Instituto de Ciências Matemáticas de São Carlos

ISSN-0109-2569

EdGraf - Editor gráfico para o ASIA

Descrição da Implementação

RENATA SPOLON
ROBERTA SPOLON
REGINA H. C. SANTANA
MARCOS JOSÉ SANTANA

Nº 20

RELATÓRIOS TÉCNICOS DO ICMSC

SÃO CARLOS
jul. / 1994

SYSNO 0868596
DATA / /
ICMC - SBAB

EdGraf - Editor Gráfico para o ASiA

Descrição da Implementação

Renata Spolon

Roberta Spolon

Regina Helena Carlucci Santana

Marcos José Santana

Grupo de Programação Concorrente e Sistemas Distribuídos
Departamento de Computação e Estatística
Instituto de Ciências Matemáticas de São Carlos
Universidade de São Paulo

São Carlos
julho de 1994

Agradecimentos à Fapesp pelo apoio ao desenvolvimento deste trabalho.

Sumário

1. Introdução 01
2. Interface entre Editor Gráfico e Gerador de Aplicação 02
2.1 Dados Referentes Às Principais Características do Modelo 02
2.2 Dados Referents Aos Recursos Definidos no Modelo 05
2.3 Dados Referentes Aos Ciclos Definidos no Modelo 10
2.4 Dados Referentes Às Distribuições Tabeladas 11
2.5 Dados Refentes Aos Eventos Definidos no Modelo 11
3. Considerações Sobre a Implementação 13
4. Arquivos Produzidos Pelo EdGraf 16
4.1 Arquivo de Especificação Gráfica 16
4.2 Arquivo de Ligações e Ramificações 17
5. Descrição dos Módulos e Principais Rotinas Implementadas 18
5.1 Módulo EDITOR.C 18
5.2 Módulo EDITARQ.C 22
5.3 Módulo LISTA. 37
5.4 Módulo LISTREC.C 45
5.5 Modulo LISTALIG.C 51
5.6 Módulo LISTDIST.C 62
5.7 Módulo LISTEVEN.C 64
5.8 Módulo DESENHA.C 66
5.9 Módulo INFOREC.C 68
5.10 Módulo CONSIST.C 69
5.11 Módulo AUXILIAR.C 76
5.12 Módulo LISTALOOP.C 79
5.13 Arquivos de Inclusão 82
5.13.1 MENSERRO.H 82
5.13.2 TIPOS.H 82

5.13.3 EDDEF.H 82
5.13.4. EDITOR.H 82
5.14 Módulo EDITOR.RC 82
5.15 Arquivo de Definições: EDITOR.DEF 83
6. Referências Bibliográficas 84
7. Bibliografia 85

1. Introdução

O EdGraf (Editor Gráfico para o ASiA) constitui o módulo de interface do ASiA (Ambiente de Simulação Automático) em desenvolvimento no Grupo de Sistemas Distribuídos e Programação Concorrente do Departamento de Computação e Estatística do ICMSC-USP [SPO93, SPO93a]. Foi desenvolvido utilizando-se a linguagem de programação C, com o compilador Borland C++ 3.1 e o Microsoft Windows 3.1, em microcomputador IBM-PC 386.

No ASiA, o Editor Gráfico é responsável pela interface entre o usuário e o ambiente, e a geração de uma estrutura de dados (uma tabela de especificação). Essa contém os dados necessários para que o Gerador de Aplicação construa o programa de simulação correspondente ao modelo definido pelo usuário [SPO93a]. A seção 2 discute como foi definida a interface entre o Editor Gráfico e o Gerador de Aplicação.

O Editor permite a definição de modelos de filas, utilizando centros de serviço com uma (ou várias) fila(s) e um (ou vários) servidor(es) para representar os recursos do modelo sendo simulado. A seção 3 discute alguns pontos importantes na implementação do EdGraf, e a seção 4 apresenta os arquivos que guardam a especificação gráfica de um modelo definido pelo usuário.

A seção 5 apresenta os módulos e as principais rotinas implementadas, com descrição dos parâmetros de entrada e valores de retorno para cada uma.

Ressalta-se que esta descrição refere-se ao projeto do Editor Gráfico, e por isso não entra em detalhes referentes à programação com o Windows 3.1.

2. Interface entre Editor Gráfico e Gerador de Aplicação

Como o trabalho do Editor consiste em captar os dados que serão utilizados pelo Gerador de Aplicação para produzir o programa de simulação, tornou-se necessário definir como armazenar esses dados e passá-los ao Gerador. Definiu-se que os dados coletados pelo Editor Gráfico sejam armazenados em arquivos, em estruturas de dados que facilitam a utilização pelo Gerador. São cinco estruturas, cada uma referente a um arquivo de dados. Tais estruturas são descritas a seguir.

