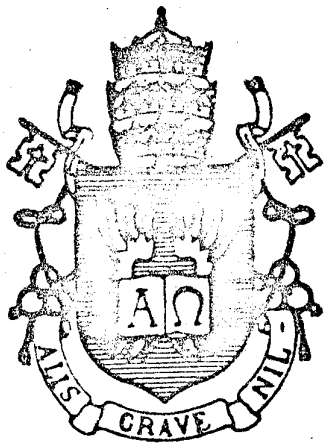


PUC

Série: Monografias em Ciência da Computação,
Nº. 14/91

JSD/PUC: UM AMBIENTE DE SOFTWARE EXPERIMENTAL PARA ESTUDO DO PROCESSO DE AUTOMATIZAÇÃO DE DESENVOLVIMENTO DE SOFTWARE

Carlos J. P. Lucena
Julio Cesar S. P. Leite
José Rodrigues Fernandes
Mário Gheiner
Antonio Francisco do Prado

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Série: Monografias em Ciência da Computação, No. 14/91
Editor: Carlos J. P. Lucena Junho, 1991

JSD/PUC : UM AMBIENTE DE SOFTWARE EXPERIMENTAL PARA ESTUDO DO PROCESSO DE AUTOMATIZAÇÃO DE DESENVOLVIMENTO DE SOFTWARE *

Carlos J. P. Lucena
Julio Cesar S. P. Leite
José Rodrigues Fernandes
Mário Gheiner
Antonio Francisco do Prado

* Trabalho patrocinado pela Secretaria de Ciêmcia e Tecnologia da Presidência da República.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

Tel.:(021)529-9386 Telex:31078 Fax:(021)511-5645
E-mail:rosane@inf.puc-rio.br

JSD/PUC: Um Ambiente de Software Experimental para Estudo do Processo de Automatização de Desenvolvimento de Software

Departamento de Informática
Pontificia Universidade Católica do Rio de Janeiro
R. Marques de S. Vicente 225 Rio de Janeiro 22453
Brasil
Maio 1991

Carlos J. P. Lucena
Julio Cesar S. P. Leite
José Rodrigues Fernandes
Mario Gheiner
Antonio Francisco do Prado

Resumo

O presente trabalho trata do estudo da automatização do processo de desenvolvimento de software, instanciado para a metodologia JSD. A arquitetura do ambiente é baseada em um programa de processos no estilo de Osterweil que integra, através de um procedimento, os métodos da metodologia JSD. O programa de processo utilizado também recorre ao uso de interfaces como especificações.

O trabalho apresenta um experimento específico conduzido no âmbito do ambiente: um modelo de dados para adequação do modelo genérico de representação de desenhos de software de Potts para permitir o registro de decisões e raciocínios adotados pelo projetista no ambiente experimental. O recurso adicionado ao ambiente JSD/PUC visa dar suporte a práticas de reutilização e manutenção no ambiente JSD/PUC.

Abstract

The present paper deals with the study of the automatization of the software development process which is instantiated to the JSD methodology.

The environment's architecture is based upon a process program expressed in the Osterweil style. This process program integrates the methods that constitute the JSD methodology. The paper also adjusts Pott's generic model for design representations to allow the record of decisions and the capture of the reasoning of a specialist that uses the experimental environment. This last feature aims at supporting reuse and maintenance practices in the JSD/PUC environment.

1 Introdução

O ambiente experimental JSD/PUC foi projetado e desenvolvido para permitir o estudo de técnicas da área de engenharia de software no ambiente acadêmico com ferramentas e aplicações de porte realista. Frequentemente, por falta de laboratórios adequados, a pesquisa universitária em engenharia de software fica confinada a problemas de porte reduzido, cuja solução não assegura a sua aplicabilidade a sistemas de tamanho real (o chamado "scale up" das soluções [CSTB 90]). A prática universal tem sido a produção de ambientes acadêmicos (ex.: [Habermann 86] , [Lamsweerde 88] e [Horowitz 86]) por equipes de estudantes de pós-graduação, ambientes estes que vão evoluindo através de gerações de projetos específicos de pesquisa.

No caso do JSD/PUC o próprio estilo do projeto de desenvolvimento foi uma atividade experimental. Usou-se a idéia do programa de processos [Osterweil 87] como integrador de métodos para a metodologia escolhida para o ambiente. Também foi praticada a técnica de interfaces como especificações ([Cabral 90] e [Bischofberger 89]) para a especificação do ambiente. Esta tarefa foi tornada viável pelo uso de software genérico para interfaces [OSF/Motif 90].

A escolha da metodologia recaiu sobre JSD [Jackson 83] em função da variedade de métodos bem definidos que ela compreende. Soma-se a isto o fato que a metodologia compatibiliza a decomposição por funções com a decomposição pela forma (baseada em objetos). A distinção clara entre estes estilos de decomposição é feita, por exemplo, em [Maher 91] .

O objetivo do trabalho aqui descrito vem sendo, portanto, o estudo do processo de automatização do desenvolvimento de software em um ambiente geral e flexível para a engenharia de software. Este artigo discute, especificamente, a técnica de programação de processos [Osterweil 87] para o estabelecimento de uma arquitetura de ambiente para a metodologia JSD e a sua integração com a instância do modelo genérico de Potts [Potts 88] e [Potts 89] que permite o registro da racionalização da atividade de desenho ("design rationale"[Lee 91]).

A escolha de uma boa representação para racionalização do design pode levar a uma melhor compreensão das questões envolvidas no espaço de desenho de software e dos princípios subjacentes à compreensão da interação homem máquina. Em particular, no caso do desenho de software a partir de métodos prescritivos (como em JSD) espera-se poder desenvolver técnicas eficientes de reutilização para suporte às atividades de manutenção.

O trabalho está organizado da seguinte forma: na Secão 2 fazemos um resumo da metodologia JSD, na Seção 3 apresentamos o programa de processo utilizado no JSD/PUC e detalhamos as atividades da criação do JSD/PUC, na Seção 4 descrevemos a arquitetura e funcionalidade do ambiente implementado, na Seção 5 falamos sobre a integração das idéias de Potts ao nosso ambiente e concluimos na Seção 6 apontando trabalhos futuros a serem efetuados sobre o ambiente.

2 Visão geral do JSD(Jackson System Development)

JSD [Jackson 83] é uma metodologia para especificação e projeto de sistemas de software que distingue 3 importantes fases:

- A Fase de Modelo, na qual enfatiza a modelagem das entidades e ações correspondentes, no dominio do problema identificado.
- A Fase de Rede, na qual a especificação do sistema é desenvolvida a nivel operacional.
- A Fase de Implementação na qual a especificação JSD é realizada considerando as restrições de recursos computacionais disponiveis, isto é, processadores e memória.

As 2 primeiras fases, por sua vez, são sub-divididas em 5 grandes passos:

- Entidade e Ação
- Estrutura Entidade
- Modelo Inicial
- Função
- Cronologia

Os 2 primeiros passos constituem a Fase de Modelo e os outros 3 a Fase de Rede.
Estes passos podem ser vistos como atividades de alto nível e representadas no próprio Diagrama da Estrutura JSD, conforme mostra a Figura 1.

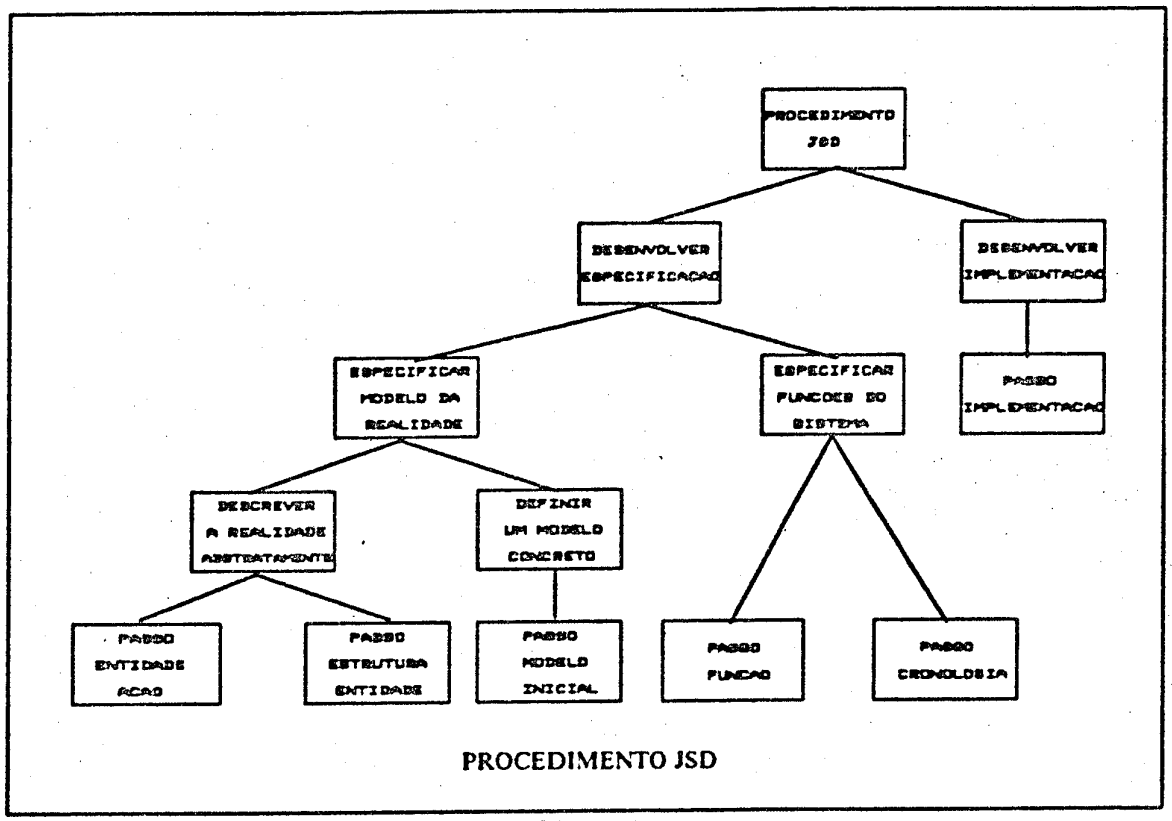


Figura 1

Em seguida serão apresentadas, resumidamente, cada uma das fases que constituem o modelo de desenvolvimento usado em nosso trabalho.

*Fase de Modelo

Nesta fase o desenvolvedor define a área de interesse do mundo real , listando as entidades e ações a partir das quais o sistema será concebido (passo Entidade e Ação). Uma vez definida a entidade, cria-se um diagrama (Diagrama de Estrutura) que representa as ações, ordenadas sequencialmente, sofridas ou executadas pela entidade (passo Estrutura da Entidade). Os passos desta Fase são repetidos para obtenção de uma lista correta e completa de entidades e ações. Parte-se, portanto, de uma lista de entidades e ações e através de heuristicas e do aprimoramento do conhecimento sobre o problema atualiza-se esta lista.

*Fase de Rede

Nesta Fase é elaborado um Modelo de Rede, onde a descrição da realidade, em termos de entidades e ações da Fase Modelo, é concretizada em um conjunto formado por processos do modelo e conexões entre o modelo e o mundo real. O diagrama da Rede, conhecido como Diagrama de Especificação do Sistema (DES), permite portanto, especificar uma simulação

do mundo real, representado por processos sequenciais, em termos dos processos do modelo. Estes são conectados com o mundo real ou entre si, de forma a poderem ser executados por um computador.

Os processos do Modelo de Rede conectam-se de duas formas:

- por fila de mensagens e

- por vetor de estado.

Fila de mensagens é a forma padrão de comunicação. Os processos do mundo real produzem mensagens para cada ação executada ou sofrida, e a fila ou cadeia dessas mensagens serve de entrada para processos do sistema.

Conforme mostra o DES da Figura 2 , a entidade E-0 do mundo real, comunica-se por fila de mensagens com o processo do modelo E-1, ou seja, escreve uma fila de mensagens DS, a qual serve de entrada para o processo do modelo E-1. Portanto E-1 lê esta cadeia de mensagens. Por convenção o sufixo 0 indica o processo do mundo real e o sufixo maior que zero indica o processo do modelo. O retângulo representa um processo e a conexão fila de mensagens é representada por um circulo.

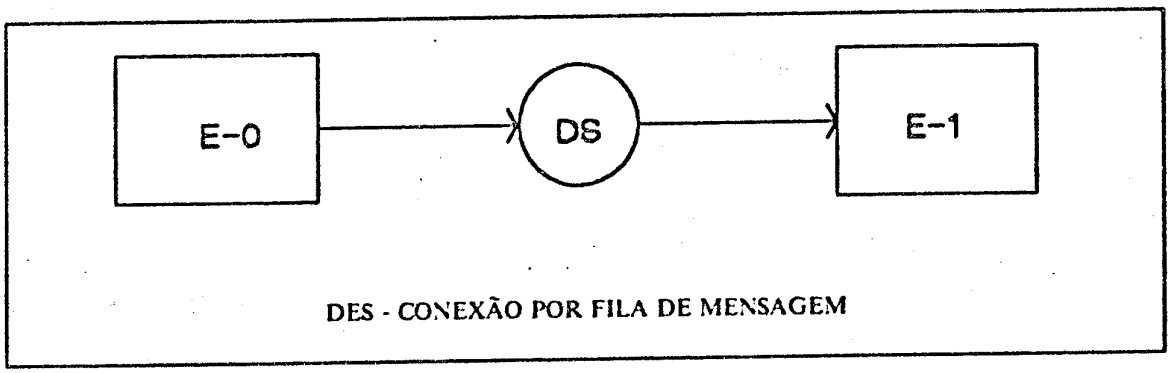


Figura 2

Na conexão por vetor de estado, mostradas na Figura 3, o processo Q inspeciona diretamente o vetor de estado, isto é, as variáveis locais internas de outro processo, ou seja de P. Neste caso a iniciativa da comunicação está inteiramente com o processo que executa a inspeção, ou seja Q. O losango indica que a conexão é feita por vetor de estado.

