Série: Monografias em Ciência da Computação
Nº 6/88

UM AMBIENTE PARA A TRANSFORMAÇÃO SEMI-AUTOMÁTICA DE DIAGRAMAS DE FLUXOS DE DADOS EM DIAGRAMAS DE OBJETO

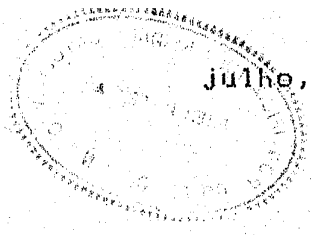
Jeferson Ferreira Soares
Roberto Ierusalimschy
Theonilla Estellita C. Pessoa

Departamento de Informática

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Série: Monografias em Ciência da Computação
N: 6/88

Editor: Paulo Augusto Silva Veloso julho, 1988



UM AMBIENTE PARA A TRANSFORMAÇÃO SEMI-AUTOMÁTICA DE DIAGRAMAS DE FLUXOS DE DADOS EM DIAGRAMAS DE OBJETO *

Jeferson Ferreira Soares
Roberto Ierusalimschy
Theonilla Estellita C. Pessoa

* Apresentado por Carlos José Pereira de Lucena.

Trabalho parcialmente financiado pelo CNPQ, pela CAPES e pela SID-Informática através do Projeto ESTRA.

Cópias de publicações :

Rosane T. L. Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC/RJ - Depto. de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

UM AMBIENTE PARA A TRANSFORMAÇÃO SEMI-AUTOMÁTICA DE DIAGRAMAS DE FLUXOS DE DADOS EM DIAGRAMAS DE OBJETO

Jeferson Ferreira Soares
Roberto Ierusalimschy
Theonilla Estellita C. Pessoa

RESUMO

Este trabalho descreve um ambiente, baseado em heurísticas, que tem por objetivo a transformação de Diagramas de Fluxos de Dados em Diagramas de Objetos. Além da apresentação do sistema e sua arquitetura, são abordados alguns aspectos da técnica utilizada neste desenvolvimento (Programação Orientada para Objetos) e algumas considerações sobre a linguagem e o ambiente de implementação (Smalltalk) usados.

Palavras-chave: Modelagem de Software, Ferramenta de Desenvolvimento de Software, Análise Estruturada, Projeto Orientado para Objetos.

ABSTRACT

This work describes a heuristics based environment whose function is to transform Data Flow Diagrams into Object Diagrams. Besides the description of the system and its architecture, some aspects of the technique used in this development (Object Oriented Programming) are discussed together with some considerations about the language and the implementation environment (Smalltalk).

Keywords: Software Designing, Software Development Tool, Structured Analysis, Object-oriented Design.

1. INTRODUÇÃO

Este artigo aborda alguns aspectos de uma experiência de desenvolvimento de uma ferramenta de apoio à construção de software.

No capítulo 2 é apresentada a motivação básica inicial do trabalho.

As principais idéias a serem experimentadas com esta ferramenta, que apoia a atividade de modelagem de software, são discutidas no capítulo 3. Neste capítulo são também comentadas as principais funções disponíveis para o usuário assim como sua forma de utilização.

O capítulo 4 apresenta uma descrição da arquitetura do software desenvolvido, discute aspectos da técnica utilizada neste desenvolvimento (Programação Orientada para Objetos) e tece algumas considerações sobre a linguagem e o ambiente (SMALLTALK) usados.

2. MOTIVAÇÃO

A compreensão e descrição do processo de desenvolvimento de software têm sido alvo de diversas pesquisas em Engenharia de Software. Este interesse no entendimento detalhado e na sistematização das atividades envolvidas em processos de produção é natural nas disciplinas de Engenharia e tem como motivação o desejo de se melhorar, continuamente, os níveis de produtividade através, principalmente :

a) da homogeneização e otimização dos processos de produção, especialmente nos trabalhos que envolvem equipes;

b) da possibilidade de se controlar qualidade nos procedimentos e resultados produzidos;

c) da possibilidade de se prever e controlar a alocação de recursos da produção.

Este problema é particularmente complexo no caso da produção de software, principalmente devido à sua natureza abstrata. As atividades de análise de requisitos e especificação/execução de testes para sistemas são bons exemplos para esta dificuldade.

As tentativas de abordagem sistemática a estes problemas têm passado, obrigatoriamente, pelas propostas de modelos de ciclo de vida, metodologias e técnicas aplicadas a atividades específicas de desenvolvimento de software. As várias correntes de trabalho nesta área partem, no entanto, de diferentes enfoques de modelagem da realidade e de distintos conceitos básicos (paradigmas) de desenvolvimento. Abordam escopos diversos dentro do conjunto de atividades previsto por cada modelo e adotam,

ainda, diferentes graus de formalização matemática nas linguagens de expressão escolhidas.

Estas características fazem com que diferentes correntes desta tecnologia explorem melhor (ou de maneira mais eficiente) aspectos diferentes do desenvolvimento. A consequência disto é uma certa tendência de especialização, isto é, para uma determinada situação de uso em que se reconheça a preponderância de um aspecto específico de desenvolvimento, procura-se recorrer à corrente de tecnologia que melhor se adapte àquele tratamento. Deste modo pode-se, por exemplo, dar prioridade, durante a construção de um software, à organização de estruturas de dados complexas, ou à estruturação de funções, para citar situações já tradicionais, ou à representação de algum tipo de conhecimento, para falar de aplicações mais avançadas.

Ocorre que nem sempre há preponderância clara de um desses aspectos isoladamente. Mesmo quando há preponderância, isto não significa a inexistência de outros aspectos a serem considerados e, portanto, não anula a importância de uma abordagem mais abrangente do desenvolvimento. Diante deste quadro, a possibilidade de se combinar de forma disciplinada diferentes enfoques de trabalho revela-se como um passo importante, no sentido de uma melhor compeensão do processo de desenvolvimento de software.

3. O USO DA FERRAMENTA

3.1 O QUE FAZ

O sistema apresentado neste artigo é um protótipo de uma ferramenta de desenvolvimento, com a qual deseja-se experimentar um tipo de uso combinado de duas linhas de construção de software que adotam paradigmas diferentes : Análise Estruturada [2] e Projeto Orientado por Objetos [6]. Busca-se dar apoio automático à atividade de modelagem de sistemas explorando pontos de complementaridade entre as duas metodologias mencionadas.

O procedimento geral de utilização é simples :

a) o usuário monta um Diagrama de Fluxos de Dados (DFD);

b) a ferramenta analisa o DFD do sistema montado e constrói a partir dele um modelo gráfico do mesmo sistema expresso em termos dos conceitos de Projeto Orientado por Objetos;

c) o usuário, se desejar, edita o modelo gráfico de objetos gerado pela ferramenta.

Explorando um pouco mais a descrição do processo geral de utilização feito acima vemos que, do ponto de vista do usuário, a ferramenta apresentada tem três componentes principais :

a) Um editor gráfico interativo específico para a montagem do DFD. Um componente importante, embutido neste editor, é uma base de dados que contém descrições (formalizadas) dos critérios de correção sintática da construção de DFD's. Com isto, o trabalho do usuário vai sendo criticado ainda no momento de edição. Construções sintáticas incorretas são rejeitadas e

indicadas por mensagens de erro.

b) Um conjunto de regras de transformação que gera um Modelo de Objetos a partir de um DFD já montado.