2.1 Dados Referentes Às Principais Características do Modelo

Algumas informações referem-se ao modelo como um todo, como por exemplo seu nome. Esse grupo de informações é armazenado em arquivo com a extensão "default" ".mod", que tem a estrutura a seguir:

```
typedef struct especifi{
        char nome_modelo[30],
        parametro_01[2],
        tempo[10];
        unsigned int num_max_entidades,
        num_voltas;
        int trace;
        char relatorio_final,
        nome_arqsaida[13],
        tempo_prim_chegada[10];
        unsigned int valores_deletados,
        tamanho_batch,
        distrib_tab,
        reesc;
        double warm_up;
} Especificacao;
```

.nome_modelo: corresponde ao nome do modelo (necessário para a implementação em SMPL);

.parâmetro_01: indica se o usuário deseja ou não ativar a interface mtr do SMPL [MAC87];

.tempo: limita a simulação por tempo de execução. Deve ser "0.0" se a simulação nao for limitada por tempo;

.num_max_entidades: limita a simulação por número de clientes. Deve ser 0 se não for o limitante da simulação;

.num_voltas: limita a simulação pelo número de ciclos ("loop") para o caso de um sistema fechado. Deve ser 0 se o modelo não especificar um sistema fechado;

.trace: ativa o "trace" do SMPL, podendo ter os seguintes valores [MAC87]:

.0: "trace" desativado;

.1: trace "free_running", as mensagens são mostradas continuamente na tela do monitor ou na impressora, dependendo de qual seja o dispositivo padrão de saída;

.2: mensagens são enviadas para a tela, ocorrendo pausa cada vez que a tela é totalmente preenchida;

.3: mensagens são enviadas para a tela, porém a pausa ocorre após a exibição de cada mensagem;

.4: atualiza contadores de linha e página e determina mudança de página ou pausa na tela, se necessário;

.relatorio_final: determina se o usuário deseja ou não obter o relatório padrão do SMPL;

.nome_arqsaida: nome do arquivo de simulação a ser gerado (no caso, em SMPL);

.tempo_prim_chegada: indica o tempo que que irá ocorrer a chegada do primeiro cliente ao sistema;

.valores_deletados: quantidade de valores a serem desprezados antes de iniciar a coleta de dados, no modelo de simulação;

.tamanho_batch: tamanho do lote, quando "Batch Means" [MAC87] for utilizado. Deve ser 0 se não for utilizado;

.distrib_tab: indica o número de distribuições tabeladas a serem utilizadas no modelo. O EdGraf considera 5 distribuições pré- definidas, e o usuário deve informar quantas distribuições tabeladas vai usar, para que o Editor Gráfico possa fazer a consistência quando aceitar os parâmetros relacionados à distribuição do tempo entre_chegadas e do tempo de serviço. A seção 2.2 apresenta as distribuicões possíveis;

.reesc: define se existem recursos sem fila no modelo (para utilização de tabelas para reescalonamento de eventos). Deve ser especificado pois a tabela é gerada em um arquivo separado a ser incluído no programa final em SMPL. Deve ser 0/1. Esse campo da estrutura não está sendo captado pelo EdGraf, que coloca sempre o valor 0. É mantido na estrutra para consistência com o Gerador de Aplicação, e futuras expansões do EdGraf;

.warm_up: caso o usuário não queira utilizar "Batch Means", mas mesmo assim queira identificar um tempo na simulação para zerar todos os contadores e medidas obtidas no SMPL, deve usar esse campo. O tempo especificado passa a ser o ínicio do intervalo de medida, e as medidas relatadas representam a atividade do modelo de simulação somente daquele ponto no tempo, até o ponto que o relatório das medidas obtidas for produzido. Pode ser utilizado para descartar dados obtidos enquanto o modelo não atingiu estado estacionário.

Todos os parâmetros que refletem as características gerais do modelo de simulação são obtidos através da caixa de mensagens Modelo, apresentada quando o usuário escolhe o item Parametrizar do menu Modelo. Internamente, todos os dados (inclusive os numéricos) são lidos dos campos da caixa de mensagens como cadeias de caracteres, para facilitar a verificação de consistência. Depois da consistência, os dados numéricos são convertidos para inteiro, inteiro sem sinal ou real, dependendo do seu tipo.

2.2 Dados Referentes Aos Recursos Definidos no Modelo

Para cada recurso definido no modelo (isto é, um centro de serviços formado por uma (várias) fila(s) e um (vários) servidor(es)) existe uma estrutura que constitue um nó de uma lista encadeada (cujas funções estão definidas no módulo Listrec.c). Os nós da lista são armazenados em arquivo, com extensão "default" ".rec". Quando o modelo possui ciclos, os dados do ciclo são armazenados em um arquivo com a extensão ".lop" (seção 2.3).

A estrutura armzenada em arquivo tem os seguintes campos (que correspondem exatamente aos campos da lista, exceto o "handle" do próximo nó):

```
typedef struct recur{
        int ident_rec;
        char nome_recurso[[10];
        int num_servidores,
        distribuição_serviço;
        char est_max_min,
        est_fila_vazia,
        media_servico[10],
        pto_triang_servico[10];
        int seq_num_aleat_servico,
        disciplina,
        escolha_fila,
        escolha_cliente,
        escolha_servidor,
        escolha_prox_rec,
        distribuição_chegada;
        char media_chegada[10],
        desvio_padrao_chegada[10],
        pto_triang_chegada[10];
        int seq_num_aleat_chegada;
        char entradas_saidas[41],
        sem_fila;
       unsigned int num_filas,
        flag_sinc;
        char libera_recurso;
```

} auxrec;

.ident_rec: identificação do recurso, utilizada pelo EdGraf para verificar se um recurso já foi ou não parametrizado. O Gerador de Aplicação não utiliza esse valor;

.nome_recurso: identificação do recurso;

.num_servidores: número de servidores do recurso;

.distribuicao_servico: distribuição do tempo de serviço, podendo ser:

.0: exponencial;

.1: erlang;

.2: hiperexponencial;

.3: uniforme;

.4: triangular.