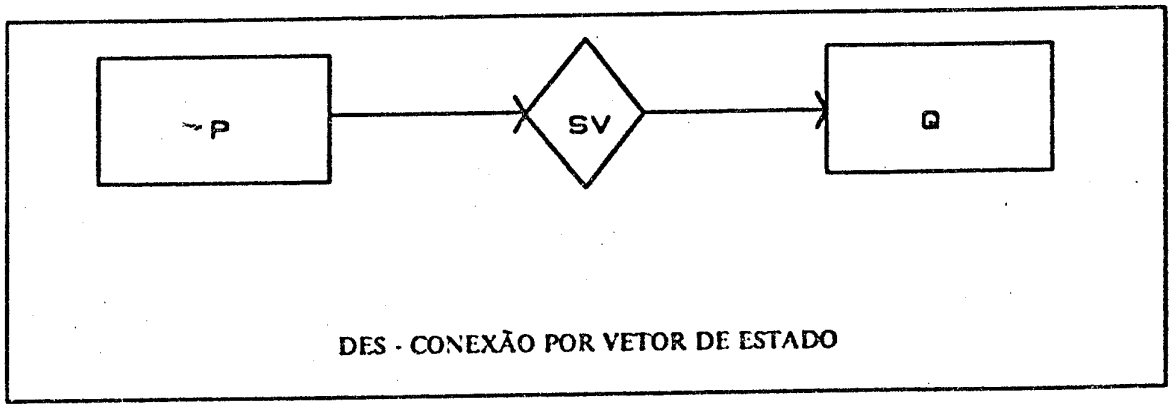


Figura 3

O Modelo Rede do sistema, contém não só os processos correspondentes às entidades, mas também processos que provém funcionalidade ao modelo.

Estes processos são estabelecidos no passo Função e tem por objetivo adequar o modelo para o suporte de atividades como saídas, filtros e interfaces externas.

Assim, por exemplo, funções são adicionadas no DES para produzir saídas.

A figura 4 mostra a inclusão do processo FUNÇÃO no DES para emitir o relatório LST. A barra dupla na linha de ORDEM para DE indica que o processo FUNÇÃO lê fila de mensagens de muitos processos ORDEM.

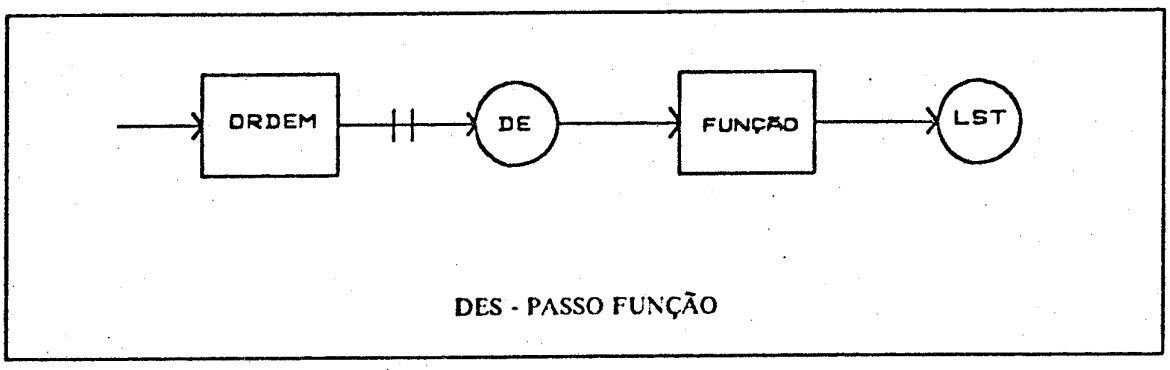


Figura 4

Funções podem ser embutidas em um processo desde que não hajam mudanças na estrutura de controle do processo hospedeiro, neste caso a função tem que ser fatorada. Por exemplo, inclusão de um estrutura de seleção que condiciona uma determinada saída. Neste caso, um novo processo, com nível 2 ou maior, é criado como uma extensão do processo nível 1.

A figura 5 ilustra o exemplo através da criação da função E-2.

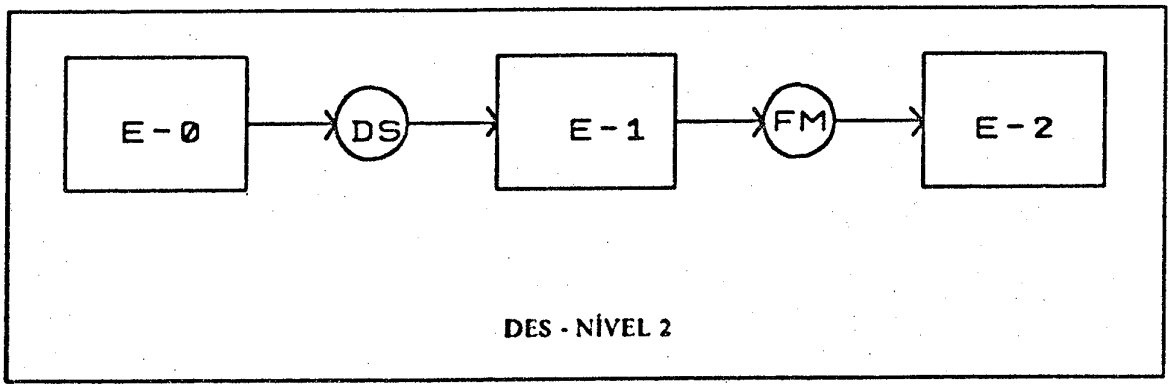


Figura 5

O passo cronologia, interposto entre os passos Função e Implementação, tem por objetivo especificar melhor as restrições de tempo na comunicação dos processos. Anotações informais a serem observadas durante a implementação ou inclusão de processos de sincronização são os meios de descrever aspectos de cronologia.

*Fase de Implementação

O diagrama de rede (DES), mostra quais são os processos, como estão conectados uns aos outros e quais as entradas e saidas na periferia do sistema. O diagrama de estrutura, com seu texto estruturado, mostra os detalhes da especificação de cada processo.

Estas especifições são então usadas para elaborar o Diagrama Estruturado da Implementação (DEI). As principais tarefas do desenvolvedor neste passo da implementação visam determinar:

- Quantos processos reais ou virtuais são usados para executar o sistema,
- Como os processos do sistema são alocados para os processadores disponiveis,
- Quais os processos alocados a cada processador e sua programação.

Tipicamente para modelos implementados em um único processador tem-se que, os vetores de estado são separados em arquivos, um processo escalonador é acrescido ao sistema e os processos do DES são invertidos em razão dos processos que os invocam.

3 Programa do Processo de Software no Desenvolvimento do Ambiente JSD/PUC.

Por ser um ambiente de desenvolvimento experimental, o ambiente JSD/PUC procurou formalizar e registrar o processo de desenvolvimento utilizado na sua concepção e desenvolvimento. A arquitetura do ambiente espelha o processo de desenvolvimento e isto permite a sua extensão para estudos especificos como o desenvolvimento de um modelo de dados (modelo de Potts) para o registro das decisões de desenho que será apresentado mais adiante.

Entende-se por processo uma abordagem sistemática para a criação de um produto ou para a execução de alguma tarefa. É importante realçar a diferença entre um processo e uma descrição de processo. Enquanto um processo é um veiculo para a realização de um trabalho, a descrição de um processo é uma especificação de como o trabalho deve ser feito [Osterweil 87]. Análogamente a receita para fazer um bolo corresponde a descrição do processo enquanto a confecção do bolo corresponde ao processo.

O enfoque de Osterweil pretende tornar descrições de processos de software rigorosas a tal ponto que seja possivel automatizá-las como um programa aplicativo.

Processos de software são descritos por programação da mesma forma que se programam aplicações no computador, usando a analogia de Osterweil que considera que processos de software são eles próprios um software ("software processes are software too").

A Figura 6 esquematiza o processo de desenvolvimento de software pela agregação dos recursos computacionais disponiveis seguindo o programa do processo de desenvolvimento de software em diferentes niveis de abstração. Cada descrição do processo é instanciada pela criação de um processo que por sua vez descreve um processo o qual é instanciado para gerar um novo processo e assim por diante, até o nivel da programação da aplicação pelo usuário.

Em nosso caso, o programa do processo a nivel geral, foi instanciado pelo engenheiro projetista do ambiente para uma descrição do processo JSD na plataforma de software UNIX, C, Motif e X-Window.

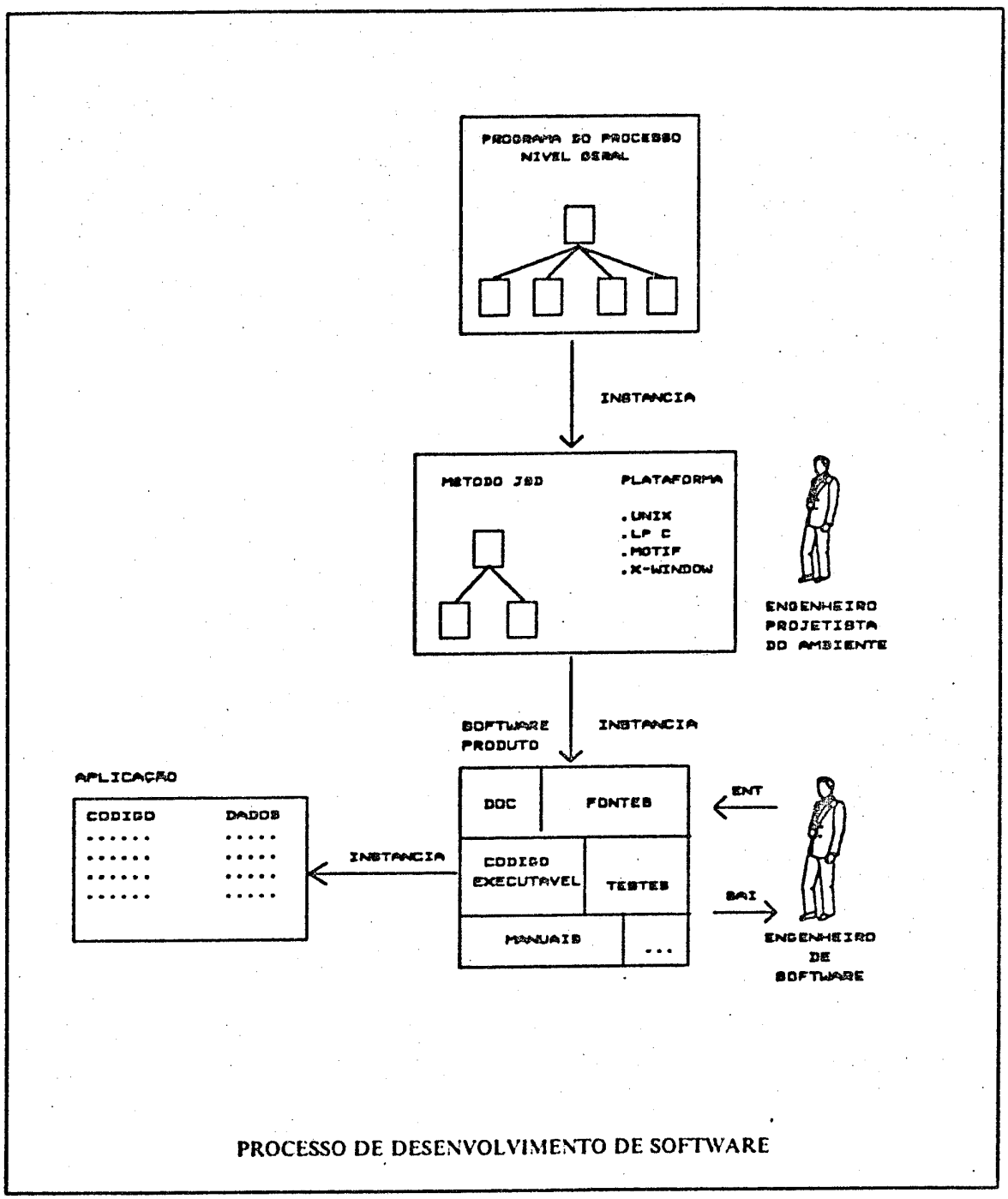
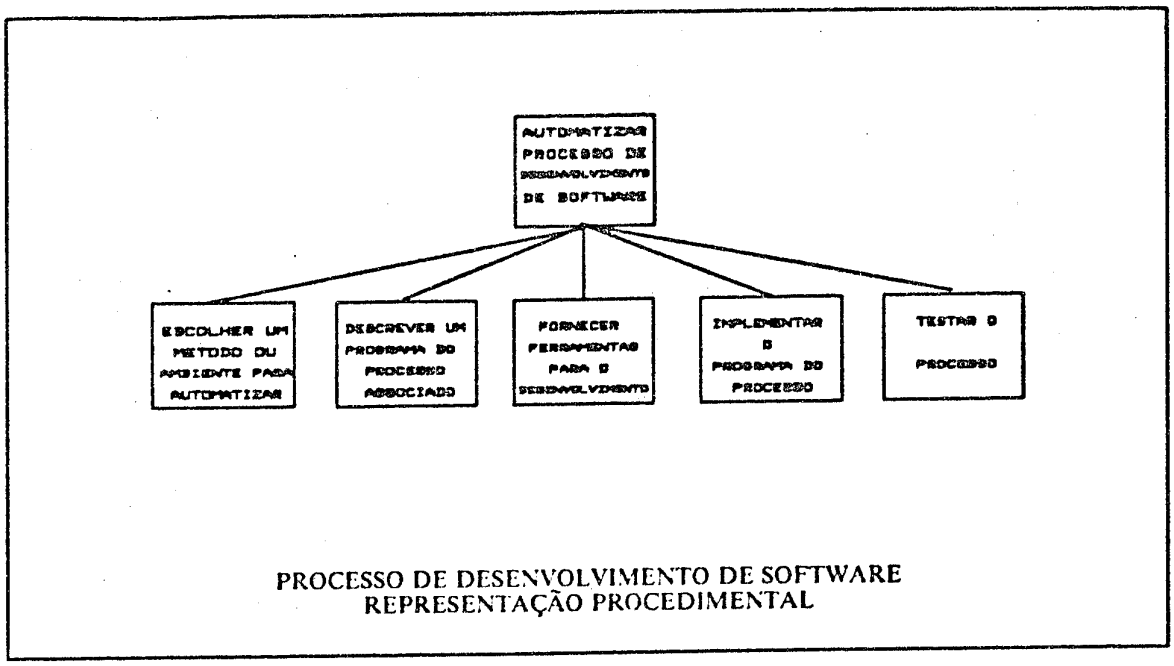


Figura 6

Em seguida, o engenheiro de software instanciou esta descrição dando origem a um software produto (ferramenta de CASE-JSD). Este por sua vez pode ser instanciado pelo usuário final numa aplicação.

Usando-se uma representação procedimental em um nivel bem alto de abstração pode-se expressar a automatização do processo de desenvolvimento de software conforme está mostrado na Figura 7.



