REUTILIZAÇÃO DE SOFTWARE NO DESENVOLVIMENTO DE SOFTWARE CIENTÍFICO

Cláudia Maria Lima Werner

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 DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Jano Moreira de Souza, Ph.D.
(Presidente)

Profa. Ana Regina C. da Rocha, D.Sc.

Prof. Zieli Dutra Thomé Filho, D.Sc.

Prof. Mário João Martins Pimenta, D.Sc.

Profa. Regina Célia de Souza Pereira, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 1992

WERNER, CLÁUDIA MARIA LIMA

Reutilização de Software no Desenvolvimento de Software Científico (Rio de Janeiro) 1992.

xii, 179 p. 29,7 cm (COPPE/UFRJ, D.Sc. Engenharia de Sistemas e Computação, 1992)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. reutilização de software 2. desenvolvimento de software científico 3. ferramentas de suporte 4. linguagem de quarta-geração

5. biblioteca de componentes

I. COPPE/UFRJ II. Título (série)

Ao Alexandre.

Agradecimentos

Ao Jano, pelas sugestões, críticas e avaliação do trabalho, permitindo que esta tese se tornasse uma realidade.

À Ana Regina, pelo incentivo dado durante todo o período de doutoramento, principalmente, no período final, fazendo críticas e sugerindo soluções quanto à estrutura da tese.

Ao Zieli, pelo carinho e atenção dada durante estes anos difíceis e pelo esforço constante na formação de um grupo forte para atuação no Projeto de Colaboração UFRJ/CERN.

Ao Mário Pimenta e João Varela, pelo apoio e incentivo durante minha permanência no CERN.

À Profa. Regina Célia por suas observações e comentários atenciosos sobre o trabalho desenvolvido.

Ao Prof. Maculan pelo apoio ao Projeto de Colaboração UFRJ/CERN.

Ao Prof. Tsichritzis por ter permitido minha participação no grupo de pesquisa do Centro Universitário de Informática da Universidade de Genebra.

A todos que trabalharam comigo no quarto nível de gatilho do Projeto DELPHI, em especial ao Philipe Gavillet que sempre incentivou meu trabalho.

A todos que trabalhei no Projeto ITHACA, principalmente, à Vicki de Mey, Oscar Nierstrasz, Jan Vitek e Betty Junod, pela amizade e incentivo.

Ao Heitor e Nelson Ebecken por me darem a primeira oportunidade de pesquisa na área científica.

Ao Flavio, por sua dedicação e inúmeras horas de trabalho na implementação do primeiro protótipo do Ambiente CAOS.

Aos companheiros de luta, em especial, Marta, Luis Carlos, Trotta, Teresa e Guilherme, pela força durante este período de eternas dúvidas.

Aos meus pais e irmãos, por terem sempre acreditado no meu trabalho e na capacidade para vencer este desafio.

Ao CNPq e World Laboratory pelo apoio financeiro.

E ao Alexandre, sem cujo apoio, incentivo, amor e carinho eu jamais poderia ter concluído esta etapa de minha vida.

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

REUTILIZAÇÃO DE SOFTWARE NO DESENVOLVIMENTO DE SOFTWARE CIENTÍFICO

Cláudia Maria Lima Werner

MARÇO, 1992

Orientador: Jano Moreira de Souza

Programa: Engenharia de Sistemas e Computação

O termo *reutilização de software* caracteriza-se pela utilização de produtos de software desenvolvidos ao longo do ciclo de vida em uma situação diferente daquela para a qual foram, originalmente, produzidos. Neste sentido, considera-se não apenas a reutilização de produtos executáveis (i.e. módulos, rotinas, fragmentos de código, etc) mas, também, produtos não executáveis (i.e. definição de requisitos, especificação, projeto, etc).

Nesta tese discutem-se duas estratégias para reutilização, propondo-se uma ferramenta de suporte para cada estratégia, e estabelecem-se as potencialidades, limites e adequação quanto ao uso das mesmas no contexto do desenvolvimento de software científico.

Apesar do intenso uso de computadores no desenvolvimento de pesquisas científicas, ainda hoje, é possível encontrar uma série de dificuldades relacionadas ao desenvolvimento de software científico. A partir da nossa experiência em projetos de pesquisa científica, propomos a adoção de uma estratégia de reutilização no desenvolvimento de software científico, a fim de minimizar estas dificuldades.

A proposta de adoção de uma estratégia de reutilização no desenvolvimento de software científico tem como principal objetivo livrar ao máximo profissionais dos diversos ramos da ciência de atividades puramente de programação, para que estes possam se dedicar mais intensamente às atividades de sua especialidade.

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

SOFTWARE REUSABILITY IN THE DEVELOPMENT OF SCIENTIFIC SOFTWARE

Cláudia Maria Lima Werner

MARCH, 1992

Thesis Supervisor: Jano Moreira de Souza

Department: Systems and Computer Engineering

Software reusability is characterized by the use of software products developed during the software life cycle in a situation different from that they were originally made for. In that sense, we consider not only the reusability of executable products (i.e. modules, routines, code fragments, etc) but also non-executable products (i.e. requirements, specifications, design, etc).

This thesis discusses two reusability approaches, having proposed a supporting tool for each approach, and establishes their potentialities, limits and adequability of use in the context of scientific software development.

Although the use of computers is intense in the development of scientific research, it is still possible to find many problems related to scientific software

development. From our experience in working in scientific research projects, we propose the adoption of a reuse strategy in the development of scientific software with the goal of minimizing some of these problems.

The main objective of this proposal is to free as much as possible professionals of various scientific domains from pure programming activities. In this way, they can dedicate themselves to activities of their own application domain.

ÍNDICE

I. Introdução

Objetivo

Esta tese tem como objetivo discutir duas estratégias para reutilização de software, propondo uma ferramenta para cada estratégia, e estabelecer as potencialidades, limites e adequação quanto ao uso das mesmas no contexto do desenvolvimento de software científico.

Para isto, fazemos inicialmente um estudo sobre o desenvolvimento de software científico e suas atuais dificuldades. Após um estudo sobre o atual estado da arte em reutilização, divide-se as tecnologias existentes em tecnologias de geração e tecnologias de composição.

Nesta tese, apresentamos duas experiências práticas no desenvolvimento de ferramentas de suporte à reutilização no contexto científico que utilizam cada um desses tipos de tecnologias.

A adoção de uma estratégia de reutilização no desenvolvimento de software científico tem como principal objetivo livrar ao máximo profissionais dos diversos ramos da ciência de atividades puramente de programação, para que estes possam se dedicar mais intensamente às atividades de sua especialidade.

Motivação

Foi no meio científico que a utilização de computadores se deu de forma imediata e crescente desde o aparecimento dos primeiros equipamentos. Estes equipamentos requeriam conhecimentos bastante específicos e complexos, não sendo facilmente operados por usuários sem um boa formação técnica.

Com o passar do tempo, os computadores foram mostrando seu potencial de processamento e armazenamento, lidando não somente com números, mas com textos e gráficos. Seu modo de operação foi sendo simplificado. Com o aparecimento de computadores pessoais e periféricos sofisticados, tais como vídeo colorido, mouse, "plotters", "scanners", "joysticks", impressoras a laser, entre outros, pode-se dizer que o computador passou a ser uma ferramenta de uso comum na maioria dos ambientes profissionais e de lazer e, portanto, parte de nosso dia a dia.

Apesar da história dos computadores poder ser contada a partir das primeiras experiências no meio científico, após tantos anos, surpreende-nos encontrar diversas referências a problemas no desenvolvimento de software científico, tal como a falta ou inadequação de métodos, técnicas e ferramentas neste domínio de aplicação [MEYE84] [TRAU84] [BROW85] [CROSS86] [MEDE87] [CERN89d] [LOKE91].

A partir da nossa participação em projetos de pesquisa científica, podemos atestar as dificuldades encontradas no desenvolvimento de software científico. Tais dificuldades vão desde questões relacionadas às diferentes especializações necessárias ao sucesso de um projeto deste gênero, até questões relacionadas à complexidade e características específicas do domínio da aplicação.

Ao identificarmos as dificuldades, atualmente, encontradas no processo de desenvolvimento de software científico propomos, como uma possível estratégia para minimizá-los, o desenvolvimento de software baseado na reutilização de produtos previamente construídos. Neste cenário, o desenvolvedor de uma nova aplicação poderá contar com o apoio de ferramentas especializadas, durante o processo de desenvolvimento, que lhe permitirão a consulta a decisões tomadas em projetos similares e o uso de produtos de software devidamente testados, documentados e construídos segundo normas de Engenharia de Software. O uso de ferramentas de suporte especializadas permitirá ao desenvolvedor preocupar-se com questões mais específicas de sua área de aplicação, ao invés de ter que lidar com questões puramente operacionais.

As linhas de pesquisa em Banco de Dados e Engenharia de Software do Programa de Engenharia de Sistemas da COPPE têm estado diretamente envolvidas em atividades de pesquisa em computação para aplicações científicas desde 1987. Nesta época, foi definida uma metodologia para desenvolvimento de software científico para um grupo de desenvolvimento do Centro de Pesquisa da PETROBRÁS (CENPES) [ROCH88]. Em 1988, foi iniciado o projeto de pesquisa da COPPE/UFRJ com o CERN (Laboratório Europeu de Altas Energias, Genebra), em colaboração com o LIP (Laboratório de Instrumentação e Física Experimental de Lisboa). Os primeiros trabalhos do grupo, no contexto deste projeto, se propunham ao estudo de linguagens dedicadas [WERN89a] [WERN90a], banco de dados para aplicações científicas [PALE90a] [PALE89] e controle da qualidade de software científico [PALE90b] [PALE90c]. Atualmente, temos pesquisas relacionadas a sistemas de apoio à documentação [MAID91] [MAID92], reutilização de software científico [WERN91a] [WERN91f] [XEXE91] e construção de um ambiente de desenvolvimento de software para física de altas energias [SOUZ90] [SOUZ91].

Esta tese foi, parcialmente, desenvolvida durante a permanência no CERN e no CUI (Centro Universitário de Informática, Universidade de Genebra), em regime de doutorado "sanduíche", tendo sido finalizada na COPPE/UFRJ (Programa de Engenharia de Sistemas e Computação).

Durante a permanência no CERN, desenvolvemos uma linguagem de quarta-geração para seleção de eventos em linha, denominada Fado (versão 2.0), no contexto do Projeto do Sistema de Aquisição de Dados da experiência DELPHI [WERN89a] [WERN89b] [WERN90a] [WERN90b] [WERN90c] [WERN90d] [WERN90e] [WERN91e]. Este trabalho foi desenvolvido de julho de 1988 a setembro de 1990, no âmbito do Projeto de Colaboração entre a COPPE/UFRJ, o CERN e o LIP.

No CUI, participamos no período de janeiro a julho de 1990 de reuniões técnicas do grupo envolvido no projeto europeu ESPRIT II, denominado ITHACA ("Integrated Toolkit for Highly Advanced Computer Applications"), para a construção de um ambiente de desenvolvimento de aplicações orientadas a objetos [PROF89] [NIER90] [STAD90] [VITE90]. Esta experiência nos permitiu amadurecer a proposta de desenvolvimento de um ambiente orientado a objetos para a composição de aplicações científicas, denominado CAOS [WERN91a], tendo sido implementado um primeiro protótipo na COPPE/UFRJ [WERN91d] [WERN92].

Organização da Tese

Esta tese está dividida em seis capítulos. O primeiro deles definiu o objetivo do presente trabalho, suas motivações e sua organização.

No segundo capítulo, descrevemos o ambiente científico tal como é descrito na literatura e como o encontramos na prática. Nesse capítulo, procuramos diferenciar

este domínio de aplicação dos demais, caracterizando o software produzido e os problemas específicos de seu desenvolvimento. Apresentamos, ainda, um exemplo concreto de um ambiente científico de desenvolvimento de software (o ambiente científico no CERN). Esse capítulo serve como base para identificar o contexto em que os demais capítulos irão se desenvolver.

No capítulo III, introduzimos a questão da reutilização no desenvolvimento de software em geral. Fazemos uma revisão da literatura sobre os principais aspectos relacionados ao tema. Apresentamos as duas principais abordagens para reutilização, definidas a partir das tecnologias aplicadas: *tecnologias de geração* e *tecnologias de composição*. Nesse capítulo, procuramos mostrar o atual estado da arte (i.e. pesquisas e sistemas desenvolvidos até a presente data), de um modo geral, e o estado da prática de reutilização no ambiente científico.

Os capítulos IV e V descrevem dois diferentes experimentos práticos de reutilização por nós vivenciados no contexto científico. O primeiro deles trata do desenvolvimento de uma linguagem de quarta-geração e seu, respectivo, ambiente de programação para aplicação específica em um sistema de aquisição de dados (abordagem de geração). O segundo provê um ambiente de apoio ao desenvolvimento de aplicações científicas, fornecendo um conjunto de componentes de software organizados em uma biblioteca, permitindo sua manipulação a partir de ferramentas de suporte especializadas (abordagem de composição). Ao descrevermos cada experiência, fazemos comentários gerais sobre o sistema implementado e seu desenvolvimento, seguido de uma análise dos principais aspectos de reutilização encontrados.

Finalmente, concluimos nosso trabalho no capítulo VI, apresentando suas principais contribuições e indicando suas perspectivas de continuidade, a partir da

discussão sobre a adoção de uma estratégia de reutilização no desenvolvimento de software científico.

II. Software Científico

Neste trabalho, utiliza-se o termo *software científico* significando todo aquele software desenvolvido para atender a aplicações científicas e de engenharia, indistintamente.

Antes de apresentar as características do software científico que o distinguam dos demais tipos de software existentes, descreve-se brevemente o cenário no qual ele se insere, ou seja, em um contexto onde o produto final (i.e. o software) faz parte de um complexo processo de avanço da ciência.

Este capítulo está dividido em três partes. A primeira delas introduz aspectos mais amplos sobre a ciência e o ambiente científico. A segunda trata do desenvolvimento de software científico e da caracterização específica deste software. Na terceira parte, apresenta-se, como exemplo típico de um ambiente de desenvolvimento de software científico, o ambiente de desenvolvimento do CERN.

II.1. A Ciência e o Ambiente Científico

O termo *ciência* é, normalmente, usado para descrever tanto o processo de aplicação do método científico quanto o conhecimento por ele gerado [STRE74]. O processo de aplicação do método científico é chamado *pesquisa*. O principal objetivo da pesquisa científica é o avanço do conhecimento.

De uma maneira geral, pode-se listar como principais fases da atividade científica as seguintes [STRE74]:

1) *Coleta de Dados*: a partir de diversas fontes (ex. relatórios, livros, artigos, dados arquivados, novos dados, etc);

2) *Exame*: envolvendo, frequentemente, a procura de certos padrões ou regularidade dos dados, ou algum evento excepcional ou surpreendente;

3) *Formulação do Problema ou Hipótese*: o problema deve ser suficientemente detalhado de forma a possibilitar o direcionamento exato das atividades resultantes. Esta fase deve fornecer subsídios para a elaboração, ou projeto, de soluções ao problema e o desenvolvimento de critérios e meios para avaliar as soluções propostas;

4) *Projeto*: envolve a invenção ou síntese de soluções-tentativas;

5) *Análise e otimização*: as diversas soluções propostas são analisadas, modificadas e refinadas, individualmente, para extrair o máximo de conceitos úteis para a solução do problema;

6) *Avaliação, Experimentação*: as soluções-candidatas são, então, avaliadas sob diversos critérios e testes;

7) *Modelagem, testes extensivos*: quando o processo de avaliação finalmente seleciona a melhor solução, uma fase de modelagem e testes extensivos é executada de forma a determinar o grau de adequação da solução escolhida e prever suas possíveis consequências;

8) *Documentação e disseminação*: a publicação do experimento e dos resultados obtidos ajuda o aumento do conhecimento científico, e

9) *Acompanhamento e responsabilidade contínua*: todos e quaisquer resultados obtidos através de uma atividade científica devem ser, constantemente, averiguados quanto aos possíveis efeitos de sua implementação em produtos tecnológicos no ambiente e no homem.

A *figura 1* mostra, esquematicamente, as fases da atividade científica.

As técnicas e ferramentas utilizadas na execução das atividades de pesquisa podem variar bastante, dependendo do ramo da ciência em que estamos executando o método científico. Com o surgimento de computadores de uso geral, pela primeira vez, se teve uma ferramenta suficientemente versátil para ser usada em todas as fases do processo científico, atendendo aos mais diversos ramos da ciência [STRE74]. Alguns dos tipos de serviço baseados em computador são: sistemas de aquisição de dados, arquivamento e gerência de dados, redução, pré-processamento e análise de dados, arquivamento e recuperação de documentação, controle de experiências, simulações, entre outros. Diversos artigos encontrados na literatura descrevem aplicações de computadores nos mais variados ramos da ciência [BART87] [BOLT70] [BROW85] [NEWM87] [TRAU84].

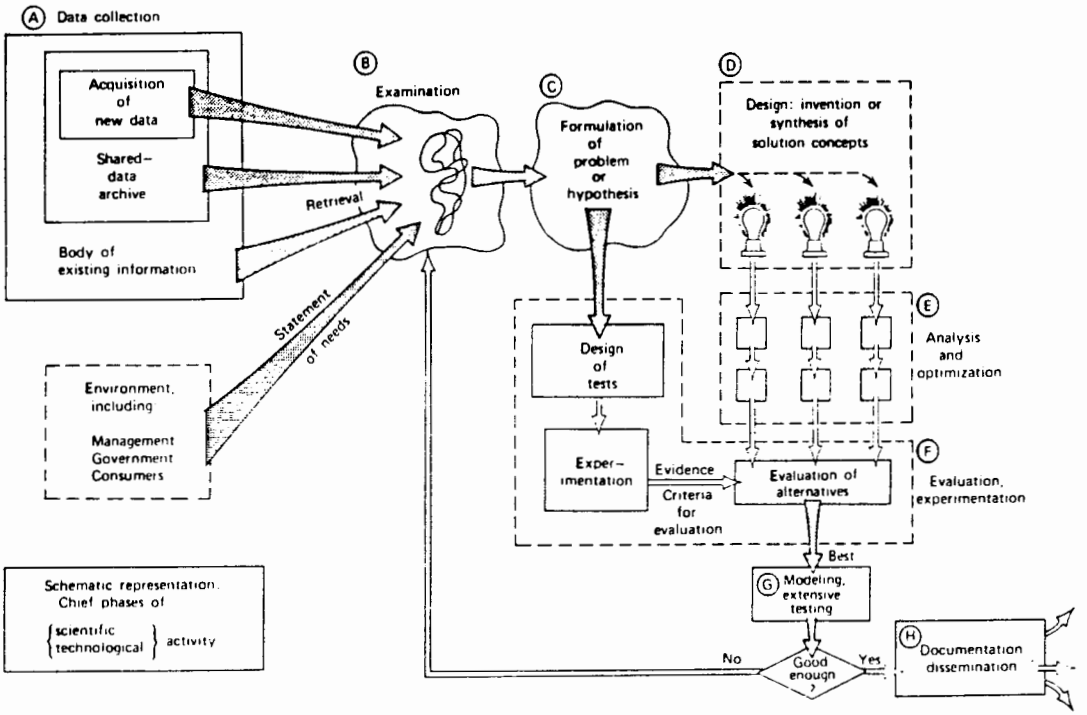


Fig. 1 - Fases da atividade científica [STRE74]

Dado que esta nova ferramenta se insere de forma bastante adequada nas mais variadas atividades científicas, é interessante notar de que forma ela contribui na

determinação de um novo perfil do cientista das áreas de física, matemática, engenharia, etc, que a utiliza.

Rodrigue et al., em [RODR80], define três grandes grupos de profissionais envolvidos no que ele denomina a "Computação Científica em Larga Escala". Estes estariam divididos em profissionais teóricos, experimentais e computacionais, existindo uma forte interação, conforme mostra a *figura 2.*

A ciência teórica desenvolve o modelo matemático, que a ciência computacional resolve numericamente. Os resultados numérico obtidos, por sua vez, podem sugerir novas teorias. Teorias sugerem experimentos, ou interpreta-os. Resultados de experiências ajudam a testar teorias, ou sugerem novas teorias.

A ciência experimental provê dados para a ciência computacional e esta pode modelar processos que são difíceis de serem executados em um laboratório, ou fazer a análise dos dados e controlar aparatos experimentais.

Na realidade, observamos que, em determinados laboratórios de pesquisa científica, existe um quase total isolamento entre esses três grupos.

Os teóricos trabalham em suas teorias, desenvolvem seus próprios programas de simulação, analisam resultados e aperfeiçoam suas teorias, individualmente.

Os profissionais envolvidos na ciência experimental desenvolvem suas experiências de maneira bastante autônoma, contando apenas com recursos básicos oferecidos por um departamento de computação, caso este exista, ou por produtos de software desenvolvidos em outras experiências. A maior parte do software usado em experiências é desenvolvido pelos próprios participantes.

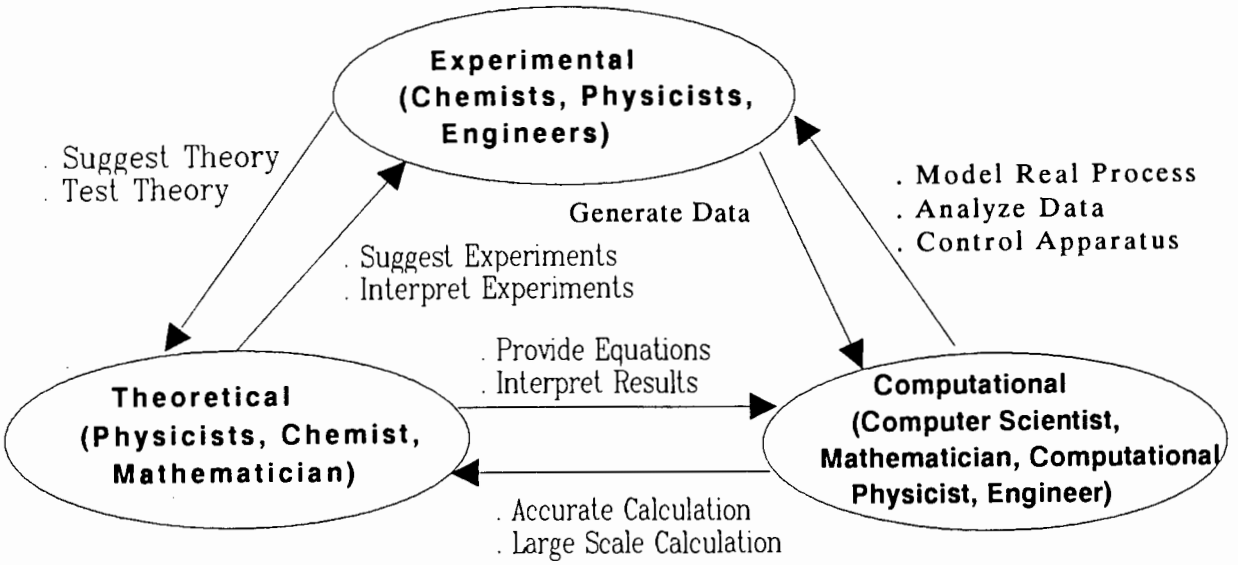


Fig. 2 - Grupos da Computação Científica em Larga Escala [RODR80]

Quanto às atividades ditas computacionais, que de acordo com Rodrigue estão associadas ao fornecimento de melhores códigos, aplicação de novas tecnologias na modelagem numérica e na engenharia de hardware e de software são executadas, na maioria das vezes, por profissionais sem educação formal em computação.

De uma maneira geral, pode-se afirmar que, apesar do computador ser considerado uma ferramenta imprescindível às atividades científicas, a presença de especialistas na Ciência da Computação quase não é verificada neste meio. Mesmo o reconhecimento da necessidade de uma apredizagem específica de computação por parte dos físicos, matemáticos, engenheiros, etc, ainda hoje, é um assunto bastante polêmico e discriminado em ambientes científicos, dado que uma grande parcela destes profissionais considera importante, apenas, sua atividade de pesquisa, não se importando, realmente, com os meios para adquiri-la.

Um outro fator que pode criar complicações no desenvolvimento de software em ambientes científicos diz respeito à questão da formação de equipes de pesquisa. É possível encontrar, atualmente, alguns projetos científicos que são desenvolvidos por equipes cujos elementos estão afiliados a diversos laboratórios de pesquisa, geograficamente distribuídos, trabalhando em um esquema de colaboração (ex. projetos no CERN, ver seção II.3). Isto se deve ao fato de que, cada vez mais, torna-se difícil a criação de grandes projetos executados por um único laboratório. O tamanho e a complexidade dos experimentos cada vez maiores, com custos exorbitantes e, quase sempre, a fundo perdido, além das diferentes qualificações profissionais exigidas, levam a uma tentativa universal em se reunir esforços em prol de uma ciência única, sem limites de fronteira. Com isto, outros problemas são inseridos no desenvolvimento do projeto, tais como problemas de comunicação, problemas de gerência e heterogeneidade de recursos, seja no nível pessoal, de máquina ou de software.

Pode-se, então, listar como características do ambiente científico em larga escala:

- divisão básica em três grandes grupos de profissionais: teóricos, experimentais e computacionais;
- atividades altamente complexas;
- exigência de diferentes qualificações para execução das atividades envolvidas;
- importância predominante das atividades de pesquisa científica;
- formação de equipes multidisciplinares, podendo ser formada por profissionais de diferentes laboratórios, geograficamente distribuídos, trabalhando em esquema de colaboração;

- dificuldades na estimativa de custos e planejamento de projetos (i.e. gerência de projetos);
- o ambiente pode ser considerado, de certa forma, conservador, auto-suficiente e um pouco "artesanal", no que diz respeito às novas técnicas de computação;
- inexistência, quase absoluta, de profissionais formados em Ciência da Computação, e
- necessidade, muitas vezes, da utilização de diferentes recursos de hardware (i.e. diferentes computadores e equipamentos) e software (i.e. diferentes sistemas operacionais, programas de suporte ao desenvolvimento, etc).

II.2. Desenvolvimento de Software Científico

Na seção anterior, identificamos uma série de características e dificuldades encontradas no ambiente científico, de uma maneira geral. Muitas dessas características e dificuldades se refletem no processo de desenvolvimento e no software científico propriamente dito.

Inicialmente, descreveremos o software científico tal como ele é encontrado na literatura. A partir de nossa experiência prática no desenvolvimento de software científico, adicionamos aspectos não totalmente abordados na literatura. Após a caracterização do software científico, discutimos os problemas relacionados ao seu desenvolvimento e os aspectos específicos relacionados aos atuais métodos, técnicas e ferramentas computacionais utilizados.

II.2.1. Caracterização do Software Científico

Para melhor organizar nossa discussão, analisamos o software científico de acordo com os seguintes aspectos:

- linguagem de programação;
- tamanho e complexidade;
- tipo de processamento e estrutura de dados;
- interface;
- aspectos de qualidade;
- documentação, e
- tipos de software existentes.

Discutimos, a seguir, cada um desses aspectos separadamente.

II.2.1.1. Linguagem de Programação

No que diz respeito à linguagem de programação usada, é consenso que o software científico é, em grande parte, desenvolvido em FORTRAN. Sabe-se que alguns grupos utilizam outras linguagens, tais como C, Pascal, PL/I e ADA, mas é o FORTRAN que, efetivamente, caracteriza o software científico.

Este fato não se traduz, em momento algum, em uma incoerência, na medida em que FORTRAN foi a primeira linguagem de alto nível a oferecer compiladores adequados para utilização em diferentes plataformas de hardware, gerando código objeto bastante eficiente. Com o passar dos anos, diversas linguagens surgiram, porém, um grande número de programas já haviam sido escritos em FORTRAN e amplamente divulgados em revistas ou jornais científicos, sendo criadas as primeiras bibliotecas de programas científicos/matemáticos [RICE71a].

Naquela época, a questão da comunicação entre as diferentes linguagens ainda não estava totalmente resolvida e a complexidade dos algoritmos envolvidos levava,

naturalmente, à utilização de código já disponível em FORTRAN. Neste caso, não havia outra solução a não ser desenvolver o novo programa, também, em FORTRAN. Mesmo hoje em dia, em que as dificuldades de comunicação entre linguagens já foram minimizadas [PERE88], esta parece ser, ainda, a postura de grande parte dos desenvolvedores de software científico.

Arsac, em [ARSA84], coloca que "uma linguagem de programação passa a definir uma comunidade de pessoas que a comprende". Assim como uma linguagem natural é o meio de comunicação entre as pessoas, uma linguagem de programação passa a ser o meio de comunicação de uma comunidade de desenvolvedores. Arsac comenta, ainda, que este tem sido um impedimento grave para um perfeito entendimento entre muitos físicos, matemáticos, etc, e especialistas em computação, mas ele acredita que "o problema não está na maneira em que dizemos as coisas mas sim no que dizemos", citando Boileau, um filósofo francês, "antes de escrever, aprenda a pensar".

Apesar de toda a universalidade do FORTRAN, este não escapou de ter sofrido problemas no que diz respeito a criação de dialetos. Mesmo os esforços no estabelecimento de padrões com o chamado FORTRAN IV, FORTRAN 77 e FORTRAN 8X, mais recentemente FORTRAN 90, não conseguiram sanar esse problema. Ainda hoje, é possível encontrarmos pessoas, cuja tarefa principal é identificar diferentes padrões de comportamento da linguagem, a partir da compilação de programas escritos em FORTRAN em diferentes computadores. Uma boa coletânea de informações sobre a história do FORTRAN pode ser encontrada em [SMIT84] [METC85] [METC89].

Trabalhos mais recentes procuram fazer uma análise atual sobre a utilização de outras linguagens de programação no meio científico [RUSS87], sendo bastante

interessante as colocações feitas em [CAIL90] e [WHIT90]. Tudo indica, entretanto, que a nova versão do FORTRAN 90 garanta, ainda por algum tempo, o uso desta linguagem, tendo revisto uma série de problemas encontrados nas versões anteriores da linguagem e adicionando uma série de novas facilidades, na tentativa de impor um novo estilo de programação [METC91].

II.2.1.2. Tamanho e Complexidade

É comum encontrarmos artigos sobre software científico que identificam o tamanho de programas científicos como sendo de porte médio (i.e. entre 5.000 e 50.000 linhas de código) [BELL85] [MEYE84]. Alguns até acreditam que a tendência deste tamanho é aumentar, devido ao aumento de complexidade do código [MEYE84]. Outros, entretanto, já os identificam como sendo de grande porte (i.e. entre 100.000 e 500.000 linhas) [RODR80] [TRAU84] [NEWM87]. Na realidade, constatamos que é possível encontrar software científico tanto de médio porte como de grande porte, dependendo do tipo da aplicação. Consideramos, entretanto, pouco significativa a classificação de um software pelo número de linhas geradas, sendo muito mais importante termos maiores informações sobre seu conteúdo (i.e. operações envolvidas, sua complexidade, etc) e sua forma (i.e. arquitetura, legibilidade, documentação, etc).

É consenso que a complexidade do código encontrado em aplicações científicas é bastante grande, dificultando seu desenvolvimento, entendimento e, por consequência, sua manutenção. O software científico, ou numérico, é comumente caracterizado por ser um algoritmo composto de algoritmos menores que, por sua vez, são compostos de outros algoritmos, formando uma estrutura complexa, onde a iteração é a principal característica [CROS86].

Pode-se afirmar que a complexidade do software científico é, em grande parte, um reflexo da complexidade dos algoritmos envolvidos. Entretanto, acredita-se que outra parte da complexidade introduzida nos produtos deve-se ao descuido com a estrutura do software e sua legibilidade, agravado pelo uso de uma linguagem como FORTRAN que não provê estruturas capazes de representar abstrações de dados [YOUS90] e que não impõe um bom estilo de programação. O FORTRAN 90, aparentemente, tenta reverter este quadro incluindo uma série de novas facilidades [METC91]. Meyer [MEYE84], entretanto, nos chama a atenção de que a solução não se traduz em apenas embutir uma série de idéias incoerentes em uma mesma linguagem. O projeto de uma linguagem não deve ser visto como um acúmulo de características (sua maior crítica às propostas iniciais do atual FORTRAN 90), mas como um conjunto regular e homogêneo de construções.

II.2.1.3. Tipo de Processamento e Estrutura de Dados

Software científico é, geralmente, caracterizado por uma grande quantidade de cálculos, requerendo muito espaço em memória principal e grande utilização de CPU [MAID88] [TRAU84]. Para este tipo de software, costumava-se usar o termo "CPU-bound" que identifica a utilização intensiva da CPU, diferentemente do denominado software comercial (ex. folha de pagamento, controle de estoque, etc) que são caracterizados pelo uso intensivo de dispositivos de entrada e saída (E/S) para manipulação de dados. Neste caso, o termo usado era "I/O-bound" que identifica a utilização intensiva de dispositivos de E/S.

A partir da nossa experiência com o desenvolvimento de software científico, verificamos, cada vez mais, o aparecimento de software com características muito mais próximas da classificação "I/O-bound" do que da "CPU-bound". Ou seja, verificamos que uma grande parcela do software científico, hoje encontrado, dedica-

se, particularmente, à manipulação de estruturas de dados, seu gerenciamento, armazenamento, questões relacionadas à entrada e saída, pré e pós-processamento, etc. Esta observação, também, foi feita por Meyer e seu grupo em [MEYE84].

Em aplicações baseadas no tratamento intensivo de dados, os recu gerência de informações são de extrema importância. Em aplicações científicas deste tipo, a integração com sistemas específicos para a gerência de dados pode ser crucial [READ89].

Acredita-se que esta transformação de uma abordagem algoritmica, onde a principal preocupação era com a otimização e eficiência de algoritmos, para uma abordagem mais gerencial, de manipulação de dados, preocupação com interfaces e com questões relacionadas à portabilidade entre diversos equipamentos, deve-se a diversos fatores, dentre os quais a rapidez de processamento dos atuais equipamentos, o aumento no tamanho e complexidade das atuais aplicações, até a proliferação de máquinas e ferramentas que exigem controles mais específicos.

Esta nova geração de software científico já não corresponde mais à definição simplificada encontrada na literatura. Além disso, a atual diversificação de equipamentos de hardware torna difícil uma caracterização única sobre o tipo de processamento científico. Diferentes configurações de hardware influem, diretamente, no tipo de processamento exigido (i.e. processamento paralelo, distribuído, interativo, em lote, etc).

II.2.1.5. Interface

Cada vez mais o software científico caracteriza-se pela necessidade de interfaces mais amigáveis para a entrada de dados e apresentação de resultados, em

sua grande maioria de forma gráfica. Este novo tipo de funcionalidade exigida pode tornar mais complexo o código do programa gerado e dificulta seu desenvolvimento devido à necessidade de um maior entendimento sobre aspectos não diretamente relacionados à atividade científica, exigindo um conhecimento específico sobre dispositivos de entrada e saída ("displays" gráficos, mouse, etc) e cuidado quanto a aspectos humanos ou ergonômicos no projeto de interfaces (tempo de resposta, aspectos visuais, orientação, etc) [SHNE87] [MONT92a].

Uma decisão de projeto pode preferir fazer uma integração de um aplicativo FORTRAN com sistemas mais sofisticados, que permitam a manipulação de gráficos e estruturas de dados complexas, tais como sistemas gerenciadores de interface e sistemas gerenciadores de banco de dados, evitando, assim, o aumento desnecessário de complexidade da parte algoritmica.

II.2.1.5. Aspectos de Qualidade

A qualidade de um produto de software deve ser medida através da avaliação de diversos atributos. Esta lista de atributos desejados varia de acordo com o domínio da aplicação, sendo inadequado definir uma lista única que atenda aos diversos tipos de software [ROCH83] [PERR83]. Rocha, em [ROCH83], define um método para avaliação da qualidade de software, que pode ser aplicado aos diferentes produtos gerados durante o processo de desenvolvimento (especificação, projeto, programas) a partir da definição de um conjunto de características a serem satisfeitas e seus respectivos processos de avaliação. As necessidades específicas de cada domínio de aplicação e/ou projeto irão priorizar determinadas características em detrimento de outras.

