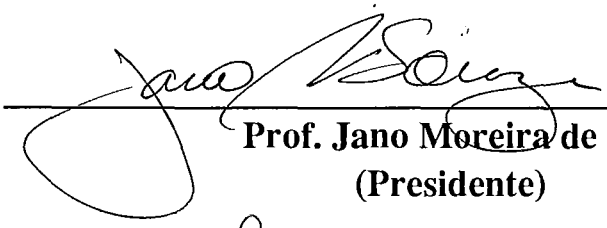
Um Sistema de Reutilização de Âmbito Global

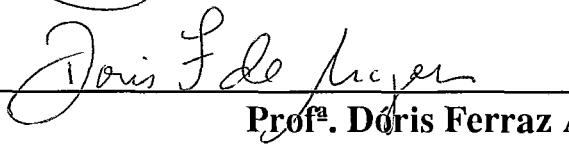
Geraldo Bonorino Xexéo

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.

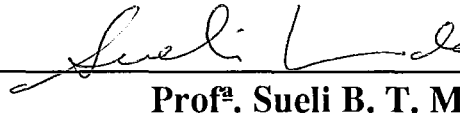
Aprovada por:



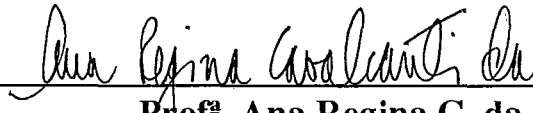
Prof. Jano Moreira de Souza, Ph.D.
(Presidente)



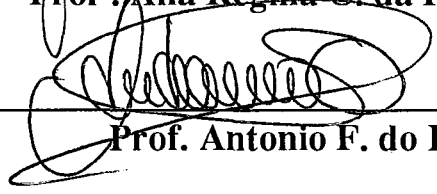
Profª. Dóris Ferraz Aragon, L.D.



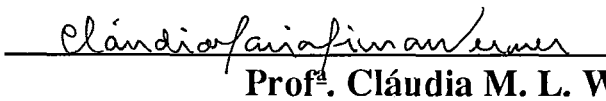
Profª. Sueli B. T. Mendes, Ph.D.



Profª. Ana Regina C. da Rocha, D.Sc.



Prof. Antonio F. do Prado, D.Sc.



Profª. Cláudia M. L. Werner, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 1994

Xexéo, Geraldo Bonorino

Um Sistema de Reutilização de Âmbito Global [Rio de Janeiro] 1992.

XI, 135 p. 29.7 cm (COPPE/UFRJ), D.Sc., Engenharia de Sistemas e Computação.

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Reutilização de Software 2. Métodos Formais 3. Hipertexto 4. VDM

I. COPPE/UFRJ II. Título (série)

Agradecimentos

Ao meu orientador, Jano, sempre disposto a ouvir e produzir idéias novas.

À professora Ana Regina, sem a qual este trabalho nunca teria se iniciado, continuado e terminado.

À Carmen, minha companheira de problemas e aventuras européias.

Ao Prof. Zieli, que me permitiu trabalhar no maior laboratório de física do mundo e nunca deixou de me apoiar.

A CAPES, CNPq e ao World Laboratory, liderado pelo Prof. Antonino Zichichi.

À professora Dóris, minha primeira orientadora na pesquisa científica.

Ao Guilherme, que mantém tudo funcionando.

Ao Eduardo, que implementou o hmake, ao Abílio, que implementou o vdmpp, e a Ana e Luíz, que trabalharam no servidor de hipertexto.

Ao Seixas, por ser ele mesmo, e aos outros brasileiros que estiveram comigo na Suiça: Koki, Alexandre, Gaspar, Denison, Trotta, Cláudia e o meio alemão Frank.

Aos colegas Karin, Marimar, Washington, Bevi, Vera, Cristina, Sérgio, Adriana, Emília(s) e John, que fizeram cursos ou dividiram laboratórios comigo, me ajudando em tarefas diversas e me dando inesquecíveis caronas, tornando o trabalho sempre mais fácil. E a outro colega, Cláudio D'Ippolito, por Douglas, Escher, and Bach.

A Cláudia, Ana Paula e Rose, que mantém a secretaria funcionando, e à Mercedes, pela paciência que todas têm comigo.

Às minhas irmãs e meus pais, sempre presentes.

À minha prima Chris, que nunca me esqueceu.

Ao Nilo e Adriano, que escutaram meus problemas.

À bras-net, em especial à Dedéa, ao Mauro, ao Marcelo Finger, ao Jaime, ao Christian e tantos outros que a fazem funcionar e viver.

À Dr. Nico Plat, Dr. Peter Gorm Larsen, Joerg, Benny, Dr. Steigerwald, que forneceram ajuda em várias ocasiões.

Aos meus amigos europeus, em especial a Giuseppe, Massimo, Franco, Bob e minha antiga chefe, Luisa Cifarelli.

A todos que desenvolvem software gratuito, em especial a Marc Anderssen (Mosaic), Richard Stallman (GNU Project) e Tim Berners-Lee (WWW).

O meu agradecimento pela ajuda e apoio ao decorrer desta tese.

Não poderia deixar de dedicar esse trabalho à minha família, que com certeza sempre continuará comigo

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

Um Sistema de Reutilização de Âmbito Global

Geraldo Bonorino Xexéo

Março, 1994

Esta tese apresenta um modelo de reutilização baseado em duas idéias principais: reutilizar componentes de software à partir de especificações formais e permitir o acesso da maior comunidade possível aos componentes reutilizáveis. É apresentada uma revisão da literatura sobre VDM e reutilização, fornecendo os meios para a definição do modelo de reutilização.

VDM foi escolhido como método de especificação por suas características de permitir níveis altos de abstração e simultaneamente poder ser utilizado do início ao fim do ciclo de vida.

Após a descrição do estado da arte em reutilização, dois sistemas são apresentados: CAB, um ambiente de desenvolvimento de software com suporte a reutilização, e Tabetá, um sistema de reutilização aplicável a qualquer ciclo de vida. Em especial, o sistema Tabetá implementa o modelo de reutilização que é definido nesta tese.

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

A Global Wide Reuse System

Geraldo Bonorino Xexéo

March, 1994

This thesis introduces a reuse model based on two main ideas: to reuse software components from formal specifications and to allow the widest possible community to access reusable components. We provide a framework for the definition of the reuse model with a review of the literature on VDM and software reuse.

We chose VDM as the specification method because it allows high abstraction levels and, at the same time, can be used from the beginning to the end of the life cycle.

After a description of the state of art in reuse, two systems are presented: CAB, a software development environment with reuse support, and Tabetá, a reuse system that can be applied to any life cycle model. In particular, the Tabetá system implements the reuse model defined in this thesis.

SUMÁRIO

I. INTRODUÇÃO

"—Podia me indicar, por favor, qual é o caminho para sair daqui?
—Isso depende muito do lugar para onde você quer ir."
Diálogo entre Alice e o Gato Cheshire, Lewis Carrol, in *Alice no País das Maravilhas.*

Este capítulo parte da idéia de desenvolvimento de software para chegar ao motivo da tese: a necessidade de um ambiente global de reutilização baseado em uma metodologia formal. Ao final, contém a organização da tese.

I.1. Reutilização de Especificações

Desenvolver software é uma tarefa essencialmente intelectual. Pessoas, baseadas em um modelo abstrato de um sistema real, desenvolvem software destinado a cumprir uma ou mais funções desse sistema.

Quando envolvidas em tarefas intelectuais, as pessoas possuem um conjunto de idéias e *pré*-conceitos a que podemos chamar **experiência**. Normalmente os problemas são resolvidos em função dessa experiência. Seja por meio de repetição, seja por meio de comparação ou analogia.

Assim, ao desenvolver software as pessoas **reutilizam** suas experiências. Em especial, reutilizam suas experiências anteriores de desenvolvimento de software.

É comumente aceito (e defendido) que a reutilização de software tem que sair do nível pessoal para passar ao nível corporativo e mesmo global. Para isso devem ser fornecidas, ao desenvolvedor de software, formas de buscar um conhecimento que não pertence a sua experiência. Pode-se dividir essas formas em **processos de reutilização**, que fornecem um paradigma de reutilização, e **ferramentas de reutilização**, que fornecem mecanismos de reutilização. **Processos** e **ferramentas** podem ser dependentes ou independentes entre si.

Também já é aceito pela maioria dos pesquisadores que quanto mais cedo, no processo de desenvolvimento de software, se inicia o processo de reuso, maior será a economia fornecida por esse processo. Por isso, esta tese defende que o processo de reutilização deve ser iniciado com a **reutilização de especificações**.

I.2. Vantagens de Reutilizar Especificações

Vários autores demonstram a vantagem de reutilizar especificações. Em especial, Yourdon [You92] cita o fato de que, uma vez reutilizada a especificação em um grau de abstração alto, praticamente todas as especificações derivadas desta e com grau de abstração menor também podem ser reutilizadas. A figura abaixo fornece uma interpretação visual para essa afirmativa: a reutilização de código equivale a reutilizar as partes mais baixas dessas hierarquias, as folhas de nossas árvores, enquanto a reutilização da especificação permite reutilizar grande parte do ramo representado!

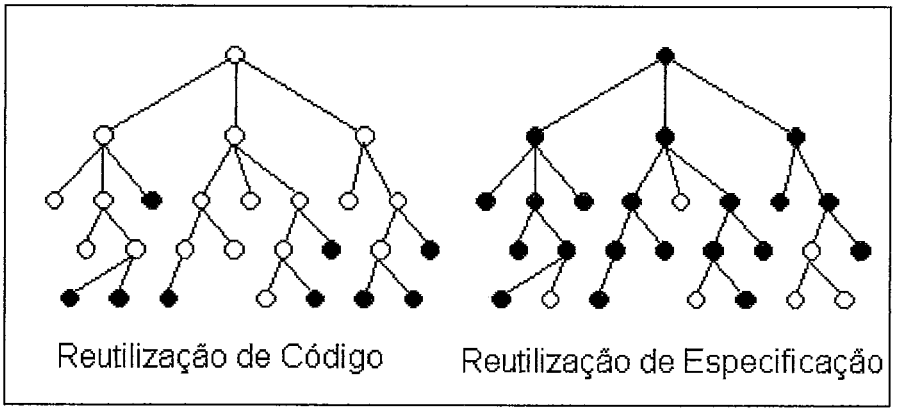


Fig 1 Reutilização de código versus reutilização de especificações[You92].

Por outro lado percebe-se que vários sistemas possuem partes semelhantes. Por exemplo, um sistema que permite a entrada de um certo tipo de dado em uma tabela, deve, em geral, permitir a exclusão do mesmo dado desta tabela. Logo, é possível **completar** especificações parciais utilizando técnicas de reutilização.

I.3. Reutilização de Especificações Formais

Muitos processos de reutilização alcançaram resultados excelentes baseando-se em **domínios de aplicação** [Nei84,Gir92]. Esses processos, porém, baseiam-se no uso de palavras e seus significados para o domínio em questão. No caso de haver

mudança de domínio, cabe ao desenvolvedor construir as analogias inter-domínios que possam auxiliá-lo no processo de reutilização.

É proposta desta tese alcançar a reutilização baseada na **semântica** de uma especificação. Assim, a semântica de especificações parciais poderá ser comparadas com a semântica de especificações anteriores, ficando a reutilização livre dos símbolos ou das estruturas existentes em sua descrição. Para implementar a reutilização semântica deve ser utilizado um método de desenvolvimento que possua um significado bem definido. É o caso dos **métodos formais.**

Assim sendo, esta tese defende a **reutilização de especificações formais.**

I.4. Ambiente Global de Desenvolvimento

Gibbs, Prevalakis e Tsichritzis [GPT89] defendem a criação de **comunidades de software**. As milhares de pessoas que hoje tem acesso à rede de comunicação Internet conhecem essa comunidade: a própria Internet. Pessoas e instituições fornecem, com ou sem fins lucrativos, softwares completos ou pedaços de software. Porém, são poucos os mecanismos que permitem a busca de um software em função de sua especificação. Na melhor hipótese, palavras chaves podem ser utilizadas para buscar o software desejado, por meio dos serviços WAIS e archie, ou ligações de hipertexto podem ser seguidos por meio do serviço WWW[Ber+92].

O que esta tese propõe é que qualquer processo de reutilização deve ser implementado no maior ambiente acessível ao reutilizador. Hoje esse ambiente é a rede global de computadores, em particular, a formada por várias subredes que utilizam ou são compatíveis com o protocolo TCP/IP, conhecida pelo termo genérico de Internet.

Para que isso seja possível, foi implementado um servidor de hipertexto que permite consultas, via Internet, distribuídas a outros servidores. Assim, depósitos de componentes em localidades diferentes podem ser consultados de forma dinâmica, não existindo a necessidade do usuário navegar por vários computadores da rede em busca do componente desejado.

Esse servidor utiliza uma extensão do protocolo HTTP e se comunica com qualquer cliente WWW, respondendo as consultas por meio da linguagem de marcação HTML

I.5. Estratégia de Reutilização

A estratégia de reutilização defendida por esta tese pode ser resumida em duas partes. A primeira garante que a busca do componente a ser reutilizado será realizada no ambiente mais amplo possível. A segunda, garante que em cada base de componentes, a busca será realizada da maneira mais completa possível. Essas duas estratégias aparecem na arquitetura do sistema caracterizadas pela divisão da tarefas do servidor entre dois componentes, um responsável pelo tratamento da comunicação entre os servidores e o outro responsável pela busca local.

A metodologia implementada fornece um mecanismo generalizado que incorpora e estende diferentes estratégias. Nosso método utiliza três aspectos de linguagens de programação, encontrados na linguagem VDM-SL, isto é, aspectos léxicos, sintáticos e semânticos, avaliando a importância de cada objeto selecionado por cada método particular de busca de acordo com a qualidade do método e com a seleção do mesmo objeto por outros métodos.

I.6. Ferramentas de Suporte

Para suportar esse método foram desenvolvidas e especificadas várias ferramentas no decorrer da tese:

- um ambiente de desenvolvimento em VDM (CAB),
- um editor gráfico de VDM incorporado ao ambiente CAB, utilizando uma notação criada para esta tese (VDM/GDL),
- um parser de um sub-set da linguagem VDM (vdmc),
- um servidor de nós de hipertexto baseado em objetos, capaz de fornecer várias visões de um mesmo nó e garantir ligações tanto relacionais entre objetos quanto internas, (objserver),
- uma interface de consulta incorporada ao servidor anterior, capaz de responder em HTML,

•um sistema de busca baseado nos três mecanismos de busca propostos:

- ◊ um mecanismo de busca semântica incorporado ao servidor objserver, baseado no conceito de sub-tipos e tipos conforme,
- ◊ um mecanismo de busca baseado em palavras e facetas,
- ◊ um mecanismo de busca baseado em sintaxe.

•uma ferramenta semelhante ao make do sistema operacional Unix, porém capaz de recuperar arquivos via HTTP ou FTP (hmake),

•um pretty-printer para VDM (vdmpp),

•um segundo compilador VDM, desta vez baseado em um parser criado por Fisher et al. [Fis+93], com nossa colaboração (vdm).

•Um servidor WWW capaz de distribuir demandas entre outros servidores, inclusive diferentes, e ao mesmo tempo repassá-las ao servidor de objetos (dhttpd).

I.7. Organização da Tese

O segundo e terceiro capítulos fazem uma revisão da literatura, sendo o segundo capítulo uma descrição do estado da arte em reutilização e o terceiro capítulo uma introdução a métodos formais e VDM.

O capítulo quatro apresenta o ambiente CAB, desenvolvido no CERN. O quinto capítulo apresenta um modelo de reutilização e um sistema que implementa esse modelo: Tabetá[1]. No último capítulo apresentamos as conclusões desta tese, suas contribuições para o estado da arte e propostas para continuação do trabalho realizado.

[1] Tabetá é a palavra Tupi-Guarani para um conjunto de tabas, podendo ser traduzida para comunidade ou tribo.

II. REUTILIZAÇÃO

"Na Natureza nada se perde, nada se cria, tudo se transforma."
Lavoisier

Neste capítulo é apresentado o estado da arte da Reutilização na Engenharia de Software. A maior preocupação é com sistemas de composição, *ou que utilizem especificações formais. Além disso, tendo em vista a proposta deste trabalho, de um ambiente global de reutilização, é feita uma revisão nos conceitos de Comunidade de Software.*

II.1. O Que é Reutilizar

Reutilizar software é desenvolver novo software utilizando partes previamente disponíveis. Os tipos de partes que podem ser utilizados não estão limitados a fragmentos de código, mas devem incluir quaisquer formas que possam ser caracterizadas como partes de um software, como manuais, especificações, projetos estruturados, etc. Podemos até parafrasear D. Berry [Ber92] dizendo que software é tudo que alguém chame, em algum momento, de software.

Peterson [Pet91] fornece um glossário de termos de reutilização do qual, neste ponto, podemos utilizar especificamente duas definições:

- **reutilização** é a aplicação de soluções existentes para os problemas de desenvolvimento de sistemas

- **reutilização de software** (1) é o processo de utilizar software pré-existente durante o desenvolvimento da implementação de novos sistemas ou componentes de software. (2) É o resultado do processo realizado em (1).

II.1.1. Desenvolvimento de software orientado pela reutilização

Segundo Basili e Rombach [BR91] "o desenvolvimento de software orientado pela reutilização assume que, dado um requisito específico de projeto x' para um objeto x, considera-se a reutilização de um objeto já existente x_k ao invés de criar x do início. Reutilizar envolve identificar um conjunto de candidatos à reutilização x_1,

$...,x_n$ de uma base de experiências, avaliando seu potencial de satisfazer x', selecionando o melhor candidato x_k e, se necessário, modificar o candidato selecionado x_k para x."

Essa visão matemática diz apenas que, em um dado momento do ciclo de vida do software, reutilizar é procurar algum componente que seja semelhante ao componente desejado em um banco de dados de componentes.

Bersoff e Davis [BD91] tentam exemplificar isso com o "Modelo de Reutilzação de Software para o Ciclo de Vida de Desenvolvimento de Software", representado pela figura 2.

Note-se, porém, a simplicidade desta figura, que não é nada mais que a introdução da atividade de reutilização no ciclo de vida tradicional, e em apenas algumas fases. Como será demonstrado no decorrer desta tese, não só a reutilização está presente, mas também pode ser formalizada, em todas as fases, sendo adequada a qualquer ciclo de vida.

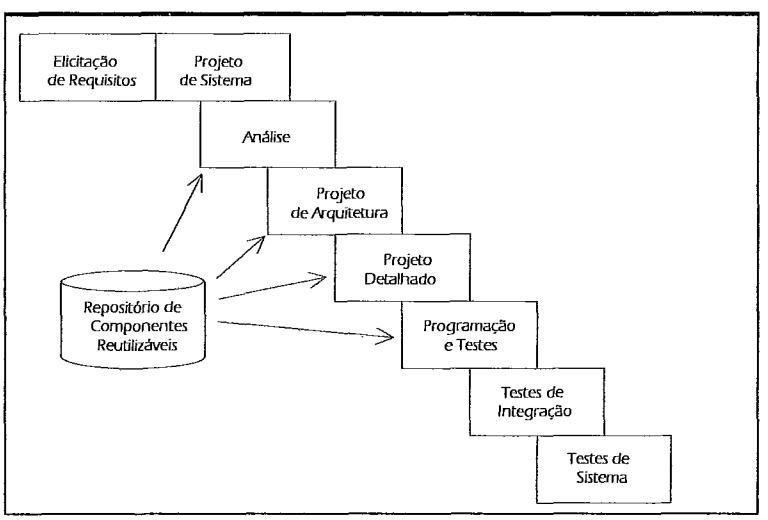


Fig 2 Modelo de Reutilização de Software do Ciclo de Vida do Desenvolvimento de Software [BD91].

II.1.2. Partes e peças não são iguais a software

A principal visão de reutilizar software é baseada na existência de partes e peças que podem ser compostas para montar um sistema, como é comum nas

engenharias eletrônica e mecânica. Essa visão, creditada a Dough McIlroy durante a conferência sobre reutilização da NATO em 1968, tornou-se ainda mais forte quando Brad Cox [Cox86] cunhou o termo *Software-IC* para representar componentes reutilizáveis de software. Porém, preparar software para ser reutilizado é uma idéia que acompanha a computação desde o início, como pode ser visto no texto a seguir, retirado de "Programming for the High-speed Digital Calculating Machines" [Bow53], um texto publicado em 1953!

> "Com o passar do tempo, uma coleção de subrotinas flexíveis, que são econômicas em requisitos de tempo e memória, fica disponível. Processos completos que, uma vez construídos, são úteis sempre que for utilizada com uma máquina específica, e sua generalidade deve ser tal que, pela modificação de poucos parâmetros, eles possam ser aplicados a uma gama de problemas similares, se permitido pelo tamanho da memória.
>
> Rotinas e subrotinas deste tipo representam o investimento de um número considerável de homens/hora. [...] Logo, quando uma decisão deve ser tomada a entre a solução de um problema por meios manuais ou automáticos — ou por uma combinação dos dois —uma biblioteca contendo programas para problemas similares pesará fortemente a favor do trabalho da máquina."

Analisando este texto podemos perceber, desde os primeiros momentos da computação, a preocupação em reutilizar para aumentar a produtividade e a constatação de que criar código reutilizável é uma tarefa de custo mais alto do que criar código normalmente.

Weide et al. [Wei+91] criticam de forma muito interessante os *"Software-IC"*, apontando o fato da indústria eletrônica adaptar os **requisitos** para corresponderem às características dos componentes disponíveis, enquanto a indústria de software se caracteriza por produtos extremamente personalizados. Essa é a diferença que torna mais complexa a reutilização de software. Tracks [Tra88] considera essa idéia um "mito", pois o número e a complexidade dos componentes de software supera enormemente o dos componentes lógicos. Além disso, a identificação e definição da interface e parâmetros dos componentes é um problema mais complexo para software. Fatores econômicos, também, tornam o processo de produção de software

distinto do processo de produção de circuitos pois, por exemplo, não existem razões práticas que obriguem o projetista de software a utilizar os componentes disponíveis.

Devemos deixar claro que a reutilização, geralmente falando, é largamente adotada[2]. Cada vez que uma linguagem de programação é utilizada, estamos reutilizando sequências de instruções em linguagem de máquina [Kru92]. A preocupação atual é como aumentar o nível de **abstração** dos artefatos reutilizados ou, visto por outra perspectiva, como aumentar a produtividade dos desenvolvedores de software através de técnicas de reutilização.

II.1.3. Abstração na reutilização

Krueger [Kru92] considera que todas as alternativas de reutilização de software utilizam alguma forma de **abstração**, sendo essa a **característica essencial** de qualquer técnica de reutilização. Tipicamente, um produto de software consiste de várias camadas de abstração construídas uma sobre as outras. Logo, quanto maior for o nível de abstração de uma tecnologia de reutilização maior será o sucesso da sua aplicação, sendo que a eficácia de uma abstração pode ser medida em termos da dificuldade intelectual (definida como a **distância cognitiva**) necessária para utilizá-la. Esse artigo propõe, ainda, algumas regras relativas a reutilização:

1. para uma técnica de reutilização de software ser eficaz, ela precisa reduzir a distância cognitiva entre o conceito inicial de um sistema e sua implementação final executável;
2. para uma técnica de reutilização de software ser eficaz, é necessário que seja mais fácil reutilizar artefatos do que desenvolver o software a partir do início;
3. para selecionar um artefato para reutilização, é necessário saber o que ele faz, e,

---

[2] Muitos autores anunciam a "falta da prática da reutilização no campo da engenharia de software". Tal afirmativa deve ser lida como "falta da utilização de ferramentas e métodos **explícitos** e **formalizados** de reutilização." Uma abordagem como a de Krueger [Kru92] nos faz verificar que a reutilização é largamente praticada, porém em geral de forma implícita, como no caso das linguagens de programação, ou informal, como no caso da cópia simples de projetos anteriores.

4. para reutilizar um artefato de software, eficazmente, é preciso que seja mais rápido encontrá-lo do que construí-lo.

II.1.4. Reutilizar como parte do processo de solução de problemas

Barnes e Bollinger [BB91], afirmam que "A característica que define a boa reutilização não é a de reutilizar software *per si,* mas a de reutilizar o processo humano de solução de problemas." Podemos analisar esta afirmação de acordo com a epistemologia. A produção de software é uma tarefa de investigação tecnológica, que pode ser dividida em seis fases [Bun87]:

1. discernir o problema
2. tratar de resolver o problema com a ajuda do conhecimento (teórico ou empírico) disponível;
3. se a tentativa anterior não for bem sucedida, elaborar hipóteses ou técnicas (ou, ainda, sistemas hipotéticos-dedutivos) capazes de resolver o problema;
4. obter uma solução (exata ou aproximada) do problema com o auxílio do novo instrumental conceitual ou material;
5. por a prova a solução (por exemplo, com ensaios de laboratório ou de campo);
6. efetuar as correções necessárias nas hipóteses ou técnicas, ou mesmo na formulação do problema original.

Fica claro que todo o processo de investigação tecnológica **inclui** a reutilização. É na segunda fase onde podemos identificar a reutilização das experiências anteriores na solução do problema desejado. Em geral, várias são as formas de buscar as soluções anteriores, como: associações, analogias, busca de referências bibliográficas e consulta a colegas.

Compreendendo o processo de investigação tecnológica e analisando a afirmativa de Barnes e Bollinger, podemos concluir que:

> **O que um bom processo de reutilização faz, na verdade, é estender e formalizar, de forma eficaz, a reutilização de experiências anteriores individuais, de uma pessoa ou comunidade, para outras pessoas ou comunidades.**

II.2. As Vantagens da Reutilização

O principal benefício de reutilizar software é o aumento de produtividade. Não podemos, porém, cair na armadilha de pensar que um grupo de desenvolvimento de software que reutilize quatro vezes mais componentes que um segundo grupo terá uma produtividade quatro vezes maior, pois existe um custo relacionado ao processo e à estrutura de suporte a reutilização [You92].

As melhores empresas de desenvolvimento de software alcançam níveis de reutilização entre 70% e 80%, enquanto uma empresa típica atinge níveis entre 20% e 30%. Levado em conta o custo da reutilização, as empresas que utilizam séria e deliberadamente técnicas de reutilização alcançam taxas de 50% a 200% de aumento na produtividade[You92].

Gaffney e Cruickshank [GC92] citam um comunicado pessoal de Allan Albrecht com dados sobre reutilização a nível mundial na IBM no período de 1984 a 1988, correspondentes à 0.5M pontos de função[Pre92] em projetos de gerência de sistema de informação realizados em mais de 50 locais de desenvolvimento. O gráfico e a respectiva tabela, a seguir, mostram os dados coletados por Albrecht:

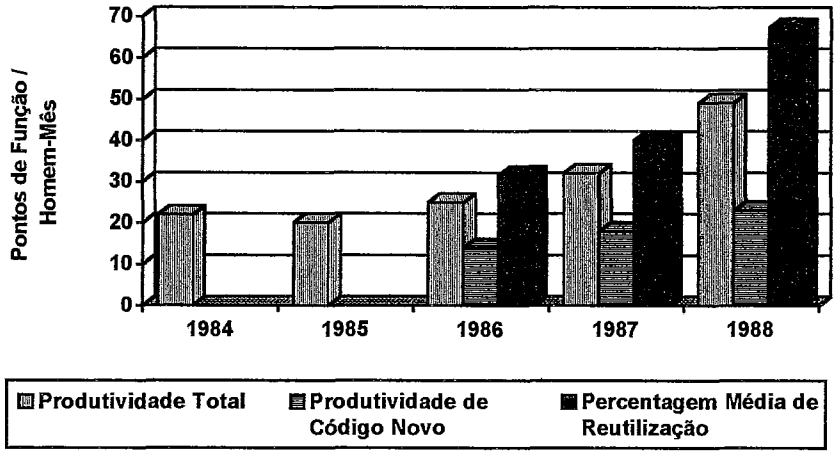


Fig. 3 Experiência mundial de produtividade e reutilização na IBM[GC92].

Ano	Produtividade Total (P) PF/HM	Produtividade de Código Novo (N) PF/HM	Porcentagem Média de Reutilização (R)
1984	22	-	-
1985	20	-	-
1986	25	14	31.5
1987	32	18	40.0
1988	49	23	67.2

Tab. 1 Experiência mundial de produtividade e reutilização na IBM, dados do figura anterior [GC92].

A correlação em cada par das variáveis P,N e R da tabela 1 é muito forte, como mostra a tabela 2, a seguir:

variáveis correlacionadas	variável constante	correlação (r)	$100r^2$
P,R	N	0.9982	98.36
P,N	R	0.9854	97.10
R,N	P	-0.9736	94.79

Tab. 2 Correlações parciais entre variáves (1986-1988) [GC92]

Analisando os dados acima, podemos detectar entre 1986 e 1988 um aumento da taxa de reutilização de 113% , da produtividade em 96% e da produtividade de código novo em 64%. A forte correlação entre as variáveis indica que o aumento de produtividade total é fortemente influenciado pelo aumento da reutilização (P,R) e do aumento da produtividade em código novo (P,N), e que esse último aumento também é influenciado pela reutilização parcial de fases anteriores do desenvolvimento (N,R).

Margono e Rhoads [MR92], também, apresentam dados concretos sobre a análise custo/benefício da reutilização no programa americano para gerência de tráfego aéreo (AAS - Advanced Automation System), onde o custo de criar software reutilizável é duas vezes maior que o de criar software não-reutilizável. Mesmo assim, eles afirmam que *até agora* os benefícios são maiores que os custos.

Também Basile et. al [Bas+92] fornecem dados reais de projetos feitos em Ada/OOD, entre 38K e 185K instruções. Enquanto de 1986 até 1990 a taxa de

reutilização total passou de 0% para 96% (sendo que 85% reutilizados sem modificação) o número de homens/hora por instrução entregue diminuiu em 58%.

Outros benefícios importantes da reutilização são: maior qualidade do produto e maior facilidade na construção de protótipos. A qualidade do software reutilizado aumenta em decorrência de duas características do processo de reutilização: a necessidade de produzir um produto com maior qualidade para que ele seja efetivamente reutilizado e a maior utilização dos componentes reutilizados, o que aumenta a quantidade de testes a que ele é submetido[You92].

A maior facilidade de construção de protótipos provém do fato que a existência de um sistema de reutilização permite que protótipos de aplicações sejam construídos, a partir dos componentes reutilizáveis, e que estes protótipos forneçam funcionalidade (quase) completa, ao invés dos protótipos tradicionais que fornecem apenas as funções de tela[You92].

Robert Pettenhill, em um seminário na Universidade de Genebra, analisou as vantagens da reutilização em uma empresa e identificou três tipos de sistemas que determinam a necessidade do investimento em reutilização:

1. sistemas com uma previsão de vida longa,
2. uma família de produtos, e
3. sistemas evolutivos.

Tais características nos parecem verdadeiras, principalmente se forem utilizadas técnicas de reutilização baseada na Análise de Domínios. Porém, como veremos mais adiante, uma setor de produção de software deve esperar reutilizar componentes entre domínios.

II.3. Empecilhos à Reutilização

Enquanto a maioria dos autores concorda em relação às vantagens da reutilização, podemos dividí-los em dois grupos, quando analisam os incentivos e empecilhos à reutilização.

O primeiro grupo considera que os fatores técnicos são o principal inibidor/incentivador da reutilização. Bertrand Meyer pode ser considerado um exemplo típico deste grupo, chegando a afirmar que:

> "Simplesmente sendo mais organizado não fará o problema de reutilizar ir embora. Os problemas são técnicos, não gerenciais. As respostas estão no projeto orientado a objetos [Mey87]."

O segundo grupo diz que o problema é basicamente sociológico (ou gerencial). Esta preocupação, é aparentemente liderada por Tom DeMarco [DeM91], que cunhou o termo *Pepleoware,* e Prieto-Diaz [Pri91].

Neste ponto é necessário tomar uma posição. Os dois grupos possuem parte da razão. Existem problemas técnicos e gerenciais extremamente importantes. Porém, a tecnologia é essencial, também, para **diminuir** a quantidade e magnitude dos problemas sociológicos. Um exemplo é a aparente falta de sucesso do método estruturado quando não está associado a ferramentas automatizadas [You92]. Logo, apesar do problema de fazer com que um grupo de desenvolvedores use determinado método seja eminentemente sociológico, e que a simples existência das ferramentas não fará com que o grupo reutilize software [Tra88], **é essencial que existam ferramentas que tornem o processo mais eficaz e eficiente**, diminuindo assim a resistência do grupo a executá-lo.

Os empecilhos mais interessantes de serem observados à reutilização são os que se referem a características psicológicas. Entre eles, o fato de "ser mais divertido desenvolver o código por si mesmo" e a "síndrome do desenvolvimento externo" (NIH - ***Not Invented Here***)[3]. Também erros gerenciais, como dar menos crédito a produção por reutilização do que a produção por criação agem **contra** a reutilização.

---

[3] Esta síndrome é muito bem representada pelas palavras de Ken Thompson, um dos criadores do sistema operacional UNIX, "A moral é óbvia. Você não pode confiar em código que não foi totalmente criado por você mesmo" [Tho87]

II.4. Classificando Reutilização

Vários autores propões classificações para formas de reutilização,. Pelo menos três delas devem ser citadas:

1. classificação facetada, proposta por Prieto-Díaz [Pri93];
2. classificação por nível de abstração, apresentada por Krueger [Kru92],
3. classificação das tecnologias de reutilização, apresentada por Biggerstaff e Richter [BR89].

A proposta de Prieto-Díaz apresenta seis facetas, que permitem avaliar várias características do processo de reutilização. A tabela a seguir, retirada de [Pri93], apresenta as facetas de forma suficientemente clara.

por substância	**por escopo**	**por modo**	**por tecnologia**	**por intenção**	**por produto**
idéias, conceitos	vertical	planejada, sistemática	composicional	caixa-preta	código-fonte
artefatos, componentes	horizontal	ad-hoc, oportunística	gerativa	caixa-branca	projeto
procedimentos					especificações
					objetos
					texto
					arquiteturas

Tab. 3 Classificação facetada da reutilização, segundo [Pri93]

A segunda classificação importante aparece na *survey* de Krueger [Kru92], que propõe a classificação de métodos de reutilização por quatro características: abstração, seleção, especialização e integração. O nível de abstração é a característica básica e apresenta oito divisões:

1. linguagens de alto-nível,
2. recuperação de projeto e código,
3. componentes em código fonte,
4. esquemas de software,
5. geradores de aplicação,
6. linguagens de nível muito alto,
7. sistemas transformacionais, e ,
8. arquiteturas de software.

Além de uma classificação, deve ser notado que Krueger apresenta também uma ordenação, baseada no nível de abstração das metodologias.

Biggerstaff e Richter apresentaram, em 1989, uma classificação bem mais simples:

características	**alternativas de reutilização**				
componente reutilizável	blocos		padrões		
natureza do componente	atomico e imutável passivo		difuso e maleável ativo		
princípio de reutilização	composição		geração		
ênfase	biblioteca de componentes	princípios de organização e composição	geradores baseados em linguagens	geradores de aplicação	sistemas transfoma-cionais
exemplos	bibliotecas de subrotinas	orientação a objeto	VHLL	Formatadores de tela	compila-dores

Tab. 4 Uma classificação para tecnologias de reutilização [BR89]

Parece claro que a classificação de Prieto-Díaz poderia incorporar as outras duas, como novas facetas. Principalmente a classificação por nível de abstração é de extrema importância, por considerar a distância cognitiva entre o usuário da metodologia de reutilização e o método utilizado.

II.5. Conceito, Conteúdo e Contexto

O modelo de referência 3C[Wei+91] é uma base para a discussão da reutilização de software proposto pelos membros da *Workshop on Methods and Tools for Reuse.* Esse modelo define e distingue três idéias:

1. **conceito**, uma declaração do que um componente de software faz, sem determinar como o faz (uma especificação abstrata do comportamento);
2. **conteúdo**, uma declaração de como um componente de software alcança o comportamento descrito em sua conceituação (o código que implementa a conceituação), e,
3. **contexto**, aspectos do desenvolvimento do software relevantes para a definição do conceito ou conteúdo que não são parte explícita de nenhum deles (informação adicional necessária para descrever a especificação comportamental ou para escrever a implementação).

O conceito de um componente pode ser visto como sua representação abstrata, enquanto o conteúdo como sua representação concreta. Em um sistema de refinamentos sucessivos, do nível mais abstrato ao nível mais concreto, é possível que uma especificação represente o conceito para um componente e um conteúdo para outro componente.

A representação do conceito de um software é importante pois é provavelmente sobre ele que devemos buscar um componente a ser reutilizado. Isto acontece porque buscamos uma representação concreta para a idéia abstrata que possuímos. **A principal diferença que iremos notar nas várias metodologias de reutilização de software está na forma escolhida para representar o conceito do software, desde simples palavras chaves até representações formais ou linguagem natural**.

II.6. O Processo de Reutilização

Considerando, ainda, a necessidade de comparar sistemas de reutilização, é possível definir, baseado em Redwine [Red89] e Faria [Far91], um processo genérico de reutilização, composto de cinco atividades:

1. **seleção**, envolvendo a identificação dos componentes a serem reutilizados (conteúdo), a partir de uma abstração ou conceito;
2. **adaptação** ou **especialização**, envolvendo a especialização dos componentes reutilizáveis para as necessidades específicas, o que inclui também a compreensão do módulo;
3. **composição** ou **integração**, envolvendo a união dos vários componentes especializados em um sistema completo;
4. **preparação**, envolvendo as tarefas relacionadas com transformação de um componente de software em um componente de software reutilizável, pois podem ter sido criados novos componentes reutilizáveis durante as atividades. Inclui generalização, classificação, manutenção ou outros passos necessários para que um componente de software possa ser selecionado pela primeira atividade, e,
5. **avaliação**, pois todas as atividades devem ser guiadas com o objetivo final de obter uma representação de qualidade do sistema especificado,

envolvendo a validação e verificação necessárias para guiar as outras atividades.

Esse processo pode ser representado pela figura a seguir:

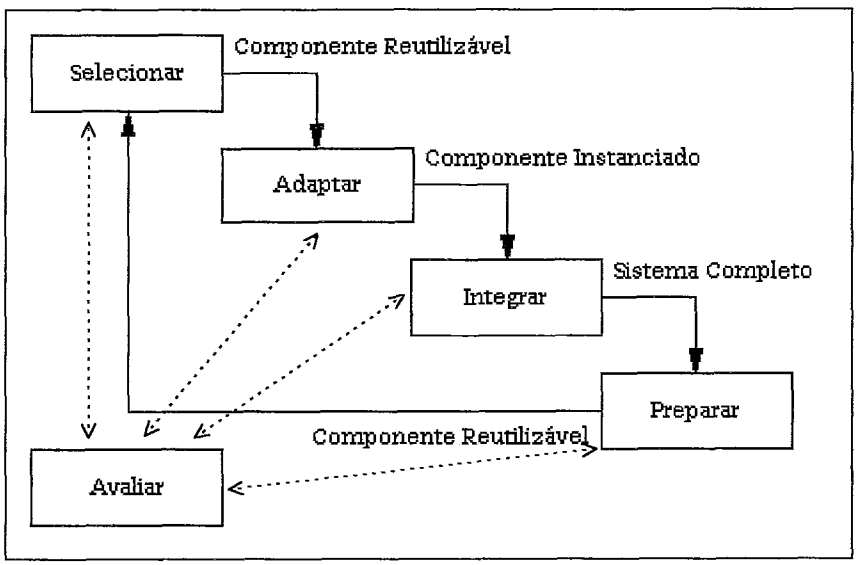


Fig 4 O processo de reutilização de software

Apesar de utilizar o termo composição, o modelo anterior pode ser utilizado para processos onde a geração é a atividade principal, pois podemos analisar a atividade de geração como uma adaptação automática integrada com a posterior composição.

É interessante notar que a maioria das propostas de reutilização apresenta um enfoque mais ligado à prática final, e consequentemente com a seleção, composição e integração de programas de computador.

Basili e Rombach [BR91] defendem a idéia de que o processo de reutilização deve possuir quatro propriedades:

1. toda experiência pode ser reutilizada,
2. os objetos reutilizáveis permitem modificação
3. é possível analisar se e quando a reutilização é apropriada, e
4. a reutilização é integrada ao desenvolvimento de software

Além disso, Basili e Rombach defendem a idéia de um **modelo de reutilização**, que, em geral, é deixado implícito na literatura. Um modelo de reutilização equivale a um modelo de dados dos objetos presentes no sistema de reutiização que suporte o processo explicitamente.

II.7. Experiências e Propostas de Reutilização

Ao mesmo tempo que os pesquisadores em Engenharia de Software desenvolvem cada vez mais projetos visando reutilizar software, a área empresarial demonstra entender que a reutilização é uma das ferramentas de aumento de produtividade mais importantes no processo de desenvolvimento. Isso faz com que sejam muitos os exemplos de sistemas de reutilização existentes, obrigando-nos a fazer uma seleção daqueles que acreditamos mais interessantes para este trabalho.

Para obter uma visão geral da **engenharia de reutilização**, definida como a sub-área da Engenharia de Software preocupada com o processo de reutilização de software, devemos fazer uma avaliação dos projetos mais importantes da área. Os projetos descritos a seguir foram escolhidos pelos seguintes motivos:

1. representarem uma experiência real, ou,
2. representarem uma experiência de grande porte, ou,
3. representarem uma experiência teoricamente importante, ou,
4. representarem uma experiência historicamente importante, ou,
5. possuirem relação direta com esta tese.

As experiências empresariais (ou governamentais) de grande porte, como o AAS[MR92], cujos resultados já foram descritos, nos fornecem dados numéricos e a capacidade de analisar as vantagens de reutilizar. Os projetos de importância teórica, como Draco [Nei91] e PARIS[KRT89], que chegam a implicar em uma nova metodologia de desenvolvimento de software, nos fornecem subsídios para defender idéias e propostas para esta tese, enquanto os projetos historicamente importantes, como o sistema operacional Unix, nos permitem analisar que fatores contribuem para o sucesso de um processo de reutilização. Finalmente, alguns projetos estão diretamente ligados ao assunto desta tese e servem como apoio às idéias aqui defendidas e como parâmetro de comparação.

Apresentaremos agora uma revisão da literatura de reutilização, analisando vários projetos selecionados de acordo com as características citadas anteriormente. Como esta tese defende o uso de especificações formais como mecanismo principal de especificação do conceito dos componentes reutilizáveis, os trabalhos com abordagem semelhantes foram deixados para a próxima seção, que trata de reutilização e métodos formais.

II.7.1. Linguagens de programação

Talvez a mais importante metodologia de reutilização seja o uso de linguagens de programação, tanto que se tornou invisível para o programador comum o que está sendo reutilizado: sequências e padrões de instruções em código de máquina. São centenas ou milhares de linguagens disponíveis, algumas altamente especializadas, outras de uso geral. O sucesso das linguagens de programação pode ser determinado pelo grande aumento de abstração que elas forneceram em relação às metodologias anteriores de programação, como a utilização de montadores. Foi este sucesso que serviu de incentivo para várias outras alternativas de reutilização, como as linguagens de quarta geração, que aumentam ainda mais o nível de abstração e as linguagens de domínio, como no projeto Draco [Nei84].

Ainda dentro de linguagens de programação, caracterizando também uma forma mais alta de abstração que as linguagens tradicionais, devemos citar a reutilização baseada na programação orientada a objetos. Baseando-se em um modelo diferente da programação tradicional a orientação a objetos, onde dados e programas estão encapsulados dentro de um objeto, permite diferentes formas de reutilização, entre elas a **herança**, onde um objeto é implementado como uma modificação de um objeto já existente, ou a **reutilização tipo caixa-preta**, onde um objeto é reutilizado com o conhecimento apenas de sua interface e com a certeza de não provocar efeitos colaterais e, ainda, **classes genéricas**, que são semelhantes a esquemas para classes. Modelos diferentes de orientação a objetos incluem objetos ativos, herança dinâmica, herança múltipla, delegação de tarefas e atores.

II.7.2. Cópia simples

A forma mais comum, e menos metodológica de reutilização, é a cópia *ad-hoc* de código ou projeto de sistemas anteriores ou de fontes de consulta [SpeX]. Apesar de aumentar a produtividade, esta forma de reutilização apresenta problemas como falta de controle de direitos autorais e propagação de erros, pois nada garante que a correção de um erro em um componente original será feita em suas cópias.

II.7.3. Bibliotecas de rotinas

Logo após as linguagens de programação, e a cópia *ad-hoc de código*, certamente as bibliotecas de rotinas são a forma mais comum de reutilização. Principalmente em domínios específicos, como software matemático. Bibliotecas de rotinas, como a IMSL, NAG e CERNlib, fornecem uma coleção de funções específicas que possuem alto grau de aplicabiblidade em seus domínios de aplicação. O uso de bibliotecas diminui muitos dos problemas detectados na simples cópia de código, principalmente quando apoiado por um sistema de gerenciamento de código eficiente.

Uma história de sucesso que não pode deixar de ser citada é a do sistema operacional Unix, cuja a característica principal é ser composto de uma miríade de pequenos programas que podem ser encadeados por um mecanismo especial (*pipe*) e compostos em aplicações mais complexas. Além disso, o sistema fornece uma coleção de funções para o programador que compõe uma biblioteca de uso geral que é largamente utilizada mesmo fora deste sistema operacional. A grande vantagem de cada módulo do sistema operacional é a sua grande generalidade ou, ao inverso, a sua grande especificidade. Juntando módulos específicos (por exemplo, o **ls** que lista um diretório), com módulos gerais (como o **grep**, que busca padrões de caracteres), por meio do mecanismo de pipe, o usuário é capaz de produzir rapidamente ferramentas de grande utilidade.

Todos os sistemas citados até agora (linguagens, bibliotecas, etc...) não necessitam, *a priori*, do suporte de metodologias ou ferramentas de reutilização. Porém devem ser citadas algumas ferramentas que, apesar de simples, possuem grande utilidade e disseminação: as ferramentas que suportam a atividade de seleção.

Podemos citar ambientes de desenvolvimento de software, como editores estruturados com suporte de ajuda, manuais incorporados ao sistema operacional Unix e a busca de rotinas a serem reutilizadas por meio de palavras chave, como é o caso da incorporação da biblioteca CERNlib dentro do sistema de busca por palavras chaves (multi-lingual) XFIND.

A utilização de palavras chave, aliás, é a metodologia de busca de componentes de reutilização que encontra maior disseminação, fazendo parte de quase todos os sistemas de suporte a reutilização. Isso porque fornece uma fórmula fácil de **classificação** e **consulta** de bancos de dados.

II.7.4. Classificação facetada de componentes

Para possibilitar uma classificação mais organizada e uma seleção aparentemente mais fácil, Prieto-Díaz [Pri85] propõe que a classificação facetada seja utilizada para encontrar o componente reutilizável.

Nesse método são representadas várias características dos componentes por meio de facetas, que buscam representar uma dimensão dos componentes, isto é, um taxon que pode possuir um entre vários valores. Cada faceta possui várias palavras chaves, das quais uma deve ser escolhida de modo a representar o componente. A principal vantagem da classificação facetada de bibliotecas de módulos sobre as classificações tradicionais de bibliotecas é não possuir hierarquia (não havendo, portanto, a possibilidade de um componente aparecer em mais de uma posição na árvore hierarquica). Os defensores da classificação facetada, também, acreditam que o fato de criar facetas com um número finito de palavras chave diminui a dificuldade do usuário no processo de seleção, sendo então um processo melhor que o simples uso de palavras chaves ou *full-text retrieval.* Podemos imaginar vários argumentos para discutir essa hipótese, mas a verdade é que, na prática, não existem experimentos reais que possam comprová-la ou não.

O sistema proposto por Prieto-Diaz e implementado em mais de uma oportunidade [Pri85,JP88,Pri91a,Pri93a] inclui um *thesaurus* para transformar grupos de palavras nas palavras chaves esperadas pelo sistema e um espécie de rede

semântica entre as palavras, definindo distâncias cognitivas entre conceitos de forma a permitir buscas relaxadas.

Uma das críticas que podemos fazer ao sistema de classificação facetada corresponde à necessidade de aplicar distâncias cognitivas aos termos *a priori*. Essas distâncias devem, então, ser modificadas pelo responsável pela biblioteca com o decorrer do tempo. Considerando a existência de várias técnicas automáticas de calcular a distância cognitiva entre termos baseadas na teoria de *full-text retrieval*, consideramos que qualquer sistema baseado na classificação facetada de Prieto-Díaz poderia utilizar-se dessas técnicas para aumentar sua performance ou, ao menos, diminuir a necessidade de intervenção humana.

Na COPPE, a classificação facetada foi utilizada para classificar classes reutilizáveis no sistema CAOS [WS91]. No sistema Draco, que propõe uma nova metodologia de desenvolvimento de software baseado em reutilização, foi utilizada para classificar os componentes de software prontos [Kru92].

II.7.5. Um Modelo de Dados para Componentes

A classificação facetada pode também ser vista como um modelo de dados muito simples para representar um componente reutilizável, onde a entidade componente é representada por um conjunto de atributos que podem receber valores pertencentes a conjuntos previamente determinados.

Tratando a modelagem de dados de uma forma mais completa, o sub-comitê TC2 do *Reuse Library Interoperability Group* [Hob93], formado em 1991 para examinar a interoperabilidade entre bibliotecas de software, preparou um modelo de dados definindo informação sobre componentes (*assets*) que as bibliotecas devem ser capazes de fornecer de forma a suportar interoperabilidade[4]. Este modelo é conhecido como *Uniform Data Model* (UDM).

---

[4] Interoperabilidade é a capacidade de operar junto com outros sistemas do mesmo tipo. Geralmente é suportada por protocolos de comunicação e modelos de dados mínimos comuns.

O UDM define um meta modelo usando uma extensão do modelo orientado a objetos que inclue relacionamentos bidirecionais com atributos. Cada elemento possui um nome e um identificador e pertence a uma das quatro sub-classes seguintes: componente reutilizável (*asset*), partes discretas de um componente reutilizável (*element*), biblioteca ou organização.

Várias são as vantagens de possuir um modelo de dados para os componentes reutilizáveis, ao invés de considerá-los, simplesmente, itens classificados em uma biblioteca "plana". Classes, tipos e relacionamentos fornecem uma semântica adicional ao componente que não está presente em modelos "planos". Além disso, a proposta do UDM é de um modelo de interoperabilidade, isto é, um protocolo de comunicação entre bibliotecas, o que é louvável quando analisada a necessidade de implementar comunidades globais de software.

Outro modelo de componentes é o apresentado por Basili e Rombach [BR91] no sistema TAME. Em TAME um componente é modelado de acordo com oito dimensões: nome, função, tipo, mecanismo, entradas e saídas, dependencias, transferência de experiência e qualidade de reutilização. Para cada dimensão os autores descrevem uma pergunta que deve ser respondida de forma a melhor representar as características do objeto reutilizável.