PROCESSO DE DESENVOLVIMENTO DE SOFTWARE
REPRESENTAÇÃO PROCEDIMENTAL

Figura 7

Usando esta descrição genérica para criação do Ambiente de Apoio a Metodologia JSD tem-se a seguinte instância do programa de processo:

- JSD como metodologia escolhida;
- Descrição do processo JSD como descrição do programa do processo associado;
- UNIX, linguagem C, Motif e X-Window como ferramentas usadas para construir a plataforma do ambiente;
- C e Motif/X-Window como linguagens de implementação;
- Submissões de casos como testes.

Este instanciamento por sua vez, pode ser visto como uma nova descrição do processo, apresentado em forma de árvore na figura 8.

Percorrendo esta árvore encontramos nas folhas as ações básicas que constituiram as grandes atividades do trabalho de implementação do JSD/PUC. Elas se encontram listadas a seguir:

a) Programar o processo JSD - Nivel Geral
b) Especificar Interface Estática com o Ambiente
c) Programar o processo JSD - Nivel Detalhado
d) Analisar a Consistência entre o processo JSD e a Interface Estática
e) Construir o autômato que descreve a interação entre as telas da interface
f) Analisar a consistência entre o processo JSD e a interface dinâmica
g) Estudar as Ferramentas utilizadas na descrição do processo (Sistema Operacional UNIX , Linguagem C , MOTIF e X-Window)
h) Prototipar o programa do processo JSD em linguagem C
i) Prototipar a interface utilizando X-Window e Motif
j) Integrar o processo JSD em C e a interface em Motif/X-Window
k) Testar o Ambiente construido.

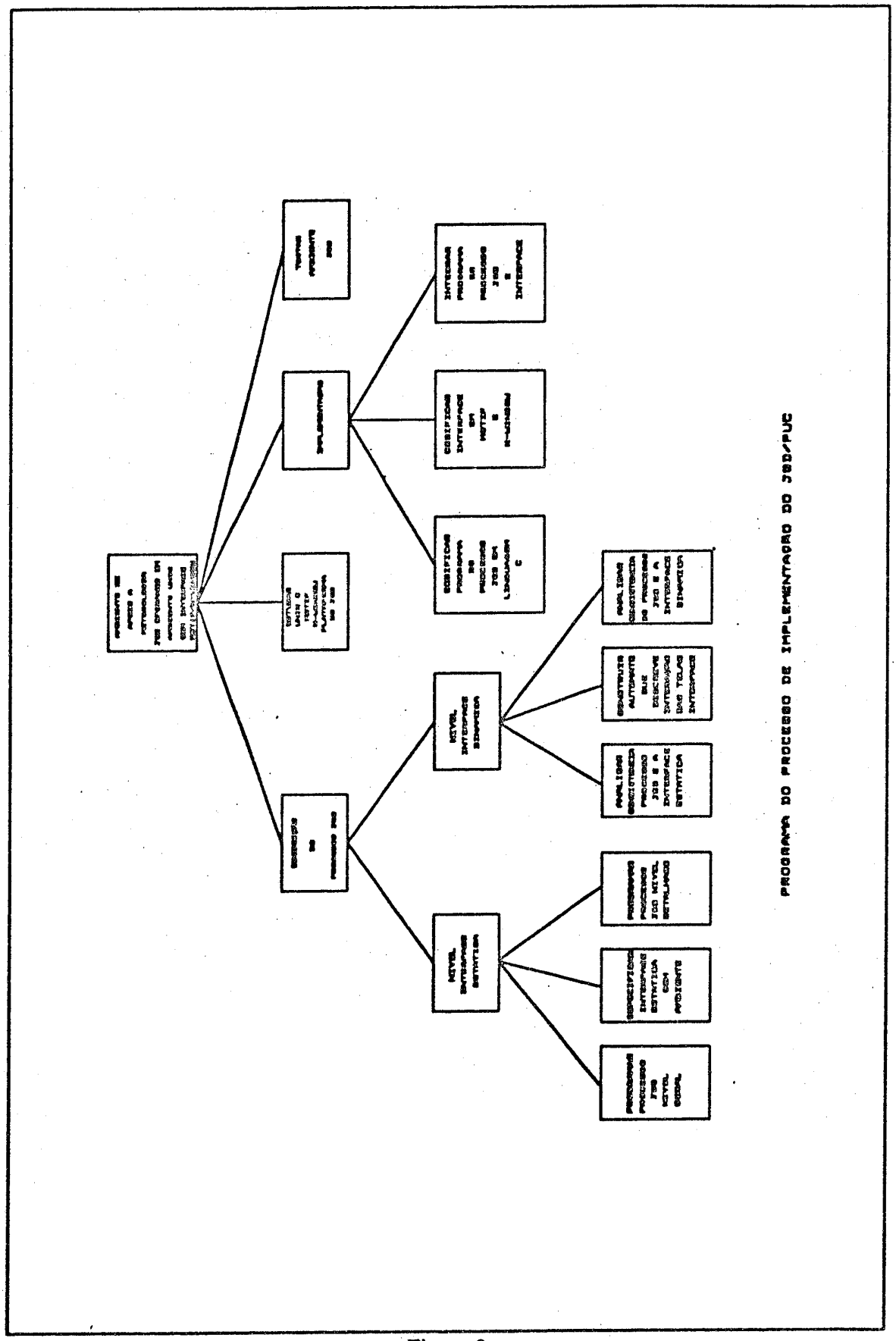


Figura 8

A seguir serão descritas cada uma das atividades mencionadas para criação do ambiente de apoio à metodologia JSD operando numa plataforma UNIX com Linguagem de Programação C, Motif e X-Window numa plataforma de Hardware com "workstations" SUN/3.

3.1 Programar o processo JSD - Nível Geral

Nesta primeira etapa do trabalho estudou-se a Metodologia JSD para que se pudesse conhecer e representar os elementos básicos de JSD.

Em seguida, usando-se uma linguagem algoritimica na forma de pseudo-código, descreveram-se as partes mais importantes do processo JSD.

Detalhes de como os resultados são obtidos e como são avaliados foram deixados para serem descritos em procedures de programa de processos de mais baixo nível.

A Figura 9 mostra um trecho do programa do processo JSD, programado nesta primeira fase.

```
proc PRINCIPAL
  Obter Opção ;
  Enquanto Opção não for = 'RETORNA' faça
    Conforme Opção
      PROJETO : Executar OBTER_PROJETO (Nome_Projeto) ;
      FASE_ENTIDADE_AÇÃO : Executar ATENDER_FASE1 ;
      FASE_ESTR_ENTIDADE : Executar ATENDER_FASE2 ;
      FASE_MOD_INICIAL   : Executar ATENDER_FASE3 ;
      FASE_CRONOLOGICA   : Executar ATENDER_FASE4 ;
      FASE_IMPLEMENTAÇÃO : Executar ATENDER_FASE5 ;
      HELP : Executar ATENDER_HELP;
      FIM : Executar ENCERRAR_JSD;
    fim_conforme
    Obter Opção ;
  fim_enquanto
fim_principal
```

PROCESSO JSD - NÍVEL GERAL

Figura 9

3.2 Especificar a Interface estática com o Ambiente

Nesta etapa foram especificados os lay-outs das telas de interface do processo. Usou-se, conforme mostra a figura 10, uma linguagem de menus para especificar cada tela de cada fase do processo JSD. Aspectos importantes a serem ressaltados nesta atividade foram as dúvidas surgidas com relação:

- à escolha das opções do "menu-bar", que deveriam ficar permanentes em todas as telas do sistema ,
- ao que deveria ser solicitado como entrada de dado ao usuário e o que o sistema deveria fornecer automáticamente,
- ao uso de "defaults" para complementar as opções dos menus.

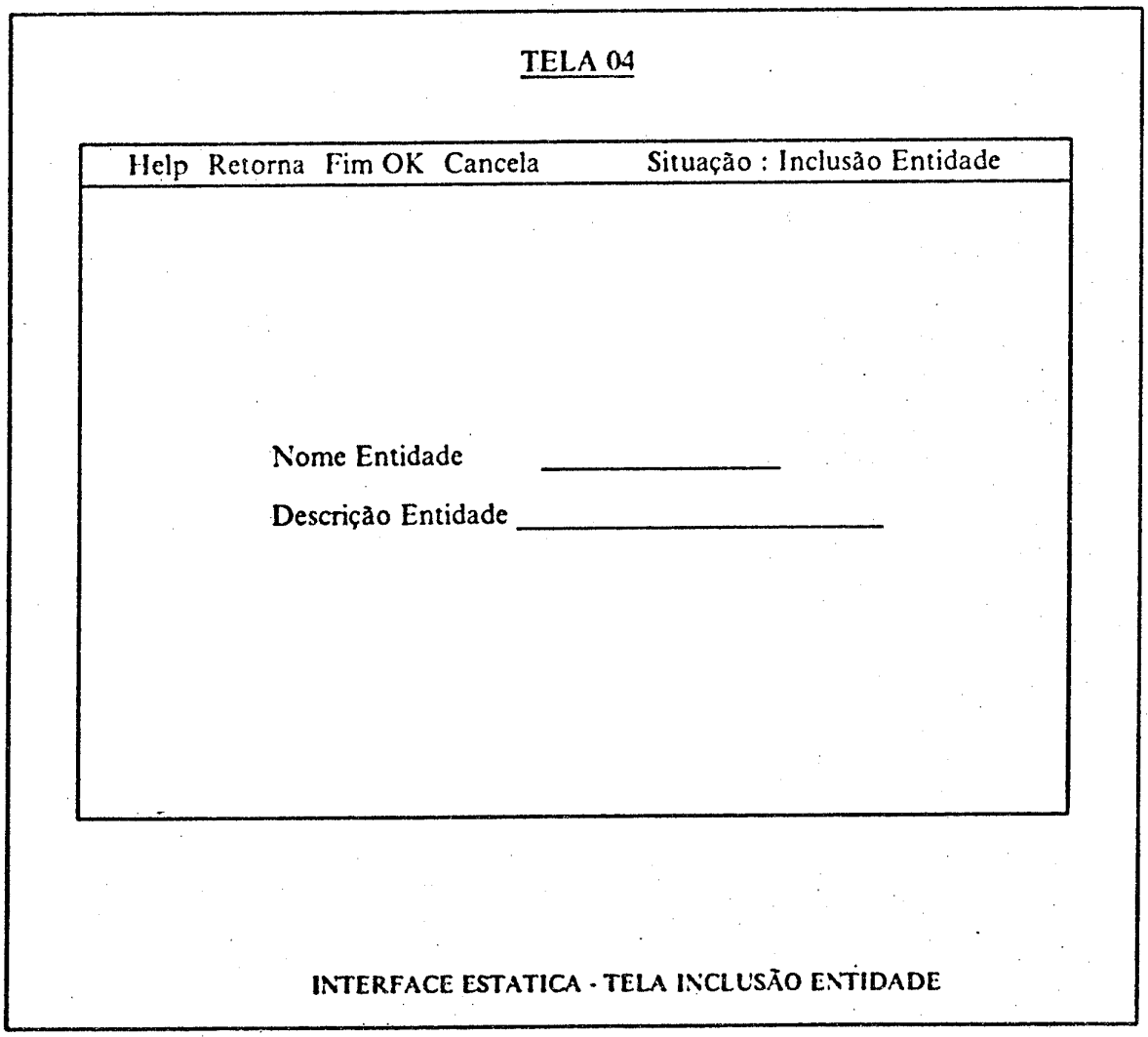


Figura 10

3.3 Programar o processo JSD - Nível Detalhado

Aqui, partiu-se do algoritmo a nível geral obtido em 3.1 e detalhou-se gradualmente a descrição do processo JSD, o que tornou mais claro o seu entendimento. Assim por exemplo, a figura 11 mostra um nivel de refinamento do procedimento Modificar-Entidade.

```
proc MODIFICAR-ENTIDADE
  Obter Opção ;
  Enquanto Opção não for = 'RETORNA' faça
   Conforme Opção
     OK :
        Obter Nome_Entidade ;
        Obter Novo_Nome_Entidade ;
        Obter Desc_Entidade ;
        Executar VALIDAR_ENTIDADE ( Novo_Nome_Entidade , Resultado );
        Se Resultado = 'OK' então
          Executar ATUALIZAR_ENTIDADE ( Nome_Entidade ,
                                        Novo_Nome_Entidade,
                                        Desc_Entidade ) ;
     fim_se

   CANCELA : Limpar Nome_Entidade, Novo_Nome_Entidade, Desc_Entidade;
   FIM : Executar ENCERRAR_JSD;
   HELP : Executar ATENDER_HELP;
  fim_conforme
  Obter Opção ;
 fim_enquanto
fim_modificar_entidade
```

PROCESSO JSD - NÍVEL DETALHADO

Figura 11

3.4 Analisar Consistência entre o processo JSD e a Interface Estática

Esta atividade do processo consistiu em integrar corretamente a interface estática no algoritmo JSD, pela descrição dos processos que fazem a recepção ou exibição de mensagens via telas da interface.

Assim, foram detalhados os procedimentos de tratamento de cada opção em cada tela da interface. Por exemplo, na tela da figura 10, foram verificadas como as opções Fim, Ok, Retorna, Help, ... são tratadas dentro do procedimento de Incluir-Entidade. Esta análise foi realizada para todas as telas que compõem a interface.

3.5 Construir o autômato que descreve a interação entre as telas da interface

Partindo da definição regular estática da interface expressa na notação de menus, construiu-se o correspondente autômato finito(AF), para mostrar seu comportamento em função das entradas recebidas pelas telas do projeto.