No contexto de software científico, alguns trabalhos já foram realizados no sentido de identificar quais os atributos que produtos de software neste domínio de aplicação devem atender [BAHI88] [MAID88] [ROCH89] [PALE90c] [ROCH91]. Na literatura encontramos preocupações, basicamente, quanto aos aspectos de integridade, robustez, usabilidade, extensibilidade, simplicidade, eficiência, portabilidade e documentação [DELP84] [RODR80] [RICE71b]. Entretanto, vários outros aspectos devem ser considerados na avaliação da qualidade de um software, que dizem respeito a questões tais como sua manutenibilidade, legibilidade, etc [ARTH85] [ROCH83]. Dado que o software científico, conforme mencionado anteriormente, é, em grande parte, desenvolvido por profissionais muitas vezes sem educação formal em computação, muitos desses aspectos são negligenciados, senão totalmente ignorados.

Muitos defendem que o software científico sofre, demasiadamente, a influência do caráter evolutivo da ciência. Inicialmente, desenvolve-se o protótipo de um programa, pois não se sabe com segurança os caminhos adequados para a solução do problema, dado que este, também, não é muito bem definido. Na medida em que novas técnicas numéricas vão surgindo, o profissional desenvolvedor tenta modificar seu programa para acompanhá-las. Assim sendo, muitas vezes, não existe tempo hábil para refazer um software para que ele atenda aos requisitos de qualidade exigidos ou os custos envolvidos nesta tarefa podem estar além do orçamento disponível pelas equipes desenvolvedoras [CROSS86]. Acreditamos, entretanto, que estes problemas podem ser minimizados se a equipe estiver sensibilizada para questões de qualidade durante o desenvolvimento e a manutenção de programas.

II.2.1.6. Documentação

Dizer que o software científico caracteriza-se por uma insuficiência e até, algumas vezes, total inexistência de documentação não é novidade. Mesmo sem querer mencionar documentos tais como especificação de requisitos e de projeto, um típico programa científico não possui nem sequer a definição das variáveis usadas, ou a definição dos parâmetros passados para uma subrotina [PALE90b]. Não existindo a menor preocupação com a forma, sendo os programas produzidos praticamente ilegíveis, o software científico caracteriza-se por ser de difícil manutenção.

Existem afirmativas no sentido de que "quanto mais precisamos documentar nossos programas, mais incapaz é a linguagem de programação na expressão dos conceitos envolvidos" [CAIL90]. Isto é, em parte, verdade. Entretanto, assumir que o código gerado em uma linguagem de programação como FORTRAN expressa com clareza o algoritmo usado é, no mínimo, uma ilusão. Reconhecendo este problema, alguns esforços vêm sendo realizados no sentido de tentar oferecer mecanismos de suporte à documentação durante a fase de desenvolvimento [MAID91] [MAID92], de forma a minimizar este problema.

II.2.1.7. Tipos de Software Existentes

Finalmente, é importante distinguirmos os diversos tipos de software científico. Na verdade, existem diversas formas de subdividir, ou classificar, o software existente a partir de determinados aspectos postos em evidência. Algumas dessas classificações para software científico são descritas a seguir.

a) *Software para ciência teórica vs software para ciência experimental*

Conforme visto na seção II.1., a atividade científica pode ser dividida em duas grandes áreas: a teórica e a experimental. O tipo de software desenvolvido em cada uma dessas áreas reflete exatamente o tipo de atividade desenvolvida.

O software desenvolvido para a ciência teórica é, normalmente, desenvolvido por um cientista, ou grupo de cientistas, para seu uso próprio, sendo seu principal objetivo o resultado teórico, obtido através da implementação do software, não existindo em momento algum uma preocupação quanto à forma deste software. Em termos computacionais, podemos comparar este tipo de software a um *protótipo descartável*, conforme descrito por Davis em [DAVI88].

Software para ciência experimental é, geralmente, um software para o controle de uma experiência, aquisição de seus dados, análise e avaliação de resultados. Não sendo de todo descartável, o software tende a ser melhor elaborado, contendo atributos de qualidade que os torne, facilmente, modificáveis para reaproveitamento em outras experiências. Na prática, entretanto, muitos desses atributos de qualidade são negligenciados, tornando sua manutenção bastante difícil.

b) *Domínios de Aplicação no Contexto Científico*

Um outro tipo de classificação é dada quanto aos diversos domínios de aplicação encontrados no contexto científico, ou seja, nos diversos ramos da ciência (ex. física, química, matemática, etc). Neste tipo de classificação, teríamos certamente a inclusão de software teórico e experimental como sub-ramos de cada um desses domínios de aplicação.

Embora possa ser útil, olhando sob um aspecto mais específico, diferenciarmos, por exemplo, o software para atender a um problema da área de química de outro que atende à área da física, dado que os conceitos envolvidos são distintos, sob um aspecto mais geral, verificamos que a base de cálculos (i.e. métodos numéricos, simulações, etc), a manipulação de dados e de gráficos é, basicamente, a mesma. Desta forma, consideramos a diferenciação entre software para ciência teórica e experimental muito mais forte do que a que considera os domínios de aplicação, dado que a primeira identifica as características específicas que cada um deles assume.

c) *Software Transitório, Permanente e Intermitente*

Pereira [PERE87] subdivide os diversos tipos de software científico em: software *transitório*, *permanente* e *intermitente*.

O primeiro tipo caracteriza aqueles em que sua aplicação ocorre, exclusivamente, para um trabalho e possui particularidades muito específicas que, dificilmente, podem ser aproveitadas em outro trabalho.

Software permanente é aquele que pode ser usado sempre que surgirem serviços relativos ao assunto abordado e que, portanto, foi desenvolvido para atender, de uma forma mais geral, a uma dada atividade.

Finalmente, *software intermitente* é aquele que, embora possa ser aplicado em diversos trabalhos sobre o mesmo assunto, restringe-se a circunstâncias pré-estabelecidas.

Como exemplo, Pereira coloca que um programa que tenha como objetivo determinar os índices pluviométricos específicos da cidade de São Paulo no ano 2000 seria do tipo transitório. Já um programa que calcula, indiscriminadamente, índices pluviométricos seria do tipo permanente. Enquanto um programa intermitente determinaria, por exemplo, os índices pluviométricos apenas das cidades litorâneas.

Pereira não entra em detalhes sobre quais as características que cada um destes diferentes tipos de software assume, não acrescentando muitas informações quanto à importância desta classificação para a caracterização do software científico.

d) *Software resultante de Pesquisa vs Software Produto*

Rocha [ROCH87] diferencia um software resultante de uma pesquisa científica do software que deverá ser o produto final desta pesquisa. O primeiro caracteriza-se pelo fato de que deve ser considerado como um protótipo, onde a maior preocupação é com seu conteúdo, ou seja, a qualidade do algoritmo, sem preocupações com aspectos relativos à sua forma. O segundo deve obedecer aos princípios e técnicas da Engenharia de Software, de forma a se ter um bom produto de software, tanto do ponto de vista do conteúdo quanto da forma.

Observa-se que este tipo de diferenciação é válida para qualquer domínio de aplicação científica. Dada esta nova visão sobre caracterização de software podemos verificar que o que chamamos, anteriormente, de software para ciência teórica tem características bastante similares ao software resultante de pesquisa, que caracteriza-se por dar ênfase ao conteúdo ao invés da forma. Já o software para ciência experimental deveria, a princípio, ser construído como software produto, já que este tende a ter vida útil longa, sendo usado por diversos usuários e sofrendo diversas modificações ao longo de sua vida útil.

Conforme já foi colocado, anteriormente, existe, na prática, um certo descuido ou não valorização de aspectos relativos à forma deste software. Assim sendo, este não pode ser considerado como um software produto.

e) *Software "Batch" vs Software Interativo*

Embora, a princípio, pareça desatualizado distinguirmos *software em lote* ("batch") de *software interativo*, em um ambiente como o científico, esta distinção é bastante significativa. O software do tipo em lote é aquele que tende a ter cálculos intensos, não dependendo da intervenção humana. Já o software interativo preocupa-se, basicamente, com as formas de interação com o usuário final, com aspectos de interface, manipulação de estruturas de dados e gráficos.

O software interativo, hoje em dia, é bastante comum no ambiente científico, devido à introdução de estações de trabalho e micro-computadores. Enquanto, antigamente, todo software científico era do tipo em lote, hoje, apenas aqueles programas que possuem características de extensa utilização de CPU é que são desenvolvidos para serem processados em lote.

f) *Software "stand-alone" vs Software "library-like"*

Existe, ainda, uma distinção entre o software desenvolvido para funcionar de forma própria ("stand-alone"), muitas vezes visto como pacotes fechados ("packages") em que o usuário final, dificilmente, tem acesso ao programa fonte, e aqueles que fazem parte de um sistema mais amplo e que, portanto, são distribuídos como rotinas de uma biblioteca ("library-like"), ou estão abertos para modificação ou refinamento por seus usuários ("open-systems").

Este tipo de diferenciação não é específica ao software científico e pode ser encontrada nas demais áreas de aplicação. A diferença básica entre estes tipos é que, quase sempre, o software desenvolvido para funcionar como um pacote fechado leva em consideração, principalmente, os aspectos de eficiência e robustez, esquecendo os demais atributos de qualidade, enquanto os sistemas abertos estão muito mais preocupados com a estrutura modular de seus subsistemas, com a geração de documentação adequada e com a legibilidade do código produzido, para facilitar sua manipulação.

II.2.2. Problemas no Desenvolvimento

As seções II.1 e II.2.1 caracterizaram dois aspectos importantes com relação ao desenvolvimento de software científico. Um deles diz respeito às características do ambiente científico, que por si só são bastante específicas. Outro aspecto importante foi verificar, dada a imensa gama de software científico hoje disponível, as características que este software adquiriu ao longo dos anos.

Parte das características incorporadas ao software científico estão relacionadas à natureza das aplicações científicas, outras foram influenciadas pelas tecnologias computacionais disponíveis a nível de hardware. Observa-se, entretanto, que o atual momento caracteriza-se por uma influência de tecnologias computacionais no nível de software. Apenas recentemente nota-se uma preocupação, por parte de alguns profissionais da área científica, quanto aos rumos que vem tomando o desenvolvimento de software científico [CROS86] [BOCK89] [KNOB91].

Alguns nos chamam a atenção para o fato de que especialistas em computação não mais se interessam pelas aplicações científicas [ARSA84]. Outros observam que

as aplicações em computação se diversificaram muito e, hoje, a computação pode, também, ser considerada uma ciência, com pesquisas próprias e, como tal, precisa de incentivos para justificar a elaboração de novas técnicas que sejam específicas a uma determinada área. Hertzberger [HERT89] sugere que o meio científico, mais especificamente o ramo da Física de Altas Energias, procure criar novas frentes de pesquisa que sejam comuns aos interesses da computação. Isto indica uma preocupação em facilitar o diálogo entre físicos, matemáticos, etc, e especialistas em computação, dado que o nível de complexidade e tamanho das aplicações científicas cresce de tal forma que torna impraticável continuar desenvolvendo software com os métodos primitivos e informais utilizados até a presente data.

Pode-se listar como principais problemas do desenvolvimento de software científico [ATAS89] [MAID88] [TRAU84] [CROSS86] [CERN89d]:

- a falta, muitas vezes, de uma definição precisa sobre o software a ser construído;
- os requisitos são, quase sempre, alterados ao longo do tempo;
- a complexidade do domínio do problema faz com que o software seja desenvolvido pelo próprio especialista na área;
- a falta de mão-de-obra especializada em computação para trabalhar nesta área;
- a falta de prática no uso ou, até mesmo, a inexistência de métodos e técnicas computacionais, modelos de ciclo de vida, etc, adequados ao desenvolvimento de software científico;
- a falta de ferramentas de software que apoiem adequadamente o desenvolvimento;
- a necessidade da execução de sistemas em diferentes plataformas de hardware e software;
- problemas gerenciais causados pela formação de grandes equipes, podendo estar trabalhando em um esquema de colaboração e distribuídas geograficamente, e

- a dificuldade de comunicação entre o especialista no domínio da aplicação e o especialista em ciência da computação.

Estes problemas estão relacionados a três aspectos:

- a natureza da aplicação;
- a questão de recursos humanos, e
- as questões computacionais (i.e. hardware, métodos, técnicas e ferramentas).

A natureza da aplicação influi diretamente no perfil do profissional desejado, que caracteriza-se por ser um sujeito pluri-apto, no que diz respeito aos conhecimentos da área de aplicação e de computação. A natureza da aplicação, também, influi diretamente nos métodos e técnicas computacionais adequados ao desenvolvimento do software e nos recursos de hardware necessários [SOUZ90] [SOUZ91].

É óbvio que os recursos humanos disponíveis irão influenciar no sucesso do uso do hardware, métodos e técnicas computacionais. A recíproca, também, é verdadeira na medida em que o treinamento e os conhecimentos específicos, necessários à utilização dessas ferramentas, irão ditar o perfil do profissional desejado. A *figura 3* mostra, claramente, a forte relação entre estes três aspectos.

Nesta tese preocupamo-nos, especificamente, com os aspectos relacionados aos métodos, técnicas e ferramentas computacionais para o atendimento dos requisitos do desenvolvimento de software científico, não sendo discutidos os aspectos de hardware.

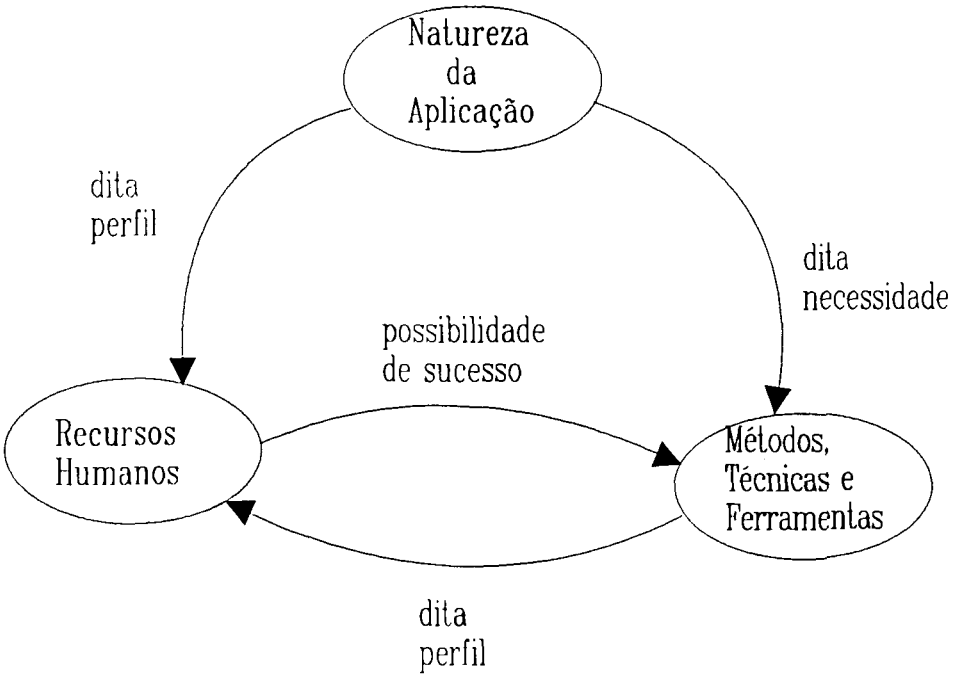


Fig. 3 - Aspectos do desenvolvimento de software científico

II.2.3. Métodos, Técnicas e Ferramentas Computacionais

O software científico, atualmente disponível, atesta de forma suficiente o tipo de métodos (ou melhor a falta dos mesmos) usados em seu desenvolvimento [MEYE84]. Embora alguns grupos admitam que, apenas recentemente, os métodos desenvolvidos para projeto de sistemas em ambientes comerciais e administrativos estejam sendo experimentados em aplicações científicas [BROW85] [MEDE87] [KNOB91], pode-se observar, ainda, um grande desconhecimento desses métodos em diversos outros grupos.

De uma maneira geral, físicos, matemáticos, engenheiros, etc, têm a opinião de que não faz parte de seus objetivos deter-se com detalhes que não dizem respeito à sua área de atuação. A falta de prática, no contexto científico, no uso dos métodos e técnicas já existentes para ambientes comerciais torna difícil a avaliação sobre sua adequação a este novo ambiente.

Assumindo que os atuais métodos e técnicas de Engenharia de Software não são, totalmente, adequados ao contexto científico, cabe-nos perguntar qual será a razão para que isto aconteça. Para respondermos a esta pergunta, é necessário situarmos a discussão em um nível adequado, procurando respostas novas e não, simplesmente, desconsiderando o uso dos métodos e técnicas da Engenharia de Software porque algumas de suas propostas não se aplicam a este contexto.

Dizer que o modelo de ciclo de vida tradicional, subdividindo o desenvolvimento de software em várias fases com produtos intermediários bem definidos, não reflete exatamente as etapas, tal como observadas no processo científico (veja II.1), é uma observação importante, na medida em que um modelo de ciclo de vida deve responder às necessidades específicas do ambiente onde é inserido. O problema é que, ao verificar que este ou outro modelo não corresponde às necessidades do contexto científico, abandona-se a busca do modelo adequado e continua-se desenvolvendo software artesanalmente.

Existe na literatura uma série de modelos de ciclo de desenvolvimento de software que podem servir de base para o estabelecimento de um modelo adequado ao processo científico [DAVI88]. Rocha, em [ROCH88], propõe uma nova metodologia para o desenvolvimento de software científico, dividindo o desenvolvimento em duas etapas. A primeira delas, denominada *etapa de pesquisa*, tem como objetivo concentrar o esforço de engenheiros na solução do problema específico. A segunda etapa garante que o software construído está de acordo com técnicas modernas de Engenharia de Software, denominando-se *etapa de produção*.

A primeira etapa deste modelo de desenvolvimento prevê duas fases [ROCH88]: a) elaboração da proposta de desenvolvimento, e b) pesquisa, incluindo

uma análise detalhada do problema, o desenvolvimento de um modelo matemático, a construção do algoritmo, verificação da correção do algoritmo e análise de desempenho. A segunda etapa segue, exatamente, as fases do ciclo de desenvolvimento tradicional (i.e. especificação, projeto, construção e avaliação).

Nesta proposta é criada a figura de um profissional intermediário que atua nos dois grupos envolvidos nas diferentes etapas do desenvolvimento [ROCH87]. Este profissional deve possuir conhecimento tanto na área de aplicação específica como na ciência de computação. Na prática, entretanto, este elemento é bastante difícil de ser encontrado, sendo necessário um treinamento rigoroso sobre os aspectos de computação, caso o elemento intermediário seja um especialista na área de aplicação, ou vice-versa.

Uma pesquisa de opinião [WERN88a], interna ao grupo de desenvolvimento da PETROBRÁS envolvido neste trabalho, identificou como posição do grupo a eliminação deste elemento intermediário, sugerindo um canal direto entre engenheiros e analistas desde a primeira etapa de desenvolvimento. O argumento usado é que a formação de uma equipe multidisciplinar possibilita a criação de um "protótipo", muito mais próximo dos padrões de qualidade desejados, já na primeira etapa de desenvolvimento. Neste caso, a etapa de produção teria como objetivo a melhoria do software quanto aos aspectos de interface e desempenho, utilizando recursos computacionais mais adequados.

Com base nas observações feitas através da pesquisa de opinião e na proposta original, iniciou-se um trabalho sobre a elaboração de um manual de desenvolvimento que servisse como guia durante o desenvolvimento de software deste grupo, detalhando um pouco mais as atividades relacionadas à etapa de pesquisa especificamente [WERN88b].

De uma maneira geral, as experiências de utilização de métodos e técnicas de Engenharia de Software em projetos científicos são, ainda, bastante recentes [MEDE87] [KNOB91]. É comum encontrar desenvolvedores de software neste ambiente que recusam-se a adotar qualquer técnica de programação que implique em trabalho extra, disciplina ou que possam restringir, de alguma forma, sua criatividade.

Um outro aspecto bastante fundamental é a disponibilidade de ferramentas computacionais para suporte ao desenvolvimento de software, sem as quais torna-se impraticável a introdução de qualquer novo método.

Com o advento das ferramentas CASE, uma série de ferramentas foram desenvolvidas e lançadas no mercado. Entretanto, ao considerarmos o ambiente científico, verificamos que algoritmos complexos, escritos em FORTRAN, são desenvolvidos utilizando-se apenas um editor e um compilador. Alguns desenvolvedores fazem uso de ferramentas um pouco mais poderosas (por exemplo, um depurador, um analisador de desempenho, ou ainda analisadores estáticos e dinâmicos [WERN91c]), mas, em geral, o que existe é um quase total desconhecimento dessas ferramentas. É bem verdade que os equipamentos de grande porte, bastante utilizados neste ambiente, muitas vezes não oferecem ferramentas de uso amigável e isso tem que ser considerado como um importante fator inibidor de uma utilização mais sistemática e frequente de ferramentas de apoio ao desenvolvimento.

Iniciativas como TOOLPACK [ILES86] [OSTE84], tem como objetivo o desenvolvimento de um ambiente integrado que dê suporte ao desenvolvimento de programas em FORTRAN. Note-se que não basta fornecer ferramentas isoladas que apoiem as diversas etapas do desenvolvimento. É necessário ter-se um ambiente de

desenvolvimento que integre estas ferramentas e auxilie no desenvolvimento de produtos com consistência. Ferramentas não integradas são de muito pouca utilidade, pois não podem garantir aspectos de consistência e integridade entre os diferentes produtos gerados durante o processo de desenvolvimento do software.

Biblioteca de rotinas é um exemplo especial de ferramenta utilizada, com bastante tradição e sucesso, no ambiente científico. Existem, hoje, diversas bibliotecas de software numérico (ex. IMSL, NAG, CERN Library, etc) que são consideradas como um dos melhores exemplos de software reutilizável [BIGG89] [CHEA89]. Exemplos de geradores de aplicação, linguagens de muito alto nível e linguagens dedicadas podem ser encontrados em algumas aplicações científicas específicas [CLEA88] [WERN89a]. Devido a restrições de utilização em determinadas áreas de aplicação, estas não são amplamente utilizadas como as bibliotecas de rotinas (ver capítulo III para maiores detalhes sobre estas ferramentas).

Não esperemos, entretanto, que todos os problemas no desenvolvimento de software científico sejam resolvidos a partir da utilização de ferramentas automatizadas, mesmo que estas sejam adequadas e de boa qualidade. É preciso compreender o processo de desenvolvimento como um todo e entender os conceitos envolvidos por trás dos métodos, técnicas e ferramentas utilizados. Conforme colocado por Arsac [ARSA84]: "uma ferramenta só é eficaz nas mãos de um bom trabalhador". É fundamental, portanto, que o desenvolvedor esteja familiarizado com os conceitos e paradigmas de desenvolvimento que motivaram a criação do método e/ou ferramenta para que seja capaz de utilizá-los correta e adequadamente.

II.3 O Ambiente Científico no CERN

CERN é um dos maiores e mais modernos laboratórios do mundo, cuja pesquisa é conhecida como Física de Partículas. Situado na fronteira entre a Suiça e a França, o CERN foi fundado em 1954, contando atualmente com 17 países membros (Áustria, Bélgica, Dinamarca, Alemanha, França, Grécia, Itália, Holanda, Noruega, Portugal, Espanha, Suécia, Suiça, Inglaterra, Polonia, Checoslovaquia e Finlândia) que colaboram em pesquisas científicas relacionadas à área de Física de Altas Energias [CERN87a]. Também colaboram, sistematicamente, diversos outros países não membros (ex. Estados Unidos, China, Brasil, etc).

O CERN tem um quadro permanente com cerca de 3500 pessoas com diferentes especialidades, dentre as quais um terço de físicos e engenheiros. Além deste grupo, cerca de 3200 físicos de mais de 200 centros de pesquisa do mundo inteiro utilizam as facilidades do CERN em suas pesquisas [CERN87b].

Diversos projetos científicos ocorrem no CERN, organizados com base na colaboração de diversos laboratórios espalhados por todo o mundo. LEP ("Large Electron-Positron Collider") é um desses projetos e constitui um dos maiores aceleradores de partícula do mundo, com 27 quilômetros de circunferência, tendo sido aprovadas 4 experiências para o estudo de colisões entre pósitrons e elétrons: ALEPH, DELPHI, L3 e OPAL [CERN89a] [CERN89b].

Para dar suporte computacional às experiências, o CERN possui uma variedade de equipamentos conectados, localmente e externamente com universidades e laboratórios de pesquisa do mundo inteiro, através de redes de computadores. Dentre os equipamentos, atualmente, disponíveis no CERN podemos citar [ZANE90]: IBMs 3090/600E, Cray X-MP48, VAXes 8800/8700/8650, diversas

estações de trabalho (Apollo, VAX, HP, IBM-Risc, Next, etc) e computadores pessoais (Macintosh e IBM-PC). Ao todo, são mais de 6000 usuários registrados no centro de computação do CERN.

O CERN tem tradição no projeto tanto de hardware especial (ex. equipamentos digitalizadores, processadores especiais, sistemas de comunicação, etc) como de software (ex. compiladores, bibliotecas de programas e aplicativos específicos da área de Física de Altas Energias) [ZANE90].

Em recente relatório sobre as atividades de computação previstas para os anos 90, grupos de trabalho avaliaram as seguintes áreas de atividade existentes no CERN [CERN89d]:

- computação para aceleradores;
- computação para engenharia;
- computação para experiências;
- computação para teóricos;
- computação para sistemas de gerência de informação, e
- redes de computadores.

O principal objetivo deste relatório foi prover informações e recomendações que pudessem auxiliar a direção do CERN no planejamento de recursos computacionais para dar suporte aos trabalhos desenvolvidos no laboratório nos próximos 5 anos. Algumas das recomendações feitas neste relatório são:

- "os recursos alocados à computação, em geral, devem ser maiores que os níveis atuais";

- "a capacidade de processamento e armazenamento de dados deve aumentar, incluindo a utilização de mais facilidades de computação paralela e o uso de volumes de alta capacidade de armazenamento";
- "facilidades de comunicação devem melhorar de forma a facilitar a descentralização das atividades de análise e processamento de dados físicos";
- "é preciso reduzir a diversidade de software através do estabelecimento de ambientes comuns de software";
- "é essencial o estabelecimento de uma estratégia para o aumento substancial do nível de profissionais em informática", e
- "o CERN deve treinar e re-treinar intensivamente os funcionários na tecnologia de informática".

A partir das recomendações acima citadas, pode-se observar diversos aspectos relacionados à caracterização deste ambiente, confirmando algumas das características de ambientes científicos descritas neste capítulo.

Observa-se a preocupação com o aumento da capacidade de processamento e armazenagem de dados e a descentralização do processamento. Isso se deve ao tipo de processamento efetuado neste laboratório (i.e. tratamento intensivo de dados) e a maneira como ele é feito (i.e. diversas equipes trabalhando em um mesmo projeto). Note-se, entretanto, a preocupação em resolver determinados problemas, tais como a diversidade dos produtos de software usados e a falta de especialistas em computação através do estabelecimento de padrões para o software, aumento do número de funcionários especialistas em computação e treinamento de todos os funcionários envolvidos no desenvolvimento de software.

Dentre as diferentes áreas de atividade da computação no CERN, pode-se dizer que a mais intensa é a de computação para experiências. Neste contexto,

diversas atividades são desenvolvidas, necessitando recursos específicos de computação. Podemos agrupar essas atividades nas seguintes áreas [CERN89e] [PIME91]:

1) *Aquisição de Dados*: envolve a construção de detetores, sistemas de aquisição de dados, controle de detetores, leitura e pré-processamento de dados;
2) *Análise de Dados*: envolve a reconstrução de eventos a partir de dados em bruto, a análise de eventos e tratamento estatístico do conjunto de eventos obtidos;
3) *Simulação*: consiste na geração de processos físicos, teóricos, e na simulação da interação de partículas com os detetores, e
4) *Visualização*: consiste na visualização gráfica de eventos e detetores.

A complexidade no desenvolvimento dessas atividades requer o envolvimento de diversos profissionais com diferentes especialidades (técnicos, engenheiros, físicos, especialistas em computação). O software desenvolvido é bastante complexo, devido à complexidade das operações envolvidas e, por isso, a gerência de dados é fundamental em todas as áreas de atividade. Grande parte do software gerado é construido pelos próprios participantes da experiência, podendo contar com alguns pacotes oferecidos pela divisão de computação (simuladores [BRUN78], pacotes para visualização gráfica [BRUN90] [BRUN88], pacotes para o gerenciamento de dados [BRUN87] [GREE89], etc).

O relatório do grupo de software para experiências apresenta dentre os problemas encontrados no desenvolvimento deste tipo de software [CERN89e]:

- a atitude gerencial dos projetos em considerar a construção de software como atividade secundária em relação, por exemplo, à construção de detetores;

- a composição de equipes formadas por físicos iniciantes com pouca ou nenhuma experiência em projetos de larga escala, sem treinamento adequado, contando com especialista locais muitas vezes sem educação formal em computação;
- a falta de uma adoção oficial de métodos e técnicas de Engenharia de Software em muitas experiências, e
- a inexistência de documentação adequada sobre o software produzido, dificultando sua manutenção.

Neste mesmo relatório, indica-se Aleph como a primeira experiência em Física de Altas Energias a adotar, em 1984, um método de engenharia de software (SA/SD). Entretanto, só recentemente, esforços no sentido de dar suporte oficial a métodos e ferramentas de análise e projeto foram realizados no CERN [CERN89e]. É importante ter em mente que o CERN é tido como o exemplo de organização no desenvolvimento computacional para a área de Física de Altas Energias, cabendo a ele implementar estratégias de desenvolvimento e estabelecer padrões para toda a comunidade científica. Logo, o suporte oficial de determinados métodos, técnicas e ferramentas computacionais, a partir de uma avaliação do atual estado da arte, oferecendo, ainda, uma estrutura adequada para treinamento, é fundamental pelo impacto que pode ter no desenvolvimento de software na área.

Dentre as ferramentas, atualmente, disponíveis no CERN, podemos citar [CERN89e]:

- gerenciadores de código fonte, que permitem o controle de versões e alterações de programas (Historian [OPCO85], CMZ [BRUN89], PATCHY [KLEI83]);
- editores sensíveis à linguagem, onde informações sobre a sintaxe específica da linguagem usada são fornecidas (LSE [DIGI87]);

- analisadores estáticos, capazes de determinar variáveis não inicializadas ou não utilizadas, trechos de programa não executados, etc (FLOPPY [BUNN89]);
- ferramentas para teste de programas, para auxílio nas atividades de teste a partir da geração automática de diversos casos de teste (PCA, DTM [DIGI88a]);
- otimizadores, que permitem a identificação de partes do programa que consomem o maior tempo de CPU (SPY, LAD [CERN89e], PCA), e
- biblioteca de programas, que fornece uma série de programas científicos úteis à comunidade (CERNLIB [CERN89c]).

Quanto à questão de ferramentas de software, as recomendações feitas no relatório dizem respeito à criação de grupos de interesse, formados por usuários de ferramentas, e de um grupo central para a avaliação de requisitos específicos da área de Física de Altas Energias e negociação de licenças de ferramentas consideradas necessárias [CERN89d].

Concluindo as informações colocadas nesta seção, pode-se dizer que o ambiente científico do CERN se enquadra bastante na caracterização geral feita nas seções anteriores, porém, com uma importante ressalva: existe uma divisão de computação alerta às dificuldades, hoje, vivenciadas por esta comunidade, abrindo novos caminhos para solução dos problemas encontrados através da conscientização de gerentes, físicos, engenheiros e técnicos quanto aos avanços tecnológicos na área de informática.

III. Reutilização de Software

O termo *reutilização de software* caracteriza-se pela utilização de produtos de software desenvolvidos ao longo do ciclo de vida em uma situação diferente daquela para a qual foram, originalmente, produzidos [FREE80]. Neste sentido, consideramos não apenas a reutilização de produtos executáveis (i.e. módulos, rotinas, fragmentos de código, etc) mas, também, produtos não executáveis (i.e. definição de requisitos, especificação, projeto, etc) [HORO89] [BIGG89] [TRAC90] [NEI91].

A reutilização de software tem sido um assunto bastante discutido na literatura técnica nos últimos 15 anos, mas que parece, ainda hoje, não ter conseguido preencher completamente suas promessas quanto ao aumento da produtividade no desenvolvimento de software e à construção de melhores produtos, tornando-os mais confiáveis, consistentes e padronizados [BIGG89].

Neste capítulo, discute-se diversos aspectos relacionados ao tema com o intuito de proporcionar uma visão geral sobre o assunto. Faz-se, inicialmente, um histórico sobre as pesquisas já desenvolvidas e, então, apresenta-se seus aspectos mais relevantes, incluindo as tecnologias atualmente disponíveis, as vantagens na adoção de uma estratégia de reutilização e os principais problemas encontrados. Finalmente, discute-se o atual estado da prática de reutilização de software no ambiente científico.

III.1 Histórico

Freeman, em [FREE87a], faz um pequeno histórico sobre as pesquisas já desenvolvidas relacionadas ao tema de reutilização de software. Ele marca o final da década de 70 como o início real das pesquisas em reutilização, que foram motivadas pela, então, chamada *"crise do software"*. Antes disso, diversos desenvolvimentos

técnicos, em áreas como linguagens de programação, estruturas de dados, sistemas operacionais, transformações de programas e técnicas de especificação, foram essenciais para o sucesso da reutilização de código. Estas pesquisas, entretanto, não tinham como proposta original a reutilização. Freeman caracteriza este período como um período *"pré-histórico"*.

A década de 80 caracteriza um período de inovações no que se refere a tecnologias, planejamento estratégico e procedimentos de reutilização. Freeman admite que ainda estamos passando por este período de descobertas, já possuindo alguns resultados bastante interessantes. Seppanem, em [SEPP87], faz uma revisão bibliográfica sobre diversos trabalhos de pesquisa na área.

Entretanto, se considerarmos a prática desses novos resultados, muito poucas organizações desenvolvedoras de software tentaram, efetivamente, explorá-los. Esta situação nos leva a caracterizar um período em que a maioria dos sistemas de apoio à reutilização, hoje disponíveis, são protótipos bastante simplificados e que não tiveram a oportunidade de ser, devidamente, validados e aperfeiçoados após uma aplicação prática, em caso real [FREE87a].

Espera-se, portanto, que a década de 90 possa ser marcada por experiências práticas de reutilização nas mais diversas áreas de aplicação, concretizando, assim, tecnologias, estratégias e procedimentos de reutilização de software e abrindo caminhos para novas abordagens.

III.2. Aspectos Básicos

Ao discutirmos sobre o tema da reutilização de software é preciso ter em mente seus aspectos mais relevantes. Em estudo preliminar sobre reutilização

[WERN91b], pudemos verificar que é preciso estabelecer nossa discussão sobre, basicamente, três diferentes aspectos:

a) o que desejamos reutilizar, isto é, qual é o objeto de reutilização;
b) como reutilizar, isto é, qual é o processo envolvido e,
c) definir o quanto e como fomos, efetivamente, capazes de reutilizar, isto é, a avaliação, ao final do processo, dos resultados obtidos determinando os fatores que facilitaram ou dificultaram o processo.

Esses aspectos são fundamentais para que se estabeleça uma plataforma mínima para discussão sobre o tema. A seguir, apresentamos cada um desses aspectos separadamente.

III.2.1. Objeto de Reutilização

Quanto ao objeto de reutilização, é possível estarmos interessados em reutilizar diversos produtos de software, ou, ainda, os recursos humanos envolvidos em projetos de software similares. Cada um desses objetos possui características específicas que irão determinar o processo de reutilização a ser usado.