Este componente do sistema utiliza-se de duas fontes de conhecimento. A primeira são as heurísticas de transformação (representadas por regras de produção) incorporadas à ferramenta. Estas heurísticas, que estão resumidas no quadro da figura 1, foram testadas manualmente antes de sua formalização e implementação. A segunda é o próprio usuário, que é chamado a interferir, sempre que necessário, no processo de geração do Modelo de Objetos. A intervenção do usuário se dá sempre através de respostas a perguntas objetivas formuladas pelo sistema.

c) Um editor gráfico específico para o Modelo de Objetos. Similarmente ao editor de DFD's tem-se, para este editor, uma base de dados contendo descrições dos critérios de correção sintática e do relacionamento estrutural entre os componentes do modelo gráfico em questão.

A edição feita pelo usuário é, portanto, monitorada pelo editor, evitando e informando a ocorrência de construções inválidas e de inconsistências internas ao modelo.

.toda entidade externa do DFD se transforma num objeto do MO, com dois métodos default : ENTRADA e SAÍDA.

.todo depósito (arquivo) do DFD se transforma num objeto do MO, com dois métodos default: CONSULTAR e ATUALIZAR.

.todo processo primitivo (sem explosão) tem duas opções: caso este processo atualize algum depósito, ele poderá transformar-se num objeto individual ou, dependendo da escolha do usuário, dar origem a um método do objeto que representa o depósito atualizado; caso contrário (o processo não atualiza depósitos), o processo se transforma, necessariamente, num objeto individual.

.para processos que tem explosão, aplicam-se as regras acima recursivamente aos elementos do DFD que o explode.

.todo fluxo do DFD dá origem a uma mensagem entre objetos. A origem da mensagem será o objeto "O" no qual a origem do fluxo no DFD se transformou. O destino da mensagem será, analogamente, o objeto "D" no qual o destino do fluxo no DFD se transformou. Caso esta origem/destino (do DFD) tenha se transformado em um método de um objeto O1, então a origem/destino da mensagem será o objeto O1. Pode acontecer que os objetos "O" e "D" sejam os mesmos, por exemplo, se tivermos um fluxo saindo de um processo e indo para um depósito, onde o deposito se transforma num objeto "D" e o processo se transforma num metodo deste mesmo objeto "D". Em casos como este o fluxo do DFD não terá representação no MO por ser interno ao objeto.

figura 1

O uso do sistema como um todo sugere uma sequência de passos, uma sistemática de trabalho, que parte da especificação funcional de um sistema (correta por construção, no que diz respeito à montagem do modelo) e a ela aplica um conjunto de regras de transformação, gerando o MO do mesmo sistema (vide fig. 2).

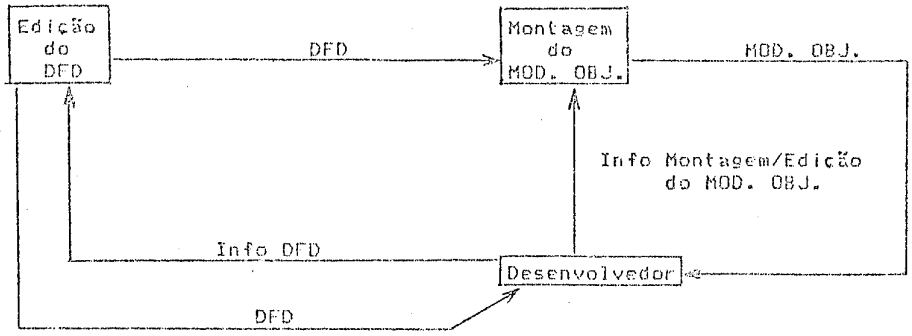


figura 2

Esta transformação realça informações sobre a aplicação sendo construída que permaneceriam pouco exploradas, caso o único modelo disponível fosse o do DFD. A complementaridade das duas especificações é importante em dois aspectos:

a) na compreensão mais precisa e abrangente do problema de processamento de informação, ao qual a aplicação sendo desenvolvida pretende atender;

b) na tomada de decisões de projeto e implementação do sistema em construção.

O usuário pode, se quiser, utilizar qualquer um dos editores isoladamente, como ferramentas independentes do resto do sistema. Pode-se, por exemplo, editar um Modelo de Objeto independentemente (fig. 3). A versão do MO, gerada a partir do DFD, é apenas uma especificação inicial, podendo ser alterada à vontade. O problema óbvio, neste caso, é que o resultado final pode ter sua funcionalidade totalmente diferente daquela da

especificação inicial. Não se está apenas complementando a especificação inicial com informações que só são vistas (explicitadas) no Modelo de Objetos, a aplicação modelada está sendo efetivamente alterada em sua definição. A vantagem é a possibilidade de uso do editor de MO nos casos em que o usuário não queira seguir a estratégia padrão de uso da ferramenta.

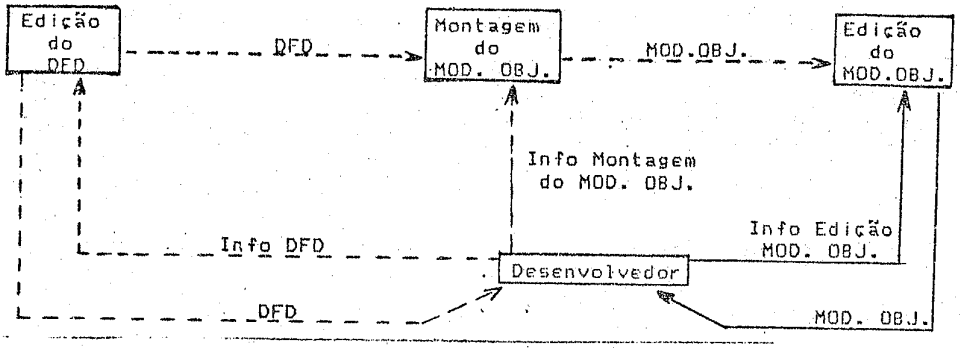


figura 3

3.2 UM EXEMPLO DE UTILIZAÇÃO

Serão apresentadas a seguir as principais funções disponíveis no software descrito neste trabalho. Esta apresentação será feita através do acompanhamento de um caso exemplo, cujo roteiro visa realçar a funcionalidade e a forma de apresentação/ativação das operações centrais do sistema, assim como a interação do usuário com as mesmas.

O protótipo descrito neste trabalho é acionado por um comando SMALLTALK. Com este comando o editor de DFD's é colocado

ativo, exibindo a tela para a edição do nível zero do Diagrama de Fluxo de Dados, relativo à aplicação a ser desenvolvida pelo usuário. O nome da aplicação do caso exemplo, vista na figura 4, é TESTE.

