UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
POS-GRADUAÇÃO EM CIENCIA DA COMPUTAÇÃO

PROPOSTA DE UM
EDITOR DIAGRAMATICO GENERALIZADO

por

WALCÉLIO LOUZADA MARTINS MELO

Dissertação submetida como requisito parcial para
a obtenção do grau de Mestre em
Ciência da Computação

Prof. Roberto Tom Price
Orientador

Porto Alegre, março de 1989

CATALOGAÇÃO NA FONTE

Melo, Walcélio Louzada Martins
Proposta de um Editor Diagramático Generalizado . Porto Alegre, PGCC da UFRGS, 1989.
1 v.
Diss. (mestr. ci. comp.) UFRGS-PGCC, Porto Alegre, BR-RS. 1989.

Dissertação: Engenharia de Software: CASE: Ambientes de Desenvolvimento: Editores diagramáticos: Diagramas;

486

681.32.063(043)
M528P

CPD
1900/4657-4
1991/09/19

À meus pais, Walter e Abigail, e
à minha esposa Keila.

AGRADECIMENTOS

Agradeço ao SERPRO - Serviço Federal de Processamento de Dados pelo apoio financeiro.

Ao CPGCC - Curso de Pós-Graduação em Ciência da Computação: agradeço aos funcionários e colegas pelo apoio e companheirismo e aos docentes pelo ensinamentos.

Aos grandes colegas e amigos Deoni, Roque, Angélica principalmente pelo início da caminhada neste curso. Ao Aristeu e Javan pela excelente amizade consolidada neste período. A Karin pelos "abstracts". Ao Alvaro, companheiro de vôo e de noites de trabalho.

Aos professores Roberto Tom Price, Raul Weber, Carla Dal Sasso Freitas e Carlos Alberto Heuser pela orientação, encorajamento e apoio.

Aos alunos da graduação Mônica Spotorno da Silva, Israel Paulino da Rosa e Silva, Silvio César Camargo e Flávio Roberto Mrack pela ajuda na revisão deste, e também pelo valoroso auxílio na programação do protótipo.

Agradeço aos meus familiares pelo apoio, sem o estímulo dos quais seria impossível a elaboração deste trabalho. A minha esposa Keila pela paciência e compreensão.

E principalmente a Deus, nosso Pai e Senhor, por ter me confortado, iluminado e protegido durante todos os momentos da minha caminhada.

SUMARIO

LISTA DE ABREVIATURAS

AC	Administrador de CASE
CASE	Computer-Aided Software Engineering
CPGCC	Curso de Pós-graduação em Ciência da Computação
ED	Editor Diagramático
EDE	Editor Diagramático Específico
EDG	Editor Diagramático Generalizado
ES	Engenharia de Software
IBM	International Business Machine
KBSA	Knowledge-Base Software Assistent
MED	Meta Editor Diagramático
PC	Personal Computer
SERPRO	Serviço Federal de Processamento de Dados
SGBD	Sistema de Gerência de Banco de Dados
UFRGS	Universidade Federal do Rio Grande do Sul

LISTA DE FIGURAS

RESUMO

Este trabalho apresenta o **EDG**, um Editor de **D**iagramas **G**eneralizado, cujo objetivo é servir de ferramenta de suporte à utilização de métodos de desenvolvimento de *software* baseados em notações diagramáticas. No trabalho são discutidos alguns problemas que influenciam a produtividade e qualidade de software, apontando-se o uso de sistemas **CASE** (*Computer-Aided Software Engineering*) como uma resposta para estes problemas. Tais sistemas são caracterizados, a fim de mostrar porque os editores diagramáticos constituem um dos seus principais componentes. As diretrizes do projeto do **EDG**, as principais características da interface com seus usuários e alguns exemplos de uso são mostrados através da descrição do protótipo implementado.

ABSTRACT

This work presents the **EDG**, a generalized diagramatic editor, whose objetive is to support the use of diagram-based techniques for software development. Some problems which influence on software productivity and quality are discussed, pointing out the use of CASE systems (*Computer-Aided Software Engineering*) as an answer for these problems. Such systems are characterized in order to show why diagramatic editors constitute one of their main components. The EDG design guidelines, the main features of the user interface, as well as some examples of its use, are presented through the description of the prototype implemented.

1 INTRODUÇÃO

Em engenharia de software, o problema fundamental tem sido a busca de maior produtividade na construção de aplicações e a obtenção de melhor qualidade nos programas construídos.

Várias metodologias de desenvolvimento de software foram elaboradas com o objetivo de aumentar a qualidade de software. Para apoiar tais metodologias várias ferramentas de desenvolvimento e manutenção foram construídas, buscando-se uma maior automatização do processo de projeto de programas na procura de uma maior facilidade de expressão com algum rigor de apresentação.

As ferramentas de desenvolvimento de software podem ser classificadas segundo a sua ação sobre os diversos objetos significativos na área de software [PRI 87], sendo estes objetos:

- Documentos.

  Objetos não estruturados para descrição de algum aspecto do sistema alvo;

- Especificações.

  Objetos construídos pelo engenheiro de software (ES), segundo algum formalismo como, por exemplo, diagramas de fluxo de dados;

- Programas.

  Objetos construídos pelos ES, possivelmente com auxílio de ferramentas de apoio, em alguma linguagem de alto nível;

- Código.

  Produto final do desenvolvimento de um software, tal como comandos de manipulação de banco de dados e código objeto produzido por compiladores.

As ferramentas desempenham ações sobre estes objetos, podendo ser classificadas em:

- Ferramentas de auxílio à preparação de objetos;
- Ferramentas de análise de objetos;
- Ferramentas de geração de objetos.

Os editores são os exemplos mais comuns de ferramentas de auxílio à preparação de objetos. Uma classe de editores bastante útil, usados durante o processo de preparação de objetos, são os editores diagramáticos (ED), pois além do apoio que estas ferramentas fornecem ao engenheiros de software no processo de projeto e refinamento de software, possibilitam uma forma de comunicação padronizada entre as equipes de desenvolvimento, e o resultado de sua utilização já é uma documentação da fase de preparação.

OS EDs são ferramentas utilizadas nas metodologias que fazem uso de notações diagramáticas, tais como notações para representar dados (diagramas Entidade-Relacionamento [CHE 77]),diagramas de estruturação de dados [JAC 75] e [WAR 85]; para especificação do fluxo de dados [GAN 83], diagramas de decomposição estruturados; diagramas para representação do controle de execução (redes de Petri); diagramas para representação da lógica de execução de módulos (diagramas de ação, tabelas e árvores de decisão, diagramas Nassi-Shneiderman [NAS 73] [CHA 74]). Vários EDs foram desenvolvidos, muitos dos quais estão disponíveis comercialmente, para apoiar algumas destas notações diagramáticas.

Como exemplo de ferramentas que fazem uso intensivo de EDs podem ser citados o Sistema Integrado para Produção de Software - SIPS [IRA 86] e o MOSAICO [IES 86], ambas desenvolvidas no Brasil. Em diversas universidades brasileiras protótipos de outros ED estão em desenvolvimento, como na COPPE com a prof. Ana Rocha [AGU 87], na UFRGS com o prof. Daltro Nunes [LEA 87] e na PUC-RJ com o prof. Rubens Melo [MEL 87], somente para citar alguns.

1.1 Motivação

O processo de desenvolvimento de software é uma contínua transformação de notações, desde a especificação dos requisitos até a produção de código executável. Em ambientes automatizados estas transformações são apoiadas por ferramentas de análise e de geração.

A incorporação de ferramentas de apoio à transformação é difícil se várias formas de representação interna são utilizadas, além da problemática intrínseca às metodologias, pois muitas não são adequadas para uma efetiva automatização das transformações. Tais transformações exigem conhecimento sobre as aplicações que estão sendo desenvolvidas, além de conhecimento sobre as metodologias propriamente ditas.

EDs são ferramentas de apoio à produção de documentos. Muitos EDs disponíveis são especializados em determinadas notações diagramáticas, notações estas propostas por metodologias desenvolvidas para serem utilizadas manualmente. Como conseqüência desta especialização, os ES que necessitem utilizar mais do que um editor (por exemplo, para usar mais de uma notação, para distintas atividades de projeto) enfrentam uma série de problemas, tais quais:

a) Funções de edição incompatíveis.

Em instalações que possuam EDs distintos, pode vir a ocorrer que funções com identificadores iguais ocasionem ações distintas, ou, em caso contrário, funções com identificadores diferentes executem ações idênticas. Isto pode levar o ES a cometer erros semânticos durante o uso destes EDs e isto, além de acarretar desconforto e insegurança ao ES no uso da ferramenta, pode causar perdas na produtividade.

b) Organização da tela.

Como na gerência de cardápios, a forma com que diferentes editores fazem uso da tela do terminal, além das características da interação propriamente ditas, também exerce influência na produtividade do ES, devido ao tempo consumido na readaptação ao ED.

c) Forma da ativação ou entrada do sistema.

EDs diferentes podem usar dispositivos de entrada distintos. O ES poderá ter que trabalhar com um editor que utilize "light pen" como ativador, com outro que utilize "mouse", outro que utilize mesa digitalizadora, e assim por diante. Este fator também acarreta perda de produtividade do ES, pois exige do mesmo conhecimento e destreza de manipulação de vários dispositivos gráficos de entrada.

d) Padrões de saídas distintos.

Alguns EDs somente possuem "hard-copy" como meio de impressão dos diagramas confeccionados. Além deste inconveniente, em alguns EDs a área disponível para desenho é limitada a tamanhos padrões, por exemplo o tamanho A4.

e) Adaptabilidade às notações impostas pelo ED.

Os EDs geralmente trabalham utilizando um biblioteca fixa de símbolos gráficos, ou seja, já são construídos para manipular um universo fixo de símbolos como, por exemplo, alguns tipos de formas elípticas, formas poligonais, segmentos de reta, e assim por diante. Esta limitação pode levar uma instalação, que adquira um ED e utilize notações diagramáticas não disponíveis neste editor, a mudar o uso de suas notações para adaptar-se àquelas fornecidas pelo fabricante. Em caso contrário, terá de construir seu próprio ED, sendo esta alternativa nem sempre viável.

f) Estruturas de dados incompatíveis.

Algumas vezes, se faz necessário o transporte do trabalho realizado por um ES em um ED para outro ED. Esta necessidade é devida a facilidades e/ou limitações de um certo tipo de editor em relação a outro. A conversão das informações de um ED para outro nem sempre é uma tarefa fácil, pois a forma de armazenamento das informações de dois EDs distintos quase sempre é incompatível, além de ser, em alguns casos, oculta pelos fornecedores.

g) Formatação automática.

A maioria dos EDs não possui facilidades de formatação automática dos diagramas desenhados. Desenhar um diagrama esteticamente é uma tarefa árdua, que pode levar um ES a gastar horas de trabalho em frente a uma tela de desenho. Através de facilidades de formatação automática de diagramas este tempo pode ser minimizado. Alguns EDs possuidores desta facilidade geralmente não permitem ao ES fazer ajustes estéticos manualmente, e isto pode levar a um descontentamento do ES com o padrão estético adotado pelo ED.

Estas dificuldades levam a considerar a conveniência de construir um editor diagramático generalizado, capaz de adaptar-se às notações diagramáticas de diversas técnicas de engenharia de software. Possuidor de uma interface amigável e homogênea com os seus diversos usuários.É conveniente que tal editor possua facilidades de janelas, para permitir ao ES ver e editar várias partes de diagrama, utilize o *mouse* ou equivalente para edição, e possa imprimir de forma adequada os diagramas editados.

1.2 Estrutura Geral do Trabalho

O capítulo 2 mostra vários problemas decorrentes da baixa produtividade e qualidade de sistemas aplicação, mostrando fatores influenciadores destes problemas. A seguir, são abordadas algumas alternativas que podem ser tomadas para aumentar a produtividade dos profissionais de processamento de dados e, finalmente, aponta-se uma possível solução através do uso das ferramentas CASE (**C**omputer-**A**ided **S**oftware Engineering).

O capítulo 3 caracteriza os sistemas CASE, comenta como tais sistemas podem aumentar a produtividade e mostra seus principais componentes. Os editores diagramáticos, uma das principais ferramentas de um sistema CASE, são abordados com maior detalhe, mostrando suas características e seus usos no processo de desenvolvimento de software.

O capítulo 4 apresenta um editor diagramático generalizado, denominado **EDG**, cujo principal objetivo é prover um suporte para a construção de editores. São comentados os principais módulos e funções deste editor.

O capítulo 5 mostra com maior detalhe as principais características da interface do **EDG** com seus usuários.

O capítulo 6 mostra alguns exemplos de utilização do **EDG**. São mostradas a especificação e utilização dos métodos "Entidade e Relacionamento" [CHE 77], "Redes Marcadas" [HEU 87] e "Transporte".

O capítulo 7 comenta a implementação do **EDG**, mostrando as características da estrutura de dados e do programa.

O capítulo 8 conclui o trabalho e apresenta direções para futuras extensões ao **EDG**.

2 PRODUTIVIDADE E QUALIDADE DE SOFTWARE

Os profissionais de processamento de dados trabalham pressionados pela necessidade de desenvolver novos sistemas e, também, pela necessidade de manter e melhorar o volume sempre crescente de sistemas já desenvolvidos. Para diminuir as pressões é preciso facilitar a manutenção dos sistemas já desenvolvidos e acelerar o desenvolvimento de novos sistemas. Para que tais metas possam ser alcançadas é necessário automatizar cada vez mais as tarefas realizadas por estes profissionais [WHA 87].

Segundo MacDonald [MAC 87], se os ES empregassem em benefício próprio a tecnologia por eles utilizada para automatizar as atividades em outras áreas, eles melhorariam o processo de desenvolvimento e manutenção de sistemas e com isso, aumentariam a produtividade de seu próprio trabalho e a qualidade dos sistemas produzidos. Para a automação ser efetiva ela deve estar presente em todas as fases do ciclo de vida dos sistemas, e seus componentes integrados de forma homogênea.

A pouca automação não é o único fator determinador para baixa produtividade e qualidade dos sistemas desenvolvidos. Em [FAI 85], Fairley cita vários outros fatores, variando desde aspectos individuais até gerenciais. Abaixo são citados alguns destes fatores:

- Habilidade individual.

Como o desenvolvimento e manutenção de sistemas são ainda em grande parte atividades trabalhosas, produtividade e qualidade são funções diretamente proporcionais ao esforço e aptidão individual, sendo que a capacidade individual pode ser vista tanto do aspecto da competência ou da familiaridade do indivíduo com uma área de aplicação em particular.

- Comunicação em grupo.

Em geral, um software grande e complexo é desenvolvido por um grupo de pessoas. Logo, é possível que alguns membros do grupo não

compreendam as verdadeiras tarefas dos módulos que eles devem desenvolver, cometendo, assim, erros que somente serão identificados posteriormente. Para enfrentar tal problema muitas técnicas de engenharia de software estão disponíveis visando melhorar a comunicação entre os indivíduos de uma equipe de desenvolvimento, como por exemplo, "walkthroughs", revisão de projetos e exercícios de inspeção de código.

- Complexidade do produto.

Quanto maior a complexidade do produto a ser desenvolvido e/ou mantido menor será a produtividade.

- Notações apropriadas.

Notações adequadas podem esclarecer o relacionamento e interações dos componentes dos sistemas. Notações padronizadas provêm um dos veículos de comunicação entre os indivíduos envolvidos no projeto e facilitam o uso de ferramentas automatizadas.

- Abordagem sistemática.

O uso de métodos e técnicas formais ou semi-formais é uma forma de disciplinar o desenvolvimento de software, além de suavizar o impacto das mudanças de requisitos e de projeto sobre a qualidade do software produzido.

- Nível de tecnologia.

Estabilidade e disponibilidade do ambiente computacional exercem uma forte influência na produtividade dos programadores e qualidade de software.

- Nível de confiabilidade.

Boehm [BOE 81] estima que a taxa de produtividade é muito baixa quando um alto nível de confiabilidade é exigido. Segundo ele, esta relação pode chegar a um fator de 2:1.

- Facilidades e recursos.

Facilidade de acesso a equipamentos eficientes e instalações de trabalho adequadas são, segundo alguns estudos [COU 78], fatores motivadores no desempenho de programadores. Logo, facilidades e recursos oferecidos às equipes de desenvolvimento podem resultar em ganhos de produtividade.

- Treinamento adequado.

Muitos dos profissionais de engenharia de software não obtiveram formação acadêmica adequada, e mesmo os que a receberam não possuem a maioria dos requisitos necessários de um engenheiro de software.

- Habilidade de gerência.

Projetos de software são frequentemente gerenciados por pessoas que não possuem suficiente experiência em engenharia de software, ou, por outro lado, muitas vezes os projetos são gerenciados por excelentes engenheiros de software com baixa capacidade gerencial.

- Entendimento do problema.

Muitos usuários não conseguem expor suas necessidades com clareza, e como se isto não bastasse, muitos engenheiros de software não conhecem a área de aplicação e não entendem o que os usuários desejam. Isto pode ocasionar a entrega de produtos não adequados às necessidades dos usuários, tornando necessária a realização de trocas e correções no produto, e por conseguinte, degradando a produtividade e a qualidade do serviço.

- Tempo disponível.

Projetos de software são sensíveis ao tempo disponível para desenvolvê-los e ao número de pessoas envolvidas. As dificuldades de comunicação e o tempo de aprendizado dos envolvidos em um projeto pode acarretar perdas na produtividade. A determinação do número de encarregados e da fixação de prazos é uma importante e difícil tarefa na estimação da produtividade.

- Habilidades exigidas.

A prática de engenharia de software requer bastante perícia dos responsáveis pelo desenvolvimento e manutenção dos produtos. Devido a grande quantidade de atividades nesta área, conceituais por natureza, é exigido um alto grau de criatividade para solucionar problemas. É necessário, também, que o engenheiro de software seja sociável, comunicativo e saiba expor suas idéias com clareza, além de possuir um apurado raciocínio dedutivo.

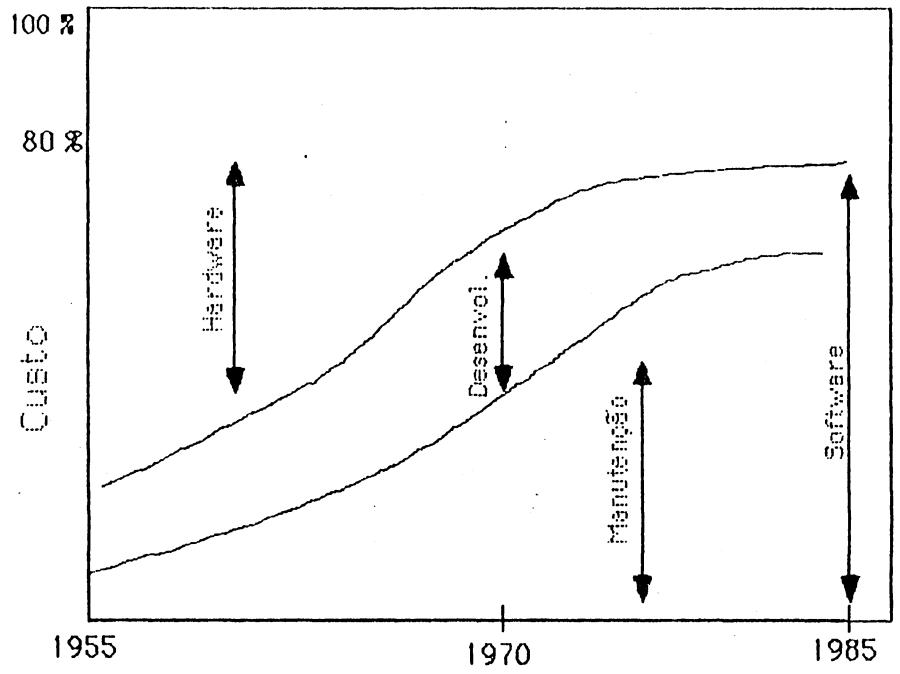


Figura 2.1 - Evolução do custo de desenvolvimento e manutenção de sistemas [BOE 81].

A preocupação com a produtividade de sistemas é devida às tendências crescentes, tanto na demanda como no custo de manutenção e desenvolvimento dos mesmos. O custo de manutenção representa uma porção considerável no custo final do software [BOE 81]. Por exemplo, Suydam [SUY 87] relata o caso da força aérea dos EUA, que investiu US$ 85 milhões no desenvolvimento de software para o caça F-16 e estima gastar US$ 250 milhões em manutenção, uma quantia quase três vezes maior que o custo de desenvolvimento. Já no ano de 1973 Barry Boehm projetou que para 1985 o custo de software (desenvolvimento e manutenção)

ultrapassaria 90% do custo total dos sistemas desenvolvidos (combinando os custos de equipamentos e programas), em parte, também, porque o custo do hardware decresceu incrivelmente, como pode ser visto na figura 2.1.

Boehm [BOE 87] coloca que atualmente a taxa de crescimento do custo de software é de 12% ao ano, sendo que em 1985 este custo totalizou, aproxidamente, US$ 11 bilhões no Departamento de Defesa dos EUA, US$ 70 bilhões nos EUA como um todo e US$ 140 bilhões em todo o mundo (figura 2.2). Um ganho de 20% na produtividade resultaria a nível mundial numa economia de US$ 90 bilhões em 1995.

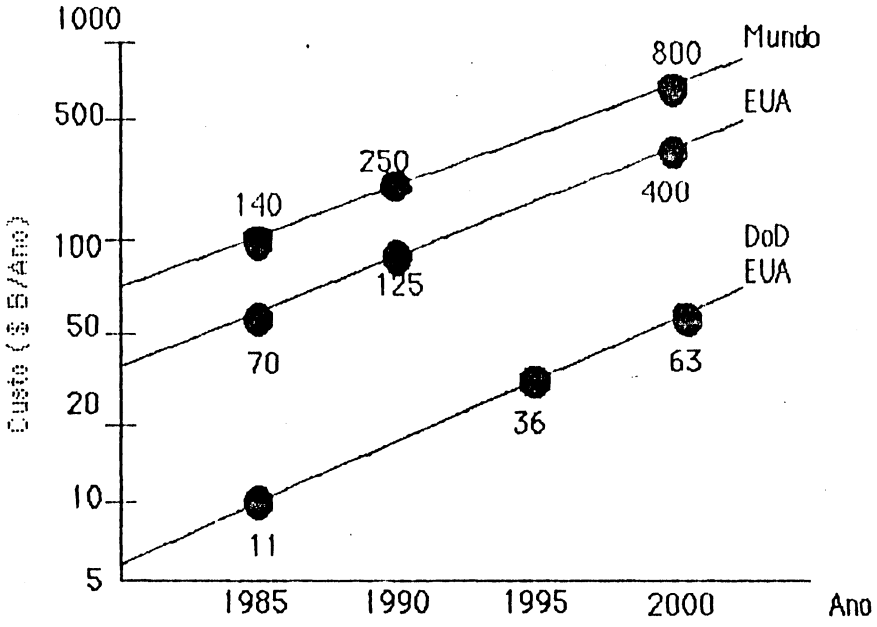


Figura 2.2 - Tendências de evolução dos custos de software [BOE 87].

O aumento crescente na demanda de aplicações comporta-se como uma curva exponencial sem saturação [BOE 87], conforme mostrado na figura 2.3. Para atender tal demanda estima-se que o número de analistas/programadores e engenheiros de software poderá chegar a 1.2 milhões em 1990 [BOE 83], ou a 2 milhões segundo previsões mais pessimistas [SCH 81], somente nos EUA, se não forem tomadas medidas que proporcionem um aumento na produtividade.

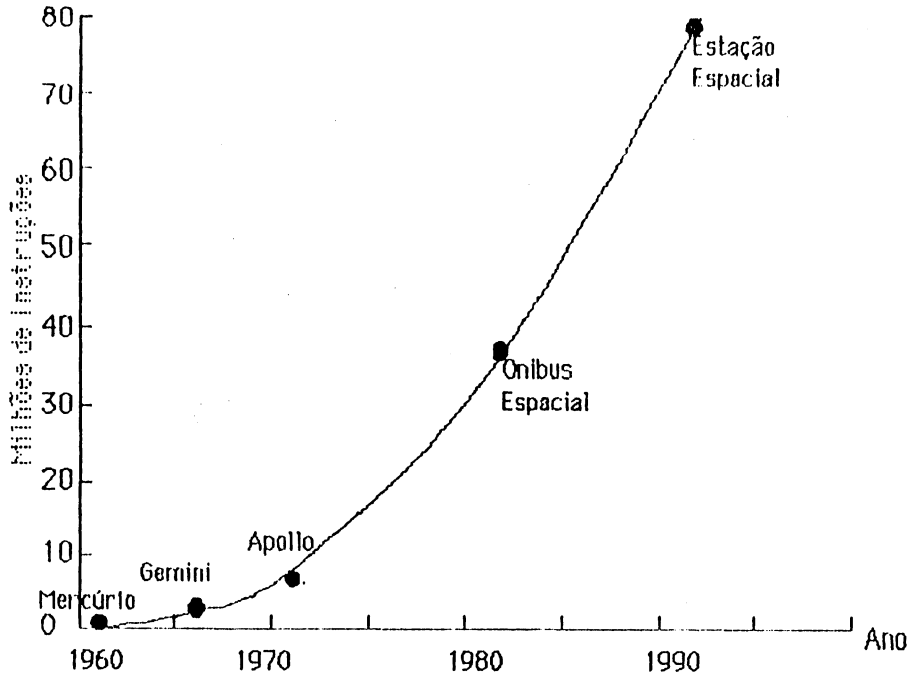


Figura 2.3 - Crescimento na demanda de sotware: programa espacial americano [BOE 87].

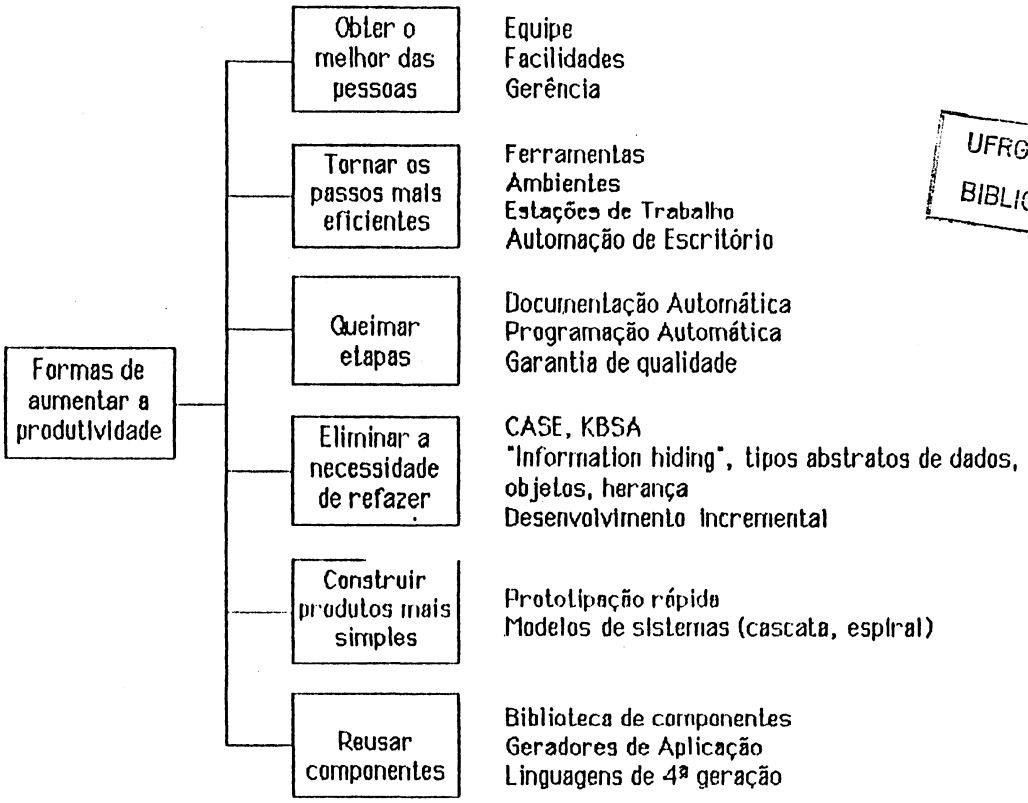


Figura 2.4 - Arvore de oportunidades para aumento de produtividade [BOE 87].

Boehm sugere, através de uma "árvore de oportunidades" [BOE 87], mostrada na figura 2.4, várias alternativas que podem ser seguidas para alcançar o objetivo de aumentar a produtividade. A árvore de oportunidades fornece as seguintes opções:

1) Obter o melhor das pessoas.

Selecionar adequadamente, motivar e gerenciar as pessoas envolvidas no processo de desenvolvimento de software pode elevar consideravelmente a produtividade. A escolha de profissionais competentes, capazes e, melhor ainda, familiarizados com as áreas de aplicação, sistemas computacionais e linguagens de programação diminui o tempo do processo de desenvolvimento, como também aumenta a qualidade do mesmo [BOE 81] [BOE 87] [FAI 85].

2) Tornar os passos mais eficientes.

Nos últimos 20 anos, várias ferramentas automatizadas de desenvolvimento de software foram construídas e distribuídas comercialmente. A maioria das ferramentas foram construídas para auxiliar a codificação, depuração e teste de programas. São exemplos deste tipo de ferramentas os editores de programas, compiladores, ligadores, depuradores, geradores de programas ou de aplicações (como as chamadas linguagens de 4ª geração [MAR 82]). O uso destas ferramentas torna as fases de codificação, teste e depuração mais rápidas e eficientes. Contudo, ferramentas que auxiliem as fases de especificação de requisitos, projeto lógico e físico, como, também, a validação do projeto/especificação, reusabalidade e eliminação de redundância de esforços ainda são pouco difundidas.

Embora algumas experiências demostrem [BOE 84] que as ferramentas são mais úteis quando integradas a um ambiente homogêneo de desenvolvimento de software, muitas ferramentas disponíveis não foram integradas, mas somente agrupadas dentro de um "tool kit", através do qual o usuário pode selecionar as ferramentas desejadas. Esta abordagem resulta na co-existência de diversas ferramentas com diferentes

linguagens de comandos e interfaces em um mesmo "ambiente", complicando, assim, tanto o processo de aprendizado das diversas ferramentas como também diminuindo a eficiência das mesmas.

3) Queimar etapas.

O uso de ferramentas automatizadas que possibilitem eliminar totalmente as tarefas manuais são de grande valia no aumento de produtividade, como, por exemplo, os compiladores na década de 50 ou as linguagens de 4ª geração [MAR 82] na década de 70. Vários esforços foram feitos no sentido de gerar automaticamente programas livres de erros a partir de especificações de alto nivel, eliminando-se assim todas as fases de projeto e teste/depuração dos mesmos. Tais esforços obtiveram resultados somente quando aplicados a um domínio muito restrito no universo de programação [BAL 83].

4) Eliminar a necessidade de refazer componentes.

Boehm [BOE 87], com base nos estudos de Porter [POR 85], coloca que mais de 50% do tempo do ciclo de vida de sistemas é gasto realizando-se ajustes em tarefas concluídas e consertando-se erros de definição ou entendimento da especificação. Logo, a maior oportunidade de aumentar a produtividade no desenvolvimento de sistemas está justamente na solução destes problemas. Para tal tem sido propostas ferramentas e ambientes como CASE (computer-aided software engineering), KBSA (knowledge-based software assistants), técnicas e linguagens de programação, utilizando conceitos de tipos abstratos de dados, herança, etc., como ADA e SMALLTALK, além da utilização de modelos de ciclo de vida de sistemas mais elaborados, como por exemplo o modelo em espiral (onde o modelo cascata é deficiente), bem como prototipação rápida e reusabilidade de software.

5) Simplificar a construção de produtos.

Usualmente a produtividade é medida através do número de linhas de código fonte produzidas em uma unidade de tempo; quanto maior a quantidade de código construída atavés dos geradores de aplicações,

linguagens de quarta geração e o uso de componentes já desenvolvidos, maior a produtividade.

6) Reusar componentes.

Bons ganhos de produtividade são conseguidos quando componentes de software já existentes são reutilizados na confecção de novos produtos. Tais componentes variam de biblioteca de programas (código objeto) até especificação de requisitos.

Embora a baixa produtividade seja um grande problema para os engenheiros de software, este não é o único. Em [BUR 87], por exemplo, Burchett identifica outros problemas não menos graves, como:

- Erros de identificação de necessidades do usuário e objetivos do sistema. A insatisfação do usuário com os sistemas produzidos e entregues é, em muitos casos, decorrente de definições inadequadas, feitas nos estágios iniciais do ciclo de desenvolvimento de sistemas;

- Baixa pradronização. Engenheiros de software estão, muitas vezes, pouco dispostos a aceitar o esforço e a disciplina necessária para manter boas padronizações;

- Documentação deficiente. É pouco frequente encontrar sistemas bem documentados, e é mais raro ainda encontrar documentação atualizada com a evolução dos sistemas.

- Baixa qualidade dos sistemas desenvolvidos. É freqüente encontar sistemas produzidos com baixa qualidade, em parte isto é devido aos erros decorridos nos procedimentos de projeto e análise.

Na década de 70 as técnicas de análise e programação estruturada trouxeram alguma disciplina ao processo de desenvolvimento de grandes e complexas aplicações [YOU 74]. De acordo com o próprio Yordon, tais técnicas não atingiram plenamente os objetivos desejados pois, apenas

10% das organizações de processamento de dados americanas praticam a análise estruturada de forma disciplinada [YOU 86]. Isso se deve principalmente à árdua carga de trabalho necessário para manter e desenvolver modelos estruturados. Segundo ele, esta "reversão" às metodologias estruturadas pode ser aplacada com o uso dos emergentes sistemas CASE. Atualmente, a maioria das ferramentas CASE visam a automatização do processo de desenvolvimento de grandes aplicações em conjunto com as técnicas estruturadas.

Barbara McNurlin [MCN 88] comenta que o processo de desenvolvimento de aplicações mudará significativamente em um futuro próximo graças aos sistemas de CASE. Ela utiliza como exemplo duas empresas americanas que conseguiram aumentar a qualidade e a velocidade do desenvolvimento com a utilização de tais sistemas. Uma das empresas analisadas, a Irving Trust, utilizava, em 1984, 70% dos recursos na manutenção de sistemas antes de iniciar o emprego de CASE. Para diminuir o custo de manutenção a empresa escolheu um CASE chamado Pacbase, após avaliar 23 produtos similares. (A Irving Trust preferiu o Pacbase porque, entre outras facilidades, ele possui a capacidade de gerar código fonte em COBOL). Após um ano de uso a Irving Trust, segundo Mcnurlin, obteve uma série de benefícios, entre os quais podem ser citados:

a) uma significante redução na manutenção de sistemas;
b) aumento da reutilização de componentes (definição de dados, especificações de programas e projetos);
c) automatização do desenvolvimento de sistemas;
d) padronização dos programas desenvolvidos.

No próximo capítulo os sistemas CASE serão analisados com maiores detalhes.

3 COMO SISTEMAS CASE AUMENTAM A PRODUTIVIDADE

CASE é uma tecnologia que utiliza ferramentas automatizadas de forma integrada, permitindo que o uso de métodos formais e semi-formais de desenvolvimento torne-se prático e econômico. Através de ferramentas de CASE os analistas e programadores obtêm uma melhor visualização dos requisitos, especificações e projetos do software, além de um acompanhamento automático nos refinamentos sofridos pelo software desde a fase de especificação de requisitos até a implementação, permitindo que a completeza e a consistência sejam verificadas automaticamente [BOE 87] [CHI 88a] [CHI 88b].

Através do uso de ferramentas CASE engenheiros de software podem aumentar a qualidade e a eficiência dos produtos, obtendo ganhos reais de produtividade na manutenção e desenvolvimento [CAS 85].

Um sistema CASE pode ser definido como um produto de software interativo cujo objetivo é auxiliar e automatizar algumas das tarefas dos engenheiros de software. Um sistema CASE deve possuir capacidades gráficas integradas a um dicionário de dados permitindo, assim, que os sub-produtos gerados pelo desenvolvimento de software sejam armazenados, recuperados e exibidos quando necessários. Também deve consistir automaticamente o trabalho dos engenheiros de software, mantendo a integridade dos produtos desenvolvidos [WHA 87].

É possível, portanto, caracterizar um sistema CASE como um macro-sistema provedor de assistência automática para as fases do ciclo de vida do software, desde a especificação até a implementação, fornecendo um suporte automatizado para o uso e execução de vários métodos, procedimentos e ferramentas de engenharia de software [CAS 85].

Para um sistema CASE prover tais facilidades ele deve possuir os seguintes elementos [CAS 85] [SHO 87] [BUR 87] [MAC 88]:

- um sistema de gerência de exibição de informações, para

apresentar uma interface amigável consistente com os diversos grupos de usuários;

- editores de diagramas e textos que tenham facilidades de desenho e edição interativa e simultânea, ou seja, possibilidade dos usuários poderem confeccionar gráficos e editar textos concomitantemente, de forma homogênea.

- um sistema de gerência de banco de dados que controle e armazene os sub-produtos gerados pelo desenvolvimento de sistemas em uma base de dados centralizada, compartilhada e com acesso "on-line";

- dicionários de dados que contenham, além das tradicionais informações de dicionários de dados - definições dos dados e referências cruzadas -, informações de decisões de projeto e análise, entre outras.

- analisadores de projeto com capacidade de validar a correção dos sub-produtos gerados pelas várias fases do ciclo de vida do desenvolvimento dos sistemas, ou seja, com funções análogas aos verificadores sintáticos em relação aos programas fontes;

- um sub-sistema de "ajuda" interativa, o qual permita aos usuários fazer acesso a qualquer método, procedimento ou ferramenta utilizada para documentação durante as sessões de trabalho;

- editores dirigidos por sintaxe, possibilitando aumento de produtividade e qualidade na fase de codificação dos programas, pois os programas são editados livres de erros de sintaxe e são formatados dentro do padrão especificado no ambiente [PRI 84] [PRI 85].

- é desejável, também, que os sistemas CASE tenham facilidades de seleção e integração com outras ferramentas, já construídas e presentes na instalação, como, por exemplo, linguagens de 4ª geração [MAR 82].

Como foi mostrado, um sistema CASE deve dispor de vários elementos integrados. Estes elementos podem ser vistos funcionalmente esquematizados na figura 3.1. As principais ferramentas que devem estar disponíveis são: editores CASE, dicionário de dados e verificadores de integridade.

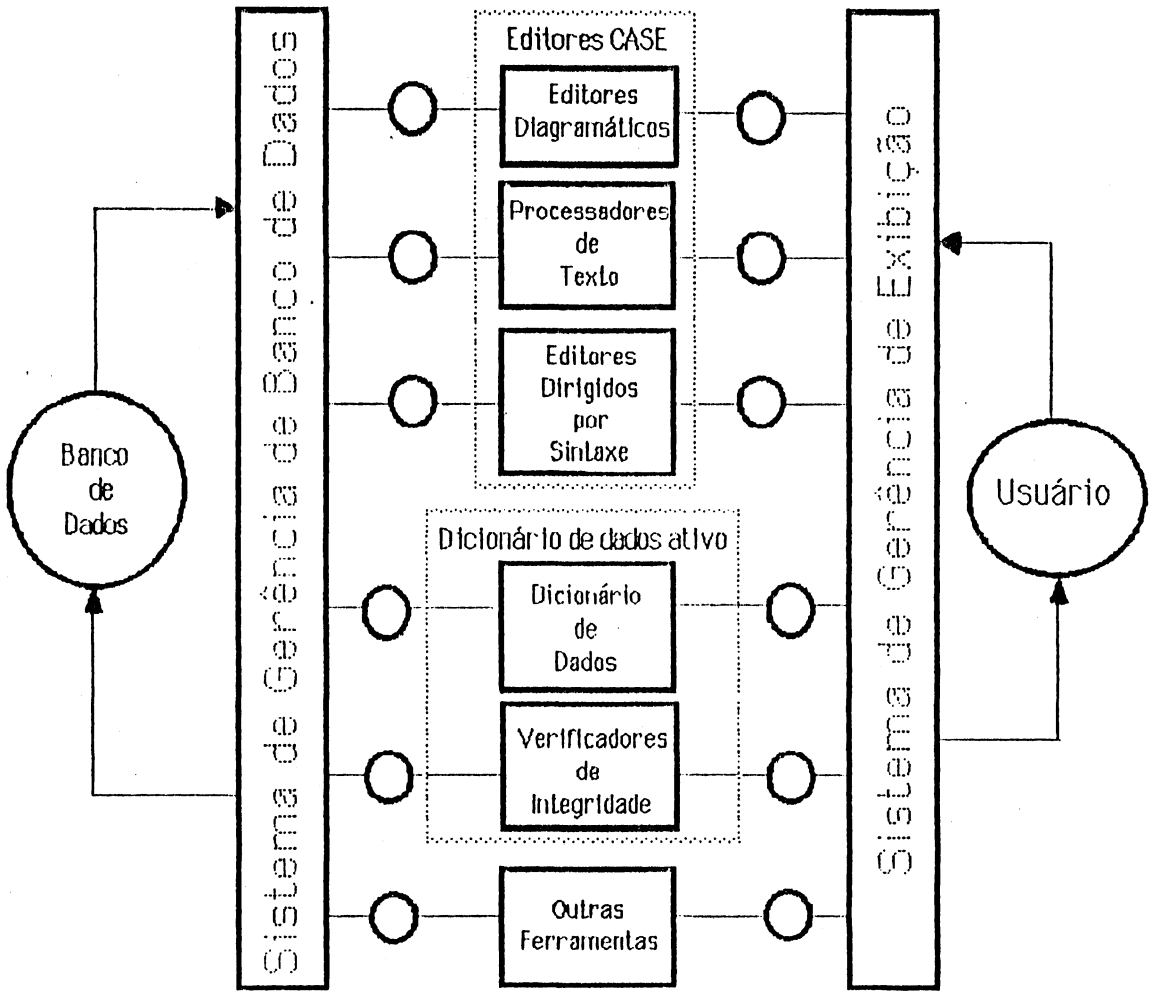


Figura 3.1 - Arquitetura funcional de um sistema CASE.

Os editores CASE [CAS 85] devem suportar editores de diagramas (sendo que textos podem ser editados juntos com os desenhos gráficos), processadores de texto e editores dirigidos por sintaxe. Com estas ferramentas integradas os editores CASE possibilitam aos engenheiros de software padronizar a criação e manutenção de quase todos os sub-produtos gerados pelo desenvolvimento de sistemas, desde os diagramas da especificação de requisitos até o código fonte dos programas.

Dicionário de dados permite que sejam manipulados todos os objetos de software, além das tradicionais informações sobre dados e estruturas de dados. É desejável que o dicionário de dados proveja as seguintes facilidades:

a) recuperação, inclusão e alteração dos vários tipos de objetos de software, como, por exemplo, descrições de fluxo de dados, processos, entidades, depósitos de dados, documentação, diagramas, organizações de formulários, relatórios e telas de exibição de informações e outros objetos criados pelos usuários;

b) gerência dos relacionamentos entre os diversos objetos de software, permitindo aos engenheiros de software identificar de forma rápida e eficiente todas as informações e documentos sobre os módulos reusáveis e redundantes;

c) responder de forma rápida e eficiente às questões dos usuários sobre as informações armazenadas no banco de dados;

d) navegação nas estruturas hierárquicas dos processos que formam os diversos níveis de decomposição da análise e projeto de software. Isto permite que sejam feitas consistências nos vários processos de refinamento e condensação a que o software é submetido durante o desenvolvimento do produto, possibilitando a obtenção das informações de relacionamento e hierarquia dos vários objetos de software usados no decorrer do desenvolvimento do produto. Assim o dicionário de dados pode servir de suporte para as ferramentas que mantêm a integridade dos diversos diagramas, que representam os vários níveis do desenvolviemnto de um determinado produto de software.

Os verificadores de integridade garantem a consistência dos diversos níveis de decomposições e transformações a que um produto de software é submetido. Através da aplicação das regras de integridade, implícitas e/ou explícitas, das diversas metodologias de desenvolvimento suportadas pelo sistema CASE. Os verificadores de integridade permitem

que a completeza e correção dos diversos sub-produtos de software sejam mantidas nos diferentes níveis do desenvolvimento e manutenção.

Um sistema CASE possuidor dos elementos citados, onde estes elementos estejam integrados de forma homogênea, pode contribuir de várias formas para aumentar a produtividade dos engenheiros de software. A seguir são mostrados alguns benefícios que um sistema CASE pode proporcionar no aumento de produtividade:

- através dos editores CASE a necessidade de aprendizado de vários editores é reduzida, minimizando o tempo gasto no processo de adaptação da interação homem-máquina, além de reduzir o tempo e esforço dispendido nas tarefas de criação e modificação dos sub-produtos de software [CAS 85];

- através das facilidades providas pelo dicionário de dados a pesquisa e acesso a componentes reusáveis pode reduzir o tempo e esforço de desenvolvimento de novos produtos de software, além de facilitar a prototipação. Os mecanismos de navegação pelos vários níveis hierárquicos dos sub-produtos de software também são fundamentais para que possam ser feitas, de forma automatizada, as consistências de projeto[CAS 85];

- as atividades de consistência que garantam a completeza e correção dos sub-produtos de software sempre foram consideradas tarefas árduas e cansativas pelos engenheiros de software. Através dos verificadores automáticos de integridade estas tarefas são realizadas durante o desenvolvimento dos produtos de software, liberando tempo e energia das equipes de desenvolvimento e manutenção para atividades mais produtivas. Com os verificadores de integridade os diagramas, programas e outros sub-produtos de software são consistidos interativamente e continuamente, evitando-se a efetivação de relacionamentos ilegais entre objetos; obrigando que todos objetos usados sejam definidos; atualizando automaticamente os objetos no dicionário de dados; e controlando as decomposições e transformações dos sub-produtos de software [CAS 85] [BUR 87];

- documentação é considerada um dos grandes problemas de desenvolvimento e manutenção de software. É comum encontrar sistemas operando com documentação inacabada ou desatualizada, sendo que em muitos casos ela é feita em fases posteriores ao término do desenvolvimento. Por exemplo, quando é necessário estender o sistema para atingir novos requisitos ou corrigir erros decorrentes das fases de desenvolvimento, os engenheiros de software deparam-se com programas cuja documentação é o próprio código fonte. Através de um sistema CASE, a documentação pode ser gerada como um sub-produto do processo de produção de software, ao invés de uma tarefa consumidora de tempo e esforço extra [REY 87] [BUR 87];

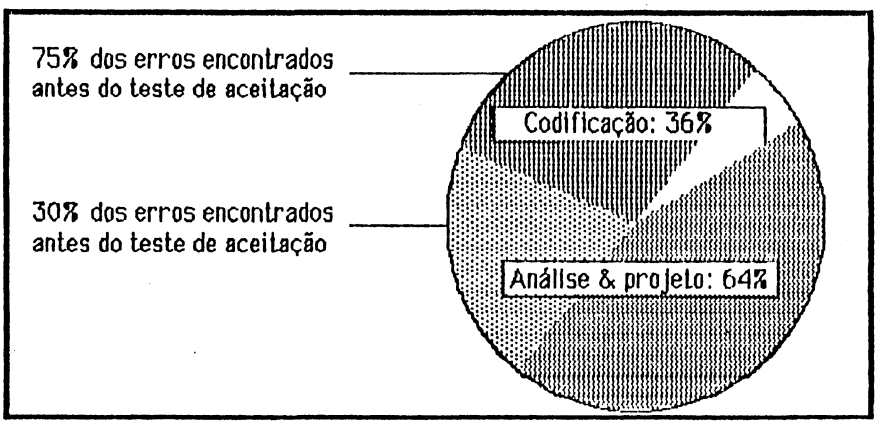


Figura 3.2 - Erros de software durante as fases de especificação e implementação [SUY 87].

- os erros decorrentes das fases de especificação e projeto são os mais difíceis e mais caros de serem corrigidos; logo quanto menor for a quantidade de inconsistências nestas fases menos tempo será perdido nas fases posteriores. Um estudo feito em 1980 pela TRW [SUY 87] mostrou que 64% dos erros de software são derivados das fases de análise e projeto, mas somente 30% destes erros foram detectados antes da entrega do software. Por outro lado, 75% dos erros de codificação, ou seja, 36% do total de erros, foram detectados antes da entrega (figura 3.2). Contudo, o custo de corrigir erros na fase de especificação é muito menor do que o de corrigí-los nas fases posteriores, devido à propagação para as demais fases. Na figura 3.3 é mostrado que o custo de se corrigir erros na fase de operação pode ser até 100 vezes maior do que o de corrigí-los durante a

fase de análise [SUY 87]. Tudo isto demostra a importância das fases iniciais do ciclo de vida do software e é justamente nestas fases que as ferramentas CASE têm mostrado serem mais efetivas. Alguns estudos mostram [CHI 88b] que o uso destas ferramentas proporciona um ganho de 30-40% (figura 3.4) nas fases de análise e projeto do sistema, aumentando também a produtividade nas outras fases.

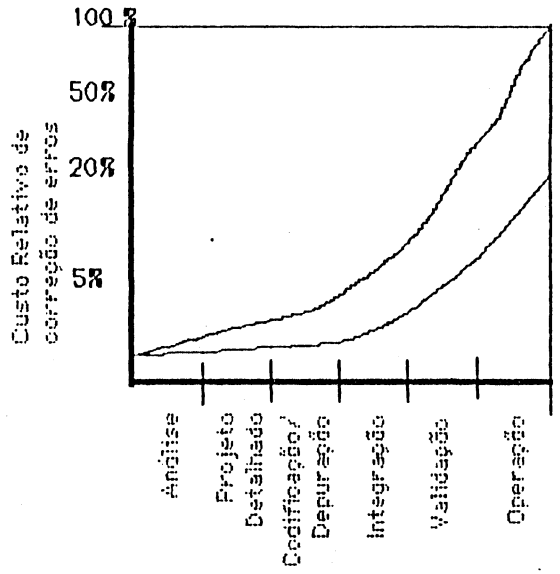


Figura 3.3 - Custo de correção de erros [SUY 87] .

Atualmente, já estão em comercialização várias ferramentas CASE. A maioria destas ferramentas são especializadas nas fases de análise, projeto e modelagem de dados. Também estão sendo comercializadas ferramentas de auxílio à construção de interfaces gráficas. Estas últimas também podem ser consideradas como ferramenta CASE, pois elas suportam um ambiente interativo de construção de interfaces, como, também, algumas delas chegam a gerar alguma espécie de código. Em [FIS 88], Fisher enumera 46 ferramentas CASE disponíveis no mercado americano, mostrando suas principais características. Muitas das ferramentas comentadas são compostas por editores diagramáticos especializados particularizados em alguns métodos diagramáticos de engenharia de software, integrados com dicionários de dados passivos. No Brasil, embora em menor número, também já se encontram disponíveis comercialmente algumas ferramentas CASE, como, por exemplo, o

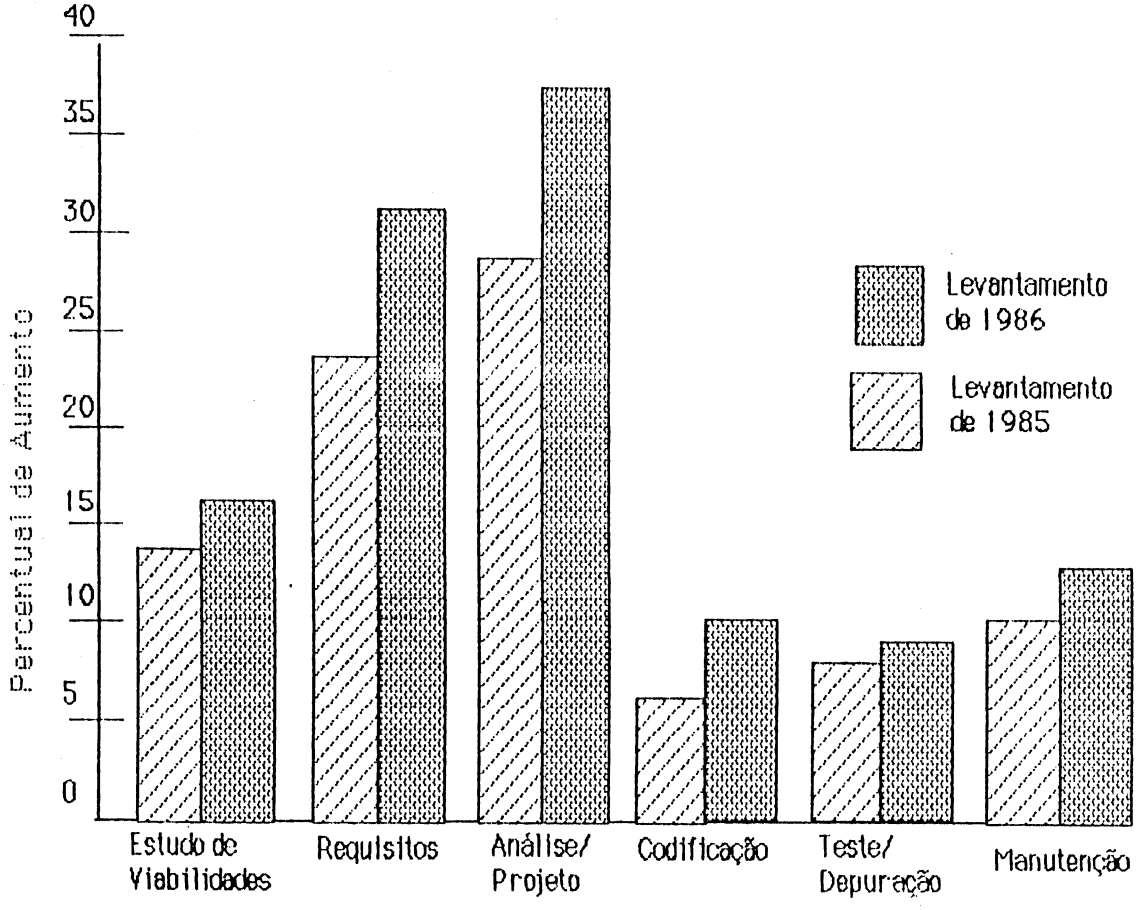


Figura 3.4 - Aumento de produtividade nas fases do ciclo de vida de sistemas proporcionado pelas ferramentas CASE [CHI 88b].

3.1 Diagramas estruturados e editores diagramáticos

Na década passada vários métodos formais e semi-formais foram desenvolvidos, a maioria baseados em diagramas, no intuito de tornar o processo de desenvolvimento de sistemas menos anárquico e mais formal.

Diagramas são ferramentas essenciais na construção de programas e sistemas complexos. Abaixo são citadas e comentadas algumas funções desempenhadas pelas técnicas estruturadas baseadas em notações diagramáticas [MAR 85a] [MAR 85b][THE 87]:

- Diagramas facilitam a clareza de raciocínio.

Através do uso de técnicas diagramáticas estruturadas o engenheiro

- Diagramas facilitam a clareza de raciocínio.

Através do uso de técnicas diagramáticas estruturadas o engenheiro de software tem a sua disposição uma poderosa ferramenta de auxílio ao seu raciocínio, pois ele poderá ver e expressar seu pensamento com maior clareza. A utilização de notações diagramáticas adequadas pode aumentar a rapidez e a qualidade dos trabalhos realizados pelos engenheiros de software.

-Diagramas auxiliam a comunicação entre os membros das equipes de desenvolvimento.

Sistemas e programas complexos são, em geral, desenvolvidos por um grupo de pessoas, neste caso diagramas são ferramentas essenciais de comunicação. O uso de técnicas diagramáticas é bastante útil no sentido de facilitar aos engenheiros de software a troca de idéias de forma padronizada e, também, auxiliar o acoplamento de componentes desenvolvidos separadamente.

-Diagramas tornam a tarefa de manutenção menos árdua.

Diagramas bem estruturados podem ser de grande ajuda aos engenheiros de software nas tarefas de manutenção. Os diagramas os auxiliam a descobrir como os programas foram desenvolvidos e projetados, permitindo-lhes, assim, entender e prever melhor os efeitos colaterais ocasionados devido às atualizações realizadas.

-Diagramas facilitam a depuração.

Diagramas bem estruturados são valiosas ferramentas de ajuda à depuração de programas, pois auxiliam os encarregados desta tarefa entender melhor os requisitos e o projeto do programa. Portanto, os diagramas facilitam a compreensão dos objetivos e funcionamento dos programas, desta forma a área a ser depurada é mais facilmente delimitada.

- Diagramas facilitam a documentação.

Diagramas bem estruturados são muito importantes como ferramentas de documentação. Eles podem ser utilizados para definir as

especificações e para representar o projeto. Eles permitem que as descrições arquiteturais e detalhadas dos programas sejam feitas de maneira fácil e consistente.

Apesar destas vantagens diagramas são extremamente difíceis de criar e manter manualmente. Os editores diagramáticos foram justamente desenvolvidos para suportar interfaces gráficas com as quais analistas e programadores interagem desenhando diagramas, descrevendo e definindo funções e dados, relacionando e identificando componentes do sistema. Através dos editores diagramáticos a documentação é gerada graficamente e consiste na "verdade" sobre o sistema. As equipes de desenvolvimento são treinadas de modo efetivo no uso dos métodos diagramáticos e as tarefas de verificação de inter-conexões entre módulos são realizadas sem aborrecê-las ou sobrecarregá-las, pois são feitas durante o processo de edição dos diagramas, liberando tempo e energia para outras atividades de desenvolvimento.

Através de editores diagramáticos, diagramas podem ser armazenados e recuperados facilmente e módulos de um sistema podem ser utilizados no desenvolvimento de outros, reforçando a prototipação e reusabilidade de componentes. Por exemplo, em [LUQ 88] Luqi e Monhammad Ketabchi apresentam um sistema de prototipação auxiliado por computador, onde a especificação dos sistemas-alvo é feita através de diagramas de fluxo de dados estendidos (PSDL - Prototype-System Description Language), armazenados de forma normalizada para que um sub-sistema de gerência organize e recupere os componentes (reusáveis) da base de software.

3.2 Alguns problemas dos editores diagramáticos atuais

Apesar das vantagens que os editores diagramáticos apresentam no aumento da produtividade eles apresentam várias limitações que reduzem muito seus benefícios. Entre algumas desvantagens podem ser citadas as mencionadas em [MAR 88] e [MEL 88a], mostradas a seguir:

a) Os editores diagramáticos são confeccionados para suportar um conjunto pré-definido de técnicas diagramáticas; logo podem não suportar técnicas que o usuário deseje e necessite para determinadas aplicações. Por exemplo, o Excelarator da Index Technology Corporation possui vários editores especializados em várias técnicas estruturadas de projeto/análise, mas se os usuários desta ferramenta CASE necessitarem ou desejarem utilizar Redes de Petri como método de análise estruturada [HEU 87] não terão o suporte necessário.

b) A maioria dos editores diagramáticos atuais não possui facilidades para formatação automática, e quando possuem não permitem que o usuário faça os ajustes estéticos desejados.

c) A primeira geração de editores diagramáticos somente enfatiza a interação com o usuário, e em sua maioria são ineficientes na produção de documentação impressa.

d) Os editores diagramáticos de primeira geração são difíceis de integrar com outros ambientes, pois são construídos de maneira verticalizada.

Estes problemas nos levam a considerar uma nova geração de editores diagramáticos que proporcionem maiores acréscimos de produtividade pela superação das deficiências citadas. Uma maneira adequada de abordar tais problemas é através de um gerador de editores diagramáticos.

3.3 Características fundamentais de um gerador de editores diagramáticos

A principal função de um gerador de editores diagramáticos, ou "meta-system" [SOR 88], é gerar automaticamente a maior parte de um ambiente específico de desenvolvimento de software, onde a palavra "específico" pode ser entendida como uma ferramenta CASE especializada em uma técnica ou método particular.

Um gerador de editores diagramáticos caracteriza-se por possuir um administrador de CASE [MAR 88], que exerce funções parecidas com o administrador de banco de dados em um SGBD (Sistema de Gerência de Banco de Dados). É ele o responsável pela criação e manutenção de novas ferramenta CASE no ambiente. Por intermédio dele são definidas as convenções e as regras de integridade dos tipos de técnicas diagramáticas que o ambiente suporta, como também as verificações de consistência e integridade que devem ser realizadas nos refinamentos e transformações que um diagrama pode sofrer. Desta forma, é eliminada a deficiência dos editores atuais de se adaptarem a novas técnicas de desenvolvimento de software.

Além disto, um gerador de editores diagramáticos deve possuir facilidades de *soft-copy* [MAR 88], ou seja, capacidade de gerar documentos formatados e estilizados, que reproduzam a qualidade e a hierarquia/decomposição dos diagramas editados. Ele deve, também, prover facilidades de desenho automático, mas intercambiáveis com o desenho interativo, para que o usuário não fique preso aos padrões de desenho oferecidos pela ferramenta, permitindo que sejam feitos os ajustes estéticos desejados.

Alguns geradores de editores diagramáticos que visam oferecer algumas ou todas as facilidades citadas acima estão em desenvolvimento [SOR 88]. No próximo capítulo será mostrado o projeto de implementação de um Editor de Diagramas Generalizado (EDG), construído na UFRGS com o objetivo de alcançar alguns dos atributos de um gerador de editores diagramáticos.

4 EDG: UM EDITOR DIAGRAMATICO GENERALIZADO

Muitos métodos formais de desenvolvimento de software são baseados em notações diagramáticas. A maioria destas notações podem ser abstraídas como grafos. As diferenças entre estes grafos são encontradas em três itens: os tipos de nodos, ou seja, sua forma gráfica; os tipos de arcos, que também possuem formas gráficas, que variam para métodos distintos; e as regras de integridade.

Evidentemente as técnicas diagramáticas possuem outras características diferenciadoras além dos três itens citados acima, principalmente quanto às particularidades semânticas de cada técnica, as quais podem divergir radicalmente de uma técnica para outra. Através de um mecanismo que suporte a especificação daqueles itens uma grande família de técnicas pode ser abordada de maneira uniforme, para efeitos de automatização.

Muitas técnicas diagramáticas, principalmente as surgidas na década de 70, estão passando na década de 80 por um processo de automatização. Várias ferramentas de CASE têm sido propostas e construídas para suportar um conjunto pré-determinado de técnicas. Contudo, a possibilidade de construir uma ferramenta com a capacidade de gerar editores diagramáticos, particularizados em técnicas específicas, parece mais conveniente que construir um editor exclusivo para um pequeno grupo de técnicas.

Seguindo esta direção, foi iniciada, em fevereiro de 1988, a construção de um editor generalizado de diagramas, denominado **EDG** (Editor **D**iagramático **G**eneralizado). O **EDG** é uma ferramenta construída com a capacidade de gerar editores diagramáticos especializados. O **EDG** [MEL 88a] [MEL 88b] [MEL 88c][MEL 88d] [MEL 89] provê mecanismos pelos quais diagramas possíveis de serem abstraídos como grafos possam ser descritos. Como base nesta descrição o **EDG** provê facilidades para a edição de diagramas na técnica especificada.

A figura 4.1 mostra o fluxo de funcionamento do **EDG**. Como pode ser visto, ele é dividido funcionalmente em dois módulos principais: o **M**eta **E**ditor **D**iagramático (**MED**) e o **E**ditor **D**iagramático Específico (**EDE**).

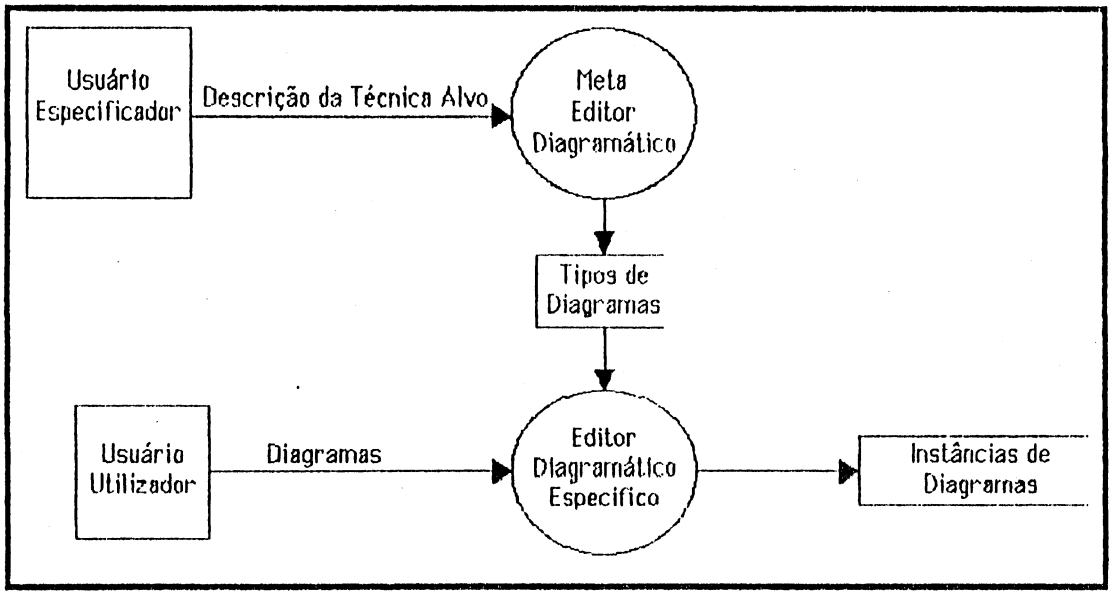


Figura 4.1 - Diagrama de fluxo de dados do **EDG**.

Através do **MED** o usuário especificador ou administrador de CASE (**AC**) [MAR 88] descreve os tipos de nodos, arcos e regras de ligação de uma técnica particular.

Utilizando o **EDE**, o usuário da técnica alvo interage com o **EDG** criando e mantendo diagramas. O **EDE** somente suporta a edição de diagramas de técnicas especificadas no **MED**. A interface do **EDE** é uniforme para todas as técnicas suportadas, divergindo, é claro, somente quanto aos tipos de diagramas que podem ser editados.

Através de sua interface, o **EDG** provê uma série de facilidades gráficas, tanto para especificação dos tipos de diagramas como para edição dos diagramas propriamente ditos. As características desta interface são examinadas no capítulo 5. As seções 4.1 e 4.2 descrevem os módulos **MED** e **EDE**.

4.1 O MED: Meta Editor Diagramático

Por intermédio do **MED** são realizadas as especificações dos tipos de diagramas suportados no **EDG**, ou seja, as técnicas diagramáticas sustentadas pelo **EDG** são definidas pelo usuário especificador ou administrador de CASE (**AC**) através do **MED**.

Através do **EDG** somente podem ser utilizadas técnicas baseadas em grafos. Para definir tais técnicas no **EDG** o **MED** suporta cinco entidades: os tipos de nodos, arcos, decorações, pontas e regras de ligações. Tais entidades possibilitam ao **AC** definir as principais características diferenciadoras das técnicas desejadas.

Os tipos de nodos são modelados através da composição de objetos gráficos primitivos disponíveis no **MED**. Um tipo de nodo pode ser formado por vários objetos de tipos diferentes. Através de instanciamentos, deformações e movimentações dos objetos formadores do nodo, o **AC** forma o estrutura deste.

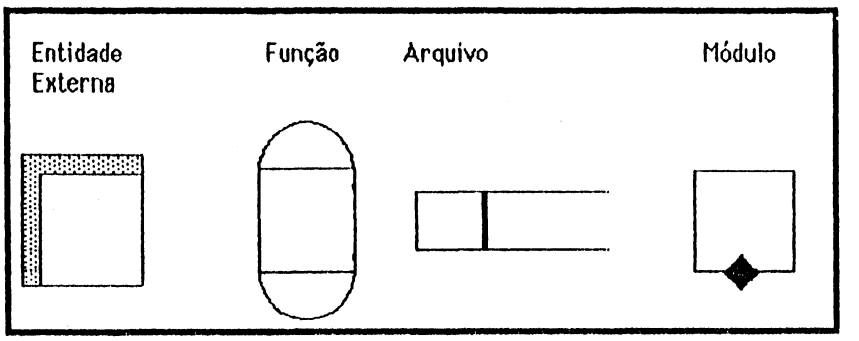


Figura 4.2 - Exemplo de tipos de nodos. No exemplo, cada nodo é composto por mais de um objeto gráfico.

A figura 4.2 mostra alguns exemplos de tipos de nodos. Cada tipo foi desenhado através da composição de objetos gráficos. A posição relativa e a ordem de criação dos objetos são fatores determinantes na formação do leiaute do nodo. O usuário especifica, também, quais objetos são portas de ligação do nodo, ou seja, através de quais objetos o nodo pode conectar-se com outros nodos.

Como pode ser visto na figura 4.2, o nodo do tipo "módulo" é formado por dois objetos gráficos: um retângulo e um losango. O losango é a única porta de ligação do nodo, ou seja, todas as conexões a serem realizadas no **EDE** somente poderão ser efetivadas neste objeto. O nodo do tipo "arquivo" é formado por três objetos gráficos: um retângulo e dois segmentos de reta. O retângulo é a porta de ligação do nodo "arquivo".

O **MED** suporta nove tipos de objetos gráficos primitivos: polígonos, polígonos regulares, retângulos, retângulos chanfrados, elipses, segmentos de retas, segmentos curvos, textos e meta-texto. Cada objeto criado no **MED** possui atributos de visualização, como cor de fundo, cor e largura da(s) linha(s) de contorno, dimensões, além de atributos de identificação e número possível de conexões mínimas e máximas. O nodo do tipo "entidade externa", exemplificado na figura 4.2, é formado por dois retângulos, sendo que um deles foi especificado como padrão de preenchimento cinza. Na atual versão implementada o **MED** dispõe de 24 padrões distintos. Estes padrões são obtidas através de facilidades embutidas no sistema operacional do Macintosh, mais precisamente na "toolbox" [INS 85].

Os textos podem ser editados com vários tipos de impressão, estilos e tamanhos diferenciados. O **EDG** provê uma série de facilidades de edição de textos, como, por exemplo, seleção de blocos de textos, procura e troca de "strings" e ajustes. Os textos definidos no **MED** não podem ser alterados ou editados no **EDE**.

Os meta-textos são objetos através dos quais poderão ser editados textos formais. Através dos meta-textos é especificado o local e a regra de produção associada. A regra de produção deverá ser especificada através dos mecanismos providos pelo Editor Dirigido por **S**intaxe (**EDS**), ou seja, através da **L**inguagem **d**e Especificação (**LDE**). Na atual versão do **MED** os meta-textos somente informam os locais onde o usuário poderá editar textos vinculados aos nodos e decorações. No momento, não existe uma associação à gramática do texto, o que deverá ser feito em uma futura versão do **EDG**. Além dos meta-textos, todos os objetos que compõem o nodo podem ter textos associados.

Decorações são figuras formadas pelos objetos primitivos do **MED**, que podem estar associadas a certos tipos de arcos. Sua função é fornecer sentido semântico às conexões em determinadas técnicas. As decorações não podem existir sem a presença do arco do qual são dependentes. Os tipos de decorações são criados no **MED** da mesma forma que os tipos de nodos. Um exemplo típico de decorações é encontrado na técnica de projeto estruturado [STE 85], onde as váriáveis de dados e controle são representadas por figuras diferenciadoras. A figura 4.3 mostra um exemplo destes dois tipos de decorações.

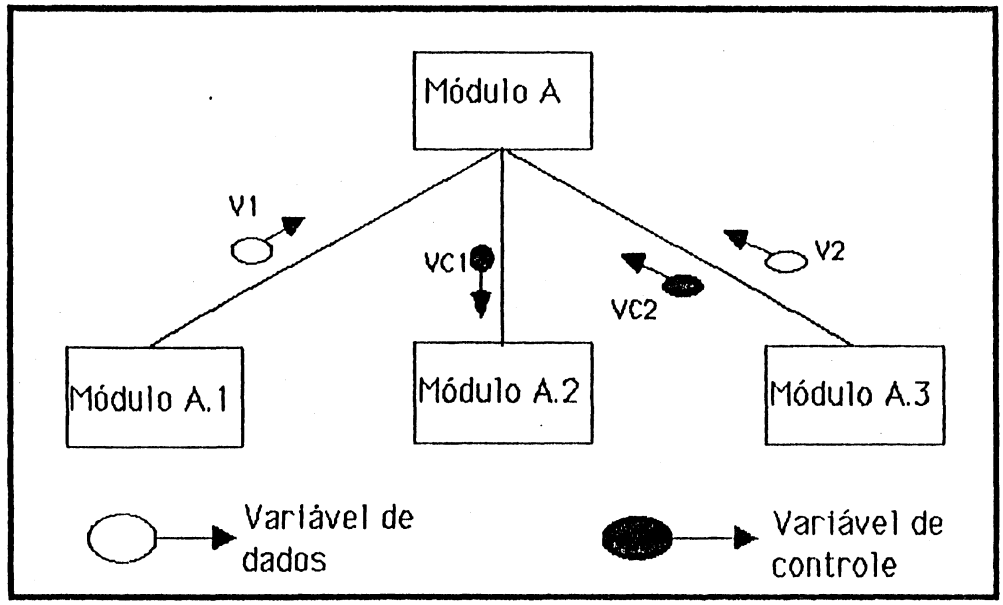


Figura 4.3 - Exemplos de decorações. A figura mostra dois tipos de decorações, as quais são utilizadas na técnica de projeto estruturado, a variável de dados e de controle [STE 85].

Diferentes tipos de técnicas diagramáticas podem usar formas distintas de pontas de arcos. O **AC** pode desenhar as pontas utilizadas pela técnica alvo. Os tipos de pontas são desenhados da mesma forma que os tipos de nodos, contudo o **MED** não permite que as pontas sejam modeladas com composição de textos, meta-textos, retângulos e retângulos chanfrados (isto porque tais objetos não são rotacionados no **EDE**). As pontas são construídas supondo uma ligação horizontal, e são rotacionadas no **EDE** de acordo com o ângulo de inclinação do arco a qual estão vinculadas. A figura 4.4 apresenta alguns exemplos de tipos de pontas

modelados no **MED**, e a figura 4.5 demonstra como as pontas são exibidas no **EDE**.

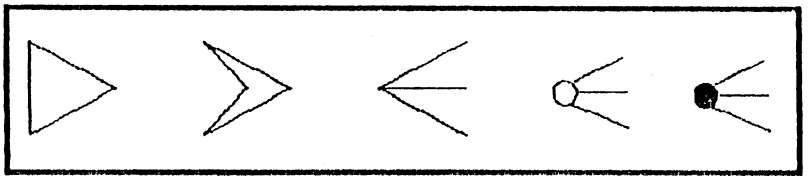


Figura 4.4 - Exemplos de alguns tipos de pontas desenhadas no **MED**.

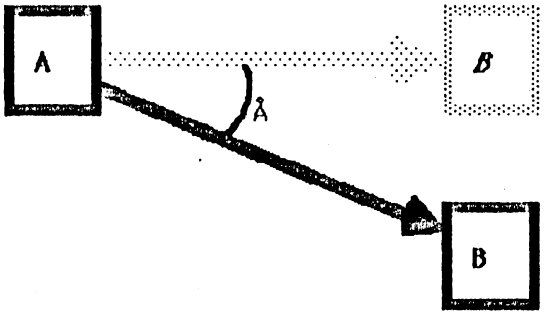


Figura 4.5 - Exemplo de ligação de um arco no **EDE**. A ponta associada é rotacionada de acordo com o ângulo formado entre os dois nodos alvos da conexão.

A maioria das técnicas possuem restrições de conexão entre nodos, isto é, certos tipos de nodos só podem ser conectados a determinados tipos de nodos através de específicos tipos de arcos. Alguns nodos ainda possuem portas de ligação, por onde devem ser realizadas as conexões. As regras de ligações são descritas no **MED** através de uma tabela de ligações possíveis. Através desta tabela o **AC** informa quais nodos podem ser conectados através de quais arcos. A figura 4.6 mostra um exemplo desta tabela, e a figura 5.14 mostra como a tabela é interativamente construída.

Tipo Nodo	Objeto	Tipo Nodo	Objeto	Tipo Arco
Função	Meio	Entidade Externa		Entrada
Função	Meio	Depósito de Dados	Cabeça	Saída
Função	Meio	Função	Meio	Entrada/Saída
...	...	...	...	...

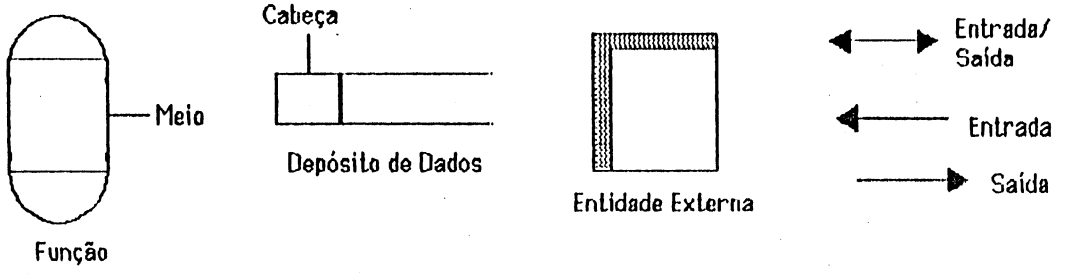


Figura 4.6 - Exemplo da tabela de restrições de ligações. O exemplo mostra a definição das ligações possíveis na técnica de Diagrama de Fluxo de Dados. Todos os tipos de nodos e arcos devem estar definidos antes da especificação das regras de ligações. No exemplo está sendo especificado, entre outras descrições, que o nodo do tipo "função" , através do objeto "meio", pode ser ligado ao nodo do tipo "depósito de dados", através do objeto "cabeça", pelo arco do tipo "saída".

4.2 O EDE: Editor Diagramático Específico

O usuário do método alvo interage com o **EDG** através do **EDE**, criando e mantendo diagramas. O **EDE** somente permite que sejam manipulados diagramas de métodos especificados no **MED**.

A entidade manipulada pelo **EDE** é o documento. Um documento é representado em forma de árvore onde cada nó é um diagrama. Ou seja, um documento pode ter vários níveis de diagramas.

Em cada diagrama o usuário pode reproduzir vários nodos, decorações e arcos de um tipo previamente especificado no **MED**. Os nodos podem ser refinados em diagramas de níveis inferiores. Todos os diagramas de um documento são do mesmo tipo, isto é, são ocorrências de uma mesma técnica especificada no **MED**.

O usuário desenha os diagramas interativamente, criando, movendo, excluindo, conectando, condensando e refinando diagramas. As verificações de consistência são realizadas automaticamente, sempre que o usuário realizar uma alteração de conteúdo do diagrama.

Cada diagrama é editado em um janela particular, sendo que cada diagrama pode possuir várias páginas de desenho.

Quando o usuário solicita a ligação de dois nodos com um tipo particular de arco, a conexão somente é realizada se estiver definida como uma ligação válida. Caso seja válida a ligação, o **EDE** realiza a conexão entre os objetos desejados, arrumando as pontas dos arcos de acordo com a inclinação da ligação. A partir daí todas as operações realizadas nos nodos ligados serão refletidas nas ligações.

O **EDE** possui facilidades de "soft-copy" [MAR 88]. Os diagramas são impressos como foram desenhados, sendo que os vários níveis de refinamento são mostrados de acordo com o desenho. Caso um diagrama possua mais que uma página de impressão o usuário poderá solicitar a redução deste para somente uma página, ou imprimí-lo em várias páginas.

O **EDE** também dispõe de um editor de textos com todas as facilidades de navegação, procura e troca de caracteres. A edição de textos pode ser realizada dentro do limite do objeto (menor retângulo que contém todos os pontos do objeto) ou livre no diagrama. No primeiro caso, o texto não pode ultrapassar os limites, logo o texto é rolado dentro do objeto quando necessário. No segundo caso, os textos são editados livremente dentro do diagrama. Os textos podem ser estilizados, alinhados e ter os fontes trocados. Aqueles que estiverem associados a nodos, arcos e decorações acompanharão as translações que estes sofrerem.

À medida que o diagrama é editado, o usuário pode solicitar o refinamento de alguns nodos. O **EDE** cria uma nova janela para o edição do diagrama de refinamento do nodo desejado. O novo diagrama conterá todas as interfaces do nodo pai no diagrama superior. As operações de exclusão

de ligações no nodo pai afetam os diagramas filhos, e também, as alterações na interface do diagrama filho alteram as conexões no diagrama pai. Desta forma a consistência sintática do documento é mantida. A figura 5.6.2 mostra um exemplo de um documento sendo refinado e condensado no **EDE**.

Uma importante característica dos nodos instanciados no **EDE** é que eles não são *carimbos*. Isto é, as dimensões de nodo (ou de uma decoração) não são fixas. As dimensões definidas no **MED** podem ser alteradas no **EDE**, permitindo maior flexibilidade de desenho. Os objetos formadores dos nodos e das decorações também podem ser excluídos, movidos ou duplicados no **EDE**, desde que especificados com esta capacidade. Através desta facilidade técnicas como o projeto estruturado [STE 85] podem ser suportadas pelo **EDE**.

A figura 4.7 mostra um exemplo de um diagrama onde um nodo do tipo "módulo" (especificado na figura 4.2) teve sua anatomia modificada no **EDE**, pela duplicação de um de seus objetos. Tal metamorfose foi possível devido a uma prévia descrição no **MED**. Os nodos que não forem especificados com esta capacidade nunca terão suas formas modificadas, a exceção das dimensões de todos os objetos que os compõem.