No caso do usuário indicar que vai utilizar distribuição tabelada, o número da distribuição pode ser 5, 6 etc, até atingir o número (4 + número de distribuições tabeladas que o usuário fornecer na caixa Modelo);

.est_max_min: indica se o usuário deseja estatísticas sobre o tamanho máximo e mínimo da fila. Deve ser usado apenas quando um se tratar de um centro de serviço com uma fila, pois SMPL não implementa centros de serviço com várias filas. Os dados são apresentados ao final da execução da simulação em forma de relatório, após o relatório padrão do SMPL;

.est_fila_vazia: indica se o usuário deseja estatísticas sobre a porcentagem de vezes em que o cliente encontra a fila vazia. Também deve ser utilizado quando se tratar de um centro de serviço com apenas uma fila. A estatística é apresentada ao final da execução da simulação, após o relatório padrão do SMPL;

.media_servico: tempo médio de serviço;

.desvio_padrao_servico: desvio padrão do tempo de serviço;

.pto_triang_servico: ponto mais provável da distribuição triangular. Caso outra distribuição seja utilizada, esse valor deve ser 0.0;

.seq_num_aleat_servico: sequência para geração de números aleatórios para o tempo de serviço. 0 indica que a escolha será feita internamente pelo SMPL, 1-15 indicam diferentes sequências.

.disciplina: indica o algoritmo de escalonamento para a formação de filas e atendimento dos clientes (no SMPL, utiliza-se FCFS ("First Come First Served) mas deixou-se como opção para futuras expansões do ASiA a outras extensões e linguagens de simulação). Poderão também ser consideradas as seguintes disciplinas [SPO93]:

.LCFS: ("Last Come First Served") : essa disciplina tem a estrutura de uma pilha, onde o último a chegar é o que entrará em serviço;

.RR ("Round Robin"): onde cada cliente é atendido por um pequeno intervalo de tempo, denominado "quantum". Caso não tenha sido suficiente, o cliente volta para o final da fila de espera, e o cliente seguinte entra para receber seu "quantum". O processo se repete até que o serviço requisitado pelo cliente se complete;

.Tempo de Serviço: o usuário com maior (ou menor) tempo de serviço esperado é atendido primeiro;

.Uso de prioridades: as formas de escalonamento com prioridade são utilizadas para dar preferência a alguns clientes, podendo ser com ou sem preempção.

.escolha_fila: algoritmo para determinar de qual fila retirar o próximo cliente a ser atendido (para o caso de centros de serviço com mais de uma fila). Com o uso do SMPL para gerar os programas de simulação, este parâmetro não é utilizado, mas deve ser mantido para futuras expansões do ASiA a outros sistemas de simulação;

.escolha_cliente: algoritmo para determinar em qual fila o cliente irá entrar, caso o centro de serviço tenha mais de uma fila. Também não é utilizado, mas é mantido para futuras expansões do ASiA;

.escolha_servidor: indica a maneira de escolher um servidor quando mais de um está disponível (SMPL escolhe o primeiro em uma pesquisa sequencial na estrutura que implementa um centro de serviço). Essa informação também foi mantida prevendo-se expansões do ASiA;

.escolha_prox_rec: determina o modo de decisão para escolha do próximo recurso. Ou seja, caso a saída de um recurso esteja ligada a vários outros, deve hever uma decisão para escolher para qual recuso um cliente do sistema sendo simulado deve se dirigir. Pode ser:

.0: utiliza intervalos de probabilidade. No caso de o recurso estar ligado a apenas um, esse campo deve ser 0. Se o usuário digita um valor diferente, o EdGraf detecta automaticamente, e coloca 0. Primeiro o EdGraf pede pelas probabilidades de todas as ramificações, e depois da ligação (EdGraf permite somente uma ligação saindo de um recurso, as demais devem ser ramificações). As ramificações são parametrizadas na ordem que foram desenhadas;

.1: por meio de ciclos, isto é, um cliente entra na fila do recurso, é atendido e volta para a fila. Repete esse procedimento até que atinjam o número especificado de vezes. EdGraf apresenta a caixa Ciclo, que pede por esse número;

.2: utilizado para sistemas fechados, deve ser especificado para o primeiro recurso definido no modelo.