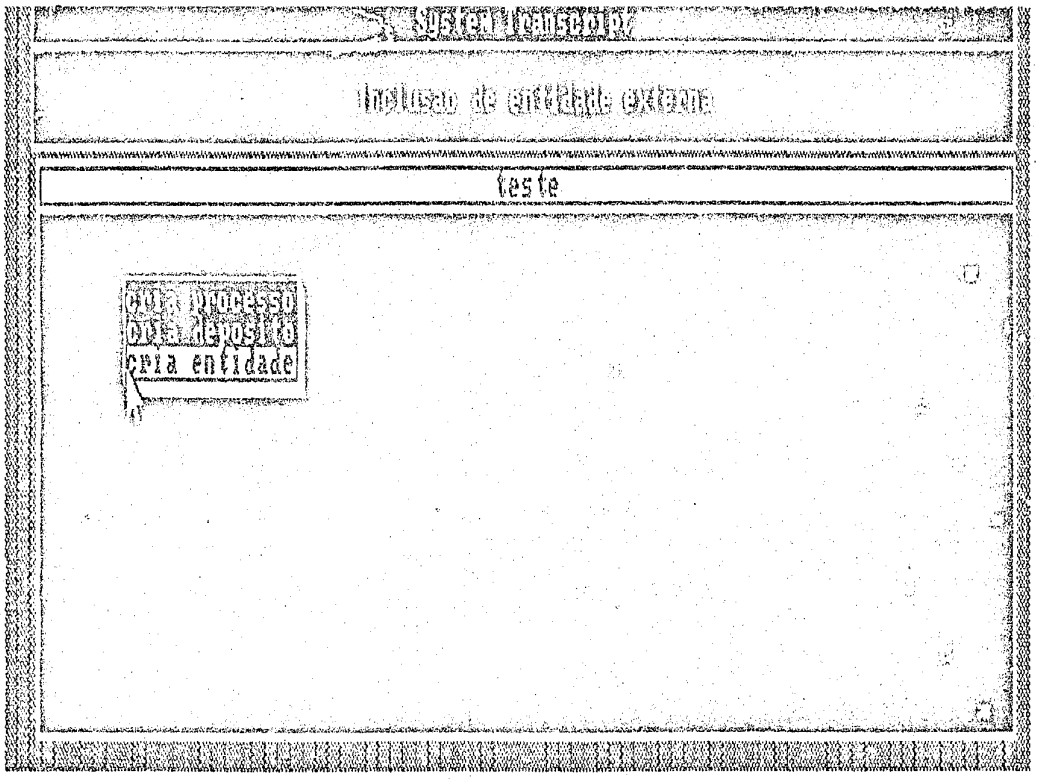


figura 4

As três operações disponíveis neste momento são as de criação de processo, depósito e entidade externa. O procedimento de ativação é : o usuário posiciona o cursor (seta branca) na posição em que deseja inserir o novo elemento e aperta a tecla "+". Desta maneira ativa-se o menu do tipo pop-up, mostrado na figura 4, com o qual pode-se escolher o tipo do elemento a ser criado.

O exemplo mostra a criação de uma entidade externa. As figuras 5 e 6 mostram, respectivamente, a escolha do nome para o elemento criado e o resultado da operação. É importante notar que ao final da criação o cursor permanece sobre o elemento criado. Em qualquer momento durante a edição de um DFD o posicionamento do cursor sobre um elemento indica ser este o elemento corrente de edição, isto é, o objeto sobre o qual atuará a próxima operação de edição ativada.