Tracz, em [TRAC90], faz uma interessante análise sobre os diversos objetos para reutilização de software. Ele constata que a reutilização no nível de código é, certamente, um lugar seguro para se começar a reutilização e é, na maioria dos casos, onde ela termina. É, entretanto, interessante ressaltar que a cadeia de decisões de projeto, utilizadas ao longo do processo de desenvolvimento, não mais se encontra no código [PRIE90] e que, na verdade, a maior parte do tempo gasto no desenvolvimento de software está, justamente, nas etapas anteriores à codificação [HORO89].

Portanto, a questão sobre a reutilização de software não está, somente, na possibilidade de reutilizar linhas de código já escritas, mas em reutilizar todo o conhecimento adquirido durante o desenvolvimento desse código. Neste sentido, é que os estudos desenvolvidos, denominados na literatura como *Análise de Domínio* [PRIE90], na identificação de termos, objetos e operações de uma classe de sistemas similares, em um certo domínio de problema, oferecem uma boa plataforma para a obtenção de melhores ganhos no processo de reutilização.

Draco [NEIG84] [NEIG89] é um exemplo de sistema que implementa uma abordagem para a reutilização de análise e projeto de aplicações em um certo domínio de aplicação. Nesta abordagem, três novos figurantes são inseridos no desenvolvimento de aplicações: o analista do domínio de aplicação, o analista de modelagem do domínio e o projetista do domínio. O primeiro deles examina as necessidades e requisitos de uma coleção de sistemas similares, definindo objetos e operações básicas. O segundo executa uma função similar à do primeiro, sendo que este está mais preocupado com notações e técnicas que obtiveram mais sucesso na modelagem de aplicações. A partir das informações fornecidas pelos dois primeiros, o projetista do domínio define diferentes implementações para os objetos e operações do domínio da aplicação. Desta forma, o desenvolvimento de novas aplicações é guiado pelo conhecimento armazenado no sistema sobre seu domínio de aplicação específico.

Freeman define como objetos de reutilização [FREE87b]: fragmentos de código, estruturas lógicas (i.e. processos e estruturas de dados), arquitetura funcional, conhecimento externo (i.e. área de aplicação, conhecimento sobre o desenvolvimento) e informações ambientais (i.e. utilização do software, transferência de tecnologia).

Outro tipo de reutilização usado com bastante sucesso diz respeito ao conhecimento adquirido pela mão-de-obra envolvida no desenvolvimento de projetos similares [TRAC90] [WEGN89] [TRAC88a]. Considerar a reutilização de mão-de-obra como sendo uma das principais razões para a obtenção de sucesso no desenvolvimento de software é uma decisão sábia, na medida em que, ainda hoje, o desenvolvimento é, em grande parte, realizado por pessoas que detêm o conhecimento específico sobre o desenvolvimento de software em determinadas áreas de aplicação (i.e. arquiteturas, projetos e sistemas similares). Selby, em [SELB89], identifica como um dos fatores para o sucesso da reutilização em projetos da NASA, entre outros, a pouca movimentação de profissionais nesta organização.

Notamos que quanto mais abstrato é um produto de software mais dificuldade temos em reutilizá-lo. Isto se deve em parte à falta de representações adequadas de informações sobre os estágios iniciais do desenvolvimento de software que possam ser, devidamente, processadas por computador e, portanto, reutilizadas [BIGG89] [WEBS88].

III.2.2. Processo de Reutilização

Definido o tipo de objeto de reutilização em questão, define-se como *processo de reutilização* todas as atividades envolvidas durante a reutilização de tal objeto.

Cohen, conforme apresentado por Peterson [PETE91], apresenta três modelos para o processo de reutilização:

a) *processo adaptativo*: feito através da adaptação de produtos de software de um certo domínio para criação de sistemas modificados ou para transferência de um sistema a uma outra plataforma ou ambiente de execução.

b) *processo de parametrização*: consiste na criação de produtos padrão, denominados "recursos de domínio", cuja reutilização é feita através de uma atividade de parametrização.

c) *processo de engenharia*: envolve atividades diversas, tais como a determinação de teorias, tecnologias atuais e relevantes, o conhecimento de especialistas em um certo domínio de aplicação, etc.

Basili e Rombach, em [BASI91], incluem como principais atividades do processo de reutilização:

- a compreensão de um certo alvo de reutilização (ou especificação);
- identificação de candidatos para reutilização;
- avaliação do potencial de reutilização de cada um dos candidatos e seleção do objeto mais adequado, caso exista;
- modificação do objeto selecionado, e
- integração do objeto modificado ao processo de desenvolvimento em andamento.

Biggerstaff e Richter [BIGG89] dividem as atuais abordagens para reutilização em dois grandes grupos, baseando-se no tipo de tecnologia aplicada: *tecnologias de composição* e *tecnologias de geração*. Cada uma dessas abordagens implica em um processo de reutilização distinto. As seções que se seguem apresentam cada uma delas separadamente.

III.2.2.1. Tecnologias de Composição

Este grupo de tecnologias caracteriza-se pela composição de aplicações a partir de componentes atômicos, bem definidos, idealmente, não modificados ao serem reutilizados [BIGG89]. Neste grupo de tecnologias, é fundamental a existência de um mecanismo de composição, não sendo suficiente a disponibilidade, pura e simples, de uma coleção de componentes atômicos (ex. biblioteca de componentes).

Exemplos de tecnologias de composição são o mecanismo de "pipe" utilizado no sistema UNIX [RICE89], ambientes de suporte a bibliotecas de componentes [PROF89] [WERN91a], linguagens para interconexão de módulos [PRIE86] e programação orientada a objetos [COX 86].

O sistema UNIX oferece um grande número de programas que executam funções específicas. A partir de seu mecanismo de "pipe", onde a saída de um programa é ligada à entrada de um segundo, é possível construir programas com funções mais complexas [RICE89].

Um ambiente de reutilização baseado no uso de bibliotecas de componentes deve prover um esquema adequado para classificação de componentes, assim como ferramentas de suporte para a recuperação, compreensão, modificação e, principalmente, composição desses componentes [GOGU86].

Um esquema de classificação adequado permite a localização rápida e eficiente de componentes de uma biblioteca. Prieto-Diaz propõe como esquema de classificação um esquema baseado em "facetas" (i.e. perspectivas, dimensões, pontos de vista) [PRIE85] [PRIE89]. Este esquema consiste em descrever relações genéricas básicas entre elementos, classificando-os a partir de uma montagem, ou síntese, de

suas classes elementares. Prieto-Diaz introduz, ainda, o conceito de *distância conceitual* entre termos, que permite a seleção de componentes fortemente relacionados. Outros trabalhos desenvolvidos no sentido de prover mecanismos para a localização de componentes similares de uma biblioteca são descritos em [PINT88] [MAAR89].

Um outro tipo de esquema de classificação é conhecido como *esquema enumerativo*. Este é o esquema mais tradicional e caracteriza-se pelo fato de todo o universo de conhecimento ser descrito através de classes que são sucessivamente sub-divididas, refletindo todas as possibilidades de combinação [PRIE89]. Um esquema deste tipo pode ser exemplificado pela taxonomia introduzida por Darwin para classificação de animais.

A compreensão de componentes é uma das atividades mais difíceis no processo de reutilização, pois exige um esforço mental muito grande no entendimento do modelo computacional de cada componente, na tentativa de utilizá-lo adequadamente. Ferramentas do tipo hipertexto [CONK87] [MEND91] podem auxiliar bastante neste contexto, provendo uma rede de informações variadas, incluindo textos, grafos, figuras, etc, que auxiliam na compreensão da funcionalidade dos componentes, sua utilização e comportamento.

O processo de modificação de componentes é, extremamente, dependente do ser humano, tornando difícil o fornecimento de mecanismos de suporte a esta atividade. É possível, entretanto, prover ferramentas periféricas, tais como:

- controladores de alterações e históricos, que guardam informações sobre as alterações de componentes;

- analisadores de impactos e simuladores, que são capazes de determinar os impactos causados por alterações de componentes, e
- linguagens capazes de restringir os efeitos de alterações em áreas facilmente localizáveis (i.e em módulos, cápsulas, etc).

Freeman [FREE87a] considera esta área ainda pouco explorada, merecendo uma atenção especial nos próximos anos.

Finalmente, a atividade de composição de uma aplicação a partir de componentes pré-fabricados é, por nós, considerada a principal etapa no processo de reutilização por composição. A dificuldade em reutilizar componentes de software em uma nova aplicação é diretamente proporcional à dificuldade em interconectar os diferentes componentes envolvidos. O suporte à composição de componentes pode ser feito de duas maneiras: através do uso de ferramentas interativas (ex. Vista [MEY 91], LegoShell [DRUM89], Teamworks [STAD91], Gluons [PINT91]) ou de linguagens de composição (ex. Linguagens para Interconexão de Módulos [PRIE86] [XEXE91] e Linguagens de Programação Orientada a Objetos [GOLD84] [COX86]).

As Linguagens para Interconexão de Módulos propõem-se a solucionar problemas relacionados ao desenvolvimento de software em larga escala, onde a implementação de diferentes partes do projeto fica sob a responsabilidade de diferentes equipes. Neste caso, uma aplicação é vista como um conjunto de módulos que cooperam através da troca de dados e controle, denominados *recursos*. A definição da estrutura de uma aplicação em módulos e a troca de recursos entre eles são descritos através de sentenças da linguagem. Exemplos de linguagens deste tipo podem ser encontrados em [PRIE86].

As Linguagens de Programação Orientada a Objetos (LPOOs) têm a reutilização como centro no desenvolvimento de novas aplicações, provendo mecanismos para a organização e decomposição de sistemas em componentes bem encapsulados (i.e. objetos e classes) e mecanismos para a modificação e composição incremental de aplicações (i.e. herança, generalidade, ligação dinâmica) [TSIC88]. A programação orientada a objetos parte do princípio que classes e objetos pré-definidos podem ser reutilizados integralmente ou especializados de forma a compor uma nova aplicação. Pode-se dizer que o paradigma de troca de mensagens entre objetos comporta-se como o elemento de ligação que compõe uma aplicação em LPOOs.

A *figura 4* mostra, esquematicamente, os elementos típicos de um sistema que adota uma tecnologia de composição.

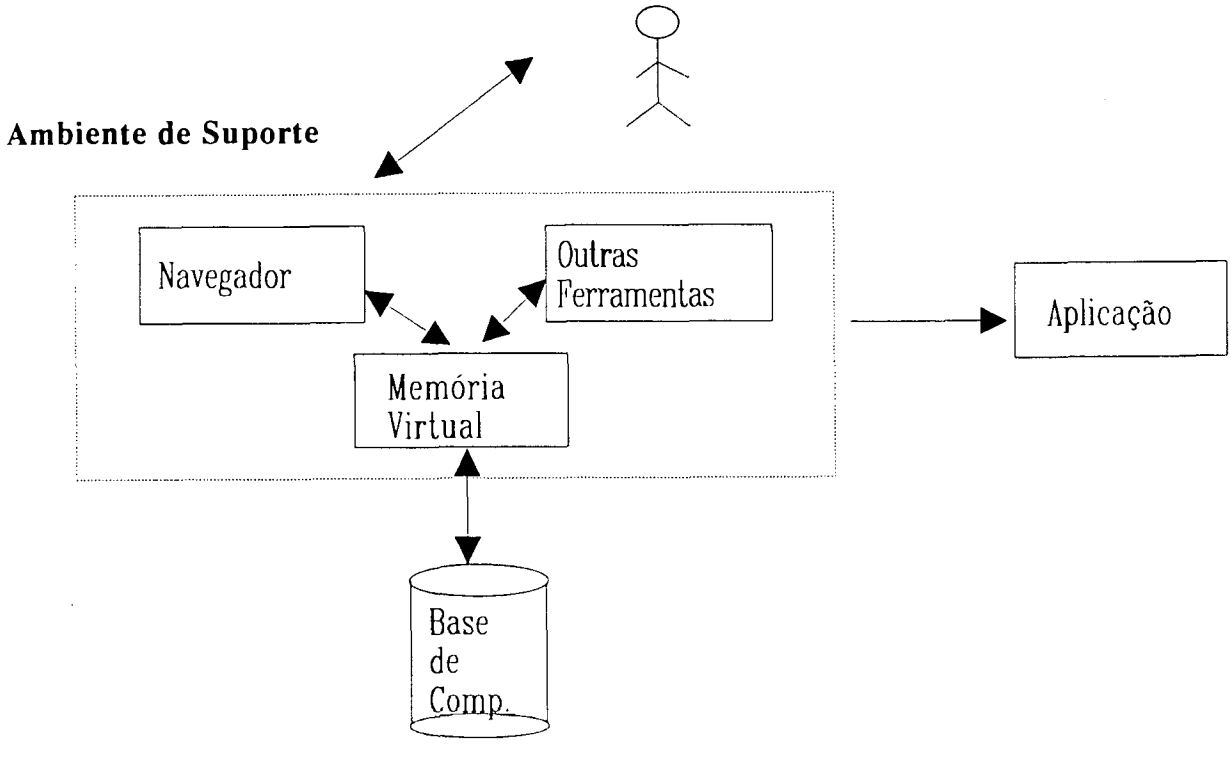


Fig. 4 - Elementos típicos de uma abordagem de composição

III.2.2.2. Tecnologias de Geração

Este grupo caracteriza-se pela geração de programas a partir de padrões não facilmente identificados após seu uso (ex. padrões de código, padrões em regras de transformações, etc) [BIGG89]. Sistemas deste tipo escondem em sua estrutura padrões que vão sendo reutilizados segundo os requisitos específicos da aplicação. Ao final do processo, é possível identificar apenas uma certa semelhança na arquitetura final dos programas gerados.

De uma maneira geral, as tecnologias de geração se caracterizam por sua natureza declarativa, partindo-se de uma descrição do problema até se alcançar sua implementação, sendo este processo realizado de forma automática ou semi-automática. Trabalhos anteriores realizados na área conhecida como *programação automática* [BALZ85] podem ser vistos como percursores deste grupo de tecnologias.

Exemplos de tecnologias de geração são geradores de aplicações [LUKE86], linguagens de quarta geração [MART85] e sistemas baseados em transformações [CHEA89].

Geradores de aplicação são pacotes de software projetados para auxiliar o usuário na construção de aplicações de um certo domínio [LUKE86]. Este tipo de ferramenta transforma especificações de um problema em programas aplicativos. As especificações podem ser definidas a partir de um diálogo interativo com o usuário, elaboração de diagramas, ou ainda utilizando-se linguagens específicas, tais como linguagens de quarta-geração, linguagens de muito alto nível ou linguagens dedicadas [CLEA88]. Uma arquitetura padrão para geradores de aplicação pode ser encontrada em [LUKE86].

Um exemplo de gerador de aplicação é o DCGS ("Dialogue-Code Generation System"), desenvolvido nos Laboratórios Bell, em 1983 [CLEA88]. Devido ao sucesso na implantação deste primeiro gerador de aplicação nos Laboratórios Bell, foi desenvolvida uma ferramenta genérica para a construção de outros geradores de aplicação, Stage, facilitando o uso desta tecnologia em diversos projetos.

Enquanto que componentes, em tecnologias de composição, são livremente compostos em diferentes combinações, em geradores de aplicação, as combinações são mais restritas, sendo determinadas pelo sistema que utiliza, apenas, certos padrões de combinação. Neste caso, o conhecimento do domínio da aplicação é dito estar *embutido* no sistema.

Sistemas baseados em transformações sugerem a transformação sucessiva de programas abstratos (i.e. não executáveis) até se atingir programas concretos (i.e. executáveis) [CHEA89]. A partir da variação das transformações envolvidas durante o processo, é possível gerar uma família de programas bastante distintos, apesar de gerados de uma mesma fonte.

Um exemplo de sistema baseado em transformações é o sistema PARIS [KATZ89], desenvolvido no MCC, onde *esquemas parcialmente interpretados* são reutilizados. Esquemas parcialmente interpretados são programas com partes abstratas, ou ainda indefinidas. Diferentes interpretações destes esquemas geram diferentes programas. PARIS conta, ainda, com um provador de teoremas ("Boyer-Moore Theorem Prover") para auxiliar no processo de interpretação de esquemas.

A *figura* 5 mostra, esquematicamente, elementos típicos de um sistema de geração.

III.2.3. Avaliação dos Resultados

Quanto à avaliação dos resultados obtidos através da adoção de uma estratégia de reutilização no processo de desenvolvimento de software, infelizmente, observamos a escassez de experiências práticas, apresentando resultados quantitativos sobre reutilização de software [SELB88] [SELB89], determinando os fatores que influenciam este processo [PRIE85] [ARTH85], ou sugerindo processos e meios para controlar a avaliação dos resultados obtidos.

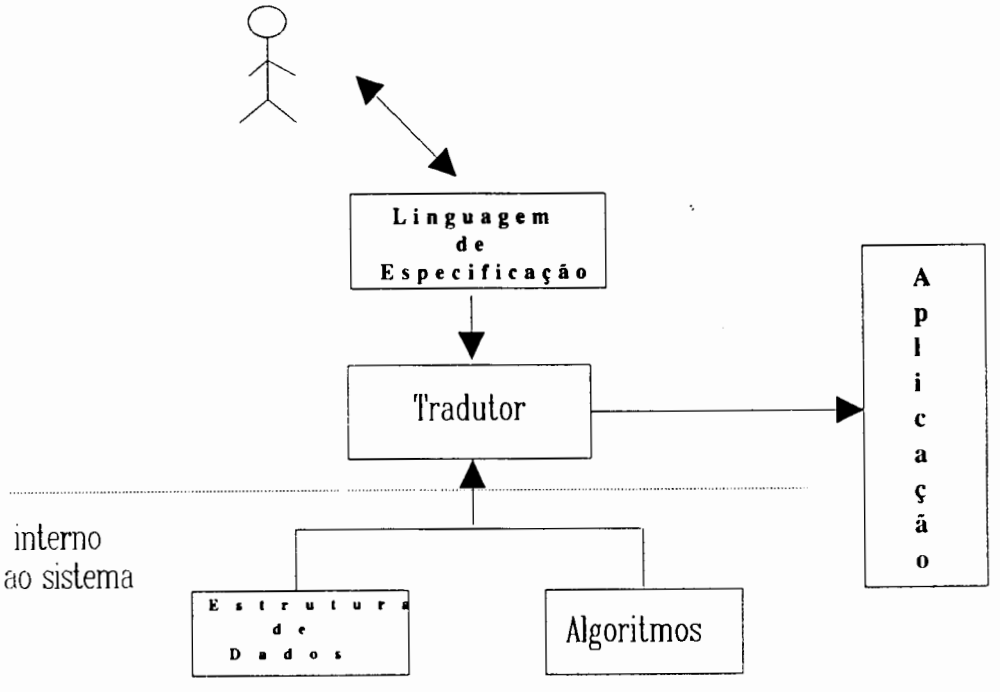


Fig. 5 - Elementos típicos de uma abordagem de geração

Prieto-Diaz [PRIE85] é um dos poucos autores que tentam identificar os fatores que influenciam, diretamente, o esforço gasto na tentativa de reutilização. Ele sugere como principais fatores:

a) *Tamanho do programa*: quanto maior é o programa, maior é a chance dele se tornar complexo e específico, dificultando, assim, sua compreensão e/ou modificação, reduzindo suas possibilidades de reutilização;

b) *Estrutura do programa*: do mesmo modo, a estrutura de um programa influi, diretamente, na sua compreensão e/ou modificação, podendo reduzir sua chance de ser reutilizado;

c) *Linguagem de programação*: bibliotecas de componentes são, tipicamente, formadas por componentes escritos em diferentes linguagens de programação. Ao tentarmos reutilizar estes componentes, é possível que tenhamos que converter programas escritos em uma linguagem para outra. Este esforço de conversão, deve ser levado em conta quando calculamos o esforço de reutilização;

d) *Documentação*: a insuficiência de documentação, tanto interna (i.e. comentários) como externa (i.e. especificação, documentos de projeto, etc), de programas pode impossibilitar sua reutilização;

e) *Agente da reutilização*: quanto mais experiente for o agente de reutilização, tanto na área de aplicação específica quanto nos aspectos computacionais, mais facilmente ele será capaz de compreender e/ou modificar componentes e, portanto, reutilizá-los.

Outros fatores que podem influenciar o esforço de reutilização de objetos são seu grau de generalidade e independência, tanto do ponto de vista de hardware quanto do software [ARTH85]. Quanto mais genérico for o objeto de reutilização, maiores serão as chances dele ser reutilizado em uma aplicação diferente daquela para a qual ele foi, originalmente, criado. Entretanto, a generalidade de um componente não deve chegar ao ponto de influir em seu desempenho e eficiência. Da mesma forma, quanto mais independente de hardware e de software for o objeto, maiores serão as chances dele ser reaproveitado em diferentes plataformas.

Resultados obtidos em análise feita por Selby [SELB88] [SELB89] sobre 25 sistemas de software de um ambiente de produção da NASA, onde foi constatada uma média de 32% de reutilização, nos dão uma boa noção sobre a possível influência desses fatores no processo de reutilização. Selby constata que módulos totalmente reutilizados são pequenos, bem documentados, com interface simples e com poucos parâmetros de entrada e saída. Ele afirma que o sucesso de reutilização neste contexto deve-se, em parte, ao baixo índice de alterações relacionadas à mão-de-obra envolvida nos projetos e ao tipo de domínio de aplicação (controle de aeronaves espaciais) que possui um conjunto de algoritmos e métodos de processamento bastante conhecidos e bem definidos, facilitando bastante a reutilização de software.

Woodfield et. al. [WOOD87] observam a incapacidade de desenvolvedores de software, não treinados em reutilização, em identificar componentes potenciais para o atendimento de requisitos de uma nova aplicação, a partir de uma experiência realizada na universidade de Brigham Young. Observou-se, ainda, uma certa dificuldade das pessoas em determinar o esforço necessário para modificação de objetos reutilizáveis, subestimando-o na maioria das vezes.

Em estudo preliminar sobre alguns dos atributos de qualidade que devem ser incorporados em produtos de software [WERN88c], identificamos que um produto de software deve ser flexível o suficiente para permitir sua reutilização em uma outra situação diferente daquela para a qual foi, inicialmente, produzido, sendo necessário avaliar os seguintes atributos:

a) *coesão para reutilização* (i.e. a força que mantém unidos os elementos que formam um produto de software): produtos de software reutilizáveis devem ter uma forte coesão;

b) *acoplamento para reutilização* (i.e. o grau de interdependência entre produtos de software): produtos de software reutilizáveis devem ter um baixo acoplamento;

c) *abrangência* (i.e. nível de generalização): produtos de software reutilizáveis não devem ser nem muito restritos, nem muito genéricos;

d) isolamento (i.e. grau de necessidade do conhecimento sobre detalhes técnicos para utilização do produto): produtos de software reutilizáveis devem ser o mais isolado possível.

Conforme mencionado anteriormente, muito poucos trabalhos são encontrados na literatura que indicam os principais fatores que influenciam o processo de reutilização, entretanto, consideramos de extrema importância termos uma lista completa sobre esses fatores, de forma a podermos melhor direcionar o desenvolvimento de componentes de software reutilizáveis. Semelhante situação ocorreu na identificação dos primeiros atributos de qualidade de software há alguns anos. Naquela época, poucos autores se preocupavam em determinar, exatamente, quais os atributos de qualidade que um produto de software devia ter [BOEH78] [ARTH85] [ROCH85]. Entretanto, o controle da qualidade em software se mostrou, extremamente, importante na consolidação dos conhecimentos sobre o processo de desenvolvimento de software.

III.3. Vantagens e Problemas encontrados na Adoção de uma Estratégia de Reutilização de Software

A adoção de uma estratégia de reutilização no desenvolvimento de software, em tese, pode trazer muitas vantagens, tais como [BIGG89] [AGRE88] [TRAC88a] [TRAC88c]:

- melhores índices de produtividade no desenvolvimento de software;

- produtos de melhor qualidade, mais confiáveis, consistentes e padronizados;
- redução dos custos e tempo envolvidos no desenvolvimento de software, e
- maior flexibilidade na estrutura do software produzido, facilitando, assim, sua manutenção e/ou evolução.

Observa-se, entretanto, que muitas das promessas feitas ao longo dos anos sobre o tema reutilização no processo de desenvolvimento de software, ainda hoje, não puderam ser, completamente, cumpridas [BIGG89] [TRAC88b]. Muitos autores tentam determinar as possíveis barreiras que estariam impedindo a efetiva implantação de uma estratégia de reutilização no desenvolvimento de software [TRAC88c] [MEYE89] [LUBA88]. Alguns associam estas barreiras a fatores técnicos como a insuficiência de ferramentas adequadas ao suporte da reutilização (ferramentas de busca de componentes adequados, ferramentas de composição, etc), além da baixa qualidade dos componentes, atualmente, disponíveis para reutilização.

Outros consideram uma série de fatores psicológicos, sociológicos e econômicos que impedem a reutilização [BIGG89] [TRAC88b]. Dentre os fatores mais citados temos: o alto custo inicial para o desenvolvimento de componentes reutilizáveis, a não credibilidade em componentes não desenvolvidos pelo próprio programador e a falta de direções claras quanto à estratégia mais adequada a ser adotada. Outros autores, ainda, mencionam aspectos políticos e legais envolvidos, tais como diversas leis protecionistas e políticas contratuais do governo [GIBB90b] [LUBA88].

Ao se decidir adotar uma estratégia de reutilização no desenvolvimento de software, deve-se estar consciente do custo e tempo adicionais envolvidos no desenvolvimento dos primeiros componentes reutilizáveis, além da necessidade de uma mudança de postura relacionada à maneira como o software é, tradicionalmente,

desenvolvido. A seguir, discutimos os aspectos relacionados à relação custo/benefício da reutilização e à definição de um ciclo de desenvolvimento adequado ao processo de reutilização.

III.3.1. Relação Custo/Benefício da Reutilização

Se, por um lado questionamos o fato de que poucas empresas estão, hoje, fazendo esforços, no sentido de introduzir uma estratégia de reutilização em seu processo de desenvolvimento de software, por outro lado devemos estar conscientes dos custos envolvidos na implantação de tal estratégia. O custo e o tempo gastos no treinamento de pessoal e no desenvolvimento de ferramentas de suporte adequadas e de componentes reutilizáveis só serão compensados se o mercado para reutilização desses produtos, em outros projetos, puder ser definido ou, pelo menos, parcialmente caracterizado [BOLL90].

Um gerente de projetos de desenvolvimento de software interessado em fazer investimentos para reutilização depara-se com os seguintes problemas [BOLL90]:

- investimento para reutilização, ao contrário do que se imagina, representa um custo extra. Onde obter financiamento? Com o cliente ou na empresa?
- investimento para reutilização é custo opcional (i.e. não é necessário para o término bem sucedido do projeto, na verdade, pode até atrasar a entrega do produto final) uma constatação, também, diferente do que se imagina.

Logo, a decisão em adotar uma estratégia de reutilização, por parte de um gerente de projeto, deve poder estar baseada em uma análise custo/benefício positiva.

Bollinger e Pfleeger [BOLL90] desenvolveram um modelo para estimar custos relacionados à reutilização, baseado no conceito de *domínios de compartilhamento de custos* (CSDs - "Cost-Sharing Domains"). Um CSD é um grupo de projetos, presentes ou futuros, cujos custos são somados e tratados como um único valor. Do ponto de vista de reutilização, um CSD representa um conjunto de grupos que estão interessados na reutilização de um dado conjunto de componentes reutilizáveis. O capital inicial para construção desses produtos reutilizáveis é financiado por uma entidade, denominada um *banco CSD*. A alocação de financiamento para um CSD é um mecanismo similar a um empréstimo, que é pago mediante a entrega do produto reutilizável ao banco CSD (i.e. o banco é o proprietário do produto). Bollinger e Pfleeger alegam que esta visão sobre reutilização (i.e. como um investimento financeiro) é bastante interessante mas que, raramente, foi levada em consideração, anteriormente.

Um outro aspecto a ser levado em consideração diz respeito à necessidade de se criar uma infra-estrutura adequada para dar suporte à reutilização. A criação desta infra-estrutura pode envolver tanto o desenvolvimento ou compra de ferramentas de suporte adequadas à reutilização como o treinamento de uma equipe especializada para o acompanhamento das atividades relacionadas à reutilização de software.

Prieto-Diaz [PRIE91] identifica como elementos necessários para uma infra-estrutura adequada de reutilização os seguintes:

1) um *grupo de suporte gerencial* que dá incentivos, financiamentos e regras para a reutilização;
2) *ferramentas de suporte* adequadas à reutilização (ex. um sistema de apoio a uma biblioteca de software);

3) um *grupo para identificação e qualificação*, responsável pela qualidade dos componentes reutilizáveis, identificando, ainda, áreas potenciais para reutilização, coletando e avaliando novos candidatos ao acervo de produtos reutilizáveis;

4) um *grupo de manutenção* que mantém e atualiza componentes reutilizáveis;

5) um *grupo de desenvolvimento* que cria novos componentes reutilizáveis;

6) um *grupo de suporte a desenvolvedores de sistemas* (re-usuários) que fornece assistência técnica e treinamento.

A *figura 6* mostra a participação desses grupos no processo de reutilização de software.

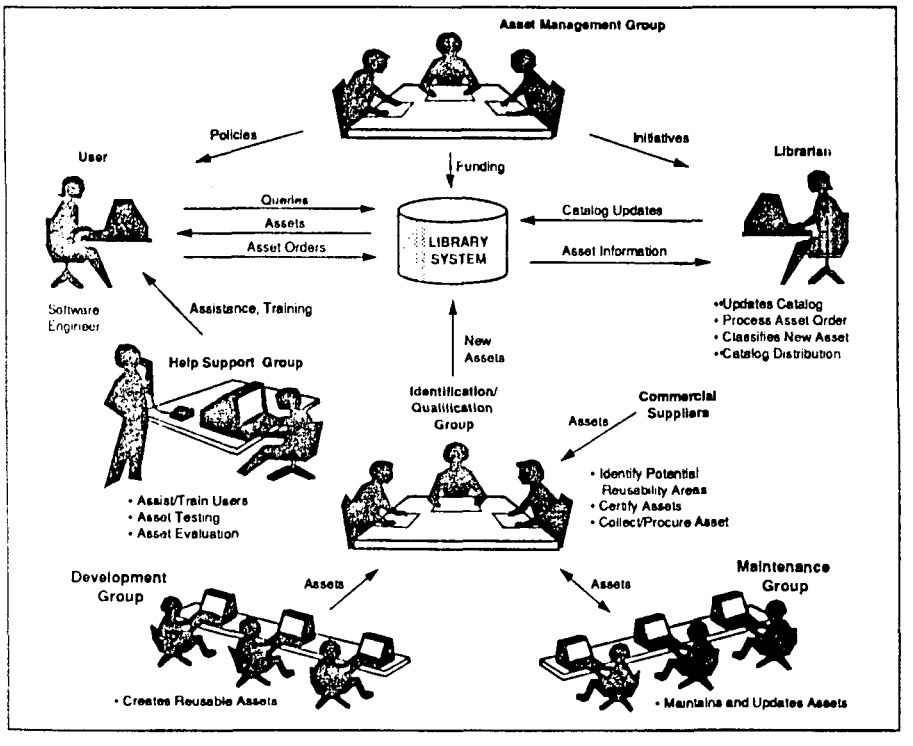


Fig. 6 - Grupos de suporte à reutilização [PRIE91]

As atividades executadas por cada um desses grupos e questões relacionadas à formação da equipe de trabalho passam por uma discussão semelhante à criação dos

primeiros grupos para avaliação da qualidade de software [PERR83] [ARTH85]. Os primeiros grupos de controle da qualidade em departamentos de processamento de dados surgiram em 1968, sendo as funções gerais desses grupos [PERR83]:

- garantir o cumprimento de métodos aprovados para a construção de aplicações;
- assegurar um nível razoável de desempenho nos procedimentos adotados, e
- assegurar que os objetivos da organização e dos usuários do sistema são satisfeitos.

Similarmente, os grupos de suporte à reutilização devem garantir o cumprimento de métodos e técnicas de reutilização recomendados para a construção de aplicações, assegurar um nível razoável de desempenho na adoção de tal estratégia e assegurar que os objetivos da organização e dos usuários são satisfeitos.

De acordo com Perry [PERR83] existem quatro alternativas para a formação de grupos deste tipo:

1) *Grupos de trabalho*: formados a cada projeto, permitem a indicação de pessoas especializadas em cada tipo de projeto. Esta alternativa é interessante na medida em que pode servir como atividade de treinamento de desenvolvedores, entretanto, pode influenciar a continuidade das atividades e no estabelecimento de procedimentos padrão, devido aos elementos do grupo estarem envolvidos com outras atividades e a tendência de se estabelecer procedimentos próprios em cada grupo de trabalho;

2) *Grupos permanentes em tempo integral*: garante a continuidade do trabalho e o estabelecimento de procedimentos padrão. Por outro lado, a equipe formada perde aos poucos a prática em outras atividades de desenvolvimento e nem sempre possue o conhecimento específico a uma certa área de aplicação;

3) *Comitês permanentes*: Pode ser vista como uma generalização da alternativa 1), criando-se comitês permanentes para a avaliação de projetos de uma mesma área de aplicação, garantindo a inclusão de especialistas e o estabelecimento de procedimentos padrão. A questão da divisão do tempo com outras atividades permanece e, de certa forma, um comitê não pode ser autoritário como um grupo em tempo integral.

4) *Combinação entre tempo integral e tempo parcial*: esta alternativa combina funcionários trabalhando em regime de tempo integral e funcionários trabalhando em regime de tempo parcial. Os membros em tempo integral garantem a continuidade do trabalho, enquanto que os membros em tempo parcial acrescentam conhecimento específico sobre certas áreas de aplicação. Os problemas relacionados à divisão do tempo dos membros em tempo parcial e a perda de prática em outras atividades de desenvolvimento dos membros em tempo integral permanecem. Entretanto, este esquema de trabalho tem se mostrado bastante interessante na formação de equipes.

Prieto-Diaz menciona, ainda em [PRIE91], a importância de incentivos financeiros, necessários ao sucesso de uma estratégia de reutilização. Em experiência realizada no Serviço de Dados da GTE [PRIE91], valores de 50 a 100 dólares eram pagos ao autor de cada programa aceito pelo grupo de identificação e qualificação. Além disso, "royalties" eram pagos cada vez que este programa era reutilizado em um novo sistema. Similarmente, gerentes de projeto eram incentivados, através de aumentos de orçamento e promoções, quando seus projetos obtinham um grande percentual de componentes reutilizados. Re-usuários eram promovidos ou ganhavam prêmios se fizessem uso de componentes reutilizáveis. Desta forma, toda uma

máquina para reutilização era alimentada através de incentivos financeiros e promocionais.

Mesmo com todos esses gastos extras, na criação de uma infra-estrutura de suporte à reutilização, treinamento de pessoal e incentivos financeiros, Prieto-Diaz afirma que uma economia de 1 milhão e meio de dólares foi alcançada no primeiro ano da experiência e uma economia estimada em 10 milhões de dólares é esperada para o final de 5 anos.

III.3.2. Ciclos de Desenvolvimento para Reutilização

Existem, hoje, algumas discussões sobre a adequação do modelo clássico de ciclo de desenvolvimento de software, mais conhecido como o *modelo em cascata* ("the waterfall model"), em desenvolvimentos que tenham como base a reutilização [TSIC89] [TRAC90].

Uma das críticas a este modelo baseia-se na idéia de que ele pressupõe uma perfeita compreensão sobre a aplicação, indicando o desenvolvimento do software como uma sequência bem estabelecida de etapas, com produtos bem definidos, além de uma sólida plataforma de hardware e software onde o sistema é desenvolvido [TSIC89]. Durante o ciclo, presume-se que os requisitos da aplicação permanecem inalterados e o desenvolvimento é feito, tipicamente, segundo uma abordagem *de cima para baixo* ("top-down").