II.7.6. Draco, análise de domínio e os geradores de aplicação

Draco [Nei84] é um sistema de reutilização criado por Neighbors, propondo um novo ciclo de vida orientado para reutilização e baseado na criação automática de software a partir de componentes gerados por uma análise de domínio.

Para possibilitar a automação do processo de produção de software e consequente reutilização, o sistema Draco advoga a execução da **análise de domínio**, isto é, do processo de identificar e organizar conhecimento sobre uma classe de problemas, **o domínio de aplicação**, e apoiar a descrição e solução destes problemas.

Ao analisar o domínio de aplicação o engenheiro de software adquire conhecimento suficiente para criar uma linguagem específica e um conjunto de componentes reutilizáveis para o domínio. Esta tarefa é, certamente, mais complexa

que especificar apenas uma aplicação do domínio e deve ser realizada por um analista experiente. A linguagem própria do domínio é (re)utilizada, junto com geradores (de programas) e os componentes reutilizáveis para produzir novas aplicações no mesmo domínio.

Os conceitos de domínio de aplicação, analista de domínio e análise de domínio podem ser vistos como a principal colaboração do projeto Draco. Apesar dessa formalização ser importante, sistemas menores, semelhantes a "uma instância" do esquema proposto por Neighbors, sempre estiveram presentes na ciência da computação, basicamente representados por programas que geram programas (geradores). Entre as propostas de maior sucesso podemos citar os *compiler-compiler* e geradores de máquinas de estado, como YACC e lex. Todos possuem uma linguagem própria, representam um domínio de aplicação, utilizam componentes pré-fabricados e foram criados a partir de teorias que são semelhantes a uma análise de domínio.

Draco-PUC [Lei+93], desenvolvido na PUC-Rio, é uma máquina que implementa parcialmente o paradigma Draco, podendo ser vista como um gerador de geradores de aplicação. Foi criado a partir da reengenharia do sistema Draco original, com o objetivo de melhorar seu desempenho. Atualmente na versão 2.0, Draco PUC possui versões iniciais de domínios básicos, como Banco de Dados e software para a interface X-Windows.

Semelhante à proposta de Neighbors, existe o projeto ITHACA[Gir92], onde a área de aplicação é representada por uma coleção de *Generic Application Frames (GAFs),* e a análise de domínio é chamada **engenharia de aplicação**. A partir dos GAFs, que são selecionados e instanciados com auxílio de uma ferramenta automatizada (RECAST), aplicações podem ser construídas utilizando uma linguagem de programação (CooL) ou um editor de roteiros visuais (VISTA). Cada aplicação corresponde a um *Specific Application Frame*, que podem então ser utilizado para criar ou modificar GAFs [TN92].

O uso de roteiros visuais nos parece mais fácil que a programação em uma linguagem de domínio. Enquanto a primeira trabalha com blocos fechados que são

compostos por meio de ligações, apresentando um gráfico semelhante a um fluxograma, a segunda recai em todos os problemas relativos à compilação/interpretação e semântica de linguagens de programação.

II.7.7. Seleção baseada no comportamento de funções

Uma das mais interessantes experiências de reutilização encontradas utiliza a semelhança estatística entre uma séria de exemplos de pares de entrada e saída e do resultado da aplicação de funções nos valores de entrada fornecidos [PP92,PP93].

Assim, o sistema faz a seleção dos candidatos à reutilização baseado na semântica implementada pelos componentes reutilizáveis, não por meio de análise de código ou especificações, mas sim analisando a função como uma caixa preta.

Nesse sistema o usuário especifica a interface desejada para uma rotina, dado que é utilizado para determinar um conjunto de rotinas candidatas. O conjunto de rotinas é então aplicado a uma seleção aleatória dos exemplos de comportamento fornecidos, utilizando análise estatística para determinar a função mais semelhante a desejada pelo usuário.

II.7.8. Esquemas

Esquemas ou *templates*, em sua versão mais complexa, são especificações generalizadas de componentes de software que podem ser especializadas a partir de parâmetros.

Muitos sistemas e linguagens utilizam esquemas para permitir a criação de programas generalizados. C++ permite a criação de classes e funções generalizadas por meio de templates[Str91], enquanto as classes generalizadas de Eiffel [Mey88] incorporam dentro de si este conceito.

Na COPPE, o sistema MARTE [Far91] implementa uma biblioteca de esquemas de programas em Pascal, cujo método de seleção é baseado na classificação facetada incluindo, porém, o conceito de hierarquia para uma faceta, o que permite otimizar a consulta.

A arquitetura do ambiente MARTE inclui um gerenciador de templates, incorporando um editor e um esquema de classificação, um ambiente de desenvolvimento, que inclui editor de telas de help, um selecionador de templates e um gerador de código, a partir da parametrização da template e, finalmente, um módulo de manutenção.

II.7.9. Ferramentas utilizando hipertexto

Várias são as propostas de utilizar hipertextos [Con87, Nel88], e *hypermedia*[5] em sistemas de desenvolvimento de software [Con87,MM89]. Segundo Rada et al. [Rad+92] uma abordagem a reutilização baseada em hipertexto requer que a informação seja representada como uma rede de conceitos, o que nos parece muito mais próximo da forma humana de pensar do que, por exemplo, textos lineares ou tabelas relacionais.

Em hipertexto um conceito pode ser representado por um nó ou por um grupo de nós relacionados (subconceitos) por ligações (*links*). A princípio as ligações podem partir de qualquer posição e alcançar qualquer outra posição de qualquer nó. Existem porém vários modelos de dados que podem ser aplicados a hipertextos [DL91,GPS93,Sch+93] que podem controlar as formas das ligações.

Uma das principais características que podemos identificar em reutilização é a de reorganizar em um novo sistema partes de antigos sistemas. O *Practitioner Project* [Rad+92] detecta essa característica como típica de hipertexto e propõe que a reutilização se componha da reorganização, por meio de sistemas de hipertextos, de componentes previamente organizados. Esse sistema permite a busca de componentes a partir de palavras no corpo dos documentos (*full-text retrieval*) ou nos seus cabeçalhos.

---

[5] Devido a falta da existência de um termo em português adequado para a tradução de *hypermedia* utilizaremos a palavra **hipertexto** para representar qualquer forma de documento não sequencial, incluindo os conceitos de *hypermedia* e *hypertext*.

Tradicionalmente, a maioria dos artigos de hipertexto tem se concentrado sobre a capacidade navegacional. Consideramos, como atualmente a maioria dos autores, que um sistema de hipertexto deve incluir a capacidade de busca.

WWW é um sistema de hipertexto formado por uma coleção de protocolos de comunicação e representação desenvolvido dentro da Internet [Ber+92]. Várias ferramentas que podem ser consideradas de reutilização (inclusive esta tese) estão sendo implementadas dentro do que se pode chamar "contexto" WWW, isto é, são capazes de fornecer informações a um cliente de consulta WWW. Entre elas podemos citar a implementação dos manuais da biblioteca de software WWW dentro do próprio sistema e a possibilidade de realizar FTP por meio do cliente de consulta Mosaic (v 2.1). Porém, esta tese foi construída **especificamente** para utilizar a tecnologia de hipertexto com escopo global fornecida por WWW para reutilização, enquanto os outros trabalhos citados fornecem apenas interfaces de um sistema para WWW.

II.7.10.Utilizando analogias

Um paradigma alternativo ao de Análise de Domínios, o qual está se tornando o principal objetivo da tecnologia atual de reutilização, devido as altas taxas de retorno, é o de utilizar **analogias** para reutilização de especificações. Maiden e Sutcliffe [Mai91,MS92] apresentam um sistema onde descrições em linguagem natural são comparadas por sua estrutura, de forma a determinar analogias entre sistemas e determinar assim candidatos à reutilização.

Maiden defende que as alternativas disponíveis para reutilização não são adequadas para reutilização de especificações. Em especial, alternativas baseadas em palavras chaves são simples demais para a complexidade de especificações e a análise de domínio não permite a reutilização **entre** domínios, principalmente no caso de se desejar reutilizar, em um domínio mal compreendido, componentes de domínios onde a experiência já aumentou o nível de compreensão.

II.8. Sistemas Utilizando Especificações Formais

O uso de especificações formais para definir o conceito dos componentes de software é defendido por vários autores [Jon90,KRT89,Wei+91,Ste91,Ste92,WHSX]. Apesar disto, é surpreendente constatar que, apesar de várias experiências descritas aqui demonstrarem o contrário, o grupo de trabalho em métodos formais da *Second International Workshop on Software Reusability* [Tra93] apresentou um relatório decidindo que especificações formais:

1. ajudam a capturar o significado do projeto,
2. **não ajudam a recuperar um componente**,
3. ajudam a compreender um componente,
4. ajudam na manutenção,
5. ajudam a verificar um nível de aceitação e confiabilidade,
6. ajudam a impedir o mau uso de componentes, e,
7. servem de documentação valiosa.

Essa decisão que foi fortemente contestada por alguns membros do grupo e, esperamos demonstrar, por meios dos sistemas a seguir e dessa tese, não é verdade.

Aqueles que decidiram pelo "não" afirmaram que em uma alternativa baseada nas técnicas tradicionais de busca de informação as consultas seriam mais facilmente escritas e mais econômicas que consultas baseadas em técnicas incluindo especificações formais.

Apesar de considerarmos que a afirmativa acima é verdadeira, ela não leva obrigatoriamente a conclusão a que chegaram. Nesta tese, por exemplo, métodos tradicionais de busca de informação são utilizados junto com métodos de busca por especificações formais.

II.8.1. Esquemas e especificações formais em PARIS

O projeto PARIS[KRT89] apresenta um sistema baseado na reutilização de esquemas descritos formalmente, com um método de busca baseado no provador de teoremas de Boyer-Moore.

Nesse sistema, uma descrição do problema fornecida pelo reutilizador é comparada com as definições formais dos esquemas pertencentes ao sistema. Caso seja encontrado uma descrição compatível, este esquema é apresentado ao usuário para transformação em um programa.

PARIS apresenta várias características de interesse para esta tese. Em primeiro lugar, seus componentes reutilizáveis são definidos formalmente. Isso representa a separação de conceito e conteúdo, de forma que o usuário é capaz de compreender o que faz um componente analisando, apenas, sua especificação. Em segundo lugar, utiliza mecanismos de comparação semântica entre a especificação do problema e a especificação dos componentes. Atualmente, PARIS pode ser considerado como a referência básica na área de reutilização utilizando especificações formais para busca de componentes.

II.8.2. Componentes reutilizáveis representando conceito

Weide et a. [Wei+91], a partir do modelo de 3C para a reutilização de software, propõe que componentes reutilizáveis sejam claramente divididos em conceito, conteúdo e contexto. Assim um programador, cliente de uma companhia que forneça componentes de software, poderá consultar um catálogo de componentes, descritos de forma abstrata (conceito), onde cada descrição de componente possuirá uma descrição da sua interface estrutural e de seu comportamento, suficiente para explicar o que ele faz e de que forma pode ser incorporado a um sistema, sem apresentar o código do componente, o que em computação é equivalente a fornecer o produto.

Para a implementação desse paradigma, existem duas características principais exigidas da linguagem utilizada para descrever os conceitos dos componentes reutilizáveis:

1. a especificação do componente abstrato deve ser clara, não-ambígua e compreensível para um cliente ou implementador em potencial, e,
2. a especificação do componente abstrato deve ser livre de detalhes de implementação de maneira a permitir qualquer forma de componente concreto que a implemente.

Assim, apesar da linguagem natural possuir a capacidade de parecer mais clara ao cliente, as necessidades de não ambiguidade exigem que seja utilizada uma linguagem formal (apesar do que, a linguagem natural pode continuar fazendo parte da descrição, por exemplo, como comentários).

Analisando as duas alternativas formais mais aceitas de caracterizar o conceito de um componente, especificações algébricas e especificações baseadas em modelos, Weide et. al. chegam a conclusão que especificações baseadas em modelos são mais fáceis de compreender e reutilizam as teorias matemáticas que devem ser recriadas em especificações algébricas, facilitando também a criação das mesmas.

O artigo apresenta então um modelo de reutilização baseado em componentes de software especificados por uma linguagem formal baseada em modelos (RESOLVE), permitindo a criação de catálogos de componentes.

II.8.3. Especificações algébricas em Spectrum, MENU e CAPS

Alguns sistemas de reutilização baseiam-se em especificações algébricas. Steigerwald [Ste91,Ste92] descreve uma ferramenta desenvolvida dentro do projeto CAPS onde componentes reutilizáveis são selecionados de uma base de software utilizando especificações formais em OBJ3[6] como chave de busca. A metodologia de busca é baseada em um primeiro filtro sintático que compara as assinaturas das funções com a função especificada e um segundo filtro semântico, chamado *query by consistency*, que analisa componentes baseados em seu comportamento. Assim a busca do componente fica independente de palavras chaves e não necessita ser específica a um domínio.

MENU [WHSX] é um sistema que reutiliza especificações em ASI[7] utilizando o seguinte método: um componente reutilizável é visto como uma árvore, cuja raiz é

---

[6] OBJ3 é uma linguagem de especificação formal algébrica destinada a especificar programas em Ada.

[7] ASI é uma linguagem de especificação formal algébrica que permite a descrição de sistemas de forma modular.

sua forma mais abstrata e as folhas suas implementações (parciais). Uma dada especificação é decomposta em subespecificações apropriadas e as subespecificações são comparadas com a base de componentes, em busca de componentes similares. Os subcomponentes que são encontrados, então, são compostos para formar uma implementação para a especificação que iniciou o processo.

Wirsing, um dos autores do trabalho referenciado acima, descreve também o projeto SPECTRUM [Wir92] como uma metodologia de reutilização de componentes de software baseada em métodos algébricos, funcionais e da teoria de tipos. Como em MENU, componentes de software são representados como especificações algébricas estruturadas. A comparação de assinaturas e a relação "implementa" auxiliam na seleção do componente.

II.8.4. Reutilizando especificações em VDM no projeto NORA

Fisher et al [Fis+93] apresentam um projeto de reutilizar componentes de software descritos em VDM por meio de comparação de assinaturas e pré e pós condições. Pré e pós condições são utilizadas em VDM para definir implicitamente o comportamento de funções e operações.

O sistema de reutilização é parte do projeto NORA, onde uma rede de agentes cooperativos se comunicam por meio de mensagens padronizadas e trabalham com uma biblioteca de componentes de software comum. Os componentes são o objeto da reutilização e são acompanhados da informação que os identifica: sua localização, sua assinatura e sua especificação em VDM. A chave de busca consiste em uma especificação em VDM que será comparada com a biblioteca de componentes.

O processo de busca é realizado por uma **cadeia de filtros**, da qual o artigo descreve dois elos: o comparador de assinaturas e o comparador de pré- e pós-condições. A cadeia suporta a adição de mais elos de forma a caracterizar outro tipo de informação.

A partir de contato com estes autores, desenvolvemos uma colaboração que resultou no desenvolvimento de um *parser* VDM por um dos membros do grupo de

Fisher. Esse parser está sendo utilizado agora para obter as árvores sintáticas de especificações VDM que são utilizadas nesta tese.

Esse trabalho apresenta algumas semelhanças com essa tese, sendo a mais importante a utilização de VDM como **conceito** principal para a busca dos componentes reutilizáveis. Esse fato foi reconhecido entre os autores, porém representam apenas a concordância entre políticas de reutilização, pois os métodos utilizados são diferentes e independentes.

II.8.5. Organizando especificações formais

Cheng e Jeng [CJ92] descrevem um sistema onde especificações formais são classificadas por meio de técnicas de clusterização. Utilizando algoritmos baseados em lógica, uma hierarquia de dois níveis é criada. O primeiro nível descreve a relação de generalidade entre os componentes, enquanto o segundo nível utiliza um mecanismo tradicional de clusterização para organizar os componentes que são nós das árvores de generalidade, criando um gráfico conexo que pode ser navegado pelo usuário.

Mittermeir, Mile e Mili [MMM93], partindo do princípio que a chave para uma recuperação de informação eficiente é a disponibilidade de uma ordenação entre as elementos de um banco de dados, propõe ordenar os componentes de software segundo uma ordem parcial, dando ao repositório uma estrutura de *lattice* onde nós correspondem a especificações e programas estão ligados à especificação mais alta para a qual são uma implementação correta. Esta ordem parcial é a relação de refinamento.

A seleção consiste em navegar automaticamente o *lattice* utilizando um provador de teoremas para comparar as especificações com uma especificação desejada.

II.8.6. Que técnicas utilizar?

Quase todos os métodos de reutilização de especificações formais utilizam-se da comparação das assinaturas como primeiro filtro de seleção. Isto reduz a

quantidade de funções disponíveis, o mesmo objetivo de ordenar os componentes, seja por meio de clusterização ou por uma ordem parcial.

Claramente, a verificação da semelhança entre uma especificação desejada e uma especificação pertencente à base de dados pode utilizar algoritmos baseados em lógica, como os fornecidos por provadores de teoremas, já que estes algoritmos caracterizam os aspectos semânticos e sintáticos da linguagem utilizada. Estes algoritmos, porém, são por vezes lentos ou ineficazes para um grupo muito grande de teoremas, podendo ser auxiliados ou mesmo substituídos por outros algoritmos mais rápidos. Por exemplo, uma busca utilizando palavras, no estilo de *full-text retrieval* pode auxiliar, em muito, a determinação da semelhança entre dois componentes.

II.9. Comunidades de Software e a Internet

Uma **comunidade de software** é um grupo de pessoas com uma cultura de software similar [TG90]. Comunidades de software podem se reunir em torno de produtos, como sistemas operacionais, aplicações (como, por exemplo, software para física), ou outros interesses comuns, como passatempos.

Com a tendência em direção a sistemas abertos, é cada vez mais impossível prever o aumento do tamanho dessas coletividades, pois barreiras artificiais como a marca de um produto vão gradualmente desaparecendo.

As maiores comunidades de software usam hoje a rede Internet para trocar mensagens e software, em formatos variados como o de correio eletrônico (*email*), notícias *(news postings*), redes de hipertexto, *relay chats*, ambientes de realidade virtual (*MUD - Multi User Dungeons*). Uma das principais funções dessas comunidades é a de trocar software e informação sobre software para que eles sejam reutilizados por seus participantes. Isso caracteriza, na reutilização, uma perspectiva social [Ara93] raramente explorada em sistemas de apoio à reutilização.

Gibbs, Prevalakis e Tsichritzis [GPT89] defendem a necessidade de sistemas de informação de software, como suporte à reutilização por parte dessas comunidades. Muitos desses sistemas já estão implantados na Internet [Ara93,SK93], e.g.:

1. **ftp** é um protocolo de transferência de arquivos que é a forma mais utilizada de se conseguir um software para reutilização.
2. **archie** é um sistema de busca baseado na comparação de string de caracteres de uma palavra dada com uma base de dados de nomes de arquivos disponíveis por FTP.
3. **Gopher** é um protocolo de comunicação e um sistema que mantém uma base de dados de informações que muitas vezes é utilizado para informar sobre software.
4. **WAIS** é um sistema de busca de informação por *full-text retrieval* muitas vezes utilizado para guardar informações sobre software, como páginas de manual e descrições informais.
5. **WWW** é um protocolo para criação de sistemas de hipertexto com suporte a buscas, muitas vezes utilizado para reutilização de software.

Observando a Internet [Ara93] podemos ficar certos que as comunidades de software, altamente informais, disponíveis na rede são o padrão *de facto* de reutilização. A busca é realizada mediante o uso de consultas a sistemas **independentes** ou envio de mensagens. As respostas são múltiplas e muitas vezes a falta de resposta ou uma resposta negativa **não** quer significar que não existe um software do tipo desejado[8].

II.10. Perspectiva Final

Ao revisar a literatura chega-se a um conjunto de conclusões que explica a abordagem desta tese. Em primeiro lugar, um componente de software possui três partes principais, seu conceito, seu conteúdo e seu contexto. Quando o componente a ser reutilizado é uma parte de um programa, a melhor forma de representar o conceito do componente, ou seja, a abstração que este componente representa, é a definição por meio de uma linguagem formal.

[8] A busca de um programa na Internet é um problema que só é resolvido com uma resposta positiva, pois as respostas negativas tem um conhecimento apenas parcial.

Por outro lado, todas as várias representações de um componente devem estar interligadas de alguma forma, e um sistema de hipertexto nos parece o melhor paradigma para a ferramenta de consulta, contanto que ele inclua a possibilidade de preparar diferentes tipos de consultas e a de organizar e reorganizar documentos. Entre os tipos de consulta devem estar presentes consultas baseadas em métodos tradicionais de busca de informação, que são simples de realizar e de custo barato, e consultas baseadas na criação de especificações (parciais) utilizando métodos formais, que permitem buscas independentes do significado de palavras, como buscas baseadas na estrutura sintática e buscas baseadas na semântica de componentes.

A terceira conclusão é que componentes reutilizáveis devem estar disponíveis, globalmente, alcançando a maior comunidade possível. Junto com o conceito de comunidade de software vem a idéia de serviços pagos de fornecimento de componentes de software, que entendemos exigir o uso de especificações formais para seu funcionamento ideal.

Assim sendo, desenvolvemos um modelo baseado nessas idéias e um sistema implementando o modelo, que será apresentado no capitulo V. Antes porém, para melhor compreensão do trabalho, descreveremos o método VDM no próximo capítulo.

III. MÉTODOS FORMAIS E VDM

"O problema dos anos vindouros será o de fixar um significado real num mar de símbolos neutros"
O Sonho de Descartes, Philip J. Davis e Reuben Hersh

O objetivo deste capítulo é servir de introdução a métodos de especificação formal, em especial ao Vienna Development Method.

III.1. Linguagens e Métodos Formais

Especificações formais usam uma notação matemática e métodos formais de raciocínio para especificar e desenvolver sistemas, com o objetivo de que a implementação final da especificação corresponda a especificação inicial[9].

Berzins e Luqi [BL90] apresentam as seguintes vantagens na utilização de especificações formais:

1. apoio ao raciocínio formal e processamento automático;
2. registro explícito de acordos feitos entre clientes, projetistas e programadores sobre o comportamento esperado do sistema;
3. ajuda à compreensão e uso dos objetos abstratos independentemente de detalhes de implementação;
4. proteção do programador, trabalhando em um determinado nível (de abstração), de detalhes de implementação em um nível inferior, e,
5. proteção dos usuários de conceitos irrelevantes de implementação.

Além disso, ferramentas para projetos computacionais (CASE) que detectam erros de projeto exigem uma sintaxe e semântica bem definida.

[9] O problema da adequação da especificação inicial ao problema não é tratado pela abordagem formal, mas sim por técnicas que são utilizadas em paralelo, como análise essencial e prototipagem da aplicação

Já Andrews e Ince [AI92] apresentam as seguintes vantagens: exatidão, eliminação de influências causadas pela implementação, apoio ao racíocio formal e sucintez.

Um método é formal se possui uma base matemática robusta, tipicamente dada por uma linguagem formal de especificação, que fornece os meios de definir precisamente noções como consistência, completude, especificação, implementação e correção.

Wing [Win90] define uma linguagem de especificação da seguinte forma:

> Uma **linguagem de especificação formal** é uma tripla, $\langle Syn,Sem,Sat \rangle$, onde *Syn* e *Sem* são conjuntos e $Sat \subseteq Syn \times Sem$ é uma relação entre os dois. *Syn* é chamado o **domínio sintático da linguagem**, *Sem* o **domínio semântico** e *Sat* a **relação de satisfação**.

Menos formalmente, uma linguagem de especificação formal fornece uma notação (domínio sintático), um universo de objetos (domínio semântico) e regras precisas indicando que objetos satisfazem a quais especificações (a relação de satisfação).

Outras definições importantes encontradas neste trabalho de Wing são as de especificação não-ambígua e especificação consistente:

> Dada uma linguagem de especificação $\langle Syn,Sem,Sat \rangle$, uma especificação $syn \in Syn$ é **não-ambígua** se e somente se *Sat* mapeia *syn* em apenas um conjunto de especificandos.
>
> Dada uma linguagem de especificação $\langle Syn,Sem,Sat \rangle$, uma especificação $syn \in Syn$ é **consistente** se e somente se *Sat* mapeia *syn* para um conjunto não vazio.

Ou seja, para que uma especificação seja útil, e em geral podemos qualificar essa utilidade como não-ambiguidade e consistência, deve haver um e apenas um significado para essa especificação. O problema da existência de um significado único é muito importante na definição de uma linguagem de especificação formal, como veremos mais adiante.

Apesar disto, é possível que o especificador deseje projetar uma abstração que admita mais de um modelo. A isto chamamos de **especificações fracas** (*loose specifications*) [LL91].

Vários métodos formais, e suas correspondentes linguagens, podem ser encontrados na literatura. Para uma visão global, porém mais detalhada, sugerimos Cohen et al [CHJ86] e Mendes [MA89]. Para uma introdução, o artigo citado anteriormente de Wing [Win90] é bastante adequado.

Nas próximas seções trataremos mais profundamente da definição do domínio sintático e do domínio semântico à luz da **semântica denotacional**.

III.2. Semântica Denotacional

As linguagens de programação têm 3 características principais:

1. **sintaxe**[10], que determina sua aparência e a estrutura de suas sentenças,
2. **semântica**, que atribui um significado a estas sentenças, e,
3. **pragmática**, que indica a forma de uso, como a área de aplicação, da linguagem.

A definição semântica de uma linguagem tem como objetivo fornecer uma referência para uso, implementação, documentação e uma ferramenta de projeto e análise que permita que diferentes interpretadores (pessoas ou programas) dêem um mesmo significado a uma sentença arbitrária.

Para especificar a semântica de uma linguagem, um dos métodos mais atraentes é construir um modelo. Esse modelo pode ser operacional ou denotacional.

A **Semântica Operacional** utiliza um interpretador para definir a linguagem. O significado de uma sentença é dado pela sequência de configurações internas do interpretador correspondentes à sentença. Esse interpretador é uma máquina abstrata

---

[10] A sintaxe pode ser vista como composta de duas partes: léxica e sintática (propriamente dita). Enquanto a léxica caracteriza os símbolos, a sintática caracteriza a estrutura das sentenças construídas com os símbolos. Como a diferença entre símbolo e estrutura pode ser definida arbitrariamente, é comum considerar os conceitos léxicos como parte da sintaxe.

de estados, com vários componentes e um conjunto de operações primitivas. Logo, o modelo é uma máquina virtual capaz de executar a linguagem que está sendo definida.

A **Semântica Denotacional** tem como objetivo mapear as sentenças em seu significado, sendo formalizada associando um objeto conveniente a cada frase da linguagem. Diz-se que a frase **denota** o objeto associado. O objeto é chamado a **denotação** da frase. Uma denotação pode ser, por exemplo, um número, uma função ou mesmo uma nota musical. Para fazer esse mapeamento é utilizada uma **função de avaliação** (ou **de satisfação**). Além disso, para ganhar um visão "referencialmente transparente" da linguagem a ser definida, denotações de frases compostas devem ser definidas somente em termos das denotações das sub-frases.

O problema principal para estabelecer uma semântica denotacional para uma dada linguagem é encontrar objetos matemáticos adequados, que podem servir como denotações.

Por ser baseada em matemática e lógica, a semântica denotacional é mais abstrata que a semântica operacional, baseada em uma máquina abstrata.

III.2.1. Interpretações

É necessário deixar claro a utilidade da semântica denotacional, que muitas vezes pode parecer estar realizando algo óbvio. Quando uma sentença diz, por exemplo:

"uma lista é um conjunto finito e ordenado de números,"

imediatamente associa-se significados as palavras desta sentença da forma mais próxima a experiência diária, o que é chamado de interpretação natural. Nada, porém, nos garante que as palavras da sentença possuam os significados dados por esta interpretação. Um ser de outro planeta poderia interpretar a palavra *finito*, por exemplo, como *desordenado*, e determinar que uma lista não pode existir. Isso acontece porque um significado é fornecido por um isomorfismo entre um conjunto de símbolos governados por regras (sintaxe) e objetos do mundo real [Hof79] (concretos ou abstratos) ou, melhor, objetos conceituais (**construtos**) [Bun87].

Quanto mais complexo esse isomorfismo, mais complexo deve ser o mecanismo necessário para extrair significado dos símbolos.

Quando lidamos com sistemas formais não podemos nos permitir uma interpretação ambígua, por isso a necessidade de formalizar o significado dos símbolos que utilizamos na linguagem e sua relação com os objetos, matemáticos em nosso caso, que possuem os significados (denotações) que desejamos.

III.2.2.Sintaxe

Para especificar uma linguagem formalmente, a primeira necessidade é a definição de um conjunto de simbolos básicos e de estruturas que podem ser construídas com esses símbolos. Como foi dito anteriormente, estas duas características definem a **sintaxe** de uma linguagem. Os símbolos e estruturas são geralmente definidos por meta-linguagens[11] como **BNF** [Tre85] (*Backus-Naur Form*) ou **Diagramas de Sintaxe** [Wir88] (*Railroad Diagrams*). Essa definição é chamada **definição sintática concreta**, pois determina a forma como as sentenças aparecem.

Por exemplo, imaginando uma linguagem aritmética simples, poderíamos ter a seguinte definição sintática concreta (em BNF):

```
<algarismo>   ::=   0 | 1 | 2 | 3 | 4 | 5 | 6 |
                    7 | 8 | 9
<operador>    ::=   + | - | × | ÷
<número>              ::=   <algarismo> |
                    <algarismo> <número>
<expressão>   ::=   <número> |
                    ( <expressão> ) |
<expressão> <operador> <expressão>
```

Fig. 5 Exemplo de definição sintática concreta

A estrutura e os símbolos, junto com a sintaxe concreta, não possuem nenhum significado *per si*, porém permitem a construção de uma **árvore sintática**. Várias técnicas [Tre85,ASU88] permitem que, a partir de uma definição da sintaxe concreta, árvores sintáticas possam ser criadas automaticamente em função de sentenças da

[11] As meta-linguagens são também linguagens, não escapando à característica de possuirem características léxicas, sintáticas e semânticas, porém são linguagens especiais destinadas a descrever outras linguagens.

linguagem. Essa árvore é uma representação que em papel muitas vezes toma uma forma gráfica, permitindo a análise semântica da sentença, isto é, a descoberta do seu significado.

Por exemplo, a expressão 1 + 2 x 3, possui as seguintes árvores sintáticas segundo o exemplo anterior:

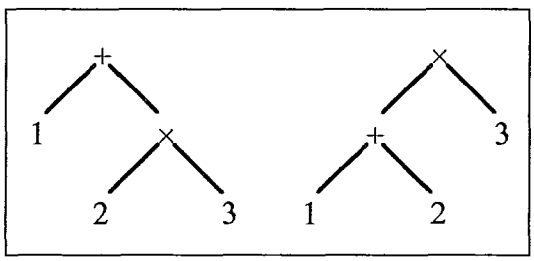


Fig. 6 Árvores sintáticas para a sentença 1+2×3 segundo a sintaxe definida anteriormente.

Schmidt [Sch86] sustenta que as verdadeiras sentenças de uma linguagem estão nas árvores sintáticas, sendo que as sequências de caracteres são apenas abreviações das árvores, muitas vezes ambíguas (como no exemplo dado, onde a sentença é ambígua, porém cada uma das árvores é não-ambígua).

Em geral, definições ambíguas em BNF podem ser reescritas em formas não ambíguas, mas para isso elas têm que conter níveis artificiais de estrutura. Uma definição como a anterior é suficiente para especificar a estrutura de sentenças, mas não para determinar uma árvore sintática única para uma sentença.

Uma **definição sintática abstrata**, não possui referências à representação simbólica utilizada pela linguagem, mas apenas à estrutura da mesma. Ela não lida com os símbolos, mas apenas com a estrutura. Novamente, Schmidt [Sch86] lembra que a verdadeira definição da linguagem é a abstrata. Além disso, a definição abstrata tem a vantagem de não tratar problemas técnicos de *parsing*. É a definição abstrata da sintaxe que é usada para analisar as árvores sintáticas e buscar o significado das sentenças.

III.3. VDM

III.3.1. A História de VDM

Segundo Plat e Larsen [Pla93], VDM é um termo genérico, que indica vários dialetos e metodologias originadas do trabalho do grupo de Heinz Zemanek, no laboratório de Viena da IBM.

Em 1970, a partir da idéia de utilizar uma semântica operacional para definir a linguagem PL/I, o grupo liderado por Zemanek criou uma meta-linguagem chamada *Vienna Definition Language* (*VDL*). Apesar do sucesso do método, considerou-se que a semântica operacional criava complicações desnecessárias ao raciocínio formal. Em 1972 o mesmo grupo tentou uma nova abordagem, chamada **semântica denotacional**, desenvolvida por Dana Scott e C. Strachey na Universidade de Oxford, definindo a linguagem **Meta-IV** (agora chamada VDM-SL). A definição formal de PL/I seguindo um estilo denotacional é considerada o marco do nascimento de VDM.

Com a dissolução natural do grupo, a cultura de VDM se espalhou, principalmente na Europa, e o método foi utilizado em várias áreas de aplicação, o que levou à criação de vários dialetos. Essa proliferação de dialetos acabou por diminuir a sua aceitação industrial, fazendo com que, em 1986, fosse iniciado o esforço de desenvolver uma versão padrão da linguagem, VDM-SL (*Standard Language*).

A padronização foi iniciada pelo *British Standards Institute* (*BSI*), sendo que em 1991 a necessidade de um padrão foi reconhecida pelo *Joint Technical Committee 1 of the International Standards Organization and the International Electro-technical Commission (ISO/IEC JTC1)*. Assim, foi formado o grupo de trabalho SC22/WG19, que vem trabalhando de forma conjunta com o grupo BSI IST/5. Atualmente o grupo ISO conta com a participação de membros da Inglaterra, Holanda, Reino Unido, Dinamarca, Canadá, Estados Unidos, Japão, Nova Zelândia e Brasil, tendo realizado seu segundo encontro em abril de 1993 em Odense, Dinamarca.

III.3.2.A Linguagem e o método

VDM é um método formal baseado na semântica denotacional, orientado para a criação de modelos. O desenvolvimento em VDM é feito pelo refinamento sucessiva de modelos abstratos até que se chegue a uma implementação concreta [Win90]. A principal ferramenta desse método é a linguagem de especificação VDM-SL, antes chamada Meta-IV[12].

Um modelo em VDM é construído a partir de duas partes, que representam o modelo de dados e o modelo funcional, e recebem o nome de **Abstração Representacional** e **Abstração Operacional.** Na verdade, o que se define é, respectivamente, um modelo estático e um modelo dinâmico para o sistema.

A abstração representacional é feita a partir de tipos de dados, sendo que a linguagem suporta seis mecanismos de estruturação de dados, ou de construção de tipos, a saber: conjuntos, sequências, mapeamentos, composição, produto cartesiano e união. Os tipos são construídos a partir de vários tipos básicos, incluindo a enumeração. Tipos compostos de dados são chamados de **domínios** em VDM. Domínios definem classes, em geral infinitas, de objetos, outra denominação para um **tipo abstrato de dados**. **Invariantes**, que são expressões lógicas que devem ser verdadeiras em qualquer estado do sistema, podem ser utilizadas para criar subclasses.

A abstração operacional é realizada por meio de funções e operações, que podem ser definidas implicitamente, por meio de pré e pós condições, ou explicitamente, por meio de construções aplicativas para funções ou construções imperativas para operadores.

As operações podem modificar o estado do sistema, definido como uma coleção de objetos globais. Às funções não é permitido alterar o estado global, significando

---

[12] Leia-se *Metaphor*, ou metáfora, um trocadilho feita pelos criadores da linguagem. Nunca houve uma linguagem chamada Meta-I, II ou III.

que uma função não pode ter efeitos colaterais (na verdade, uma expressão não pode conter efeitos colaterais).

III.4. VDM e o Desenvolvimento de Software

Segundo Cohem et al [CHJ86], o método padrão de desenvolver programas em VDM pode ser representado pela Figura 7. A idéia é passar por vários níveis de projeto e, a cada estágio, adicionar mais detalhes de implementação (processo chamado de **reificação**).

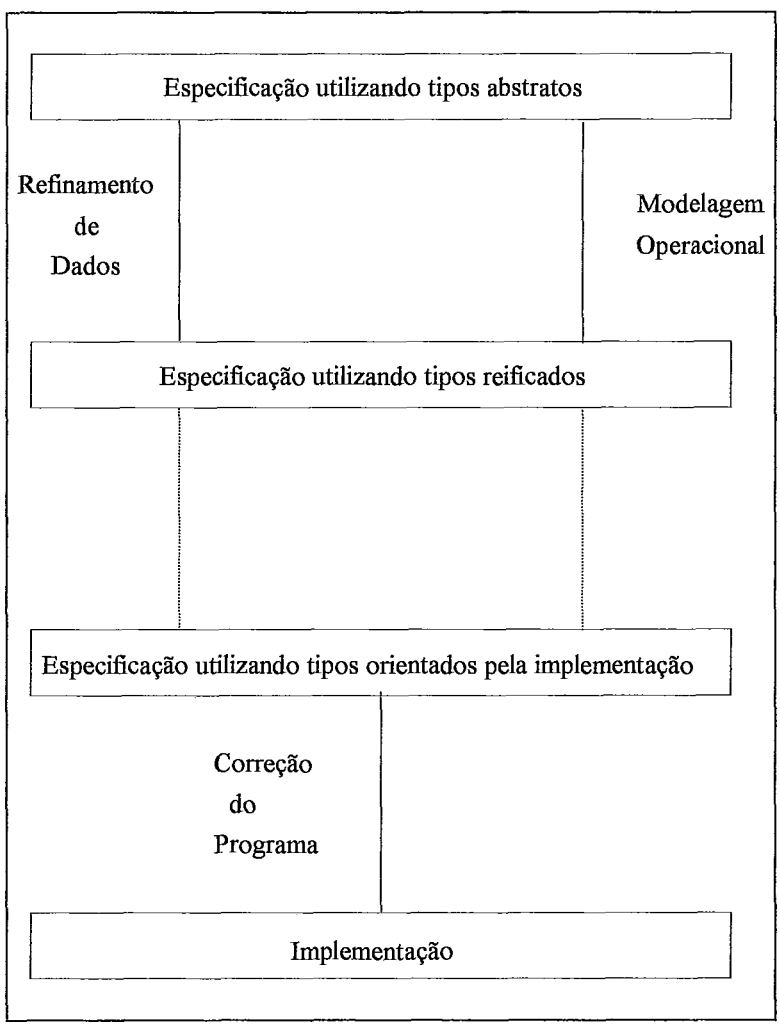


Fig. 7 Método padrão de desenvolver sistemas em VDM, segundo Cohen et al. [CHJ86]

A cada passo, estruturas abstratas de dados são refinadas em estruturas mais orientadas para a implementação (conjuntos em árvores, árvores em matrizes, por exemplo) e, cada vez que isto é feito, as novas especificações de operações devem ser

formuladas nos termos das estruturas de dados reificadas correspondentes às do nível mais abstrato. Em um determinado ponto as estruturas de dados especificadas são suficientemente concretas para serem representadas em uma linguagem de programação, permitindo que o programa seja escrito.

A cada estágio do refino dos dados, é necessário formular "funções de recuperação" que permitam provar que cada valor do tipo abstrato pode ser representada com a estrutura reificada (uma noção chamada **adequação**). Além disso também é possível verificar que as operações reificadas modelam corretamente os efeitos das operações abstratas. No último estágio é necessário provar que o código escrito executa corretamente a especificação menos abstrata.

Outro método, encontrado em [AI92] utiliza uma técnica estruturada para guiar o refino de dados e as necessidades de prova. Cliff Jones, em [Jon90], a principal referência da linguagem, fornece maiores explicações sobre funções de recuperação, adequação e o método utilizado para o refino de dados.

VDM é um método de desenvolvimento de software aplicável de forma geral[PKT92] e que pode ser utilizado junto com outro método ou como método base de metodologias padronizadas de desenvolvimento de software. Nico Plat, Peter Gorm Larsen, Jan van Katwijk e Hans Toetenel [Pla93,PT92,PLTs] estão desenvolvendo trabalhos para aplicar VDM junto com Diagramas de Fluxo de Dados, seguindo o método SASD e segundo o padrão DoD-STD-2167A.

III.5. Estrutura da Linguagem

A linguagem VDM modela um sistema ou um método de acordo com os dois **sub-modelos** inter-relacionados. O primeiro é o modelo estático, que indica os estados possíveis do sistema por meio de estruturas abstratas de dados. O segundo é o modelo dinâmico, que indica o comportamento do sistema por meio de funções e operações.

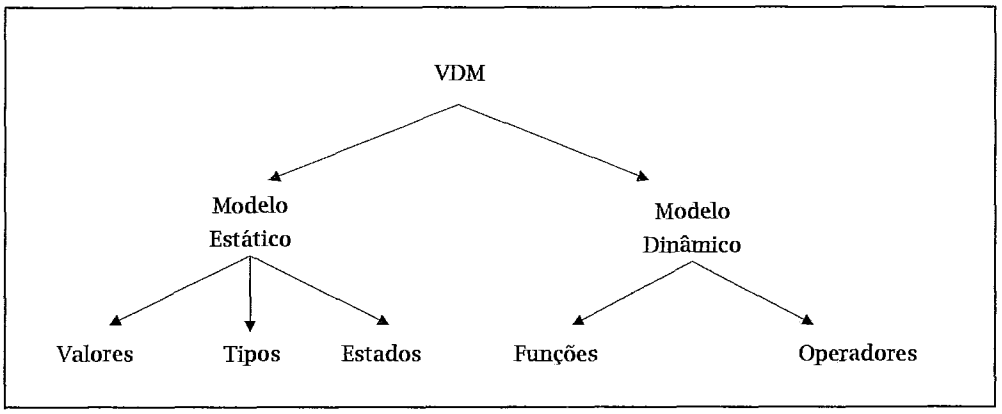


Fig. 8 Uma especificação em VDM é composta de um modelo estático, composto de valores, estados e tipos e de um modelo dinâmico, composto de funções e operadores.

III.6. O Padrão ISO

Nesta seção descrevemos o documento atual que define a linguagem VDM. Esta descrição tem os seguintes objetivos:

1. descrever o padrão como um todo, mostrando o significado de cada parte e as relações entre as partes, caracterizando como elas se unem para formar a definição da linguagem, e,
2. mostrar como cada parte cumpre o que se espera de uma definição pela semântica denotacional, como foi descrito nas seções anteriores.

III.6.1. Descrição geral

Segundo [PL92] o padrão pode ser dividido em cinco componentes principais: sintaxe, representação simbólica, semântica estática, semântica dinâmica e mapeamentos sintáticos.

III.6.2. Sintaxe Abstrata

A sintaxe é descrita em duas formas: a sintaxe concreta em EBNF e a sintaxe abstrata em um sub-conjunto de VDM-SL. Existem dois níveis de complexidade sintática: a sintaxe propriamente dita é chamada "**Outer Abstract Syntax**" (**OAS**) e uma versão simplificada, que é utilizada para a definição da semântica formal da linguagem, a "**Core Abstract Syntax**" (**CAS**). Isto fornece um compromisso entre a facilidade de se determinar uma semântica formal para uma especificação e a

facilidade de escrever uma especificação. Podemos dizer, invertendo o raciocínio anterior, que a OAS é uma CAS "açucarada" (com mais *syntatic sugar*).

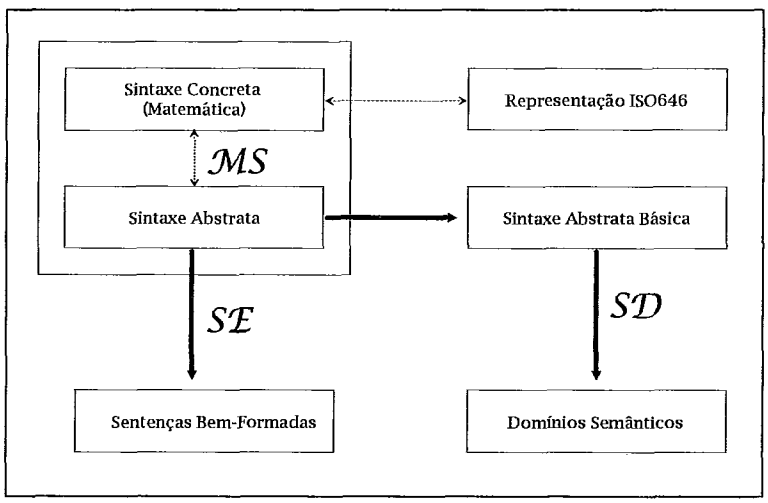


Fig. 9 A estrutura do padrão VDM

III.6.3.Sintaxe Concreta

A representação simbólica, ou sintaxe concreta, define a representação dos símbolos da linguagem em duas versões: uma, chamada **matemática**, que usa símbolos matemáticos, e outra, **ISO 646**, que utiliza apenas símbolos desse mesmo conjunto de caracteres padronizados (semelhante ao conjunto ASCII).

A representação matemática é a referência para a representação ISO 646, só existindo a representação em EBNF para a primeira. Normalmente, os participantes do grupo ISO chamam a sintaxe ISO 646 de *ASCII Sintax,* como veremos mais adiante. Fica claro que o objetivo de uma versão ASCII da sintaxe é permitir que terminais comuns de computadores sejam utilizados para criar definições VDM.

III.6.4.Os Domínios Semânticos

Existem dois domínios semânticos. O mais importante forma o **Domínio da Semântica Dinâmica (DSD)**, e fornece as denotações para as especificações. O segundo, **Domínio da Semântica Estática (DSE,** ou ***SSD*)**, permite que alguns testes

de existência de denotação para uma especificação sejam realizados a partir da análise estática de uma especificação.

III.6.5.Os Mapeamentos

O mapeamento sintático (**MS**) fornece a transformação de sentenças em OAS para CAS, permitindo então que um significado formal seja dado às especificações por meio da **Semântica Dinâmica (SD** ou ***DS*)**. Para que seja possível verificar que uma sentença, sintaticamente correta, em VDM é bem-formada, utiliza-se a **Semântica Estática (SE** ou ***SS*)**.

De acordo com as seções anteriores, a SE e SD, são as relações de satisfação entre os domínio sintáticos OAS e CAS e os domínios semânticos correspondentes, DSD e DSE.

III.7. Exemplos da Linguagem

A melhor forma de conhecer uma linguagem, rapidamente, é por meio de exemplos. Nesta seção apresentaremos alguns exemplos que devem esclarecer tanto a estrutura de uma especificação VDM, quanto formas de utilizar essas especificações para determinar o comportamento de um componente de software.

Em VDM uma pilha de inteiros pode ser especificada da seguinte forma, bastante abstrata:

```
types

Pilha = ℤ* -- Pilha é uma sequência de inteiros que pode ser vazia

-- isto é um comentário

state

  p : Pilha


operations


-- new  inicializa a pilha para a sequência vazia
new()
ext wr  p : Pilha
post p = [ ]
```

```
-- push insere um elemento na pilha
push(e : ℤ)
ext wr p : Pilha
post p =  [e] ^ p~

-- a linha anterior diz que o valor
-- de p deve ser igual a concatenação
-- da sequência formada pelo inteiro e com
-- o valor antigo de p

-- pop devolve o último elemento posto,
-- se a pilha não estiver vazia
pop() e : ℤ
ext wr p : Pilha
pre p ≠ [ ]
post  p~ = [e] ^ p

-- neste caso dizemos que o valor antigo de p
-- deve ser igual a [e] concatenado com o valor
-- novo de p! Uma forma muito abstrata de
-- retirar [e] de p.
```

Segundo o método, a uma representação abstrata deve se seguir uma mais concreta. Poderíamos continuar com a mesma representação para a pilha, porém utilizar agora os operadores **hd** e **tl**, que representam o primeiro elemento da pilha e a pilha sem o primeiro elemento, para definir a função pop:

```
push(e : ℤ)
ext wr p : Pilha
post p =  [e] ^ p~

pop() e : ℤ
ext wr p : Pilha
pre p ≠ [ ]
post  (p = tl p~) ∧ ( e = hd p~)
```

Admitindo que a pilha seria implementada em um vetor de até 100 itens, poderíamos trocar essas definições para:

```
types

Pilha = ℤ* -- Pilha é uma sequência de inteiros que pode ser vazia
inv  length(mk-Pilha) < 100

state
  p : Pilha
  s : ℕ
inv length(p) = s
```

```
operations

-- new  inicializa a pilha para a sequência vazia

new()
ext wr  p : Pilha
     wr s : ℕ
post (p = [ ]) ∧ (s = 0)

-- push insere um elemento na pilha

push(e : ℤ)
ext wr p : Pilha
     wr s : ℕ
post (s = s~ + 1) ∧
       (p[1] = e) ∧
       (let j : ℕ in j < s ⇒ p[j] = p~[j-1])

-- pop devolve o último elemento posto,
-- se a pilha não estiver vazia
pop() e : ℤ
ext wr p : Pilha
     wr s : ℕ
pre p ≠ [ ]
post  (e = p~[1]) ∧
       ( s = s~  - 1) ∧
       (let j : ℕ in j < s ⇒ p[j] = p~[j+1])
```

E, finalmente, querendo determinar o algoritmo a ser implementado para push, são válidas as seguintes definições:

```
operations
-- push insere um elemento na pilha
-- 'p' e 's' são variáveis globais
 push(e : ℤ)
( dcl j : ℕ;
  dcl pp : Pilha;
  s  := s + 1;
  j := 1;
  pp[1]:=e;
  if s > 1 then
    while j <= s do
      pp[j] = p[j-1];
    end
  else
     skip
  end;
  p = pp
)
```

III.8. Ferramentas para VDM

Nico Plat e Hasn Toetenel [PT89] descreveram em 1989 diversas ferramentas disponíveis que suportam o desenvolvimento em VDM ou em dialetos de VDM. Esse trabalho descreve 16 ambientes ou coleções de ferramentas, classificadas segundo o tipo de suporte oferecido, a integração e o dialeto de VDM utilizado.

Dos 16 ambientes, 5 eram financiados por projetos ESPRIT, 3 apenas por universidades, 3 por colaborações entre universidades e indústrias e 3 apenas por indústrias. Os autores caracterizaram que apenas empresas e universidades localizadas na Europa mostram algum interesse por VDM.

Entre os ambientes analisados, 11 possuiam suporte sintático e 10 suporte semântico. Todos os ambientes possuem *type-checkers*, editores orientados a sintaxe e *pretty-printers*. O projeto RAISE [Bjo+85] ainda desenvolveu suporte à pragmática.

Atualmente, o conjunto mais fácil de ser encontrado é a IPTES VDM-SL Toolbox [IPT92], composto de um analisador sintático, um interpretador, um *pretty-printer*, um *debugger* e um analisador da semântica estática de VDM.