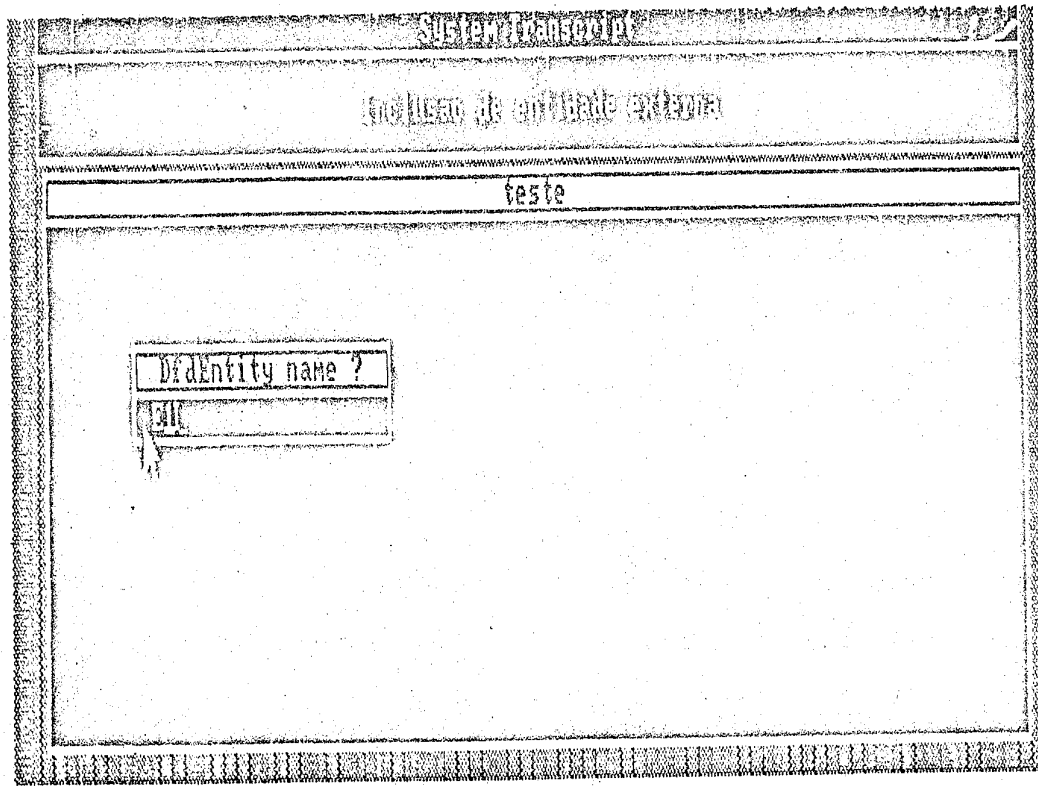


figura 5

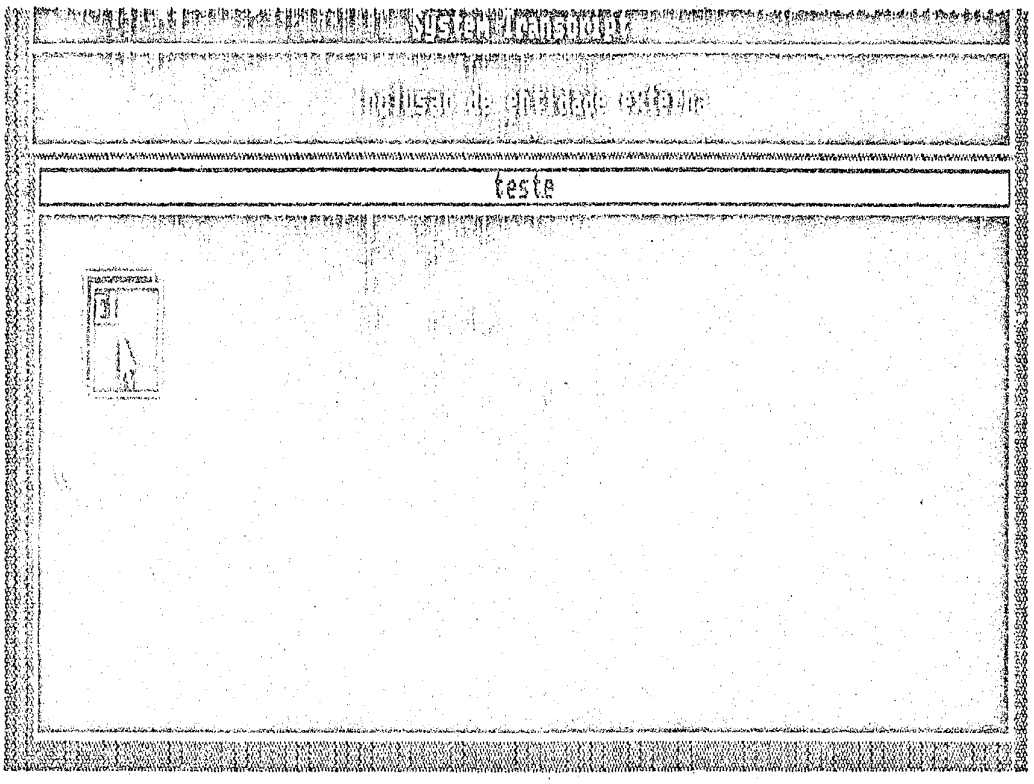


Figura 6

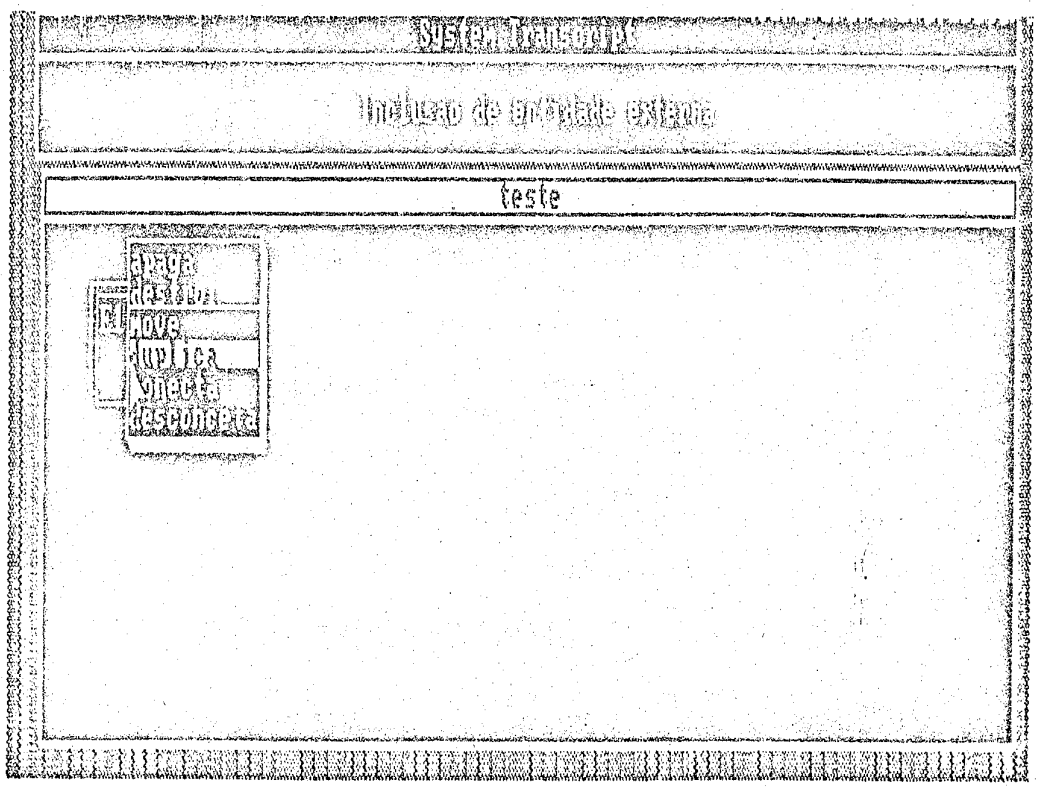


Figura 7

Na figura 7 está exibido o menu de opções com as operações de edição específicas para entidades externas. Estas operações são :

.duplica : cria na posição desejada na tela (a ser indicada pelo cursor) uma cópia do elemento corrente.

.move : permite que se altere a posição de um elemento na tela. O usuário ativa a função, posiciona o cursor na nova posição desejada e finaliza a operação.

.conecta : cria um fluxo de dados conectando o elemento corrente a algum outro elemento exibido na tela. Uma vez ativada a operação, o usuário pode marcar sobre a tela, na sequência desejada, os pontos que servirão de vértices do fluxo sendo especificado. Um vértice marcado sobre um elemento (processo, depósito ou entidade externa) indica ser este o elemento destino do fluxo. Conexões entre elementos incompatíveis são rejeitadas e dão origem a mensagens de erro, que informam ao usuário a razão da rejeição.

.desconecta : esta operação apaga todos os fluxos com origem no elemento corrente.

.apaga : apaga da tela o elemento corrente. Esta operação só é válida se o elemento corrente estiver totalmente desconectado dos outros

elementos do DFD, isto é, ele não pode ser origem nem destino de fluxos.

.destrói : apaga da tela o elemento corrente e todas as suas cópias, geradas com a operação "duplicar".

A figura 8 mostra o resultado da criação de um processo. Esta operação, assim como a criação de depósitos de dados, se dá de forma análoga à inserção de entidades externas, já vista anteriormente.

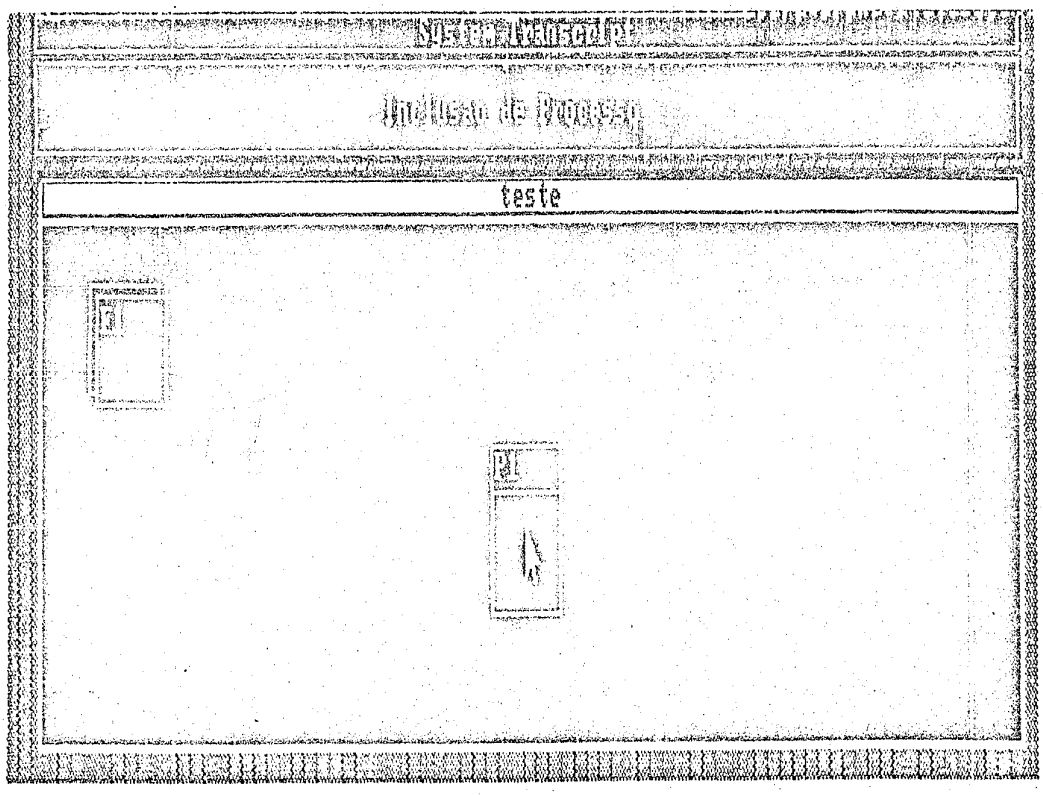


figura 8

As operações de edição específicas para processo são

exibidas na figura 9. Estão disponíveis, neste caso, todas as operações disponíveis para entidades externas e mais :

.explode : cria uma janela, com o nome do processo corrente, na qual será construído o DFD que corresponde a sua explosão. A edição deste DFD pode utilizar todas as funções disponíveis à edição do DFD de nível zero, com pequenas particularidades exemplificadas na figura 12.

.des...(explode) : apaga um DFD gerado em um procedimento de explosão de processo.