Formalmente denotamos o AF pela quintupla (Q,I,T,E0,F), onde:

Q = conjunto de estados, isto é, conjunto de telas,

I = alfabeto de entrada, isto é, opções possíveis dos menus,

T = funções de transição que mapeiam Q x I para Q, isto é, funções que possibilitam a navegação de uma tela para outra,

E0 = estado inicial ou seja tela inicial e

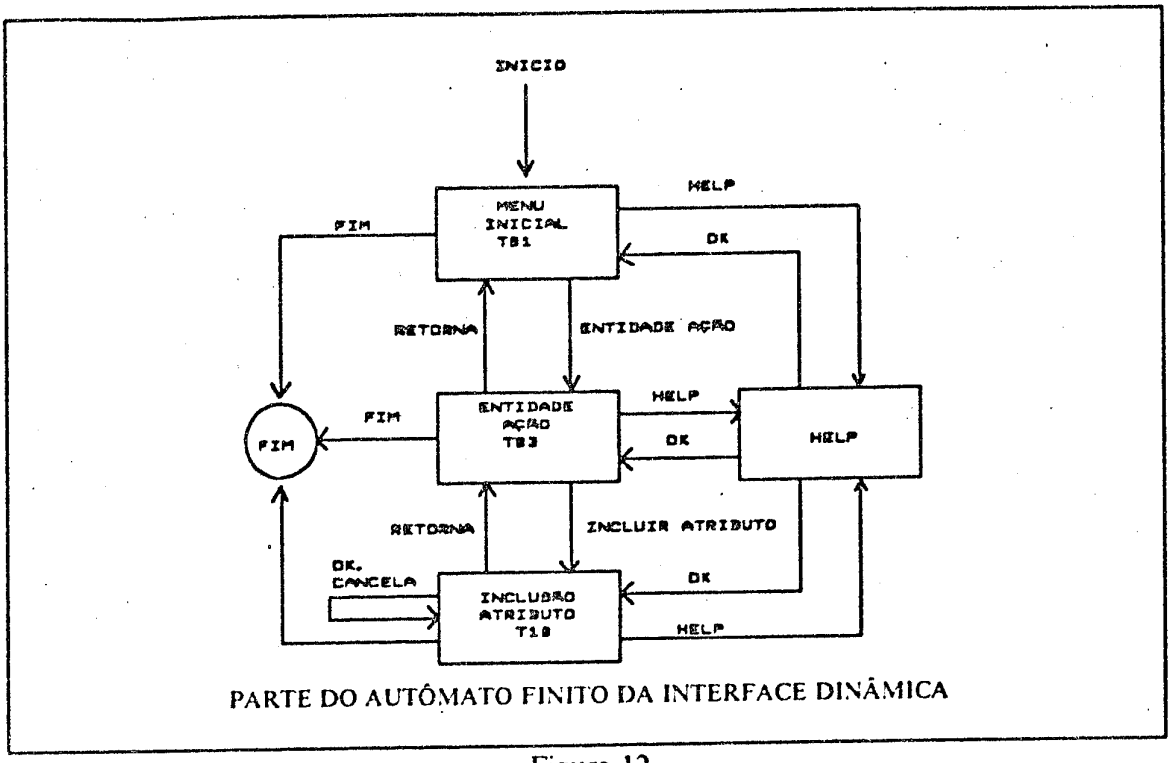
F = estado final, ou seja, tela final.

Desse modo, foi construido o diagrama de transição, mostrado em parte no grafo direcionado da figura 12, que associa ao conjunto de telas um conjunto de estados finitos e um conjunto de transições de um estado para outro, que ocorrem em função das entradas recebidas pelas telas.

O AF permite verificar a correta navegação de uma tela para outra, em função das opções escolhidas em cada situação. O AF mostra também um estado inicial, indicado pela seta rotulada com "INÍCIO" e um estado final com o nome "FIM".

3.6 Analisar consistência entre o processo JSD e a Interface dinâmica

Nesta fase do processo de desenvolvimento realizou-se uma verificação das transições previstas no AF, com o algoritmo do processo onde são descritas as transições que levam de um estado para outro, em função das entradas recebidas.



PARTE DO AUTÔMATO FINITO DA INTERFACE DINÂMICA

Figura 12

3.7 Estudar as Ferramentas utilizadas na descrição do processo (Sistema Operacional UNIX, Linguagem C, MOTIF e X-Window)

Nesta fase, iniciou-se o processo de aquisição de conhecimento das ferramentas MOTIF e X-Window, com o objetivo de dar suporte a implementação da interface do Ambiente e da linguagem C para permitir a implementação do programa do processo JSD. O conhecimento do Sistema Operacional UNIX também foi necessário uma vez que constitui a base na qual estão instaladas as demais ferramentas.

3.8 Prototipar o programa do processo JSD do Ambiente em linguagem C

Nesta fase em que já se conheciam os recursos disponíveis pode-se implementar o programa do processo JSD.

Para permitir o desenvolvimento em paralelo da interface(Motif/X-Window) e do programa do processo JSD(linguagem C), o algoritmo do processo JSD foi implementado gerando um protótipo, incluindo uma interface provisória no próprio C, enquanto se desenvolvia a interface definitiva em X-Window e Motif.

A figura 13 mostra um trecho do protótipo desenvolvido, o qual foi mantido atualizado até a fase de integração com a interface definitiva.

A interface provisória, apesar de simples, permitiu que fossem realizados os testes de verificação durante a implementação e integração do código, pois pode-se contar com eficientes depuradores que facilitaram a identificação e correção de erros.

```
void Tela4 ()
{
 printf ("\nNome da Entidade: \0" );
 scanf("%19s", NomeEntidade);
 fflush(stdin);
 printf ("\nDescrição da Entidade: \0");
 scanf("%s", DescEntidade);
 fflush(stdin);
 printf ("\n");
 printf ("1 - OK \n");
 printf ("0 - Fim \n");
 printf ("\n");
}

void GravarArqEntidade ()
{
 PontEntidade = (T_PontEntidade) malloc (tT_PontEntidade);
 strcpy (PontEntidade-> NomeEntidade , NomeEntidade);
 strcpy (PontEntidade-> DescEntidade , DescEntidade);
 PontEntidade-> PontArvore = NULL;
 PontEntidade-> PontRedeInicio = NULL;
 PontEntidade-> PontRedeFim = NULL;
 PontEntidade-> PontMerge = NULL;
 PontEntidade-> PontListaAção = NULL;
 PontEntidade-> PontListaEntidade = PontIniEntidade;
 PontIniEntidade = PontEntidade;
}

void IncluirEntidade ()
 {

 Tela4();
 Opção = getchar();
 fflush(stdin);
 if ( Opção = = '1')
 {
     ValidarEntidade();
     if (Resultado = = 'S')
        GravarArqEntidade();
     ListarEntidades();
 }

 }

 void Tela5 ()
 {
   printf ("\nNome da Entidade : \0");
   scanf("%19s", NomeEntidade );
   fflush(stdin);
   printf ("\n");
   printf ("1 - OK \n");
   printf ("0 - Fim \n");
   printf ("\n");
 }
```

TRECHO DO PROGRAMA DO PROCESSO JSD
LINGUAGEM C

Figura 13

3.9 Prototipar a interface utilizando X-Window e Motif

A codificação do programa de processo da interface seguiu o algoritmo especificado pelo autômato com os lay-outs das telas definidas na especificação estática da interface.

Para isto, foi criado um arquivo na linguagem UIL descrevendo os objetos que compõem a interface. Paralelamente, o fonte C foi alterado a fim de receber as chamadas de funções do Motif, X Intrinsics e Xlib. Graças a ferramenta Motif que possui uma linguagem de especificação de interface bastante amigável e eficiente foi possível construir a interface usando um processo altamente interativo. Isto permitiu testar e validar rápidamente a interface do sistema.

A figura 14 constitui um exemplar bem típico de interface que se pode criar de posse destas ferramentas. Na parte superior da figura ve-se o título "JSD" que faz parte da especificação de uma widget da classe CONSTRAINT. Esta classe é do tipo "Composite" e portanto está pronta para conter e gerenciar outras widgets. Neste exemplo foram criadas as seguintes classes:

a) Na parte mais ao alto da figura encontra-se uma widget da classe "menubar" que contém um conjunto de opções (help,retorna, ok, cancela, fim) que foram padronizados em todas as telas.

b) No canto esquerdo alto abaixo da "menubar" foi criado uma widget da classe "radio box" que permite que o usuário selecione apenas uma das opções apresentadas.

c) Um pouco abaixo da "radio box" encontra-se uma widget da classe "Label" que permite que qualquer conjunto de caracteres fixo (string) seja exibido na tela.

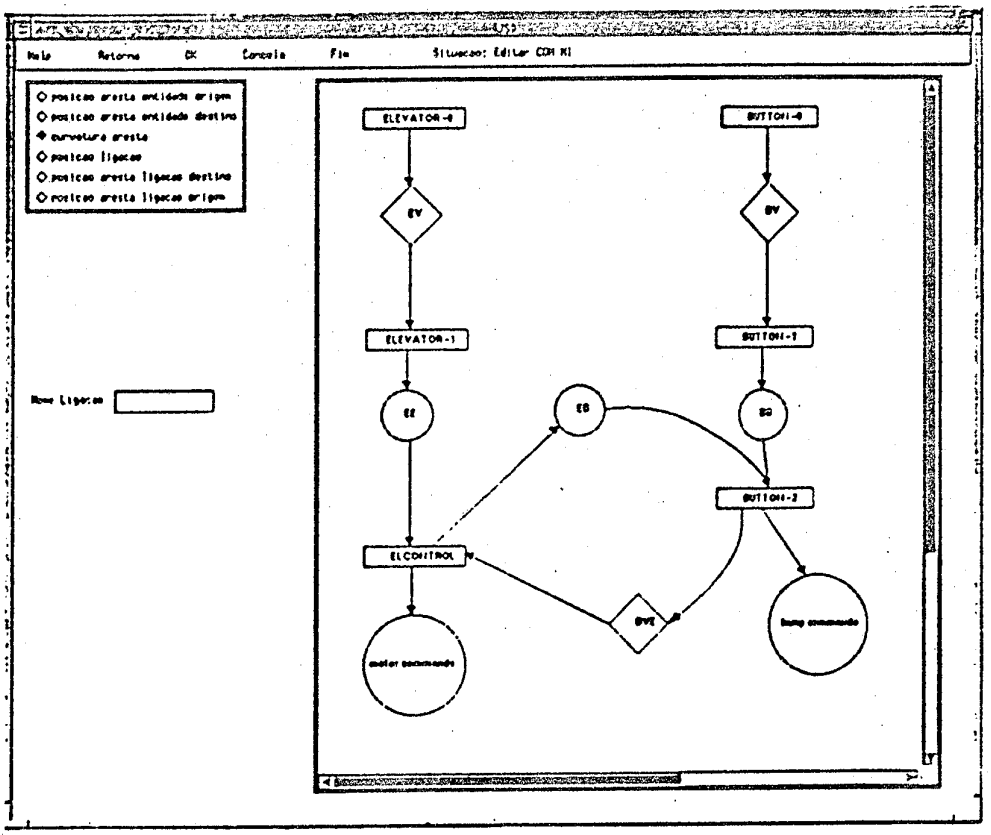
d) Ao lado do texto foi criado uma outra widget da classe "Text" que permite tanto a exibição como recepção de texto do usuário.

e) O retângulo maior à direita contém uma widget da classe "DrawingArea".

A classe "DrawingArea" faz parte de um conjunto de classes genéricas que o Motif dispõe para permitir que o analista possa criar interfaces que não estejam contidas dentro daquelas já padronizadas pelo Motif.

Neste caso, interessava criar um editor gráfico. Em tais situações, é necessário que o analista crie todas as rotinas para dar suporte a esta interface. Para facilitar este trabalho, o analista tem à sua disponibilidade não só as rotinas disponíveis na linguagem base que se está utilizando, neste caso C, como também as rotinas disponíveis por cada uma das ferramentas que compõe o ambiente de programação, no caso Xlib, X Intrinsics e Motif.

No problema em questão, utilizou-se principalmente a linguagem C e as rotinas da Xlib para criação da funcionalidade do editor. Vale ressaltar que neste nível mais baixo é importante um bom entendimento da potencialidade de cada uma das ferramentas assim como o conhecimento do seu funcionamento interno e das suas interfaces. Esta tarefa foi uma das que consumiu mais tempo no projeto devido a necessidade de se adquirir experiência com cada uma das ferramentas.



INTERFACE EDITOR GRÁFICO

Figura 14

3.10 Integrar o processo JSD em Linguagem C e a Interface em Motif/X-Window

Esta fase do processo consistiu em fazer a integração do protótipo implementado em C com a interface implementada em X-Window e Motif. Aqui a interface provisória em C do algoritmo JSD, foi substituida pela interface definitiva.

Os cuidados necessários nesta fase estão relacionados principalmente aos nomes de variáveis, procedimentos, tamanho e tipo dos dados, enfim a todos os cuidados relativos a manutenção de um eficiente dicionário de dados, agora não mais da interface ou do código C isolados mas do Produto integrado.

Para facilitar o trabalho, o sistema foi dividido em módulos, compilados separadamente e "link-editados". Assim o sistema ficou constituido por 2 grandes módulos:

- Módulo correspondente ao programa de processo que implementa o modelo JSD
- Módulo contendo as trocas de mensagens com o ambiente Motif/X-Window, correspondente à geração da interface do sistema.

3.11 Testar o Ambiente construído

Nesta última fase do processo alguns exemplos foram utilizados para testar o ambiente permitindo a identificação e correção de falhas no processo de desenvolvimento, de forma a validar o protótipo.

4 Arquitetura e Funcionalidade do Ambiente implementado

A arquitetura do ambiente de software criado permite que o analista de software desenvolva sistemas instanciados no modelo JSD, desde a especificação até o desenho do projeto. O ambiente é constituído de 4 Módulos principais, que abrangem os 5 primeiros passos da metodologia JSD:

- Entidade Ação
- Estrutura Entidade
- Modelo Rede
  - Modelo Inicial
  - Função
- Cronologia

A figura 15 mostra a tela inicial do ambiente implementado onde podem ser vistas além das opções correspondentes aos passos da metodologia JSD, opções correspondentes ao módulo que tem como função salvar em memória secundária o projeto que está sendo desenvolvido (Salva Sistema) e ao módulo que cria um novo projeto ou carrega um projeto existente.

A figura 16 mostra uma tela que permite a entrada das entidades do sistema. No passo Entidade Ação o ambiente permite que o desenvolvedor crie, remova ou modifique as entidades, ações e atributos, bem como suas descrições. O sistema verifica automáticamente a associação entidade e ação; se não existem entidades repetidas ou ações repetidas para uma determinada entidade. Remoções ou modificações de entidades, ações ou atributos inexistentes não são permitidas. Também não se pode remover uma entidade sem antes retirá-la do diagrama DES e remover seu diagrama de estrutura.

Heurísticas, em forma de "check-list", permitem que o analista verifique a validade ou não das entidades e ações.

No passo Estrutura Entidade o sistema permite a construção do diagrama de estrutura de cada entidade com suas ações, definidas obrigatóriamente na fase Modelo. A figura 17 mostra uma tela exibida no passo Estrutura Entidade.