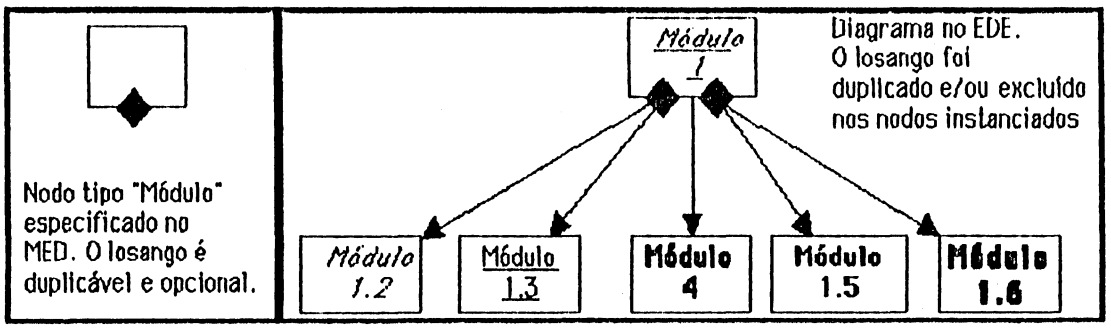


Figura 4.7 - Exemplo de um diagrama no **EDE**. O nodo "módulo 1" do tipo "módulo" teve seu leiaute modificado, no **EDE**, pela duplicação do losango, e os demais nodos do diagrama tiveram tal objeto excluído.

5 DESCRIÇÃO DA INTERFACE DO EDG

Este capítulo descreve as linhas gerais da interface do **EDG** com os seus usuários. Na seção 5.1 são mostradas as características gerais da interface do **EDG**, válidas tanto para a interface do **MED** quanto para o **EDE**. Na seção 5.2 são comentadas as técnicas interativas de construção de objetos gráficos. A seção 5.3 descreve as formas de seleção de entidades gráficas (textos, ícones e objetos gráficos) utilizadas. Na seção 5.4 uma facilidade para a alteração das dimensões dos objetos gráficos, denominada pegadores, é apresentada. Na seção 5.5 e 5.6 os cardápios de funções e as principais janelas do **MED** e do **EDE** são respectivamente comentados com maiores detalhes.

5.1 Linhas gerais

O **EDG** é projetado para executar sobre um ambiente gráfico (WINDOWS, GEM, MACINTOSH) utilizando as facilidades providas por estes ambientes. Como o protótipo do **EDG** foi implementado em um equipamento tipo Macintosh, ele possui as características das aplicações que executam neste ambiente.

O Macintosh fornece uma série de facilidades para construção de programas gráficos, como exemplo: gerência de janelas; gerência de cardápios; facilidades de edição de textos; facilidades de impressão gráfica; e "driver" para "mouse".

No restante desta seção serão mostradas as principais características da interface do **EDG** no ambiente Macintosh, sendo que maiores informações sobre os dipositivos de entrada e saída, como também sobre a gerência de cardápios e janelas, entre outras, podem ser encontradas em [INS 85].

Os usuários interagem com o **EDG** através dos seguintes dispositivos:

- "mouse";
- teclado;
- impressora e
- vídeo.

O "mouse" é utilizado como dispositivo de apontamento e desenho. A maioria das ações dos usuários são ativadas e efetuadas com a ajuda deste dipositivo; como exemplo, a seleção de itens do cardápio ou a edição de diagramas.

O teclado é utilizado, principalmente, para entrada de textos. Contudo, algumas teclas possuem funções especiais, sendo utilizadas para ativação de ítens dos cardápios ou como parâmetros para a mudança de comportamento das ações do mouse.

Através de algumas teclas especiais do teclado o **EDG** pode interpretar de forma diferente as ações do mouse. Portanto, uma mesma operação com o mouse pode gerar comportamento distinto quando acompanhada da ativação de alguma tecla especial. Por exemplo, a tecla "shift" é utilizada para informar seleção de objetos por extensão (seção 5.3.2).

O teclado também é utilizado para seleção de ítens dos cardápios, sendo neste caso mais rápida sua utilização que a seleção feita através do mouse. Esta facilidade permite uma melhor adaptação dos usuários ao **EDG**, pois os menos experientes, que não conheçem as principais opções do cardápio, utilizam o mouse para escolher os ítens desejados, enquanto os já iniciados podem utilizar as teclas especiais para uma rápida seleção.

O **EDG** imprime as definições dos tipos de diagramas e os diagramas com o mesmo padrão gráfico com que estes são desenhados. Para tal, o **EDG** utiliza os "drivers" de impressora providos pelo equipamento.

O **EDG** utiliza uma tela gráfica de alta resolução na interação com seus usuários. A tela é dividida em duas regiões: uma região é usada para exibição das opções do cardápio (denominada "menu bar") do tipo "pull-down" e a outra (denominada "desk-top") é usada para a exibição das janelas. Estas duas regiões nunca se sobrepõem. A figura 5.2 mostra um exemplo de uma tela como cinco janelas na "desk-top" e a região das opções do cardápio.

O cardápio possui um conjunto de opções, sendo que cada opção pode possuir vários comandos, ou seja, o cardápio é implementado com dois níveis hierárquicos. Os comandos são ativados pela seleção feita através do mouse ou, em alguns casos, através de uma tecla especial. Alguns comandos necessitam de paramêtros do usuário para sua execução. Para obtenção destes paramêtros, o **EDG** ativa uma caixa de diálogo, por intermédio da qual o usuário fornece as informações necessárias para a execução do mesmo.

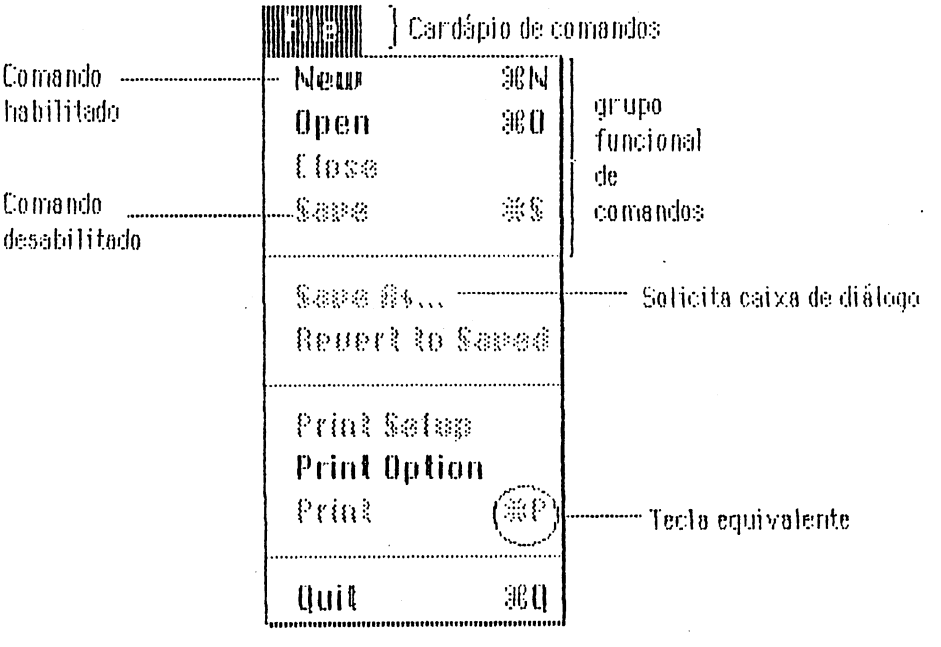


Figura 5.1 - Exemplo de uma opção do cardápio do **EDE**.

A figura 5.1 mostra um exemplo de uma opção do cardápio do **EDE**. A opção é dividida em grupos funcionais. Cada grupo contém os comandos que possuem comportamentos semelhantes ou agem sobre os mesmos elementos. A figura mostra alguns comandos habilitados e outros não. Os

comandos não habilitados são mostrados de forma atenuada e não podem ser selecionados pelo usuário. Alguns comandos podem ser ativados por teclas funcionais específicas, como é o caso do comando **Print**, o qual pode ser ativado pela tecla de função em conjunto com a letra **P**. O exemplo também mostra um comando que necessita de parâmetros extras para sua execução (**Save As**). Neste caso o nome do comando é seguido de "...".

É através das janelas que os usuários desenham os tipos de diagramas (tipos de nodos, pontas, decorações e arco) e os diagramas . Toda a interação, à exceção das caixas de diálogo, é feita através das janelas. Uma janela pode estar aberta ou fechada. Embora possam ser exibidas várias janelas abertas ao mesmo tempo, que podem estar sobrepostas ou não, somente uma janela pode estar ativa de cada vez. A janela ativa é a janela alvo das operações do **EDG**. Qualquer janela aberta pode tornar-se ativa, bastando para isto uma seleção do usuário. A janela ativa está sempre em posição de destaque em relação às outras janelas abertas, nunca estando sobreposta por outra janela aberta.

O usuário pode mover, redimensionar o tamanho e fechar a janela ou rolar o seu conteúdo quando desejar, selecionando ou arrastando os ícones apropriados pela tela.

A figura 5.2 mostra uma tela com várias janelas abertas, uma janela ativa e a região do cardápio de opções. Como pode ser notado, a janela ativa está em evidência em relação às outras janelas abertas mostradas na tela.

A figura 5.3 mostra o exemplo de uma janela onde são indicados os ícones utilizados para movimentar, redimensionar, fechar e rolar a janela e seu conteúdo. O usuário pode selecionar qualquer dos ícones desejados em qualquer janela do **EDG**. No caso dos "scroll bar", os quais são utilizados para rolar o conteúdo da janela, tanto horizontalmente quanto verticalmente, o usuário pode selecionar as setas indicadas ou, no caso de deseja maior rapidez, o retângulo branco. Para movimentar a janela pela tela, o usuário seleciona a região do título da janela. Quanto a alteração do

tamanho da janela ou o fechamento da mesma, o usuário seleciona os ícones apropriados ("close" e "size" respectivamente).

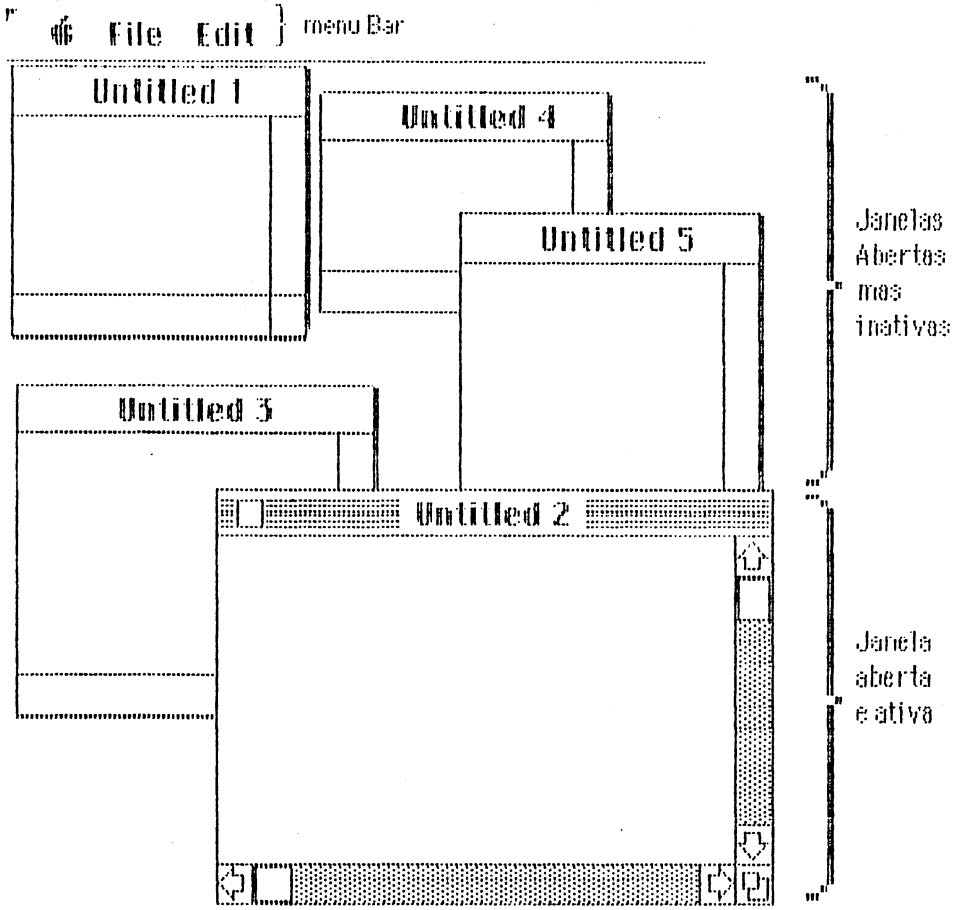


Figura 5.2 - Exemplo de uma tela com várias janelas abertas e uma janela ativa.

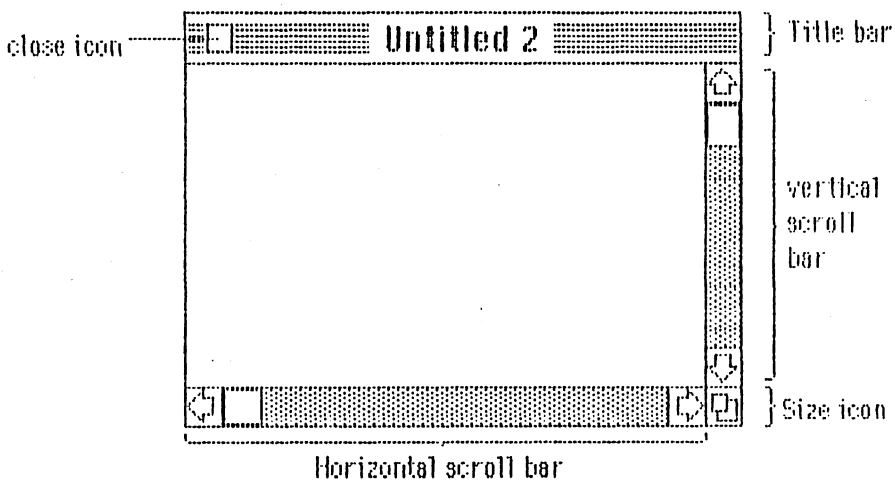


Figura 5.3 - Exemplo de uma janela ativa. São mostrados os ícones utilizados pelo usuário para movimentar, redimensionar, fechar a janela e rolar o seu conteúdo.

O **EDG** também utiliza um tipo especial de janela denominado caixa de diálogo. Caixa de diálogo é uma janela utilizada para obtenção de parâmetros de execução de determinados comandos do cardápio de opções do **EDG**. Somente uma caixa de diálogo pode estar aberta, sendo que quando aberta ela é sempre a janela ativa. No **EDG**, uma caixa de diálogo não pode ter suas dimensões alteradas, não pode ser movimentada, e ela é sempre fechada pela seleção da opção "OK" ou "CANCEL". As figuras 5.14, 5.15, 5.20, 5.21 e 5.24 mostram algumas caixas de diálogos utilizadas no **EDG**.

5.2 Técnicas interativas de construção de objetos gráficos utilizadas no EDG

O **EDG** provê várias técnicas interativas para construção de figuras gráficas, visando facilitar o trabalho de seus usuários na tarefa de edição de tipos de diagramas e também de diagramas propriamente ditos. Todas as técnicas comentadas nesta seção são do tipo **WYSIWYG** ("what you see is what you get"). Utilizando o "mouse" como dispositivo de edição, o **EDG** suporta as técnicas de posicionamento, "rubber-band", "sketching" e arrasto, usando a nomeclatura encontrada em [HEA 86]. No restante desta seção estas técnicas são comentadas e exemplificadas.

5.2.1 Posicionamento

Esta técnica é utilizada pelo usuário para informar ao **EDG** a localização de criação de um novo objeto gráfico na janela.

A mecânica de funcionamento desta técnica consiste no usuário movimentar o cursor, através do "mouse", para a posição desejada na janela ativa e então pressionar o botão do "mouse" soltando-o em seguida. Desta forma é sinalizada a localização, no sistema de coordenadas (X,Y) da janela ativa, onde o novo objeto deve ser colocado.

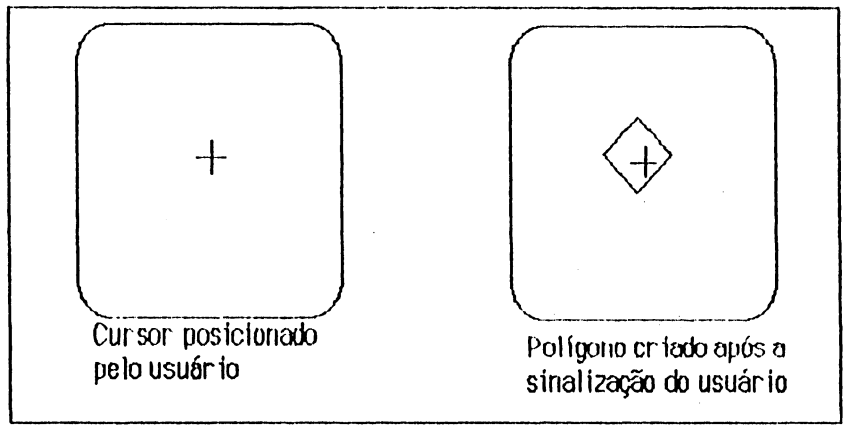


Figura 5.4 - Exemplo de funcionamento da técnica de criação de objetos por posicionamento no **EDG**.

A figura 5.4 ilustra o funcionamento desta técnica através de um exemplo de criação de um polígono regular no **MED**.

5.2.2 "Rubber-band"

A maioria dos objetos instanciados no **EDG** são criados através desta técnica. Após o usuário colocar o "mouse" na posição desejada da janela, ele inicia o processo pressionando o botão do "mouse" e termina soltando-o. A medida que o usuário movimenta o "mouse", com o botão pressionado, é mostrado na janela o desenho que está sendo formado. Esta técnica é bastante interessante, pois possibilita ao usuário uma visão imediata do que ele está desenhando no processo de edição.

Geralmente o **EDG** utiliza a técnica "rubber-band" em complementação à técnica de criação de objetos por posicionamento, comentada na seção anterior. Como os objetos criados por posicionamento são mostrados em um dimensão fixa, pré-determinada no **EDG**, o usuário pode alterar as dimensões do objeto criado no instante de sua reprodução, através da técnica "rubber-band".

A figura 5.5 ilustra o funcionamento desta técnica através da criação de um segmento de reta e de um retângulo no **MED**.

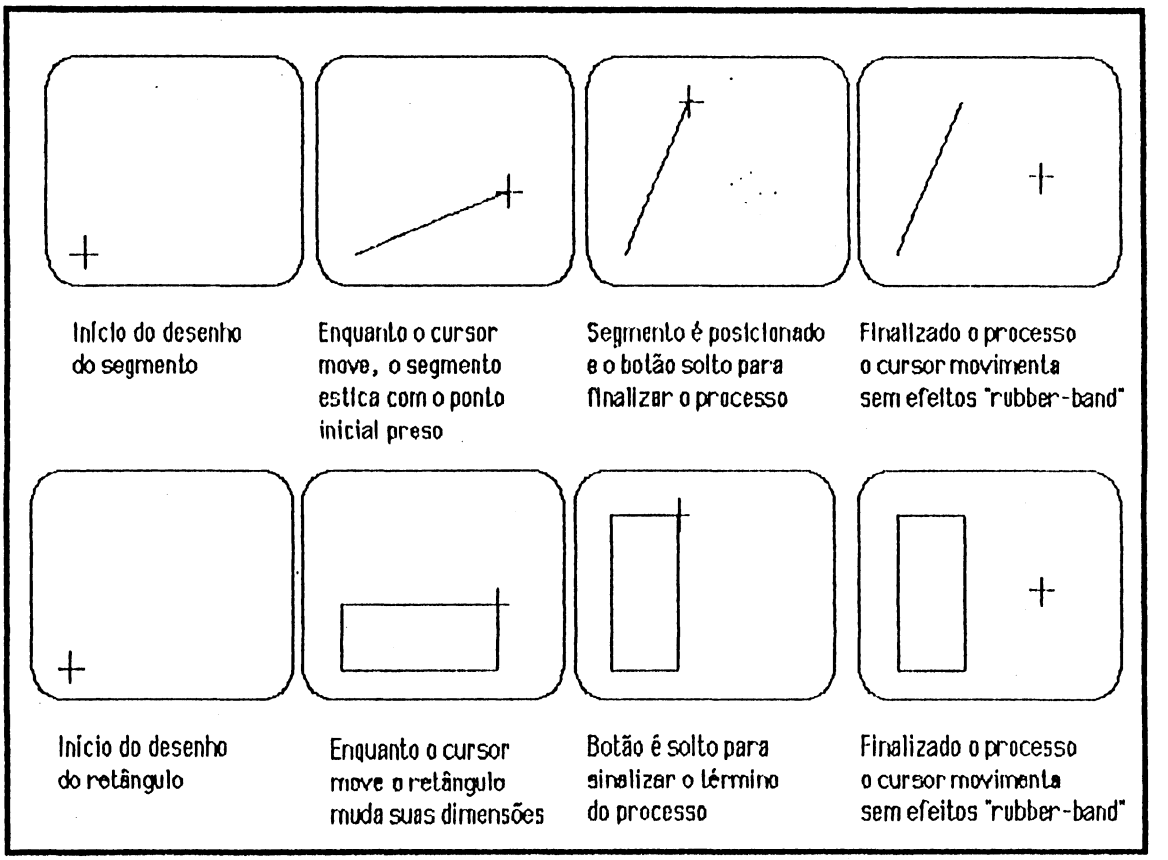


Figura 5.5 - Exemplo de funcionamento da técnica "rubber-band" implementada no **EDG**.

5.2.3 "Sketching"

Esta técnica consiste na extensão da técnica "rubber-band". Ela é utilizada na criação de polígonos irregulares no **MED** e de segmentos de arcos no **EDE**. Consiste no usuário criar um série de segmentos de reta ligados pela ordem de apontamento em um único processo de criação. Cada segmento é criado de acordo com a técnica "rubber-band". Quando o usuário informa o fim de um segmento imediatamente inicia-se a criação de outro, e assim por diante, até que o usuário sinalize o fim do processo, o que é feito através do duplo pressionamento do botão do "mouse" ("duplo-click") em um intervalo de tempo especificado no sistema operacional.

A técnica de "sketching" somente é utilizada no **EDG** para criação de segmentos de reta. A figura 5.6 ilustra o funcionamento desta técnica através da criação de um polígono irregular no **MED**.

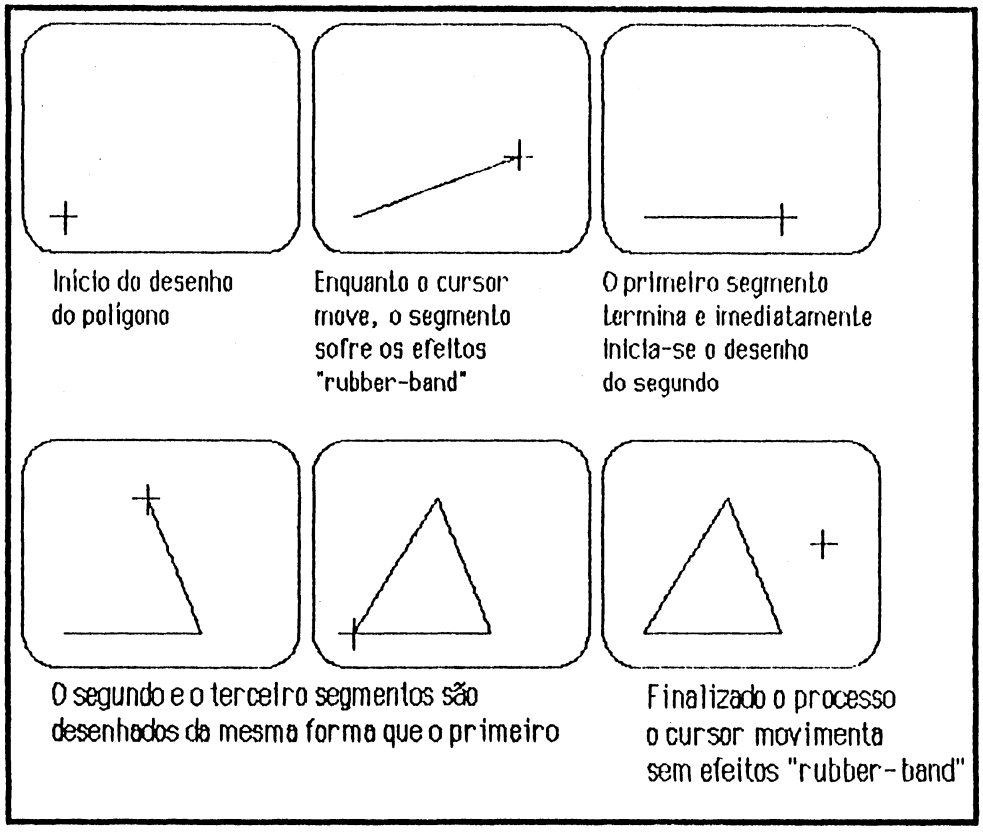


Figura 5.6 - Exemplo de funcionamento da técnica "sketching" implementada no **EDG**.

5.2.4 Arrasto ("Dragging")

Esta técnica é bastante utilizada no **EDG** para transladar elementos arrastando-os pela tela com o cursor. É utilizada tanto para transladar janelas na tela quanto objetos dentro das janelas. No caso do usuário desejar transladar objetos dentro da janela, ele deve apontar o objeto desejado e movimentar o cursor com o "mouse" pela janela, com o botão deste pressionado, soltando-o quando o cursor estiver na posição desejada. À medida que o cursor é movimentado, o desenho do objeto selecionado acompanha o movimento do cursor. O translado de janelas é semelhante ao dos objetos, só que o usuário deve apontar o ícone apropriado para isto, ou seja, a região do título da janela.

A figura 5.7 ilustra o funcionamento desta técnica.

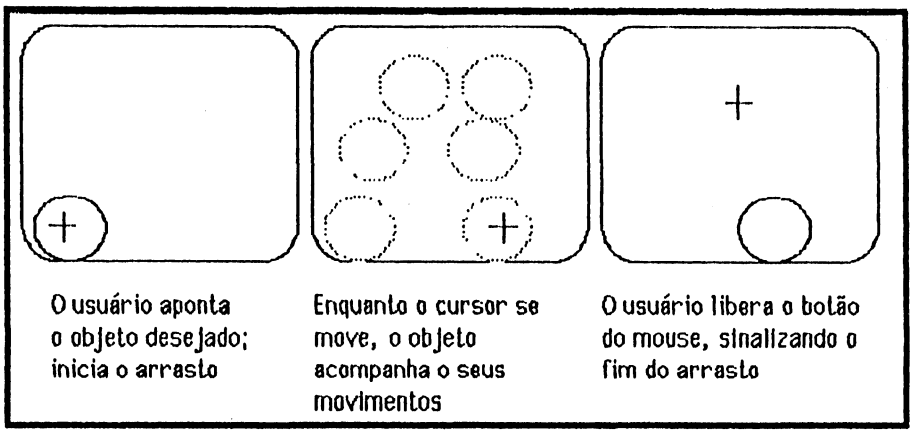


Figura 5.7 - Exemplo de funcionamento da técnica de arrasto implementada no **EDG**. O usuário pressiona o botão do mouse sobre o objeto desejado, e movimenta o cursor com o botão pressionado. Quando o objeto estiver adequadamente posicionado o usuário solta o botão, finalizando o processo. O objeto é arrastado pela janela acompanhando o movimento do cursor.

5.3 Implementação da Seleção de Entidades no EDG

O **EDG** utiliza diferentes métodos de seleção dependendo da entidade que o usuário deseja selecionar. As entidades suportadas pelo **EDG** podem ser reunidas em três grupos: ícones, textos e objetos gráficos. Embora a técnica de seleção utilizada pelo usuário seja semelhante nos três casos, o comportamento e o retorno do **EDG** difere. As técnicas de seleção de textos são implementadas pelo próprio Macintosh, enquanto as técnicas de seleção de objetos gráficos e ícones são implementadas pelo **EDG** com base nos padrões de interfaces recomendados para as aplicações que executam sobre o Macintosh. No restante desta seção são comentados e exemplificados os métodos de seleção adotados pelo **EDG**, para cada tipo de entidade alvo.

5.3.1 Seleção de textos

Em conjunto com o editor gráfico o **EDG** suporta também um editor de textos. Algumas vezes durante a edição de textos é necessário selecionar um bloco de caracteres para efetuar alguma operação. Para realizar a seleção de blocos de textos são utilizadas várias formas

aplicavéis a diferentes situações. A figura 5.8 mostra diferentes seqüências de caracteres selecionados pelas distintas técnicas utilizadas no **EDG**. Estas técnicas são implementadas através das facilidades já providas pelo sistema Macintosh [INS 85].

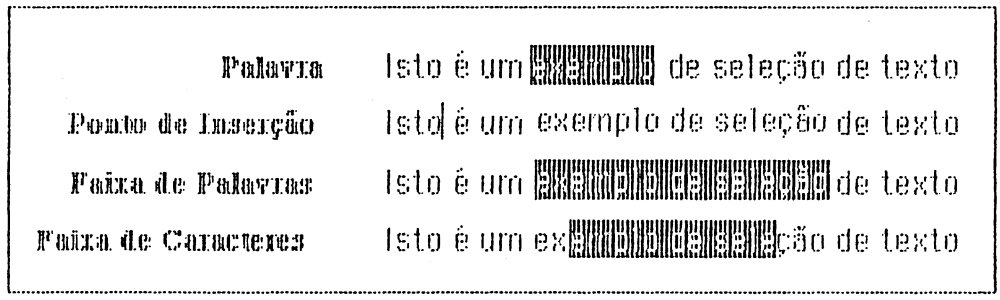


Figura 5.8 - Diferentes seqüências de caracteres selecionadas de distintas formas.

Ponto de inserção é um caso particular de seleção. Através desta forma de seleção nenhum caracter é selecionado, mas é sinalizado o local onde novos caracteres digitados serão incorporados ao texto e, também, onde dados copiados anteriormente serão inseridos. O ponto de inserção é colocado através de um simples apontamento sobre o texto desejado, ou seja, o usuário posiciona o "mouse" no local desejado e aperta o botão deste.

Uma palavra é selecionada através de um "double-click" sobre a área onde está a palavra desejada. A palavra selecionada é colocada em evidência.

Um faixa de palavras ou caracteres é seleciona através de uma operação de arrasto do cursor sobre o bloco de texto desejado. Durante a operação, à medida que o texto vai sendo selecionado, o mesmo é colocado em evidência.

5.3.2 Seleção de objetos gráficos

O **EDG** fornece três maneiras de selecionar objetos gráficos. O usuário pode fazer a seleção por simples apontamento, por faixa de objetos ou por extensão. Esta técnicas são exemplificadas na figura 5.9.

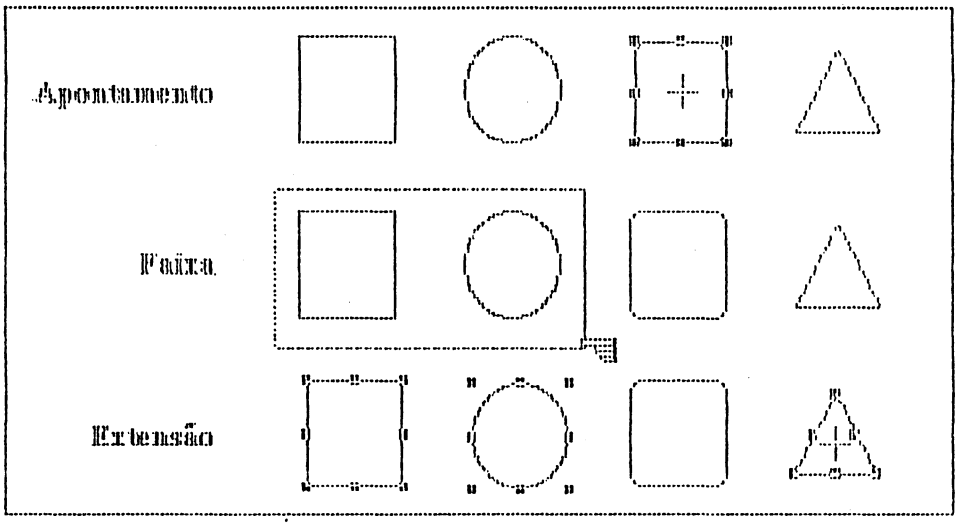


Figura 5.9 - Técnicas de seleção de objetos gráficos.

A figura 5.9 mostra as três técnicas de seleção de objetos gráficos implementadas no **EDG**. Através de apontamento o usuário seleciona um objeto de cada vez. Através da faixa de objetos o usuário pode selecionar todos os objetos que estejam dentro do retângulo por ele definido. Através da extensão o usuário pode selecionar objetos dispersos pela janela. Como pode ser visto na figura, os objetos selecionados são evidenciados pelos pegadores colocados no contorno do objeto.

Independentemente da maneira como os objetos gráficos são selecionados, eles são colocados em destaque em relação ao outros objetos através dos "pegadores". Além de evidenciar os objetos gráficos selecionados, os pegadores são utilizados, também, para auxiliar no processo de deformação dos objetos gráficos selecionados. A função dos pegadores no processo de edição gráfica será comentada com maiores detalhes na seção 5.4.;

5.3.3 Seleção de ícones

Ícones são figuras que têm por função representar alguma ação ou elemento. No **EDG** os ícones estão agrupados funcionalmente em conjunto. Cada conjunto possui somente um ícone selecionado a cada momento. A

seleção de um dado ícone em uma conjunto é realizada por um apontamento simples.O ícone selecionado é sempre colocado em evidência em relação aos outros através da inversão de todo ícone. A figura 5.10 mostra um exemplo do processo de seleção de ícones.

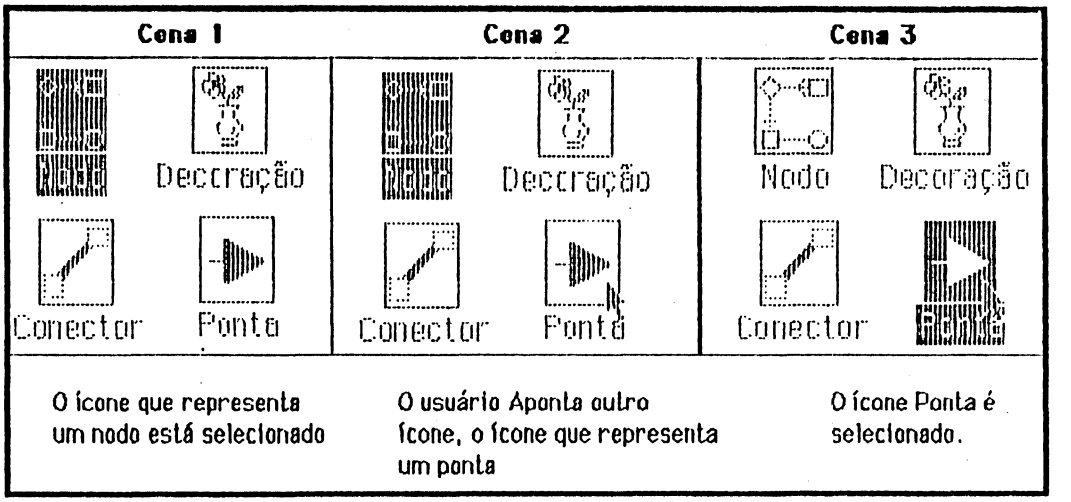


Figura 5.10 - Exemplo de seleção de ícones. A figura mostra a seleção do ícone que representa as pontas no **MED**.

5.4 Pegadores: o que são e como funcionam

Pegadores são figuras (pequenos retângulos pretos) colocadas em torno dos objetos selecionados. Sua função é evidenciar os objetos selecionados em relação aos outros e ajudar no processo de deformação das dimensões destes. Cada objeto selecionado é mostrado com oito pegadores, colocados no meio e nos vértices do menor retângulo que compreende todos os pontos do objeto, ou seja, em torno do envelope. Exceção é feita nos polígonos, nos quais são mostrados dois pegadores em cada lado do polígono, um no vértice e outro no meio da aresta.

Através dos pegadores o usuário pode alterar as dimensões dos objetos, puxando, com o cursor, o pegador desejado. A figura 5.9 mostra alguns objetos selecionados evidenciados com pegadores.

Através da figura 5.11 é exemplificada uma alteração através de pegadores: no caso, a alteração das dimensões de uma elipse.

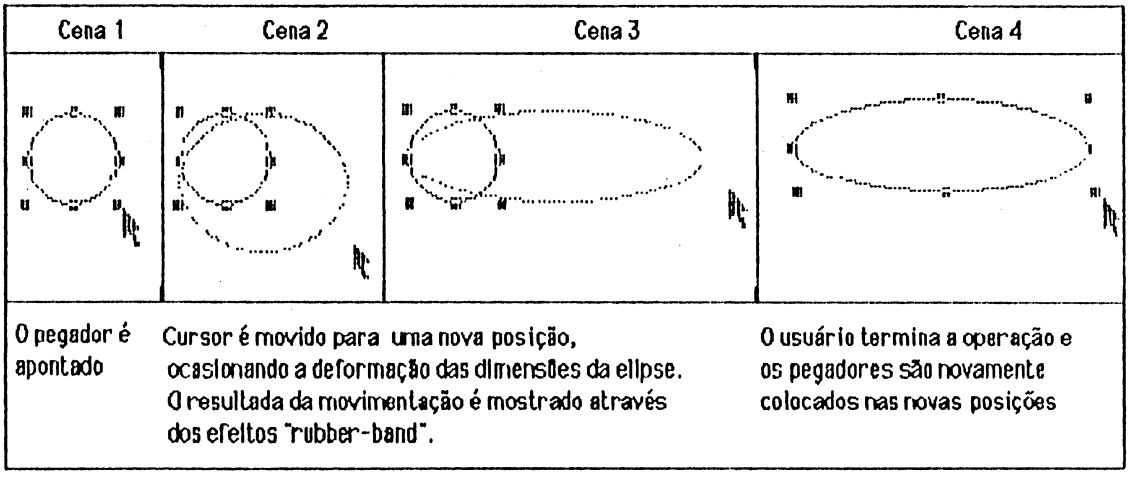


Figura 5.11 - Exemplo da função dos pegadores na alteração de objetos. A figura mostra uma elipse sendo deformada através do arrasto do pegador do canto inferior direito do envelope da elipse.

5.5 Descrição da Interface do MED

O **MED** é usado para especificar os tipos de diagramas suportados no **EDG**. O usuário especificador descreve os tipos de nodos, decorações, pontas e arcos do método alvo através deste editor. Para cumprir tal função, a interface do **MED** provê facilidades gráficas de desenho interativo. Tais facilidades incluem a utilização de janelas e de várias opções de cardápio, por onde o usuário pode interagir construindo graficamente a forma dos nodos, decorações, pontas, arcos e as regras de ligações.

Para mostrar as principais características da interface do **MED**, o restante desta seção é dividido da seguinte forma: a seção 5.5.1 mostra e exemplifica as principais janelas do **MED**, e a seção 5.5.2 descreve o cardápio de opções deste editor mostrando as opções disponíveis, sendo que os comandos mais importantes de cada opção são comentados com maiores detalhes e exemplificados.

5.5.1 As principais janelas

O usuário define um método interagindo com o **MED** através das seguintes janelas: a de catálogo do método; a de edição de elementos; a de definição das regras de ligação; e a de definição dos arcos.

É através da janela de catálogo que o usuário informa os elementos que compõem o método desejado. Cada elemento, ou seja, o tipo de nodo, decoração, arco e ponta, é representado por um ícone apropriado. A janela divide-se em duas principais regiões: a região dos ícones, que apresenta os tipos de elementos disponíveis, e a região do diretório, onde são mostrados os elementos de um método particular, com seus respectivos nomes e ícones.

A figura 5.12 mostra um exemplo da janela cátalogo do **MED**, no qual está sendo definido um método denominado "Fluxo de Dados". O método possui os nodos: função, entidade externa, depósito de dados; os arcos:

fluxo de entrada e fluxo de entrada/saída; e as pontas seta entrada e seta saída. No lado esquerdo da janela está a região dos tipos de elementos que um método pode ter. Também nesta região é mostrado o ícone da lixeira, onde o usuário coloca os elementos para exclusão. A maior parte da janela está ocupada com a região do diretório do método, onde são mostrados os elementos que definem o método. Cada elemento está representado com seu ícone apropriado e com o texto que o denomina. Os textos que denominam os elementos são digitados com todos os recursos de um editor de textos.

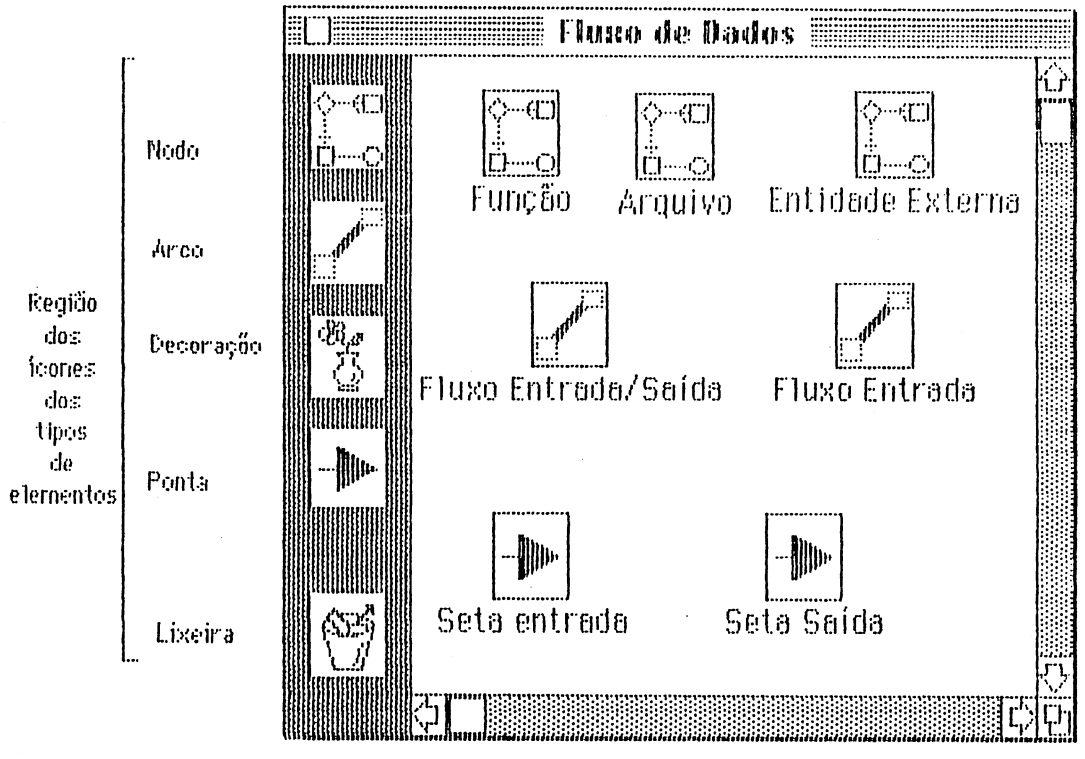


Figura 5.12 - Janela cátalogo do **MED**. A janela mostra os elementos que compõem o método "fluxo de dados".

É através da janela de edição de elementos que o usuário define a forma gráfica dos nodos, decorações e pontas. A janela subdivide-se em quatro regiões: a região dos ícones da biblioteca de objetos gráficos disponíveis; a região dos padrões de preenchimento dos objetos e das arestas destes, a região das larguras das linhas, e a região de desenho propriamente dita.

A figura 5.13 fornece um exemplo da janela de edição de elementos, onde é mostrado a definição gráfica do nodo "função", do método "fluxo de dados". Como pode ser visto na figura, o usuário definiu o nodo função através de três objetos gráficos: duas elipses e um retângulo. O usuário dipõe de vários objetos gráficos (polígono, polígono regular, retângulo, retângulo com os cantos arrendondados, elipse, arco, segmento de reta, texto e meta texto), de 24 padrões de preenchimento distintos e de 8 larguras de linhas diferentes para definir graficamente um elemento.

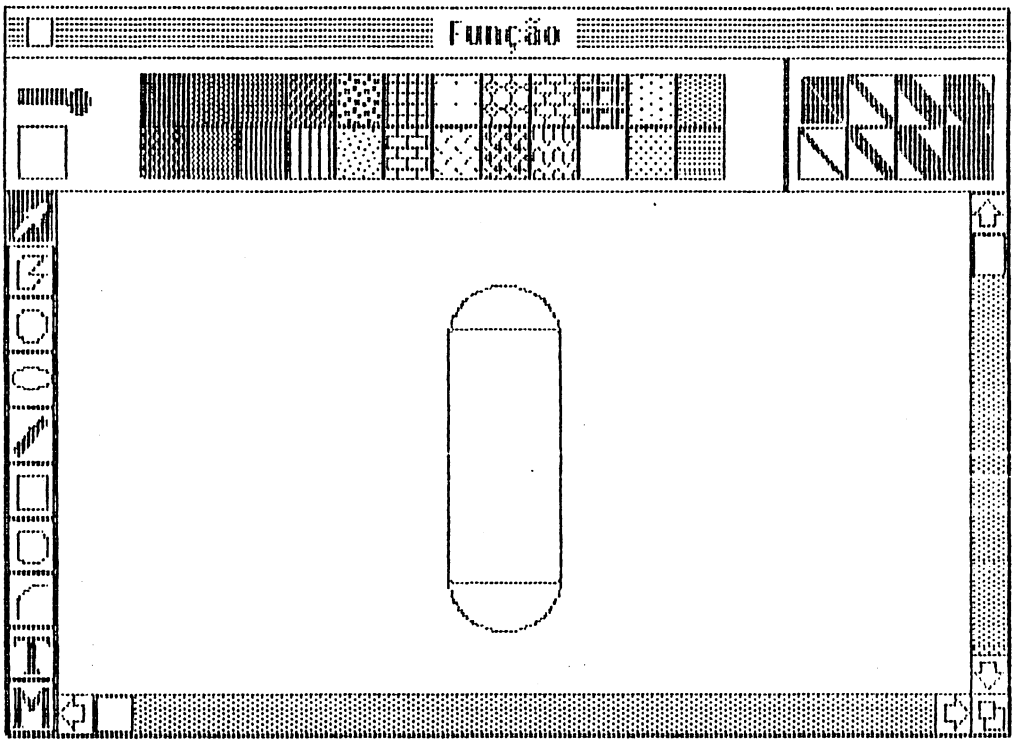


Figura 5.13 – Exemplo de uma janela de edição de elementos. É mostrada a definição gráfica do nodo "função". O nodo é especificado através da composição de objetos gráficos fornecidos pelo **MED**.

Através da janela de definição das regras de ligação (figura 5.14) o usuário informa quais nodos podem ser conectados através de quais arcos no método alvo. O usuário define, também, por quais objetos pode se dar a ligação. A janela é subdividida em quatro partes: duas áreas onde são mostrados os nodos definidos no método, uma área onde são mostrados os arcos disponíveis no método; e uma área central da janela, onde são mostrados os dois nodos alvos da definição.

A janela 5.14 mostra um exemplo de definição de uma regra de ligação.O exemplo mostra que o nodo função pode ser conectado com outro nodo função através do arco "fluxo entrada/saída", por intermédio do retângulo formador do referido nodo.

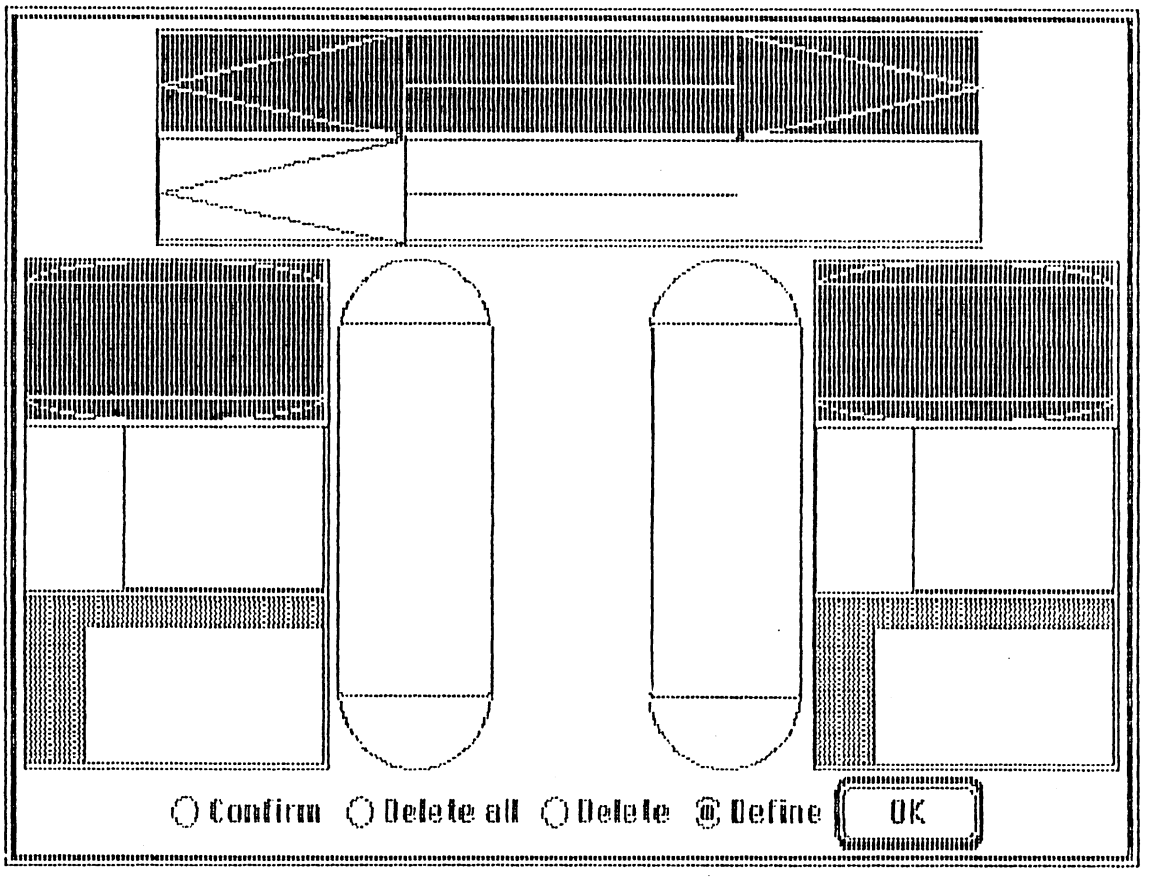


Figura 5.14 - Janela de definição das regras de ligação.

A janela de definição de arcos é usada para definir quais serão as pontas do arco alvo, a largura dos seus segmentos e o padrão de preenchimento destes segmentos. Como pode ser visto na figura 5.15, a janela mostra as pontas definidas no método, os padrões de preenchimento e as larguras das linhas diponíveis. Caso o método possua mais pontas do que é possível mostrar no diretório, o usuário pode rolar esta área para melhor escolher a ponta desejada. As pontas selecionadas são mostradas abaixo do diretório das mesmas. O arco pode ser definido, também, sem nenhuma ponta.

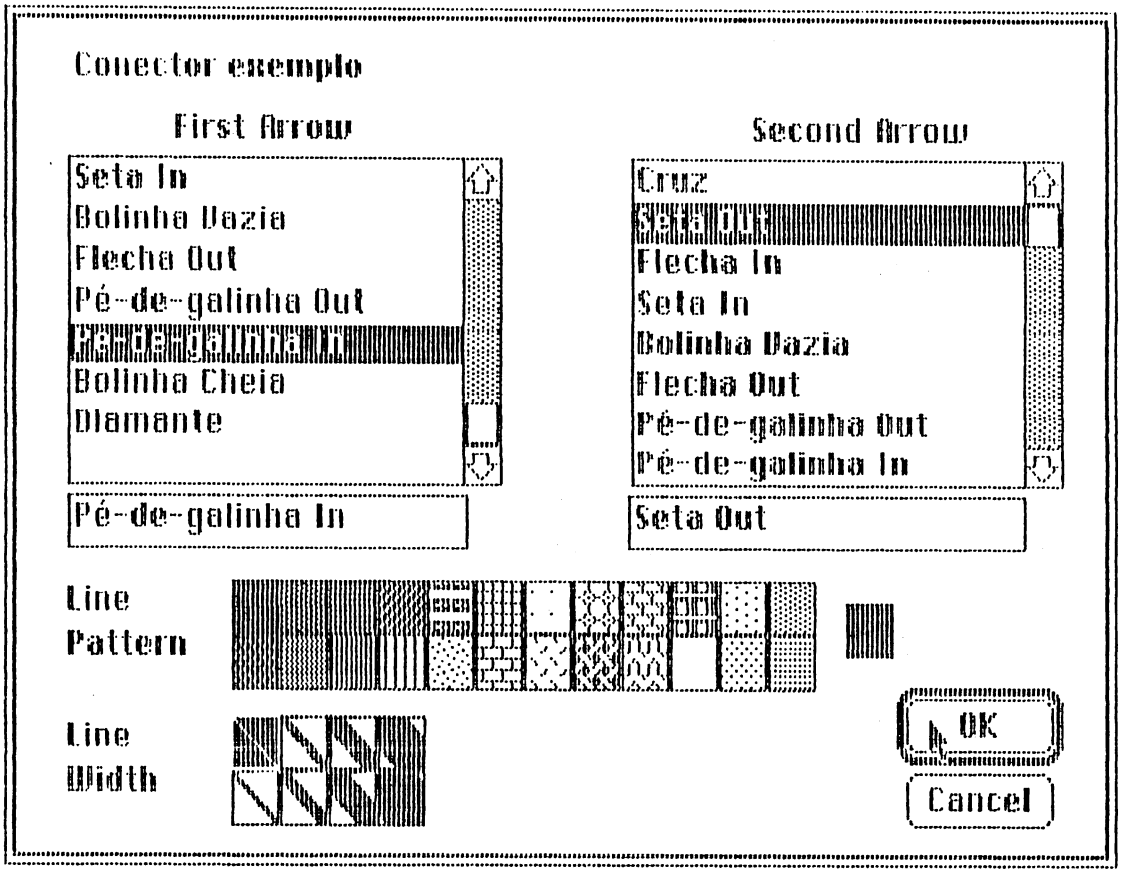


Figura 5.15 - Janela de definição de arcos. A figura mostra a definição do arco "Exemplo", na qual são selecionadas as pontas "Pe-de-galinha In" e "Seta Out".

5.5.2 O cardápio de opções

5.5.2.1 Opção FILE

Os comandos da opção FILE afetam os métodos como um todo. Através dos seus comandos, esta opção permite aos usuários criar, modificar, gravar e imprimir os métodos suportados pelo **EDG**. O **MED** somente permite a atualização de um método de cada vez. Atualmente a opção FILE possui os seguintes comandos:

- Comando NEW.

Permite abrir um novo método, inicialmente sem denominação. Tal abertura implica na exibição da janela de catálogo do novo método.

- Comando OPEN.

Este comando abre um método previamente definido. Sua ativação ocasiona a aparição de uma caixa de diálogo, a qual mostra o diretório de métodos já definidos, para facilitar a escolha do método desejado. Quando o usuário abre um determinado método, o **MED** carrega para a memória todo o contexto deste, abre a janela de catálogo do método e exibe o último estado desta. A figura 5.12 mostra um exemplo da abertura de um método, denominado **"Fluxo de Dados"**.

- Comando CLOSE.

Este comando fecha o método corrente. A operação ocasiona a liberação de toda a memória ocupada pelo método, e o fechamento de todas as janelas da tela. Caso o método não tenha sido salvo anteriormente o **MED** ativa automaticamente o comando SAVE AS.

- comando SHOW CONTENTS

Este comando ativa a janela de catálogo do método corrente.

- comando SAVE.

O comando faz uma cópia do método corrente em disco. Se o método não possui denominação o **MED** automaticamente ativa o comando SAVE AS.

- comando SAVE AS.

Este comando é utilizado para salvar um método novo ou para salvar um método com outra denominação. Sua ativação ocasiona a exibição de uma caixa de diálogo onde são exibidos o nome do método corrente, se existir, o diretório de métodos e um espaço para digitação do novo nome. O usuário tem as facilidades de rolagem da janela de diretório para poder visualizar os métodos já definidos. Este comando é bastante útil para realização de cópias extras do método corrente, ou para criação de novos com elementos análogos. O resultado da operação é uma cópia do método corrente com outro nome; a transação não remove a cópia do método corrente.

- Comando REVERT TO SAVED.

Este comando remove o método corrente da memória e carrega a última cópia, anteriormente salva. A transação equivale à ativação de um comando CLOSE, sem gravação, seguido de um comando OPEN. Porém o comando REVERT TO SAVED é mais eficiente e cômodo para o usuário que a sequência mencionada, pois dispensa as confirmações solicitadas pelo comando CLOSE e OPEN. Após a sua ativação é mostrada uma caixa de diálogo onde o usuário deve confirmar a transação.

- Comando PRINT SETUP.

O comando é usado para determinar como um método deve ser impresso. Através de uma caixa de diálogo o usuário determina o tipo de formulário usado, se deseja que a impressão seja reduzida em 50% e a orientação da impressão em relação ao formulário.

- Comando PRINT.

Imprime o método corrente de acordo com os parâmetros fornecidos através do comando PRINT SETUP. Através de uma caixa de diálogo o usuário fornece o número de cópias, as páginas desejadas e a qualidade de impressão. São impressos os nodos, decorações, pontas e arcos definidos.

- Comando QUIT.

Fecha o método corrente e encerra o **MED**. Caso o usuário tenha realizado alguma alteração posterior a última gravação do método corrente, o **MED** solicita a resposta do usuário sobre a necessidade de gravação do método.

5.5.2.2 Opção EDIT

Os comandos desta opção somente têm efeito sobre objetos gráficos e textos selecionados. Cada janela do **EDG** pode possuir vários objetos gráficos selecionados ao mesmo tempo, mas somente um texto pode estar selecionado em cada uma. Os comandos desta opção, quando ativados somente produzem efeitos sobre os objetos selecionados na janela corrente. Logo, os objetos selecionados nas outras janelas abertas, mas

não ativas, permanecem inalterados. Atualmente, a opção EDIT possui os seguintes comandos:

- Comando UNDO.

Este comando permite desfazer a última alteração realizada na janela ativa.

- Comando REDO.

Este comando permite refazer a alteração desfeita pelo comando UNDO.

- Comando CUT.

Remove os objetos selecionados da estrutura de dados e os coloca à disposição na "clipboard". (Obs.: "clipboard" é uma facilidade provida pelo sistema operacional, a qual permite a importação e exportação de dados entre aplicações no Macintosh).

- Comando COPY.

Faz uma cópia dos objetos selecionados e os coloca à disposição na "clipboard".

- Comando PASTE.

Coloca uma cópia do conteúdo da "clipboard" na janela ativa.

- Comando CONECTION RULES.

Este comando abre uma caixa de diálogo onde são definidas para o método ativo as regras de ligação entre os nodos com os arcos . A figura 5.14 mostra um exemplo de uma caixa de diálogo ativada por este comando

5.5.2.3 Opção ICON.

Os comandos da opção ICON afetam somente o ícone ativo da janela de catálogo do método. Atualmente, essa opção possui os seguintes comandos:

- Comando OPEN.

Este comando ativa a janela de definição do elemento representado pelo ícone ativo. A figura 5.13 exibe um exemplo da ativação deste comando.

- comando CLOSE.

Este comando fecha a janela de definição do elemento representado pelo ícone ativo.

- comando REMOVE.

Este comando exclui o elemento representado pelo ícone ativo da janela de catálogo do método corrente. O elemento excluído deixa de fazer parte do método.

- comando DUPLICATE.

Este comando duplica o elemento corrente, representado pelo ícone ativo. Todos as definições e objetos do elemento são duplicados juntamente com o elemento. A denominação do elemento permanece inalterada, sendo que, o usuário tem a responsabilidade de nomear a cópia do elemento.

5.5.2.4 Opção FONT

Os comandos desta opção somente têm efeito sobre o texto selecionado, ou sobre os objetos do tipo texto. A quantidade de comandos desta opção é dependente do sistema operacional residente em memória. Cada comando possui o nome do tipo dos caracteres; por exemplo, Geneva, Courier e Helvetica. Através desta opção é permitido ter em uma mesma janela vários textos distintos com diferentes fontes. Esta facilidade permite ao usuário tornar a edição dos métodos e diagramas mais estética. A figura 5.16 mostra um exemplo de edição de um elemento com vários textos com tipos de impressão, estilos, tamanhos e ajustamentos distintos.

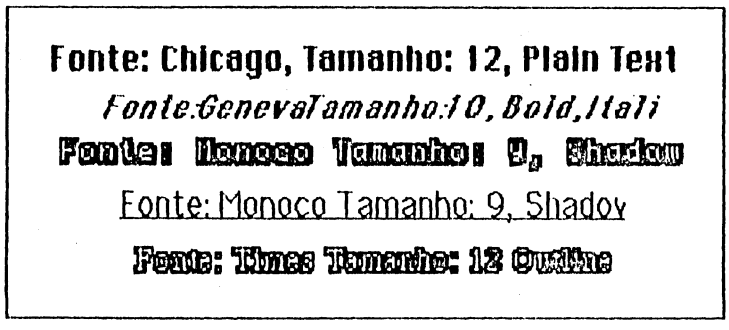


Figura 5.16 - Vários textos com fontes, estilos, tamanhos e ajustes distintos.

5.5.2.5 Opção STYLE

Da mesma forma que os comandos da opção FONT, os comandos desta opção somente têm efeito sobre um texto selecionado ou sobre um conjunto de objetos selecionados do tipo texto. Os comandos estilizam o texto da forma desejada pelo usuário.

Os comandos, ou seja, os estilos disponíveis são:

- PLAIN TEXT (texto sem estilização);
- BOLD (negrito);
- ITALIC (itálico);
- UNDERLINE (sublinhado);
- OUTLINE (com contorno);
- SHADOW (sombreado).

5.5.2.6 Opção SIZE

Seus comandos agem sobre os mesmos alvos dos comandos das opções FONT e STYLE. Os nomes dos seus comandos representam o tamanho dos caracteres em pontos. Como nem todos os tipos de impressão de caracteres possuem todos os tamanhos disponíveis nesta opção, os tamanhos mostrados em estilo "outline" são diretamente disponíveis no tipo, enquanto que os outros tamanhos são mostrados sem nenhuma estilização.

5.5.2.7 Opção JUST

Seus comandos agem sobre os mesmos alvos dos comandos das opções FONT,STYLE e SIZE. O texto é ajustado dentro do menor retângulo do próprio texto. Os ajustes disponíveis são: à esquerda (LEFT), à direita (RIGHT) e ao centro (CENTER).

Nota : *os comandos da opção FONT, STYLE, SIZE e JUST são implementados utilizando as facilidades de edição de texto providas pelo Macintosh.*

5.5.2.8 Opção ALIGN

Os comandos desta opção agem sobre os objetos selecionados na janela de desenho corrente. Os comandos possuem a função de alinhar os objetos selecionados a um outro objeto apontado pelo usuário (no caso do comando BETWEEN o usuário tem de apontar dois objetos ao invés de somente um). Como todos os objetos podem ser limitados por um retângulo, o qual contém todos os pontos do objetos (denominado envelope), todos os alinhamentos são realizados em relação aos lados (esquerdo, direito, topo e base) ou ao centro destes retângulos, ou seja, os alinhamentos são realizados em relação aos envelopes dos objetos.

Os alinhamentos disponíveis possuem o mesmo nome do comando, a saber:

- horizontal (HORIZONTAL);
- vertical (VERTICAL);
- esquerda com esquerda (LEFT TO LEFT);
- esquerda com direita (LEFT TO RIGHT);
- direita com esquerda (RIGHT TO LEFT);
- direita com direita (RIGHT TO RIGHT);
- topo com topo (TOP TO TOP);
- topo com Base (TOP TO BOTTOM);
- base com topo (BOTTOM TO TOP);

- base com base (BOTTOM TO BOTTOM);
- entre dois objetos (BETWEEN);
- no centro de um objeto (CENTER).

Os comandos Horizontal, Vertical e Between realizam o alinhamento em relação ao centro dos envelopes dos objetos, enquanto que os demais realizam o alinhamento em relação as arestas dos envelopes. A figura 5.17 mostra alguns exemplos destes alinhamentos.

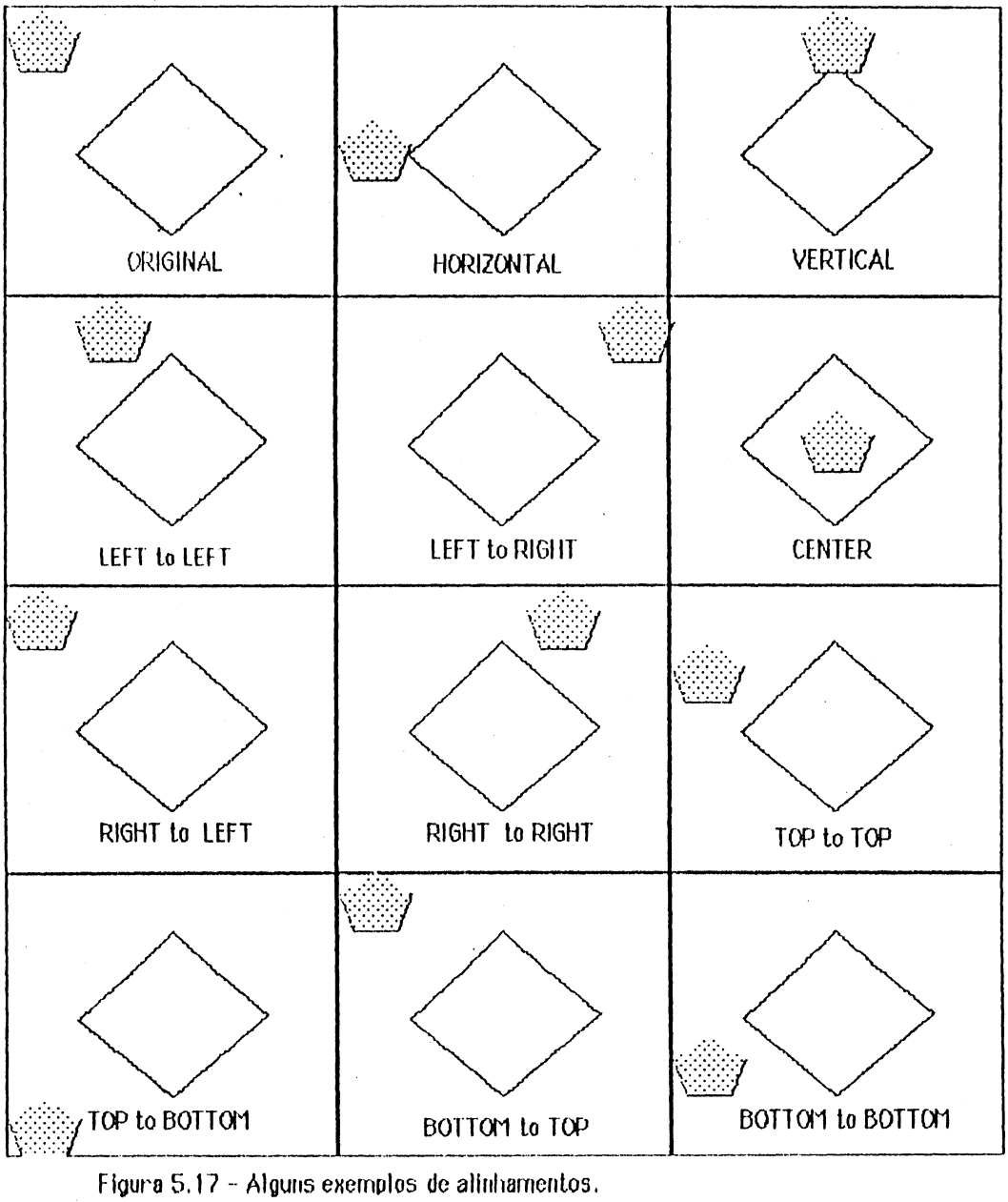


Figura 5.17 - Alguns exemplos de alinhamentos.

5.5.2.9 Opção ARRANGE

Os comandos desta opção, a exemplo dos comandos da opção ALIGN, também agem sobre os objetos selecionados na janela de desenho corrente. Atualmente, esta opção dispõe dos seguintes comandos:

- Comando SELECT.

Seleciona um objeto apontado pelo usuário. Para auxiliar esta tarefa, o usuário pode solicitar que alguns objetos sejam apagados temporariamente da janela.

- Comando DRAG.

É utilizado para transladar objetos na janela corrente. Geralmente é usado para movimentar objetos pequenos.

- Comando BRING FORWARD e SEND BACKWARD.

Coloca os objetos selecionados na frente ou atrás de um objeto apontado pelo usuário. A figura 5.18 mostra um exemplo de um objeto antes e depois de ser movido para frente e para trás de outro objeto.

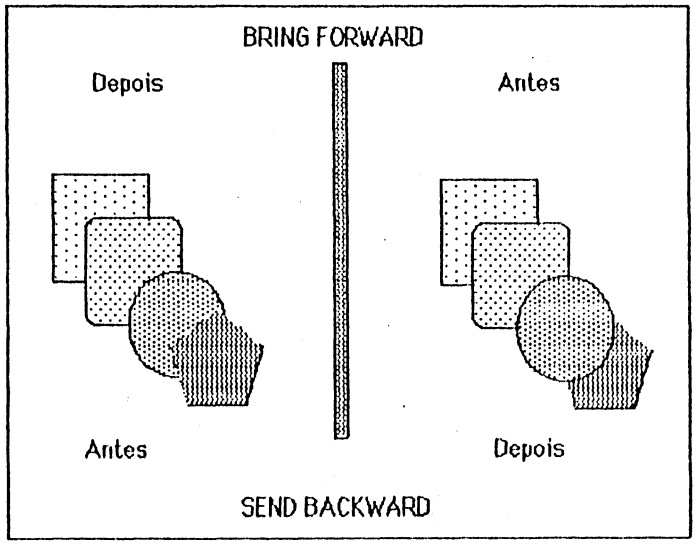


Figura 5.18 - Exemplo dos comandos BRING FORWARD e SEND BACKWARD.

- Comandos FLIP HORIZONTAL e FLIP VERTICAL .

Realizam uma operação de espelhamento nos objetos selecionados em relação a um eixo horizontal ou vertical. A figura 5.19 mostra um exemplo de um objeto antes e depois de submetido a um espelhamento horizontal e outro vertical.

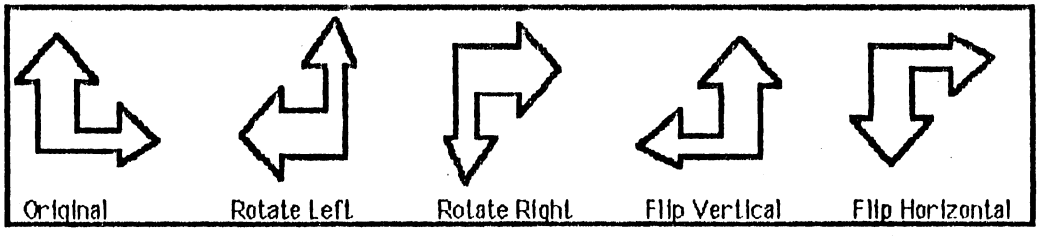


Figura 5.19 - Exemplo de execução dos comandos de rotação à esquerda e à direita e espelhamento vertical e horizontal.

- Comandos ROTATE LEFT e RIGHT.

Realizam uma operação de rotação de -90°, no caso de ROTATE LEFT, ou 90°, no caso de ROTATE RIGHT, nos objetos selecionados. A figura 5.19 mostra um exemplo de um objeto antes e depois de submetido a uma rotação à esquerda e à direita.

- Comando ROTATE FREE.

Realiza uma operação de rotação nos objetos selecionados a critério do usuário. À medida que o usuário solicita a rotação, o **MED** mostra o resultado da operação. Os objetos podem ficar deformados devido às operações de rotação livre.

5.5.2.10 Opção ATTRIBUTE

Através dos comandos desta opção o usuário pode mudar as características gráficas dos objetos selecionados e fornecer dados sobre as ligações e operações que podem ser realizadas sobre estes objetos.

A opção "Attribute" atualmente possui dois comandos:

- Comando GENERAL.

Este comando ativa uma caixa de diálogo onde o usuário pode configurar a apresentação gráfica de objetos selecionados. A figura 5.20 mostra um exemplo desta caixa de diálogo.

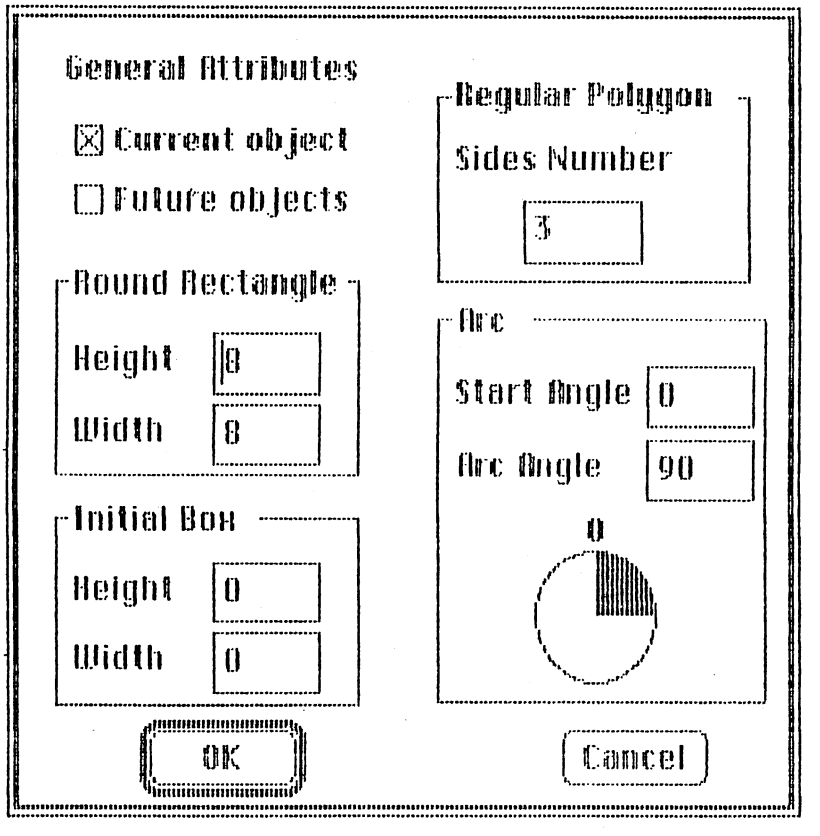


Figura 5.20 - Caixa de diálogo apresentada pela ativação do comando GENERAL.

O comando possibilita ao usuário mudar o número de lados dos polígonos regulares, o diâmetro de curvatura dos cantos dos retângulos chanfrados, os valores dos arcos de circunferência e as dimensões dos envelopes iniciais dos objetos gráficos. O usuário pode trocar as características citadas nos objetos já desenhados ou somente nos futuros objetos. A figura 5.21 mostra como os parâmetros "height" e "width" dos retângulos chanfrados são interpretados pelo **MED**.

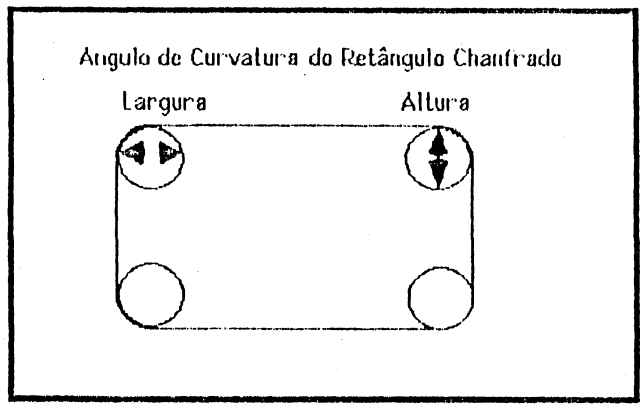


Figura 5.21 - Formação do ângulo de curvatura dos cantos dos retângulos chanfrados.

- Comando OBJECT.

Este comando ativa uma caixa de diálogo por onde o usuário fornece algumas informações quanto às operações que podem ser submetidas sobre objeto selecionado. A caixa de diálogo é mostrada na figura 5.22.

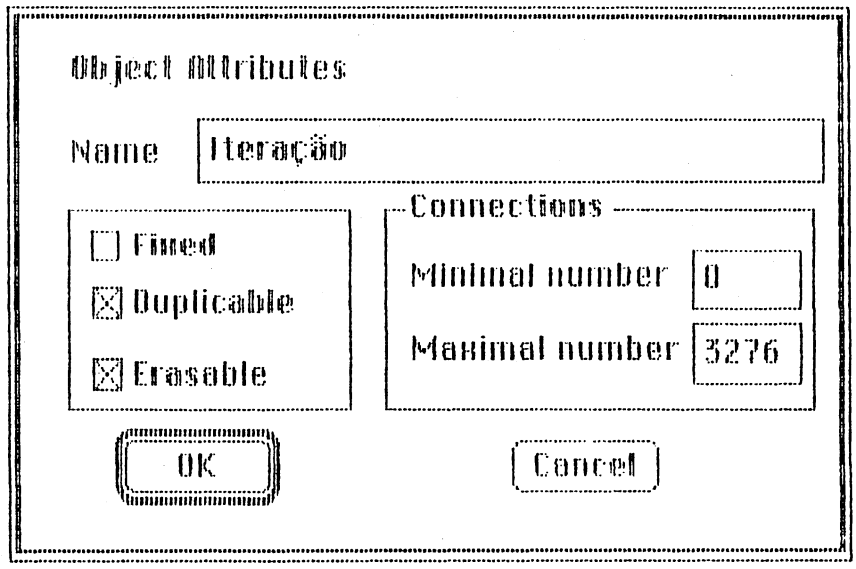


Figura 5.22 - Caixa de diálogo ativada pelo comando OBJECT da opção ATTRIBUTES. É mostrado os atributos do losango que forma o tipo de nodo "módulo", definido na figura 4.7.

O comando possibilita ao usuário informar: se o objeto pode ser duplicado ou excluído no **EGE**; se ele é fixo em relação aos outros objetos do nodo; o nome do objeto; e o número de conexões mínimas e máximas que os objetos devem e/ou podem ter nos diagramas do **EGE**.

Por exemplo, o nodo do tipo "módulo", definido na figura 4.7, possui dois objetos: um retângulo e um losango. O losango é definido por este comando como sendo duplicável, excluível, e não fixo, ou seja, móvel; poderá ter de zero a 3.276 ligações; e possui o nome de "Iteração"

5.6 Descrição da Interface do EDE

É através do **EDE** que são editados os diagramas. O **EDE** somente manipula diagramas dos tipos especificados anteriormente no **MED**, ou seja, o **EDE** somente suporta diagramas pertencentes aos métodos definidos no **MED**.

Os diagramas são representados em forma de árvore sendo que cada nó é um diagrama. Existe um diagrama que é a raiz da árvore. Os diagramas filhos são gerados a partir dos refinamentos dos nodos do diagrama pai. Desta forma, cada diagrama pode ter vários nodos refinados em diagramas de níveis inferiores. Todo o processo de refinamento e condensação de diagramas é feito pelo **EDE**; desta forma é mantida a integridade sintática de todos os níveis.

O usuário interage com o **EDE** através de quatro janelas básicas: a janela dos nodos, a dos arcos, a das decorações do método alvo e a janela de edição dos diagramas. A figura 5.22 mostra um exemplo de uma tela do **EDE** no modo de edição de um diagrama pertencente ao método "fluxo de dados". Na figura pode ser visto a janela dos nodos e a dos arcos no lado esquerdo da tela, e a janela de edição de diagramas, no caso a maior das três, ocupando o restante da tela. Como o método "fluxo de dados" não possui decorações esta janela não é mostrada.

A figura 5.23 também mostra algumas das facilidades providas pelo **EDE**. São exemplificados os vários tipos de ligações suportadas pelo **EDE**, ou seja, conexões com um ou vários segmentos e curvas suavizadas. São também exemplificadas algumas das facilidades de edição de textos, como pode ser visto, pois vários objetos possuem textos com estilização, tamanho e fonte distintos. A figura também mostra alguns textos vinculados a determinadas ligações, outros soltos no diagrama e alguns dentro de nodos.

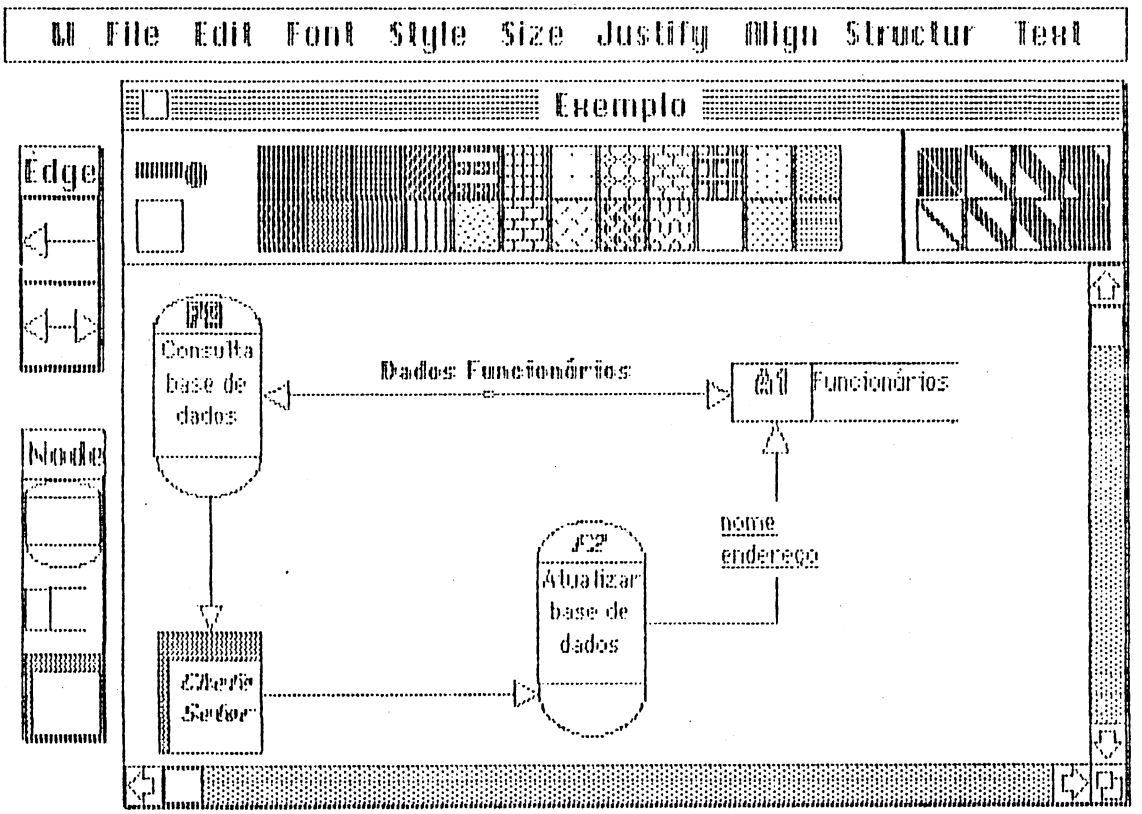


Figura 5.23 - Exemplo de uma tela no **EDE**.