III.9. Extensões a VDM

Muitos são os dialetos e extensões existentes para a linguagem e para o método VDM. Atualmente, devido a existência de um padrão da linguagem [ISO92] , a tendência é o desenvolvimento de extensões ao padrão que resolvam as questões que não foram respondidas por ele, como a implementação de módulos ou o uso de objetos.

•Módulos

Várias são as extensões que sugerem a utilização de módulos, que foram explicitamente deixados de fora do padrão VDM/SL. como método de encapsulamento em VDM. Jones [Jon90] descreve uma possível implementação de módulos.

Basicamente existem dois tipos de módulos que podem ser introduzidos na linguagem VDM plana. Plat e Toetenel [PT89], descrevendo quatro dialetos distintos de VDM, incluido VDM/SL, apresentam o primeiro como apenas uma coleção de definições, com um mecanismo de importação e exportação, e o segundo como módulos que permitem definir um tipo abstrato de dados.

•RAISE

RSL (RAISE Specification Language) [Bjo+85,NT89] é o dialeto de VDM criado para o projeto RAISE, desenvolvido com o objetivo de suportar o desenvolvimento de software grande e complexo pela indústria. A linguagem extende VDM principalmente no sentido de criar um **método** bem definido.

•VDM++

VDM++ [DK92,Dür92] é uma extensão orientada a objetos de VDM que oferece classes, objetos e herança simples e ainda um formalismo adicional para especificar a sequência de invocação de métodos permitida. Uma especificação em VDM++ pode ser automaticamente transformada em uma especificação em VDM/SL, tendo então uma semântica bem definida.

IV. CAB: Um Ambiente de Desenvolvimento em VDM

"Processos computacionais são seres abstratos que habitam os computadores. Enquanto eles evoluem, manipulam outras abstrações chamadas dados. A evolução dos processos é dirigida por um padrão de regras chamado programa. Pessoas criam programas para dirigir os processos. Em efeito, nós conjuramos os espíritos do computador com nossos feitiços."
H. Abelson, G.J. Sussman e J. Sussman, in *Structure and Interpretation of Computer Programs.*

CAB, acrônimo para Comprehensive Application Builder, é um ambiente de desenvolvimento que implementa o método VDM. Em CAB, o desenvolvedor pode criar especificações formais, modelando uma aplicação e, a partir dessas especificações, desenvolver um programa em qualquer linguagem de programação.

Para facilitar o trabalho de modelagem de dados, CAB introduz VDM/GDL, uma representação gráfica para o representação abstracional de VDM. Para permitir a manutenção de uma relação entre a implementação e a especificação, CAB introduz CAB/DEL, uma linguagem capaz de descrever como um módulo de um programa implementa uma especificação em VDM.

Vários protótipos de CAB foram construídos e são descritos neste capítulo, incluindo suporte ao método estruturado e ao conceito de domínio de aplicações.

IV.1. Histórico

A primeira etapa desta tese, realizada no CERN - Centro Europeu de Física de Partículas, em Genebra, teve como objetivo definir e implementar um ambiente de desenvolvimento baseado em VDM. CAB [XLS90,XLS91] (*Comprehensive Application Builder*), é uma ferramenta CASE que suporta a especificação, o refinamento e a implementação de um sistema utilizando VDM como método de desenvolvimento.

A implementação foi realizada utilizando um meta-ambiente de desenvolvimento que utiliza um banco de dados baseado no modelo de entidades e relacionamento (com extensões de herança e hierarquias), editores gráficos e editores estruturados de texto com suporte a hipertexto e capacidade de coordenação entre a parte gráfica e a parte textual. Esse meta-ambiente, chamado YPSIS Toolbuilder [YPS92], é disponível em diferentes máquinas que suportam o sistema operacional Unix com X-Windows. A implementação foi testada em workstations Sun IPX, DecStation 3100 e HP-9000.

O ambiente CAB foi desenvolvido segundo o ciclo de vida de prototipações evolutivas, de modo a investigar opções e requisitos esperados de um ambiente de desenvolvimento de software. O objetivo principal foi construir um ambiente de desenvolvimento que permitisse a reutilização de componentes.

Para que os componentes pudessem ser corretamente compreendidos, determinamos que deveriam ser formalmente especificados e adotamos VDM como linguagem de especificação. Esse método, porém, não cumpre o papel de fazer a ligação completa entre a especificação formal e a implementação, pois as regras de sintaxe e semântica das linguagens de programação são bastante diferentes das regras de VDM. Para isso desenvolvemos uma linguagem que descreve como um programa implementava essa especificação. Essa linguagem foi chamada de CAB-*Description Language*, ou CAB/DEL.

Outro ponto que notamos era a falta de uma notação gráfica em VDM, principalmente para a abstração representacional (modelo de dados). Tendo em vista a grande facilidade de analisar modelos de dados que as notações gráficas fornecem [Shu88,Cha90], decidimos definir e implementar uma representação gráfica para a abstração representacional de VDM, chamada VDM/*Graphical Data Language*, ou VDM/GDL.

Um ambiente suportando a definição do modelo de dados em VDM/GDL foi implementado, onde a descrição em VDM/GDL era automaticamente transformada em VDM/SL. A descrição em VDM/SL era feita por um editor estruturado, evitando assim erros de sintaxe. As especificações podiam ser refinadas até que fossem

implementadas em uma linguagem de programação. No último passo (VDM/SL para linguagem de programação) era feita uma descrição em CAB-DEL identificando como o módulo definido na linguagem de programação implementava a definição em VDM.

Finalmente, tendo o ambiente pronto, investigamos outras possibilidades de utilização de VDM. Uma extensão foi feita utilizando o método de desenvolvimento estruturado, alcançando uma proposta semelhante as de Plat [Pla93] e Andrew e Ince [AI92].

IV.2. Um ambiente de desenvolvimento para VDM

Descrevemos agora a versão básica do ambiente CAB que admite, basicamente, oito atividades:

1. identificar banco de dados e usuário,
2. criar/editar sistemas,
3. criar/editar abstração representacional (modelo de dados) em VDM/GDL,
4. criar/editar operações e funções em VDM,
5. criar/editar módulos do programa, em Fortran ou C,
6. criar/editar descrições CAB-DEL,
7. criar/editar o modelo de dados em VDM/SL (opcional), e,
8. manutenção do conjunto de palavras-chaves

Uma seção completa no sistema CAB inicia-se quando o usuário escolhe uma base de dados, passando então a uma tela inicial onde se identifica. O passo de escolha do banco de dados é imposto pelo meta-ambiente[13], enquanto o passo de identificação do usuário tem como objetivo permitir que mais de um usuário acesse o mesmo banco e os mesmos sistemas. Em CAB, a um dado momento, cada usuário pode acessar qualquer sistema no modo de leitura, porém cada sistema só pode ser acessado por um usuário no modo de escrita e edição.

[13] Um meta-ambiente é um ambiente de desenvolvimento destinado a produzir ambientes de desenvolvimento de software.

Cada usuário possui uma lista de sistemas disponíveis para trabalho. Os sistemas são identificados por um nome e um comentário. Um sistema é formado por um modelo de dados (em VDM/GDL e com sua contra-parte em VDM/SL), um conjunto de operações e funções (em VDM), um conjunto de módulos (na linguagem de programação) e um conjunto de descrições em CAB/DEL.

A atividade normal de criação do sistema deve se iniciar com a definição do modelo de dados em VDM/GDL. O paradigma do meta-ambiente define a existência, a cada instante, de duas telas, uma gráfica e uma textual. Utilizando esse paradigma, a descrição em VDM/SL é automaticamente gerada na tela textual quando os objetos gráficos são criados. Isso permite uma rápida assimilação da ferramenta gráfica para usuários habituados com a notação tradicional de VDM/SL.

A partir da descrição de dados, o usuário pode definir suas funções e operações em VDM/SL. Apenas os tipos de dados definidos no modelo de dados podem ser utilizados nessas definições, garantindo parte da correção semântica (estática) da definição.

Quando o usuário possui uma especificação suficientemente completa, pode decidir iniciar a programação. Para cada módulo criado deve ser feita também a declaração em CAB/DEL descrevendo que função ou procedimento em VDM/SL está sendo implementada.

A sequência de atividades está representada pela figura 10.

Cada tarefa é suportada, basicamente por uma tela que lista o conjunto de itens disponíveis (funções, operações, módulos, etc...) com um nome e um comentário. Clicando neste nome, é possível navegar para uma tela que fornece a implementação do item.

O apêndice I contém exemplos das telas durante a especificação de um sistema hipotético.

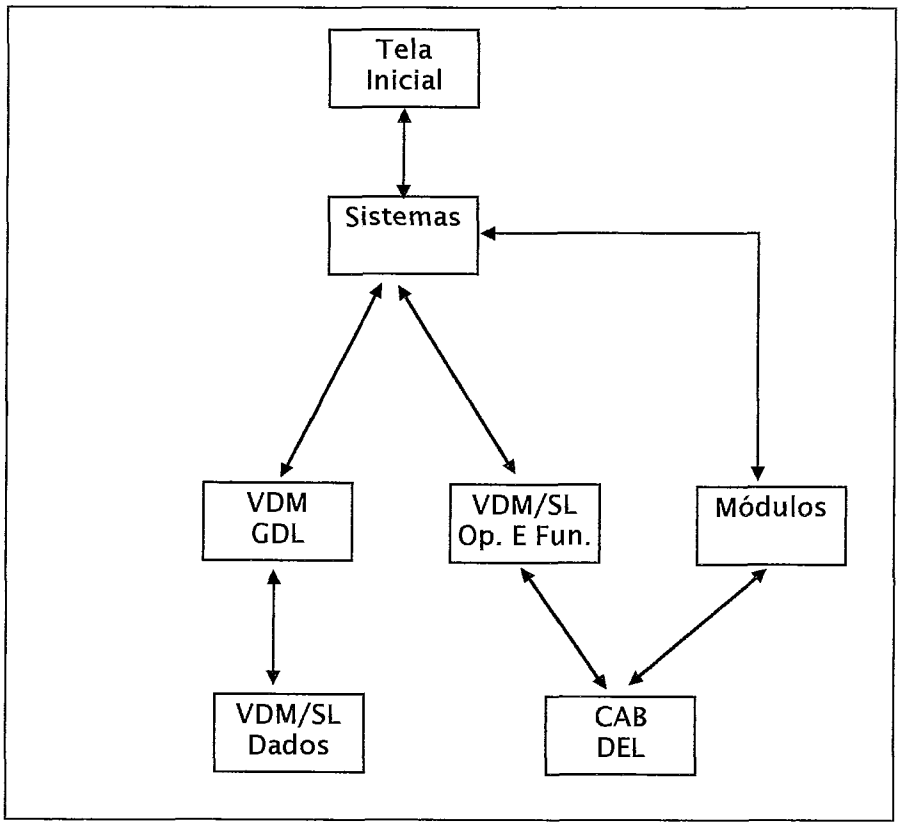


Fig. 10 Sequência de tarefas de criar/editar que o usuário podia seguir por meio de ligaçõess de hipertexto.

IV.3. A linguagem VDM/GDL

Tendo em vista facilitar a modelagem de dados em VDM, foi desenvolvida uma linguagem gráfica capaz de representar toda a abstração representacional da linguagem (com exceção de invariantes e estados iniciais que podem, também, ser vistas como uma parte híbrida com a abstração operacional).

Muitas são as vantagens de utilizar linguagens gráficas. Shu [Shu88] e Chang [Cha90] apresentam vários sistemas e formas de abordagens que demonstram características como maior facilidade de comprensão, diminuição da distância cognitiva entre o usuário e a linguagem, e maior facilidade de manipulação. Por outro lado, o sucesso de métodos de análise e projeto baseados em representações gráficas indica que "falta algo" no método VDM. Entendendo esse fato, outras linguagens formais, como LOTOS, já apresentam representações gráficas em sua definição [VSC93,Pur92].

VDM/GDL é melhor analisada como uma nova representação para uma parte da linguagem VDM. A relação entre especificações VDM/GDL e VDM/SL é de um para um, a menos da ordem. O problema da ordem pode ser tratado, no gráfico, considerando um ponto inicial em cada figura e o sentido dos ponteiros do relógio como uma forma de ordenação, porém não é importante para a semântica da especificação gráfica, apesar de ser importante para a semântica da especificação escrita. O mesmo problema pode ser encontrado na representação gráfica do modelo de entidades e relacionamentos e não é um empecilho a sua utilização.

IV.3.1. Definição da representação gráfica

Inicialmente, foram definidos símbolos para os tipos básicos de VDM e para as construções de *record* e *composite* e *named type.* Este símbolos são mostrados na figura 11.

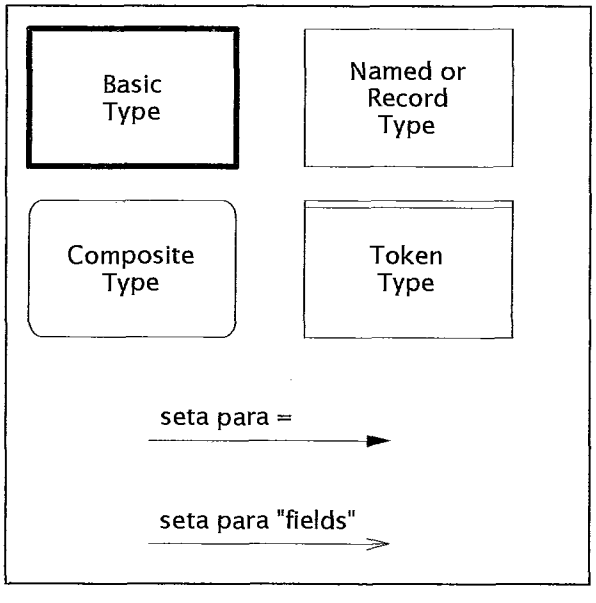


Fig 11 Construções mais simples em VDM

Assim sendo, o gráfico da figura 12 e a seguinte descrição VDM são equivalentes:

```
numero = ℤ
data ::      dia : numero
             mes : numero
             ano : numero

valorNulo = NULO
```

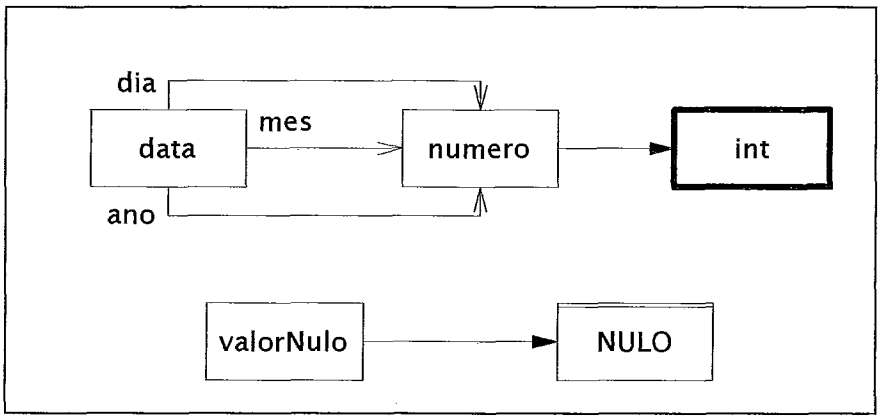


Fig 12 Exemplo de definições simples em VDM-GDL

A seguir, definimos os símbolos que representam os construtores de tipos, que podem ser vistos na figura 13.

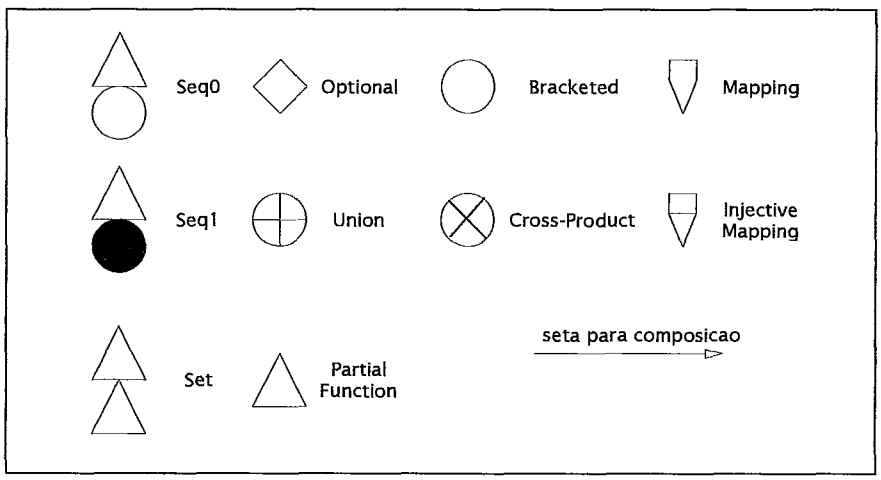


Fig 13 Símbolos para os construtores de tipos

Para exemplificar, apresentamos a especificação em VDM produzida a partir do modelo de entidades e relacionamentos do programa MEGA [Ans91], e o gráfico da figura 14, que são equivalentes. Nesse exemplo podemos notar, também, uma notação reduzida para uma declaração de um tipo igual a um tipo básico. O uso de notações reduzidas tem como finalidade diminuir o "peso" do gráfico.

```
types
    Vertex ::       ProducedBy : [ Kine-set],
                               : 3-D_Position
    Kine :: ProducedAt    : Vertex-set,
            Daughter      : [Kine-set],
            Decay : ℝ,
            P             : Four_Momentum
    3-D_Position  ::   x      : ℝ,
                       y      : ℝ,
                       z      : ℝ
    Four_Momentum ::   : 3-D_Position,
            Energy:  Energy
    Energy = ℝ
```

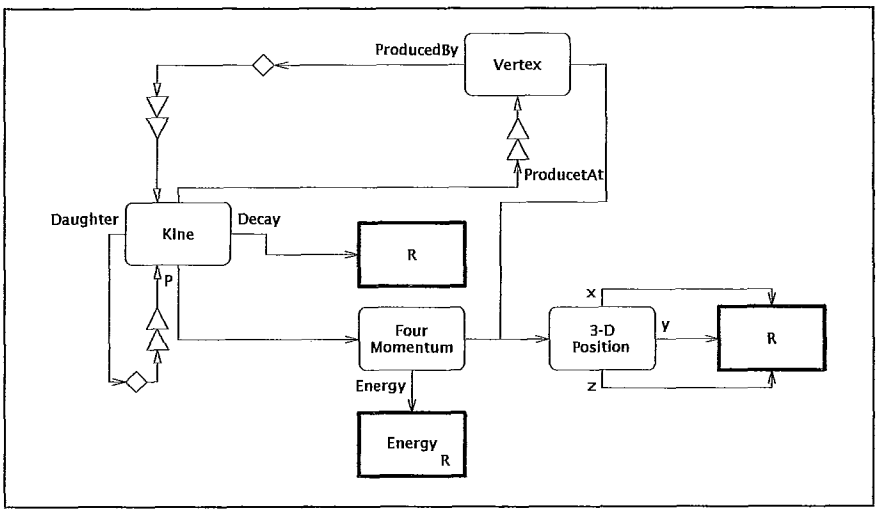


Fig 14 Exemplo de gráfico em VDM/GDL

IV.3.2. Utilizando VDM/GDL para representar sintaxes

Uma das características da abstração representacional de VMD/SL é servir para a definição da sintaxe abstrata de linguagens de programação. Como foi citado no segundo capítulo, usualmente dois tipos de representação são utilizados para representar sintaxe: BNF (e suas extensões) e diagramas de sintaxes. É possível traduzir os diagramas para BNF, porém eles não representam a mesma linguagem, não sendo essa tradução direta.

VDM/GDL permite que sejam construídas sintaxes utilizando a mesma semântica, apenas com representações diferentes. Como comprovação desta possibilidade de aplicação, o Apêndice II mostra um gráfico em VDM/GDL que

descreve exatamente a abstração representacional de VDM como está no padrão ISO. Para permitir a comparação, esse Apêndice apresenta também as páginas correspondentes no padrão.

IV.4. A linguagem CAB/DEL

CAB-DEL, acrônimo para CAB-*Description Language*, é uma linguagem de sintaxe simples destinada a descrever como um módulo[14] de programa, em uma linguagem de programação qualquer, implementa uma função ou operação em VDM.

A linguagem pode ser comparada a uma linguagem de interconexão de módulos (*module interconection language, MIL*) [PN86], pois permite descrever valores semânticos para as variáveis sintáticas, baseada em descrições em VDM, exigências de uso e, como extensão para suportar buscas, palavras chaves para os módulos.

Nas figuras 15 e 16, apresentamos dois exemplos em CAB-DEL, que descrevem funções em C e Fortran que implementam uma função de concatenação de cadeias de caracteres, como especificada pela seguinte definição em VDM:

```
types

string = seq of char

functions

-- append recebe duas strings e responde com a string
-- equivalente a concatenação da segunda ao final da primeira

append( string1 : string, string2 :  string) res : string

post res = string1 ^ string2;
```

[14] Utilizamos a palavra módulo para representar simultaneamente programas, funções e subrotinas.

```
function strcat implements append;
keywords are append, string, character;
language is C;
syntax is
  arguments are str1,str2,str3;
  type char* str1;
  type char* str2;
  type char* str3;
semantic is
 str1 = string1;
 str2 = string2;
 res = str3;
use is
 declare "#include <string.h>";
end.
```

Fig, 15 Exemplo de descrição em CAB/DEL para uma funcão em C.

```
subroutine STRCNT implements append;
keywords are append, string, character;
language is Fortran;
syntax is
  arguments are str1,str2,str3;
  type character*80 str1;
  type character*80 str2;
  type character*80 str3;
semantic is
        str1 = string1;
 str2 = string2;
 res = str3;
use is
 include "CHARACTER*80 STRCNT";
 library  "libstring.a";
 source "/users/code/fortran/strcnt.f";
end.
```

Fig. 16 Exemplo de uma descrição em CAB-DEL para uma subrotina em Fortran.

IV.4.1. Descrição da linguagem.

Cada descrição da linguagem divide-se em quatro partes: identificação, descrição sintática, descrição semântica e descrição do uso. O início da sintaxe para o programa em BNF [Tre85,ASU88] é o seguinte:

```
CABDEL ::=    identificação sintaxe
              semântica [uso] "end";
```

Fig. 17 Sintaxe inicial de CAB/DEL

IV.4.1.1.A indentificação

O objetivo desta parte é identificar o módulo, oferecendo informações que permitam ao usuário ter uma visão rápida do que o módulo implementa.

A sintaxe para a identificação é a seguinte

```
identificação   ::= cabeçalho palavras-chave
                        linguagem;
cabeçalho       ::= tipo nome "implements"
                        nomeVDM;
tipo            ::= "program" |
                        "function" |
                        "subroutine";
nome            ::= identificador
nomeVDM         ::= identificador
palavras-chave  ::= "keywords" verbo
                        lista-palavras;
verbo           ::= "is" | "are";
lista-palavras  ::= palavra |
                        palavras lista-palavras;
linguagem       ::= "language is"
                        nomeLinguagem;
nomeLinguagem   ::= "Fortran" | "C";
```

18. Definição em BNF para o bloco de identificação de CAB/DEL

A descrição se inicia dizendo que tipo de módulo está sendo descrito. São possíveis 3 palavras chaves: *program*, *function* e *subroutine*. A príncipio, programas em C só possuem *functions*, enquanto programas em Fortran admitem os três tipos.

A seguir são descritas as palavras chaves para esse módulo e a linguagem de programação em que ele foi implementado. A sintaxe atual de CAB/DEL suporta apenas Fortran e C.

IV.4.1.2.A sintaxe

O objetivo desta parte é descrever a sintaxe do módulo, isto é, que variáveis usa e quais os tipos destas variáveis (de acordo com a linguagem de programação). A especificação em BNF para a parte sintática é apresentada na figura 19.

```
sintaxe          ::= "syntax is"
                     lista-argumentos tipos;
lista-argumentos ::= "arguments are"
                          lista-vars;
lista-vars       ::= var | var lista-vars;
var              ::= identificador
tipos            ::= tipo | tipo tipos;
tipo             ::= "type" lista-id;
lista-id         ::= identificador |
                          identificador lista-id;
```

Fig 19. Definição em BNF da parte de descrição sintática de CAB/DEL.

IV.4.1.3.A semântica

A função desta parte é ligar a definição sintática do módulo com a sua especificação em VDM, dando assim uma semântica as variáveis do módulo.

Apenas dois tipos de declaração são permitidos: atribuir uma variável em VDM a uma variável sintática, indicando uma variável de entrada, ou atribuir uma variável sintática a uma variável presente na especificação VDM, indicando uma variável de saída.

A especificação em BNF para esta parte é:

```
semântica      ::= "semantic is" lista-atrib;
lista-atrib    ::= atrib | atrib lista-atrib;
atrib          ::= atrib-entrada |
                        atrib-saída;
atrib-entrada  ::= nomeVDM "=" nome;
atrib-saída    ::= nome "=" nomeVDM;
```

Fig. 20 Definição em BNF da descrição semântica em CAB/DEL.

IV.4.1.4.O uso

Esta parte indica características de utilização dos módulos. Na sintaxe atual permite a descrição de linhas que devem ser incluídas nos arquivos que utilizam o módulo, o nome da biblioteca a ser utilizada na link-edição e o arquivo que contém o código fonte, se este não estiver integrado no ambiente. A especificação em BNF é:

```
uso            ::= "use is" [include] [library]
                        [source];
include        ::= "include" lista-id;
library        ::= "library" lista-id;
source         ::= "source" lista-id;
```

Fig. 21 Definição da parte de descrição de uso de CAB/DEL em BNF

IV.4.2. Utilização das descrições em CAB/DEL

Além da utilidade planejada de fazer a ligação entre a especificação em VDM e a implementação em uma linguagem de programação habitual, CAB/DEL foi implementada ainda com o objetivo de auxiliar na reutilização de especificações e de módulos.

As palavras-chaves foram introduzidas na linguagem de forma a permitir que fossem selecionados módulos que pudessem ser identificados desta forma, sem que fosse preciso alterar as sintaxes de VDM ou da linguagem de programação (já fortemente formalizadas). Além disso, a existência de um nível de ligação entre a especificação e a implementação possibilita uma maior flexibilidade na reutilização de ambas.

IV.5. Introduzindo o Conceito de Domínio de Aplicação

A primeira extensão realizada na série de protótipos desenvolvidos introduziu o conceito de domínio de aplicação. Um domínio de aplicações é caracterizado, no sistema, por um modelo de dados único (uma abstração representacional única), devendo os sistemas que fazem parte deste domínio utilizar apenas os dados disponíveis no modelo.

Por ser uma linguagem plana, isto é, sem a possibilidade de definir módulos independentes, podemos discutir se VDM se presta para o desenvolvimento de domínios de aplicação. Por um lado o modelo plano facilita a relação do domínio de aplicação com as aplicações, pois as últimas não podem se "fechar"em módulos, devendo apresentar uma estrutura completamente aberta. Por outro lado, é impossível utilizar técnicas de modelagem como ocultação da informação.

Neste segundo protótipo, o incentivo a reutilização foi dado pela pré-existência do modelo de dados, considerado geral para a área de aplicação. VDM possuí construtores suficientes, como tipos opcionais e união de tipos, para permitir que modelos de dados generalizados fossem construídos, tanto pelo método analítico quanto pela síntese dos modelos de um conjunto de sistemas.

Por exemplo, supondo que duas aplicações possuissem os seguintes modelos para pessoa:

```
pessoa ::   nome : string
            endereço : string
string =    char*
```

e

```
pessoa ::   nome : char+
            idade : integer
```

um modelo geral poderia ser criado, trivialmente, com a seguinte definição:

```
pessoa = pessoa1 | pessoa2

pessoa1 ::  nome : string
            endereço : string
string =    char*

pessoa2 ::  nome : char+
            idade : integer
```

ou, com um trabalho mais adequado de síntese:

```
pessoa =    nome : string | char+
            endereço : [string]
            idade : [integer]
```

Isto, em VDM/GDL é representado pelo diagrama da figura 22.

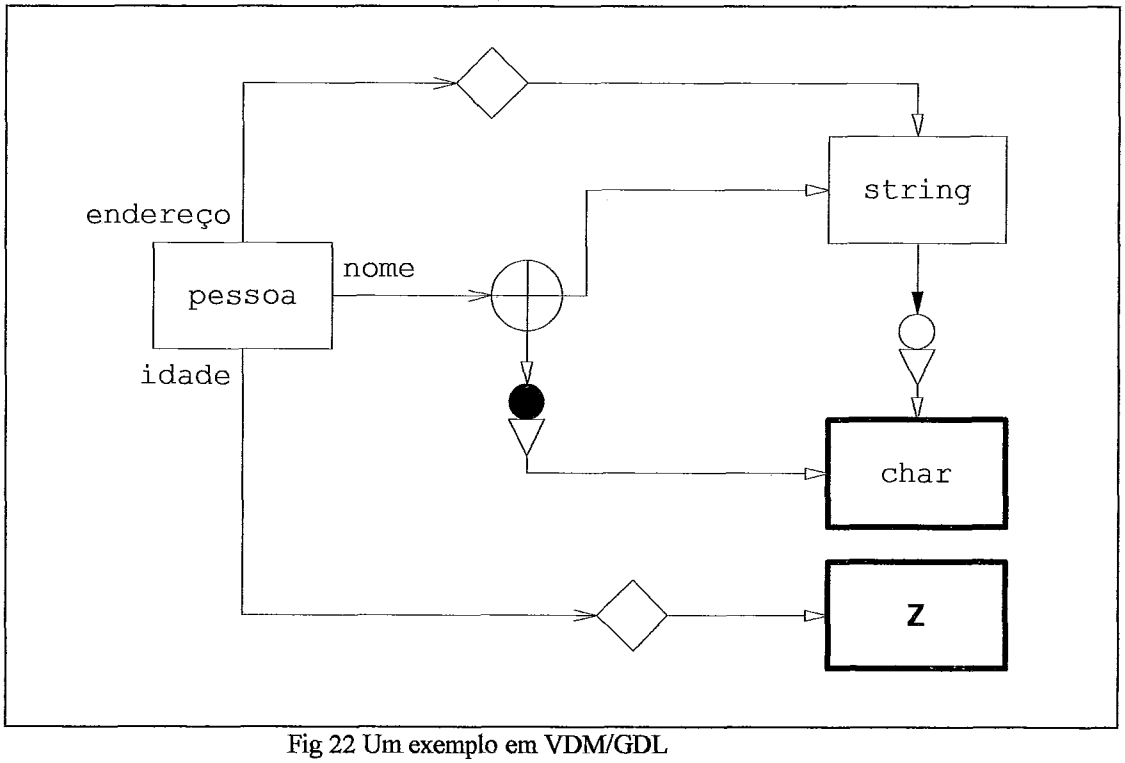


Fig 22 Um exemplo em VDM/GDL

IV.6. Unindo VDM e o Método Estruturado

Um terceiro protótipo foi construído tendo como objetivo unificar a linguagem VDM/SL (e VDM/GDL e CAB/DEL) com o método de desenvolvimento estruturado. Este trabalho foi baseado em [AI92] e assemelha-se a proposta, posteriormente publicada, de [Pla93].

Para explicar o método desenvolvido, devemos notar que as especificações estruturadas, na fase de Análise, são representadas basicamente por um Diagrama de Fluxo de Dados (DFD), diagramas de Entidade e Relacionamentos, e um Dicionário de Dados [You90]. Em VDM, a representação básica é por meio de tipos de dados e operações. Acreditamos que o mapeamento pode ser simples (na verdade, bem mais simples do que o proposto por Plat).

Primeiramente mapeamos cada fluxo de dados em um tipo de dados VDM. Os fluxos de dados dos DFD representam a entrada e saída de dados entre processos e repositórios, ou seja, a interação entre o modelo estático (de dados) e o modelo dinâmico (funcional). Tal mapeamento é equivalente a preencher o dicionário de dados com os fluxos.

O segundo passo é mapear os **depósitos de dados** do DFD em estados do sistema. Para isso, devemos admitir um método com vários estados. A alternativa existente é considerar que **todos** os arquivos constroem o estado, formando uma partição. O mapeamento provavelmente exige a construção de mais tipos de dados. Os tipos de dados dos depósitos de dados e os tipos de dados dos fluxos de dados são relacionados, pois cada dado presente em cada fluxo que entra ou sai de um depósito de dados deve estar presente nesse depósito.

O terceiro e último passo consiste em mapear os processos do DFD em processos VDM. Para isso, só utilizamos os processos de últimos nível, o que torna as descrições em VDM/SL equivalente às especificações de processo do método estruturado.

A partir do ponto em que possuimos cada especificação de processo, devemos trabalhar no sentido de torná-las menos abstratas, seguindo então o método VDM tradicional.

As figuras do Apêndice I demonstram a implementação deste protótipo, pois ele incorpora todas as versões anteriores.

Outra possibilidade, analisada mas não prototipada, é a de só utilizar VDM/SL na fase de projeto, como especificação dos módulos que aparecem no Gráfico de Estrutura.

Este método demonstrou características muito boas. Em primeiro lugar a utilização de VDM fornece um formalismo não existente no método estruturado. Permite também que seja controlado o nível de abstração das especificações dos processos, pela análise do nível de abstração das especificações em VDM. O modelo de dados de VDM é também mais forte que o modelo de entidades e relacionamentos, indicado em [You90] como parte do método estruturado, por apresentar um número maior de construtores de tipos e formalizar a existência de tipos básicos.

A principal diferença de nosso método para o método de Plat [Pla93] é que estamos preocupados em utilizar uma versão estruturada do método VDM, usando a linguagem VDM/SL para formalizar especificações antes fornecidas em linguagem natural ou em linguagens não formalizadas, enquanto Plat está preocupado em encontrar um significado (uma denotação) em VDM para as especificações estruturadas.

IV.7. CAB como um ponto de partida

Devido ao ciclo de vida de prototipações sucessivas do ambiente CAB, fomos capazes de realizar várias outras extensões menores ao sistema. Uma das extensões permite que o usuário ative uma aplicação WWW de dentro do sistema. Com essa extensão verificamos que o ambiente WWW era de grande qualidade para a implementação de sistemas de busca e navegação. Este foi o primeiro passo em direção ao sistema que apresentaremos no capítulo seguinte.

V. UM MODELO DE REUTILIZAÇÃO GLOBAL

"Um registro, se deve ser útil para a ciência, deve ser continuamente estendido, deve ser guardado e, sobre tudo, deve ser consultado"
Vannebar Bush, in *As We May Think*

Neste capítulo, inicialmente é feita uma análise crítica de certas características ausentes no sistema CAB, como acesso externo e um modelo conceitual para os objetos reutilizáveis.

Esta crítica gera um modelo conceitual para a reutilização de software, capaz de ser utilizado em qualquer ciclo de vida, por ser baseado no processo humano de solução de problemas, mais especificamente, na metodologia de investigação tecnológica e não sobre métodos específicos de desenvolvimento. O sistema Tabetá implementa esse modelo de forma complementar ao sistema CAB.

V.1. Fundamentos

No capítulo II, ao realizar a revisão da literatura sobre reutilização, chegamos a algumas conclusões que guiam esta tese.

A primeira delas é nossa definição de um bom processo de reutilização:

> **O que um bom processo de reutilização faz, na verdade, é estender e formalizar, de forma eficaz, a reutilização de experiências anteriores individuais, de uma pessoa ou comunidade, para outras pessoas ou comunidades.**

Durante o processo de desenvolvimento do sistema CAB, verificamos que ele não fornecia uma formalização suficientemente clara, pois seu modelo de componentes era plano, dividido apenas em especificação e implementação. O ambiente foi implementado em um sistema fechado, que apenas com certa dificuldade permitia consultas externas, evitando que mais de uma comunidade tivesse acesso aos componentes reutilizáveis. Logo, apesar de cumprir o papel de um ambiente de desenvolvimento com suporte a reutilização interna, CAB não fornece um ambiente de reutilização global, ou com um modelo conceitual suficientemente poderoso para responder as nossas expectativas.

A partir da afirmação que fazemos sobre o que é um bom processo de reutilização, desenvolvemos os requisitos básicos do método de reutilização proposto nesta tese. Estes requisitos estão divididos entre os que estendem o conhecimento e os que formalizam este conhecimento. Os requisitos servem de guia para um modelo de reutilização baseado na visão epistemológica da investigação tecnológica, que, por sua generalidade e fundamento filosófico, fornece simultaneamente formalidade ao processo e aplicabilidade em qualquer ciclo de vida, por não considerar métodos específicos, mas a própria natureza do método científico.

Outros requisitos, relacionados com a forma com que software é produzido, como a existência de problemas de patente e direitos autorais, são tratados como problemas contextuais, pois não são inerentes à atividade de produção de software *per si* mas sim às atividades econômicas em geral (o contexto em que o software é produzido).

V.1.1. Requisitos de extensão do conhecimento

Por requisitos de extensão do conhecimento, representamos aqueles que tem como objetivo alcançar o maior número possível de pessoas.

V.1.1.1.Larga extensão de área

É praticamente equivalente a dizer "atingir o maior número possível de pessoas". Este requisito busca alcançar a maior **comunidade** possível de usuários, tendo em vista a idéia de que quanto maior o grupo de usuários, maior será a base de componentes e maior será o número de vezes que cada componente será reutilizado. Para alcança-lo, o serviço deve estar aberto ao público, certamente disponível em rede e utilizando protocolos padronizados.

V.1.1.2.Ter interface ampla

Por "amplitude" queremos dizer que qualquer usuário possa encontrar, facilmente, uma forma de interagir com o sistema. Por interface ampla também indicamos que várias *linguagens* devem estar disponíveis ao usuário para consultar e compreender um componente, onde por **linguagem** entendemos um método qualquer

de troca de informação (linguagens de programação, linguagens visuais, linguagens iconicas, linguagens de consulta, etc.)

V.1.1.3.Ser integrado com outros sistemas

Com isso, permite-se que pessoas sem acesso ao sistema escolhido para implementação do sistema de reutilização possam, também, utilizá-lo por meio de tecnologias de integração.

V.1.2. Requisitos de formalização

Por requisitos de formalização, entendemos aqueles que se referem à forma como os componentes reutilizáveis estão descritos, formalizados e disponíveis para seleção.

V.1.2.1.Possuir um modelo conceitual simples e claro

A existência de um modelo conceitual permite que o usuário desenvolva um paradigma mental de utilização do sistema. Poucos conceitos devem estar presentes, sendo a simplicidade uma necessidade para tornar o modelo compreensível.

V.1.2.2.Possuir uma metodologia de reutilização ampla

Como o modelo conceitual deve ser simples, a metodologia de utilizar o modelo deve ser ampla, isto é, geral o suficiente para ser utilizada em qualquer modelo de ciclo de vida.

V.1.2.3.Possuir suporte a métodos formais e informais

A necessidade de métodos formais está clara na necessidade de definição **não ambígua** dos componentes, enquanto os métodos informais facilitam a compreensão.

V.1.3. Requisitos Contextuais

Basicamentes trazidos pela necessidade do sistema de incorporar a realidade econômica do desenvolvimento de software. Em uma sociedade idealizada poderíamos imaginar que todo o software seria gratuito (ou que **qualquer bem** seria gratuito), o que chega a ser verdade dentro de pequenas comunidades científicas.

Porém, a realidade implica que, apesar da existência de software gratuito, o desenvolvimento de software é, em geral, uma tarefa remunerada.

Assim, o sistema deve incorporar suporte a seleção e fornecimento de software pago.

V.2. Contratos de Desenvolvimento de Sistemas

Segundo a legislação brasileira, contratos de desenvolvimento de sistemas são contratos de prestação de serviços. No código civil, não sendo o contrato de prestação de serviço preciso quanto ao tipo de trabalho a ser efetuado, ficará o prestador obrigado a prestar **qualquer** serviço compatível com suas condições [Cer93]. A partir destas afirmações, podemos imaginar inúmeras questões que podem ser levantadas quando um contrato de desenvolvimento de sistemas permite múltiplas interpretações.

Além disso, devido a incapacidade das partes contratante e contratada de prever a complexidade de sistemas contratados, bem como decorrentes prazos e custos, e a capacidade técnica de desenvolvimento dos mesmo pela parte contratada, Cerqueira[Cer93] afirma que "As varas cíveis do Rio e São Paulo, principalmente, acham-se povoadas por demandas que possuem como objeto contratos de desenvolvimento de sistemas descumpridos pelas empresas prestadoras de serviços de desenvolvimento de software por encomenda".

O principal problema de relacionamento entre contratado e contratante é a ambigüidade dos contratos de desenvolvimento, que não proporciona às partes envolvidas uma idéia precisa do objeto do contrato, causando expectativas e previsões erradas de ambas as partes, com o conseqüente descontentamento final com o produto.

V.2.1. Contratos e métodos formais

Como visto no segundo capítulo, um método formal pode ser visto como uma forma de modelar o **conceito** de um produto de software, o que ele faz, sem definir seu **conteúdo**, a forma como é implementado[Wei91].

Métodos formais adequam-se, de forma exemplar, ao **modelo contratual de desenvolvimento de software** [CHJ86]. Nesse modelo, considera-se o ciclo de vida do desenvolvimento de software como uma seqüência de fases. Como as fronteiras entre cada fase são identificadas por produtos, cada uma das fases pode ser vista como um contrato. O uso de um método formal permite especificar de forma não ambígua o conceito do produto a ser implementado pelo contratado.

Apesar de ainda em seus primeiros passos, é cada vez maior a aceitação de métodos formais por parte de empresas. Os exemplos mais importantes são LOTOS e Estelle, padrões ISO que vem sendo utilizados na especificação de sistemas de telecomunicações e a recente obrigatoriedade da utilização de métodos formais na especificação de sistemas onde a segurança é crítica, determinada pelo Ministério da Defesa do Reino Unido.

O registro **explícito** de acordos feitos entre clientes, projetistas e programadores sobre o comportamento do sistema, fornecido pelos métodos formais, serve para retirar do contrato qualquer ambigüidade sobre o objetivo do mesmo.

Assim, a relação conceito/conteúdo pode ser utilizada para apoiar a contratação de software, mantendo-se no banco de dados apenas os conceitos do software, que assumem então o papel de um 'catálogo' especializado, permitindo que os clientes consultem este banco de dados, e oferecendo as implementações mediante contratos baseados nas especificações formais, fornecidas dentro do conceito.

V.3. Geração x Composição

Uma importante decisão é a escolha entre a implementaçãoo de um sistema gerativo ou compositivo. As experiências bem-sucedidas sobre sistemas gerativos são um grande ponto a favor dos sistemas gerativos [Nei+84,Gir92,Lei+94].

Entretanto, o sistema não tem a intenção de restringir o grupo de usuários com a escolha inicial de técnicas. Assim, supomos que o sistema básico de reutilização é um sistema compositivo, que pode ser controlado por um sistema gerativo.

A afirmação de que todo sistema gerativo é uma aplicação realizada sobre um sistema compositivo nos parece forte o suficiente para merecer uma justificativa. Os sistemas gerativos são **programas de computador**, certamente incapazes de realizar tarefas criativas. Assim as tarefas criativas são simuladas, por meio da inclusão do sistema de padrões que são modificados e **compostos** de forma a produzir o software especificado. É desta forma que funcionam os principais exemplos de sistemas gerativos[15], como Draco[Nei84].

Logo, ao fornecer um sistema básico, capaz de ser adaptado a múltiplos ciclos de vida, esse sistema deve ser certamente **compositivo**, ficando a cargo de extensões desenvolver aplicativos que permitam sua utilização na construção de sistemas gerativos.

V.4. Reutilização e Investigação Tecnológica

Construiremos nosso modelo de reutilização baseado na visão epistemológica da investigação tecnológica, como apresentada em [Bun87], tendo como objetivo poder incorporá-lo aos vários modelos de ciclo de vida de software [BD91]. Isso nos permitirá construir um modelo já integrado com a forma humana de resolver problemas, sem introduzir artificialidades desnecessárias. Além disso, escolhemos a versão de seis passos do método de investigação tecnológica, de forma a manter a simplicidade conceitual do modelo, enquanto uma segunda versão, de quinze passos [Bun87], introduz artificialidade em função de compreender sub-processos dos citados na versão menor.

A visão epistemológica da investigação científica já foi apresentada como uma sequência de atividades.

1. discernir o problema

---

[15] Na verdade, acreditamos que **todo** sistema gerativo deve seguir esse paradigma, mas admitimos a hipótese de que outras técnicas possam ser utilizadas, como por exemplo "gerar todas as soluções prováveis" e buscar a mais apropriada, técnica semelhante à prova computacional de um teorema. Mesmo assim, apesar da possível controvérsia desta afirmação, ainda vemos os padrões presentes e sendo compostos,

2. tratar de resolver o problema com a ajuda do conhecimento (teórico ou empírico) disponível;
3. se a tentativa anterior não for bem sucedida, elaborar hipóteses ou técnicas (ou, ainda, sistemas hipotéticos-dedutivos) capazes de resolver o problema;
4. obter uma solução (exata ou aproximada) do problema com o auxílio do novo instrumental conceitual ou material;
5. por a prova a solução (por exemplo, com ensaios de laboratório ou de campo);
6. efetuar as correções necessárias nas hipóteses ou técnicas, ou mesmo na formulação do problema original.

Porém, para deixar mais clara a interação entre as fases e sua relação com a reutilização, apresentamos o diagrama de fluxo de dados da figura 23, onde o dado de entrada "Problema" foi representado como uma nuvem de forma a demonstrar que não necessariamente ele estará bem definido.

"Discernir Problema" é uma atividade basicamente mental que deve resultar numa especificação do problema (e **não** da solução). "Selecionar Técnicas" é a atividade básica de reutilização, enquanto "Elaborar Técnicas" é a atividade básica de criação. Ambas podem ser vistas como atividades relacionadas à análise (busca da solução) e projeto (busca da implementação da solução). "Corrigir Hipotéses" pode ser vista como uma atividade mista, incluindo a alteração das técnicas existentes. É equivalente a modificar a solução anterior de acordo com o feedback fornecido pelo teste da solução. "Obter Solução" pode ser visto como uma atividade semelhante a codificação, enquanto "Testar Solução" representa as atividades de teste e verificação de uma solução.

O desenvolvedor do sistema realiza o ciclo investigação tecnológica várias vezes, de acordo com o ciclo de vida escolhido. Por exemplo, utilizando o ciclo de vida em cascata (Análise, Projeto, Implementação, Testes), cada fase apresenta um problema a ser resolvido com as técnicas disponíveis mais eventuais novas técnicas que devem ser desenvolvidas, como novos algoritmos.

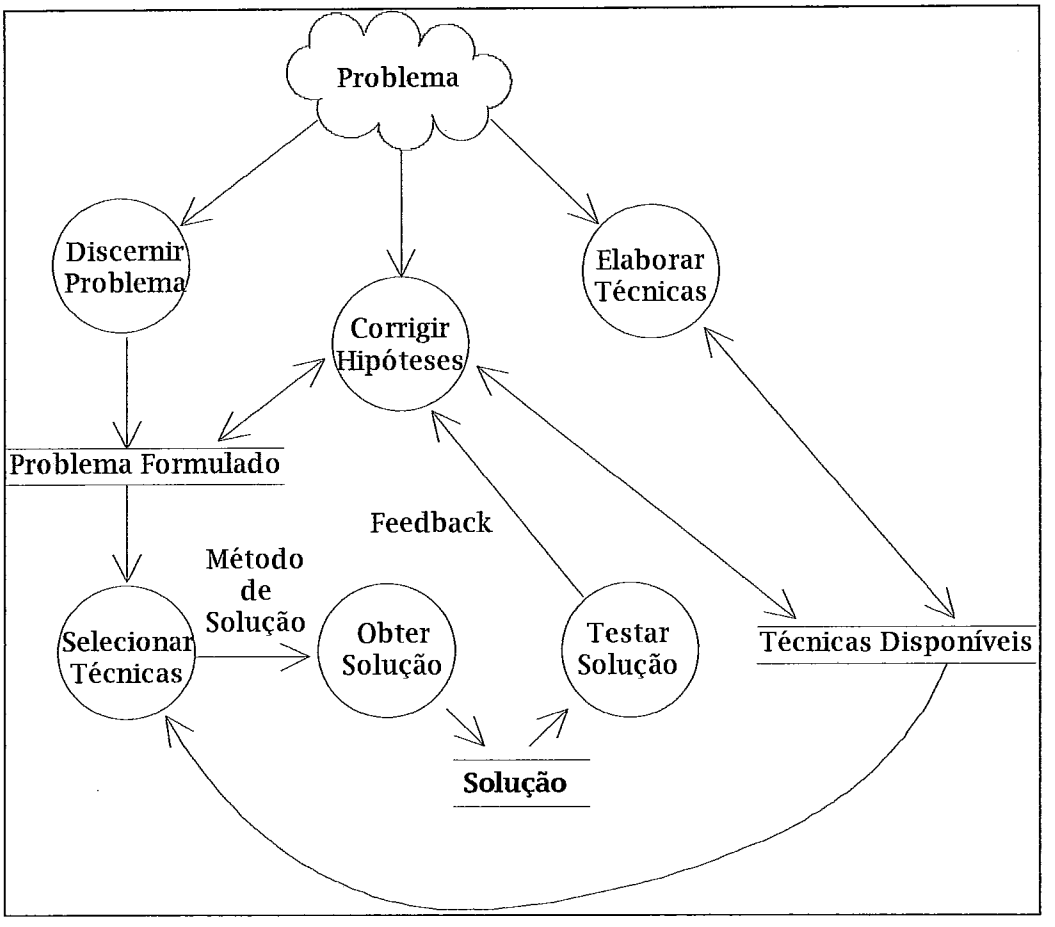


Fig 23 Método de investigação tecnológica, sem considerar ordem dos eventos.

V.5. Ciclo de vida apropriado

Ao contrário de modelos muito específicos [Nei84,Nei91,WHSX,Gir92,etc.] que exigem um determinado ciclo de vida para o desenvolvimento de software, ou de modelos tão gerais que preferem desconsiderar o assunto [Pri85,Mai91], um sistema geral de reutilização deve se encaixar em qualquer ciclo de vida. Para isso, visualizamos cada fase do ciclo de vida como um processo de investigação tecnológica distinto onde se tenta produzir uma implementação que resolva um problema dado. De acordo com o modelo 3C de reutilização dizemos que o objetivo deste processo é implementar (como conteúdo) o conceito fornecido pela fase anterior de acordo com algumas restrições fornecidas pelo contexto.

Assim, aplicar a reutilização ao processo de executar uma fase do ciclo de vida de um software equivale a buscar um conjunto de componentes de software cujos

conceitos mais se assemelhem ao conceito que se deseja implementar, e consequentemente, utilizar seus conteúdos para implementar o conteúdo desejado.

V.6. Abstrações

"Uma **abstração** é um processo mental utilizado quando selecionamos algumas características e propriedades de um conjunto de objetos e excluímos outras que não são consideradas relevantes" [BCN92]. Ao reutilizar software, fica claro que a metodologia de seleção deve utilizar abstrações, pois admitir o caso contrário significa dizer que o novo sistema é **igual** ao anterior. Admitindo que existam diferenças, devemos então analizar que tipos de abstrações são comuns, e verificar como são implementadas em nosso modelo.