Dado que existem, hoje, diversas aplicações, ditas *abertas* ("open-systems") [TSIC89], cuja plataforma de desenvolvimento, tanto de hardware como de software, está em aberto, geograficamente distribuída, cujo domínio não é muito bem conhecido com requisitos que mudam muito rapidamente, não é possivel assumir que

ao final de um longo ciclo de desenvolvimento, ainda, sejamos capazes de atender os requisitos iniciais da aplicação. Neste caso, é preciso construir o software rapidamente, com baixo custo e flexibilidade suficiente para acompanhar sua evolução e permitir seu desenvolvimento contínuo.

A adoção de uma estratégia de reutilização no desenvolvimento deste tipo de software pode reduzir, consideravelmente, o tempo e o custo de desenvolvimento, na medida em que produtos de software similares podem ser totalmente reutilizados ou adaptados nas diversas fases do desenvolvimento.

O desenvolvimento de software baseado em uma estratégia de cima para baixo inibe, de certa forma, a reutilização de produtos de software já existentes, na medida em que tende a estabelecer requisitos muito específicos, dificilmente encontrados em produtos já disponíveis. Para reutilizar produtos de software, é preciso, em algum instante, observar os produtos existentes, de forma a poder reutilizar o máximo possível com esforço reduzido (i.e. utilizando uma abordagem *de baixo para cima*) [TSIC89]. Assim sendo, são propostas adaptações ao modelo de desenvolvimento clássico para dar suporte ao desenvolvimento de aplicações abertas, baseando-se na reutilização de software, onde:

- a análise de requisitos é desenfatizada, na medida em que não se tem requisitos muitos claros;
- a especificação é feita em termos de componentes e da descrição de suas propriedades, assumindo a existência de uma base de software com componentes reutilizáveis, devidamente documentados, e um mecanismo de seleção capaz de localizar componentes relevantes à aplicação em questão (i.e. abordagem de composição);

- o projeto dá ênfase à composição estrutural e não ao refinamento sucessivo (i.e. *de baixo para cima* ao invés *de cima para baixo*);
- a programação é desenfatizada, na medida em que o trabalho de construção deve minimizar o esforço de programação, enfatizando-se o aproveitamento de componentes já existentes;
- o esforço de integração é, ainda, necessário e bastante importante neste contexto;
- a monitoração do comportamento da aplicação continua sendo feita através de procedimentos de teste e validação, e
- o desenvolvimento segue continuamente na medida em que adaptações vão se tornando necessárias.

Um modelo de ciclo de vida, baseado neste tipo de proposta, foi definido para dar suporte ao ambiente de desenvolvimento de aplicações orientadas a objetos, ITHACA, onde as seguintes etapas foram estabelecidas [PROF89] [FUGI91]:

- *engenheiros de aplicação* desenvolvem classes e "grades de aplicação" (i.e. uma espécie de "template" que guarda informações sobre a aplicação) referentes a um certo domínio de aplicação, armazenando-as em uma Base de Informações de Software (BIS);
- *desenvolvedores de aplicação* selecionam uma grade de aplicação a partir de uma breve descrição sobre os requisitos da aplicação, procurando e navegando através da BIS;
- a grade de aplicação guia a coleta de requisitos e a especificação, segundo especificações e projetos genéricos, pré-existentes na BIS, direcionando a seleção de classes reutilizáveis;
- As classes selecionadas são modificadas de forma incremental, utilizando os recursos de herança e fornecendo parâmetros adequados;

- As classes selecionadas são compostas a partir de um *script* que especifica a cooperação entre os objetos, e
- O comportamento da aplicação é, então, monitorado através de testes e procedimentos de validação e o desenvolvimento segue, continuamente, adaptando a aplicação segundo modificações de seus requisitos.

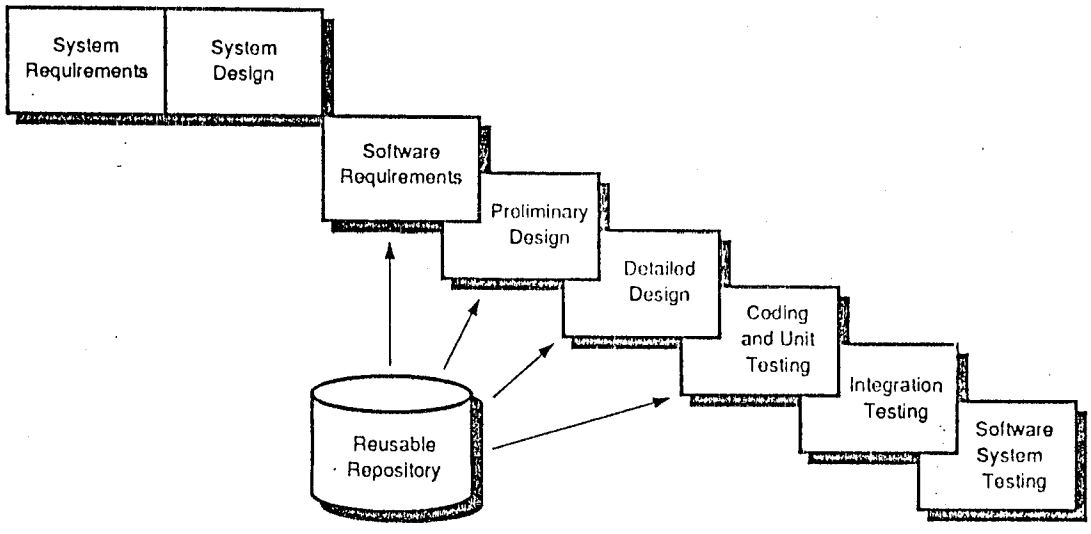
Bersoff, em [BERS91], apresenta um modelo de componentes de software reutilizáveis como uma dentre outras tentativas (ex. diferentes modos de prototipagem e síntese automática de software) na redução dos custos e tempo envolvidos no modelo de ciclo de desenvolvimento tradicional. Em sua descrição deste novo modelo de ciclo de vida, as tradicionais fases de requisitos, projeto preliminar, projeto detalhado e codificação/teste têm o suporte de um repositório de produtos reutilizáveis, podendo o desenvolvedor, a cada uma dessas fases, contar com produtos de software reutilizáveis. Bersoff, entretanto, nada diz sobre as novas atividades, introduzidas em cada uma dessas fases, relacionadas à adoção de uma estratégia de reutilização (i.e. seleção, entendimento, adaptação e composição).

Note que este modelo de reutilização reflete uma abordagem de composição, sendo a abordagem de geração melhor representada pelo modelo apresentado por Davis como *síntese automática de software*. A *figura 7* mostra cada um desses modelos.

Tracz, em [TRAC90], considera, apenas, necessária a inclusão de mais uma fase no modelo tradicional em cascata. Esta nova fase, denominada *análise de domínio*, é adicionada, exclusivamente, para dar suporte explícito à reutilização. Como uma generalização da análise de requisitos, ao invés de analisar os requisitos específicos de uma determinada aplicação, os requisitos de uma aplicação genérica em um certo domínio são determinados. Novamente, o enfoque a ser dado nas demais

fases de desenvolvimento de software ao se adotar uma estratégia de reutilização não é mencionado.

a)



b)

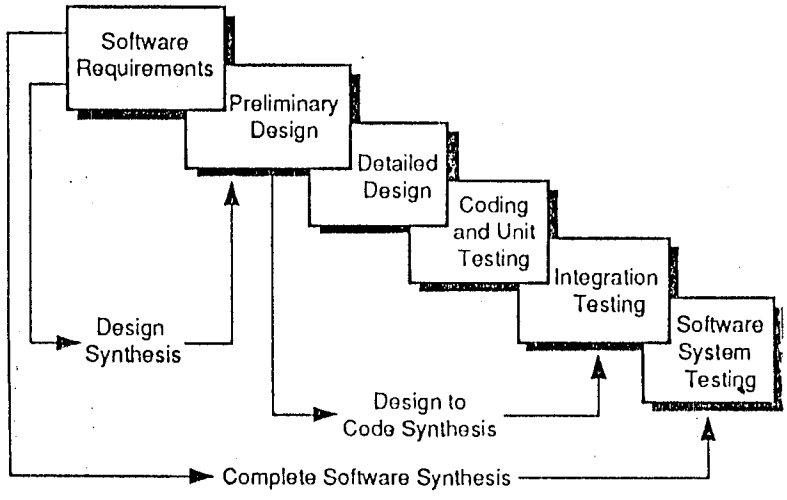


Fig. 7 - Modelos de ciclo de vida [BERS91]: a) componentes de Software reutilizáveis e b) síntese automática.

Seja qual for o modelo de ciclo de vida adotado, é certo que este deve estar adaptado à realidade do ambiente de desenvolvimento em questão e às tecnologias e ferramentas de apoio à reutilização disponíveis.

III.4. Reutilização no Contexto Científico

Bibliotecas de rotinas matemáticas são referenciadas na literatura como um dos poucos exemplos concretos de componentes reutilizáveis [BIGG89] [CHEA89]. Acredita-se que este sucesso esteja associado às seguintes questões [BIGG89]:

- o domínio em questão é bastante restrito;
- o domínio é bastante entendido, existindo uma plataforma matemática bem definida, e
- a tecnologia usada é bastante estática, crescendo e evoluindo lentamente de maneira que a tecnologia existente permanece inalterada (compatibilidade vertical da tecnologia).

A restrição da pesquisa sobre reutilização em domínios de aplicação específicos parece ser, atualmente, uma das poucas maneiras para minimizar os problemas associados a esta nova tecnologia [ISCO88] [PRIE90] [NEIG91]. A princípio, parece-nos ser ineficiente, senão impossível, construir-se uma biblioteca única, genérica, contendo os diversos componentes específicos a cada tipo de aplicação.

Um domínio bastante entendido, cuja plataforma de termos e conceitos é bem definida, oferece uma boa base para a determinação de componentes básicos, capazes de gerar componentes mais complexos. Tal formalismo, entretanto, inexiste em

diversas outras áreas de aplicação, muitas vezes, devido a sua própria natureza (ex. aplicações em inteligência artificial) [HALL87].

Quanto à tecnologia usada em computação numérica ser estática e verticalmente compatível, pode-se citar um exemplo recente onde esta afirmação não é constatada. Com a introdução de máquinas paralelas, muitas das técnicas e métodos numéricos até então utilizados mostraram-se impróprios para o aproveitamento eficiente desta nova arquitetura, sendo necessária uma total reformulação de diversas rotinas já estabelecidas [ALMA89] [ALLE84].

Um fator, talvez, determinante no ambiente científico é a existência de uma forte corrente a favor do estabelecimento de padrões, o que permite uma rápida aceitação da comunidade de padrões impostos, tornando-os reutilizadores em potencial.

Um outro fator que impulsiona a prática de reutilização no meio científico é a complexidade das aplicações desenvolvidas (conforme discutido no capítulo II), impedindo que a cada nova aplicação desenvolva-se toda uma coleção de operações numéricas básicas, a um custo altíssimo. Portanto, tradicionalmente, físicos, metemáticos, etc, recorrem a bibliotecas de programas que os auxiliam em suas atividades de desenvolvimento de aplicações.

É inegável que o sucesso na criação de bibliotecas no ambiente científico tenha como principal fator o uso de uma linguagem de programação única (i.e. FORTRAN) [DICK71] [BELL85] [TRAU84] [METC91].

Inicialmente, bibliotecas de rotinas caracterizavam-se por conter rotinas puramente algébricas [CODY71] [KUKI71]. No decorrer dos anos, bibliotecas mais

especializadas foram surgindo, contendo rotinas específicas a uma determinada área de aplicação [SAMM71] [CERN89c]. Hoje, é possível dispor de bibliotecas de renome tais como IMSL [RICE83] e NAG [FORD79], em que toda uma infra-estrutura de suporte é montada para garantir a qualidade de programas, sua documentação, gerência, manutenção e divulgação [FORD79] [BRUN91].

Muitas bibliotecas de software científico são obtidas, por laboratórios ou instituições de ensino e pesquisa, através de contratos de utilização totalmente gratuitos ou através do pagamento de pequenas taxas para a cobertura de despesas com material. Tal facilidade na obtenção de componentes reutilizáveis é, certamente, um outro fator positivo na decisão a ser tomada quanto à utilização de software previamente construído. Entretanto, muitas dessas bibliotecas podem não ter a qualidade esperada, sendo as vezes necessária uma avaliação detalhada de seus programas [CODY71].

Recorrer às bibliotecas de renome pode ser a solução, porém, além do custo de aquisição envolvido, é possível que os algoritmos utilizados estejam obsoletos em relação ao estado da arte, na medida em que para se manter a qualidade dessas bibliotecas pode ser necessário privilegiar programas baseados em métodos numéricos bem estabelecidos aos programas baseados em métodos mais modernos e inovadores [BATI71].

A credibilidade de bibliotecas está muitas vezes associada à sua fonte de recursos (i.e. fabricantes, grupos de usuários, centros de computação, disseminação informal, programas descritos em livros texto, publicações, etc) [RICE71c] [LUBA88]. Outros fatores extremamente importantes, que podem determinar o uso de bibliotecas de software, dizem respeito à qualidade de sua documentação e sua estrutura, pois esta pode facilitar sua compreensão e modificação [RICE83].

Bibliotecas de software são, normalmente, encontradas na literatura como exemplo de tecnologia de composição para reutilização [BURT87] [PRIE87] [HORO84]. Na seção III.2.2, entretanto, fizemos questão de ressaltar o fato de que, para nós, a disponibilidade, pura e simples, de uma coleção de componentes de software não é condição suficiente para caracterizarmos uma abordagem de composição para reutilização. É preciso existir algum tipo de mecanismo capaz de compor uma aplicação a partir desses componentes.

O uso tradicional das bibliotecas científicas assume o desenvolvimento de um programa principal em FORTRAN que se encarrega de ativar as rotinas desejadas, misturando-se aspectos convencionais de programação com aspectos mais específicos de composição (i.e. verificação de parâmetros formais, conversões entre estruturas de dados, encapsulamento, etc), nem sempre bem tratados pela linguagem. A atual versão de FORTRAN 90 inclui uma série de mecanismos interessantes do ponto de vista de composição (ex. módulos, blocos de interface, tratamento de argumentos de rotinas, etc) [METC91].

Um conjunto de ferramentas interativas de suporte às atividades relacionadas à escolha, entendimento, adaptação e composição de componentes de uma biblioteca pode reduzir, drasticamente, o esforço atual para utilização dessas bibliotecas. No capítulo V, apresentamos um exemplo de ambiente interativo de composição de aplicações científicas, baseado em conceitos de orientação a objetos e na utilização de uma biblioteca de componentes, que satisfaz os requisitos de uma abordagem de composição para reutilização de software.

Outras ferramentas de suporte à reutilização de código encontradas em determinadas áreas de aplicação no contexto científico pertencem ao grupo de

tecnologias de geração. São geradores de aplicação [CLEA88] [XEXE91] e linguagens de muito alto nível, de quarta-geração ou dedicadas [WERN89a] [SCHW86] [MART85]. Estas ferramentas partem da especificação de uma aplicação e são capazes de gerar programas executáveis (ver seção III.2.2 para maiores detalhes).

Devido ao seu uso, muitas vezes, restrito a determinadas aplicações, estas ferramentas não são amplamente utilizadas como as bibliotecas de rotinas. Ferramentas deste tipo, entretanto, estão muito mais próximas dos conceitos e abstrações entendidos por especialistas de sua área de aplicação do que linguagens de programação convencionais, em que o conhecimento sobre aspectos de programação é fundamental. Assim sendo, este tipo de ferramentas é, facilmente, utilizado por usuários leigos em computação.

PROTRAN é um exemplo de linguagem de muito alto nível introduzida pela IMSL para facilitar o uso de suas rotinas e, simultaneamente, melhorar a credibilidade e legibilidade de programas [RICE83]. Sentenças em PROTRAN são orientadas à solução de problemas matemáticos e o sistema se encarrega de verificar questões computacionais tais como a verificação de dimensões de vetores e matrizes, a chamada de rotinas da biblioteca e a verificação de seus argumentos.

O sistema PROTRAN foi construído em FORTRAN. Um programa PROTRAN pode conter sentenças FORTRAN e PROTRAN, sendo as sentenças PROTRAN identificadas por um sinal de dolar ("$"). O tradutor PROTRAN, portanto, funciona como um pré-processador do FORTRAN. Um programa em PROTRAN é muito menor que um programa FORTRAN. A *figura 8* mostra como exemplo um mesmo programa escrito nas duas linguagens.

Apesar do uso intensivo de bibliotecas de rotinas e o aparecimento de algumas ferramentas úteis à reutilização de software (basicamente, reutilização de código), o termo *reutilização* raramente é utilizado no meio científico. Logo, por um lado, temos um quadro de intensa pesquisa sobre reutilização de software na ciência da computação, sem muitas experiência práticas sobre sua adequação nas diversas áreas de aplicação. Por outro lado, temos o ambiente científico que, apesar de referenciado como um dos poucos exemplos onde tem sucesso a reutilização de código através do uso de bibliotecas de rotinas, parece desconhecer, totalmente, as questões teóricas relacionadas ao tema.

Os diversos desafios, hoje impostos, para a solução dos problemas encontrados no desenvolvimento de software científico, conforme descrito no capítulo II, são suficientes para deter a atenção dos poucos especialistas em computação envolvidos neste domínio de aplicação. Questionamos, entretanto, se uma possível solução para minimização destes problemas não poderia resultar da adoção de uma estratégia de reutilização no desenvolvimento de software científico, criando-se uma infra-estrutura para incentivo à reutilização e fornecendo-se ferramentas de suporte adequadas. Os capítulos que se seguem descrevem duas experiências práticas no desenvolvimento de ferramentas de suporte à reutilização no contexto científico.

```
      $DECLARATIONS
      BAND MATRIX DE(N=50,1,1)
      VECTOR X(N=50), Y(N=50) $
C
C                          SET UP PROBLEM TO BE SOLVED
      N = 20
      H = 1./(N+1)
      $ASSIGN  X(I) = I*H
C                          DEFINE MATRIX FOR DISCRETE APPROXIMATION
      $ASSIGN  DE(I,J) =  1. - (IABS(I-J)-1)*(3. - H**2*COS(X(I))
C                     INITIALIZE SOLUTION Y TO BE THE RIGHT SIDE
      $ASSIGN  Y(I) = H**2 * ALOG(X(I)+4.) $
C                     ADD BOUNDARY CONDITION TERM AT X=1. TO
C                     THE LAST RIGHT SIDE
      Y(N) = Y(N) - 1.0
C                     SOLVE THE LINEAR SYSTEM AND PLOT RESULT
      $LINSYS  DE*Y = Y ; NOSAVE
      $PLOT Y ; VS X;  SYMBOL = 'Y' ; XRANGE = (0.,1.)
          TITLE = 'SOLUTION OF DIFFERENTIAL EQUATION - EXAMPLE 6.4'
      $END
```

```
                    FORTRAN VERSION

      REAL X(50)
            DECLARATIONS FOR IMSL LEQT1B
      REAL DE(50,3), Y(50), WORK(100)
C                 DECLARATIONS FOR LINPACK SGBCO AND SGBSL
      REAL DEL(4,50), YL(50),Z(50)
      INTEGER IPVT(50)
C
C             DECLARATIONS AND SET-UP FOR IMSL PLOT ROUTINE USPLO
      REAL RANGE(4)
      DATA RANGE/4*0.0/
      RANGE(2) = 1.0
C
C                    DISCRETIZE THE INTERVAL (0,1)
      N = 20
      H = 1./(N+1)
      DO 5 I = 1,N
    5    X(I) = I*H
C
C                    SET-UP AND SOLUTION BY IMSL LEQT1B
      DO 10  I = 1,N
          DE(I,1) = DE(I,3) = 1.0
          DE(I,2) = -2. + H**2*COS(X(I))
          Y(I)    =  H**2*ALOG(X(I)+4.)
   10 CONTINUE
      Y(N) = Y(N)-1.0
      CALL LEQT1B(DE,N,1,1,50,Y,1,50,0,WORK,IER)
      IF ( IER .NE. 0 )  PRINT 16
   16     FORMAT(//,25(3H **)/20X'LEQT1B SAYS MATRIX IS SINGULAR'//)
C
C               PLOT SOLUTION WITH IMSL LIBRARY ROUTINE
      CALL USPLO(X,Y,1,N,1,1,
     A           47HSOLUTION OF DIFFERENTIAL EQUATION - EXAMPLE 6.4, 47,
     B           1HX, 1, 1HY, 1, RANGE, 1HY, 1, IER)
C
C                   SET-UP AND SOLUTION BY LINPACK SGBCO AND SGBSL
      DO 110 I = 1,N
          DEL(4,I) = DEL(2,I) = 1.0
          DEL(3,I) = -2. + H**2*COS(X(I))
          YL(I)    = H**2*ALOG(X(I)+4.)
  110 CONTINUE
      YL(N) = YL(N) - 1.0
      CALL SGBCO(DEL,4,N,1,1,IPVT,COND,Z)
      IF(  1.+ COND .EQ. 1. )  PRINT 116
  116     FORMAT(//,25(3H **)/20X'LINPACK SAYS MATRIX IS SINGULAR'//)
      CALL SGBSL(DEL,4,N,1,1,IPVT,YL,0)
C
C               PLOT SOLUTION WITH IMSL LIBRARY ROUTINE
      CALL USPLO(X,Y,1,N,1,1,
     A           47HSOLUTION OF DIFFERENTIAL EQUATION - EXAMPLE 6.4, 47,
     B           1HX, 1, 1HY, 1, RANGE, 1HY, 1, IER)
      STOP
      END
```

Fig. 8 - Exemplo comparativo de um programa PROTRAN [RICE83]

IV. A Linguagem Fado: uma experiência no desenvolvimento de um sistema de geração

Neste capítulo, descrevemos nossa experiência no desenvolvimento de uma linguagem de quarta-geração, denominada Fado, e de seu respectivo ambiente de programação, para aplicação em um sistema de aquisição de dados de um experimento de Física de Altas Energias.

Linguagens de quarta-geração pertencem ao grupo de tecnologias de geração em reutilização de software, conforme descrito no capítulo anterior. Este tipo de linguagem é conhecido por oferecer um alto grau de benefício em termos de reutilização, porém restrito a uma área de aplicação específica [MART85] [WEBS88]. Sua principal vantagem em relação a linguagens de programação convencionais (i.e. FORTRAN, Pascal, etc) é que uma linguagem de mais alto nível, dedicada a uma aplicação em especial, é capaz de incorporar conceitos e abstrações muito mais próximas da área de aplicação, podendo esconder detalhes específicos de programação, sendo de fácil aprendizado para usuários não especialistas em computação. Um programa escrito em uma linguagem deste tipo é, normalmente, auto-descritivo e fácil de ler.

Em termos de reutilização, o aspecto principal de linguagens de quarta-geração é a sua capacidade em reutilizar determinados padrões de código que são repetidos nas diversas instâncias de programas que atendem a uma certa área de aplicação. O trabalho envolvido no projeto de uma linguagem deste tipo é, basicamente, o de identificar estes blocos de código que se repetem, a partir de uma modelagem da área de aplicação (i.e. identificação de suas estruturas básicas, operações, etc). Feito isto, identificam-se conceitos e abstrações existentes na área de

aplicação que melhor representem esses blocos de código, estabelecendo uma relação entre sentenças de muito alto nível com suas respectivas implementações.

A linguagem Fado 2.0 e seu ambiente de programação, descritos neste capítulo, fazem parte de um sistema para seleção, em tempo real, de eventos de Física de Altas Energias considerados promissores. Apresentamos, inicialmente, o sistema FADO ("Fast Analysis of Data Objects") [BARA88], no contexto do Projeto DELPHI ("DEtector with Lepton, Phothon and Hadron Identification") [DELP90]. Em seguida, descrevemos nosso trabalho no desenvolvimento da linguagem de quarta-geração e de seu ambiente de programação. Comentários gerais sobre o sistema implementado são feitos na seção IV.3, seguido de uma análise dos principais aspectos de reutilização presentes no mesmo.

IV.1. O Sistema FADO no contexto do Projeto DELPHI

DELPHI é um dos quatro projetos aprovados para o estudo de colisões entre pósitrons e elétrons que ocorrem dentro do acelerador de partículas LEP, construído pelo CERN na fronteira entre a França e a Suiça [CERN89b]. A experiência DELPHI ocupa uma área subterrânea equipada com um sofisticado detetor de partículas, cuja descrição detalhada pode ser encontrada em [DELP90], para coleta e observação de eventos resultantes das colisões. Cerca de 650 colaboradores, associados a 40 instituições de pesquisa em diversos países europeus, estão envolvidos nesta experiência. Nossa participação neste projeto deu-se através de um projeto de colaboração entre a COPPE/UFRJ e o Laboratório de Instrumentação e Física Experimental de Lisboa (LIP).

Sistemas de aquisição de dados eficientes e confiáveis são essenciais em experiências de Física de Altas Energias. Para que esses sistemas possam atender aos

requisitos de velocidade e confiabilidade das informações, é preciso dividir as tarefas envolvidas no tratamento dos dados em camadas, denominadas *níveis de gatilho* ("triggers") [GAVI85] [ALLAB87]. O sistema de aquisição de dados da experiência DELPHI possui quatro níveis de gatilho, conforme mostra o esquema da *figura 9*. Cada um desses níveis executa um tipo diferente de filtragem, eliminando informações errôneas e ruídos durante o processo de aquisição. Detalhes sobre o sistema de aquisição de dados da experiência DELPHI podem ser encontrados em [DELPHI85].

O sistema FADO é executado no quarto nível de gatilho. Neste nível de gatilho nenhuma informação é descartada, porém um rápido processamento é feito sobre as informações obtidas de forma a verificar a existência de eventos físicos interessantes. Este nível de gatilho, portanto, funciona como um nível de seleção de eventos considerados promissores que, ao serem identificados pelo sistema, são copiados para fitas magnéticas. Estas fitas são, então, analisadas por físicos através de sistemas de análise de dados [BARA88] [KUNZ91].

Para atender aos requisitos de tempo impostos pelo quarto nível de gatilho da experiência DELPHI (da ordem de 0.5 segundos por evento), o sistema FADO implementa um mecanismo de reconstrução seletiva de dados de eventos a partir de critérios físicos estabelecidos para seleção de eventos [BARA88]. A especificação dos critérios físicos, a serem usados no processo de seleção, são escrito em uma linguagem de quarta-geração como um conjunto de condições que classificam o evento. Cada uma dessas condições aciona, apenas, a reconstrução de uma parte do evento, não sendo necessário reconstruir todo o evento.

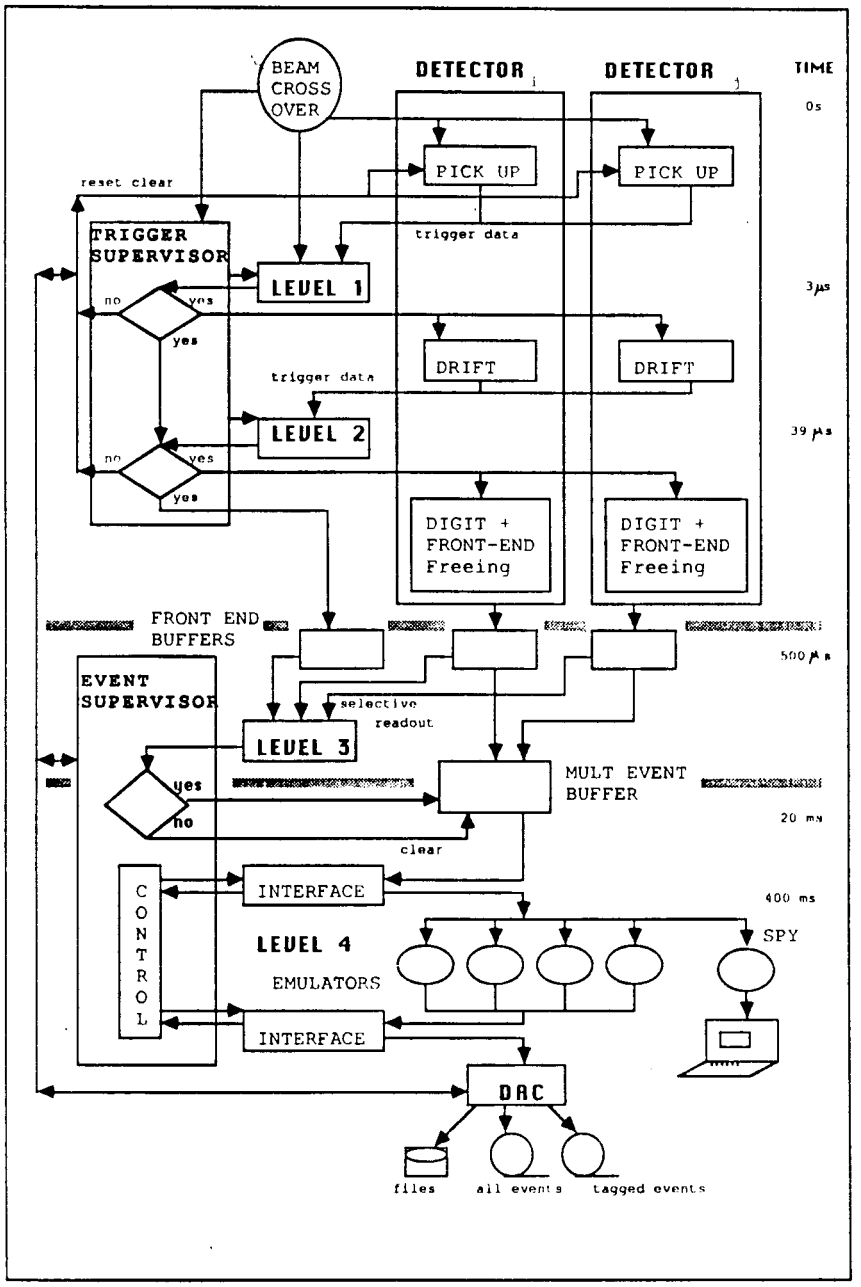


Fig. 9 - O Sistema de Aquisição de Dados do DELPHI

O sistema FADO é composto por dois ambientes de software distintos, conforme mostra a *figura 10* [WERN90a]:

1) um *ambiente de definição*, para descrição dos critérios físicos a serem usados, sendo composto de uma linguagem de quarta-geração e seu ambiente de programação, e
2) um *ambiente de execução*, que controla o processo de reconstrução e a validação de critérios físicos, sendo formado por um sistema de controle, uma estrutura de dados de eventos reconstruídos e uma biblioteca de algoritmos de reconstrução otimizados [DELP89].

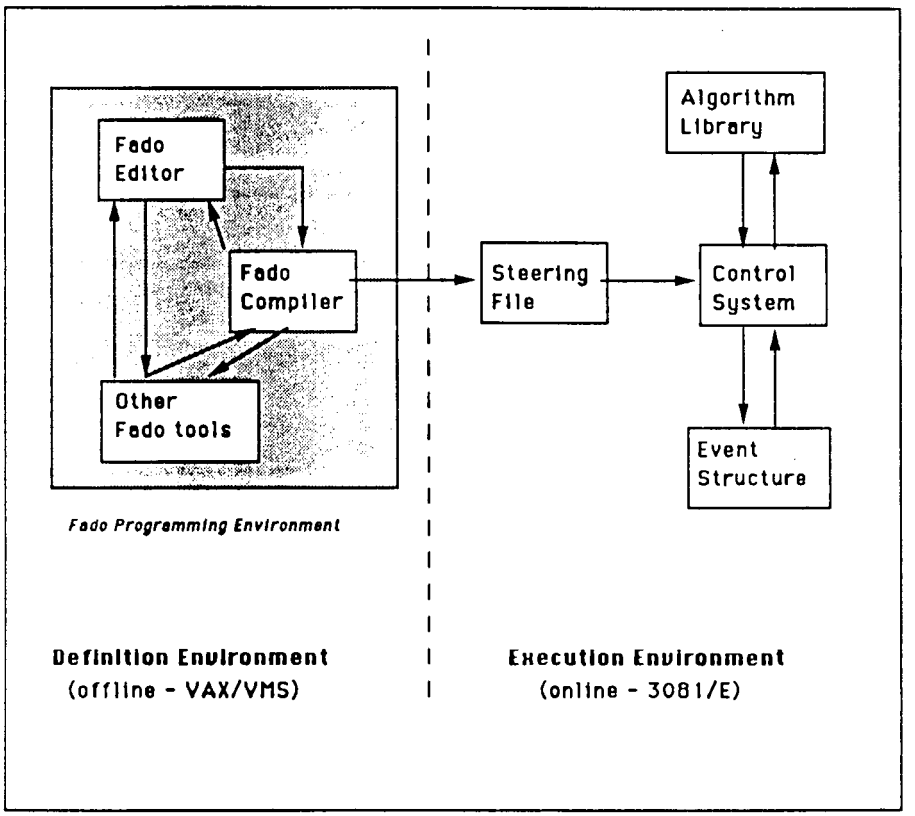


Fig. 10 - Ambientes de software do Sistema FADO

A descrição de critérios físicos é feita através de um programa escrito em uma linguagem de seleção, construído com o apoio de ferramentas de suporte à programação. A tradução deste programa gera uma espécie de arquivo de comandos de baixo nível, que define a estratégia a ser adotada durante o processo de reconstrução dos dados. Este arquivo é, então, carregado no ambiente de execução

que decodifica os comandos de baixo nível através de um sistema de controle. Este sistema tem como função controlar todo o processo de reconstrução de um evento, ativando os algoritmos de reconstrução apropriados, organizando as informações em bancos da estrutura de dados de eventos e verificando os critérios físicos estabelecidos.

Desta forma, a lógica (ou controle) envolvida na definição de critérios de seleção fica, totalmente, desacoplada da implementação do processo de reconstrução de dados, resultando em uma arquitetura bastante modular onde existe uma clara distinção entre "o que fazer" e "como fazê-lo".

O sistema FADO é executado, parcialmente, em uma plataforma fora de linha (i.e. VAX/VMS) e, parcialmente, em uma plataforma em linha (i.e. emuladores 3081/E) (veja a *figura 10*).

Nosso trabalho no projeto do sistema FADO constou do desenvolvimento de seu ambiente de definição, tendo sido o ambiente de execução desenvolvido pelo grupo do LIP. Detalhes sobre o ambiente de execução podem ser encontrados em [BARA88].

IV.2. A Linguagem Fado 2.0 e seu Ambiente de Programação

Uma primeira tentativa para a criação de uma linguagem de seleção tinha sido realizada, no início de 1988, pelo grupo do LIP. Neste período, uma linguagem operacional, bastante próxima do sistema de controle e muito dependente dos mecanismos de execução de programas (i.e. uma espécie de assembler simbólico), foi criada [BARA88]. Neste período, foi constatada a necessidade de uma linguagem de mais alto nível, mais natural e concisa, que pudesse ser facilmente usada pelos físicos.

Esta primeira versão da linguagem, entretanto, permitiu o teste e a validação dos mecanismos básicos do ambiente de execução, provando sua operacionalidade.

O desenvolvimento de uma linguagem de mais alto nível foi iniciado em meados de 1988, a partir de nosso ingresso no projeto. Após uma primeira análise sobre a área de aplicação específica do sistema FADO (i.e. seleção de eventos em Física de Altas Energias) e de seu contexto de execução, optamos por basear o projeto da linguagem Fado 2.0 em conceitos e abstrações encontrados na lógica matemática e na teoria de conjuntos. Consideramos esta plataforma bastante apropriada para a descrição das propriedades de um evento em Física de Altas Energias e natural o suficiente para ser usada com facilidade pelos físicos.

