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

Carlos José Pereira de Lucena, Julio Cesar Sampaio do Prado Leite,
José Rodrigues Fernandes, Mario Gheiner e Antonio Francisco do Prado

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

Resumo: O presente trabalho procura mostrar o uso da automação aplicada à Engenharia de Software. Em particular focamos nosso estudo na automatização do processo de desenvolvimento de software, instanciado para o método JSD. A arquitetura do ambiente é baseada em um programa de processos no estilo de Osterweil que integra, através de um procedimento, as etapas do método 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 à 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

A engenharia de software levou algum tempo para aplicar a si própria os benefícios da automação. Ao contrário das outras

engenharias que desde de muito já se utilizam de ferramentas de desenho (*Computer Aided Design*) e produção (*Computer Aided Manufacturing*), o apoio à produção de software através de ferramentas CASE (*Computer Aided Software Engineering*) (Chikofsky, 1993) só recentemente chegou ao mercado, e ainda é um tema que vem demandando esforços de pesquisas. Essas pesquisas são normalmente associadas ao termo ambientes de software, que procuram não só apoiar a produção de documentos mas procuram também automatizar o processo de produção.

Nesse artigo reportamos o desenho e implementação de um desses ambientes, que além de dar suporte a produção de documentos aliados à um método de produção de software JSD (Masiero, 1992) oferece também facilidades para guiar e registrar as decisões de desenho tomadas no processo de construção. Nossos resultados apontam para a viabilidade de ferramentas CASE mais poderosas e que de fato assistam o projetista de software na árdua tarefa de projetar arquiteturas de software.

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, 1990)). A prática universal tem sido a produção de ambientes acadêmicos (exemplos: Habermann et alii, 1986; Lamsweerde, 1988 e Horowitz, 1986) por equipes de estudante 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, 1987) como integrador de métodos para a metodologia escolhida para o ambiente. Também foi praticada a técnica de interfaces como

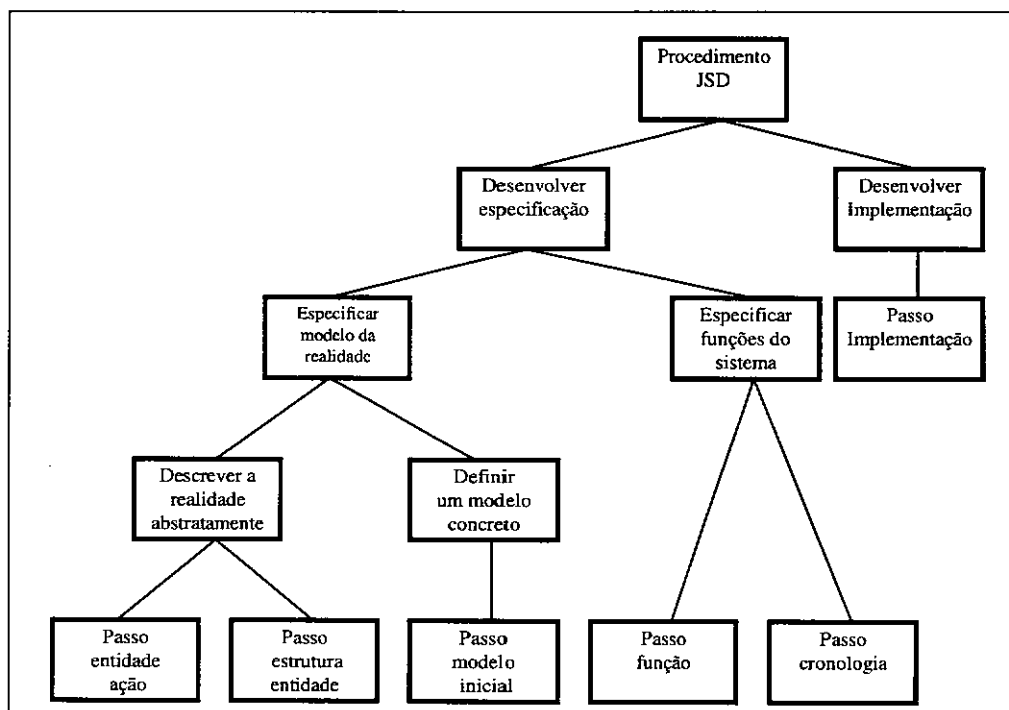


Figura 1 - Procedimento JSD

especificações(Cabral *et alii*, 1990 e Bischofberger & Keller, 1989) para a especificação do ambiente. Esta tarefa foi tornada viável pelo uso de software genérico para interfaces OSF/Motif, 1990).

A escolha da metodologia recaiu sobre JSD (Jackson, 1983) em função da variedade de métodos bem definidos que ela compreende. Soma-se a isto o fato de 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 (1990).

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, 1987) 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 & Bruns, 1988 e Potts, 1989), que permite o registro da racionalização da atividade de desenho ("design rationale" (Lee & Lai, 1991)).

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: no capítulo 2 fazemos um resumo da metodologia JSD; no capítulo 3 apresentamos o programa de processo utilizado no JSD/PUC e detalhamos as atividades da criação do JSD/PUC; no capítulo 4 descrevemos a arquitetura e funcionalidade do ambiente implementado; no capítulo 5 falamos sobre a integração das idéias de Potts ao nosso ambiente; e concluímos no capítulo 6

apontando trabalhos futuros a serem efetuados sobre o ambiente JSD/PUC.

2 Visão geral do JSD (Jackson System Development)

JSD (Jackson, 1983) é uma metodologia para especificação e projeto de sistemas de software que distingue 3 importantes fases:

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

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

- Entidade Ação
- Estrutura da 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, representadas no próprio Diagrama da Estrutura JSD, conforme mostra a 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 seqüencialmente, 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 heurísticas 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 seqüenciais, 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 círculo.

Na conexão por vetor de estado, mostrada 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.

O Modelo Rede do sistema contém não só os processos correspondentes às entidades, mas também processos que

fornecem funcionalidade ao modelo.

Estes processos são estabelecidos no passo Função e têm 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.

Funções podem ser embutidas em um processo desde que não haja mudanças na estrutura de controle do processo hospedeiro; neste caso a função tem que ser fatorada. Por exemplo, inclusão de uma 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.

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 saídas na periferia do sistema. O diagrama de estrutura, com seu texto estruturado, mostra os detalhes da especificação de cada processo.

Estas especificações 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 disponíveis,
- 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 acrescentado ao sistema e os processos do DES são invertidos em razão dos processos que os invocam.