Quando o usuário efetua uma operação de translação de nodos, todas as conexões dos nodos transladados são atualizadas automaticamente, e os textos e decorações vinculados às conexões também acompanham a atualização.

A exclusão de um arco resulta também na retirada de todos os textos e decorações vinculados ao arco excluído.

A exclusão de um nodo gera a exclusão de todos os arcos conectados ao mesmo, como também de todos os diagramas gerados pelo refinamento do nodo excluído.

O **EDE** provê duas maneiras básicas de edição de textos: texto dentro de objetos e textos livres. A edição de textos dentro de objetos é limitada às dimensões do mesmo. Para que o usuário edite textos maiores que a dimensões do objeto, o **EDE** provê facilidades de enquadramento do texto

numa janela. Esta operação consiste na criação de uma janela temporária com as mesmas dimensões do objeto, onde o texto é rolado quando necessário. Os textos livres não possuem limitação de área, logo não necessitam de janelas de edição. A figura 5.24 mostra um exemplo de edição de textos dentro de um retângulo. É criada uma janela temporária para o enquadramento do texto juntamente com barras de rolagem vertical e horizontal.

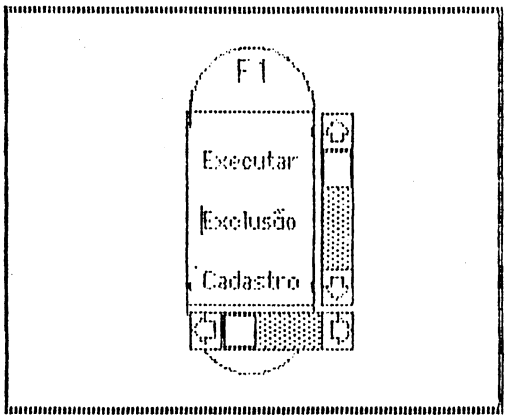


Figura 5.24 - Exemplo de um enquadramento de texto. É criada uma janela temporária por intermédio da qual o usuário pode editar textos de até 32 K (limitação do Macintosh). O usuário pode rolar o texto quando desejado através de barras de rolagem vertical e horizontal, criadas junto com a janela temporária.

O processo de reprodução de elementos do método no diagrama se dá por apontamento. Ou seja, o usuário aponta o elemento desejado na janela apropriada (por exemplo, aponta o nodo desejado na janela dos nodos), e então sinaliza na janela de edição o local e as dimensões onde o novo elemento deve ser reproduzido. No caso especial dos arcos, o usuário deve apontar, na janela de edição, os dois nodos alvos da ligação. O **EDE** então verifica se a ligação pode ser concluída, com base nas definições feitas anteriormente no **MED**. A figura 5.25 mostra um exemplo de criação de arcos no **EDE**.

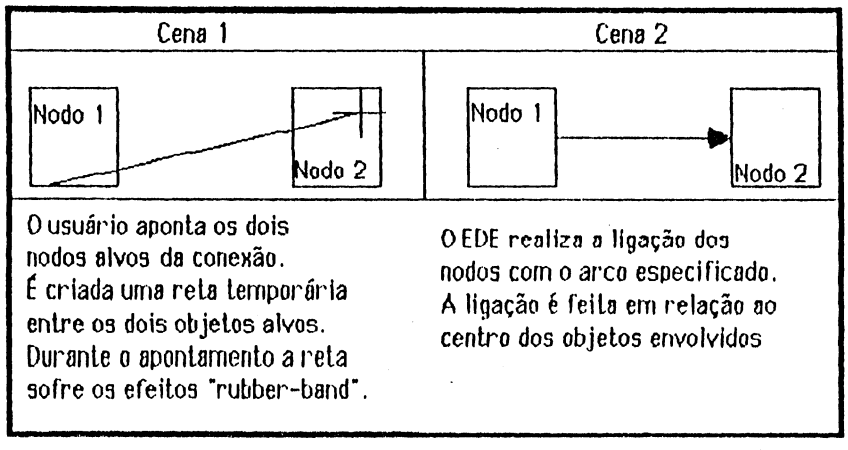


Figura 5.25 - Exemplo de ligação entre dois nodos. A cena 1 mostra o processo de apontamento dos nodos alvos da ligação e a cena 2 mostra como os nodos ficam conectados com o arco escolhido pelo usuário. A conexão somente é concretizada se a ligação está especificada na tabela de ligações válidas.

O usuário pode solicitar a reprodução de elementos com as dimensões diferentes daquelas especificadas no **MED**, dando-se assim maior liberdade de desenho. Após a reprodução, o novo elemento fica selecionado, com os seus respectivos pegadores.

O **EDE** mantém a consistência sintática dos diagramas editados. Isto é feito através de dois mecanismos. No primeiro, o usuário somente pode conectar nodos com os arcos especificados no **MED**. O outro mecanismo é o controle dos refinamentos e condensações, realizado automaticamento pelo **EDE**. Por exemplo, se o usuário exclui um nodo em um diagrama pai, todos os diagramas filhos do nodo são também excluídos.

O **EDE** permite quebrar uma conexão em vários segmentos. Desta forma, o usuário pode traçar o caminho das conexões, para que elas passem nos pontos indicados.

A próxima seção mostra as opções do cardápio do **EDE**. Algumas opções e comandos são idênticos aos do **MED**, logo somente serão citados.

5.6.1 O Cardápio de Opções

5.6.1.1 Opção FILE

Os comandos desta opção afetam os diagramas como um todo. Através dos seus comandos é possível criar novos diagramas, modificar, gravar e imprimí-los. Como o **EDE** somente permite a edição de diagramas cujos métodos tenham sido previamente especificados no **MED**, o usuário necessita informar qual o método desejado antes de iniciar qualquer operação sobre os diagramas. Atualmente esta opção consiste dos seguintes comandos:

- New: cria um novo diagrama;

- Open: abre um diagrama já existente;

- Save: salva em disco o diagrama editado;

- Save As: realiza uma cópia em disco do diagrama.

- Revert to Saved: recupera para a memória a última cópia em disco do diagrama.

- Print SetUp: configura a impressora.

- Print: Imprime o diagrama.

- Quit: sai do **EDE**.

5.6.1.2 Opção EDIT

Esta opção está dividida no **EDE** em dois grupos de comandos. O primeiro grupo de comandos afeta os objetos gráficos e textos selecionados, e são os mesmos comandos do **MED**, ou seja, undo, redo, cut, copy e paste. O segundo grupo possui somente o comando "smooth". O

comando "smooth" é usado para suavizar ligações. Quanto ativado, ele age sobre o arco selecionado, suavizando-o. A suavização de arcos consiste na aplicação de um algoritmo [EMM 86] de curvas B-Spline sobre os pontos que formam os segmentos destes.

5.6.1.3 Opções FONT, STYLE, SIZE, JUST e ALIGN

Estas opções possuem os mesmos comandos das opções com o mesmo nome do **MED** e agem sobre os mesmos objetos no **EDE**. Os comandos da opção Allgn no **EDE** possuem comportamento diferenciado do **MED**. No **MED** os alinhamentos são realizados em relação ao envelope de cada objeto selecionado, enquanto no **EDE** os alinhamentos são realizados em relação ao envelope que envolva todos os objetos que formam os nodos e/ou decorações envolvidas no processo. Desta forma um nodo que possua, por exemplo, três objetos será alinhado como se fosse um objeto único.

5.6.1.4 Opção DIAGRAM

Os comandos desta opção têm efeito sobre os nodos selecionados no diagrama corrente e também sobre a forma como os diagramas são exibidos na janela. Atualmente esta opção possui os seguintes comandos:

- Comando CHILD.

Torna corrente o diagrama filho do nodo apontado. O diagrama corrente é o alvo das operações do **EDE**.

- Comando PARENT.

Chama o pai do diagrama corrente, colocando-o como o novo diagrama corrente.

- Comando SHOW STRUCTURE.

Mostra a árvore de diagramas. O usuário pode apontar o diagrama que ele deseja colocar como corrente.

- Comando REFINE.

Cria um diagrama filho do nodo apontado. O EDE cria uma nova janela para edição do diagrama filho e ,automaticamente, reproduz no novo diagrama todas as interfaces que estão presentes no nodo pai.

- Comando CONDENSE.

Transfere o diagrama filho para o diagrama pai, isto é, todos os elementos criados no diagrama filho sobem para o diagrama pai, mantendo as conexões com as interfaces. A figura 5.26 exemplifica a execução deste comando e do comando COARSEN. O nodo em destaque na cena 1 representa o nodo alvo do comando COARSEN. Após a execução deste comando, uma nova janela é criada para a edição do diagrama filho, representada pela cena 2 na figura. O diagrama filho herda as conexões do nodo pai. A cena 3 mostra o diagrama filho após a criação e conexão de outros nodos. A cena 4 mostra o diagrama pai após a execução do comando REFINE. Todos os nodos e conexões do diagrama filho sobem para o diagrama pai e o diagrama filho é excluído.

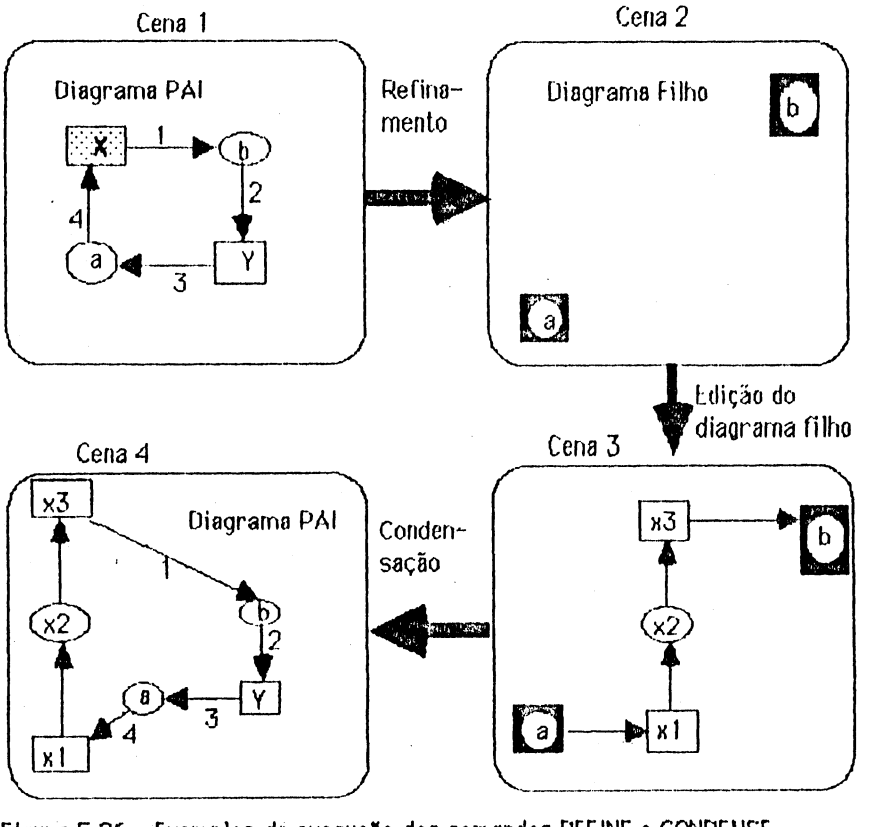


Figura 5.26 - Exemplos de execução dos comandos REFINE e CONDENSE.

- Comando SHOW.

Mostra todo o diagrama corrente em uma única janela. O usuário pode apontar a parte do diagrama que ele deseja ver com maiores detalhes.

- Comando REDUCE.

Reduz em 25% as dimensões do diagrama corrente, permitindo ao usuário visualizar uma maior área deste. O máximo de redução permitida é 75%.

- Comando ENLARGE.

Aumenta em 50% as dimensões do diagrama corrente. O usuário pode ver com maiores detalhes áreas menores do diagrama. O máximo de aumento permitido é 200%.

- Comando REDUCE TO FIT

Reduz as dimensões do diagrama para que toda a sua área possa ser visualizada e editada pelo usuário.

5.6.1.5 Opção TEXT

Os comandos desta opção tem efeito sobre os textos dos diagramas. Atualmente possui os seguintes comandos:

- Comando TURN ON ou TURN OFF.

Respectivamente coloca ou retira o **EDE** do modo de edição de textos.

- Comando FIT BOX TO TEXT.

Este comando somente é válido para textos dentro de objetos gráficos. O objeto selecionado tem suas dimensões alteradas para comportar todo o texto que está a ele vinculado.

- Comando ATTACH.

Este comando somente tem efeito nos textos não vinculados aos objetos gráficos. O objeto texto selecionado é vinculado à conexão apontada. A partir daí, todas as alterações no arco terão efeito sobre o

objeto texto. Assim sendo, as translações das conexões ou exclusões destas refletirão sobre os objetos textos vinculados.

- Comando DETACH.

Desvincula os objetos texto selecionados das conexões, desta forma as translações ou exclusões nas conexões não terão efeito sobre eles. Os textos alvo deste comando são colocados como textos livre no diagrama.

- Comando FIND

Procura nos textos indicados um conjunto de caracteres especificado através do comando FIND WHAT.

- Comando REPLACE.

Troca o texto selecionado por um conjunto de caracteres especificados no comando FIND WHAT.

- Comando REPLACE EVERY WHERE

Ao contrário do comando REPLACE, que somente faz a troca de caracteres no texto selecionado, este comando faz a troca em todos os textos indicados.

- Comando FIND WHAT.

Este comando ativa uma caixa de diálogo por onde o usuário descreve o conjunto de caracteres que ele deseja procurar, o conjunto de caracteres que ele deseja trocar e os textos onde ele deseja que seja efetuada a operação. O usuário pode indicar textos restritos aos objetos selecionados, a todo o diagrama e também a todos os níveis do diagrama. A caixa de diálogo ativada por este comando é mostrada na figura 5.27.

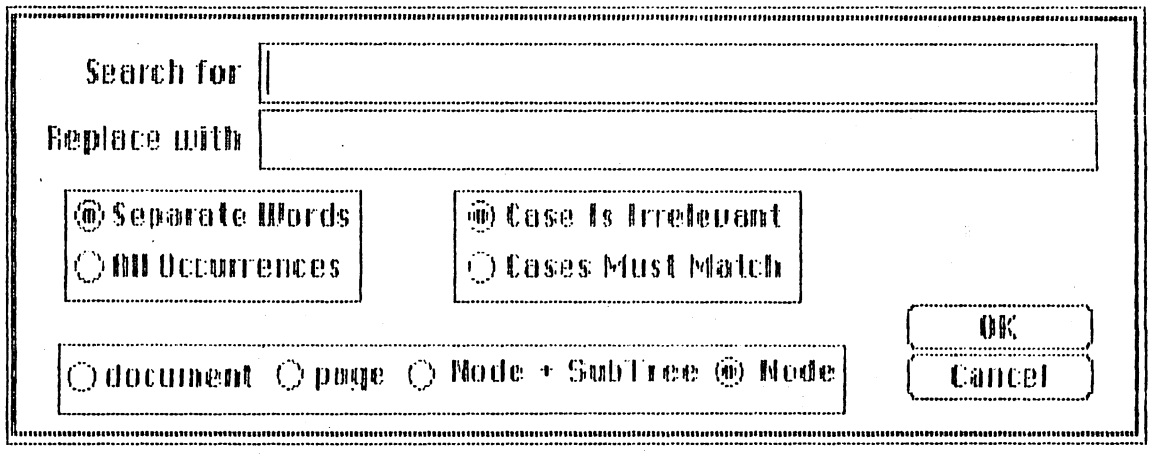


Figura 5.27 - Caixa de diálogo ativada pelo comando FIND WHAT. O usuário especifica a "string" a ser procurada, "string" a ser trocada pela "string" procurada. O usuário também indica em quais textos deve se proceder as ações de troca e procura. Também é indicado se a procura deve ser efetuada somente em palavras ou conjuntos de caracteres, e se as letras malúsculas e minúsculas são relevantes para a procura.

6 EXEMPLOS DE USO

Este capítulo apresenta alguns exemplos de utilização do **EDG**. A seção 6.1 mostra, de forma mais detalhada, um exemplo prático de utilização do **EDG**, através da descrição e edição de diagramas do tipo Entidade-Relacionamento. A seção 6.2 apresenta um outro exemplo prático de utilização do **EDG**, através da especificação e edição de diagramas do tipo Redes Marcadas [HEU 87]. A seção 6.3 mostra a definição de um método denominado "transporte". O método "transporte" não possui nenhuma aplicação no processo de desenvolvimento de software, e servirá apenas para mostrar algumas das facilidades gráficas providas pelo **EDG** na descrição e utilização de diagramas.

6.1 Primeiro Exemplo: Entidade Relacionamento

Esta seção mostrá a definição do método "Entidade Relacionamento" [CHE 77]. A subseção 6.1.1 mostra a definição do método no **MED**, e a subseção 6.1.2 mostra a definição de um diagrama do método. Ambas as subseções apresentam a utilizando do **EDG** de forma bastante detalhada.

6.1.1 Utilizando o MED

Para criar um novo método no **MED**, o usuário deve escolher o comando **NEW** da opção **FILE** do cardápio. O **MED** abre uma nova janela chamada "untitled", que corresponde ao novo método.

Para indicar a criação do nodo "relacionamento" procede-se da seguinte forma:

- seleciona-se com "mouse" o ícone que representa os tipos de nodos;

- arrasta-se o ícone para dentro da região de desenho da janela. A medida que o ícone é arrastado o **MED** mostra um retângulo que

representa o ícone;

- solta-se o botão do "mouse" na posição desejada. Após liberado o "mouse" o **MED** cria a representação de um novo tipo de nodo. Na posição onde o usuário indicou é criado um ícone do tipo nodo com a denominação "Node";

- digita-se o nome do novo nodo, no caso "relacionamento", sobre o texto selecionado. Desta forma o nome "node" é trocado por "relacionamento" (figura 6.1);

A criação dos outros ícones do método é feita efetuando os mesmos passos da criação do ícone do relacionamento.

Criado o ícone representativo do nodo "relacionamento" é necessário especificar graficamente a sua forma. Para tal o usuário escolhe o comando **Open** da opção **Icon** do cardápio, ou fornece um "double-click" sobre ícone do nodo Relacionamento. O **MED** então abrirá uma nova janela, nomeada "Relacionamento", para a edição do mesmo, como mostra a figura 6.2.

O nodo Relacionamento possui a forma de um losango, ou seja um polígono regular de quatro lados. Para especificar sua forma gráfica, são efetuados os seguintes passos:

- seleciona-se o ícone de polígono regular na região dos ícones dos objetos primitivos;

- troca-se o default do número de lados de um polígono regular, de 3 para 4 lados. Para tal, escolhe-se o comando **General** da opção **Attributes**. Muda-se o número de lados do polígono regular para 4 e assinala-se a configuração para futuros objetos. Confirma-se as mudanças feitas apontando o botão **OK** (figura 6.3);

- e cria-se o objeto losango na janela de desenho, através de posicionamento (figura 6.4). Caso se deseje alterar a forma do losango, por exemplo, tornando-o mais achatado, basta pegar um dos pegadores com o "mouse" e arrastá-lo até onde desejar, como na figura 6.5.

É necessária também a criação de portas de ligação, pois o nodo Relacionamento só pode ser ligado com outros nodos pelas suas extremidades. Essas portas serão pequenos triângulos nas pontas do losango.

Para fazer a **porta de ligação** da ponta direita do losango:

- escolhe-se o comando **General** da opção **Attributes**, mudando-se o número de lados do polígono regular para 3 e a largura e a altura do envelope inicial para 10;

- cria-se então o pequeno polígono regular, que será a porta, apontando-se na janela de desenho;

- centraliza-se a porta em relação ao losango através do comando **center** da opção **Align**;

- alinha-se o lado direito do envelope da porta com o lado direito do envelope do losango, através do comando **Right to Right** da opção **Align**.

Após estas últimas operações o nodo fica com a forma mostrada na figura 6.6.

Para fazer a **porta de ligação** da ponta esquerda do losango:

- cria-se um polígono regular de 3 lados;

- vira-se o polígono, utilizando o comando **Flip Vertical** da opção

Arrange;

- ajusta-se a porta em relação ao centro do losango, através do comando **Center** da opção **Align**;

- ajusta-se a porta com o lado esquerdo do losango, através do comando **Left to Left** da opção **Align**.

A porta de ligação da ponta de cima do losango é feita pela criação do polígono regular de 3 lados e sua rotação para esquerda. Para isto escolhe-se o comando **Rotate Left** da opção **Arrange**. Para ajustar a porta à ponta de cima do losango, aponta-se um pegador e arrasta-se de modo a arrumar a porta. Alinha-se o objeto em relação ao losango utilizando os comandos **Center** para centralizar, e **Top to Top** da opção **Align** para alinhar com a ponta de cima do losango.

A porta da ponta de baixo do losango é criada de maneira similar, utilizando-se o comando **Rotate Right** da opção **Arrange** para rotacionar o polígono para direita, e os comandos **Center** e **Bottom to Bottom** da opção **Align** para alinhar em relação ao losango.

Ao final destas operações o nodo "relacionamento" toma a forma mostrada na figura 6.7.

Após a definição gráfica do nodo, é necessário definir os atributos dos objetos que formam o nodo. Para abrir a caixa de diálogo de definição de atributos dos objetos, seleciona-se um objeto, por exemplo, o losango. Escolhe-se então o comando **Object** da opção **Attributes**. Aparecerá na tela a caixa de diálogo correspondente aos atributos do losango. Dá-se um nome ao objeto como, por exemplo, "Losango do Relacionamento". Define-se o número mínimo e máximo de conexões, no exemplo 0 e 0, pois o losango não pode ser conectado a nenhum objeto. Define-se o losango como sendo um objeto fixo. Confirma-se as definições com um **OK** (figura 6.8).

Os atributos das portas de ligação são definidos de maneira similar, sendo que as portas possuem número mínimo de conexões igual a 0 e número máximo de conexões igual a 32767.

Resta então a definição dos outros nodos do modelo E-R. O nodo Entidade é definido com um único objeto, um retângulo. O nodo Dependência Funcional é definido com dois retângulos. Depois de definir os atributos dos objetos deste nodo, passa-se a definição das pontas do método. O Modelo E-R possui apenas um tipo de ponta, que chama-se "Seta". A figura 6.9 mostra a forma gráfica destes três elementos.

Após a criação da ponta, cria-se os conectores do método, que são de dois tipos : um com ponta, outro sem nenhuma ponta. O conector sem ponta chama-se "arco" e o com ponta chama-se "arco dependente". Na figura 6.10 é mostrada a definição do arco "arco dependente".

As regras de ligação são definidas escolhendo-se o comando **Conect Rules** da opção **Edit**. Para definir uma ligação, seleciona-se os nodos e o conector que participam dela (figura 6.11). Realiza-se a ligação apontando-se no objeto do nodo origem, arrastando o cursor até o objeto no nodo destino (figura 6.12). Deste modo são definidas todas as ligações possíveis entre os diversos nodos (figura 6.13).

A nomeação do método é feita selecionando-se o comando **Save as** da opção **File**. Escreve-se o nome desejado e seleciona-se no botão **Save** (figura 6.14).

6.1.2 Utilizando o EDE

Para criar um novo diagrama no **EDE**, o usuário deve escolher o comando **New** da opção **File**. O **MED** abre a janela de escolha dos métodos existentes, como mostra a figura 6.15. Escolhe-se o método Modelo E-R e seleciona-se o botão "Open". Uma nova janela chamada "untitled", que corresponde ao novo diagrama, é aberta (ver figura 6.16).

Seleciona-se a janela "Node" para torná-la ativa e então aponta-se no nodo "Entidade" (figura 6.17). Para criar uma nova entidade, aponta-se na janela do diagrama (figura 6.18). Esta nova entidade será chamada "EMPLOYEE". Para tal, é necessário criar um texto com esse nome dentro dela. Escolhe-se o comando **Turn On** da opção **Text**, o que provoca a abertura de uma região de texto dentro do nodo. Digita-se o nome do nodo, podendo-se centralizá-lo escolhendo o comando **Center** da opção **Justify** do cardápio (figura 6.19). Escolhe-se o comando **Turn Off** da opção **Text** e a nova entidade está pronta (figura 6.20). Do mesmo modo cria-se uma entidade chamada **PROJECT** e um relacionamento entre as duas entidades chamado **PROJECT-WORKER** (figura 6.21). Os três nodos criados são alinhados usando-se os comandos da opção **Align** (figura 6.22).

Para realizar a ligação entre a entidade "EMPLOYEE" e o relacionamento "PROJECT-WORKER", seleciona-se o conector "Arco" na janela "Edge" (figura 6.23). Realiza-se a ligação apontando-se a entidade "EMPLOYEE" e arrastando o cursor até a ponta do relacionamento "PROJECT-WORKER" (figura 6.24). A ligação obtida é mostrada na figura 6.25. Como o relacionamento entre as entidades é de "M" trabalhadores para "N" projetos, cria-se os rótulos "M" e "WORKER" associados ao arco entre "EMPLOYEE" e "PROJECT-WORKER", escolhendo-se o comando **Turn On** da opção **Text** e apontando na janela do diagrama (figura 6.26). Escreve-se os textos desejados e escolhe-se o comado **Turn Off** da opção **Text** (figura 6.27). Do mesmo modo cria-se a ligação entre o relacionamento e a entidade "PROJECT", com os rótulos "N" e "PROJECT". O diagrama completo é mostrado na figura 6.28. Basta então gravá-lo escolhendo-se o comando **Save As** da opção **File** do cardápio.

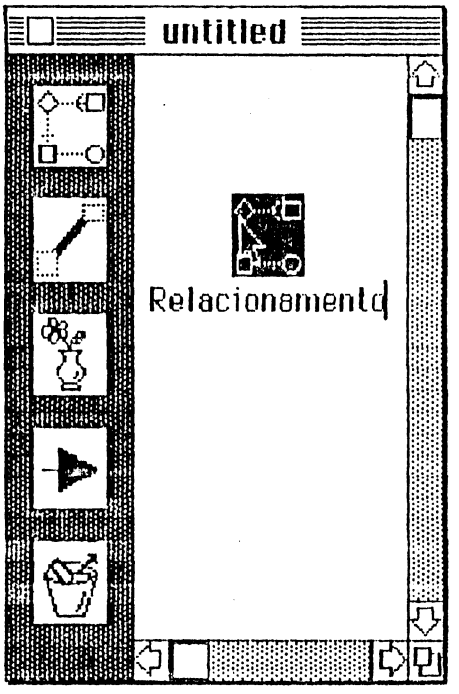


Figura 6.1 - Edição do nome de um tipo de nodo.

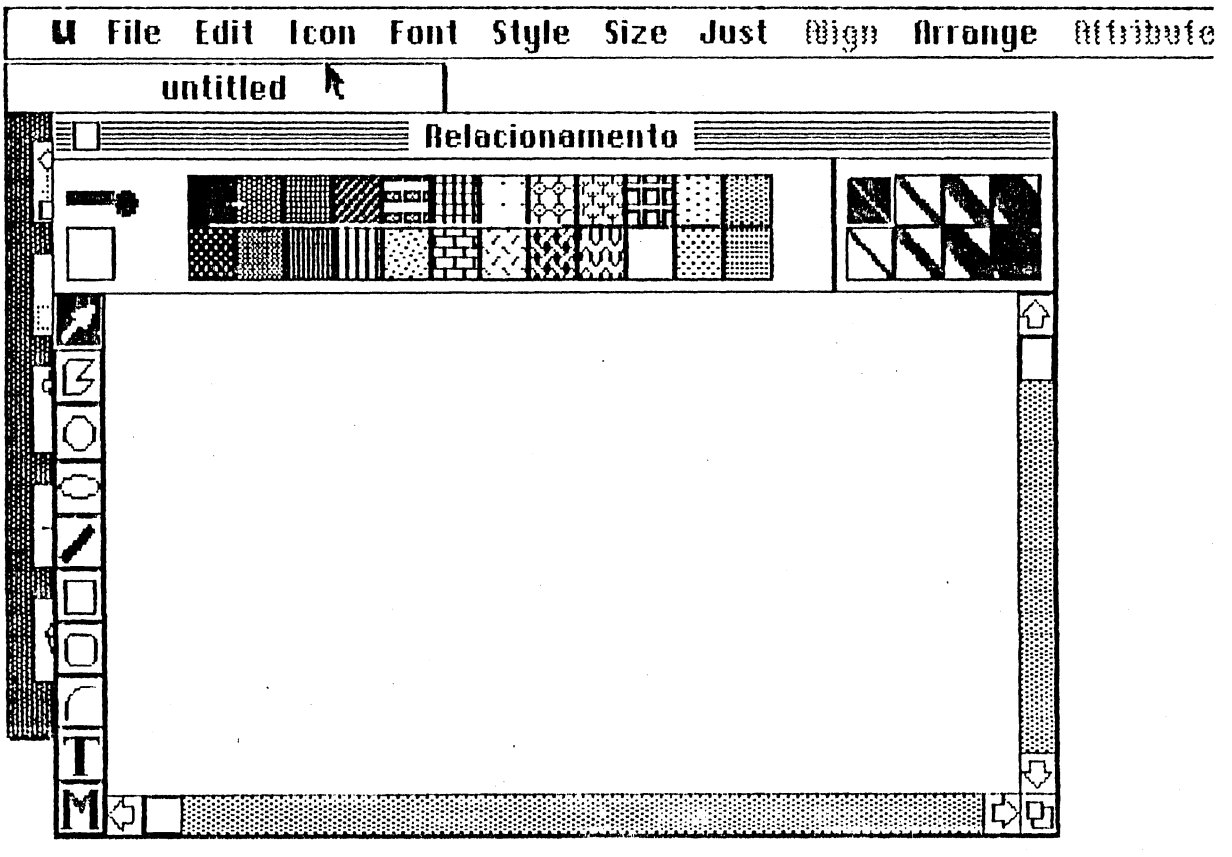


Figura 6.2 - Janela aberta pela ativação do comando **OPEN** da opção **ICON**.

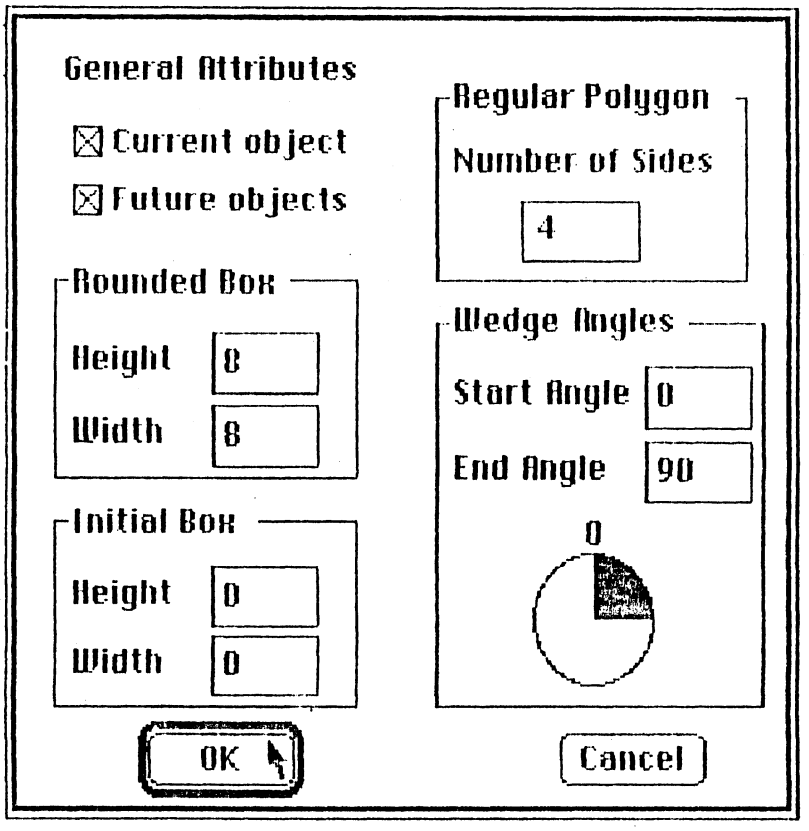


Figura 6.3 - Fixação de quatro lados para os polígonos regulares.

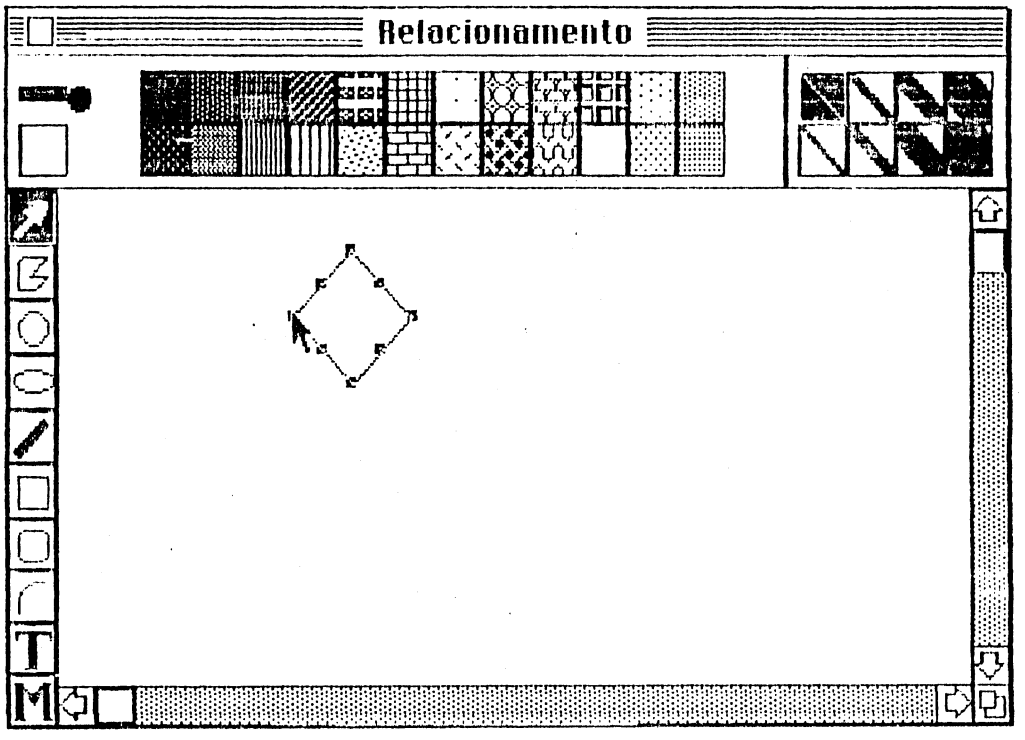


Figura 6.4 - Criação de um polígono regular de quatro lados.

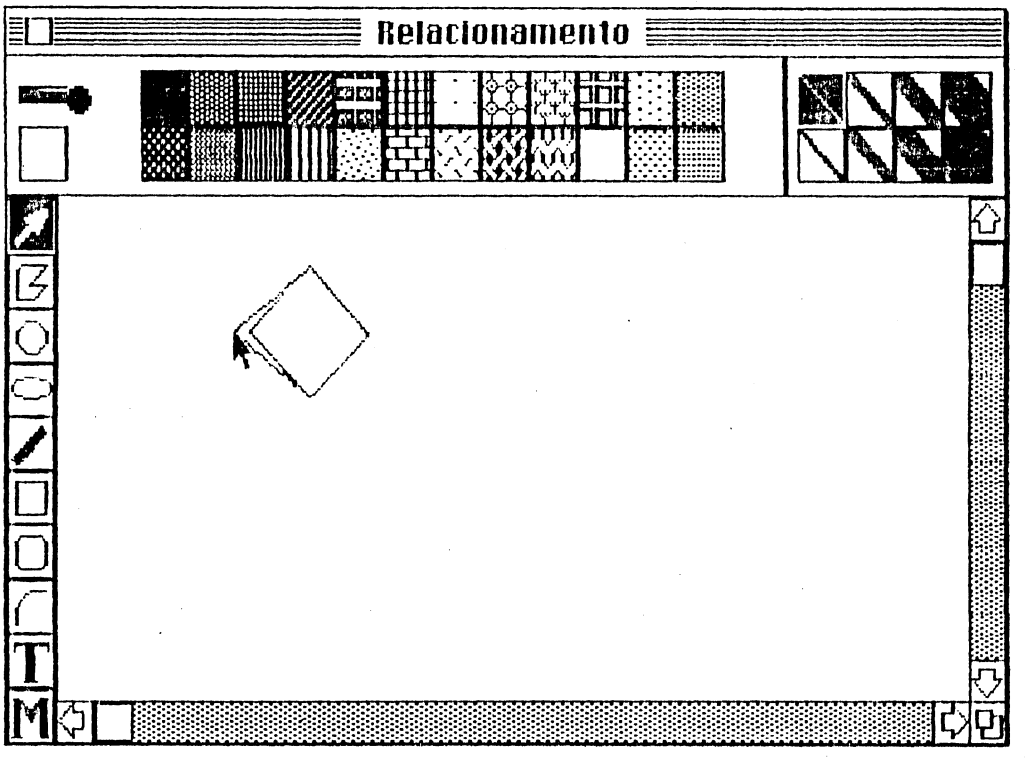


Figura 6.5 - Alteração das dimensões de um polígono.

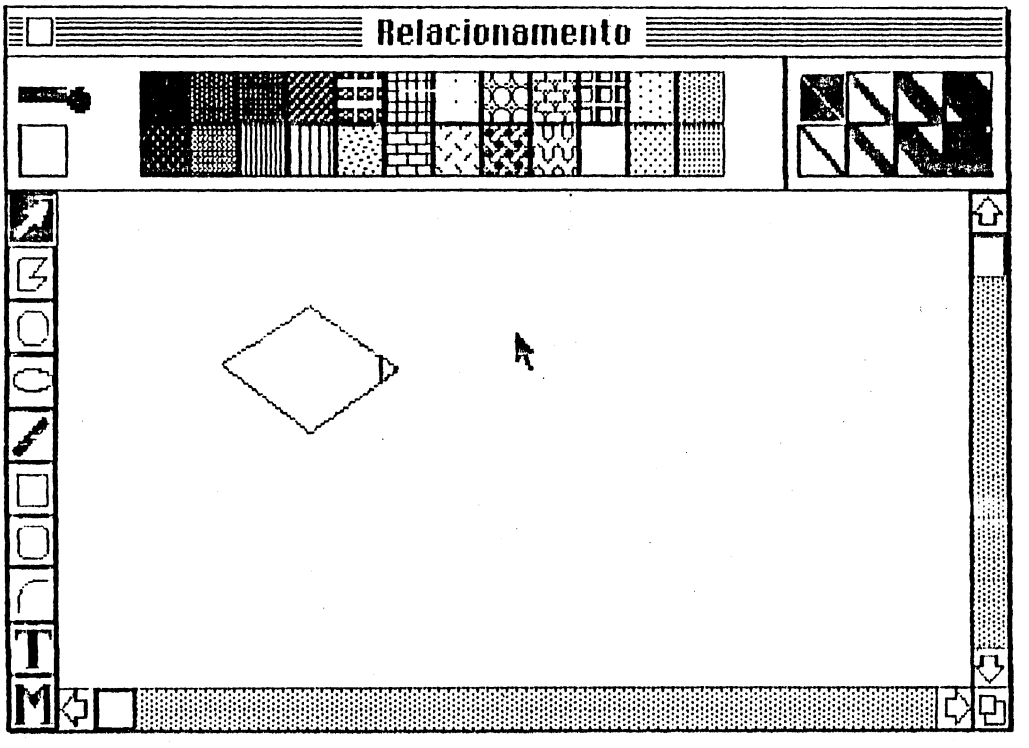


Figura 6.6 - Nodo "Relacionamento" após a colocação da primeira porta de ligação.

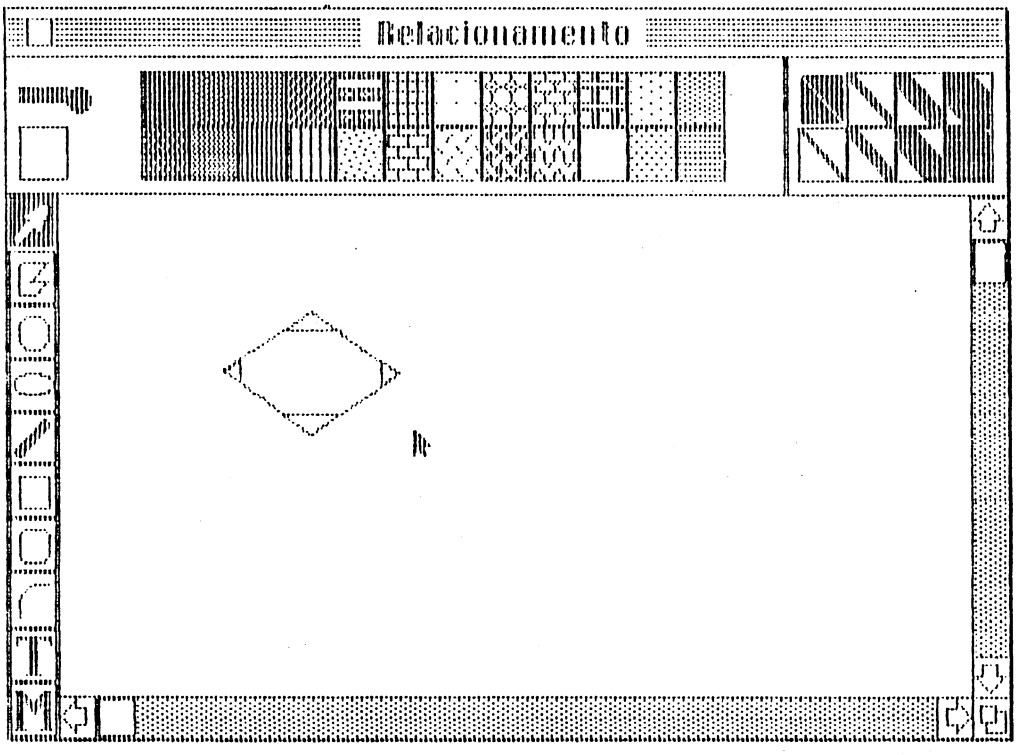


Figura 6.7 - Nodo "Relacionamento" após a colocação das quatro portas de ligação.

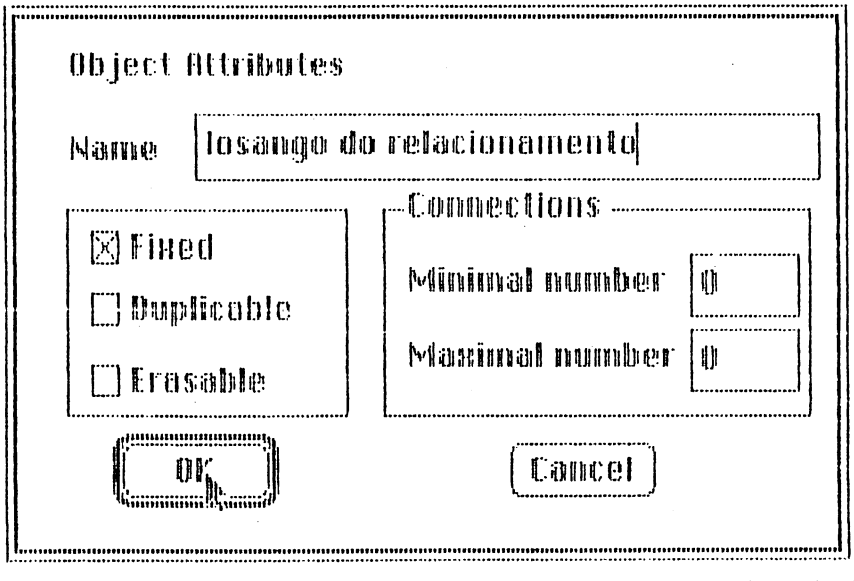


Figura 6.8 - Definição dos atributos do losango formador do nodo "Relacionamento".

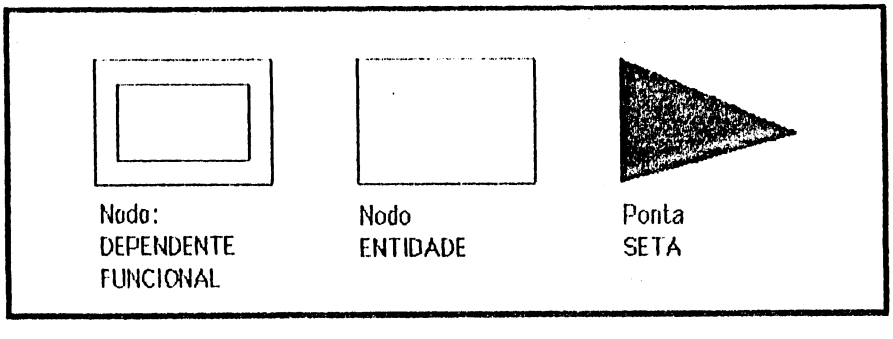


Figura 6.9 - Elementos do método Entidade-Relacionamento.

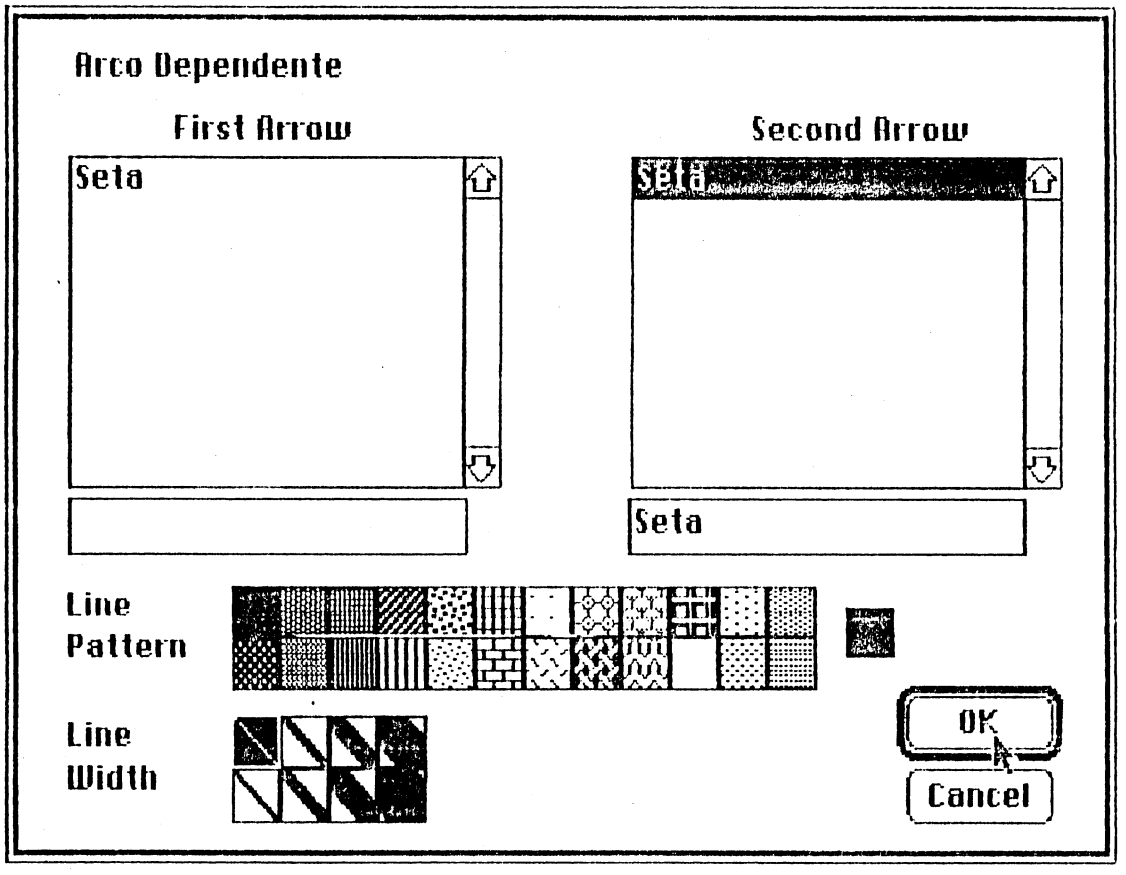


Figura 6.10 - Definição do "Arco Dependente".

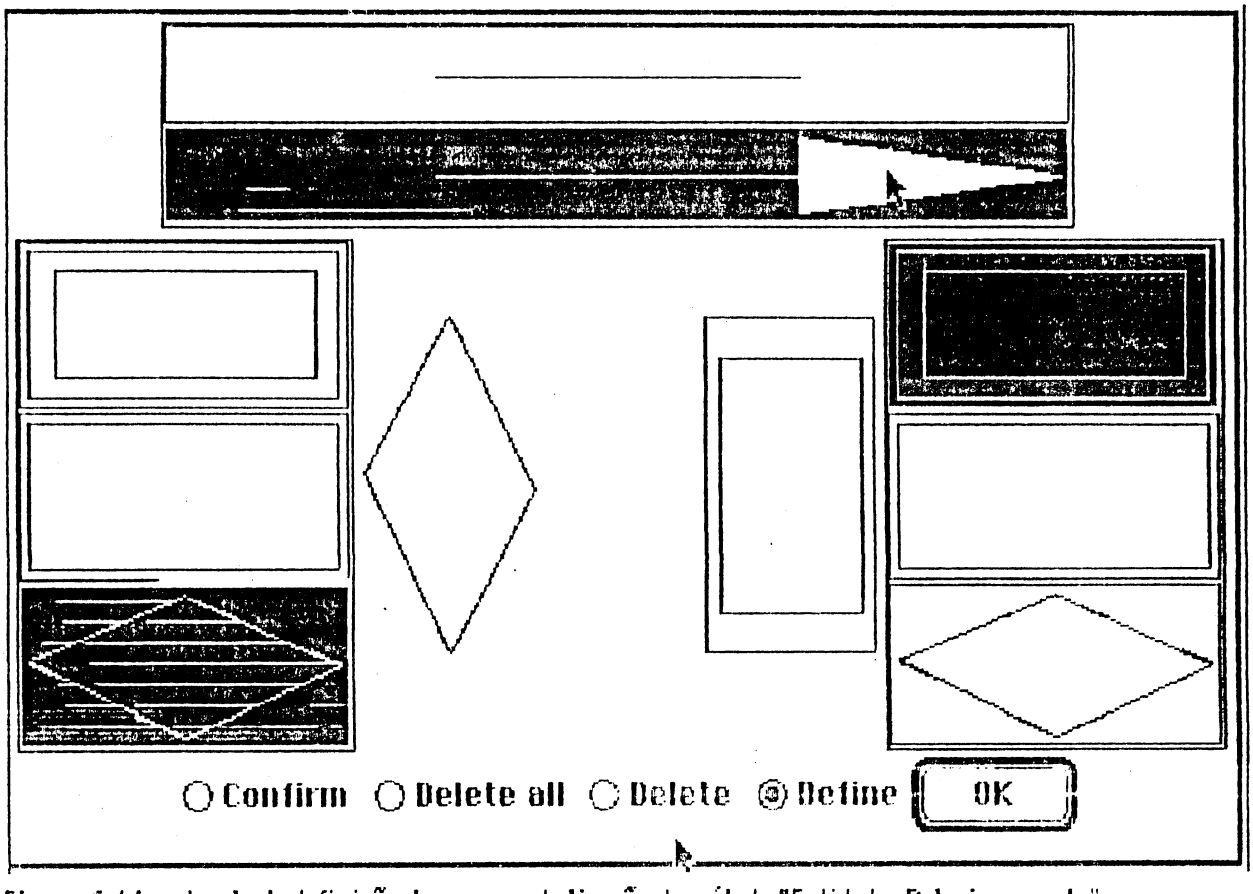


Figura 6.11 - Janela de definição das regras de ligação do método "Entidade-Relacionamento".

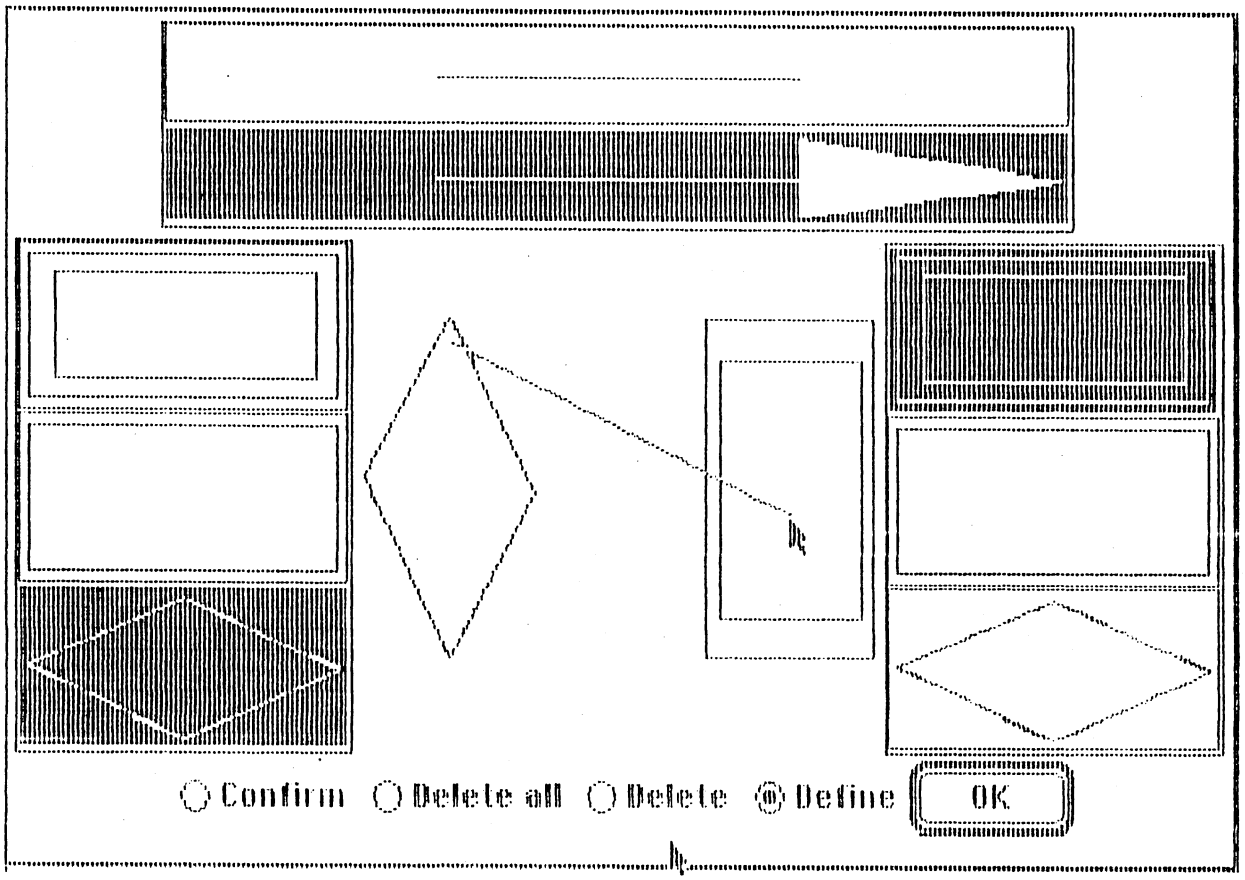


Figura 6.12 - Confecção de uma regra de ligação.

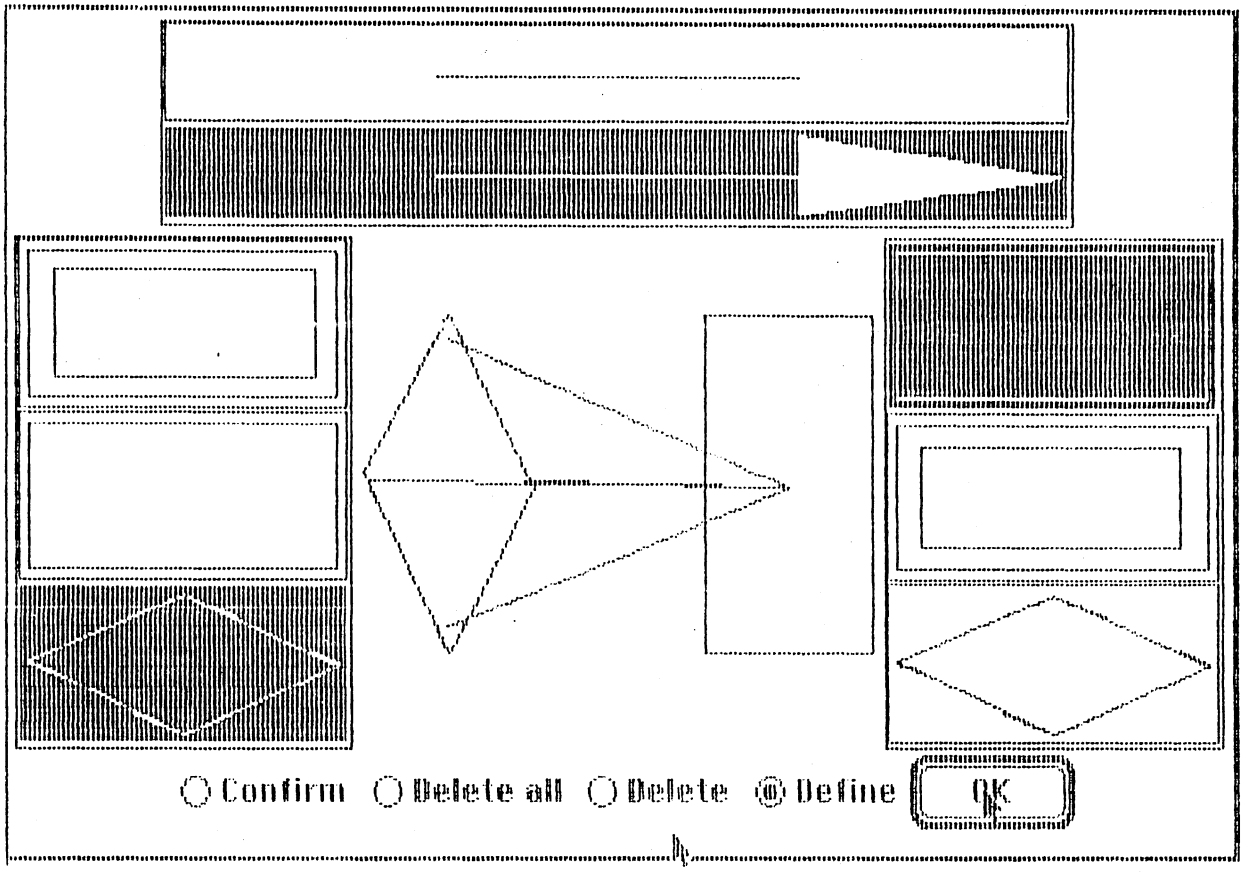


Figura 6.13 - Tipos de ligações entre os nodos "entidade" e "relacionamento".

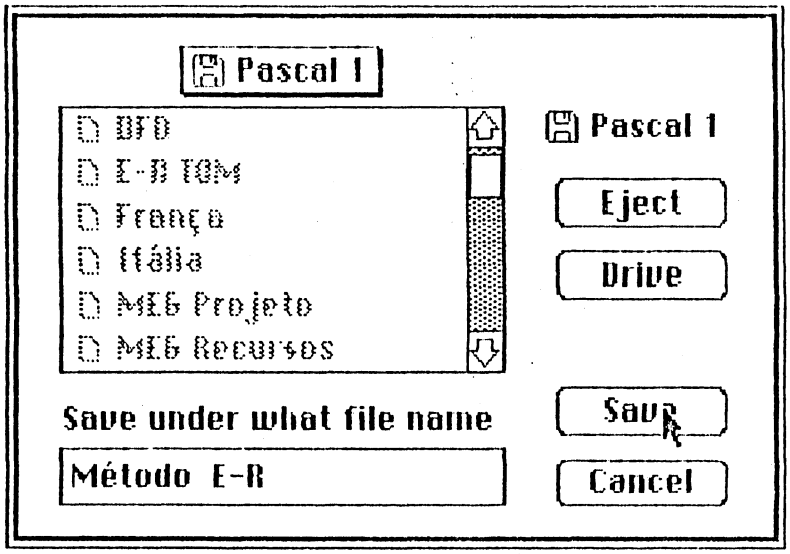


Figura 6.14 - Gravação do método "Entidade-Relacionamento".

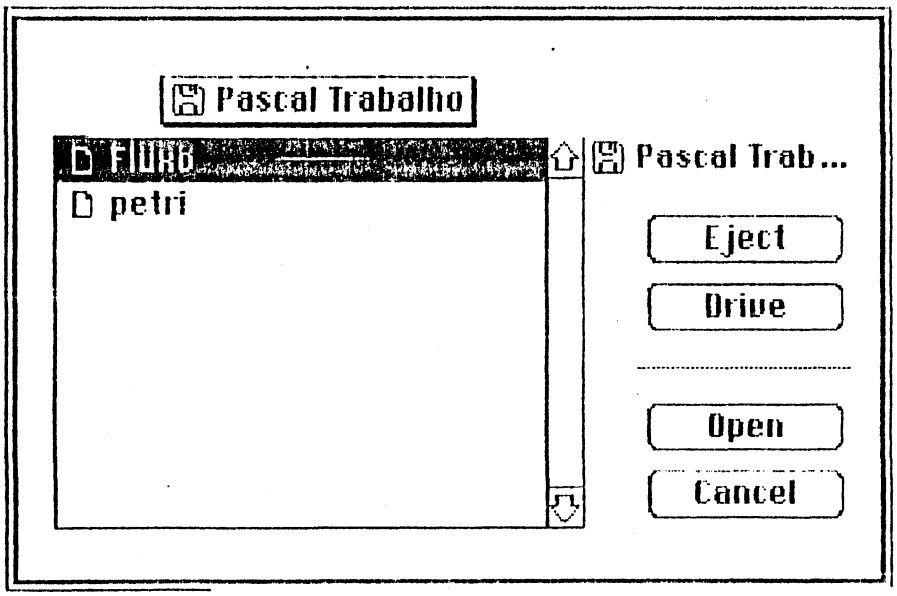


Figura 6.15 - Solicitação de criação de um novo diagrama.

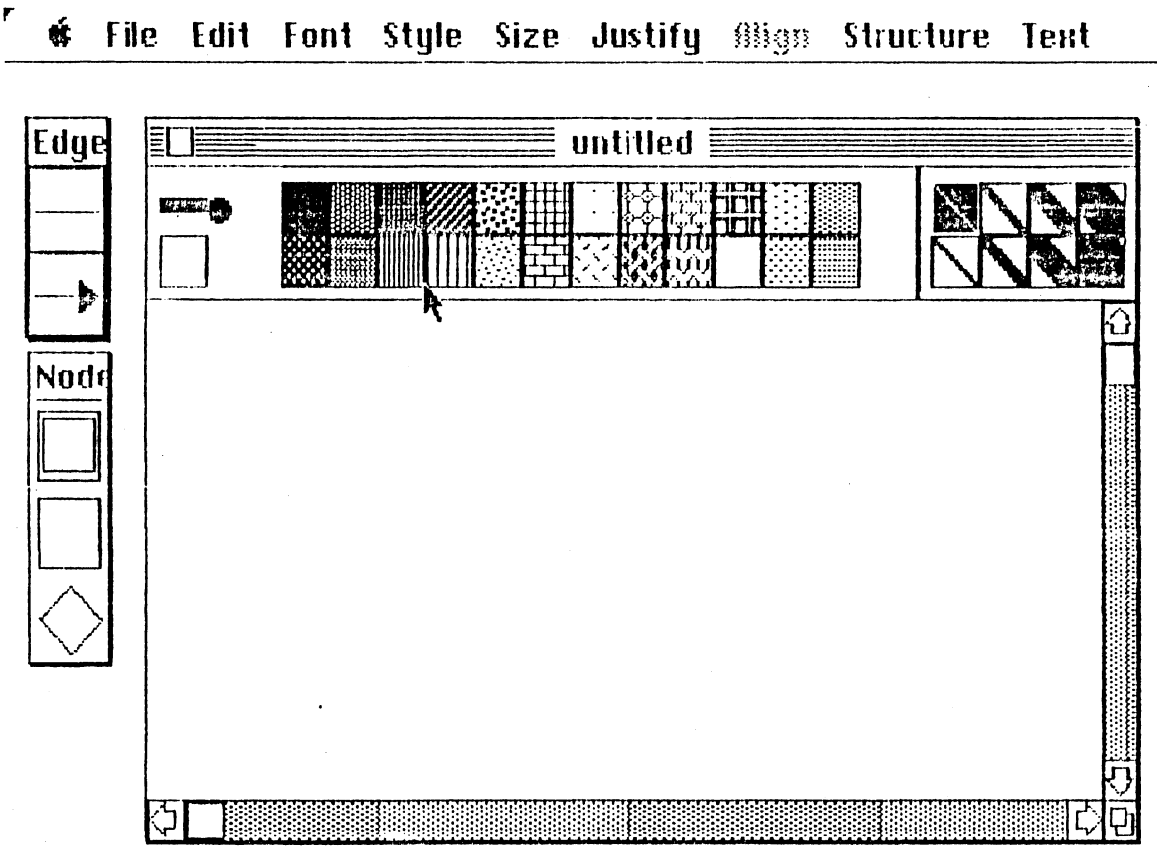


Figura 6.16 - **EDE** no estado de edição de um novo diagrama.

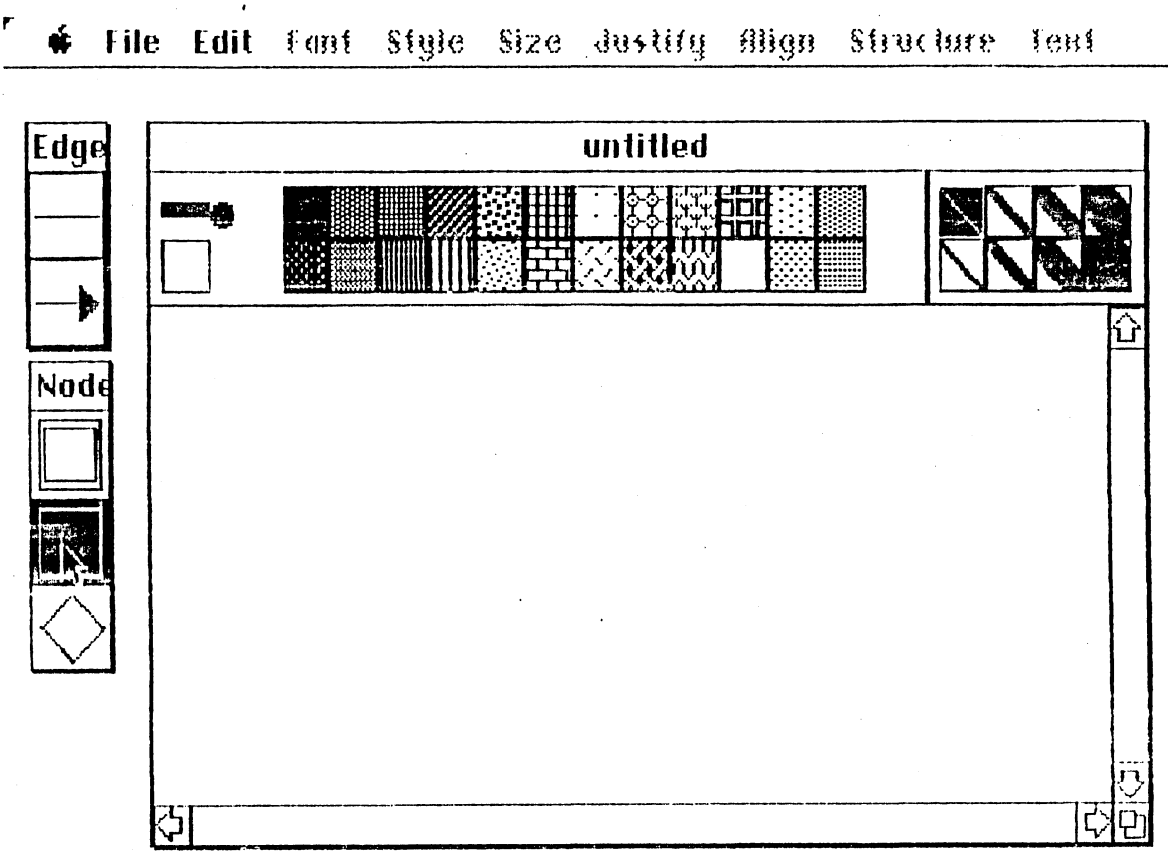


Figura 6.17 - Seleção do nodo "entidade" na janela de tipos de nodos.

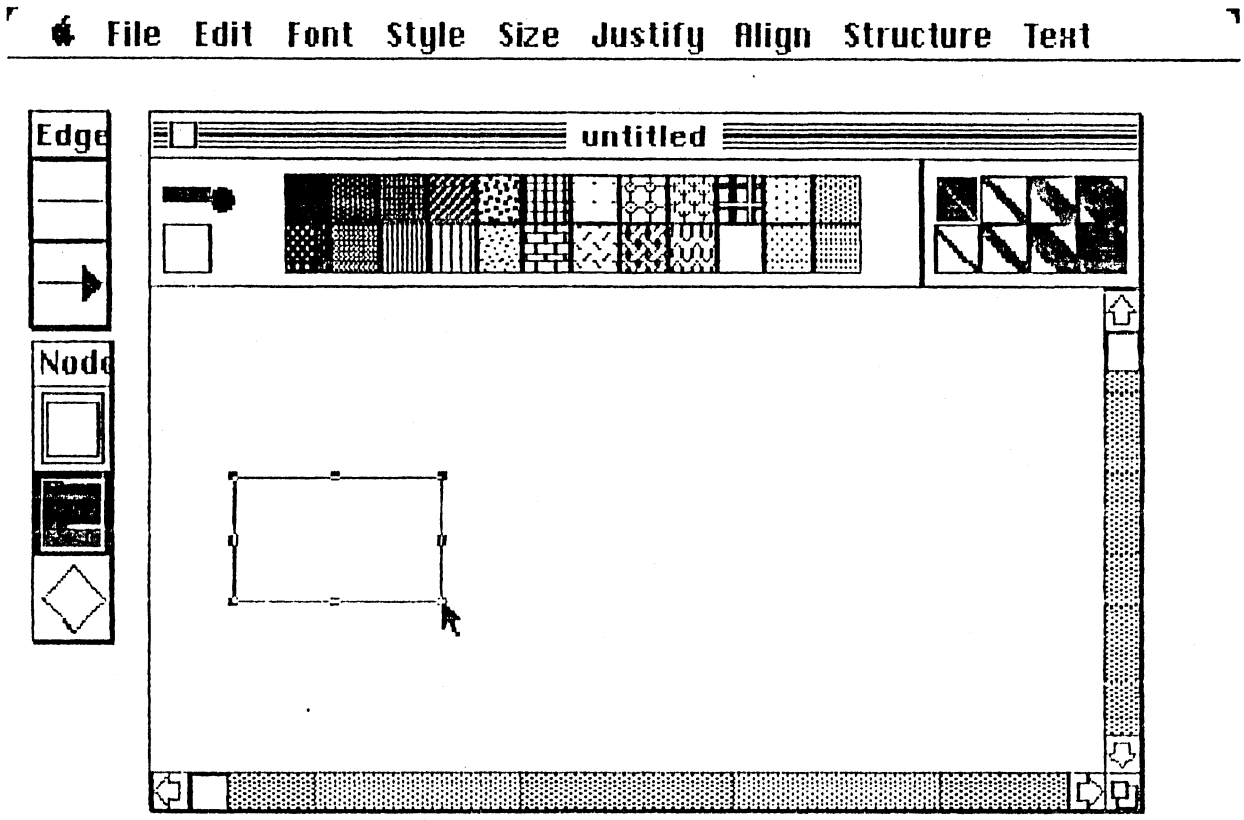


Figura 6.18 - Criação de um nodo entidade na janela de desenho.

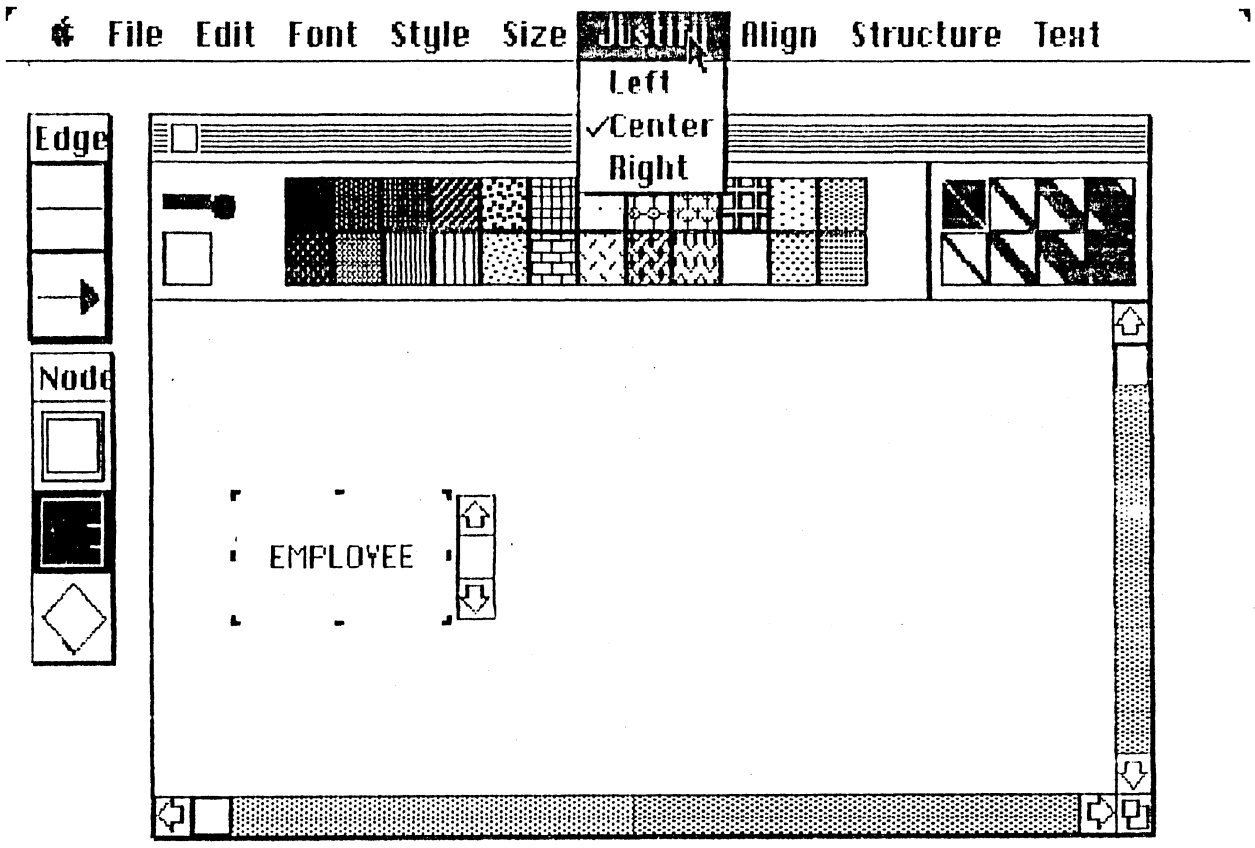


Figura 6.19 - Edição de texto em um nodo do tipo entidade.

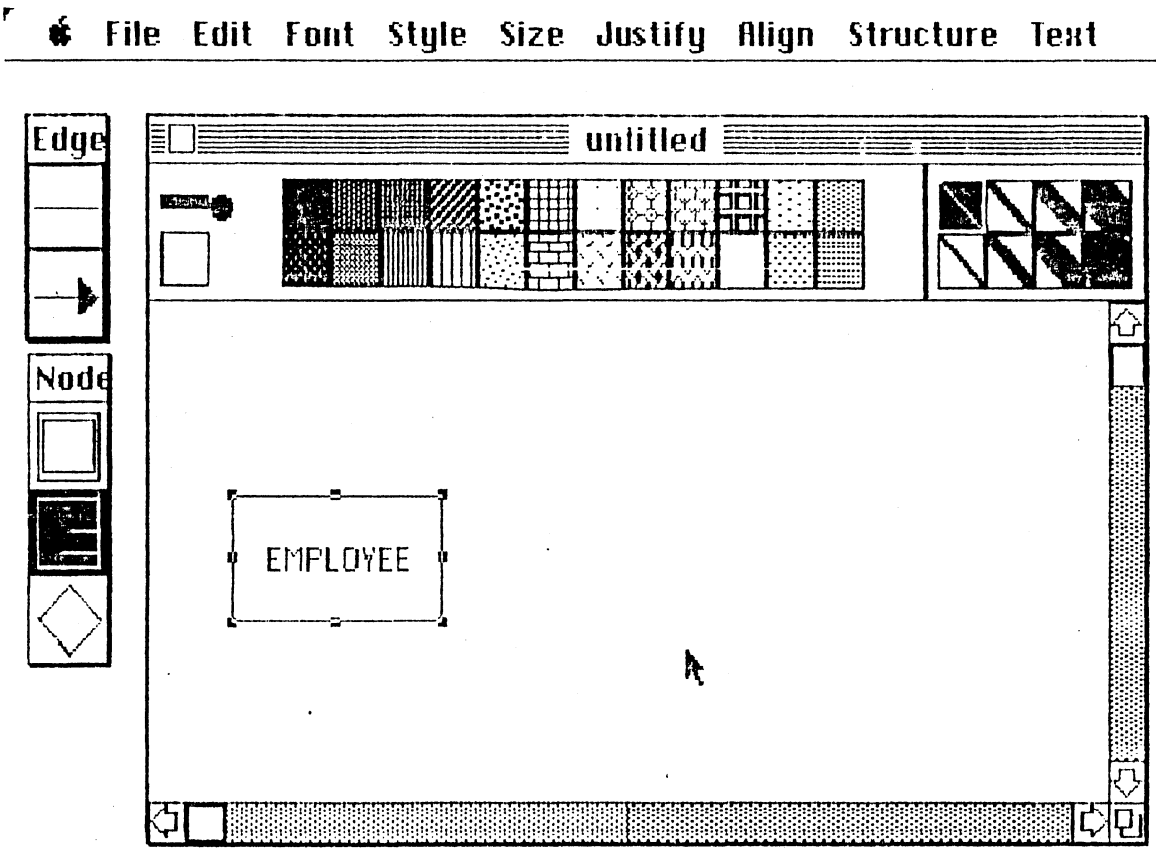


Figura 6.20 - Nodo do tipo entidade após a edição de texto.

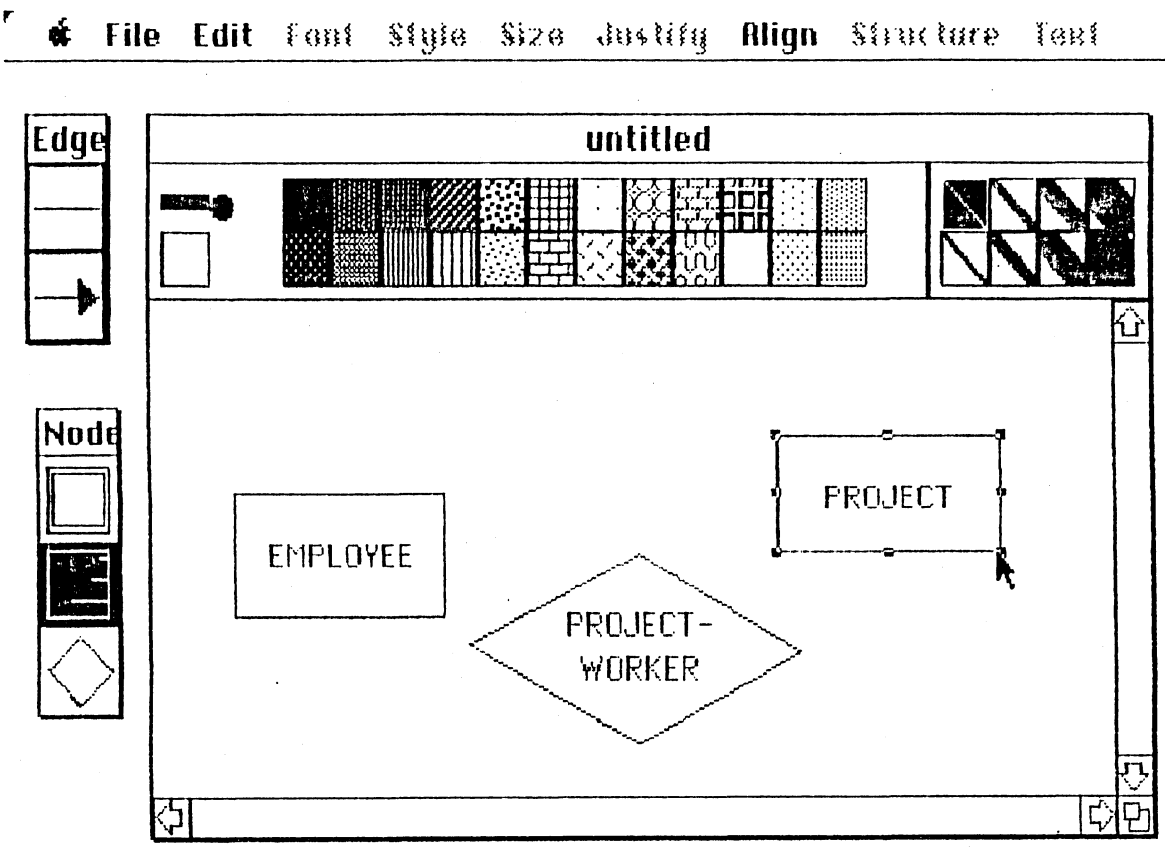


Figura 6.21 - Diagrama com três nodos criados.

