUI`.

Caso o usuário deseje utilizar outro formatador de textos, um mesmo arquivo contendo a documentação pode ser gerado no sistema operacional, sem comandos LaTeX, através do comando *CTOT* do CMZ.

O apêndice A apresenta um exemplo de como produzir documentos através da ferramenta SIM.

V.4 Ferramentas utilizadas no desenvolvimento

As seguintes ferramentas computacionais foram utilizadas na construção de SIM tanto como parte da implementação do código quanto como sistemas de apoio ao seu desenvolvimento:

- CMZ: para gerenciamento e manutenção do código da ferramenta;
- KUIP: utilizado para a gerência da interface de SIM;
- BISON [52] (versão avançada do YACC [53]) e FLEX [54] (versão avançada do LEX [55]): ferramentas utilizadas para gerar os analizadores sintático e léxico das palavras-chave dos roteiros;
- TeX e LaTeX: utilizadas para definir os macros que descrevem o formato do manual gerado por SIM;
- PATCHY: para o desenvolvimento e a manutenção do código-fonte; (diferentes instruções para os ambientes computacionais IBM, VAX, etc.; e definição de *COMMON blocks*.)

- ZEBRA: sistema para gerenciamento de estruturas de dados; (incorporado ao CMZ.)
- DecWindows: sistema de interface com o usuário baseado em X-Windows;
- Sistemas Operacionais VMS (VAX) e VM/CMS (IBM).

V.5 Métodos e Técnicas de Engenharia de Soft-wase Envolvidos no Besenvolvirnento de SIM

Para a integração das ferramenta SIM e CMZ, foi realizado um estudo dessa última através de técnicas de *Engenharia Reversa* [56][57], pois não há nenhuma documentação referente à especificação ou ao projeto desse sistema. CMZ é um exemplo típico, e bastante comum no CERN, de um programa que cresceu em um ambiente científico a partir de um pequeno protótipo, sem especificação de requisitos. A partir da análise do código e da utilização da ferramenta, foi possível entender o funcionamento e estrutura da mesma.

Objetivando-se manter uma compatibilidade com as rotinas do CMZ, para a implementação das rotinas do SIM foram utilizadas técnicas de reutilização [58] de software. Em cada rotina do CMZ que realiza funções similares às da ferramenta SIM foram identificados seus procedimentos (ou seja, partes do código) e esses reutilizados nas rotinas do SIM.

A funcionalidade das rotinas que compõem os comandos do SIM são mostradas a seguir:

1. comando **DOCUMENT,** formado por:
   - **SDOCU** (similar à rotina **CEDIT** do CMZ): verifica os seguintes requisitos mínimos:

     se o arquivo CMZ já está conectado;

     se o sistema não está sendo processado em modo *batch;*

     · se há permissão para escrever no arquivo CMZ.

- SIMEXE (parte modificada da rotina CMEXEC do CMZ): lê os parâmetros da linha de comandos, que podem ser:

  opções do CMZ: número da versão, verificando sua validade;

  - *decks*, diretórios ou um arquivo inteiro a serem documentados.

- SIMARG (similar à rotina CMXARG do CMZ): verifica se a opção fornecida pelo usuário corresponde a uma das opções do comando DOCUMENT.
- SMDECK (parte modificada da rotina CMEXEC do CMZ): cria o diretório de documentação (caso não exista) e processa os parâmetros a serem documentados.
- SMWRRO: escreve o roteiro default para documentação de rotinas em um vetor.
- SMWRGU (similar à rotina CMMPAT do CMZ): cria o *deck* ($ROU_GUI) contendo o roteiro para documentação de rotinas (caso não exista) no diretório de documentação.
- SMWRMA: escreve o roteiro default para documentação de manuais em um vetor.
- SMWRMGU (similar à rotina CMMPAT do CMZ): cria um *deck* ($MAN_GUI) contendo o roteiro para documentação de manuais (caso não exista) no diretório de documentação.
- SMWRSGU (similar à rotina CMMPAT do CMZ): cria um *deck* ($STY_GUI) contendo a definição do formato dos manuais (caso não exista) no diretório de documentação.
- SMEXCP (similar à rotina CMEXCP do CMZ): processa cada *deck*, caso o parâmetro seja um diretório.
- SIMDOC: gera a documentação de um *deck* (contendo código Fortran) e a escreve no arquivo do editor.

2. comando SETDOCDIR, formado por:

- SDOCDIR: verifica a sintaxe do parâmetro e armazena o nome do novo diretório de documentação.

3. comando DTOT, formado por:

- SIMEXE, SIMARG, SMDECK, SMWRRQ, SMWRGU, SMWRMA, SMWRMGU, SMWRSGU e SMEXCP descritas acima.
- SMWRMC: cria vetores auxiliares ao processamento contento os ítens da documentação, em sua ordem padrão (caso os roteiros não existam no diretório de documentação).
- SMMCWR: cria vetores auxiliares ao processamento contento os ítens da documentação, na ordem definida pelo usuário (caso os roteiros já existam no diretório de documentação).
- SMEXSTY: exporta o *deck* contendo a definição do formato físico para o sistema operacional.
- SIMTEX: cria o arquivo da documentação com comandos LaTeX.

e as funções :

- SMCOMP: compara dois **strings** e retorna o número de caracteres que são iguais.
- SMNOSL: retorna um **string** sem o caracter de controle \.
- SMLTSPC: retorna um **string** tratando os comandos especiais do LaTeX.

Essas rotinas do SIM também chamam algumas rotinas dos seguintes sistemas:

- *CMZ*: CREPAD, EDITOR, CVERS, IVERSN, CMDLST, TOPTUV, CHKOPT, CMDECK, ITOC, GETCYC, CMMPAT, CMZVIN, CEDITB, COMMNT, WTIMPLT, CPLINE e DPLINE;
- ZEBRA: RZCDIR, RZSAVE, RZINK, RZMDIR, RZVIN e RZVOUT;
- *KUIP*: KUPATL e KUGETC;

- *CERNLIB*: ZITOH, UHTOC, UCTOH, UCOPY, SPACES e CUTOL;
- *VAX/VMS*: LIB$WAIT.

Durante a construção da ferramenta SIM foram aplicadas técnicas de Engenharia de Software. A especificação de SIM foi definida através de um relatório técnico para o Grupo de Simulação de Monte Carlo [59], onde foram apresentados:

- objetivo e proposta da ferramenta;
- uma proposta da arquitetura do sistema, descrevendo a funcionalidade de cada módulo;
- uma proposta da linguagem para a documentação;
- o conteúdo da documentação a ser gerenciada;
- uma proposta para o processo de desenvolvimento, de forma incremental, da ferramenta.

A proposta inicial foi discutida entre os membros do grupo e a idéia de integrar SIM à ferramenta CMZ, para o gerenciamento do código, foi apresentada. A partir das sugestões discutidas, a proposta inicial foi modificada e, após a aprovação do grupo partiu-se para a construção do primeiro protótipo da ferramenta. A idéia final foi, então, apresentada na conferência "Computing in High Energy Physics '91" [60], no Japão.

Durante a fase de projeto, os comandos da ferramenta, implementados através do programa de interfaces KUIP, foram definidos e apresentados em outro relatório técnico para o Grupo de Simulação de Monte Carlo [61], permitindo que o grupo acompanhasse o desenvolvimento da ferramenta e contribuísse para a construção de uma ferramenta adequada às necessidades dos usuários.

Para avaliação da qualidade do código, durante a fase de implementação, as seguintes ferramentas, disponíveis no CERN, foram utilizadas com os seguintes objetivos:

- Floppy *Coding Convention Checker*: para verificar se o código da ferramenta segue as convenções padrões (correspondem aos ítens precedidos por $*$) apresentados na tabela 111.1.

- CMZ:

  - identificação de variáveis indefinidas (sem inicialização no código) ou variáveis utilizadas antes de sua inicialização (comando UNDEFINED);
  - geração da árvore de chamada a rotinas pelo comando TREE;
  - identação do código pelo comando INDENT;
  - renumeração dos labels pelo comando LABEL.

Para validação da ferramenta SIM, durante a fase de testes, foram utilizados os programas do Grupo de Simulação de Monte Carlo: PHOTOS (Photon Radiation in Decays) [62] e *PDKDB* (Data Base for Particles and Decays Channels). Os códigos desses programas foram organizados na ferramenta CMZ em *decks*, diretórios e arquivos CMZ e a sua documentação foi gerada através da ferramenta SIM.

Os programas PHOTOS e PDKDB foram escolhidos pois representam sistemas típicos para pesquisas em física e reúnem as características de programas científicos, tais como, desenvolvimento incremental, código pouco documentado e grande utilização de fórmulas matemáticas.

Através da geração da documentação relativa a esses programas, pode-se verificar que os resultados obtidos corresponderam aos esperados. Coube aos desenvolvedores desses programas completar a documentação produzida para a geração dos manuais completos, atividade que fugia ao escopo da avaliação da ferramenta SIM. Através dessa validação, verificou-se que os roteiros oferecidos por SIM foram facilmente adaptados às aplicações e a documentação produzida foi satisfatória, atendendo às necessidades dos usuários.

Capítulo VI

Conclusões

São apresentadas, neste capítulo final, as conclusões do trabalho de tese e sugestões para trabalhos futuros.

VI.1 Importância e Aplicabilidade do Trabalho

Esta tese apresenta a ferramenta SIM e as pesquisas necessárias à escolha de uma implementação adequada. Teve-se como objetivo definir um sistema para produção semi-automática da documentação de produtos de software, que respeitasse as características de ambientes de desenvolvimento de software científico.

Para tanto, foi essencial identificar as características dos processos de desenvolvimento de software científico e estudar a produção da documentação dos produtos de software.

O trabalho apresentado está inserido no Projeto de Colaboração Internacional da UFRJ com o CERN, em Genebra, Suíça. No CERN são realizadas as mais avançadas pesquisas na área de Física de Altas Energias e a computação é uma ferramenta indispensável. Muitos dos sistemas computacionais utilizados nesse centro são desenvolvidos pelos próprios físicos que, certamente, dominam a teoria de sua área mas carecem de conhecimento formal em ciência da computação.

A experiência de trabalho no CERN permitiu identificar diversas características do desenvolvimento de software científico, tais como: carência de documentação (principalmente nas fases de especificação e projeto), falta de um método específico de desenvolvimento, baixa qualidade da interface com o usuário e dos manuais de utilização do software, bem como a tendência dos produtos a sofrer diversas modificações.