.distribuicao_chegada: distribuição do tempo entre-chegadas, e que segue as mesmas considerações para o tempo de serviço;

.media_chegada: tempo médio entre-chegadas de novos clientes ao sistema;

.desvio_padrao_chegada: desvio padrão do tempo entre-chegadas;

.seq_num_aleat_chegada: sequência para geração de números aleatórios para o tempo entre-chegadas no sistema. 0 indica que a escolha será feita internamente pelo SMPL, 1-15 indicam diferentes sequências;

.entradas_saidas: utilizado para armazenar o número do próximo recurso, e se existe um ciclo, o número de vezes em que um cliente deve passar pelo mesmo recurso. Assim, para cada ligação ou ramificação do recurso para

outro, esse campo recebe um grupo de 4 caracteres: os dois primeiros são usados para identificar o próximo recurso, e os dois outros dígitos dependem do valor de escolha_prox_rec:

.esc_prox_rec=0: nesse caso, vai haver uma escolha através de probabilidade. Os dois dígitos seguintes representam um intervalo para a probabilidade, isto é, 10% para o primeiro recurso seguinte significa "11", 20% significa "12" etc. O EdGraf aceita somente dezenas, isto é, 10%, 20%, 30% etc. As probabilidades são fornecidas através das caixas de diálogo Ramificação e Ligação Principal, apresentadas após a caixa Recurso, do submenu Parametrizar (menu Recursos).

.esc_prox_rec = 1: os dois dígitos seguintes podem ser:

."00": quando cria um ciclo (o número de passagens por esse ciclo é pedido pela caixa Ciclo, após fornecer os parâmetros requisitados pela caixa Recursos);

."10": quando uma entrada deve ser gerada após o término do ciclo, isto é, após o cliente passar as n vezes pelo recurso, ele deve ir para outro ou sair do sistema.

.esc_prox_rec=2: utilizado para modelos fechados. Deve ser determinado para o primeiro recurso do modelo. Segue a mesma sintaxe do caso 1, e devem ser escalonados eventos para ocorrer. Para definir esses eventos, o menu Outros apresenta o submenu Evento.

.sem_fila: indica se o recurso tem ou não fila. Ainda não é captado pelo EdGraf, portanto esse campo recebe um valor nulo;

.num_filas: indica o número de filas do recurso;

.flag_sinc: deve indicar com quantos recursos o recurso em questão está sincronizado. Não está sendo utilizado pelo EdGraf;

.libera_recurso: indica se deve liberar um recurso passivo logo após a sincronização com outros recursos (um recurso passivo é aquele que não oferece serviço, mas deve estar alocado ao cliente quando este está recebendo serviço de um outro recurso). Esse campo também não está sendo utilizado pelo EdGraf, e recebe um valor nulo.

Esses parâmetros são obtidos pelas caixas de diálogo Recurso, Chegada, Fila e Serviço, apresentadas pelos ítens do submenu Parametrizar, do menu Recursos. Os dados numéricos são lidos como cadeias de caracteres, para facilitar a verificação de consistência.

2.3 Dados Relativos Aos Ciclos Definidos No Modelo

Tem-se um ciclo quando por exemplo um cliente executa um número determinado de passagens no mesmo recurso do sistema. Conforme o EdGraf capta os dados do ciclo, armazena os mesmos em uma lista encadeada (cada ciclo corresponde a uma estrutura que consiste em um nó de uma lista encadeada, definida no módulo Listloop.c). Todas as informações mantidas na lista são armazenadas em arquivo com a extensão "lop". O arquivo é criado automaticamente quando os parâmetros do recurso são gravados, mesmo que não tenham sido fornecidos parâmetros para ciclos.

Cada estrutura gravada no arquivo tem os campos:

```
typedef struct tot_loop{
        char n_rec[10];
        unsigned int n_loop;
} total_loop;
```

.n_rec: identificação do recurso, é utilizado pelo Gerador de Aplicação para identificar uma variável contadora do número de vezes que um cliente deve passar pelo ciclo;

.n_loop: número de vezes que um cliente deve passar pelo ciclo.

A identificação do recurso é obtida através da indicação com o "mouse". O número de voltas no ciclo é obtido através da caixa Ciclo, apresentada após a caixa Dados do Recurso quando Escolha do Próximo Recurso tem o botão Ciclo selecionado.

2.4 Dados Relativos Às Distribuições Tabeladas

Uma facilidade oferecida pelo EdGraf é possibilitar ao usuário definir suas próprias distribuições de probabilidade, através de tabelas. Cada entrada da tabela, com o valor observado e o número de ocorrências, corresponde a uma estrutura que consiste em um nó de uma lista encadeada na memória (módulo Listdist.c). Para cada distribuição, um arquivo de dados é produzido, com extensão "default" ".dis". Cada nó da lista é gravado no arquivo.

Cada estrutura gravada no arquivo tem os campos:

```
typedef struct distrib_tab {
        char valor[5],
        prob[6];
} D_Tab;
```

.valor: valor cujo número de ocorrências foi observado;

.prob: frequência da ocorrência deste valor, dentre o total de observações efetuadas.