Batini, Ceri, Navathe [BCN92] identificam três tipos de asbtrações: classificação, agregação e generalização. **Classificação** é o processo de definir um conceito como uma classe de objetos reais. **Agregração** é o processo de definir uma classe nova como um conjunto de outras classes que são seus componentes. **Generalização** é o processo de definir uma relação de sub-conjunto entre elementos de duas ou mais classes.

Nós estamos utilizando o conceito de classificação quando utilizamos conjuntos de palavras chaves para definir um elemento, pois o elemento passa a pertencer a classe de elementos definidos por aquela palavra. Um exemplo típico de classificação é "programas em C" ou "programas em Pascal".

Agregação representa a relação "é composto de". Exemplos típicos de agregação são as "compositions" e "records" em VDM. Fichas de dados pessoais utilizam agregação para representar uma "pessoa" como um conjunto de características (nome, idade, cor, etc.).

Generalização representa a abstração que fazemos quando dizemos: programas em C, programas em Pascal, programas em Fortran. Nesse caso, todas essas classes podem ser **generalizadas** pela palavra **programa**.

Ao buscar um componente reutilizável, espera-se que o desenvolvedor vá utilizar um desses métodos de abstração. Por exemplo, se utilizar palavras chaves, estará utilizando mecanismos de classificação.

Nosso modelo inclui suporte para os três mecanismos de abstração da seguinte forma:

V.6.1. Classificação

Os mecanismos de classificação utilizados são vários. A um nível inicial, os componentes são classificados implicitamente pelos valores de suas características agregadas, podendo essa classificação ser explicitada por meio de uma seleção. Assim, componentes são classificados entre si.

Em um nível mais abstrato, a relação conceito-conteúdo entre uma especificação e suas várias implementações apresenta uma classificação que encontramos internamente em um componente. Como componente é um conceito recursivo (um componente pode ser contexto, conteúdo ou conceito de outro componente), utilizamos o próprio conceito de componente para fornecer classificações.

V.6.2. Agregação

O mecanismo de agregação é implícito quando criamos componentes agregados, e implícita quando definimos um componente de software como um conjunto de 3 características. Cada uma dessas caractecterísticas é uma generalização de sub-características, que podem ser ou não conceitos agregados.

V.6.3. Generalização

Outro mecanismo utilizado em vários níveis. Conceito, contexto e conteúdo são generalizações que representam o papel que um componente tem para outro componente. Componente, por si só, é o conceito principal que pode ser visto como uma generalização de todos os outros.

Mais internamente, a relação conceito-conteúdo entre duas especificações pode ser, além de uma classificação, uma generalização. Isso porque especificações em VDM podem ser construídas de forma generalizadas

V.6.4. Utilizando abstrações

Em geral, o usuário não está ciente do tipo de abstração que está utilizando no sistema. Porém, ao encontrar um componente semelhante ao que deseja, ele pode procurar componentes mais ou menos gerais, ou componentes que pertençam a mesma classe, ou suas partes ou componentes do qual faça parte.

V.7. Um Modelo Conceitual da Reutilização

Desenvolvimento de software é uma tarefa realizada em fases. De acordo com as características dessas fases e de seu sequenciamento, existe um ciclo de vida. Cada fase pode ser vista como um ciclo de investigação tecnológica onde é apresentado um problema a ser resolvido. À especificação desse problema damos o nome de conceito. À sua solução damos o nome de conteúdo.

Em geral, existem outras restrições não inerentes ao problema, mas sim ao seu ambiente, as quais damos o nome de contexto. A composição do conceito, conteúdo e contexto damos o nome de **componente reutilizável de software**.

Um componente de software representa completa ou parcialmente uma fase do desenvolvimento de software. Para um sistema específico pode ser necessário o desenvolvimento de zero, um ou mais componentes para completar uma fase.

A figura 24 apresenta este modelo conceitual utilizando o modelo de entidade e relacionamento [BCN92].

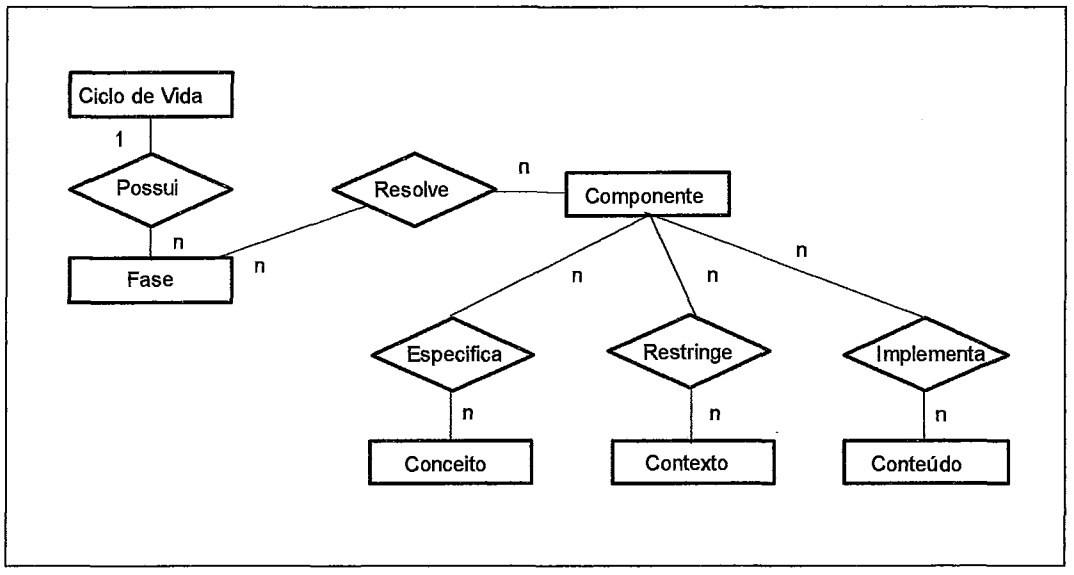


Fig 24 Um Modelo Conceitual para reutilização de software

Resta agora a pergunta: o que são Conceito, Contexto e Conteúdo? Tendo em vista serem partes do processo de desenvolvimento de software, nosso modelo propõe que são apenas papéis assumidos por outros componentes de software!

Assim, utilizando a recursividade, podemos utilizar componente de software como uma generalização de qualquer artefato decorrente do processo de desenvolvimento de software, e utilizá-los para construir outros componentes.

Utilizando também a noção de que Conceito, Conteúdo e Contextos são apenas papéis assumidos pelos componentes de software, possibilitamos que uma especificação intermediária, por exemplo, seja conceito para um certo componente e conteúdo para outro componente.

V.7.1. Exemplo

Utilizaremos o exemplo do capítulo III para exemplificar Conceito e Contexto como papéis de um componente. A função **push** foi definida em vários níveis de abstração. O nível intermediário é simultaneamente conceito para o nível mais próximo da implementação e conteúdo para o nível de abstração mais alto. Isto pode ser visto na figura 25.

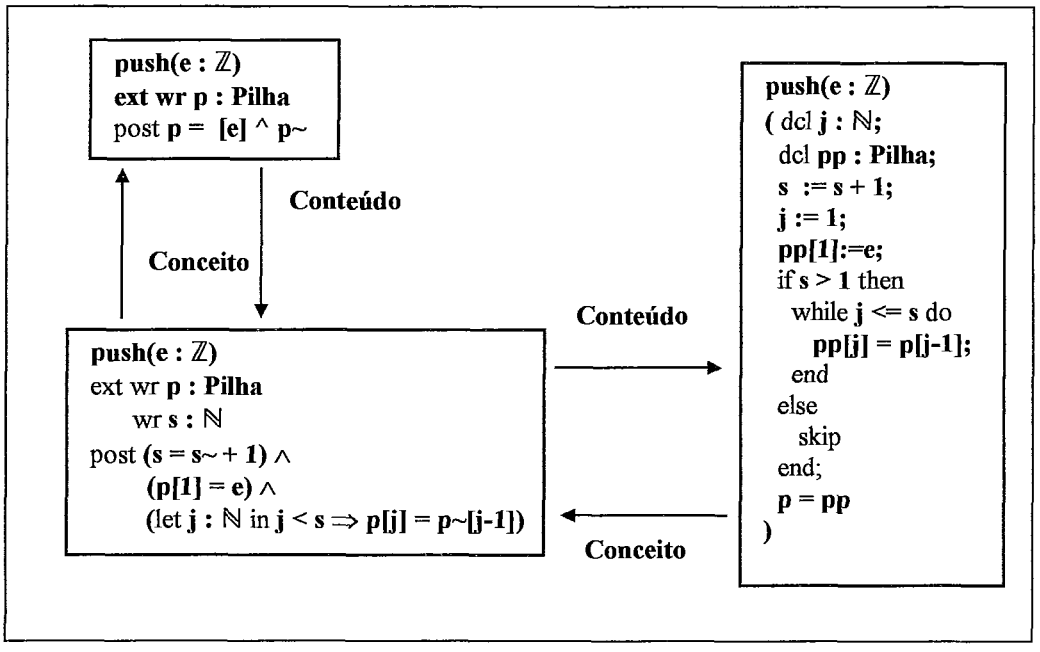


Fig. 25 Conceito e Conteúdo podem ser papéis assumidos pelo mesmo componente.

V.8. Um método de reutilização

Dado o nosso modelo conceitual para introduzir a reutilização dentro dos ciclos de vida nos parece que fica claro qual será o método de reutilização.

Para cada fase o desenvolvedor recebe um problema, já formalizado ou não. O primeiro passo é a especificação do problema de acordo com as normas daquela fase, produzindo o que denominanos **conceito**, e que no método de investigação tecnológica é conhecido como **especificação do problema**.

É o conceito que o desenvolvedor utilizará para realizar a seleção e elaboração de técnicas para a solução. A próxima atividade é a composição das técnicas selecionadas, que podemos descrever como "implementar solução". Baseado no método de investigação tecnológica assuminos que pode existir, dentro de uma fase, um ciclo para correção de hipóteses. A figura 26 mostra a sequência de ações correspondentes ao método de investigação tecnológica já com o nome das ações correspondentes ao nosso modelo de reutilização.

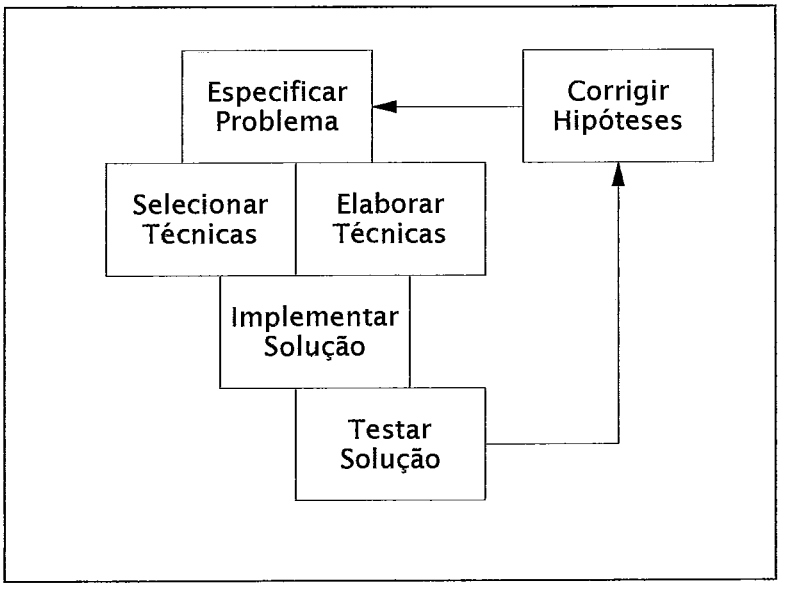


Fig 26 Representação do método de reutilização como uma sequencia de ações

As cinco atividades do processo de reutilização estão incorporadas nesse método da seguinte forma: selecionar é incorporado diretamente, como a fase de busca de componente. Ao elaborar componentes, como vimos anteriormente, eles devem ser somados

Adaptar e integrar são realizadas durante implementar solução. Preparar está incluído em elaborar técnicas, enquanto avaliar está em corrigir hipóteses.

Devemos notar, porém, que este modelo apresenta apenas uma sequência para a tarefa, e não seu fluxo de dados, e que não indica como a reutilização é gerenciada,

A tarefa de cumprir uma fase do desenvolvimento de software equivale a criar o componente especifico daquela fase. A metodologia científica permite dois métodos básicos: reutilizar ou criar. Para reutilizar é preciso encontrar o que reutilizar e incorporá-lo ao seu sistema.

V.9. Levando a reutilização à comunidade

Levando em conta que nosso modelo conceitual exige larga extensão de área, devemos ser capazes de fornecer a comunidade os componentes a serem reutilizados. Para isso consideramos que os componentes devem ser "servidos" à comunidade por um sistema que permita consulta, navegação e recuperação.

Segundo Arango [Ara93], o padrão de fato da reutilização é o de utilizar sistemas altamente informais na Internet. Acreditamos que a melhor opção de implementação do nosso sistema é a própria Internet, aproveitando o impulso já existente na utilização deste meio.

V.10.Um Modelo de Objetos

Devido a característica dos componentes assumirem papéis, projetamos um modelo orientado a objetos, utilizando a metodologia OOA [CY91], para os componentes, apresentado na figura 27.

No modelo, agregação, classificação e um objeto que apresente as características do modelo C3 são especializações de um component comum, enquanto generalizações são características de qualquer componente.

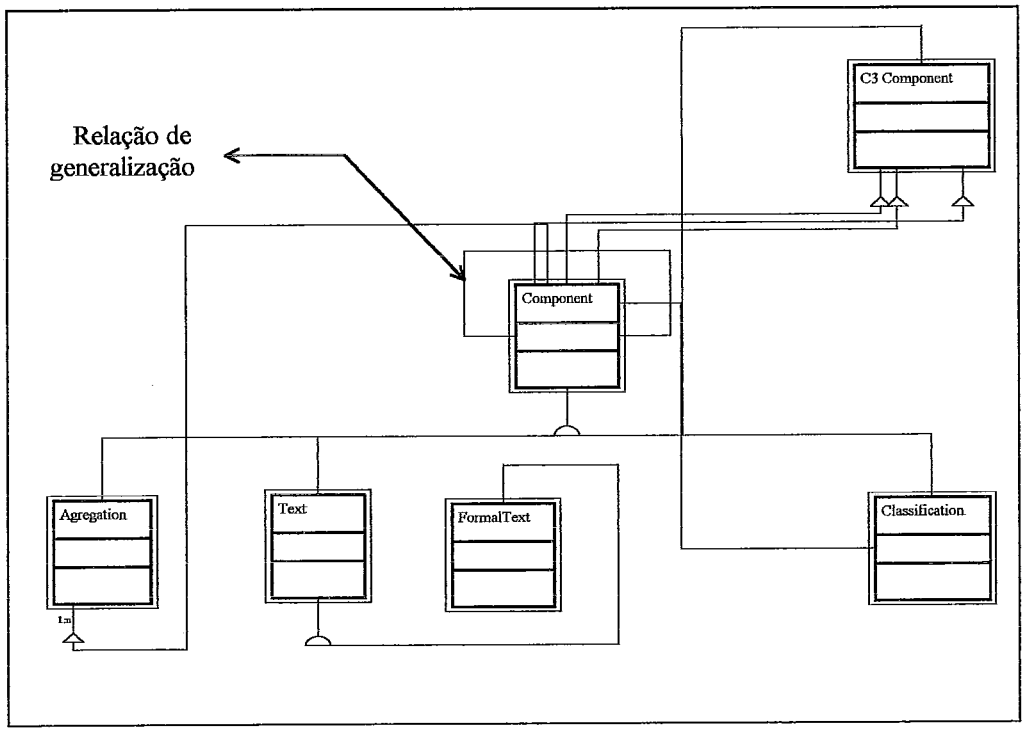


Fig. 27 Um modelo de objetos para componentes de software.

V.11.Tabetá

Tabetá é a palavra em Guarani que representa um conjunto de Tabas, sendo um conceito para uma tribo ou grupo indígena. Tabetá também é o nome do primeiro

protótipo de um sistema de reutilização que visa fornecer componentes de softwares disponíveis em bases de componentes distribuídas pela rede Internet. A idéia é que cada comunidade local (tribo) pode se reunir a outras comunidades locais e reutilizar suas experiências.

O sistema apresenta características complementares ao sistema CAB. Idealmente, deveria ser uma extensão que complementasse as características do ambiente de desenvolvimento com suporte a reutilização e fornecimento de componentes, porém CAB foi desenvolvido no CERN, utilizando os sistemas lá disponíveis, enquanto Tabetá foi desenvolvido na UFRJ, onde as ferramentas de software disponíveis são outras. CAB e Tabetá poderiam ser implementados em um sistema único como na figura

V.12. Descrição do Sistema

O sistema Tabetá é implementado por um ou mais servidores de objetos, capazes de formar entre si uma sub-rede, que podem ser acessados por qualquer cliente WWW, de acordo com protocolos padronizados. Isso significa que o usuário pode estar em qualquer tipo de computador que acesse a Internet, contanto que possua um programa que compreenda os protocolos definidos no sistema WWW. Para verificar a generalidade deste modelo de implementação, pode ser citada a disponibilidade de clientes WWW para máquinas Unix simples, Unix com X-Windows, Unix com NeXTStep (Display Postscript), IBM-PC DOS, IBM-PC e Microsoft Windows e Apple Macintoshs.

Cada objeto servido é um componente de software ou uma informação sobre componentes de software. Ao entrar no sistema o usuário pode escolher um entre vários métodos de busca, desde a simples navegação linear entre componentes até buscas baseadas em especificações formais. O sistema também mantém informação sobre outras bases de componentes, podendo indicar ao usuário onde encontrar outros componentes que não os mantidos por eles. Esse segundo tipo de informação funciona como uma espécie de catálogo de bases de informação, ou diretório.

V.13.Um pequeno glossário

Antes de iniciar a descrição do sistema, faremos um pequeno glossário de palavras e siglas que aparecerão muitas vezes neste capítulo:

1. CERN: *European Center for Nuclear Research*, laboratório onde foi criado o sistema WWW.
2. FTP: *File Transfer Protocol*, um protocolo de transferência de arquivos na Internet;
3. HTML: *Hypertext Mark-up Language*, uma linguagem, definida a partir do padrão SGML, capaz de definir como uma página deve ser escrita e suportando características de hipertexto, como ligações e formulários;
4. HTTP (1.0): *Hypertext Transfer Protocol*, o protocolo de comunicação cliente-servidor do sistema WWW;
5. httpd: um servidor WWW produzido pelo NCSA;
6. Mosaic: um cliente de leitura WWW capaz de mostrar figuras e sons, caracterizando um sistema de *hypermedia*, produzido pelo NCSA;
7. NCSA: *National Center for Supercomputing and Applications*, um laboratório americano de computação;
8. URL: *Uniform Resource Locator*, um padrão de sintaxe universal que pode ser utilizado para referenciar objetos disponíveis utilizando quaisquer dos protocolos existentes na Internet. As vezes citado como *Universal Resource Locator*. Cada URL deve corresponder a um objeto específico que pode ser acessado pela Internet;
9. WWW: *World Wide Web*, um sistema de hipertexto funcionando dentro do protocolo TCP/IP, formado não por aplicaçõres específicas mas por um conjunto de protocolos que definem uma forma padrão de comunicação entre clientes e servidores (HTTP e HTML);

V.14.Propósito do Sistema

O objetivo do sistema Tabetá **é fornecer, para reutilização, à maior comunidade de usuários possível componentes de software de características variadas**. O sistema suporta várias formas de visualizar um componente, desde sua especificação, formal ou informal, até sua implementação. Em especial, objetos

incluídos no sistema Tabetá possuem as visões do modelo de referência 3C de reutilização: Conceito, Contexto e Conteúdo.

O sistema é acessado por meio de um conjunto de protocolos conhecido com o nome genérico de "protocolos WWW". Tais protocolos definem um **sistema de hipertexto com suporte a consultas**. Ao contrário de sistemas tradicionais, o sistema Tabetá não tem uma interface com o usuário própria, mas utiliza qualquer interface compatível com o protocolo WWW, que se tornam clientes do servidor de componentes. Outras aplicações do sistema Tabetá, como hmake e vdmpp, possuem interfaces de usuário simples, características das operações específicas às quais foram destinados.

Os usuários do sistema Tabetá são todos os desenvolvedores de software com acesso a rede Internet e a, pelo menos, um cliente de leitura WWW. Usuários que desejem estabelecer uma base própria devem possuir, ainda, o software de banco de dados O2, que foi utilizado para implementação do sistema. Apenas o servidor Tabetá faz exigências de sistema operacional (Unix) e suporte a banco de dados (O2), não sendo feitas exigências aos clientes.

V.15. Descrição Geral

V.15.1. Interface do sistema Tabetá com outros sistemas

O centro do sistema Tabetá é um servidor de componentes capaz de entender o protocolo HTTP 1.0 e que responde com textos e imagens formatados segundo o protocolo HTML.Ambos os protocolos estão em fase final de padronização pela Internet. Isto significa que o sistema é aberto, podendo ser acessado por qualquer software cliente capaz de se comunicar nesses protocolos.

As ferramentas auxiliares funcionam como software independentes, algumas com características de programas clientes de algum protocolo cliente servidor. A ferramenta hmake, por exemplo, é um cliente ftp.

V.15.2.Funções do Sistema Tabetá

A principal função do sistema é suportar, por meio do servidor de componentes, a busca e recuperação de componentes de software. Em especial, é feito o suporte a buscas baseadas na especificação formal dos componentes em VDM.

Outros aplicativos completam a funcionalidade do sistema. "vdmpp" é um *pretty-printer* para VDM, "hmake" é uma extensão do programa "make", do sistema operacional Unix, capaz de manter um produto de software em dia com componentes de software disponíveis em outros computadores, utilizando os protocolos ftp e http.

O DFD, a seguir, representa o contexto do sistema Tabetá, cujo objetivo é fornecer componentes de software baseado em informações sobre estes componentes que representam especificações parciais.

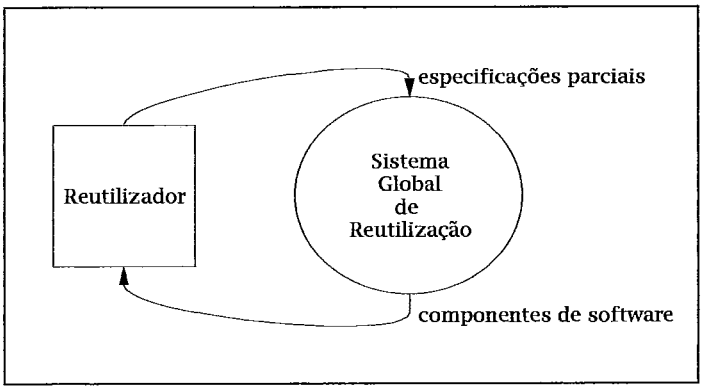


Fig. 28 DFD de contexto do sistema Tabetá

V.16.Exemplo do Sistema

Para utilizar o sistema Tabetá o usuário deve possuir um cliente WWW ou dar um "login" em um servidor público. Nos nossos exemplos estaremos utilizando o servidor Mosaic, desenvolvido pelo NCSA. Para iniciar uma seção o usuário deve navegar para a URL **http://guarani.cos.ufrj.br:8000/GRS/GRS.html.** Quando o servidor responder o usuário verá a seguinte tela[16]:

[16] Apresentaremos versões em português das telas, mas o sistema, para permitir uma utilização global, também implementa telas em inglês.

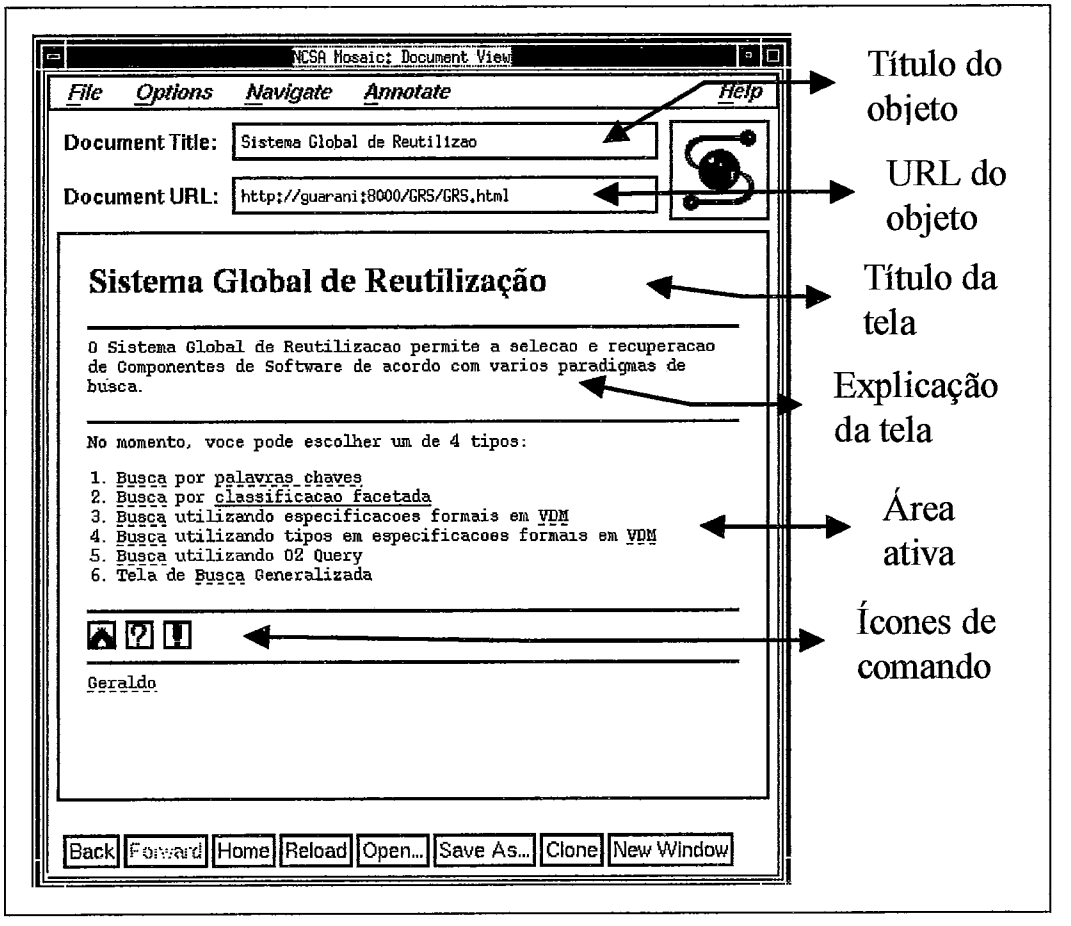


Fig. 29 A tela de entrada do sistema Tabetá

Na tela da figura 29 estão apresentados os principais comandos do sistema. Seis tipos de busca estão disponíveis, sendo que o último tipo engloba todos os outros tipos. O sistema é projetado para que a busca seja feita normalmente pela forma generalizada, porém alguns usuários podem preferir utilizar um tipo especial de busca apenas.

Desta tela o usuário pode navegar para várias outras, seguindo as ligações de hipertexto, que estão sublinhados na figura 29, e que, no vídeo, aparecem também com uma distinção de cor. As sequências possíveis de telas podem ser encontradas na figura 30

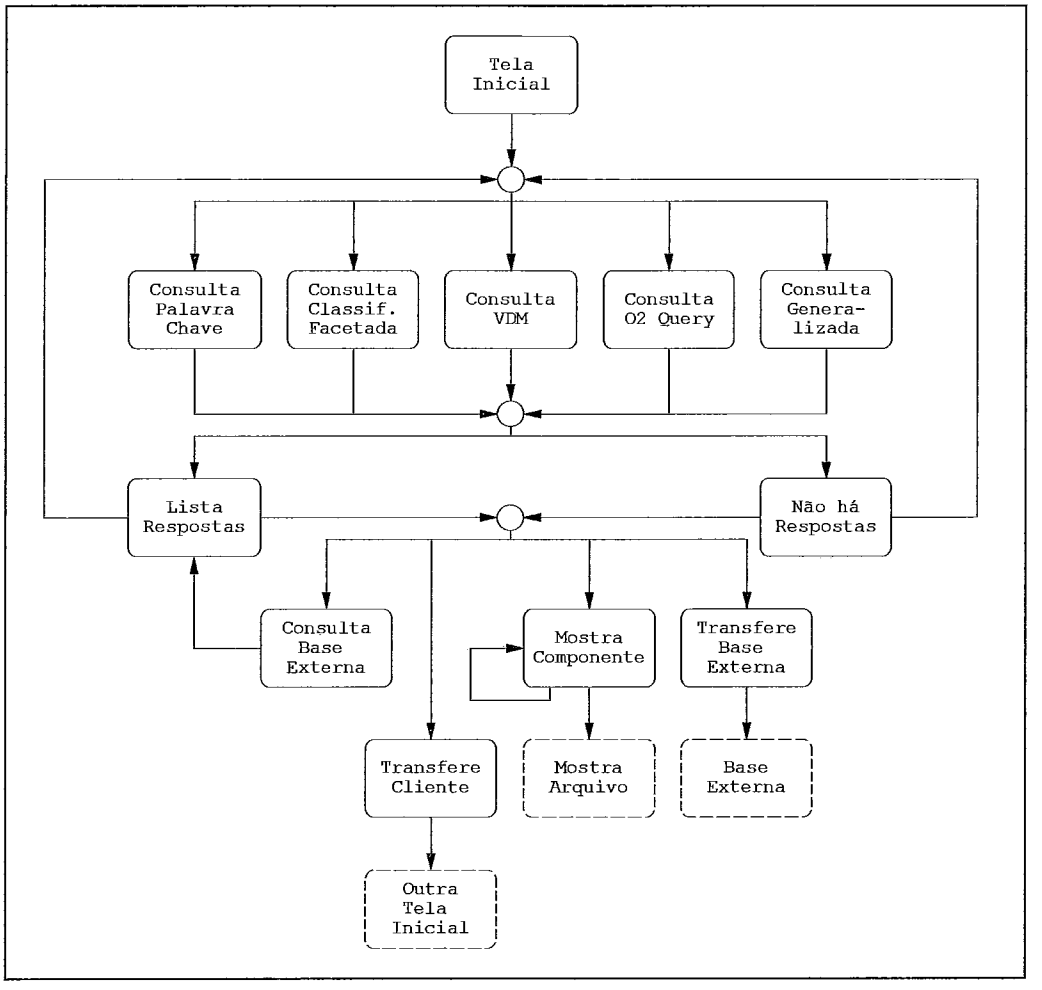


Fig. 30 Sequências possíveis dos tipos principais de tela, excluindo telas de ajuda e exemplo. Telas indicadas por um retângulo tracejado representam navegações para fora do sistema.

A figura 31 apresenta a tela de busca generalizada, onde o usuário pode entrar com dois tipos de dados, uma descrição completa ou uma URL indicando uma descrição completa que será recuperada pelo cliente no momento da busca. Também estão disponíveis, para o usuário, várias opções de busca, que podem ser ligadas ou desligadas de acordo com sua vontade, de modo a conseguir buscas mais completas ou mais rápidas.

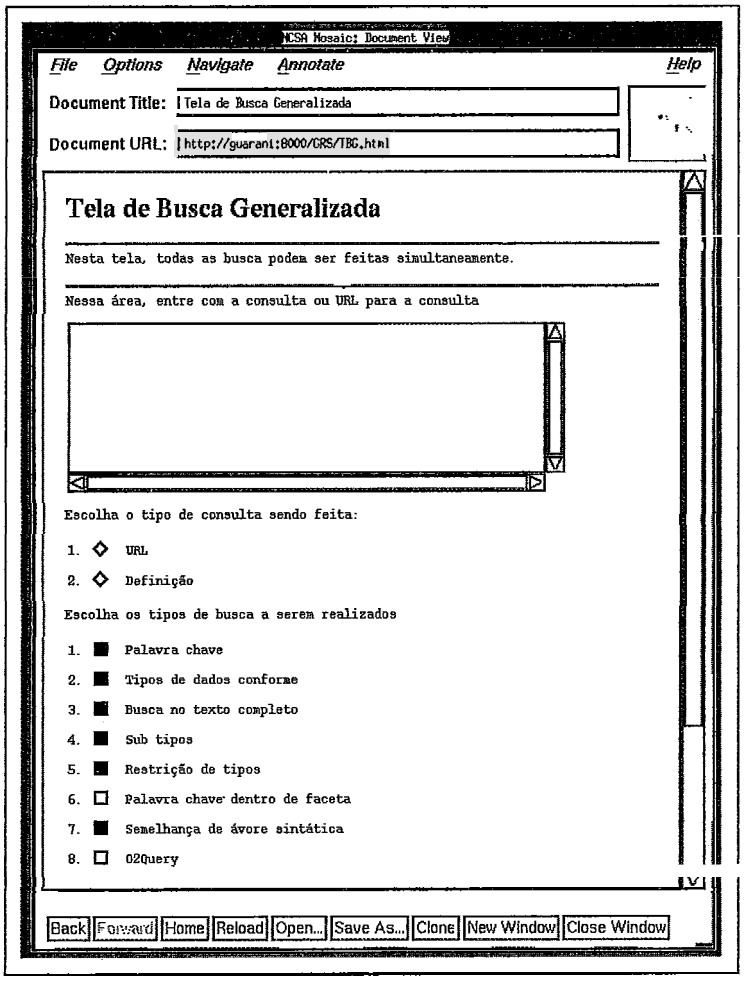


Fig. 31 Tela de busca generalizada

Caso uma busca resulte em uma lista de componentes reutilizáveis, o usuário verá a tela da figura 32. Nesta figura mostramos apenas a área ativa.

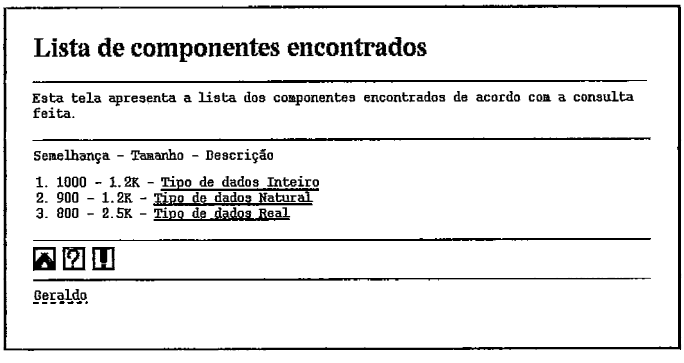


Fig. 32 Tela listando os componentes encontrados

As outras telas do sistema podem ser vistas no apêndice III.

V.17.Arquitetura do Sistema

O sistema segue uma arquitetura cliente-servidor. O objetivo do servidor é manter um banco de componentes com múltiplas formas de consulta e acesso, capaz de consultar outros bancos de componentes, sejam eles integrados ou não.

Clientes e servidores comunicam-se pelo protocolo HTTP 1.0. Assim, os clientes WWW podem ser utilizados sem modificações. Servidores podem se comunicar entre si com uma extensão do protocolo HTTP 1.0, que chamaremos de HTTPX.

Os servidores, pertencentes ao sistema, formam uma rede própria de comunicação. Isto tem como objetivo suportar o fato de que um único local não pode manter informação sobre toda e qualquer espécie de componentes de software. Assim, cada servidor pode manter dois tipos de informação básica: informações sobre componentes e informações sobre outros servidores. A segunda, entre outras coisas, indica de forma generalizada o tipo de componente que pode ser encontrado em cada servidor e pode ser vista como um "catálogo" de servidores.

Além disso existem na Internet vários outros sistemas que fornecem componentes de software. Esses sistemas também podem ser cadastrados, sendo possível indicar formas de consulta que eles aceitam e maneiras de traduzir consultas no nosso protocolo para o protocolo deles.

Quando um cliente se conecta a um servidor esse servidor passa a ser conhecido, para aquele cliente, como servidor central ou servidor principal. Isso significa que o cliente, *a priori*, deseja acessar a rede de servidores de componentes por meio deste servidor específico.

Os outros servidores participantes da rede de servidores assumem o papel de servidores remotos. De acordo com a estratégia adotada pelo cliente o Servidor Principal pode ou não acessar os Servidores Remotos. Os servidores estão sempre cadastrados entre si, na forma de componentes, podendo o servidor principal encontrar, durante uma busca, servidores que respondam melhor a uma certa consulta.

Outros bancos de dados sobre componentes ou de componentes também podem ser acessados. Para isso são cadastrados com regras que permitem a tradução da consulta.

Se o desenvolvedor possuir um servidor local, ele é capaz de criar modificações aos componentes integradas ao sistemas, utilizando a capacidade de comentário do sistema WWW. Comentários são controlados por "Servidores de Comentário", instâncias especiais de servidores WWW, e são textos em HTML, que podem então conter referências ao objeto que é fornecido pelo usuário. Outra opção é o usuário fornecer informações ao responsável pelo servidor.

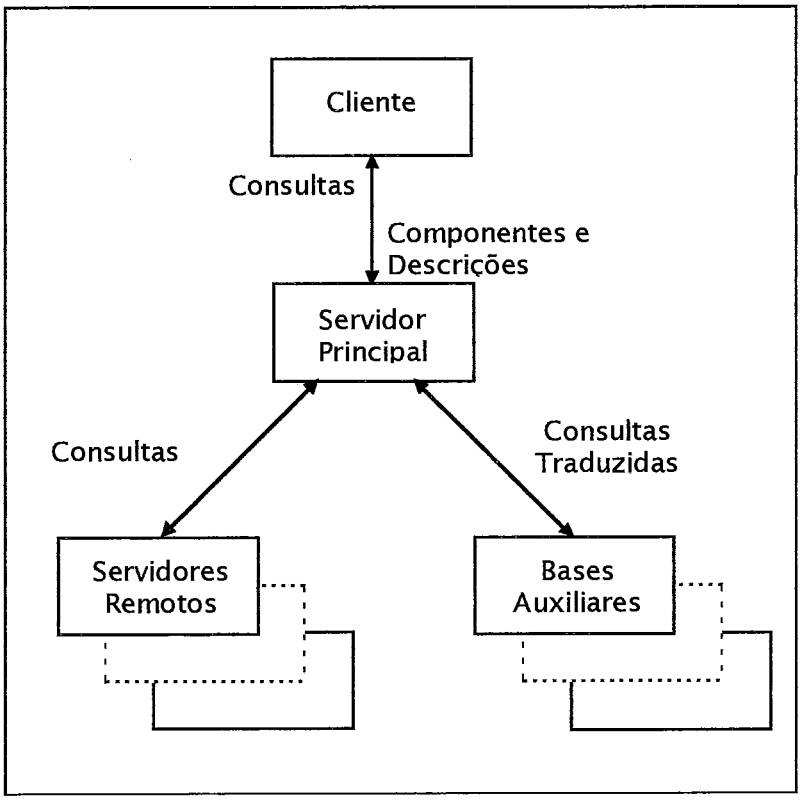


Fig. 33 Os processos participantes do sistemas podem assumir vários papéis diferentes.

A arquitetura do sistema é indicada pela figura 34. Nela vemos a representação de três cópias do sistema, uma assumindo o papel de servidor central, outra assumindo o papel de servidor remoto e uma terceira assumindo o papel de servidor local, que, para o servidor central é apenas outro servidor remoto, mas que permite que o usuário faça modificações internas, por meio do próprio sistema ou por meio de editores, por pertencer a esse usuário.

Os servidores se comunicam por meio do protocolo HTTP. Cada servidor pode ser visto em três estágios: um servidor de rede, um banco de dados (implementado em O2) e os arquivos do sistema operacional (Unix). A figura apresenta três tipos de clientes: um cliente comum de leitura, um cliente de escrita e um cliente hmake.

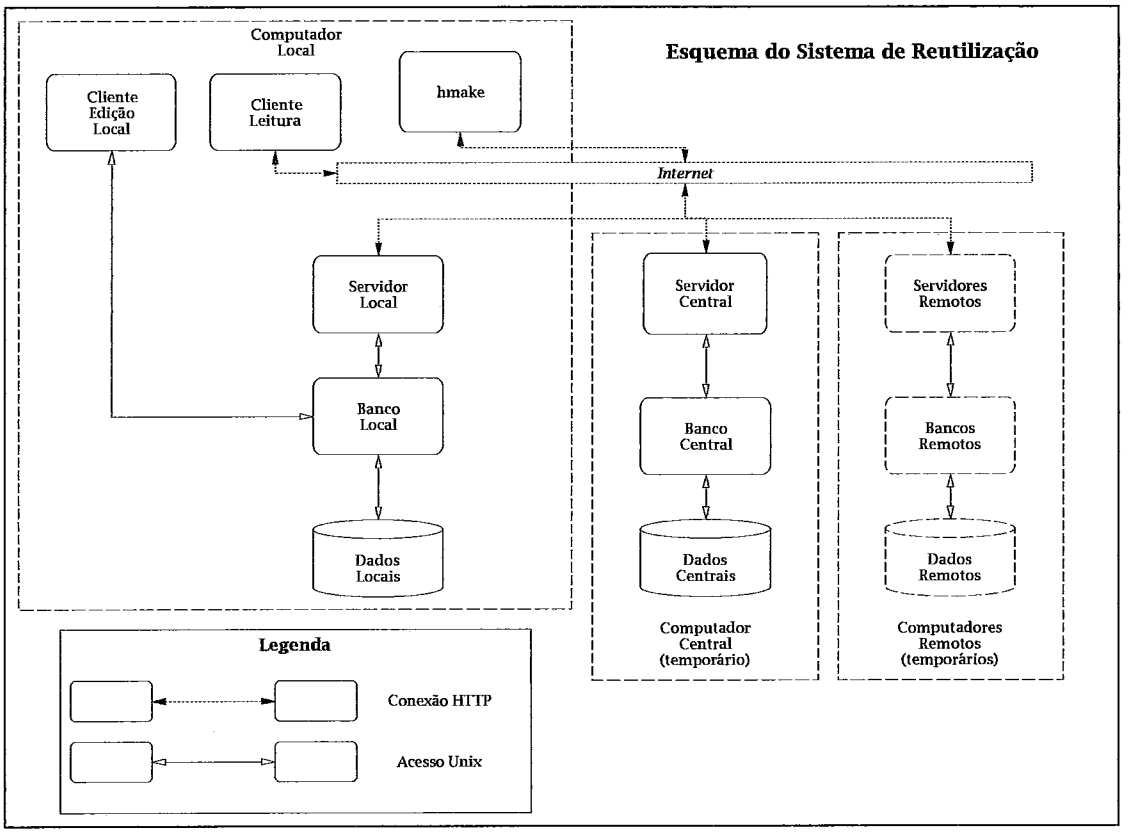


Fig 34 Arquitetura do Sistema Tabetá

V.17.1.1.O Modelo de Objetos Completo

Certamente, a implementação do sistema tem que levar em conta vários detalhes relativos a padrões de comunicação, sistemas operacionais e paradigma do banco de dados utilizado. O apêndice IV apresenta o modelo de objetos completo do banco de dados implementado.

V.18.Servidor de Componentes

O servidor de componentes, cuja arquitetura em três estágios pode ser vista na figura 34, foi implementado da seguinte forma:

1. o servidor de rede, a partir das bibliotecas WWW fornecidas pelo CERN e do servidor httpd fornecido pelo NCSA,
2. o banco de dados, utilizando o banco de dados orientado a objetos O2,
3. arquivos externos, apontados a partir do banco de dados por meio de URLs.

O servidor de rede funciona como um *daemon*, chamado dhhtpd, que fica esperando, em uma porta pré-determinada, que seja invocado por um cliente. Com base no URL o servidor de rede é capaz de transferir, imediatamente, o objeto solicitado ou, se necessário, ativar o banco de dados.

Neste ponto fazemos a observação que o banco de dados utilizado na implementação **também** funciona com a tecnologia cliente servidor. Isso significa que o servidor de redes pode ser obrigado, algumas vezes, a ativar o servidor O2, além do cliente, que sempre é necessário.

Ativado o banco de dados, cujo aplicativo se chama 'objserver', o servidor de redes transfere a parte de consulta da URL para o primeiro. Cabe então ao 'objserver' realizar a busca e responder com o(s) objeto(s) encontrados. Caso o usuário deseje, o 'objserver' pode pedir ao servidor de redes que repita a mesma pergunta para outros servidores de hipertexto distribuídos pela rede.

O sistema de busca do servidor incorpora múltiplos métodos. Cada um desses métodos será analisado mais tarde neste capítulo.

V.19.O programa hmake

O programa 'hmake' é uma extensão do programa 'make' com suporte a URLs. 'make' é um programa que permite a manutenção de um ou mais programas ou arquivos atualizados com as modificações em outros programas ou arquivos. O usuário do programa 'make' fornece uma descrição formada por listas de dependências entre arquivos e instruções para serem realizadas caso o arquivo dependente seja mais antigo de que o antigo do qual ele depende, indicando que não está em dia com as modificações realizadas no último.

'hmake' permite que ao invés de arquivos sejam indicados URLs. Assim, por exemplo, pode se construir um software baseado em componentes distribuídos pela Internet, cabendo ao programa testar as datas e recuperar os objetos correspondentes às URLs, se necessário.

Tecnicamente são dois os problemas principais na implementação de hmake: a utilização das URLs, pois podem indicar qualquer protocolo, e a determinação da data e a hora correta da última atualização de um determinado objeto, dado muitas vezes não fornecidos pelos protocolos.

A versão atual é capaz de utilizar os protocolos ftp e http. URLs indicadas por ftp podem ter suas datas facilmente checadas segundo o protocolo, já corrigidos os problemas de fuso horário. URLs indicadas por http não admitem, ainda, a verificação de datas e horários, porém desenvolvemos essa capacidade em nosso servidor 'dhttpd'.

V.20.Os métodos de busca

A principal característica da metodologia de busca do sistema Tabetá é não possuir uma metodologia específica, mas sim um modelo ao qual os métodos de busca são adaptados. Esse método pode ser representado pela seguinte instrução em O2SQL, uma extensão da linguagem padrão SQL capaz de suportar objetos:

```
SELECT SIMILARITY,SIZE,NAME,DESCRIPTION,URL
FROM OBJECT IN COMPONENTS
WHERE SIMILAR( <especificação parcial>) GREATER THAN <limite>;
```

Isto significa que, a cada objeto, é enviada uma mensagem de medida de similaridade com uma especificação, cabendo ao objeto determinar qual é o valor desta similaridade.

Assim, novos tipos de objetos podem ser incorporados ao sistema, admitindo novos tipos de comparações, sem que seja necessário alterar o esquema de busca.

Cada componente, a princípio, tem mecanismos para realizar três tipos de buscas: buscas léxicas, buscas sintáticas e buscas semânticas. As buscas léxicas

tratam o texto do componente como unidades separadas. As buscas sintáticas tratam o texto dos componentes como estruturas sintáticas e as buscas semânticas tratam do significado do componente.

A seguir descrevemos as buscas implementadas no sistema.

V.20.1.Buscas Léxicas

V.20.1.1.Palavras chaves e classificação facetada

As buscas de palavras chave, seja pelo método tradicional, seja pelo método de classificação facetada, têm seus algoritmos suficientemente detalhada na literatura e por isso não é preciso descreve-las [Pri87,Sal68]. Note-se, porém, que modelos adotamos para essas busca. No caso das palavras chave, dois tipos de busca podem ser feitos: o primeiro utiliza *full-text retrieval*, buscando as palavras chave em meio a textos, o segundo utiliza o conceito de palavras chaves propriamente dito, buscando-as entre palavras destacadas pelo modelo, na forma de *classificações.*

A busca por faceta utiliza um modelo semelhante ao implementado pelo sistema CAOS [Wer92], adaptado para o modelo de componentes implementado no sistema Tabetá, porém não deixando de incluir a distância cognitiva entre as palavras, como indicado por Prieto-Diaz, de forma a possibilitar buscas relaxadas.

Para implementar as distâncias cognitivas foram utilizados mecanismos de busca da informação capazes de propor um valor de ligação entre duas palavras a partir da distância e frequência dos pares de palavras em textos [Sal68], essa implementação evita que o usuário seja obrigado a definir os valores de antemão e pode, ainda, ser feita de forma dinâmica, representando a mudança do banco de dados durante o tempo.

O algoritmo implementado, baseado em [Sal68], inicia encontrando a frequência, de algumas palavras de alta frequência, em um conjunto de documentos, sendo criada um grafo, representando o peso de cada palavra em cada documento analisado. Coeficientes de similaridade são então computados utilizando estes dados, sendo dependentes da frequência em que os termos aparecem juntos em um documento.

V.20.1.2.Similaridade como Fractais

Peter Kokol [Kok94] apresenta um fractal "como uma forma feita de partes similares ao todo de algum modo. Desta forma, fractais precisam ter alguma invariancias: à rotação, à escala, à translação , à divisão, etc.

Levando em conta nomes de variáveis em programas de computador, ou ainda palavras em especificações formais, podemos perceber que existem alguns invariantes:

1. rotação, podemos rodar as letras de uma variável e ainda ter um nome de variável,
2. divisão, podemos retirar um parte do nome de uma variável e ainda obter um nome de variável, e,
3. concatenação, podemos juntar dois nomes de variáveis e o resultado será um nome de variável.

Isto significa que estes tipos de palavras podem ser modelados como um conjunto de Cantor com a dimensão fractal:

$$D = \frac{1}{1 - \frac{\log(1-p_0)}{\log N}}$$

onde N é o número de letras legais utilizadas na construção dos nomes de variáveis e p_0 é a probabilidade de aparecer um símbolo delimitador"

Isto significa que é possível medir a semelhança entre duas palavras utilizando a teoria dos fractais.

V.20.2.Buscas Sintáticas

V.20.2.1.Semelhança de árvore sintática

Cada especificação VDM é guardada no sistema junto com sua representação na forma de uma árvore sintática abstrata. Assim , quando o usuário apresenta uma especificação parcial, a árvore sintática desta especificação pode ser comparada as

árvores sintáticas presentes no sistema em busca de uma árvore, ou um conjunto de árvores, que apresente certa semelhança.

Atualmente a medida de semelhança é medida pelo número de nós que podem ser encontrados nas mesmas posições relativas entre duas árvores comparadas. O algoritmo abstrai as diferenças de operadores do mesmo tipo (que possuem a mesma assinatura) e trata o conceito de semelhança como o de uma igualdade relaxada, inicialmente tentando que as duas árvores 'encaixem' perfeitamente, e depois tentando retirar os ramos que evitem encontrar a igualdade.