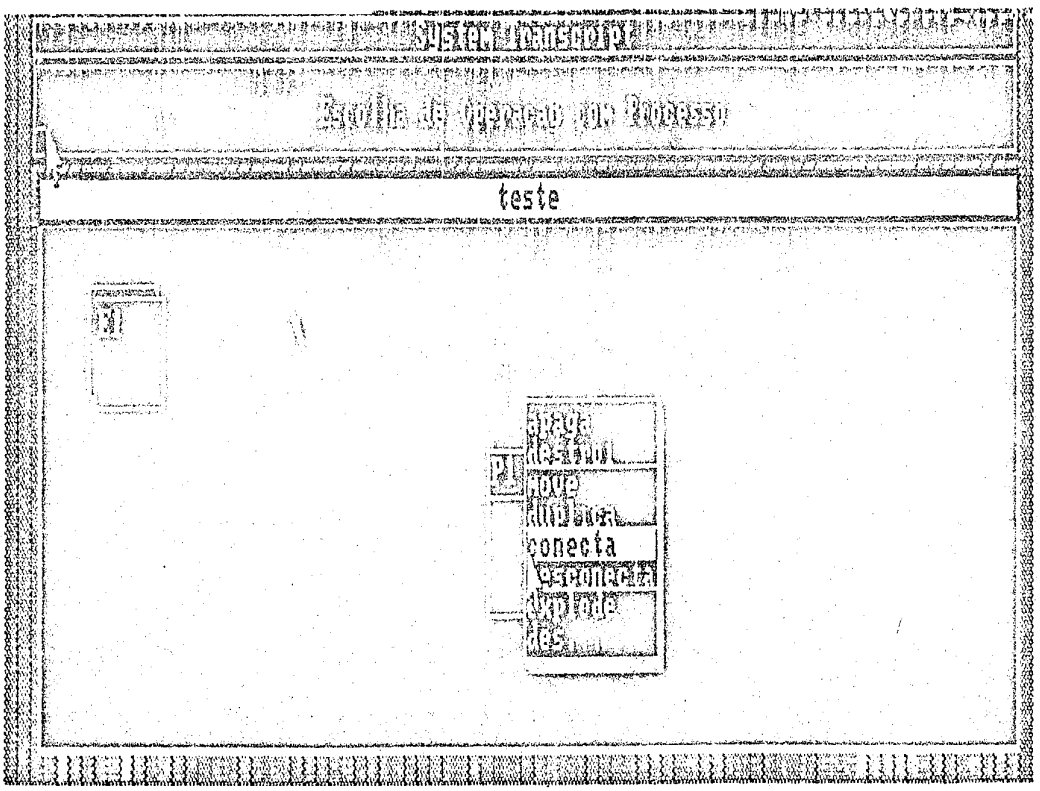


figura 9

A figura 10 mostra o resultado de uma ativação da função conectar disparada a partir do processo "P1" e com um vértice na trajetória do fluxo.

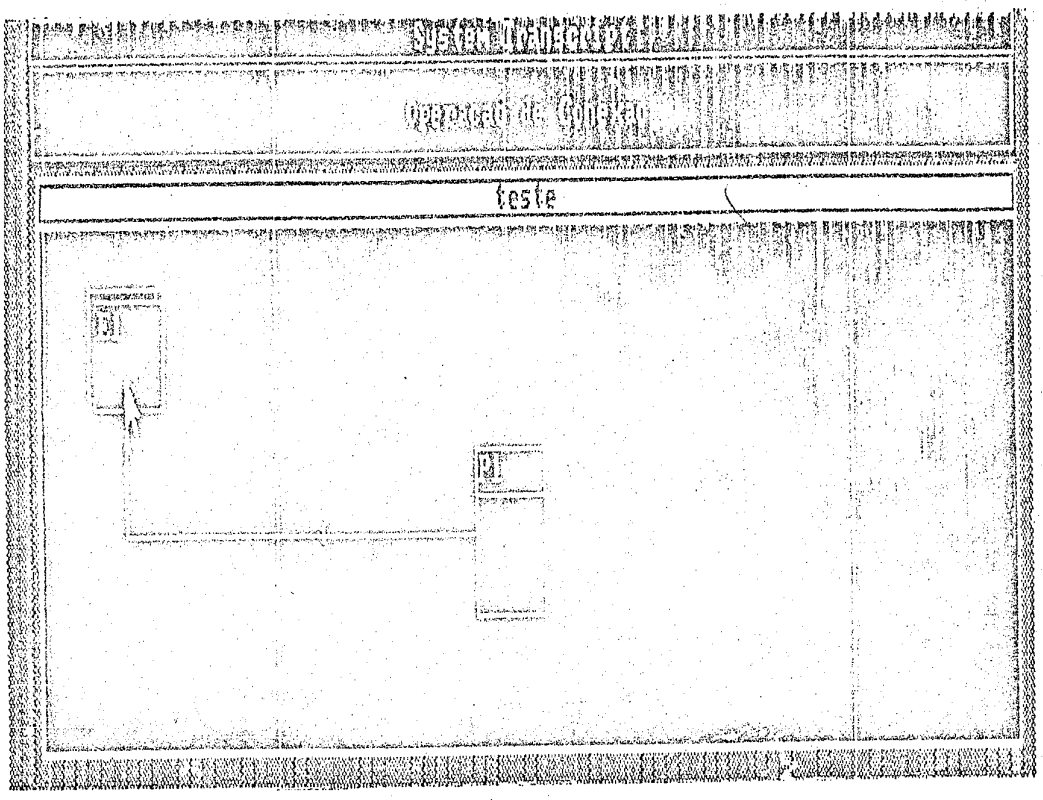


figura 10

O nível zero completo do DFD usado como exemplo é mostrado na figura 11.

O resultado da ativação da função explode a partir do processo "P1" está na tela da figura 12. Note-se, na lateral esquerda da tela, as entidades E1 e E2 representadas em fundo branco. Esta representação tem o objetivo de associar ao DFD de uma explosão os elementos com os quais o processo que deu origem à explosão se comunica em seu nível original. No exemplo, o processo "P1" comunica-se com E1 e E2 no nível zero do DFD.