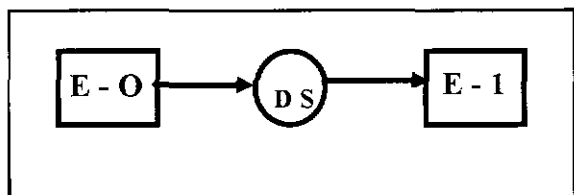


Figura 2 - DES - Conexão por Fila de Mensagem

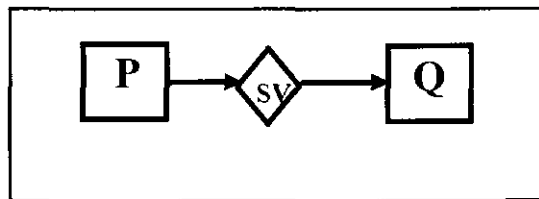


Figura 3 - DES - Conexão por Vetor de Estado

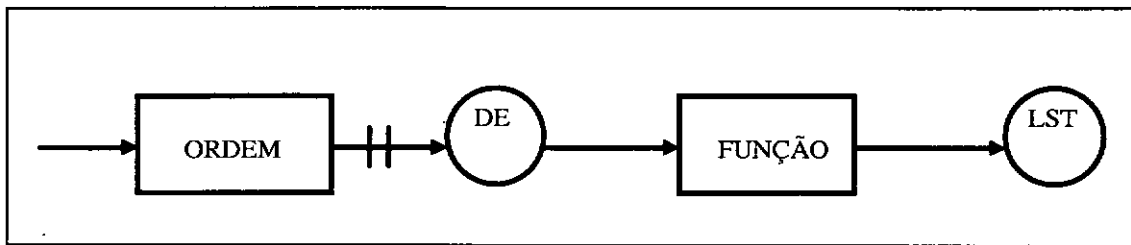


Figura 4 - DES - Passo Função

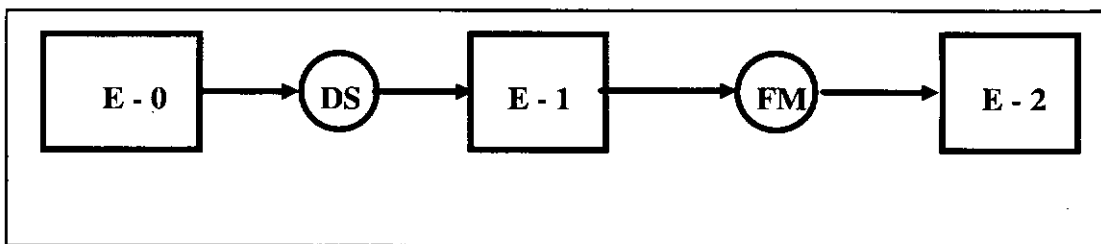


Figura 5 - DES - Nível 2

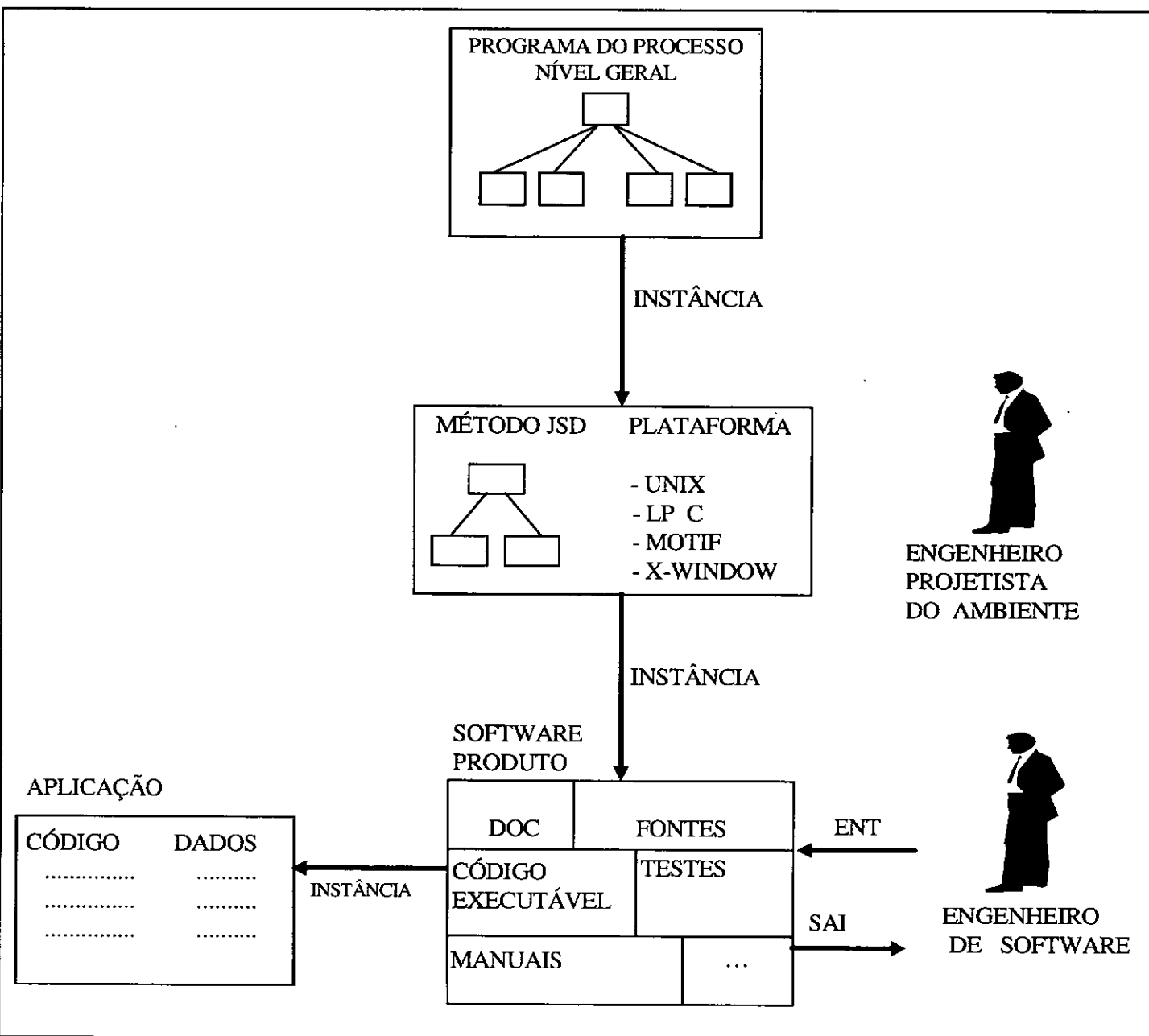


Figura 6 - Processo de Desenvolvimento de Software

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 específicos 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 veículo para a realização de um trabalho, a descrição de um processo é uma especificação de como o trabalho deve ser feito (Osterweil, 1987). Analogamente, a receita para fazer um bolo corresponde à 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 possível 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 disponíveis seguindo o programa do processo de desenvolvimento de software em diferentes níveis 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 nível da programação da aplicação pelo usuário.

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

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 nível bem alto de abstração pode-se expressar a automatização do processo de desenvolvimento de software conforme está mostrado na Figura 7.

Usando esta descrição genérica para criação do Ambiente de Apoio à 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 constituíram as grandes atividades do trabalho de implementação do JSD/PUC. Elas se encontram listadas a seguir:

- a) Programar o processo JSD - Nível Geral
- b) Especificar Interface Estática com o Ambiente
- c) Programar o processo JSD - Nível 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 construído

A seguir será descrita 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.