Um programa em Fado 2.0 é, basicamente, uma coleção de descrições de reações físicas. Cada reação é descrita a partir de um conjunto de condições lógicas feitas sobre quantidades físicas. Por exemplo, a determinação de que o número de elétrons encontrados no evento deve ser dois e que a energia total deve ser maior que 75 GeVs é descrita, em Fado 2.0, da seguinte forma:

```
REAÇÃO exemplo
        MULTIPLICIDADE(ELÉTRON) = 2
        ENERGIA_TOTAL(ELÉTRON) > 75 GeVs
FIM_REAÇÃO
```

No exemplo acima, ELÉTRON é a *lista* (ou conjunto) de partículas elétrons encontradas no processo de reconstrução do evento. Cada elemento de uma lista é denominado *objeto*[1]. Cada objeto possui valores associados descrevendo valores físicos do objeto, tais como sua energia, sua massa e coordenadas no espaço. Esses

1 O termo *objeto* neste sistema não está relacionado à definição encontrada em linguagens de programação orientada a objetos.

valores são chamados *parâmetros* do objeto. Uma lista, por sua vez, também possui valores associados que descrevem características gerais sobre a lista, tais como o número de elementos (MULTIPLICIDADE) e a energia total dos elementos (ENERGIA_TOTAL). Estes valores associados a listas são chamados *atributos* de lista.

Listas são divididas em *listas de sistema* e *listas definidas pelo usuário*. Listas de sistema podem, ainda, ser *elementares* ou *compostas*. *Listas elementares* são aquelas em que seus elementos são traços individuais, identificados por algoritmos de reconhecimento de padrões. No caso específico da aplicação em DELPHI, temos uma lista elementar associada a cada sub-detetor (i.e. TPC, OD, HPC, EMF, etc). *Listas compostas* são definidas a partir da associação entre diversos elementos de diferentes listas. O sistema fornece uma série de definições de listas compostas, pré-definidas, bastante utilizadas na descrição de critérios físicos. Exemplos de listas compostas do sistema são: ELÉTRON, MUÓN, PHÓTON, etc.

Listas definidas pelo usuário são definidas a partir de listas de sistema ou outras listas, previamente, definidas pelo usuário. Um exemplo de definição de uma lista de usuário é:

ELÉTRON_ENERGIA_MAIOR_5GEVS = = { X : X em ELÉTRON
& ENERGIA(X) > 5 GeVs }

Esta lista descreve o conjunto de elétrons cuja energia é maior que 5 GeVs. Observe a notação usada para descrição de conjuntos. A definição acima pode ser lida, verbalmente, como "a lista de elétrons de energia maior que 5 GeVs é o conjunto de x tal que x está em ELÉTRON e a energia de x é maior que 5 GeVs".

Na descrição de uma lista, é possível utilizarmos os operadores *E* e *OU* como, por exemplo:

CALORÍMETRO_EM = = { X : X OU(em HPC, em EMF) }

Existem, ainda, listas cujos elementos são formados por pares, trincas, quadruplos, etc, de objetos. Estas listas são denominadas *listas de tuplas*. Neste caso, *relações* entre objetos podem existir. Como exemplo de uma lista de tupla tem-se:

PAR = = { (X,Y) : X em ELÉTRON
Y em ELÉTRON
& X VIZINHO_DE Y }

No exemplo acima, VIZINHO_DE é uma relação definida pelo usuário, que deve ser descrita em uma unidade de programação apropriada, denominada *bloco de relações*. O sistema provê algumas relações pré-definidas (ex. EXTRAPOLAÇÃO, ASSOCIAÇÃO). Uma lista completa dos identificadores de sistema pode ser encontrada em [WERN90c].

Programadores Fado 2.0 podem, ainda, desenvolver *funções* para o cálculo de valores específicos, podendo fazer uso de funções disponíveis no sistema, tais como funções matemáticas (ex. SEN, COS, SQRT, etc), funções especiais (ex. MAX, MIN, SOMA) e funções para cálculo de valores físicos (ex. PT, ACOP). As únicas estruturas de programação permitidas em funções são a estrutura de "SE_SENÃO_ENTÃO" e atribuições.

Finalmente, é possível atribuir valores numéricos constantes a identificadores para uso nas diversas partes de um programa Fado 2.0 (ex. ENERGIA_DO_FEIXE

= 91). Portanto, um programa escrito na linguagem Fado 2.0 é composto pelas seguintes unidades de programação:

- Bloco de Reações, contendo a descrição de reações (pode ser visto como um programa principal);
- Bloco de Listas, contendo a definição de listas do usuário;
- Bloco de Relações, contendo a definição de relações do usuário;
- Bloco de Constantes, contendo a definição de constantes do usuário;
- Bloco de Funções, contendo a definição de funções do usuário.

Todos os blocos, exceto o bloco de reações, são opcionais. A ordem dos blocos, entretanto, deve ser respeitada. Comentários podem ser colocados em qualquer ponto do programa, desde que envolvidos por pontos de exclamação (!).

A descrição detalhada de outras facilidades da linguagem podem ser encontradas em [WERN90c] (ex. listas temporárias, listas compostas definidas pelo usuário, objetos constituintes, funções com valor constante, comandos de saída, marcador de tempo, etc). Esta descrição não foi incluida no texto por tratar de mecanismos muito específicos ou relacionados a aspectos de eficiência da linguagem.

Um exemplo completo de programa em Fado 2.0 pode ser encontrado no apêndice A.

IV.2.1. O Ambiente de Programação Fado

Para o desenvolvimento de um programa Fado 2.0, um físico deve poder contar com o apoio de ferramentas em suas atividades de programação. Vários ambientes integrados estão, hoje, disponíveis, provendo uma série de ferramentas de suporte às diversas atividades de programação. Um estudo geral sobre ambientes de programação pode ser encontrado em [WERN91c].

O Ambiente de Programação Fado é formado por seis ferramentas. São elas [WERN89b]:

- um editor sensível à linguagem;
- um compilador/tradutor;
- um depurador/verificador;
- um configurador;
- um otimizador, e
- um instalador.

Da lista apresentada acima, apenas, as duas primeiras ferramentas são essenciais ao funcionamento do sistema FADO, sendo as demais desejáveis para ter-se um sistema mais completo, genérico e adaptável a outras áreas de aplicação.

Como requisitos gerais do projeto do Ambiente de Programação Fado temos: a) uma forte integração das ferramentas do ambiente (conforme mostra a *figura 11*) e, b) a possibilidade de uso diferenciado para usuários com diferentes níveis de experiência (i.e. iniciantes e especialistas).

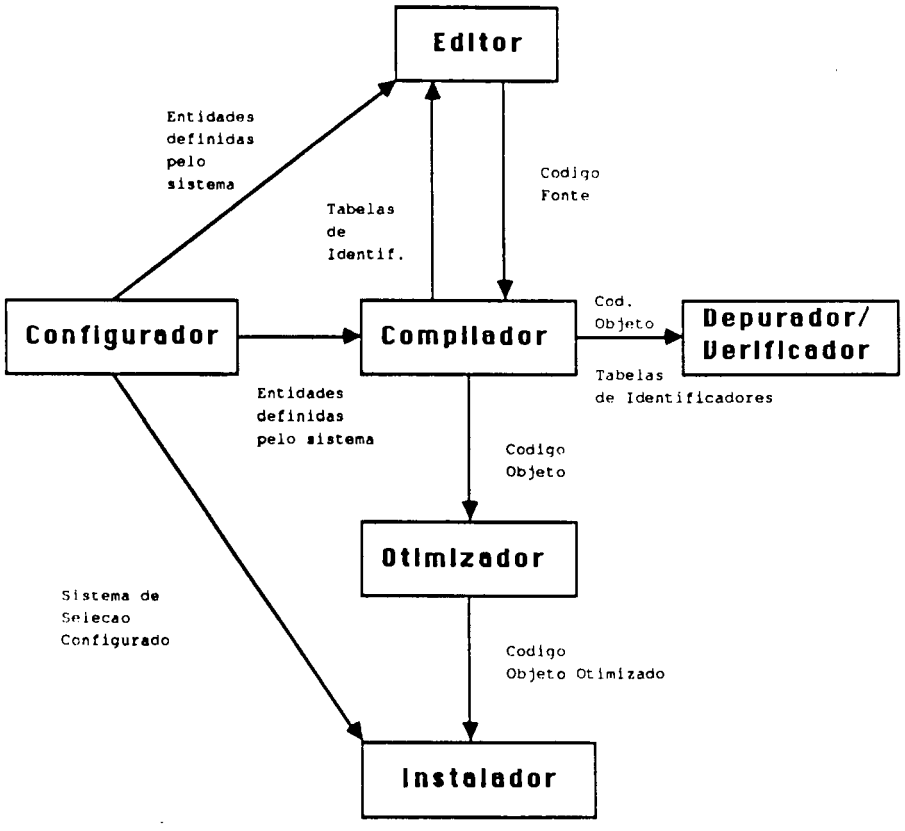


Fig. 11 - Integração entre ferramentas do ambiente de programação Fado

A seguir, apresentamos em detalhes a implementação do editor sensível à linguagem e do compilador/tradutor e descrevemos, brevemente, o objetivo das demais ferramentas do ambiente.

IV.2.1.1. O Editor Sensível à Linguagem

Editores sensíveis à linguagem surgiram a partir da observação de que "programas não são apenas um conjunto de caracteres que formam um texto, mas sim composições hierárquicas de estruturas computacionais" [TEIT84]. Como tal, é possível construí-los com base nessas estruturas, criando editores que provêm mecanismos que os tornam mais ativos durante o processo de codificação.

O editor sensível à linguagem Fado 2.0 incorpora uma coleção de construtores pré-definidos da linguagem (i.e. "templates"), que refletem sua gramática. Estes construtores poupam o programador Fado de saber, à priori, detalhes sintáticos da linguagem.

Os conceitos básicos envolvidos na implementação do editor são:

1) "*Buffer*": uma área temporária que provê um espaço de trabalho para edição de texto;
2) *Janela*: uma seção do vídeo que mostra o conteúdo de um "buffer";
3) *Marcadores*: correspondem a marcas no código fonte que indicam locais onde o usuário pode inserir textos de programa (marcadores são gerados, automaticamente, pelo editor), e
4) "*Tokens*": palavras-chaves escritas pelo programador que, quando *expandidas*, trazem construtores específicos da linguagem (ex. a sintaxe de um certo comando).

Marcadores podem ser *obrigatórios* ou *opcionais*. Marcadores obrigatórios são envolvidos pelos símbolos "<" e ">" e representam locais onde textos de programa devem ser fornecidos. Marcadores opcionais são envolvidos por "[" e "]" e representam locais onde construtores adicionais podem ser fornecidos. Estes últimos podem ser apagados do "buffer" de edição.

São três as possíveis operações com marcadores: *expan*são, *deleção* ou *escrita por cima*. Esta última opção substitui o marcador pelo texto inserido, automaticamente. Quando uma *expansão* ocorre, uma dentre três situações pode ocorrer:

a) o marcador é substituído por um "template" correspondente (caracteriza um marcador *não terminal*);

b) um texto aparece em uma outra janela, para auxiliar o usuário no fornecimento de um valor (caracteriza um marcador *terminal*) ou,

c) um cardápio aparece em uma outra janela, provendo possíveis opções para expansão em diferentes "templates" (caracteriza um marcador *cardápio*).

É possível caminhar, entre as opções de um cardápio, utilizando as teclas de setas, para cima e para baixo. A seleção de um ítem é feita através da tecla de retorno ou da tecla definida para expansão (ex. Ctrl_E). A não seleção de ítens é feita através da barra de espaço. Uma expansão pode ser desfeita através da tecla definida para desfazer expansões (ex. PF1 + Ctrl_E). O mesmo pode ser dito em relação à deleção de um marcador.

Diversos "tokens" foram definidos no editor sensível à linguagem Fado. Para obter uma lista dos "tokens" disponíveis é possível emitir um comando para listagem dos "tokens" (Ver [WERN90d] para maiores detalhes sobre a operação do editor).

O editor Fado pode ser usado em dois diferentes modos: no *modo dirigido* (modo padrão para iniciantes) ou no *modo livre* (para programadores especialistas). A diferença básica entre os dois modos diz respeito ao tipo de interação entre o usuário e o sistema. No primeiro, a interação é feita através da expansão dos construtores da linguagem e da seleção de opções de cardápios. Este modo é bastante útil para programadores iniciantes, entretanto, pode se tornar bastante cansativo para programadores com mais experiência em programação Fado. Logo, um modo livre é, também, oferecido, fazendo o editor Fado funcionar como um editor convencional.

De fato, uma mistura desses dois modos é, normalmente, usada. Sempre que uma construção específica é esquecida, no modo livre, é possível fazer uso da facilidade de "tokens" para obter a sintaxe desejada. Do mesmo modo, sempre que o programador tem certeza da sintaxe adequada à linguagem, no modo guiado, é possível escrevê-la por cima dos marcadores, substituindo-os automaticamente.

O editor Fado foi construido com base no editor LSE ("VAX Language-Sensitive Editor") [DIGI87], tendo sido previstas algumas facilidades adicionais de edição especializada, dentre elas:

- o carregamento automático de identificadores do sistema, pré-definidos, durante a fase de configuração do sistema Fado;
- a inclusão automática de identificadores definidos pelo usuário em uma lista de identificadores válidos porém, ainda, não definidos;
- a expansão automática de construtores de definição para os identificadores, ainda, não definidos.

Detalhes sobre a implementação do editor e uma discussão sobre as possibilidades de incorporação das facilidades adicionais, utilizando rotinas escritas na linguagem VAXTPU ("VAX Text Processing Utility") [DIGI88b], podem ser encontradas em [WERN90d]. Discutimos, ainda, em [WERN90d], questões relativas à integração do editor com o compilador.

IV.2.1.2. O Compilador

A complexidade da construção de compiladores já é bastante conhecida [TREM85]. Sabe-se, ainda, que a complexidade e natureza do processo de compilação depende, em grande parte, da linguagem de programação usada. Durante

a construção do compilador Fado, deparamo-nos com uma série de problemas comuns ao processo de compilação que puderam ser resolvidos com base na literatura existente sobre o assunto. Entretanto, diversos outros tipos de problemas específicos à linguagem tiveram que ser solucionados com base na nossa própria experiência e na experiência ganha no desenvolvimento do tradutor da primeira versão da linguagem [BARA88].

Um compilador é por definição um programa que traduz, ou converte, um programa escrito em uma linguagem fonte em um programa escrito em uma linguagem objeto. O compilador Fado tem como entrada a linguagem Fado 2.0 e como saída a linguagem Fado Assembler (descrita em [PIME90] e decodificada pelo ambiente de execução do sistema FADO).

O compilador Fado é composto pelos seguintes módulos:

1) *Analisador Léxico* ("scanner"): lê o programa fonte e reconhece unidades léxicas, denominadas "tokens" (i.e. identificadores, palavras-chaves, operadores, etc);
2) *Analisador Sintático* ("parser"): recebe as unidades léxicas do analisador léxico e as agrupa em unidades sintáticas (ex. sentenças), construindo uma *árvore sintática*, e
3) *Analisador Semântico*: verifica regras de contexto (ex. verifica o número e os tipos dos argumentos de uma função) e gera o código objeto correspondente.

Os dois primeiros módulos foram construídos com base na teoria de máquinas abstratas de reconhecimento, mais conhecidas como *autômatos finitos* [TREM85]. Portanto, a gramática Fado 2.0 é, previamente, definida e carregada em tabelas léxicas e sintáticas (em uma etapa anterior à compilação, denominada *etapa de carregamento*), sendo utilizadas durante o processo de compilação. Esta estrutura

facilitou bastante o acompanhamento do processo de evolução da linguagem de seleção até que um ponto mais estável de desenvolvimento fosse atingido.

O compilador desenvolvido utiliza uma estratégia de compilação de cima para baixo, para uma classe de gramáticas ESLL(1) ("Extended Simple LL(1)") [LEWI68]. Gramáticas LL(1) simples podem ser analisadas de uma forma determinística, através da verificação do próximo símbolo da cadeia de entrada a ser analisada. Este tipo de gramática tem sido bastante usado na definição de linguagens de programação por já existirem algoritmos de análise eficientes[2]. Além disso, técnicas de detecção e recuperação de erros podem ser, facilmente, adicionadas ao algoritmo de análise [TREM85].

O processo de compilação se constitui, basicamente, em percorrer um grafo sintático da gramática, formado por nós *não-terminais*, que correspondem a sub-grafos, e nós *terminais*, que correspondem a símbolos da linguagem. Os grafos sintáticos da linguagem Fado 2.0 podem ser encontrados em [WERN90e].

Quatro diferentes tipos de estratégia para recuperação de erros foram implementadas no compilador Fado. Cada uma delas é experimentada uma após a outra. Se nenhuma delas consegue corrigir o erro, o símbolo de entrada é, simplesmente, ignorado e as estratégias são experimentadas mais uma vez. As estratégias adotadas são [SETZ83]:

1) *Inserção de símbolos*: assume que o programador cometeu um erro, esquecendo um símbolo que deve ser inserido. A partir da lista de símbolos esperados, verifica-se se o símbolo que causou o erro é um sucessor dos elementos da lista.

---

2 O desempenho dos algoritmos existentes é uma função linear sobre o tamanho da cadeia de entrada.

2) *Troca de símbolos*: assume que o programador escreveu um símbolo errado. Neste caso, a estratégia 1 é aplicada no próximo símbolo da cadeia de entrada. Se a resposta for positiva, ao invés de inserir um símbolo, uma mensagem é enviada, avisando que o símbolo errado foi substituído pelo símbolo terminal que teve seu sucessor igual ao próximo símbolo de entrada.
3) *Procura por um delimitador*: verifica se o símbolo que causou o erro corresponde ao símbolo sucessor de algum não terminal procurado, ou se pertence a algum caminho alternativo desses sucessores.
4) *Eliminação de símbolos*: assume que o programador cometeu um erro escrevendo uma cadeia de entrada com um símbolo extra. Portanto, o próximo símbolo de entrada é comparado com a lista de símbolos esperados. Se o símbolo pertence a esta lista, uma mensagem é enviada, indicando que o símbolo foi eliminado.

As estratégias 1, 2 e 4 resolvem a maior parte dos erros que ocorrem, entretanto, é a estratégia 3 que garante a não eliminação de muitos símbolos de entrada [SETZ83].

O compilador Fado utiliza uma técnica para execução de análise semântica em que o tipo da análise de contexto executada e a natureza do código produzido são especificados para cada produção da gramática. Portanto, durante a aplicação das regras de produção da linguagem (i.e. percurso no grafo sintático), o "parser" aciona rotinas apropriadas à análise de contexto e geração de código. Este tipo de compilação é conhecido na literatura como *compilação direcionada à sintaxe* [TREM85].

Finalmente, devido a questões de integração entre as ferramentas do ambiente de programação, o compilador pode ser acionado em quatro diferentes modos. São eles:

- *modo padrão*: para compilação de programas Fado;
- *modo de depuração*: para compilação de programas Fado que serão depurados em uma sessão do depurador/verificador Fado;
- *modo de condição do depurador*: para compilação de uma condição enviada pelo depurador/verificador Fado (utilizado pelo comando de parada condicional [NUCC89]), e
- *modo de definição de listas compostas*: para compilação de um programa de definição de listas compostas do sistema (ver [WERN90c]).

O compilador foi, totalmente, desenvolvido em Pascal padrão. Uma descrição completa sobre sua implementação pode ser encontrada em [WERN90e]. Maiores informações sobre sua operação são encontradas em [WERN90c].

IV.2.1.3. Demais Ferramentas

Das ferramentas que compõem o ambiente de programação Fado, somente o editor sensível à linguagem e o compilador se encontram na atual versão do sistema FADO, por terem sido considerados fundamentais ao funcionamento imediato do sistema. Nesta seção descrevemos, brevemente, os objetivos das demais ferramentas (i.e. depurador, configurador, otimizador e instalador), que deverão estar presentes na evolução do atual sistema para outro mais completo e genérico, adaptável a outras áreas de aplicação.

O objetivo do *depurador/verificador* é prover um ambiente para verificação e depuração de programas envolvidos no sistema FADO. Esta ferramenta deve prover quatro diferentes níveis para depuração [WERN89b]:

- Nível 0: para testes e estatísticas de eventos;
- Nível 1: para depuração de programas Fado 2.0;
- Nível 2: para depuração do código gerado em Fado Assembler, e
- Nível 3: para depuração da máquina virtual e algoritmos de reconstrução (i.e. ambiente de execução).

A especificação e o projeto do depurador/verificador foram, totalmente, desenvolvidos tendo como resultado um projeto final de curso de um aluno da Universidade de Udine, Itália [NUCC89].

O objetivo do *configurador* é prover uma ferramenta, baseada em cardápios, capaz de integrar as diferentes etapas de configuração do sistema FADO (i.e. definição de parâmetros do sistema, definição de identificadores da linguagem de seleção, acoplamento de algoritmos, etc) e gerar, automaticamente, as tabelas de informações necessárias ao funcionamento do sistema. Embora esta ferramenta não tenha sido desenvolvida nesta primeira versão do sistema FADO, toda a estrutura das tabelas de informações para a armazenagem dos parâmetros gerais do sistema e identificadores da linguagem Fado 2.0 foi criada, assim como mecanismos para criação de uma biblioteca de listas compostas do sistema [WERN90c].

O objetivo do *otimizador* é, a partir do arquivo de comandos gerado pelo compilador, verificar a possibilidade de reorganização de comandos, fazendo uma análise do custo envolvido (em termos de tempo) para sua execução. Desta forma, sería possível descartar eventos através da verificação de critérios físicos "menos custosos" (i.e. mais rápidos).

O objetivo do *instalador* é fornecer uma ferramenta de suporte à atividade de carregamento do sistema de seleção em sua máquina hospedeira. No caso específico

da experiência DELPHI, esta ferramenta auxiliaría a instalação de todo o sistema FADO nos emuladores 3081/E, tarefa esta que envolve operações em diferentes equipamentos (i.e. IBM, VAX, emuladores).

IV.3. Comentários Gerais sobre o Sistema FADO

Nesta seção, são feitos comentários gerais sobre o Sistema FADO e suas características mais inovadoras. Inicialmente, apresentamos a abordagem tradicional para seleção de eventos em Física de Altas Energias, descrevendo as vantagens do Sistema FADO em relação a esta abordagem. Em seguida, uma rápida análise sobre a possibilidade de uso do atual sistema em outras áreas de aplicação é estabelecida. Finalmente, fazemos alguns comentários sobre a experiência no desenvolvimento do Sistema FADO.

IV.3.1. Uma nova abordagem para seleção de eventos

O Sistema FADO pode ser visto como uma nova abordagem para seleção de eventos em Física de Altas Energias. A abordagem "tradicional" baseia-se no desenvolvimento de programas completos (normalmente, escritos em FORTRAN) que analisam todo um evento para, então, serem capazes de classificá-lo. Neste processo, são pertinentes os seguintes comentários [WERN90b]:

- o processo é lento, na medida em que todos os dados são processados antes que uma decisão possa ser tomada;
- devido à complexidade do detetor, a tradução dos critérios de seleção em uma forma executável leva à definição de algoritmos de redução de dados muito complexos, tornando difícil a identificação dos critérios de seleção iniciais;

- as definições de critérios de seleção estão sujeitas a modificações frequentes, devido ao fato de que, durante a vida de uma experiência, físicos tentam desenvolver novas estratégias para deteção de novos tipos de eventos;
- os físicos que definem os critérios de seleção possuem como maior obstáculo o domínio da linguagem de programação (i.e. FORTRAN) e de formatos de dados envolvidos, ao invés das dificuldades impostas pelo domínio da aplicação, e
- os programas de reconstrução, longos e complexos, precisam ser continuamente refinados.

Além dos problemas relacionados à codificação e manutenção de programas de seleção, existiam restrições quanto ao tempo de processamento disponível no quarto nível de gatilho de DELPHI, o que impedia o uso dos algoritmos de reconstrução disponíveis.

Logo, se compararmos o Sistema FADO com os demais sistemas para seleção de eventos em Física de Altas Energias construídos da forma tradicional, podemos citar como principais características:

- o uso de algoritmos de reconstrução otimizados, para atendimento às restrições de tempo impostas [DELP89];
- mecanismo para reconstrução seletiva de partes do evento, para atendimento aos critérios estabelecidos, baseado na idéia de que as informações necessárias para classificar um evento fazem parte de um sub-conjunto das informações fornecidas pelo sistema de aquisição;
- a facilidade de manutenção dos critérios de seleção, a partir do uso de uma linguagem de muito alto nível;
- a transparência dos algoritmos de reconstrução na descrição dos critérios de seleção, e

- a facilidade de manutenção dos algoritmos de reconstrução, para permitir a exploração do conhecimento obtido através da resposta dos detetores em teste.

IV.3.2. Uso do Sistema em outras Áreas de Aplicação

Embora o Sistema FADO tenha sido desenvolvido no contexto de um Projeto de Sistema de Aquisição de Dados em Física de Altas Energias, ele possui características que podem ser interessantes a outras áreas de aplicação.

O tipo de problema que o sistema se propõe a solucionar é aquele em que é preciso selecionar ou filtrar informações mais significativas a partir de um grande volume de dados. Isto é obtido através da aplicação de algoritmos numéricos capazes de analisar, reconstruir e/ou reduzir informações, seguindo uma ordem específica de ativação, que pode variar de acordo com a estratégia para análise dos dados a ser adotada.

Uma área de aplicação típica, em que um sistema deste tipo poderia ser usado, é a área de processamento de imagens. O sistema pode servir como ferramenta para identificação de formas e classificação de imagens, em problemas de reconhecimento de padrões em geral. Este tipo de aplicação, normalmente, envolve grandes quantidades de informações e algoritmos de tratamento de dados complexos que são ativados segundo uma certa estratégia.

Um outro tipo de aplicação identificada, de uso quase imediato, ainda, dentro da área de Física de Altas Energias, diz respeito a uma das etapas envolvidas na análise de dados em experiências, denominada a *etapa de seleção e filtragem de eventos fora de linha*. A análise de dados em experiências envolve a redução de centenas de "gigabytes" de dados em bruto em poucos traços, ou em alguns números. Os dados

passam através de um estágio conhecido como *análise de DSTs* ("Data Summary Tapes") para, depois, passarem por um outro estágio, onde apenas um sub-conjunto das informações disponíveis é selecionado, servindo à uma aplicação específica (i.e. estágio de seleção e filtragem de eventos). Apenas, após estas fases, é que a análise física dos dados começa.

A adequação do sistema para atendimento dessa nova aplicação, no projeto DELPHI, implicaria, basicamente, na substituição da atual biblioteca de algoritmos otimizados por algoritmos mais adequados à análise de dados, e na substituição das rotinas de manipulação de dados por rotinas que manipulem estruturas TANAGRA [BERT86]. No nível da linguagem de seleção, poucas seriam as alterações previstas para introdução de novos conceitos, caso fossem necessárias.

IV.3.3. Comentários sobre o Desenvolvimento

O projeto do Sistema FADO pode ser visto como um exemplo de sucesso no desenvolvimento de um sistema científico executado por uma equipe multidisciplinar. A equipe desenvolvedora envolveu especialistas na área de aplicação (i.e. físicos e técnicos) trabalhando em diferentes ramos de atividade (i.e. análise de dados, reconstrução de eventos, especialistas em detetores, etc) e especialistas em computação (i.e. programadores e engenheiros de software).

O esforço necessário a cada membro da equipe, no sentido de eliminar dúvidas quanto a atividades não específicas a sua área, foi minimizado pelo constante diálogo com os demais membros da equipe, inclusive através do uso de facilidades de correio eletrônico durante os períodos em que a equipe ficou geograficamente distribuída.

Outros aspectos que consideramos importantes para o sucesso do projeto foram:

- a decisão de desenvolvimento em plataformas padrão (ex. uso de padrões de linguagens, disponíveis em diferentes equipamentos);
- a disponibilidade de ferramentas computacionais adequadas ao desenvolvimento (ex. VAX Toolkit [DIGI88a]), e
- a flexibilidade da arquitetura adotada para o sistema, permitindo o acompanhamento da evolução do sistema.

No que diz respeito à nossa participação no projeto, dada a variedade de aspectos envolvidos no projeto, foi necessário estudarmos, com detalhes, os seguintes assuntos, antes de iniciarmos as atividades de desenvolvimento da linguagem Fado 2.0 e de seu ambiente de programação:

- o software previamente construído até o momento de nosso ingresso no projeto (i.e. ambiente de execução do Sistema FADO, seu funcionamento, primeira versão da linguagem, etc);
- o software de suporte disponível no CERN (i.e. ferramentas computacionais disponíveis nos diversos equipamentos);
- conceitos, abstrações e sistemas existentes na área de aplicação em questão (i.e. seleção de eventos em Física de Altas Energias);
- projeto de linguagens de programação;
- a construção de compiladores/tradutores, e
- ambientes de suporte à programação.

De uma maneira geral, podemos concluir que um dos fatores que levou ao sucesso desta experiência, além dos fatores já mencionados, foi o estabelecimento de

atividades específicas à área de conhecimento de cada membro da equipe, responsabilizando-os por diferentes aspectos do sistema final.

IV.4. Reutilização de Software no Sistema FADO

Nossa experiência no desenvolvimento da linguagem Fado 2.0 e de seu ambiente de programação, no contexto do Sistema FADO, nos fez melhor compreender as questões relacionadas ao desenvolvimento de uma ferramenta de geração para reutilização de software em uma área de aplicação científica.

Se, por um lado, constatamos o grande benefício obtido com a implementação de um sistema como o descrito neste capítulo, seja na facilidade de desenvolvimento de novas aplicações ou nos índices de reutilização alcançados, por outro lado, observamos o esforço necessário a sua construção.

O projeto do Sistema FADO, conforme dito anteriormente, envolveu uma equipe de profissionais com diferentes especialidades, trabalhando em um esquema de colaboração e, durante uma certa parte do projeto, distribuídos geograficamente. O Sistema envolveu diversos equipamentos, sendo seus principais requisitos relacionados aos aspectos de eficiência, portatibilidade, reutilizabilidade, manutenção e evolução do sistema. Este contexto de projeto implicou em uma forte disciplina no que diz respeito ao cumprimento de padrões estabelecidos, na determinação de interfaces bem definidas entre os sub-sistemas e, principalmente, no diálogo constante entre os membros da equipe.

O aspecto mais importante do projeto, que permitiu a efetiva reutilização de código na área de seleção de eventos em Física de Altas Energias, está relacionado à arquitetura definida para o sistema, onde a parte referente ao processo de

reconstrução de eventos e a interação com o sistema de aquisição de dados está, totalmente, independente de sua lógica de controle (i.e. a estratégia de reconstrução).

O físico, usuário final do sistema, que deseja selecionar eventos com determinadas características físicas não deve precisar ter conhecimento específico sobre detetores, algoritmos complexos de reconstrução ou formatos de dados. Todo o conhecimento existente sobre o processo de reconstrução, a manipulação de estruturas de dados e de seus algoritmos associados deve poder ser reutilizado no desenvolvimento de diferentes estratégias para a seleção de eventos. Através da especificação de critérios de seleção, utilizando a linguagem de quarta-geração Fado 2.0, o usuário do Sistema FADO tem acesso implícito a esse conhecimento.

O compilador Fado tem embutido em suas rotinas semânticas e de geração de código a transformação das sentenças de mais alto nível em sequências de comandos de mais baixo nível, que são então decodificados pelo sistema de controle do ambiente de execução FADO. Podemos, portanto, dizer que o elemento de ligação final da aplicação (i.e. o programa para seleção de uma determinada classe de eventos) é o sistema de controle, que tem total controle sobre o acesso às informações do evento e dos algoritmos de reconstrução, associação e ajuste de eventos.

Linguagens de quarta-geração, assim como geradores de aplicação, possuem como principal característica a reutilização eficiente do conhecimento específico a uma certa aplicação através de mecanismos bastante simples, facilmente usados por usuários leigos em computação. Entretanto, o custo a ser pago está na restrição de seu uso em determinadas áreas de aplicação.

Existem formas de se tentar amenizar o problema da rigidez da estrutura implementada por tecnologias de geração. Faria [FARI91], por exemplo, utiliza

"templates" externos parametrizáveis para representação de padrões de arquitetura de geradores de aplicação, apliando o potencial de reuso destas ferramentas. O próprio Sistema FADO cria a figura de uma linguagem intermediária de mais baixo nível, decodificada por um sistema de controle que ativa os diferentes algoritmos de reconstrução e manipulação de dados, permitindo a adaptação do sistema para outras áreas de aplicação. Esta adaptação, entretanto, não se dá de forma imediata, sugerindo a criação de uma ferramenta específica para a solução dos problemas operacionais existentes (i.e. configurador).

No contexto do desenvolvimento de software científico, vemos o desenvolvimento de ferramentas de geração do tipo da descrita neste capítulo bastante útil devido aos seguintes aspectos:

- o próprio processo de desenvolvimento de uma ferramenta deste tipo serve como instrumento para um estudo aprofundado sobre a área de aplicação específica, podendo ser comparado, em parte, ao esforço envolvido nas atividades referenciadas na literatura como *análise de domínios* [PRIE90] [NEIG91];
- a possibilidade de geração automática de aplicações, a partir de uma descrição do problema, facilita o desenvolvimento de software por parte de usuários leigos em computação, e
- desde que o desenvolvimento de ferramentas deste tipo possa ser realizado por uma equipe multidisciplinar com formação tanto na área de aplicação como na área de computação, porém bem integrada, é possível criar aplicações cuja qualidade está dentro dos padrões de Engenharia de Software e cujo conteúdo está de acordo com o estado da arte na área de aplicação específica.

V. O Ambiente CAOS: uma experiência no desenvolvimento de um sistema de composição

Neste capítulo, descrevemos nossa experiência no desenvolvimento de um ambiente orientado a objetos de apoio ao desenvolvimento de aplicações científicas, denominado CAOS ("Composing Application-Objects' System") [WERN91a]. E ste ambiente fornece uma série de ferramentas especializadas que dão suporte à organização e manipulação de uma biblioteca de componentes de software reutilizáveis, facilitando o desenvolvimento de aplicações a partir da composição de software já existente.

Ambientes de suporte a bibliotecas de componentes pertencem ao grupo de tecnologias de composição, descrito no capítulo III, cuja principal característica é a possibilidade de criação de aplicações a partir da composição de componentes atômicos, previamente construidos. Ambientes deste tipo devem oferecer ferramentas de suporte a todas as atividades envolvidas na busca, compreensão, modificação e composição de componentes.

Conforme discutido anteriormente, é intenso o uso de bibliotecas de programas no contexto científico. A razão óbvia para explicar este fato deve-se ao alto grau de complexidade do software desenvolvido neste domínio de aplicação, tornando impraticável o desenvolvimento de novas aplicações a partir do nada.

Na seção III.4 discutimos o uso de bibliotecas de programas no contexto científico. Naquela seção, questionamos a inclusão de bibliotecas de componentes, pura e simplesmente, como exemplo de tecnologia de composição, na medida em que os aspectos principais de composição não são colocados em evidência. Bibliotecas de componentes, a nosso ver, só podem ser consideradas como exemplo de tecnologia de

composição se acompanhadas de ferramentas de suporte adequadas às atividades relacionadas à escolha, entendimento, adaptação e, principalmente, composição de componentes.

O ambiente descrito neste capítulo é um exemplo de tecnologia de composição, na medida em que provê uma bilbioteca de componentes acompanhada de ferramentas de suporte a todas as atividades de composição. As primeiras idéias sobre um sistema de apoio ao desenvolvimento de software científico, que fosse capaz de reutilizar programas científicos já existentes, foram amadurecidas durante nossa participação no grupo de trabalho do Centro Universitário de Informática da Universidade de Genebra (CUI), no contexto do projeto ITHACA (um projeto ESPRIT II) [PROF89], acrescida de resultados apresentados por Prieto, na implementação de um esquema de classificação por facetas [PRIE85], e de nossa experiência no desenvolvimento de software científico, tendo resultado em uma primeira proposta do sistema para composição de aplicações científicas, descrita em [WERN91a].

Neste capítulo, descrevemos o ambiente CAOS e seus principais elementos, apresentando seus requisitos e o primeiro protótipo do sistema implementado. Em seguida, fazemos alguns comentários gerais sobre o ambiente CAOS, descrevendo trabalhos relacionados e questões sobre o desenvolvimento do protótipo. Na seção V.3., resumimos as principais características do ambiente CAOS e avaliamos o uso de ferramentas deste tipo no contexto científico.