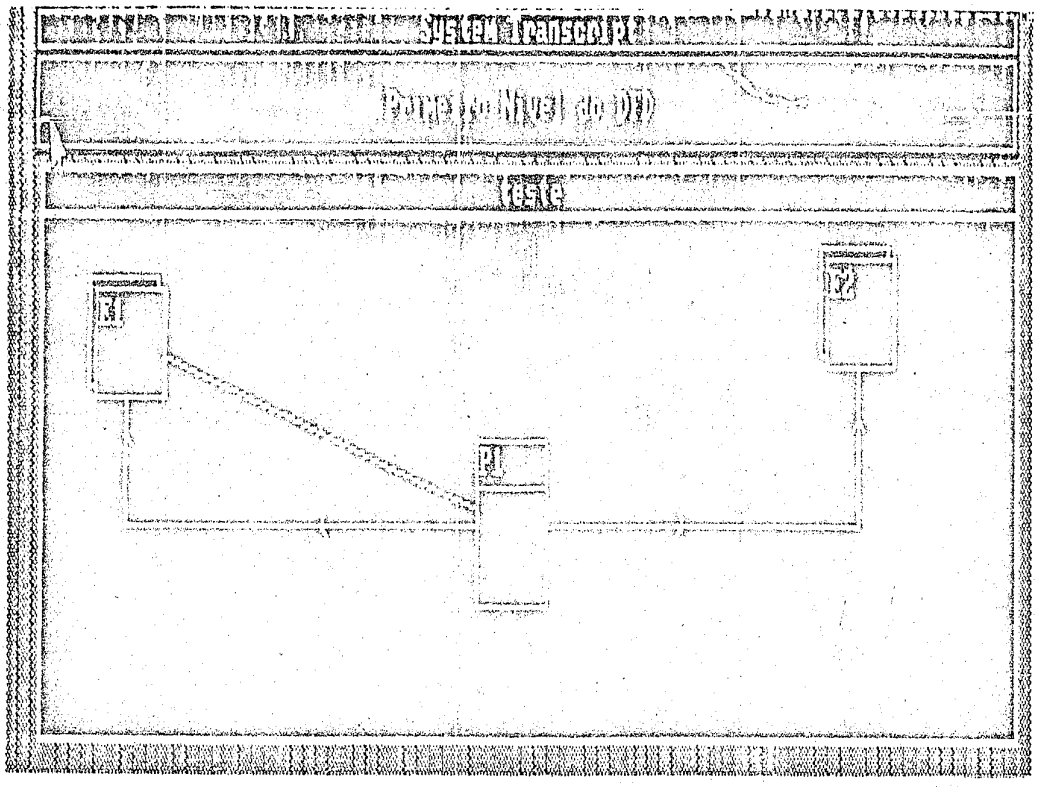


figura 11

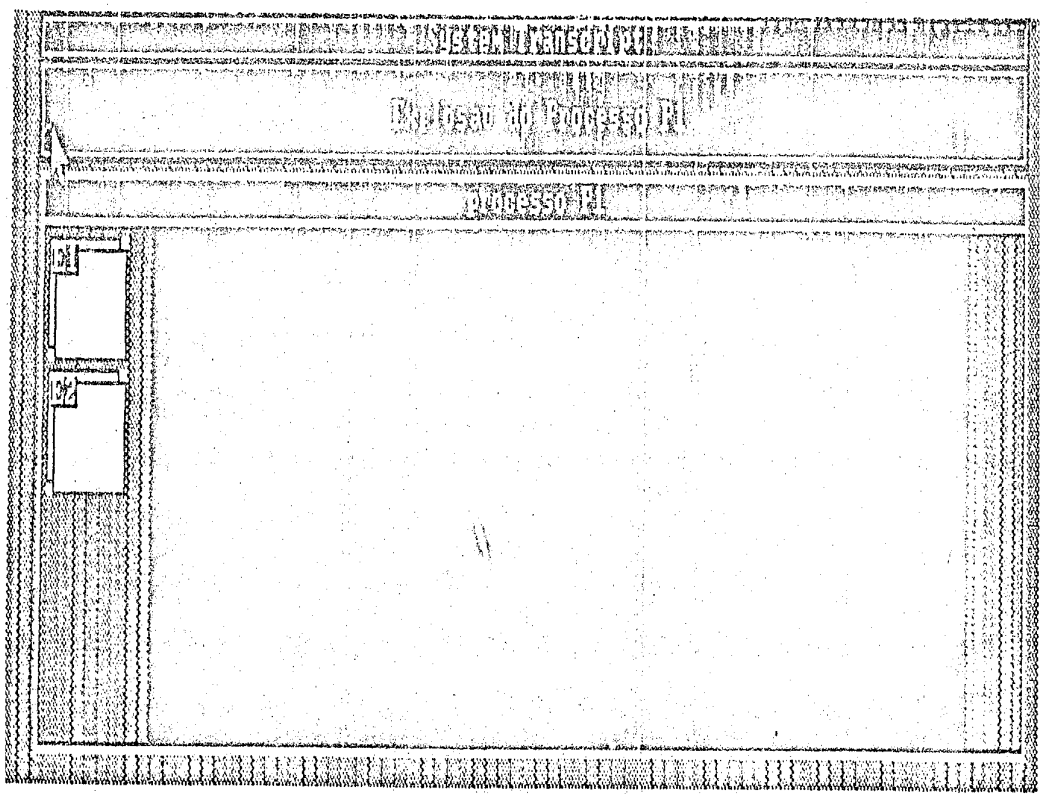


figura 12

As representações de E1 e E2 são chamadas "fantasmas". As operações disponíveis para "fantasmas" (mostradas na figura 13), sejam eles de entidades externas, processos ou depósitos de dados são : mover, conectar e desconectar. As verificações sobre conexões inválidas são também aplicadas a fantasmas. A coerência entre fluxos que entram e saem de fantasmas e os fluxos que entram e saem dos elementos que dão origem aos "fantasmas" é de responsabilidade do usuário.

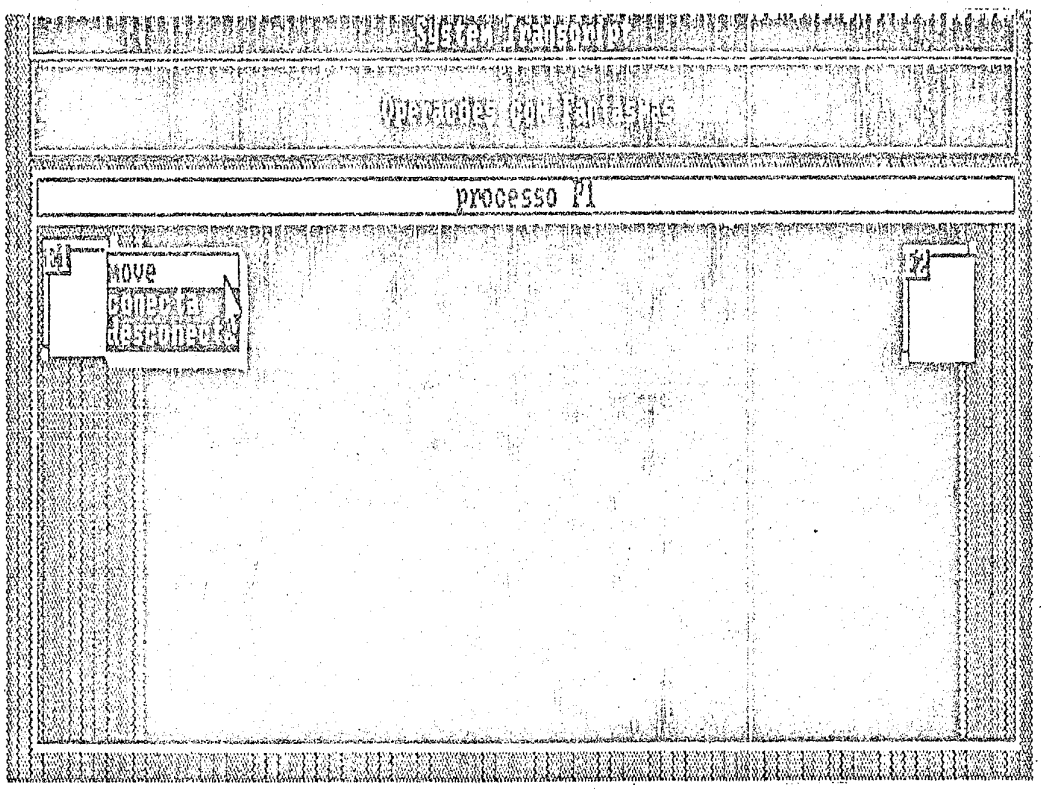


figura 13

De acordo com a figura 14 vê-se que as operações de criação de elementos em telas de explosão são : cria processo e cria depósito. Não faz sentido criar entidades externas em DFD's de explosão.