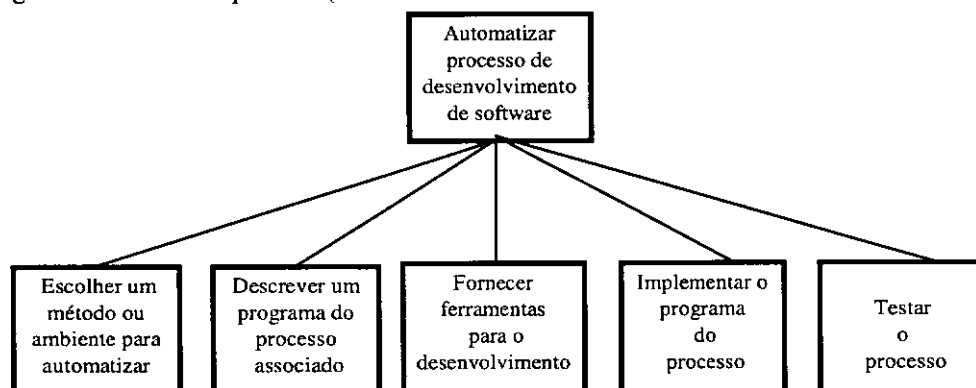


Figura 7 - Processo de Desenvolvimento de Software - Representação Procedimental

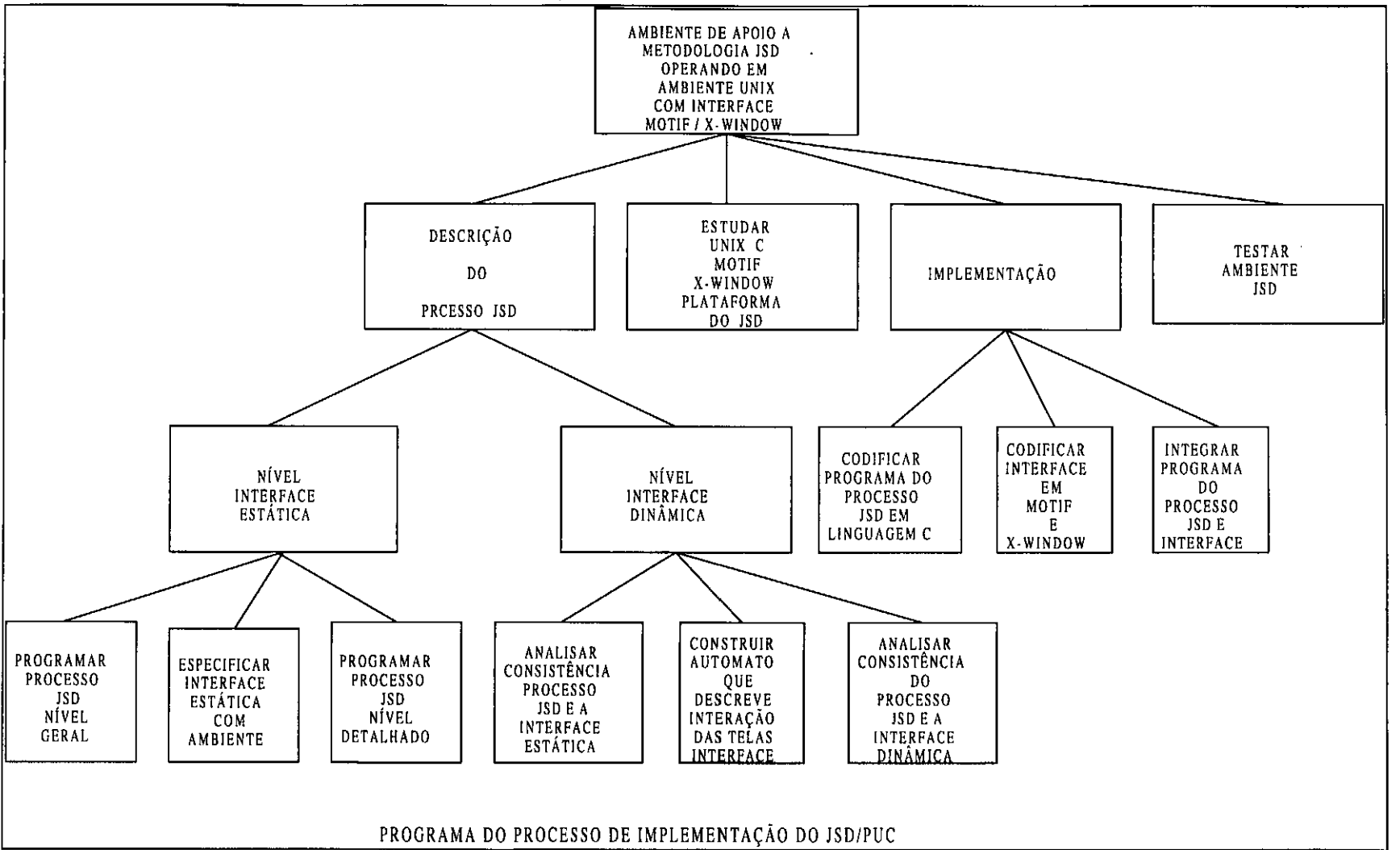


Figura 8 - Programa do Processo de Implementação do JSD/PUC

```

Proc PRINCIPAL

Obter Opção ;

Enquanto Opção não for = "RETORNA" faça

    Conforme Opção

        PROJETO : Executar OBTER_PROJETO (Nome_Projeto) ;

        PASSO_ENTIDADE_AÇÃO : Executar ATENDER_PASSO1 ;

        PASSO_ESTR_ENTIDADE : Executar ATENDER_PASSO2 ;

        PASSO_MOD_INICIAL : Executar ATENDER_PASSO3 ;

        PASSO_FUNÇÃO : Executar ATENDER_PASSO4 ;

        PASSO_CRONOLOGIA : Executar ATENDER_PASSO5 ;

        PASSO_IMPLANTAÇÃO : Executar ATENDER_PASSO6 ;

        HELP : Executar ATENDER_HELP ;

        FIM : Executar ENCERRAR_JSD ;

    fim_conforme

Obter Opção ;

fim_enquanto

fim_PRINCIPAL

```

Figura 9 - Processo JSD - Nível Geral

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 algorítmica 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 procedimentos de programa de processos de mais baixo nível.

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

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 do usuário e a que o sistema deveria fornecer automaticamente,
- ao uso de "defaults" para complementar as opções dos menus.

Cabe observar o uso da interface como especificação, ou seja, através da definição de telas e das opções do menu estamos também especificando o produto.

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 nível de refinamento do procedimento Modificar-Entidade

Help	Retorna	Fim	OK	Cancela	Situação : Inclusão Entidade
<hr/>					
Nome Entidade		_____			
Descrição Entidade		_____			

Figura 10 - Interface Estática - Tela Inclusão de 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

      CANCELAR : 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

```

Figura 11 - Processo JSD - Nível Detalhado

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 $Q \times I$ 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 construído 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.

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 à 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 pôde-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 durante a implementação e integração do código, pois pôde-se contar com eficientes depuradores que facilitaram a identificação e correção de erros.

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 à 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 rapidamente 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 vê-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 padronizadas em todas as telas.
- b) No canto esquerdo alto abaixo da "menubar" foi criada uma "widget" da classe "radio boxe" que permite que o usuário selecione apenas uma das opções apresentadas.
- c) Um pouco abaixo da "radio boxe" 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 criada uma outra "widget" da classe "Text" que permite tanto a exibição como a 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 de 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 em cada uma das ferramentas que compõem 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 à necessidade de se adquirir experiência com cada uma das ferramentas.

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 substituída 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 à 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 constituído 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 e Ação
- Estrutura da Entidade
- Modelo Rede
- Modelo Inicial
- Função
- Cronologia

O passo implementação não foi implementado, ficando para trabalho futuro, após se obter experiências com o uso do ambiente.

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.

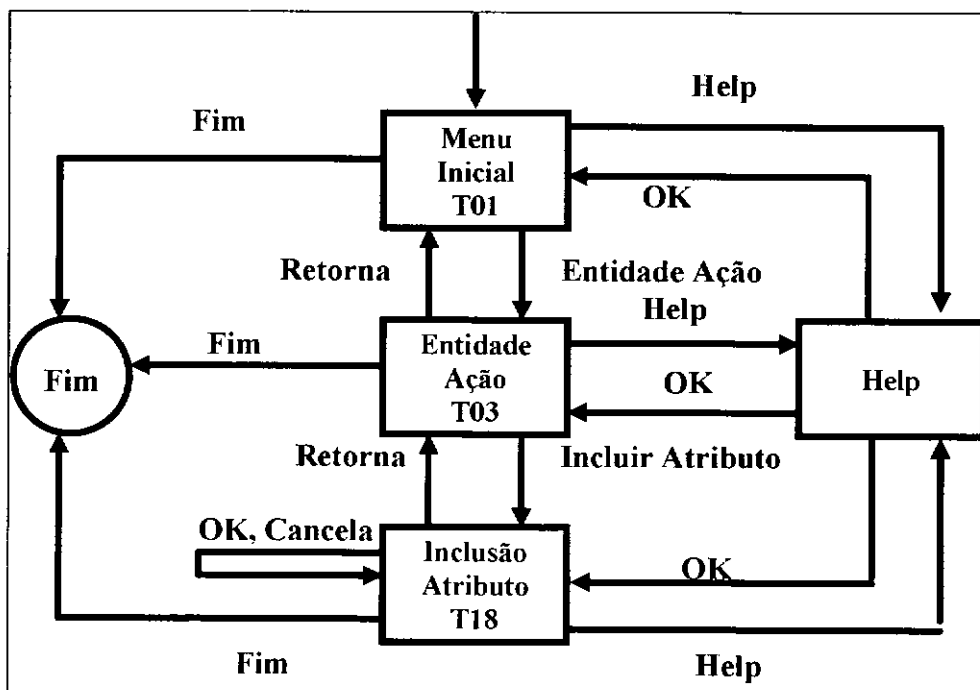


Figura 12 - Parte do Autômato Finito da Interface Dinâmica