O CERN comporta uma excelente infra-estrutura computacional tanto no que se refere a hardware (IBM, VAX, VaxStation, CRAY, APOLLO, SUN, MacIntosh,

NeXT, etc), quanto a software (sistemas operacionais, ferramentas gráficas, aplicativos, editores e processadores de textos, compiladores, etc). O desenvolvimento da tese nesse centro possibilitou, portanto, acessar diferentes sistemas para documentação disponíveis em diversas plataformas.

A integração de profissionais de ciência da computação com os cientistas que desenvolvem aplicações para apoio a suas pesquisas torna-se essencial dado que a quantidade de dados e a complexidade dos experimentos é cada vez maior. Uma das funções essenciais da participação de especialistas em computação, nesses ambientes, é possibilitar a introdução de métodos e técnicas atuais de desenvolvimento de software nessess contextos.

A principal característica da ferramenta SIM é o fato de ter sido desenvolvida considerando as especificidades dos ambientes onde se desenvolve software científico e dos próprios desenvolvedores.

A proposta apresentada por SIM procura levar o usuário a buscar a qualidade de seu produto, ou seja, que o próprio usuário se dê conta que, após seguir um método adequado à construção do software, seu produto se torna mais fácil de ser modificado, entendido e utilizado. Dessa forma, o desenvolvedor de aplicações científicas é encorajado a buscar novas ferramentas e métodos.

Através dos roteiros, a atividade de produzir um manual torna-se simples e eficiente. Permite-se, dessa forma, a geração de documentos padronizados, facilitando seu entendimento e manipulação.

A integração das ferramentas para produção do código (CMZ) e da documentação (SIM) permite ao desenvolvedor verificar que essas atividades devem ser realizadas concomitantemente e a importância de considerar o usuário durante a implementação de um sistema.

As ***tags*** oferecidas por SIM permitem que os usuários aproveitem seus trabalhos anteriores sem ter que redefiní-los para utilizar a nova ferramenta (caso clássico da documentação inserida do código).

A produção de um documento com alta qualidade de tipografia (através do LaTeX), sem exigir do usuário um conhecimento prévio de comandos de formatação,

facilita a existência de documentação pois, geralmente, se o esforço da geração é muito grande, essa tarefa é facilmente ignorada.

VI.2 Futuras Pesquisas

O ambiente integrando as ferramentas SIM e CMZ apoia a fase de construção do software, a qual engloba as atividades de iinplementação do código e geração dos manuais. Esse ambiente considera as características do desenvolvimento de software científico, no que diz respeito tanto à produção dos programas (desenvolvimento incremental) quanto dos documentos (roteiros adaptados à área científica).

O desenvolvimento de SIM baseia-se no método de versões sucessivas. A primeira versão, objeto desta tese, foi construída e avaliada. Durante sua implementação, idéias e sugestões foram cadastradas para serem introduzidas nas futuras versões. Algumas delas são apresentadas a seguir:

- documentação de código em C;
- inclusão de novos roteiros para:
  - documentação de sistemas compostos por diversos programas;
  - planejamento de testes do programa;
  - interface com o usuário.
- geração das diferenças entre as informações do código e sua documentação (ao invés de sempre gerar a documentação);
- geração de documentos em SGML e outros formatadores de texto;
- inclusão de comandos dentro do ambiente CMZ+SIM que permitam a visualização do documento (XDVI, por exemplo);
- geração de arquivos para a interface com o usuário.

Em acréscimo a esses ítens, SIM também pode ser estendido para apoiar testes do programa, verificar a integração de suas rotinas e fornecer um banco de dados para gerenciamento de suas entradas e resultados. Um sistema computacional na área científica corresponde, em geral, a um conjunto de procedimentos que são executados de acordo com os dados de entrada. Em outras palavras, um grupo de instruções do código ou um conjunto de rotinas é executado dependendo de determinada entrada. Faz-se, portanto, necessário manter todas as informações de entrada possíveis verificando-se, assim, se todos os procedimentos do código são corretos.

Para fases anteriores, tais como, as de especificação e de projeto, novos ambientes podem ser construídos, sempre considerando as características dos sistemas científicos. Um trabalho bastante útil será construir-se novas ferramentas cujas saídas sejam compatíveis com o sistema CMZ+SIM.

Não deve-se desconsiderar que, em um ambiente de desenvolvimento de software científico, a construção do código é vista como a atividade principal e talvez, mesmo, a única relevante. Devido a esse fato, os métodos implementados em ferramentas para apoio às fases de especificação e projeto não devem ser complexos e nem muito detalhados. A utilização, mais uma vez, de roteiros auxiliaria os cientistas a organizar suas idéias e definir a estrutura do produto a ser construído.

Finalmente, pode-se considerar SIM, nessa sua primeira versão, como uma ferramenta útil e adequada ao desenvolvimento de software científico e, em especial, ao ambiente CERN.

Referências Bibliográficas

[1] Rocha, **A.** R. C.; Aguiar, T. C.; Palermo, S.; Souza, J. M.; Falcão, J. F.; D'Ipólito, C.; *Uso de Técnicas de Engenharia de Software no Desenvolvimento de Software Científico*; XV Conferência Latino-Americana de Informática; Santiago, Chile; 1989.

[2] Rocha, **A.** R. C.; Palermo, S.; *Software Quality Assurance in High Energy Physics;* Computing in High Energy Physics; Oxford, England; 1989.

[3] Howard, J.; *What is good documentation?;* BYTE; março 1991.

[4] Retting, M.; *Nobody reads Documentation;* Communications of the ACM, julho 1989.

[5] Loy, P. H.; *A Comparison of Object-Oriented and Structured Development Methods;* ACM Sigsoft - Software Engineering Notes; janeiro 1990.

[6] Lamport, L.; LaTeX*: A Document Preparation System;* Addison-Wesley, Reading, Massachusetts; 1986.

[7] CMZ: A Source Code Management System - User's Guide & Reference Manual; CERN, Genebra, Suíça; 1990.

[8] Humphrey, W. S.; *The Software Engineering Process: Definition and Scope;* in proceedings of the *4th International Software Process Workshop*; *ACM* SIGSOFT Software Engineering Notes; junho 1989.

[9] *Guidelines for Documentation of Computer Programs and Automated Data Systems;* National Bureau of Standards; Federal Information Processing Standards Publications 38; 1976.

[10] *Text, ConText, and HyperText: writing with and for the computer;* Edward Barrett editor; MIT Press; Massachusetts, EUA; 1988.

[11] Cunto, J.; Araujo, J.; Giovannetti, F; Rivero, J.; FPLUS: A Programming Environment for *Scientific* Applications; IEEE SOFTWARE, setembro 1991.

[12] Houghton, R. C.; Wallace, D. R.; Characteristics and Functions of Software Engineering Environments: An Overview; ACM SIGSOFT Software Engineering Notes; janeiro 1987.

[13] Price, J.; How to write a computer manual: a handbook of software documentation; The Benjamin/Cummings Publishing Company; 1984.

[14] Document Preparation Systems: A Collection of Survey Articles; Nievergelt, J.; Coray, G., Nicoud, J.-D.; Shaw, A.C. editors; North-Holland Publishing Company; Amsterdam; 1982.

[15] Knuth, D. E.; The *TEXbook*; Addison-Wesley, Reading, Massachusetts; 1984.

[16] Schwarz, N.; Introduction to *TEX*; Addison-Wesley, Reading, Massachusetts; 1989.

[17] Knuth, D. E.; *TEX* and METAFONT: New Directions in Typesetting; American Mathematical Society and Digital Press; 1979.

[18] André, J.; Furuta, R.; Quint, V. editors; Structured Documents; Cambridge, Cambridge University Press; New York, USA; 1989.

[19] Teskey, F. N.; *Principles* of Text Processing; Ellis Horwood series in computers and their applications; 1982.

[20] Design Management Environment - Technical Overview; versão 3.03; Mentor Graphics Corporation; 1990.

[21] Scaling Up: A Research Agenda For Software Engineering; National Academy Press; Washington, 1989.

[22] Loken, S. C. et al.; Computing for High Energy Physics in the 1990s; in Proceedings of Summer Study on High Energy Physics in the 1990s, Sharon Jensen ed.; Snowmass, Colorado, USA; 1990.

[23] Zanella, P.; 30 Years of Computing at CERN; CERN - Computing & Networks Division, CN/90/2; CERN, Genebra, Suíça; 1990.

[24] Jenkins, J. S.; Davis, M.; DiGiacomo, N. J.; Killian, K.; Software Development in Large Engineering Projects: Lessons Learned; in Proceedings of Summer Study on High Energy Physics in the 1990s, Sharon Jensen ed.; Snowmass, Colorado, USA; 1990.

[25] Palermo, S.; Rocha, A. R. C.; An *Experience* on Software Quality Assurance; International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics; Lyon, França; 1990.

[26] Palermo, S.; Rocha, A. R. C.; A Proposal to Evaluate Program Quality for DELPHI *off-line* Environment; Relatório Técnico do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Brasil; 1988.

[27] Bunn, J. J.; Floppy *User* Guide (5.00); CERN; DD/US/112; Genebra, Suíça; 1988.

[28] Klein, H. J.; Zoll J.; Patchy - Reference Manual; CERN, Genebra, Suíça; 1988.

[29] Brun, R.; Zoll, J.; ZEBRA User's Guide - version 3.53; CERN, Genebra, Suíça; 1987.

[30] Zoll, J.; ZEBRA Memory Management Reference Manual - version 3.58; CERN, Genebra, Suíça; 1988.

[31] Zoll, J.; ZEBRA Error Diagnostics Reference Manual - version 3.61; CERN, Genebra, Suíça; 1989.

[32] Zoll, J.; ZEBRA Sequential I/O Reference Manual - version 3.62; CERN, Genebra, Suíça; 1989.

[33] Brun, R.; Zanarini; *KUIP (Kit* for an User Interface Package) - User's Guide, version 1.0; CERN - Data Handling Division, Genebra, Suíça; 1988.

[34] Text-Processing: An overview; Computer Newsletter articles no. 202_5_1; CERN, Genebra, Suíça; 1990.

[35] Jonghe, J. de; *Using* LaTeX on CERNVM; Computing & Networks Division, CERN/DD/US/121; CERN, Genebra, Suíça; 1988.

[36] Goossens, M.; at CERN; Computing & Networks Division, CERN/CN/US/136; CERN, Genebra, Suíça; julho 1991.

[37] McGrew, P. C.; McDaniel W. D.; In-house publishing in a mainframe environment; Macmillan database/data communications series; 1987.

[38] Rocha, A. R. C.; Análise e Projeto Estruturado de Sistemas; Editora Campus; Rio de Janeiro, Brasil; 1987.