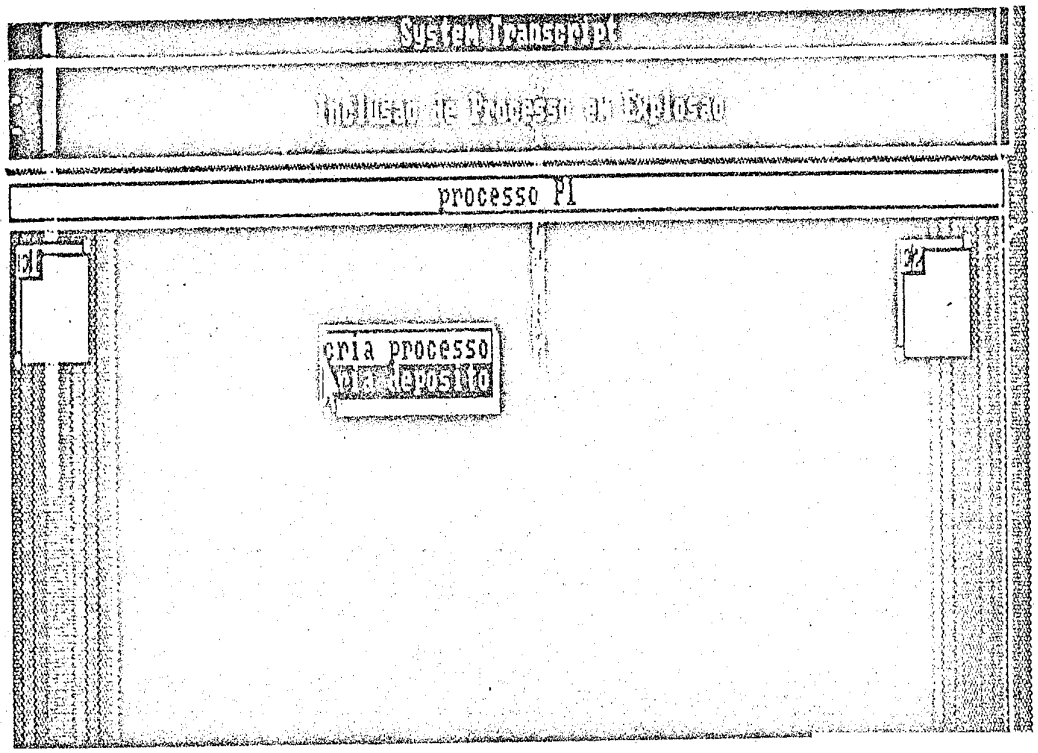


Figura 14

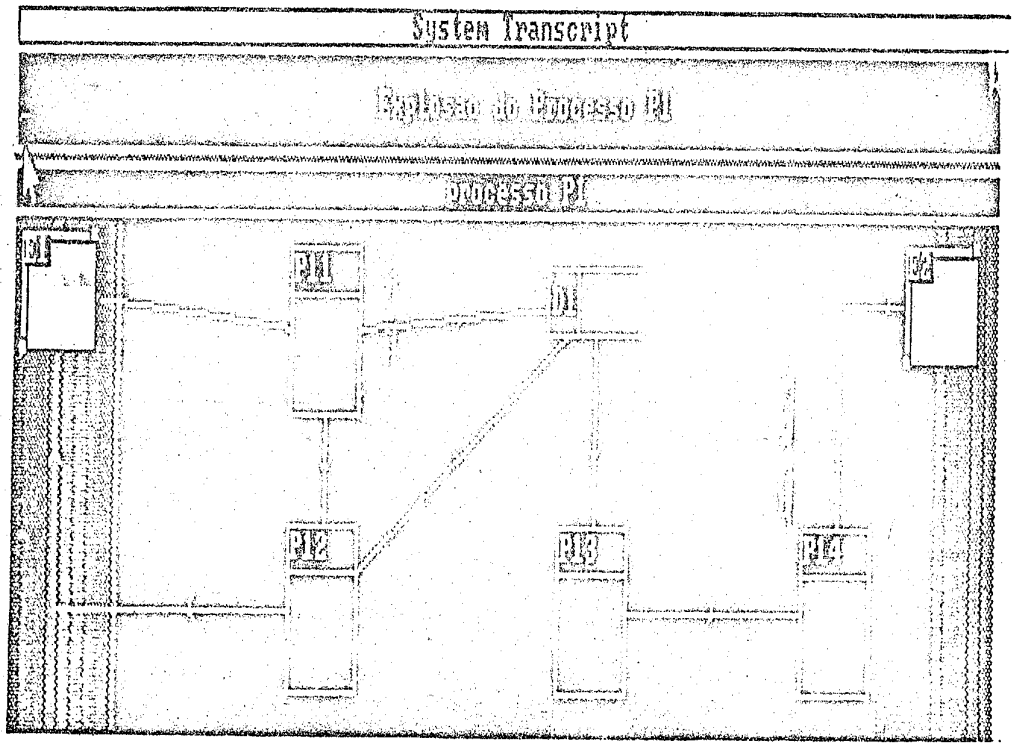


Figura 15

O DFD completo da explosão do processo "P1" está na tela da figura 15 e com isso conclui-se a especificação do DFD exemplo usado.

A chamada das regras heurísticas de transformação, que na versão atual do protótipo é feita por comando SMALLTALK, gera o Modelo de Objetos relativo ao DFD construído, visto na figura 16.

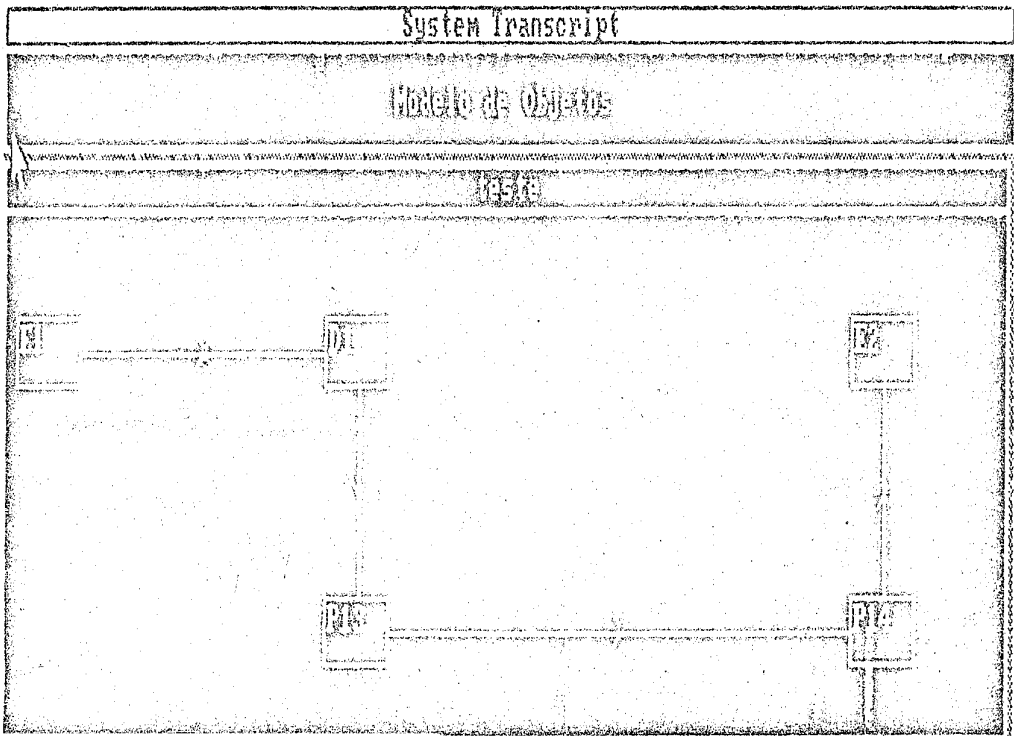


Figura 16

As operações de edição sobre o MO são as mesmas disponíveis para o DFD. Das operações de edição específicas de elementos, as únicas não disponíveis são as de "explosão" e "des..." (des-faz explosão), que não fazem sentido para objetos em um MO.

5. ARQUITETURA DA IMPLEMENTAÇÃO

Este protótipo, conforme já dito, foi implementado em Smalltalk-80 [3]. A principal razão para esta escolha foi realizar uma experiência paralela sobre orientação à objetos a nível de linguagem de programação. Nesta seção descrevemos a estrutura do programa em Smalltalk (basicamente as classes incorporadas ao sistema) e nossas impressões sobre o uso da linguagem em um sistema de razoável porte.