```

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

void Gravar ArqEntidade ( )
{
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 Incluir Entidade ( )
{
Tela4 ( );
Opção = getchar ( );
fflush ( stdin );
if ( Opção == '1' )
{
ValidarEntidade ( );
if ( Resultado == 'S' )
GravarArqEntidade ( );
ListarEntidade ( );
}
}

void Telas5 ( )
{
printf ( "\n Nome da Entidade : \0" );
scanf ( "% 19s" , NomeEntidade );
fflush ( stind );
printf ( "\n" );
printf ( "1 - OK \n" );
printf ( "0 - Fim \n" );
printf ( "\n" );
}

```

Figura 13 - Trecho do Programa do Processo JSD - Linguagem C

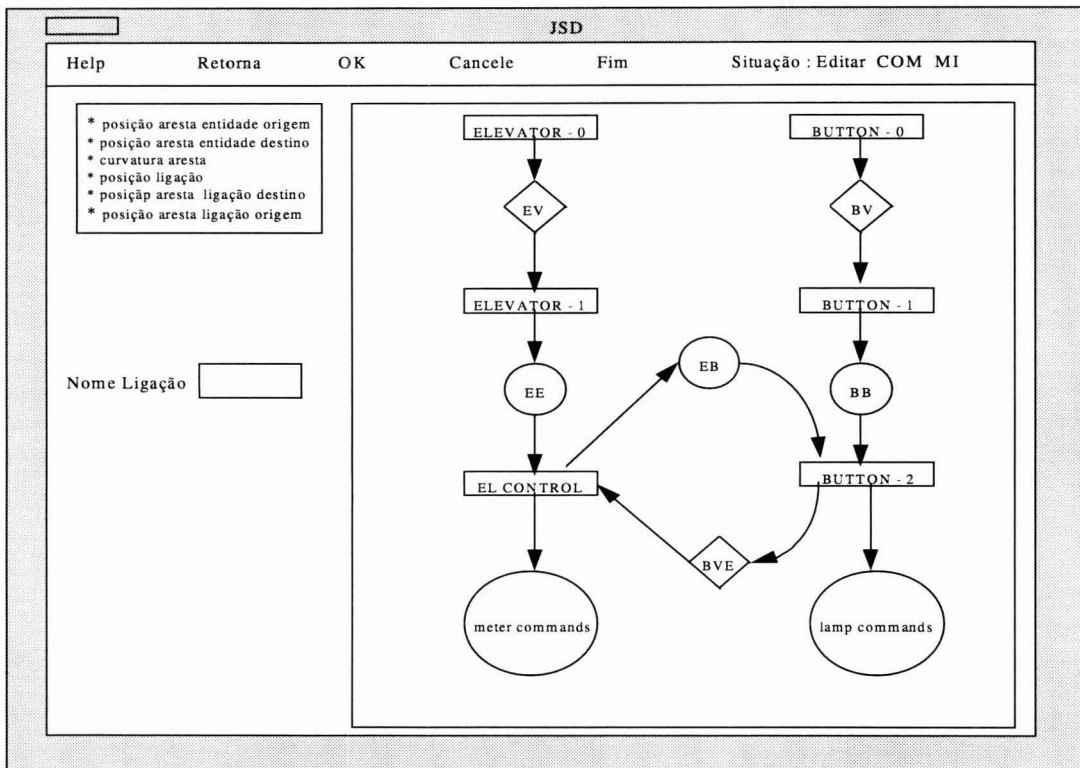


Figura 14 -Interface Editor Gráfico

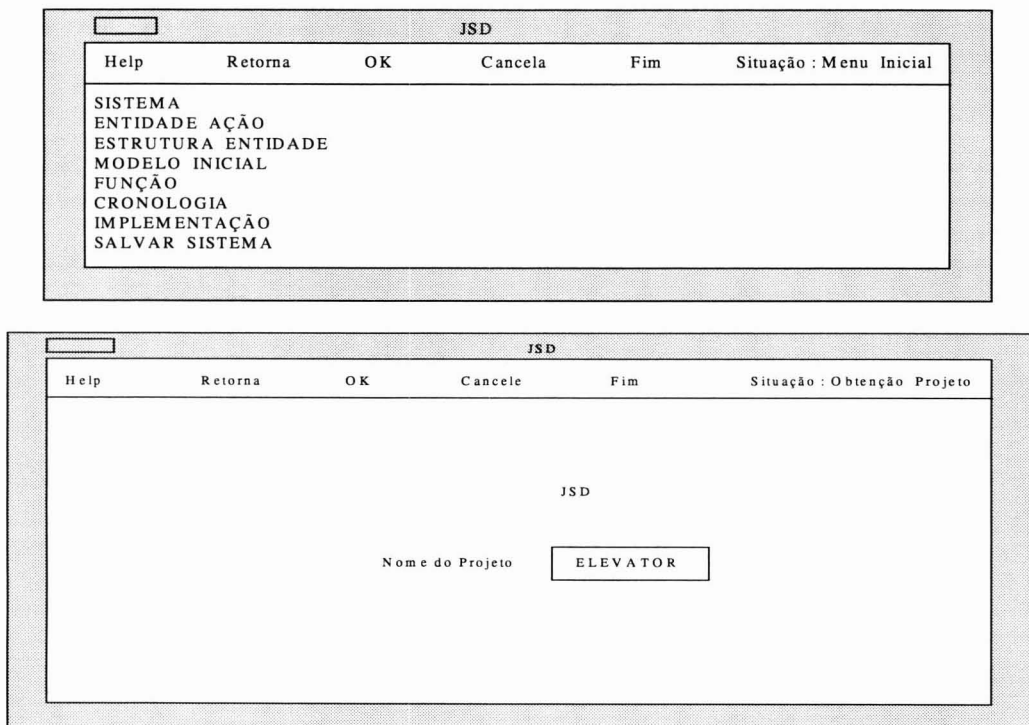


Figura 15 - Tela Inicial - JSD / PUC

A Figura 16 mostra uma tela que permite a entrada das entidades do sistema. No passo Entidade e Ação o ambiente permite que o desenvolvedor crie, remova ou modifique as entidades, ações e atributos, bem como suas descrições. O sistema verifica automaticamente 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 da Entidade o sistema permite a construção do diagrama de estrutura de cada entidade com suas ações, definidas obrigatoriamente na fase Modelo. A Figura 17 mostra uma tela exibida no passo Estrutura da 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 automaticamente. Na remoção é possível remover um nó sem filho ou toda uma subárvore do diagrama de estrutura da entidade. A modificação dos nomes e tipos dos nós intermediários, seleção ou repetição é possível; 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 incluída no passo Entidade e 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 rede, 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 de estrutura de entidade, auxiliado por um texto inicial gerado automaticamente. Esta geração automática, a partir do diagrama de estrutura de 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 necessária 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 a editoração gráfica foi construída utilizando-se da técnica de manipulação direta, tornando o sistema mais amigável para os usuários

5 Integração do Modelo Potts ao Ambiente JSD/PUC