V.1. O Ambiente CAOS

O ambiente CAOS é um ambiente orientado a objetos de suporte à reutilização de software científico que provê uma série de ferramentas especializadas

para o atendimento das atividades de organização e manipulação de uma biblioteca de componentes de software reutilizáveis.

O uso de conceitos e técnicas de orientação a objetos, no sistema CAOS, teve como principal motivação os diversos aspectos incorporados neste paradigma que facilitam a reutilização de software. A programação orientada a objetos, através de seus mecanismos para organização e decomposição de sistemas em entidades encapsuladas (i.e. objetos e classes) e mecanismos para a modificação e composição incremental de software (i.e. herança, generalização e ligação dinâmica) [TSIC88], provê facilidades básicas ao desenvolvimento de aplicações baseado na reutilização de componentes de software.

No contexto científico, começam a surgir as primeiras tentativas de utilização das técnicas de orientação a objetos no desenvolvimento de aplicações científicas [ALVE90] [ATWO90] [ATWO91] [KATA91] [OLEY91]. A discussão quanto à adoção de linguagens de programação orientadas a objetos (LPOOs) passa por uma antiga discussão sobre a adoção de qualquer outra linguagem de programação que não seja FORTRAN. Os argumentos, em geral, são [METC91]:

1) que o aprendizado de uma nova linguagem, principalmente quando ela incorpora conceitos diferentes, pode levar muito tempo e que, portanto, não se sabe ao certo o quanto isso implica em atrasos no desenvolvimento de novas aplicações;

2) que os compiladores FORTRAN são, atualmente, os mais eficientes para a geração de código para aplicações científicas, e

3) dada a imensa gama de programas científicos, atualmente, disponíveis em FORTRAN, como fazer uso desta preciosa fonte de recursos, na medida em que

muitas das atuais LPOOs não consideram a comunicação com outras linguagens de programação?

O objetivo do ambiente CAOS é permitir o uso incremental das facilidades oferecidas pelo sistema, incluindo mecanismos que possibilitam a reutilização de programas gerados externamente ao sistema (i.e. escritos em uma linguagem procedimental como FORTRAN ou C). Desta forma, os impasses criados para utilização de sistemas orientados a objetos no contexto científico são minimizados.

O ambiente CAOS provê ferramentas de suporte às atividades de recuperação, compreensão, modificação e composição de componentes armazenados em uma base de componentes. Os elementos básicos do ambiente CAOS são [WERN91a] [WERN91d]:

- uma base de componentes de software;
- um gerente de componentes;
- um navegador;
- um acoplador de rotinas externas;
- um gerente de composição de aplicações;
- uma linguagem de composição de aplicações, e
- um monitor.

A *figura 12* mostra, esquematicamente, os elementos do ambiente CAOS.

As sub-seções que se seguem apresentam o modelo de desenvolvimento de software assumido pelo sistema CAOS, os requisitos de cada um dos elementos do ambiente e a descrição do protótipo implementado.

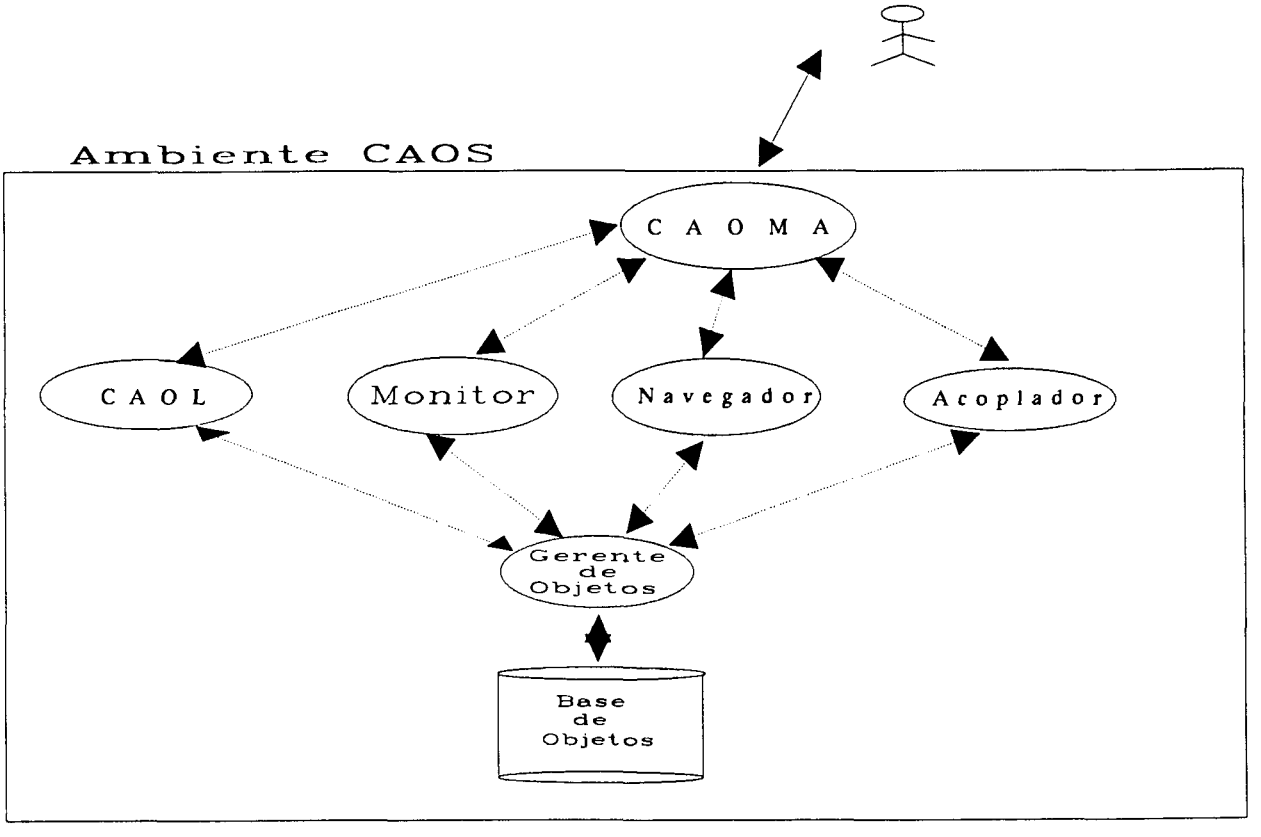


Fig. 12 - Elementos do ambiente CAOS

V.1.1. O Modelo de Desenvolvimento em CAOS

Dada a discussão, colocada no capítulo III, sobre a necessidade de uma mudança de perspectiva relacionada à maneira em que o software é, tradicionalmente, desenvolvido quando se deseja adotar uma estratégia de reutilização de software, observamos que, realmente, o desenvolvimento de aplicações, totalmente, baseado em uma estratégia *de cima para baixo* (i.e. por refinamentos sucessivos e na decomposição do problema inicial em sub-problemas, até que estes possam ser inteiramente solucionados), leva-nos a uma solução muito particular, dificilmente composta a partir de código já existente.

Por outro lado, uma abordagem para o desenvolvimento de aplicações seguindo uma abordagem *de baixo para cima* (i.e. através da localização de componentes reutilizáveis e composição de objetos mais complexos) deve estar apoiada em uma estratégia central para a solução do problema, capaz de guiar e organizar o processo de desenvolvimento.

Acreditamos, portanto, que uma abordagem ideal para o desenvolvimento de aplicações, baseado na composição de software, deva incorporar aspectos tanto de uma estratégia *de cima para baixo* como *de baixo para cima*, de forma que os requisitos da aplicação possam, em algum momento, ser atendidos por componentes já existentes.

O modelo de desenvolvimento utilizado no sistema CAOS assume a existência de uma base de componentes reutilizáveis, munidos de sua documentação, e de um mecanismo de seleção capaz de localizar componentes relevantes à aplicação. Podemos listar como principais características deste modelo [WERN91d]:

- especificação da aplicação baseada na identificação de componentes básicos e na descrição das propriedades a serem satisfeitas pelos mesmos;
- seleção de componentes potencialmente úteis;
- compreensão e modificação de componentes selecionados;
- construção de novos componentes, caso seja necessária;
- composição da aplicação a partir dos componentes selecionados, adaptados ou construídos;
- observação do comportamento da aplicação através da sua execução, e
- continuação do desenvolvimento através da substituição e/ou evolução dos componentes.

Neste modelo, a ênfase é dada ao desenvolvimento contínuo, a partir da substituição ou evolução de componentes, não sendo gasto muito tempo na especificação da aplicação. A construção de uma aplicação, neste modelo, é vista como o resultado de experiências na composição, especialização e generalização de componentes de um certo domínio de aplicação.

O sucesso deste modelo depende não somente da qualidade dos componentes disponíveis na base de componentes mas, também, de mecanismos de seleção suficientemente capazes de, a partir da descrição das propriedades dos componentes da aplicação feita na fase de especificação, localizar componentes potencialmente úteis.

V.1.2. A Base de Componentes de Software e o Gerente de Componentes

A iniciativa de se criar uma base, contendo diversos componentes de software, servindo como fonte de recursos durante o processo de reutilização não é nova. A organização, recuperação e manipulação desses componentes é que se torna um aspecto fundamental no desenvolvimento de bases de componentes, de forma que o acesso a esses recursos seja feito de maneira adequada e eficiente.

Uma base de componentes que possui, apenas, informação sobre o código fonte ou código objeto de componentes e cujo acesso é realizado através de um mecanismo de indexação simples (ex. por palavra-chave) não é, suficientemente, útil ao processo de desenvolvimento baseado em reutilização de componentes, por dois aspectos.

Primeiro, ao analisarmos um componente, a fim de estabelecer sua utilidade em uma dada aplicação, passamos por diversos estágios que vão desde a compreensão de sua função básica e observações sobre decisões de projeto, até detalhes mais específicos de sua implementação. Note que o conhecimento sobre a análise e projeto de um componente não mais se encontra em seu código fonte [PRIE90], dificultando, assim, a tarefa de determinação da adequação das características do componente aos requisitos da nova aplicação.

Segundo, o acesso a componentes através de um mecanismo de indexação simples, somente, é capaz de representar componentes sobre um determinado aspecto (i.e. a palavra-chave correspondente). É necessário, porém, descrever componentes sobre diferentes aspectos, ou visões [ROUS90], além de ser capaz de manter uma certa correspondência entre estas diferentes visões.

Um outro fator importante na implementação de uma base de componentes diz respeito ao volume de componentes armazenados e a facilidade e eficiência para sua recuperação. Assumindo que o volume de componentes reutilizáveis de uma base tende a crescer com o tempo e que, portanto, seu acesso em disco pode se tornar demasiadamente ineficiente, estabelece-se a necessidade de um *gerente de componentes* capaz de manter componentes referenciados em memória, decidindo o momento adequado para o processo de transferência de componentes do disco para memória e reduzindo ao máximo o número de acessos ao disco. Outra função deste gerente de componentes é controlar a criação, modificação ou remoção de componentes da base de componentes.

Esquemas de agrupamento ("clusters") e mecanismos de acesso a disco similares aos implementados em sistemas tais como [HARR89], [KAEH87] e

[MONT92b] podem servir de base para a implementação de um gerente de componentes deste tipo.

Assim sendo, estabelece-se como principais requisitos da base de componentes de software do ambiente CAOS os seguintes:

1) capacidade de armazenagem de informações diversas sobre o componente (i.e. não apenas seu código fonte mas informações gerais sobre sua funcionalidade, detalhes de projeto, exemplos de utilização, etc);
2) mecanismos para organização de componentes em coleções que possam ser referenciadas sobre diferentes aspectos, determinando diferentes visões ou contextos, mantendo o relacionamento entre as diversas coleções, e
3) mecanismos para transferência de componentes entre o disco e a memória (i.e. necessidade de um gerente de componentes).

No atual protótipo do ambiente CAOS, descrito na seção V.1.7, os dois primeiros requisitos são atendidos através da adoção de uma rede de informações do tipo hipertexto como unidade organizacional e de um esquema de classificação por facetas, tal como descrito por Prieto-Diaz [PRIE85]. Nesta versão, assumimos que todos os objetos do sistema residem em memória, sendo o gerente de componentes responsável por manter atualizada a lista de todos os objetos do sistema.

V.1.3. O Navegador

De uma maneira geral, existem duas formas para recuperação de elementos de uma base de componentes, tal como descrita na seção anterior:

1) através de consultas diretas à base de componentes, utilizando-se, por exemplo, uma linguagem de consulta capaz de exprimir as características do componente procurado, e
2) através da navegação sobre a estrutura organizacional da base, à procura de componentes com características, ainda, não muito bem estabelecidas.

Esses dois métodos não são exclusivos, podendo ser utilizados conjuntamente, de forma a se complementarem. Assim sendo, a partir de uma primeira consulta tentativa, começa-se a percorrer a base à procura de componentes mais específicos ou, da mesma forma, a partir de uma primeira visualização das características encontradas em objetos observados durante um processo de navegação, opta-se pela definição de uma consulta mais específica.

No ambiente CAOS, é prevista a implementação destes dois métodos em um mesmo elemento do ambiente, denominado o *navegador*. É a partir desta ferramenta de navegação que o usuário determina, através da definição de critérios de seleção do esquema de facetas implementado, um conjunto mínimo de elementos a serem explorados, caminhando pelos relacionamentos entre componentes até que o componente procurado seja encontrado.

O navegador tem, portanto, como principais funções:

1) localizar componentes reutilizáveis similares, candidatos a uma determinada função ou atividade;
2) permitir a compreensão desses componentes, e
3) permitir a navegação entre os diversos componentes da base.

O conceito de componentes similares pode ser encontrado em diversos sistemas sobre diferentes formas, quer como afinidades léxicas, no sistema GURU [MAAR89], funções de afinidade, no "Affinity Browser" [PINT90], atribuições de peso ao grau de atendimento das características e propriedades do componente em relação à sua classe, no sistema ESF-ROSE [MOIN89], ou na atribuição de pesos em arestas de um grafo que representa a estrutura hierárquica de classes, como encontrado no trabalho de Prieto-Diaz [PRIE85].

No protótipo do ambiente CAOS, o estabelecimento da similaridade entre componentes é feito através da criação de ligações diretas (i.e. arestas) entre os nós do hipertexto que caracterizam cada um dos componentes envolvidos na relação de similaridade (ver seção V.1.7).

As funções do navegador, no atual protótipo, são atendidas através da implementação dos mecanismos de recuperação e organização de componentes (i.e. o esquema de classificação por facetas e a rede de informações do tipo hipertexto). Através da determinação de critérios de seleção por facetas, o usuário localiza um conjunto mínimo de candidatos a serem explorados, navegando sobre as informações existentes do componente e sobre seus relacionamentos, quer hierárquicos (i.e. ancestral, descendentes) ou semânticos (i.e. componentes similares, relações de referência, etc), a partir da navegação sobre as ligações do hipertexto.

V.1.4. O Acoplador de Rotinas Externas

Uma das principais características do ambiente CAOS é sua capacidade de encapsular rotinas externas em objetos manipuláveis pelo sistema. Esta característica permite a reutilização de código previamente definido em outras linguagens de programação (ex. FORTRAN ou C).

Pode-se dizer que a população da base de componentes, descrita anteriormente, é feita a partir de duas fontes de recursos. A primeira delas é o próprio sistema CAOS que, a partir da composição de aplicações e criação de novos componentes, armazenará seus resultados na base para futuras utilizações. A segunda fonte de recursos está relacionada à fontes externas. Neste caso, os recursos precisam ser previamente tratados, de forma que possam ser, devidamente, manipulados pelo sistema e por seus usuários.

A ferramenta, denominada *Acoplador*, tem como principal função preparar o software externo ao sistema CAOS, integrando-o ao sistema. Como principais requisitos desta ferramenta, temos:

1) a possibilidade de acoplamento (i.e. integração) em dois níveis:
   a) *acoplamento mínimo*: descreve o protocolo de comunicação de rotinas externas, definindo seu nome e parâmetros formais e,
   b) *acoplamento estruturado*: define objetos complexos a partir de rotinas externas já acopladas e objetos disponíveis no sistema;
2) alimentação da rede de informações sobre o novo componente acoplado, para o atendimento do requisito 1) da base de componentes (i.e. armazenagem de informações diversas sobre o novo componente), e
3) determinação da classificação do novo componente segundo o esquema de facetas utilizado.

No atual protótipo do sistema CAOS todos esses requisitos foram satisfeitos, utilizando-se rotinas escritas em C para o teste de acoplamento (ver seção V.1.7).

V.1.5. O Gerente e a Linguagem de Composição de Aplicações

Existem dois grandes problemas encontrados na abordagem de composição para reutilização [MOIN89]. O primeiro diz respeito à especificação ou descrição de componentes de forma a permitir sua recuperação, compreensão e adaptação. O segundo problema está relacionado à definição de princípios básicos que regem a combinação desses componentes para formação de componentes mais complexos.

Diversas abordagens para a descrição de componentes existem e incluem o uso de palavras-chaves, esquemas de indexação múltipla [PRIE85] [ROUS90], e métodos mais inovativos tais como esquemas parcialmente interpretados no sistema PARIS (ver seção III.2.2.2) [KATZ89]. No atual protótipo do ambiente CAOS, a descrição de componentes é feita através da definição de valores de facetas associadas à classes de objetos (ver seção V.1.7.).

Quanto à definição de princípios básicos de composição (o segundo problema citado), este é, ainda, um assunto de intensa pesquisa, apesar de já existirem algumas propostas encontradas na literatura [MEY 91] [STAD91] [PINT91] [HELM90] [VITE90] [RAJ 89]. De acordo com Stadelmann [STAD91], existem duas possíveis maneiras de abordar o problema de composição de componentes em linguagens orientadas a objetos:

1) através da extensão de construtores da linguagem (ex. Contracts [HELM90] e Jade [RAJ 89]), ou
2) através de ferramentas interativas (ex. Vista [MEY 91], Teamworks [STAD91] e Gluons [PINT91]).

No ambiente CAOS, adota-se uma solução híbrida para este problema que fornece, além de uma ferramenta interativa para auxílio durante o processo de composição, denominado *gerente de composição de aplicações* (CAOMA - "Composing Application-Objects' MAnager"), uma linguagem de programação capaz de resolver questões sobre a composição de objetos, denominada CAOL ("Composing Application-Objects Language"), que utiliza os mecanismos tradicionais de composição de linguagens orientadas a objetos (i.e. troca de mensagem e herança) e introduz um mecanismo para verificação de tipos, em tempo de compilação, antecipando possíveis erros de execução e possibilitando a avaliação do impacto de mudanças durante a adaptação de componentes.

A linguagem para composição de aplicações do ambiente CAOS tem como principais requisitos ser uma linguagem fortemente orientada a objetos, aproveitando ao máximo as características do paradigma, e prover mecanismos adequados ao atendimento das seguintes atividades importantes ao desenvolvimento de aplicações científicas:

a) definição de algoritmos;
b) definição de estruturas de dados complexas;
c) composição de objetos complexos e,
d) conversão de dados.

Portanto, o modelo de objetos desta linguagem deve seguir, essencialmente, o modelo de linguagens de programação orientadas a objetos, tal como Smalltalk [GOLD84] ou Actor [WHIT91], no que concerne os conceitos de classe, objeto e mecanismo de herança. A linguagem deve, ainda, oferecer um conjunto mínimo de estruturas de dados úteis à conversão de dados em aplicações científicas, incluindo: matriz, pilha, fila, array, conjunto, coleção ordenada, grafo, árvore, registro, etc.

No protótipo do ambiente CAOS, a compilação de classes em CAOL resulta na geração de classes correspondentes em Actor. Este processo de geração, entretanto, pode ser modificado de forma a gerar classes correspondentes em outras linguagens de programação orientadas a objetos (ex. C++) ou, ainda, em linguagens convencionais (ex. Pascal, C, FORTRAN, etc). Maiores detalhes sobre o processo de geração das classes em Actor podem ser encontrados na seção V.1.7.

V.1.6. O Monitor

O objetivo da ferramenta de monitoração é permitir a execução de aplicações construidas no ambiente CAOS, de forma a possibilitar a observação de seu comportamento. Para isto, é preciso criar um mecanismo de controle do conjunto de instâncias de classes da aplicação criadas durante sua execução, provendo um ambiente similar ao espaço de trabalho da linguagem Actor [WHIT91].

Baseado nos resultados obtidos durante a execução da aplicação, o desenvolvedor poderá decidir entre a inclusão da nova aplicação na base de componentes do ambiente, para futura reutilização, e/ou a continuação do processo de desenvolvimento, substituindo e/ou evoluindo componentes até que obtenha um resultado satisfatório.

V.1.7. O Protótipo do Ambiente

A partir dos requisitos do ambiente CAOS descritos nas sub-seções anteriores, implementou-se um primeiro protótipo do ambiente na linguagem Actor 3.0 [WHIT91], rodando sobre MS-Windows 3.0 [MICR90], em uma plataforma de microcomputadores 486.

O protótipo do ambiente CAOS é formado por uma coleção de classes em Actor que implementam os conceitos necessários para o atendimento dos requisitos do ambiente. A *figura 13* mostra, esquematicamente, os principais conceitos envolvidos no protótipo do ambiente CAOS. O diagrama usado nesta figura é uma adaptação do diagrama proposto em [COAD92]. Nem todos os conceitos apresentados nesta figura são classes de objetos em sua implementação em Actor (ex. ligação, critério de faceta, valor de faceta, etc), entretanto, auxiliam na compreensão do esquema como um todo. Para manter a legibilidade da figura, alguns relacionamentos foram suprimidos.

A característica principal do ambiente está relacionada à sua organização geral sobre a forma de hipertexto. Portanto, os conceitos de *nó* e *ligação* entre nós são o ponto de partida para compreensão dos demais elementos do ambiente.

Um nó é uma unidade atômica de informação sobre um componente. Todo nó possui uma janela associada, que consiste na interface do usuário para acesso às operações permitidas de um nó, através de cardápios de comandos e movimentos de mouse. Nós podem ser do tipo descrição, classe, método, papel, faceta ou rotina.

Um *nó de descrição* consiste em um conjunto de palavras que descrevem informações textuais sobre um componente. Nós de descrição podem estar associados entre si através de *ligações* do tipo "palavra-nó", ou seja, determinadas palavras de um nó podem levar a outros nós de informação relacionados. Um nó de descrição pode ser ativado *somente para leitura* ou *para edição*, permitindo ou não o acesso ao seu conteúdo, dependendo se a ativação do nó for feita para atender a um processo de navegação ou de edição.

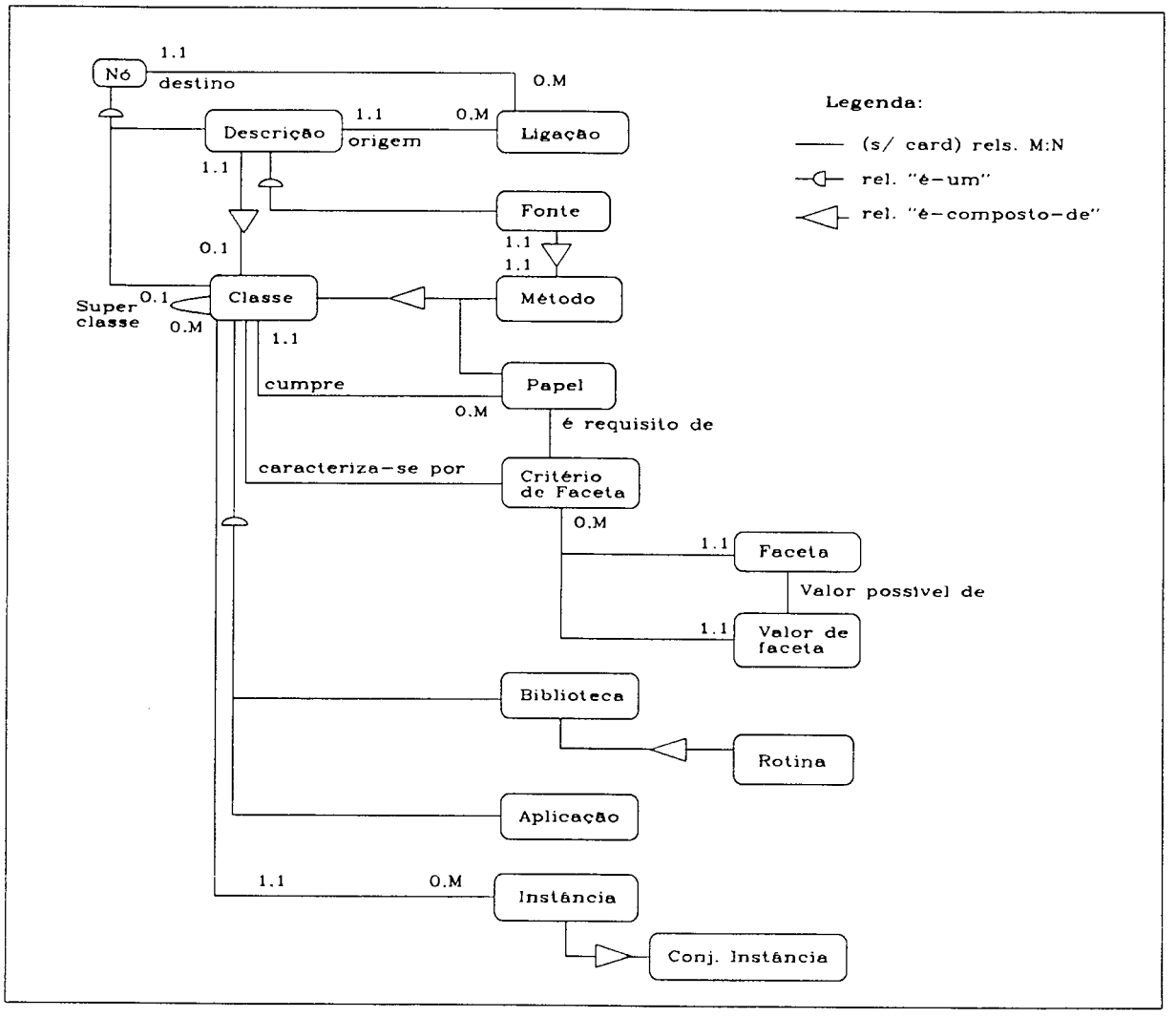


Fig. 13 - Principais conceitos do protótipo do ambiente CAOS.

Um nó do tipo *classe* contém informações estruturais e comportamentais sobre um componente. Um nó de classe tem sempre um nó de descrição associado que, a partir da ligação com outros nós de descrição, permite a criação de uma rede de informações textuais sobre o componente. Classes (i.e. nós do tipo classe) podem estar relacionadas entre si através de ligações do tipo "palavra-nó" de seu nó de

descrição associado. Ligações deste tipo podem ser estabelecidas de forma a implementar relações de similaridade entre classes.

As informações estruturais e comportamentais de uma classe consistem em um conjunto de *métodos* e ***papéis***. *Métodos* em CAOS traduzem o mesmo significado de métodos em outras linguagens de programação orientadas a objetos (LPOOs). Cada método de uma classe possui um *fonte* associado, que é uma especialização de um nó de descrição com capacidade para compilação de seu conteúdo.

Papéis correspondem à variáveis de instância em outras LPOOs e podem ser atendidos por uma classe já definida ou, ainda, por definir. O nome ***papel*** evidencia a possibilidade de reutilização de classes já definidas a partir do cumprimento de determinados requisitos estabelecidos para o papel. Cada papel possui um nome e uma descrição feita a partir da definição de *critérios de faceta*. Através da consulta das classes existentes, cujos critérios de facetas atendam aos requisitos do papel, é possível associar uma classe ao papel.

Classes caracterizam-se por um conjunto de *critérios de faceta* que são usados no processo de seleção de classes úteis para reutilização. A determinação de critérios de faceta é feita através da associação de valores, escolhidos a partir de uma lista de *valores possíveis* para cada faceta, a uma ou mais facetas disponíveis no sistema. Uma faceta pode estar associada a um nó de descrição, possibilitando a inclusão de descrições textuais sobre a mesma.

Facetas, no protótipo do ambiente CAOS, implementam o esquema de classificação proposto por Prieto-Díaz [PRIE85] em um ambiente orientado a objetos para recuperação de classes. O esquema de classificação por facetas consiste em descrever relações genéricas básicas entre elementos a partir da montagem, ou

síntese, de suas classes elementares. A tabela a seguir mostra um exemplo de facetas e valores possíveis para classificação de componentes de software científico.

Conforme identificado por Prieto-Díaz [PRIE91], o esquema de classificação por facetas é mais eficiente quando sua aplicação se restringe a coleções específicas, aumentando seu poder descritivo. Na medida em que o ambiente CAOS pode servir a diferentes áreas de aplicação, assume-se que a lista de facetas e de seus possíveis valores será adaptada ao tipo de componentes armazenados no sistema. Portanto, é possível criar e remover facetas e valores possíveis associados a partir de uma opção de comando, no cardápio de comandos da janela do nó principal do sistema (ver exemplo da tabela 1).

Classes em CAOS geram classes em Actor que as implementam. O processo de geração das classes Actor correspondentes é ativado a partir de uma outra opção de comando, no cardápio de comandos da janela do nó principal do sistema. Neste processo, a hierarquia de herança das classes de CAOS não é refletida na hierarquia de classes do Actor, havendo um processo de *achatamento* de todos os métodos e papéis herdados, através da cópia dos mesmos em cada classe de Actor gerada.

Esta característica permite não somente a implementação de aplicações de CAOS em Actor como em qualquer outra linguagem de programação orientada a objetos em que se deseja implementar a aplicação (ex. C++), ou, ainda, em linguagens de programação convencionais (neste caso, questões relacionadas ao polimorfismo de métodos devem ser cuidadosamente tratadas). Nesta estrutura de geração, é, também, possível criar um mecanismo de herança múltipla para classes em CAOS (neste caso, questões relacionadas a colisões de nomes devem ser tratadas), mecanismo este não considerado fundamental para implementação nesta primeira versão do sistema.

DOMÍNIO	SUB-DOMÍNIO	ATIVIDADE
Engenharia	Eng. Civil	Análise de Dados
Física	Eng. de Sistemas	Análise de Sistemas
Matemática	Fís.de Partículas	Aquisição de Dados
Química	Fís. Nuclear	Especificação
.	.	Programação
.	.	Projeto de Sistemas
.	.	Simulação
.	.	Visualização

FUNÇÃO	OBJETO	LINGUAGEM
Criação	Dados	C
Impressão	Documento	CAOL
Modificação	Estrutura	FORTRAN
Remoção	Gráfico	.
Rotação	Matriz	.
.	Módulo	.
.	Nó	.
.	Partícula	.
.	Programa	.
.	.	.

Tab. 1 - Exemplo de esquema de facetas

Operações permitidas em classes do ambiente CAOS são: criação, modificação e remoção de sub-classes, métodos, papéis e critérios de facetas e navegação em relações hierárquicas e relações de similaridade.

Bibliotecas em CAOS são formadas por um conjunto de rotinas. *Rotinas* são rotinas externas acopladas ao sistema. Bibliotecas são responsáveis pelo gerenciamento de suas rotinas, acoplando-as, removendo-as e permitindo a navegação a qualquer rotina especificada. O acoplamento de rotinas é feito através do fornecimento de seu nome e nome e tipo de seus parâmetros. A associação de um nó de descrição a uma rotina permite a criação de informações textuais sobre a mesma.

Uma biblioteca em CAOS é, na verdade, uma especialização da classe *Classe* e considera o conjunto de rotinas acopladas como parte das informações estruturais e comportamentais da classe, juntamente com métodos e papéis. Esta facilidade permite o atendimento do requisito de acoplamento mínimo, descrito na seção V.1.4. A classe *Biblioteca*, também, permite a criação de novos métodos e papéis, satisfazendo, assim, o requisito de acoplamento estruturado (i.e. definição de objetos mais complexos a partir de rotinas já acopladas e objetos do sistema).

Uma *aplica*ção em CAOS é um outro tipo de especialização da classe *Classe*, consistindo de uma classe com um único método. Uma aplicação pode ser vista como um programa em linguagens de programação convencional. Este conceito facilita a organização de aplicações fechadas, ativadas através de um comando de execução enviado a partir do monitor do sistema. Todo o controle do *conjunto de instâncias* criadas durante a execução de uma aplicação é feito pelo monitor, provendo um ambiente similar ao espaço de trabalho da linguagem Actor [WHIT91].

Finalmente, todas as classes, bibliotecas e aplicações geradas em CAOS são mantidas por um *gerente de objetos*. Este elemento mantém, ainda, o relacionamento entre as classes de CAOS e as classes de Actor que as implementam.

No apêndice B, incluimos um exemplo de uma aplicação escrita na linguagem de composição CAOL. Maiores detalhes sobre a implementação do protótipo do ambiente CAOS podem ser encontrados em [WERN92].

V.2. Comentários Gerais sobre o Ambiente CAOS

Nesta seção, fazemos comentários gerais sobre o ambiente CAOS. Inicialmente, apresentamos alguns dos trabalhos relacionados que serviram como principal fonte de inspiração do ambiente descrito neste capítulo. Em seguida, discutimos os principais aspectos envolvidos na experiência de desenvolvimento do protótipo do ambiente.

V.2.1. Trabalhos Relacionados

Conforme dito anteriormente, o ambiente CAOS teve suas idéias iniciais amadurecidas durante nossa participação no projeto ITHACA [PROF89] [FUGI91], cujo objetivo é construir um ambiente integrado de suporte ao desenvolvimento de aplicações orientadas a objetos. O sistema consiste de um núcleo orientado a objetos (composto de uma linguagem de programação orientada a objetos e seu ambiente de execução), uma base de informações de software, um ambiente de desenvolvimento de aplicações (composto por um conjunto de ferramentas de suporte ao desenvolvimento) e um ambiente de suporte à aplicação (composto de uma interface e facilidades para coordenação de atividades).

ITHACA propõe um modelo de desenvolvimento onde "engenheiros de aplicação" são responsáveis pelo desenvolvimento de software genérico e reutilizável, para um dado domínio de aplicação, e "desenvolvedores de aplicação" são seus clientes, reutilizando classes de objetos e "esqueletos pré-projetados de aplicações" ("frames" ou "templates") que guiam o desenvolvedor na construção de aplicações, estabelecendo padrões de construção [FUGI91] (ver seção III.3.2).

As classes de objetos e "esqueletos de aplicações" são armazenados, gerenciados e manipulados pela base de informações de software e sua ferramenta de seleção associada [CONS89]. Uma ferramenta para coleta de requisitos e especificação, denominada RECAST, negocia um "tour guiado" através da base de informações, na tentativa de construir um "esqueleto de aplicação" específico, e uma ferramenta visual de script, denominada Vista, que provê um editor gráfico de manipulação direta para a construção interativa de aplicações a partir de componentes recuperados e instanciados [MEY 91] [NIER91] [FUGI91].

Os domínios de aplicação para validação do sistema proposto são: administração pública, aplicações financeiras e sistemas de escritório. O projeto teve início em janeiro de 1989 e tem duração prevista de cinco anos.

Além da participação nas discussões técnicas no contexto do projeto ITHACA, diversos trabalhos em desenvolvimento pelo grupo de trabalho liderado pelo Prof. Tsichritzis puderam enriquecer as idéias desenvolvidas para a proposta do ambiente CAOS como, por exemplo, os trabalhos relacionados ao "Affinity Browser" de Xavier Pintado [PINT88] [PINT90] e à gerência de coleções de classes do grupo liderado por Simon Gibbs [ARAP88] [GIBB90a] [GIBB90c] [GIBB90d] [CASA90], entre outros.

Nosso trabalho no desenvolvimento do ambiente CAOS, se comparado ao ambiente proposto no projeto ITHACA, tem como principais contribuições:

1) A introdução de um mecanismo que possibilita o acoplamento de rotinas externas ao sistema, escritas em linguagens convencionais, através de seu encapsulamento.

2) A introdução da tecnologia de hipertexto, além da tecnologia de orientação a objetos encontrada em ITHACA, como alicerce da construção do ambiente, tendo se mostrado perfeitamente adequada aos conceitos envolvidos em ambientes de reutilização por composição.