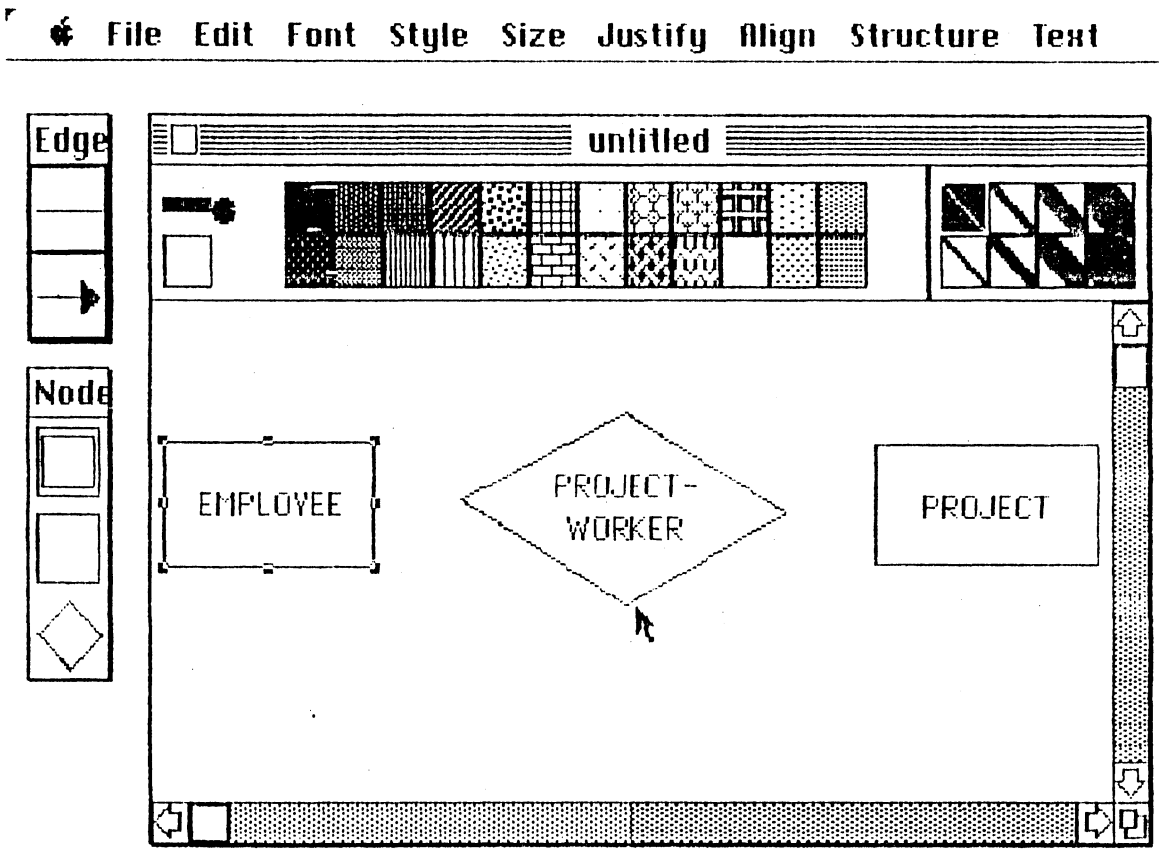


Figura 6.22 - Diagrama com os nodos alinhados horizontalmente.

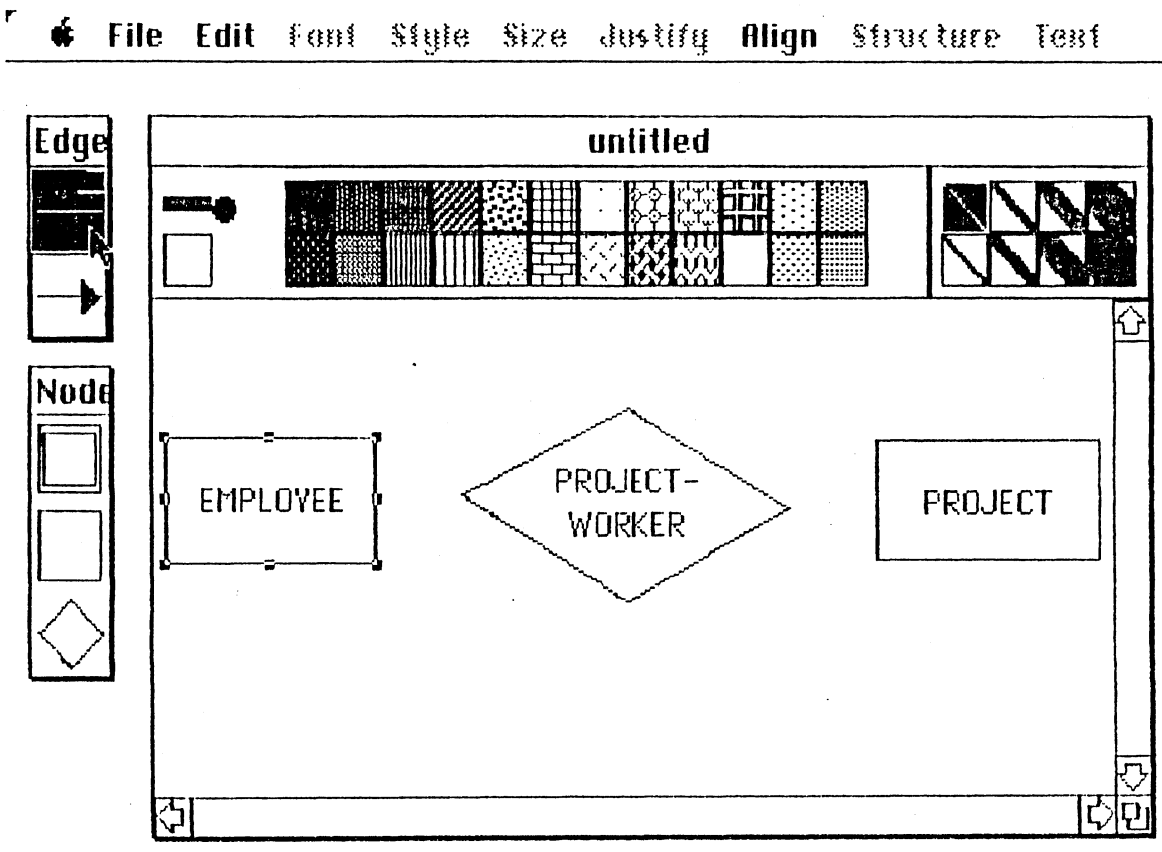


Figura 6.23 - Seleção do arco "arco" na janela de tipos de arcos.

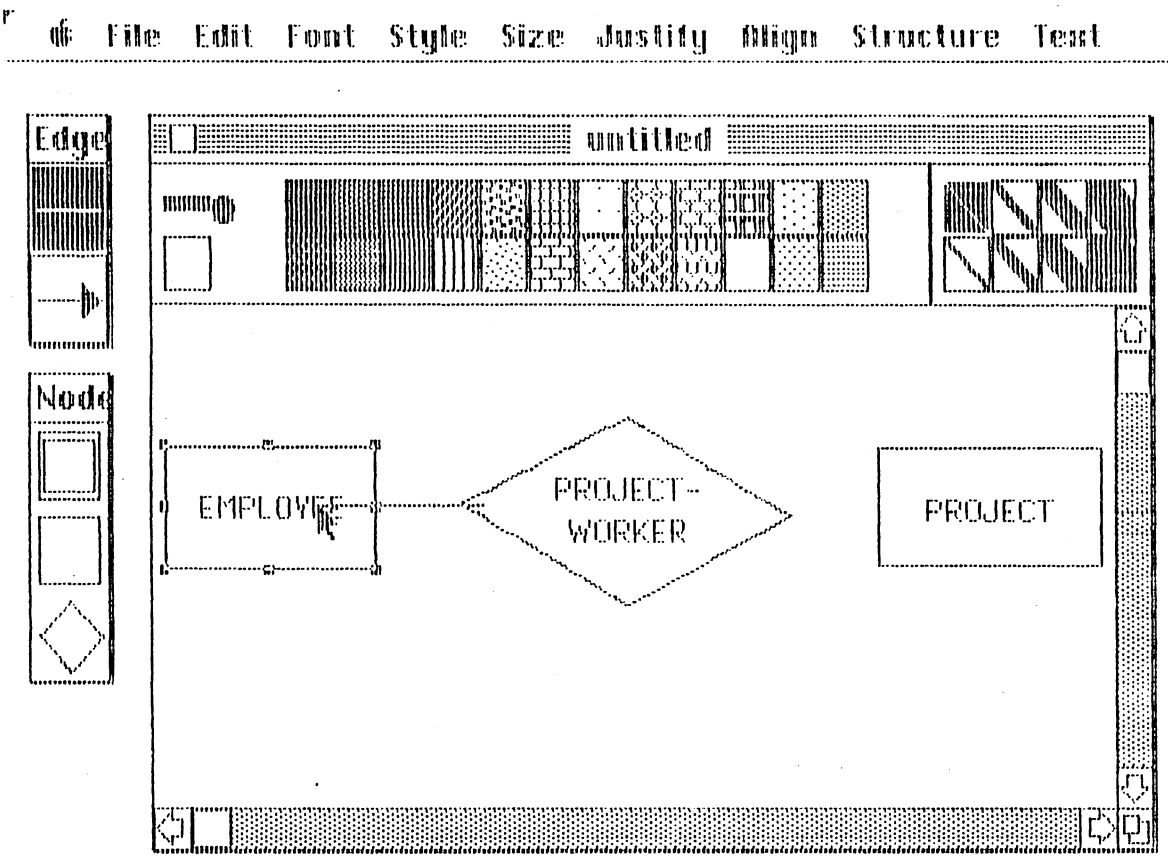


Figura 6.24 - Solicitação de conexão entre os nodos "employee" e "project-worker" através de um arco do tipo "arco".

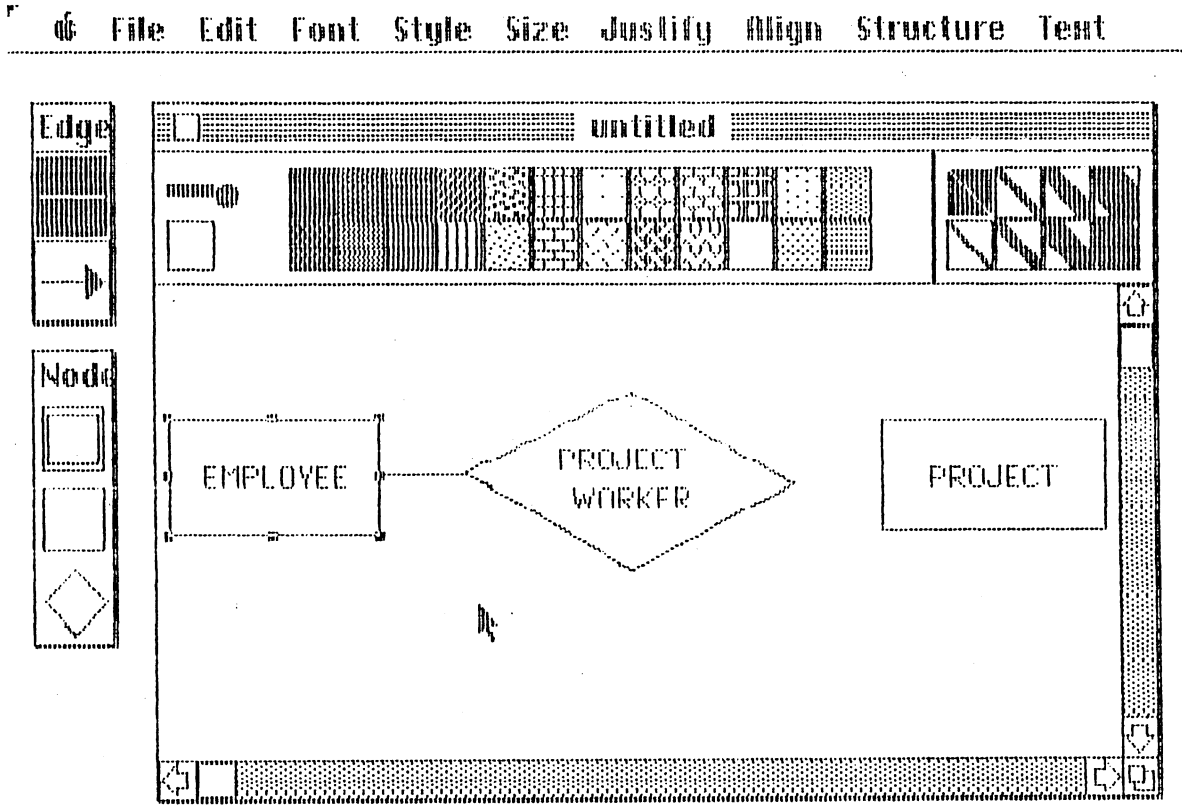


Figura 6.25 - Efetivação da conexão entre os nodos "employee" e "project-worker" através de um arco do tipo "arco".

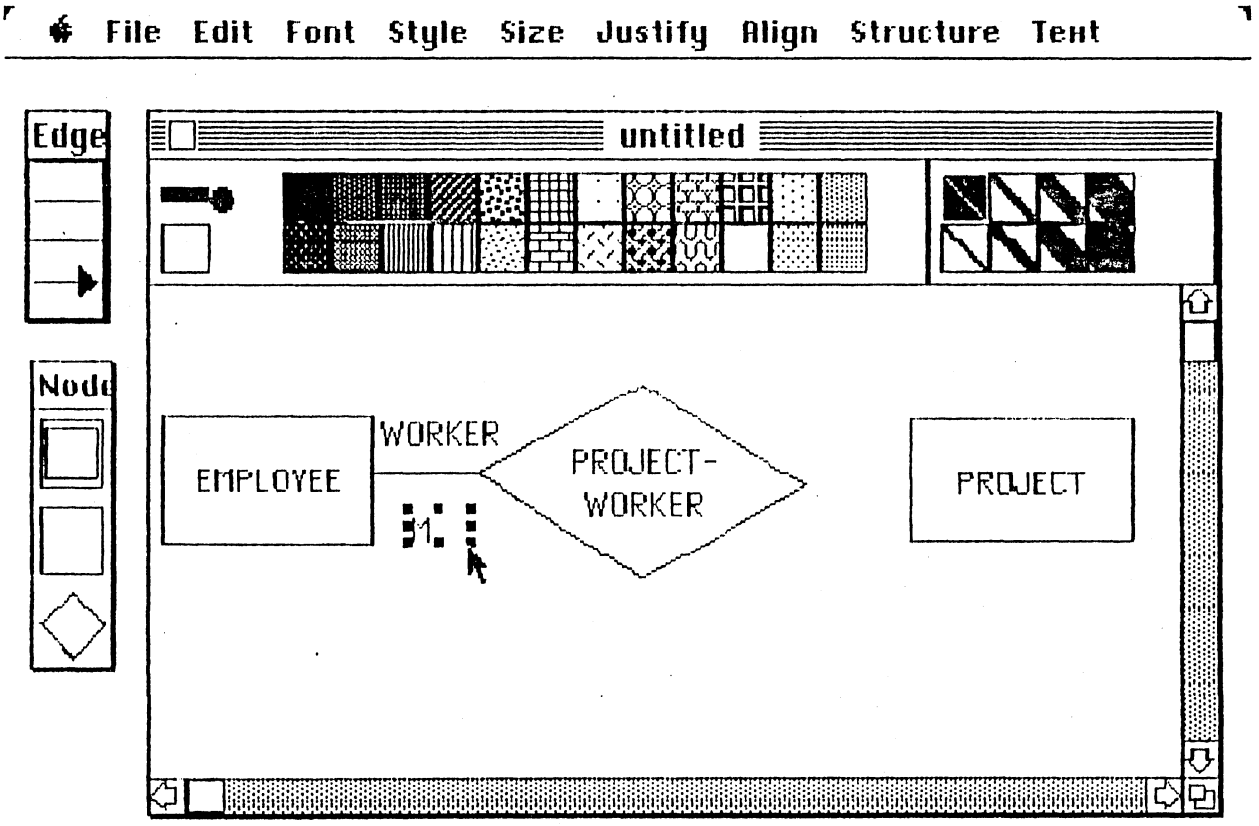


Figura 6.26 - Criação de rótulos vinculados a ligações.

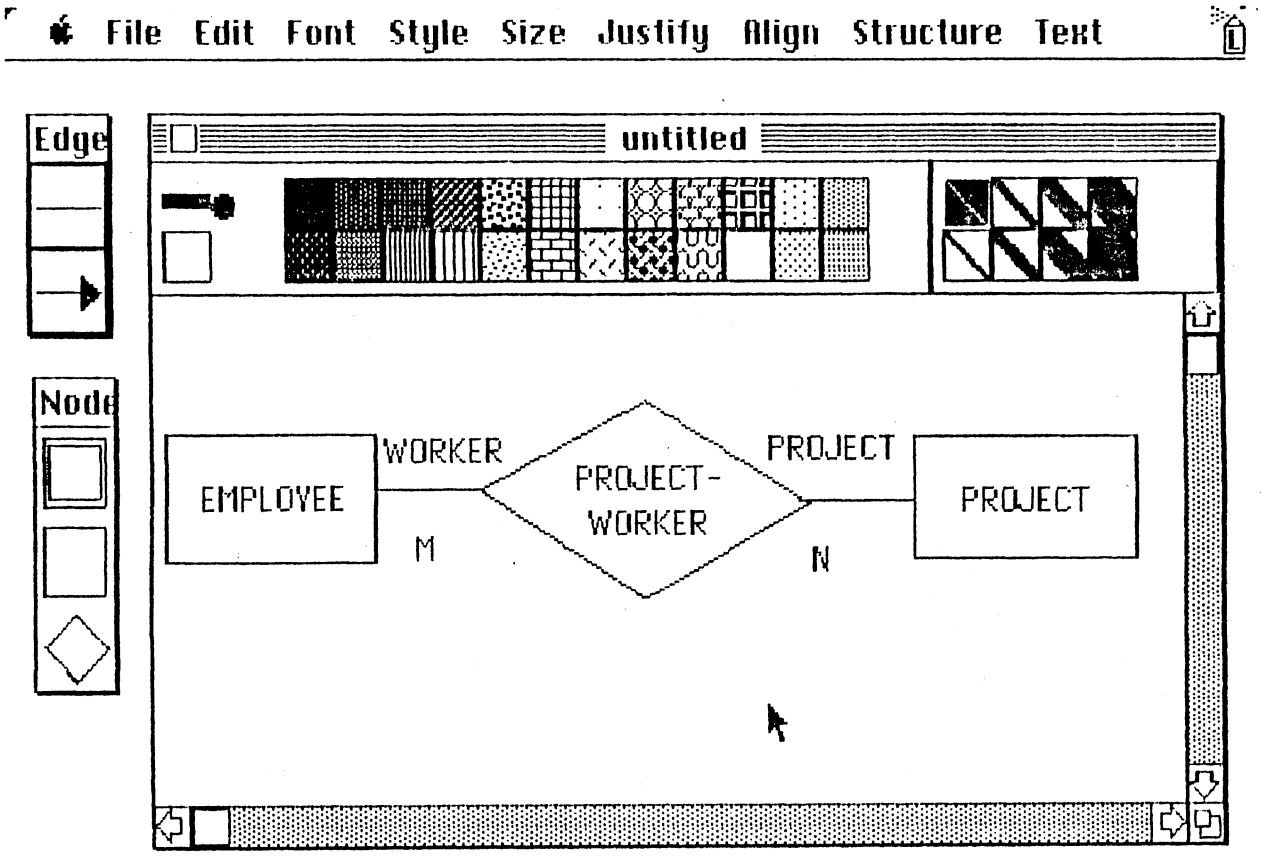


Figura 6.27 - Um diagrama após a edição dos textos e ligações dos nodos envolvidos.

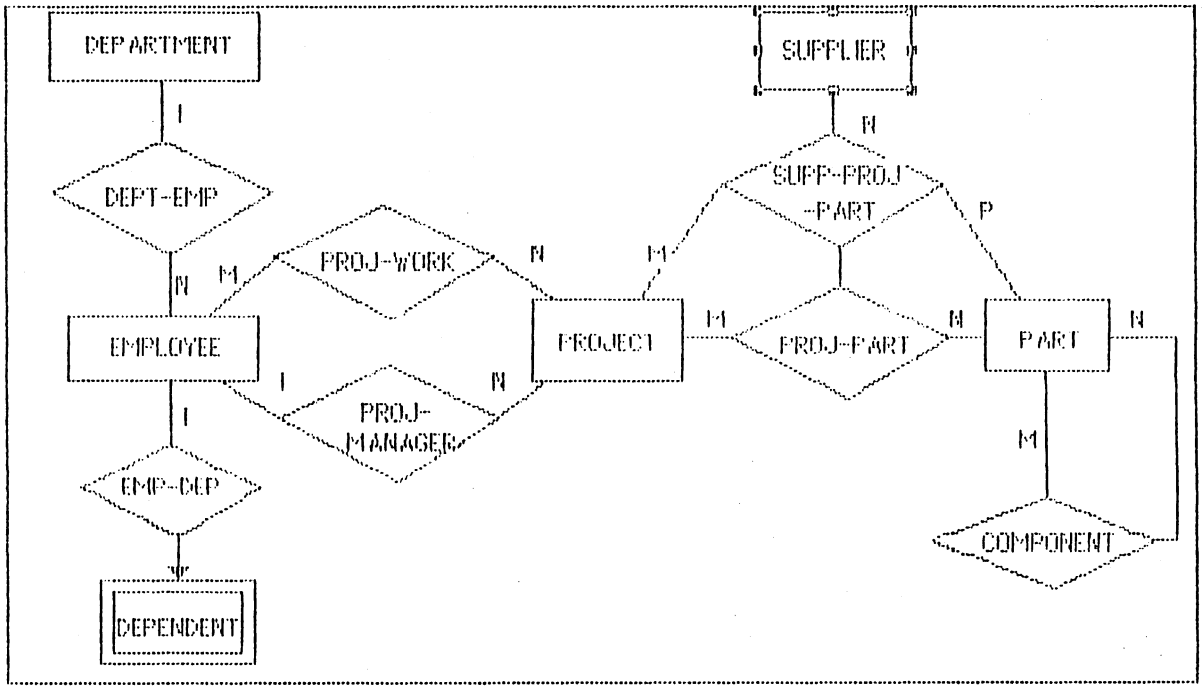


Figura 6.28 - Um diagrama de "Entidade-Relacionamento" de uma empresa fictícia (exemplo retirado de [CHE 77]).

6.2 Segundo Exemplo: Redes Marcadas

As Redes Marcadas [HEU 87] são um tipo de Redes de Petri. Elas possuem três tipos de nodos: lugar, transição e asserção estática. A figura 6.29 mostra a forma gráfica destes três tipos de nodos.

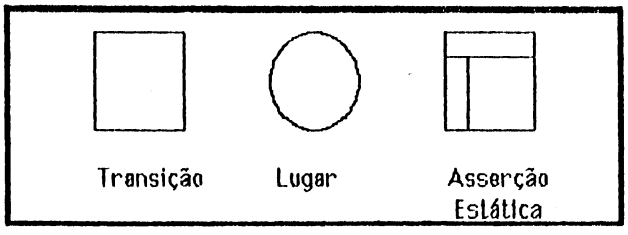


Figura 6.29 - Tipos de nodos das redes marcadas [HEU 87]. No **MED** o nodo "asserção estática" é formado por três retângulos.

As redes marcadas possuem quatro tipos de arcos: porta alteradora de entrada, porta alteradora de saída, porta restauradora de entrada e porta restauradora de saída. A figura 6.30 mostra a definição do tipo de arco "porta restauradora de saída". A definição da ponta "porta restauradora" é mostrada na figura 6.31. Para formar esta ponta o usuário utilizou dois poligonos regulares, para formar as setas, e um segmento de reta.

Como regra de ligação entre os nodos, nas redes marcadas somente são permitidas ligações entre os nodos do tipo transição com lugares e entre os nodos do tipo asserção estática com lugares. A figura 6.32 mostra um exemplo de definição destas regras.

A figura 6.33 mostra um exemplo de edição de um diagrama deste tipo de redes no **EDE**.

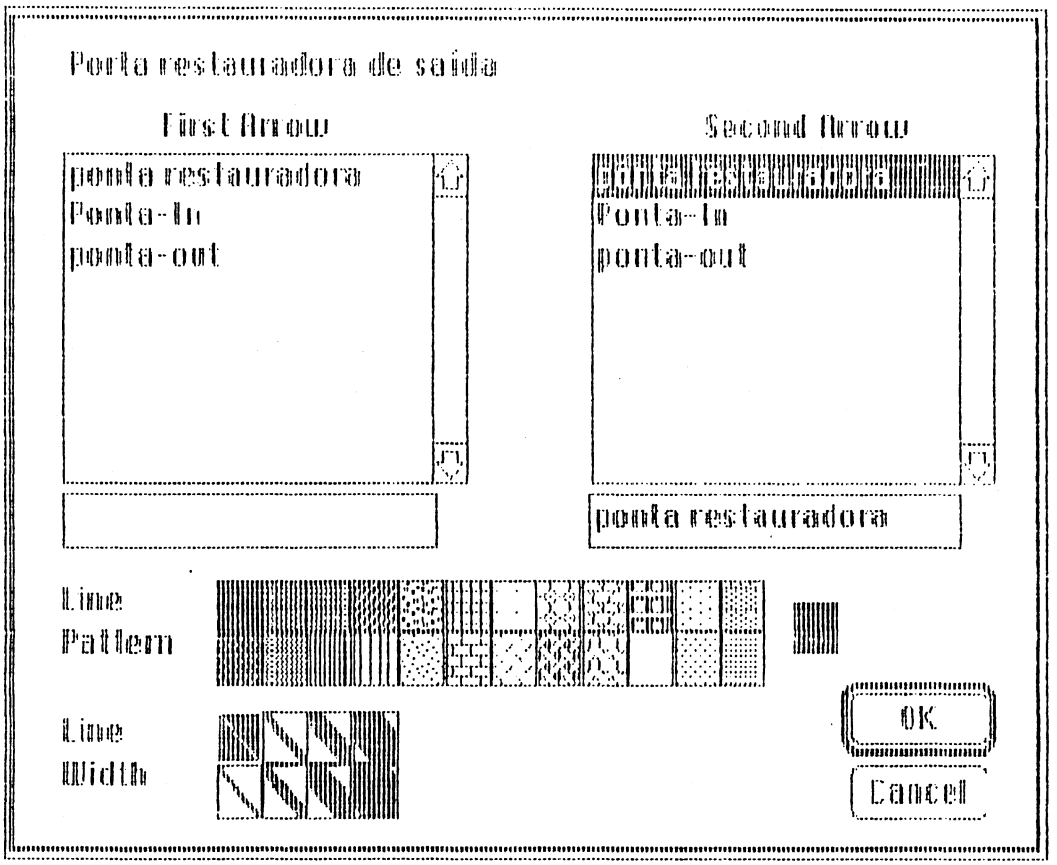


Figura 6.30 - Definição do tipo de arco: "porta restauradora de saida".

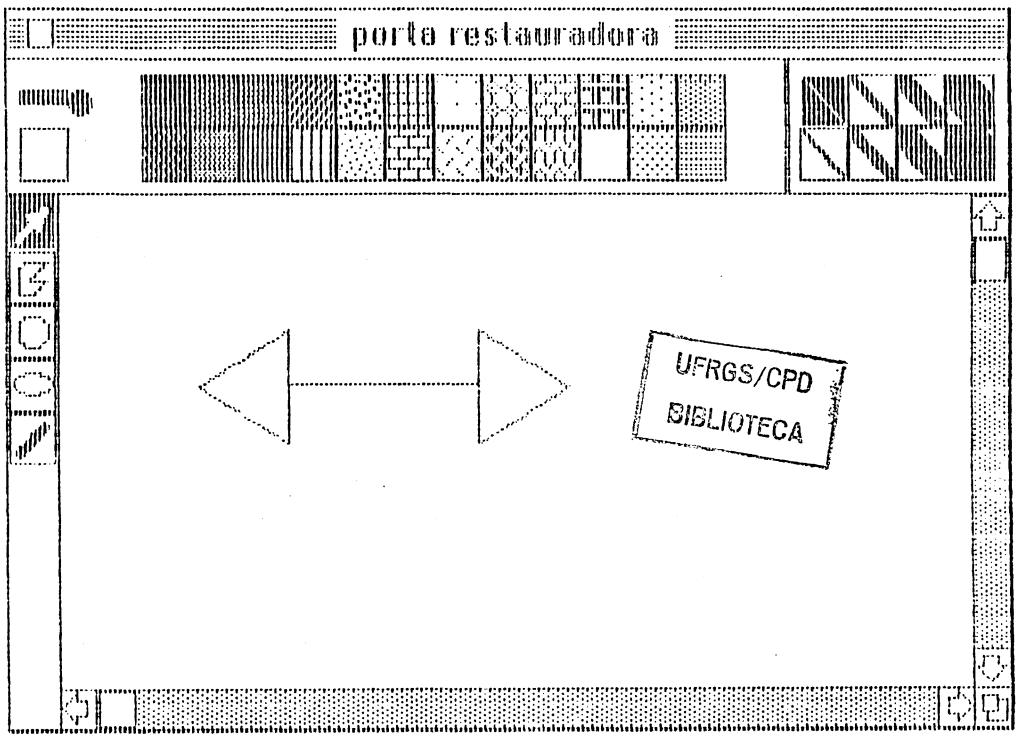


Figura 6.31 - Definição do tipo de ponta "porta restauradora".

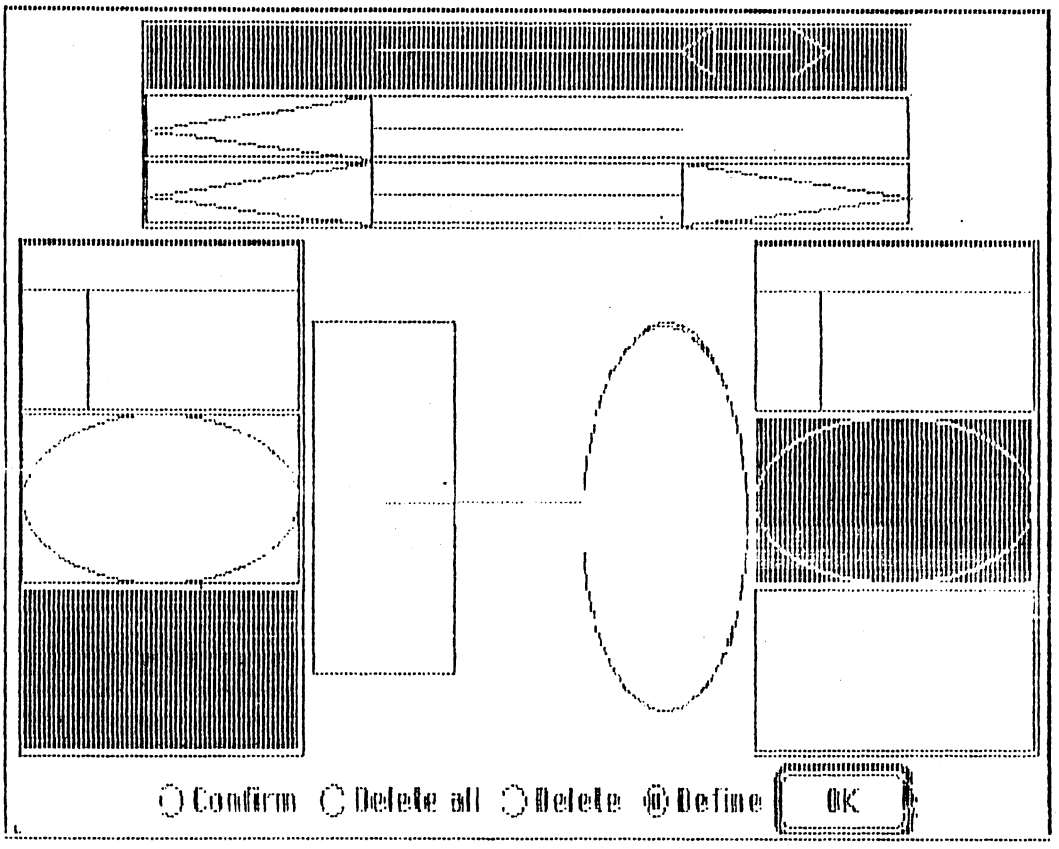


Figura 6.32 - Definição das regras de ligação ente os nodos das redes marcadas.

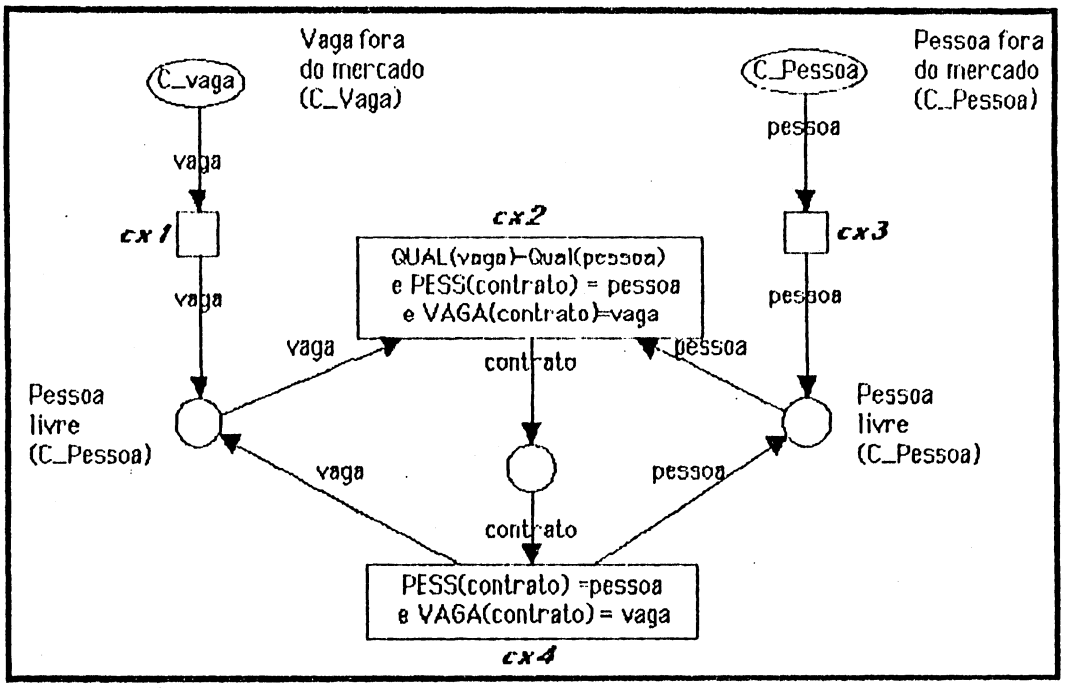


Figura 6.33 - Exemplo de edição de um diagrama do método "redes marcadas" (exemplo retirado de [HEU 87]).

6.3 Terceiro Exemplo: O Método Transporte

Supondo a existência de uma método possuidor de três nodos: armazém, fábrica e supermercado; três decorações: navio, trem e caminhão; duas pontas, a ponta "seta-in" e "seta-out"; e dois arcos: arco "vai" e o arco "vai-e-vem". Supondo, ainda, que o nodo supermercado somente pode ser ligado com o nodo armazém, e este somente com o nodo fábrica. O **EDG** possibilita a um engenheiro de software desenhar os nodos e decorações deste método ficticio, e especificar as restrições de conexões, como também permite a edição de diagramas deste método.

Cada nodo é descrito através da composição de vários objetos gráficos primitivos providos pelo **MED**. O engenheiro de software dispõe de facilidades de coloração de objetos para facilitar o processo de desenho O nodo fábrica é formado por um polígono irregular. O engenheiro de software desenha o polígono segmento por segmento, utilizando a facilidade de "sketching", descrita na seção 5.2.2. O nodo "supermercado" é formado por quatro retângulos com cores diferentes. O nodo também possui um objeto do tipo texto. O texto definido no **MED** não poderá ser editado no **EDE**. O nodo "armazém" é formado por um retângulo, um polígono de três lados (o teto do armazém) e um objeto texto. A figura 6.34 o desenho destes três nodos.

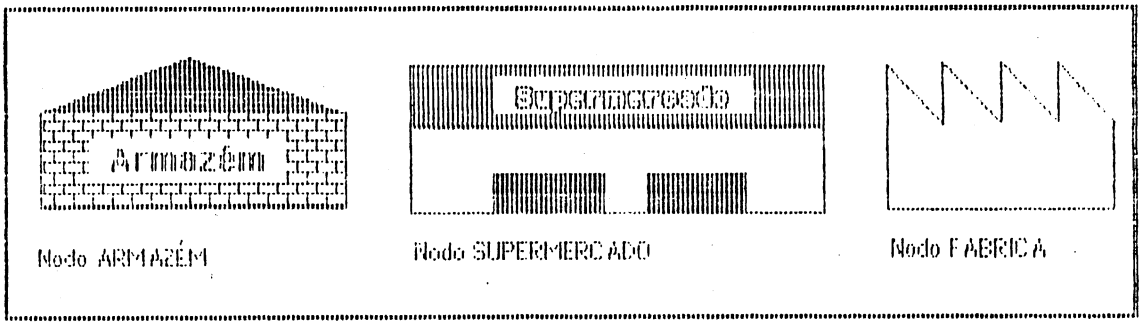


Figura 6.34 - Nodos do método "transporte" desenhados no **MED**.

As figura 6.35 mostra a definição de uma decoração utilizada no método, chamada "caminhão". A janela de definição de decorações possui as mesmas caracteristicas da janela de definição de nodos. Da mesma forma

que os nodos, os objetos textos definidos na decorações pelo **MED** não podem ser editados no **EDE**.

As outras decorações, "navio" e "trem", são mostradas na figura 6.36.

A figura 6.37 mostra a edição de um diagrama do método "transporte" no **EDE**.

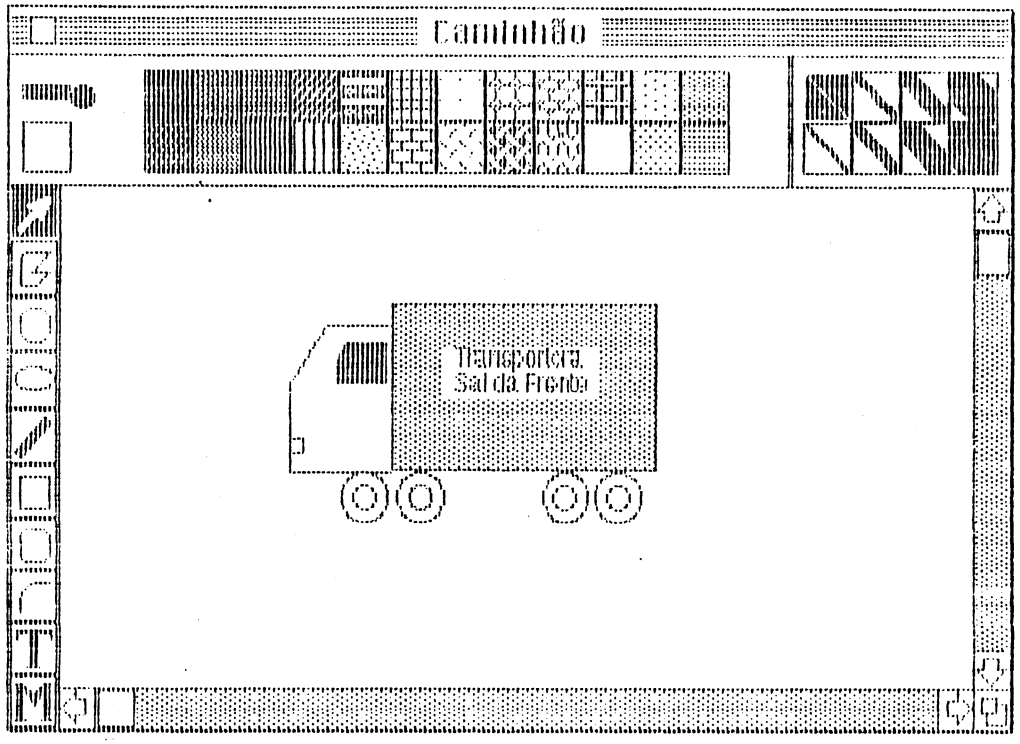


Figura 6.35 - Janela de edição da decoração "caminhão" do método "transporte".

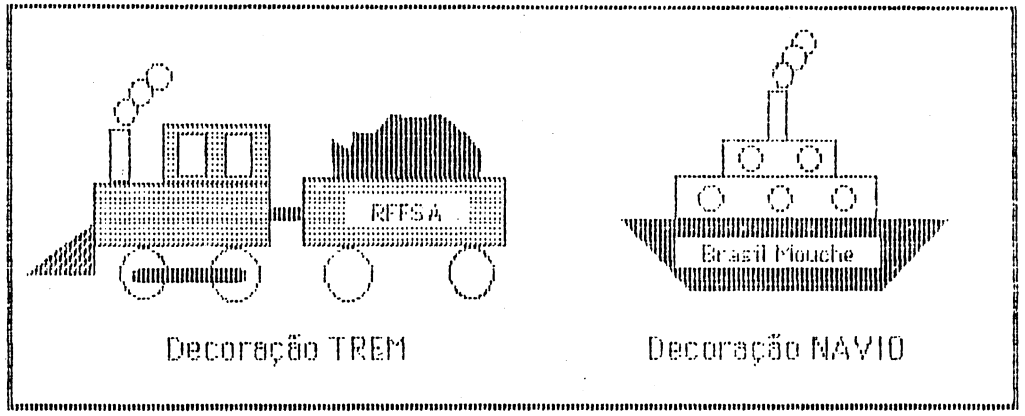


Figura 6.36 - Decoração "TREM" e "NAVIO" do método transporte.

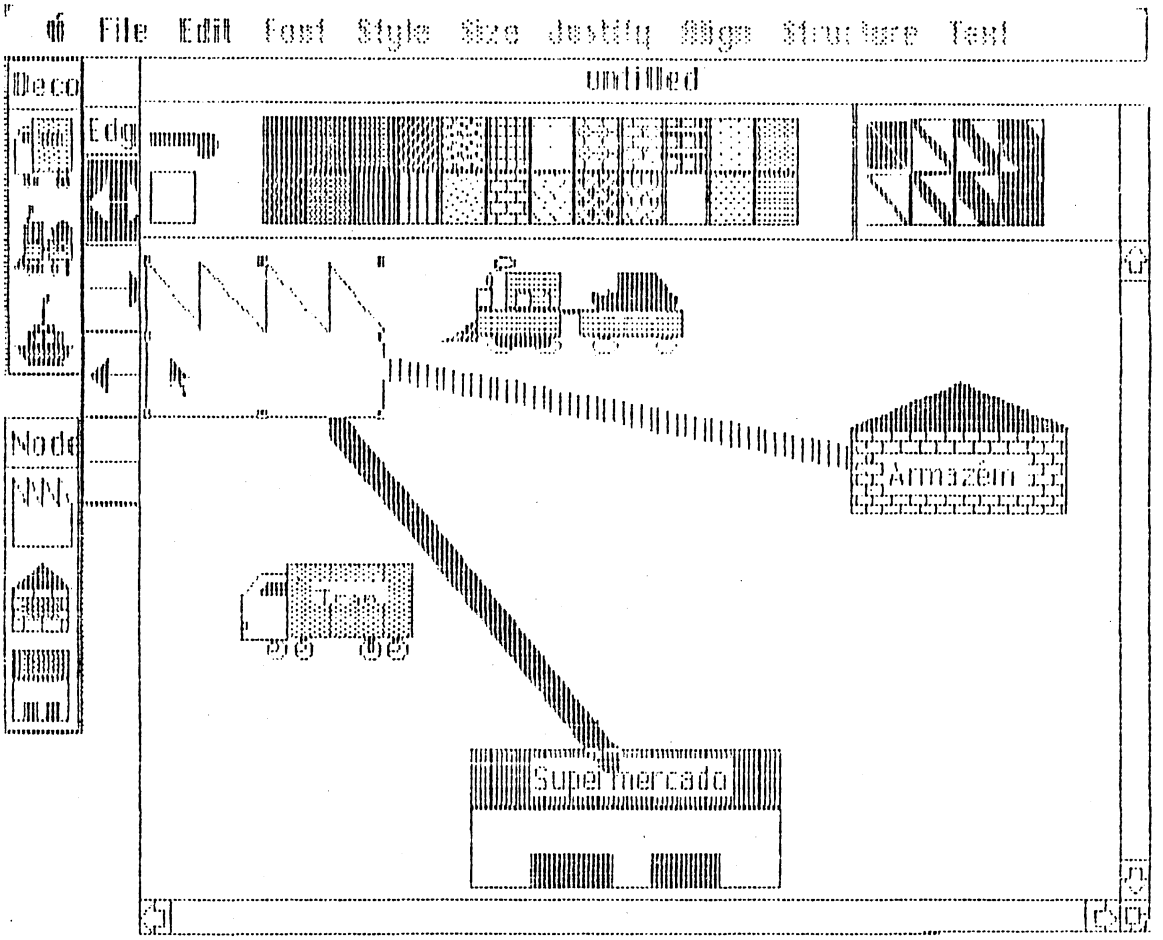


Figura 6.37 - Exemplo de um diagrama do método transporte.

7 IMPLEMENTAÇÃO

O protótipo do **EDG** foi implementado em um equipamento Macintosh. Tal equipamento fornece uma série de facilidades para a construção de interfaces gráficas. A seção 7.1 mostra a arquitetura do equipamento Macintosh, comentando as principais facilidades providas por ele. As informações constantes nesta seção foram retiradas do Inside Macintosh [INS 85]. A seção 7.2 apresenta a estrutura de dados e a arquitetura dos programas que implementam o **MED** e o **EDE**.

7.1 A Arquitetura do Sistema Macintosh

As rotinas disponíveis no sistema Macintosh são agrupadas funcionalmente em módulos, denominados "managers". A figura 7.1 mostra a arquitetura deste módulos.

O sistema operacional é a camada mais baixa. Ele realiza as tarefas básicas de entrada e saída, gerência da memória e manipulação de interrupções. A "User Interface Toolbox" é uma camada acima do sistema operacional. A "Toolbox" ativa o sistema operacional para realizar operações de baixo nível. O **EDG** também utiliza diretamente algumas rotinas do sistema operacional, como, por exemplo, o gerente de arquivos.

Além disto, o sistema Macintosh também coloca à disposição do programador rotinas em RAM. Estas rotinas são utilizadas para realizar operações especializadas, como, por exemplo, aritmética de ponto-flutuante.

A "User Interface Toolbox" possui várias facilidades para auxiliar a construção de programas que seguem os padrões de interface do Macintosh. Ela oferece um conjunto de rotinas, algumas das quais o **EDG** utiliza para implementar a sua interface (descrita no capítulo 5). A figura 7.2 mostra como suas rotinas são funcionalmente agrupadas em um nível de hierarquia relativo. As próximas seções apresentarão sucintamente cada módulo da "Toolbox".

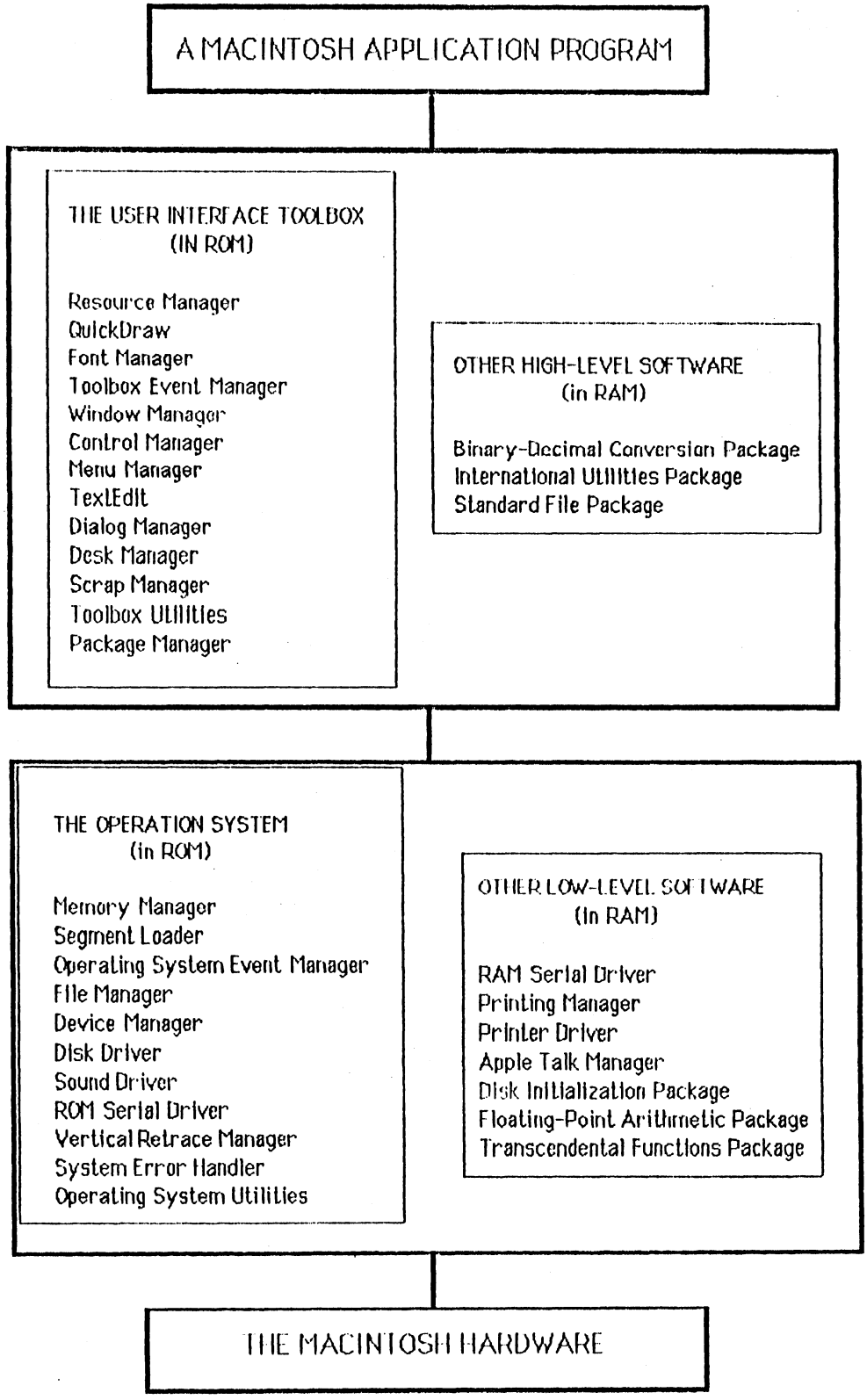


Figura 7.1 - Arquitetura do Sistema Macintosh [INS 85].

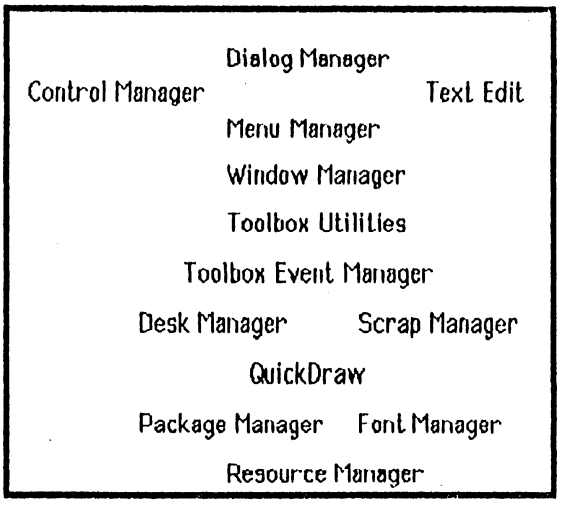


Figura 7.2 - Módulos da "toolbox" em uma estrutura relativa de hierarquia [INS 85].

7.1.1 Gerente de recursos

Aplicações que executam em equipamento Macintosh geralmente utilizam muitos recursos, tais quais menus, tipos de caracteres e ícones. Estes recursos são armazenados em **arquivos de recursos**. Por exemplo, os ícones mostrados na figura 5.12 são residentes em um arquivo de recursos como uma matriz de bits de 32 x 32. Em alguns casos os recursos consistem de informação descritiva, como, por exemplo, os nomes das opções do cardápio. O gerente de recursos obtém os recursos no arquivo respectivo, e provê rotinas que possibilitam ao **EDE** e a outras partes da "Toolbox" manipulá-los.

Existe um arquivo de recursos associado a cada aplicação, bem como um **arquivo de recursos do sistema**, o qual contém os recursos compartilhados por todas as aplicações. Por exemplo, as cores de preenchimentos dos objetos gráficos fornecidas pelo **EDG**, mostradas nas figuras 5.13, 5.15 e 5.23, são exemplos de **recursos do sistema**.

Os recursos usados por uma aplicação são criados e atualizados separadamento do código da aplicação. Esta separação é a principal vantagem de se usar recursos. Por exemplo, a troca dos nomes das opções do cardápio do **EDG** não requer qualquer recompilação do programa.

7.1.2 "Quick-draw"

Todas as operações gráficas no Macintosh são realizadas pelo **quickdraw**. Através dele podem ser desenhados:

- textos em vários tipos de impressão espaçados proporcionalmente, com vários tipos de estilos;

- linhas retas de qualquer tamanho, largura e cor;

- vários tipos de formas gráficas, incluindo retângulos, retângulos chanfrados, elipses, segmentos de elipses (arcos) e polígonos. Todos estes objetos gráficos podem ter vários tipos de linhas de contorno e várias cores de preenchimento;

- figuras compostas de qualquer combinação dos objetos gráficos acima citados.

Além destas facilidades, o "quickdraw" possui várias outras capacidades comuns a outros pacotes gráficos, tais como:

- capacidade de definir qualquer número distinto de "portas gráficas" na tela. Cada porta tem seu ambiente de desenho, que inclui, entre outras propriedades: sistema de coordenadas próprio, localização de desenho, estilo e tipo de caracteres;

- recorte em áreas arbitrárias ("clipping").

7.1.3 Gerente de tipos de caracteres ("font manager")

A principal função do gerente de tipos de caracteres é prover um suporte à manipulação de caracteres para as rotinas do "quickdraw". Tipo de caracteres, ou "font", é um conjunto completo de caracteres de um padrão gráfico. Usualmente não inclui qualquer variação no estilo. Isto é feito pelas rotinas do "quickdraw".

7.1.4 Gerente de eventos da "toolbox"

O gerente de eventos da "toolbox" é o elo entre o **EDG** e os seus usuários. Toda vez que o usuário aperta o botão do "mouse", digita uma tecla ou insere um disquete, o **EDG** é notificado por meio de uma mensagem. Assim sendo, os programas que implementam o **MED** e o **EDE** são dirigidos por mensagens.

À medida que os eventos são recebidos eles são colocados em uma fila de mensagens. O gerente de eventos retira os eventos desta fila e os coloca à disposição do **EDG**. O **EDG** então responde ao evento chamando as rotinas adequadas. Por exemplo, chamando a rotina encarregada de fechar uma janela quando é recebida uma mensagem para isto.

7.1.5 Gerente de janelas ("window manager")

Toda a informação mostrada pelo **EDG** é apresentada em forma de janelas. Para criar janelas, ativá-las, movê-las, alterar suas dimensões ou fechá-las, o **EDG** ativa as rotinas do gerente de janelas. Ele monitora a sobreposição de janelas na tela, desta forma o **EDG** não necessita saber como as janelas estão dispostas na tela para poder manipulá-las. Por exemplo, o gerente de janelas avisa ao gerente de eventos (comentado na seção 7.1.4) quando informar ao **EDG** que uma janela precisa ser redesenhada. Também, quando o usuário pressiona o botão do "mouse", o **EDG** chama o gerente de janelas para saber em qual parte de qual janela o botão foi pressionado.

Os tipos de janelas utilizados no **EDG** são pré-definidos pelo Macintosh. Um exemplo destes tipos de janelas é a "janela documento", mostrada na figura 5.3. Um outro tipo é a janela denominada "caixa de diálogo", exemplificada na figura 5.20.

7.1.6 Gerente de controles ("control manager")

O gerente de controles é o módulo da "toolbox" que manipula os controles. Controle é um objeto na tela do Macintosh com o qual o usuário, utilizando o "mouse", pode causar ações instantâneas com resultados visíveis ou trocar o estado do programa para causar ações futuras.

O **EDG** somente utiliza os tipos de controle já pré-definidos pelo Macintosh, são eles:

- Botões.

São utilizados para causar uma resposta imediata quando acionados com o "mouse". São exemplos deste tipo de controles os botões de "OK" e "Cancel" utilizados em várias caixas de diálogos do **EDG**.

- Retângulos de marcação ("check box").

Servem para fixar um estado de ativo ou inativo, por exemplo os mostrados na figura 5.22.

- Botões de rádio ("radio button").

Servem para indicar uma escolha entre várias opções. Geralmente são organizados em grupos, nos quais somente uma propriedade do grupo pode estar ativa ao mesmo tempo. Por exemplo, os mostrados na figura 5.14.

- "Dials".

Servem para mostrar valores, magnitude ou posição de algum elemento. No **EDG** o único tipo de "dial" utilizado é a barra de rolagem ("scroll bar"), para rolar o conteúdo das janelas tanto verticalmente como horizontalmente.

7.1.7 Gerente de cardápios ("menu manager")

O gerente de cardápios controla todo o processo de escolha de itens do cardápio. Quando o usuário escolhe um item ele avisa a aplicação, no

caso o **EDG**, qual o item escolhido a fim de que o programa possa realizar a ação correspondente à escolha.

7.1.8 Editor de textos ("TextEdit")

O editor de texto provido pela "toolbox" é um conjunto de rotinas e tipos de dados que possibilitam a edição e formatação de textos utilizados no **EDG**. Entre outras capacidades, estas rotinas permitem:

- inserção de novos textos;
- exclusão de caracteres;
- seleção dinâmica de textos, através de arrasto do "mouse";
- rolagem de textos dentro de janelas;
- possibilidade de copiar e excluir textos selecionados;
- formatação e ajustes de textos;
- importar e exportar blocos de textos entre aplicações, através da "scrapbook" (seção 7.1.10).

7.1.9 Gerente de diálogos ("Dialog Manager")

O Gerente de Diálogos é uma ferramenta de manipulação das caixas de diálogos e dos alertas mostrados no MacInstosh. Caixas de diálogos são utilizadas para a obtenção de parâmetros de execução de comandos. Sempre que o usuário ativar um comando do cardápio de opções e este comando necessitar de informações complementares para executar suas tarefas, o programa de aplicação ativa uma caixa de diálogo para captar os dados necessários. Toda a interação do usuário com a caixa de diálogo é feita pelo **gerente de diálogos**. Ao final da interação o controle retorna para o programa de aplicação.

O gerente de diálogos também é utilizado pelo programa de aplicação para mostrar instruções e alertas para o usuário. Para tal é utilizado um tipo de janela especial denominado **caixa de alerta**. Uma caixa de alerta é similar a uma caixa de diálogo, mas somente é mostrada quando o usuário efetua uma operação inválida ou quando o programa deseja chamar a

atenção deste para alguma situação especial.

7.1.10 Gerente de rascunho ("Scrap Manager")

O gerente de rascunho é um conjunto de rotinas e tipos de dados que permite a importação e exportação de informações pelas aplicações utilizando a "scrapbook". A "scrapbook" é um arquivo que funciona como um veículo de transferência de dados entre duas aplicações. Através deste recurso o **EDG**, por exemplo, pode importar textos digitados em um processador de textos, e exportar diagramas para o mesmo.

7.1.11 Utilitários da "Toolbox"

A "toolbox" fornece uma série de rotinas tais como: aritmética de ponto flutuante, manipulação de "strings", operações booleanas em mapas de bits, entre outras.

7.1.12 Gerente de Pacote ("Package Manager")

O gerente de pacotes permite a utilização de software, denominados pacotes, residentes em RAM. São eles:

- Pacote de arquivos padronizados.

Conjunto de rotinas utilizadas para criar, abrir e fechar arquivos documentos, e interface padrão de interação com o usuário para a seleção de arquivos.

- Pacote de conversão binário-decimal.

Conjunto de rotinas que permitem converter números inteiros para "strings" decimais, e vice-versa.

- Pacote de utilitários internacionais.

Permite obter informações relativas a alguns países tais como: padrões de impressões de números, datas, horas e moedas.

7.2 A Implementação do Protótipo do EDG

O protótipo do **EDG** foi implementado através de dois programas: um programa que implementa o **MED** e outro que implementa o **EDE**. As seções 7.2.1 e 7.2.2 respectivamente descrevem a estrutura de dados e as arquiteturas do programa **MED** e do programa **EDE**.

7.2.1 A Implementação do MED

Esta seção descreve alguns detalhes de implementação do **MED**. A seção 7.2.1.1 mostra a estrutura de dados do **MED** e a seção 7.2.1.2 mostra como esta estrutura é armazenada, para que possa ser utilizada pelo **EDE**.

7.2.1.1 A Estrutura de dados

Um método pode possuir vários tipos de nodos, decorações, pontas , arcos e regras, a qual possui as regras de ligação entre os nodos. Cada tipo de nodo, ponta e decoração é formado pela composição de objetos primitivos disponíveis no **MED** (retângulos, retângulos chanfrados, elipses, segmentos de reta, polígonos regulares e irregulares, elipses, textos e arcos de elipses). Cada tipo de arco possui atributos de visualização (cor e largura da linha) e duas pontas. As regras de ligação associam os tipos de nodos com os tipos de arcos, e através delas são informadas as ligações possíveis entre os nodos, objetos e arcos do método.

A seguir será mostrado como as entidades do **MED** são implementadas.

A entidade método é implementada da seguinte forma:

```
Type Metodo = RECORD
        identificador  : Nome;
        primNodo       : NodoHandle;
        primArco       : ArcoHandle;
        primDecoracao  : DecoracaoHandle;
```

```
        primPonta     : PontaHandle;
    END;
```

"Identificador" contém o nome do método. "PrimArco", "primNodo", "primDecor" e "primPonta" são ponteiros para a lista de tipos de arcos, nodos, decorações e pontas.

Cada tipo de nodo de um método possui a seguinte estrutura de dados:

```
Type EstTipoNodo = RECORD
        identificador : Nome;
        primObjeto    : ObjetoHandle;
        proxNodo      : NodoHandle;
END;
```

O campo "identificador" contém o nome do tipo do nodo. "PrimObjeto" é um ponteiro para a lista de objetos que forma o tipo de nodo. "ProxNodo" é um ponteiro para o próximo tipo de nodo da lista.

Cada tipo de decoração possui a seguinte estrutura de dados:

```
Type EstTipoDecor = RECORD
        identificador       : Nome;
        primObjDecoracao : ObjetoHandle;
        proxDecoracao       : DecoracaoHandle;
END;
```

O campo "identificador" contém o nome do tipo de decoração. O campo "primDecoracao" é um ponteiro para o primeiro objeto da lista de objetos que formam a decoração, e o campo "proxDecoracao" contém um ponteiro para o próximo elemento da lista de decorações.

Cada tipo de arco do método possui a seguinte estrutura de dados:

```
Type EstTipoArco = RECORD
        identificador  : Nome;
        tipoPonta      : ARRAY[1..2] OF PontaHandle;
```

```
        larguraLinha  : Integer;
        padraoLinha   : Pattern;
        proxArco      : ArcoHandle;
    END;
```

O campo "identificador" contém o nome do tipo de arco. "TipoPonta" são ponteiros para os dois tipos de pontas que foram o tipo de arco. "LarguraLinha" e "padraoLinha" contém a largura e a cor da linha do arco do tipo especificado. "ProxArco" é um ponteiro para o próximo elemento da lista de tipos de arcos.

Cada tipo de ponta possui a seguinte estrutura de dados:

```
Type EstTipoPonta = RECORD
        identificador  : Nome;
        primObjPonta : ObjetoHandle;
        proxPonta      : PontaHandle;
    END;
```

O campo "identificador" contém o nome do tipo da ponta, "primObjPonta" é um ponteiro para o primeiro objeto da lista de objetos que formam a ponta e o campo "proxPonta" é um ponteiro para a próxima ponta da lista de tipos de pontas do método.

Todo o método especificado no **MED** possui uma tabela de ligações possíveis. Esta tabela possui a seguinte estrutura de dados:

```
Type TabLigação = RECORD
        origem         : NodoHandle;
        objetoSaida    : ObjetoHandle;
        destino        : NodoHandle;
        objetoChegada: ObjetoHandle;
        tipoArco       : ArcoHandle;
        proxDefinicao : DefinicaoHandle;
    END;
```

Os campos "origem" e "destino" são ponteiros para os tipos de nodos que podem ser conectados. "ObjetoSaida" e "objetoChegada" são ponteiros para os dois objetos através dos quais poderá ser efetuada a ligação. "TipoArco" é um ponteiro para o tipo de arco que pode ser utilizado na conexão entre os dois nodos e objetos especificados. "ProxDefinição" é um ponteiro para a próxima tupla da tabela.

O **EDE** utiliza esta tabela sempre que for solicitada a conexão entre dois nodos.

Cada objeto formador das pontas, decorações e nodos possui a seguinte estrutura de dados:

```
Type EstTipoObjeto = RECORD
        nomeObjeto    : Nome;
        envelope      : Rect;
        tipoLinha     : Integer;
        padraoLinha   : Pattern;
        padraoPreen   : Pattern;
        fixo          : Boolean;
        duplicavel    : Boolean;
        opcional      : Boolean;
        numConecMin   : Integer;
        numConecMax   : Integer;
        txFonte       : Integer;
        txFace        : Style;
        txTam         : Integer;
        txJust        : Integer;
        textoHandle   : TEHandle;
        proxObjeto    : ObjetoHandle;
        CASE primitivo: Objetos OF
        retangulo, elipse, texto, metaTexto : ();
        retanguloArrend     : (ovalWidth, ovalHeight : Integer);
        poligReg, poligono  : (NumLados : Integer;
        primeiroVertice     : VerticeHandle);
        reta                : (p1, p2 : Point);
```

```
        arco                : (startAngle, arcAngle : Integer);
    END;
```

A estrutura contém informações referentes aos atributos de visualização do objeto (largura e cor da linha envoltória e cor de preenchimento do interior do objeto), atributos de estilização do texto que ficará associado ao objeto (estilo, ajuste, tamanho, tipo de impressão ("font")), tipo de objeto primitivo (reta, retângulo, retângulo chanfrado, elipse, arco de elipse, texto, meta-texto, polígono regular e irregular). Dependendo do tipo de objeto primitivo, também são armazenadas outras informações. Como por exemplo, o tipo polígono requer uma lista de vértices. Meta-texto é um tipo especial de objeto, será utilizado para associar um texto a uma produção de uma gramática de atributos, para edição de textos formais (não foi implementado). Os outros campos informam o tipo de comportamento que o texto poderá ter no **EDE**. Por exemplo, o campo opcional informa se o objeto pode ser excluído no **EDE**. Todos os campos serão explicados na apresentação da estrutura de dados do **EDE**.

7.2.1.2 Gravação da estrutura de dados do MEG

O nome do arquivo gravado é o nome do método. O arquivo possui três partes: a lista de elementos do método, e o próprio método.

A estrutura da lista de elementos gravada é a seguinte :

- número de elementos;
- número do elemento selecionado;

- os elementos, os quais são formados por :
  - retângulo do ícone;
  - tipo do elemento ("nodo", "decoração", "ponta", "arco");
  - nome do elemento.

A estrutura de dados do método é gravada da seguinte forma:

- lista de nodos;

- lista de decorações;
- lista de pontas;
- lista de conectores.

Para cada nodo grava-se:
- nome do nodo;
- tipo de elemento ("nodo", "decoração", "ponta", "arco");
- lista de objetos.

Uma lista de objetos é composta por :
- número de objetos;
- os objetos.

Cada objeto possui as seguintes informações :
- número do objeto;
- nome do objeto;
- envelope;
- tipo de linha;
- padrão da linha;
- padrão de preenchimento;
- booleana indicando se o objeto é fixo;
- booleana indicando se o objeto é duplicável;
- booleana indicando se o objeto é opcional;
- número mínimo de conectores;
- número máximo de conectores;
- número de conectores;
- tipo de impressão (padrão gráfico);
- estilo;
- tamanho;
- ajuste;
- o tipo primitivo do objeto (retângulo, elipse, texto, meta-texto, retângulo chanfrado, reta, arco, polígono, polígono regular);

- dependendo do tipo primitivo do objeto, tem-se :
  - se o tipo for retângulo, elipse ou meta-texto, não grava-se nada;
  - se for retângulo chanfrado, grava-se a largura e a altura do ângulo de curvatura da elipse formadora do canto;
  - se for reta, grava-se o ponto inicial e o ponto final;
  - se for arco, grava-se o ângulo inicial e o ângulo do arco;
  - se for texto, grava-se uma indicação se o texto existe ou não. Se existir, grava-se seu tamanho e o texto em si;
  - se for polígono ou polígono regular, grava-se a lista de vértices.

A lista de decorações é formada por :
- nome da decoração;
- tipo de elemento;
- lista de objetos.

A lista de pontas é formada por :
- nome da ponta;
- tipo de elemento;
- lista de objetos.

A lista de arco é composta por :
- nome do arco;
- tipo de elemento;
- número do arco;
- nome da ponta inicial;
- nome da ponta final;
- largura da linha;
- padrão da linha.

As regras de ligação são gravadas na forma de uma lista, que contém :
- o número de regras de ligação definidas;

- as regras, formadas por :
  - número do nodo origem;
  - número do objeto origem;
  - número do nodo destino;
  - número do objeto destino;
  - número do tipo de arco através do qual poderão ser realizadas as conexões.

7.2.2 A Implementação do EDE

Esta seção mostra alguns detalhes de implementação do **EDE**. A seção 7.2.2.1 mostra a estrutura de dados utilizada para a representação dos diagramas e a seção 7.2.2.2 mostra como esta estrutura é armazenada.

7.2.2.1 A estrutura de dados do EDE

É através do **EDE** que são editados os diagramas. Um diagrama pode possuir vários nodos, arcos, decorações e rótulos. Um nodo pode ser formado por vários objetos e pode possuir um diagrama. Um arco pode possuir várias decorações, rótulos, duas pontas, e pode ligar-se a dois nodos por meio de dois objetos. Uma decoração pode ser formada por vários objetos. Um rótulo possui atributos tipo de impressão do texto, estilo, ajuste, tamanho e o texto propriamente dito. O anexo I mostra o diagrama de entidades que representa a estrutura de dados do **EDE**.

Do ponto de vista de implementação, a estrutura de dados do **EDE** é gerenciada como uma árvore, onde cada nó é um diagrama. Existe um diagrama que é a raiz da árvore. Cada diagrama possui a seguinte estrutura:

```
Type EstDiagrama = RECORD
     listaNodos    : HdlDesNodo;
     listaArcos    : HdlDesArco;
     listaDecors   : HdlDesDecor;
     listaLabels   : HdlDesTexto;
     numPagina     : Integer;
```

```
        nomeDiagrama: Nome;
        metodoDiag    : Nome;
        nodoPai       : HdlNodo;
        DiagPai       : HdlDiagrama;
        PontJanela    : WindowPtr;
    END;
```

"ListaNodos" é um ponteiro para o descritor da lista de nodos do diagrama. "ListaArcos" é um ponteiro para o descritor da lista de arcos. "ListaDecor" é um ponteiro para o descritor da lista de decorações. "Lista Labels" é um ponteiro para o descritor da lista de rótulos. (Rótulo é um texto livre no diagrama). "NumPagina" representa o número da página do diagrama. O **EDE** fornece este número de forma sequencial para cada diagrama criado. "NomeDiag" é o nome do diagrama. "MetodoDiag" representa o nome do método a qual o diagrama pertence. "NodoPai" e "DiagPai" são respectivamente ponteiros para o nodo e diagrama pai. Estas informação são utilizadas quando o diagrama representa o refinamento de um nodo. O diagrama raiz é o único que não as possui. "PontJanela" é um ponteiro para a janela através da qual o diagrama é editado.

Os descritores da lista de nodos, arcos, decorações e rótulos possuem ponteiros para o primeiro e último elemento da lista, e um contador de elementos.

Cada nodo do diagrama possui a seguinte estrutura:

```
Type EstNodo = RECORD
        nodoAnterior, nodoPosterior: HdlNodo;
        identificador  : Nome;
        centro         : Point;
        tipo           : boolean;
        listaObjetos   : HdlDesObjeto;
        DiagPai        : HDlDiagrama;
        DiagFilho      : HdlDiagrama;
        ArcoPai        : HdlArco;
    END;
```

"NodoAnterior" e "nodoPosterior" são ponteiros para o nodo anterior e posterior da lista de nodos. "Identificador" é o tipo do nodo definido no método. "Centro" é o ponto que representa o centro do envelope do nodo. "ListaObjetos" é um ponteiro para o descritor da lista de objetos gráficos que forma o nodo. "DiagPai" é um ponteiro para o diagrama a que o nodo pertence. "DiagFilho" é um ponteiro para o diagrama que representa o refinamento do nodo. "Tipo" indica se o nodo é uma interface de um diagrama ou não. "ArcoPai" é um ponteiro para o arco, do diagrama pai que o nodo representa.

Por exemplo, a figura 5.26 mostra um diagrama no processo de refinamento e condensação de nodos. A cena 2 desta figura mostra dois nodos representando a interface do nodo que está sendo refinado. Estes dois nodos representam os arcos "1" e "4" da cena 1. Os nodos "a" e "b" da cena 2 possuem, em sua estrutura de dados, o campo "ArcoPai" apontando para os arcos "1" e "4" e o campo "tipo" com valor verdadeiro. Desta forma, qualquer modificação nas ligações do nodo pai podem ser refletidas no diagrama filho, e vice-versa. Caso o usuário solicite a exclusão dos nodos "a" e "b" do diagrama mostrado na cena 2, o programa, com base nestes dados, pode excluir os arcos "1" e "4" do diagrama mostrado na cena 1. Da mesma forma, caso o usuário solicite a exclusão do arco 1 da cena 1, o **EDE** exclui o nodo "a" da cena2.

Cada arco de um diagrama possui a seguinte estrutura:

```
Type EstArco = RECORD
        arcoAnterior  : HdlArco;
        arcoPosterior : HdlArco;
        identificador : Nome;
        nodosLig      : ARRAY[1..2] OF HdlNodo;
        objetosLig    : ARRAY[1..2] OF HdlObjeto;
        pontasLig     : ARRAY[1..2] OF HdlPonta;
        nodoVirtual   : HdlNodo;
        PI            : ARRAY[1..2] OF Point;
        ListaEmbDeco  : HdlDesEmbDecor;
        ListaVertices : HdlDesVertice;
```

```
        ListaLabel     : HdlDesTexto;
        padraoLinha    : Pattern;
        larguraLinha   : Integer;
        tipoConector   : ConecTipo;
        visivel        : Boolean;
        DiagPai        : HdlDiagrama;
END;
```

"ArcoAnterior" e "arcoPosterior" são ponteiros para o arco anterior e posterior da lista de arcos do diagrama. "Identificador" é o nome do tipo de arco ao qual ele pertence. "NodosLig" são ponteiros para os dois nodos alvos da conexão. "ObjetosLig" são ponteiros para os dois objetos alvos da conexão. "PontasLig" são ponteiros para os descritores da lista de objetos que forma as pontas do arco. "Pi" possui os pontos de intersecção do arco com os objetos alvos da ligação. "ListaEmbDeco" é um ponteiro para o descritor da lista de apontadores das decorações vinculadas ao arco. "ListaVertices" é um ponteiro para a lista de pontos que forma os segmentos do arco. "ListaLabel" é um ponteiro para o descritor da lista de rótulos vinculados ao arco. "PadraoLinha" contém a cor dos segmentos de reta que formam o arco. "LarguraLinha" contém a espessura dos segmentos de reta. "TipoConector" indica o tipo de ligação do arco, se o arco é suavizado, segmentado ou elíptico. O arco segmentado é composto por um ou mais segmentos de reta, o arco suavizado é formado por uma curva B-Spline, e o arco elíptico é formada por um quarto de elipse. "DiagPai" é um ponteiro para o diagrama ao qual pertence o arco.

O campo "NodoVirtual" contém um ponteiro para o nodo que representa a ligação em um diagrama filho. Por exemplo, na figura 5.26 o arco "1" e "2" são representados na cena 2 pelos nodos "a" e "b". O campo "nodoVirtual" do arco "1" possui um ponteiro para o nodo "b" do diagrama mostrado na cena 2.

Cada arco pode ter duas pontas. A estrutura de cada ponta é a seguinte:

```
Type EstPonta = RECORD
        envelope      : Rect;
        angulo        : real;
        listaObjetos  : HdlDesObjeto;
        arcoPai       : HdlArco;
END;
```

"Envelope" é o menor retângulo que contém todos os objetos formadores da ponta. "Angulo" é o valor do ângulo de inclinação da ponta. "ListaObjetos" é um ponteiro para o descritor da lista de objetos formadores da ponta. "ArcoPai" é um ponteiro para o arco ao qual a ponta pertence.

Cada decoração de um diagrama possui a seguinte estrutura:

```
Type EstDecor = RECORD
        decorAnterior, decorPosterior : HdlDecor;
        Identificador  : Nome;
        centro         : Point;
        listaObjetos   : HdlDesObjeto;
        ArcoPai        : HdlArco;
        DiagPai        : HdlDiagrama;
END;
```

"DecorAnterior" e "decorPosterior" são ponteiros para a decoração anterior e posterior da lista de decorações. "Identificador" é o nome do tipo ao qual pertence a decoração. "Centro" é o ponto central do envelope que compreende todos os objetos da decoração. "ListaObjetos" é um ponteiro para o descritor da lista de objetos que formam a decoração. "ArcoPai" é um ponteiro para o arco ao qual a decoração está vinculada. "DiagPai" é um ponteiro para o diagrama ao qual a decoraçáo pertence.

As pontas, decorações e nodos são formados graficamente por uma lista de objetos. Cada objeto possui a seguinte estrutura de dados:

```
EstObjeto = RECORD
        objetoAnterior: HdlObjeto;
        objetoPosterior : HdlObjeto;
        listaConec      : HdlDesConector;
        visivel         : Boolean;
        atribObjeto     : objAtributos;
        CASE pertence : TiposDeElementos OF
                Nodo            : (nodoPai : HdlNodo);
                Decoracao       :(decorPai : HdlDeco);
                Ponta           : (pontaPai : HdlPonta);
        END;
```

Os campos "objetoAnterior" e "objetoPosterior" são ponteiros para o objeto anterior e posterior da lista de objetos. "ListaConec" é um ponteiro para o descritor da lista de apontadores das conexões dos objetos. Cada objeto pode estar ligado a vários arcos, por esta razão o **EDE** mantém uma lista contendo os endereços dos arcos que estão conectados ao objeto. "Pertence" é utilizado para informar se o objeto faz parte de um nodo, decoração ou uma ponta, sendo que, dependendo de qual elemento ele pertença, é informado o endereço do elemento. "Visível" informa se o objeto é visível ou não. "AtribObjeto" é uma sub-estrutura que contém os atributos gráficos do objeto, sendo descrita a seguir.

```
Type ObjAtributos = RECORD
        nomeObjeto    : Nome;{Nome do objeto}
        envelope : Rect; {Box do objeto}
        tipoLinha : Integer;{Altura e largura da PEN}
        padraoLinha : Pattern;{Padrão da linha}
        padraoPreen : Pattern;{Padrao de prenchimento}
        fixo : Boolean;{Objeto pode ser movido?}
        duplicavel:Boolean;{Objeto  pode ser duplicavel?}
        opcional : Boolean;{Objeto pode ser apagado?}
        numConecMin : Integer;{Numero mínimo de conexões}
        numConecMax : Integer;{Numero máximo de conecções}
        numConec : Integer;{Contador de conexões}
```

```
textoHandle : TEHandle;{Handle para o texto}
valVerSBar : integer;{valor do "vertical scroll bar"}
valHorSBar : integer;{valor do "horizontal scroll bar"}
CASE primitivo : Objetos OF {Tipo de objeto}
        retangulo, elipse, texto, metaTexto : ();   {Não armazena nada}
        retanguloArrend : (ovalWidth, ovalHeight : Integer );
        poligReg, poligono : (listaVertices : HdlDesVertice;);
        reta : (p1, p2 : Point);
        arco : (startAngle, arcAngle : Integer);
END;
```

"NomeObjeto" contém a identificação do objeto feito no **MED**. "Envelope" contém o menor retângulo que compreende todos os pontos do objeto. "TipoLinha", "padraoLinha" e "padraoPreen" são informações pertinentes à largura da linha de contorno do objeto, a cor da linha e a cor de preenchimento do interior do mesmo. "Fixo", "duplicavel" e "opcional" são atributos de anatomia do objeto. Se o objeto é fixo ele somente pode ser transladado no diagrama em conjunto com os outros objetos que formam o nodo ou a decoração. Se o objeto é duplicável significa que o usuário pode duplicar o objeto, mantendo-o vinculado ao nodo ou decoração ao qual pertence. Se o objeto é opcional o usuário pode solicitar a sua exclusão do nodo ou da decoração a qual ele pertence. "NumConecMin", "numConecMax" e "numConec" informam o número mínimo, máximo e atuais de conexões do objeto. "TextoHandle" é um ponteiro para a estrutura de dados que contém o texto vinculado ao objeto. "ValVerSBAR" e "valHorSBar" contém os valores das barras de rolagem da janela temporária de enquadramento de texto. "Primitivo" informa o tipo de objeto gráfico que ele é. Os tipos primitivos suportados pelos **EDE** são: retângulos, elipses, textos, retângulos chanfrados, polígonos regulares e irregulares, segmentos de reta e arco de elipses. Dependento do tipo de objeto gráfico primitivo são mantidos outros dados. Como por exemplo, no caso dos polígonos é armazenado um ponteiro para o descritor da lista de vértices.