Nesta seção mostraremos de que forma o método de Potts integra-se ao ambiente para prover facilidades ao processo de desenho da arquitetura de software.

The image shows two overlapping screenshots of the JSD/PUC software interface. The top screenshot displays a menu titled 'JSD' with options: Help, Retorna, OK, Cancela, Fim, and Situação: Entidade Ação. The menu items are: INCLUIR ENTIDADE, EXCLUIR ENTIDADE, MODIFICAR ENTIDADE, INCLUIR AÇÃO, EXCLUIR AÇÃO, MODIFICAR AÇÃO, INCLUIR ATRIBUTO, EXCLUIR ATRIBUTO, and MODIFICAR ATRIBUTO. The bottom screenshot shows a form titled 'JSD' with options: Help, Retorna, OK, Cancele, Fim, and Situação: Inclusão Entidade. It has two input fields: 'Nome Entidade' and 'Descrição Entidade'. To the right is a checklist with four questions: 'Entidade executa ou sofre ação em um tempo significativo, ordenadamente?', 'Entidade existe no mundo real (fora do sistema)?', 'Entidade pode ser considerada individual?', and 'Sistema usa ou produz informação sobre a entidade?'.

Figura 16 - Passo Entidade Ação - JSD / PUC

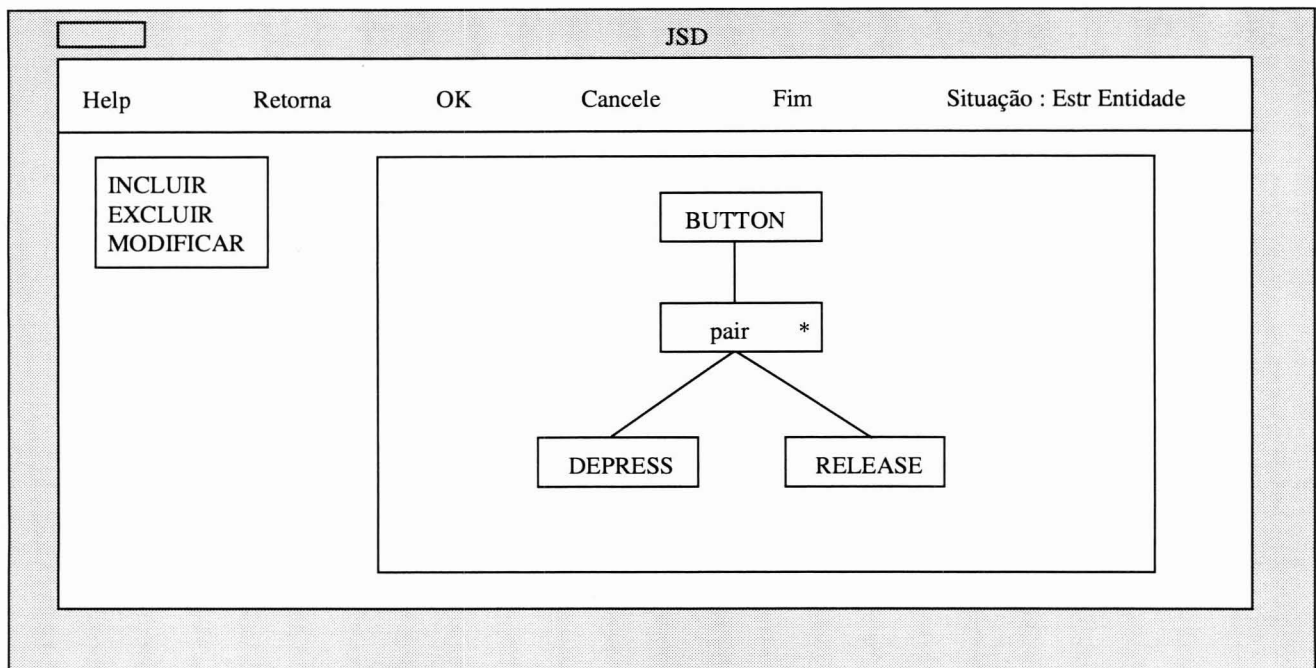


Figura 17 - Passo Estrutura Entidade - JSD / PUC

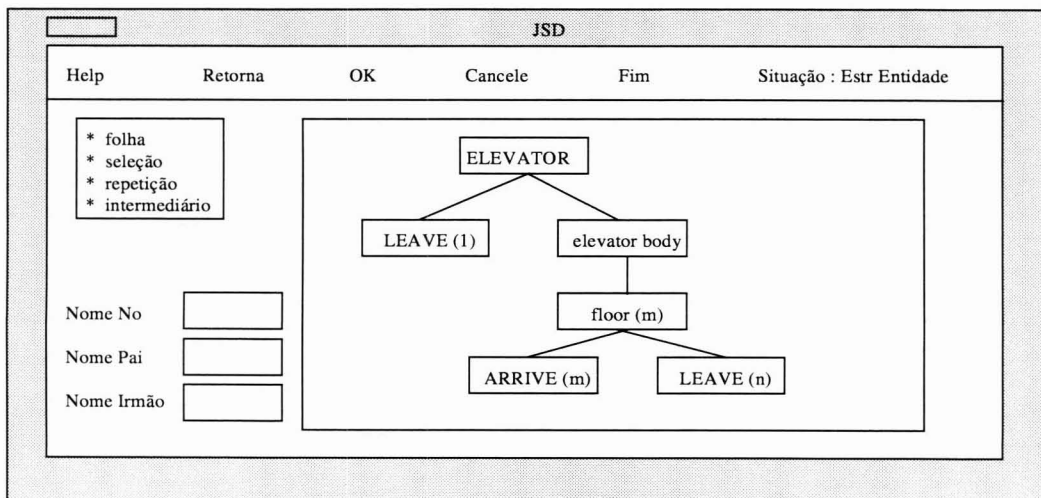
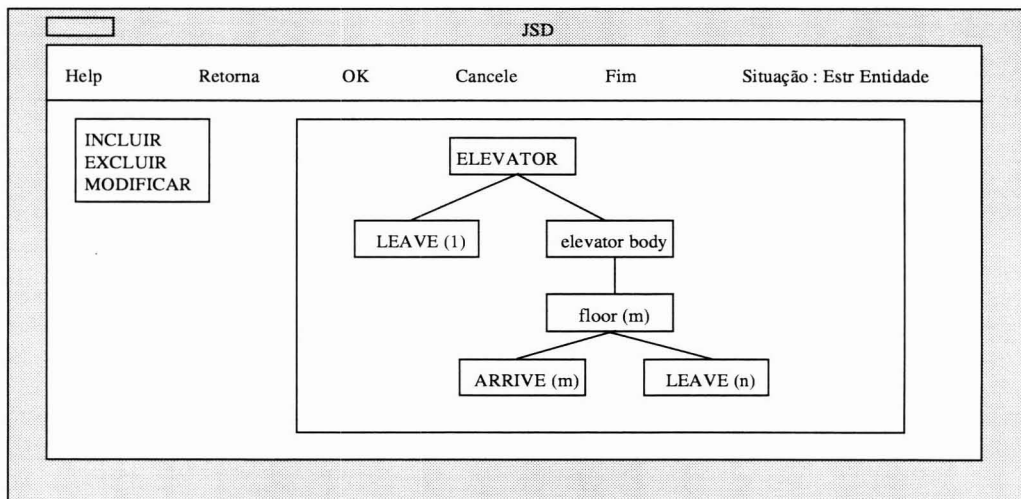


Figura 18 - Construção Diagrama Estrutura Entidade - JSD / PUC

5.1 Visão Geral do paradigma Potts

Aqui apresenta-se um modelo de dados para o método de Potts (Potts & Bruns, 1988 e Potts, 1989) 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 distinguem-se pelo menos 3: descrição do projeto, estratégia e progressão, e heurísticas e raciocínios sobre as decisões de projeto. Por outro lado, segundo Potts, em geral as tentativas de codificar as informações do processo, têm 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 heurísticas 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 de tal forma que sua análise possa ser reutilizada para manutenções ou desenvolvimento de novos projetos (por analogia à 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 de entidade, os quais constituem 5 classes básicas:

- Artefato
- Questão
- Passo
- Posição
- Argumento

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

- Revisa
- Sugere
- Responde a
- Cita
- Contribui para
- Suporta
- Rejeita
- Modifica