V.20.3.Buscas Semânticas

As buscas semânticas estão baseadas na semântica estática de VDM e são aplicadas a tipos abstratos de dados construídos a partir dos tipos de dados, funções e operadores definidos em uma especificação VDM. Para definir um tipo abstrato de dados, é utilizado um grafo representando um diagrama ADJ [Jon90], que indica não só o tipo de dados como estrutura de dados, mas também o conjunto de operadores que podem ser aplicado sobre este tipo e o tipo dos resultados dessas aplicações. Duas formas de analisar tipos de dados foram implementadas: determinação de sub-tipos e determinação de tipos conforme. Caso um tipo proposto e um tipo do banco de dados possuam uma destas relações, o último será apresentado ao usuário.

V.20.3.1.Sub-tipos

A relação sub-tipo para VDM está bem definida no padrão da linguagem[ISO92] e pode ser utilizada para comparar dois tipos. A interpretação do padrão basicamente diz que um tipo S é um sub-tipo de outro tipo T se for uma restrição desse tipo T. Isso significa que todos os valores de S são válidos como do tipo$T$, mas nem todos os valores de T são válidos como do tipo S.

V.20.3.2.Tipos Conforme

Um tipo S é **conforme** com um tipo T se:

1. S fornece, pelo menos, todas as operações fornecidas por T;
2. para cada operações em T, a operação correspondente em S tem o mesmo número de argumentos e resultados;

3. os tipos abstratos dos resultados das operações de S são compatíveis com os tipos abstratos dos resultados das operações correspondentes em T, e,
4. os tipos abstratos dos argumnetos das operações de T são compatíveis com os tipos abstratos dos argumentos das operações correspondentes de S, i.e., os argumentos devem ser conformes no sentido inverso.

Por essa definição, uma extensão de um tipo é conforme com esse tipo, i.e., se um tipo S ;é a extensão de um tipo T, então S é conforme com T. Além disso, todo tipo é confrome com si mesmo e , se S é conforme com T, em geral T não é conforme com S, e se isso acontecer, S e T são tipos equivalentes (ou o mesmo tipo).

V.20.4.Exemplos das buscas

Voltemos a analisar o exemplo apresentado no capítulo III para verificar como as buscas seriam realizadas.

O usuário, desejando implementar uma pilha de reais poderia apresentar a seguinte definição:

```
push(e : ℝ)
ext wr p : Stack
post p = [e] ^ p~
```

neste caso, o sistema produziria as seguintes informações para a busca:

1. palavras chave: *push*, *stack* (*e* e *p* são desprezados por possuírem menos de 2 caracteres);
2. tipos: não houve a definição total de um tipo, porém o sistema pode inferir que Stack é um tipo e que **push** é uma operação de Real x Stack resultando em Stack;
3. sintática: o sistema gera a árvore sintática para a definição.

Supondo que o sistema possui-se (apenas) as definições de Pilha como apresentadas no capítulo III, ele encontraria:

1. pelo método de palavras chaves: todas as definições com a palavra *push*, que são 3. Nenhuma com a palavra *Stack*. Caso houvesse uma entrada no

thesaurus dizendo que *Stack* e *Pilha* são palavras semelhantes, poderia encontrar todas as definições com a palavra *Pilha*;

2. pelo método sintático: como as árvores são iguais, mesmo com os identificadores diferentes, encontraria uma especificação de *push* para *Inteiros*;
3. pelo método de tipo: não encontraria nenhum tipo semelhante;

Neste ponto o usuário recebe uma lista de componentes, que pode ser consultada. Cada componente apresenta, então, não só sua descrição, mas também as ligações, geradas pelos objetos, indicando que abstrações podem ser aplicadas. A função *push*, por exemplo, com certeza apresentaria as abstrações "Conceito" e "Conteúdo", como representado na figura 25. Caso o tipo *Pilha* estivesse implementado em função dela, como uma agregação, apresentaria também uma ligação para "Agregação".

De tal forma, o usuário, e deve, utilizar especificações parciais para buscar especificações completas. Da mesma forma, já que o algoritmo de comparação sintática é capaz de encontrar uma árvore da qual a árvore apresentada seja sub-árvore, as especificações das funções podem também ser parciais, indicando apenas características essenciais.

V.21.O Modelo do Usuário

Parte do sistema é dedicada a manter um modelo do usuário, caracterizado por suas consultas e as respostas as suas consultas e a sequências dos nós visitados.

Na presente implementação o modelo do usuário pode ser utilizados pelo próprio usuário, ao pedir informações sobre estados anteriores, e por um usuário a nível de gerente interessado em calcular fatores de desempenho do banco de dados. Nas conclusões desta tese sugere-se que este modelo seja utilizado por um sistema especialista para otimizar as buscas e facilitar o uso do sistema, ou por um tutor inteligente para orientar o usuário nos caminhos a navegar.

V.22. Uso do sistema

O sistema está implementado utilizando o sistema operacional Unix, o banco de dados O2, softwares de uso público como os compiladores GCC e G++, GNU make, CERN http, NCSA httpd e Mosaic.

O uso inicial do sistema demonstrou sua utilidade e permitiu que fossem tiradas várias conclusões, apresentadas no próximo capítulo.

VI. CONCLUSÃO

"'Onde se encontra a Árvore do Conhecimento, ali é o Paraíso': assim falam as mais antigas e as mais novas serpentes"
Friedrich Nietzche, in *Além do Bem e do Mal*

Neste capítulo apresentamos as conclusões finais, enfatizando as contribuições originais desta tese e indicando caminhos para trabalhos que continuem as idéias aqui defendidas.

VI.1. Uma visão geral do trabalho

Para avaliar as contribuições dadas por esta tese, é importante analisar o trabalho como um todo. Em primeiro lugar, foi desenvolvido, por meio de protótipos de um ambiente de desenvolvimento, um conjunto de idéias sobre reutilização. Essas idéias foram utilizadas então para definir um modelo conceitual baseado no modelo de referência 3C e um sistema foi implementado de forma a suportar esse modelo.

Estas três fases implicaram em uma série de contribuições originais, algumas de ordem conceitual, outras de ordem prática. As contribuições de ordem conceitual originais desta tese são:

1. a criação e uso de uma linguagem gráfica para descrever a abstração representacional de VDM, denominada VDM/GDL,
2. a determinação da necessidade e a definição de uma linguagem de ligação entre as especificações formais e as implementações em uma linguagem de programação,
3. a definição de um método simples de utilizar a análise estruturada junto com o método VDM,
4. a conceituação da atividade de desenvolver software como uma atividade de investigação tecnológica e a decorrente verificação de que **sempre** existe reutilização,

5. a definição de um processo de desenvolvimento suportando a criação e a reutilização baseado no modelo epistemológico da investigação tecnológica que pode ser aplicado a qualquer ciclo de vida,
6. a definição dos requisitos de um **bom método** de reutilização,
7. a definição de um método de reutilização baseado no uso da linguagem VDM para representar o **conceito** dos componentes reutilizáveis, e
8. a formalização dos conceitos de modelagem, i.e., classificação, agregação e generalização, como conceitos essenciais em um sistema de reutilização.

As contribuições de ordem prática originais desta tese são:

1. a especificação e implementação de um ambiente de desenvolvimento para o método VDM e suas extensões suportando reutilização, domínios de aplicação e método estruturado,
2. a especificação e implementação de um servidor de componentes reutilizáveis em um banco de dados orientado a objetos,
3. a especificação e implementação de um servidor de hipertexto capaz de interagir com o servidor de componentes reutilizáveis e com outros servidores,
4. a especificação e implementação de uma extensão do programa make, hmake, capaz de recuperar componentes por ftp e http,
5. A extensão do método de classificação facetada automatizando o cálculo das distâncias cognitivas entre os termos,
6. a utilização de mecanismos de comparação de estruturas sintáticas para encontrar candidatos a reutilização,
7. a utilização dos conceitos de tipo conforme, sub-tipo e restrição para encontrar candidatos a reutilização, e,
8. a especificação e a implementação, utilizando os itens 2 a 6, de um sistema de reutilização baseado em hipertexto de âmbito global.

VI.2. O modelo de reutilização e sua implementação

Esta tese apresenta um modelo de reutilização suficientemente geral para ser aplicado em qualquer ciclo de vida.

Baseado no modelo epistemológico da pesquisa tecnológica, ou seja, baseado no método científico, esse modelo é ao mesmo tempo geral, para acomodar-se dentro dos mais diferentes ciclos de vida, e formal, para que possa ser estudado e avaliado.

Outras duas importantes contribuições, ainda no nível do modelo apresentado, foram a implementação de um sistema realmente baseado no modelo 3C de reutilização onde Conceito, Contexto e Conteúdo são **papéis** assumidos por componentes de software, e não características intrínsicas a uma ou outra forma de software, e a formalização, junto com esse modelo, dos conceitos de abstração (classificação, generalização e agregação), apresentados ao usuário no sistema Tabetá como ligações de hipertexto.

A alternativa de utilizar um sistema de hipertexto para implementar o modelo acaba por criar um sistema altamente flexível, **onde aplicações podem ser criadas apenas pela reorganização de componentes reutilizáveis**. O fato desse sistema ser especificamente construído para ser acessado via Internet também é de grande importância para a tese, pois assim usuários em vários pontos da rede não precisam mais copiar componentes, mas apenas apontá-los.

VI.3. As Ferramentas e seu uso

Na área de comunidade de software, apresentamos um servidor de hipertexto capaz de interagir com outros servidores, sejam eles do mesmo tipo ou não, e com o usuário. Para isso, foi necessário modificar o conceito de servidor passivo e estático para o de servidor ativo e dinâmico, capaz de avisar ao cliente que seu estado foi modificado.

Para suportar a manutenção de programas desenvolvidos com os componentes fornecidos pelo sistema Tabetá foi criada a ferramenta hmake, que aumenta o poder da ferramenta make, de larga utilização na comunidade de programação para Unix. Utilizando estsa ferramenta o usuário não precisa copiar para si os componentes a serem reutilizados, podendo acessá-los, de desejar, apenas no exato momento em que

são necessários. A ferramenta permite também que seja mantida uma cópia local e essa cópia seja comparada ao original (remoto).

VI.4. Os métodos de busca

A criação de um mecanismo de busca generalizado é outra importante contribuição na seleção, principalmente quando esse mecanismo é baseado na aplicação conceitual da linguistica dividindo as caracterizações léxicas, sintáticas e semânticas. O uso de um banco de dados orientado a objeto permitiu a implementação da busca generalizada, utilizando a técnica de responsabilizar cada objeto por definir sua semelhança com o que o usuário deseja, por meio do encapsulamento das funções de comparação dentro das classes.

Entre as características léxicas mais importantes, a proposta de automatizar o cálculo das distâncias cognitivas para o sistema de classificação facetada também é uma contribuição importante, na medida em que retira do bibliotecário a necessidade de reavaliar as distâncias inicialmente fornecidas.

O uso da semântica estática de VDM para determinar a possibilidade de reutilização de um tipo abstrato de dados é de extrema importância para essa tese, por demonstrar que a reutilização pode ser feita sem se basear nas estruturas artificiais da linguagem, definidas por sua sintaxe concreta, ou mesmo nas estruturas definidas pela sintaxe abstrata, mas apenas no seu significado.

VI.5. A linguagem VDM/GDL

Durante o decorrer desta tese, uma das maiores certezas é o da necessidade da linguagem VDM/GDL. Um dos maiores empecilhos a utilização de VDM tem sido a dificuldade dos usuários de ler uma especificação. Utilizar uma linguagem gráfica permite uma representação que não "assusta"o usuário por sua carga matemática.

A linguagem foi utilizada na definição do sistema CAB e na modelagem de sistemas ou partes de sistemas menores. Ela apresenta maior capacidade de

representação que linguagens semelhantes, como o modelo de entidades e relacionamentos, por implementar todo o modelo representacional de VDM.

VI.6. Propostas para continuação do trabalho

VI.6.1. Novos e melhores métodos de busca

As principais extensões possíveis a este trabalho estão relacionadas a implementação e otimização dos métodos de busca utilizados e ao processo de comunicação entre os servidores de componentes.

Vários algoritmos podem ser encontrados na literatura para medir a semelhança entre dois objetos. De interesse maior, a medida da quantidade de informação dos objetos[SW69], o método da mensagem de tamanho mínimo para transformar um objeto em outro [Pat91,PW91] e o algoritmo de mapeamento de estruturas de Falkenhainer, Forbus e Gentner [FFG89] para avaliar a semelhança entre duas estruturas sintáticas parecem ser aplicáveis a esta tese. Ainda, provadores de teoremas podem ser utilizados para encontrar duas especificações semelhantes e sistemas de classificação mais rebuscados [Bro89,BR89], incluindo classificação por meio de redes neurais [HKP91], podem ser utilizados para classificar as palavras chaves em grupos com ou sem interseção.

É importante que seja notada a semelhança do problema de encontrar estruturas reutilizáveis e o problema de integração de esquemas em banco de dados [Sou86], pois ambos buscam encontrar objetos semelhantes. O uso de técnicas de redução das estruturas sintáticas a estruturas mínimas pode ser útil, tanto como o uso de técnicas de canonização (normalização) das estruturas sintáticas.

Em especial, a transformação da sintaxe abstrata descrita em OAS para a sintaxe abstrata definida na CAS e a seguinte ordenação das estruturas em uma forma normal poderia facilitar os algoritmos de busca sintática e permitir outros algoritmos de busca baseados na semântica além da comparação de tipos.

Ainda outra possibilidade, é a de modelar os tipos mais complexos de VDM a partir dos tipos mais simples, reduzindo a quantidade de tipos possíveis e buscando maior capacidade de comparação semântica. Por exemplo, listas (*sequences*) podem ser modelas como o produto cartesiano do conjunto de naturais com o tipo que origina a lista.

VI.6.2. Suporte a orientação a objetos

O sistema implementado suporta qualquer tipo de componente de software, porém a linguagem de especificação formal utilizada não permite a criação de especificações modulares ou orientadas a objetos. Acreditamos que uma extensão de VDM para suportar orientação a objetos é uma contribuição de grande valor para esse sistema. Analisamos a linguagem VDM++ [DK92,Dür92], porém ela não apresenta herança múltipla e ainda não foi publicada em sua totalidade.

VI.6.3. Suporte a software pago

Outra extensão para este trabalho é o tratamento do processo de compra de componentes de software pelo sistema. Atualmente, software pago é suportado pela ocultação da implementação. Em linhas gerais, esse método já está delineado e contaria com as seguintes características:

1. o usuário só teria acesso as especificações dos produtos,
2. se, consultando o banco de dados, o usuário desejasse utilizar um produto de uma empresa X, ele se comunicaria com a empresa e solicitaria uma quantidade de *tokens* que permitisse a utilização desse produto,
3. esses *tokens* são licensas de acesso, fornecida pela empresa de venda e que utilizam o sistema criptografia de chave pública [DH76,Luc86] como garantia de autenticidade, e
4. o usuário, ao requisitar um componente, utiliza um dos tokens fornecido pela empresa que o vende para ter acesso ao produto.

Tal sistema poderia ser estendido para permitir a criação de contratos de software, utilizando a especificação formal como definição do objeto do contrato e novamente o sistema criptográfico de chave pública para assiná-los.

VI.6.4. Suporte inteligente

Outra extensão útil a este trabalho seria inclusão de um sistema especialista que controlasse as buscas, inclusive avaliando o modelo do usuário. Tal sistema poderia analisar cada objeto ou conjunto de objetos em relação a consulta feita, decidindo que método aplicar antes, de forma a reduzir de forma rápida a quantidade de opções que respondem uma consulta de forma errada.

O sistema especialista poderia também analisar o odelo do usuário de forma a procurar que tipo de componentes ele procura, tentando otimizar a busca baseado nesse modelo.

VI.7. Reutilização Global

A comunidade de produção de software vive um momento de crise, relacionado à tendência americana de aceitar **patentes** de algoritmos e **copyright** de interfaces com o usuário e linguagens de programação. Oficialmente o orgão responsável pelas patentes americanas está aceitando os pedidos, mas o caso ainda está sendo discutido nos tribunais.

Algumas pessoas e sociedades, como a Free Software Foundation, lutam para o desenvolvimento de software **sem** patentes. Basicamente, estas pessoas e sociedades acreditam que o principal trabalho de desenvolver software está em **bem reutilizar** algoritmos. A patente de algorimos, ao contrário da patente de mecanismos, seria então um empecilho ao progresso da área.

Simultaneamente, estas empresas põe a disposição do público, software a ser reutilizado, sendo os exemplos principais representados pelo produtos do GNU Project, muitos utilizados no desenvolvimento desta tese.

Concordando com o fato de que patentes são prejudiciais ao desenvolvimento de software, acredito que existem duas frentes de luta: a conscientização da sociedade em geral, e dos poderes decisórios em particular, de que as patentes são prejudiciais e a exemplificação das vantagens da reutilização total, por meio da implementação de softwares reutilizáveis e de sistemas de apoio a reutilização.

Espero, assim, que esta tese não seja apenas uma demonstração de conceitos e técnicas, mas também a defesa de uma política: a de que software é um produto intelectual, sendo suas representações candidatas a direitos autorais, mas que a patente de algoritmos é um empecilho ao progresso da ciência de desenvolvimento de software, por proibir a reutilização e permitir que forças econômicas dominem uma atividade especialmente intelectual: o desenvolvimento e implementação de algoritmos.

VII.BIBLIOGRAFIA

[ASU88] Aho,A. V.; Sethi,R.; Ullman,J. D. "Compilers: Principles, Techniques and Tools" . Addison-Wesley Publishing Company, Reading, Massachusetts, 1988.

[Ale89] Alencar,P. S. C.; Lucena,C. J. P. "Métodos Formais para o Desenvolvimento de Sistemas" . EBAI, Campinas, 1989.

[And93] Anderssen,M. NCSA Mosaic Technical Summary, 1993 ftp://ftp.ncsa.uiuc.edu/Mosaic.

[AI92] Andrews,D.; Ince,D. "Practical Formal Methods with VDM" . The McGraw-Hill International Series in Software Engineering, McGraw-Hil Book Company, London, 1992.

[And+92] Andrews,D. et al. VDM Specification Language - Mathematical Concrete Syntax and Corresponding Outer Abstract Syntax, 1992 Personal Communication.

[Ans91] Anselmo,F. et al. MEGA: Monte Carlo Event Generator Adaptor, 1991 CERN/LAA Technical Report CERN/LAA-MSL/01-08.

[Ara79] Aragon,D. F. "Computabilidade" . ICMSC/USP, São Paulo, 1979.

[Ara93] Arango,G. "Networks and Information Technology Redefine The Practice of Reuse" in WISR 93 - 6th Annual Workshop on Software Reuse. available for anonymous ftp, 1993.

[AP91] Arango,G.; Prieto-Díaz,R. "Introduction and Overview: Domain Analysis Concepts and Research Directions" in Domain Analysis and Software Systems Modelling, eds. Prieto-Díaz, R.; Arango, G. IEEE Computer Society Press, Los Alamitos, California, 1991.

[BR89] Banfield,J. D.; Raftery,A. E. "Model-Based Gaussian and Non-Gaussian Clustering" . Technical Report 186, Department of Statistics, University of Washington, Seattle, Washington, 1989.

[BB91] Barnes,B. H.; Bollinger,T. B. Making Reuse Cost Effective, IEEE Software ; Jan 1991; pp. 13-24.

[Bar+87] Barnes,B. et al. "A Framework and Economic Foundation for Software Reuse" in Proceedings fo the Workshop of SOtware REusability and Mantainaility. National Institute of Software Quality adn Productivity, USA, 1987.

[Bas90] Basili,V. R. Viewing Maintenance as Reuse-Oriented Software Development, IEEE Software ; Jan 1990; pp. 19-25.

[BR91] Basili,V. R.; Rombach,H. D. Support for Comprehensive Reuse, Software Engineering Journal ; Sep 1991; pp. 303-316.

[Bas+92] Basili,V. R. et al. "The Software Engineering Laboratory - An Operational Software Experience Factory" in Proceedings of the 14th International Conference on Software Engineering. ACM, Melbourne, Australia, 1992.

[Bea92] Beach,B. W. "Connecting Software Components with Declarative Glue" in Proceedings of the 14th International Conference on Software Engineering. ACM, Melbourne, Australia, 1992.

[Ber+92] Berners-Lee,T. et al. World-Wide Web: The Information Universe, Electronic Networking: Research, Applications and Policy,1(2); Spring 1992.

[Ber92] Berry,D. M. "Academic Legitimacy of the Software Engineering Discipline" . Technical report CMU/SEI-92-TR-34 ESC-TR-92-034, November 1992, Software Engineering Institute - Carnegie Mellon Univeristy, Pittsburgh, Pennsylvania, 1992.

[BD91] Bersoff,E. H.; Davis,A. M. Impacts of Lyfe Cycle Models on Software, Communications of ACM,34(8); Aug 1991; pp. 104-118.

[BCN92] Bertini,C.; Ceri,S.; Navathe,S. B. "Conceptual Database Design" . The Benjamin/Cummings Publishing Company, redwood City, California, 1992.

[BL90] Berzins,V.; Luqi. An Introduction to the Specification Language Spec, IEEE Software ; Março 1990; pp. 74-84.

[BR89] Biggerstaff,T. J.; Richter,C. "Reusability Framework, Assessment and Directions" in Software Reusability, eds. Biggerstaff, T. J.; Perlis, A. ACM Press, 1989.

[BjoX] Bjorner,D. "The Vienna Development Method (VDM): Software Specification & Program Synthesis" in Mathematical Studies of Information Processing. Springer-Verlag, Berlim, 1978.

[Bjo+82] Bjorner,D. et al. "Formal Specification and Software Development" . Prentice-Hall International, Inc., London, 1982.

[Bjo+85] Bjorner,D. et al. "The RAISE Project: Fundamental Issues and Requirements" . ESPRIT Report RAISE/DDC/EM/1/V6,, Lyngby, Denmark, 1985.

[BP91] Bollinger,T. B.; Pfleeger,S. L. Economics of Reuse: issues and alternatives, Information and Software Technology,32(12); December 1991; pp. 643-652.

[Bow53] Bowden,B. V. "Faster Than Thought, A Symposium on Digital Computing Machines" . Sir Isaac Pitman & Sons, Ltd., London, 1953.

[Jon89] Bowen,J. POS - formal specification of a Unix tool, Software Engineering Journal ; January 1989.

[Bro89] Brooks,D. R. "Manual de Metodologia Cladística" . Workshop sobre Metodologia Cladística, Academia Brasileira de Ciências, Rio de Janeiro, 1989.

[BSI92] BSI IST/5/50. "VDM Specification Language: Proto Standard" ., 1992.

[Bun87] Bunge,M. "Epistemologia, Curso de Atualização" 2nd. Biblioteca de Ciências Naturais, 4, T.A. Queiroz, Editor, São Paulo, 1987.

[CB91] Caldiera,G.; Basili,V. R. Identifying and Qualifying Reusable Software Components, IEEE Computer ; February 1991; pp. 61-70.

[Cer93] Cerqueira. "Software - Direito Autoral e Contratos" . ADCOAS, Rio de Janeiro, 1993.

[Cha90] Chang,S. K. "Principles of Visual Programming Systems" . Prentice-Hall International, Inc., London, 1990.

[CJ92] Cheng,B. H. C.; Jeng,J. "Formal Methods Applied to Reuse" in WISR'92:5th Annual Workshop on Software Reuse, eds. Griss, M.; Tracz, W., Palo Alto, California, 1992.

[Che92] Cheng,J. Parameterized Specs for SW Reuse, ACM SigSoft - Software Engineering Notes,17(4); Oct 1992; pp. 53-59.

[CM84] Clocksin,W. F.; Mellish,C. S. "Programming in Prolog" 2nd. Springer-Verlag, Berlin, 1984.

[Coh89] Cohen,B. Justification of formal methods for system specification, Software Engineering Journal,January 1989a; pp. 26-35.

[Coh89a] Cohen,B. A Rejustification of formal methods for system specification, Software Engineering Journal,January 1989b; pp. 26-35.

[CHJ86] Cohen,B.; Harwood,W. T.; Jackson,M. I. "The Specification of Complex Systems" . Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[Con87] Conklin,J. "A Survey of Hypertext" . MCC, 1987.

[Cox86] Cox,B. J. "Object-Oriented Programming: An Evolutionary Approach" . Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[DeM91] DeMarco,T. "Non-Technological Issues in Software Engineering" in . IEEE, 1991.

[DH76] Diffie,W.; Hellman,M. E. New Directions in Cryptography, IEEE Transactions of Information Theory,IT-22(6) 1976; pp. 644-654.

[Dür92] Dürr,E. Syntactic Description of the VDM++ Language, 1992 Cap Gemini Innovation document.

[DK92] Dürr,E.; Katwijk,J. van. VDM++, A Formal Specification Language for Object-Oriented Design, 1992 comunicação pessoal.

[DL91] Dürr,M.; Lang,S. M. "Hypertext and Object-Orientation: The Dual Approach" in Proc. Intl. Conference on Database Systems in Office, Technology and Science. Springer, Kaiserslautern, Germany, 1991.

[FFG89] Falkenhainer,B.; Forbus,K. D.; Gentner,D. The Structure Mapping Engine: Algorithm and Examples, Artificial Intelligence,41 1989; pp. 1-63.

[Far91] Faria,E. C. MARTE: Um Meta-Gerador de Aplicações baseado na Reutilização de Templates. MSc Thesis, Coppe/Universidade Federal do Rio de Janeiro 1991

[FE92] Fields,B.; Elvang-Goranson,M. A VDM Case Study in mural, IEEE Transactions on Software Engineering,18(4); April 1992; pp. 279-295.

[Fis+93] Fischer,B. et al. Software Component Retrieval based on Pre- and Postconditions (VERY) Draft Status Report, 1993 Avaible from the authors (fisch,gode,mkiever,rhimeir)@ips.cs.tu-bs.de.

[Fis+93] Fisher,B. et al. Software Component Retrieval based on Pre and Postconditions (VERY) Draft Status Report, 1993 personnal communication.

[GC92] Gaffney Jr.,J. E.; Cruickshank,R. D. "A General Economics Model of Software Reuse" in Proceedings of the 14th International Conference on Software Engineering. ACM, Melbourne, Australia, 1992.

[GPS93] Garzotto,F.; Paolini,P.; Schwabe,D. HDM - A Model-Based Approach to Hypertext Application Design, ACM Transactions on Information Systems,11(1); January 1993; pp. 1-26.

[GPT89] Gibbs,S.; Prevalakis,V.; Tsichritzis,D. "Software Information Systems: A Software Community Perspective" in, eds. Tsichritzis, D. Centre Universitaire D'Informatique, Université de Genève, Genève, 1989.

[GTwp] Gibbs,S.; Tsichritzis,D. Software Licensing versus Software Reuse working paper.

[Gir92] Girardi,R. "Application Engineering: Putting Reuse to Work" in Object Frameworks, eds. Tsichritzis, D. Université de Genève, Centre Universitaire d'Informatique, Genève, 1992.

[HKP91] Hertz,J.; Krogh,A.; Palmer,R. G. "Introduction to the Theory of Neural Computation" . Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[Hob93] Hobbs,E. T. "A Uniform Data Model for Reuse Library Interoperability" in WISR 93 - 6th Annual Workshop on Software Reuse. available for anonymous ftp, 1993.

[Hof79] Hofstadter,D. R. "Gödel, Escher, Bach: An Eternal Golden Braid" . Penguim Books, London, 1979.

[IPT92] IPTES. The IPTES VDM-SL Toolbox, 1992.

[ISO93] ISO WG/5/19. Draft Minutes of the Second ISO Meeting held in Odense on 27th April 1993, 1993.

[Jon90] Jones,C. B. "Systematic Software Development Using VDM" 2nd. Prentice Hall International Series is Computer Science, Prentice Hall, New York, 1990.

[JS90] Jones,C. B.; Shaw,R. C. F. "Case Studies in Systematic Software Development" . Prentice Hall International Series in Computer Science, Prentice Hall, Cambridge, 1990.

[JP88] Jones,G.; Prieto-Diaz,R. "Building and Managing Software Libraries" in COMPSAC 88: The twelfth Annual International Computer Software and Applications Conference. IEEE Computer Society Press, Chicago, Illinois, 1988.

[KRT89] Katz,S.; Richter,C. A.; The,Khe-S. "PARIS: A System for Reusing Partially Interpreted Schemas" in Software Reusability, eds. Biggerstaff, T. J.; Perlis, A. ACM Press, 1989.

[Kok94] Kokol,P. The Self-Similarity and Computer Programs, ACM SIGPLAN Notices,29(1); January 1994; pp. 9-12.

[Kru92] Krueger,C. W. Software Reuse, ACM Computing Surveys,24(2); June 1992; pp. 131-184.

[LL91] Larsen,P. G.; Lassen,P. B. "An Executable Subset of Meta-IV with Loose Specification" in VDM 91 - Formal Software Development Methods. Springer-Verlag, Berlin, 1991.

[LP93] Larsen,P. G.; Plat,N. Standards for Non-Executable Specification Languages, The Computer Journal,35(6) 1993; pp. 567-573.

[Lei+93] Leite,J. C. S. P. et al. "PROJETO DRACO-PUC: Draco-Puc, v 2.0 e Domínios de Interface e Bancos de Dados" in VII Simpósio Brasileiro de Engenharia de Software. SBC - Sociedade Brasileira de Computação, 1993.

[Luc86] Lucchesi,C. L. "Introdução a Criptografia Computacional" . Editora da Unicamp, Campinas, 1986.

[Mai91] Maiden,N. Analogy as a Paradigm for Specification Reuse, Software Engineering Journal ; Jan 1991.

[MS92] Maiden,N. A.; Sutcliffe,A. G. Exploiting Reusable Specifications Through Analogy, Communications of ACM,34(4); Apr 1992; pp. 3-15.

[MR92] Margono,J.; Rhoads,T. E. "Software Reuse Economics: Cost-Benefit Analysis on A Large-Scale Ada Project" in Proceedings of the 14th International Conference on Software Engineering. ACM, Melbourne, Australia, 1992.

[MM89] Mattoso,A. L. Q.; Monte,L. C. M. Hiperficha: Hipermídia para Desenvolvimento de Software, 1989 Relatórios Técnicos do Programa de Engenharia de Sistemas e Computação. COOPE, UFRJ, Rio de Janeiro.

[MA89] Mendes,S.; Aguia,T. C. "Métodos para Especificação de Sistemas" . Editora Edgard Blücher Ltda., São Paulo, 1989.

[Mey87] Meyer,B. Reusability: The Case for Object-Oriented Design, IEEE Software ; March 1987; pp. 50-63.

[Mey88] Meyer,B. "Object-Oriented Software Construction" . Prentice-Hall International Ltd., Englewood Cliffs, NJ, 1988.

[MMM93] Mittermeir,R. T.; Mili,R.; Mili,A. "Building a Repository of Software Components, A Formal Specifications Approach" in WISR 93 - 6th Annual Workshop on Software Reuse. available for anonymous ftp, 1993.

[Nei84] Neighbors,J. M. The Draco Approach to Constructing Softwre from Reusable Components, IEEE Transactions on Software Engineering,SE-10(5); September 1984; pp. 564-574.

[Nei91] Neighbors,J. "The Evolution from Sofware Components to Domain Analysis" in V Simpósio Brasileiro de Engenharia de Software. Sociedade Brasileira de Computação, Ouro Preto, 1991.

[Nel87] Nelson,T. H. "Literary Machines" XU 87.1. Xanadu Project, Indiana, 1987.

[Ost+92] Ostertag,E. et al. Computing Similarity in a Reuse Library System: An AI-Based Approach, ACM Transactions on Software Engineering and Methodology,1(3); July 1992; pp. 205-228.

[Pat91] Patrick,J. D. "SNOB: A program for discrimination between classes" . Technical Report 91/151, Department of Computer Science, Monash University, Clayton, Victoria, Australia, 1991.

[PW91] Patrick,J. D.; Wallace,C. S. "Coding Decision Trees" . Technical Report 91/153, Department of Computer Science, Monash University, Clayton, Victoria, Australia, 1991.

[Pet91] Peterson,A. S. Coming to Terms with Software Reuse Terminology: a Model-Based Approach, ACM SigSoft - Software Engineering Notes,16(2); Apr 1991; pp. 45-51.

[PKT92] Plat,N.; van Katwijk,J.; Toetenel,H. Application and Benefits of Formal Methods in Software Development, Software Engineering Journal ; Sep 1992.

[PL92] Plat,N.; Larsen,P. G. An Overview of the ISO VDM-SL Standard, ACM SIGPLAN Notices,27(8); August 1992; pp. 76-82.

[PLTs] Plat,N.; Larsen,P. G.; Toetenel,H. A Formal Semantics of Data Flow Diagrams, submitted to Formal Aspects of Computing .

[PT89] Plat,N.; Toetenel,H. "Tool Support for VDM" . Delft University of Technology Technical Report, Report 89-91, Delft University of Technology, 1989.

[PT92] Plat,N.; Toetenel,H. A Formal Transformation from the BSI/VDM-SL Concrete Syntax to the Core Abstract Syntax (revised version), 1992 Delft University of Technology, Faculty of Technical Mathematics and Informatics Report 92-07.

[PP92] Podgurski,A.; Pierce,L. "Behavior Sampling: A Technique for Automated Retrieval of Reusable Components" in Proceedings of the 14th International Conference on Software Engineering. ACM, Melbourne, Australia, 1992.

[PP93] Podgurski,A.; Pierce,L. Retrieving Reusable Software by Sampling Behavior, ACM Transactions on Software Engineering and Methodology,2(3) 1993; pp. 286-303.

[Pre92] Pressman,R. S. "Software Engineering - A Practitioner's Approach" 3rd. McGraw-Hill, Inc., New York, 1992.

[Pri85] Prieto-Diaz,R. A Software Classification Scheme. Ph.D. Dissertation, University of California, Irvine 1985

[Pri87] Prieto-Diaz,R. "Domain Analysis for Reusability" in COMPSAC 87: The eleventh Annual International Computer Software and Applications Conference. IEEE Computer Press, Tokyo, Japan, 1987.

[Pri90a] Prieto-Diaz,R. Domain Analysis: An Introduction, ACM SigSoft - Software Engineering Notes,15(2); April 1990a; pp. 47-54.

[Pri90] Prieto-Diaz,R. Implementing Faceted Classification for Software Reuse, IEEE Software ; May 1990b; pp. 300-304.

[Pri91a] Prieto-Diaz,R. Implementing Faceted Classification for Software Reuse, Communications of ACM,34(5); May 1991a; pp. 88-97.

[Pri91] Prieto-Diaz,R. Making Software Reuse Work: An Implementation Model, ACM SigSoft - Software Engineering Notes,16(3); Jul 1991b; pp. 61-68.

[Pri93a] Prieto-Diaz,R. Software Reusability, Classification and Domain Analysis, 1993a VII SBES Mini-Tutorial.

[Pri93] Prieto-Diaz,R. Status Report: Software Reusability, IEEE Software ; May 1993b; pp. 61-66.

[PriX] Prieto-Diaz,R. Estrategias para la construcción de bibliotecas de componentes.

[PA91] Prieto-Diaz,R.; Arango,G. "Domain Analysis and Software Systems Modeling" . IEEE Computer Society Press, Los Alamitos, Califórnia, 1991.

[PF87] Prieto-Diaz,R.; Freeman,P. Classifying Software for Reusability, IEEE Software ; Jan 1987; pp. 6-16.

[PJX] Prieto-Diaz,R.; Jones,G. A. Breathing new life into Old Software, GTE Journal of Services and Technology,1.

[PN86] Prieto-Diaz,R.; Neighbors,J. M. Module Interconnection Languages, The Journal of Systems and Software,6(4) 1986; pp. 307-334.

[Pur92] Purvis,J. B. The use od LOTOS for Specification of Graphics Software, 1992 RAL-91-094.

[Rad+92] Rada,R. et al. Software Reuse: from text to hypertext, Software Engineering Journal,7(5); September 1992; pp. 311-321.

[Red89] Redwine Jr,S. T.; Riddle,W. E. Sotware Reuse Process, ACM SigSoft - Software Engineering Notes,14(4); Jun 1989; pp. 133-135.

[Ric83] Rich,E. "Artificial Intelligence" . McGraw-Hill Book Company, New York, 1983.

[Rol92] Rolland,F. D. "Programming with VDM" . MacMillan Computer Science Series, The MacMillan Press Ltd., London, 1992.

[Sal68] Salton,G. "Automatic Information Organization and Retrieval" . Mcgraw-Hill Book Company, New York, 1968.

[SM88] Scheiderman,B.; Marchionini,G. Finding Facts vs. Browsing Knowledge in Hypertext Systems, IEEE Computer ; January 1988; pp. 70-79.

[Sch86] Schmidt,D. A. "Denotational Semantics - A Methodology for Language Development" . Allyn and Bacon, Inc., Boston, 1986.

[Sch+93] Schnase,J. L. et al. Semantic Data Modeling of Hypermedia Associations, ACM Transactions on Information Systems,11(1); January 1993; pp. 27-50.

[SW69] Shannon,C. E.; Weaver,W. "The Mathematical Theory of Communication" 4th. Illini Books, IB-13, Original copyright in 1949, The University of Illinois Press, Urbana, 1969.

[Shu88] Shu,N. C. "Visual Programming" . Van Nostrand Reinhold Company, New York, 1988.

[Sou86] Souza,J. M. de. Software Tools for Schema Integration. Ph.D. Dissertation, School of Information Systems, University of East Anglia 1986

[SpeX] Spencer,H. How to Steal Code or Inventing the Wheel Only Once copy distributed by the author via Internet's FTP.

[Ste91] Steigerwald,R. A. Reusable Software Component via Normalized Algebraic Specifications. Ph.D. Dissertation, Naval Postgraduate School, Monterrey, California 1991

[Ste92] Steigerwald,R. A. "Reusable Component Retrieval with Formal Specifications" in WISR'92:5th Annual Workshop on Software Reuse, eds. Griss, M.; Tracz, W., Palo Alto, California, 1992.

[Sti93] Stillman,M. "Reuse and Formal Methods for Ada" in WISR 93 - 6th Annual Workshop on Software Reuse. available for anonymous ftp, 1993.

[SK93] Stockwell,T.; Krause,M. "Internet Information Discovery and Retrieval Tools - Cost Effective Building Blocks for Asset Libraries" in WISR 93 - 6th Annual Workshop on Software Reuse. available for anonymous ftp, 1993.

[SD89] Stratton,S. D.; Dunsmore,H. E. "The Use of Hypertext in Software Development" . SERC TR-36-P, Dep. of Computer Science, Purdue University, Indiana, 1989.

[Str91] Stroustrup,B. "The C++ Programming Language" 2nd. Addison-Wesley Publishing Company, Reading, Massachussets, 1991.

[Tho87] Thompson,K. "Reflections on Trusting Trust" in ACM Turing Award Lectures: The First Twenty Years 1966-1985. ACM PRess, Reading, Massachusetts, 1987.

[Tra87] Tracz,W. "Software Reuse: Motivators and Inhibitors" in COMPCON S'87. IEEE, 1987.

[Tra88] Tracz,W. Software Reuse Myths, ACM SigSoft - Software Engineering Notes,13(1); Jan 1988; pp. 17-21.

[Tra90] Tracz,W. Where Does Reuse Start, ACM SigSoft - Software Engineering Notes,15(2); April 1990; pp. 47-54.

[Tra92] Tracz,W. Domain Analysis Working Group Report - First International Workshop on Software Reusability, ACM SigSoft - Software Engineering Notes,17(3); Jul 1992.

[Tra93] Tracz,W. 2nd International Workshop on Software Reuse, ACM SigSoft - Software Engineering Notes,18(3); Jul 1993; pp. A-73/A-78.

[TG93] Tracz,W.; Griss,M. Workshop on Software Reuse, ACM SigSoft - Software Engineering Notes,18(2); April 1993; pp. 74-85.

[Tre85] Tremblay,J.; Sorenson,P. G. "Compiler Writing" . McGraw-Hill Book Company, New York, 1985.

[TN92] Trotta,C.; Nierstrasz,O. "Object-Oriented Support for Generic Application Frames" in Object Frameworks, eds. Tsichritzis, D. Centre Universitaire d'Informatique, Université de Genève, Genève, 1992.

[TG90] Tsichritzis,D.; Gibbs,S. "Towards Integrated Software Communities" in Object Management, eds. Tsichritzis, D. Centre Universitaire d'Informatique, Université de Genève, Genève, 1990.

[VSC93] Valenzano,A.; Sisto,R.; Ciminiera,L. Rapid Prototyping of Protocols from LOTOS Specifications, Software Practice and Experience,23(1); January 1993; pp. 31-54.

[Wei+91] Weide,B. W.; Ogden,W. F.; Zweben,S. H. Reusable Software Components, Advances in Computers,33 1991; pp. 1-65.

[Wer92] Werner,C. M. L. Reutilização de Software no Desenvolvimento de Software Científico. D.Sc. Thesis, COPPE/UFRJ Rio de Janeiro 1992

[WS91] Werner,C. M. L.; Souza,J. M. de. "An Object-Oriented Composition Environment for Scientific Applications" in Computing in High Energy Physics'91. Universal Academy Press, Tokyo, Japan, 1991.

[Win90] Wing,J. M. A Specifier's Introduction to Formal Methods, IEEE Computer ; September 1990; pp. 8-24.

[Win92] Winston,P. H. "Artificial Inteligence" 3rd. Addison-Wesley Publishing Company, Reading, Massachusetts, 1992.

[Wir92] Wirsing,M. "Spectrum: A Formal Approach to Software Development and Reuse" in WISR'92:5th Annual Workshop on Software Reuse, eds. Griss, M.; Tracz, W., Palo Alto, California, 1992.

[WHSX] Wirsing,M.; Hennicker,R.; Stabt,R. MENU- An example for the Systematic Reuse of Specifications Unknow reference. Part of DRAGON Esprit project.

[WJ88] Wirth,N.; Jansen,K. "Pascal ISO Manual do Usuário e Relatório" . Editora Campus, Rio de Janeiro, 1988.

[Woo89] Woodcock,J. C. P. Structuring Specifications in Z, Software Engineering Journal ; Jan 1989 1989; pp. 51-66.

[XLS90] Xexéo,G.; La Commare,G.; Souza,J. de. "The CAB Database" in Proc. of the 14th Workshop of the INFN Eloisatron Project: Data Structures for Particle Physics Experiments: Revolution or Evolution. World Scientific, Erice, Sicilia, 1990.

[XLS91] Xexéo,G.; La Commare,G.; Souza,J. de. "CAB: The Cosmos Application Builder" in Proc. of the CHEP Conference. KEK, Tsukubo, Japan, 1991.

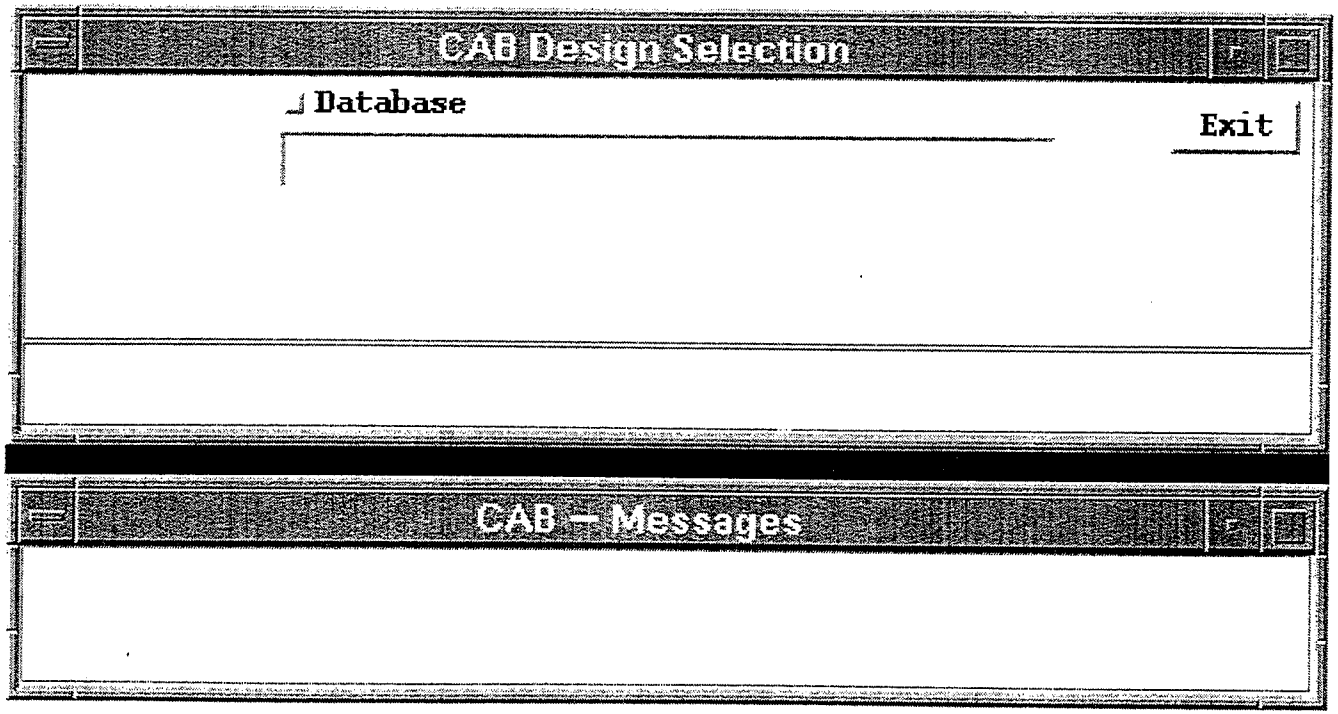
[Yeh73] Yeh,R. T.; Preparata,F. P. "Introduction to Discrete Structures for Computer Science and Engineering" . Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1973.

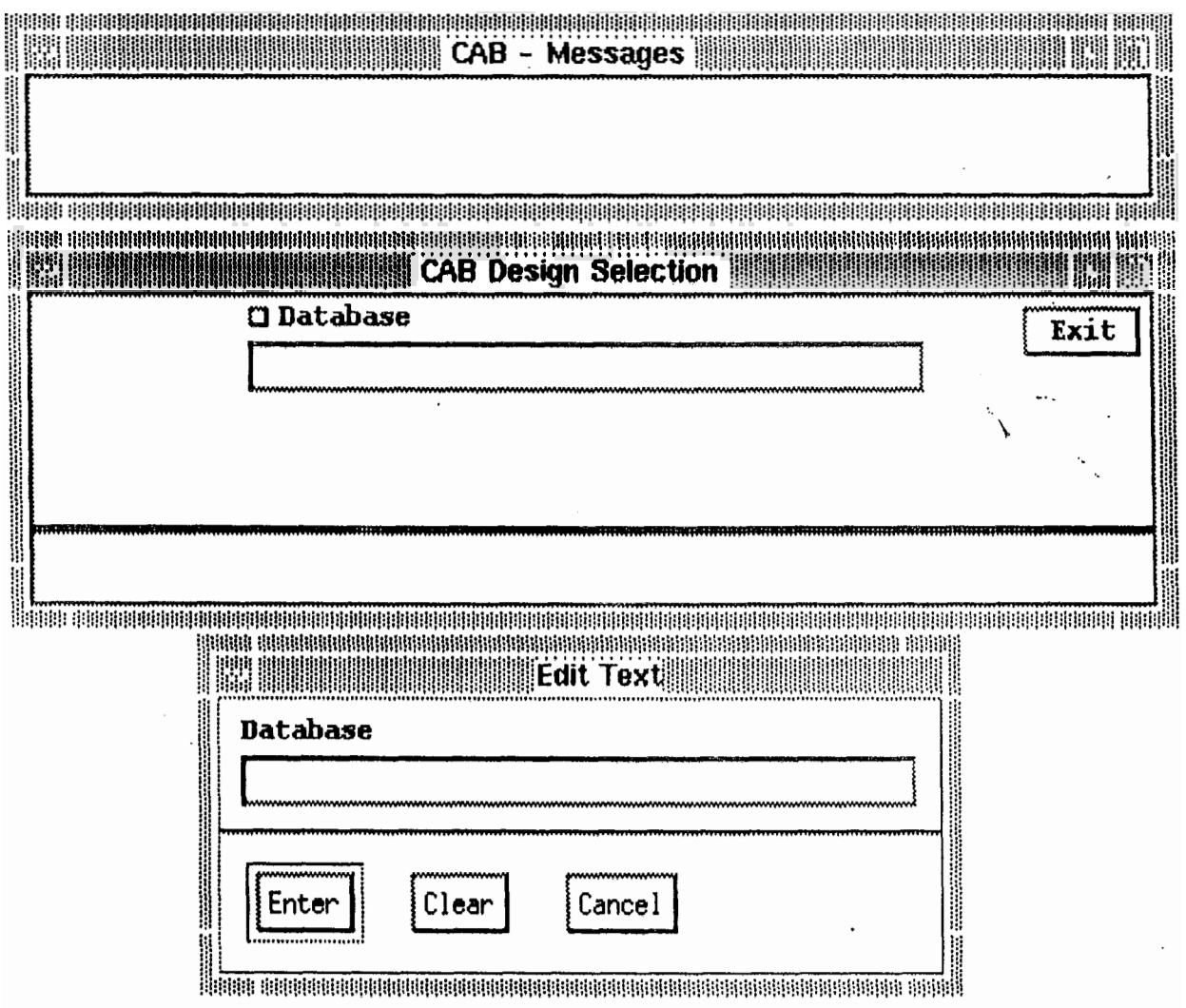
[You90] Yourdon,E. "Análise Estruturada Moderna" 3rd. Editora Campus, Rio de Janeiro, 1990.