O EdGraf obtém do usuário o valor e o número de ocorrências para aquele valor, para todas as entradas da tabela, através da caixa Distribuição, apresentada com o item Nova, do submenu Distribuição Tabelada (menu Outros). Quando o usuário termina de digitar, EdGraf obtém a distribuição de frequência relativa aos dados fornecidos (dividindo-se cada número de ocorrência pelo número total de ocorrências). Esta distribuição é passada para o Gerador de Aplicação, que obtém a distribuição de frequência acumulada [SPO93a].

2.5 Dados Relativos aos Eventos Definidos no Modelo

Para modelos de sistemas fechados, normalmente alguns eventos são escalonados antes do início da simulação. O usuário pode definir esses eventos através do submenu Evento Chegada, do menu Outros da barra de menus. Cada evento definido no modelo corresponde a uma estrutura que consiste em um nó de uma lista encadeada (módulo Listeven.c). O arquivo de eventos tem extensão "default" ".eve".

Cada estrutura gravada no arquivo tem os campos:

```
struct escalona_eventos{
        int num_recurso;
        char tempo_ocorrencia[9];
} Temp_Eventos;
```

.num_recurso: número do recurso para o qual um evento deve ser escalonado, como a chegada de um cliente ou a utilização deste recurso;

.tempo_ocorrencia: tempo para o qual o evento especificado será escalonado.

O número ou identificação do recurso é obtido através do "mouse", quando o usuário indica o recurso para definir evento. O tempo ocorrência é obtido através da caixa Evento Chegada, apresentada com o item Definir do submenu Evento Chegada (menu Outros). O tempo de ocorrência é obtido como uma cadeia de caracteres, para que possa ser verificada sua consistência.

3. Considerações Sobre A Implementação

O EdGraf foi desenvolvido utilizando-se a linguagem C, com o compilador Borland C++ versão 3.1, em microcomputador IBM-PC 386. Para a interface ser mais amigável, fez-se uso dos recursos oferecidos pelo MS-Windows 3.1. O Windows oferece inúmeras rotinas internas que permitem o uso de menus, caixas de diálogos, ícones, cursores e outros componentes de uma interface amigável [PET93].

Através dessas ferramentas, o Editor Gráfico apresenta, na tela do monitor, uma barra de menus, uma barra de ferramentas, uma barra de título e uma janela de edição gráfica.

A janela de edição gráfica (que ocupa a maior parte do vídeo) permite ao usuário desenhar a especificação do modelo desejado. À esquerda dessa janela, o Editor apresenta uma janela contendo um conjunto de ícones que irão auxiliar o usuário na tarefa de compor um modelo de simulação. Os ícones representam primitivas gráficas para especificar o modelo, e formam a barra de ferramentas.

A barra de menus, apresentada acima das janelas de ícones e edição gráfica, apresenta os menus que irão auxiliar o usuário na tarefa de parametrizar o modelo. A barra de título situa-se acima da barra de menus, e contém a informação referente ao nome do sistema de edição gráfica: "EdGraf - Editor Gráfico ASiA".

Para facilitar o trabalho do usuário, todas as caixas de diálogo que tratam de arquivos são as caixas padrão, idênticas às caixas de outros aplicativos Windows.

A barra de ferramentas é composta por dez ícones, que representam os tipos de centros de serviço para serem desenhados, ligações, ramificações, uma seta para seleção e o ícone de uma borracha, para apagar símbolos, ligações ou ramificações. Todos os ícones foram projetados através do Resource Workshop [RES91] e estão descritos no módulo Editor.rc (seção 5.14). A barra de menus, com todos os submenus e caixas de diálogo, também foram projetadas utilizando-se todas as facilidades do Resource Workshop.

Com relação à janela de edição gráfica, do ponto de vista do programador é uma janela filha da janela principal que contém a barra de título e a barra de menus. Para manipular os desenhos, foi necessário definir o modo de mapeamento para utilizar as

coordenadas da janela. Um atributo do dispositivo de contexto que afeta virtualmente todo o desenho feito na janela de edição é o modo de mapeamento. Esse deixa implícitas uma origem e uma orientação para os eixos de coordenadas x e y. O modo de mapeamento utilizado foi MM_TEXT (que tem esse nome por causa da orientação dos eixos). A figura 3.1 mostra a orientação dos eixos x e y [PET93].

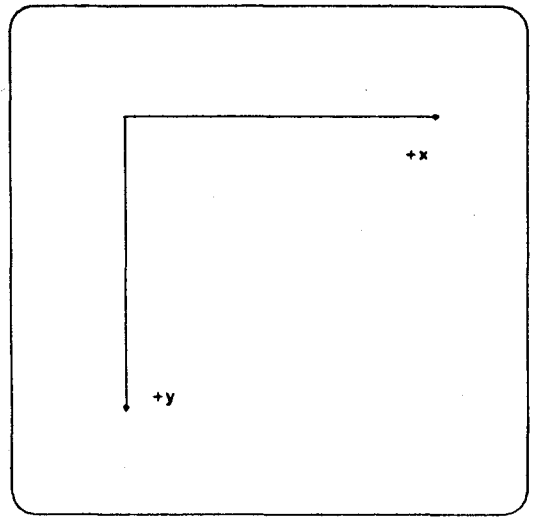

Figura 3.1: Orientação dos eixos no modo MM_TEXT

Para cada símbolo desenhado (de centros de serviço com uma ou várias filas, e um ou vários servidores) tem-se um retângulo imaginário cujo canto superior esquerdo corresponde exatamente ao canto superior esquerdo do desenho. É esse ponto que é armazenado na lista de símbolos (seção 5.3). As dimensões de cada símbolo dependem do seu tipo, e são utilizadas para desenhar e verificar se o botão do "mouse" foi pressionado sobre o símbolo.