Objetos específicos do método são todas 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 à 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 foi selecionada;

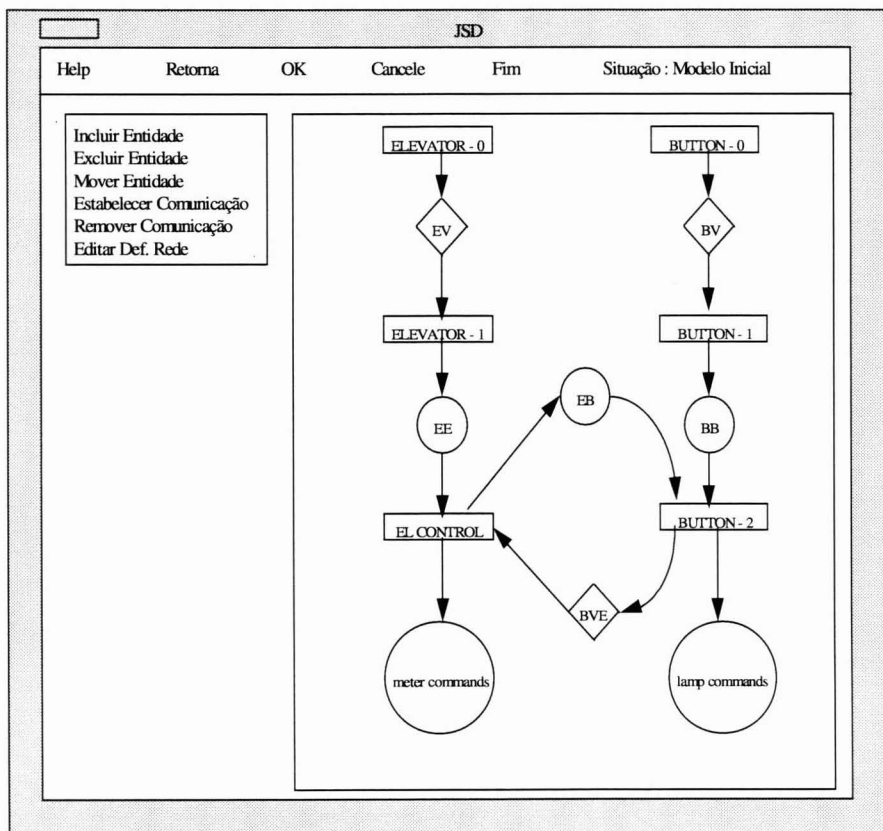


Figura 19 - Fase Rede - JSD / PUC

- Argumentos: dão suporte às 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 é instanciado o modelo para o caso JSD.

5.2 Adequação do paradigma Potts ao paradigma procedimental de Osterweil

Para fazer registro das decisões e raciocínios 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 possível 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 heurísticas 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 raciocínios adotados para dar suporte ou rejeitar as posições.

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

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

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

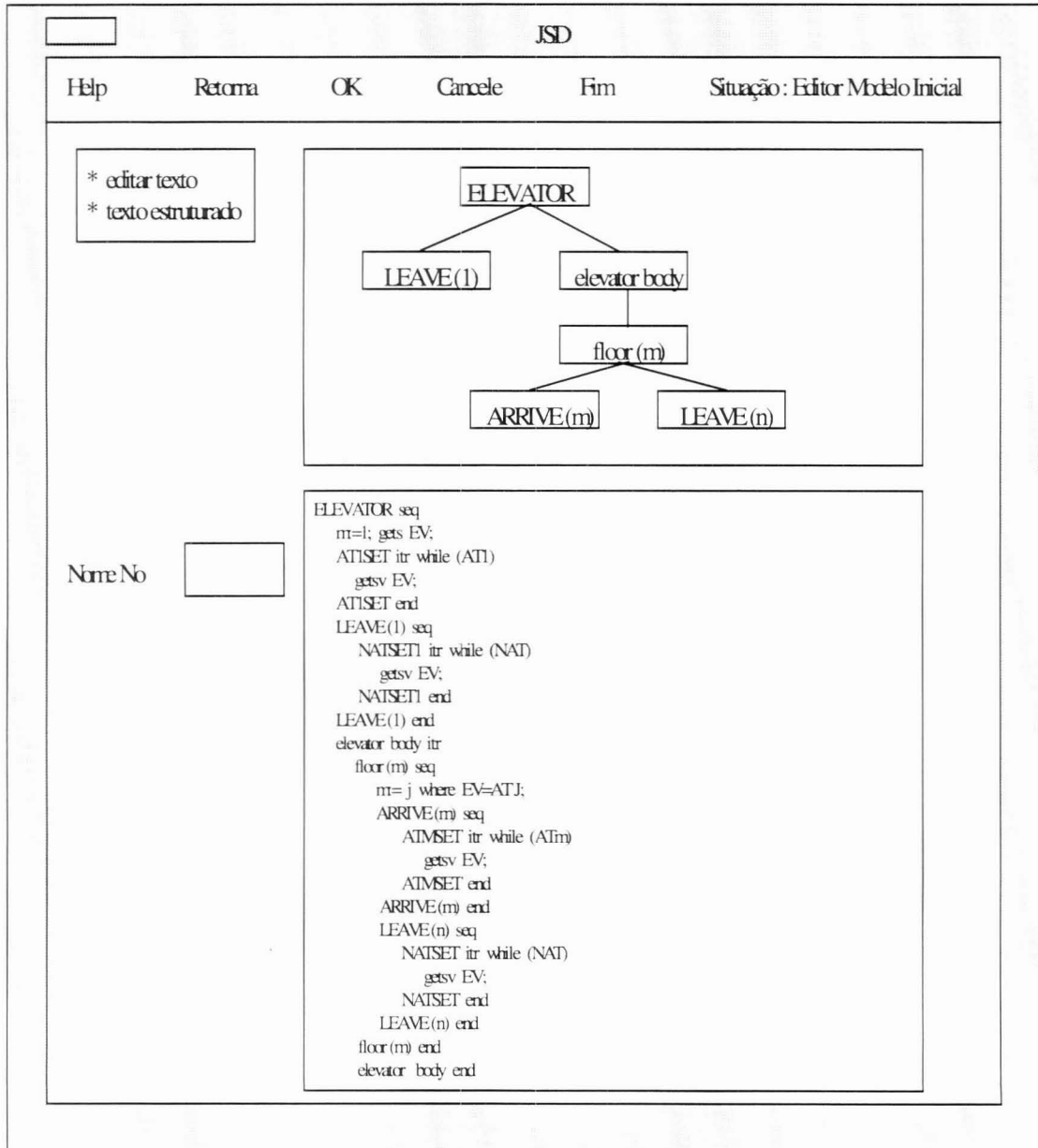


Figura 20 - Texto Estruturado - JSD / PUC

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.

Conforme se pode ver, a entidade LEITOR foi incluída na Lista de Entidades. A resolução desta Questão sugere uma nova Questão sobre a validade da candidata com relação às 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 às 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 JUÍZES, já selecionada", a entidade SESSÃO é excluída. 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.

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, 1987), e do conceito de interfaces como especificações. O programa de

processo utilizado foi expresso através de um diagrama de estrutura de 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.

É importante salientar que o método preconizado por Osterweil, isto é, descrição procedural de processos de software, ajudou a organizar as etapas de desenvolvimento do JSD/PUC. Portanto acreditamos que a grande vantagem do uso do programa do processo, especializado para o JSD/PUC, foi exatamente possibilitar uma sistematização e maior controle no desenvolvimento. Cabe salientar que utilizando a noção de processo, criamos um processo particular para o problema em questão.

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 *et alii*, 1991), 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.