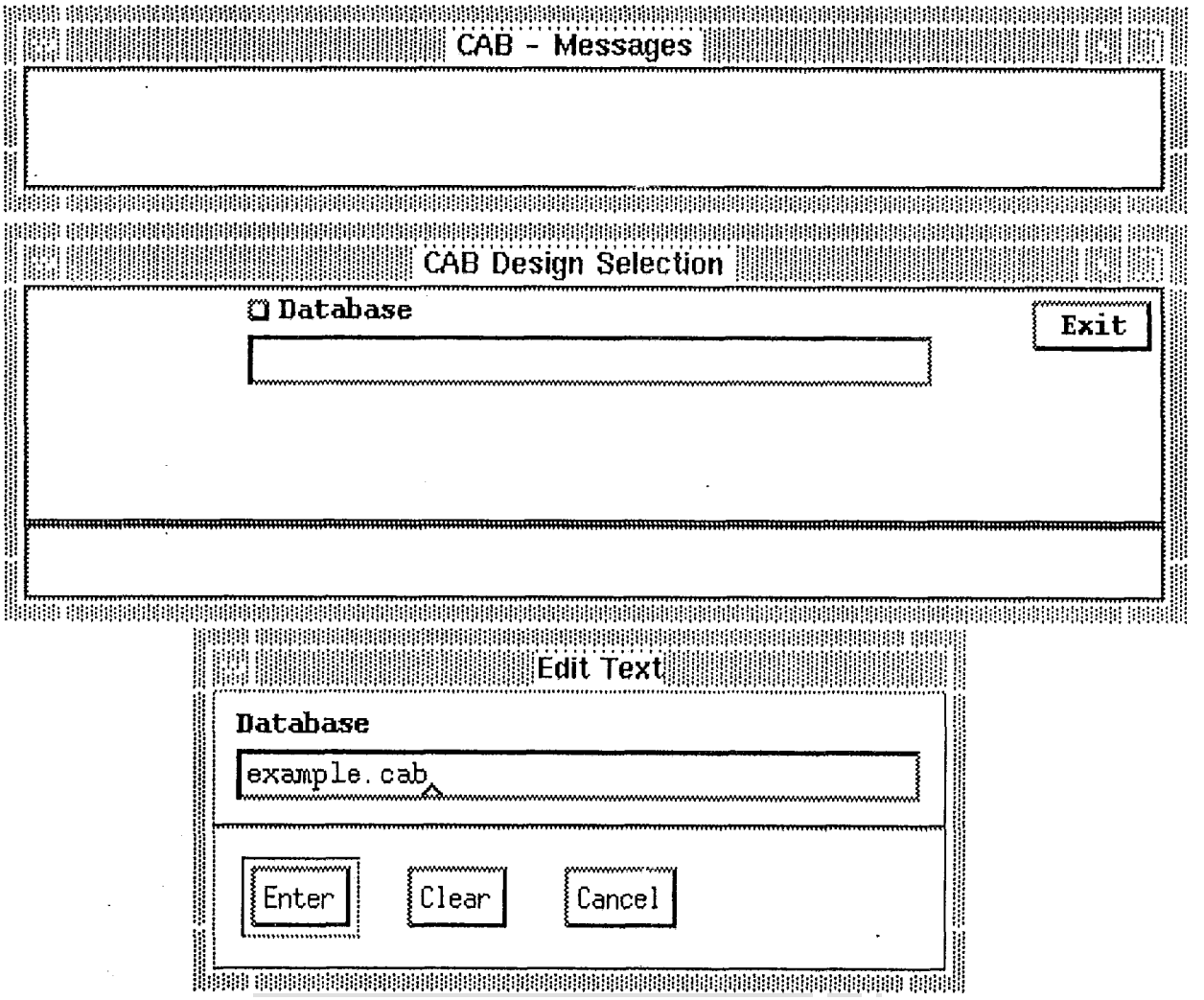
[You92] Yourdon,E. "Decline and Fall of the American Programmer" . Yourdon Press e Prentice-Hall, New York, 1992.

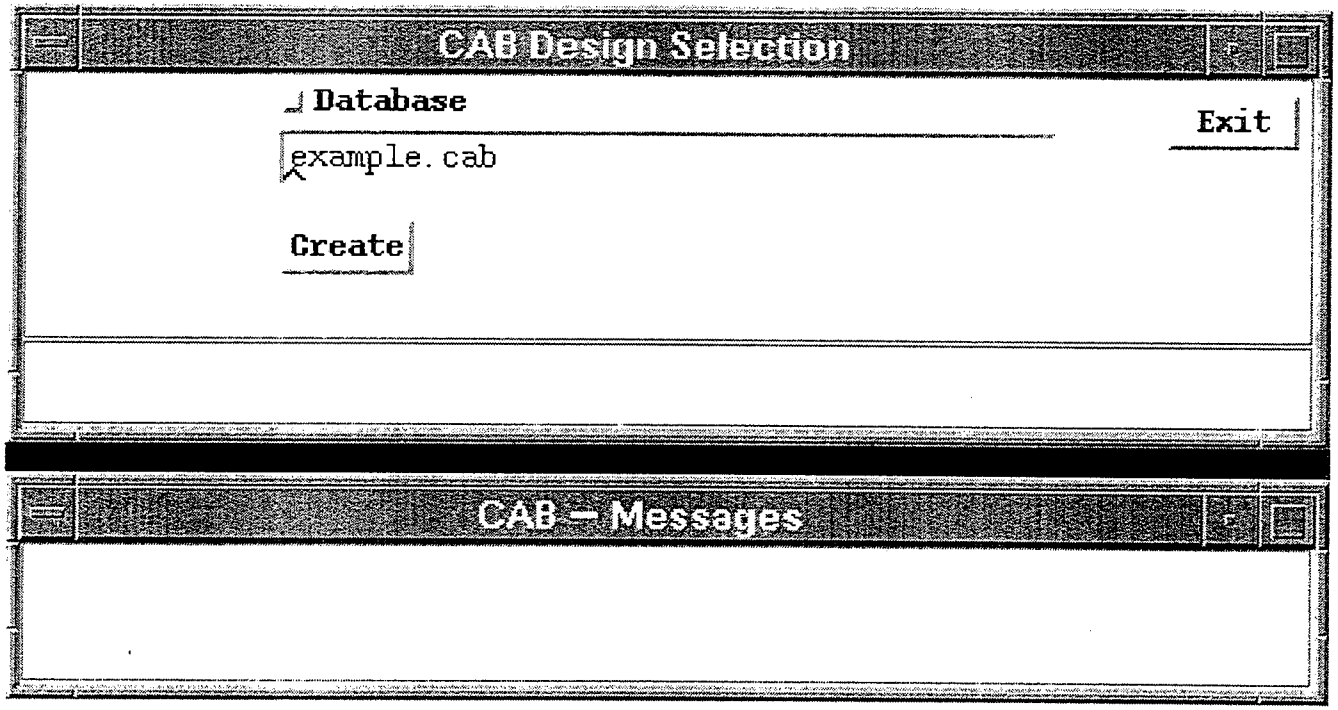
[YPS92] YPSIS. "YPSIS Toolbuilder Manual" . YPSIS, 1992.

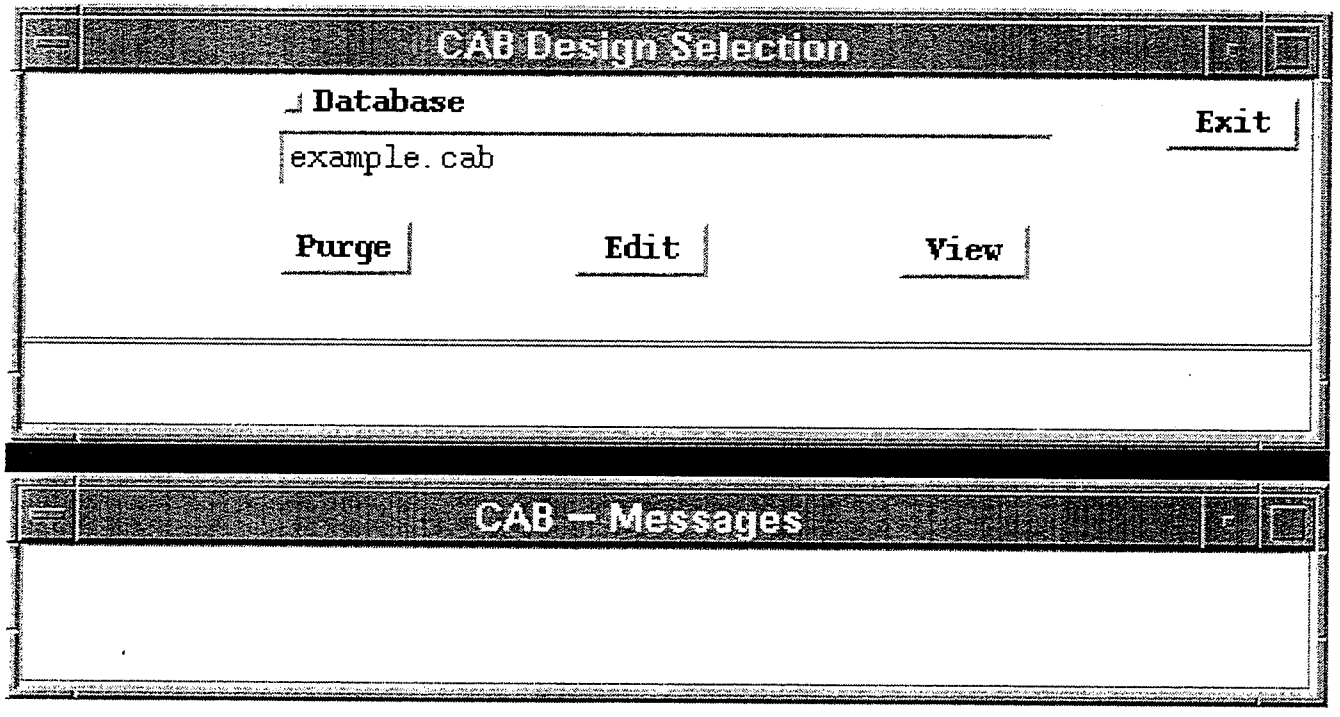
APÊNDICE I: TELAS DO SISTEMA CAB

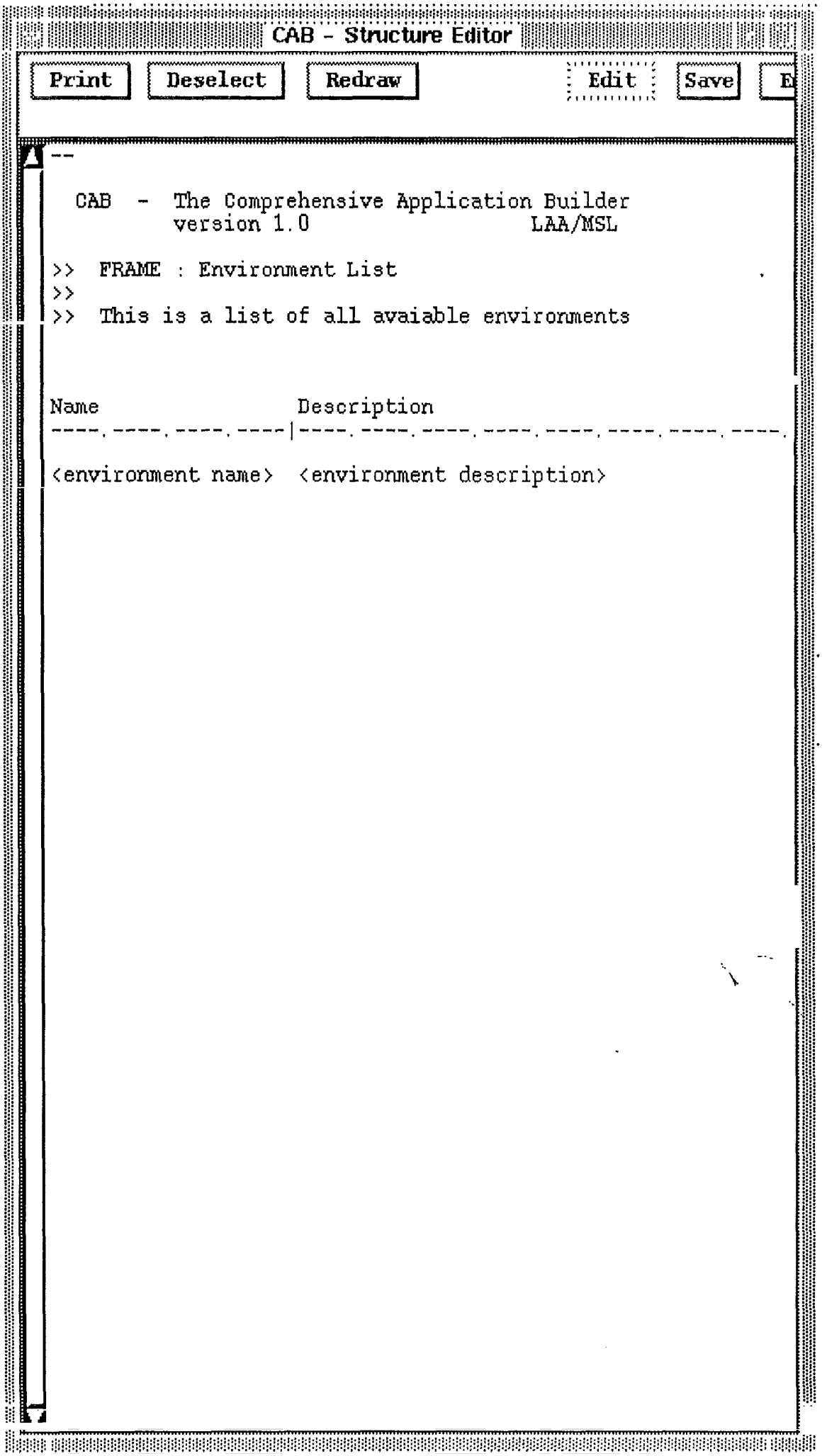

CAB Design Selection
Database
Exit
CAB — Messages

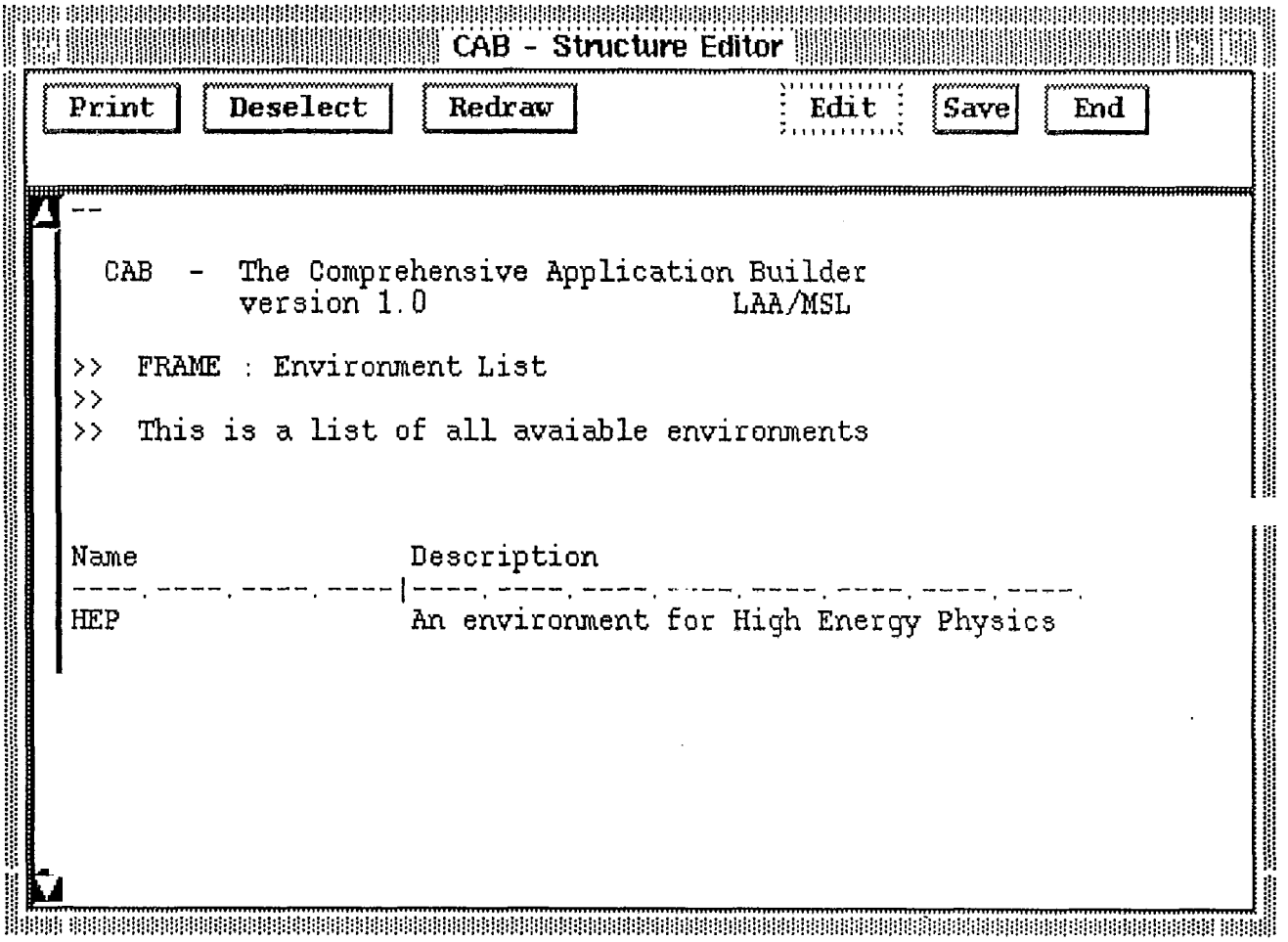

CAB - Messages
CAB Design Selection
Database
Exit
Edit Text
Database
Enter
Clear
Cancel

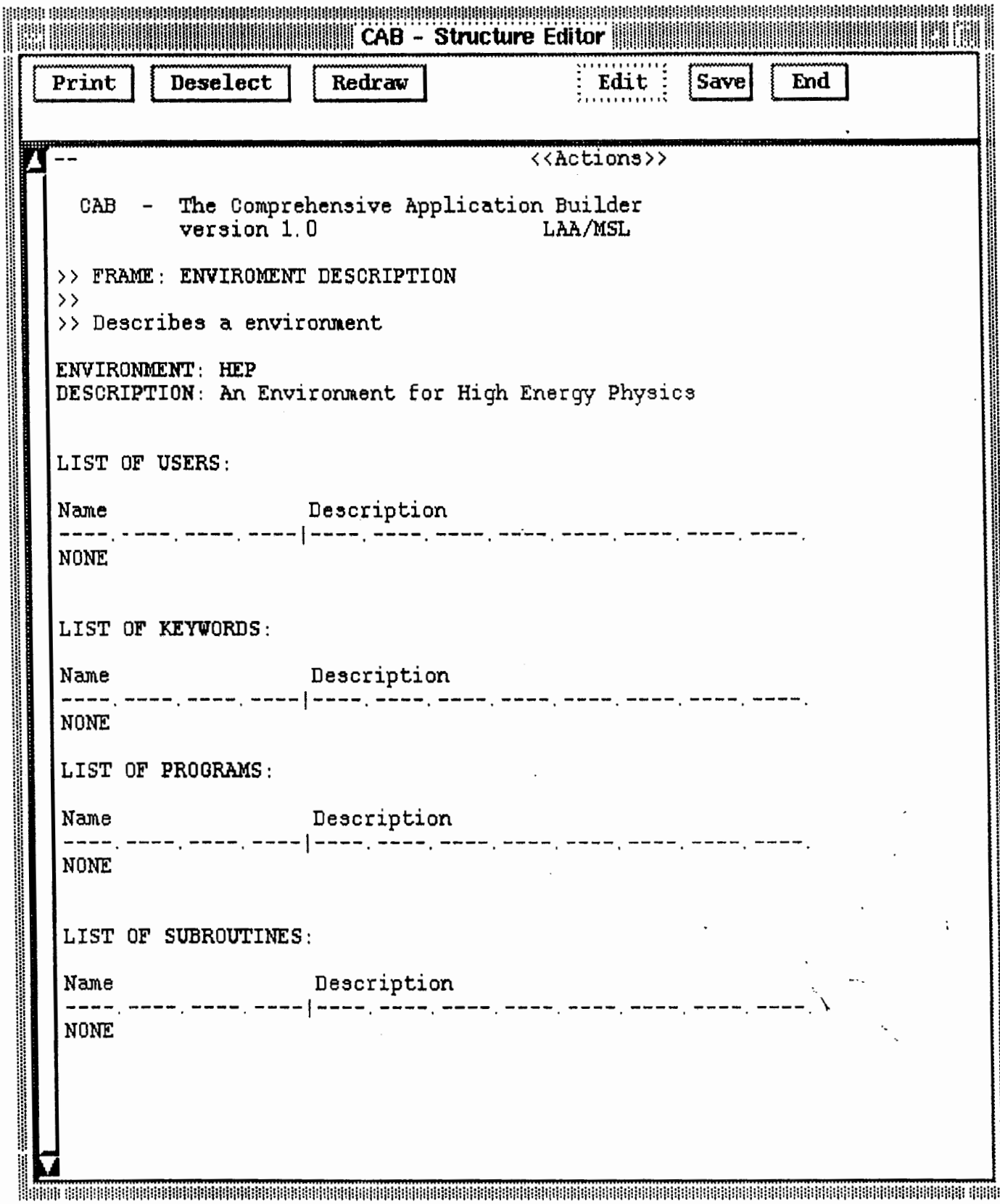

CAB - Messages
CAB Design Selection
Database
Exit
Edit Text
Database
example.cab
Enter
Clear
Cancel


CAB Design Selection
Database
example.cab
Exit
Create
CAB — Messages


CAB Design Selection
Database
Exit
example.cab
Purge
Edit
View
CAB — Messages


CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
E
--
CAB - The Comprehensive Application Builder
version 1.0 LAA/MSL
>> FRAME : Environment List
>>
>> This is a list of all avaiable environments
Name Description
<environment name> <environment description>


CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
CAB - The Comprehensive Application Builder
version 1.0 LAA/MSL
>> FRAME : Environment List
>>
>> This is a list of all avaiable environments
Name
Description
HEP
An environment for High Energy Physics


CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: ENVIROMENT DESCRIPTION
>>
>> Describes a environment
ENVIRONMENT: HEP
DESCRIPTION: An Environment for High Energy Physics
LIST OF USERS:
Name
Description
NONE
LIST OF KEYWORDS:
Name
Description
NONE
LIST OF PROGRAMS:
Name
Description
NONE
LIST OF SUBROUTINES:
Name
Description
NONE

CAB - Structure Editor

Print | Deselect | Redraw | Edit | Save | End

-- <<Actions>>

CAB - The Comprehensive Application Builder
version 1.0 LAA/MSL

>> FRAME: ENVIROMENT DESCRIPTION
>>
>> Describes a environment

ENVIRONMENT: HEP
DESCRIPTION: An environment for High Energy Physics

LIST OF USERS:

Name	Description
Geraldo Xexeo	The Creator
Giuseppe La Commare	Main Collaborator
J. Random Hacker	Standard good user
Edgar Luser	Not a good user

LIST OF KEYWORDS:

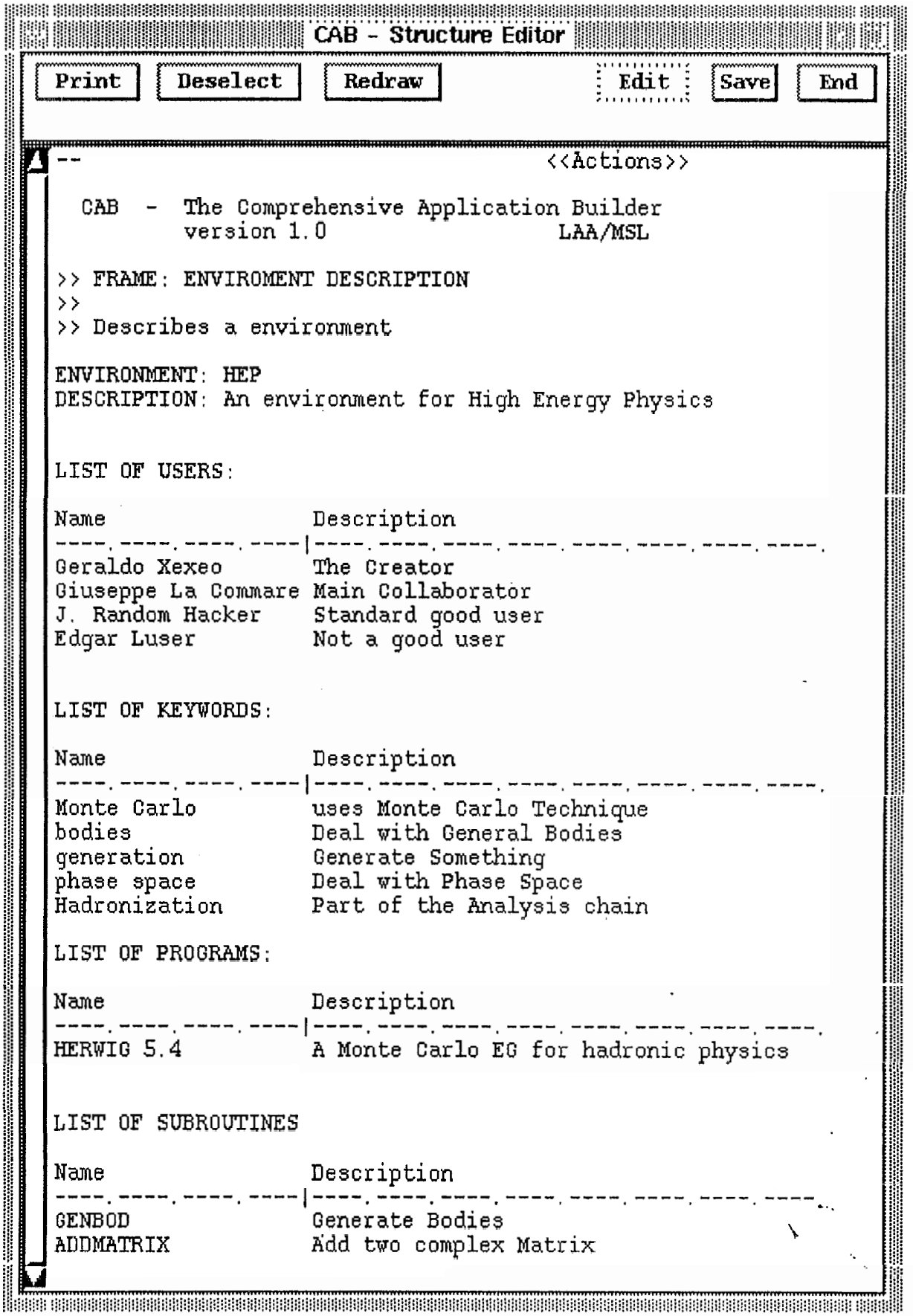
Name	Description
Monte Carlo	uses Monte Carlo Technique
bodies	Deal with General Bodies
generation	Generate Something
phase space	Deal with Phase Space
Hadronization	Part of the Analysis chain

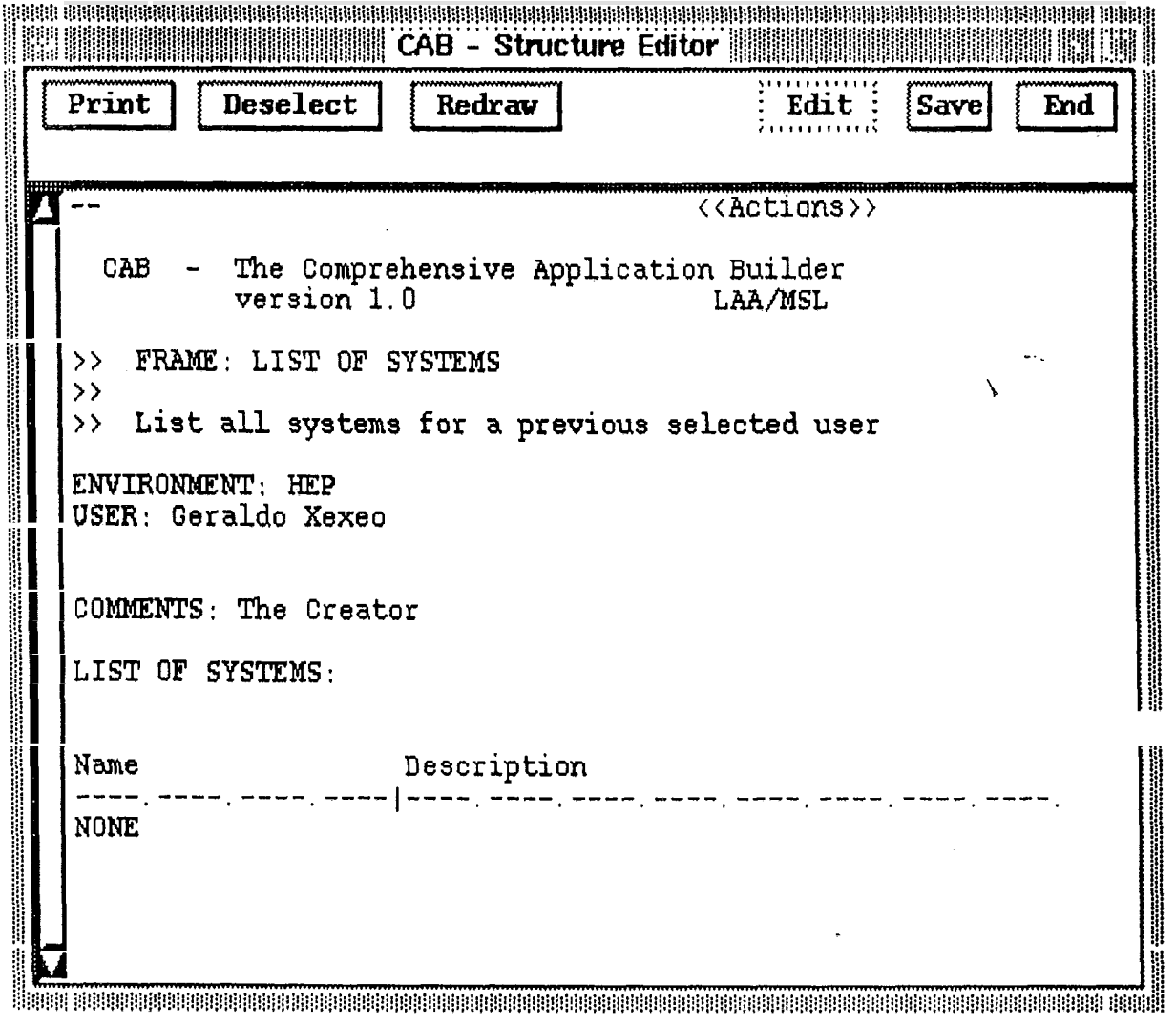
LIST OF PROGRAMS:

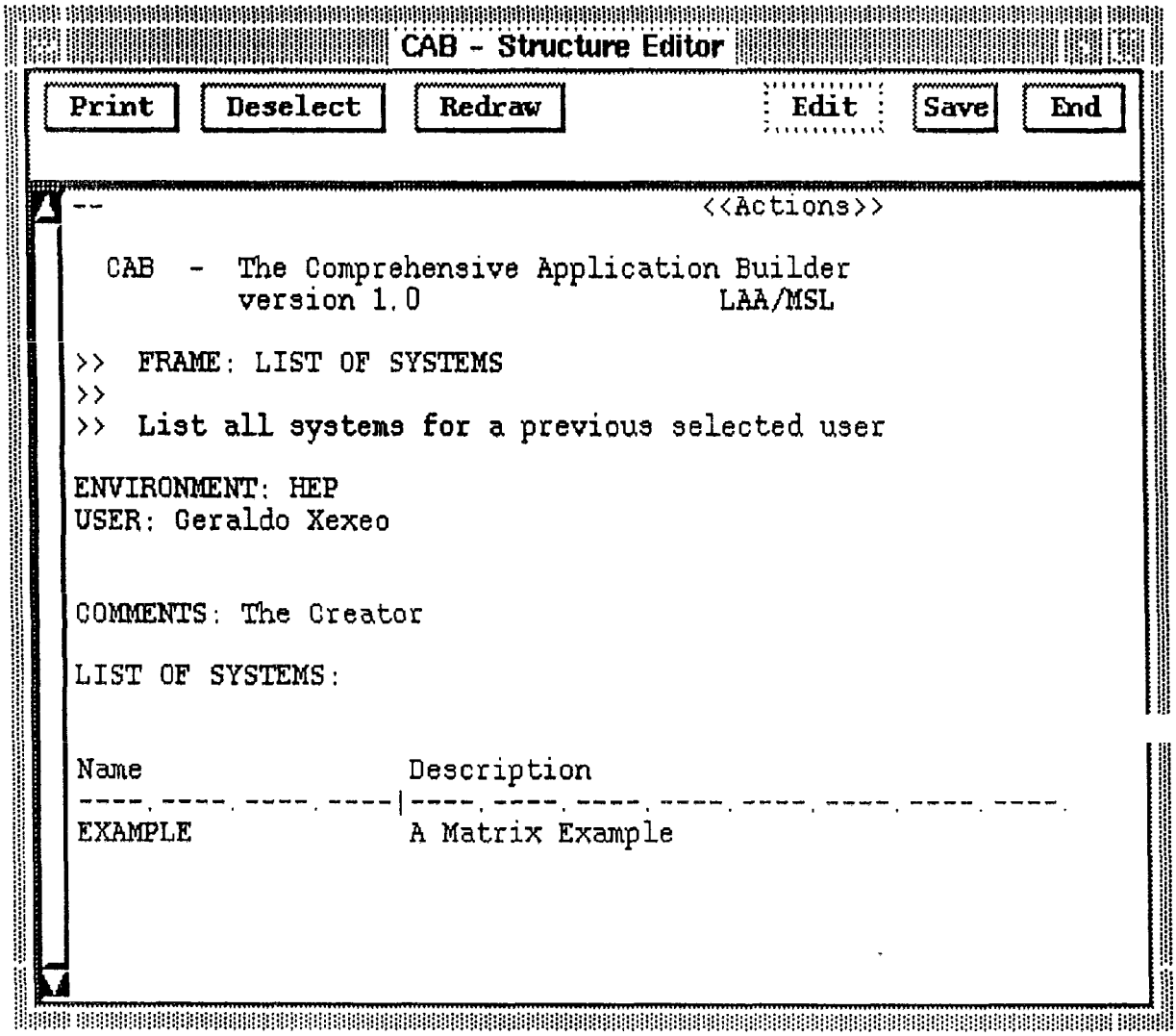
Name	Description
NONE	

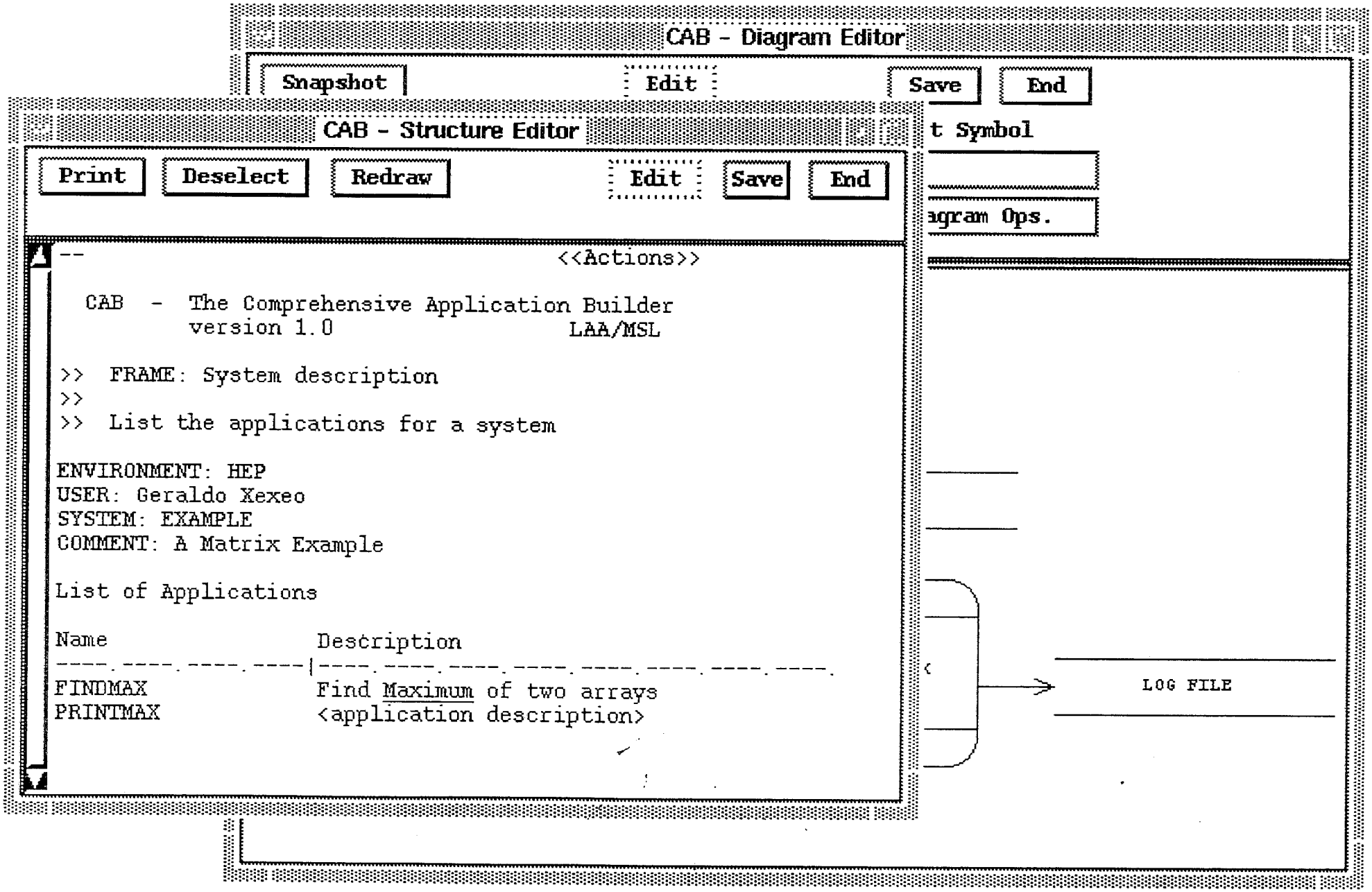
LIST OF SUBROUTINES:

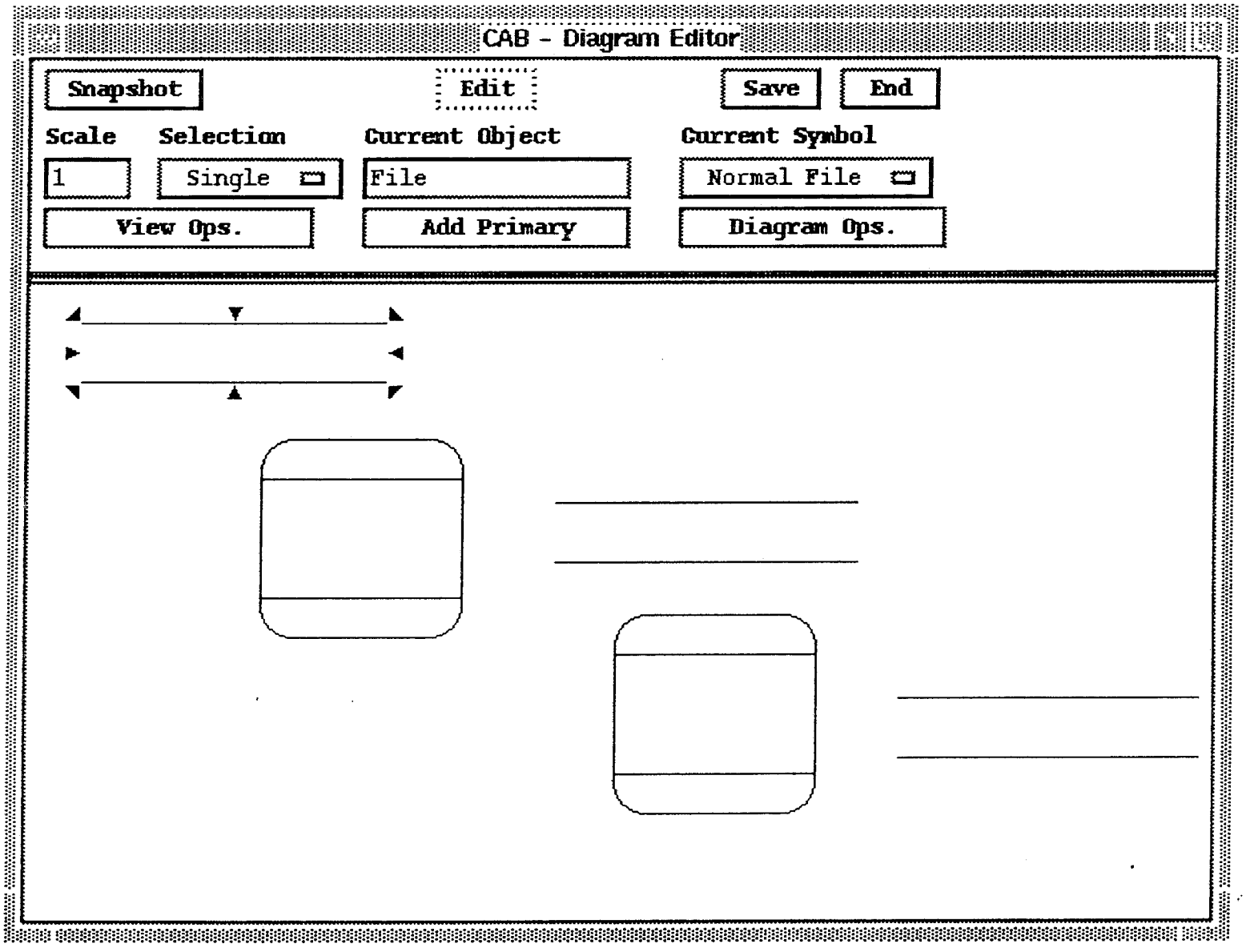
Name	Description
GENBOD	<subroutine description>
ADDMATRIX	<subroutine description>

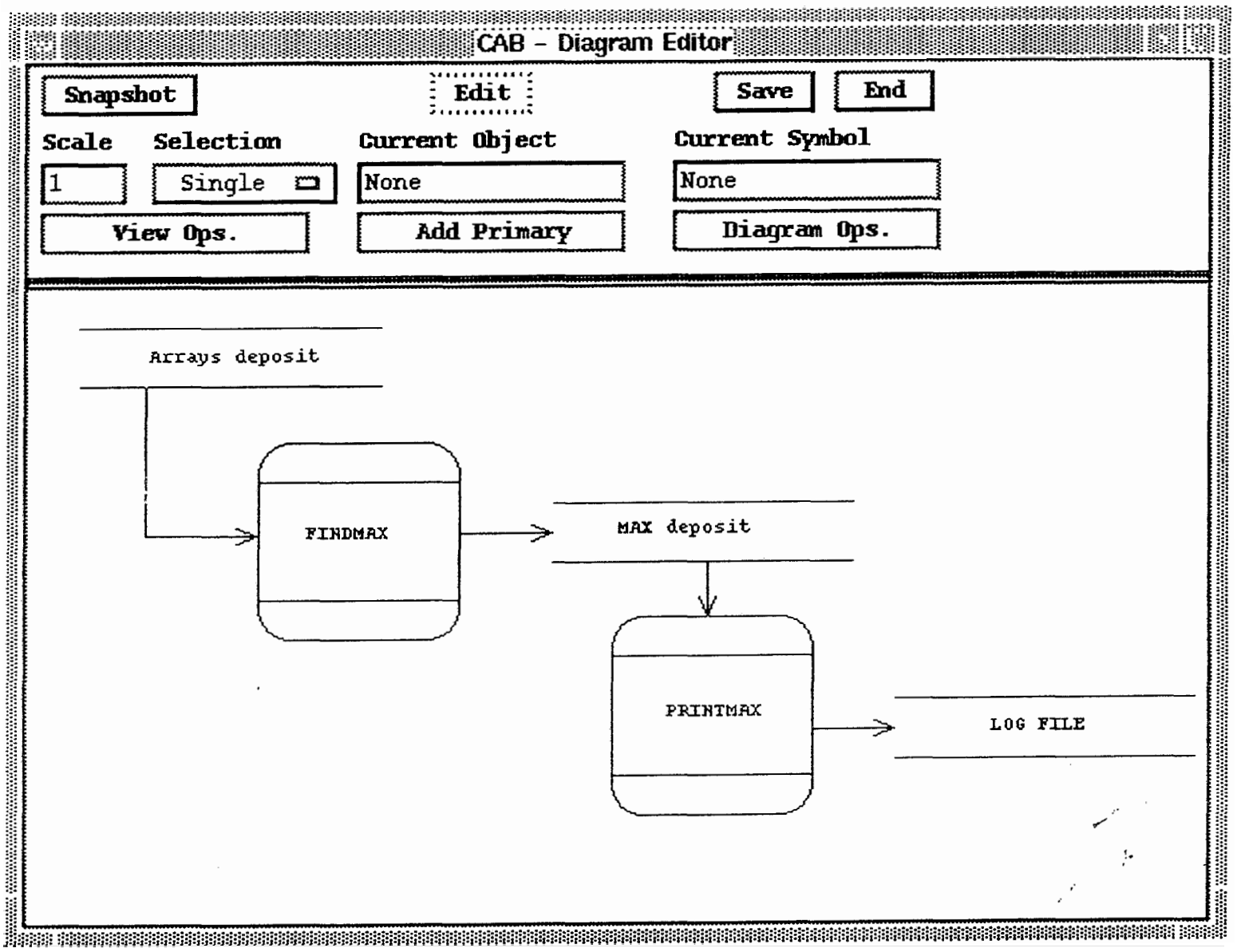

CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: ENVIROMENT DESCRIPTION
>>
>> Describes a environment
ENVIRONMENT: HEP
DESCRIPTION: An environment for High Energy Physics
LIST OF USERS:
Name
Description
Geraldo Xexeo
The Creator
Giuseppe La Commare
Main Collaborator
J. Random Hacker
Standard good user
Edgar Luser
Not a good user
LIST OF KEYWORDS:
Name
Description
Monte Carlo
uses Monte Carlo Technique
bodies
Deal with General Bodies
generation
Generate Something
phase space
Deal with Phase Space
Hadronization
Part of the Analysis chain
LIST OF PROGRAMS:
Name
Description
HERWIG 5.4
A Monte Carlo EG for hadronic physics
LIST OF SUBROUTINES
Name
Description
GENBOD
Generate Bodies
ADDMATRIX
Add two complex Matrix

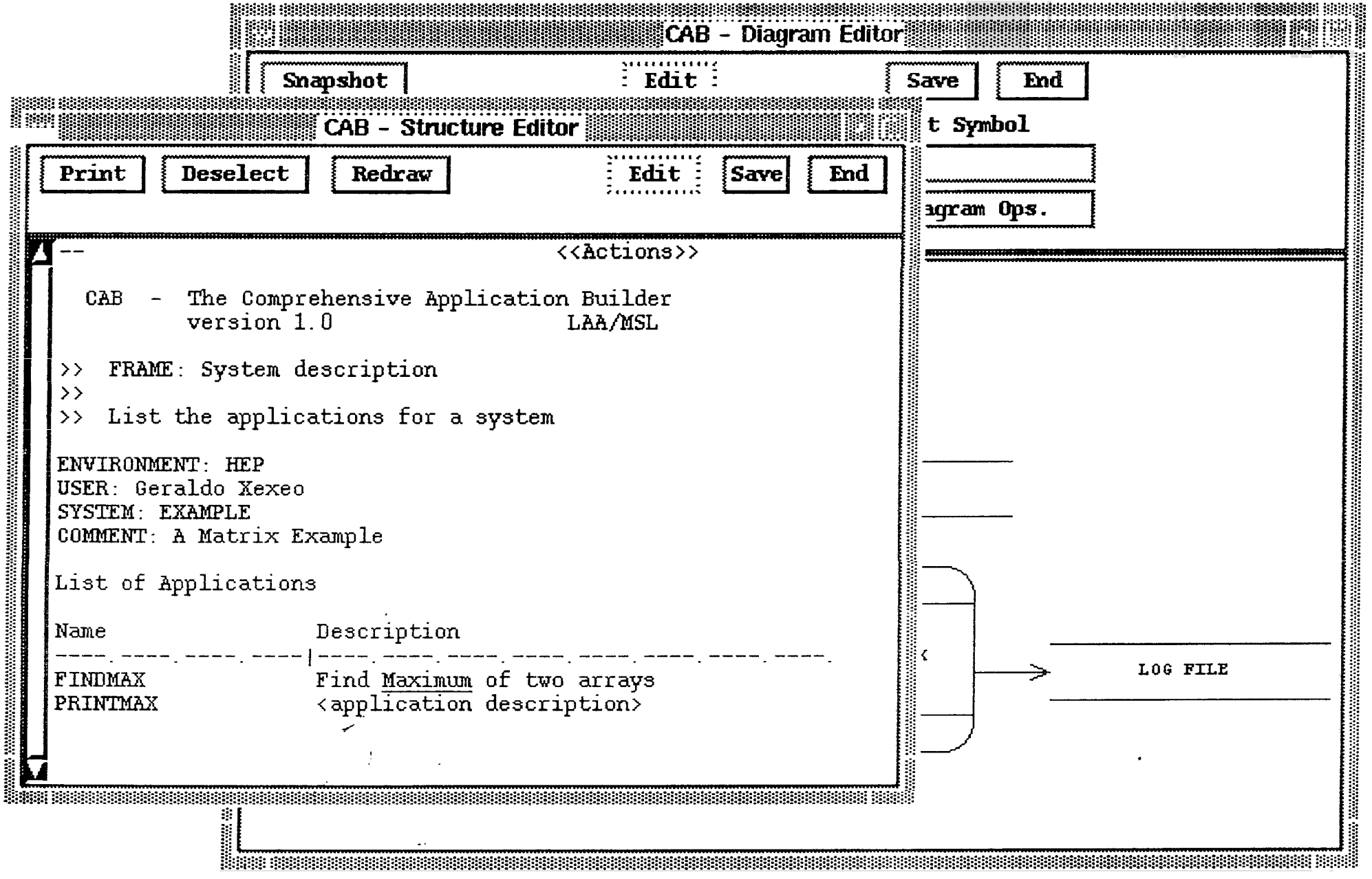

CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: LIST OF SYSTEMS
>>
>> List all systems for a previous selected user
ENVIRONMENT: HEP
USER: Geraldo Xexeo
COMMENTS: The Creator
LIST OF SYSTEMS:
Name
Description
NONE