Outra observação importante a fazer é que quando um símbolo é removido, todas as ligações e ramificações que chegam e/ou partem dele também são removidas. Caso o usuário tenha parametrizado algum recurso, toda a parametrização também é removida. Isso porque a parametrização deve ser feita na ordem em que os símbolos foram desenhados, isto é, o primeiro símbolo deve ser o primeiro a ser parametrizado e assim sucessivamente.

Com relação às ligações e ramificações desenhadas, EdGraf utiliza para cada recurso definido no modelo uma variável do tipo cadeia de caracteres, que indica os recursos aos quais o primeiro está ligado e como é feita a escolha para o próximo (através de probabilidade, ou ciclos). Dessa forma, o EdGraf limita a dez o número de ligações e ramificações que partem de um recurso. E o desenho na janela de edição não é inteligente, ou seja, não verifica se vai sobrepor outro.

4. Os Arquivos da Especificação Gráfica

O EdGraf possibilita ao usuário gravar em arquivo a especificação gráfica do modelo. Dois arquivos são produzidos, com extensões ".esp" e ".lig", sendo que o segundo é um arquivo auxiliar, sempre produzido automaticamente pelo Editor e com o mesmo nome do primeiro, mudando-se apenas a extensão. Assim, a leitura e gravação do arquivo ".lig" é transparente ao usuário, pois o EdGraf sempre trata genericamente como arquivo de especificação. Ambos são descritos a seguir.

4.1 Arquivo de Dados Sobre Os Símbolos Desenhados

Para cada símbolo de recurso que o usuário desenha na janela de edição gráfica, corresponde uma estrutura que é um nó da lista de especificacão gráfica (cujas rotinas estão descritas no módulo Lista.c). As informações de cada nó são gravadas em arquivo que possui a extensão "default" ".esp".

A estrutura gravada em arquivo tem os campos (contém os mesmos campos da estrutura dos nós da lista de especificacão gráfica, exceto o "handle"):

```
typedef struct simb{
        int x,
        int y;
        char tipo;
        int identrec;
} auxsimb;
```

.x, y: coordenadas x e y do canto superior esquerdo de um retângulo imaginário, que envolve o símbolo e serve para delimitar o espaço no caso de identificar um símbolo através do "mouse";

.tipo: o tipo do símbolo (definidos em EDITOR.H). Pode ser:

.REC1: símbolo de um centro de serviço com uma fila e um servidor;

.REC2: simbolo de um centro de serviço com uma fila e vários servidores;

.REC3: símbolo de um centro de serviço com várias filas e um servidor;

.REC4: símbolo de um centro de serviço com várias filas e vários servidores.

.identrec: a identificação, que indica a ordem em que foram desenhados e vai ser utilizada para identificar o recurso na parametrização. É esse valor que preenche o campo ident_rec, da lista de recursos, quando o recurso é parametrizado;

4.2 Arquivo de Ligações e Ramificações

Quando o usuário define ligações e ramificações entre os recursos definidos no modelo, as informações referentes a tais ligações e ramificações são armazenadas em uma lista encadeada (rotinas do módulo Listalig.c). As informações de cada nó são gravadas em arquivo com a extensão "default" ".lig".

A estrutura gravada em arquivo tem os campos (contém os mesmos campos da estrutura dos nós da lista de especificacão gráfica, exceto o "handle"):

```
typedef struct lig{
        int recurso1,
            recurso2,
            x1, y1,
            x2, y2;
        char tipo;
} auxlig;
```

.recurso1: identificação do recurso inicial;

.recurso2: identificação do recurso final;

.x1, y1: ponto inicial da ligação ou ramificação;

.x2, y2: ponto final da ligação ou ramificação;

.tipo: indica se é ligação (LIGACAO) ou ramificação (RAMIFICACAO).

5. Descrição dos Módulos e Principais Rotinas Implementadas

Esta seção descreve cada módulo e as principais rotinas implementadas no projeto do EdGraf.

5.1 Módulo EDITOR.C

Este é o módulo principal do Editor Gráfico. Contém os protótipos das funções dos demais módulos (que são utilizadas nesse módulo), todas os procedimentos responsáveis pelas caixas de diálogo, o procedimento WinMain e os procedimentos de janela (janela principal, janela de edição gráfica e janela de ícones - onde está a barra de ferramentas).

O procedimento WinMain consiste no ponto de entrada de um programa Windows, exatamente como main() em um programa escrito em C. É nesse procedimento que são registradas os três tipos de classes de janelas utilizadas no Editor: a janela principal, e as janelas filhas de edição gráfica e a janela onde está a barra de ferramentas (os ícones). A classe de janela vai identificar o procedimento de janela que processa as mensagens para aquela janela [PET93]. Os procedimentos de janela no EDITOR.C são ProcJan, ProcJanFilha e ProcJanFIcones.