[39] Molich, R; Nielsen, J.; Improving a Human-*Computer* Dialogue; Communications of the ACM; março 1990.

[40] Menezes, I. M. G.; Rocha, A. R. C.; Como Produzir Documentação de Boa Qualidade; Relatório Técnico ES-107/86 do Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Brasil; 1986.

[41] Poston, R. M.; When does more documentation mean less *work?*; IEEE Software; outubro 1984.

[42] Sufrin, B.; Formal Methods in System Design and Implementation; Oxford University Programming Research Group.

[43] Brooks, K. P.; Lilac: a Two-View Document Editor; IEEE COMPUTER; junho 1991.

[44] Fidge, C.; Heagerty; Everything You Ever Wanted to Know About Documentation (and should have *asked)*; Telecom Australia, Research Laboratories, Austrália; 1986.

[45] Katzin, E.; How to write a really good user's manual; Van Nostrand Reinhold Company; New York, USA; 1985.

[46] Esteveny, L.; CERNDOC - the DD Documentation Scheme; CERN, Genebra, Suíça; 1987.

[47] CERN Support Group; CERNPAPER *User's* Guide; CERN, Genebra, Suíça; 1985.

[48] Herwinjnen, E. van; SGML (Standard Generalized *Markup* Language) - Text Processing System based on ISO Standard 8879; CERN, Genebra, Suíça; 1986.

[49] Department of Computing Services, University of Waterloo; Waterloo SCRIPT version 83.1 - Reference Manual; Waterloo, Ontario, Canadá; 1984.

[50] Knuth, D. E.; Larabee, T.; Roberts, P. M.; Mathematical Writing; Mathematical Association of America; 1989.

[51] Adobe Systems; *PostScript* language: Reference *Manual*; 2^{nd} edition; Addison-Wesley Publishing Company, Inc.; Palo Alto, USA; 1990.

[52] Donnely, C.; Stallman, R.; BISON; versão 1.10; Free Software Foundation; Cambridge, USA; 1990.

[53] Johnson, S. C.; *Yacc:* Yet Another Compiler-Compiler; Computer Science Technical Report; Bell Laboratories, Murray Hill, New Jersey, USA; 1975.

[54] Paxson, V. et al.; Flex - *fast* lexical analyzer generator; versão 2.3; University of California, Berkeley, USA; 1990.

[55] Lesk, M. E.; Schimidt, E.; Lex - A Lexical Analyzer Generator; Computer Science Technical Report; Bell Laboratories, Murray Hill, New Jersey, USA; 1975.

[56] Choi, S.; Scacchi, W.; Extracting and Restructuring the Design of Large Systems; IEEE Software; janeiro 1990.

[57] Chikofsy, E.; Cross, J.; *Reverse* Engineering and Design Recovery: a Taxonomy; IEEE Software; janeiro 1990.

[58] Redwine, S. T.; Riddle, W. E.; Software Reuse Process; ACM Sigsoft - Software Engineering Notes; junho 1989.

[59] La Commare, G.; Maidantchik, C.; Rocha, A. R. C.; Xexéo, G.; SIM: A Software Information Manager; Relatório Técnico CERN/LAA-MSL/90-16, CERN, Genebra, Suíça; 1990.

[60] Maidantchik, C.; Rocha, A. R. C.; La Commare, G.; Souza, J. M.; SIM: A Software Information Manager; "Computing in High Energy Physics '91" Universal Academy Press, Inc.; Tókio, Japão; 1991.

[61] Maidantchik, C.; La Commare, G.; An Introduction to the Software Information Manager (SIM); Relatório Técnico CERN/LAA-MSL/91-03; CERN, Genebra, Suíça; 1991.

[62] Barberio, E.; van Eijk, B.; Was; Zbigniew; PHOTOS version *1.3* - *User* Guide and Reference Manual; CERN/LAA-MSL/90-04, Genebra, Suíça; 1990.

[63] CERN Computer Center; CERN Program Library; CERN, Genebra, Suíça; 1989.

Apêndice A
Exemplo de Utilização da Ferramenta SIM

Neste apêndice, é apresentado um exemplo de utilização da ferramenta SIM, que descreve os procedimentos que o usuário deve seguir para gerar a documentação de um programa.

O programa que foi escolhido como exemplo se denomina PHOTOS, foi desenvolvido por físicos do Grupo de Simulação de Monte Carlo e está disponível no CERN através da CERNLIB [63], a biblioteca de programas do CERN.

PHOTOS está escrito em Fortran-77 e se encontra em formato Patchy o que permite sua instalação em diferentes plataformas.

A ferramenta CMZ foi adotada como sistema de gerenciamento de código-fonte. Dentro do ambiente CMZ, o arquivo correspondente ao programa também é denominado PHOTOS. Para efeito de simplificação, são apresentadas somente quatro das 16 rotinas e uma das 7 funções que compõem o programa. As rotinas e funções encontram-se no diretório CODE.

Para ativar o programa CMZ+SIM, o usuário chama o comando CMZSIM do sistema operacional (figura A.1).

```
VxCern> CMZSIM
```

Figura A.1: Entrada no sistema CMZ+SIM

O programa informa o número da versão, modificações e novas facilidades incluídas em relação à versão anterior, e o prompt é alterado (CMZ [0]), informando ao usuário que agora esse se encontra no ambiente CMZ+SIM (figura A.2).

Dentro do ambiente CMZ+SIM, o usuário se conecta ao arquivo PHOTOS através do comando FILE (figura A.3).

```
**** CMZ Code Manager version  1.35/01  (07/11/90) ****

*** Bug fixes and new features since 1.35/00 ***

Oct 30: new options -B and -V in READ (see  HELP READ)

CMZSIM [0]
```

Figura **A.2:** Inicializações do ambiente CMZ+SIM

```
CMZSIM [0] FILE PHOTOS
photos [1]
```

Figura **A.3:** Conexão do arquivo PHOTOS dentro do ambiente CMZ+SIM

O arquivo PHOTOS é composto pelo diretório CODE, que contém as rotinas PHOTOS, PHOINI, PHOCIN, PHOENE e a função PHOFAC (figura **A.4).**

```
photos [1] DIR
Current Working Directory = //PHOTOS
Following subdirectories :
    CODE
Following DECKS :
Number of DECKS =   0 Number of CYCLES =   0
photos [1] CDIR CODE
photos/code [2] DIR
00_patch;1  PHOTOS;3    PHOINI;2    PHOCIN;1    PHOENE;1    PHOFAC;1
Number of DECKS =   6 Number of CYCLES =   6
```

Figura **A.4:** *Decks* que compõem o arquivo CMZ PHOTOS

No diretório CODE, que contém as rotinas, o usuário lista, através do comando READ, o deck PHOTOS (figura **A.5).**

```
photos/code C31 READ PHOTOS
```

Figura **A.5:** Comando para listar um ***deck***

O deck PHOTOS corresponde à rotina PHOTOS, implementada em Fortran, que já contém a tag *$ERR, identificando mensagens de erro (figura **A.6):**

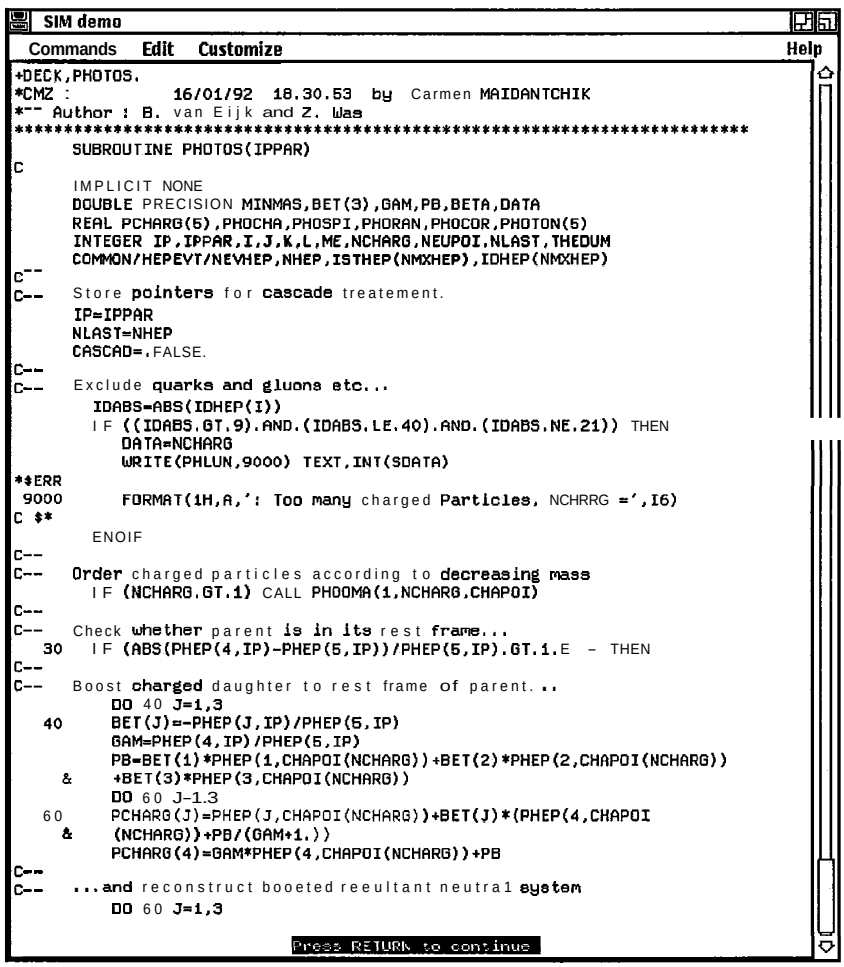

Figura A.6: ***Deck*** PHOTOS contendo a rotina PHOTOS

A rotina PHOTOS também contém a tag *$PHOTON, que identifica comentários incluídos no código sobre cálculos com a partícula *PHOTON*. O usuário chama o comando *DOCUMENT* (figura **A.7**) que:

- cria o diretório de documentação $DOCDIR e os roteiros $ROU_GUI, $MAN_GUI e $STY_GUI, caso não existam;
- gera a documentação relativa à rotina PHOTOS;
- cria um novo *deck*, com o mesmo nome do que contém o código no diretório $DOCDIR, contendo a documentação.