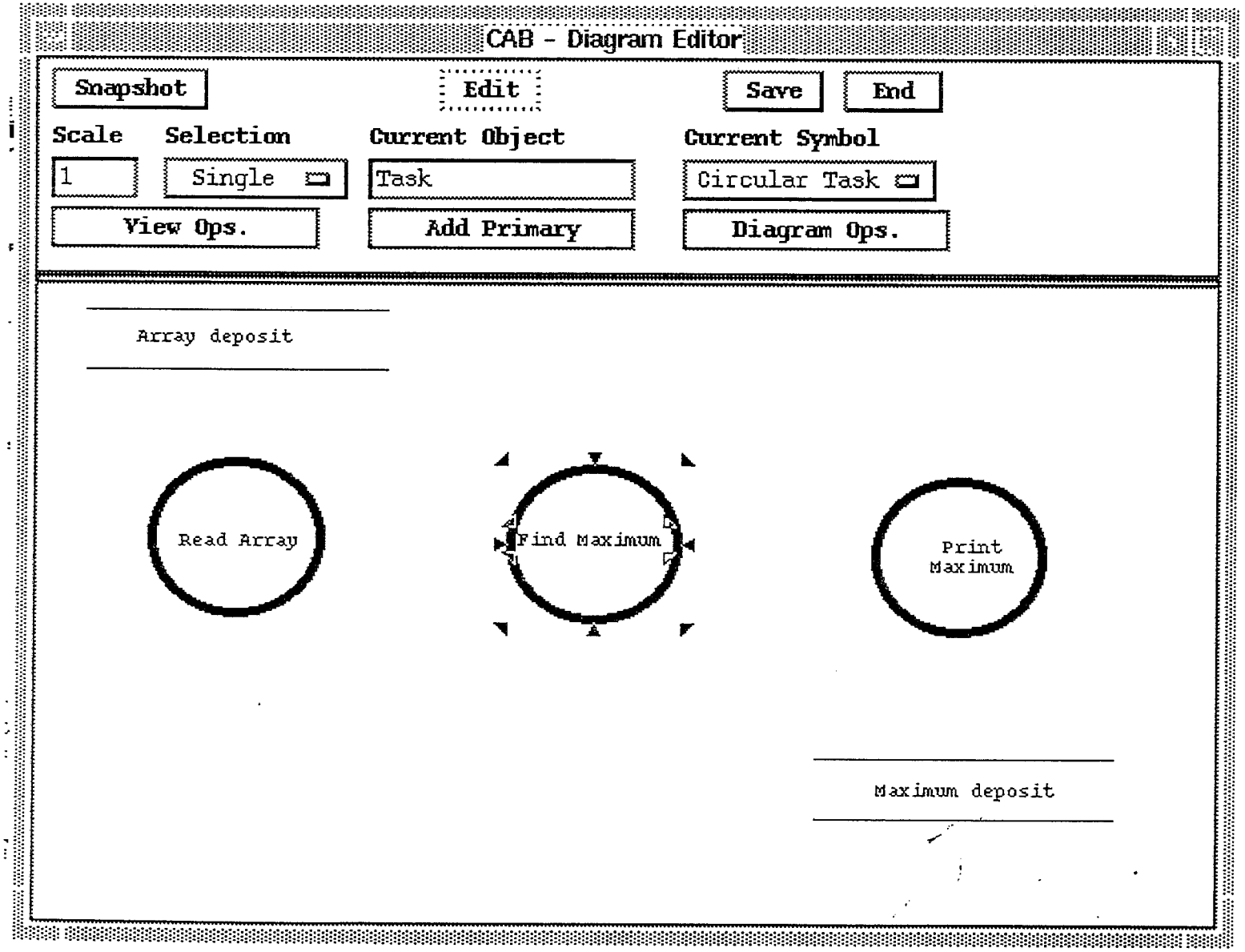

CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: LIST OF SYSTEMS
>>
>> List all systems for a previous selected user
ENVIRONMENT: HEP
USER: Geraldo Xexeo
COMMENTS: The Creator
LIST OF SYSTEMS:
Name
Description
EXAMPLE
A Matrix Example

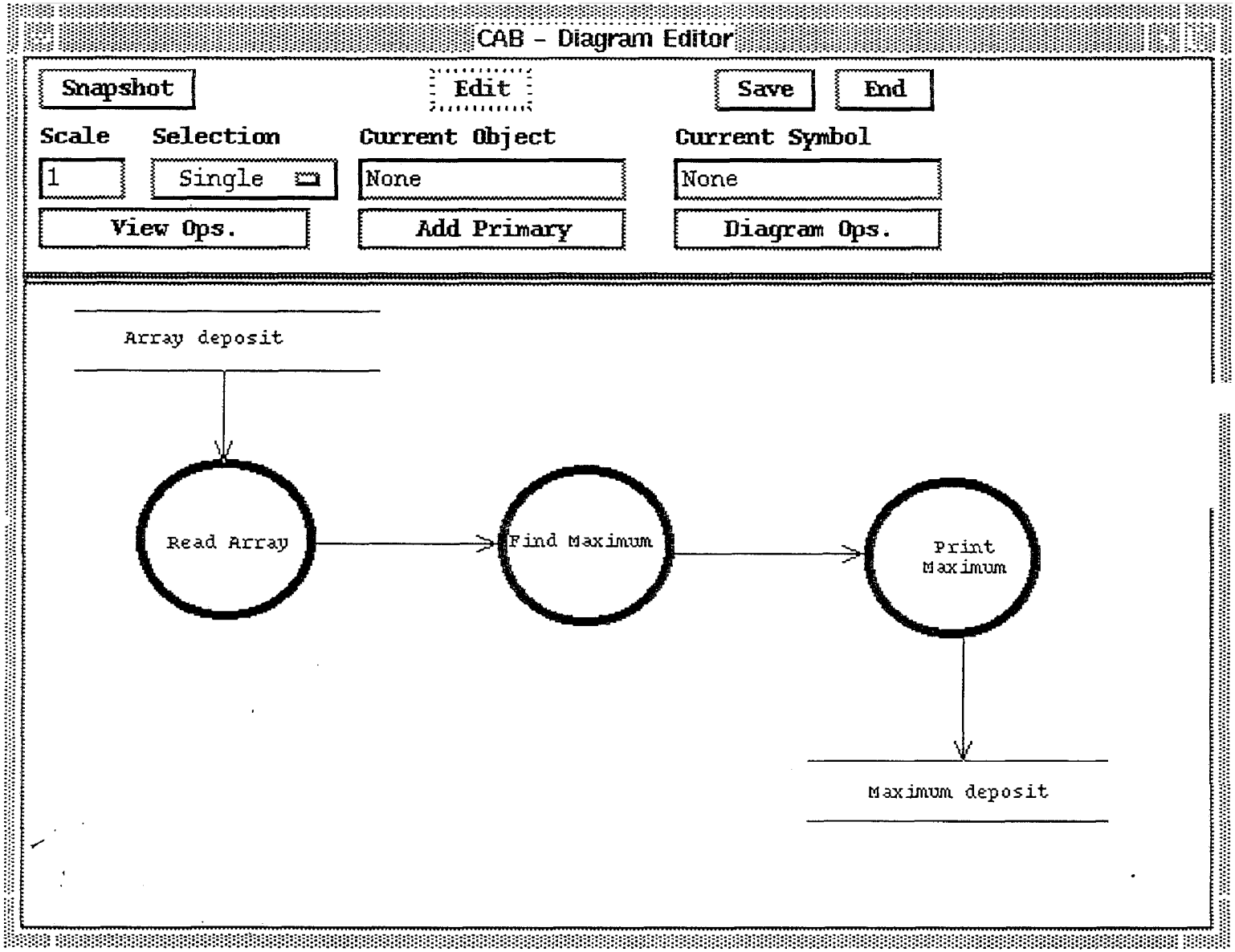

CAB - Diagram Editor
Snapshot
Edit
Save
End
t Symbol
agram Ops.
LOG FILE
CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: System description
>>
>> List the applications for a system
ENVIRONMENT: HEP
USER: Geraldo Xexeo
SYSTEM: EXAMPLE
COMMENT: A Matrix Example
List of Applications
Name
Description
FINDMAX
Find Maximum of two arrays
PRINTMAX
<application description>

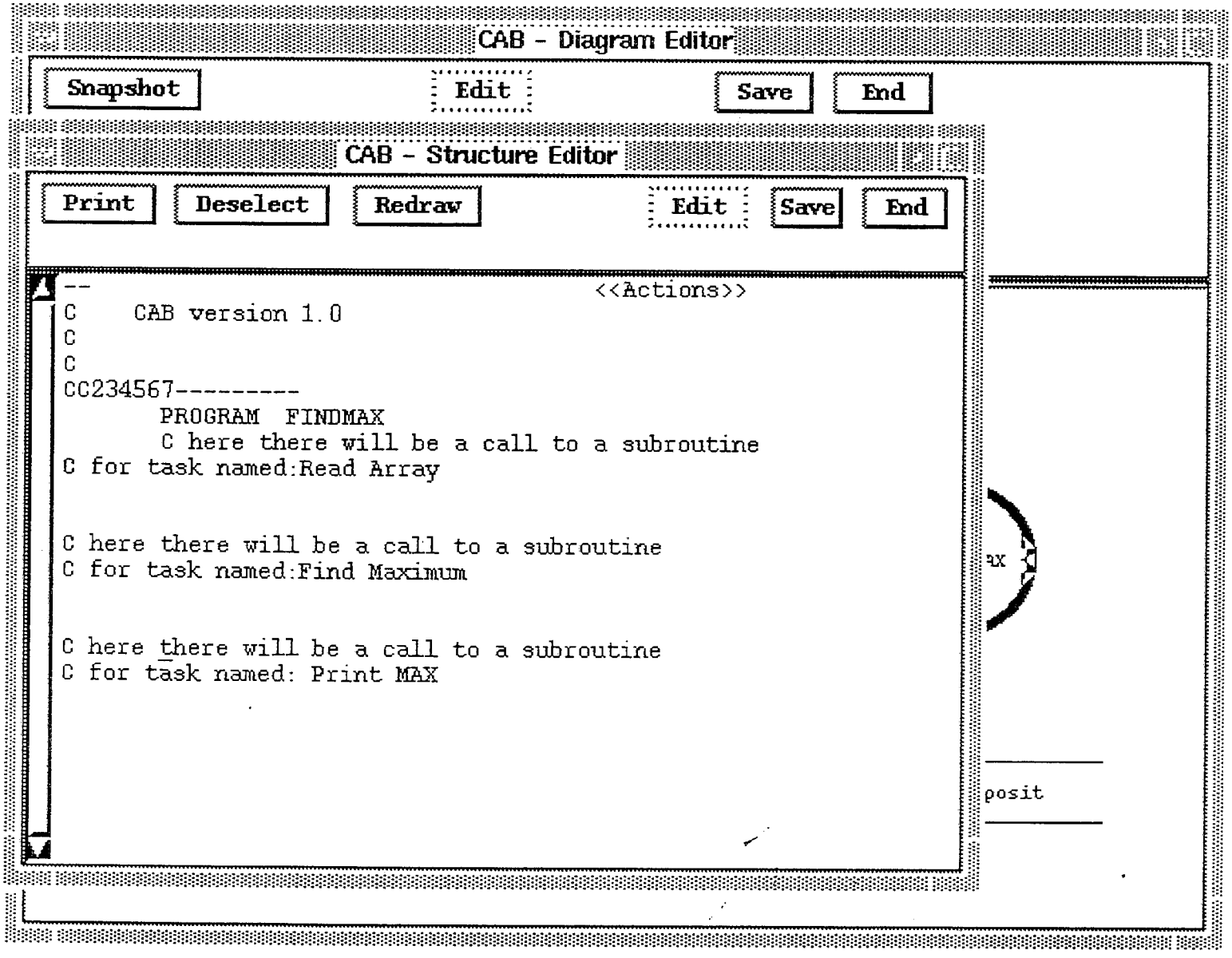

CAB - Diagram Editor
Snapshot
Edit
Save
End
Scale
Selection
Current Object
Current Symbol
1
Single
File
Normal File
View Ops.
Add Primary
Diagram Ops.

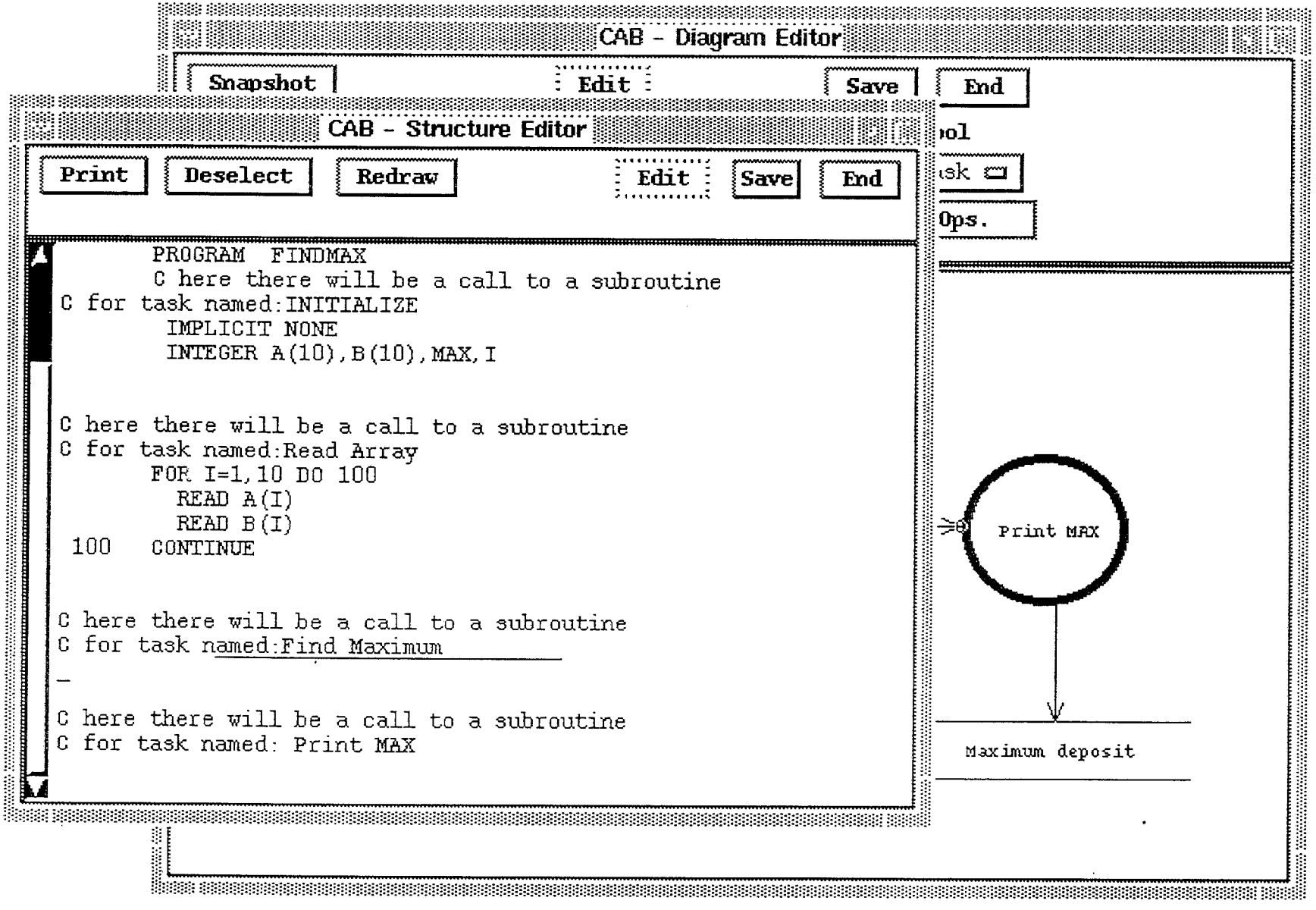

CAB - Diagram Editor
Snapshot
Edit
Save
End
Scale
Selection
Current Object
Current Symbol
1
Single
None
None
View Ops.
Add Primary
Diagram Ops.
Arrays deposit
FINDMAX
MAX deposit
PRINTMAX
LOG FILE

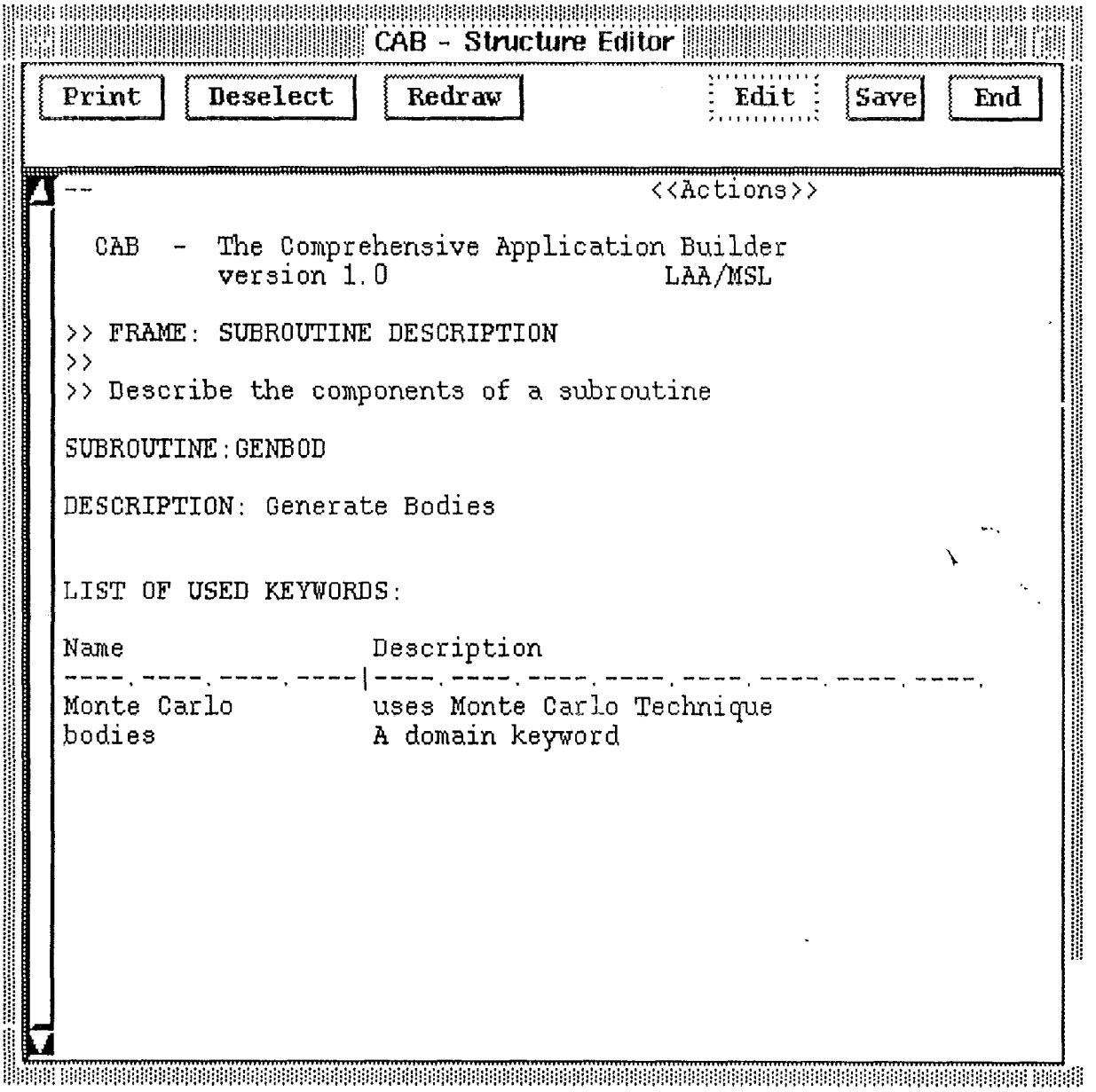

CAB - Diagram Editor
Snapshot
Edit
Save
End
t Symbol
agram Ops.
CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: System description
>>
>> List the applications for a system
ENVIRONMENT: HEP
USER: Geraldo Xexeo
SYSTEM: EXAMPLE
COMMENT: A Matrix Example
List of Applications
Name
Description
FINDMAX
Find Maximum of two arrays
PRINTMAX
<application description>
LOG FILE

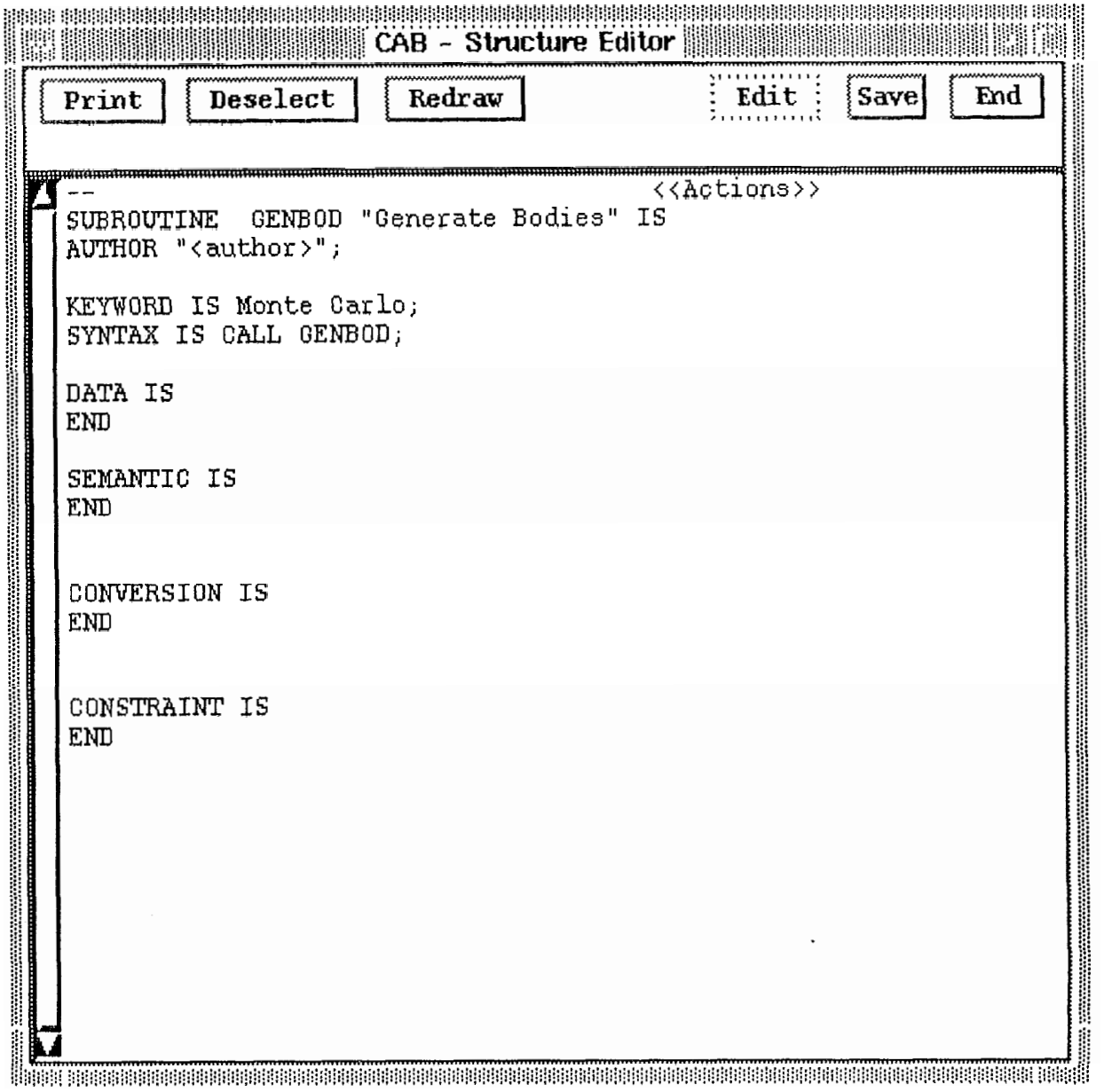

CAB - Diagram Editor
Snapshot
Edit
Save
End
Scale
Selection
Current Object
Current Symbol
1
Single
Task
Circular Task
View Ops.
Add Primary
Diagram Ops.
Array deposit
Read Array
Find Maximum
Print Maximum
Maximum deposit

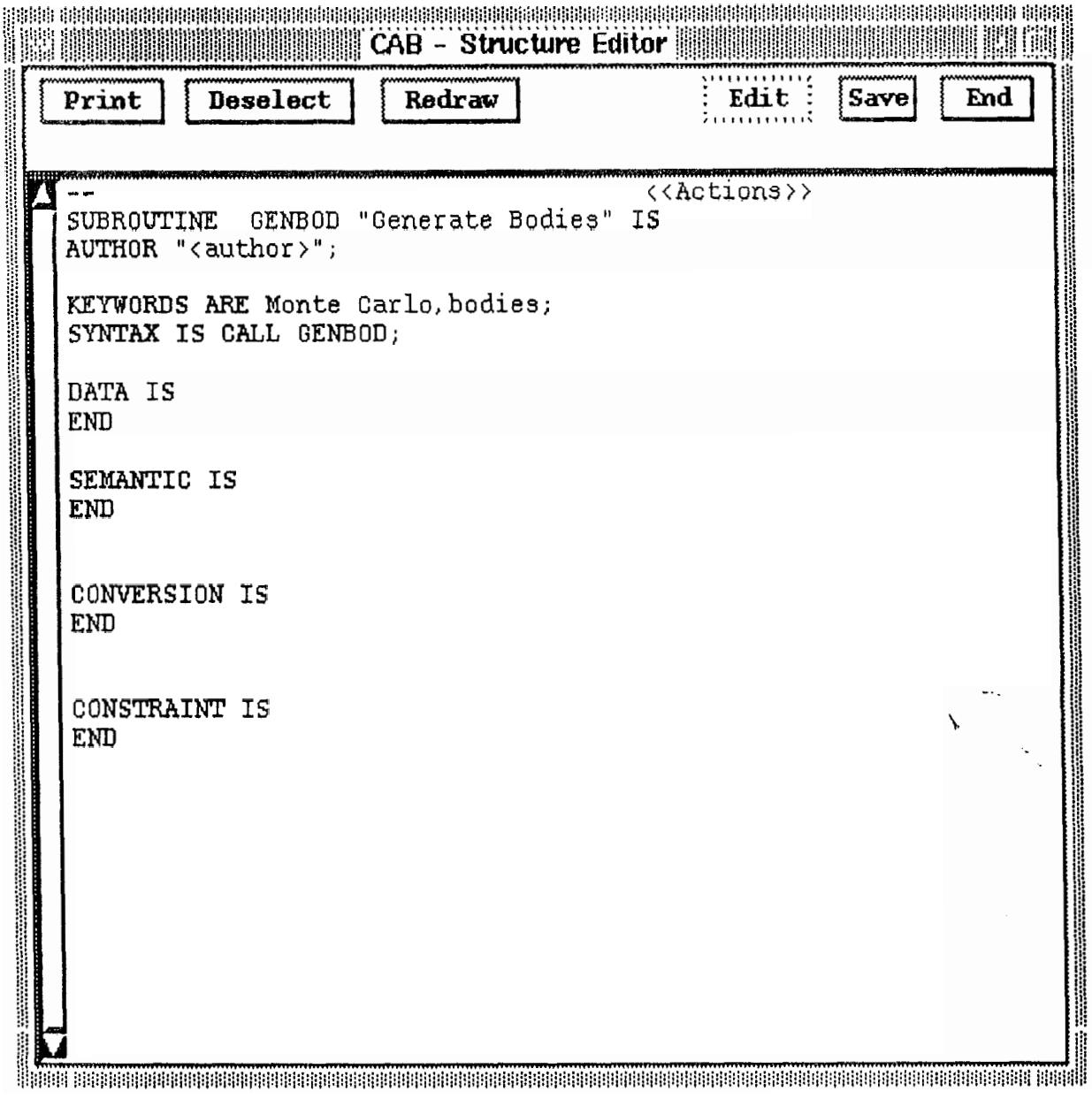

CAB - Diagram Editor
Snapshot
Edit
Save
End
Scale
Selection
Current Object
Current Symbol
1
Single
None
None
View Ops.
Add Primary
Diagram Ops.
Array deposit
Read Array
Find Maximum
Print Maximum
Maximum deposit

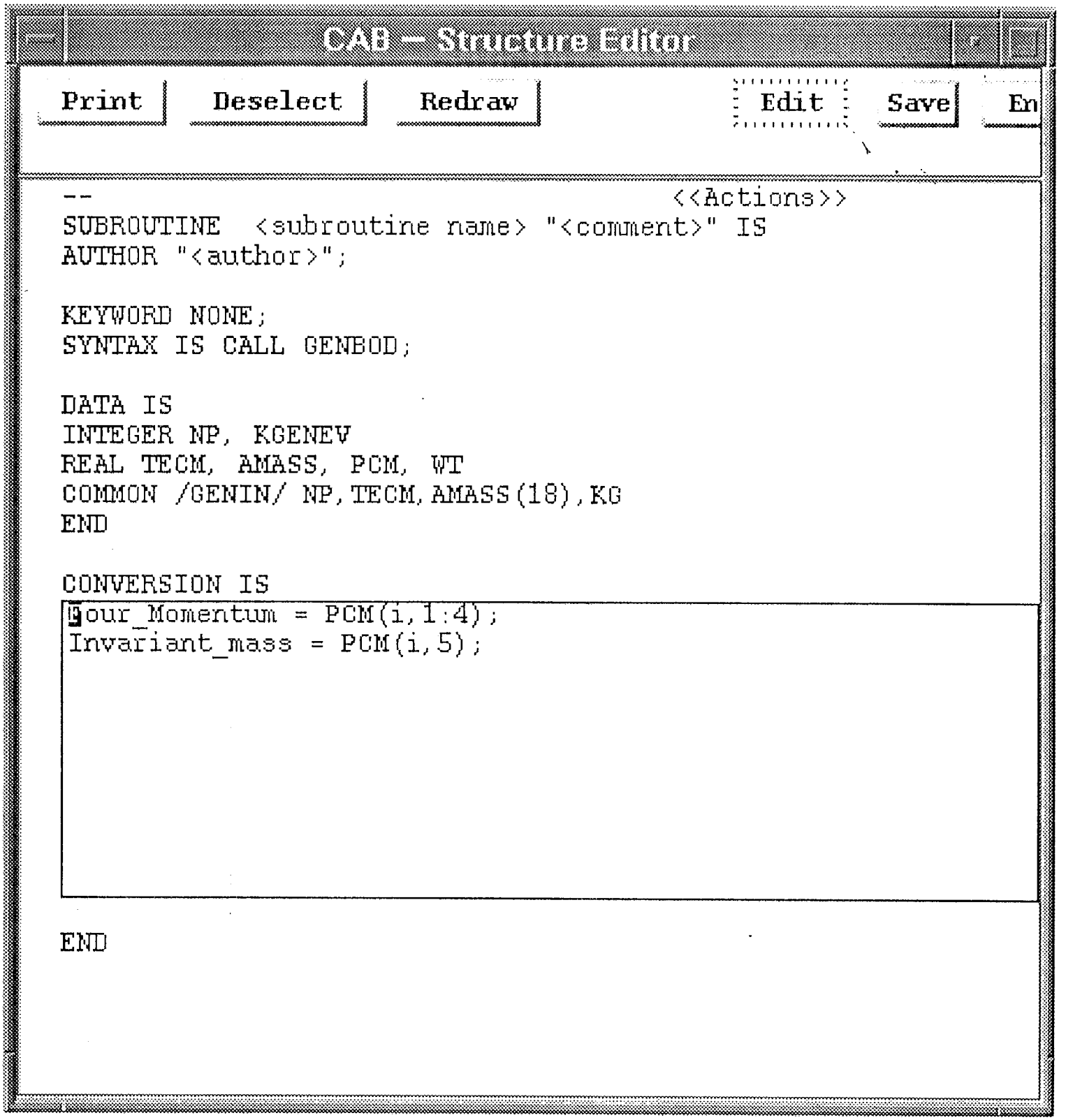

CAB - Diagram Editor
Snapshot
Edit
Save
End
CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
C CAB version 1.0
C
C
CC234567---------
PROGRAM FINDMAX
C here there will be a call to a subroutine
C for task named:Read Array
C here there will be a call to a subroutine
C for task named:Find Maximum
C here there will be a call to a subroutine
C for task named: Print MAX
posit

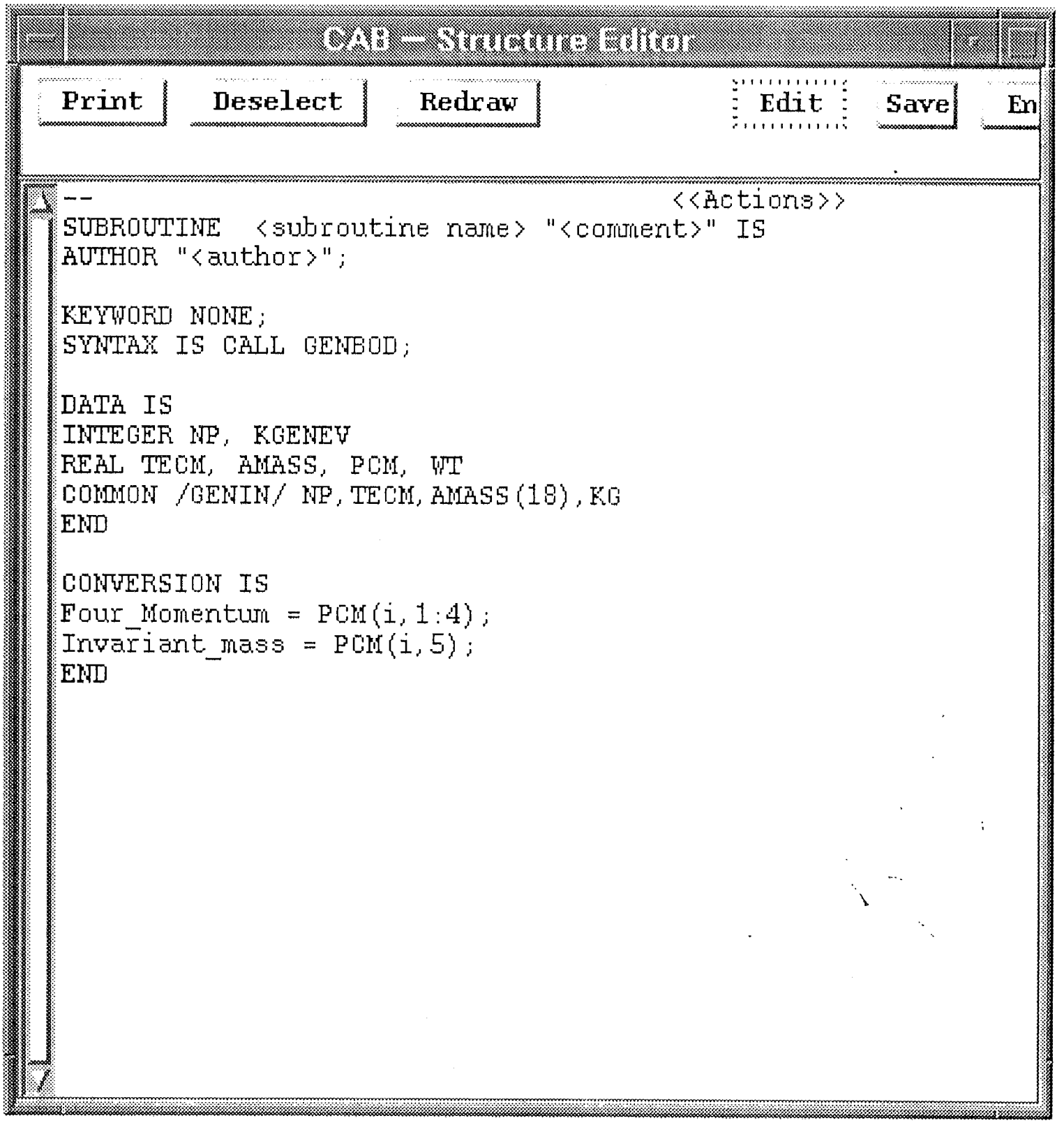

CAB - Diagram Editor
Snapshot
Edit
Save
End
CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
ol
sk
Ops.
PROGRAM FINDMAX
C here there will be a call to a subroutine
C for task named:INITIALIZE
IMPLICIT NONE
INTEGER A(10),B(10),MAX,I
C here there will be a call to a subroutine
C for task named:Read Array
FOR I=1,10 DO 100
READ A(I)
READ B(I)
100 CONTINUE
C here there will be a call to a subroutine
C for task named:Find Maximum
C here there will be a call to a subroutine
C for task named: Print MAX
Print MAX
Maximum deposit

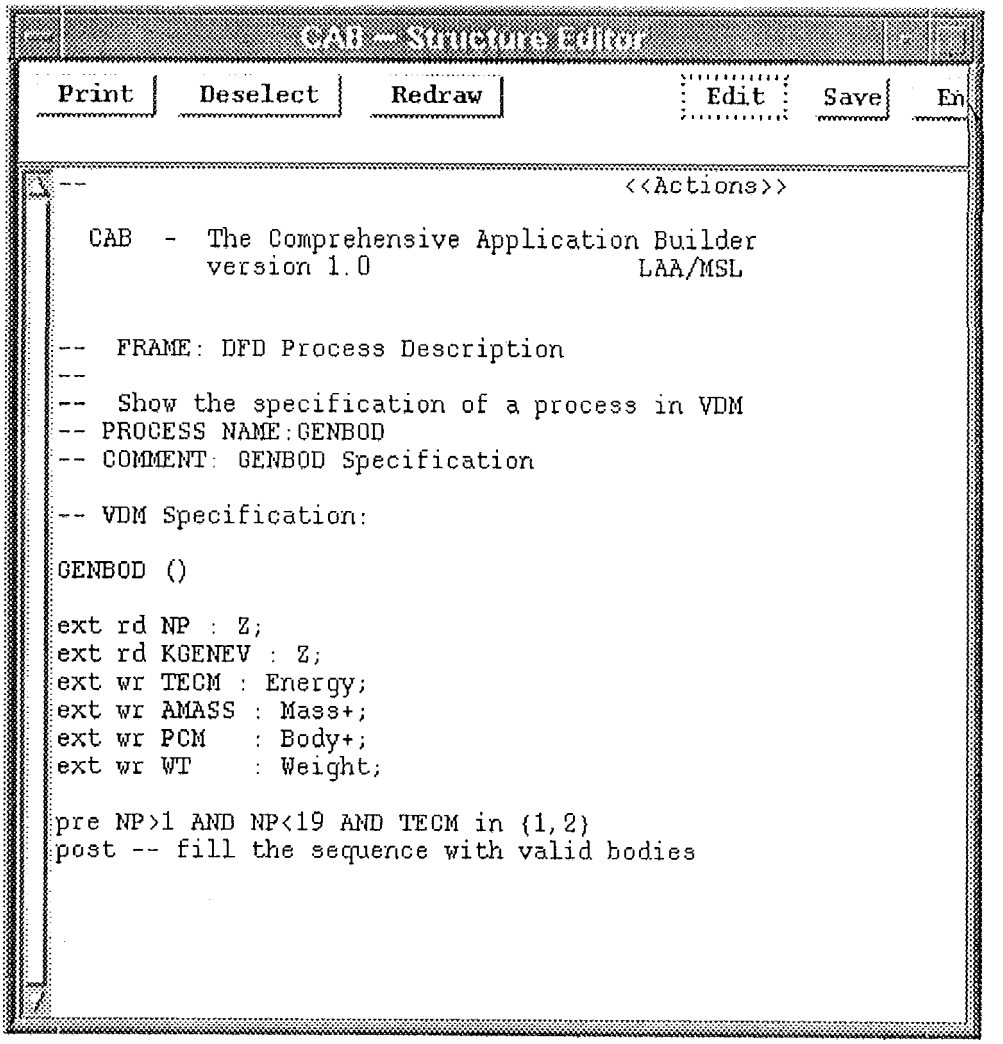

CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: SUBROUTINE DESCRIPTION
>>
>> Describe the components of a subroutine
SUBROUTINE:GENBOD
DESCRIPTION: Generate Bodies
LIST OF USED KEYWORDS:
Name
Description
Monte Carlo
uses Monte Carlo Technique
bodies
A domain keyword

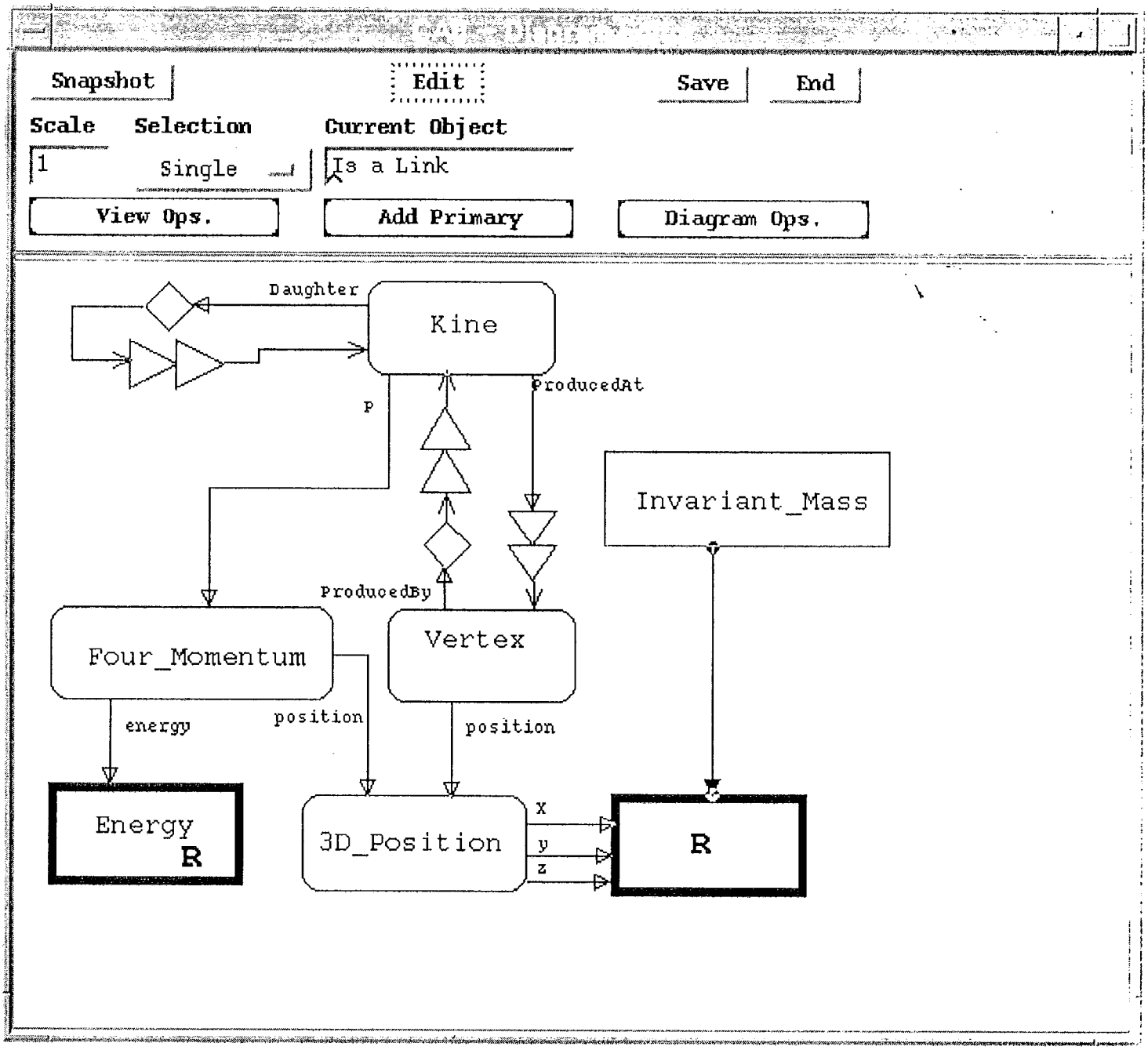

CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
SUBROUTINE GENBOD "Generate Bodies" IS
AUTHOR "<author>";
KEYWORD IS Monte Carlo;
SYNTAX IS CALL GENBOD;
DATA IS
END
SEMANTIC IS
END
CONVERSION IS
END
CONSTRAINT IS
END

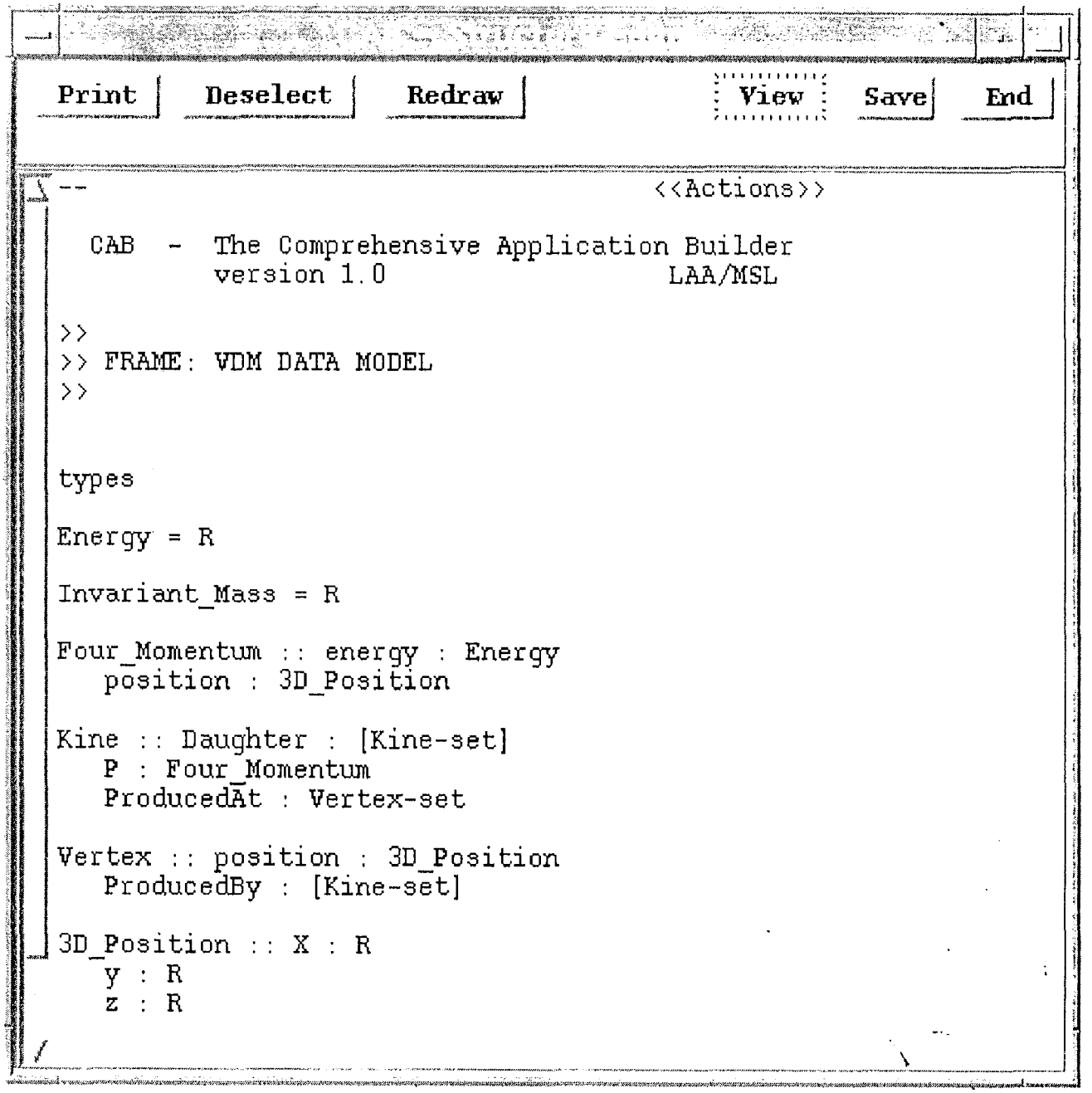

CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
SUBROUTINE GENBOD "Generate Bodies" IS
AUTHOR "<author>";
KEYWORDS ARE Monte Carlo,bodies;
SYNTAX IS CALL GENBOD;
DATA IS
END
SEMANTIC IS
END
CONVERSION IS
END
CONSTRAINT IS
END

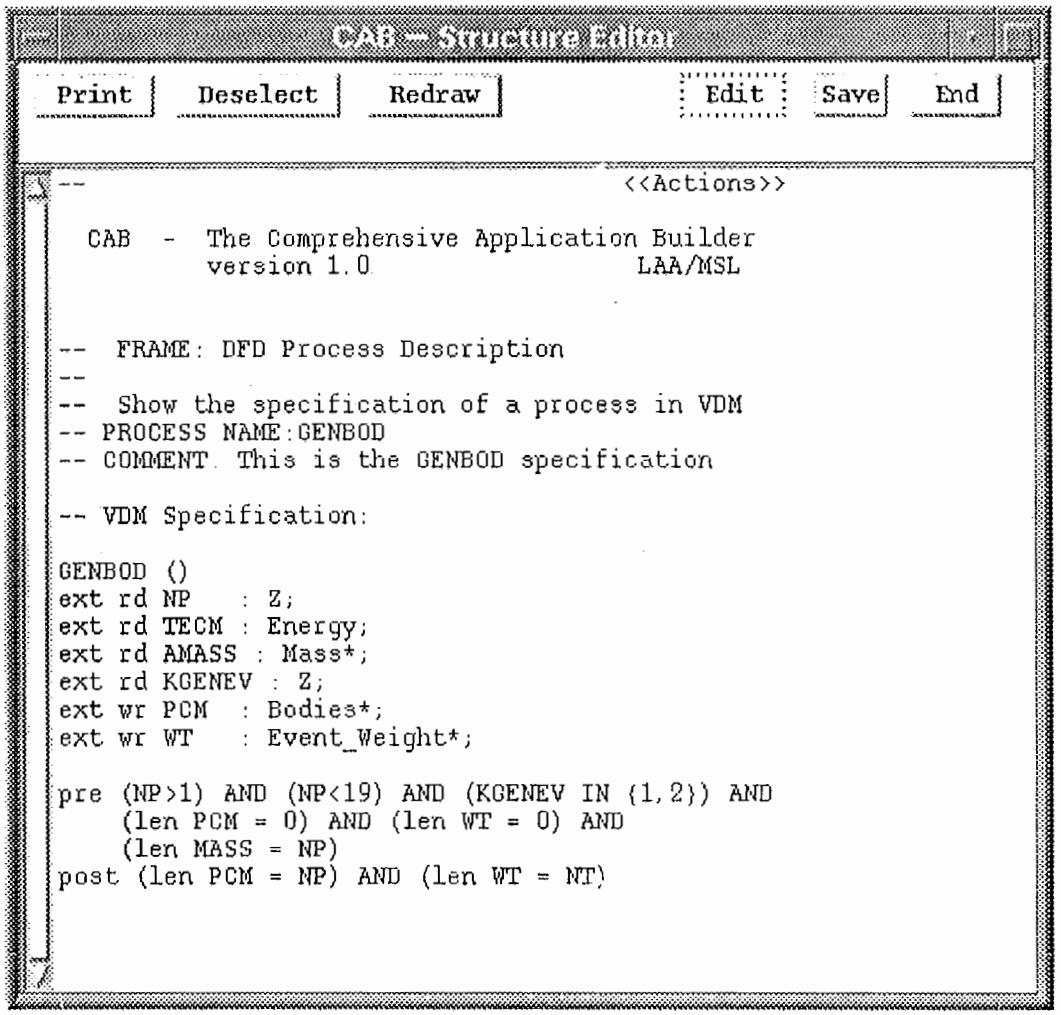

CAB — Structure Editor
Print
Deselect
Redraw
Edit
Save
En
--
<<Actions>>
SUBROUTINE <subroutine name> "<comment>" IS
AUTHOR "<author>";
KEYWORD NONE;
SYNTAX IS CALL GENBOD;
DATA IS
INTEGER NP, KGENEV
REAL TECM, AMASS, PCM, WT
COMMON /GENIN/ NP,TECM,AMASS(18),KG
END
CONVERSION IS
Four_Momentum = PCM(i,1:4);
Invariant_mass = PCM(i,5);
END