O **EDE** carrega para a memória esta estrutura de dados sempre que é solicitada a abertura de um documento, e armazena em um arquivo em disco

sempre que é solicitada uma gravação do mesmo. O EDE não possui, ainda, um suporte de banco de dados para manter sua estrutura de dados, por isto grava-se um arquivo sequencial para armazenar os diagramas editados. O formato deste arquivo é fornecido a seguir.

7.2.2.2 Gravação da estrutura de dados do EDE

Grava-se primeiro o diagrama principal, seguido de todos os seus diagramas filhos. Cada diagrama é gravado com a seguinte estrutura:

- nome do método;
- nome do diagrama;
- número da página;
- lista de nodos;
- lista de arcos;
- lista de decorações;
- lista de labels.

A lista de nodos é formada por :
- número de nodos da lista;
- os nodos.

Cada nodo possui as seguintes informações :
- número do nodo;
- identificador do nodo (nome do tipo do nodo);
- centro do nodo;
- número do arco pai (zero se este não existir);
- booleana indicando se tem diagrama filho;
- lista de objetos do nodo.

Uma lista de objetos é composta por :
- número de objetos da lista;
- os objetos.

Cada objeto possui as seguintes informações :
- número do objeto;
- booleana indicando se é visível;

- os atributos do objeto, relacionados abaixo :
    - nome do tipo do objeto;
    - envelope;
    - tipo de linha;
    - padrão da linha;
    - padrão de preenchimento;
    - booleana indicando se o objeto é fixo;
    - booleana indicando se o objeto é duplicável;
    - booleana indicando se o objeto é opcional;
    - número mínimo de conectores;
    - número máximo de conectores;
    - número de conectores;
    - o tipo primitivo do objeto (retângulo, elipse, texto, meta-texto, retângulo chanfrado, reta, arco, polígono, polígono regular);
    - o valor da barra de rolamento vertical;
    - o texto do objeto;
    - dependendo do tipo primitivo do objeto, tem-se :
        - se o tipo for retângulo, elipse, texto ou meta-texto, não grava-se nada;
        - se for retângulo chanfrado, grava-se a largura e a altura do ângulo de curvatura da elipse formadora do canto;
        - se for reta, grava-se o ponto inicial e o ponto final;
        - se for arco, grava-se o ângulo inicial e o ângulo do arco;
        - se for polígono ou polígono regular, grava-se a lista de vértices.

Textos são gravados com o seguinte formato :
- indicação se o texto existe ou não;
- se o texto existir, tem-se :
    - tamanho do texto;
    - tipo de impressão (padrão gráfico ou *font*);
    - estilo;

- tamanho;
- tipo de ajuste (centro, esquerda ou direita);
- retângulo de visão;
- retângulo destino;
- o texto propriamente dito.

A lista de vértices é composta por :
- número de vértices;
- os pontos dos vértices (coordenada X e Y).

A lista de arcos é formada por :
- número de arcos da lista;
- os arcos.

Cada arco possui as seguintes informações :
- número do arco;
- objeto origem;
- objeto destino;
- ponto de interseção origem;
- ponto de interseção destino;
- padrão da linha;
- largura da linha;
- nome do tipo de conector;
- booleana indicando se é visível;
- booleana indicando se é suavizado;
- as pontas do arco;
- lista de vértices;
- lista de labels.

As duas pontas dos arcos são guardadas da seguinte forma :
- indicação se ela existe (pode existir arcos sem pontas);
- envelope;
- ângulo de inclinação;
- lista de objetos.

A lista de labels é composta por :

- número de labels;
- os textos dos labels.

A lista de decorações é formada por :

- número de decorações;
- as decorações.

Cada decoração é formada por :

- nome do tipo da decoração;
- centro;
- número do arco pai (zero se não existir);
- lista de objetos.

8 CONCLUSÃO

Neste capítulo é feita uma retrospectiva do trabalho apresentado mostrando os resultados obtidos, bem como são propostos desenvolvimentos futuros.

8.1 Retrospectiva

Este trabalho foi elaborado com o propósito de atingir os seguintes objetivos:

- Automatizar o processo de confecção de diagramas. Diagramas são ferramentas essenciais na construção de programas e sistemas complexos. Diagramas auxiliam:

  - a clareza de raciocínio;
  - a comunicação entre os membros das equipes de desenvolvimento;
  - a manutenção;
  - a depuração;
  - e a documentação.

  Apesar destas vantagens, diagramas são extremamente difíceis de criar e manter manualmente. Devido a isto, um dos objetivos do trabalho é prover uma ferramenta de auxílio à construção de diagramas.

- Minimizar os problemas enfrentados pelos ES no uso de EDs. Muitos EDs são especializados em determinados tipos de diagramas. Como consequência os ES que necessitem utilizar mais do que uma ferramenta deste gênero enfrentam vários obstáculos, entre os quais:

  - funções de edição incompatíveis;
  - organização da tela;

- forma de ativação do sistema;
- padrões de saída distintos;
- adaptabilidade às notações impostas pelo EDs;
- estrutura de dados incompatíveis;
- e formatação não automatizada.

O trabalho desenvolvido visou resolver, com exceção da formatação automática, estes problemas.

- Servir de suporte à geração de editores diagramáticos. Um sistema CASE possui várias ferramentas integradas com a finalidade de automatizar, o máximo possível o processo de desenvolvimento e manutenção de programas e sistemas. Os editores são um dos principais tipos de componentes de um sistema CASE, pois o processo de produção de sistemas e programas consiste de várias etapas, nas quais são utilizados vários tipos de notações diagramáticas.

Para alcançar estes objetivos foi implementado um protótipo de um **E**ditor de **D**iagramas **G**eneralizado, denominado **EDG**. O **EDG** é composto por dois módulos principais: o **M**eta **E**ditor **D**iagramático (**MED**) e o **E**ditor **D**iagramático **E**specífico (**EDE**).

Através do **MED**, o usuário especificador (também chamado administrador de CASE) descreve o tipo de notação diagramática. Utilizando-se de uma interface gráfica, o **AC** desenha os nodos, arcos, pontas de arcos e as decorações que formam um determinado tipo de notação. O **MED** somente suporta as notações que podem ser abstraídas como grafos, logo, somente provê recursos para que sejam especificadas notações formadas por nodos, arcos e decorações. Além disso, o **MED** provê mecanismos por intermédio dos quais o **AC** pode especificar as restrições de ligações entre os nodos, ou seja, que tipos de nodos podem ser conectados através de que tipos de arcos.

Com base nas definições feitas no **MED**, o **EDE** possibilita a edição de diagramas de várias notações distintas. Através das facilidades providas pelo **EDE**, o usuário (o programador ou analista) desenha os diagramas interativamente, criando, movendo, excluindo, conectando, condensando e refinando diagramas. As verificações de consistência são realizadas automaticamente, sempre que o usuário realizar uma alteração de conteúdo do diagrama. Com base na tabela de ligações entre nodos, construída pelo **AC** no **MED**, os diagramas editados no **EDE** são confeccionados corretamente. O **EDE** realiza o processo de condensação e refinamento de diagramas, mantendo a integridade de todos os seus níveis hierárquicos.

O usuário não sofre os problemas de adaptação a vários editores, já que a interface do **EDE** é igual para todos os tipos de diagramas suportados. As funções de edição, a organização da tela, os dispositivos de ativação e manipulação do **EDE** são idênticos para qualquer tipo de diagrama editado pelo usuário. O **EDG**, através das facilidades providas pelo sistema Macintosh, permite a impressão dos diagramas com a mesma qualidade que estes são mostrados na tela. Como os tipos de diagramas são especificados no **EDG**, através do **MED**, o engenheiro de software pode descrever as notações diagramáticas que ele deseja, resolvendo o problema de adaptabilidade a notações impostas pelos **EDs**. Não foi possível, ainda, resolver os problemas de formatação automática no protótipo do **EDG**. Este problema deverá ser solucionado através de um futuro trabalho a ser desenvolvido no **CPGCC**.

O **EDG**, através de sua capacidade de geração de **ED**s para métodos baseados em grafos, serve como uma ferramente de suporte a geração de editores CASE.

E finalmente, através do **EDE**, diagramas podem ser editados, corrigidos e recuperados facilmente.

A implementação do protótipo do **EDG** foi realizada com auxílio de uma equipe de três alunos do curso de graduação em ciência da computação.

O **EDG** foi apresentado, através de artigo técnico publicado, em quatro congressos nacionais (VIII Congresso da Sociedade Brasileira de Computação, II Simpósio Brasileiro de Engenharia de Software, V Congresso Serpro de Informática e III Congresso Internacional do Software). O protótipo conta atualmente com aproximadamente 25.000 linhas de código fonte em PASCAL, e foi implementado em equipamento Macintosh.

8.2 Desenvolvimentos Futuros

Durante a elaboração deste trabalho surgiram quatro questões as quais podem ser melhor investigadas no futuro: (1) a formatação automática de diagramas, (2) gerente de hipertextos, (3) gerente de banco de dados, e (4) mecanismos para verificação sintática e semântica dos diagramas.

Através do **EDG** vários tipos de diagramas podem ser suportados. Diagramas em forma de árvores binárias, árvores n-árias, grafos direcionados e não direcionados são desenhados interativamente pelo usuário através das facilidades gráficas providas pelo **EDG**. A criação de nodos e ligações no **EDE** é responsabilidade do usuário. Como consequência disto, a construção de diagramas legíveis pode tornar-se uma tarefa difícil e consumidora de tempo. Para que o **EDG** supere está limitação é necessário estendê-lo com a capacidade de formatar os diagramas automaticamente. O uso de tal capacidade pode diminuir o custo de geração de diagramas, e prover uma efetiva integração da fase de concepção e produção de diagramas. Com o objetivo de formatar diagramas com boa legibilidade vários algoritmos tem sido propostos [BAT 85] [BAT 86a] [BAT 86b] [CHI 86] [MEY 83] [NAR 84] [SUP 83] [TAM 87a] TAM 87b] [WET 79], sendo que cada tipo de algoritmo trata uma classe especifica de grafos, adota um padrão gráfico particular e seus próprios aspectos estéticos de legibilidade. O **EDG** manipula todas as classes de grafos, isto é, árvores binárias, grafos planares, grafos hierárquicos e grafos não direcionados, logo é necessário estendê-lo com vários tipos de algoritmos para que ele possa suportar a formatação automática destes distintos tipos de grafos.

Hipertexto é uma maneira prática e eficiente de documentar várias porções de um sistema ou programa sem as restrições dos editores de textos lineares [CON 87]. A integração de *hipertexto* com o **EDG** poderá facilitar a documentação, permitindo que as decisões do projeto sejam ligadas com a documentação do programa e com o código fonte do mesmo. Um diagrama poderá possuir um **pedaço** (*node*) [BIG 88] [CON 87] do hipertexto associado a cada nó da árvore de diagramas. Desta forma, o **EDE** poderá fazer a gerência dos hipertextos também como uma árvore, onde as **ligações** (*links*) são os refinamentos dos diagramas. No **EDE**, um diagrama pode ter vários diagramas filhos, gerados pelos refinamentos dos nodos no diagrama pai. Assim, os diagramas podem ser detalhados até o código fonte do programa que implementará o problema. Com isto, toda a documentação textual vinculada às decisões até a solução poderá ser mantida através do *hipertexto* associado aos diagramas.

Os métodos descritos no **MED** são armazenados em um arquivo seqüencial, e os diagramas editados no **EDE** também são armazenados da mesma forma. Tal mecanismo é bastante deficiente para ser usado como interface com outras ferramentas. Para que o **EDG** possa ser integrado com outras ferramentas é necessário estendê-lo com um gerente de banco de dados. A experiência tem mostrado que os sistemas convencionais, baseados nos modelos de dados clássicos (relacional, rede, hierárquico), não são adequados para utilização de entidades complexas [DAT 81] [HAS 82], o que é o caso das entidades manipuladas pelo **EDG**. É necessário, então, que o **EDG** seja integrado a um gerente de banco de dados, que possa prover suporte a: estruturação hierárquica dos diagramas; tratamento de versões [BEC 88]; dados não estruturados, como, por exemplo, textos; e transações longas [KLA 85] [KIM 84] ou transações conversacionais [HAS 82] [LOR 82].

O **EDG** provê uma tabela de ligações possíveis para manter a consistência sintática dos diagramas editados. Embora este mecanismo seja simples e eficiente ele, por si só, não é suficiente para a manutenção da integridade sintática e semântica dos diagramas editados. Sendo assim, é necessário a criação de outros mecanismos para cumprir este objetivo.

Em [PRI 88], Price propõe que as verificações sintáticas e semânticas dos diagramas sejam feitas com o auxílio de gramática de atributos. Atualmente, tal mecanismo tem sido usado em editores dirigidos por sintaxe [PRI 84]. Outro método descrito em [MEL 88a], prevê a execução de regras em PROLOG. É necessário, então, experimentar estes e outros mecanismos para se verificar o mais adequado ao **EDG.**

BIBLIOGRAFIA

[AGU 87] AGUIAR, T.C.; BLASCHECK, J.R.; NOGUEIRA, D.; ROCHA, A.R.C.; Ferramentas automatizadas para apoiar a fase de análise de projeto. In: CONGRESSO NACIONAL de INFORMATICA, 20., São Paulo, Ago. 31-Set. 6, 1987. Anais. São Paulo, SUCESU, 1987. p. 1065-70.

[BAL 83] BALZER, R.; CHEATHAM, T.E.; GREEN, C. Software technology in the 1990's: using the new paradigm. Computer, Los Alamitos, 16(11):39-45, Nov. 1983.

[BAT 85] BATINI, C.; FURLANI, L.; NARDELLI, E. What is a good diagram? In: INT. CONF. on THE ENTITY RELATIONSHIP APPROACH, 4., Chicago, 28-30 Oct. 1985. Proceedings. Silver Spring, IEEE Comput. Soc., 1985, p. 312-19.

[BAT 86] BATINI, C.; NARDELLI, E.; TAMASSIA, R. A layout algorithm for data flow diagrams. Trans. Soft. Engineering, New York, SE-12(4):538-46, Apr. 1986.

[BEC 88] BECKER, Karin. Tratamento de versões em um ambiente de PAC para sistemas digitais. Porto Alegre, PGCC da UFRGS, 1988. (Dissertação de mestrado).

[BIG 88] BIGELOW, J. Hypertext and case. IEEE Software, Los Alamitos, 5(2):23-27, Mar. 1988.

[BOE 81] BOEHM, B.W. Software engineering economics. Englewood Cliffs, Prentice-Hall, 1981.

[BOE 83] BOEHM, B. & STANDISH, T.A. Software technology in the 1990s: Using an evolutionay paradigm. Computer, Los Alamitos, 16(11):30-7, Nov. 1983.

[BOE 84] BOEHM, B.; et al. A software development environment for improving productivity. Computer, Los Alamitos, 17(6):30-42, June 1984.

(4) [BOE 87] BOEHM, B. Improving software productivity. Computer, Los Alamitos, 26(9):43-57, Sept. 1987.

[BUR 87] BURCHETT, R.H. The need for automation of analysis. In: ANALYST workbenches. Oxford, Pergamon Infotech, 1987. p. 33-39. (State of the Art Report, 15:1).

[CAS 85] CASE, A.F. Computer-aided software engineering (CASE): technology for improving software development productivity. Data base, New York, 16(1):35:43, fall 1985.

[CHA 74] CHAPIN, N. New format for flowcharts. Software - practice and experience, Sussex, 4(4):341-57, Oct.-Dec. 1974.

[CHE 77] CHEN, Peter Pin-Shan. The entity-relationship mode - toward a unified view of data. ACM Trans. on Database Systems, New York, 1(1):9-36, Mar. 1977.

[CHI 85] CHIBA, N.; ONOGUCHI, K.; NISHIZEKI, T. Drawing planar graphs nicely. Acta Informatica, Berlin, 22(2):187-201, June1985.

(5) [CHI 88a] CHIKOFSKY, E. J. Software techonology people can really use. IEEE Software, Los Alamitos, 5(2):8-10, Mar. 1988.

(6) [CHI 88b] CHIKOFSKY, E.J. & RUBENTEIN, B.L. Case: reliability engineering for information systems. IEEE Software, Los Alamitos, 5(2):8-10, Mar. 1988.

[CON 87] CONKLIN, J. Hypertext: an introduction and survey. Computer, Los Alamitos, 20(9):17-41, Sept. 1987.

[COU 78] COUGAR, D. & ZAWACKI, R. What motivates DP professionals. Datamation, New York, 24(9):116-123, Sept. 1978. In: [FAI 85].

[DAT 81] DATE, C.J. An introduction to data base systems. 3. ed. Massachussets, Addison-Wesley, 1981. v. 1.

[ENN 86] ENNS, Steve. Free-form curves on your micro. BYTE, Peterborough, 11(13):225-30, Dec. 1986.

[FAI 85] FAIRLEY, R.E. Software engineering concepts. Singapore, MacGraw-Hill, 1985.

[FIS 88] FISHER, Alan S. CASE: using the newest tools in software development. New York, Wiley, 1988.

[GAN 83] GANE, Chris. Análise estruturada sistemas. Rio de Janeiro, LTC, 1983.

[HAS 82] HASKIN, R.L. & LORIE, R. A. On extending the functions of a relational database system. In: ACM SIGMOD INTERNATIONAL CONFERENCE on MANAGEMENT of DATA, Orlando, June 2-4, 1982. Proceedings. New York, ACM, 1982. p. 207-12.

[HEA 86] HEARN, Donald & BAKER, Pauline. Computer graphics. Englewood Cliffs, Prentice-Hall, 1988.

[HEU 87] HEUSER, Carlos Alberto. Análise estruturada de sistemas com redes de Petri. In: CONGRESSO NACIONAL de INFORMATICA, 20., São Paulo, Ago. 31-Set. 6, 1987. Anais. São Paulo, SUCESU, 1987. p. 668-75.

[IES 86] IESA - Tecnologia de Sistemas Ltda. Mosaico - Manual do usuário. Rio de Janeiro, Set. 1986.

[INS 85] INSide Macintosh. Reading, Addison-Wesley. 1985. v. 1, 2 e 3.

[JAC 75] JACKSON, M. Principles of program design. London, Academic Press, 1975.

[KIM 84] KIM, W. et al. A transaction mechanism for engineering design databases. In: INTERNATIONAL CONFERENCE on VERY LARGE DATA BASE, 10., Singapore, Oct. 28-31, 1984. Proceedings. Singapore, ACM, 1985. p. 355-62.

[KLA 85] KLAHOLD, P.; SCHLAGETER, G.; WILKES, W. A general model for version management in databases. IN: ACM SIGMOD INTERNATIONAL CONFERENCE on MANAGEMENT of DATA. Austin, May 28-31, 1985. Proceedings. New York, ACM, 1985. p. 388-401.

[LEA 87] LEANDRO, M.A.C. & NUNES, D. Ambiente de desenvolvimento de software utilizando a metodologia SADT. In: SIMPOSIO BRASILEIRO de ENGENHARIA de SOFTWARE, 1., Petropólis, Out. 23-23, 1987. Anais. Petrópolis, SBC, 1987. p. 100-2.

[LOR 82] LORIE, R. & PLOUFFE, W. Complex objects and their use in design transactions. San Jose, IBM, 1982. RJ 3706.

[LUQ 88] LUQI & KETABCHI, M. A computer-aided prototyping system. IEEE Software, Los Alamitos, 5(2):66-72, Mar. 1988.

[MAC 87] MACDONALD, I.G. Automating the information engineer - IPSEs and Workbenches in context. In: ANALYST workbenches. Oxford, Pergamon Infotech, 1987. p. 123-45. (State of the Art Report, 15:1).

[MAC 88] MACNURLIN, Barbara Carnning. Improving large application development. I/S Analyzer, Maryland, 26(3):1-12, Mar. 1988.

[MAR 82] MARTIN, James. Application development without programmer. Englewood Cliffs, Prentice-Hall, 1982.

[MAR 85a] MARTIN, J. & MACCLURE, C. Diagramming techniques for analysts and programmers. Englewood Cliffs, Prentice-Hall, 1985.

[MAR 85b] MARTIN, J. & MACCLURE, C. Techniques computing. Englewood Cliffs, Prentice-Hall, 1985.

[MAR 88] MARTIN, F. C. Second-generation CASE tools: a challenge to vendors. IEEE Software, Los Alamitos, 5(2):46-49, Mar. 1988.

[MEY 83] MEYER, C. A browser for directed graphs. Berkeley, Univ. of California, EECS Dept. 1983. M.S. project report.

[MEL 87] MELLO, R. N. Um ambiente de ferramentas integradas para desenvolvimento de sistemas interativos. In: SIMPOSIO BRASILEIRO de ENGENHARIA de SOFTWARE, 1., Petropólis, Out. 22-23, 1987. Anais. Petrópolis, SBC, 1987. p. 59-69.

[MEL88a] MELO, W.L.M. & PRICE, R. T. Considerações sobre um editor gráfico adaptável a notações diagramáticas. Porto Alegre, CPGCC/UFRGS, fev. 1988. (não publicado)

[MEL 88b] MELO, W.L.M. & PRICE, R.T. O editor gráfico generalizado. In: CONGRESSO da SOCIEDADE BRASILEIRA de COMPUTAÇÃO, 8., Rio de Janeiro, Julho 17-22, 1988. Anais. Rio de Janeiro, SBC, 1988. p. 56-66.

[MEL 88c] MELO, W.L.M. & PRICE, R.T. Implementação de um editor de diagramas generalizado. In: SIMPOSIO BRASILEIRO de ENGENHARIA de SOFTWARE, 2., Canela, Out. 27-28, 1988. Anais. Canela, Departamento de Informática da UFRGS, 1988. p. 123-134.

[MEL 88d] MELO, W.L.M. & PRICE, R.T. Aumentando a produtividade através de um editor diagramático generalizado. In: CONGRESSO SERPRO de INFORMATICA, 5., Brasília, Nov. 7-10, 1988. Anais. Brasília, SERPRO, 1988. p. 56-86.

[MEL 89] MELO, W.L.M. & PRICE, R.T. EDG: editor diagramático generalizado: uma ferramenta de CASE. In: CONGRESSO INTERNACIONAL da TECNOLOGIA do SOFTWARE TELEMATICA e INFORMAÇÃO, 3., Rio de Janeiro, Mar. 14-17, 1989. Anais. Rio de Janeiro, FAIR Feiras e empreendimentos LTDA, 1989. p. 70-89.

[NAR 84] NARDELLI, E. & TALAMO, M. A fast algorithm for planarization of sparse diagrams. Rome, 1984. Technical report R. 105, IASI-CNR.

[NAS 73] NASSI, I. & SHNEIDERMAN, B. Flowchart techniques for structured programming. ACM SIGPLAN Notices, New York, 8(8):12-26, Aug. 1973.

[POR 85] PORTER, M. Competiteve advantage. New York, Free Press, 1985.

[PRI 84] PRICE, R. T., A prototype syntax driven language editor. Brighton, University of Sussex, 1984. (Tese de doutorado).

[PRI 85] PRICE, R. T. De um editor dirigido por sintaxe a um ambiente de desenvolvimento de software. In: CONGRESSO NACIONAL de INFORMATICA, 18., São Paulo, Out. 28-31, 1985. Anais. São Paulo, SUCESU, 1985. p. 645-59.

[PRI 87] PRICE, Roberto Tom & FAVERO, Eloi L. Uma introdução a linguagem de especificação do ambiente de desenvolvimento de software. IN: CONGRESSO da SOCIEDADE BRASILEIRA de COMPUTAÇÃO, 7., Salvador, Julho 17-22, 1987. Anais. Salvador, SBC, 1987. p. 79-96.

[PRI 88] PRICE, Roberto Tom & FAVERO, Eloi L. Editores diagramáticos baseados em formalismos gramaticais. IN: JORNADAS ARGENTINAS de INFORMATICA e INVESTIGACIONS, 17., Buenos Aires, Sept. 26-30, 1988. Anais. Buenos Aires, SADIO, 1988. p. 509-27

[REY 87] REYNOLDS, Lou. CASE tools boost productivity in any environment. Computer Design, Littleton, 26(1):60, Jan. 1987.

[SCH 81] SCHINDLER, M. The software decade. Electronic Design, New York, 29(1):190-9, Jan. 1981.

[SHO 87] SHORT, K.W. Analyst workbenches- design and other mapping issues. In: ANALYST workbenches. Oxford, Pergamon Infotech, 1987. p. 147-161. (State of the Art Report, 15:1).

[SOR 88] SORENSON, P.G.; TREMBLAY, J.P.; MACALLISTER, A.J. The metaview for many specification environments. IEEE Software, Los Alamitos, 5(2):30-38, Mar. 1988.

[STE 85] STEVENS, Wayne P. Projeto Estruturado de Sistemas. Rio de Janeiro, Campus, 1985.

[SUP 83] SUPOWIT, K. & REINGOLD, E. The complexity of drawing trees nicely. Acta Informatica, Berlin, 18(4):377-92, Jan.1983.

[SUY 87] SUYDAM, W. CASE makes strides toward automated software development. Computer Design, Littleton, 26(1):49-70, Jan. 1987.

[TAM 87] TAMASSIA, R. On embedding a graph in the grid with the minimum number of bends. SIAM Journal on Computing, New York, 16(3):421-44, June 1987.

[THE 87] THE SCOPE of workbench products. In: ANALYST workbenches. Oxford, Pergamon Infotech, 1987. p. 199-313. (State of the Art Report, 15:1).

[TRA 86] TRAINA Jr., C.; et al. SIPS - estado atual de desenvolvimento. In: CONGRESO NACIONAL de INFORMATICA, 19., São Paulo, Out. 23-26, 1986. Anais. São Paulo, SUCESU, 1986. p. 145-60.

[WAR 85] WARNIER, Jean-Dominique. Guia dos usuários de sistemas de informação. Rio de Janeiro, Campus, 1985.

[WET 79] WETHERELL, C & SHANNON, A. Tidy drawing of trees. IEEE Trans. on Software Engineering, New York, 5(5):514-20, Sept. 1979.

[WHA 87] WHAT is an analyst wokbench? In: ANALYST workbenches. Oxford, Pergamon Infotech, 1987. p. 169:197. (State of the Art Report, 15:1).

[YOU 74] YOURDON, Edward & CONSTANTINO, Larry L. Structured Design. Englewood Cliffs, Printice-Hall, 1974.

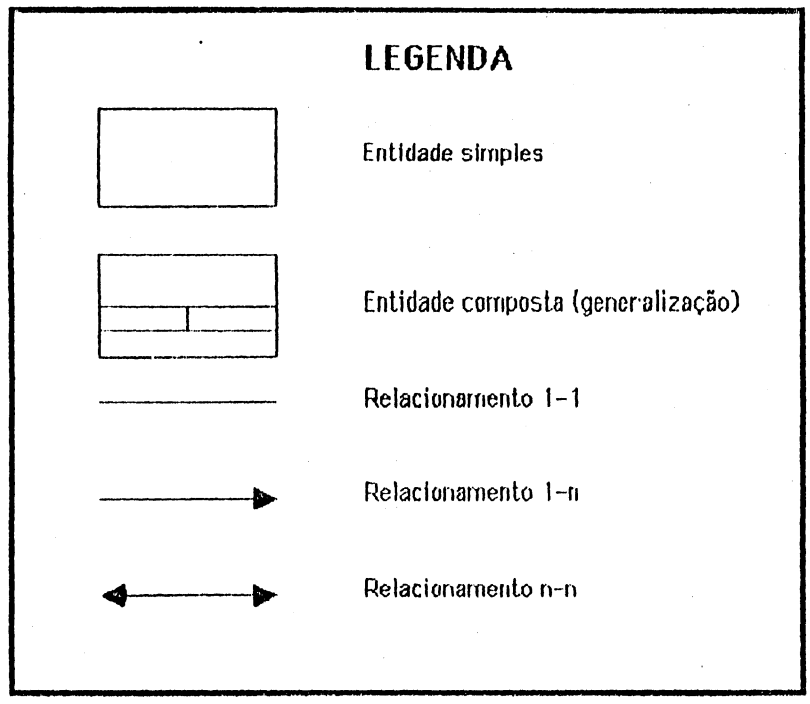
[YOU 86] YOURDON, Edward. What ever happened to structured analysis? DATAMATION, New York, 32(11):133-8, June 1986.

ANEXO 1

ESTRUTURA DE DADOS DO EDG

Este anexo mostra a estrutura de dados do **EDG** através de diagramas de entidades. Primeiro são mostradas as entidades manipuladas pelo **MED**, e depois as manipuladas pelo **EDE**.

O diagrama utiliza a seguinte notação:



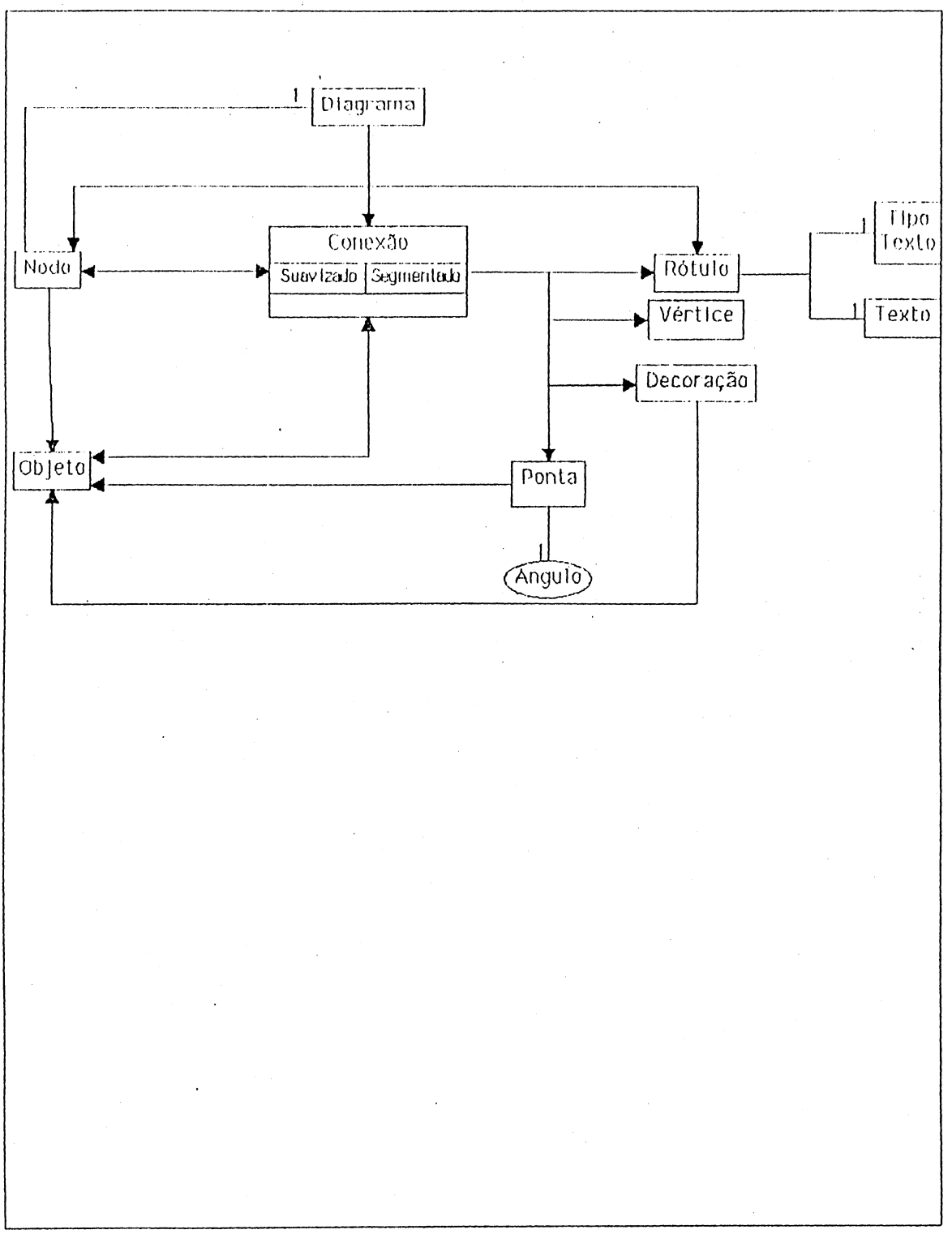
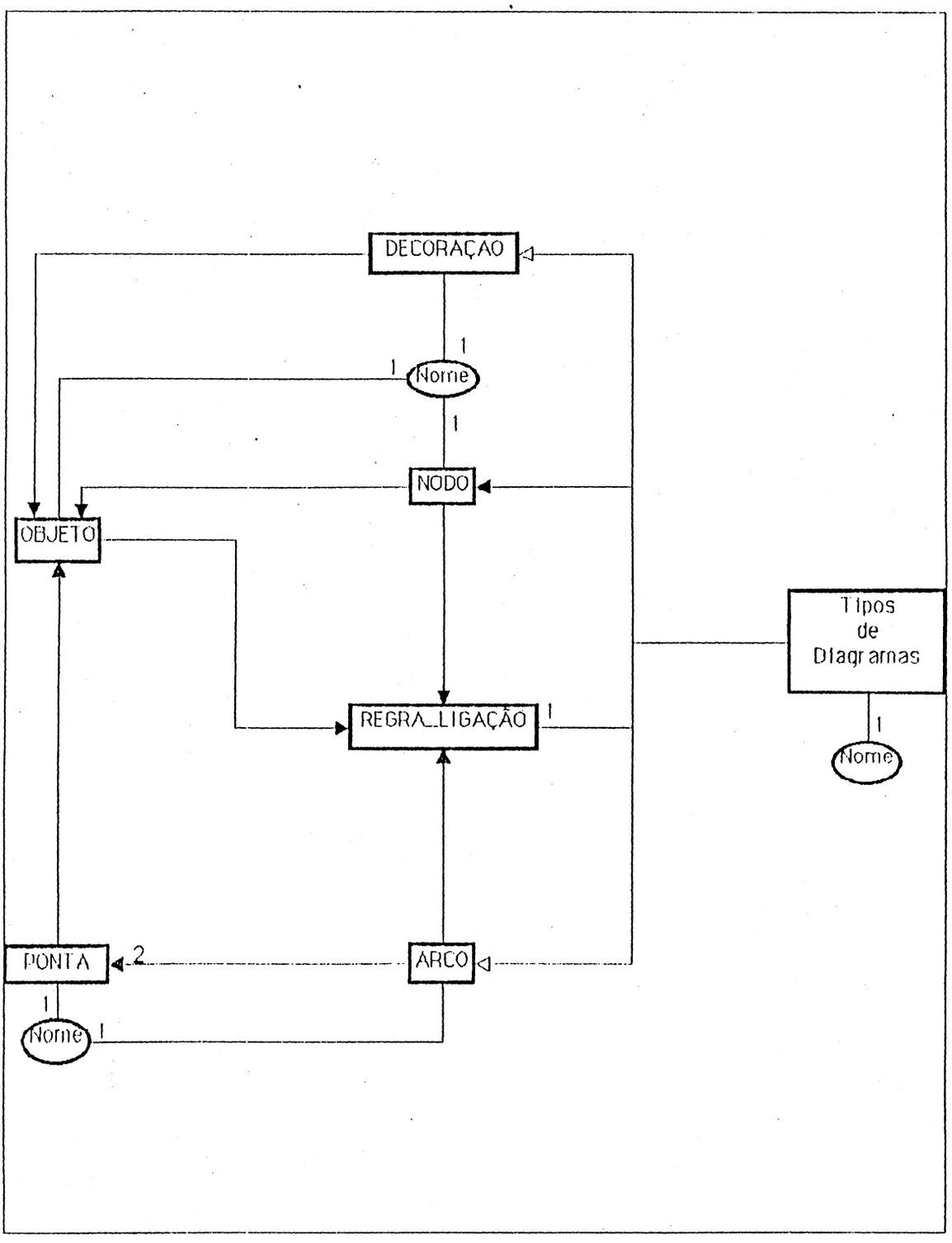


Diagrama
Nodo
Conexão
Suavizado
Segmentado
Rótulo
Tipo Texto
Texto
Vértice
Decoração
Objeto
Ponta
Angulo

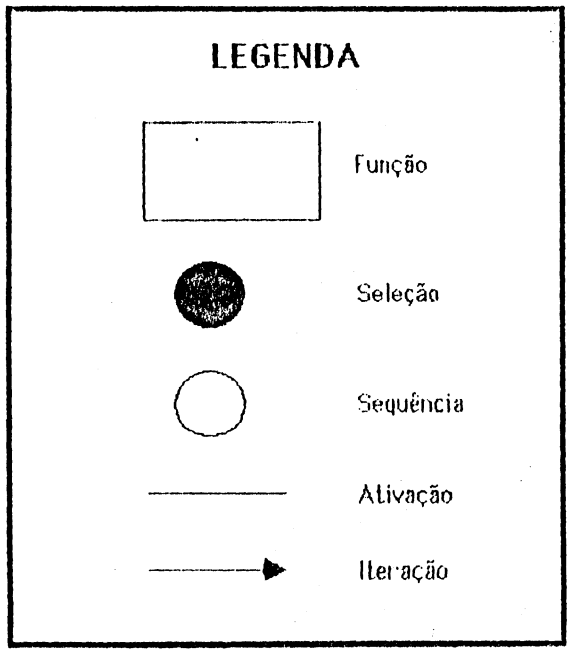

DECORAÇAO
1
1
Nome
1
NODO
OBJETO
Tipos
de
Diagramas
REGRA_LIGAÇÃO
1
1
Nome
PONTA
2
ARCO
1
Nome
1

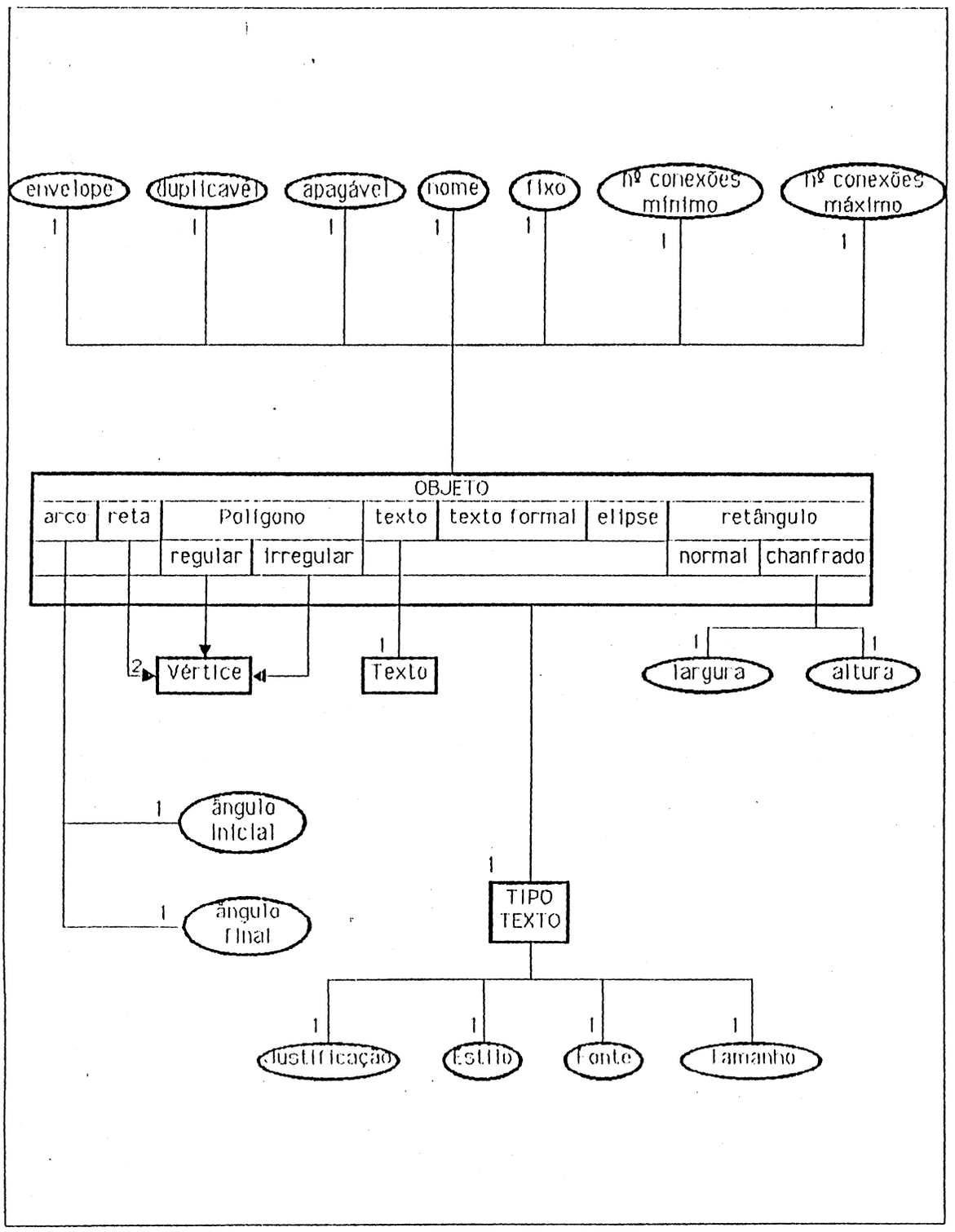
ANEXO 2

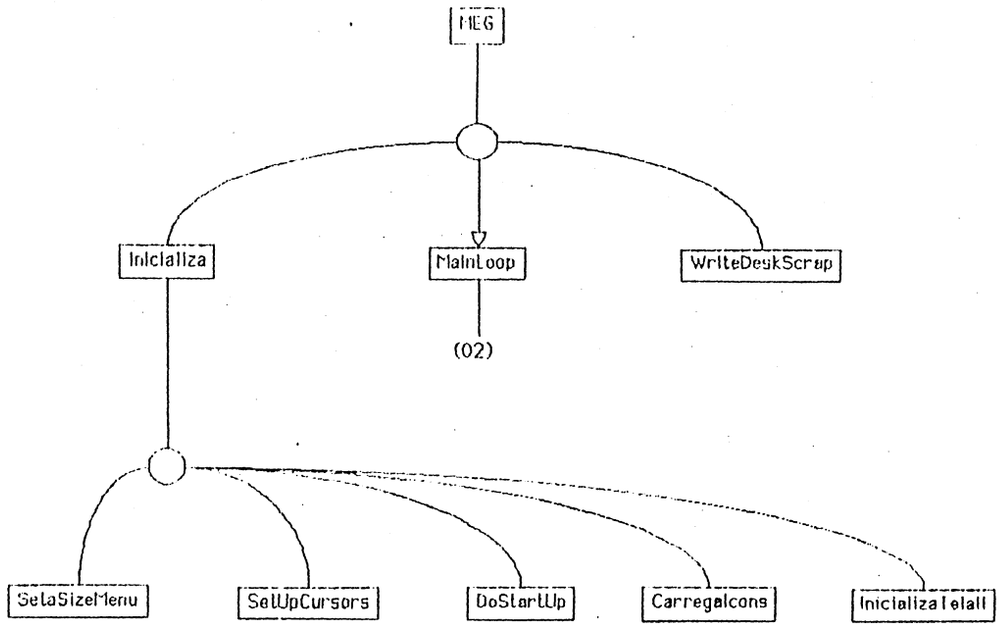
DIAGRAMAS DO PROJETO DO MED

Este anexo mostra os diagramas de decomposição funcional do **MED**.

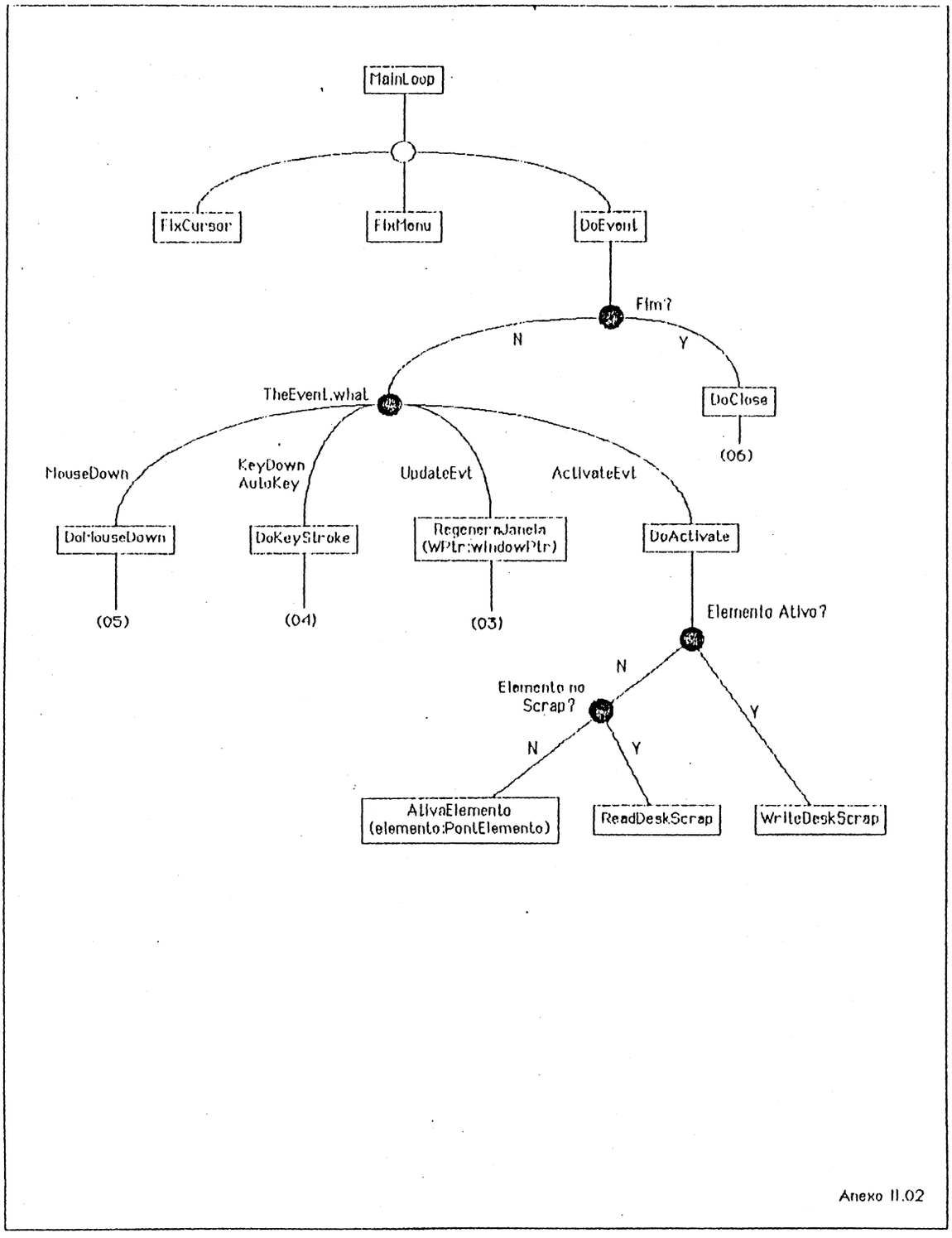
Os diagramas utilizam-se da seguinte notação:



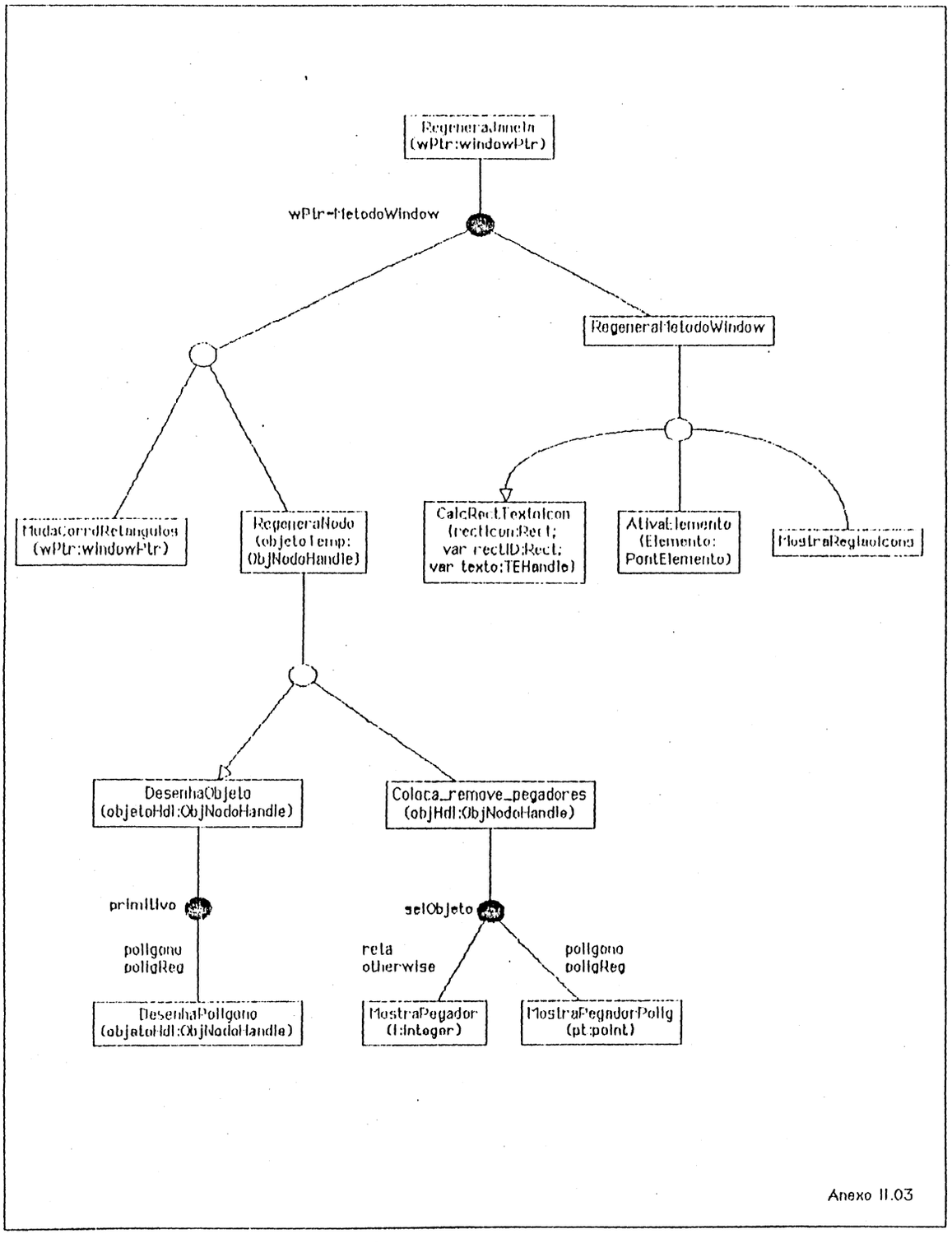

envelope
duplicável
apagável
nome
fixo
nº conexões mínimo
nº conexões máximo
1
OBJETO
arco
reta
Polígono
regular
irregular
texto
texto formal
elipse
retângulo
normal
chanfrado
2
Vértice
Texto
largura
altura
ângulo inicial
ângulo final
TIPO TEXTO
Justificação
Estilo
Fonte
Tamanho



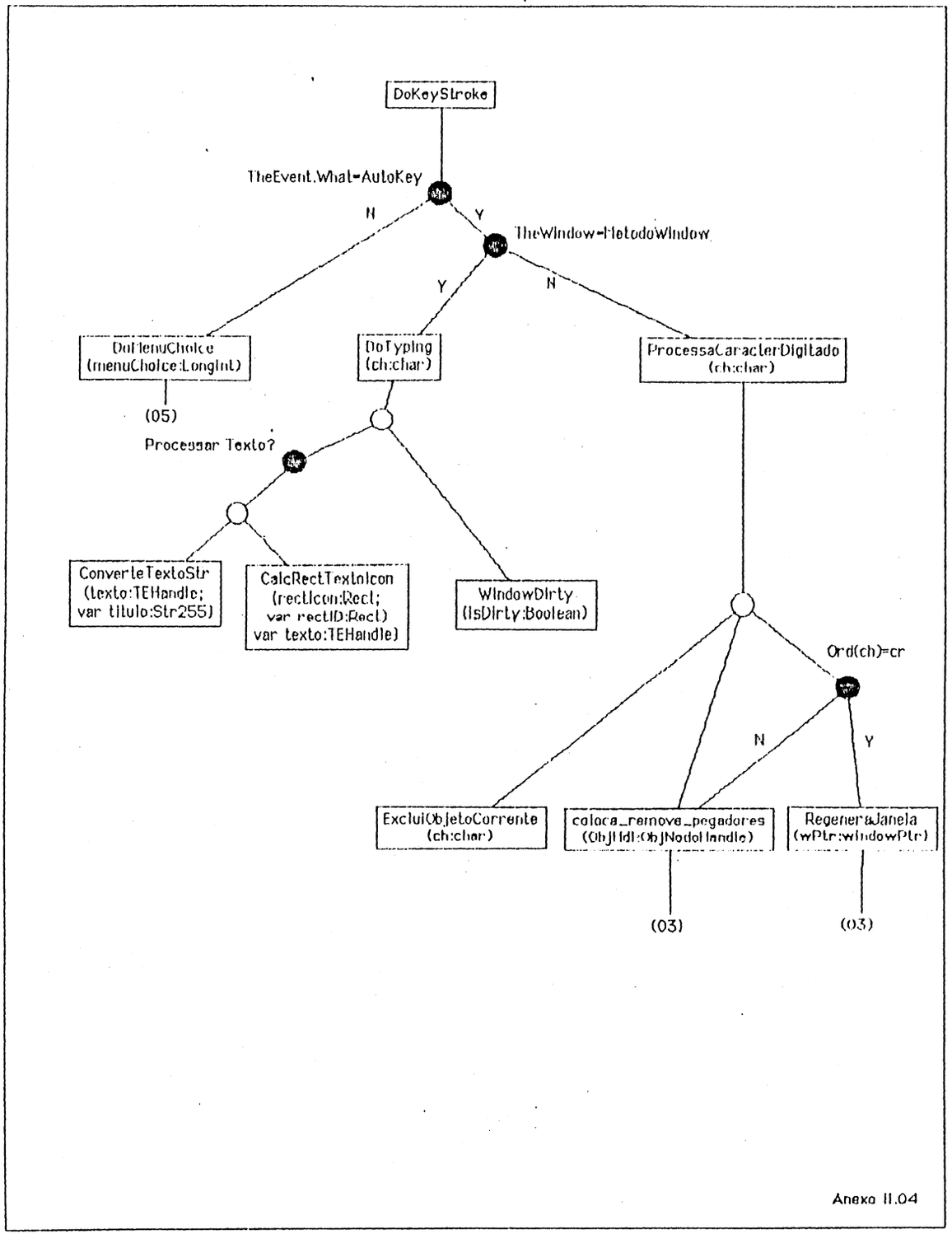
Anexo II.01


MainLoop
FixCursor
FixMenu
DoEvent
Fim?
N
Y
TheEvent.what
DoClose
(06)
MouseDown
KeyDown
AutoKey
UpdateEvt
ActivateEvt
DoMouseDown
DoKeyStroke
RegenerarJanela
(WPtr:WindowPtr)
DoActivate
(05)
(04)
(03)
Elemento Ativo?
N
Elemento no
Scrap?
Y
N
Y
AtivaElemento
(elemento:PontElemento)
ReadDeskScrap
WriteDeskScrap

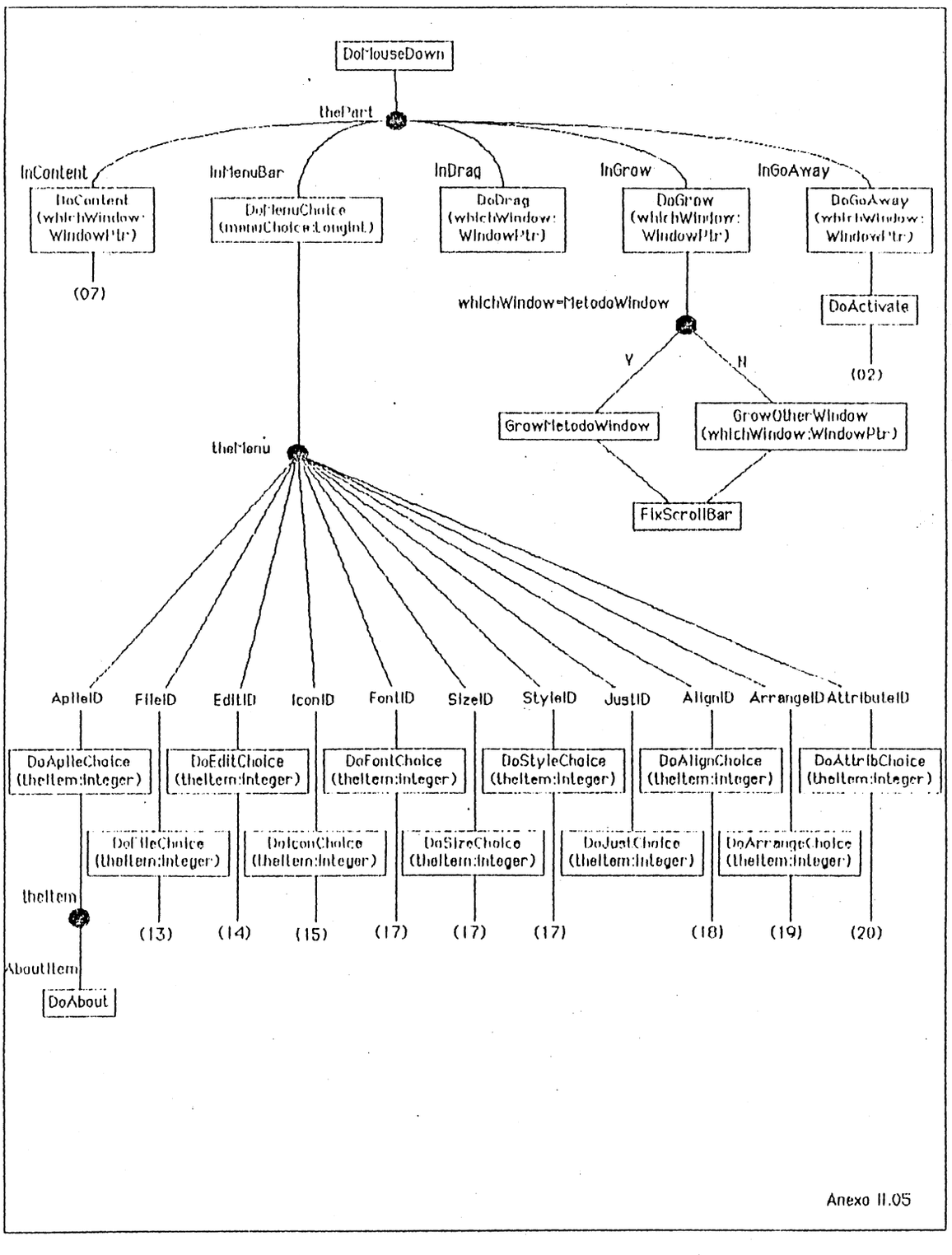
Anexo II.02


RegeneraJanela
(wPtr:windowPtr)
wPtr=MetodoWindow
RegeneralMetodoWindow
MudaCorrdRetangulos
(wPtr:windowPtr)
RegeneraNodo
(objetoTemp:
ObjNodoHandle)
CalcRectTextoIcon
(rectIconRect;
var rectID:Rect;
var texto:TEHandle)
AtivaElemento
(Elemento:
PontElemento)
MostraReglaoIcone
DesenhaObjeto
(objetoHdl:ObjNodoHandle)
Coloca_remove_pegadores
(objHdl:ObjNodoHandle)
primitivo
selObjeto
poligono
poligReg
reta
otherwise
poligono
poligReg
DesenhaPoligono
(objetoHdl:ObjNodoHandle)
MostraPegador
(i:integer)
MostraPegadorPolig
(pt:point)

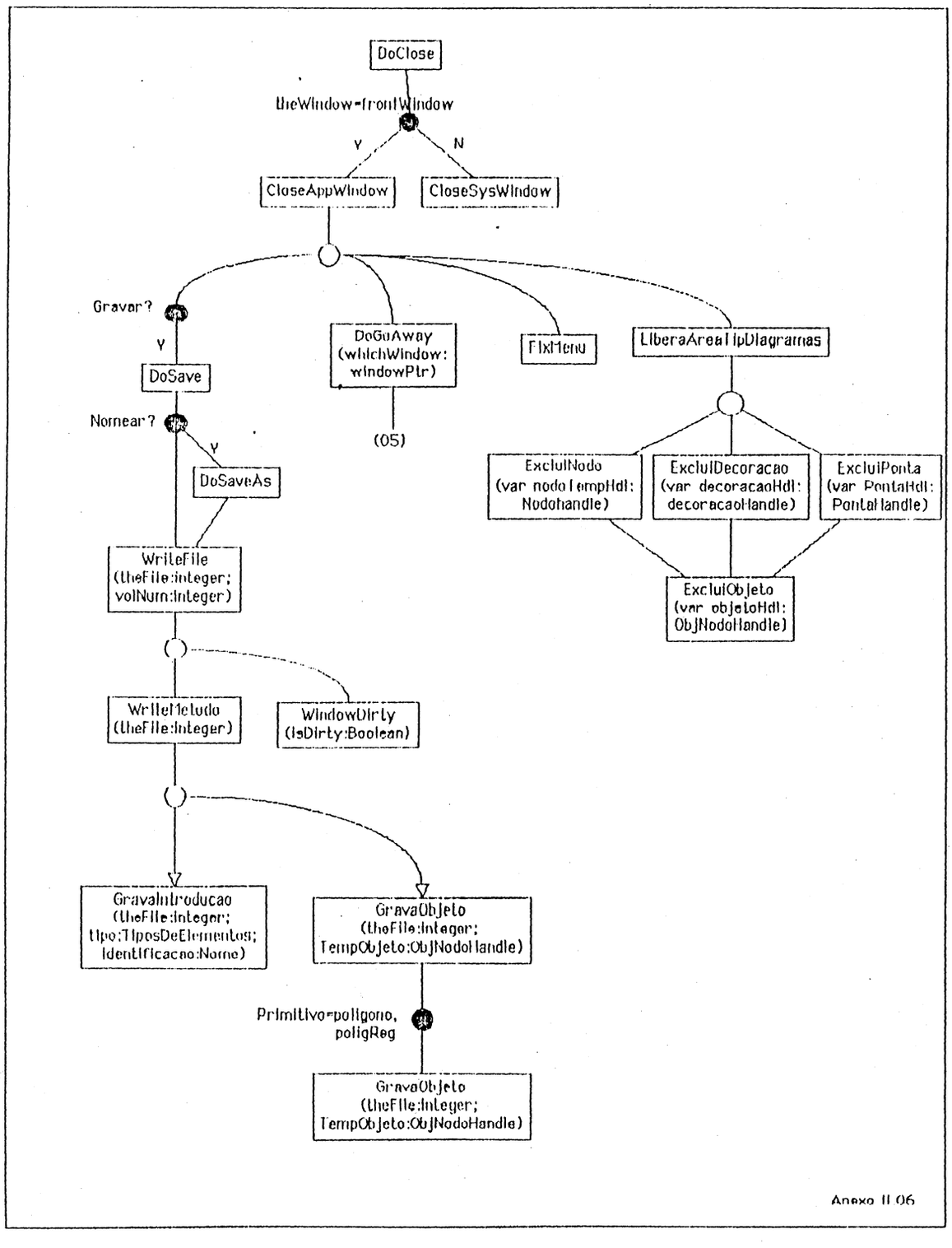
Anexo II.03


DoKeyStroke
TheEvent.What=AutoKey
N
Y
TheWindow=MetodoWindow
Y
N
DoMenuChoice
(menuChoice:LongInt)
DoTyping
(ch:char)
ProcessaCaracterDigitado
(ch:char)
(05)
Processar Texto?
ConverteTextoStr
(texto:TEHandle;
var titulo:Str255)
CalcRectTextoIcon
(rectIcon:Rect;
var rectID:Rect)
var texto:TEHandle)
WindowDirty
(IsDirty:Boolean)
Ord(ch)=cr
N
Y
ExcluiObjetoCorrente
(ch:char)
coloca_remove_pegadores
(ObjHdl:ObjNodeHandle)
RegenerarJanela
(wPtr:WindowPtr)
(03)
(03)

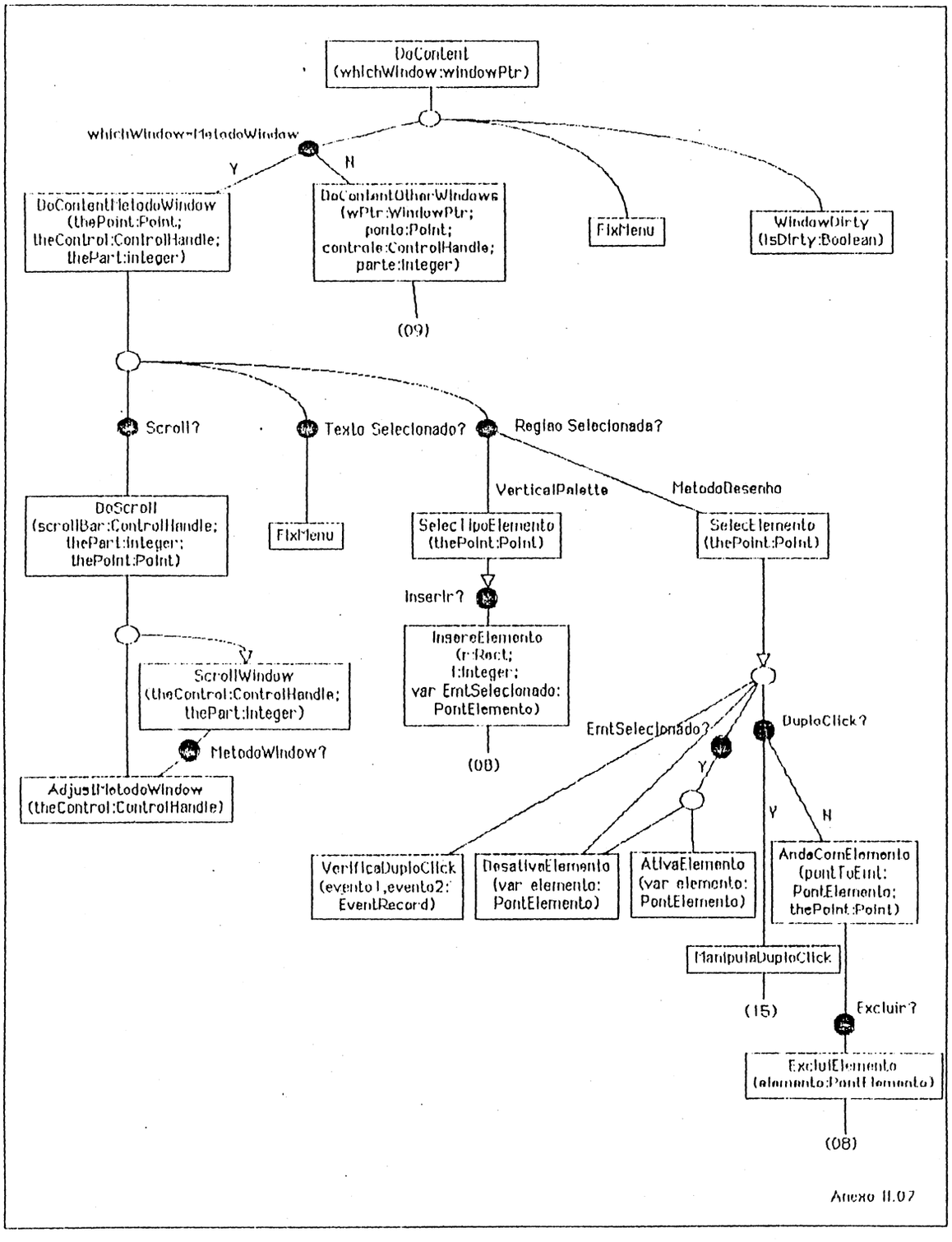
Anexo II.04


DoMouseDown
thePart
InContent
InMenuBar
InDrag
InGrow
InGoAway
DoContent (whichWindow: WindowPtr)
DoMenuChoice (menuChoice:LongInt)
DoDrag (whichWindow: WindowPtr)
DoGrow (whichWindow: WindowPtr)
DoGoAway (whichWindow: WindowPtr)
(07)
whichWindow=MetodoWindow
DoActivate
(02)
Y
N
GrowMetodoWindow
GrowOtherWindow (whichWindow:WindowPtr)
FixScrollBar
theMenu
ApMeID
FileID
EditID
IconID
FontID
SizeID
StyleID
JustID
AlignID
ArrangeID
AttributeID
DoApMeChoice (theItem:Integer)
DoEditChoice (theItem:Integer)
DoFontChoice (theItem:Integer)
DoStyleChoice (theItem:Integer)
DoAlignChoice (theItem:Integer)
DoAttribChoice (theItem:Integer)
DoFileChoice (theItem:Integer)
DoIconChoice (theItem:Integer)
DoSizeChoice (theItem:Integer)
DoJustChoice (theItem:Integer)
DoArrangeChoice (theItem:Integer)
theItem
(13)
(14)
(15)
(17)
(17)
(17)
(18)
(19)
(20)
AboutItem
DoAbout

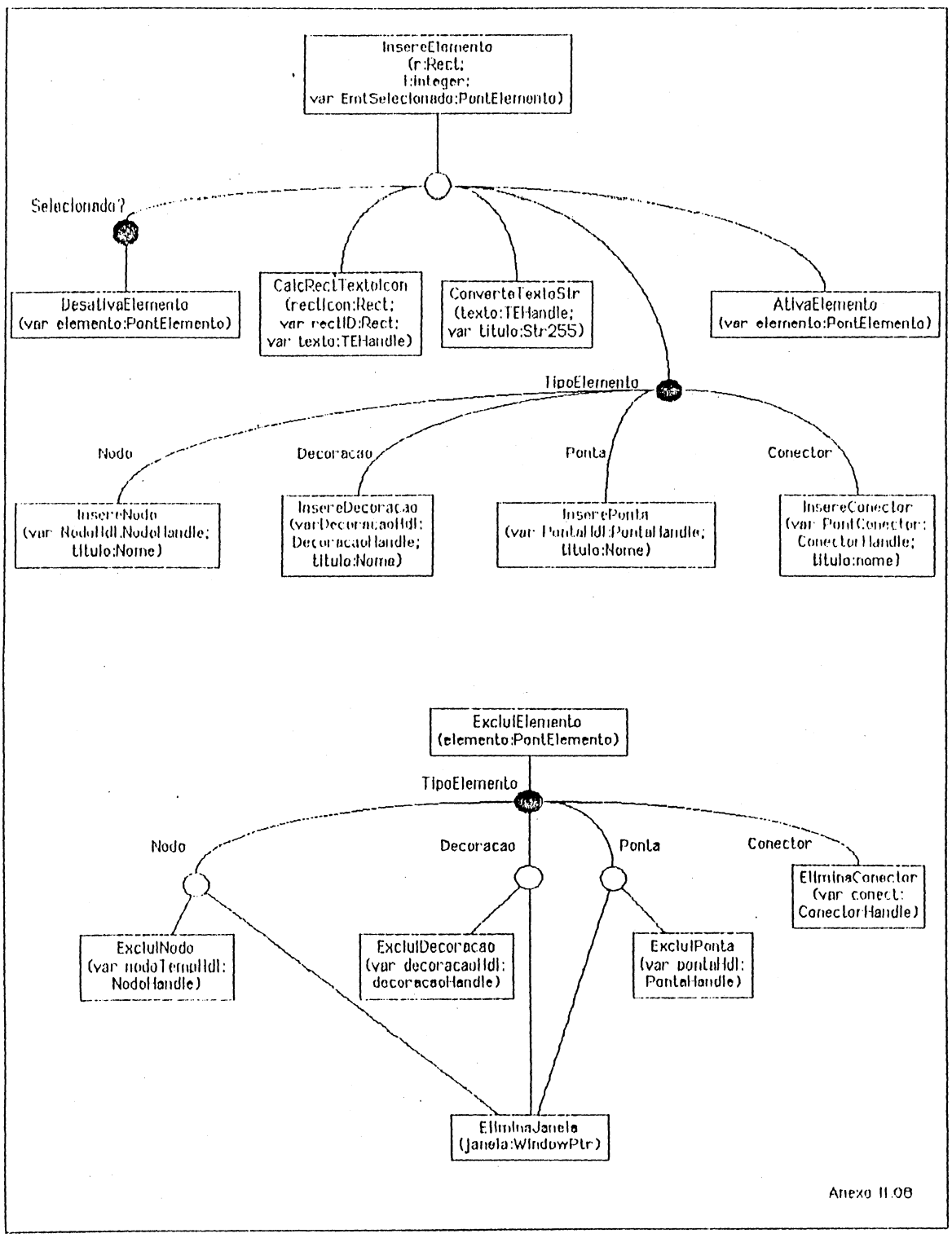
Anexo II.05


DoClose
theWindow=frontWindow
Y
N
CloseAppWindow
CloseSysWindow
Gravar?
Y
DoSave
Nomear?
Y
DoSaveAs
WriteFile
(theFile:integer;
volNum:integer)
WriteMeludo
(theFile:integer)
WindowDirty
(isDirty:Boolean)
GravaIntroducao
(theFile:integer;
tipo:TiposDeElementos;
identificacao:Nome)
GravaObjeto
(theFile:integer;
TempObjeto:ObjNodoHandle)
Primitivo=poligono,
poligReg
GravaObjeto
(theFile:integer;
TempObjeto:ObjNodoHandle)
DoGoAway
(whichWindow:
windowPtr)
(05)
FixMenu
LiberaAreaTipDiagramas
ExcluiNodo
(var nodoTempHdl:
NodoHandle)
ExcluiDecoracao
(var decoracaoHdl:
decoracaoHandle)
ExcluiPonta
(var PontaHdl:
PontaHandle)
ExcluiObjeto
(var objetoHdl:
ObjNodoHandle)

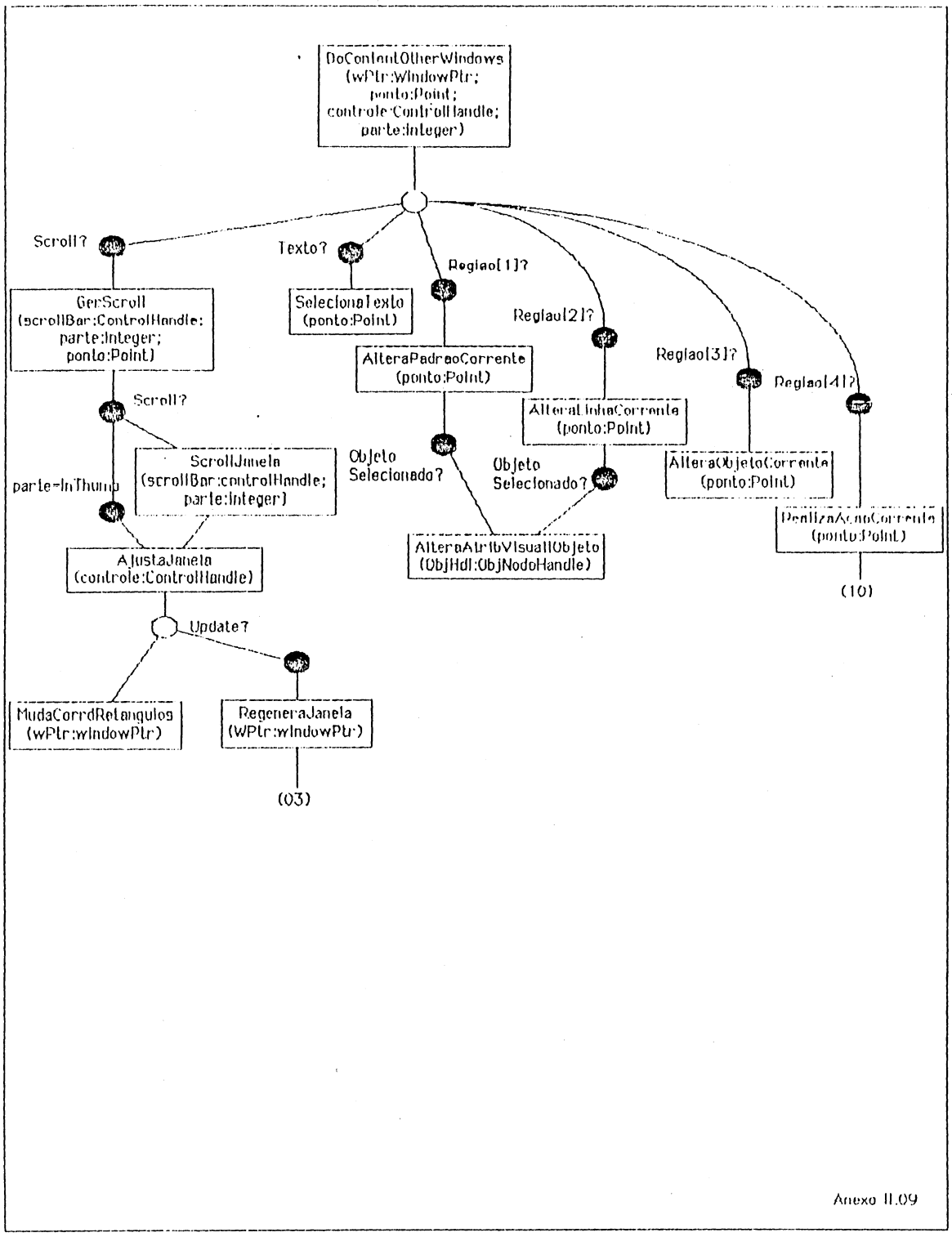
Anexo II.06


DoContent
(whichWindow:windowPtr)
whichWindow=MetodoWindow
Y
N
DoContentMetodoWindow
(thePoint:Point;
theControl:ControlHandle;
thePart:integer)
DoContentOtherWindows
(wPtr:WindowPtr;
ponto:Point;
controle:ControlHandle;
parte:Integer)
FixMenu
WindowDirty
(IsDirty:Boolean)
(09)
Scroll?
Texto Selecionado?
Regiao Selecionada?
VerticalPalette
MetodoDesenho
DoScroll
(scrollBar:ControlHandle;
thePart:Integer;
thePoint:Point)
FixMenu
SelectTipoElemento
(thePoint:Point)
SelectElemento
(thePoint:Point)
Inserir?
InsereElemento
(r:Rect;
i:Integer;
var EntSelecionado:
PontElemento)
(08)
ScrollWindow
(theControl:ControlHandle;
thePart:Integer)
MetodoWindow?
AdjustMetodoWindow
(theControl:ControlHandle)
EntSelecionado?
DuploClick?
Y
Y
N
VerificaDuploClick
(evento1,evento2:
EventRecord)
DesativaElemento
(var elemento:
PontElemento)
AtivaElemento
(var elemento:
PontElemento)
AndaComElemento
(pontToEnt:
PontElemento;
thePoint:Point)
ManipulaDuploClick
(15)
Excluir?
ExcluiElemento
(elemento:PontElemento)
(08)