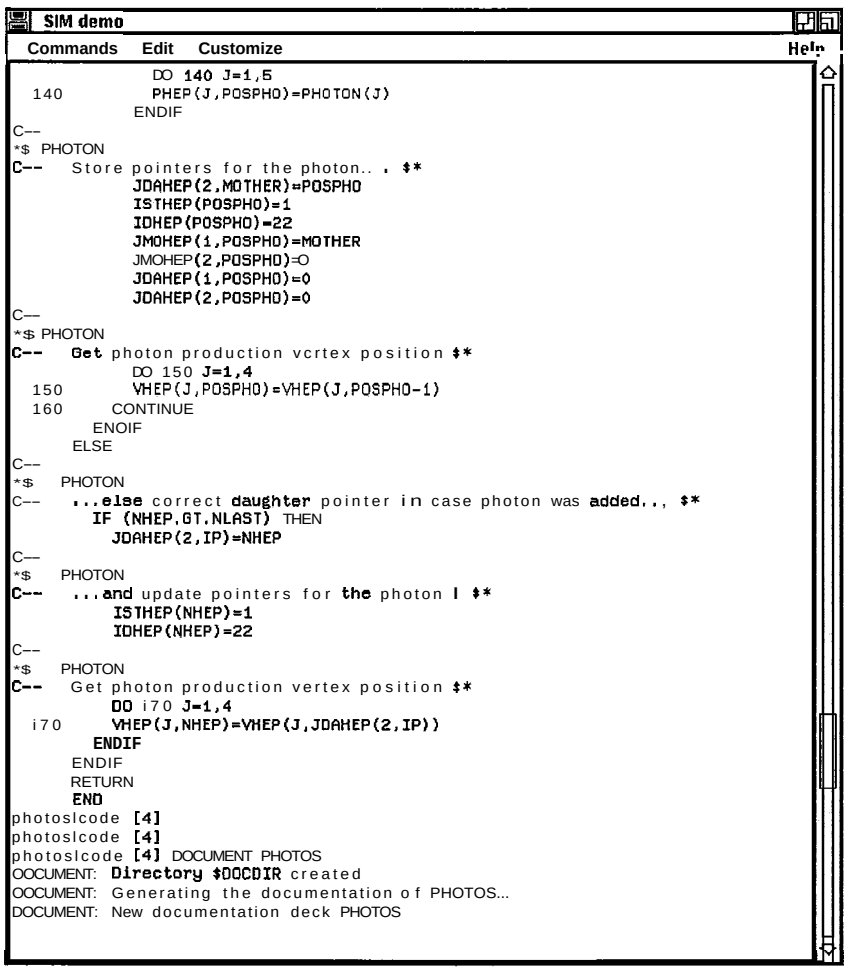

Figura **A.7:** Exemplo da execução do comando *DOCUMENT*

O comando DOCUMENT chama diretamente o editor, contendo o *deck* com a documentação, permitindo que o usuário inclua informações adicionais (figura **A.8):**

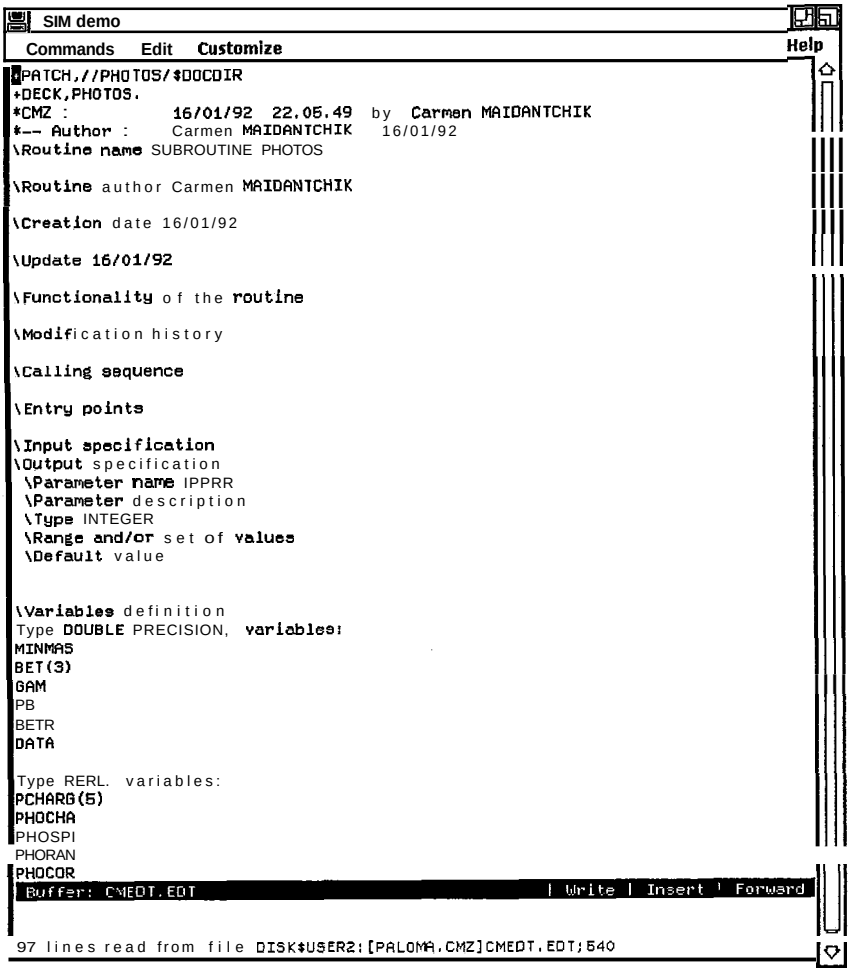

Figura **A.8:** Documentação gerada e editada através do editor local

A figura **A.9** mostra a documentação anterior com as informações incluídas pelo próprio usuário:

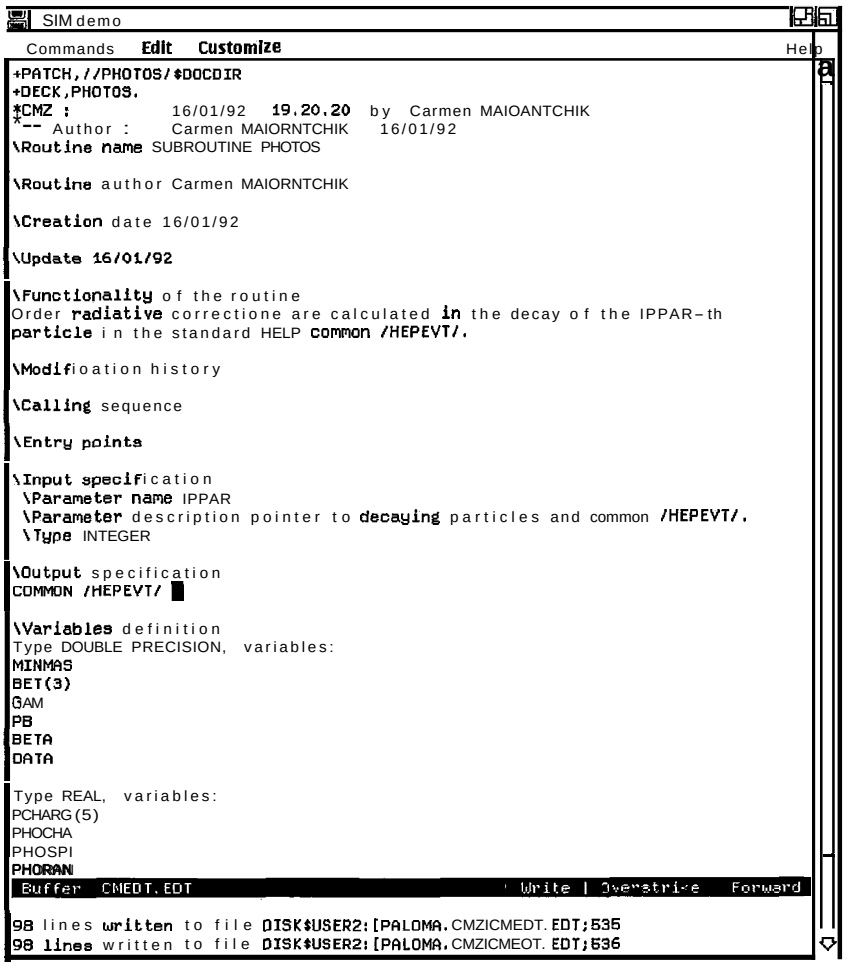

Figura **A.9:** Documentação com informações incluídas pelo usuário

A figura A.10 mostra que o diretório $DOCDIR, os roteiros $ROU_GUI, $MAN_GUI e $STY_GUI e o *deck* PHOTOS, contendo a documentação relativa à rotina PHOTOS, foram criados através do comando ***DOCUMENT.***

Caso o roteiro para documentação de rotinas não seja adequado à aplicação, o usuário pode modificar o ***deck*** $ROU_GUI através de um editor de textos, chamado pelo comando ***EDIT*** (figura A.lO):

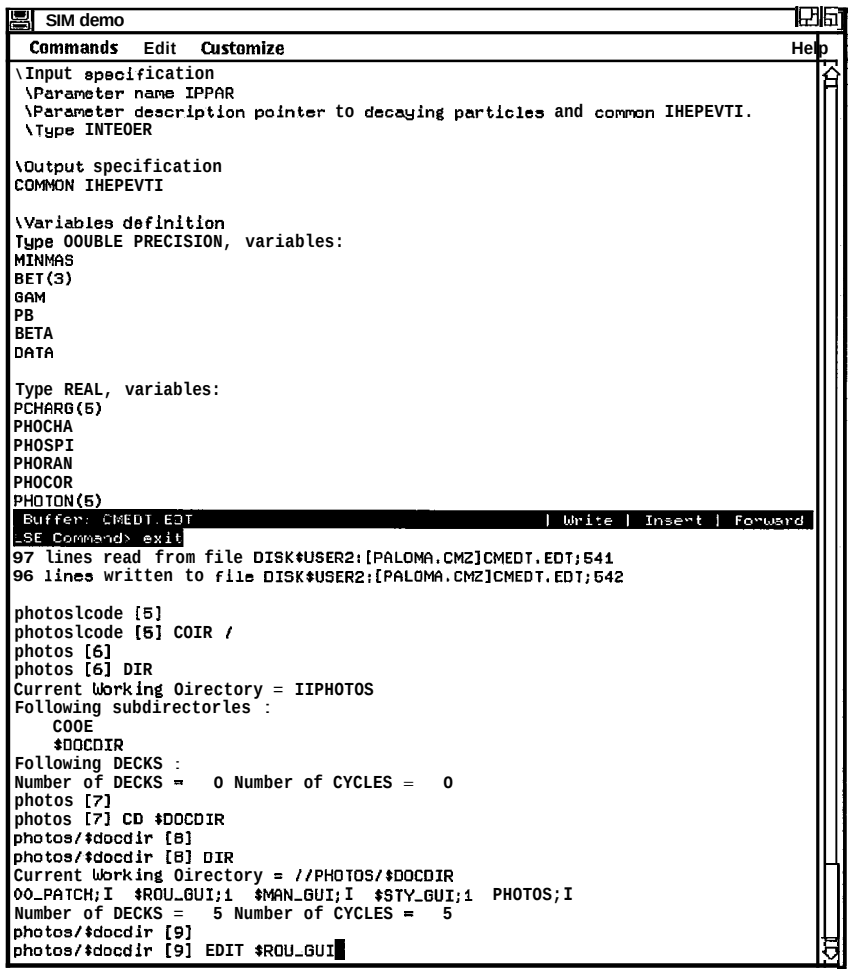

Figura A.10: Chamada ao comando ***EDIT***

Pelo editor de textos, o usuário pode modificar o roteiro para documentação de rotinas (`$ROU_GUI`), acrescentando as tags `*$ERR` e `*$PHOTON` e deletando outros ítens como é mostrado na figura **A.ll:**