3) A adoção de uma solução híbrida para o problema de composição de componentes, fornecendo um ambiente interativo e uma linguagem para suporte à atividade de composição, sendo a composição no projeto ITHACA feita somente através da ferramenta visual de script [MEY 91].

Uma outra fonte de inspiração para a elaboração da proposta do ambiente CAOS está relacionada ao trabalho desenvolvido por Prieto-Diaz na elaboração de um esquema de classificação de bibliotecas baseado em facetas [PRIE85] [PRIE87] [PRIE89] e sua aplicação prática nos Laboratórios GTE, Serviços de Dados GTE e Centro de Tecnologia Contel [PRIE88] [PRIE91].

Conforme colocado no capítulo III, Prieto-Diaz relata detalhes sobre sua experiência na aplicação do esquema proposto em grandes centros de desenvolvimento de software, chamando a atenção para os principais fatores envolvidos na introdução de uma estratégia de reutilização no desenvolvimento de software (ex. incentivos financeiros, necessidade de criação de uma infra-estrutura organizacional, etc).

Prieto-Diaz, em [PRIE91], faz uma análise de seu esquema de classificação por facetas, considerando-o mais eficiente em coleções de um domínio específico. Quando aplicado à coleções muito diversas, o esquema torna-se muito genérico e perde sua precisão descritiva. A partir da experiência prática, verificou-se a necessidade do desenvolvimento de ferramentas de suporte à criação e manutenção de esquemas de facetas.

Prieto-Diaz resume, ainda, sua experiência com bibliotecas de componentes como um excelente instrumento para auxílio na compreensão sobre a forma em que se desenvolve software em uma área de aplicação específica, sendo o esquema de classificação em facetas a possibilidade de determinar um vocabulário único de termos, utilizado por toda a equipe de desenvolvimento.

O trabalho desenvolvido por Prieto-Diaz consiste na elaboração de um esquema de classificação para recuperação de recursos de uma biblioteca e proposta de uma infra-estrutura organizacional de suporte ao uso de bibliotecas. O sistema CAOS se propõe a ser um ambiente de suporte ao desenvolvimento de aplicações baseado na reutilização por composição. Portanto, oferece-se não só um mecanismo para recuperação de componentes de uma biblioteca mas, também, outras ferramentas de suporte para o atendimento das atividades de compreensão, modificação e composição desses componentes.

Nossa experiência no desenvolvimento de software científico nos faz comprovar as diversas dificuldades, hoje, encontradas neste contexto, quer pela complexidade do domínio da aplicação, quer pelas necessidades específicas do ambiente científico (ver capítulo II). Algumas dessas dificuldades podem ser minimizadas através do uso de bibliotecas de programas.

Bibliotecas de programas científicos são comuns no ambiente científico. Entretanto, a formação dessas bibliotecas requer a criação não só de uma infra-estrutura de suporte adequada para a garantia da qualidade dos programas, documentação, gerência, manutenção, etc [FORD79] [PRIE91] [BRUN91] como, também, de ferramentas de suporte adequadas.

A proposta do ambiente CAOS surge a partir da necessidade de ferramentas de suporte à bibliotecas mais eficientes e amigáveis, que permitam uma fácil manipulação dos recursos disponíveis, utilizando-se tecnologias modernas e avançadas.

V.2.2. Comentários sobre o Desenvolvimento

Diferente do tipo de problemas encontrados no desenvolvimento do Sistema FADO, descrito no capítulo anterior, não existiram problemas relacionados à formação de equipe ou dificuldades impostas por uma área de aplicação específica, no desenvolvimento do protótipo do ambiente CAOS. As dificuldades encontradas no desenvolvimento estão relacionadas ao desenvolvimento orientado a objetos e incluem a falta de métodos e técnicas mais práticos, que estejam apoiados em ferramentas de suporte adequadas (apesar de existirem diversas propostas de métodos na literatura [BAIL89] [SEID89] [SHLA90] [PERN90] [COAD92] [BOOC91], pouco se sabe sobre a disponibilidade de ferramentas de suporte adequadas [MATT90]).

Inicialmente, foi preciso desmistificar alguns dos aspectos relacionados à programação orientada a objetos, através das seguintes constatações [NIER91]:

1) "A programação orientada a objetos (POO) é tão difícil quanto a programação convencional (i.e. com linguagens de programação imperativas, estruturadas, etc)". Talvez ela seja, ainda, mais difícil que as demais, devido à necessidade da compreensão de classes de objetos disponíveis para um aproveitamento satisfatório dos recursos da linguagem. Este fato está relacionado ao baixo nível que, ainda, precisa-se trabalhar na implementação de aplicações (i.e. programando-as).

2) "O sucesso da POO é altamente sensível aos resultados da análise e projeto orientado a objetos." Visto que os atuais métodos dependem muito da aptidão de quem os aplica, baseando-se em regras muito empíricas e informais. Portanto, o resultado obtido em POO é um reflexo direto da capacidade do programador em programar orientado a objetos.

3) "Não existem garantias de que o software gerado seja reutilizável". Embora a POO dê subsídios para o encapsulamento, composição e evolução de objetos, nada pode ser dito quanto a qualidade final do software gerado, sendo comum a necessidade de recriar objetos e classes devido a incapacidade de adaptá-los aos novos requisitos.

Uma outra dificuldade encontrada foi a falta de ferramentas de suporte à programação orientada a objetos que pudessem ser especializadas de forma a satisfazer os requisitos do ambiente proposto. Portanto, foi preciso desenvolver as ferramentas de suporte à reutilização de software, praticamente, a partir do nada.

As atuais linguagens de programação orientada a objetos, apesar de basearem o desenvolvimento de aplicações a partir de classes de objetos já existentes, oferecem poucas ferramentas de suporte às atividades relacionadas à busca, compreensão e modificação de classes de objetos. Linguagens como Smalltalk [GOLD84] e Actor

[WHIT91] provêm um "browser" para auxílio na exploração das classes disponíveis do sistema. Entretanto, apesar de útil, esta ferramenta está longe de atender todos os requisitos descritos neste capítulo. Existem, ainda, algumas bibliotecas de classes disponíveis para linguagens tal como C++, Objective-C ou Eiffel, que servem de base para o desenvolvimento de novas aplicações. Estas bibliotecas, entretanto, não possuem nenhum tipo de ferramenta interativa para auxílio na manipulação das mesmas [ARAP88].

V.3. Reutilização de Software no Ambiente CAOS

A elaboração de um ambiente orientado a objetos para composição de aplicações científicas teve como principal motivação fornecer um conjunto de ferramentas adequadas ao desenvolvimento baseado em bibliotecas de componentes, visto que as atuais bibliotecas científicas não atendem completamente às necessidades deste tipo de desenvolvimento.

Um ambiente de suporte a uma biblioteca de componentes pertence ao grupo de tecnologias de composição para reutilização de software e deve fornecer ferramentas adequadas à busca, compreensão, modificação e, principalmente, composição de componentes.

O ambiente proposto baseou-se no uso de conceitos e técnicas de orientação a objetos, enfatizando os aspectos envolvidos na construção de aplicações a partir de componentes bem definidos, atômicos, previamente definidos. Nossa experiência no desenvolvimento do primeiro protótipo do sistema mostrou que a adoção de um paradigma de orientação a objetos, por si só, não garante a reutilização de software por composição, sendo necessário o desenvolvimento de todo um conjunto de

ferramentas de suporte às atividades inerentes à reutilização por composição e de uma metodologia correspondente.

O ambiente introduziu, ainda, o uso da tecnologia de hipertexto como base de sua construção, servindo como estrutura organizacional dos elementos do sistema. Esta abordagem mostrou-se, perfeitamente, adequada à implementação dos conceitos envolvidos em ambientes de composição.

Um aspecto a ser enfatizado, quando se discute reutilização de software por composição, diz respeito à diferença entre "software reutilizado" e "software reutilizável", bem apresentada por Tracz em [TRAC90]. Existe uma grande diferença entre a possibilidade de se reutilizar software já existente, quando este não foi construído com o objetivo de ser reutilizado posteriomente, e de se desenvolver software de tal forma que este se torne reutilizável. Tracz prefere não considerar a reutilização do primeiro tipo de software. Entretanto, em ambientes tradicionais, como é o caso do ambiente científico e comercial, é este o tipo de software predominante e é este o tipo de software que se deseja reutilizar em um primeiro instante. Reconstruir tudo que já existe torna-se impraticável. Ambientes deste tipo só podem considerar a adoção de uma nova tecnologia com o intuito de melhorar o desenvolvimento de novas aplicações, mas precisam contar com mecanismos que permitam a comunicação entre as novas aplicações e as já existentes.

O ambiente CAOS, descrito neste capítulo, teve como um de seus objetivos atender este requisito, permitindo o acoplamento de rotinas externas ao sistema e assumindo uma transição gradual entre o desenvolvimento tradicional e o orientado a objetos. Para isso, foi preciso construir um ambiente fácil de ser usado, onde as atividades envolvidas no uso de bibliotecas de componentes são devidamente suportadas por ferramentas especializadas.

Tecnologias de composição oferecem mecanismos mais flexíveis para a reutilização de software do que as tecnologias de geração (ver seção IV.4), permitindo seu uso em diferentes domínios de aplicação. Entretanto, sua utilização requer o conhecimento de aspectos operacionais de mais baixo nível que as tecnologias de geração, podendo se tornar de mais difícil aprendizagem para usuários leigos em computação.

Trabalhos existentes na literatura [ISCO88] [PRIE90] indicam a criação de bibliotecas específicas a certas áreas de aplicação mais eficientes que a criação de bibliotecas genéricas. Neste caso, é possível criar linguagens que incorporam conceitos e abstrações existentes no domínio de aplicação específico e que são capazes de manipular adequadamente estas bibliotecas [NEIG91]. Tais observações nos levam a sugerir o uso de técnicas associadas tanto às tecnologias de composição como tecnologias de geração para o atendimento dos requisitos de um ambiente científico (ver seção VI.4).

Ferramentas de suporte à reutilização por composição, do tipo da apresentada neste capítulo, são bastante úteis ao desenvolvimento de software científico devido aos seguintes aspectos:

- tal como sugerido por Prieto-Diaz [PRIE91], bibliotecas de componentes servem como excelente instrumento de auxílio à compreensão de como é desenvolvido software em uma dada aplicação (similarmente à experiência no desenvolvimento da ferramenta de geração, ver seção IV.4);
- mecanismos genéricos para o atendimento das atividades de busca, compreensão, modificação e composição de aplicações dão flexibilidade ao uso de tecnologias de composição em diferentes áreas de aplicação, e

- o desenvolvimento de software, a partir de uma biblioteca de componentes de boa qualidade, além de facilitar o desenvolvimento em geral, reduz o custo e o tempo envolvidos neste processo e tem como resultado produtos de qualidade que atendem aos padrões de Engenharia de Software.

VI. Conclusões

Esta tese foi, parcialmente, desenvolvida durante a permanência no CERN e no CUI (Centro Universitário de Informática, Universidade de Genebra), em regime de doutorado "sanduíche", tendo sido finalizada na COPPE/UFRJ (Programa de Engenharia de Sistemas e Computação). Seu objetivo foi discutir duas estratégias para reutilização, propondo uma ferramenta para cada estratégia, e estabelecer as potencialidades, limites e adequação quanto ao uso das mesmas no contexto do desenvolvimento de software científico.

Fêz-se, inicialmente, um estudo sobre as principais dificuldades, atualmente, encontradas no desenvolvimento de software científico. A partir desse estudo, estabeleceu-se a hipótese da adotação de uma estratégia de reutilização no desenvolvimento de software científico, a fim de minimizar os problemas encontrados.

Após um estudo sobre o atual estado da arte em reutilização de software, apresentou-se duas experiências práticas no desenvolvimento de ferramentas de suporte à reutilização no contexto científico. A primeira delas, uma linguagem de quarta-geração (Fado), mostrou-se bastante útil devido a diversos aspectos, principalmente pela incorporação de conceitos e abstrações de muito alto nível específicos à área de aplicação da Física, sendo por isso de fácil utilização.

O projeto da linguagem Fado foi baseado em conceitos e abstrações da lógica matemática e da teoria de conjuntos por ser considerada uma plataforma apropriada para a descrição das propriedades de um evento em Física de Altas Energias e natural o suficiente para ser usada por físicos. A linguagem é acompanhada de um ambiente de suporte à programação, que facilita a criação de programas em Fado, e está

inserida no contexto de um sistema para seleção de eventos em linha, também denominado FADO [BARA88].

A reutilização de código no sistema FADO se dá através do desacoplamento da estratégia de seleção de eventos físicos da implementação do processo de reconstrução. Desta forma, todo o conhecimento sobre este processo, incluindo a manipulação de estruturas de dados e algoritmos associados, são implicitamente reutilizados pelo sistema. Além disso, a arquitetura do sistema como um todo facilita bastante sua manutenção, seja em termos da evolução do sistema em relação ao processo de reconstrução ou na definição de novas estratégias para seleção de eventos.

A segunda ferramenta apresentada, o ambiente CAOS, baseia-se em uma tecnologia de composição para reutilização de software. Este ambiente é um exemplo de ambiente de suporte à biblioteca de componentes e provê um conjunto de ferramentas de suporte para todas as atividades envolvidas na busca, compreensão, modificação e composição de componentes.

O ambiente implementado baseou-se em conceitos e técnicas de orientação a objetos devido aos diversos aspectos incorporados neste paradigma que facilitam a reutilização de componentes de software (i.e. objetos, classes, herança, generalização, etc). A programação orientada a objetos, entretanto, não garante a reutilização de software, sendo necessária a implementação de ferramentas especializadas de suporte às diversas atividades envolvidas no processo de reutilização por composição. Poucos são os ambientes deste tipo, atualmente, disponíveis.

Uma das principais contribuições na implementação desta ferramenta de composição foi colocar lado a lado as tecnologias de orientação a objetos e

hipertextos como alicerces para construção de um ambiente de composição. O trabalho desenvolvido no projeto ITHACA [PROF89], do qual participamos, contribuiu para a discussão dos principais aspectos da aplicação da tecnologia de orientação a objetos neste contexto, abrindo uma série de oportunidades para pesquisas na área. A introdução da tecnologia de hipertextos mostrou-se, também, bastante oportuna na medida em que abre, ainda, outras oportunidades de pesquisa, tendo se mostrado perfeitamente adequada aos conceitos envolvidos em ambientes de reutilização por composição.

A adoção do esquema de classificação por facetas, proposto por Prieto-Díaz [PRIE85], adaptado para recuperação de classes de objetos, abriu caminhos para pesquisas relacionadas à recuperação de classes de objetos para reutilização. Nesta linha já está definida, como continuação de nosso trabalho, uma tese de mestrado para avaliação de outros esquemas existentes quanto a sua adequação em ambientes orientados a objetos, devendo resultar na proposta final de um esquema para recuperação de componentes de software neste contexto.

No contexto científico, o ambiente CAOS contribui no sentido de prover um instrumento mais completo, para reutilização de rotinas científicas, do que as atuais bibliotecas de rotinas. Isso se dá na medida em que provê uma série de ferramentas que facilitam a procura, compreensão, modificação e composição de rotinas, previamente encapsuladas em objetos do sistema, além de permitir o uso gradual de facilidades encontradas no desenvolvimento orientado a objetos.

Conforme já observamos anteriormente, a proposta de adoção de uma estratégia de reutilização no desenvolvimento de software científico tem como principal objetivo livrar, ao máximo, profissionais dos diversos ramos da ciência de

atividades puramente de programação, para que estes possam se dedicar mais intensamente às atividades de sua especialidade.

Uma proposta deste tipo, entretanto, deve considerar uma série de fatores, que incluem:

- a criação de grupos de suporte à reutilização;
- a definição de um modelo de ciclo de desenvolvimento para software científico que considere questões de reutilização;
- o estabelecimento de incentivos à reutilização, e
- o desenvolvimento de ferramentas de suporte apropriadas.

Concluindo este trabalho, apresentamos brevemente nossa opinião sobre cada um desses fatores, incluindo algumas perspectivas quanto à continuidade do trabalho de desenvolvimento de ferramentas de suporte apropriadas e indicando algums dos problemas, atualmente, encontrados no desenvolvimento de software científico que podem ser minimizados com a adoção de tal estratégia.

VI.1. Grupos de Suporte à Reutilização

A partir da necessidade de criação de grupos de suporte à reutilização, apresentados na seção III.3.1, sugerimos os seguintes objetivos, atividades e formação de equipes a cada um desses grupos no contexto científico:

a) o *grupo de suporte gerencial* deve estabelecer políticas, procedimentos e regras para reutilização adequadas a este ambiente, resolvendo inclusive questões relacionadas ao financiamento de projetos de sofware reutilizável. Esta equipe deve ser formada por pessoas que tenham uma boa visão sobre todas as atividades desenvolvidas na

instituição científica e bom conhecimento sobre aspectos políticos, gerenciais e técnicos.

b) o objetivo do *grupo de identificação e qualificação* deve ser manter a qualidade do acervo de produtos reutilizáveis e identificar áreas potenciais para reutilização. Sua principal atividade é avaliar a qualidade de componentes de software candidatos à inclusão em uma base de componentes reutilizáveis, assim como a qualidade de ferramentas de suporte, adquiridas ou desenvolvidas, sugeridas para uso no processo de reutilização. Caso o produto avaliado não apresente o grau de reutilizabilidade necessário, este grupo deve ser capaz de propor alterações de forma que as características determinantes para reutilização possam ser incorporadas. Devido aos diversos aspectos de qualidade a serem avaliados, tanto no que diz respeito à forma como ao conteúdo, esta equipe deve ser formada por especialistas na área de computação e na área de aplicação específica, sendo treinados em atividades de avaliação e acompanhamento de projetos, técnicas de inspeção, "walkthrough", etc [ACKE84] [FAGA86].

c) o *grupo de manutenção* deve se responsabilizar somente pela manutenção e atualização daqueles produtos que fazem parte do acervo de software reutilizável, sendo que dependendo do tipo de manutenção exigida (i.e. adaptação ou evolução de algoritmos) o(s) autor(es) do componente deve(m) ser envolvido(s) nesta atividade. Assumindo que a maior parte das atividades de manutenção estará relacionada aos aspectos de computação (i.e. criação ou manutenção de versões específicas para diferentes equipamentos), este grupo poderá ser formado, basicamente, por especialistas em computação.

d) o *grupo de desenvolvimento*, responsável pela criação de novos componentes reutilizáveis, pode ser formado por contribuintes autônomos, que ao

desenvolverem um certo produto o submetem ao grupo de qualificação, para determinação de seu grau de reutilizabilidade, ou por grupos de interesse criados em certas áreas de aplicação para promover o desenvolvimento de software reutilizável na área. Em ambos os casos, as atividades de desenvolvimento poderão ser supervisionadas pela equipe de qualificação. Este grupo, portanto, será formado, basicamente, por especialistas na área de aplicação, podendo contar com o apoio de especialistas do grupo de suporte à desenvolvedores de sistemas.

e) o *grupo de suporte à desenvolvedores de sistemas* representa o elo de comunicação entre os *produtores* de software e os *consumidores*, isto é, aqueles que promovem a prática de reutilização na comunidade científica, divulgando o software existente, dando assistência técnica e treinando pessoas para o uso de métodos, técnicas e ferramentas de reutilização. Esta atividade é essencialmente computacional, porém, pode exigir conhecimento mais detalhado sobre o conteúdo dos componentes ou ferramentas de software específicas a uma certa área de aplicação. Portanto, a inclusão de especialistas de diferentes áreas de aplicação deve ser incentivada.

Sugerimos, ainda, a criação de um outro grupo de identificação e desenvolvimento de ferramentas de suporte à reutilização. Este grupo seria responsável pela proposta de ferramentas de suporte à reutilização a serem utilizadas no desenvolvimento de software, a partir da avaliação de novas ferramentas adquiridas ou desenvolvidas para atender os requisitos da comunidade científica. A atividade de desenvolvimento de ferramentas de suporte é uma atividade tipicamente computacional e, portanto, desenvolvida por especialistas em computação. Durante a avaliação das ferramentas, propõe-se a colocação das mesmas para uso e avaliação por parte de especialistas em áreas de aplicação específicas por um certo período.

Das alternativas propostas por Perry [PERR83] para a formação da maioria destes grupos, consideramos a combinação de profissionais trabalhando em regime de tempo integral com profissionais em tempo parcial a mais adequada para adoção no ambiente científico. Desta forma, será possível manter uma pequena equipe permanente, formada por especialistas da área de computação e de áreas de aplicação científica, podendo contar com a inclusão de membros em tempo parcial para o acréscimo de conhecimento específico em determinados projetos.

VI.2. Modelo de Desenvolvimento

Para que a adoção de uma estratégia de reutilização possa se dar de forma efetiva, é preciso que se estabeleça um modelo comum de desenvolvimento de software científico, adaptando-o aos aspectos de reutilização de software, conforme discutido no capítulo III.

Neste sentido, pode-se assumir a adoção de um modelo tal como o proposto por Rocha [ROCH88] (ver seção II.2.3), sendo necessárias algumas adaptações às questões de reutilização. Na medida em que a segunda etapa deste modelo (etapa de produção) segue, exatamente, as fases do ciclo de desenvolvimento tradicional, as adaptações sugeridas no capítulo III (seção III.3.2) podem ser igualmente utilizadas.

Na primeira etapa (etapa de pesquisa), observando-se a reutilização do conhecimento como base do processo científico (ver fases da atividade científica na seção II.1), propõe-se que as atividades relacionadas à reutilização sejam destacadas e colocadas como atividades específicas, cujo objetivo é reutilizar ao máximo o conhecimento já existente ou adquirido durante o processo científico (ex. fase de coleta de dados e exame, síntese de soluções-tentativas, extração de conceitos úteis à solução do problema na fase de análise e otimização, etc).

VI.3. Incentivos à Reutilização

Os incentivos financeiros à reutilização de software, descritos no capítulo III, sob a forma de pagamentos ao autor de programas aceitos para inclusão no acervo de software reutilizável, promoções, prêmios ou aumentos no orçamento de projetos que alcançam um grande percentual de componentes reutilizados, funciona em um ambiente onde o aumento da produtividade, através de uma estratégia de reutilização de software, implica diretamente em um aumento no lucro da empresa, seja pela redução dos gastos do projeto e de seu ciclo de desenvolvimento ou pela obtenção de produtos de melhor qualidade, confiáveis, manuteníveis, competitivos, etc.

No contexto científico, a motivação para reutilização de software, a princípio, tem bases diferentes daquela do contexto comercial e diz respeito à satisfação pessoal de indivíduos, facilidade no desenvolvimento de novas aplicações, solução de questões relacionadas aos aspectos de complexidade, manutenção e características gerais da qualidade do software resultante, possibilitando a liberação de especialistas dos diferentes ramos da ciência de atividades puramente computacionais.

Apesar das diferentes motivações, ambos ambientes precisam solucionar os aspectos relacionados aos custos adicionais inseridos na adoção de uma estratégia de reutilização.

No caso do ambiente científico, pode-se considerar a proposta de Bollinger e Pfleeger [BOLL90] (ver seção II.3.1), sobre a criação de domínios de compartilhamento de custos, os chamados *CSDs*, para atender às necessidades do desenvolvimento de componentes de software retilizáveis e ferramentas de suporte específicas a um certo grupo de interesse. Neste caso, entretanto, o pagamento do

financiamento, através da entrega do produto a um *banco CSD*, não tem sentido no atual contexto científico, na medida em que não é possível recuperar o dinheiro gasto no desenvolvimento de produtos reutilizáveis através de sua revenda. O resultado da criação de melhores produtos, reutilizáveis, no ambiente científico, traduz-se, portanto, no prestígio criado perante a comunidade científica, por deter um acervo de software de grande valor para esta comunidade e na minimização de diversas dificuldades, hoje, encontradas no desenvolvimento de sofware.

Logo, pode-se dizer que a reutilização no desenvolvimento de software, em ambientes comerciais, significa a possibilidade de redução dos custos associados a um certo projeto, enquanto que, no ambiente científico, uma estratégia deste tipo representa a possibilidade de construção de melhores produtos, talvez, pelo mesmo custo.

VI.4. Ferramentas de Suporte

Um outro aspecto bastante determinante, na discussão sobre a adoção de uma estratégia de reutilização no desenvolvimento de software, está associado à disponibilidade de ferramentas de suporte adequadas às atividades envolvidas no processo de reutilização. Este aspecto pode, inclusive, implicar na criação de mais um grupo na lista de grupos de suporte à reutilização, conforme dito anteriormente. Este grupo seria responsável pela identificação e desenvolvimento de ferramentas apropriadas para suporte à reutilização.

Nos capítulos IV e V, descrevemos o desenvolvimento de duas ferramentas de suporte à reutilização no contexto científico, baseadas em cada um dos tipo de tecnologias, hoje, conhecidos (i.e. tecnologias de geração e tecnologias de

composição). Mostrou-se que as duas abordagens e, também, ambas ferramentas são adequados ao uso neste contexto.

Dado o atual estado da arte em reutilização de software, não existem, ainda, ferramentas de suporte comercialmente disponíveis, no que se refere a domínios de aplicação específicos e nem mesmo em termos gerais. No capítulo III, apresentamos os trabalhos, atualmente, considerados mais relevantes no desenvolvimento de ferramentas genéricas para reutilização [PRIE85] [NEIG85] [CLEA88]. Nota-se, entretanto, uma escassez absoluta de trabalhos referentes a ferramentas específicas a determinadas áreas de aplicação. Isto é, particularmente, verdade no que se refere à área científica.

O trabalho desenvolvido nesta tese serve como base para discussão sobre o desenvolvimento de ferramentas para reutilização aplicadas ao domínio científico. Além dos trabalhos apresentados na concepção de ferramentas de geração e de composição, acreditamos ser de extrema importância a construção de ferramentas capazes de integrar as técnicas associadas a ambos os tipos de tecnologias (i.e. sistemas híbridos), aproveitando as características mais importantes de cada uma delas e minimizando algumas das dificuldades encontradas.

Como um possível seguimento futuro deste trabalho, propomos a construção de um sistema híbrido capaz de reunir diferentes linguagens de especificação de aplicações, baseadas no projeto de linguagens de muito alto nível, específicas a diferentes áreas de aplicação, utilizando um mecanismo genérico para composição automática de aplicações. Este mecanismo, a partir de bases de componentes específicas, contendo informações sobre a arquitetura de aplicações das diferentes áreas, guiará o processo de especificação.

Terminada a especificação da aplicação, o sistema construirá, automaticamente, uma proposta de solução, a partir da composição de componentes já existentes na base. Esta solução é, então, apresentada ao usuário que poderá modificá-la através da substuição ou adaptação de componentes. Caso o sistema não possa gerar uma proposta de aplicação, devido à inexistência de determinados componentes na base, o usuário será comunicado quanto à necessidade de construção de novos componentes, sendo apresentados alguns possíveis candidatos para adaptação, de forma a satisfazer os requistos, inicialmente, determinados.

A *figura 14* mostra, esquematicamente, os elementos envolvidos em um sistema deste tipo.

A proposta de construção de um sistemas como este está, ainda, distante do atual estado da arte em reutilização consistindo, portanto, em interessante tema de pequisa. Esta proposta, no entanto, integra uma série de conceitos já encontrados em sistemas tais como [NEIG85] [KATZ89] [ISCO88] [SOUZ90].

Resultados significativos de pesquisas relacionadas à integração de tecnologias de geração e de composição são esperados nos próximos anos na área de reutilização de software [TRAC88a] [NEIG89] [FARI91]. Alguns autores [BATZ83] [JONE88] afirmam que a tecnologia de composição deve ser considerada como o objetivo a curto prazo, pois consiste em criar os primeiros blocos de contrução de determinados domínios de aplicação. A tecnologia de geração, portanto, pode ser vista como o objetivo a longo prazo, na medida em que só somos capazes de construir sistemas de geração quando conhecemos bem suas partes. De qualquer forma, a criação de mecanismos capazes de reunir o alto nível de abstração e facilidade de uso das tecnologias de geração com a flexibilidade e manutenibilidade da estrutura de

tecnologias de composição proporcionará um grande avanço na construção de ferramentas de suporte à reutilização.

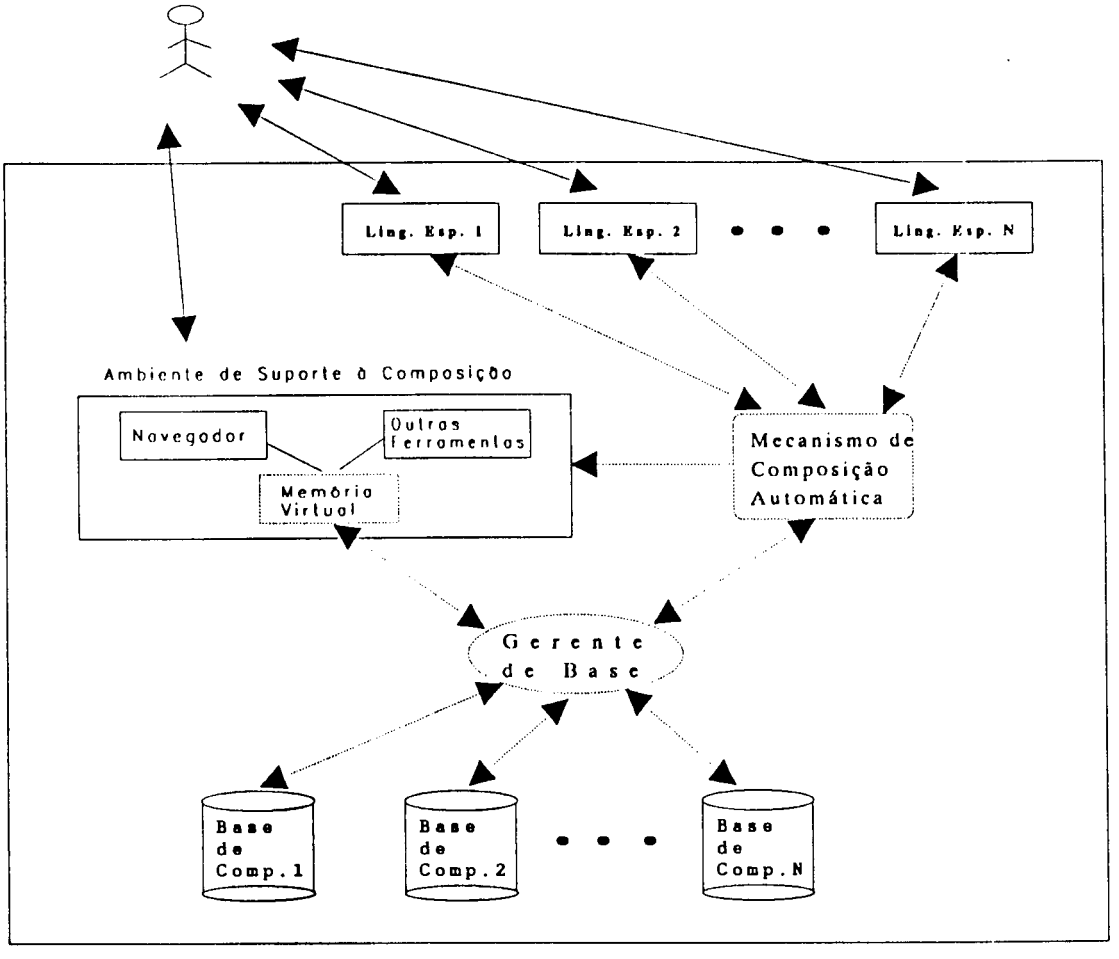


Fig. 14 - Elementos do sistema híbrido

Os resultados obtidos através da pesquisa na área de análise de domínios [PRIE90] [NEIG91], acrescidos dos resultados da pesquisa de mecanismos concretos que tornam viável a construção de sistemas genéricos, irão possibilitar, a médio prazo, a solução de grande parte dos problemas de caráter técnico da reutilização de software. Espera-se que, ao longo deste percurso, os fatores psicológicos, sociológicos e econômicos que impedem, atualmente, a reutilização possam, também, ser resolvidos, para que ao final da década de 90 tenhamos um quadro de intensa reutilização no desenvolvimento de software nas mais diversas áreas de aplicação.

VI.5. Minimização de Problemas no Desenvolvimento

Finalmente, dentre os problemas que podem ser minimizados com a adoção de uma estratégia de reutilização no contexto científico, temos:

- questões relacionadas à qualidade do software gerado, através da criação de grupos especializados de suporte à reutilização, do fornecimento de ferramentas adequadas, da construção de novas aplicações a partir de software garantido pelos grupos de suporte e da implantação de uma metodologia básica para o desenvolvimento de software científico;
- questões relacionadas ao diálogo entre especialistas da área de aplicação e de computação, a partir da padronização de conceitos, termos e arquiteturas de aplicações utilizados;
- questões relacionadas à manutenção e evolução de sistemas, na medida em que a melhoria na estrutura e qualidade do software gerado facilitará a manutenção e evolução dos mesmos, e
- questões relacionadas à perda de tempo de especialistas no domínio da aplicação com aspectos, essencialmente, técnicos de Ciência da Computação, na medida em que estes terão disponível o suporte de ferramentas computacionais e grupos especializados.

Para que iniciativas deste tipo possam se tornar uma realidade no ambiente científico, é preciso que gerentes de projetos considerem os aspectos computacionais fundamentais ao desenvolvimento da atividade científica. Esta conscientização já começa a surgir em algumas comunidades científicas [CERN89d] [KUNZ89] [LOKE89] [LOKE91].

Um outro fator importante diz respeito à formação de profissionais capazes de atuar nos diferentes grupos de suporte à reutilização, sendo necessária a formação de engenheiros, físicos, matemáticos, etc, cuja função específica é trabalhar com aspectos computacionais, não sendo, portanto, descriminados perante a comunidade por desenvolver tal função [CROS86]. O mesmo pode ser dito sobre a especialização de determinados profissionais de informática em certos domínios de aplicação, reduzindo-se, assim, as dificuldades de diálogo entre elementos de diferentes especialidades necessários ao desenvolvimento de software científico.

Esperamos que os resultados obtidos nesta tese sirvam como instrumentos para a compreensão sobre a importância da reutilização de software no desenvolvimento de software científico, traduzindo-se na possibilidade de solução de diversas dificuldades, hoje, encontradas, e que uma estratégia de reutilização possa, em breve, ser estabelecida neste contexto.

Referências Bibliográficas

[ACKE84] Ackerman, A.F.; e outros "Software Inspections and the Industrial Production of Software", em "Software Validation", (ed.) Hausen, H.L., North-Holland, 1984.

[AGRE88] Agresti, W.W.; McGarry, F.E. "The Minnowbrook workshop On Software Reuse: A Summary Report", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[ALLA87] Allaby, J. "Data Acquisition and Analysis at LEP", CERN School of Computing, (ed.) Verkerk, C.; Troia, Portugal, 1987.

[ALLE84] Allen, J.R.; Kennedy, K. "PSC: A Program to Convert FORTRAN to Parallel Form", em "Supercomputers: Design and Application" (ed.) Hwang, K.; IEEE Computer Science Press, 1984.

[ALMA89] Almasi, G.S.; Gottlieb, A. "Highly Parallel Computing", The Benjamin/Cummings Publishing Company, 1989.

[ALVE90] Alves Filho, J.S.R.; Devloo, P. "Object-Oriented Programming in Scientific Computations: the beginning of a new era", XI Congresso Ibero Latino Americano sobre Metodos Computacionais para Engenharia, Rio de Janeiro, outubro 1990.

[ARAP88] Arapis, C.; Kappel, G. "Organizing Objects in an Object Software Base", em "Active Object Environments", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, junho 1988.

[ARSA84] Arsac, J. "Informatique et Calcul Scientific", em "Tools, Methods and Languages for Scientific and Engineering Computation", (eds.) Ford, B. e outros, Elsevier Science Publishers, 1984.