O histórico do projeto construído no ambiente JSD/PUC pode ser usado como fonte de informações para a manutenção do próprio sistema e reutilização das especificações do projeto em sistemas similares. O histórico do projeto, disponível no modelo Potts, pode ser consultado sobre as decisões de projeto e suas justificativas. Na manutenção, por exemplo, caso se queira modificar alguma função do sistema tem-se acesso aos argumentos que justificaram sua implementação.

Na reutilização em sistemas similares, o acesso a justificativas facilita a tomada de decisões quanto ao senso ou não das especificações do projeto. É também fato que o acesso ao histórico das decisões pode servir de base, portanto senso, para decisões similares em outros projetos.

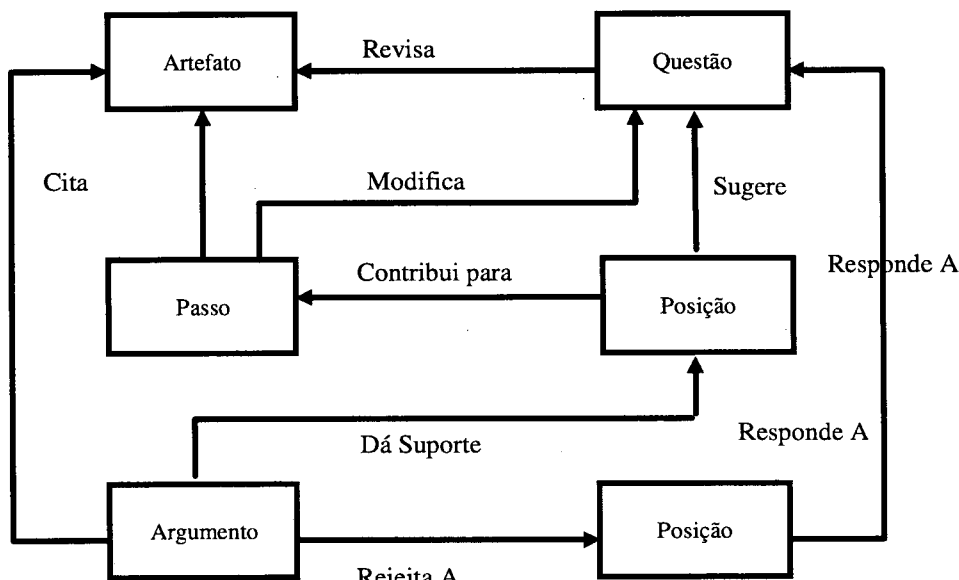


Figura 21 - Esquema Geral De Potts

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. Um dos trabalhos consequentes ao aqui descrito é a extensão da

metodologia JSD e do modelo de Potts para dar suporte a um processo de desenvolvimento cooperativo (Duarte *et alii*, 1992).

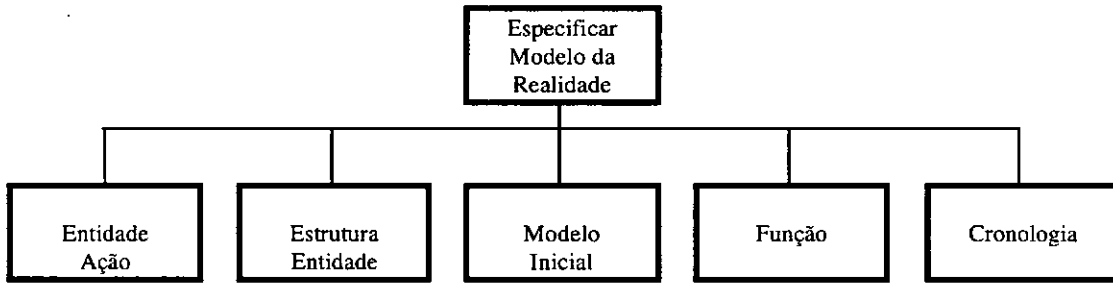


Figura 22 - Hierarquia da Sub-Classe Passo do Procedimento JSD

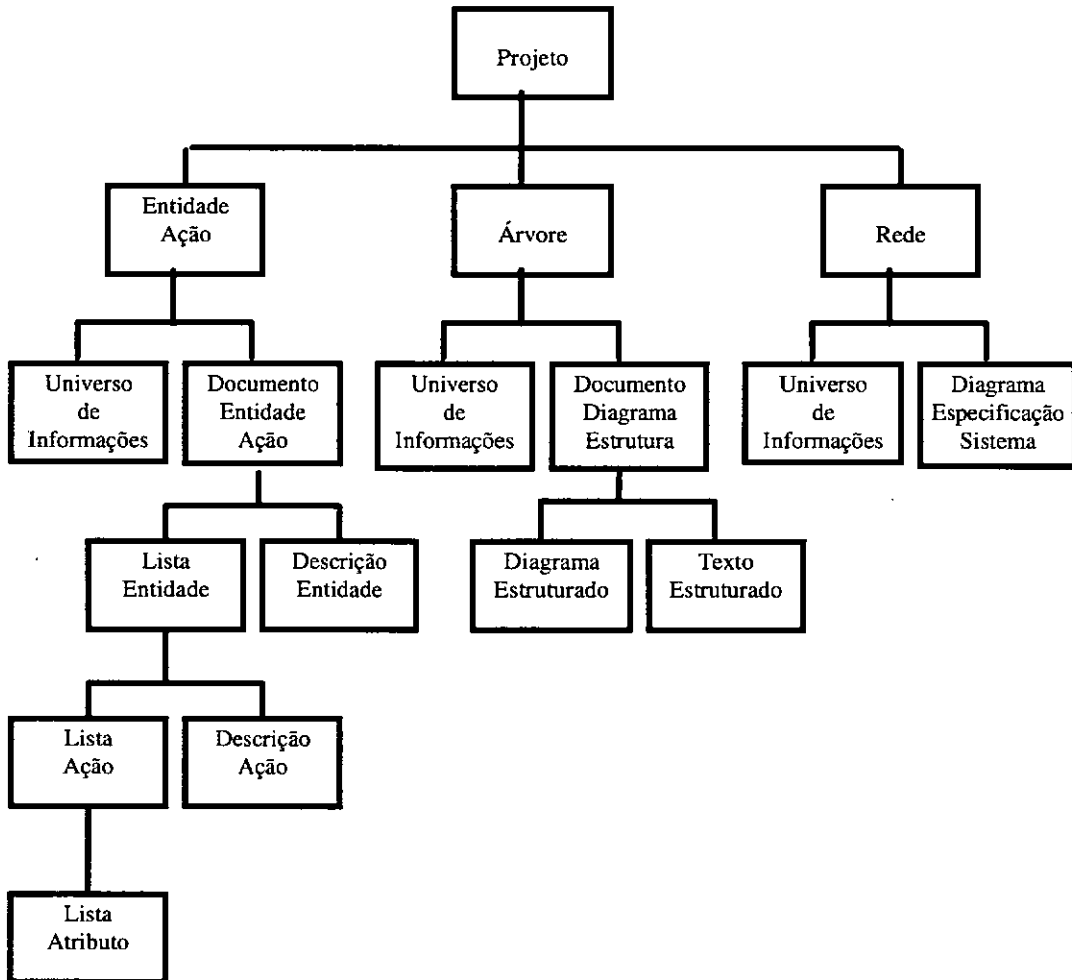


Figura 23 - Hierarquia Da Sub-Classe Artefato Do Processo JSD

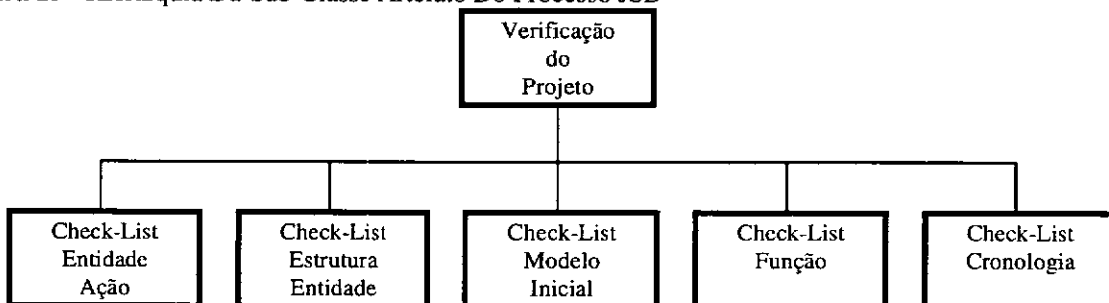


Figura 24 - Hierarquia da Sub-Classe Questão do Processo JSD

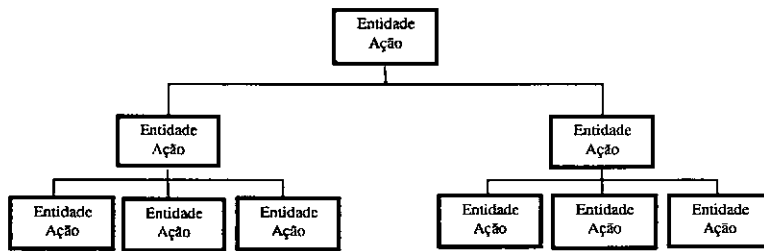


Figura 25 - Hierarquia da Sub-Classe para Entidade Ação

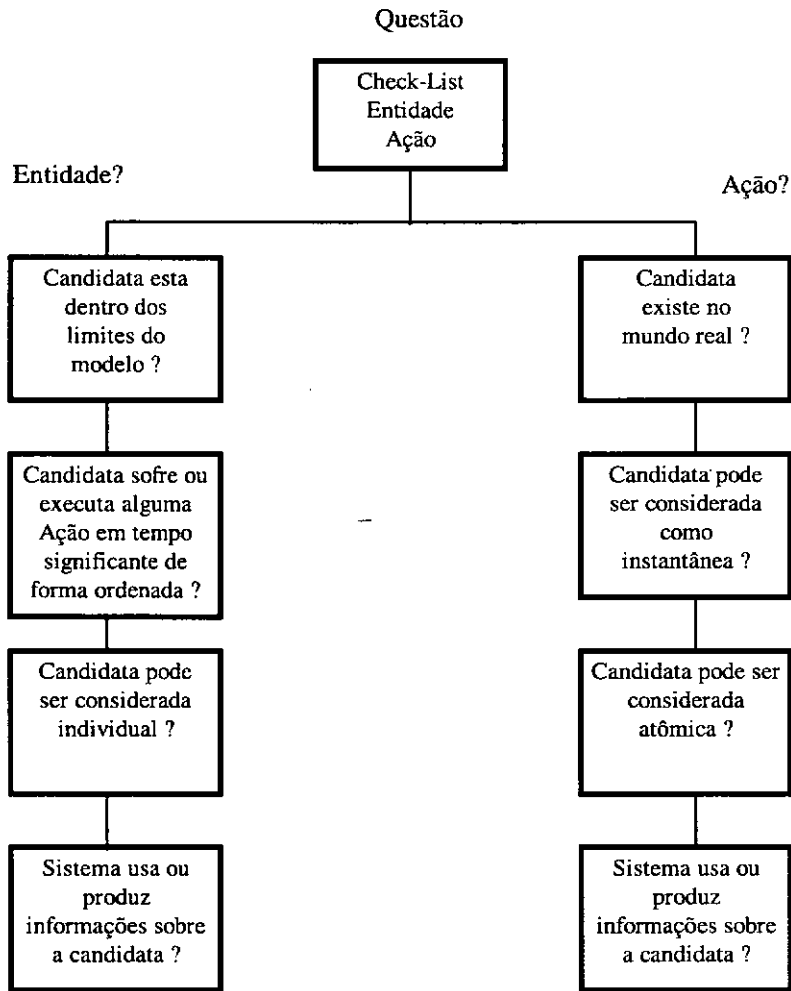


Figura 26 - Hierarquia da Sub-Classe Questão para Entidade Ação

7 Referências Bibliográficas

- Bischofberger, W.; Keller, R., 1989; "Enhancing The Software Life Cycle by Prototyping", *Structured Programming* 1, 47-59.
- Cabral, R. H. B.; Campos, I.M.; Cowan, D.D.; Lucena, C.J.P., Abr. 1990, "Interfaces as Specifications in The MIDAS USER Interface Development System", *Software Engineering Notes*, ACM SIGSOFT, Vol. 15, no. 2.
- Chikofsky, Elliot., 1993 "Computer-Aided Software Engineering", IEEE Computer Society Press.
- Computer Science Technology Board, , Mar. 1990 "Scaling Up: A Research Agenda for Software Engineering", *Communications of the ACM*.