Os nós da árvore que representa a estrutura da entidade podem ser: folha, intermediário, seleção ou repetição (Figura 18). A raiz contém sempre o nome da entidade e as folhas conterão sempre as ações sofridas ou executadas pela entidade, não podendo logicamente ter filhos. Esta verificação de consistência é também feita automáticamente. Na remoção é possivel remover um nó sem filho ou toda uma subárvore do diagrama estrutura entidade. A modificação dos nomes e tipos dos nós intermediários, seleção ou repetição é possivel, contudo não é permitido modificar o nome de um nó folha, uma vez que se trata de uma ação. Caso isto seja necessário esta ação deve ser modificada ou então incluida no passo Entidade Ação.

Na fase Rede, o desenvolvedor pode construir o Modelo Inicial do DES e/ou adicionar funções (passo Função) ao Modelo Inicial para permitir criar as saídas do projeto, como relatórios, consultas, etc, em função das entradas de dados ou em função das informações retiradas do próprio projeto.

A figura 19 mostra uma tela da fase Rede onde são exibidas as opções de edição de entidade e edição de ligações.

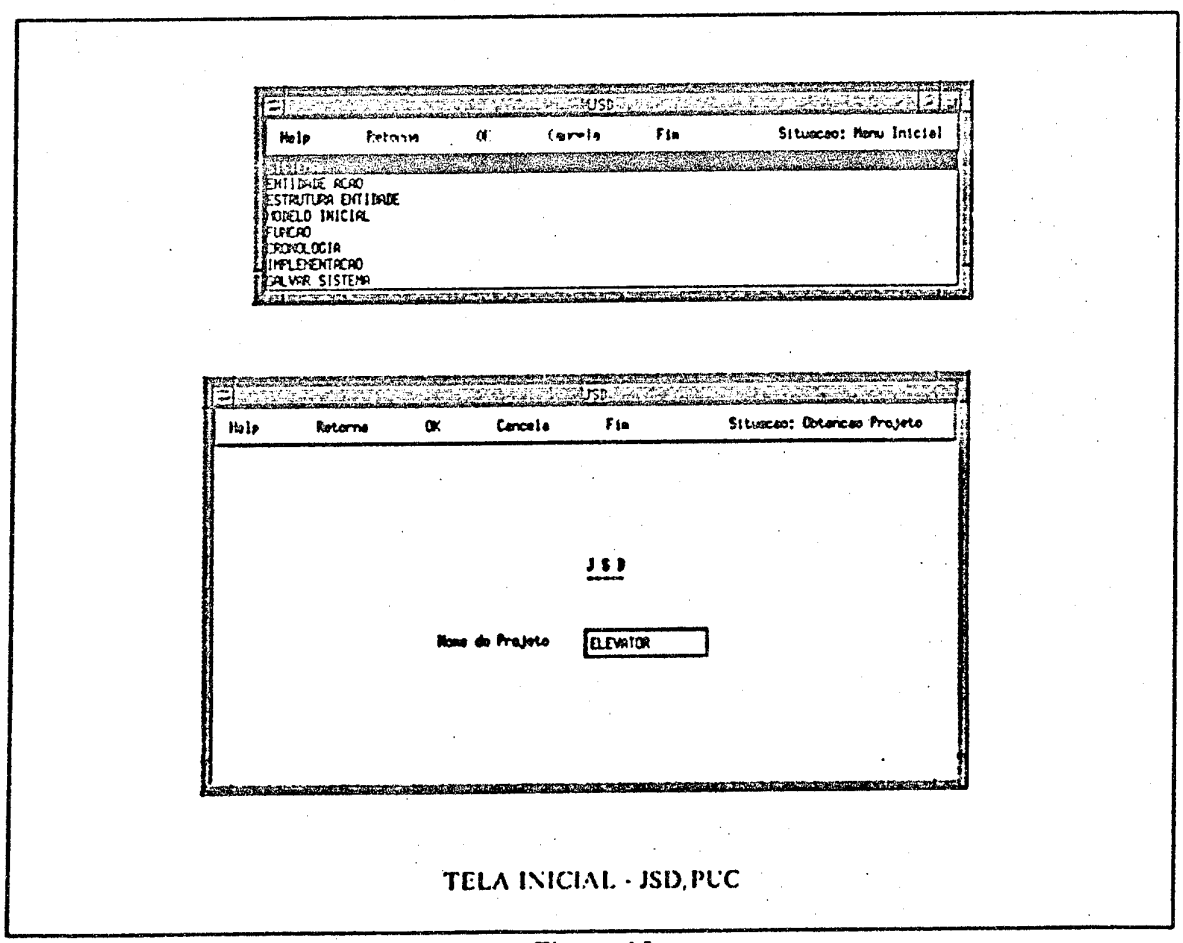
O sistema permite além da conexão dos processos por fila de mensagens("data-stream") e vetor de estado("state-vector") as ligações "fixed ou data merge" e "rough merge", de 2 ou mais processos. A manutenção da Rede é facilitada pela existência de opções de remoção de um processo e/ou de suas comunicações com outros processos. Também é possível refazer completamente o "lay-out" do diagrama da rêde, alterando-se as posições dos processos e ligações.

Caso esteja no passo Função, ainda na fase rede, pode-se incluir novas funções no sistema. Para tal basta definir suas ações e atributos e construir seu diagrama de estrutura à semelhança do que é feito no caso das entidades.

Nesta fase ainda, o desenvolvedor elabora o texto estruturado do diagrama estrutura entidade, auxiliado por um texto inicial gerado automáticamente. Esta geração automática, a partir do diagrama estrutura entidade , contém as estruturas de controle da árvore da entidade, aninhadas para facilitar seu entendimento (Figura 20).

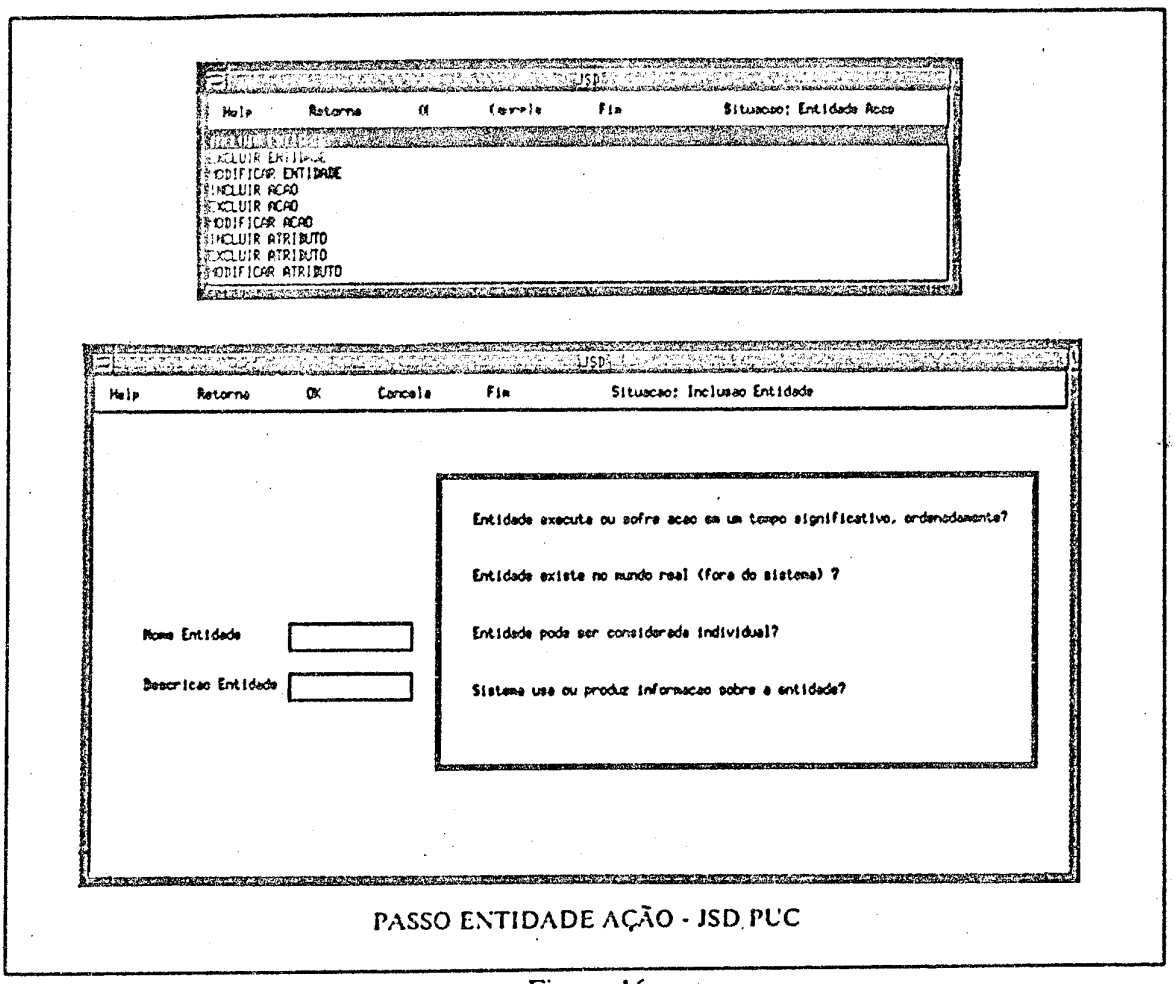
No passo cronologia um editor "full-screen" é oferecido ao desenvolvedor para que o mesmo possa anotar as restrições de tempo a serem observadas quando da implementação do projeto. Caso seja necessário a inclusão de processos de sincronização de atividades, estes devem ser adicionados no passo Função.

Todo o projeto em qualquer passo do desenvolvimento pode ser salvo em memória secundária , evitando perda em casos de acidentes ou quedas de tensão. Convém ressaltar que toda editoração gráfica foi construida utilizando-se da técnica de manipulação direta, tornando o sistema mais amigável para os usuários.