[ARTH85] Arthur, L.J. "Measuring Programmer Productivity and Software Quality", John Wiley & Sons, 1985.

[ATAS89] Atas do Curso de Ambiente de Desenvolvimento para Software Científico, Genebra, Suiça, 1989.

[ATWO90] Atwood, W.; e outros "The Reason Project", International Conference on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, Lyon, França, 1990.

[ATWO91] Atwood, W.; e outros "The Gismo Project: Applications of OOP to HEP Detector Design, Simulation, and Reconstruction", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[BAIL89] Bailin, S. "An Object-Oriented Requirements Specification Method", Communications of the ACM, vol.32, no.5, maio 1989.

[BAHI88] Bahia, A.S.; Controle de Qualidade do Software para a Área Científica, II Simpósio Brasileiro de Engenharia de Software, outubro 1988.

[BALZ85] Balzer, R. "A 15 Year Perspective on Automatic Programming", IEEE Transactions on Software Engineering, vol.SE-11, no.11, novembro 1985.

[BARA88] Barao, F.; e outros "FADO Emulators Tagging System", DELPHI note 88-55 DAS 87, Genebra, Suiça, 1988.

[BART87] Bartholdi, P. "Applications of Computers in Astronomy", CERN School of Computing, (ed.) Verkerk, C.; Troia, Portugal, setembro 1987.

[BASI91] Basili, V.R.; Rombach, M.D. "Support for Comprehensive Reuse", Software Engineering Journal", setembro 1991.

[BATI71] Batiste, E.L. "The Production of Mathematical Software for a Mass Audience" em "Mathematical Software" (ed.) Rice, J.R., Academic Press, New York, 1971.

[BATZ83] Batz, J.C. e outros "The Application-Specific Task Area", IEEE Computer, vol.16, no.11, novembro 1983.

[BELL85] Bell, K. "Some Thoughts on Design, Development and Maitenance of Engineering Software", em "Engineering Software IV", (ed.) Adey, R.A.; Springer-Verlag, Berlin, 1985.

[BERS91] Bersoff, E.; Davis, A. "Impacts of Life Cycle Models on Software", Communications of the ACM, agosto 1991.

[BERT86] Bertrand, D.; Pape, L. "TANAGRA, Track Analysis and Graphics Package Manual", DELPHI note 86-75 PROG 55, 1986.

[BIGG89] Biggerstaff, T.J.; Richter, C. "Reusability, Framework, Assessment, and Directions", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[BOCK89] Bock, R.K. "Bringing together high energy physicist and computer scientist. A summary of the Oxford conference on Computing in High Energy Physics", Computing in High Energy Physics Conference, Oxford, Inglaterra, 1989.

[BOEH78] Boehm, B.W. "Characteristics of Software Quality", North-Holland, 1978.

[BOLL90] Bollinger, T.B.; Pfleeger, S.L. "Economics of reuse: issues and alternatives", Information and Software Technology, vol.32, no.10, dezembro 1990.

[BOLT70] Bolt, B.A. "The Use of Computers in Studies of the Earth", em "Computers and their role in the Physical Sciences" (eds.) Fernbach, S.; Taub, A.; Gordon & Breach Science Publishers, New York, 1970.

[BOOC91] Booch, G. "Object Oriented Design", The Benjamin/Cummings Publishing Company, Inc, 1991.

[BROW85] Browning, D.J.; e outros "Software Systems Development in Petroleum Engineering", Computer Physics Communications, 38, North-Holland, Amsterdam, 1985.

[BRUN78] Brun, R.; e outros "GEANT: Simulation Program for Particle Physics Experiments - User Guide and Reference Manual", CERN-DD/78/2, 1978.

[BRUN87] Brun, R.; Zoll, J. "ZEBRA User Guide", 1987.

[BRUN88] Brun, R.; Lienart, D."HBOOK Users Guide, 1988.

[BRUN89] Brun, R.; e outros "CMZ - a source code management system", Computing in High Energy Physics Conference, Oxford, Inglaterra, 1989.

[BRUN90] Brun, R.; e outros "Visualization of Scientific Data for High Energy Physics: PAW, a General-Purpose Portable Software Tool for Data Analysis and Presentation", International Conference on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, Lyon, França, 1990.

[BRUN91] Brun, R.; e outros "Experience in Managing a Large Scientific Library", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[BUNN89] Bunn, J. "FLOPPY User Guide", CERN/DD/US/12, 1989.

[BURT87] Burton, B.; e outros "The Reusable Software Library", IEEE Software, julho 1987.

[CAIL90] Cailliau, R. "Languages for HEP", International Conference on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, Lyon, França, 1990.

[CASA90] Casais, E. "Managing Class Evolution in Object-Oriented Systems", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[CERN87a] Brochura CERN "Presenting CERN", European Laboratory for Particle Physics - Publications Section, Genebra, Suiça, 1987.

[CERN87b] Brochura CERN "CERN/LEP", European Laboratory for Particle Physics - Publications Section, Genebra, Suiça, 1987.

[CERN89a] Brochura CERN "The LEP in short", (eds.) Buhler-Broglin, M.; Lajust, D.; Lewis, R.; Sadag, Bellegarde, França, 1989.

[CERN89b] CERN "Large Electron-Positron storage ring - Technical Report", CERN Publications, Genebra, Suiça, 1989.

[CERN89c] CERN Computer Centre "CERN Program Library Manual", Genebra, Suiça, 1989.

[CERN89d] CERN "Computing at CERN in the 1990s", Genebra, Suiça, 1989.

[CERN89e] CERN "Computing for Experiments Final Report", em "Computing at CERN in the 1990s", Genebra, Suiça, 1989.

[CHEA89] Cheatham Jr., T.E. "Reusability through Program Transformations", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[CLEA88] Cleaveland, J.C. "Building Application Generators", IEEE Software, vol.5, no.4, julho 1988.

[COAD92] Coad, P.; Yordon, E.; Análise Baseada em Objetos, (trad.) CTI Informática, Editora Campus, 1992.

[CODY71] Cody, W.J. "Software for the Elementary Functions" em "Mathematical Software" (ed.) Rice, J.R. , Academic Press, New York, 1971.

[COMM91] Commerlato, C.A.; Uma Ferramenta para Avaliação da Qualidade de Programas Fortran, Workshop em Desenvolvimento, Avaliação e Manutenção de Software Científico, COPPE/UFRJ, Rio de Janeiro, 1991.

[CONK87] Conklin, J. "Hypertext: a survey and introduction", Computer, vol.20, no.9, setembro 1987.

[CONS89] Constantopoulos, P. e outros "The Ithaca Software Information Base: Requirements Functions and Structuring Concepts", Ithaca Report ITHACA.FORTH.89.E2#1, 1989.

[COX 86] Cox, B.J. "Object-Oriented Programming: An Evolutionary Approach", Addison-Wesley, 1986.

[CROS86] Cross, M.; e outros "Software engineering methodologies for Scientific and Engineering computation", Appl. Math. Modelling, vol.10, 1986.

[DAVI88] Davis, A.M.; e outros "A Strategy for Comparing Alternative Software Development Life Cycle Models", IEEE Transactions on Software Engineering, vol.14, no.10, outubro 1988.

[DELP84] DELPHI note 84-3 PROG 2 "Final Report of the DELPHI Software Planning Group", Genebra, Suiça, 1984.

[DELP85] DELPHI Collaboration "Status Report on Data Acquisition & Computing" DELPHI note 85-17 GEN-20, Genebra, Suiça, 1985.

[DELP89] DELPHI note 89-92 DAS 101 "DELPHI Event Tagging", Genebra, Suiça, 1989.

[DELP90] DELPHI Collaboration "The DELPHI Detector at LEP", CERN-PPE note 90-128, Genebra, Suiça.

[DICK71] Dickinson, A.W.; e outros "The Development and Maintenance of a Technical Subprogram Library" em "Mathematical Software" (ed.) Rice, J.R. , Academic Press, New York, 1971.

[DIGI87] Digital Equip. Corporation "Guide to VAX LSE and VAX SCA", Massachusetts, 1987.

[DIGI88a] Digital Equip. Corporation "A Methodology for Software Development Using VMS Tools", Massachusetts, 1988.

[DIGI88b] Digital Equip. Corporation "VAX Text Processing Utility Reference Manual", Massachusetts, 1988.

[DRUM89] Drummond, R.; LegoShell: Linguagem de Computações, Relatório Técnico do Projeto A_HAND, Departamento de Ciência da Computação, UNICAMP, 1989.

[FAGE86] Fagan, M.E. "Design and code inspections to reduce errors in program development", IBM Systems Journal, vol.15, no.3, 1986.

[FARI91] Faria, E.C. "MARTE: Um Meta-gerador de Aplicações baseado na Reutilização de Templates", Tese de Mestrado da COPPE/UFRJ, setembro 1991.

[FORD79] Ford, B.; e outros "The NAG Library Machine", Software-Practice and Experience, vol.9, fevereiro 1979.

[FREE80] Freeman, P. "Reusable Software Engineering: A statement of Long-Range Research Objectives", Tech.Rep. TR-159, Irvine, Univ. California, ICS Dept., 1980.

[FREE87a] Freeman, P. "A Perspective on Reusability", em "Tutorial: Software Reusability" (ed.) Freeman, P.; 1987.

[FREE87b] Freeman, P. "Reusable Software Engineering: Concepts and Research Directions", em "Tutorial: Software Reusability" (ed.) Freeman, P.; 1987.

[FUGI91] Fugini, M.G.; e outros "Application Development through Reuse: the Ithaca Tools Environment", em "Object Composition", (ed.) Tsichritzis, D.;

Centre Universitaire d'Informatique, Universidade de Genebra, junho 1991.

[GAVI85] Gavillet, Ph. "Trends in New Collider Experiment Data Acquisition Systems", Nuclear Instruments and Methods in Physics Research, A235, North-Holland, 1985.

[GIBB90a] Gibbs, S.; e outros "Class Management for Software Communities", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[GIBB90b] Gibbs, S.; Tsichritzis,D. "Software Licensing versus Software Reuse", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[GIBB90c] Gibbs, S.; Prevelakis, V. "Xos: An Overview", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[GIBB90d] Gibbs, S. "Querying Large Class Collections", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[GOGU86] Goguen, J.A. "Reusing and Interconnecting Software Components", Computer, fevereiro 1986.

[GOLD84] Goldberg, A. "Smalltalk-80: The Interactive Programming Environment", Addison-Wesley, Massachusetts, 1984.

[GREE89] Green, M.G. "The ADAMO Data System", ALEPH note 89-47, 1989.

[HALL87] Hall, P. "Software components and reuse - getting more out of your code", Information and Software Technology, vol.29, no.1, 1987.

[HARR89] Harrison, W.H.; e outros "Good News, Bad News: Experience Building a Software Development Environment Using the OO Paradigm", OOPSLA'89 Proceedings, outubro 1989.

[HELM90] Helm, R.; e outros "Contracts: Specifying Behavioral Compositions in Object-Oriented Systems", ACM SIGPLAN Notices, OOPSLA/ECOOP'90, vol. 25, no. 10, outubro 1990.

[HERT89] Hertzberger, L.O. "Does HEP still hold challenges for computer science?", Computing in High Energy Physics Conference, Oxford, Inglaterra, 1989.

[HORO89] Horowitz, E.; Munson, J.B. "An Expansive View of Reusable Software", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[ILES86] Iles, R.M.J.; Hague, S.J. "Toolpack: The First Public Release", Computer Physics Communications 41, North-Holland, Amsterdam, 1986.

[ISCO88] Iscoe, N. "Domain-Specific Reuse: An Object-Oriented and Knowledge-Based Approach", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[JONE88] Jones, G. "Methodology/Environment Support for Reusability", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[KAEH87] Kaehler, T. "Virtual Memory for Object-Oriented Language", em "Tutorial: Object Oriented Computing - vol.2" (ed.) Peterson, G.; IEEE Computer Society Press, 1987.

[KATA91] Katayama, N. "Object-Oriented Approach to B reconstruction", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[KATZ89] Katz, S.; e outros "PARIS: A System for Reusing Partially Interpreted Schemas", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[KLEI83] Klein, H.J.; Zohl, J. "PATCHY: Reference Manual", CERN-23, 1983.

[KNOB91] Knobloch, J. "Reality of Software Engineering in High Energy Physics", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[KUKI71] Kuki, H. "Mathematical Function Subprograms for Basic System Libraries-Objectives, Constraints, and Trade-off" em "Mathematical Software" (ed.) Rice, J.R. , Academic Press, New York, 1971.

[KUNZ89] Kunz, P.F. "Software Management Issues", Computing in High Energy Physics Conference, Oxford, Inglaterra, 1989.

[KUNZ91] Kunz, P.F. "Physics Analysis Tools", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[LEWI68] Lewis, P.M.; Stearns, R.E.; Journal of the ACM 15-3, 265, 1968.

[LOKE89] Loken, S.C. "Computing for High Energy Physics in The 1990s", em "Proceedings of the Summer Study on HEP in the 1990s (eds.) Snowmass, C.; Jensen, S.; World Scientific, 1989.

[LOKE91] Loken, S.C. "Conference Summary", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[LUBA88] Lubars, M.D. "Code Reusability in the Large versus Code Reusability in the Small", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[LUKE86] Luker, P.A.; Burns, A. "Program Generators and Generation Software", The Computer Journal, vol.29, no.4, 1986.

[MAAR89] Maarek, Y.S.; e outros "Automatically Generating Software Libraries without Pre-Encoded Knowledge", Research Report RC 14990(#66631), IBM Research Division, Almaden, 1989.

[MAID88] Maidantchik, C.; Controle da Qualidade de Software Científico, Relatório Técnico do Programa de Eng. de Sistemas e Computação, ES-167/88, COPPE/UFRJ, Rio de Janeiro, 1988.

[MAID91] Maidantchik, C.; e outros "SIM: A Software Information Manager", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[MAID92] Maidantchik, C.; SIM: Um Gerador Semi-Automático de Documentos, Tese de Mestrado da COPPE/UFRJ, março 1992.

[MATT90] Mattoso, A.L.Q.; TABA_OBJ: Um Ambiente de Desenvolvimento de Software com Orientação a Objetos, Tese de Mestrado da COPPE.UFRJ, agosto 1990.

[MART85] Martin, J. "Fourth-Generation Languages", vol. I, Prentice-Hall, 1985.

[MEDE87] Medes, J.S.; e outros "A Survey of the Methods Used for the Development of Engineering Software in the United Kingdom", 1st International Conference on Reliability & Robustness of Engineering Software, setembro 1987.

[MEND91] Mendes, E.; Souza, J.M.; HIPPEA: HIpermeios aPlicados ao Processo de Ensino-Aprendizagem, Workshop Latinoamericano de Sistema Expertos y Multimedios, 1991.

[METC85] Metcalf, M. "Has FORTRAN a Future?", Computer Physics Communications, 38, North-Holland, Amsterdam, 1985.

[METC89] Metcalf, M. "Recent progress in Fortran standardization", Computing in High Energy Physics Conference, Oxford, Inglaterra, 1989.

[METC91] Metcalf, M. "Why FORTRAN 90?", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[MEY 91] Mey, V.; e outros "The Implementation of Vista - A Visual Scripting Tool", em "Object Composition" (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, junho 1991.

[MEYE84] Meyer, B. "Software Engineering for Engineering Software" em "Tools, Methods and Languages for Scientific and Engineering Computation", (eds.) Ford, B. e outros; Elsevier Science Publishers, 1984.

[MEYE89] Meyer, B. "Reusability: the case for object-oriented design", em "Software Reusability", vol. II, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[MICR90] Microsoft Corporation, Microsoft Windows: Guia do Usuário, USA, 1990.

[MOIN89] Moineau, T.; e outros "Towards a Generic and Extensible Reuse Environment", Proc. 2nd International Workshop on Software Engineering & its Applications, Toulouse, França, dezembro 1989.

[MONT92a] Monte, L.C.M.; A Comunicação Homem-Computador por Manipulação Direta; Relatório Técnico da Unversidade Federal Fluminense, 1992.

[MONT92b] Monte, L.C.M. e outros; Contribuição do PROTOGEO ao Projeto do SGBDOO GEOTABA, aceito para publicação no VII Simpósio Brasileiro de Banco de Dados, maio 1992.

[NEIG84] Neighbors, J.M. "The Draco Approach to Constructing Software from Reusable Components",IEEE Transactions of Software Engineering, vol.SE-10, no.5, setembro 1984.

[NEIG89] Neighbors, J.M. "Draco: a method for engineering reusable software systems", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[NEIG91] Neighbors, J.M. "The Evolution from Software Components to Domain Analysis", V Simpósio Brasileiro de Engenharia de Software, Ouro Preto, outubro 1991.

[NEWM87] Newman, H.B. "Computing for HEP Experiments: 1987-1997", Computer Physics Communications, 45, North-Holland, Amsterdam, 1987.

[NIER90] Nierstrasz, O.; e outros "Visual Scripting Towards Interactive Construction of Object-Oriented Applications", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[NIER91] Nierstrasz, O.; e outros "Objects + Script = Applications", em "Object Composition", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, junho 1991.

[NUCC89] Nucci, A. "FADOBUG: The FADO debugger", Projeto Final de Curso, Univ. Udine, Dept. Informática, Itália, 1989.

[OLEY91] Oleynik, G.A. "Object-Oriented Design and Programming for Experiment Online Applications-Experiences with a Prototype Application Online Support Department", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[OPCO85] Opcode, Inc. "Historian-Plus Users-Manual, 1985.

[OSTE84] Osterweil, L.; Clemm, G. "An Extensible Toolset and Environment for the Production of Mathematical Software", em "Tools, Methods and Languages for Scientific and Engineering Computation", (eds.) Ford, B. e outros; Elsevier Science Publishers, 1984.

[PALE89] Palermo, L.I.; Eijndhoven, N.V. "DELOFB DELPHI Off-line Bookeeping User's Guide", DELPHI note 89-78 PROG 145, Genebra, Suiça, 1989.

[PALE90a] Palermo, L.I.; Souza, J.M. "Recording DELPHI Off-line Production Activity in a Meta-Database", International Conference on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, Lyon, França, 1990.

[PALE90b] Palermo, S.; Rocha, A.R.C. "Evaluating Quality of Programs in DELPHI off-line Software", DELPHI note 90-18 PROG 151, Genebra, Suiça, 1990.

[PALE90c] Palermo, S.; Rocha, A.R.C. "An Experience on Software Quality Assurance", International Conference on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, Lyon, França, 1990.

[PERE87] Pereira, J.C.; Metodologia para Desenvolvimento de Software para Engenharia, VIII Congresso Latino-Americano e Ibérico sobre métodos computacionais para Engenharia, Rio de Janeiro, 1987.

[PERE88] Pereira, R.C.S.; Programação Multi-Linguagem: Uma Proposta; Tese de Doutorado da COPPE/UFRJ, setembro 1988.

[PERN90] Pernici, B. "Class Design and Meta-Design", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[PERR83] Perry, W.E. "Effective Methods of EDP Quality Assurance", Prentice-Hall, New Jersey, 1983.

[PETE91] Peterson, A.S. "Coming to Terms with Software Reuse Terminology: a Model-Based Approach", Software Engineering Notes, vol.16, no.2, abril 1991.

[PIME90] Pimenta, M.; Varela, J. "FADO 2.0 Assembler Instructions", notas de trabalho, 1990.

[PIME91] Pimenta, M.; Comunicação escrita particular, 1991.

[PINT88] Pintado, X.; Tsichritzis, D. "An Affinity Browser", em "Active Object Environments", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, junho 1988.

[PINT90] Pintado, X. "Selection and Exploration in an Object-Oriented Environment: The Affinity Browser", em "Object Management", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[PINT91] Pintado, X. "Gluons: Connecting Software Components", em "Object Composition", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, junho 1991.

[PRIE85] Prieto-Diaz, R. "A Software Classification Scheme", Ph.D.thesis, Universidade da California, 1985.

[PRIE86] Prieto-Diaz, R.; Neighbors, J.M. "Module Interconnection Languages", em "Software Reusability", (ed.) Freeman, P.; Computer Society Press of the IEEE, 1986.

[PRIE87] Prieto-Diaz, R.; Freeman, P. "Classifying Software for Reusability", IEEE Software, vol.4, no.1, janeiro 1987.

[PRIE88] Prieto-Diaz, R.; Jones, G.A. "Breathing new life into old software", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[PRIE89] Prieto-Diaz, R. "Classification of Reusable Modules", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[PRIE90] Prieto-Diaz, R. "Domain Analysis: An introduction", ACM SIGSOFT SOFTWARE ENG. Notes, vol.15, no.2, 1990.

[PRIE91] Prieto-Diaz, R. "Implementing Faceted Classification for Software Reuse", Communications of the ACM, vol.34, no.5, maio 1991.

[PROF89] Profrock, A.; e outros "ITHACA: An Integrated Toolkit for Highly Advanced Computer Applications", em "Object Oriented Development" (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1989.

[RAJ 89] Raj, R.K.; Levy, H.M. "A Compositional Model for Software Reuse", Third European Conference on Object-Oriented Programming, Cambridge University Press, Nottingham, julho 1989.

[READ89] Read, B.J. "Data Structures and Organisation: Special Problems in Scientific Applications", Computing in High Energy Physics Conference, Oxford, Inglaterra, 1989.

[RICE71a] Rice, J.R. (ed.) "Mathematical Software", Academic Press, New York, 1971.

[RICE71b] Rice, J.R. "The Challenge for Mathematical Software" em "Mathematical Software" (ed.) Rice, J.R. , Academic Press, New York, 1971.

[RICE71c] Rice, J.R. "The Distribution and Sources of Mathematical Software" em "Mathematical Software" (ed.) Rice, J.R. , Academic Press, New York, 1971.

[RICE83] Rice, J.R. "Numerical Methods, Software, and Analysis: IMSL Reference Edition", McGraw-Hill, New York, 1983.

[RICE89] Rice, J.R.; Schwetman, H. "Interface Issues in a Software Parts Technology", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[ROCH83] Rocha, A.R.C.; Um Modelo para Avaliação da Qualidade de Especificações; Tese de Doutorado, PUC-RJ, junho 1983.

[ROCH87] Rocha, A.R.C.; e outros; Algoritmos Numéricos x Software Numérico: Uma visão do Engenheiro de Software, VIII Congresso Latino-Americano e Ibérico sobre métodos computacionais para Engenharia, Rio de Janeiro, 1987.

[ROCH88] Rocha, A.R.C.; e outros; Metodologia para desenvolvimento de software científico, Relatórios do Projeto Coppetec ET-21045, COPPE/UFRJ, 1988.

[ROCH89] Rocha, A.R.C.; Palermo, S. "Software Quality Assurance in HEP", Computer Physics Communications, vol.57, no.1-3, North-Holland, Amsterdam, 1989.

[ROCHA91] Rocha, A.R.C.; Palermo, S. "An Environment for Software Quality Evaluation in HEP", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[RODR80] Rodrigue, G.; e outros "Perspectives on Large-Scale Scientific Computation", Computer, outubro 1980.

[ROUS90] Rousseau, B.; Rechenmann, F. "Building the Scientist Computing Environment", International Conference on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, Lyon, França, 1990.

[RUSS87] Russell, J.J. "Programming Languages: Time for a Change?", Computer Physics Communications 45, North-Holland, Amsterdam, 1987.

[SAMM71] Sammet, J.E. "Software for Nonnumerical Mathematics" em "Mathematical Software" (ed.) Rice, J.R. , Academic Press, New York, 1971.

[SCHW86] Schwartz, J.; e outros "Programming with Sets: An Introduction to SETL", Springer-Verlag, New York, 1986.

[SEID89] Seidewitz, E. "General Object-Oriented Software Development: Background and Experience", The Journal of Systems and Software, no.9, 1989.

[SELB88] Selby, R.W. "Empirically Analyzing Software Reuse", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[SELB89] Selby, R.W. "Quantitative Studies of Software Reuse", em "Software Reusability", vol. II, (ed.) Biggerstaff,T.J.; Perlis, A.J.; Addison Wesley, 1989.

[SEPP87] Seppanen, V. "Reusability in Software Engineering", em "Tutorial: Software Reusability" (ed.) Freeman, P.; 1987.

[SETZ83] Setzer, V.M.; Melo, H.; A Construção de um Compilador, Ed. Campus, Rio de Janeiro, 1983.

[SHLA90] Shlaer, S.; Mellor, S.J.; Análise de Sistemas Orientada para Objetos; (trad.) Giova, A.T.; McGraw Hill/Newstec Editora Ltda, 1990.

[SHNE87] Schneiderman, B. "Designing the User Interface: Strategies for Effective Human-Computer Interaction", Addison-Wesley, 1987.

[SMIT84] Smith, B.T. "Fortran 8x: A Continuing Language for Numerical Software", em "Tools, Methods and Languages for Scientific and Engineering Computation", (eds.) Ford, B. e outros; Elsevier Science Publishers, 1984.

[SOUZ90] Souza, J.M.; Rocha, A.R.C. "TABA-HEP: a Heuristic Workstation for Developing HEP Software", International Conference on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, Lyon, França, 1990.

[SOUZ91] Souza, J.M.; Rocha, A.R.C. "The Architecture of Taba-HEP Workstation, Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[STAD90] Stadelmann, M.; e outros "VST: A Scripting Tool Based on the UNIX Shell", em "Object Management" (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1990.

[STAD91] Stadelmann, M. "TeamWorks: Towards A Framework for Reuse", em "Object Composition" (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, junho 1991.

[STRE74] Streeter, D.N. "The Scientific Process and the Computer", John Wiley & Sons, New York, 1974.

[TEIT84] Teitelbaum, T.; Reps, T. "Interactive Programming Environments, McGraw Hill, New York, 1984.

[TRAC88a] Tracz, W. "RMISE workshop on Software Reuse: Meeting Summary", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[TRAC88b] Tracz, W. "Software Reuse Myths", ACM SIGSOFT SOFTWARE ENGINEERING NOTES, vol.13, no.1, janeiro 1988.

[TRAC88c] Tracz, W. "Software Reuse: Motivators and Inhibitors", em "Software Reuse: Emerging Technology", (ed.) Tracz, W.; 1988.

[TRAC90] Tracz, W. "Where Does Reuse Start ?", ACM SIGSOFT SOFTWARE ENGINEERING NOTES, vol.15, no.2, abril 1990.

[TRAU84] Trauboth, H. "Development of Scientific and Engineering Software for Nuclear Applications", em "Tools, Methods and Languages for Scientific

and Engineering Computation", (eds.) Ford, B. e outros; Elsevier Science Publishers, 1984.

[TREM85] Tremblay, J.P.; Soreson, P.G. "The Theory and Practice of Compiler Writing", McGraw Hill, New York, 1985.

[TSIC88] Tsichritzis, D.C.; Nierstrasz, O.M. "Application Development Using Objects", em "Information Technology for Organizational Systems", Proceedings EURINFO'88, Elsevier Science Publishers, 1988.

[TSIC89] Tsichritzis, D. "Object-Oriented Development for Open Systems", em "Object Oriented Development", (ed.) Tsichritzis, D.; Centre Universitaire d'Informatique, Universidade de Genebra, julho 1989.

[VITE90] Vitek, J.; e outros "Events and Sensors: Enhancing the reusability of objects", em "Object Management", (ed.) Tsichritzis, D.C.; Centre Universitaire d'Informatique, Universidade de Genebra, Suiça, julho 1990.

[WEBS88] Webster, D.E. "Mapping the Design Information Representation Terrain", Computer, dezembro 1988.

[WEGN89] Wegner, P. "Capital-Intensive Software Technology", em "Software Reusability", vol. I, (ed.) Biggerstaff, T.J.; Perlis, A.J.; Addison Wesley, 1989.

[WERN88a] Werner, C.M.L.; O Desenvolvimento de Software Científico no SEDEM (pesquisa de opinião), Relatório Interno do Setor de Desenvolvimento e

Métodos do Centro de Pesquisa da PETROBRÁS (CENPES), Rio de Janeiro, 1988.

[WERN88b] Werner, C.M.L.; Proposta para Elaboração do Manual de Desenvolvimento de Software no SEDEM, Relatório Interno do Setor de Desenvolvimento e Métodos do Centro de Pesquisa da PETROBRÁS (CENPES), Rio de Janeiro, 1988.

[WERN88c] Werner, C.M.L.; Rocha, A.R.C. "Flexibility and its relation to other Software Project Quality Factors", Seventh International Conference of the Israel Society for Quality Assurance, Israel, 1988.

[WERN89a] Werner, C.M.L.; e outros "FADO 2.0: A High Level Tagging Language", Computer Physics Communications, vol.57, no.1-3, North-Holland, Amsterdam, 1989.

[WERN89b] Werner, C.M.L.; e outros "A Programming Environment for a Tagging Language", VI Conferencia Cientifica de Ingenieria y Arquitetura, Ciudad de La Habana, Cuba, 1989.

[WERN90a] Werner, C.M.L.; e outros "Fado Tagging System: decoupling logic from code", International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics, Lyon, França, 1990.

[WERN90b] Werner, C.M.L.; e outros "Fado Tagging System: a new Tagging Approach", Computing in High Energy Physics conference, Santa Fe, Estados Unidos, 1990.

[WERN90c] Werner, C.M.L.; e outros "Fado Tagging System: User's Guide", DELPHI note 90-47 DAS 105, Genebra, Suiça, 1990.

[WERN90d] Werner, C.M.L. "Fado 2.0 Editor", LIP-90/04 Technical Report, Lisboa, Portugal, 1990.

[WERN90e] Werner, C.M.L. "Fado 2.0 Compiler", LIP-90/03 Technical Report, Lisboa, Portugal, 1990.

[WERN91a] Werner, C.M.L.; Souza, J.M. "An Object-Oriented Composition Environment for Scientific Applications", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[WERN91b] Werner, C.M.L.; Reutilização em Software, Relatório Técnico do Programa de Eng. de Sistemas e Computação, ES-233/91, COPPE/UFRJ, Rio de Janeiro, 1991.

[WERN91c] Werner, C.M.L.; Um estudo sobre Ambientes de Programação, C.M.L. Werner, Relatório Técnico do Projeto TABA, COPPE/UFRJ, Rio de Janeiro, 1991.

[WERN91d] Werner, C.M.L.; e outros; Protótipo do CAOS: Um Ambiente para Composição de Aplicações Científicas Orientado a Objetos, Workshop em Desenvolvimento, Avaliação e Manutenção de Software Científico, COPPE/UFRJ, Rio de Janeiro, 1991.

[WERN91e] Werner, C.M.L.; e outros "Fado Tagging System: Selecting HEP Events Online", Relatório Técnico do Programa de Eng. de Sistemas e Computação, ES-245/91, COPPE/UFRJ, Rio de Janeiro, 1991.

[WERN91f] Werner, C.M.L.; O Desenvolvimento de Software Científico: Um caso de reutilização de software, Workshop em Pesquisas de Tese em Engenharia de Software, COPPE/UFRJ, Rio de Janeiro, 1991.

[WERN92] Werner, C.M.L.; Mattos, F.A.; "Protótipo do Ambiente CAOS: Relatório de Implementação", Relatório Técnico do Programa de Eng. de Sistemas e Computação, em preparação, COPPE/UFRJ, Rio de Janeiro, 1992.

[WHIT90] White, B. "The Comparison and Selection of Programming Languages for High Energy Physics Applications", International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics, Lyon, França, 1990.

[WHIT91] The Whitewater Group "Actor User's Manual, Evanston, USA, 1991.

[WOOD87] Woodfield, S.N.; e outros "Can Programmers Reuse Software?", IEEE Software, julho 1987.

[XEXE91] Xexeo, G.; e outros "CAB: The COSMOS Application Builder", Computing in High Energy Physics Conference, Tsukuba City, Japão, 1991.

[YOUS90] Youssef, S. "The TYPES Users Guide: A Data Abstraction Package in FORTRAN", Florida State University, FSU-SCRI-90-03, Tallahassee, USA, 1990.

[ZANE90] Zanella, P. "30 Years of Computing at CERN", CERN School of Computing, Bad Herrenalb, Germany, 1989.

Apêndice A - Exemplo de um Programa em Fado 2.0

Eventos com léptons isolados com alto momento são importantes para o estudo de decaimentos de Z0 em quarks pesados. O exemplo apresentado mostra como selecionar eventos com ao menos um elétron isolado com alto momento. O critério de isolamento é definido como o ângulo mínimo entre o elétron identificado e o traço carregado mais próximo.

O programa principal tem apenas duas condições: um corte na multiplicidade de traços carregados para seleção de eventos hadrônicos e um corte no número de elétrons isolados.

```
! ******** PROGRAMA PRINCIPAL FADO ********* !
! Programa para seleção de eventos com electrons isolados !
! = = = = = = = = = = = = = = = = = = = = = = = = = = = = = !

                    ! Definição da Reação !

Define_Reaction Electron_Unico
  Multiplicity($TRACKS) > = 5
  Multiplicity($ISOL_ELECTRON) > = 1
End_Reaction

                    ! Definição de Listas !

Define_Lists
  $ISOL_ELECTRON = = { _x : _x AND(in $ELECTRON,
                                   not_in { (_x1,_y1) :
                                            _x1 = _x in $ELECTRON
                                            _y1   in $TRACKS
                                            & AND(p(_y1) > 0.1,
                                                  Angle(_x1,_y1) < 10)
                                   & p(_x) > 2 }
  $EM_CLUSTERS = = { _x : _x OR(in $HPC, in $EMF) }
End_Define

                    ! Definição de Funções !

Define_Functions
  Function Angle(_x1,_x2)
   Int_Prod = Px(_x1)*Px(_x2) + Py(_x1)*Py(_x2) + Pz(_x1)*Pz(_x2)
   Cos_Angle = Int_Prod / (p(_x1)*p(_x2))
   Angle = ACOS(Cos_Angle)
  End_Function
End_Define
End_Prog
```

```
! ***** BIBLIOTECA DE LISTAS COMPOSTAS DO SISTEMA ****** !

Define_Composite
  $ELECTRON = = {_x=OR((_x1, _x2) : _x1 in $TRACKS
                                    & P(_x1) > Pcut_EL
                                        _x2 in $HPC
                                    & AND(_x1 Associate _x2,
                                          Energy(_x2) > = 0.6*p(_x1),
                                          Energy(_X2) < 1.5*p(_x1)),
                              (_x1, _x2) : _x1 in $TRACKS
                                    & p(_x1) > Pcut_EL
                                        _x2 in $EMF
                                    & AND(_x1 Associate _x2,
                                          Energy(_x2) > = 0.6*p(_x1),
                                          Energy(_X2) < 1.5*p(_x1)) )}
$TRACK_IN_B = = {_x=(_x1,_x2,_x3) : _x1 in $TPC
                                    & p(_x1) > Pcut_TPC
                                        _x2 in $OD
                                    & _x1 Associate _x2
                                        _x3 in $IDJ
                                    & _x1 Associate _x3}
End_Define
                              ! Definição de Listas !
Define_Lists
  $TPC_TRACKS = = {_x : _x AND(in $TPC,
                                not_in CONSTITUENTS of $TRACK_IN_B)
                          & p(_x) > Pcut_TPC}
  $TRACKS = = { _x : _x OR(in $TRACK_IN_B,
                                in $TPC_TRACKS)}
End_Define
                              ! Definição de constantes !
Define_Constants
  Pcut_EL  = 0.5
  Pcut_TPC = 0.1
End_Define
End_Prog
```

Apêndice B - Exemplo de uma Aplicação CAOL

```
Define_Object Histogram
  Operations_are
     Create;
     Display;
     Print;

Structure_is
     x : array of integer;
     y : array of integer;
     z : array of integer;

Implementation_is
     Create = CHIST(x,y,z);
     Display = DHIST(x,y,z);
     Print = PHIST(x,y,z);

Classification_is
     Language = FORTRAN;
     Domain = Graphics;
     Function = Print, Create, Display;
     Object = Histogram;
End_Object


Define_Application histogram
{
   hist : Histogram
   Create(hist);
   Display(hist);
   Print(hist);
}
```