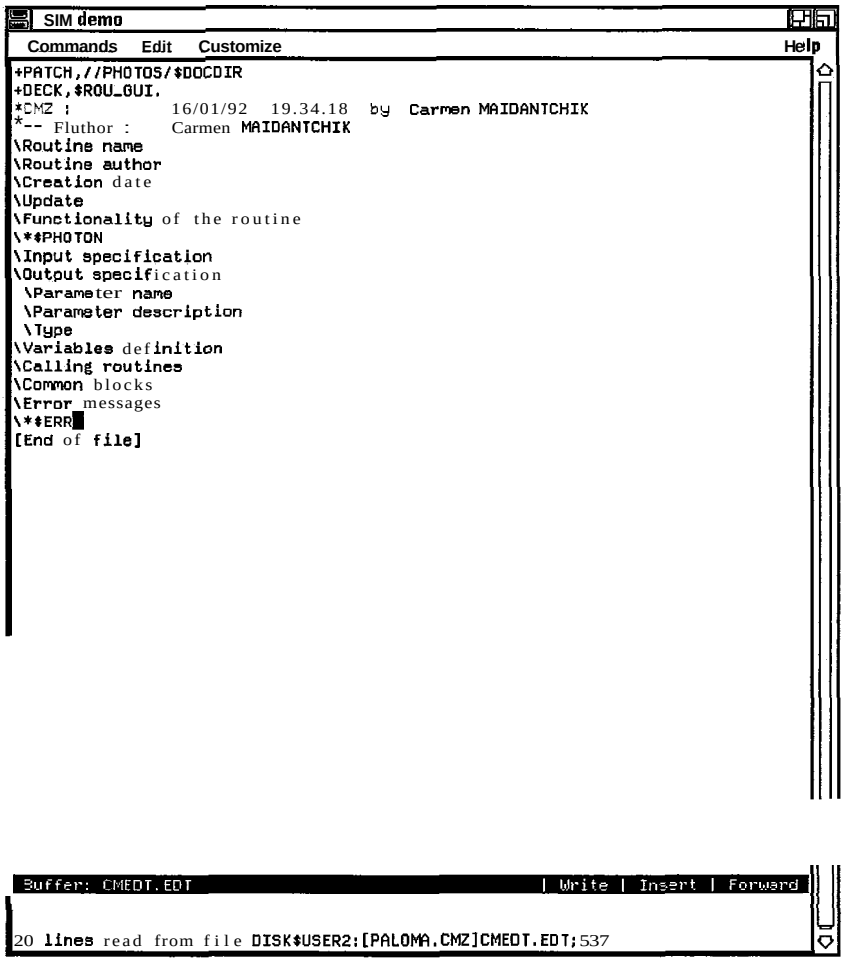

Figura A.ll: Exemplo do roteiro `$ROU_GUI` modificado pelo usuário

Na figura A.12 é apresentada a nova documentação gerada, de acordo com as modificações definidas no roteiro $ROU_GUI, quando o comando DOCUMENT é chamado novamente:

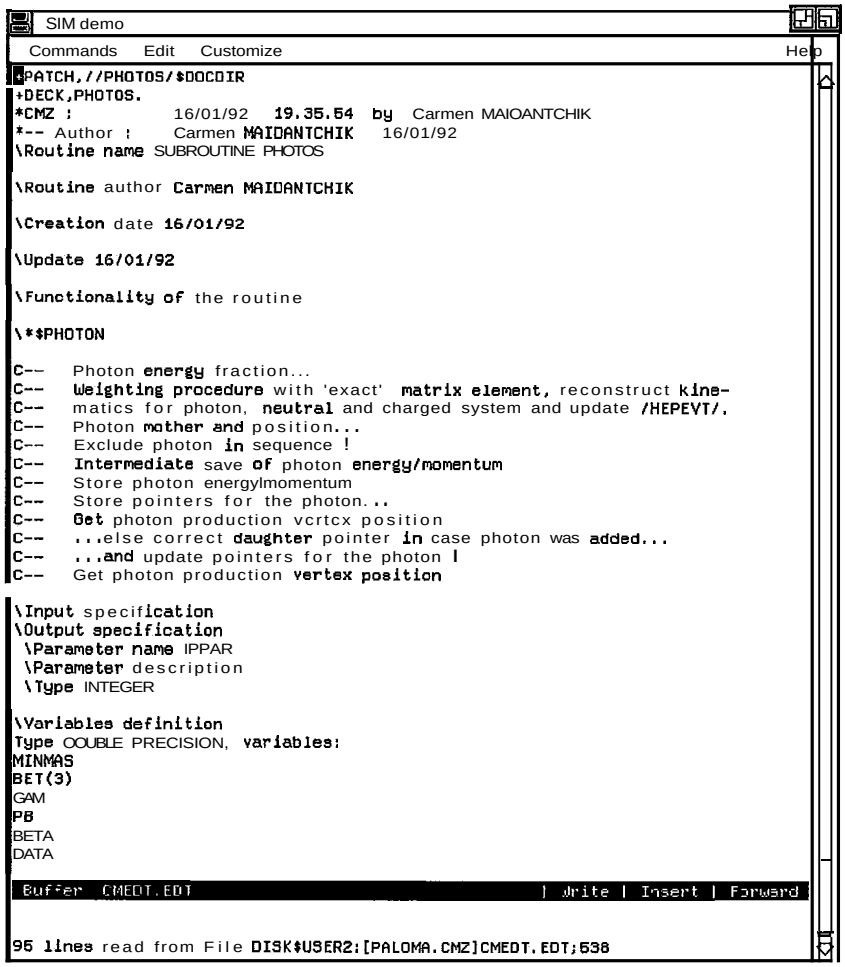

Figura A.12: Documentação gerada a partir das modificações no $ROU_GUI

A parte introdutória do manual pode ser elaborada com base no roteiro $MAN_GUI. A partir desse *deck*, o usuário pode completar com as informações referentes ao seu programa (figura **A.13):**

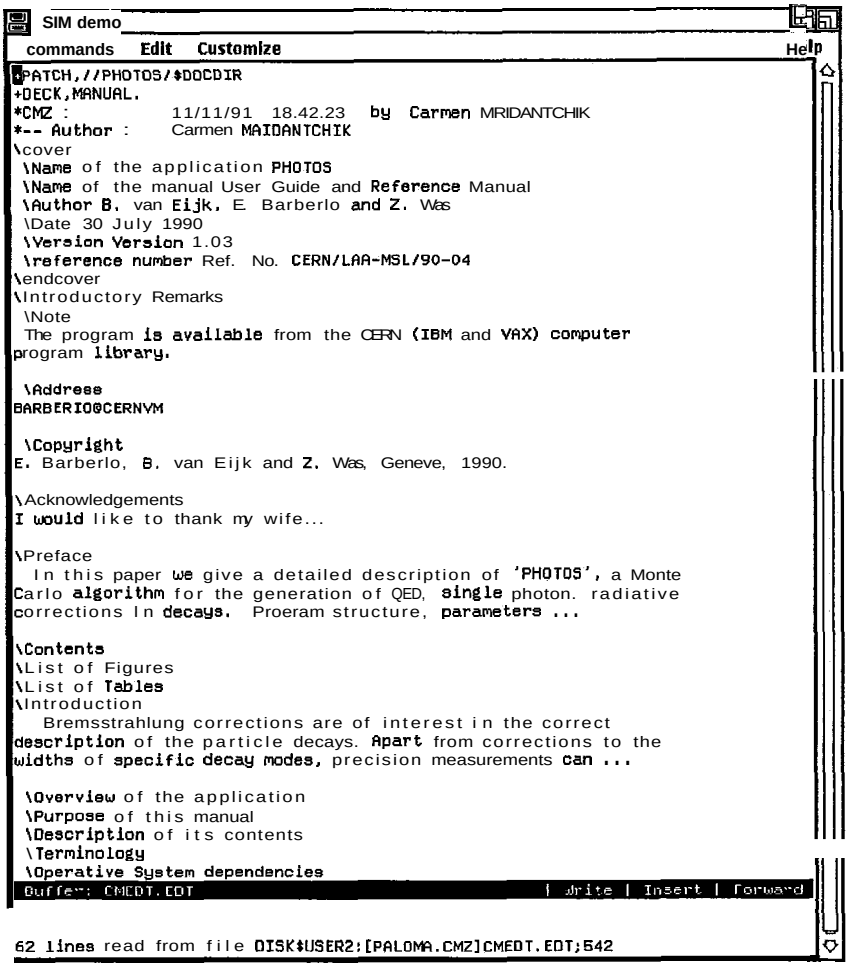

Figura **A.13:** Exemplo da elaboração da parte introdutória de um manual

Após documentar, com base no roteiro **$ROU_GUI**, as rotinas que devem estar no documento final e elaborar a parte introdutória do manual (descrita no roteiro **$MAN_GUI**), o usuário pode gerar um arquivo em formato LaTeX através do comando *DTOT*. O formato do arquivo gerado segue o estilo descrito no roteiro **$STY_GUI**.

A figura A.14 mostra o conteúdo do diretório de documentação **$DOCDIR** já contendo a documentação das rotinas PHOCIN, PHOENE, PHOFAC, PHOINI e PHOTOS e a parte introdutória do manual descrita no deck MANUAL. O usuário chama o comando *DTOT* com os seguintes parâmetros:

- arquivo a ser gerado em formato LaTeX (PHOTOS.TEX);
- nome do deck, contendo a parte introdutória (MANUAL);
- nome dos decks, contendo a documentação das rotinas (PHOCIN, PHOENE, PHOFAC, PHOINI e PHOTOS).