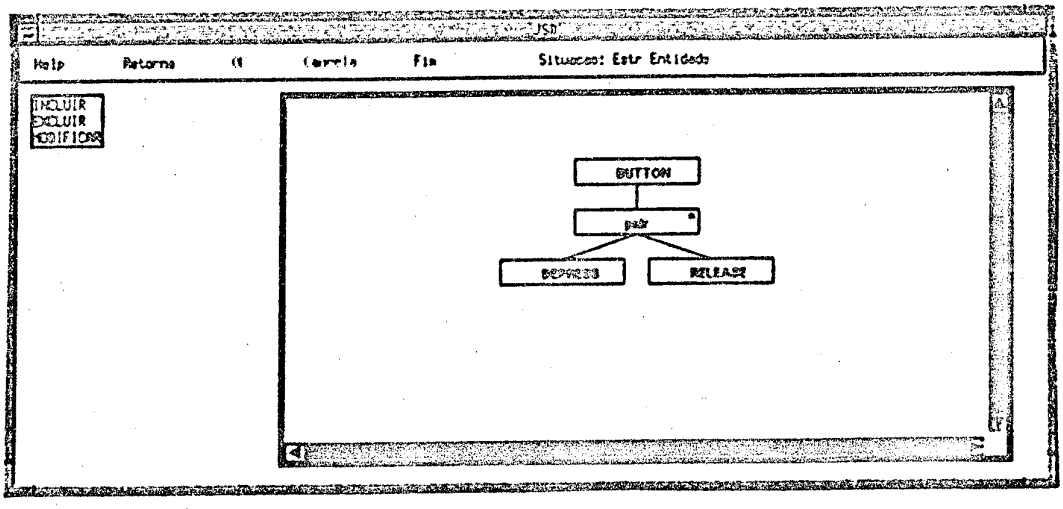
TELA INICIAL - JSD,PUC

Figura 15



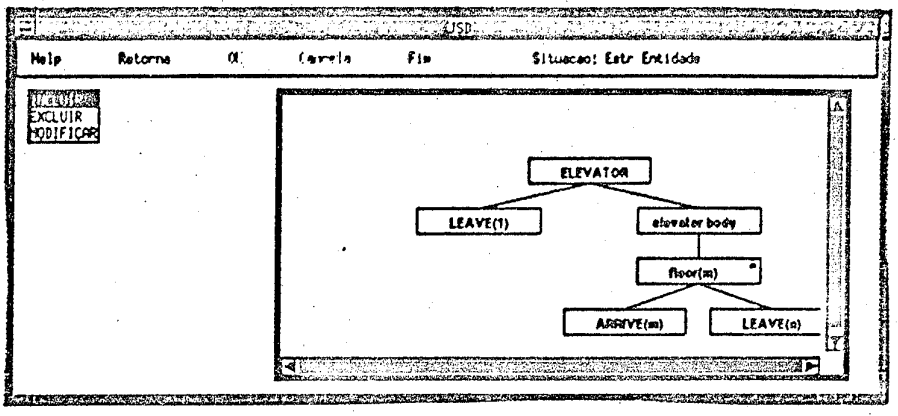
PASSO ENTIDADE AÇÃO - JSD PUC

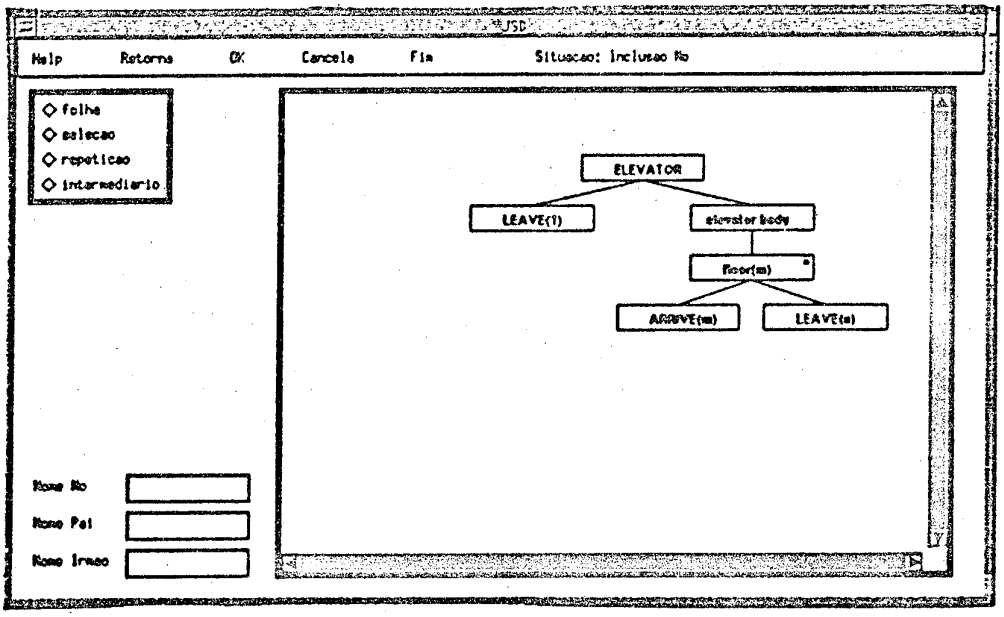
Figura 16



PASSO ESTRUTURA ENTIDADE - JSD PUC

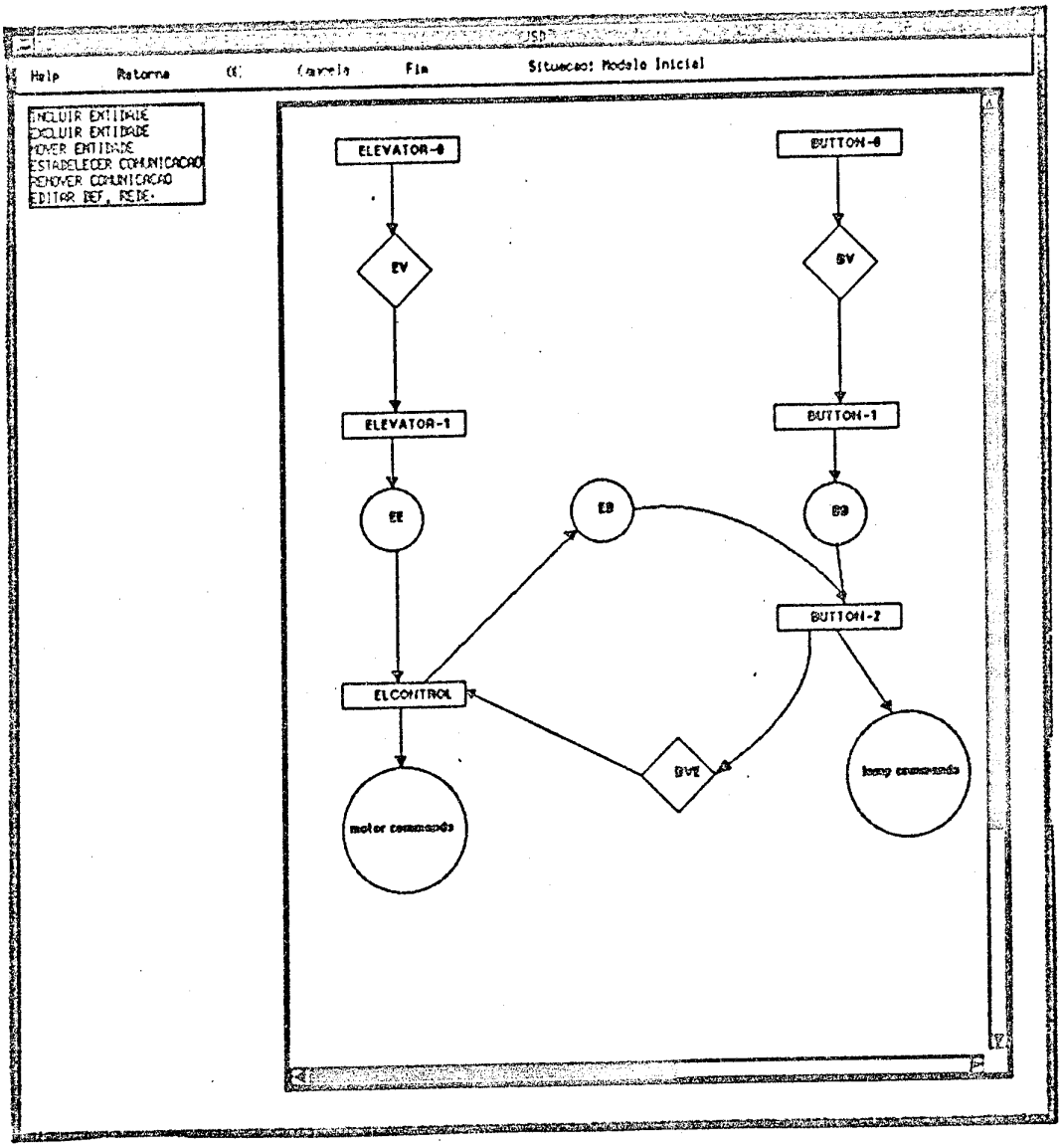
Figura 17





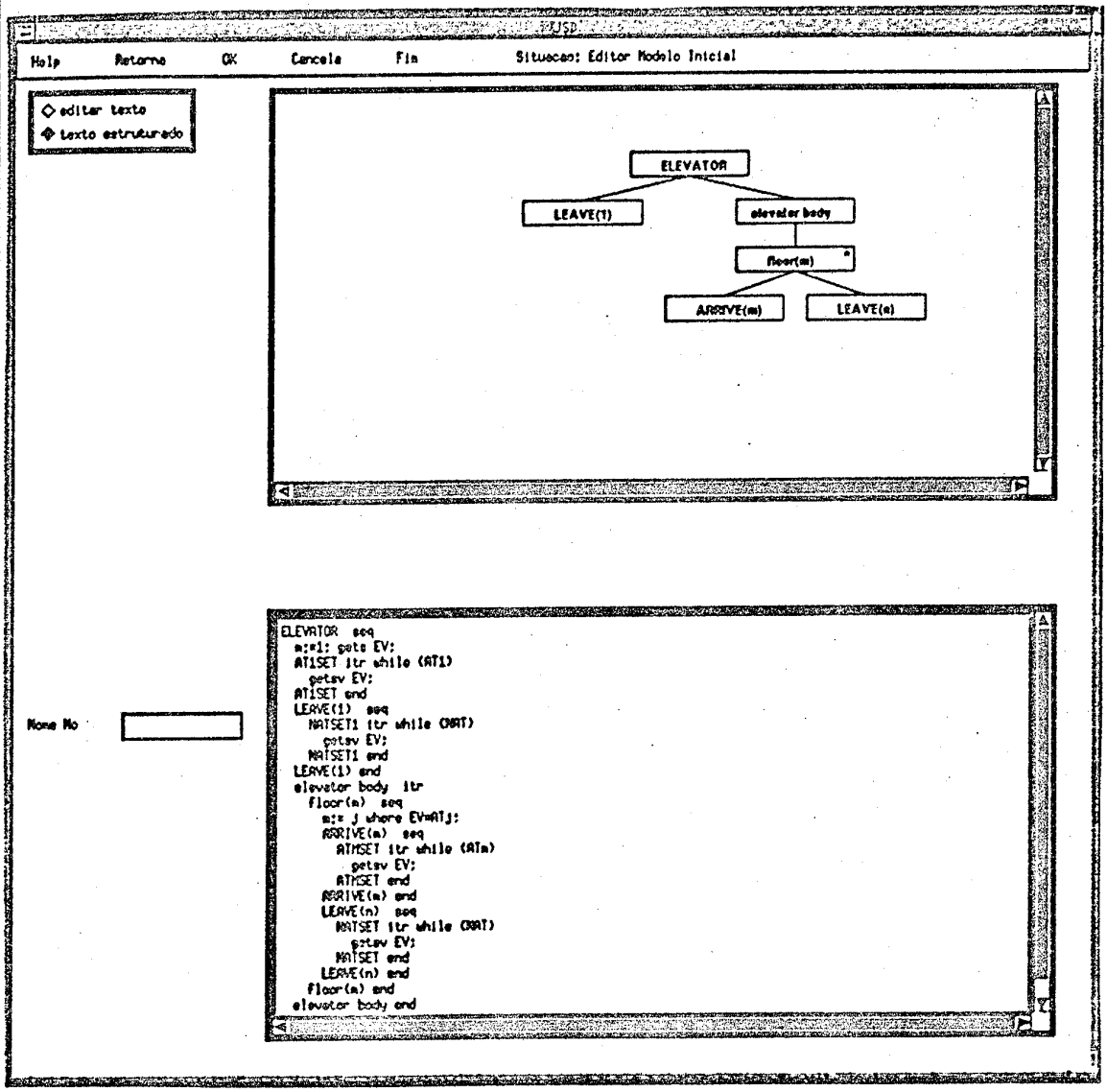
CONSTRUÇÃO DIAGRAMA ESTRUTURA ENTIDADE - JSD PUC

Figura 18



FASE REDE - JSD PUC

Figura 19



TEXTO ESTRUTURADO - JSD PUC

Figura 20

5 Reutilização da Análise do Projeto

5.1 Visão Geral do paradigma Potts

Nesta seção apresenta-se um modelo de dados para o método de Potts [Potts 88] e [Potts 89] para registro das decisões de desenho que ilustra como o ambiente JSD/PUC permite a incorporação de experimentos em engenharia de software. É a estruturação da arquitetura do ambiente na forma de um programa de processo que viabiliza projetos como o que está descrito a seguir.

Uma metodologia deve assistir o projetista de várias formas entre as quais distiguem-se pelo menos 3: descrição do projeto, estratégia e progressão, e heuristicas e raciocinios sobre as decisões de projeto.

Por outro lado, segundo Potts, em geral as tentativas de codificar as informações do processo estratégico, tem adotado um dos 3 paradigmas:

- Codificação procedural

- Gramatical

- Especificação declarativa

Conforme foi enfatizado adotamos o paradigma procedimental também conhecido como processo de programação de Osterweil, para codificar os dois componentes, descrição do projeto e estratégia e progressão dos métodos de projeto do nosso trabalho.

Para endereçar o terceiro componente dos métodos de projeto, suas heuristicas e raciocínios das decisões de projeto, propõe-se um modelo de dados para a incorporação do modelo genérico baseado no paradigma de Potts. A combinação deste paradigma com o de Osterweil torna possível o registro histórico do desenvolvimento do projeto, tal que sua análise possa ser reutilizada para manutenções ou desenvolvimento de novos projetos (por analogia a racionalização usada em outros projetos).

O modelo genérico de representação dos métodos de projeto proposto por Potts pode ser representado (ver Figura 21) com 5 tipos entidade, os quais constituem 5 classes básicas:

Artefato
Questão
Passo
Posição
Argumento

Do inter-relacionamento entre estas entidades pode-se extrair 8 relacionamentos binários:

Revisã
Sugere
Responde a
Cita
Contribui para
Suporta
Rejeita
Modifica

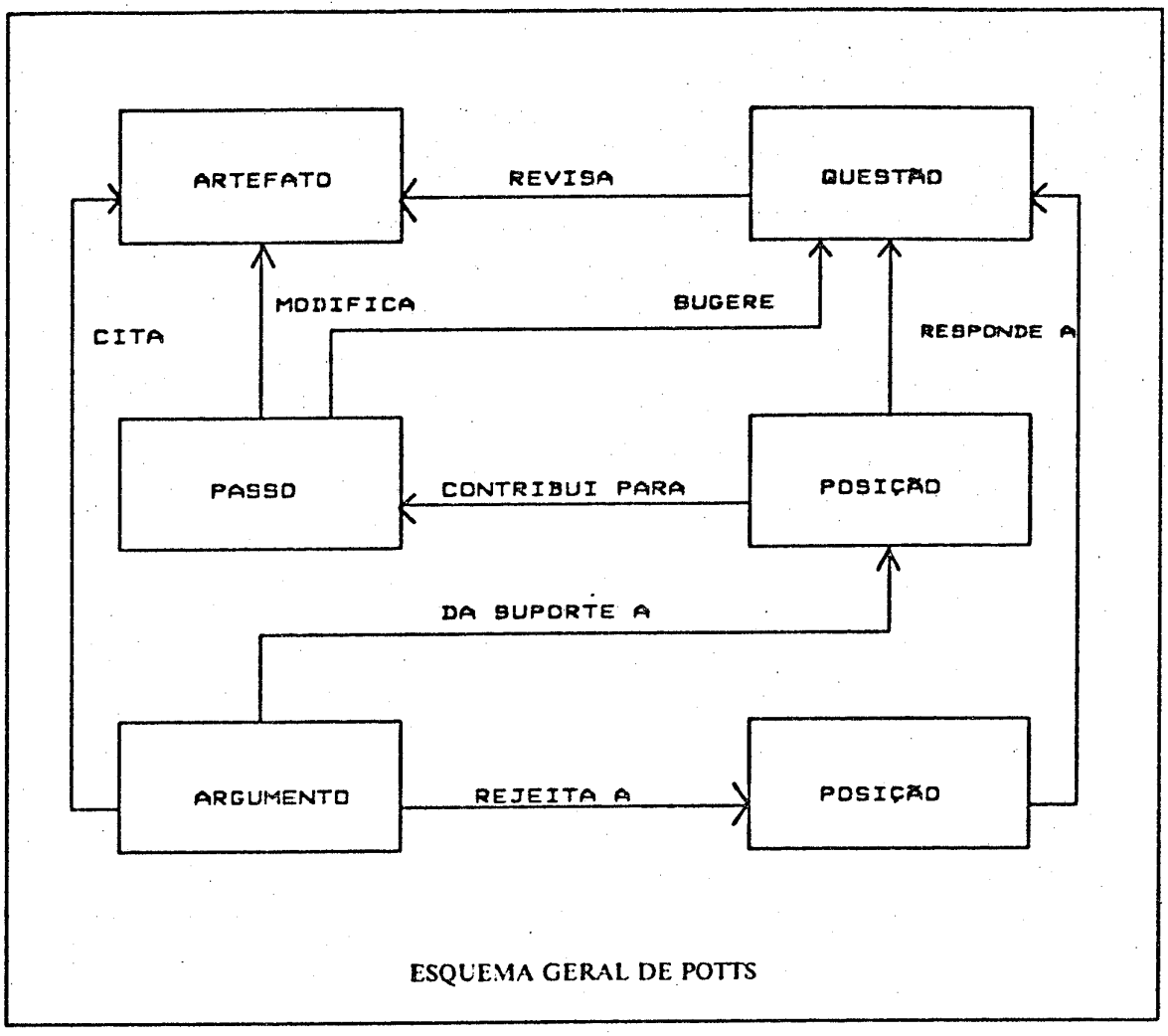


Figura 21

Objetos específicos do método são todos especializações de um destes tipos de entidades. Artefatos são os documentos específicos do método. Um passo do desenho modifica ou cria um Artefato. As representações dos raciocínios e decisões de projeto são expressadas pelas Questões, Posições e Argumentos. Uma Posição é uma resposta candidata a Questão. Uma decisão é feita quando uma Posição é tornada verdadeira (selecionada verdadeira), resolvendo uma Questão.

Resumidamente tem-se que:

- Passos: modificam ou criam um novo artefato;
- Questões: são sugeridas pelas execuções dos Passos; revisam os Artefatos;
- Posições: respondem as Questões; contribuem para os Passos, uma vez que Passo é executado quando uma série de Posições foram selecionadas;
- Argumentos: dão suporte as Posições; rejeitam as Posições; citam Artefatos (baseiam-se ou referem-se aos documentos).