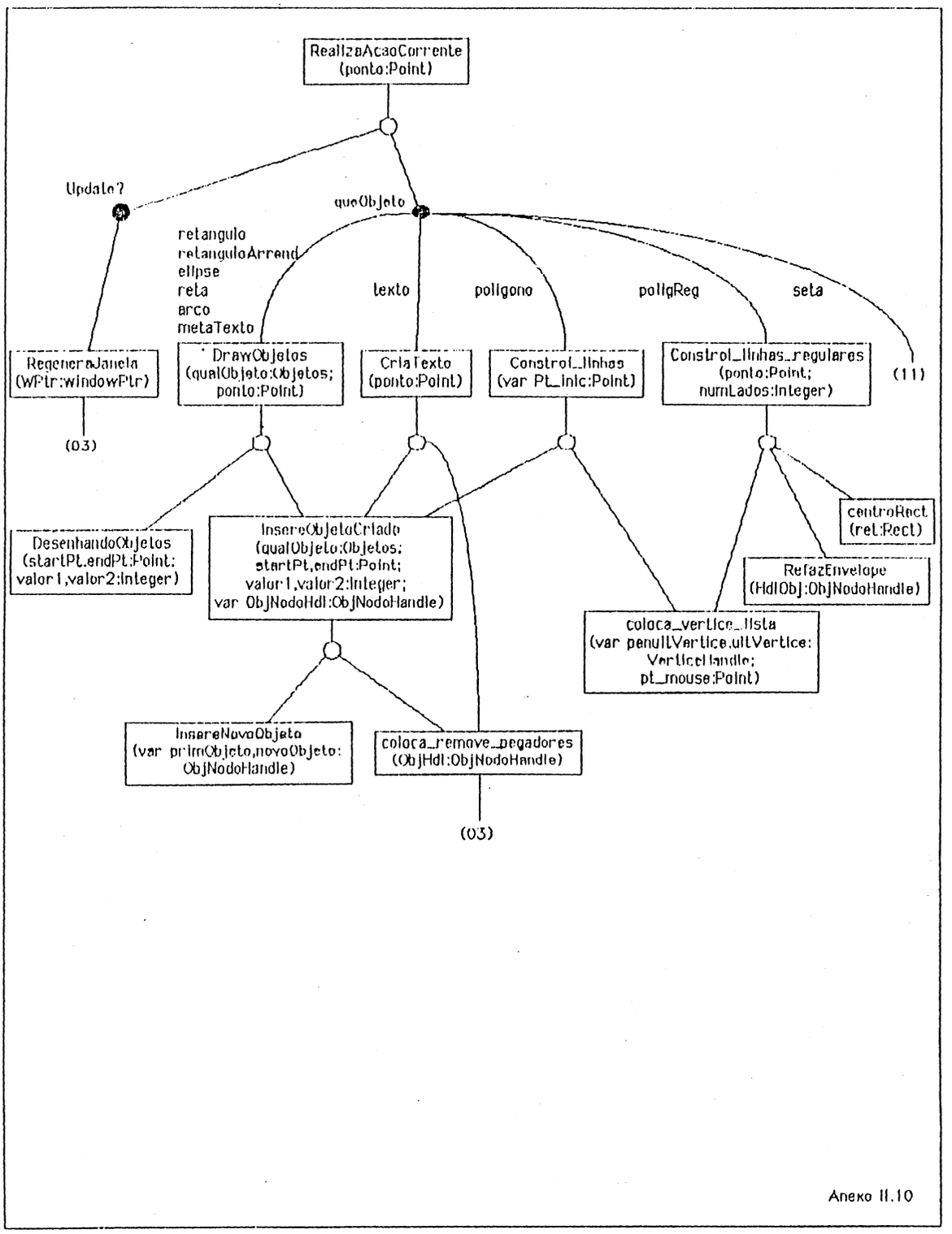
Anexo II.07



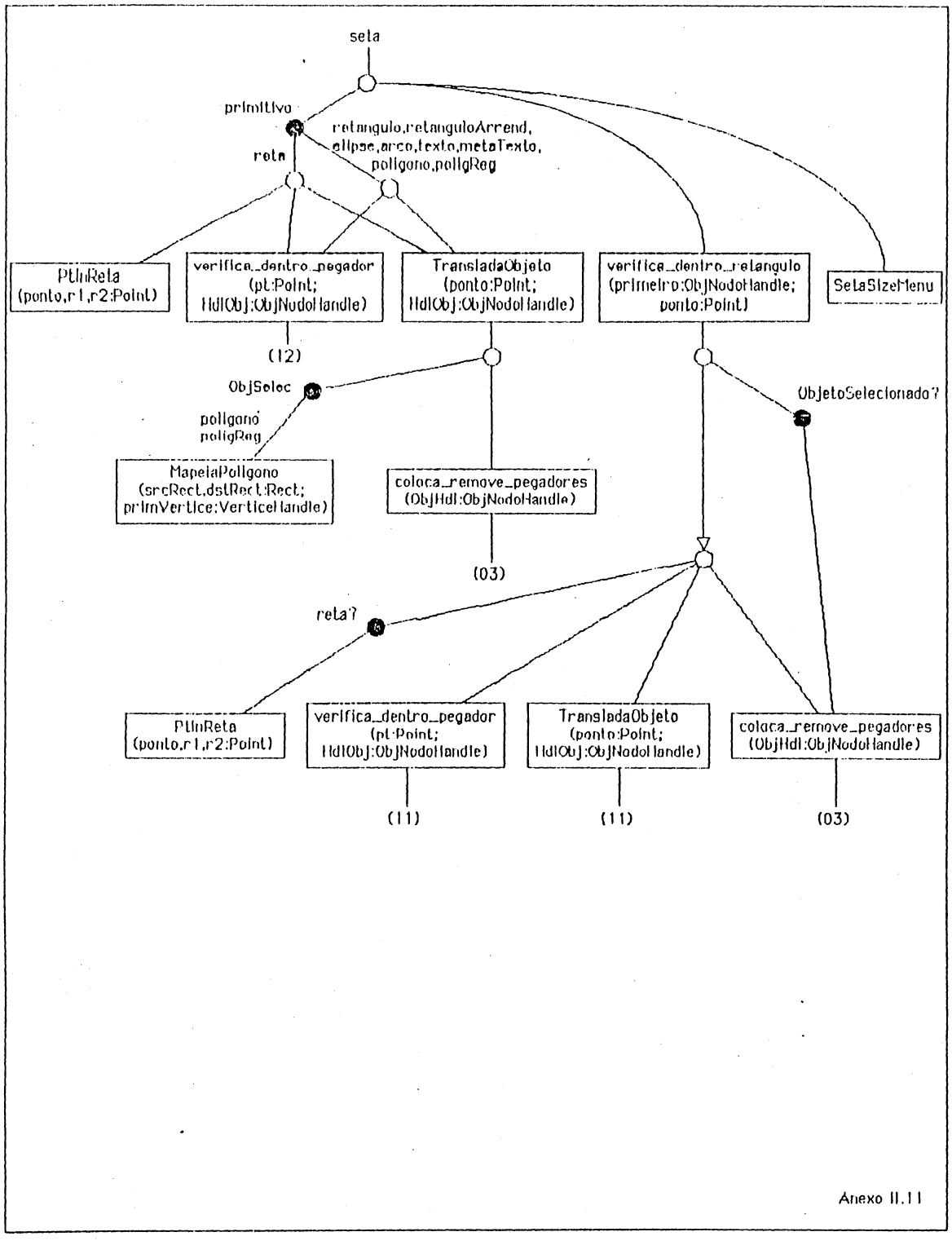
Anexo II.08


DoContentOtherWindows
(wPtr:WindowPtr;
ponto:Point;
controle:ControlHandle;
parte:Integer)
Scroll?
Texto?
Regiao[1]?
Regiao[2]?
Regiao[3]?
Regiao[4]?
GerScroll
(scrollBar:ControlHandle;
parte:Integer;
ponto:Point)
SelecionaTexto
(ponto:Point)
AlteraPadraoCorrente
(ponto:Point)
AlteraLinhaCorrente
(ponto:Point)
AlteraObjetoCorrente
(ponto:Point)
RealizaAcaoCorrente
(ponto:Point)
Scroll?
parte=InThumb
ScrollJanela
(scrollBar:controlHandle;
parte:Integer)
Objeto
Selecionado?
Objeto
Selecionado?
AlteraAtribVisualObjeto
(ObjHdl:ObjNodoHandle)
AjustaJanela
(controle:ControlHandle)
Update?
MudaCorrdRetangulos
(wPtr:WindowPtr)
RegeneraJanela
(WPtr:WindowPtr)
(03)
(10)

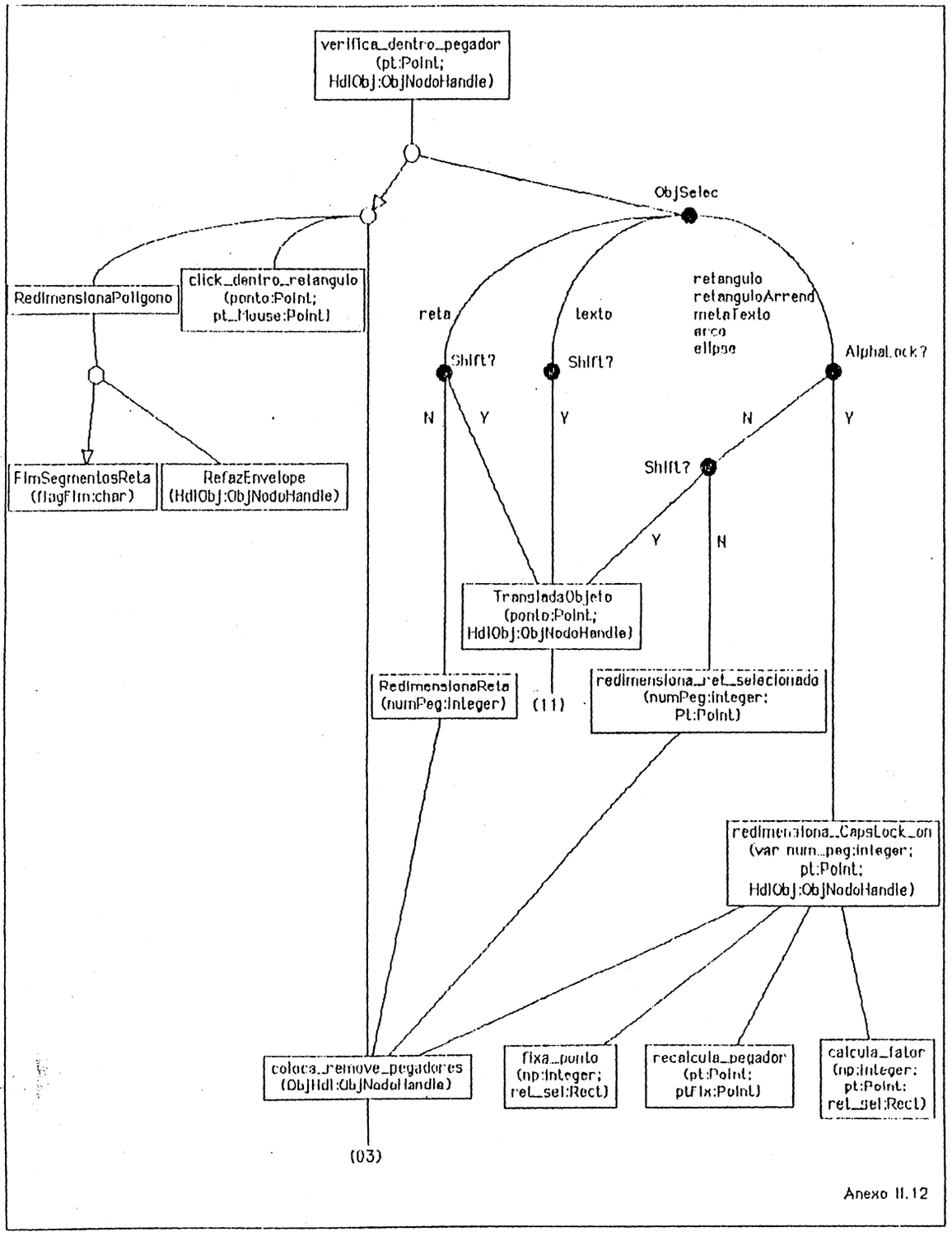
Anexo II.09


RealizaAcaoCorrente
(ponto:Point)
Update?
queObjeto
retangulo
retanguloArrend
elipse
reta
arco
metaTexto
texto
poligono
poligReg
seta
RegeneraJanela
(WPtr:windowPtr)
DrawObjetos
(qualObjeto:Objetos;
ponto:Point)
CriaTexto
(ponto:Point)
Control_linhas
(var Pt_inic:Point)
Control_linhas_regulares
(ponto:Point;
numLados:Integer)
(11)
(03)
DesenhandoObjetos
(startPt,endPt:Point;
valor1,valor2:Integer)
InsereObjetoCriado
(qualObjeto:Objetos;
startPt,endPt:Point;
valor1,valor2:Integer;
var ObjNodoHdl:ObjNodoHandle)
centroRect
(ret:Rect)
RefazEnvelope
(HdlObj:ObjNodoHandle)
coloca_vertice_lista
(var penultVertice,ultVertice:
VerticeHandle;
pt_mouse:Point)
InsereNovoObjeto
(var primObjeto,novoObjeto:
ObjNodoHandle)
coloca_remove_pegadores
(ObjHdl:ObjNodoHandle)
(03)

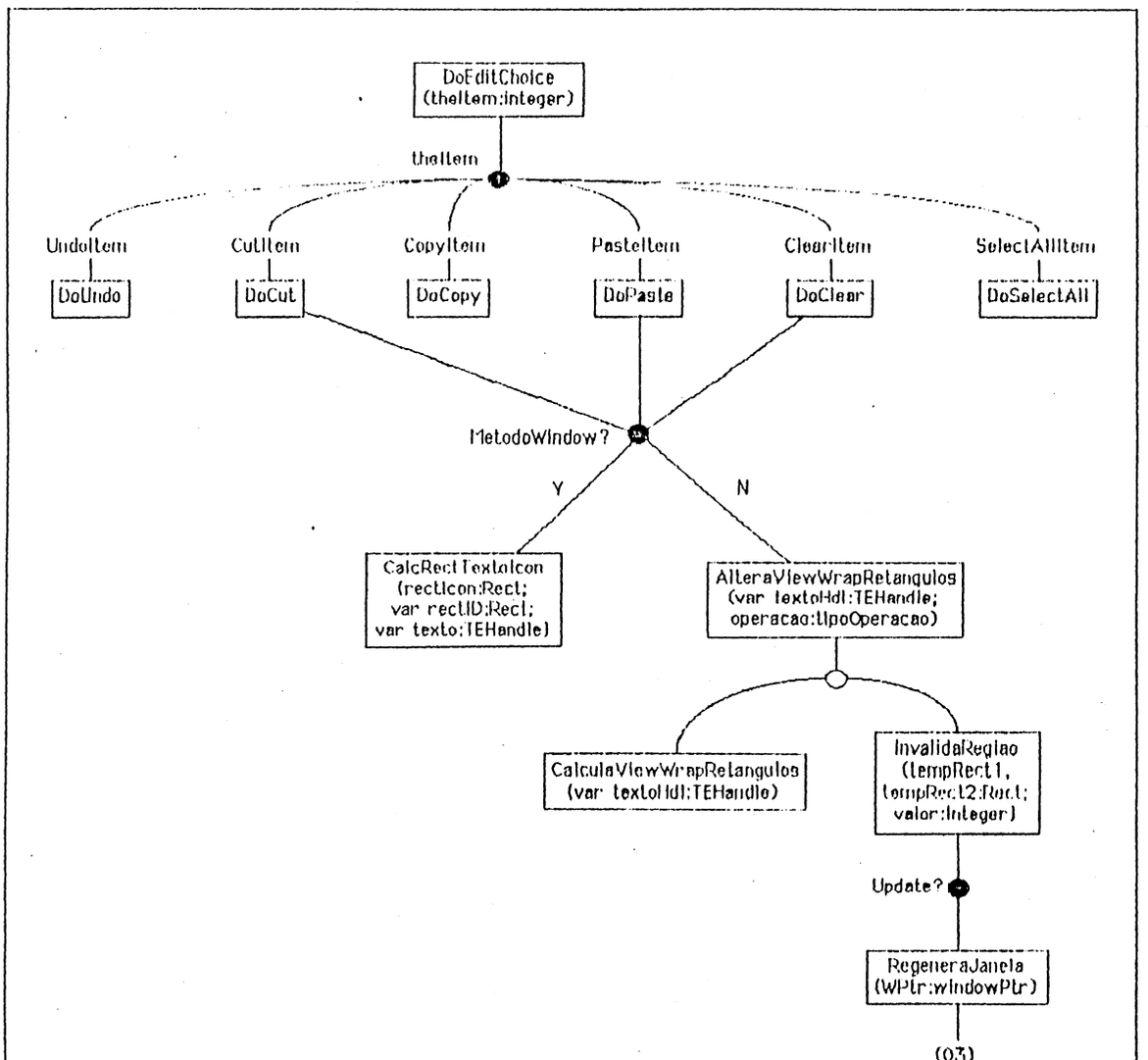
Anexo II.10


seta
primitivo
retangulo,retanguloArrend,
elipse,arco,texto,metaTexto,
poligono,poligReg
reta
PtInReta
(ponto,r1,r2:Point)
verifica_dentro_pegador
(pt:Point;
HdlObj:ObjNodoHandle)
TransladaObjeto
(ponto:Point;
HdlObj:ObjNodoHandle)
verifica_dentro_retangulo
(primeiro:ObjNodoHandle;
ponto:Point)
SetaSizeMenu
(12)
ObjSelec
poligono
poligReg
MapeiaPoligono
(srcRect,dstRect:Rect;
primVertice:VerticeHandle)
coloca_remove_pegadores
(ObjHdl:ObjNodoHandle)
ObjetoSelecionado?
(03)
reta?
PtInReta
(ponto,r1,r2:Point)
verifica_dentro_pegador
(pt:Point;
HdlObj:ObjNodoHandle)
TransladaObjeto
(ponto:Point;
HdlObj:ObjNodoHandle)
coloca_remove_pegadores
(ObjHdl:ObjNodoHandle)
(11)
(11)
(03)

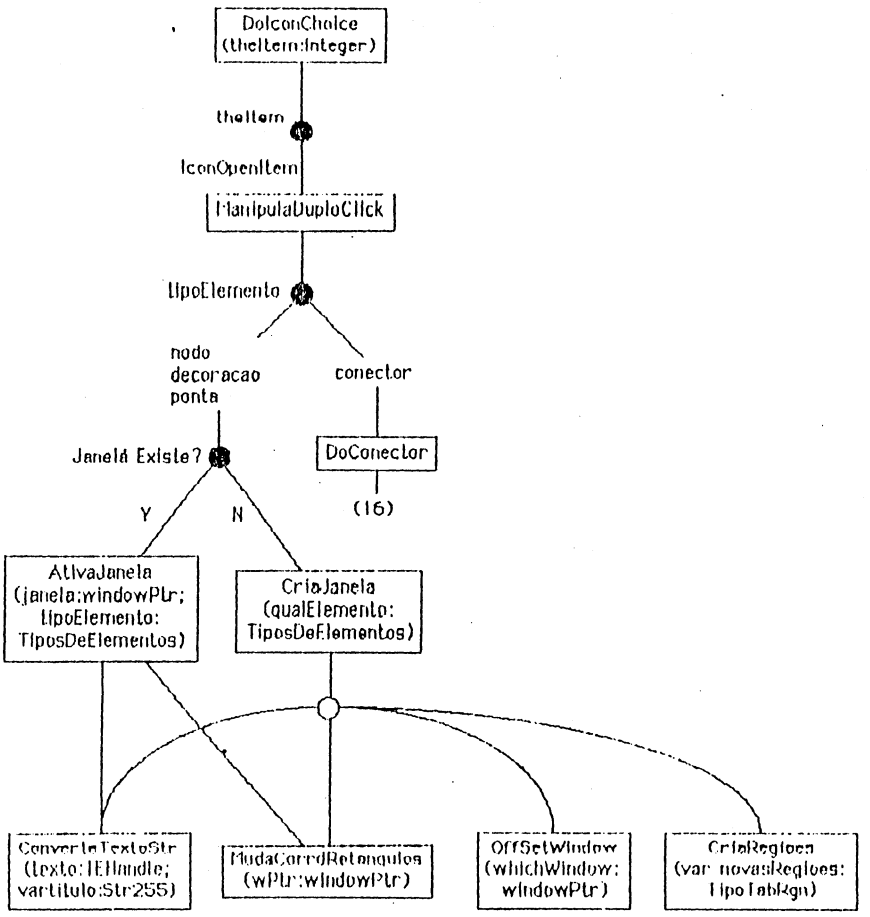
Anexo II.11


verifica_dentro_pegador
(pt:Point;
HdlObj:ObjNodoHandle)
ObjSelec
RedimensionaPoligono
click_dentro_retangulo
(ponto:Point;
pt_Mouse:Point)
reta
texto
retangulo
retanguloArrend
metaTexto
arco
elipse
Shift?
Shift?
AlphaLock?
N
Y
Y
N
Y
Shift?
Y
N
FimSegmentosReta
(flagFim:char)
RefazEnvelope
(HdlObj:ObjNodoHandle)
TransladaObjeto
(ponto:Point;
HdlObj:ObjNodoHandle)
RedimensionaReta
(numPeg:Integer)
(11)
redimensiona_ret_selecionado
(numPeg:Integer;
Pt:Point)
redimensiona_CapsLock_on
(var num_peg:Integer;
pt:Point;
HdlObj:ObjNodoHandle)
coloca_remove_pegadores
(ObjHdl:ObjNodoHandle)
fixa_ponto
(np:Integer;
rel_sel:Rect)
recalcula_pegador
(pt:Point;
ptFix:Point)
calcula_fator
(np:Integer;
pt:Point;
rel_sel:Rect)
(03)

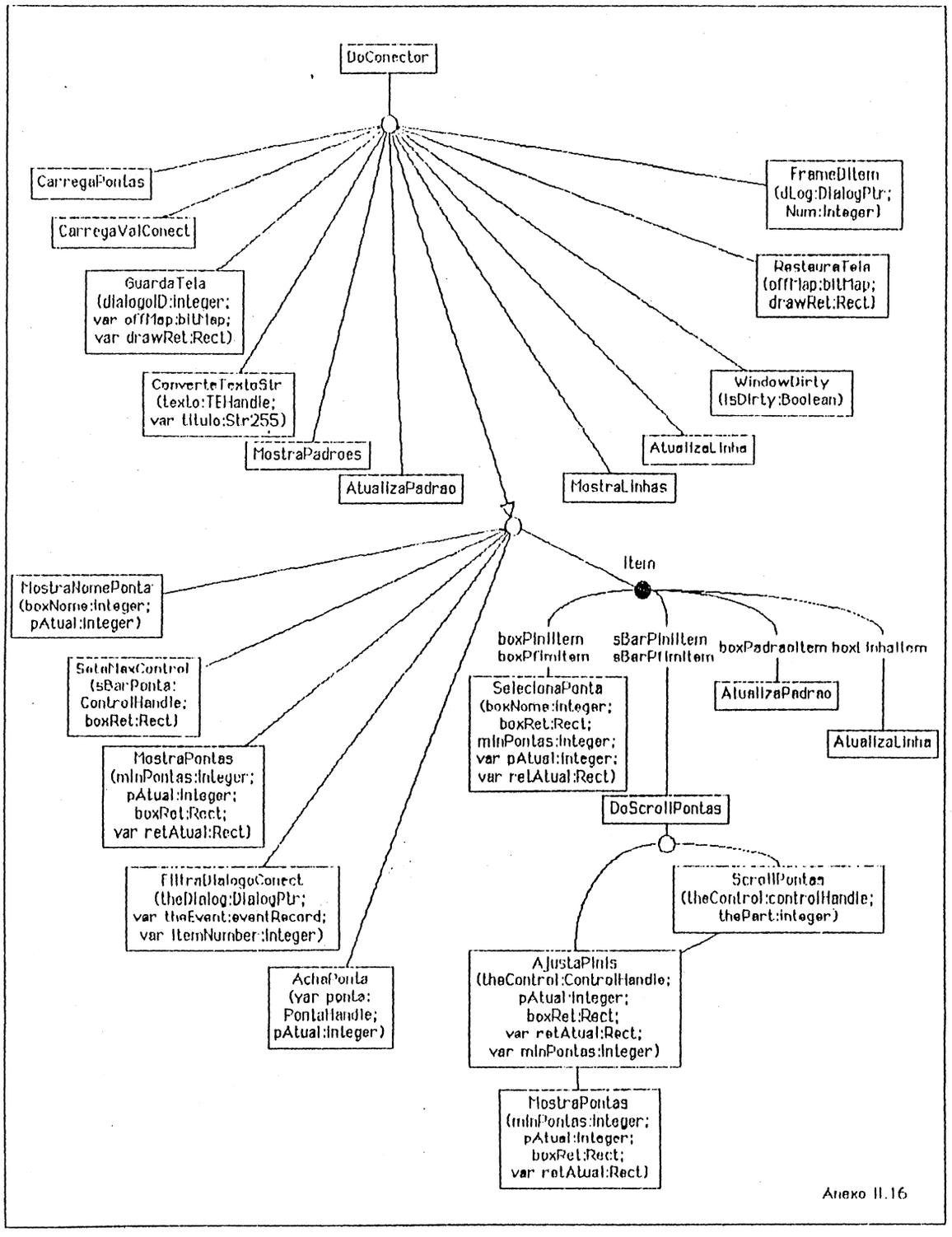
Anexo II.12


DoEditChoice
(theItem:Integer)
theItem
UndoItem
CutItem
CopyItem
PasteItem
ClearItem
SelectAllItem
DoUndo
DoCut
DoCopy
DoPaste
DoClear
DoSelectAll
MetodoWindow?
Y
N
CalcRectTextoIcon
(rectIcon:Rect;
var rectID:Rect;
var texto:TEHandle)
AlteraViewWrapRetangulos
(var textoHdl:TEHandle;
operacao:tipoOperacao)
CalculaViewWrapRetangulos
(var textoHdl:TEHandle)
InvalidaRegiao
(tempRect1,
tempRect2:Rect;
valor:Integer)
Update?
RegeneraJanela
(WPtr:WindowPtr)
(03)

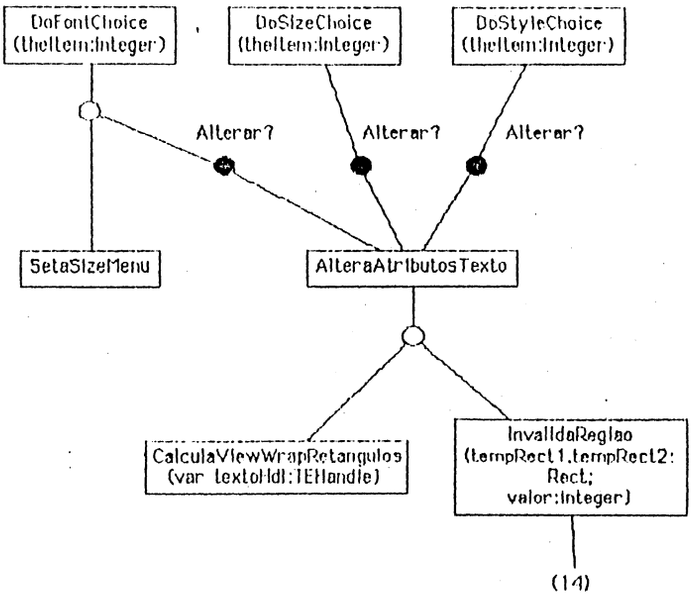
Anexo II.14


DoIconChoice
(theItem:Integer)
theItem
IconOpenItem
ManipulaDuploClick
TipoElemento
nodo
decoracao
ponta
conector
Janela Existe?
DoConector
(16)
Y
N
AtivaJanela
(janela:windowPtr;
tipoElemento:
TiposDeElementos)
CriaJanela
(qualElemento:
TiposDeElementos)
ConverteTextoStr
(texto:IEHandle;
varTitulo:Str255)
MudaCorrdRetangulos
(wPtr:wIndowPtr)
OffSetWindow
(whichWindow:
wIndowPtr)
CriaRegioes
(var novasRegioes:
TipoTabRgn)

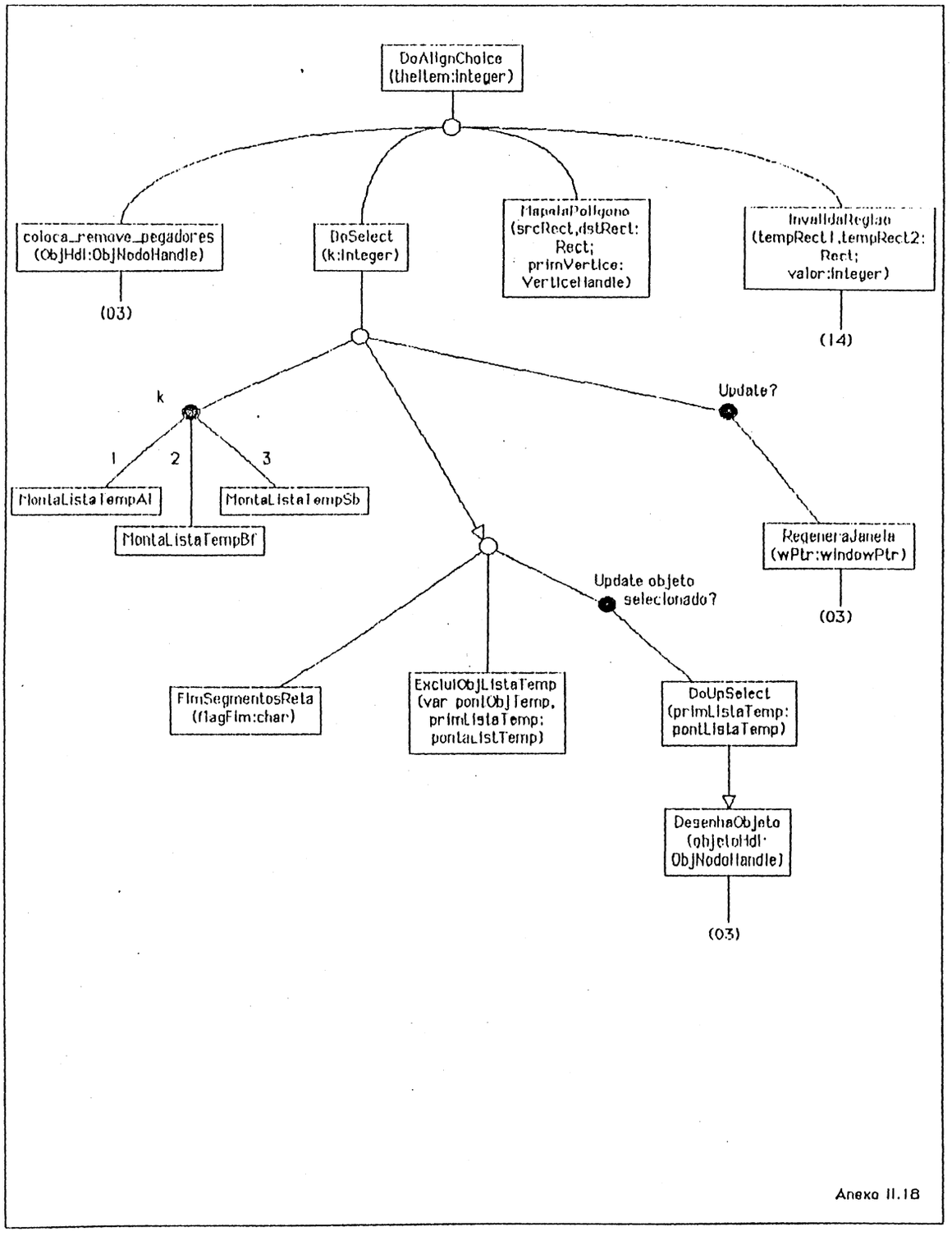
Anexo II.15


DoConector
CarregaPontas
CarregaValConect
GuardaTela
(dialogoID:Integer;
var offMap:bitMap;
var drawRet:Rect)
ConverteTextoStr
(texto:TEHandle;
var titulo:Str255)
MostraPadroes
AtualizaPadrao
FrameDItem
(dLog:DialogPtr;
Num:Integer)
RestauraTela
(offMap:bitMap;
drawRet:Rect)
WindowDirty
(IsDirty:Boolean)
AtualizaLinha
MostraLinhas
Item
MostraNomePonta
(boxNome:Integer;
pAtual:Integer)
SetaMaxControl
(sBarPonta:
ControlHandle;
boxRet:Rect)
MostraPontas
(minPontas:Integer;
pAtual:Integer;
boxRet:Rect;
var retAtual:Rect)
FiltraDialogoConect
(theDialog:DialogPtr;
var theEvent:eventRecord;
var ItemNumber:Integer)
AchaPonta
(var ponta:
PontaHandle;
pAtual:Integer)
boxPinItem
boxPfimItem
sBarPinItem
sBarPfimItem
boxPadraoItem
boxLinhaItem
SelecionaPonta
(boxNome:Integer;
boxRet:Rect;
minPontas:Integer;
var pAtual:Integer;
var retAtual:Rect)
AtualizaPadrao
AtualizaLinha
DoScrollPontas
ScrollPontas
(theControl:controlHandle;
thePart:integer)
AjustaPinIs
(theControl:ControlHandle;
pAtual:Integer;
boxRet:Rect;
var retAtual:Rect;
var minPontas:Integer)
MostraPontas
(minPontas:Integer;
pAtual:Integer;
boxRet:Rect;
var retAtual:Rect)

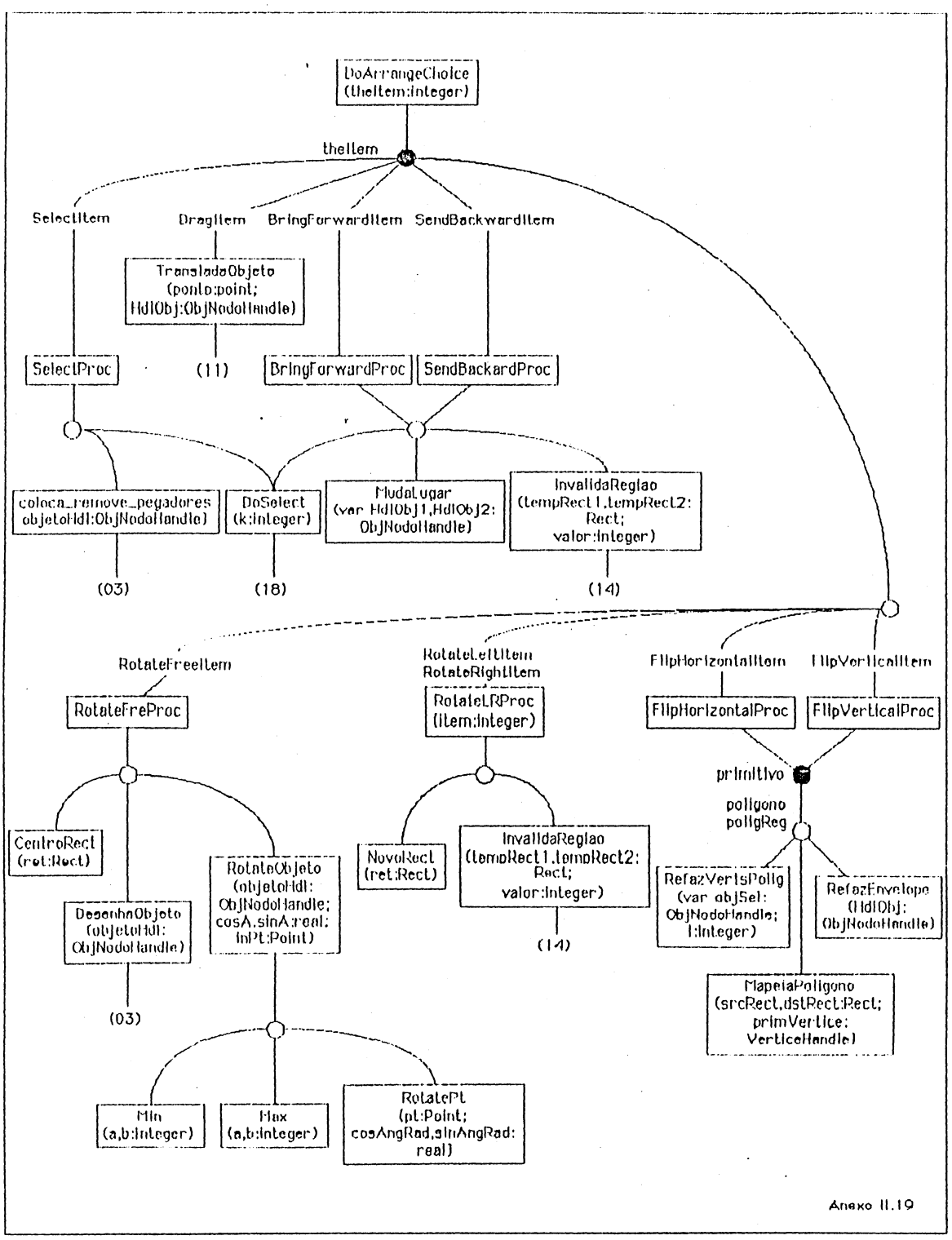
Anexo II.16


DoFontChoice
(theItem:Integer)
DoSizeChoice
(theItem:Integer)
DoStyleChoice
(theItem:Integer)
Alterar?
Alterar?
Alterar?
SetaSizeMenu
AlteraAtributosTexto
CalculaViewWrapRetangulos
(var textoHdl:TEHandle)
InvalidaRegiao
(tempRect1,tempRect2:
Rect;
valor:Integer)
(14)

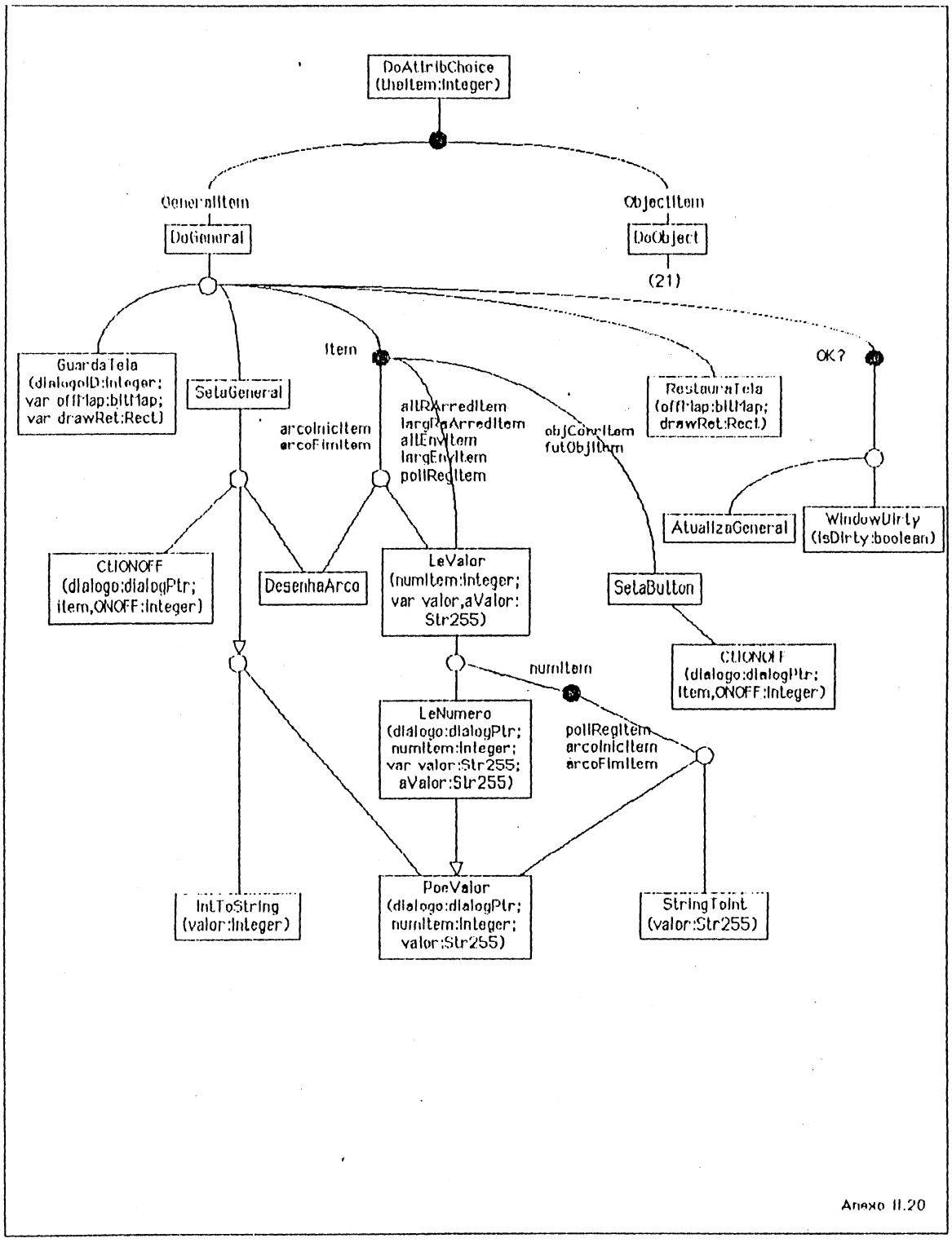
Anexo II.17


DoAlignChoice
(theItem:Integer)
coloca_remove_pegadores
(ObjHdl:ObjNodoHandle)
(03)
DoSelect
(k:Integer)
MapalaPoligono
(srcRect,dstRect:
Rect;
primVertice:
VerticeHandle)
InvalidaRegiao
(tempRect1,tempRect2:
Rect;
valor:Integer)
(14)
k
1
2
3
MontaListaTempA1
MontaListaTempB1
MontaListaTempSb
Update?
RegeneraJanela
(wPtr:windowPtr)
(03)
Update objeto
selecionado?
FimSegmentosReta
(flagFim:char)
ExcluiObjListaTemp
(var pontObjTemp,
primListaTemp;
pontaListTemp)
DoUpSelect
(primListaTemp;
pontListaTemp)
DesenhaObjeto
(objetoHdl:
ObjNodoHandle)
(03)

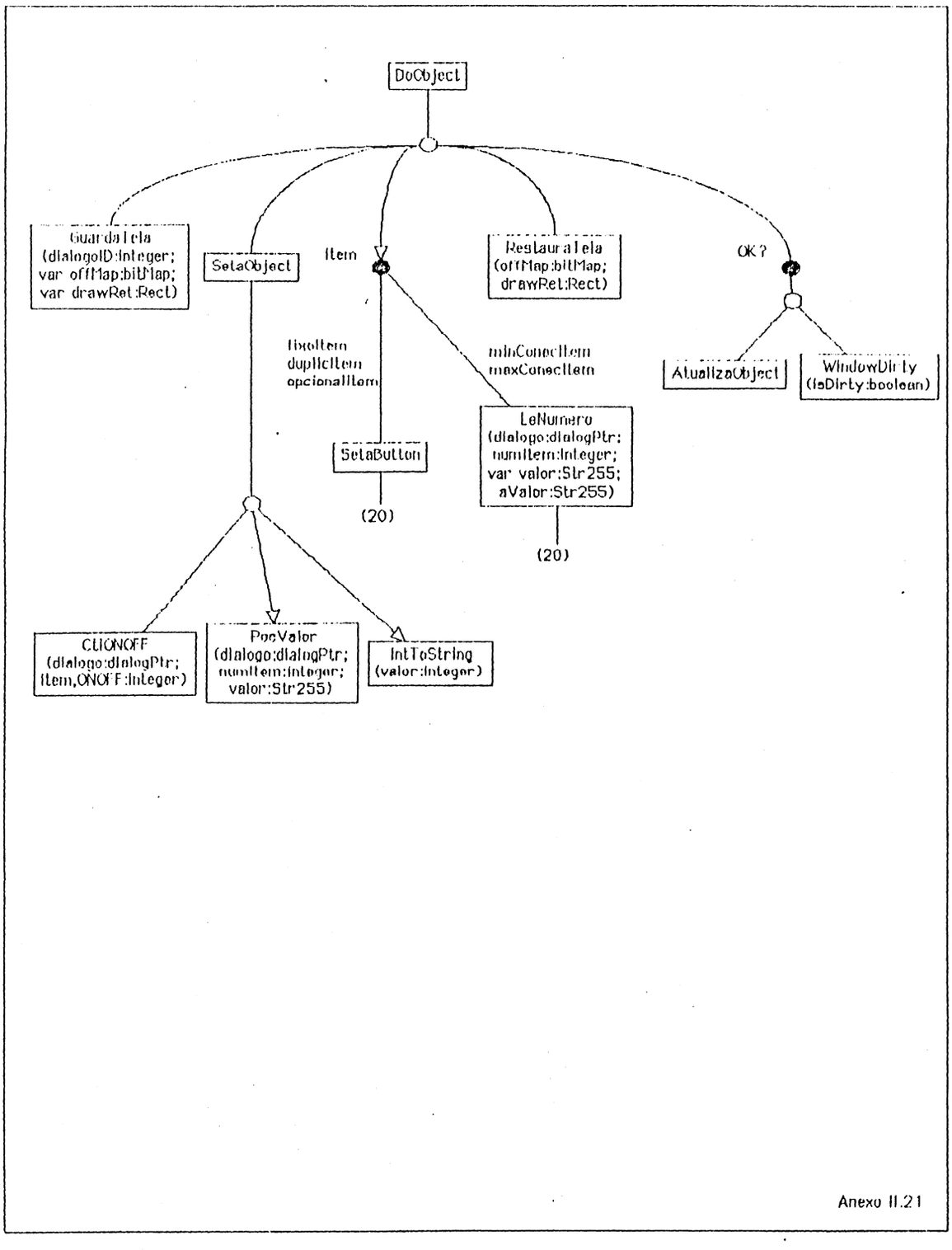
Anexo II.18


DoArrangeChoice
(theItem:Integer)
theItem
SelectItem
DragItem
BringForwardItem
SendBackwardItem
TransladaObjeto
(ponto:point;
HdlObj:ObjNodoHandle)
SelectProc
(11)
BringForwardProc
SendBackardProc
coloca_remove_pegadores
objetoHdl:ObjNodoHandle)
DoSelect
(k:Integer)
MudaLugar
(var HdlObj1,HdlObj2:
ObjNodoHandle)
InvalidaRegiao
(tempRect1,tempRect2:
Rect;
valor:Integer)
(03)
(18)
(14)
RotateFreeItem
RotateFreProc
RotateLeftItem
RotateRightItem
RotateLRProc
(iItem:Integer)
FlipHorizontalItem
FlipVerticalItem
FlipHorizontalProc
FlipVerticalProc
primitivo
poligono
poligReg
CentroRect
(ret:Rect)
DesenhaObjeto
(objetoHdl:
ObjNodoHandle)
RotateObjeto
(objetoHdl:
ObjNodoHandle;
cosA,sinA:real;
inPt:Point)
NovoRect
(ret:Rect)
InvalidaRegiao
(tempRect1,tempRect2:
Rect;
valor:Integer)
(14)
RefazVertsPolig
(var objSel:
ObjNodoHandle;
i:Integer)
RefazEnvelope
(HdlObj:
ObjNodoHandle)
MapeiaPoligono
(srcRect,dstRect:Rect;
primVertice:
VerticeHandle)
(03)
Min
(a,b:Integer)
Max
(a,b:Integer)
RotatePt
(pt:Point;
cosAngRad,sinAngRad:
real)

Anexo II.19


DoAttribChoice
(theItem:Integer)
GeneralItem
ObjectItem
DoGeneral
DoObject
(21)
GuardaTela
(dialogoID:Integer;
var offMap:bitMap;
var drawRet:Rect)
SetaGeneral
Item
OK?
RestauraTela
(offMap:bitMap;
drawRet:Rect)
arcoInicItem
arcoFimItem
allRArredItem
largRaArredItem
allEnvItem
largEnvItem
pollRegItem
objCorrItem
futObjItem
AtualizaGeneral
WindowDirty
(isDirty:boolean)
CtlONOFF
(dialogo:dialogPtr;
item,ONOFF:Integer)
DesenhaArco
LeValor
(numItem:Integer;
var valor,aValor:
Str255)
SetaButton
CtlONOFF
(dialogo:dialogPtr;
Item,ONOFF:Integer)
numItem
LeNumero
(dialogo:dialogPtr;
numItem:Integer;
var valor:Str255;
aValor:Str255)
pollRegItem
arcoInicItem
arcoFimItem
IntToString
(valor:Integer)
PoeValor
(dialogo:dialogPtr;
numItem:Integer;
valor:Str255)
StringToInt
(valor:Str255)

Anexo II.20


DoObject
GuardaTela
(dialogoID:Integer;
var offMap:bitMap;
var drawRet:Rect)
SetaObject
Item
RestauraTela
(offMap:bitMap;
drawRet:Rect)
OK?
tipoItem
duplicItem
opcionalItem
minConecItem
maxConecItem
AtualizaObject
WindowDirty
(isDirty:boolean)
LeNumero
(dialogo:dialogPtr;
numItem:Integer;
var valor:Str255;
aValor:Str255)
SetaButton
(20)
(20)
CLIONOFF
(dialogo:dialogPtr;
item,ONOFF:Integer)
PosValor
(dialogo:dialogPtr;
numItem:Integer;
valor:Str255)
IntToString
(valor:Integer)

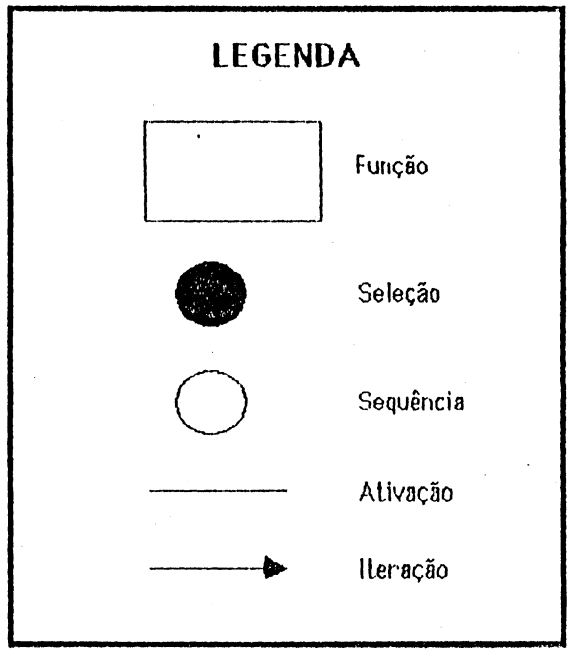
Anexo II.21

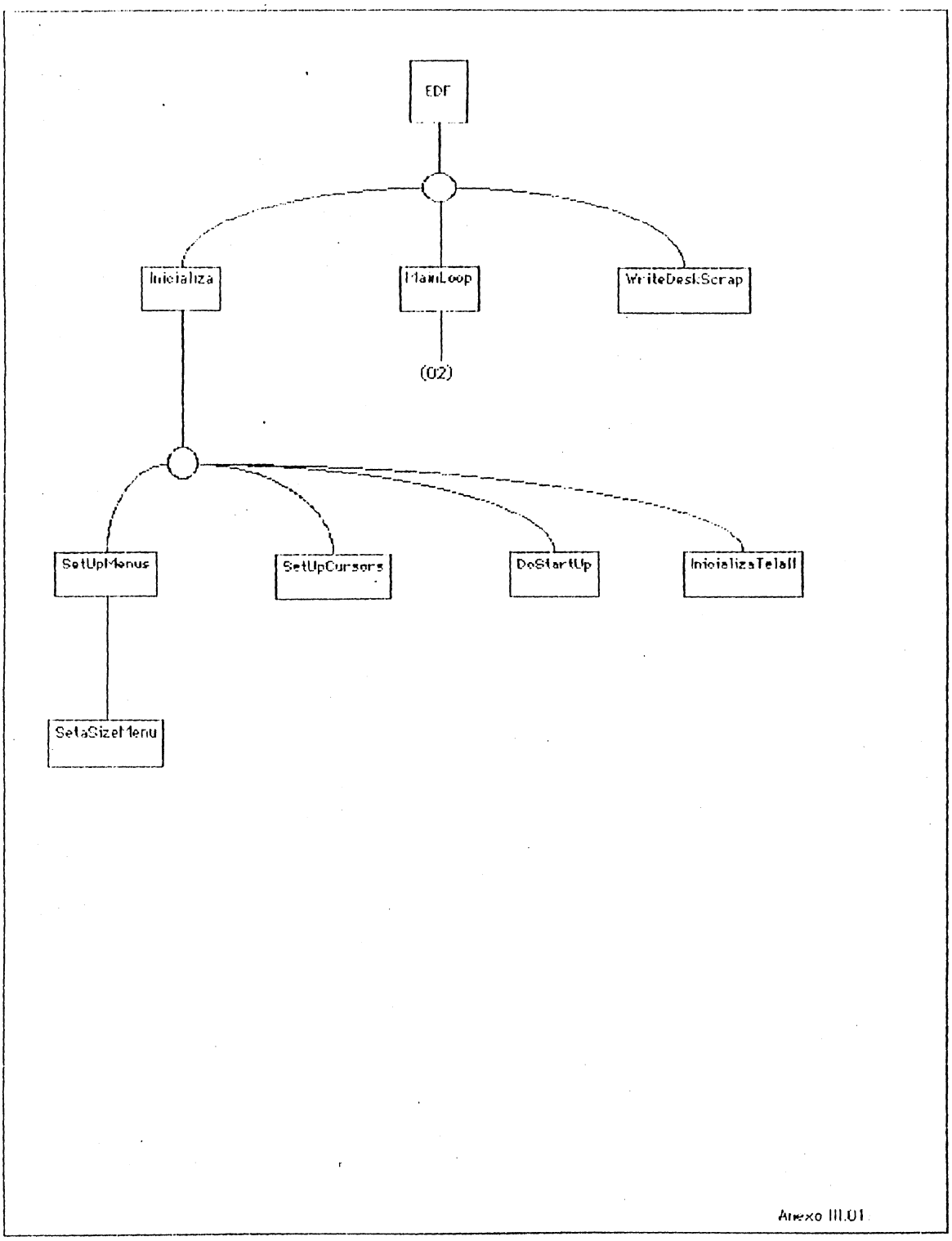
ANEXO 3

DIAGRAMAS DO PROJETO DO EDE

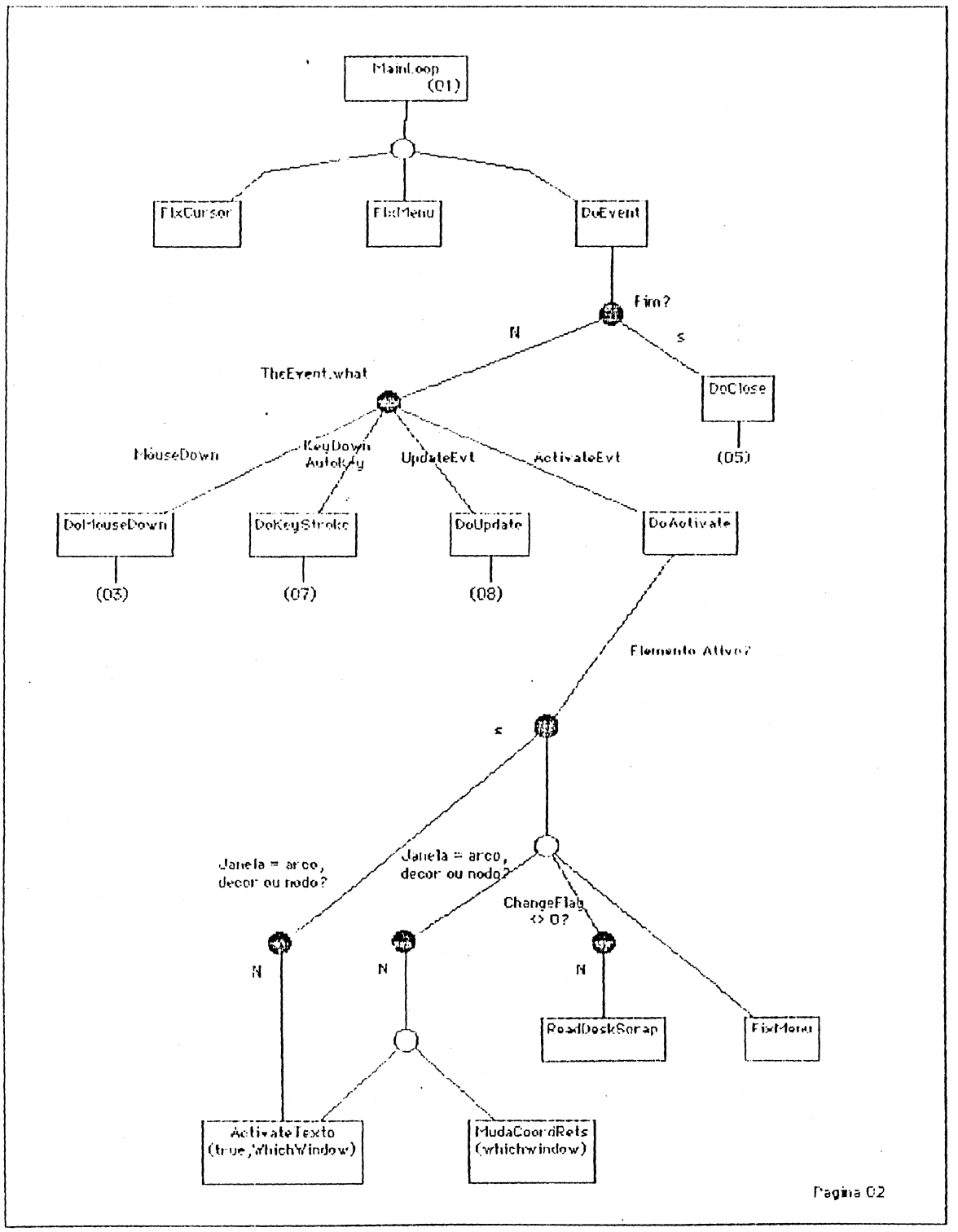
Este anexo mostra os diagramas de decomposição funcional do **EDE**.

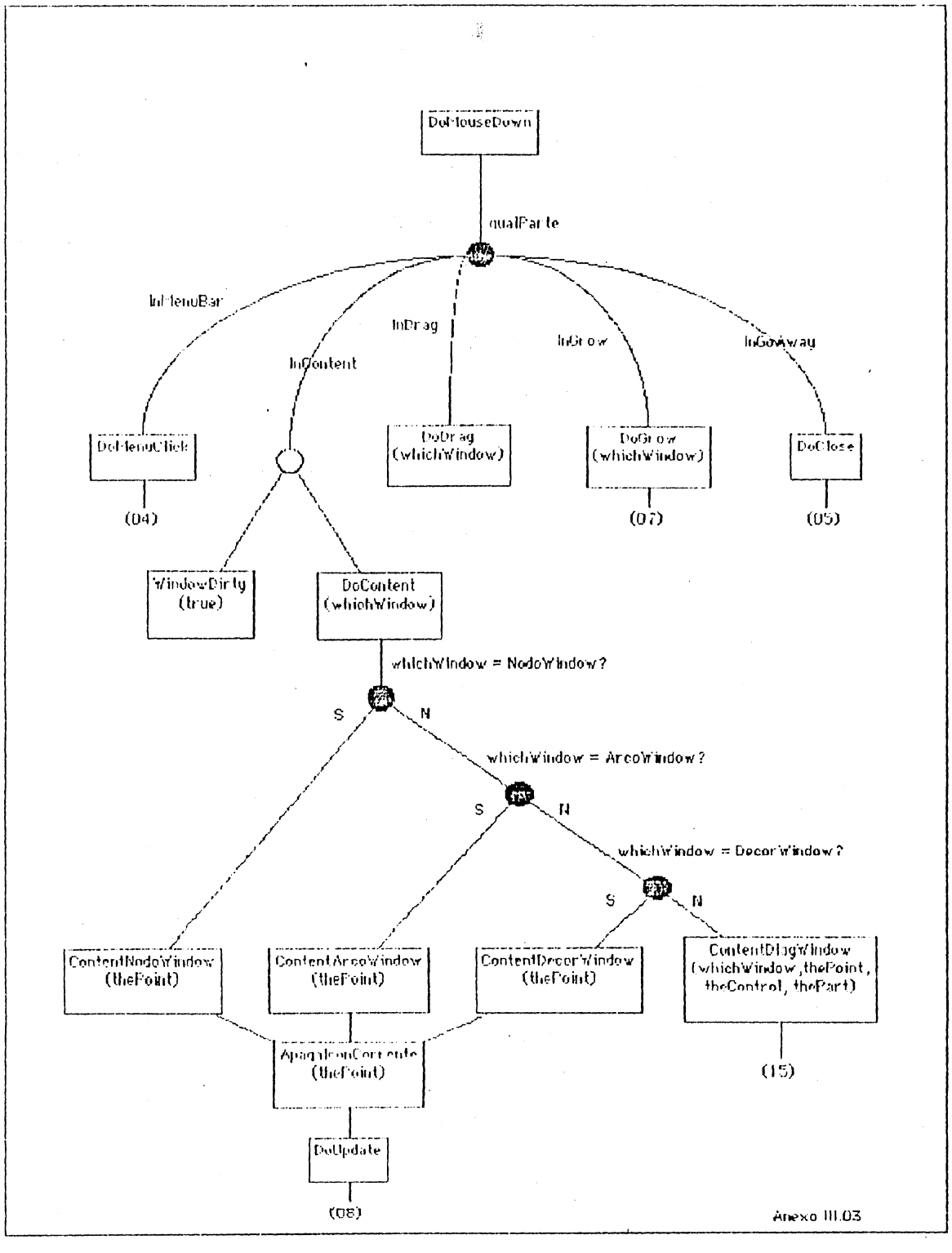
Os diagramas utilizam-se da seguinte notação:

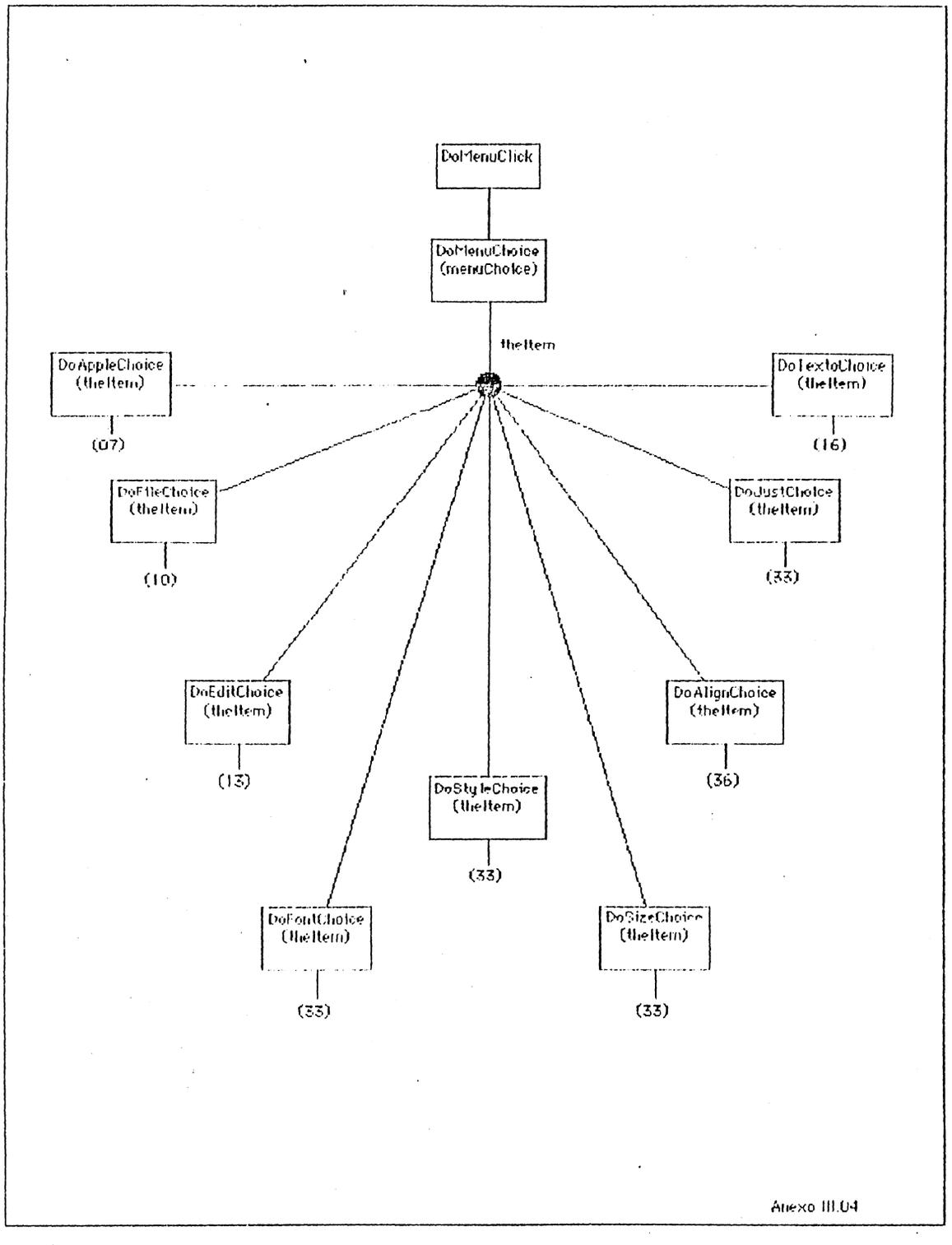



EDF
Inicializa
MainLoop
WriteDeskScrap
(02)
SetUpMenus
SetUpCursors
DoStartUp
InicializaTelaII
SetaSizeMenu

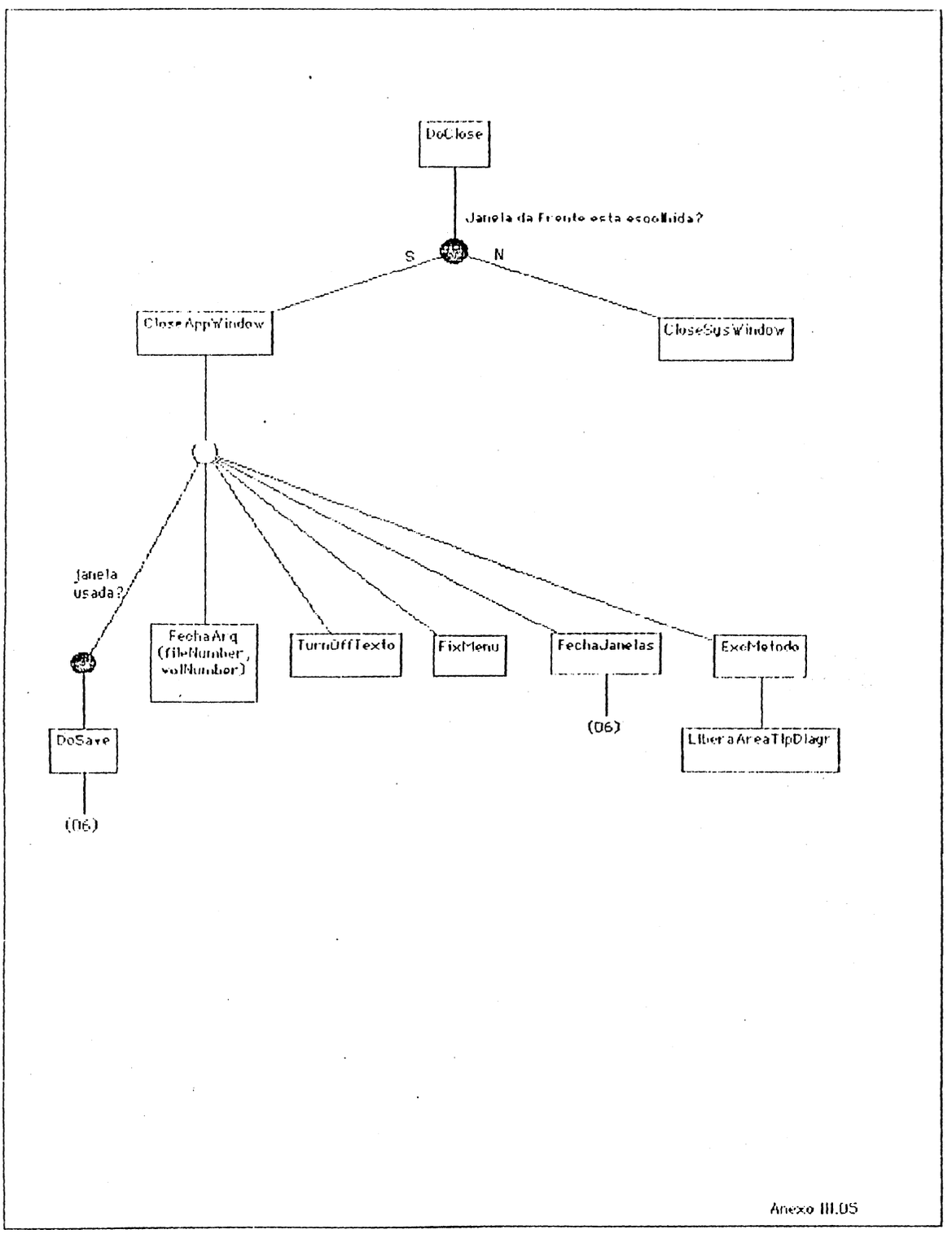
Anexo III.01


MainLoop
(01)
FixCursor
FixMenu
DoEvent
Fim?
N
S
TheEvent.what
DoClose
(05)
MouseDown
KeyDown
AutoKey
UpdateEvt
ActivateEvt
DoMouseDown
DoKeyStroke
DoUpdate
DoActivate
(03)
(07)
(08)
Elemento Ativo?
S
Janela = arco,
decor ou nodo?
Janela = arco,
decor ou nodo?
ChangeFlag
<> 0?
N
N
N
ReadDeskScrap
FixMenu
ActivateTexto
(true,WhichWindow)
MudaCoordRels
(whichwindow)
Pagina 02

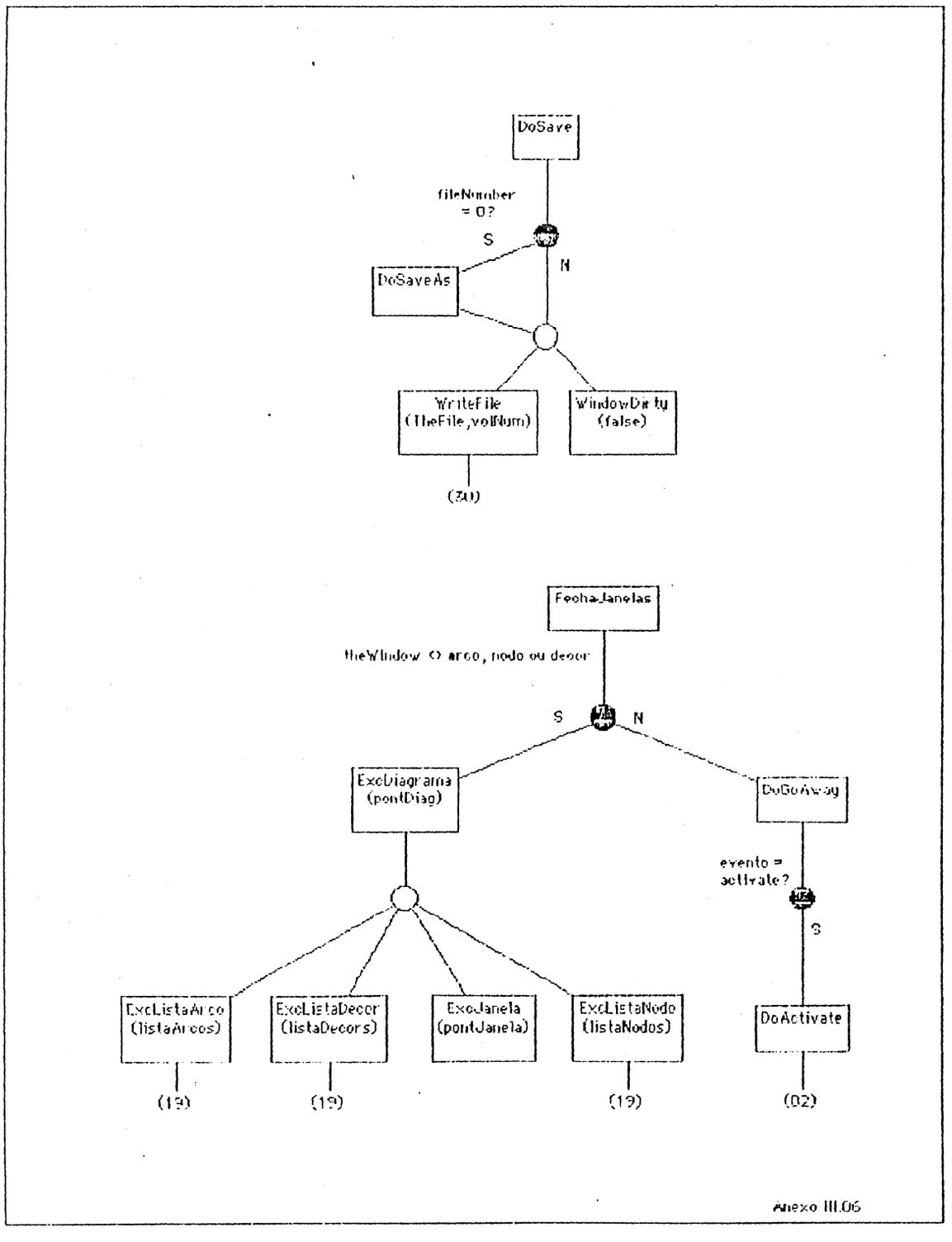

DoMouseDown
qualPart
InMenuBar
InContent
InDrag
InGrow
InGoAway
DoMenuClick
(04)
DoDrag
(whichWindow)
DoGrow
(whichWindow)
(07)
DoClose
(05)
WindowDirty
(true)
DoContent
(whichWindow)
whichWindow = NodoWindow?
S
N
whichWindow = AreaWindow?
S
N
whichWindow = DecorWindow?
S
N
ContentNodoWindow
(thePoint)
ContentAreaWindow
(thePoint)
ContentDecorWindow
(thePoint)
ContentDragWindow
(whichWindow, thePoint,
theControl, thePart)
(15)
DoUpdate
(08)
Anexo III.03

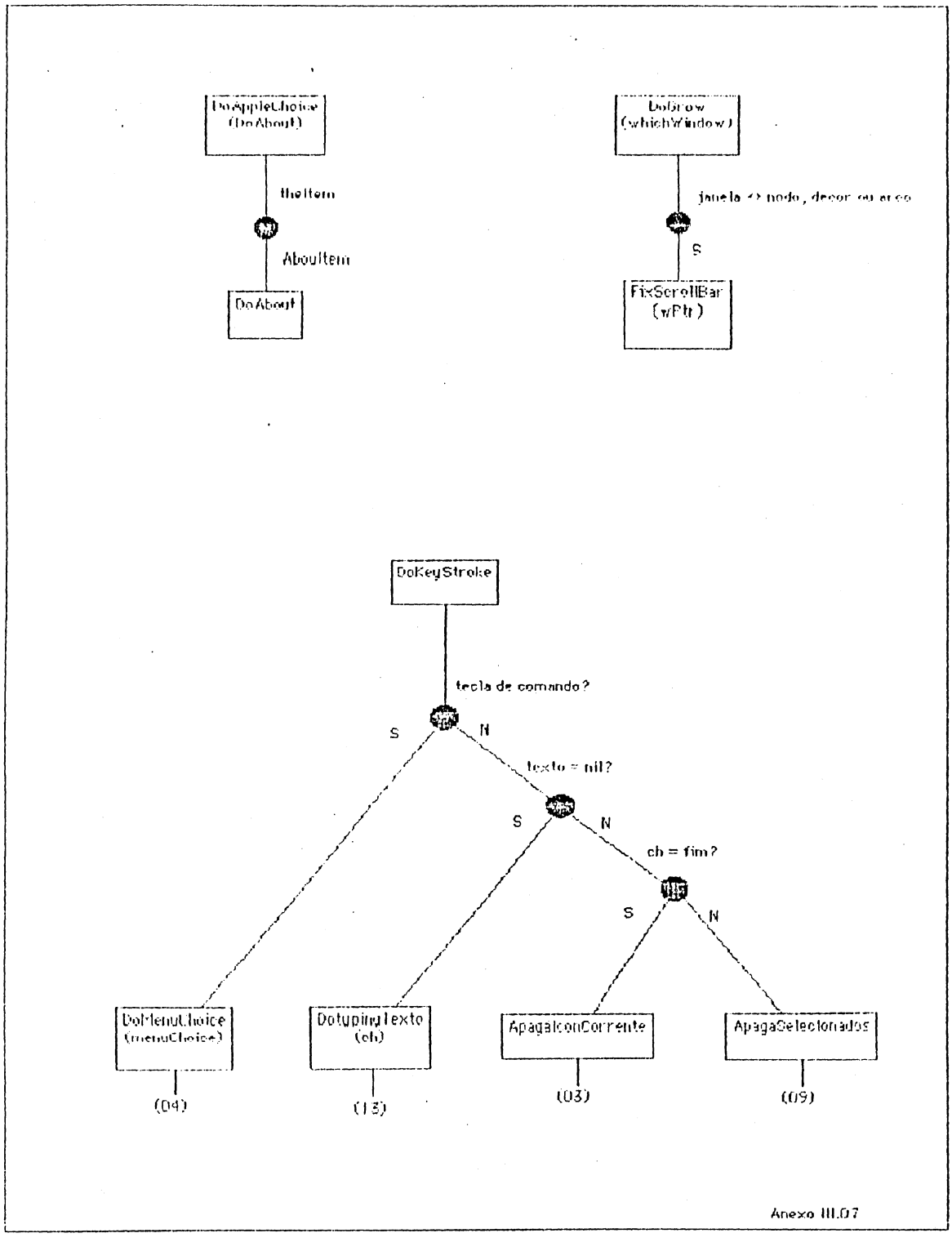

DoMenuClick
DoMenuChoice (menuChoice)
theItem
DoAppleChoice (theItem)
(07)
DoTextoChoice (theItem)
(16)
DoFileChoice (theItem)
(10)
DoJustChoice (theItem)
(33)
DoEditChoice (theItem)
(13)
DoAlignChoice (theItem)
(36)
DoStyleChoice (theItem)
(33)
DoFontChoice (theItem)
(33)
DoSizeChoice (theItem)
(33)

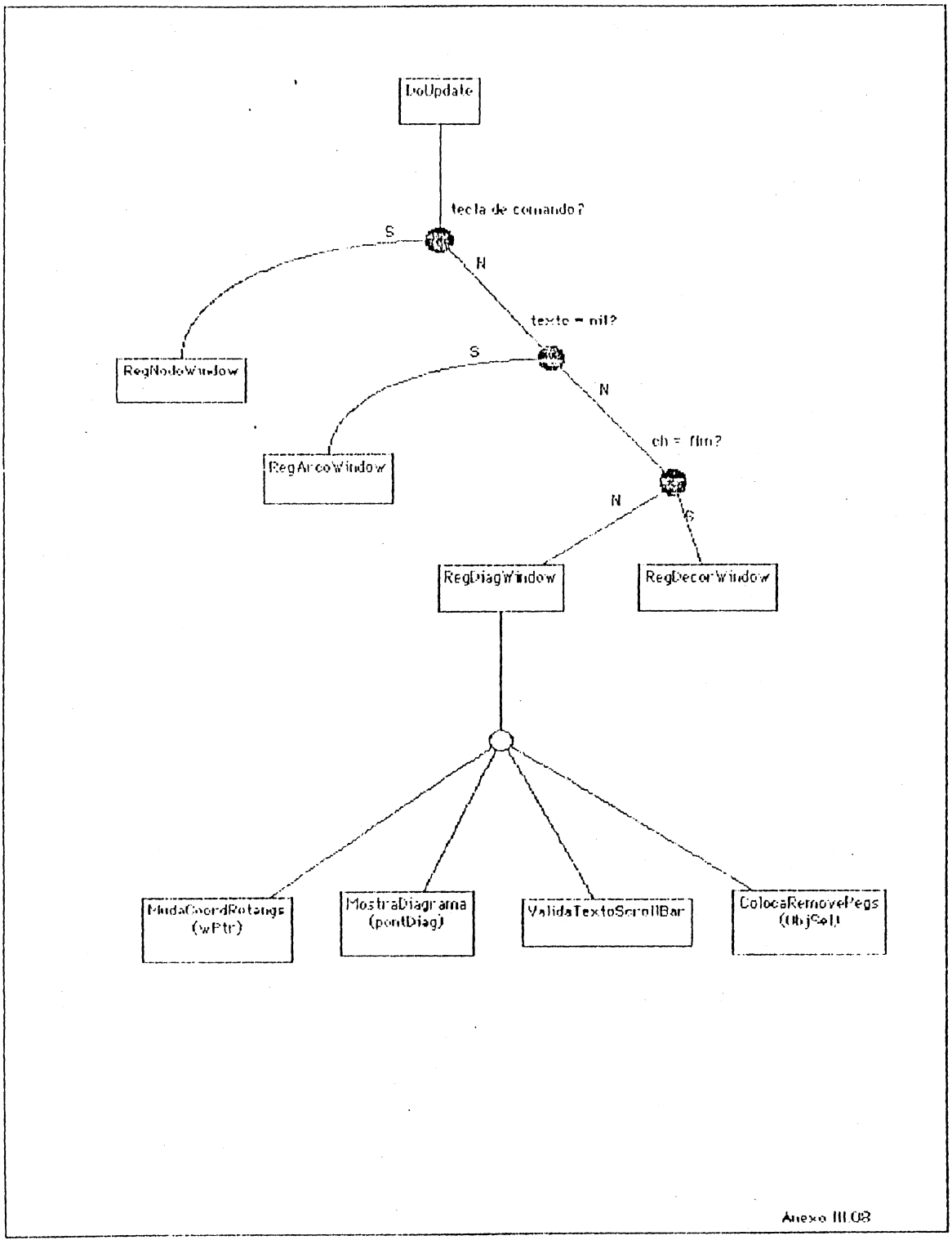
Anexo III.04


DoClose
Janela da Frente esta escolhida?
S
N
CloseAppWindow
CloseSysWindow
Janela usada?
FechaArq (fileNumber, volNumber)
TurnOffTexto
FixMenu
FechaJanelas
ExeMetodo
DoSave
(06)
(06)
LiberaAreaTipDiagr