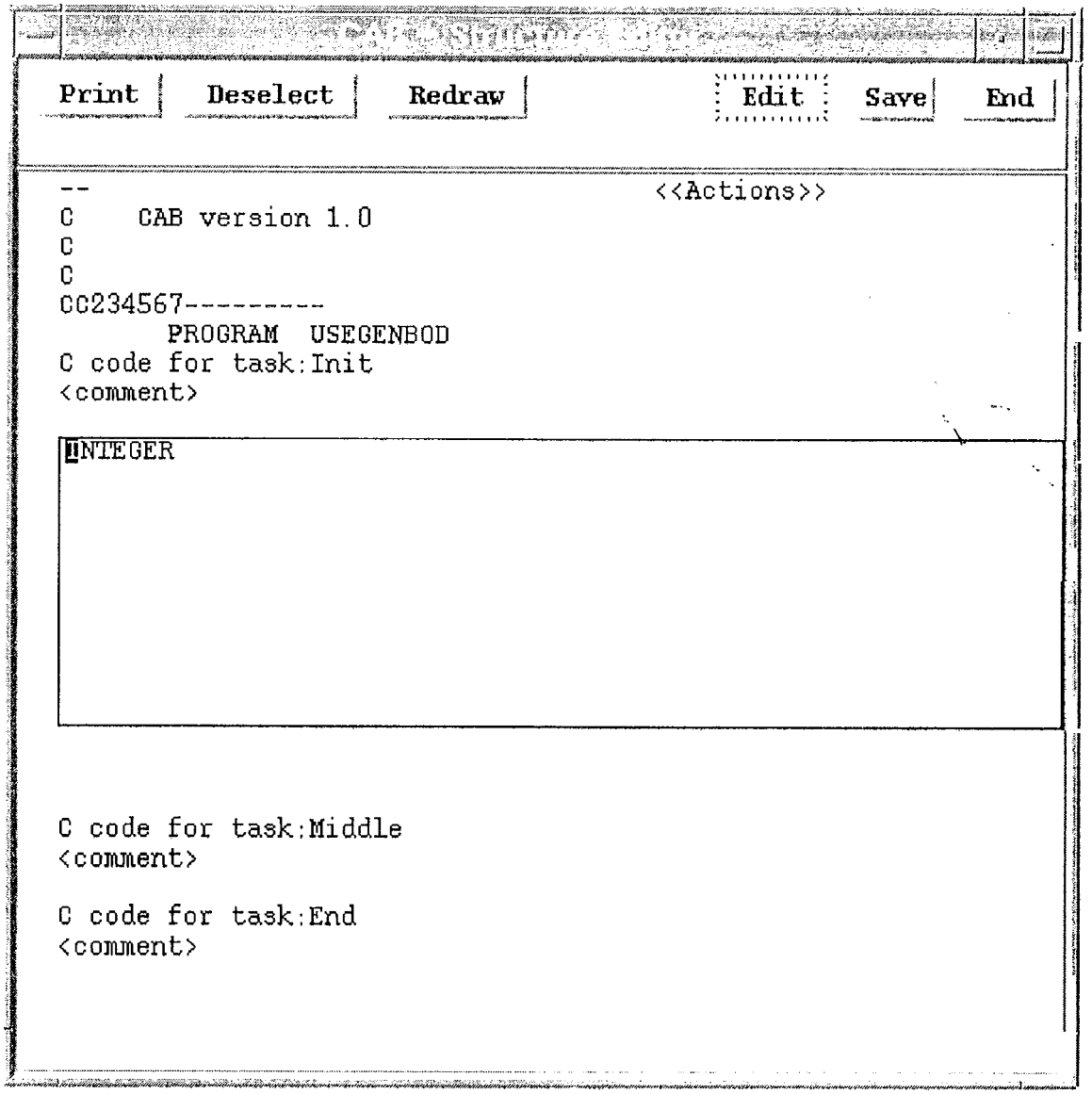

CAB — Structure Editor
Print
Deselect
Redraw
Edit
Save
En
-- <<Actions>>
SUBROUTINE <subroutine name> "<comment>" IS
AUTHOR "<author>";
KEYWORD NONE;
SYNTAX IS CALL GENBOD;
DATA IS
INTEGER NP, KGENEV
REAL TECM, AMASS, PCM, WT
COMMON /GENIN/ NP,TECM,AMASS(18),KG
END
CONVERSION IS
Four_Momentum = PCM(i,1:4);
Invariant_mass = PCM(i,5);
END

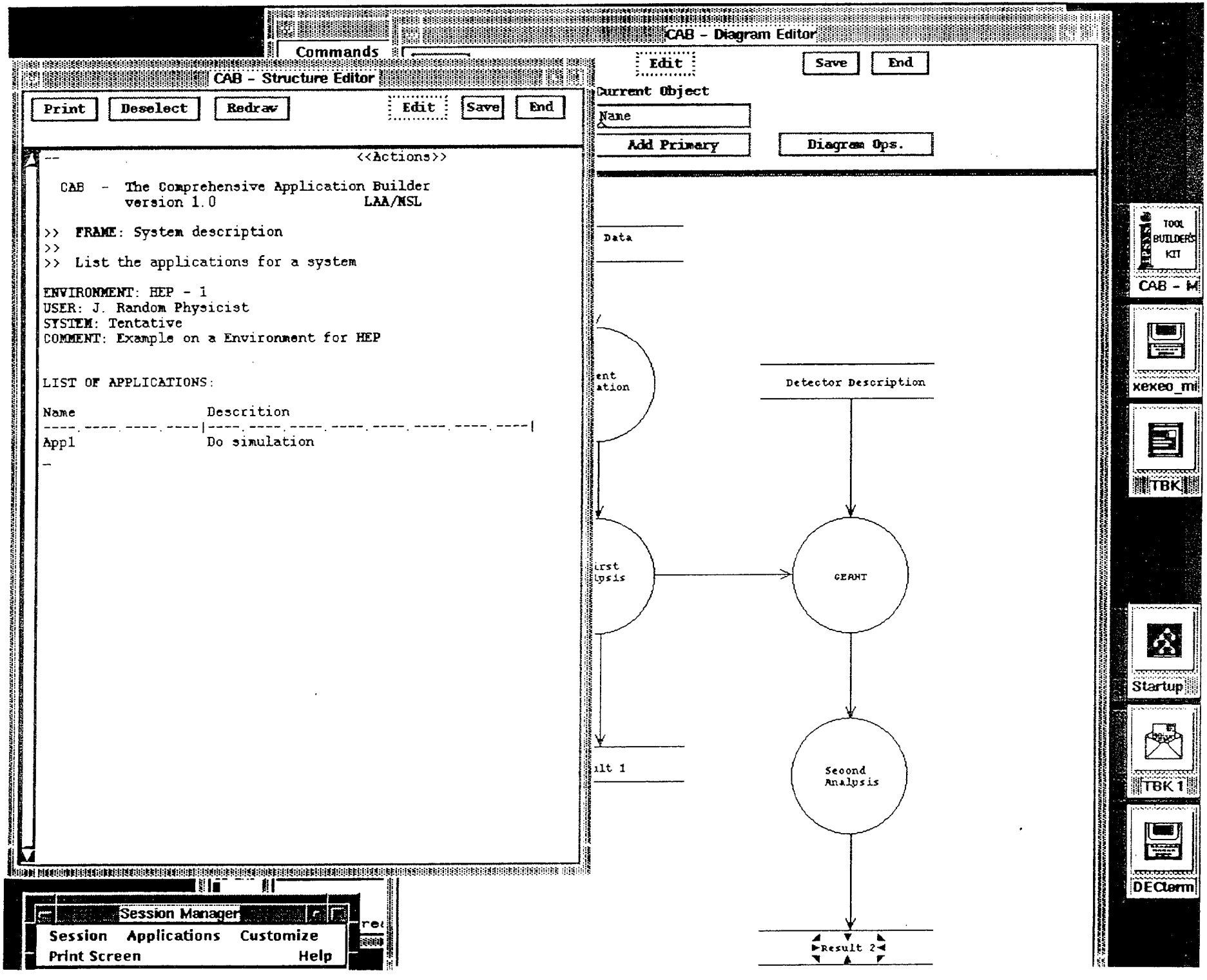

CAB — Structure Editor
Print
Deselect
Redraw
Edit
Save
En
-- <<Actions>>
CAB - The Comprehensive Application Builder
version 1.0 LAA/MSL
-- FRAME: DFD Process Description
--
-- Show the specification of a process in VDM
-- PROCESS NAME:GENBOD
-- COMMENT: GENBOD Specification
-- VDM Specification:
GENBOD ()
ext rd NP : Z;
ext rd KGENEV : Z;
ext wr TECM : Energy;
ext wr AMASS : Mass+;
ext wr PCM : Body+;
ext wr WT : Weight;
pre NP>1 AND NP<19 AND TECM in {1,2}
post -- fill the sequence with valid bodies

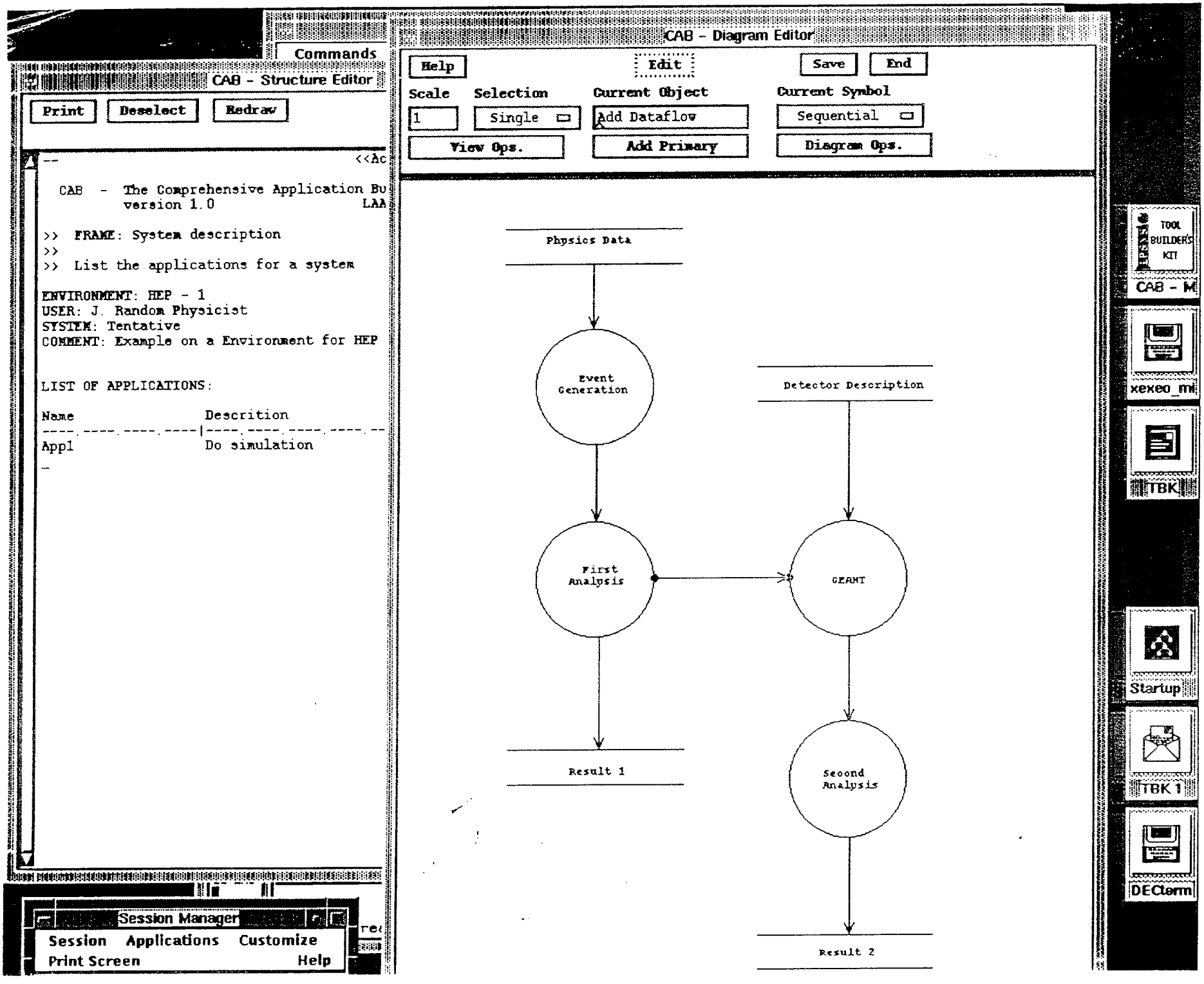

Snapshot
Edit
Save
End
Scale
Selection
Current Object
1
Single
Is a Link
View Ops.
Add Primary
Diagram Ops.
Daughter
Kine
ProducedAt
P
Invariant_Mass
ProducedBy
Four_Momentum
Vertex
energy
position
position
Energy
R
3D_Position
x
y
z
R

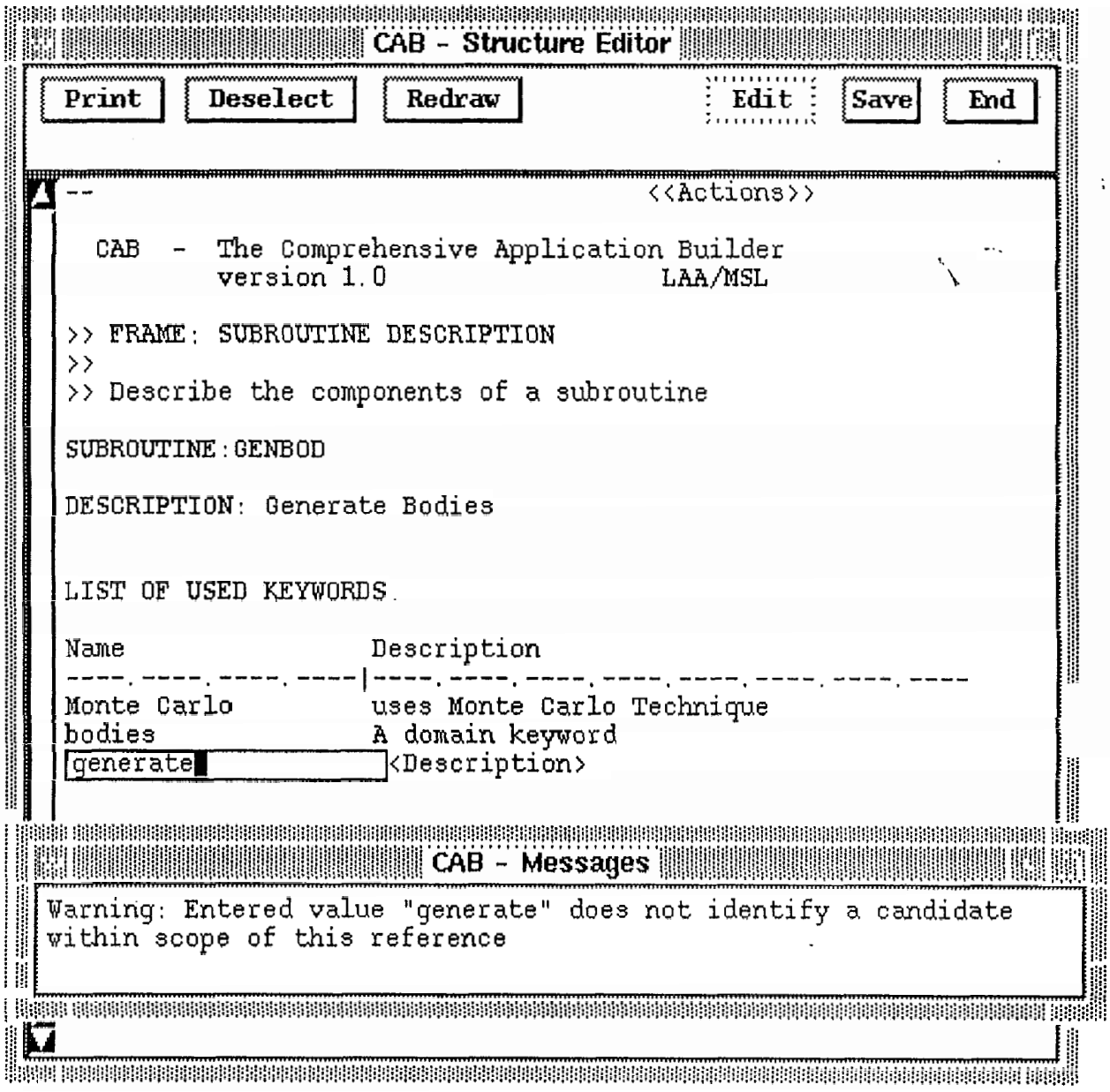

Print
Deselect
Redraw
View
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>>
>> FRAME: VDM DATA MODEL
>>
types
Energy = R
Invariant_Mass = R
Four_Momentum :: energy : Energy
position : 3D_Position
Kine :: Daughter : [Kine-set]
P : Four_Momentum
ProducedAt : Vertex-set
Vertex :: position : 3D_Position
ProducedBy : [Kine-set]
3D_Position :: X : R
y : R
z : R

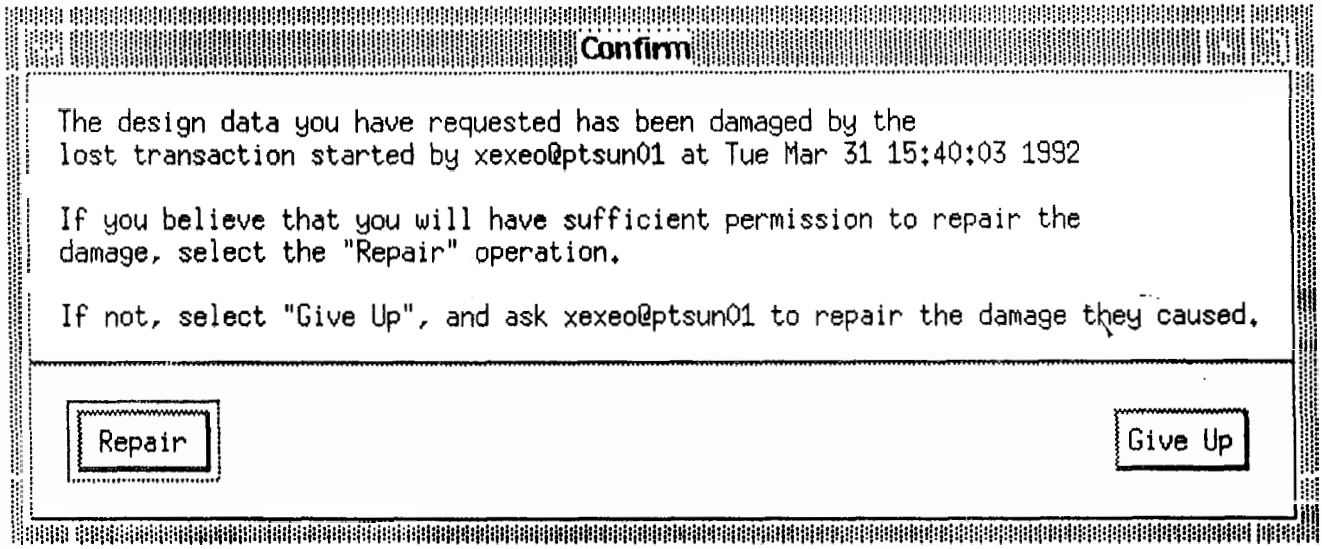

CAB — Structure Editor
Print
Deselect
Redraw
Edit
Save
End
-- <<Actions>>
CAB - The Comprehensive Application Builder
version 1.0 LAA/MSL
-- FRAME: DFD Process Description
--
-- Show the specification of a process in VDM
-- PROCESS NAME:GENBOD
-- COMMENT: This is the GENBOD specification
-- VDM Specification:
GENBOD ()
ext rd NP : Z;
ext rd TECM : Energy;
ext rd AMASS : Mass*;
ext rd KGENEV : Z;
ext wr PCM : Bodies*;
ext wr WT : Event_Weight*;
pre (NP>1) AND (NP<19) AND (KGENEV IN {1,2}) AND
(len PCM = 0) AND (len WT = 0) AND
(len MASS = NP)
post (len PCM = NP) AND (len WT = NT)


Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
C CAB version 1.0
C
C
CC234567---------
PROGRAM USEGENBOD
C code for task:Init
<comment>
INTEGER
C code for task:Middle
<comment>
C code for task:End
<comment>


Commands
CAB - Diagram Editor
CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0 LAA/NSL
>> FRAME: System description
>>
>> List the applications for a system
ENVIRONMENT: HEP - 1
USER: J. Random Physicist
SYSTEM: Tentative
COMMENT: Example on a Environment for HEP
LIST OF APPLICATIONS:
Name Descrition
App1 Do simulation
Edit
Save
End
Current Object
Name
Add Primary
Diagram Ops.
Data
Detector Description
GEANT
Seoond Analysis
Result 2
Session Manager
Session Applications Customize
Print Screen Help
TOOL BUILDERS KIT
CAB - M
xexeo_mi
TBK
Startup
TBK 1
DECterm

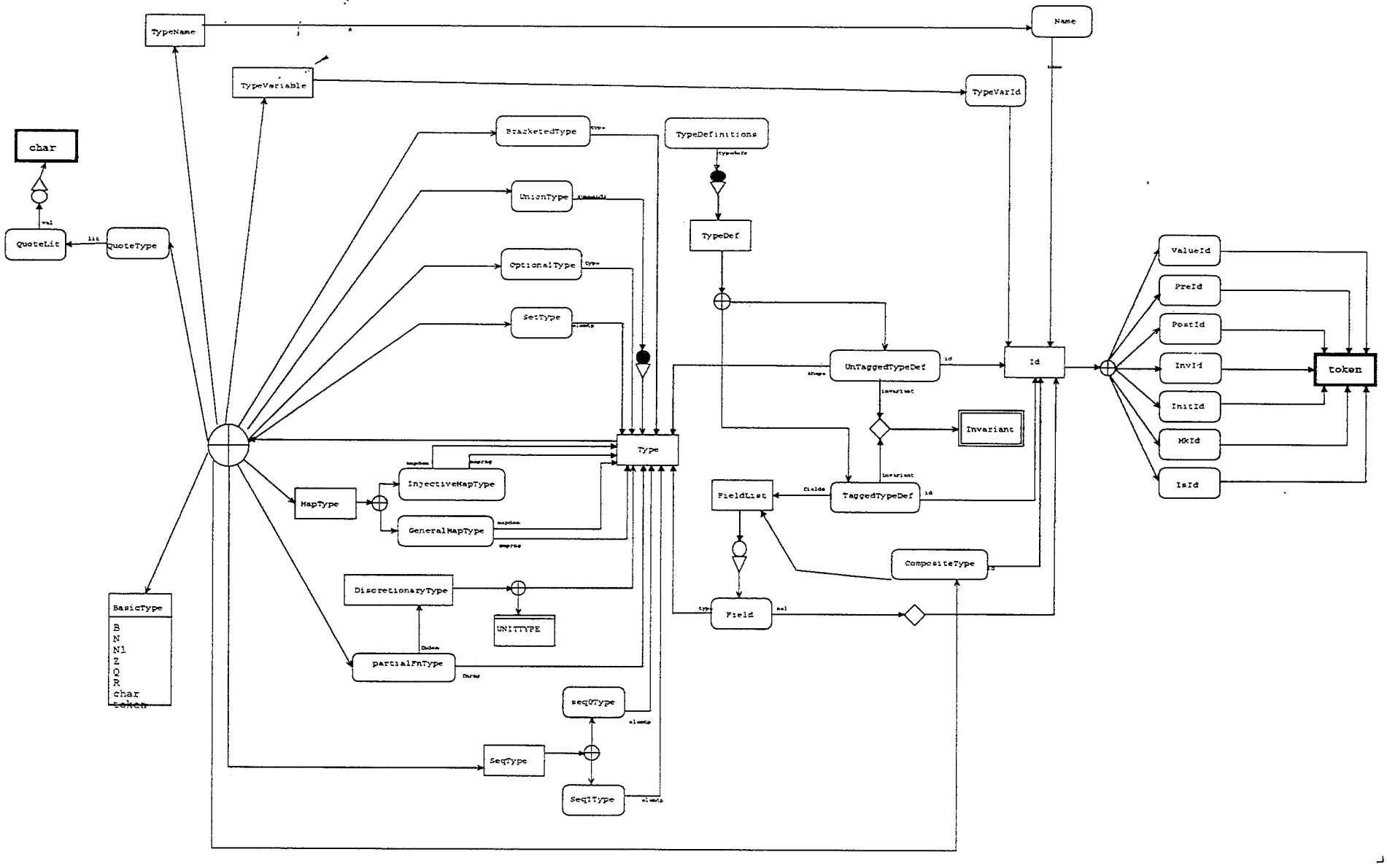

Commands
CAB - Structure Editor
Print
Deselect
Redraw
<<Ac
CAB - The Comprehensive Application Bu
version 1.0 LAA
>> FRAME: System description
>>
>> List the applications for a system
ENVIRONMENT: HEP - 1
USER: J. Random Physicist
SYSTEM: Tentative
COMMENT: Example on a Environment for HEP
LIST OF APPLICATIONS:
Name Descrition
Appl Do simulation
Session Manager
Session Applications Customize
Print Screen Help
CAB - Diagram Editor
Help
Edit
Save
End
Scale
Selection
Current Object
Current Symbol
1
Single
Add Dataflow
Sequential
View Ops.
Add Primary
Diagram Ops.
Physics Data
Event Generation
Detector Description
First Analysis
GEANT
Result 1
Seoond Analysis
Result 2
TOOL BUILDER'S KIT
CAB - M
xexeo_mi
TBK
Startup
TBK 1
DECterm


CAB - Structure Editor
Print
Deselect
Redraw
Edit
Save
End
--
<<Actions>>
CAB - The Comprehensive Application Builder
version 1.0
LAA/MSL
>> FRAME: SUBROUTINE DESCRIPTION
>>
>> Describe the components of a subroutine
SUBROUTINE:GENBOD
DESCRIPTION: Generate Bodies
LIST OF USED KEYWORDS.
Name
Description
Monte Carlo
uses Monte Carlo Technique
bodies
A domain keyword
generate
<Description>
CAB - Messages
Warning: Entered value "generate" does not identify a candidate
within scope of this reference


Confirm
The design data you have requested has been damaged by the
lost transaction started by xexeo@ptsun01 at Tue Mar 31 15:40:03 1992
If you believe that you will have sufficient permission to repair the
damage, select the "Repair" operation.
If not, select "Give Up", and ask xexeo@ptsun01 to repair the damage they caused.
Repair
Give Up

APÊNDICE II: EXEMPLO DE VDM/GDL

Partial model of VDM/SL in VDM/SL-G



Section 11

The Outer Abstract Syntax

The proposed abstract syntax for the BSI VDM specification language is defined using a subset of BSI/VDM SL type equations.

11.1 Document

$Document = DefinitionBlock^+$

$$\begin{aligned} DefinitionBlock = {} & TypeDefinitions \mid \\ & StateDef \mid \\ & ValueDefinitions \mid \\ & FunctionDefinitions \mid \\ & OperationDefinitions \end{aligned}$$

11.2 Definitions

11.2.1 Type Definitions

$TypeDefinitions :: typedefs : TypeDef^+$

$TypeDef = UnTaggedTypeDef \mid TaggedTypeDef$

$$\begin{aligned} UnTaggedTypeDef :: {} & id : Id \\ & shape : Type \\ & typeinv : [Invariant] \end{aligned}$$

$$\begin{aligned} TaggedTypeDef :: {} & id : Id \\ & fields : FieldList \\ & typeinv : [Invariant] \end{aligned}$$

$$\begin{aligned} Type = {} & BracketedType \mid BasicType \mid QuoteType \mid CompositeType \mid UnionType \mid \\ & ProductType \mid OptionalType \mid SetType \mid SeqType \mid MapType \mid PartialFnType \mid \\ & TypeName \mid TypeVar \end{aligned}$$

$BracketedType :: type : Type$

$BasicType = \text{BOOLEAN} \mid \text{NAT} \mid \text{NATONE} \mid \text{INTEGER} \mid \text{RAT} \mid \text{REAL} \mid \text{CHAR} \mid \text{TOKEN}$

$QuoteType :: lit : QuoteLit$

$CompositeType :: id : Id$
$\quad fields : FieldList$

$FieldList = Field^*$

$Field :: sel : [Id]$
$\quad type : Type$

$UnionType :: summands : Type^+$

$\quad \text{inv } mk\text{-}UnionType(ts) \triangleq \text{len } ts \geq 2$

$ProductType :: factors : Type^+$

$\quad \text{inv } mk\text{-}ProductType(ts) \triangleq \text{len } ts \geq 2$

$OptionalType :: type : Type$

$SetType :: elemtp : Type$

$SeqType = Seq0Type \mid Seq1Type$

$Seq0Type :: elemtp : Type$

$Seq1Type :: elemtp : Type$

$MapType = GeneralMapType \mid InjectiveMapType$

$GeneralMapType :: mapdom : Type$
$\quad maprng : Type$

$InjectiveMapType :: mapdom : Type$
$\quad maprng : Type$

$FnType = PartialFnType \mid TotalFnType$

$PartialFnType :: fndom : DiscretionaryType$
$\quad fnrng : Type$

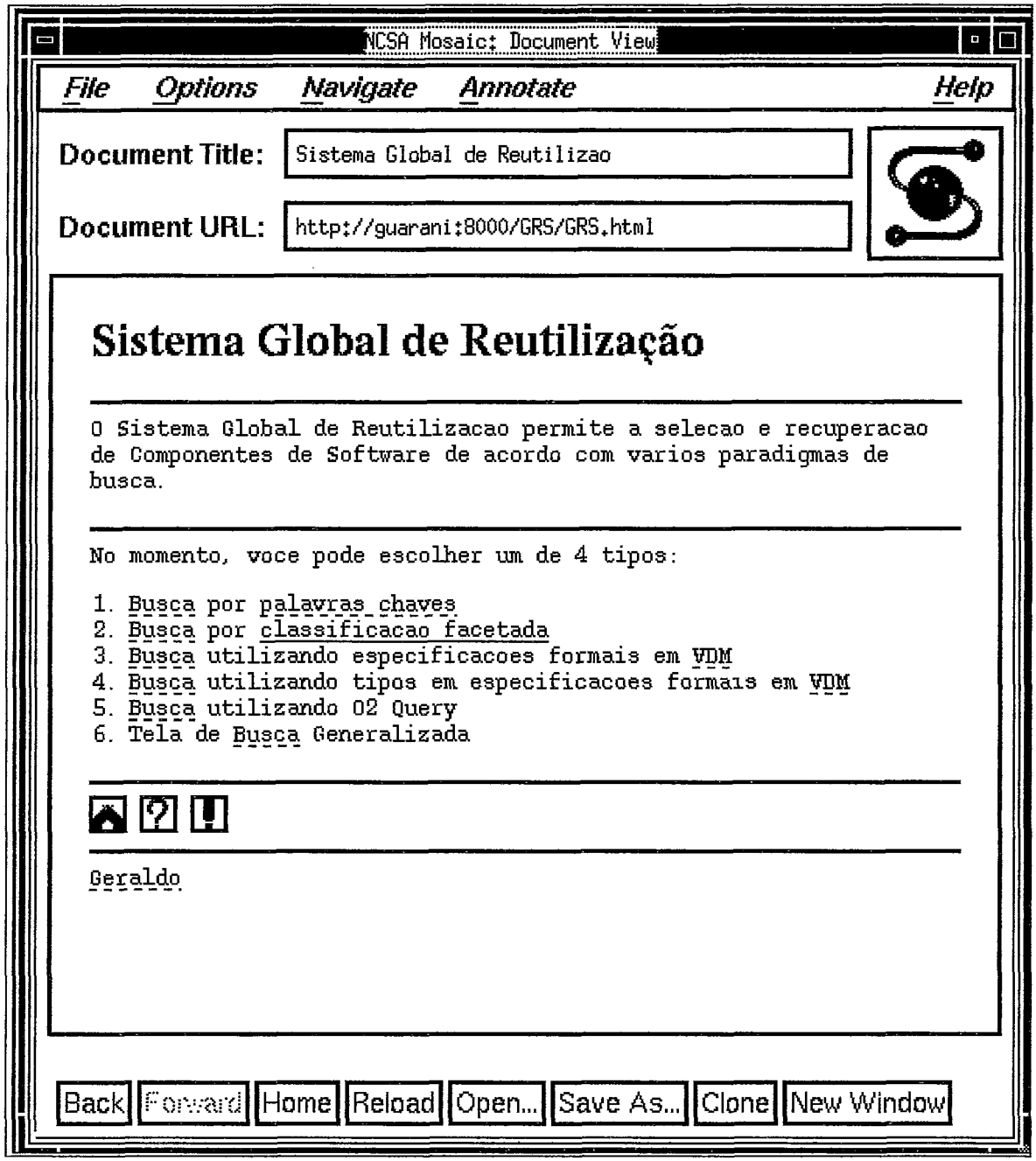
$TotalFnType :: fndom : DiscretionaryType$
$\quad fnrng : Type$

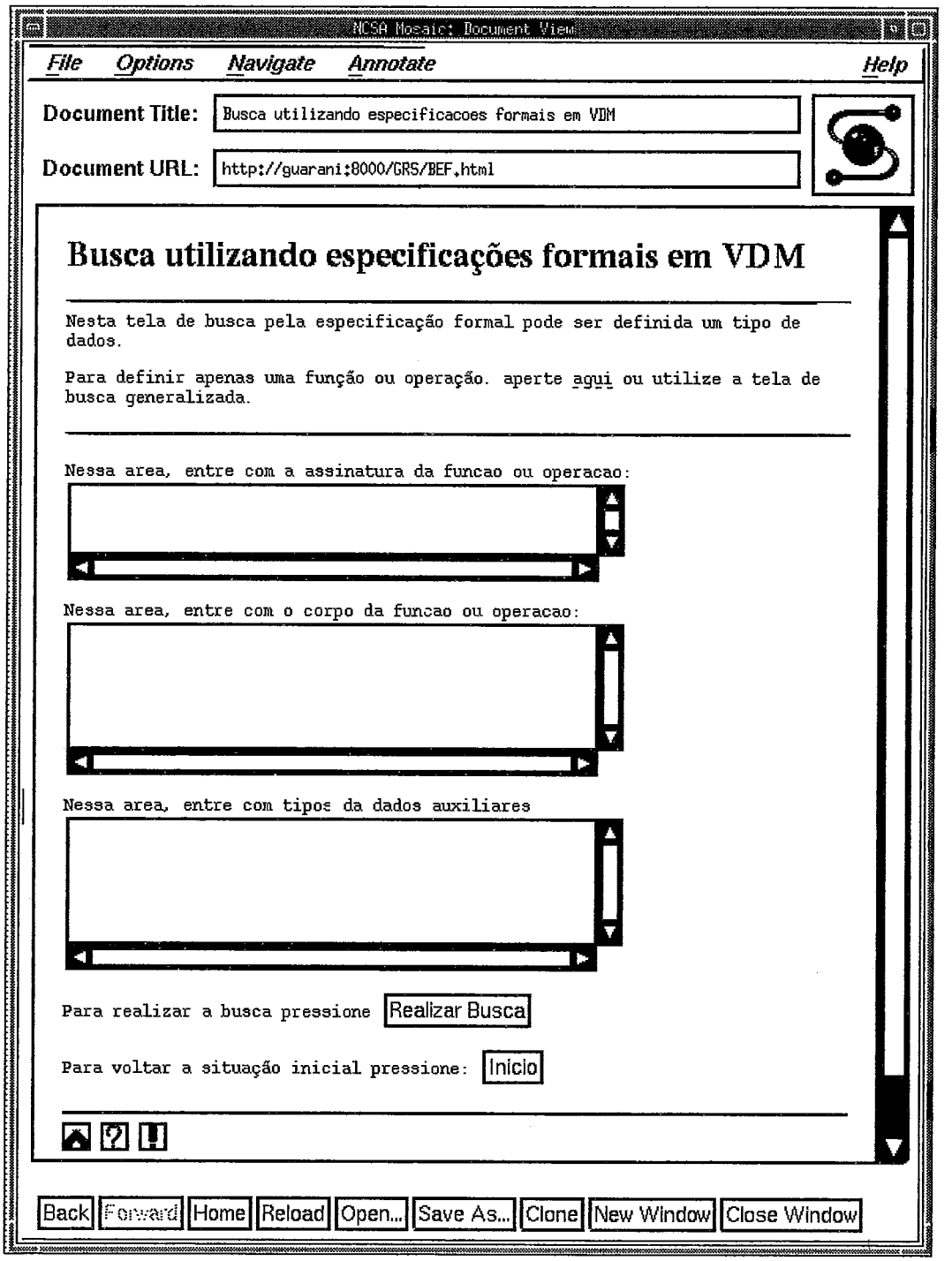
$DiscretionaryType = Type \mid \text{UNITTYPE}$

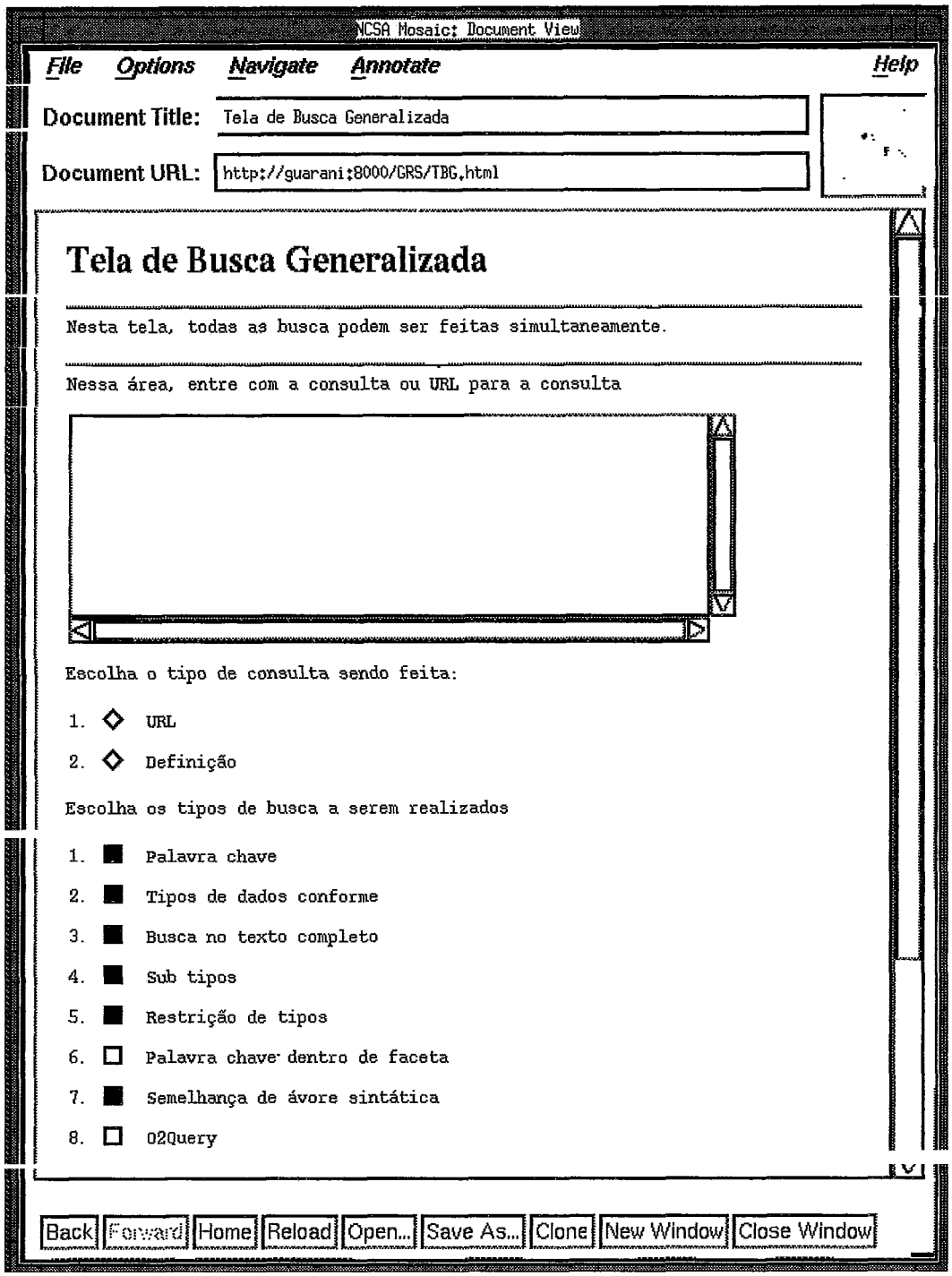
$TypeName = Name$

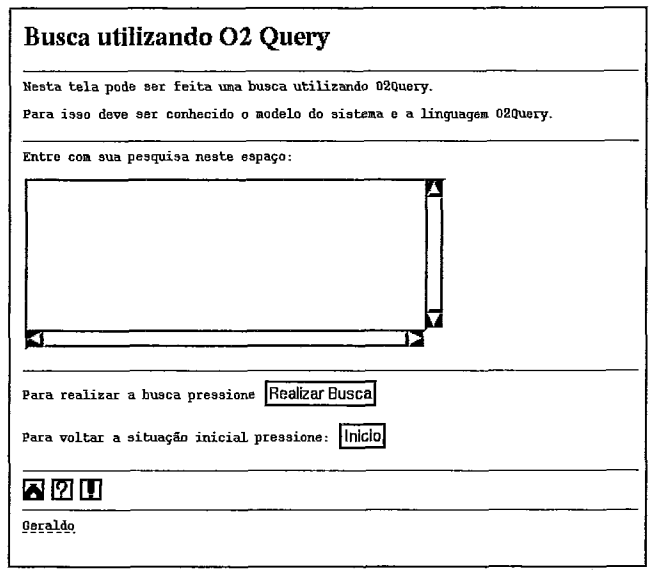
$TypeVar = TypeVarId$

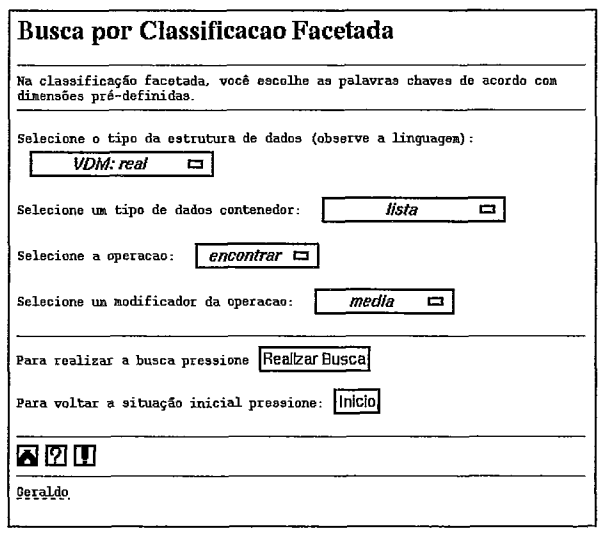
APÊNDICE III: TELAS DO SISTEMA TABETÁ

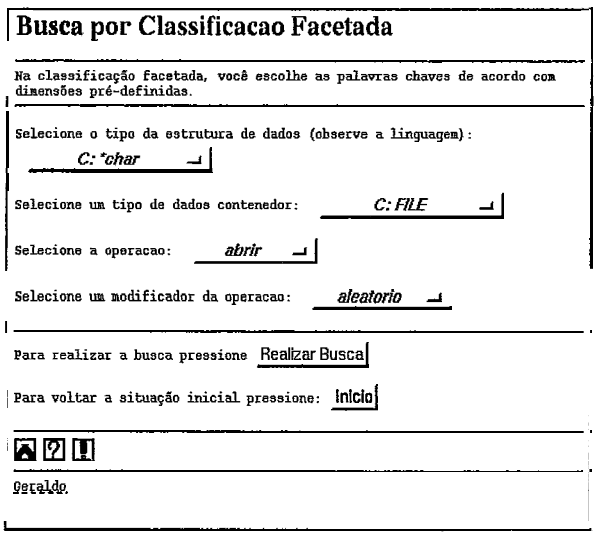

NCSA Mosaic: Document View
File
Options
Navigate
Annotate
Help
Document Title:
Sistema Global de Reutilizao
Document URL:
http://guarani:8000/GRS/GRS.html
Sistema Global de Reutilização
O Sistema Global de Reutilizacao permite a selecao e recuperacao de Componentes de Software de acordo com varios paradigmas de busca.
No momento, voce pode escolher um de 4 tipos:
1. Busca por palavras chaves
2. Busca por classificacao facetada
3. Busca utilizando especificacoes formais em VDM
4. Busca utilizando tipos em especificacoes formais em VDM
5. Busca utilizando O2 Query
6. Tela de Busca Generalizada
Geraldo
Back
Forward
Home
Reload
Open...
Save As...
Clone
New Window

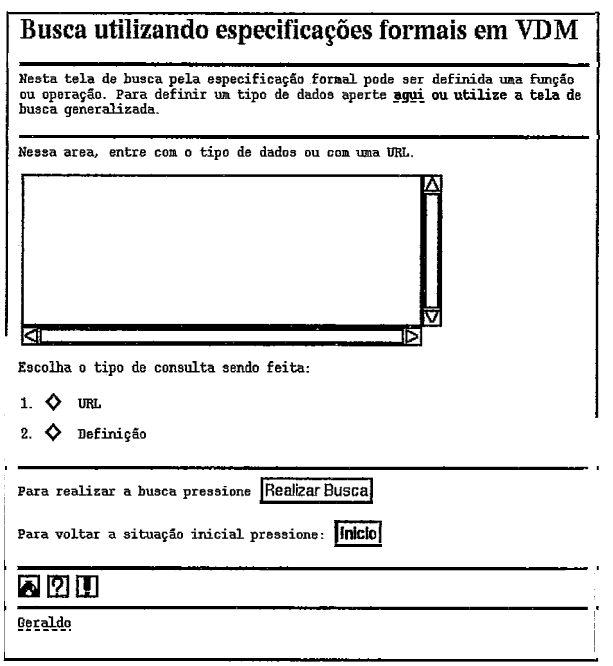

NCSA Mosaic: Document View
File
Options
Navigate
Annotate
Help
Document Title:
Busca utilizando especificacoes formais em VDM
Document URL:
http://guarani:8000/GRS/BEF.html
Busca utilizando especificações formais em VDM
Nesta tela de busca pela especificação formal pode ser definida um tipo de dados.
Para definir apenas uma função ou operação. aperte agui ou utilize a tela de busca generalizada.
Nessa area, entre com a assinatura da funcao ou operacao:
Nessa area, entre com o corpo da funcao ou operacao:
Nessa area, entre com tipos da dados auxiliares
Para realizar a busca pressione
Realizar Busca
Para voltar a situação inicial pressione:
Início
Back
Forward
Home
Reload
Open...
Save As...
Clone
New Window
Close Window


NCSA Mosaic: Document View
File Options Navigate Annotate Help
Document Title: Tela de Busca Generalizada
Document URL: http://guarani:8000/GRS/TBG.html
Tela de Busca Generalizada
Nesta tela, todas as busca podem ser feitas simultaneamente.
Nessa área, entre com a consulta ou URL para a consulta
Escolha o tipo de consulta sendo feita:
1. URL
2. Definição
Escolha os tipos de busca a serem realizados
1. Palavra chave
2. Tipos de dados conforme
3. Busca no texto completo
4. Sub tipos
5. Restrição de tipos
6. Palavra chave dentro de faceta
7. Semelhança de ávore sintática
8. O2Query
Back Forward Home Reload Open... Save As... Clone New Window Close Window


Busca utilizando O2 Query
Nesta tela pode ser feita uma busca utilizando O2Query.
Para isso deve ser conhecido o modelo do sistema e a linguagem O2Query.
Entre com sua pesquisa neste espaço:
Para realizar a busca pressione
Realizar Busca
Para voltar a situação inicial pressione:
Inicio
Geraldo


Busca por Classificacao Facetada
Na classificação facetada, você escolhe as palavras chaves de acordo com dimensões pré-definidas.
Selecione o tipo da estrutura de dados (observe a linguagem):
VDM: real
Selecione um tipo de dados contenedor:
lista
Selecione a operacao:
encontrar
Selecione um modificador da operacao:
media
Para realizar a busca pressione
Realizar Busca
Para voltar a situação inicial pressione:
Inicio
Geraldo


Busca por Classificacao Facetada
Na classificação facetada, você escolhe as palavras chaves de acordo com dimensões pré-definidas.
Selecione o tipo da estrutura de dados (observe a linguagem):
C: *char
Selecione um tipo de dados contenedor:
C: FILE
Selecione a operacao:
abrir
Selecione um modificador da operacao:
aleatorio
Para realizar a busca pressione
Realizar Busca
Para voltar a situação inicial pressione:
Inicio
Geraldo


Busca utilizando especificações formais em VDM
Nesta tela de busca pela especificação formal pode ser definida uma função ou operação. Para definir um tipo de dados aperte agui ou utilize a tela de busca generalizada.
Nessa area, entre com o tipo de dados ou com uma URL.
Escolha o tipo de consulta sendo feita:
1. URL
2. Definição
Para realizar a busca pressione
Realizar Busca
Para voltar a situação inicial pressione:
Inicio
Geraldo

Busca utilizando especificações formais em VDM

Nesta tela de busca pela especificação formal pode ser definida uma função ou operação. Para definir um tipo de dados aperte agui ou utilize a tela de busca generalizada.

Nessa area, entre com o tipo de dados ou com uma URL.

http://guarani:8000/esp.vdm

Escolha o tipo de consulta sendo feita:

1. ◆ URL
2. ◇ Definição

Para realizar a busca pressione [Realizar Busca]

Para voltar a situação inicial pressione: [Inicio]

Geraldo

Busca por palavras chave

Use qualquer palavra chave nesta busca.

No campo a seguir entre com palavras chave importantes

No campo a seguir entre com palavras chave secundarias

Para realizar a busca pressione [Realizar Busca]

Para voltar a situação inicial pressione: [Inicio]

Geraldo

Lista de componentes encontrados

Esta tela apresenta a lista dos componentes encontrados de acordo com a consulta feita.

Semelhança - Tamanho - Descrição

1. 1000 - 1.2K - Tipo de dados Inteiro
2. 900 - 1.2K - Tipo de dados Natural
3. 800 - 2.5K - Tipo de dados Real

Geraldo

Busca utilizando especificações formais em VDM

Nesta tela de busca pela especificação formal pode ser definida uma função ou operação. Para definir um tipo de dados aperte aqui ou utilize a tela de busca generalizada.

Nessa area, entre com o tipo de dados ou com uma URL.

Escolha o tipo de consulta sendo feita:

1. URL
2. Definição

Para realizar a busca pressione Realizar Busca

Para voltar a situação inicial pressione: Inicio

Geraldo

APÊNDICE IV: MODELO DE OBJETOS

Tabetá Object Model