Outro conceito importante no modelo de Potts é a hierarquia de especialização em que as classes entidades dos métodos de projeto podem ser customizadas formando uma hierarquia de classes para cada entidade. Este conceito será melhor visto a seguir, quando instanciando o modelo para o caso JSD.

5.2 Adequação do paradigma Potts ao paradigma procedimental de Osterweil

Para fazer registro das decisões e raciocinios tomados pelo desenvolvedor quando ele instancia o programa do processo JSD no ambiente automatizado, propõe-se a customização do modelo genérico de Potts de tal forma que seja possivel a reutilização da análise do desenho para futura manutenção ou desenvolvimento de um novo desenho.

Para adequar o modelo propõe-se a seguinte instanciação:

- A classe Passo é customizada segundo os passos do Método JSD, conforme hierarquia mostrada na figura 22;
- A classe Artefato é customizada para os documentos constantes do projeto desenvolvido no ambiente, conforme hierarquia mostrada na figura 23;
- A classe Questão é customizada segundo as verificações de projeto, baseadas em heuristicas do método JSD, em forma de checklist, conforme hierarquia mostrada na figura 24;
- A classe Posição é customizada pelas decisões tomadas quando as questões são levantadas;
- A classe Argumento, é customizada como o registro dos raciocinios adotados para dar suporte ou rejeitar as posições.

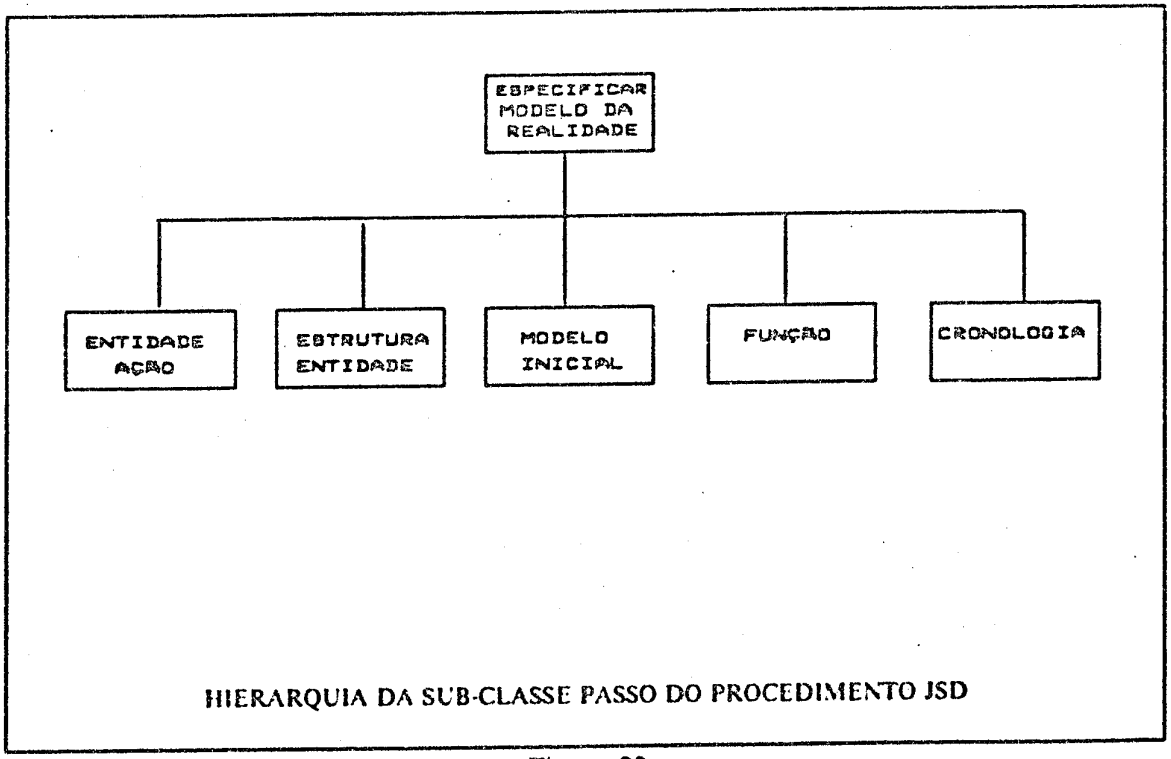
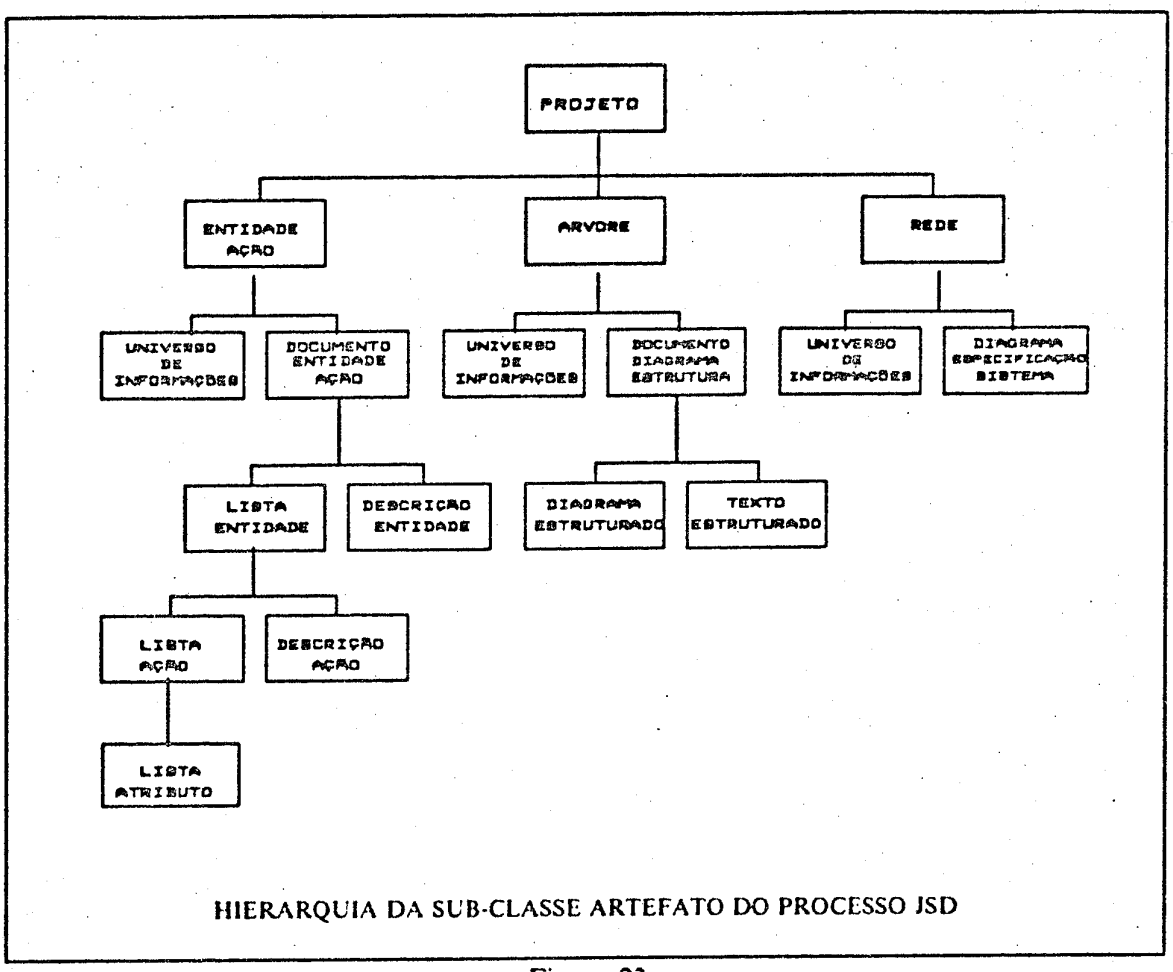
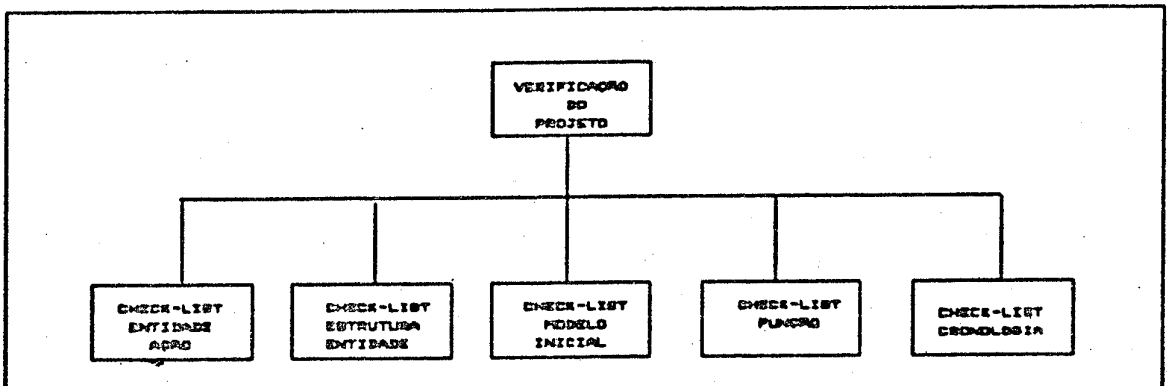


Figura 22



HIERARQUIA DA SUB-CLASSE ARTEFATO DO PROCESSO JSD

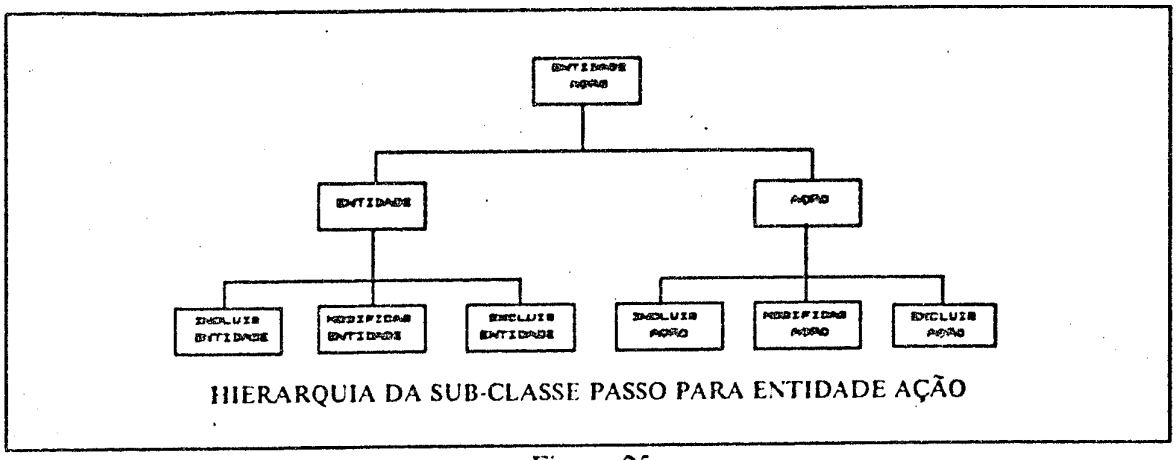
Figura 23



HIERARQUIA DA SUB-CLASSE QUESTÃO DO PROCESSO JSD

Figura 24

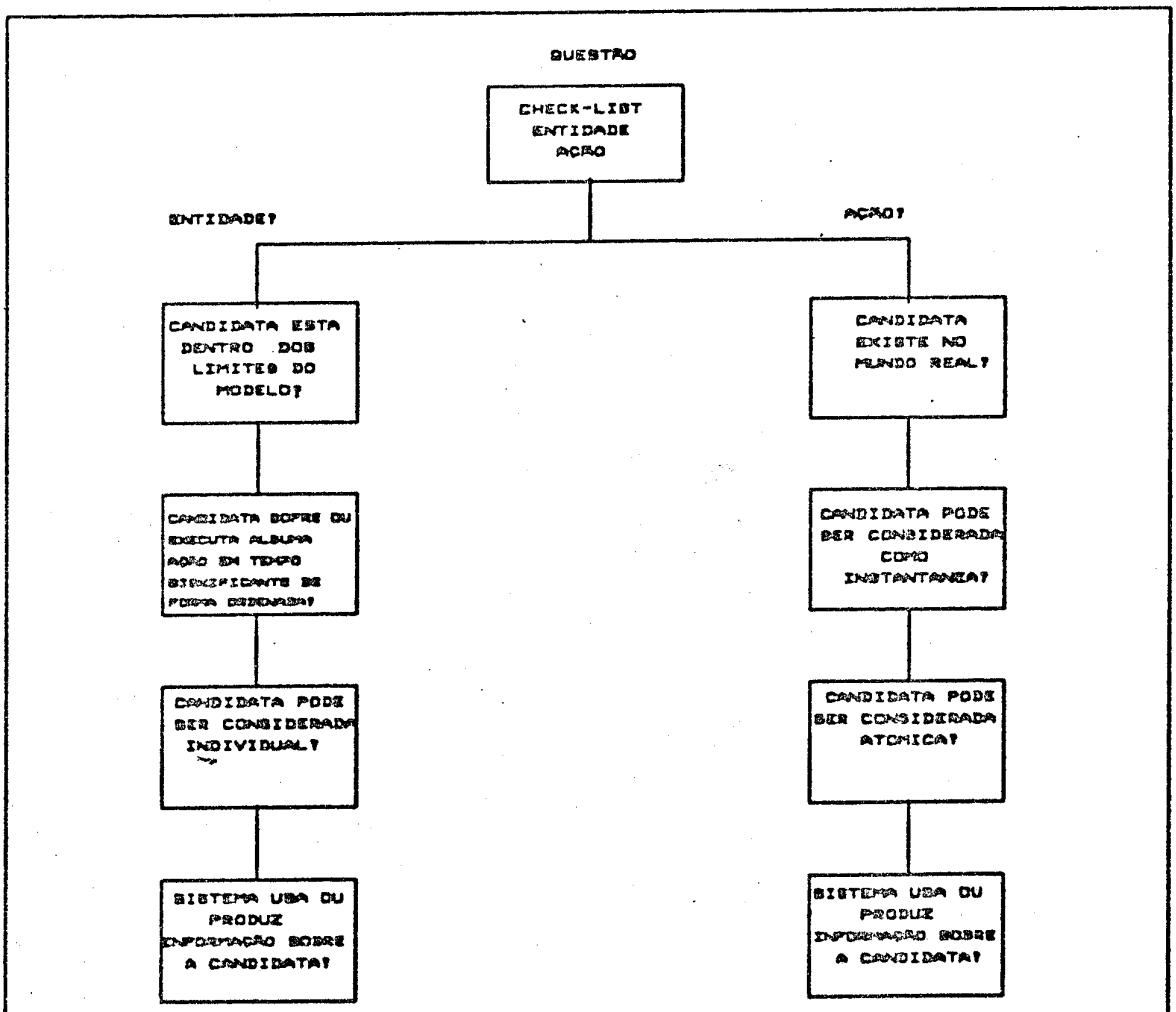
Dentro do conceito de especialização na classe tem-se por exemplo, a Figura 25 que mostra a sub-classe passo no nivel Entidade-Ação.