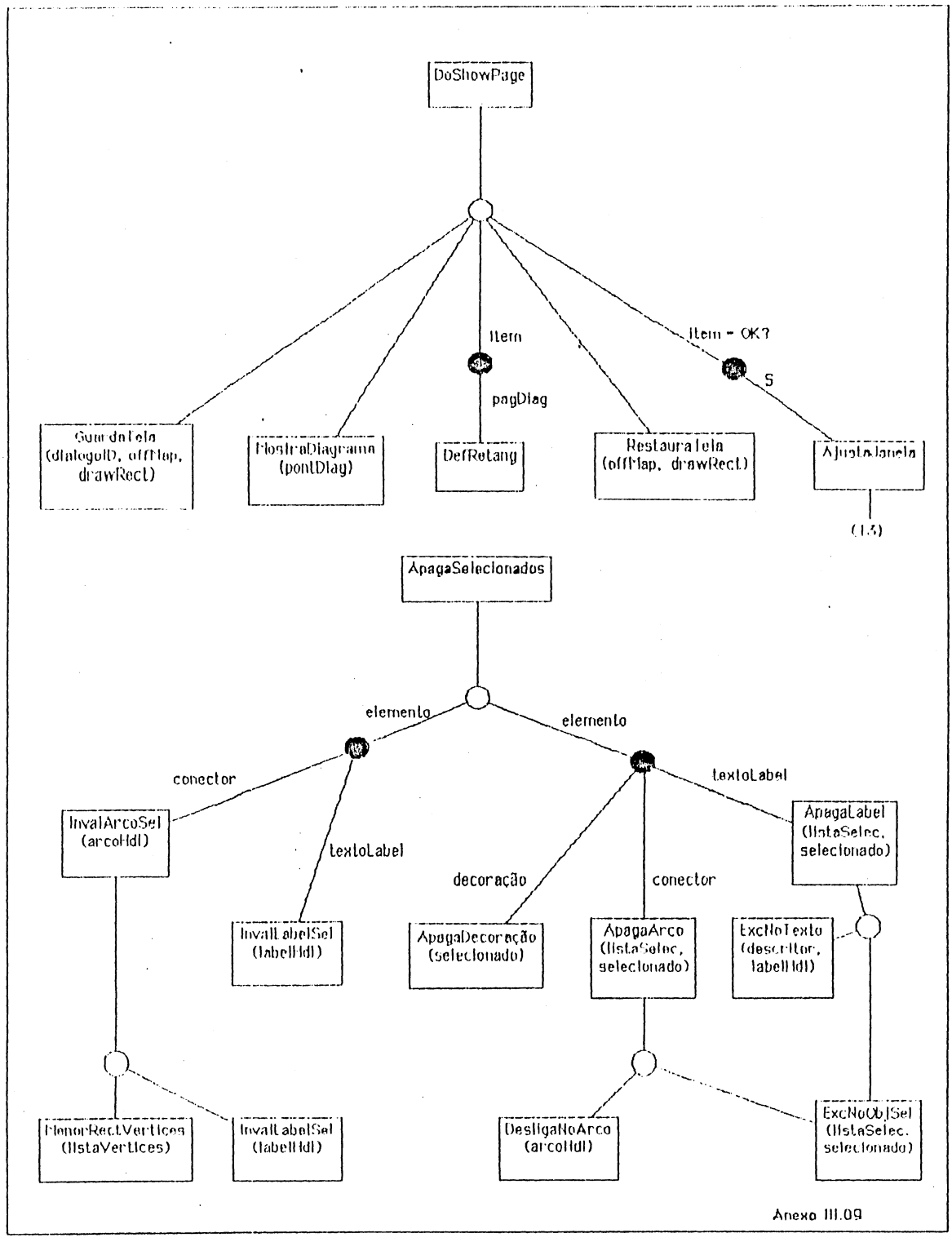
Anexo III.05


DoSave
fileNumber
= 0?
S
N
DoSaveAs
WriteFile
(TheFile,volNum)
WindowDirty
(false)
(20)
FechaJanelas
theWindow ◇ arco, nodo ou decor
S
N
ExcDiagrama
(pontDiag)
DoGoAway
evento =
activate?
S
ExcListaArco
(listaArcos)
ExcListaDecor
(listaDecors)
ExcJanela
(pontJanela)
ExcListaNodo
(listaNodos)
DoActivate
(19)
(19)
(19)
(02)
Anexo III.06

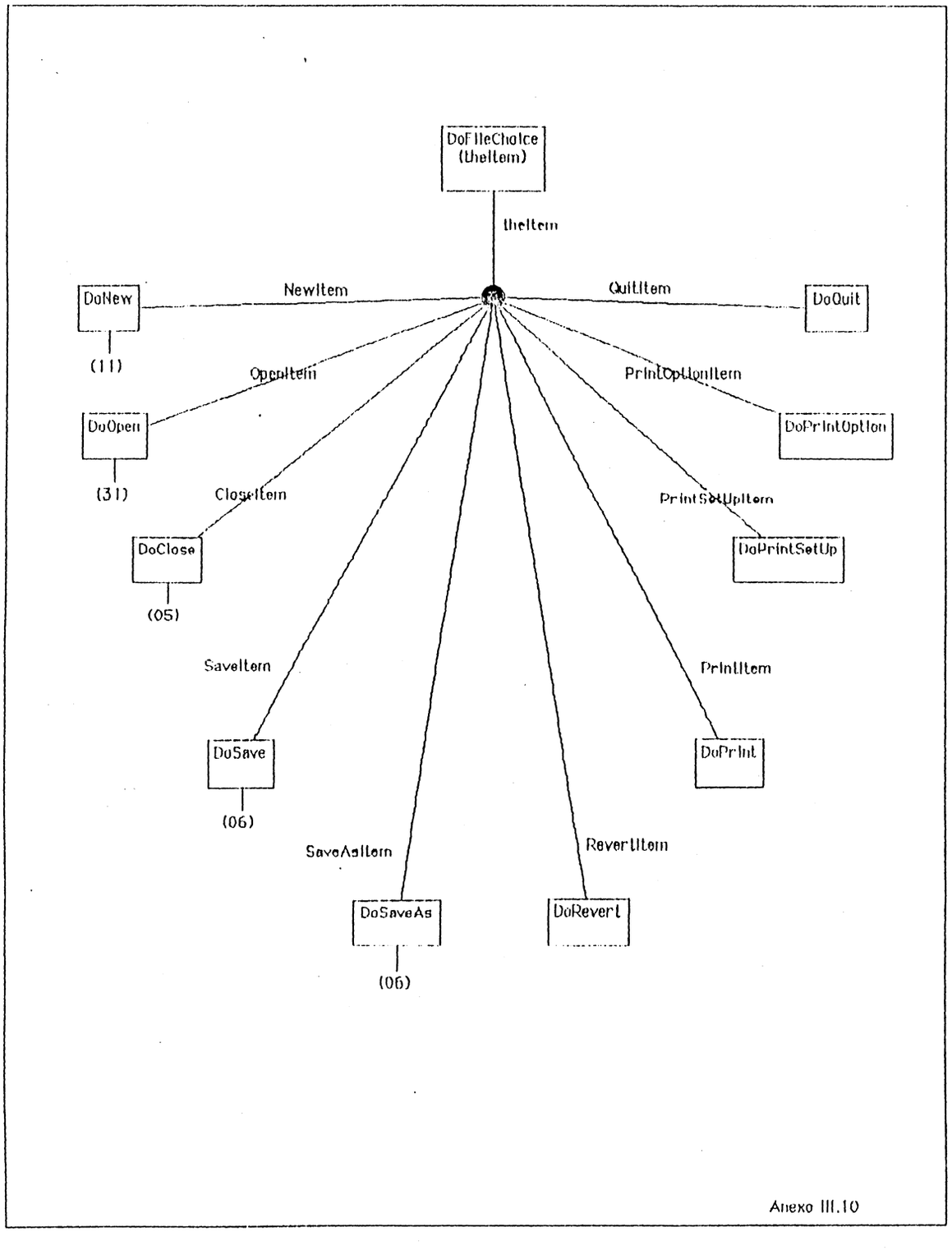

DoAppleChoice
(DoAbout)
theItem
AboutItem
DoAbout
DoGrow
(whichWindow)
S
FixScrollBar
(wPtr)
DoKeyStroke
tecla de comando?
S
N
texto = nil?
S
N
ch = fim?
S
N
DoMenuChoice
(menuChoice)
DoTypingTexto
(ch)
ApagaIconCorrente
ApagaSelecionados
(04)
(13)
(03)
(09)
Anexo III.07


DoUpdate
tecla de comando?
S
N
texto = nil?
S
N
RegNodoWindow
RegArcoWindow
ch = fim?
N
S
RegDiagWindow
RegDecorWindow
MudaCoordRotangs
(wPtr)
MostraDiagrama
(pontDiag)
ValidaTextoScrollBar
ColocaRemovePegs
(ObjSel)

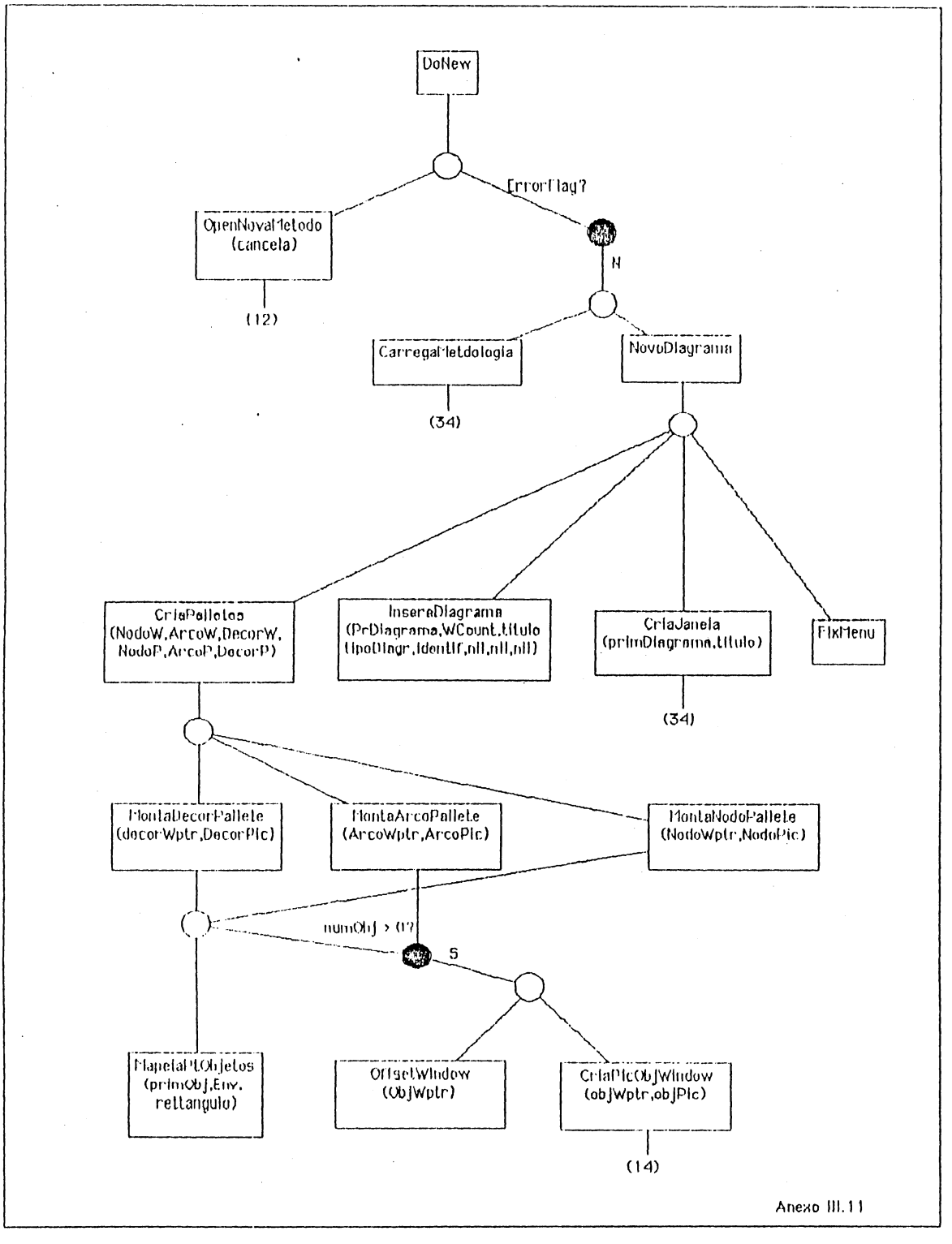
Anexo III.03


DoShowPage
iItem
iItem = OK?
S
pagDiag
GuardaTela
(dialogID, offMap, drawRect)
MostraDiagrama
(pontDiag)
DefRetang
RestauraTela
(offMap, drawRect)
AjustaJanela
(1.3)
ApagaSelecionados
elemento
elemento
conector
textoLabel
decoração
conector
textoLabel
InvalArcoSel
(arcoHdl)
InvalLabelSel
(labelHdl)
ApagaDecoração
(selecionado)
ApagaArco
(listaSelec, selecionado)
ExcNoTexto
(descritor, labelHdl)
ApagaLabel
(listaSelec, selecionado)
MenorRectVertices
(listaVertices)
InvalLabelSel
(labelHdl)
DesligaNoArco
(arcoHdl)
ExcNoObjSel
(listaSelec, selecionado)

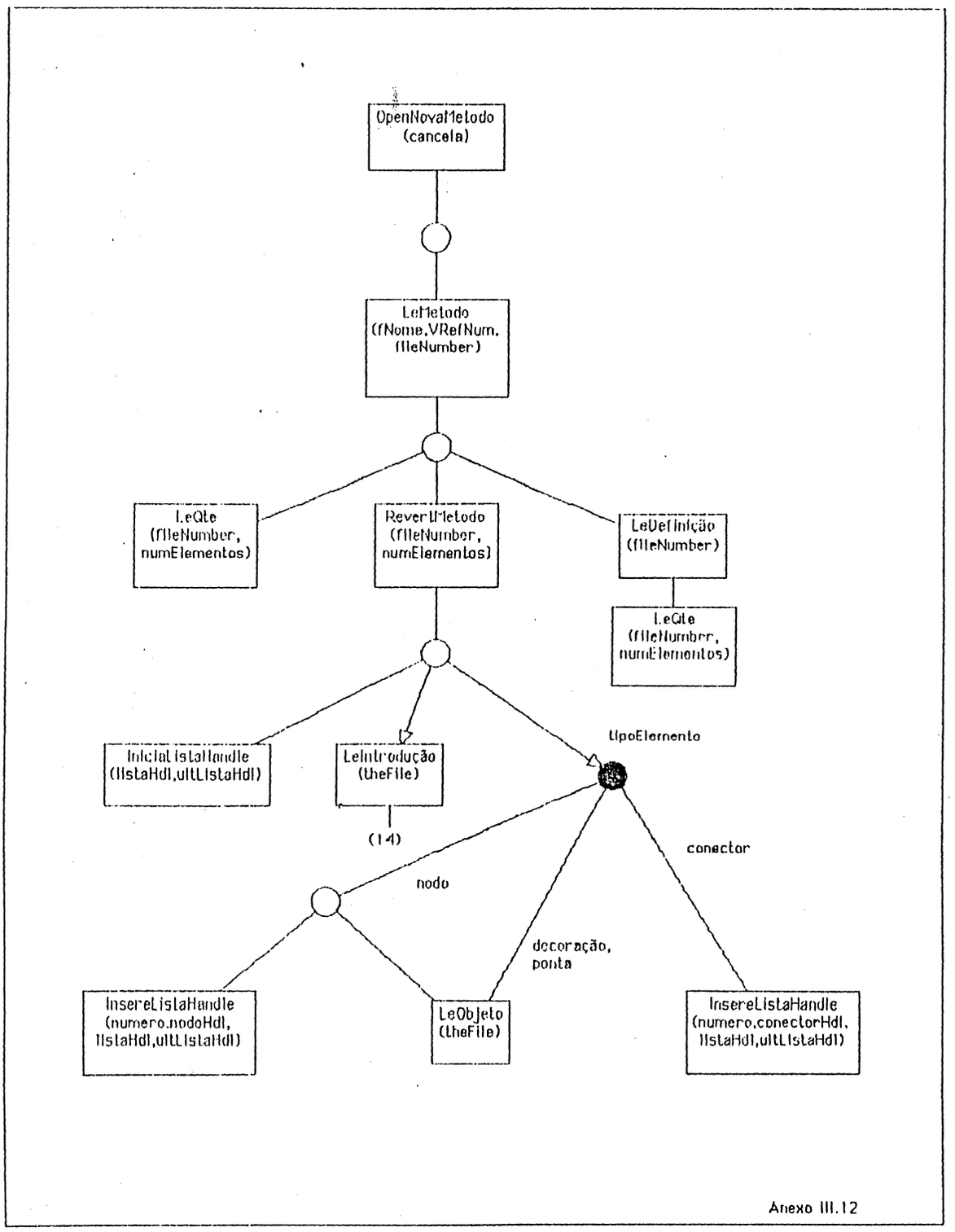
Anexo III.09


DoFileChoice
(theItem)
theItem
NewItem
QuitItem
DoNew
DoQuit
(11)
OpenItem
PrintOptionItem
DoOpen
DoPrintOption
(31)
CloseItem
PrintSetUpItem
DoClose
DoPrintSetUp
(05)
SaveItem
PrintItem
DoSave
DoPrint
(06)
SaveAsItem
RevertItem
DoSaveAs
DoRevert
(06)

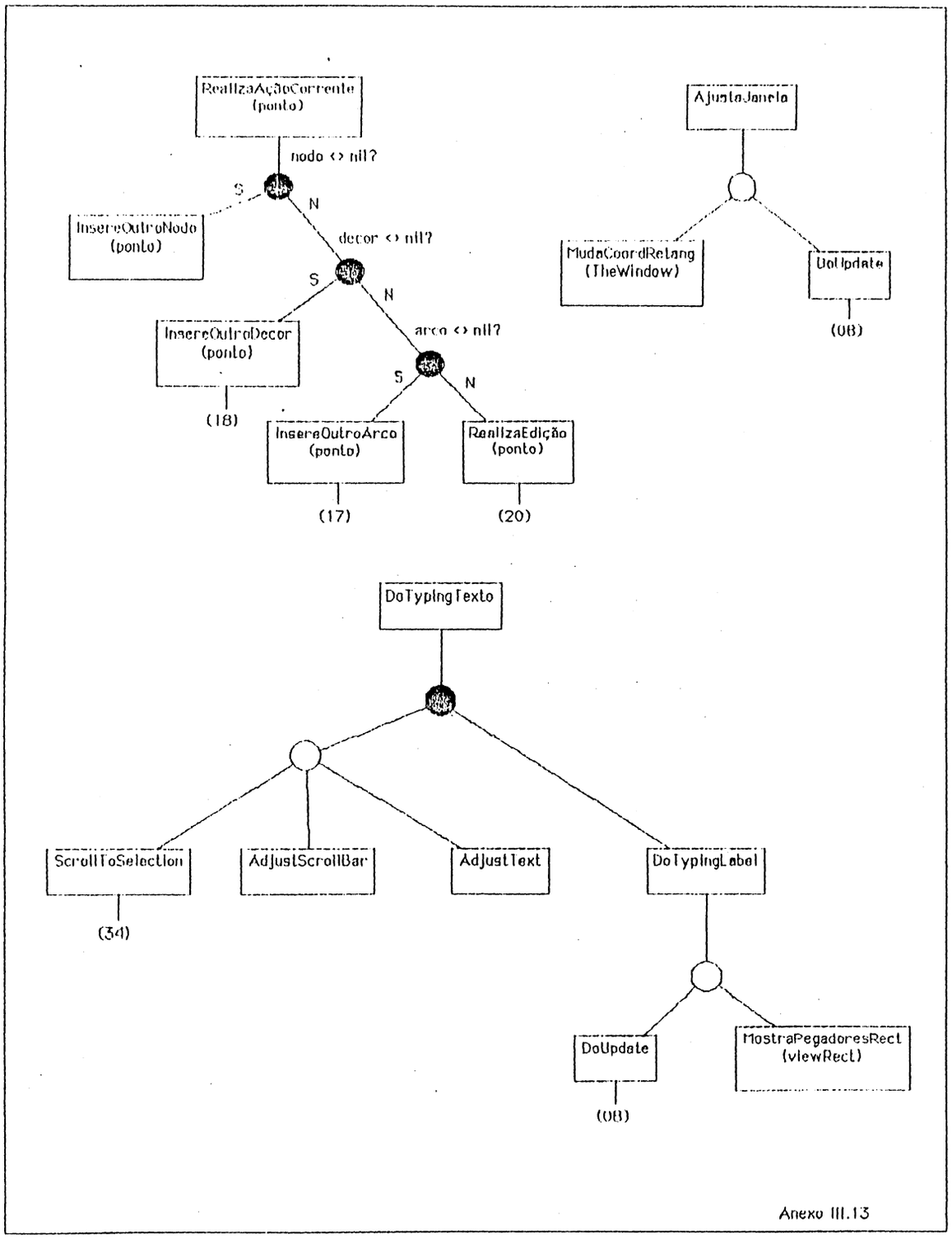
Anexo III.10


DoNew
ErrorFlag?
OpenNovaMetodo
(cancela)
N
(12)
CarregaMetdologia
NovoDiagrama
(34)
CriaPalletes
(NodoW,ArcoW,DecorW,
NodoP,ArcoP,DecorP)
InsereDiagrama
(PrDiagrama,WCount,titulo
tipoDiagr,IdentIf,nil,nil,nil)
CriaJanela
(primDiagrama,titulo)
FixMenu
(34)
MontaDecorPallete
(decorWptr,DecorPic)
MontaArcoPallete
(ArcoWptr,ArcoPic)
MontaNodoPallete
(NodoWptr,NodoPic)
numObj > 0?
S
MapeiaPtObjetos
(primObj,Env.
rettangulo)
OffsetWindow
(ObjWptr)
CriaPicObjWindow
(objWptr,objPic)
(14)

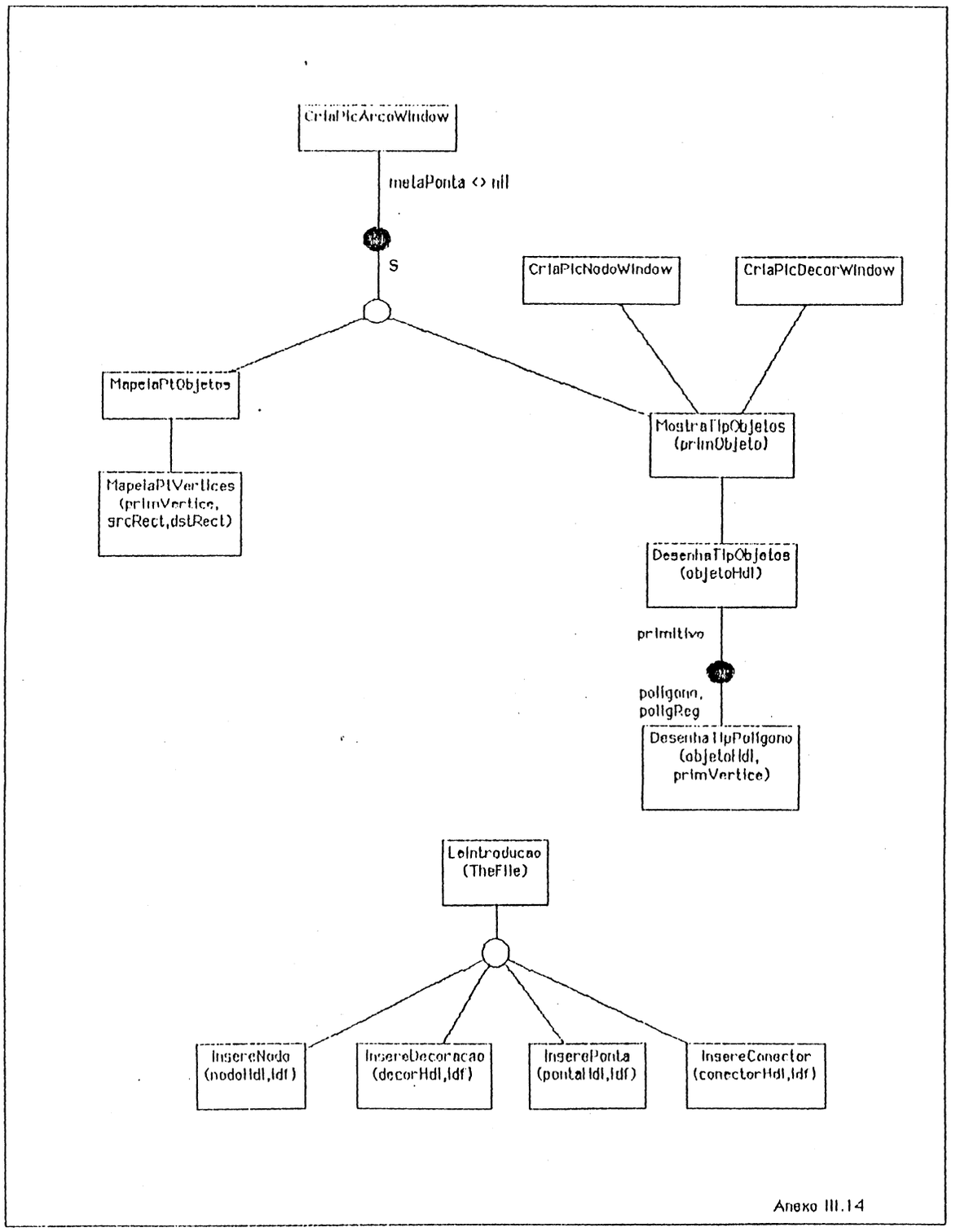
Anexo III.11


OpenNovaMetodo
(cancela)
LeMetodo
(fNome,VRefNum,
fileNumber)
LeQte
(fileNumber,
numElementos)
ReverMetodo
(fileNumber,
numElementos)
LeDefinição
(fileNumber)
LeQte
(fileNumber,
numElementos)
IniciaListaHandle
(listaHdl,ultListaHdl)
LeIntrodução
(theFile)
(14)
tipoElemento
nodo
conector
decoração,
ponta
InsereListaHandle
(numero,nodoHdl,
listaHdl,ultListaHdl)
LeObjeto
(theFile)
InsereListaHandle
(numero,conectorHdl,
listaHdl,ultListaHdl)

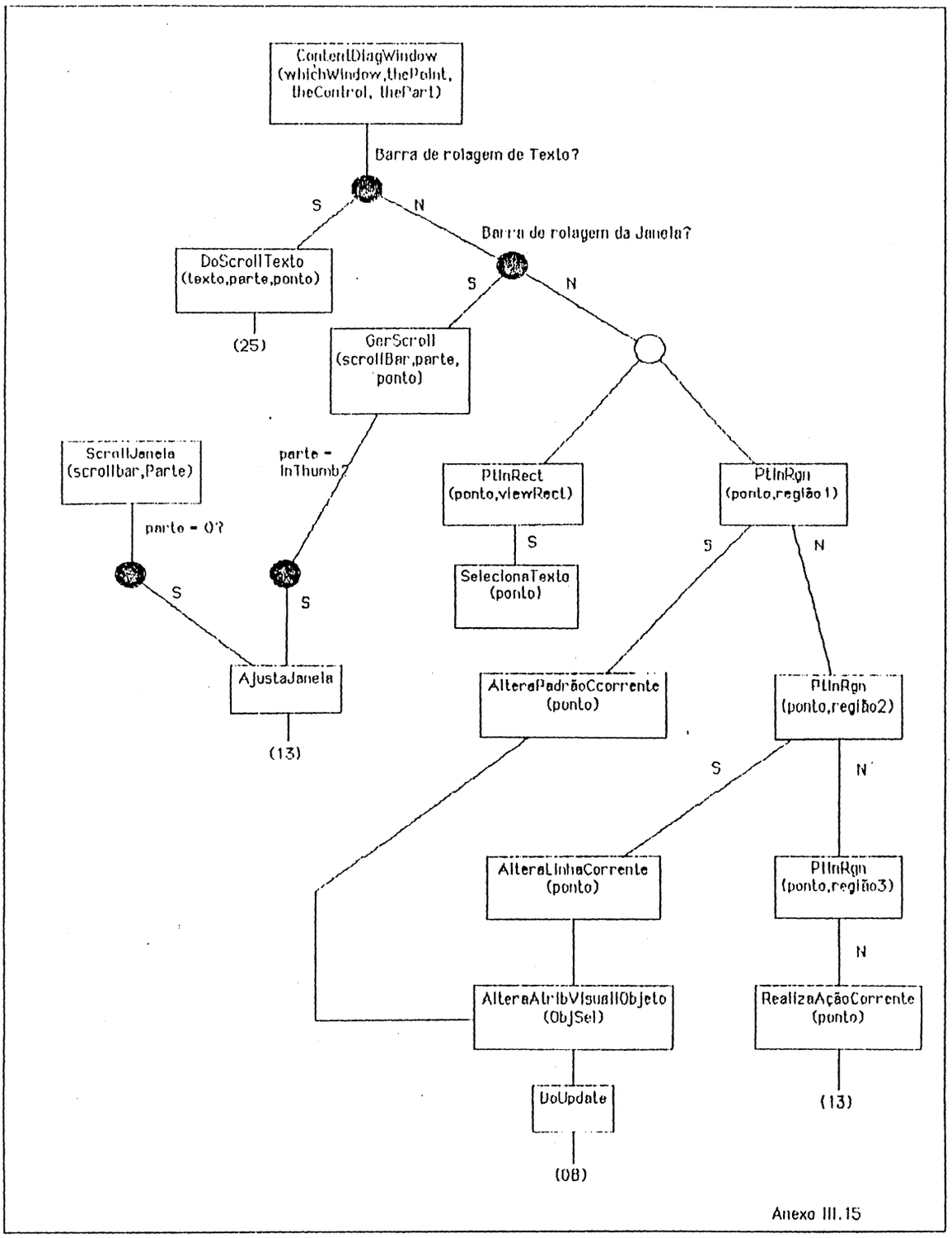
Anexo III.12


RealizaAçãoCorrente
(ponto)
nodo <> nil?
S
N
InsereOutroNodo
(ponto)
decor <> nil?
S
N
InsereOutroDecor
(ponto)
(18)
arco <> nil?
S
N
InsereOutroArco
(ponto)
(17)
RealizaEdição
(ponto)
(20)
AjustaJanela
MudaCoordRetang
(TheWindow)
DoUpdate
(08)
DoTypingTexto
ScrollToSelection
(34)
AdjustScrollBar
AdjustText
DoTypingLabel
DoUpdate
(08)
MostraPegadoresRect
(viewRect)

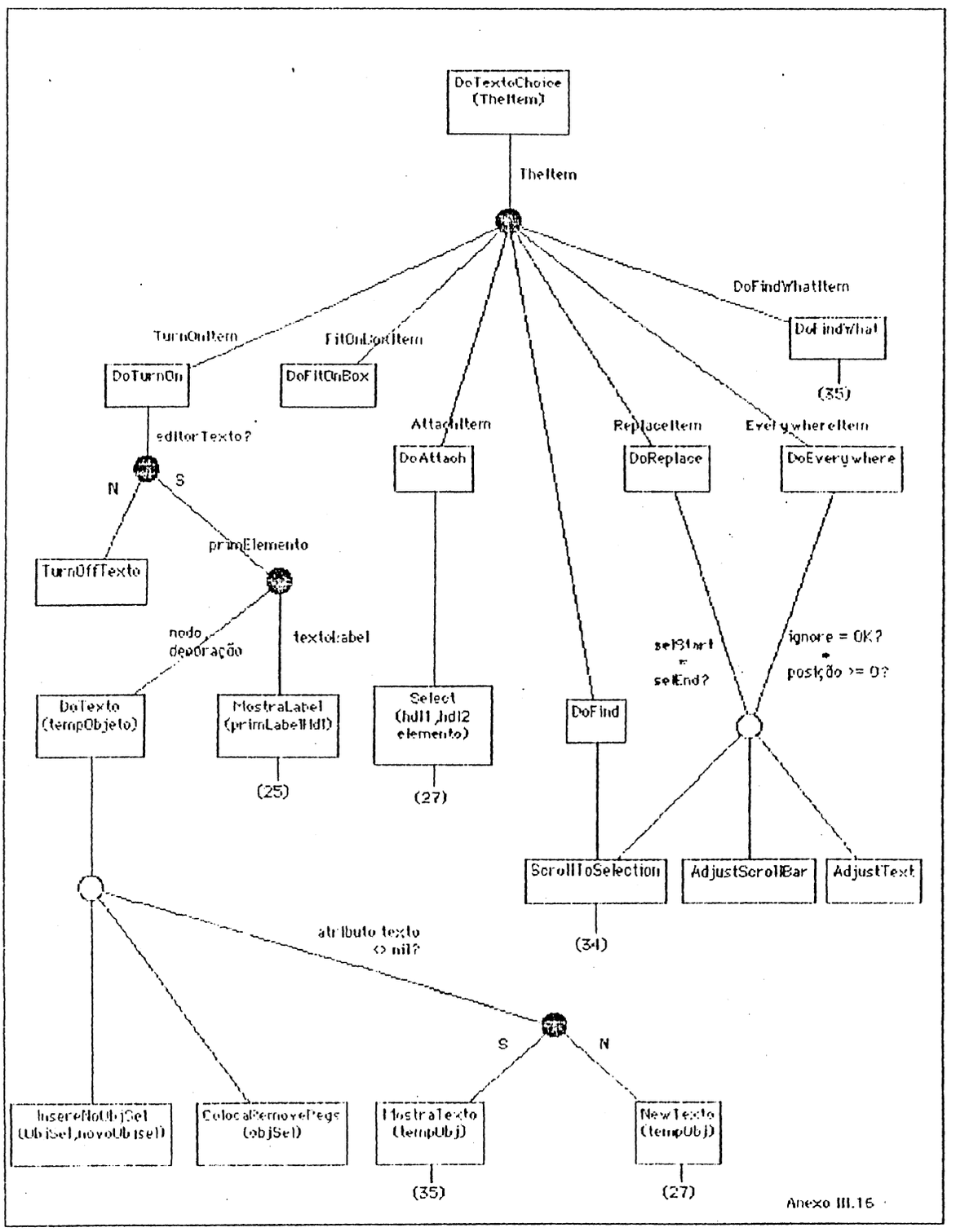
Anexo III.13

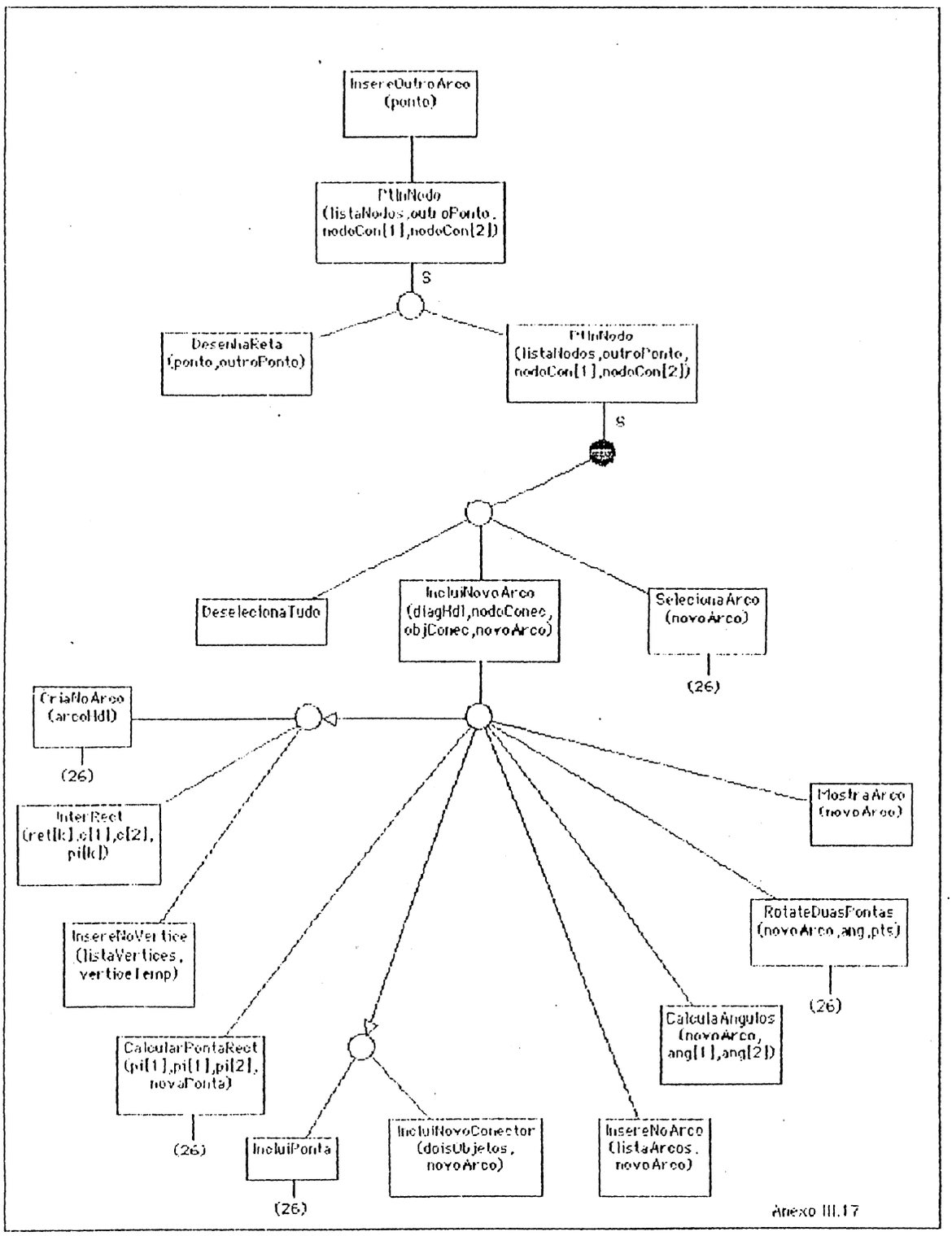

CriaPicArcoWindow
metaPonta <> nil
S
CriaPicNodoWindow
CriaPicDecorWindow
MapeiaPtObjetos
MostraTipObjetos
(primObjeto)
MapeiaPtVertices
(primVertice,
srcRect,dstRect)
DesenhaTipObjetos
(objetoHdl)
primitivo
poligono,
poligReg
DesenhaTipPoligono
(objetoHdl,
primVertice)
LeIntroducao
(TheFile)
InsereNodo
(nodoHdl,idf)
InsereDecoracao
(decorHdl,idf)
InserePonta
(pontaHdl,idf)
InsereConector
(conectorHdl,idf)

Anexo III.14

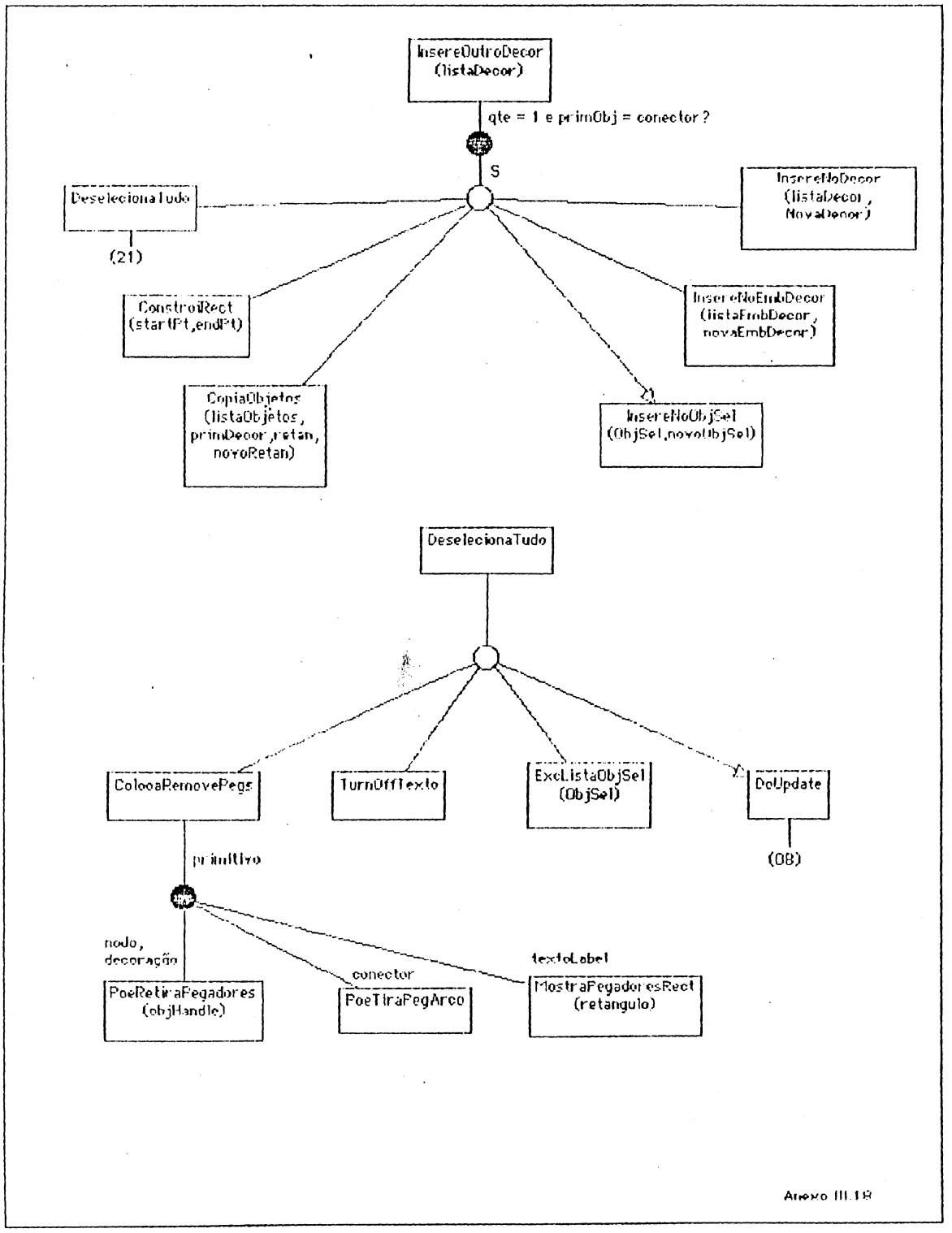

ContentDiagWindow
(whichWindow,thePoint,
theControl, thePart)
Barra de rolagem de Texto?
S
N
DoScrollTexto
(texto,parte,ponto)
(25)
Barra de rolagem da Janela?
S
N
GerScroll
(scrollBar,parte,
ponto)
ScrollJanela
(scrollbar,Parte)
parte = 0?
S
parte =
InThumb?
S
AjustaJanela
(13)
PtInRect
(ponto,viewRect)
S
SelecionaTexto
(ponto)
PtInRgn
(ponto,região1)
S
N
AlteraPadrãoCcorrente
(ponto)
PtInRgn
(ponto,região2)
S
N
AlteraLinhaCorrente
(ponto)
PtInRgn
(ponto,região3)
N
AlteraAtribVisualIObjeto
(ObjSel)
RealizaAçãoCorrente
(ponto)
(13)
DoUpdate
(08)

Anexo III.15

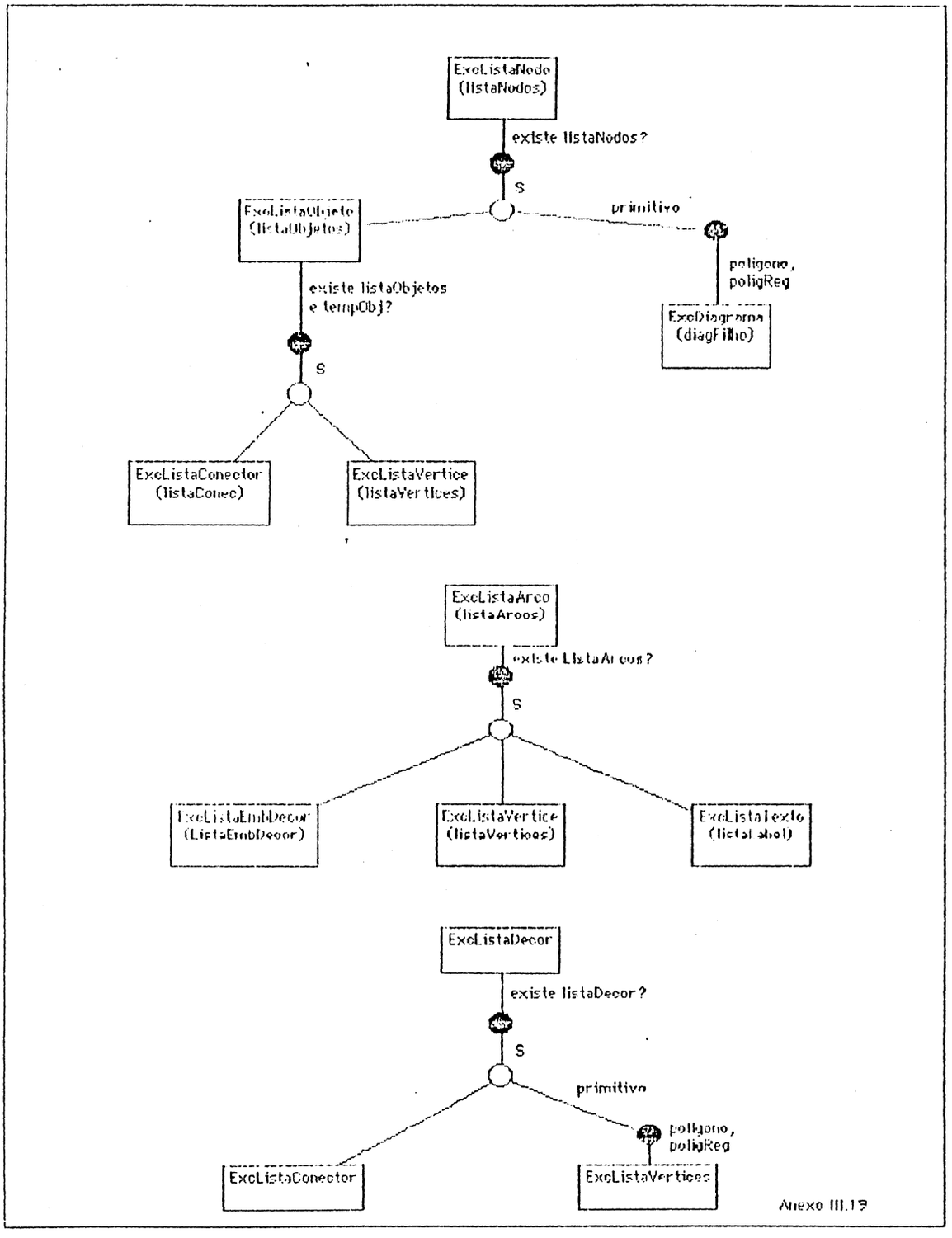

DoTextoChoice (TheItem)
TheItem
DoFindWhatItem
DoFindWhat
(35)
TurnOnItem
FitOnLiorItem
DoTurnOn
DoFitOnBox
editorTexto?
N
S
AttachItem
ReplaceItem
EverywhereItem
DoAttach
DoReplace
DoEverywhere
TurnOffTexto
primElemento
nodo depuração
textoLabel
selStart = selEnd?
ignore = OK? e posição >= 0?
DoTexto (tempObjeto)
MostraLabel (primLabel hdl)
Select (hdl1 ,hdl2 elemento)
DoFind
(25)
(27)
ScrollToSelection
AdjustScrollBar
AdjustText
(34)
atributo texto <> nil?
S
N
InsereNoObjSel (ObjSel,novoObjSel)
ColocaRemoveRegs (objSel)
MostraTexto (tempObj)
NewTexto (tempObj)
(35)
(27)
Anexo III.16


InsereOutroArco
(ponto)
PtInNodo
(listaNodos,outroPonto,
nodoCon[1],nodoCon[2])
S
DesenhaReta
(ponto,outroPonto)
PtInNodo
(listaNodos,outroPonto,
nodoCon[1],nodoCon[2])
S
DeselecionaTudo
IncluiNovoArco
(diagHdl,nodoConec,
objConec,novoArco)
SelecionaArco
(novoArco)
(26)
CriaNoArco
(arcoHdl)
(26)
InterRect
(ret[k],c[1],c[2],
pi[k])
InsereNoVertice
(listaVertices,
verticeTemp)
MostraArco
(novoArco)
RotateDuasPontas
(novoArco,ang,pts)
(26)
CalculaAngulos
(novoArco,
ang[1],ang[2])
CalcularPontaRect
(pi[1],pi[1],pi[2],
novaPonta)
(26)
IncluiPonta
(26)
IncluiNovoConector
(doisObjetos,
novoArco)
InsereNoArco
(listaArcos,
novoArco)

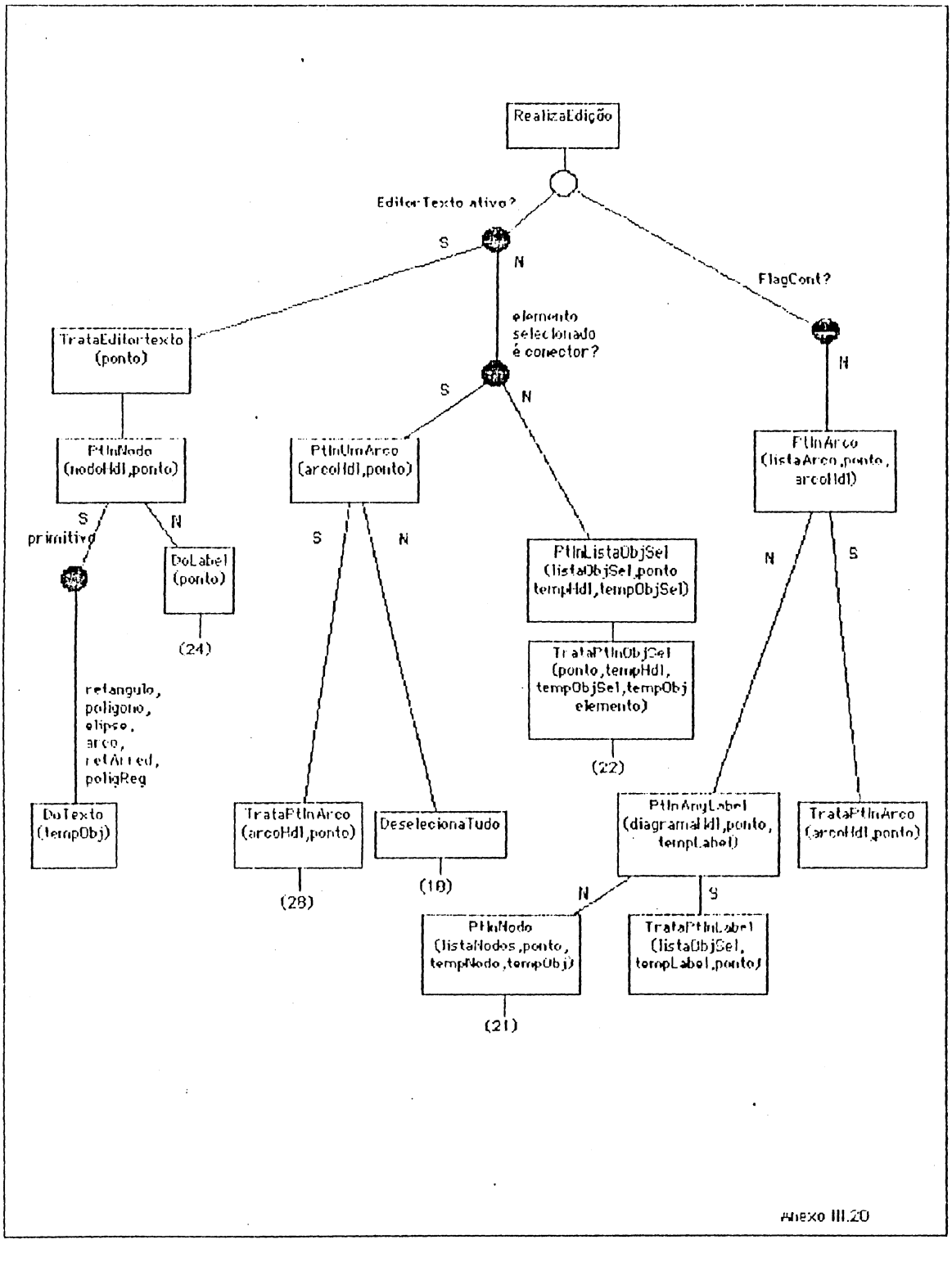
Anexo III.17


InsereOutroDecor
(listaDecor)
qte = 1 e primObj = conector ?
S
DeselecionaTudo
(21)
InsereNoDecor
(listaDecor,
NovaDecor)
ConstroiRect
(startPt,endPt)
InsereNoEmbDecor
(listaEmbDecor,
novaEmbDecor)
CopiaObjetos
(listaObjetos,
primDecor,retan,
novoRetan)
InsereNoObjSel
(ObjSel,novoObjSel)
DeselecionaTudo
ColocaRemovePegs
TurnOffTexto
ExcListaObjSel
(ObjSel)
DoUpdate
(08)
primItiro
nodo,
decoração
conector
textoLabel
PoeRetiraPegadores
(objHandle)
PoeTiraPegArco
MostraPegadoresRect
(retangulo)

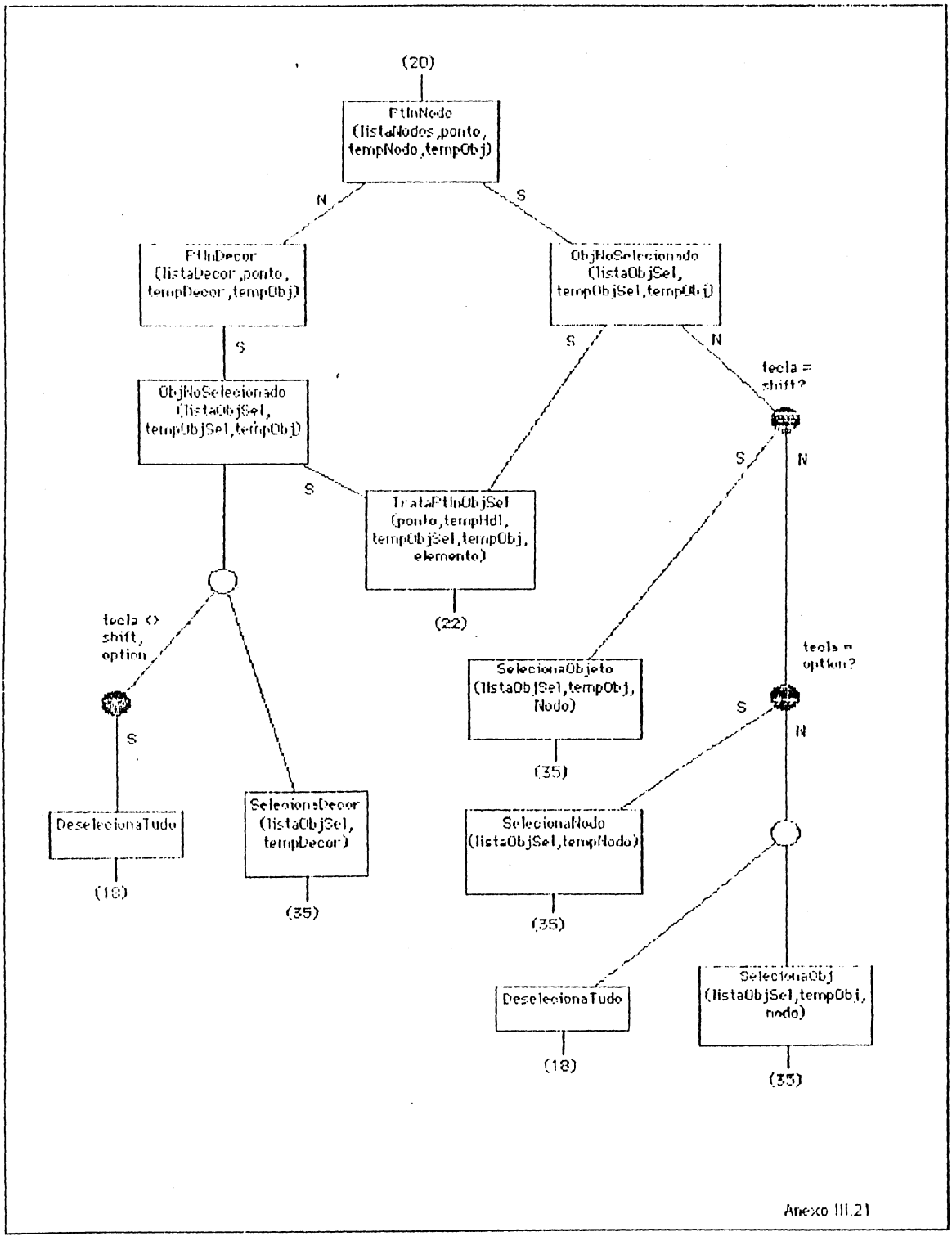
Anexo III.18


ExeListaNodo
(listaNodos)
existe listaNodos?
S
ExeListaObjeto
(listaObjetos)
primitivo
poligono,
poligReg
existe listaObjetos
e tempObj?
ExeDiagrama
(diagFilho)
S
ExeListaConector
(listaConec)
ExeListaVertice
(listaVertices)
ExeListaArco
(listaArcos)
existe listaArcos?
S
ExeListaEmbDecor
(listaEmbDecor)
ExeListaVertice
(listaVertices)
ExeListaTexto
(listaLabel)
ExeListaDecor
existe listaDecor?
S
primitivo
poligono,
poligReg
ExeListaConector
ExeListaVertices

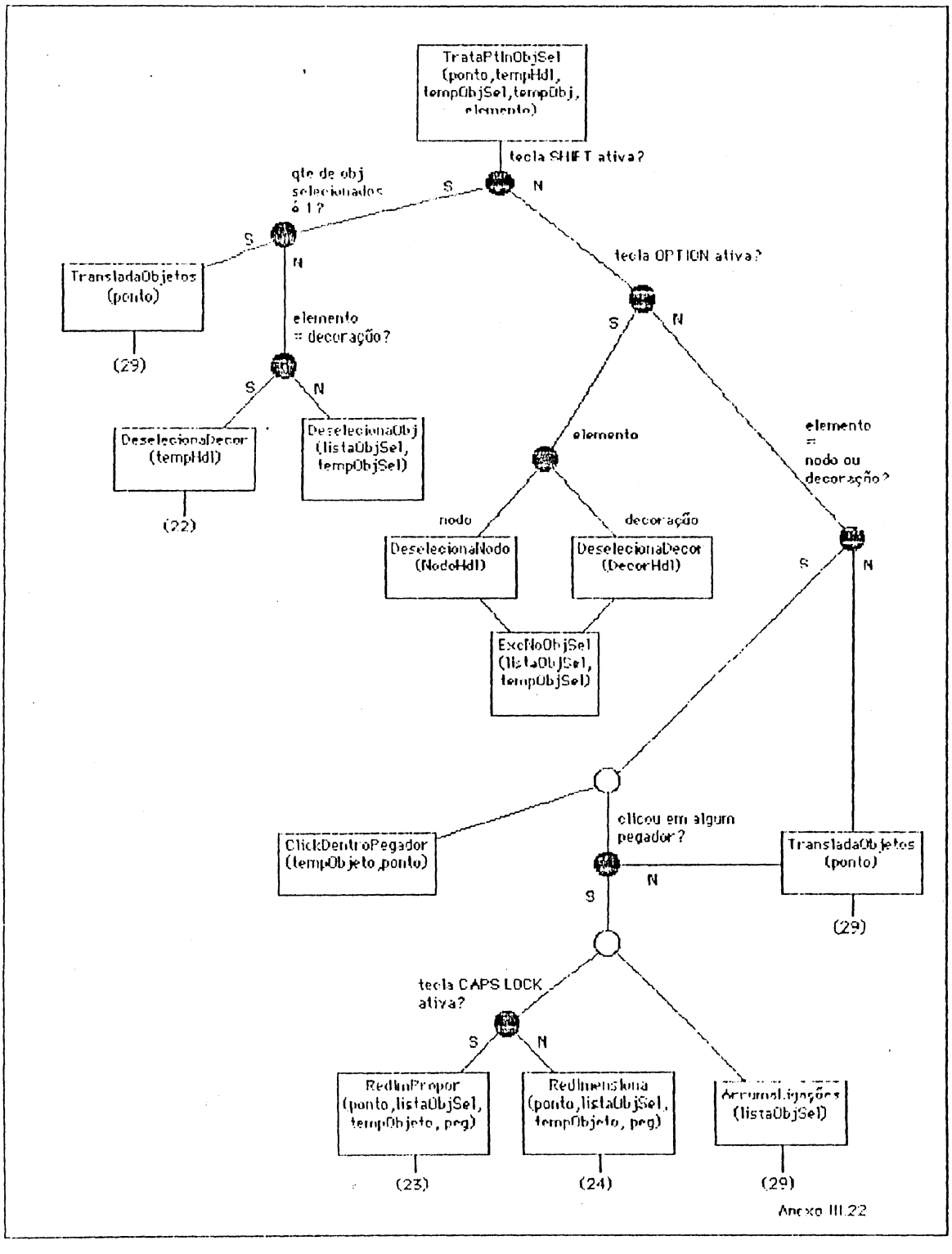
Anexo III.19


RealizaEdição
EditorTexto ativo?
S
N
FlagCont?
TrataEditorTexto (ponto)
elemento seleccionado é conector?
S
N
N
PtInNodo (nodoHdl,ponto)
PtInUmArco (arcoHdl,ponto)
PtInArco (listaArco,ponto, arcoHdl)
S
primitivo
N
S
N
N
S
DoLabel (ponto)
PtInListaObjSel (listaObjSel,ponto tempHdl,tempObjSel)
(24)
TrataPtInObjSel (ponto,tempHdl, tempObjSel,tempObj elemento)
(22)
retangulo, poligono, elipse, arco, retArred, poligReg
DoTexto (tempObj)
TrataPtInArco (arcoHdl,ponto)
DeselecionaTudo
PtInAnyLabel (diagramaHdl,ponto, tempLabel)
TrataPtInArco (arcoHdl,ponto)
(28)
(18)
N
S
PtInNodo (listaNodos,ponto, tempNodo,tempObj)
TrataPtInLabel (listaObjSel, tempLabel,ponto)
(21)

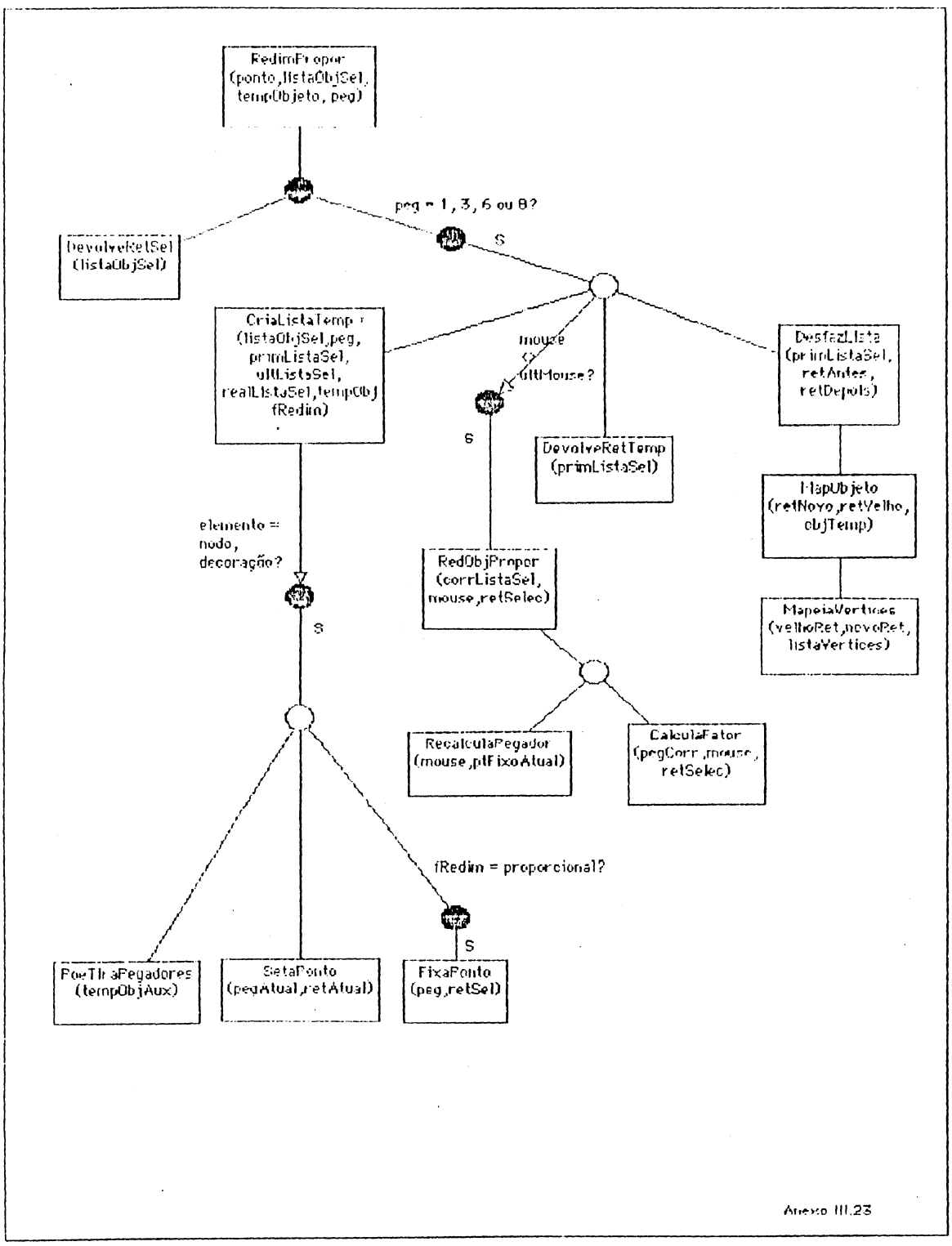
Anexo III.20


(20)
PtInNodo
(listaNodos,ponto,
tempNodo,tempObj)
N
S
PtInDecor
(listaDecor,ponto,
tempDecor,tempObj)
ObjNoSelecionado
(listaObjSel,
tempObjSel,tempObj)
S
S
N
tecla =
shift?
ObjNoSelecionado
(listaObjSel,
tempObjSel,tempObj)
S
S
N
TrataPtInObjSel
(ponto,tempHdl,
tempObjSel,tempObj,
elemento)
(22)
tecla <>
shift,
option
SelecionaObjeto
(listaObjSel,tempObj,
Nodo)
tecla =
option?
S
S
N
(35)
DeselecionaTudo
SelecionaDecor
(listaObjSel,
tempDecor)
SelecionaNodo
(listaObjSel,tempNodo)
(18)
(35)
(35)
DeselecionaTudo
SelecionaObj
(listaObjSel,tempObj,
nodo)
(18)
(35)

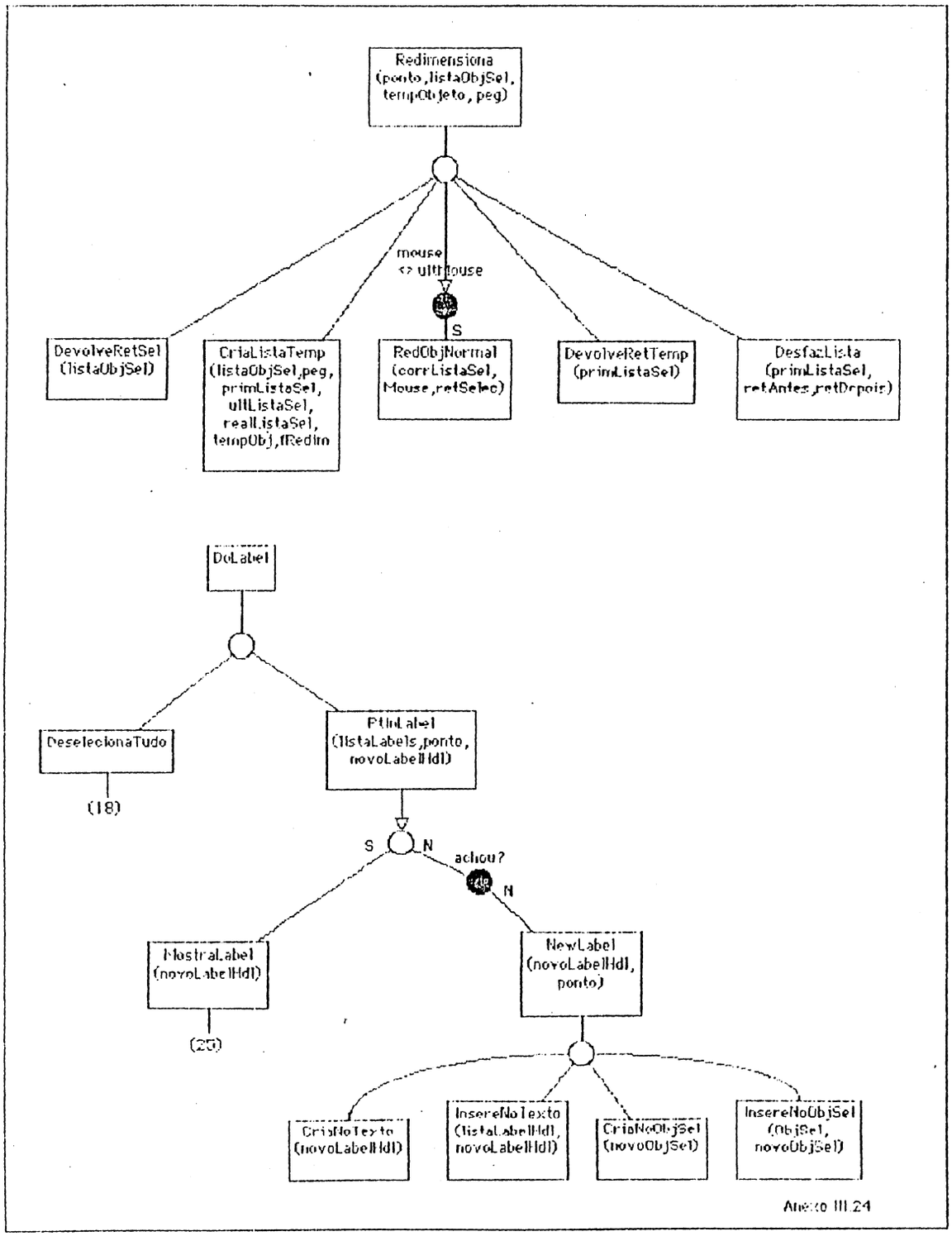
Anexo III.21

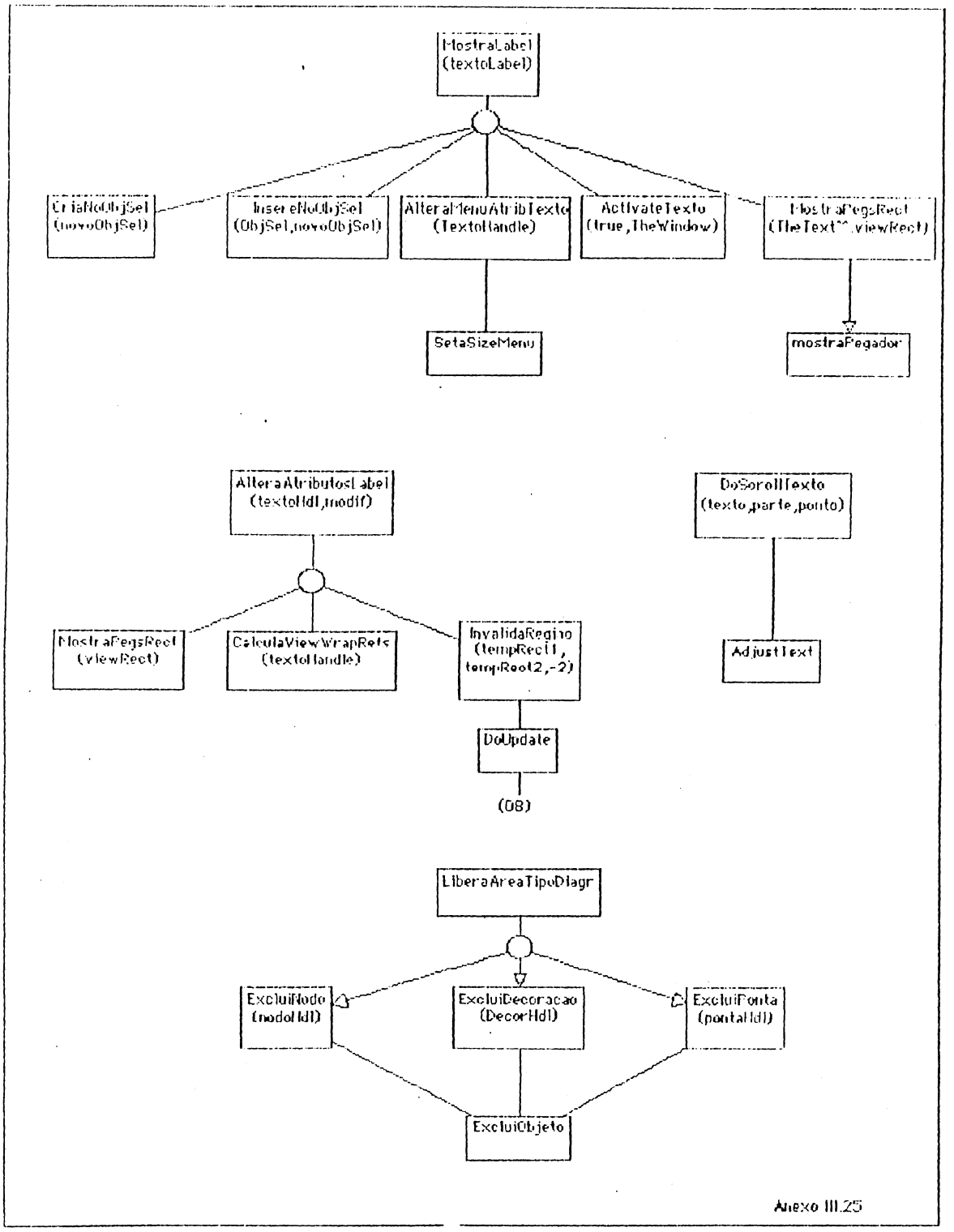

TrataPtInObjSel (ponto,tempHdl, tempObjSel,tempObj, elemento)
tecla SHIFT ativa?
S
N
qte de obj selecionados é 1?
S
N
TransladaObjetos (ponto)
(29)
elemento = decoração?
S
N
DeselecionaDecor (tempHdl)
(22)
DeselecionaObj (listaObjSel, tempObjSel)
tecla OPTION ativa?
S
N
elemento
nodo
decoração
DeselecionaNodo (NodoHdl)
DeselecionaDecor (DecorHdl)
ExcNoObjSel (listaObjSel, tempObjSel)
elemento = nodo ou decoração?
S
N
ClickDentroPegador (tempObjeto,ponto)
clicou em algum pegador?
S
N
TransladaObjetos (ponto)
(29)
tecla CAPS LOCK ativa?
S
N
RedimPropor (ponto,listaObjSel, tempObjeto, peg)
(23)
RedimensIona (ponto,listaObjSel, tempObjeto, peg)
(24)
Acrescentações (listaObjSel)
(29)

Anexo III.22

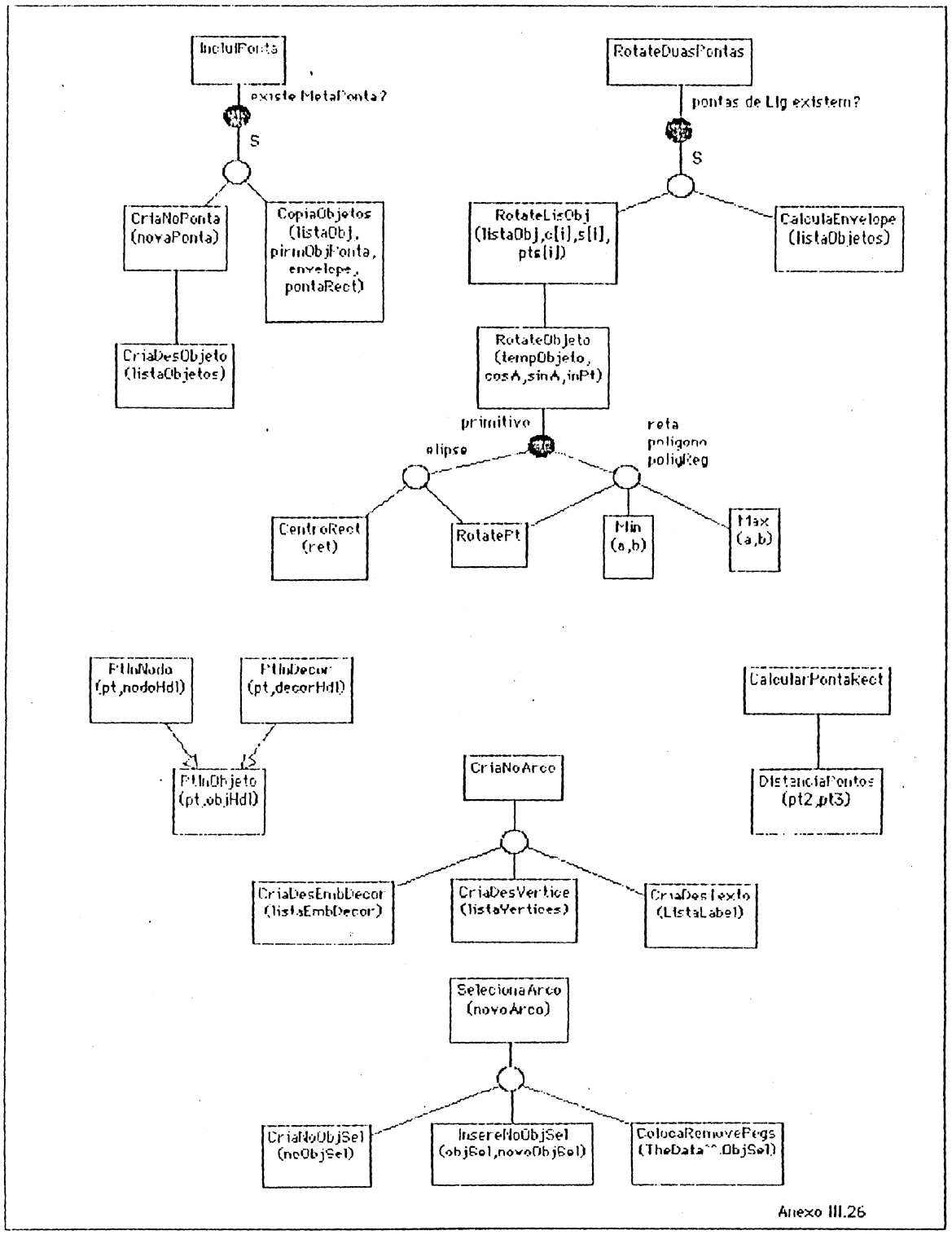

RedimPropor
(ponto,listaObjSel,
tempObjeto, peg)
peg = 1, 3, 6 ou 8?
S
DevolveRetSel
(listaObjSel)
CriaListaTemp
(listaObjSel,peg,
primListaSel,
ultListaSel,
realListaSel,tempObj
fRedim)
mouse
<>
ultMouse?
S
DevolveRetTemp
(primListaSel)
DesfazLista
(primListaSel,
retAntes,
retDepois)
MapObjeto
(retNovo,retVelho,
objTemp)
elemento =
nodo,
decoração?
S
RedObjPropor
(corrListaSel,
mouse,retSelec)
MapeiaVertices
(velhoRet,novoRet,
listaVertices)
RecalculaPegador
(mouse,ptFixoAtual)
CalculaFator
(pegCorr,mouse,
retSelec)
fRedim = proporcional?
S
PoeTiraPegadores
(tempObjAux)
SetaPonto
(pegAtual,retAtual)
FixaPonto
(peg,retSel)

Anexo III.23

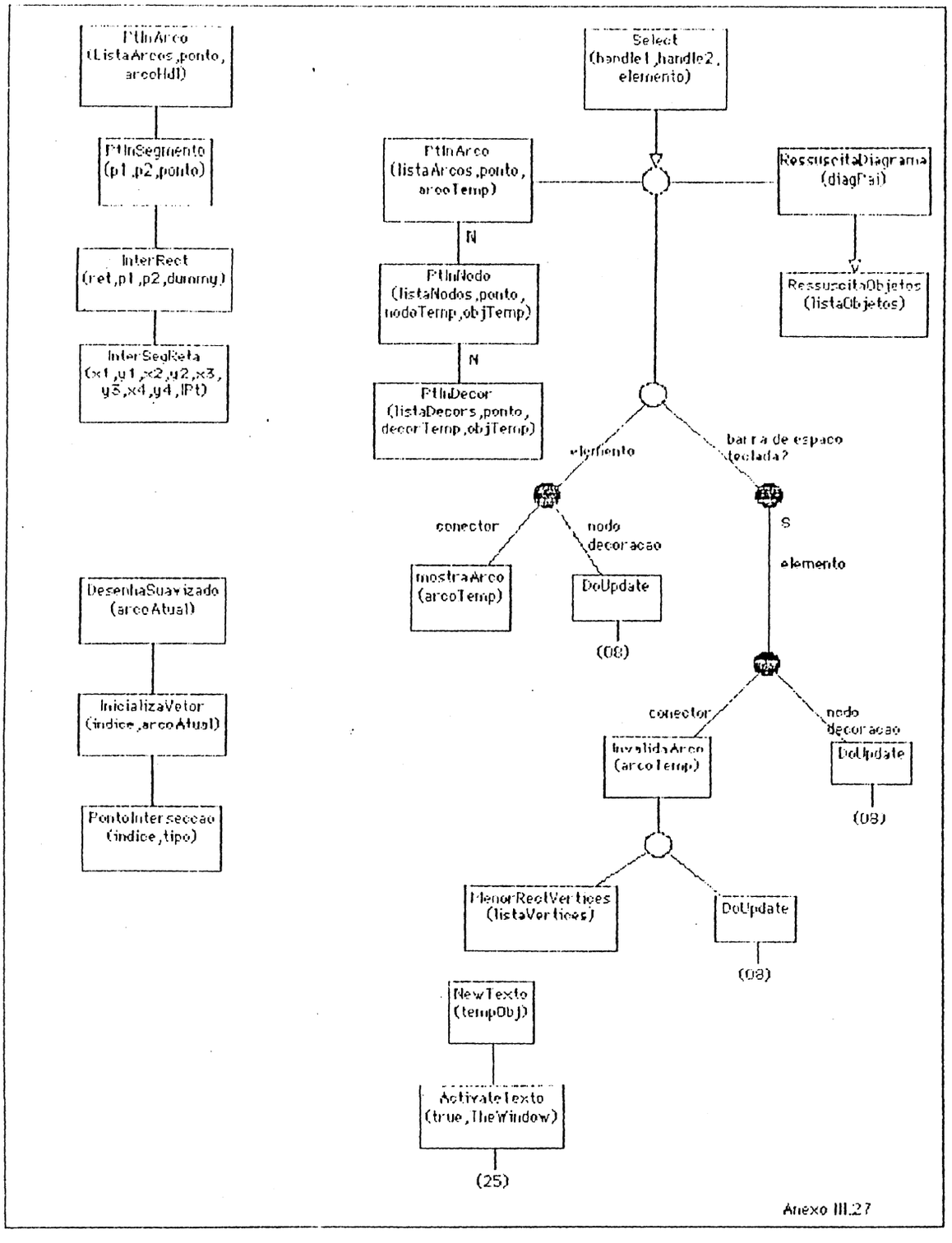

Redimensiona
(ponto,listaObjSel,
tempObjeto, peg)
mouse
<> ultMouse
S
DevolveRetSel
(listaObjSel)
CriaListaTemp
(listaObjSel,peg,
primListaSel,
ultListaSel,
reallistaSel,
tempObj,fRedim
RedObjNormal
(corrListaSel,
Mouse,retSelec)
DevolveRetTemp
(primListaSel)
DesfazLista
(primListaSel,
retAntes,retDepois)
DoLabel
DeselecionaTudo
(18)
PtInLabel
(listaLabels,ponto,
novoLabelHdl)
S
N
achou?
N
MostraLabel
(novoLabelHdl)
(25)
NewLabel
(novoLabelHdl,
ponto)
CriaNoTexto
(novoLabelHdl)
InsereNoTexto
(listaLabelHdl,
novoLabelHdl)
CriaNoObjSel
(novoObjSel)
InsereNoObjSel
(ObjSel,
novoObjSel)
Anexo III.24