Praticamente todas as funções da ferramenta operam sobre uma representação interna para DFD's e uma para Modelos de Objetos (MO's). Este fato, aliado a Smalltalk ser orientado a objetos, faz a escolha de uma representação adequada para estas estruturas um ponto fundamental no desenvolvimento deste sistema.

Devido às grandes semelhanças entre DFD's e MO's, optamos pela criação de uma classe denominada Grafo, que reune as características comuns às estruturas de DFD's e MO's. A partir desta base comum foram definidas duas especializações, uma englobando as peculiaridades de DFDs e outra englobando as peculiaridades de MOs. Esta escolha tem duas importantes vantagens, sob o ponto de vista de reaproveitamento de software. A primeira, já mencionada, consiste na descrição única, via classe Grafo, de várias rotinas comuns a DFDs e MOs. A segunda, a mais longo prazo, é a possibilidade de futuros usos desta classe como base para ferramentas que manipulam outros tipos de grafos, como Redes de Petri ou diagramas Entidade-Relacionamento. Em outras palavras, com a implementação desta classe temos como produto colateral uma espécie de editor gráfico generalizado.

Para implementar a classe Grafo foram usadas duas classes auxiliares, Nó e Arco. Assim, um grafo é composto por um conjunto de nós e um conjunto de arcos; um nó mantém seus arcos de entrada e saída, e um arco mantém seus nós origem e destino. Além disto, todas estas classes têm um campo nome para identificação.

Seguindo a hierarquia acima, desenvolvemos um editor geral para grafos, que foi posteriormente especializado para editar DFDs e MOs. O editor geral se utilizou amplamente das facilidades gráficas já existentes no ambiente Smalltalk.

A interface com o editor é feita por uma janela gráfica ("GraphPane") que, por ser da biblioteca do sistema, apresenta todas as facilidades comuns a janelas em Smalltalk. Esta janela se comunica com o Grafo sendo editado, passando a ele qualquer ação do usuário que não possa ser tratada localmente. Quando o grafo recebe esta mensagem, ele verifica se existe algum objeto (nó ou arco) na posição do cursor. Se existir, o controle é passado a este objeto, que apresenta seu próprio menu e realiza a operação desejada sobre si próprio. Caso não exista nenhum objeto na posição selecionada, o grafo apresenta o chamado "menu de fundo", com operações gerais sobre o grafo e operações de criação de novos objetos. Com esta estrutura, a interface fica bastante modularizada, e cada objeto pode apresentar um menu próprio, contendo apenas as operações adequadas para a situação (p.e., apenas os processos tem a opção de explodir em seus menus).

A rotina de desenho do grafo também é bastante modular. Cada objeto é dotado de um método para se auto desenhar, e o grafo simplesmente varre seus objetos invocando esta rotina. Novamente,

qualquer particularidade no desenho de um determinado objeto fica limitada ao método de desenho deste objeto, sem afetar o resto do sitema.

A impressão final sobre Smalltalk como linguagem de programação teve altos e baixos. Como ponto positivo podemos citar o alto grau de reaproveitamento de código alcançado, fato que acelerou bastante o tempo de desenvolvimento do protótipo. Como pontos negativos destacamos a falta de maiores verificações em tempo de compilação e, contra a impressão geral, alguns problemas de modularidade. A seguir explicamos o porquê destas impressões.

O mecanismo de herança se mostrou estremamente útil. Conforme relatado anteriormente, toda a arquitetura do sistema foi apoiada neste conceito, permitindo um alto grau de reaproveitamento, bem como um desenvolvimento mais incremental. A medida que as classes mais gerais iam sendo testadas passavamos à implementação de suas especializações, tendo que testar apenas as modificações introduzidas, e não a classe toda.

Outro ponto forte para reaproveitamento é a biblioteca do sistema. Em especial, o programa inteiro foi desenvolvido sem necessidade de implementarmos nenhuma estrutura de dados, usando apenas as fornecidas pela biblioteca (classe Collection e suas subclasses). Para isto também colaborou o alto grau de polimorfismo permitido pela linguagem, que por outro lado tem vários defeitos.

A falta de tipagem estática é uma das consequências danosas do polimorfismo de Smalltalk. Apesar de existirem sistemas de tipagem estática para linguagens polimórficas (p. e. [4]),

Smalltalk optou pela tipagem dinâmica. Com isto, a compilação perde boa parte do seu poder de detetar erros, aumentando o tempo gasto em testes e correções. Com o mecanismo de "late-binding", mesmo erros de digitação nos seletores de mensagens só são detetados nos testes.

Finalmente, temos a questão da modularidade. Muito se tem elogiado a modularidade de Smalltalk, onde cada objeto encapsula suas variáveis, que só podem ser acessadas por métodos do próprio objeto. O problema sentido foi que, apesar de cada objeto encapsular suas variáveis, ele não cuida dos objetos ao qual estas váriaveis se referem, que podem ser modificados externamente. Desta forma, modificações no estado de um objeto podem se propagar para outros objetos (que se referem a este) de formas muitas vezes inesperadas, e sempre não explícitas. Com isto, muitas vezes pequenas alterações na especificação da ferramenta obrigavam a modificações em muitas diferentes partes do sistema.

5. CONCLUSÃO

O trabalho apresentado descreve uma ferramenta de apoio automático à atividade de modelagem de sistemas de software.

As diferentes correntes de tecnologia que suportam a atividade de modelagem adotam, normalmente, conjuntos de conceitos básicos que enfatizam aspectos específicos dos sistemas a serem modelados.

Frequentemente é importante poder abordar vários destes aspectos em um mesmo caso de desenvolvimento, seja por sua importância individual ou pela forte complementaridade com que estes se apresentam em determinadas aplicações práticas.

Neste sentido, justifica-se a proposta de apoio ao desenvolvimento de software apresentada. Nela combinam-se duas linhas de construção baseadas em paradigmas distintos (aplicados à mesma atividade) e eventualmente complementares.

As expansões mais imediatas vislumbradas para a ferramenta apresentada são :

a) A construção de um interpretador que possibilite a simulação da execução de MO's gerados com o auxílio da ferramenta.

b) O desenvolvimento de uma nova versão do componente gráfico responsável pelo desenho dos MO's. Esta versão implementaria uma heurística de distribuição espacial dos objetos de um MO na tela.

c) Melhora nos mecanismos de verificação de consistência de explosões de processos.

REFERENCIAS BIBLIOGRÁFICAS

1 - BOOCH, G. Object-oriented development. IEEE Transactions on Software Engineering, SE-12 (2):211-221, feb. 1986.

2 - GANE, C. & SARSON, T. Análise estruturada de sistemas. Rio de Janeiro, Livros Técnicos e Científicos Editora, 1984. 253p.

3 - GOLDBERG, A. Smalltalk-80 The language and its implementation. Massachusetts, Addison-Wesley, 1984.

4 - MILNER, R. A Theory of Type Polimorphism in Programming. Journal of Computer and System Sciences, 17(3), 1978.

5 - PASCOE, G.A. Elements of object-oriented programming. BYTE: 139-144, aug. 1986.

6 - ROTENBERG, H.B.; STAA, A.; IERUSALIMSCHY, R. Modelo orientado a objetos para o processo de desenvolvimento. Anais do IV Encontro de Trabalho do Projeto ETHOS. Petrópolis, abril 1987.

7 - STEFIK, M. & BOBROW, D.G. Object-oriented programming: themes and variations. The AI Magazine: 40-62, 1984.