O comando *DTOT* gera, fora do ambiente **CMZ+SIM**, ou seja, no sistema operacional o arquivo PHOTOS.TEX, que deve ser processado pelo comando LATEX para ser impresso, gerando assim, o documento final.

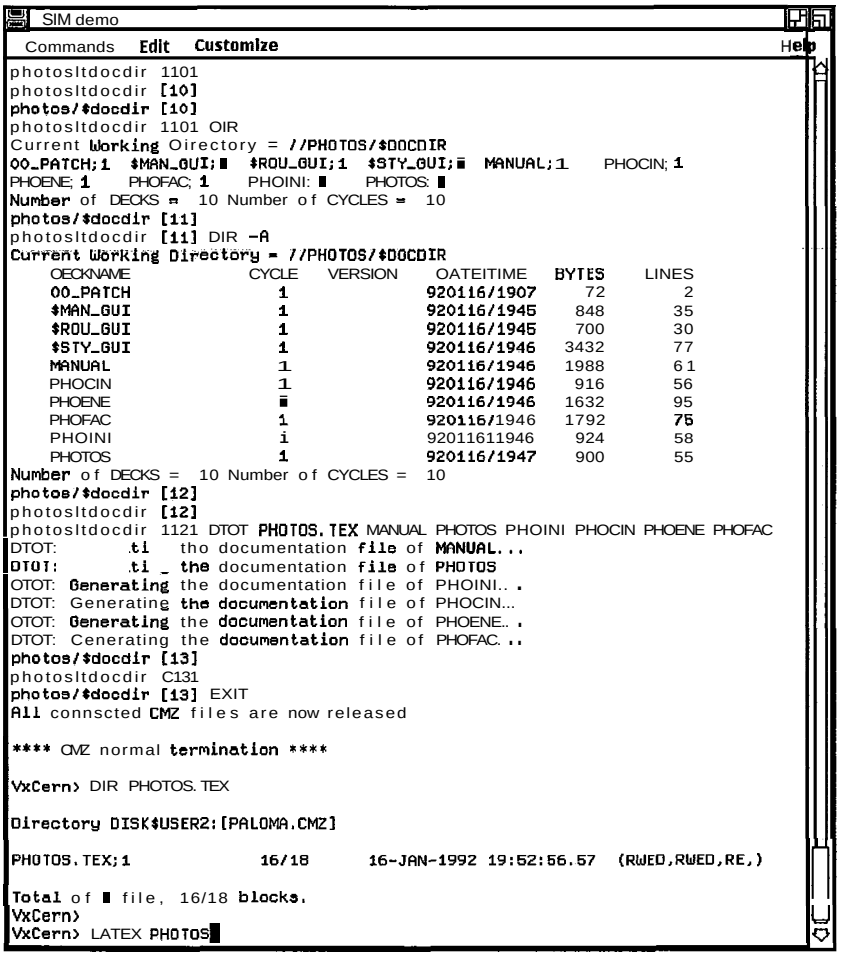

Figura **A.14:** Exemplo da execução do comando DTOT

Um exemplo de um documento gerado pela ferramenta, contendo todos os ítens para documentação de programas, criados automaticamente, e seguindo o formato padrão SIM é apresentado no apêndice B.

O apêndice C corresponde ao manual da aplicação PHOTOS, cuja elaboração foi auxiliada por SIM. Para elaboração desse documento, o usuário modificou os roteiros oferecidos por SIM, para que seu conteúdo e formato sejam adequados ao uso, e inseriu informações adicionais às geradas pela ferramenta.

Apêndice B

Exemplo de um documento "vazio" gerado automaticamente por SIM

O documento apresentado neste apêndice foi gerado automaticamente pela ferramenta SIM, sem envolver a modificação dos roteiros `$MAN_GUI`, `$ROU_GUI` e seguindo o formato oferecido por SIM, descrito no *deck* `$STY_GUI`. Todas as informações presentes foram geradas pela ferramenta e correspondem à documentação de uma parte do programa PHOTOS.

Na folha de rosto, as informações entre colchetes ('[' e ']') podem ser substituídas pelo nome da aplicação, nome do manual (Manual do Usuário, de Referência, etc.), nome do autor, número da versão e de referência. Caso o usuário não especifique a data, SIM gera (através de um comando do LaTeX) a clata atual.

Como SIM foi desenvolvida no CERN, um centro de pesquisas internacional, a documentação produzida pela ferramenta é escrita em inglês. Por fidelidade ao trabalho realizado, o documento é aqui apresentado sem tradução.

[Application name]

[name of the manual]

[author]

December 11, 1991

[version]

[reference number]

Introductory Remarks

Note:

Address:

Acknowledgements

Preface

Contents

List of Figures

List of Tables

1. Introduction

1.1 Overview of the application

1.2 Purpose of this manual

1.3 Description of its contents

1.3.1 Terminology

1.4 Operative System dependencies

1.4.1 Installation

1.5 Revision History

1.6 Basic Concepts

2. Description of the Application

2.1 Functionality of the system

2.2 Architecture of the system

2.3 Initialization

2.4 Execution

3. Description of Each Command

3.1 SUBROUTINE PHOTOS

Routine author: B. van Eijk and Z. Was

Creation date: 26/11/90

Update: 26/03/90

Functionality of the routine:

```
\*$PHOTON
C--   Photon energy fraction...
C--   Weighting procedure with 'exact' matrix element, reconstruct kine-
C--   matics for photon, neutral and charged system and update /HEPEVT/.
C--   Photon mother and position...
C--   Exclude photon in sequence !
C--   Intermediate save of photon energy/momentum
C--   Store photon energy/momentum
C--   Store pointers for the photon...
C--   Get photon production vertex position
C--   ...else correct daughter pointer in case photon was added...
C--   ...and update pointers for the photon !
C--   Get photon production vertex position
```

Modification history: *** no records ***

Input specification:

Output specification:

- **Parameter name:** IPPAR

  . **Parameter description:**

  . **Type:** INTEGER

- **Range and/or set of values:**
- **Default value:**

Variables definition:

Type DOUBLE PRECISION, variables: MINMAS, BET(3), GAM, PB, BETA, DATA.
Type REAL, variables: PCHARG(5), PHOCHA, PHOSPI, PHORAN, PHOCOR, PHOTON(5).
Type INTEGER, variables: IP, I, J, K, L, ME, NCHARG, NEUPOI, NLAST, THEDUM.

Called by:

Calling routines:

CALL PHOOMA, variables: 1, NCHARG, CHAPOI.
CALL PHOOMA, variables: 1, NCHARG, CHAPOI.
CALL PHOKIN, variables: IP, NCHARG.

Calling functions:

Accessed files (I/O): PHLUN

Common blocks:

Common HEPEVT, variables: NEVHEP, NHEP, ISTHEP(NMXHEP), IDHEP(NMXHE

Included libraries:

Error messages:

```
\*$ERR
 9000      FORMAT(1H,A,': Too many charged Particles, NCHARG =',I6)
```

Example:

3.2 SUBROUTINE PHOINI

Routine author: B. van Eijk and Z. Was

Creation date: 26/11/89

Update: 12/04/90

Functionality of the routine:

```
\*$PHOTON
*** no tag statements ***
```

Modification history: *** no records "**

Input specification: *** no parameters ***

Output specification: *** no parameters ***

Variables definition:

Type INTEGER, variables: INIT.

Called by:

Calling routines:

CALL PHOCIN, variables: INIT.

Calling functions:

Accessed files (I/O): *** no accessed files ***

Common blocks:

*** no common blocks ***

Included libraries:

Error messages:

```
\*$ERR
*** no tag statements ***
```

Example:

3.3 SUBROUTINE PHOCIN

Routine author: **B.** van Eijk

Creation date: **12/04/90**

Update: **12/04/90**

Functionality of the routine:

*$PHOTON
*** no tag statements ***

Modification history: *** no records ***

Input specification: *** no parameters ***

Output specification: *** no parameters ***

Variables definition:

Type INTEGER, variables: INIT.

Called by:

Calling routines:
*** no calling routines ***

Calling functions:

Accessed files (I/O): *** no accessed files ***

Common blocks:
*** no common blocks ***

Included libraries:

Error messages:

*$ERR
*** no tag statements ***

Example:

3.4 SUBROUTINE PHOENE

Routine author: B. van Eijk, S. Jadach and Z. Was

Creation date: 01/01/89

Update: 26/03/90

F'unctionality of the routine:

```
\*$PHOTON
C--   No photon... (ie. photon too soft)
C--   Hard  photon... (ie.  photon  hard enough).
C--   Calculate  Altarelli-Parisi Kernel
```

Modification history: *** no records ***

Input specification:

Output specification:

- **Parameter name:** MPASQR
  - **Parameter description:**
  - **Type:** DOUBLE PRECISION
  - **Range and/or set of values:**
  - **Default value:**
- **Parameter name:** MCHREN
  - **Parameter description:**
  - **Type:** DOUBLE PRECISION
  - **Range and/or set of values:**
  - **Default value:**
- **Parameter name:** BETA
  - **Parameter description:**
  - **Type:** DOUBLE PRECISION
  - **Range and/or set of values:**
  - **Default value:**

Variables definition:

Type DOUBLE PRECISION, variables: BIGLOG, DATA.

Type REAL, variables: MPASQR, MCHREN, BIGLOG, BETA, DATA, PRSOFT, PRHARD, PHORAN, PHOFAC.
Type INTEGER, variables: IWT1, IRN, IWT2.

Called by:

Calling routines:

CALL PHOERR, variables: 2, 'PHOENE', DATA.

Calling functions:

Accessed files (I/O): *** no accessed files ***

Common blocks:

*** no common blocks ***

Included libraries:

Error messages:

```
\*$ERR
 9000      FORMAT(1H,A,': Too much Bremmstrahlung required, PRSOFT =',I6)
```

Example:

3.5 FUNCTION PHOFAC

Routine author: B. van Eijk and Z. Was

Creation date: 01/01/89

Update: 13/02/90

Functionality of the routine:

```
\*$PHOTON
*** no tag statements ***
```

Modification history: *** no records ***

Input specification:

Output specification:

- **Parameter name:** MODE

. **Parameter description:**

. **Type:** **INTEGER**

. **Range and/or set of values:**

. **Default value:**

Variables definition:

Type REAL, variables: PHOFAC, FF, PRX.

Called by:

Calling routines:

*** no calling routines ***

Calling functions:

Accessed files (I/O): *** no accessed files ***

Common blocks:

*** no common blocks ***

Included libraries:

Error messages:

```
\*$ERR
*** no tag statements ***
```

Example:

Apêndice C
Exemplo de um manual produzido com o apoio de SIM

O manual apresentado neste apêndice corresponde à documentação de uma parte do programa PHOTOS, cujos processos para produzí-la foram descritos no apêndice A.