MostraLabel
(textoLabel)
CriaNoObjSel
(novoObjSel)
InsereNoObjSel
(ObjSel,novoObjSel)
AlteraMenuAtribTexto
(TextoHandle)
ActivateTexto
(true,TheWindow)
MostraPegsRect
(TheText^.viewRect)
SetaSizeMenu
mostraPegador
AlteraAtributosLabel
(textoHdl,modif)
DoScrollTexto
(texto,parte,ponto)
MostraPegsRect
(viewRect)
CalculaViewWrapRets
(textoHandle)
InvalidaRegion
(tempRect1,
tempRect2,-2)
AdjustText
DoUpdate
(08)
LiberaAreaTipoDiagr
ExcluiNodo
(nodoHdl)
ExcluiDecoracao
(DecorHdl)
ExcluiPonta
(pontaHdl)
ExcluiObjeto

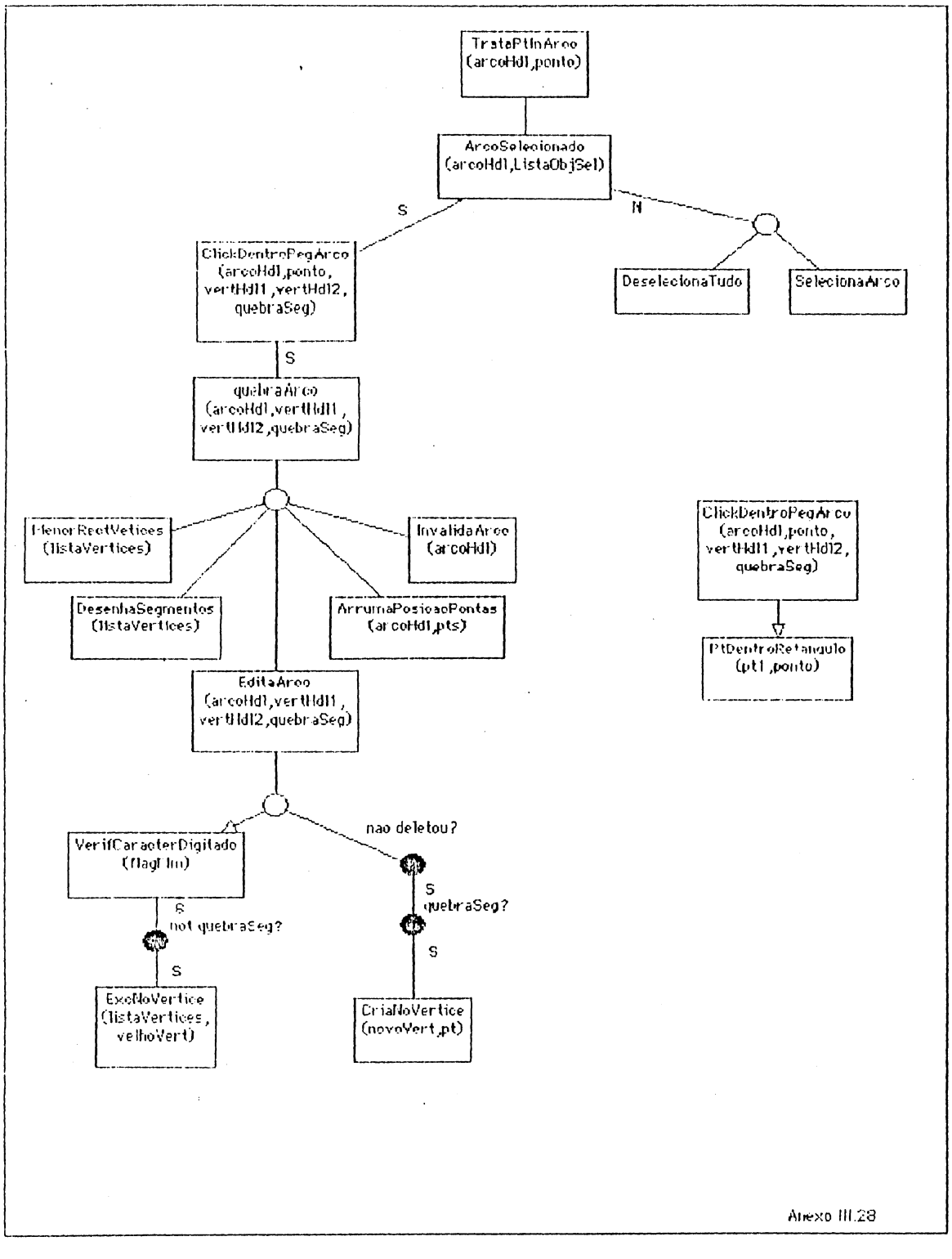
Anexo III.25


IncluiPonta
existe MetaPonta?
S
CriaNoPonta (novaPonta)
CopiaObjetos (listaObj, primObjPonta, envelope, pontaRect)
CriaDesObjeto (listaObjetos)
RotateDuasPontas
pontas de Lig existem?
S
RotateLisObj (listaObj,c[i],s[i], pts[i])
CalculaEnvelope (listaObjetos)
RotateObjeto (tempObjeto, cosA,sinA,inPt)
primitivo
elipse
reta
poligono
poliReg
CentroRect (ret)
RotatePt
Min (a,b)
Max (a,b)
PtInNodo (pt,nodoHdl)
PtInDecor (pt,decorHdl)
PtInObjeto (pt,objHdl)
CriaNoArco
CriaDesEmbDecor (listaEmbDecor)
CriaDesVertice (listaVertices)
CriaDesTexto (ListaLabel)
CalcularPontaRect
DistanciaPontos (pt2,pt3)
SelecionaArco (novoArco)
CriaNoObjSel (noObjSel)
InsereNoObjSel (objSel,novoObjSel)
ColocaRemovePegs (TheData^.ObjSel)

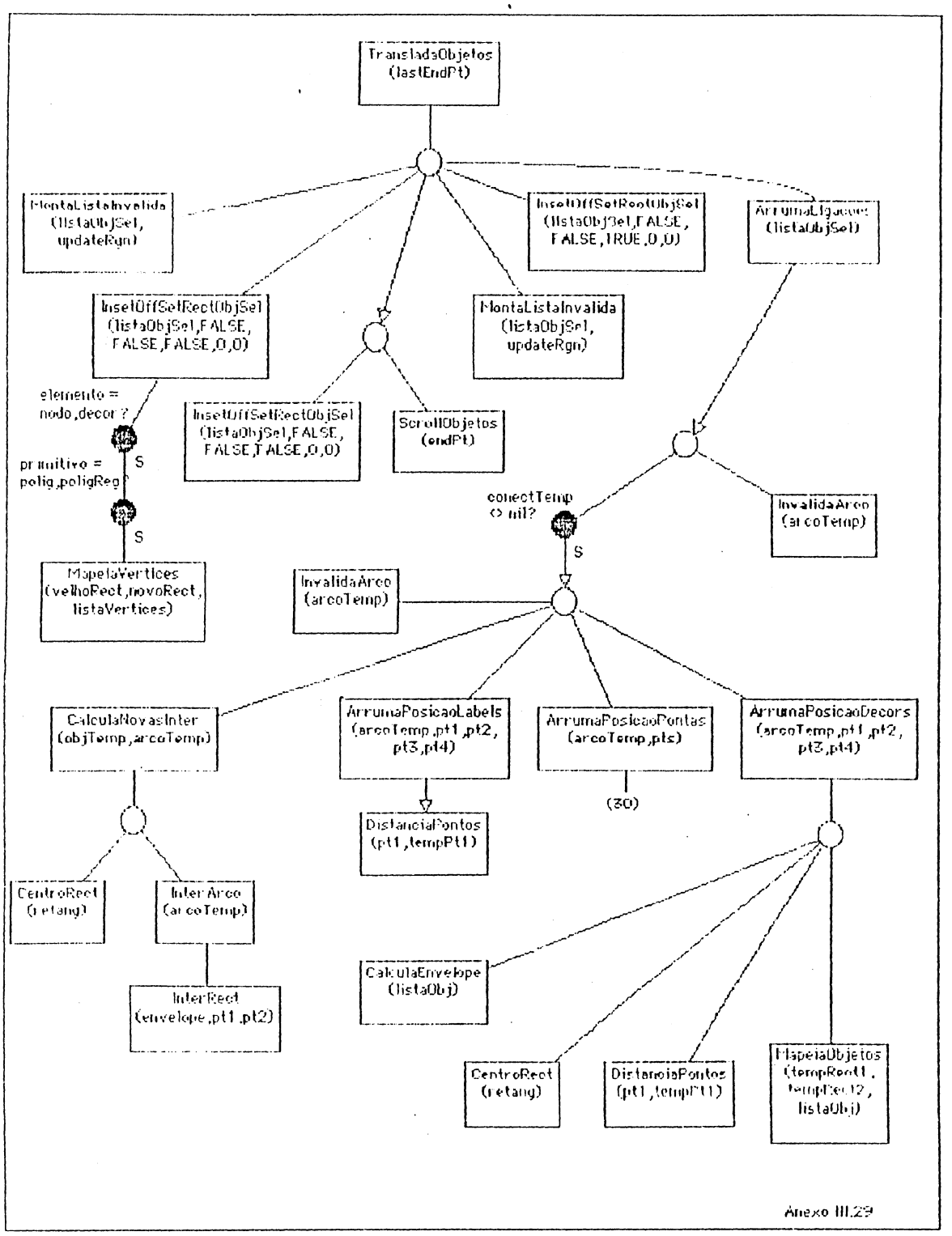
Anexo III.26


PtInArco
(ListaArcos,ponto,
arcoHdl)
PtInSegmento
(p1,p2,ponto)
InterRect
(ret,p1,p2,dummy)
InterSegReta
(x1,y1,x2,y2,x3,
y3,x4,y4,IPt)
DesenhaSuavizado
(arcoAtual)
InicializaVetor
(indice,arcoAtual)
PontoInterseccao
(indice,tipo)
Select
(handle1,handle2,
elemento)
PtInArco
(listaArcos,ponto,
arcoTemp)
N
PtInNodo
(listaNodos,ponto,
nodoTemp,objTemp)
N
PtInDecor
(listaDecors,ponto,
decorTemp,objTemp)
RessuscitaDiagrama
(diagPai)
RessuscitaObjetos
(listaObjetos)
elemento
barra de espaco
teclada?
conector
nodo
decoracao
S
elemento
mostraArco
(arcoTemp)
DoUpdate
(08)
conector
nodo
decoracao
InvalidaArco
(arcoTemp)
DoUpdate
(08)
MenorRectVertices
(listaVertices)
DoUpdate
(08)
NewTexto
(tempObj)
ActivateTexto
(true,TheWindow)
(25)

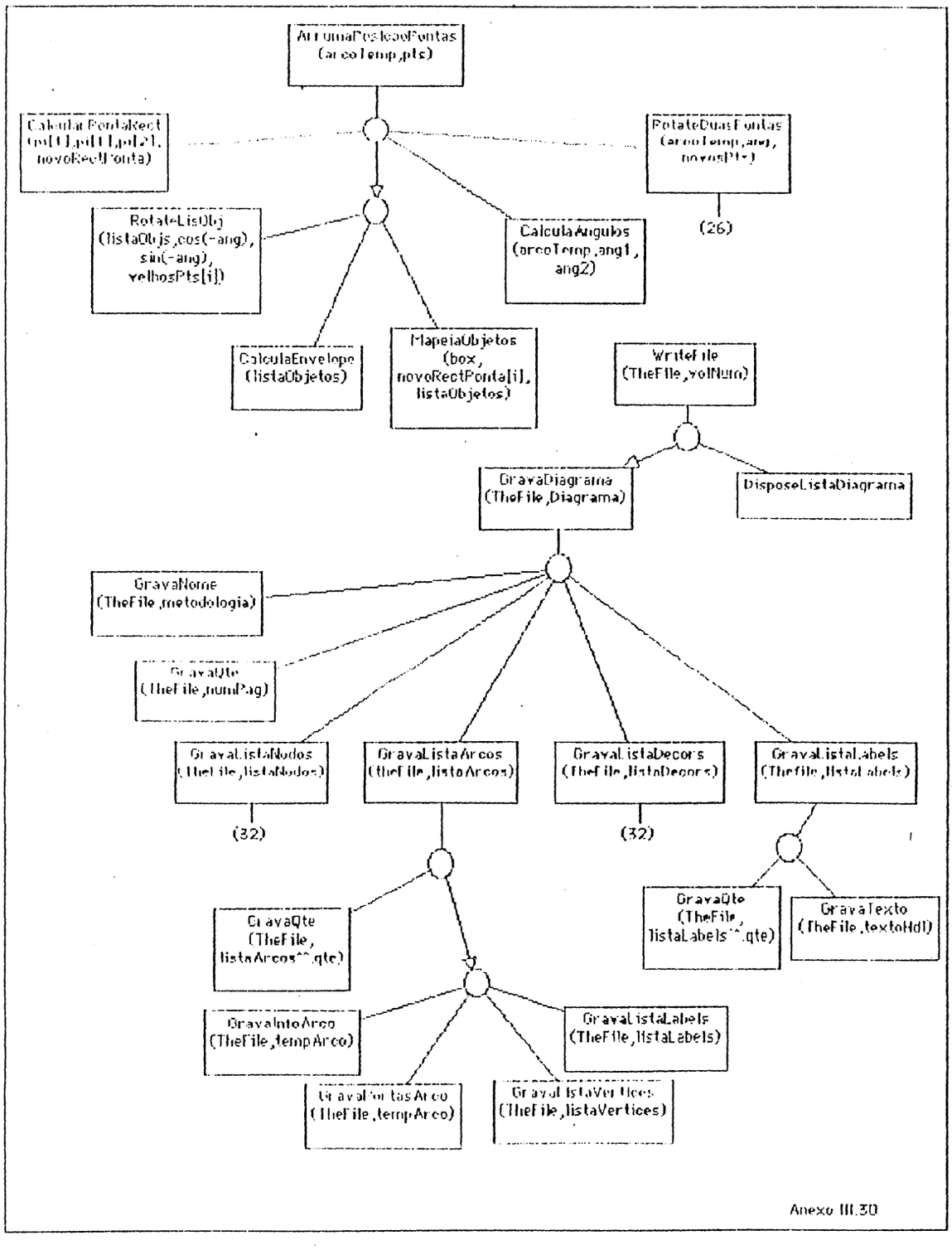
Anexo III.27


TrataPtInArco
(arcoHdl,ponto)
ArcoSelecionado
(arcoHdl,ListaObjSel)
S
N
ClickDentroPegArco
(arcoHdl,ponto,
vertHdl1,vertHdl2,
quebraSeg)
DeselecionaTudo
SelecionaArco
S
quebraArco
(arcoHdl,vertHdl1,
vertHdl2,quebraSeg)
MenorRectVetices
(listaVertices)
InvalidaArco
(arcoHdl)
DesenhaSegmentos
(listaVertices)
ArrumaPosicaoPontas
(arcoHdl,pts)
EditaArco
(arcoHdl,vertHdl1,
vertHdl2,quebraSeg)
ClickDentroPegArco
(arcoHdl,ponto,
vertHdl1,vertHdl2,
quebraSeg)
PtDentroRetangulo
(pt1,ponto)
VerifCaracterDigitado
(flagFim)
nao deletou?
S
not quebraSeg?
S
quebraSeg?
S
S
ExcNoVertice
(listaVertices,
velhoVert)
CriaNoVertice
(novoVert,pt)

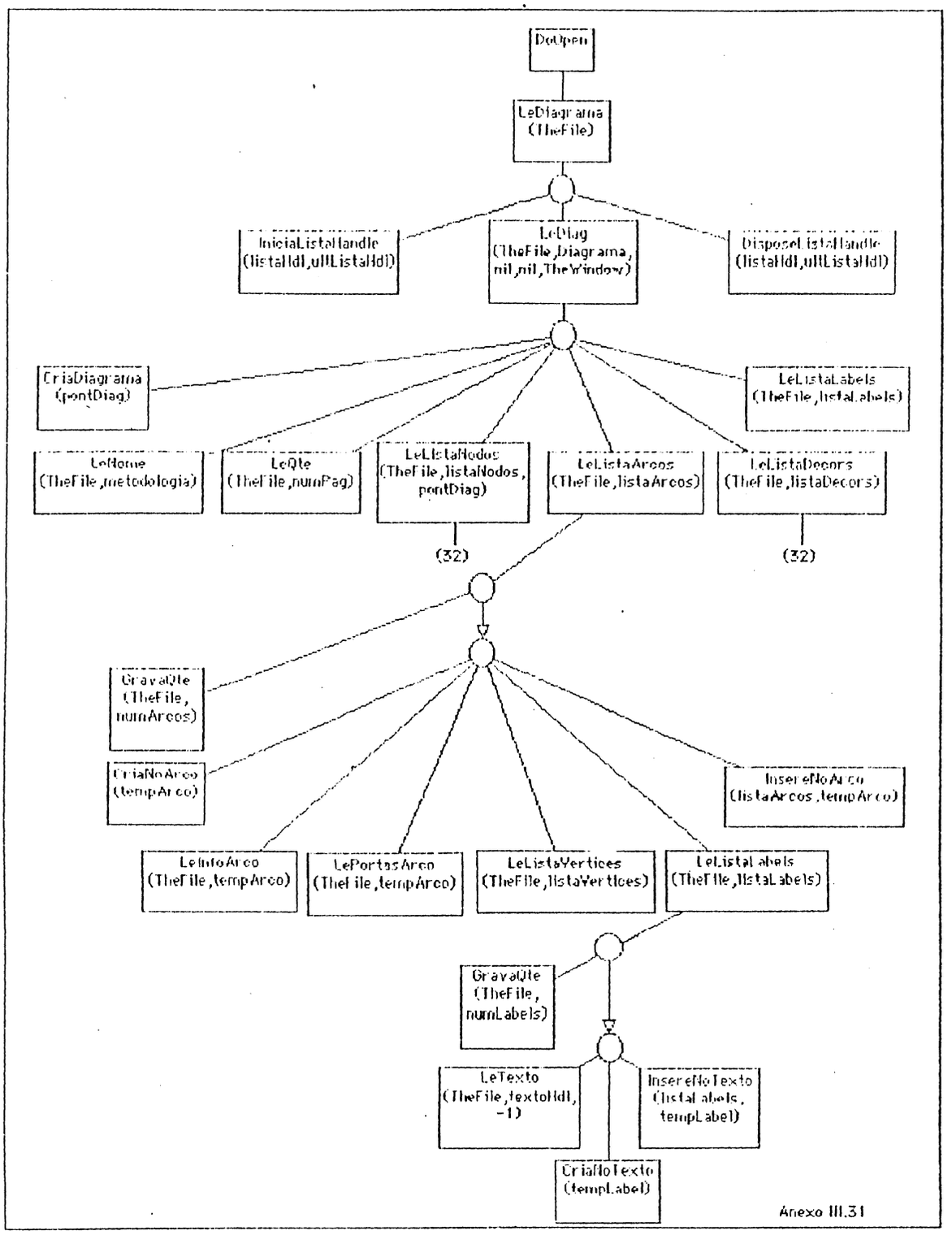
Anexo III.28


TransladaObjetos (lastEndPt)
MontaListaInvalida (listaObjSel, updateRgn)
InsetOffSetRectObjSel (listaObjSel,FALSE, FALSE,TRUE,0,0)
ArrumaLigacoes (listaObjSel)
InsetOffSelRectObjSel (listaObjSel,FALSE, FALSE,FALSE,0,0)
MontaListaInvalida (listaObjSel, updateRgn)
elemento = nodo,decor ?
InsetOffSelRectObjSel (listaObjSel,FALSE, FALSE,FALSE,0,0)
ScrollObjetos (endPt)
S
primitivo = polig,poligReg ?
S
conectTemp <> nil?
S
InvalidaArco (arcoTemp)
MapeiaVertices (velhoRect,novoRect, listaVertices)
InvalidaArco (arcoTemp)
CalculaNovasInter (objTemp,arcoTemp)
ArrumaPosicaoLabels (arcoTemp,pt1,pt2, pt3,pt4)
ArrumaPosicaoPontas (arcoTemp,pts)
ArrumaPosicaoDecors (arcoTemp,pt1,pt2, pt3,pt4)
(30)
DistancialPontos (pt1,tempPt1)
CentroRect (retang)
InterArco (arcoTemp)
InterRect (envelope,pt1,pt2)
CalculaEnvelope (listaObj)
CentroRect (retang)
DistanciaPontos (pt1,tempPt1)
MapeiaObjetos (tempRect1, tempRect2, listaObj)

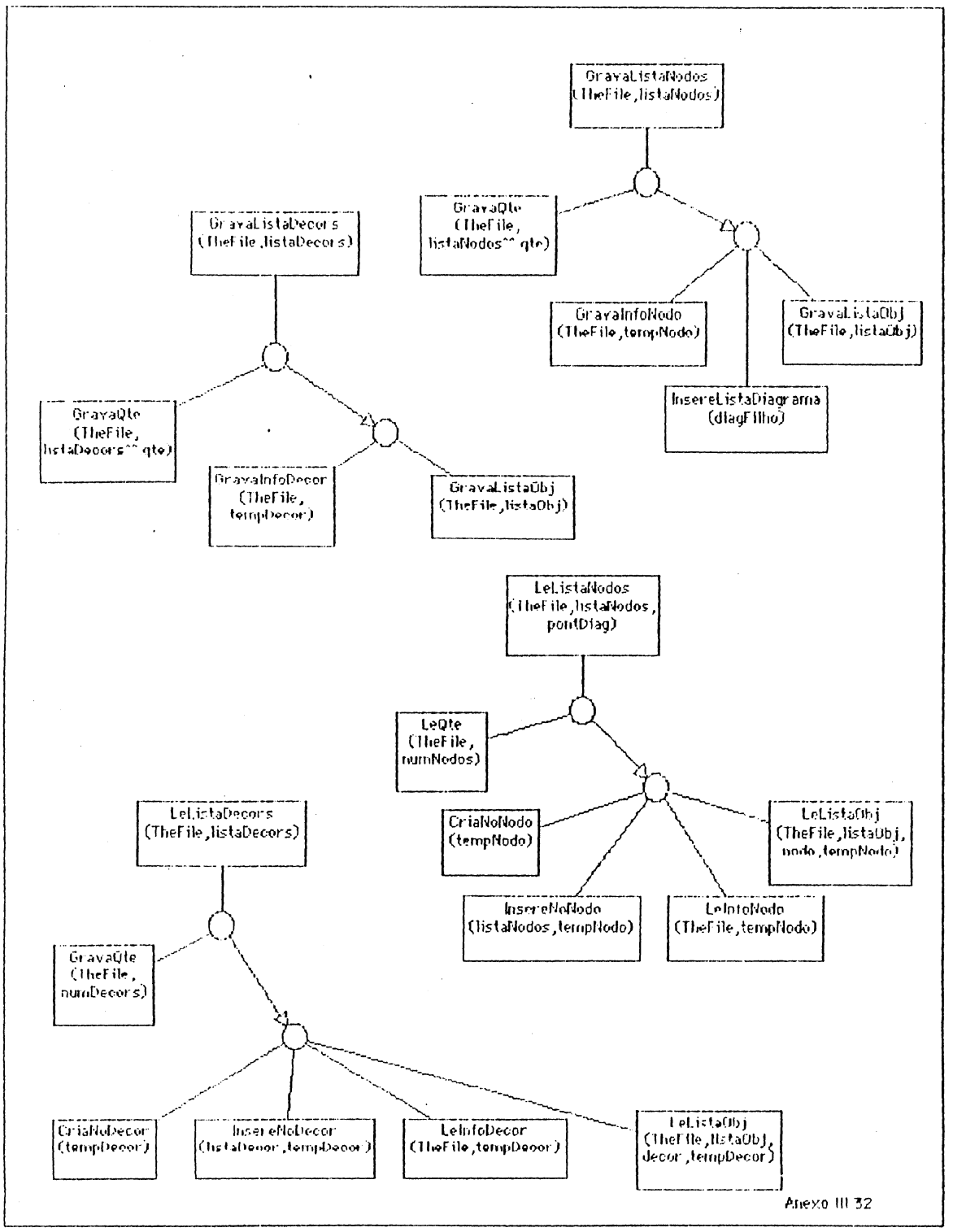
Anexo III.29


ArrumaPosicaoPontas
(arcoTemp,pts)
CalculaPontaRect
(pt[1],pt[2],
novoRectPonta)
RotateDuasPontas
(arcoTemp,ang,
novosPts)
(26)
RotateListObj
(listaObjs,cos(-ang),
sin(-ang),
velhosPts[i])
CalculaAngulos
(arcoTemp,ang1,
ang2)
CalculaEnvelope
(listaObjetos)
MapeiaObjetos
(box,
novoRectPonta[i],
listaObjetos)
WriteFile
(TheFile,volNum)
GravaDiagrama
(TheFile,Diagrama)
DisposeListaDiagrama
GravaNome
(TheFile,metodologia)
GravaQte
(TheFile,numPag)
GravaListaNodos
(TheFile,listaNodos)
GravaListaArcos
(TheFile,listaArcos)
GravaListaDecors
(TheFile,listaDecors)
GravaListaLabels
(TheFile,listaLabels)
(32)
(32)
GravaQte
(TheFile,
listaArcos^.qte)
GravaQte
(TheFile,
listaLabels^.qte)
GravaTexto
(TheFile,textoHdl)
GravaIntoArco
(TheFile,tempArco)
GravaListaLabels
(TheFile,listaLabels)
GravaPontasArco
(TheFile,tempArco)
GravaListaVertices
(TheFile,listaVertices)

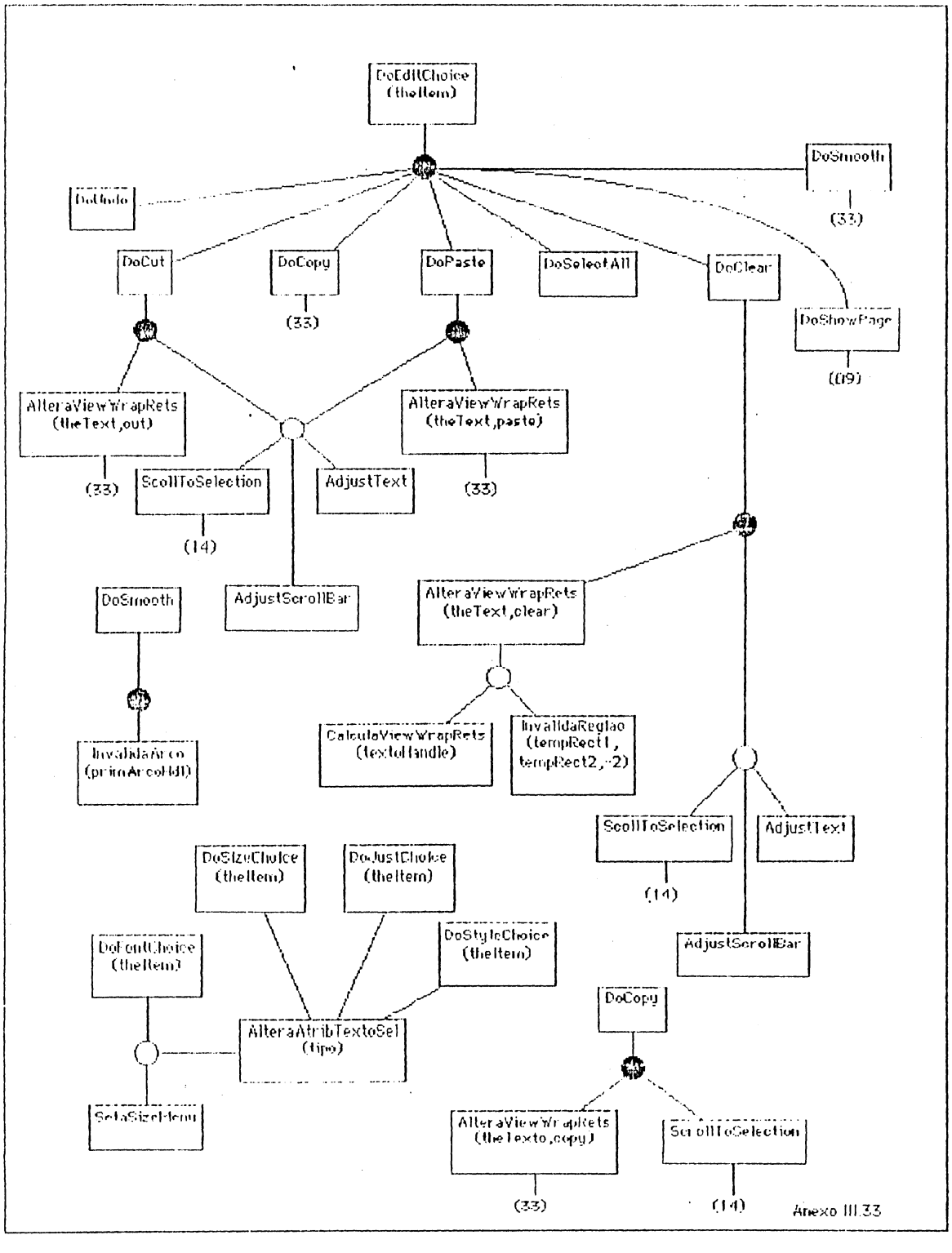
Anexo III.30

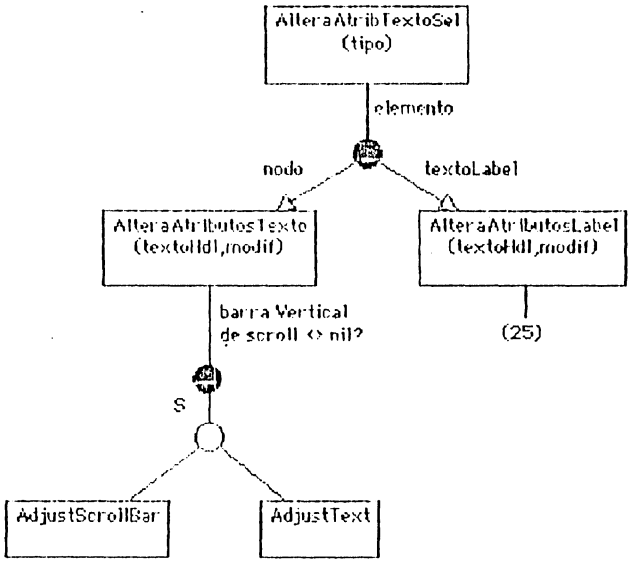

DoOpen
LeDiagrama (TheFile)
IniciaListaHandle (listaHdl,ulIListaHdl)
LeDiag (TheFile,Diagrama, nil,nil,TheWindow)
DisposeListaHandle (listaHdl,ulIListaHdl)
CriaDiagrama (pontDiag)
LeListaLabels (TheFile,listaLabels)
LeNome (TheFile,metodologia)
LeQte (TheFile,numPag)
LeListaNodos (TheFile,listaNodos, pontDiag)
LeListaArcos (TheFile,listaArcos)
LeListaDecors (TheFile,listaDecors)
(32)
(32)
GravaQte (TheFile, numArcos)
CriaNoArco (tempArco)
InsereNoArco (listaArcos,tempArco)
LeInfoArco (TheFile,tempArco)
LePortasArco (TheFile,tempArco)
LeListaVertices (TheFile,listaVertices)
LeListaLabels (TheFile,listaLabels)
GravaQte (TheFile, numLabels)
LeTexto (TheFile,textoHdl, -1)
InsereNoTexto (listaLabels, tempLabel)
CriaNoTexto (tempLabel)

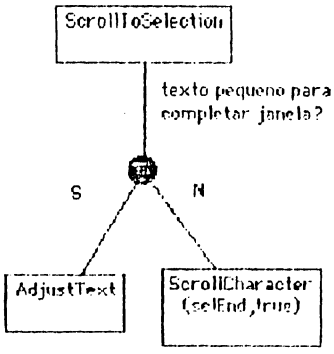
Anexo III.31

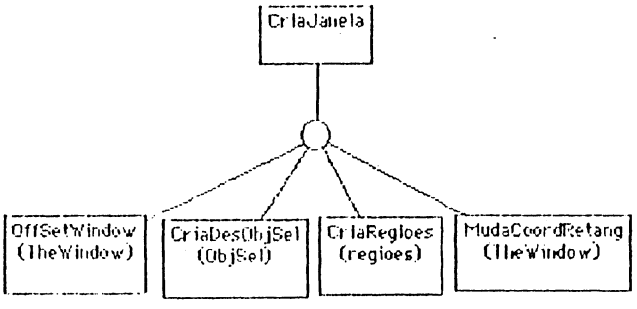

GravaListaNodos (TheFile,listaNodos)
GravaQte (TheFile, listaNodos^^ qte)
GravaListaDecors (TheFile,listaDecors)
GravaInfoNodo (TheFile,tempNodo)
GravaListaObj (TheFile,listaObj)
InsereListaDiagrama (diagFilho)
GravaQte (TheFile, listaDecors^^ qte)
GravaInfoDecor (TheFile, tempDecor)
GravaListaObj (TheFile,listaObj)
LeListaNodos (TheFile,listaNodos, pontDiag)
LeQte (TheFile, numNodos)
LeListaDecors (TheFile,listaDecors)
CriaNoNodo (tempNodo)
LeListaObj (TheFile,listaObj, nodo,tempNodo)
InsereNoNodo (listaNodos,tempNodo)
LeInfoNodo (TheFile,tempNodo)
GravaQte (TheFile, numDecors)
CriaNoDecor (tempDecor)
InsereNoDecor (listaDecor,tempDecor)
LeInfoDecor (TheFile,tempDecor)
LeListaObj (TheFile,listaObj, decor,tempDecor)

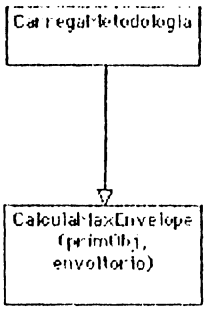
Anexo III 32


DoEditChoice (theItem)
DoSmooth
(33)
DoUndo
DoCut
DoCopy
DoPaste
DoSelectAll
DoClear
DoShowPage
(09)
(33)
AlteraViewWrapRets (theText,out)
AlteraViewWrapRets (theText,paste)
(33)
(33)
ScollToSelection
AdjustText
(14)
DoSmooth
AdjustScrollBar
AlteraViewWrapRets (theText,clear)
InvalidaArco (primArcoHdl)
CalculaViewWrapRets (textoHandle)
InvalidaRegiao (tempRect1, tempRect2,-2)
ScollToSelection
AdjustText
(14)
DoSizeChoice (theItem)
DoJustChoice (theItem)
DoStyleChoice (theItem)
DoFontChoice (theItem)
AdjustScrollBar
AlteraAtribTextoSel (tipo)
DoCopy
SetaSizeItem
AlteraViewWrapRets (theTexto,copy)
ScrollToSelection
(33)
(14)
Anexo III.33

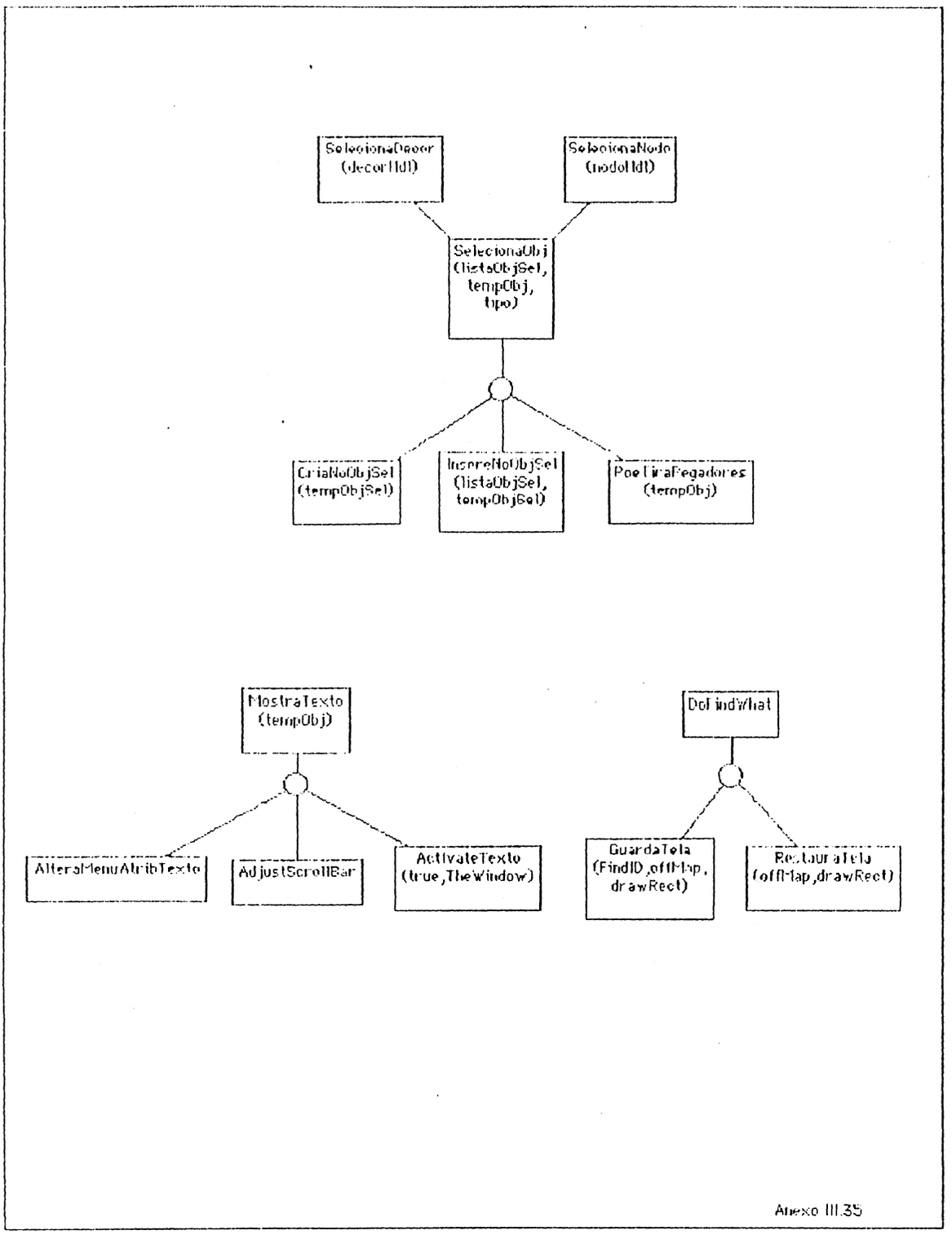

AlteraAtribTextoSel
(tipo)
elemento
nodo
textoLabel
AlteraAtributosTexto
(textoIdl,modif)
AlteraAtributosLabel
(textoIdl,modif)
barra Vertical
de scroll <> nil?
(25)
S
AdjustScrollBar
AdjustText


ScrollToSelection
texto pequeno para
completar janela?
S
N
AdjustText
ScrollCharacter
(selEnd,true)

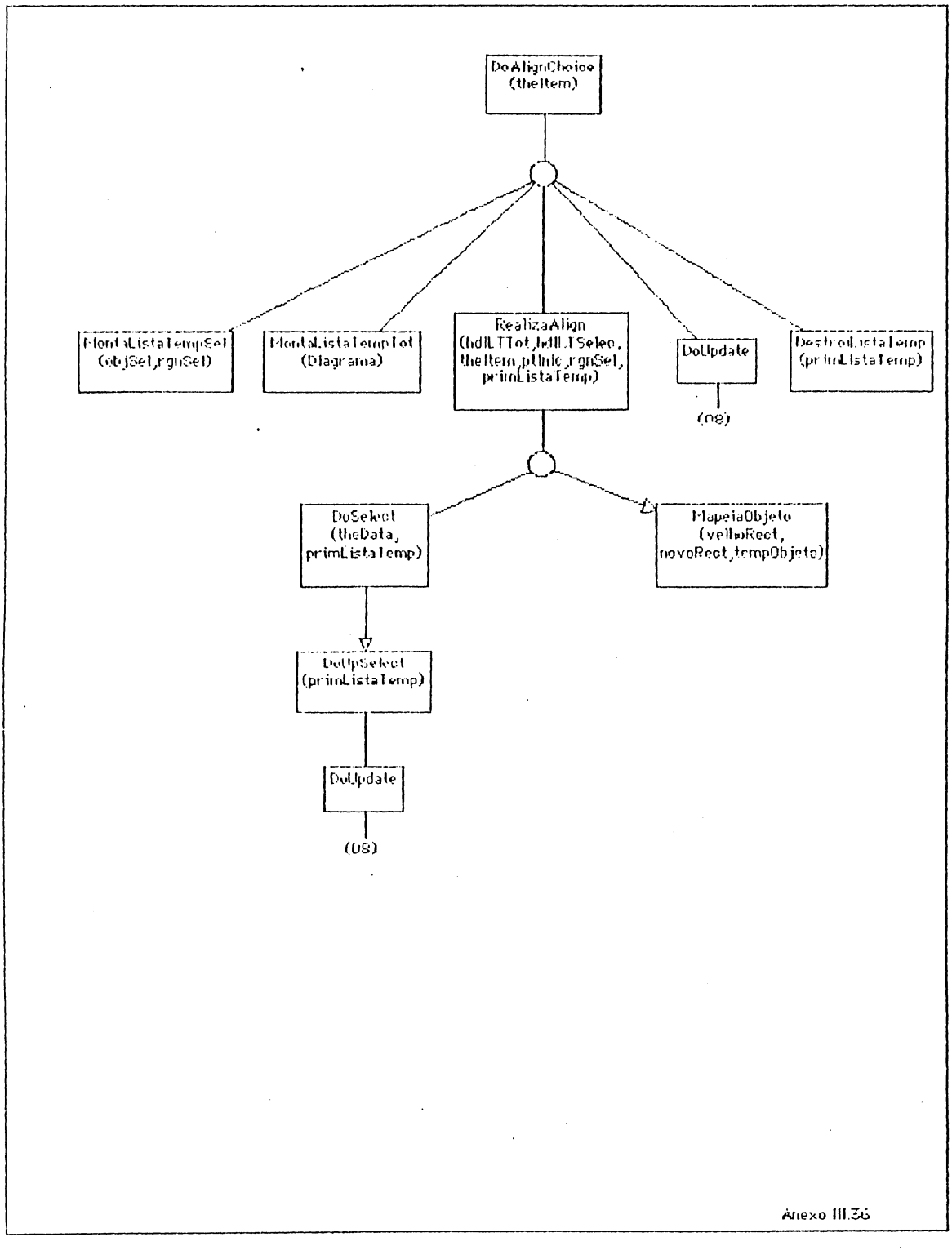

CriaJanela
OffSetWindow
(TheWindow)
CriaDesObjSel
(ObjSel)
CriaRegioes
(regioes)
MudaCoordRetang
(TheWindow)


CarregaMetodologia
CalculaMaxEnvelope
(primObj,
envoltorio)

Anexo III.34


SelecionaDecor (decorIdI)
SelecionaNodo (nodeIdI)
SelecionaObj (listaObjSel, tempObj, tipo)
CriaNoObjSel (tempObjSel)
InsereNoObjSel (listaObjSel, tempObjSel)
PoeLinkPegadores (tempObj)
MostraTexto (tempObj)
AlteraMenuAtribTexto
AdjustScrollBar
ActivateTexto (true,TheWindow)
DoFindWhat
GuardaTela (FindID,offMap, drawRect)
RestauraTela (offMap,drawRect)

Anexo III.35


DoAlignChoice
(theItem)
MontaListaTempSel
(objSel,rgnSel)
MontaListaTempTot
(Diagrama)
RealizaAlign
(hdlLTTot,hdlLTSelec,
theItem,ptInic,rgnSel,
primListaTemp)
DoUpdate
(nil)
DestroiListaTemp
(primListaTemp)
DoSelect
(theData,
primListaTemp)
MapeiaObjeto
(velhoRect,
novoRect,tempObjeto)
DoUpSelect
(primListaTemp)
DoUpdate
(nil)

Anexo III.36

ANEXO 4

LISTAGEM DAS PRINCIPAIS ROTINAS DO MED

Este anexo mostra a listagem do programa principal e as principais rotinas do programa que implementa o **MED**.

```
program MEG;
  uses
    MEGGlobais, MEGGlobaisII, MEGInicializa, MEGFile, MEGEdicao, MEGAlign, MEGArrange, MEGDoContent,
       MEGDoContentII, MEGRotinas, MEGUpdate, MEGEliminaElementos, MEGAttrib, MEGImpressao;
  procedure FixCursor;
  forward;
  procedure DoEvent;
  forward;
  procedure DoMouseDown;
  forward;
  procedure DoMenuClick;
  forward;
  procedure DoMenuChoice (menuChoice : LongInt);
  forward;
  procedure DoFileChoice (TheItem : Integer);  {Handle choice from File menu}
  forward;
  procedure DoAppleChoice (theItem : Integer);
  forward;
  procedure DoAbout;
  forward;
  procedure DoIconChoice (theItem : integer);
  forward;
  procedure DoDrag (whichWindow : WindowPtr);
  forward;                    {handle click in drag region}
  procedure DoGrow (whichWindow : WindowPtr);
  forward;                    {handle click in Grow region}
  procedure FixScrollBar;
  forward;
  procedure DoKeystroke;          {Handle keystroke}
  forward;
  procedure DoTyping (ch : CHAR);
  forward;
  procedure ProcessaCaracterDigitado (ch : Char);
  forward;
  procedure ExcluiObjetoCorrente (ch : Char);
  forward;

  procedure MainLoop;
{Rotina principal do programa. Inicializa o programa e transfere o controle para}
{a rotina DoEvent, a qual é a responsável de responder os eventos.}
  begin {MainLoop}
    FixCursor;
    SystemTask;

    if TheText <> nil then
      TEIdle(TheText);
    FixMenu;
    DoEvent
  end; {MainLoop}

{-------------------------------------------------------------------------------
```

```
      ----}

  procedure FixCursor;
{Ajusta o cursor dependendo do local onde este esteja.}
    label
      999;
    var
      mousePoint, pt : Point;              {Current mouse position in window coordinates}
      textRect : Rect;                     {Active window's text rectangle}
      wPeek : WindowPeek;
      dataHandle : Handle;
  begin
    if Quitting then
      goto 999;
    if FrontWindow = nil then
      InitCursor
    else  {Is one of our windows active?}
      begin
        GetMouse(mousePoint);                        {Get mouse positon}
        if TheText <> nil then
          begin
            textRect := TheText^^.viewRect;          {Get window's text rectangle}
            if PtInRect(mousePoint, textRect) then
              SetCursor(Ibeam^^)
            else
              InitCursor;
          end
        else
          InitCursor;
      end;{if}
999:
  end;{FixCursor}

{------------------------------------------------------------------------------
      ----}

  procedure DoEvent;
{Rotina encarregada de responder os eventos no MED. Cada tipo de evento é respondido por}
{uma rotina particular. A DoEvent encarrega-se de verificar qual o tipo de evento e ativar}
{a rotina apropriada}
    var
      wal : Integer;
  begin
    ErrorFlag := False;                          {Clear I/O error flag}
    if GetNextEvent(EveryEvent, TheEvent) then   {Get next event}
      case TheEvent.what of
        MouseDown :
          if not quitting then
            DoMouseDown;                 {Handle mouse-down event}
        KeyDown, AutoKey :
          if not quitting then
```

```
        DoKeystroke;                    {Handle keystroke}
      UpdateEvt :
        DoUpdate;                       {Handle update event}
      ActivateEvt :
        DoActivate; {Handle activate/deactiva event}
      otherwise {Do nothing}
    end{case}
  else if Quitting then
    begin
      DoClose;
      if Quitting then
        Finished := True;
    end;
  EventoAnterior := TheEvent;
end;{DoEvent}

{-------------------------------------------------------------------------------
    ----}

 procedure DoMouseDown;
{Rotina encarregada de responder os eventos do tipo "MouseDown". É verificado onde o botão}
{do mouse foi pressionado, ativando-se a rotina apropriada de executar os procedimentos}
{necessários.}
  var
    whichWindow : WindowPtr;        {Window that mouse was pressed in}
    thePart : Integer;              {Part of screen where mouse was pressed}
 begin
  thePart := FindWindow(TheEvent.where, whichWindow);    {Where on the screen was mouse pressed?}
  case thePart of
    InDesk :
      ;{Do nothing}
    InMenuBar :
      DoMenuClick; {Handle click in menu bar}
    InSysWindow :
      SystemClick(TheEvent, whichWindow); {Handle click in system window}
    InContent :
      DoContent(whichWindow); {Handle click in content region}
    InDrag :
      DoDrag(whichWindow);  {handle click in drag region}
    InGrow :
      DoGrow(whichWindow);  {handle click in size region}
    InGoAway :
      DoGoAway(whichWindow);   { Handle click in close region}
  end {case}
 end; {DoMouseDown}

{-------------------------------------------------------------------------------
    ----}

 procedure DoMenuClick;
{Rotina ativada quando o usuário aperta o botão do mouse sobre alguma opção do cardápio. }
```

```
{A rotina verifica qual o comando e opção do cardápio foi escolhida e ativa a rotina DoMenuChoice}
{para a execução dos procedimentos necessários}
   var
     menuChoice : LongInt;
  begin
    menuChoice := MenuSelect(TheEvent.where);
    DoMenuChoice(menuChoice)
  end;{DoMenuClick}

{-------------------------------------------------------------------------------
      ----}

  procedure DoMenuChoice;
{{menuChoice : LongInt)}
{Com base na variável "menuchoice" a rotina descobre qual opção do cardápio o usuário escolheu}
{e ativa a rotina adequada de realizar os procedimentos necessários}
   var
     theMenu : Integer;              {Menu Id of selected menu}
     theItem : Integer;              {item number of selected item}
  begin
    if menuChoice <> 0 then
      begin
        theMenu := HiWord(menuChoice);
        theItem := LoWord(menuChoice);

        case theMenu of
          AppleID :
            DoAppleChoice(theItem);
          FileID :
            DoFileChoice(theItem);
          EditID :
            DoEditChoice(theItem);
          IconID :
            DoIconChoice(theItem);
          FontID :
            DoFontChoice(theItem);
          StyleID :
            DoStyleChoice(theItem);
          SizeID :
            DoSizeChoice(theItem);
          JustID :
            DoJustChoice(theItem);
          AlignID :
            DoAlignChoice(theItem);
          ArrangeID :
            DoArrangeChoice(theItem);
          AttributeID :
            DoAttribChoice(theItem);

        end; {case}
        HiliteMenu(0);              {Unhighlight menu title}
```

```
    end {if}
  end;{DoMenuChoice}

{-------------------------------------------------------------------------------
    ----}
  procedure DoFileChoice;
{(theItem : Integer)}
{Rotina ativada quando o usuário escolhe algum comando da opção File do cardápio. A rotina}
{ativa o procedimento encarregado de responder ao usuário dependendo do comando escolhido}
  begin
    case theItem of
      NewItem :
        DoNew;
      OpenItem :
        DoOpen;
      CloseItem :
        DoClose;
      ShowItem :
        DoShowMetodo;
      SaveItem :
        DoSave;
      SaveAsItem :
        DoSaveAs;
      RevertItem :
        Dorevert;
      PrintItem :
        DoPrint;
      PrintSetUpItem :
        DoPrintSetUp;
      QuitItem :
        DoQuit;
    end; {case}
  end; {DoFileChoice}
{-------------------------------------------------------------------------------
    ----}
  procedure DoAppleChoice;
 {(theItem : Integer)}
{Rotina ativada quando o usuário escolhe algum comando do sistema Macintosh (Desk Acessory).}
    var
      accName : Str255;
      accNumber : Integer;
  begin
    case theItem of
      AboutItem :
        DoAbout;          {Handle About MiniEdit... command}
      otherwise
        begin
          if FrontWindow = nil then
            begin
              EnableItem(FileMenu, closeItem);
              EnableItem(EditMenu, UndoItem);
```

```
            EnableItem(EditMenu, CutItem);
            EnableItem(EditMenu, PasteItem);
            EnableItem(EditMenu, ClearItem);
          end;

        GetItem(AppleMenu, theItem, accName);        {Get accessory name}
        accNumber := OpenDeskAcc(accName);           {Open desk accessory}
      end
    end {case}
  end; {DoAppleChoice}

  procedure DoIconChoice; {(theItem : Integer)}
{Rotina ativada para responder a um comando da opção Icon do cardápio.}
    var
      wpeek : WindowPeek;
  begin
    if EmtSelecionado <> nil then
      case theItem of
        IconOpenItem :
          ManipulaDuploClick;
        IconCloseItem :
          with EmtSelecionado^ do
            if tipoElemento in [ponta, nodo, decoracao] then
              if pontJanela <> nil then
                begin
                  wpeek := WindowPeek(pontJanela);
                  if wpeek^.visible then
                    HideWindow(pontJanela)
                end
      end
  end; {DoIconChoice}

{-------------------------------------------------------------------------------
      ----}

  procedure DoAbout;
    var
      ignore : integer;
  begin
    ignore := Alert(AboutID, nil)
  end; {DoAbout}

  procedure DoDrag;
{(whichWindow : WindowPtr)}
{Rotina ativa para responder uma solicitação de arrosto de janela}
    var
      limitRect : Rect;                              {Limit rectangle for dragging}
  begin
    {Set limit rectangle}
    SetRect(limitRect, 0, MenuBarHeight, ScreenWidth, ScreenHeight);
```

```
    {Inset by screen margin}
    InsetRect(limitRect, ScreenMargin, ScreenMargin);

    DragWindow(whichWindow, TheEvent.where, limitRect);  {let user drag the window}
  end; {DoDrag}

{-------------------------------------------------------------------------------
      ----}

  procedure DoGrow;
{(whichWindow : WindowPtr)}
{Rotina encarregada de responder as solicitações de alteração de dimensões de janelas.}
{Para mudar as dimensões da janela de catálogo é chamada a rotina GrowMetodoWindow, caso}
{contrário é chamada a rotina GrowOtherWindow para outras janelas}
  begin
    if whichWindow <> FrontWindow then
      SelectWindow(whichWindow)
    else if whichWindow = MetodoWindow then
      GrowMetodoWindow
    else
      GrowOtherWindow(whichWindow)
  end; {DoGrow}

  procedure GrowMetodoWindow;
{Rotina ativada para responder a solicitação de alteração de dimensões }
{da janela de cátalogo do método.}
{Caso a janela muda suas dimensões, os novos valores das bras de rolagens são atualizados,}
{como também as regiões de recorte e desenho.}
    var
      rgnTemp : RgnHandle;
      rgnTempPalet : RgnHandle;
      sizeRect : Rect;
      newSize : LongInt;
      newWidth : Integer;
      newHeight : Integer;
  begin
    SetRect(sizeRect, MinWidth, MinHeight, ScreenWidth, (ScreenHeight - MenuBarHeight));
    newSize := GrowWindow(MetodoWindow, TheEvent.where, sizeRect);

    if newSize <> 0 then
      begin
        rgnTemp := NewRgn;
        rgnTempPalet := NewRgn;
        with MetodoWindow^.portRect do
          begin
            SetRectRgn(rgnTemp, left, top, right, bottom);
            DiffRgn(rgnTemp, VerticalPalette, rgnTemp);
            DiffRgn(rgnTemp, MetodoDesenho, rgnTemp);
            EraseRgn(rgnTemp);
          end;
        newWidth := LoWord(newSize);                    {Extract width from low word}
```

```
      newHeight := HiWord(newSize);                    {Extract height from high word}
      SizeWindow(MetodoWindow, newWidth, newHeight, TRUE); {Adjust size of window}

      with MetodoWindow^.portRect do
        begin
          SetRectRgn(MetodoDesenho, left, top, right - (SBarWidth - 1), bottom - (SBarWidth - 1));
          SetRectRgn(rgnTemp, left, top, right, bottom);
          MetodoPaletteY2 := bottom;
          SetRectRgn(VerticalPalette, MetodoPaletteX1, MetodoPaletteY1, MetodoPaletteX2, MetodoPaletteY2);
        end;

      with MetodoIconRect[LixoItem] do
        begin
          if top < MetodoPaletteY2 - 52 then
            SetRectRgn(rgnTempPalet, MetodoPaletteX1, MetodoPaletteY1, MetodoPaletteX2, top)
          else
            SetRectRgn(rgnTempPalet, MetodoPaletteX1, MetodoPaletteY1, MetodoPaletteX2, MetodoPaletteY2 - 52);
          SetRect(MetodoIconRect[LixoItem], left, MetodoPaletteY2 - 52, right, MetodoPaletteY2 - 20);
        end;

      DiffRgn(MetodoDesenho, VerticalPalette, MetodoDesenho);
      DiffRgn(rgnTemp, rgnTempPalet, rgnTemp);
      InvalRgn(rgnTemp);                    {force update of window's contents}
      FixScrollBar;                         {Resize scroll bar}
      DisposeRgn(rgnTemp);
    end
  end;


  procedure GrowOtherWindow (wPtr : WindowPtr);
{Rotina encarregada de responder as solicitações de alteração de dimensões das janelas}
{de desenho.}
    var
      dataHandle : Handle;
      wPeek : WindowPeek;
      rgnTemp : RgnHandle;
      sizeRect, tempRect : Rect;
      newSize : LongInt;
      newWidth, newHeight : Integer;
      i, newRight, newBottom : integer;
  begin
    SetRect(sizeRect, TelaIIXMin, TelaIIYMin, ScreenWidth, (ScreenHeight - MenuBarHeight));
    newSize := GrowWindow(wPtr, TheEvent.where, sizeRect);

    if newSize <> 0 then
      begin
        wPeek := WindowPeek(wPtr);
        dataHandle := Handle(GetWRefCon(Wptr));    {Get window data}
        HLock(dataHandle);    {Lock data record}
        TheData := WDHandle(dataHandle);
```

```
      rgnTemp := NewRgn;
      with wPtr^.portRect, TheData^^ do
        begin
          SetRectRgn(rgnTemp, left, top, right, bottom);
          for i := 1 to NumRgn do
            DiffRgn(rgnTemp, regioes[i], rgnTemp);
          EraseRgn(rgnTemp);
        end;
      newWidth := LoWord(newSize);
      newHeight := HiWord(newSize);
      SizeWindow(wPtr, newWidth, newHeight, TRUE);

      with TheData^^.regioes[2]^^.rgnBBox do
        SetRect(tempRect, left, top, wPtr^.portRect.right, Bottom);
      RectRgn(TheData^^.regioes[2], tempRect);

      with TheData^^.regioes[3]^^.rgnBBox do
        SetRect(tempRect, left, top, right, wPtr^.portRect.bottom);
      RectRgn(TheData^^.regioes[3], tempRect);

      newRight := wPtr^.portRect.right - SbarWidth + 1;
      newBottom := wPtr^.portRect.bottom - SbarWidth + 1;
      with TheData^^.regioes[4]^^.rgnBBox do
        SetRect(tempRect, left, top, newRight, newBottom);
      RectRgn(TheData^^.regioes[4], tempRect);
      HUnlock(dataHandle);

      InvalRect(wPtr^.portRect);
      DisposeRgn(rgnTemp);
      FixScrollBar
    end;
  end;

  procedure FixScrollBar;
{Rotina encarregada de alterar os valores das barras de rolagens}
  begin
    HideControl(VerScrollBar);                    {Hide Vertical scroll bar}
    HideControl(HorScrollBar);                    {Hide Horizontal scroll bar}

    if TheWindow = MetodoWindow then
      with TheWindow^.portRect do
        begin
          MoveControl(VerScrollBar, right - (SbarWidth - 1), top - 1); {move top-left corner}
          SizeControl(VerScrollBar, SBarWidth, (bottom + 1) - (top - 1) - (SBarWidth - 1));
          MoveControl(HorSCrollBar, MetodoPaletteX2 - 1, bottom - (SBarWidth - 1));
          SizeControl(HorScrollBar, (right + 1) - (MetodoPaletteX2 - 1) - (SBarWidth - 1), SBarWidth);
        end{with}
    else
      with TheWindow^.portRect, TheData^^ do
        begin
          MoveControl(VerScrollBar, right - SbarWidth + 1, regioes[3]^^.rgnBBox.top - 1); {move top-left corner}
```

```
        SizeControl(VerScrollBar, SBarWidth, (bottom + 1) - (regioes[3]^^.rgnBBox.top - 1) - (SBarWidth - 1));
        MoveControl(HorSCrollBar, regioes[3]^^.rgnBBox.right - 1, bottom - SBarWidth + 1);
        SizeControl(HorScrollBar, (right + 1) - (regioes[3]^^.rgnBBox.right - 1) - SBarWidth + 1, SBarWidth);

      end;

    ShowControl(VerScrollBar);                {Redisplay vertical scroll bar}
    ShowControl(HorScrollBar);                {Redisplay horizontal scroll bar}

    ValidRect(VerScrollBar^^.contrlRect);     {Avoid updating again}
    ValidRect(HorScrollBar^^.contrlRect);

  end; {FixScrollBar}


  procedure DoKeystroke;
{Rotina encarregada de responder as mensagens de teclhado. A rotina verifica se o tecla}
{presionda é um acionador de comandos do cardápio. Caso não seja, é verificado}
{em qual tipo de janela foi digitado o caracter, chamando-se a rotina apropriada de tratar}
{a edição}
    var
      chCode : Integer;                       {Character code from event message}
      ch : Char;                              {Character that was typed}
      menuChoice : Longint;                   {Menu ID and item number for keyboard alias}
  begin
    with TheEvent do
      begin
        chCode := BitAnd(message, CharCodeMask);  {Extract character code}
        ch := CHR(chCode);                        {Convert to a character}

        if BitAnd(modifiers, CmdKey) <> 0 then    {Command key down?}
          begin
            if what <> AutoKey then               {Ignore repeats}
              begin
                menuChoice := MenuKey(ch);        {Get menu equivalent}
                DoMenuChoice(menuChoice);         {Handle as menu choice}
              end
          end
        else if TheWindow <> nil then
          if TheWindow = MetodoWindow then
            DoTyping(ch)                              {Handle as normal character}
          else
            ProcessaCaracterDigitado(ch)
      end {with}
  end; {DoKeyStroke}

  procedure DoTyping;
{(ch : CHAR)}
{Esta rotina trata as edições realizadas na janela de catálogo do método.}
    const
      backSpace = 8;
```

```
    cr = 13;
  var
    saveRgn : RgnHandle;
    titulo : Str255;
    dataHandle : Handle;
begin
  saveRgn := NewRgn;
  GetClip(saveRgn);
  if TheText <> nil then
    if ((TheText^^.teLength < MaxCaractTexto) or (ord(ch) = backSpace)) and (ord(ch) <> cr) then
      with EmtSelecionado^ do
        begin
          TEKey(ch, TheText);                          {Process character}
          SetClip(MetodoDesenho);
          TEUpdate(rectID, id);
          CalcRectTextoIcon(rectIcon, rectID, TheText);
          TEUpdate(rectID, id);
          ConverteTextoStr(ID, titulo);
          case tipoElemento of
            nodo, decoracao, ponta :
              begin
                if pontJanela <> nil then
                  begin
                    dataHandle := Handle(GetWRefCon(pontJanela));
                    TheData := WDHandle(dataHandle);
                    SetWTitle(pontJanela, titulo);
                  end;
                case tipoElemento of
                  nodo :
                    begin
                      HLock(handle(NodoHdl));
                      NodoHdl^^.identificador := titulo;
                      HUnlock(handle(NodoHdl));
                    end;
                  decoracao :
                    begin
                      HLock(handle(DecoracaoHdl));
                      DecoracaoHdl^^.identificador := titulo;
                      HUnlock(handle(DecoracaoHdl));
                    end;
                  ponta :
                    begin
                      HLock(handle(PontaHdl));
                      PontaHdl^^.identificador := titulo;
                      HUnlock(handle(PontaHdl));
                    end;
                end;
              end;
            conector :
              begin
                HLock(handle(PontConector));
```

```
            PontConector^^.identificador := titulo;
            HUnlock(handle(PontConector));
          end;
        end;
      end;
  DisableItem(EditMenu, CutItem);              {Disable menu item that operate on}
  DisableItem(EditMenu, CopyItem);             {  a nonempty selecton}
  DisableItem(EditMenu, ClearItem);
  WindowDirty(true);
  SetClip(saveRgn);
  DisposeRgn(saveRgn);
end;{DoTyping}

procedure ProcessaCaracterDigitado;
{(ch : CHAR)}
{Esta rotina trata as edições de texto que são realizadas nas janelas de desenho.}
  const
    cr = 13;
    margem = 4;
    backSpace = 8;
  label
    999;
  var
    tempRect : Rect;
    infoFonte : FontInfo;
    tempBool : Boolean;
    incr : Integer;
    retBackSpace : Rect;
begin
  if (TheText = nil) then
    begin
      ExcluiObjetoCorrente(ch);
      goto 999;
    end;
  GetFontInfo(infoFonte);
  with TheData^^ do
    with contexto do
      begin
        HLock(Handle(TheData));
        HLock(Handle(HdlObjNodoCorr));
        HLock(Handle(TheText));
        tempRect := regioes[TelaIIRegiaoDesenho]^^.rgnBBox;
        coloca_remove_pegadores(HdlObjNodoCorr);
        with TheText^^, infoFonte do
          begin
            TEKey(ch, TheText);
            TEDeactivate(TheText);
            destRect.bottom := nLines * (ascent + descent + leading) + destRect.top;
            if destRect.bottom = destRect.top then
              destRect.bottom := ascent + Descent + leading + destRect.top;
            destRect.right := destRect.left + CalculaMaiorLinha(TheText) + widMax;
```

```
          tempBool := SectRect(tempRect, destRect, viewRect);
          retBackSpace := HdlObjNodoCorr^^.envelope;
          HdlObjNodoCorr^^.envelope := destRect;
          if ord(ch) = cr then
           begin
            tempRect := destRect;
            InsetRect(tempRect, -2, -2);
            InvalRect(tempRect);
            if GetNextEvent(UpdateMask, TheEvent) then
             DoUpdate;
           end
          else
           begin
            if (ord(ch) = backSpace) then
             begin
              retBackSpace.left := TheText^^.destRect.right;
              InvalRect(retBackSpace);
             end;
            TEUpdate(TheText^^.destRect, TheText);
            TEActivate(TheText);
            coloca_remove_pegadores(HdlObjNodoCorr);
           end;
         end;
        DisableItem(EditMenu, CopyItem);
        DisableItem(EditMenu, ClearItem);
        DisableItem(EditMenu, CutItem);
        HUnLock(Handle(TheText));
        HUnLock(Handle(HdlObjNodoCorr));
        HUnLock(Handle(TheData));
       end;
999:
 end;{ProcessaCaracterDigitado}

 procedure ExcluiObjetoCorrente; {(ch : Char)}
{Esta rotina é chamada para realizar exclusões de objetos.}
   var
    updateRgn1, updateRgn2 : RgnHandle;
 begin
   if ch = FlagDelecao_GLB then
    with TheData^^, TheData^^.contexto do
     if HdlObjNodoCorr <> nil then
      begin
       updateRgn1 := newRgn;
       updateRgn2 := newRgn;
       RectRgn(updateRgn1, HdlObjNodoCorr^^.envelope);
       with EmtSelecionado^ do
        case tipoElemento of
         nodo:
          Procura_Retira_objeto(NodoHdl^^.primObjNodo, HdlObjNodoCorr);
         decoracao:
          Procura_Retira_objeto(DecoracaoHdl^^.primObjDecoracao, HdlObjNodoCorr);
```

```
            ponta^
              Procura_Retira_objeto(PontaHdl^^.primObjPonta, HdlObjNodoCorr);
          end;
        if HdlObjNodoCorr <> nil then
          RectRgn(updateRgn2, HdlObjNodoCorr^^.envelope);
        UnionRgn(updateRgn1, updateRgn2, updateRgn2);
        InsetRgn(updateRgn2, -20, -20);
        InvalRgn(updateRgn2);
      end;
  end;{ExcluiObjetoCorrente}

begin
  Inicializa;
  UnLoadSeg(@Inicializa);
  repeat
    MainLoop
  until Finished;

  if TEGetScrapLen > 0 then
    begin
      ScrapDirty := true;
      WriteDeskScrap
    end;
end.
```

ANEXO 5

LISTAGEM DAS PRINCIPAIS ROTINAS DO EDE

Este anexo mostra a listagem do programa principal e as principais rotinas do programa que implementa o **EDE**.

```
program EGE;
 uses
  EGEGlobais, EGEConecs, EGEFile, EGEEdicao, EGEDoContent, EGEUpdate, EGEMainRot, EGETexto, EGEApagaSelecao,
     EGERotinas, EGERefina, EGEAlign;
 procedure FixCursor;
 forward;
 procedure DoEvent;
 forward;
 procedure DoMouseDown;
 forward;
 procedure DoMenuClick;
 forward;
 procedure DoMenuChoice (menuChoice : LongInt);
 forward;
 procedure DoAppleChoice (theItem : Integer);
 forward.
 procedure DoAbout;
 forward;
 procedure DoDrag (whichWindow : WindowPtr);
 forward;                {handle click in drag region}
 procedure DoGrow (whichWindow : WindowPtr);
 forward;                {handle click in Grow region}
 procedure GrowDiagWindow (wPtr : WindowPtr);
 forward;                {handle click in diagrams windows}
 procedure FixScrollBar (wptr : WindowPtr);
 forward;
 procedure DoKeystroke;          {Handle keystroke}
 forward;
 procedure DoTyping (ch : CHAR);
 forward;

 procedure MainLoop;
 begin {MainLoop}
  FixCursor;
  SystemTask;

  if TheText <> nil then
   TEIdle(TheText);
  FixMenu;
  if MenuBool then
   begin
    MenuBool := false;
    DrawMenuBar
   end;
  DoEvent
 end; {MainLoop}

 procedure FixCursor;
{Ajust cursor for region of screen}
  label
   999;
```

```
  var
    mousePoint, pt : Point;              {Current mouse position in window coordinates}
    textRect : Rect;                     {Active window's text rectangle}
    wPeek : WindowPeek;
    dataHandle : Handle;
begin
  if Quitting then
    goto 999;
  if FrontWindow = nil then
    InitCursor
  else  {Is one of our windows active?}
    begin
      GetMouse(mousePoint);                          {Get mouse positon}
      if TheText <> nil then
        begin
          textRect := TheText^^.viewRect;            {Get window's text rectangle}
          if PtInRect(mousePoint, textRect) then
            SetCursor(Ibeam^^)
          else
            InitCursor;
        end
      else
        InitCursor;
    end; {if}
999:
end; {FixCursor}

procedure DoEvent;
  var
    tempoAnterior : Integer;
begin
  ErrorFlag := False;                            {Clear I/O error flag}
  if GetNextEvent(EveryEvent, TheEvent) then     {Get next event}
    case TheEvent.what of
      MouseDown :
        if not quitting then
          DoMouseDown;                 {Handle mouse-down event}
      MUpMask :
        if not quitting then
          tempoAnterior := TheEvent.When;
      KeyDown, AutoKey :
        if not quitting then
          DoKeystroke;                           {Handle keystroke}
      UpdateEvt :
        DoUpdate;                                {Handle update event}
      ActivateEvt :
        DoActivate; {Handle activate/deactive event}
      otherwise {Do nothing}
    end{case}
  else if Quitting then
    begin
```

```
      DoClose;
      if Quitting then
        Finished := True;
    end;
  EventoAnterior := TheEvent;
end;{DoEvent}

procedure DoMouseDown;
  var
    whichWindow : WindowPtr;          {Window that mouse was pressed in}
    thePart : Integer;                {Part of screen where mouse was pressed}
begin
  thePart := FindWindow(TheEvent.where, whichWindow);      {Where on the screen was mouse pressed?}
  case thePart of
    InDesk :
      {Do nothing}
    InMenuBar :
      DoMenuClick; {Handle click in menu bar}
    InSysWindow :
      SystemClick(TheEvent, whichWindow); {Handle click in system window}
    InContent :
      begin
        WindowDirty(true);
        UnloadSeg(@WindowDirty);
        DoContent(whichWindow); {Handle click in content region}
        UnloadSeg(@DoContent);
      end;
    InDrag :
      DoDrag(whichWindow);  {handle click in drag region}
    InGrow :
      DoGrow(whichWindow);  {handle click in size region}
    InGoAway :
      begin { Handle click in close region}
        if TheWindow = PrimDiagrama^^.pontJanela then
          begin
            DoClose;
            UnloadSeg(@DoClose);
          end
        else
          DoGoAway(whichWindow);{if TheWindow = PrimDiagrama^^.pontJanela}
      end;
  end {case}
end; {DoMouseDown}

procedure DoMenuClick;
  var
    menuChoice : LongInt;
begin
  menuChoice := MenuSelect(TheEvent.where);
  DoMenuChoice(menuChoice)
end;{DoMenuClick}
```

```
procedure DoMenuChoice;
{(menuChoice : LongInt)}
  var
    theMenu : Integer;          {Menu Id of selected menu}
    theItem : Integer;          {Item number of selected item}
begin
  if menuChoice <> 0 then
    begin
      theMenu := HiWord(menuChoice);
      theItem := LoWord(menuChoice);

      case theMenu of
        AppleID :
          DoAppleChoice(theItem);
        FileID :
          begin
            DoFileChoice(theItem);
            UnLoadSeg(@DoFileChoice);
          end;
        EditID :
          begin
            DoEditChoice(theItem);
            UnLoadSeg(@DoEditChoice);
          end;
        FontID :
          begin
            DoFontChoice(theItem);
            UnLoadSeg(@DoFontChoice);
          end;
        StyleID :
          begin
            DoStyleChoice(theItem);
            UnLoadSeg(@DoStyleChoice);
          end;
        SizeID :
          begin
            DoSizeChoice(theItem);
            UnLoadSeg(@DoSizeChoice);
          end;
        JustID :
          DoJustChoice(theItem);
        AlignID :
          begin
            DoAlignChoice(theItem);
            UnLoadSeg(@DoAlignChoice);
          end;
        StructurID :
          begin
            DoStructureChoice(theItem);
            UnLoadSeg(@DoStructureChoice);
```

```
        end;
      TextID :
        begin
          DoTextoChoice(theItem);
          UnLoadSeg(@DoTextoChoice);
        end;
    end; {case}
    HiliteMenu(0);                {Unhighlight menu title}
  end {if}
end;{DoMenuChoice}

procedure DoAppleChoice;
{(theItem : Integer)}
  var
    accName : Str255;
    accNumber : Integer;
begin
  case theItem of
    AboutItem :
      DoAbout;          {Handle About MiniEdit... command}
    otherwise
      begin
        if FrontWindow = nil then
          begin
            EnableItem(FileMenu, closeItem);
            EnableItem(EditMenu, UndoItem);
            EnableItem(EditMenu, CutItem);
            EnableItem(EditMenu, PasteItem);
            EnableItem(EditMenu, ClearItem);
          end;

        GetItem(AppleMenu, theItem, accName);        {Get accessory name}
        accNumber := OpenDeskAcc(accName);           {Open desk accessory}
      end
  end {case}
end; {DoAppleChoice}

procedure DoAbout;
  var
    ignore : integer;
begin
  ignore := Alert(AboutID, nil)
end; {DoAbout}

procedure DoDrag;
{(whichWindow : WindowPtr)}
  var
    limitRect : Rect;                                {Limit rectangle for dragging}
begin
  {Set limit rectangle}
  SetRect(limitRect, 0, MenuBarHeight, ScreenWidth, ScreenHeight);
```

```
    {Inset by screen margin}
    InsetRect(LimitRect, ScreenMargin, ScreenMargin);

    DragWindow(whichWindow, TheEvent.where, limitRect);  {let user drag the window}
  end; {DoDrag}

  procedure DoGrow;
{(whichWindow : WindowPtr)}
  begin
    if whichWindow <> FrontWindow then
      SelectWindow(whichWindow)
    else if (whichWindow <> NodoWindow) and (whichWindow <> DecorWindow) and (whichWindow <> ArcoWindow)
        then
      GrowDiagWindow(whichWindow)
  end; {DoGrow}

  procedure GrowDiagWindow; {(wPtr : WindowPtr)}
    var
      dataHandle : Handle;
      wPeek : WindowPeek;
      rgnTemp : RgnHandle;
      sizeRect, tempRect : Rect;
      newSize : LongInt;
      newWidth, newHeight : Integer;
      i, newRight, newBottom : integer;
      theData : WDHandle;
  begin
    SetRect(sizeRect, TelaIIXMin, TelaIIYMin, ScreenWidth, (ScreenHeight - MenuBarHeight));
    newSize := GrowWindow(wPtr, TheEvent.where, sizeRect);

    if newSize <> 0 then
      begin
        wPeek := WindowPeek(wPtr);
        dataHandle := Handle(GetWRefCon(Wptr));    {Get window data}
        HLock(dataHandle);    {Lock data record}
        TheData := WDHandle(dataHandle);

        rgnTemp := NewRgn;
        with wPtr^.portRect, TheData^^ do
          begin
            SetRectRgn(rgnTemp, left, top, right, bottom);
            for i := 1 to NumRgn do
              DiffRgn(rgnTemp, regions[i], rgnTemp);
            EraseRgn(rgnTemp);
          end;
        newWidth := LoWord(newSize);
        newHeight := HiWord(newSize);
        SizeWindow(wPtr, newWidth, newHeight, TRUE);

        with TheData^^.regions[2]^^.rgnBBox do
```

```
      SetRect(tempRect, left, top, wPtr^.portRect.right, Bottom);
    RectRgn(TheData^^.regioes[2], tempRect);

    newRight := wPtr^.portRect.right - SbarWidth + 1;
    newBottom := wPtr^.portRect.bottom - SbarWidth + 1;
    with TheData^^.regioes[3]^^.rgnBBox do
      SetRect(tempRect, left, top, newRight, newBottom);
    RectRgn(TheData^^.regioes[3], tempRect);
    HUnlock(dataHandle);

    InvalRect(wPtr^.portRect);
    DisposeRgn(rgnTemp);
    FixScrollBar(wptr)
  end;
end;

procedure FixScrollBar;{(wptr : WindowPtr)}
  var
    theData : WDHandle;
    dataHandle : Handle;
begin
  dataHandle := Handle(GetWRefCon(wptr));
  theData := WDHandle(dataHandle);
  HideControl(VerScrollBar);                    {Hide Vertical scroll bar}
  HideControl(HorScrollBar);                    {Hide Horizontal scroll bar}

  with theData^^.regioes[TelaIIRegiaoDesenho]^^, wptr^.portRect do
    begin
      MoveControl(VerScrollBar, right - SbarWidth + 1, rgnBBox.top - 1); {move top-left corner}
      SizeControl(VerScrollBar, SBarWidth, (bottom + 1) - (rgnBBox.top - 1) - (SBarWidth - 1)),
      MoveControl(HorScrollBar, left - 1, bottom - SBarWidth + 1);
      SizeControl(HorScrollBar, (right + 1) - (left - 1) - SBarWidth + 1, SBarWidth);
    end;

  ShowControl(VerScrollBar);                    {Redisplay vertical scroll bar}
  ShowControl(HorScrollBar);                    {Redisplay horizontal scroll bar}

  ValidRect(VerScrollBar^^.contrlRect);         {Avoid updating again}
  ValidRect(HorScrollBar^^.contrlRect);

end; {FixScrollBar}

procedure DoKeystroke;
  var
    chCode : Integer;                           {Character code from event message}
    ch : Char;                                  {Character that was typed}
    menuChoice : Longint;                       {Menu ID and item number for keyboard alias}
begin
  with TheEvent do
    begin
      chCode := BitAnd(message, CharCodeMask);  {Extract character code}
```

```
      ch := CHR(chCode);                    {Convert to a character}

      if BitAnd(modifiers, CmdKey) <> 0 then    {Command key down?}
        begin
          if what <> AutoKey then           {Ignore repeats}
            begin
              menuChoice := MenuKey(ch);    {Get menu equivalent}
              DoMenuChoice(menuChoice);     {Handle as menu choice}
            end
        end
      else if TheWindow <> nil then
        DoTyping(ch)                        {Handle as normal character}
    end {with}
  end; {DoKeyStroke}

  procedure DoTyping;
{(ch : CHAR)}
    const
      fim = '.';
      espaco = ' ';
  begin
    if (TheText = nil) then
      case ch of
        fim :
          ApagaIconCorrente;
        espaco :
          begin
            ApagaSelecionados;
            UnloadSeg(@ApagaSelecionados);
          end;
        otherwise
      end
    else
      begin
        DoTypingTexto(ch);
        UnloadSeg(@DoTypingTexto);
      end;
  end;{DoTyping}

begin
  Inicializa;
  UnLoadSeg(@Inicializa);
  repeat
    MainLoop
  until Finished;

  if TEGetScrapLen > 0 then
    begin
      ScrapDirty := true;
      WriteDeskScrap
    end;
```

```
end.
```

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Pós-Graduação em Ciência da Computação

Proposta de um editor diagramático generalizado

Dissertação apresentada aos Srs.

Carlos Alberto Heuser

Raul Fernando Weber

Carla Maria Dal Sasso Freitas

Roberto Tom Price

Visto e permitida a impressão
Porto Alegre, 16/06/89.

Roberto Tom Price
Orientador

Profa. Ingrid E. S. Jansch Pôrto
Coordenadora do Curso de Pós-Graduação
em Ciência da Computação