HIERARQUIA DA SUB-CLASSE PASSO PARA ENTIDADE AÇÃO

Figura 25

O mesmo vale para a sub-classe Questão mostrada na Figura 26, onde estão apresentadas algumas das heurísticas propostas pelo método JSD.



HIERARQUIA DA SUB-CLASSE QUESTÃO PARA ENTIDADE AÇÃO

Figura 26

Pode-se portanto, usando as estruturas implementadas para o Ambiente JSD, adaptá-las para incorporar o registro do histórico do projeto conforme modelo proposto por Potts.

5.3 Exemplos

Para caracterizar melhor a adequação proposta para o método geral de registro de decisões de Potts, dois episódios descrevem como é feito o registro das decisões de desenho no Ambiente JSD.

Considere o primeiro caso, mostrado na figura 27, onde se deseja incluir uma entidade LEITOR. Tem-se então:

- Passo: Incluir Entidade;
- Questão: É um nome dentro dos limites do desenho?;
- Artefato: Lista de Entidades e Documento que cita o nome LEITOR;
- Posição: Sim ;
- Argumento: Justificativa de que LEITOR contém um conjunto ordenado de ações que se deseja modelar.

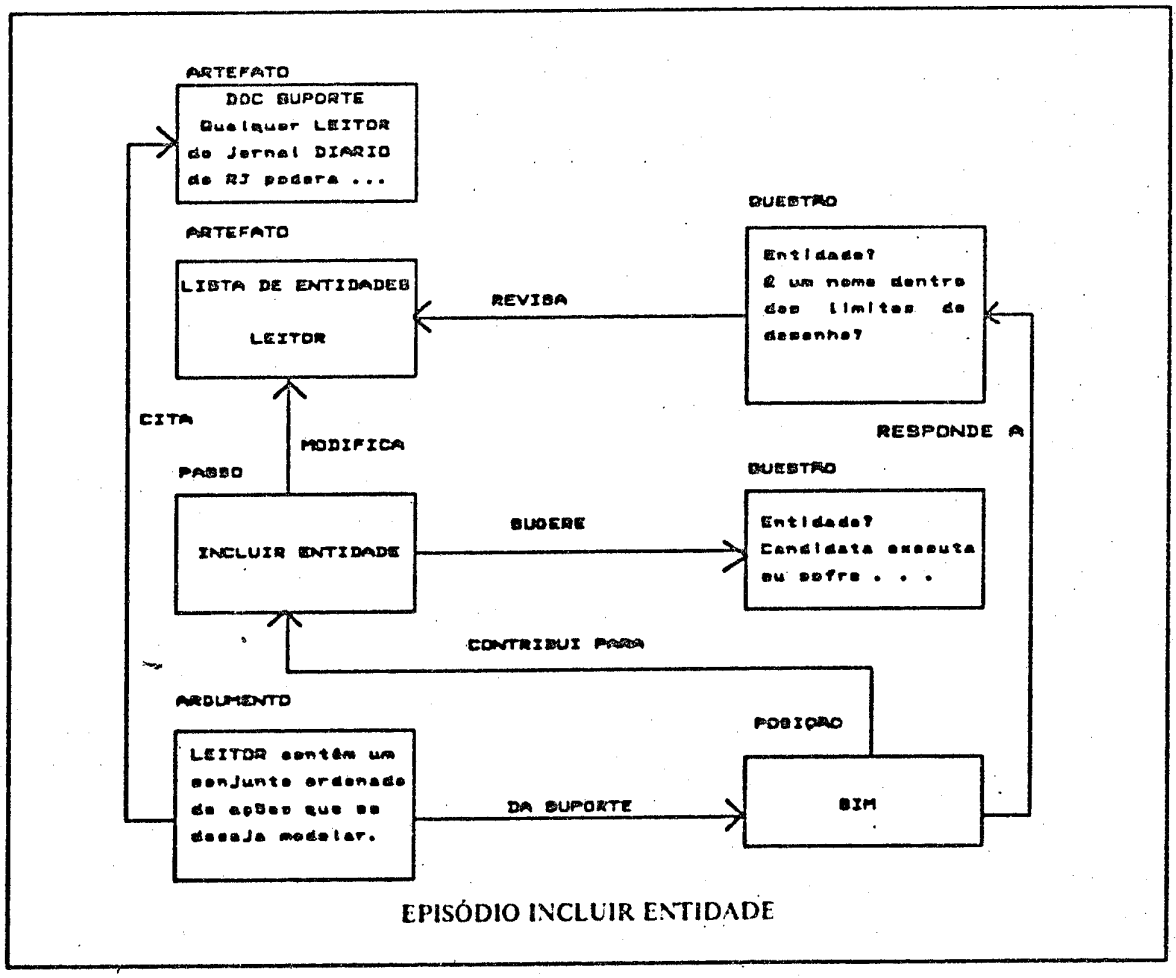


Figura 27

Conforme se pode ver a entidade LEITOR foi incluida na Lista de Entidades. A resolução desta Questão sugere uma nova Questão sobre a validade da candidata com relação as ações sofridas ou executadas e então o ciclo se repete.

O exemplo da figura 28 mostra o episódio de excluir a entidade SESSÃO da Lista de Entidades.

Semelhantemente é levantada uma Questão sobre a validade da entidade com relação as ações que ela executa ou sofre num intervalo significativo de tempo (ordenado). Com o Argumento de que "todas as ações de SESSÃO são ações da entidade JUIZES, já selecionada", a entidade SESSÃO é excluida. Novamente sugere-se uma nova Questão sobre a existência da entidade SESSÃO no Modelo Inicial(MI), uma vez que esta não pode ser retirada da lista de entidades e permanecer no MI.

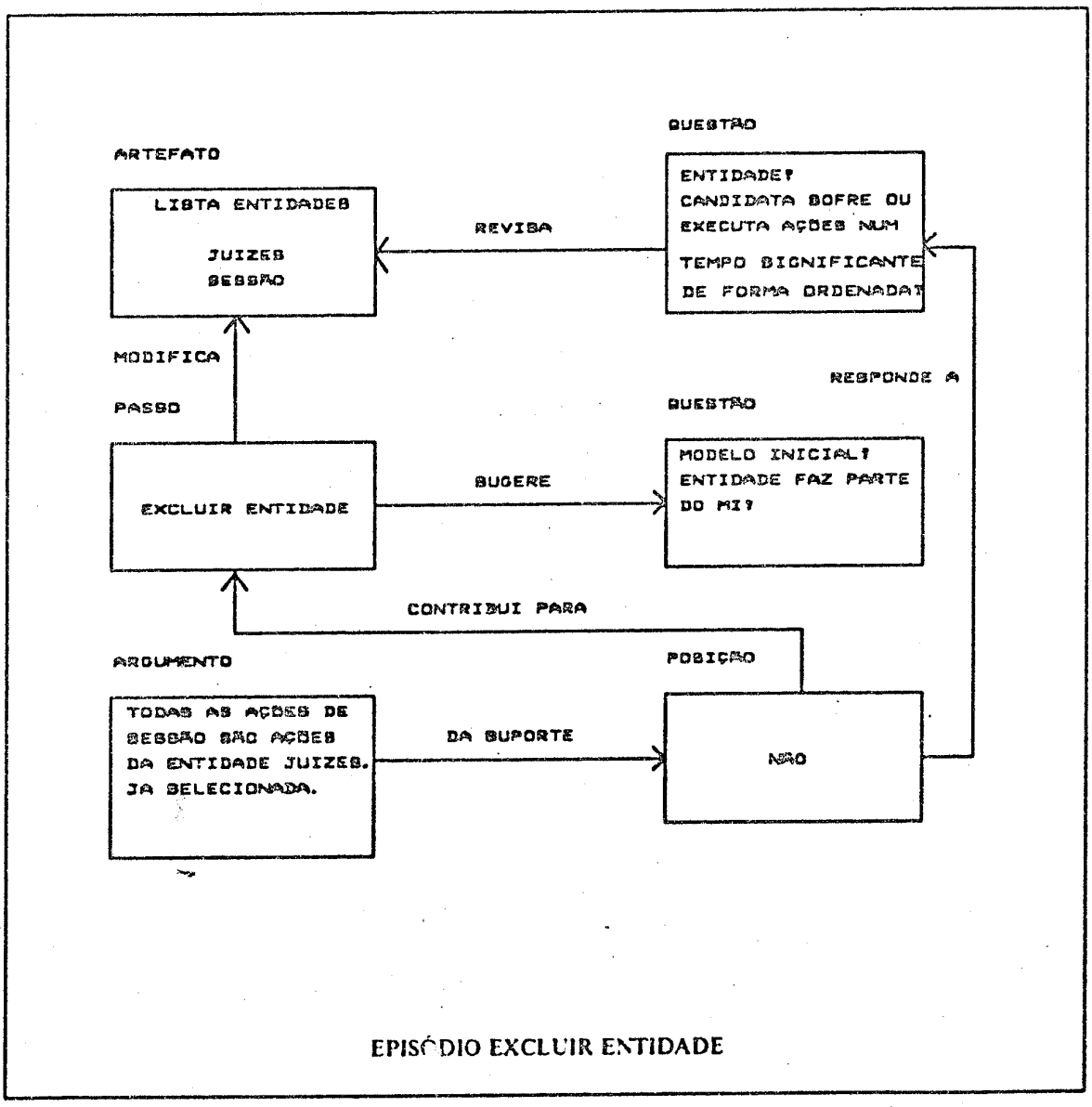


Figura 28

6 Conclusão

Apresentamos neste trabalho a concepção e a arquitetura de um ambiente acadêmico experimental voltado para a metodologia JSD. A concepção do JSD/PUC utilizou-se do programa de processo no estilo Osterweil [Osterweil 87], e do conceito de interfaces como especificações. O programa de processo utilizado foi expresso através de um diagrama estrutura/entidade e mesclou métodos pertencentes ao JSD, especificações de interfaces e conhecimento sobre as ferramentas de implementação. Procurando executar esse processo os implementadores desenvolveram o JSD/PUC na plataforma UNIX, utilizando C e Motif/X-Window.

O principal objetivo do JSD/PUC é servir como laboratório para experimentos e desenvolvimentos na área de desenho de software. O primeiro destes trabalhos [Prado 91], foi a integração no programa de processo de JSD/PUC do modelo de Potts para registro de decisões e justificativas de desenho. Observou-se que o processo de inclusão dessa funcionalidade ao ambiente, não apresentava maiores problemas, principalmente porque conseguiu-se descrever o modelo como um processo no estilo Osterweil.

Encontra-se em execução a efetiva implementação do modelo de Potts no ambiente JSD/PUC, que assim passará a dispor de facilidades de apoio a manutenção e reuso de desenhos. Dois outros trabalhos também encontram-se encaminhados, um é um exercício de Re-Engenharia [Leite 91] da própria ferramenta através do uso do próprio JSD como instância da programação de processos, e o outro é a extensão da metodologia JSD e do modelo de Potts para dar suporte a um processo de desenvolvimento cooperativo.

7 Referências Bibliográficas

[Bischofberger 89] Bischofberger, W.; Keller,R.; "Enchancing The Software Life Cicle by Prototyping", Structured Programming 1, 47-59, 1989.
[Cabral 90] Cabral, R. H. B.; Campos, I.M.; Cowan, D.D.; Lucena , C.J.P. "Interfaces as Specifications in The MIDAS USER Interface Development System ", Software Engineering NOTES , ACM SIGSOFT, Vol. 15, no. 2 , abril 90.
[CSTB 90] Computer Science Technology Board, "Scaling Up: A Research Agenda for Software Engineering", Communications of the ACM, março 1990.
[Habermann 86] Habermann, A. N. et al, "Gandalf: Software Development Environment ", IEEE Trans. on SE, Vol. 12 no. 12, 1986
[Horowitz 86] Horowitz, E. et al," SODOS: A software Documentation Support Environment - its Definition ", IEEE Trans. on Software Engineering, vol. 12 , no.8, 1986.
[Jackson 83] Jackson, M.A.;"System Development", Prentice Hall, 1983.
[Lamsweerde 88] "Generic Life Cicle Support in the ALMA Environment ", IEEE Trans. Software Engineering, Vol. 14, no. 6, 1988.
[Lee 91] Lee,V.; Lai, KoY; "Whats in Design Rationale", MIT CCS Tech. Report 118, 1991.
[Leite 91] Leite, Julio Cesar S. P., Franco, Ana Paula Moreira, Departamento de Informática, PUC-RJ, 1991.
[Maher 90] Maher,M.L.; "Process Models for Design Synthesis", AI Magazine, Winter, 1990.
[OSF/Motif 90] "Motif Programmer's Reference", Open Software Foundation, Prentice-Hall, 1990.
[Osterweil 87] Osterweil, L.; "Software processes are software too", Proc. 9th Int. Conf. Software Eng, IEEE Comp. Soc. Press, 1987.
[Perry 88] Perry, Dewayne E., Kaiser, Gail E. Models of Software Development Environments, IEEE Comp. Soc. Press, 1988.
[Potts 88] Potts,Colin; Bruns,Glenn; "Recording the Reasons for Design Decisions", IEEE Comp. Soc. Press, 1988.
[Potts 89] Potts,Colin; "A Generic Model for Representing Design Methods", ACM, 1989.
[Prado 91] Prado, Antonio Francisco; Lucena, Carlos J. P.; Leite, Julio Cesar S. P.; "Registro de Decisões e Justificativas de Desenho em Softwares Projetados com a Metodologia JSD", Monografia 10/91, Departamento de Informática, PUC-RJ, 1991.
[Willians 88] Williams, L. "Software process modeling: a behavioral approach", Proc. 10th Int. Conf. Software Eng. IEEE Comp. Soc. Press, 1988.