As mensagens que os procedimentos de janela não processam são passadas à funcão DefWinProc, do Windows.

O procedimento ProcJan recebe as mensagens endereçadas à janela principal (que é a janela mãe das janelas de edição e de ícones). As mensagens e o tratamento para cada uma delas são descritos a seguir:

.WM_CREATE: no processamento dessa mensagem, são criados todos os procedimentos responsáveis pelas caixas de diálogo, e as duas janelas-filha: de edição e de ícones. São também iniciadas as estruturas responsáveis pelas caixas de diálogo do tratamento de arquivos (modelo, recursos, especificação gráfica, eventos, distribuição tabelada);

.WM_SIZE: adapta o tamanho da janela de edição quando o usuário altera o tamanho;

.WM_INITMENUPOPUP: a cada vez que o usuário ativa um menu da barra de menus, no processamento dessa mensagem são verificadas as condições para que determinados itens de cada um dos menus sejam ativados ou desativados;

.WM_COMMAND: processa as mensagens vindas dos submenus da barra de menus. O procedimento executa as ações correspondentes a cada item dos menus;

.WM_CLOSE: nesse caso, é verificado se a especificação gráfica ou parâmetros do modelo de simulação ainda não foram gravados em arquivo. Antes de encerrar o Editor Gráfico, pergunta ao usuário se deseja gravar os dados. Também apaga da memória todas as listas encadeadas: a lista que armazena a especificação gráfica, de recursos, eventos, distribuição tabelada, ciclos, ligações;

.WM_QUERYENDSESSION: como WM_CLOSE;

.WM_DESTROY: o EdGraf finaliza sua execução.

O procedimento ProcJanFilha recebe as mensagens endereçadas à janela de edição gráfica. As mensagens e o tratamento para cada uma são descritos a seguir:

.WM_SETFOCUS: a janela de edição obtém o foco de entrada;

.WM_LBUTTONDOWN: o botão esquerdo do "mouse" foi pressionado, na área da janela. Nesse caso, pode indicar a seleção de um recurso para parametrizar ou definir evento, ou então a posição na janela para desenhar ou apagar um símbolo. Obtém a posição do cursor do "mouse", para identificar o recurso selecionado, ou a posição para desenho, e executa as ações apropriadas;

.WM_LBUTTONUP: indica que o botão esquerdo do "mouse", que estava pressionado, foi liberado. Nesse caso, desenha o símbolo escolhido pelo usuário na posição onde foi pressionado o botão do "mouse", ou então a seta que indica recurso selecionado para parametrização ou definição de evento. Determinados desenhos não podem ser repetidos, como as setas que indicam início ou fim, e ligações ou ramificações de e para os mesmos recursos. Nesse caso, uma caixa de mensagem avisa ao usuário que o

desenho já existe. Caso seja apagado um símbolo, todas as ligações e ramificações de e para o mesmo são apagadas também; e caso algum recurso já tenha sido parametrizado, toda a parametrização é cancelada (a lista de recursos é removida);

.WM_PAINT: essa mensagem informa ao Editor quando surge a necessidade de (re)pintar a janela de edição. Verifica se existem símbolos e ligações a serem desenhados, ou se existe algum recurso selecionado (para desenhar a seta que aponta para o mesmo). O mesmo ocorre quando qualquer símbolo é apagado, repinta somente a porção da janela afetada.

As mensagens para a janela que contém a barra de ferramentas são tratadas pelo procedimento ProcjanFIcones. As mensagens e o tratamento para cada uma são descritos abaixo:

.WM_CREATE: quando da criação da janela, obtém os "handles" dos ícones utilizados, bem como sua dimensão;

.WM_LBUTTONDOWN: quando o usuário pressiona o botão esquerdo do "mouse" sobre um ícone, determina qual é este ícone (através de uma função que verifica a posição do ponto indicado pelo "mouse" em relação à janela). A cor do ícone é invertida, só voltando a cor original quando outro ícone for selecionado. O cursor muda do formato de uma seta para uma cruz. Indica qual o ícone escolhido, para efetuar a operação correta (desenho ou seleção) quando houver a pressão do botão do "mouse" na janela de edição;

.WM_LBUTTONUP: quando o usuário libera o botão do "mouse", o cursor volta ao formato de uma seta;

.WM_PAINT: quando o procedimento recebe essa mensagem, os ícones da janela são (re)pintados.