- Duarte, Renato C.; Fuks, Hugo; Lucena, Carlos J.P. Nov. 1992, *Software Design Cooperativo: Um Estudo de caso*, VI Simpósio Brasileiro de Engenharia de Software, SBC (Sociedade Brasileira de Computação), Gramado, RS, pp. 97-114.
- Habermann, A. N. et al., 1986 "Gandalf: Software Development Environment", *IEEE Trans. on Software Engineering*, Vol. 12 no. 12.
- Horowitz, E. et al., 1986, "SODOS: A Software Documentation Support Environment - its Definition", *IEEE Trans. on Software Engineering*, vol. 12, no. 8.
- Jackson, M.A., 1983; "*System Development*", Prentice Hall.
- Lamsweerde, 1988, "Generic Life Cycle Support in the ALMA Environment", *IEEE Trans. on Software Engineering*, Vol. 14, no. 6.

Lee,V.; Lai, KoY; , 1991, "What's in Design Rationale", MIT CCS Tech. Report 118.

Maher,M.L. , 1990; "Process Models for Design Synthesis", AI Magazine, Winter.

Masiero, Paulo C. , 1992 "Análise Estruturada de Sistemas pelo Método de Jackson". Ed. Edgard Blucher, Ltda.

OSF/Motif, 1990, "Motif Programmer's Reference", Open Software Foundation, PrenticeHall.

Osterweil, L. , 1987; "Software processes are software too", Proc. 9th Intl. Conf. Software Engineering, IEEE Computer Society Press.

Perry, Dewayne E., Kaiser, Gail E. , 1988, Models of Software Development Environments, ", Proc. 10th Intl. Conf.

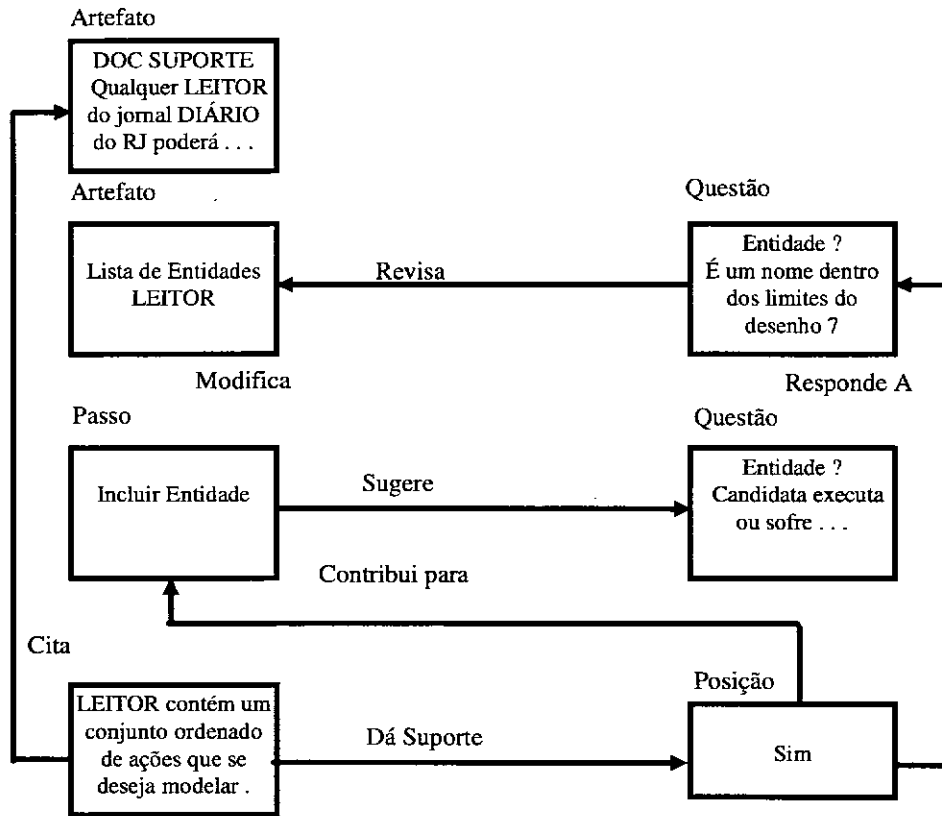


Figura 27 - Episódio Incluir Entidade

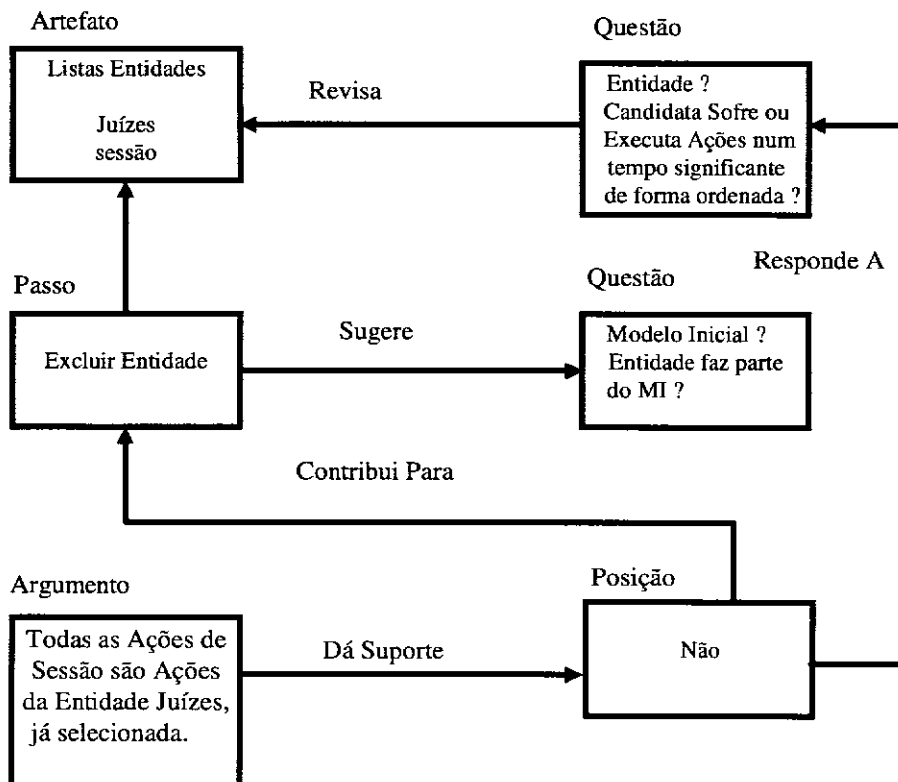


Figura 28 - Episódio Excluir Entidade

Potts, Colin; Bruns, Glenn; , 1988, "Recording the Reasons for Design Decisions", *Proc. 10th Intl. Conf. Software Engineering*, IEEE Computer Society Press.

Potts, Colin; , May, 1989 "A Generic Model for Representing Design Methods", *11th Intl. Conf. on Software Engineering*, IEEE Computer Society Press.

Prado, Antonio Francisco; Lucena, Carlos J. P.; Leite, Julio Cesar S. P.; Out 1991 "Registro de Decisões e Justificativas de Desenho em Softwares Projetados com a Metodologia JSD", *1º Simpósio Brasileiro de Engenharia de Software*, SBC (Sociedade Brasileira de Computação), Ouro Preto, MG, pp. 15-29.

Williams, L. , 1988, "Software process modeling: a behavioral approach", *Proc. 10th Intl. Conf. Software Engineering*. IEEE Computer Society Press.