PHOTOS: **User Guide and Reference Manual** não contém todos os ítens descritos nos roteiros para documentação de rotinas e para elaboração da parte introdutória do manual, `$ROU_GUI` e `$MAN_GUI` respectivamente, pois esses foram adaptados à aplicação e ao contexto de um manual de *Referência e Guia para o Usuário*. Através da edição da documentação gerada pela ferramenta e do *deck* `$STY_GUI`, o usuário de SIM acrescentou novas informações e modificou o formato do documento.

O documento PHOTOS foi produzido no CERN e serviu para verificar a aplicabilidade da ferramenta SIM. Esse manual foi apresentado aos membros do grupo com o objetivo de buscar sugestões e críticas ao trabalho. Portanto, o documento a seguir está escrito em inglês e assim como no apêndice anterior, é apresentado sem tradução, por fidelidade ao trabalho realizado.

PHOTOS

User Guide and Reference Manual

E. Barberio, B. van Eijk, and Z. Was

30 July 1990

Version 1.03

Ref. No. CERN/LAA-MSL/90-04

Introductory Remarks

Note:

PHOTOS consists of a universal Monte Carlo algorithm for the simulation of QED radiative corrections in decays. The package is written in standard Fortran and has been tested on Apollo, VAX, IBM and Cray. The organisation of the package conforms the CERN program library standards, CERN specific items can be found in the patch: P=*PHOTOS, D=CERN. I/O is performed through the standard high energy physics common: /HEPEVT/, using the PDG particle coding conventions.

Contents

List of Tables

1. Introduction

Bremsstrahlung corrections are of interest in the correct description of the dynamics of particle decays. Apart from corrections to the partia1 widths of specific decay modes, precision measurements can strongly be influenced by QED corrections. A clear example is for instance the measurement of the Michel parameter in tau lepton decays. End-point spectrum measurements in semileptonic bottom/charm hadron decays are effected by photon emission and consequently influence the rate determination of charmless/strangeless B- and C- hadron decays. Other fields of interest may be electromagnetic isolation studies of leptons and/or charged pions, while trigger rates may be affected as well.

We propose a relatively simple approach to a general treatment of QED order (alpha) corrections in decays where at least one charged particle is produced. For details on the description of the algorithm as included in the PHOTOS package, we would like to refer to ref. [1]. In this paper, comparisons with some exact calculations and the implementation of exact O(alpha) matrix elements are discussed as well. In addition, the limits on the applicability of the program are described.

This short user and reference guide is organized as follows. In chapter 2 we describe the procedure of installing the PHOTOS package on (in principle) any type of machine which is supporting Fortran-77. Chapter 3 contains a complete list of commons and parameter settings. In chapter 4 we discuss the initialisation and main calling entries of the program. Some global examples of program application are illustrated as well. Chapter 5 contains the complete list of subroutines and functions as included in the present package. In a short write-up, the main functionalities and I/O parameters are explained. A summary on run time errors and possible warning messages can be found in chapter 6. The guide concludes with a list of references. For a complete history of program updates and program modifications we refer to the patch 'HISTORY'.

1.1 Purpose of this manual

Give the user a hand in getting started.

1.2 Installation

The program may be installed as a standalone library on any **32** (or more) bit machine. The Patchy version contains the following machine selection flags:

```
IBM, VAX, MACMPW, SGI, DECS, APOLLO, IBMRT, CONVEX, SUN,
GOULD, CDC and CRAYI
```

as supported by the CERN program library. Special request for additional flags may be addressed to either one of the authors or the CERN program library. An example of a cradle as used for the creation of a library on the VAX is given below:

```
+EXE.
+USE, VAX.                              Machine selection
+USE, *PHOTOS.                          Select PHOTOS patches
+PAM, LUN=11, T=ATTACH. PHOTOS.PAM      PHOTOS Pam file
+QUIT.
```

The sequence of Patchy cards for executing a user program which either calls the PHOTOS package internally or applies PHOTOS after full events have been generated may look as follows:

```
+EXE.
+USE, VAX.                              Machine selection
+PAM, LUN=11, T=ATTACH. PHOTOS.PAM      PHOTOS Pam file
+USE, PHOCDE.                           Use PHOTOS common definitions
+USE, ...                               Other patches
+USE, USER_PATCH.                       Main steering routine
+PATCH, USER_PATCH.
      SUBROUTINE USER_SUBROUTINE.
...
...
C--
C--   PHOTOS initialisation...
      CALL PHOINI
...
...
C--
C--   Enter event loop...
...
...
C--
C--   Add QED corrections in decays...
      CALL PHOTOS(IPPAR)
```

```
...
...
      RETURN
      END
```

No 'external' routines from other libraries are needed in order to execute PHOTOS.

1.3 Cornrnon definitions

Pending the discussion whether CERN will adopt CMZ as the new code management system, PHOTOS will be supported in Patchy format. Therefore, common definitions are given in a special patch, 'PHOCDE'. If desired, this patch may easily be converted into a Fortran include file. The (KEEP) definitions are given below. We can distinguish two classes. One containing parameters the user may want to modify, the other containing transfer addresses, which are only for interna1 use. Default values are given in square brackets.

PHOTOS version number and release date:

```
+KEEP,PHOVER.
      INTEGER PHOVN1,PHOVN2
      COMMON/PHOVER/PHOVN1,PHOVN2
```

```
Logical unit for printing warnings/error messages and program info:
+KEEP,PHOLUN.
      INTEGER PHLUN
      COMMON/PHOLUN/PHLUN [6]
```

```
+KEEP,PHPICO.
      REAL PI,TWOPI
      COMMON/PHPICO/PI,TWOPI [3.14..., 6.28...]
```

Electromagnetic coupling constant:

```
+KEEP,PHOCOP.
      REAL ALPHA
      COMMON/PHOCOP/ALPHA [0.00729735039]
```

Seeds for Marsaglia and Zaman random number generator:

```
+KEEP,PHSEED.
      INTEGER ISEED,I97,J97
      REAL URAN,CRAN,CDRAN,CMRAN
      COMMON/PHSEED/ISEED(2),I97,J97,URAN(97),CRAN,CDRAN,CMRAN
      SEED(1), [1802]
      SEED(2) , [9373]
```

Photon properties:

```
+KEEP,PHOPHO,IF=DOUBLE.
      DOUBLE PRECISION COSTHG,SINTHG
      REAL XPHCUT,XPHMAX,XPHOTO
      COMMON/PHOPHD/COSTHG,SINTHG
      COMMON/PHOPHS/XPHCUT,XPHMAX,XPHOTO
+KEEP,PHOPHO,IF=SINGLE.
      REAL COSTHG,SINTHG,XPHCUT,XPHMAX,XPHOTO
      COMMON/PHOPHD/COSTHG,SINTHG
      COMMON/PHOPHS/XPHCUT,XPHMAX,XPHOTO
      XPHCUT, minimum photon energy fraction [0.001]
```

Standard HEP common definitions [3]:

```
+KEEP,HEPDIM.
      INTEGER NMXHEP
      PARAMETER (NMXHEP=2000)
+KEEP,HEPEVT.
      INTEGER IDHEP,ISTHEP,JDAHEP,JMOHEP,NEVHEP,NHEP
      REAL PHEP,VHEP
      COMMON/HEPEVT/NEVHEP,NHEP,ISTHEP(NMXHEP) ,IDHEP(NMXHEP),
     &JMOHEP(2, NMXHEP),JDAHEP(2, NMXHEP),PHEP(5, NMXHEP),
     &VHEP(4,NMXHEP)
```

Internal variables, PDMMAX is the maximum charged particle stack length:

```
+KEEP,PHOMUL.
      INTEGER PDMMAX
      PARAMETER (PDMMAX=100)
+KEEP,PHOMOM,IF=DOUBLE.
      INTEGER CHAPOI
```

```
      DOUBLE PRECISION MCHSQR,MNESQR
      REAL PNEUTR
      COMMON/PHOMOM/MCHSQR,MNESQR,PNEUTR(5),CHAPOI(PDMMAX)
+KEEP,PHOMOM,IF=SINGLE.
      INTEGER CHAPOI
      REAL MCHSQR,MNESQR,PNEUTR
      COMMON/PHOMOM/MCHSQR,MNESQR,PNEUTR(5),CHAPOI(PDMMAX)
+KEEP,PHOPRO.
      INTEGER IREP
      REAL PROBH,CORWT,XF
      COMMON/PHOPRO/IREP,PROBH,CORWT,XF
```

Data for efficient determination of particle charge and spin:

```
+KEEP,PHOPCH.
C--
C--   Array 'CHARGE' contains the charge of the first 101 particles
C--   according to the PDG particle code...
C--  (0  is added for convenience)
      REAL CHARGE(0:100)
      DATA CHARGE/ 0.,-0.3333333333,  0.6666666667, -0.3333333333,
     & 0.6666666667,-0.3333333333, 0.6666666667, -0.3333333333,
     & 0.6666666667,2*0., -1., 0., -i., 0., -i., 0., -1., 6*0., i.,
     & 12*0., i., 63*0./
+KEEP,PHOPSP.
C--
C--   Array 'SPIN' contains the spin of the first 100 particles
C--   according to the PDG particle code...
      REAL SPIN(100)
      DATA SPIN/ 8*.5, i., 0., 8*.5, 2*0., 4*1., 76*0./
```

1.4 Initialization

Before the first event is treated, the routine PHOINI has to be called. This package of routines takes care of pre-setting the parameters used in the program and prints some program version info. A block data routine has been avoided by inclusion of the routine 'PHOCIN', which is called from PHOINI. In order to modify default parameter settings, the user may want to execute PHOCIN before PHOINI is called. When this has been the case, a switch prevents a second execution of PHOCIN at the time PHOINI is called. Thus all parameters the user may want to modify are specified in EUDCIN and may be modified by overwriting the values in the appropriate common after execution of PHOCIN !

1.4.1 Steering routine

PHOTOS has one main entry. Execution of PHOTOS is invoked by the call:

```
CALL PHOTOS(IPPAR)
```

where IPPAR is the pointer to a parton/particle at position IPPAR in the common /HEPEVT/. Whether or not this particle has given energy and momenta in the lab. or local rest frame, the complete chain of daughters is scanned for charged particles. So all daughter pointers are checked until the final link is found where all decay particles are stable. After deciding whether a photon has to be emitted, the tree will be followed step by step, allowing for photon emis sion at each branching of the tree. In the present version only one photon may be emitted at a branching. After adding the photons, the complete /HEPEVT/ is re-arranged in order to achieve consistency in mother and daughter pointers. Clearly, the algorithm will be much more efficient when the routine PHOTOS is called when the calling program is simulating the decay of a particle (and preferably in the rest frame of the decaying object). Unnecessary to stress the need for a correct filling of /HEPEVT/, e.g. complete information on pointers etc..

1.5 Execution

In the following example we show the global structure of a program in which photon emission is simulated, just after the decay of a single particle at position IPPAR in /HEPEVT/. Daughter particles are located at JDAHEP(1,IPPAR) $\longrightarrow$ JDAHEP(2,IPPAR) (= NHEP in this case).

```
      PROGRAM MAIN
C--
C--   User initialisation, initialisation of event generator
      ...
C--
C--   Initialise PHOTOS
      CALL PHOINI
      ...
C--
C--   Enter event loop
      ...
C--
C--   Simulate hard process
      ...
C--
C--   Perform parton evolution
      ...
C--
```

```
C--   Hadronisation
      ...
C--
C--   Decay unstable particles...
      DO IPPAR=1, NHEP
         IF (PARTICLE .NE. STABLE) THEN
         CALL PARTICLE_DECAY
C--
C--   Add QED corrections
         CALL PHOTOS(IPPAR)
        ENDIF
      ENDDO
C--   End of event loop ? ---> stop
```

Similarly, a complete event may be generated after which PHOTOS scans for possibilities to add QED corrections:

```
      PROGRAM MAIN
C--
C--   User initialisation, initialisation of event generator
      ...
C--
C--   Initialise PHOTOS
      CALL PHOINI
      ...
C--
C--   Enter event loop
      ...
C--
C--   Simulate hard process
      ...
C--
C--   Perform parton evolution
      ...
C--
C--   Hadronisation
      ...
C--
C--   Decay unstable particles...
      ...
C--
C--   Add QED corrections, depending on which informationthe event
C--   generator is writing in the first records, IPPAR will in
C--   general be =1.
```

```
      IPPAR=I
      CALL PHOTOS(IPPAR)
C--
C--   End of event loop ? ---> stop
```

2. Description of Each Command

2.1 SUBROUTINE PHOTOS

Routine author: B. van Eijk and Z. Was

Creation date: 26/11/90

Update: 26/03/90

Functionality of the routine: Order(alpha) radiative corrections are calculated in the decay of the IPPAR- th particle in the standard HEP common /HEPEVT/. The photon is 'emitted' from one of the charged daughters of the decaying particle (at positions JDAHEP(1, IPPAR) $\longrightarrow$ JDAHEP(2,IPPAR) in /HEPEVT/. The kinematics of both parent and daughters should preferably be defined in the rest frame of the parent. If not, boosts, rotations etc. will considerably slow down the program. Cascade decays will be treated until the end of the cascade, photons may be added at each stage of the decay chain.

Working with PHOTON:

```
C--   Photon energy fraction...
C--   Weighting procedure with 'exact' matrix element,
C--   reconstruct kinematics for photon, neutral and charged
C--   system and update /HEPEVT/.
C--   Photon mother and position...
C--   Exclude photon in sequence !
C--   Intermediate save of photon energy/momentum
C--   Store photon energy/momentum
C--   Store pointers for the photon...
C--   Get photon production vertex position
C--   ...else correct daughter pointer in case photon was added...
C--   ...and update pointers for the photon !
C--   Get photon production vertex position
```

Input specification:

- **Parameter name:** IPPAR

. **Parameter description:** pointer to decaying particle and common /HEP-EVT/

. **Type:** INTEGER

Output specification:

- **Parameter name:** COMMON /HEPEVT/

. **Parameter description:** either with or without (a) photon(s) added

Variables definition:

Type DOUBLE PRECISION, variables: MINMAS, BET(3), GAM, PB, BETA, DATA.

Type REAL, variables: PCHARG(5), PHOCHA, PHOSPI, PHORAN, PHO-COR, PHOTON(5).

Type INTEGER, variables: IP, I, J, K, L, ME, NCHARG, NEUPOI, NLAST, THEDUM.

Calling routines:

CALL PHOOMA, variables: 1, NCHARG, CHAPOI.

CALL PHOKIN, variables: IP, NCHARG.

Common blocks:

Common HEPEVT, variables: NEVHEP, NHEP, ISTHEP(NMXHEP), IDHEP(NMXHEP).

Error messages:

#1, Too many charged Particles, NCHARG = *

Example: `CALL PHOTOS(IPPAR)`

2.2 SUBROUTINE PHOINI

Routine author: B. van Eijk and Z. Was

Creation date: 26/11/89

Update: 12/04/90

Functionality of the routine: Initialization steering routine for the PHOTOS QED radiation package.

Input specification: *** no parameters ***

Output specification: *** no parameters ***

Variables definition:

Type INTEGER, variables: INIT.

Calling routines:

CALL PHOCIN, variables: INIT.

Common blocks: *** no common blocks ***

Example: `CALL PHOINI`

2.3 SUBROUTINE PHOCIN

Routine author: B. van Eijk

Creation date: 12/04/90

Update: 12/04/90

Functionality of the routine: Routine for pre-setting parameters in commons.

Input specification: *** no parameters ***

Output specification:

- **Parameter name:** COMMONS /PHOLUN/, /PHOPHO/, /PHOCOP/, /PHPICO/ and /PHSEED/.

Variables definition:

Type INTEGER, variables: INIT.

Calling routines: *** no calling routines ***

Common blocks: *** no common blocks ***

Example: CALL PHOCIN

2.4 SUBROUTINE PHOENE

Routine author: B. van Eijk, S. Jadach and Z. Was

Creation date: 01/01/89

Update: 26/03/90

Functionality of the routine: Generate photon energy fraction (in (pasent mass)/2 units) for the decay bremmstrahlung.

Working with PHOTON:

```
C--   No photon... (ie. photon too soft)
C--   Hard photon... (ie. photon hard enough).
C--   Calculate Altarelli-Parisi Kernel
```

Input specification:

- **Parameter name:** MPASQR

. **Parameter description:** mass of decaying system squared.

Type: DOUBLE PRECISION

Output specification:

- **Parameter name:** MCHREN
  - **Parameter description:** renormalised mass squared
  - **Type:** DOUBLE PRECISION
- **Parameter name:** BETA
  - **Parameter description:** beta factor due to renormalisation
  - **Type:** DOUBLE PRECISION

Variables definition:

Type DOUBLE PRECISION, variables: BIGLOG, DATA.

Type REAL, variables: MPASQR, MCHREN, BIGLOG, BETA, DATA, PRSOFT, PRHARD, PHORAN, PHOFAC.

Type INTEGER, variables: IWT1, IRN, IWT2.

Calling routines:

CALL PHOERR, variables: 2, 'PHOENE', DATA.

Common blocks: *** no common blocks ***

Error messages:

#2, Too much Bremmstrahlung required, PRSOFT = *

Example: **CALL PHOENE(MPASQR ,MCHREN,BETA)**

2.5 FUNCTION PHOFAC

Routine author: B. van Eijk and Z. Was

Creation date: 01/01/89

Update: 13/02/90

Functionality of the routine: This is the control function for the photon spectrum and final weighting. It is called from PHOENE for generating the raw photon energy spectrum (MODE=0) and in PHOCOR to scale the final weight (MODE=1). The factor consists of 3 terms. Addition of the factor FF which multiplies PHOFAC for MODE=0 and divides PHOFAC for MODE=1, does not affect the results for the MC generation. An appropriate choice for FF can speed up the calculation. Note that a too small value of FF may cause weight overflow in PHOCOR and will generate a warning, halting the execution. PRX should be included for repeated calls for the same event, allowing more particles to radiate photons. At the first call IREP=0, for more than 1 charged decay products, IREP ¿= 1. Thus, PRSOFT (no photon radiation probability in the previous calls) appropriately scales strength of the bremsstrahlung.

Input specification:

- **Parameter name:** MODE

. **Parameter description:** control parameter for the photon energy spectrum and final weighting.

. **Type:** INTEGER

Output specification: Function value

Variables definition:

Type REAL, variables: PHOFAC, FF, PRX.

Calling routines: *** no calling routines ***

Common blocks: *** no common blocks ***

Example: `PHOFAC(MODE)`

3. Run time errors

In the following table, warnings and/or fatal error messages are summarised that may occur during the execution of PHOTOS.

Mess. nr.	Text (including routine name: *)	Value	Status
1	*: Too many particles, NCHARG =	ncharg	fatal
2	*: Combined Weight is exceeding 1, Weight =	prsoft	fatal
3	*: Combined Weight is exceeding 1, Weight =	weight	fatal
4	*: Error in rescaling charged and neutral Vectors		fatal
5	*: Non matching charged Particle Pointer, NCHARG =	ncharg	
6	*: Do you really work with a Particle of Spin:	spin	warning
7	*: Stack Length exceeded, NSTACK =	nstack	
8	*: Random Number Generator Seed(1) out of Range:	seed	fatal
9	*: Random Number Generator Seed(2) out of Range:	seed	fatal
#	Funny Error Message: # ! What to do ?		warning
	Fatal Error Message, I stop this Run !		fatal
-	10 Error Messages generated, I stop this Run !		fatal

Table 3.1: Run time erros

Except for error messages 1, 6, 7, 8 and 9, previous error reports will not occur. Replacing the default seeds for the random number generator by one or two values that are out of range forces the program to stop (#8 and #9). Message nr. 6, informs the user that the particle concerned will be treated as if it had a lower spin (see user and reference guide for the present version on spin treatment). Errors 1 and 7 indicate that a local array has a too small dimension. This may occur when a multi body decay generated by the decay generator exceeds 100 (default) decay particles or as in a cascade decay, the charged particle stack exceeds 100 (default) particles. The user may want to increase these dimensions.

4. References

[1] E. Barberio, B. van Eijk and Z. Was, 'PHOTOS-a universal Monte Carlo Algorithm for Q.E.D. radiative Corrections in Decays', CERN report, CERN-TH-5857/90 (1990), submitted to Comp. Phys. Commun.

[2] M. Aguilar-Benitez et al. (Particle Data Group), 'Review of Particle Properties', Phys. Lett. B204 (1988) 113.

[3] B. Bambah et al., 'QCD Generators at LEP', contribution to the workshop on LEP physics, CERN Yellow Report 89-08 Volume 3 (325) and CERN Report CERN- TH.5466/89.

[4] F.James CERN DD-Report November 1988.

[5] G.Marsaglia and A.Zaman, FSU-SCR-87-50.

[6] H.U. Bengtsson and T. Sjostrand, Comp. Phys. Comm. 46 (1987) 43.