No EDITOR.C também estão todos os procedimentos responsáveis pelo tratamento das caixas de diálogo:

.O procedimento ProcDlgApresentacao recebe as mensagens da caixa de apresentação, ativada quando o usuário inicia uma sessão do EdGraf;

.ProcDlgSobre é responsável pelas mensagens da caixa Sobre, ativada quando o usuário escolhe o item Sobre no menu Ajuda;

.ProcDlgModelo recebe as mensagens da caixa de diálogo Modelo, através da qual o usuário parametriza o modelo. É ativada quando o usuário escolhe o item Parametrizar, no menu Modelo;

.O procedimento ProcDlgRecursos recebe as mensagens da caixa Dados do Recurso, com os dados principais que identificam um recurso. É ativada através do item Recurso, do submenu Parametrizar, menu Recursos;

.Os dados da fila de um recurso são obtidos através da caixa Fila, gerenciada pelo procedimento ProcDlgRecFila. Ativada pelo item Fila, do submenu Parametrizar, menu Recursos;

.ProcDlgRecServico é responsável pela caixa de diálogo Serviço, que recebe os dados relativos ao tempo de serviço de um recurso. A caixa é ativada pelo item Serviço, do submenu Parametrizar, menu Recursos;

.O procedimento ProcDlgRecChegada recebe as mensagens da caixa Chegada, pela qual o usuário parametriza o tempo entre-chegadas de novos clientes. É ativada pelo item Chegada, do submenu Parametrizar, menu Recursos;

.O Editor permite ao usuário visualizar os dados gerais do modelo, fornecidos pelo usuário. O item Mostar, do submenu Modelo, apresenta uma caixa de diálogo com tais informações. Essa caixa é gerenciada pelo procedimento ProcDlgMostraDadosModelo;

.Os procedimentos ProcDlgRecRealimentacao e ProcDlgLigacaoPrincipal são responsáveis pelas caixas Ramificação e Ligação Principal, que obtêm os dados relativos a porcentagens quando existe escolha na saída de um recurso. São ativadas durante a parametrização do recurso, após a caixa Recursos;

.ProcDlgDistribuicao é responsável pela caixa Distribuição Tabelada, que obtém os dados para a definição de uma distribuição tabelada. É apresentada pelo item Nova do submenu Distribuição Tabelada, menu Outros;

.O procedimento ProcDlgEventos recebe as mensagens da caixa Evento, ativada pelo item Definir do submenu Evento, menu Outros. Permite definir eventos e respectivos tempos de ocorrência;

.ProcDlgLoop recebe os dados da caixa Ciclo, que obtém o número de vezes que um cliente deve passar repetidamente por um recurso. É ativado durante a parametrização do recurso, após a caixa Dados do Recurso.

5.2 Módulo EDITARQ.C

Este módulo contém as funções de manipulação de arquivos do Editor Gráfico. Utiliza as rotinas comuns de caixas de diálogos para Abrir, Salvar e Salvar Como (as caixas de diálogo padrão do Windows, definidas em COMMDLG.DLL) [MIC92, PET93].

Para cada arquivo produzido pelo EdGraf (arquivo de dados do modelo, de dados dos recursos, eventos, distribuição tabelada, especificação gráfica e dois arquivos auxiliares para a especificação e recursos) está associada uma estrutura OPENFILENAME, definida em WINDOWS.H [MIC92b]. Essa estrutura contém informações que o sistema utiliza para iniciar as caixas de diálogo Abrir, Salvar e Salvar Como, padrões do Windows 3.1. O sistema retorna nessa estrutura informações sobre as entradas do usuário, como o nome do arquivo e diretório correspondente.

A extensão "default" para todos os arquivos produzidos depende da informação armazenada:

."mod": parâmetros gerais do modelo;

."esp" e ".lig": especificação gráfica;

."rec" e ".lop": parâmetros dos recursos definidos no modelo;

."dis": distribuição tabelada;

."eve": dados dos eventos chegada definidos no modelo.

Cada estrutura associada aos arquivos tem os seguintes campos [MIC92b] (alguns desses campos são definidos quando o EdGraf é iniciado, outros através das caixas de

diálogo para abrir e salvar arquivo - são os campos que dependem o arquivo a ser aberto):

.lStructSize: tamanho da estrutura, em bytes;

.hwndOwner: especifica o handle da janela. Definido nas funções de iniciar arquivo;

.hInstance: especifica um bloco de dados para a caixa de diálogo. Não é utilizado.

.lpstrFilter: aponta para um "buffer" contendo cadeias de caracteres especificando filtros (por exemplo, "*.dat", "*.txt"). Definido nas funções de iniciar arquivo;

.lpstrCustomFilter: deve ser NULL para que não sejam salvos filtros definidos pelo usuário;

.nMaxCustFilter: tamanho, em bytes, do "buffer" especificado pelo campo anterior;

.nFilterIndex: usado para identificar o filtro padrão das janelas de arquivo;

.lpstrFile: especifica um nome de arquivo. Definido nas funções de abrir e fechar;

.nMaxFile: especifica o tamanho, em bytes, do campo anterior;

.lpstrFileTitle: título do arquivo selecionado (nome e extensão, mas nenhuma informação sobre o diretório). Definido nas funções de abrir e fechar;

.nMaxFileTitle: tamanho máximo, em bytes, do campo anterior;

.lpstrInitialDir: diretório inicial (se for nulo, usa o diretório corrente);

.lpstrTitle: título a ser colocado na barra da caixa de diálogo. Se for nulo, usa os títulos "default": Salvar, Salvar Como, Abrir;

.Flags: especificam condições para iniciar as caixas de diálogo. Definido nas funções de abrir e fechar;

.nFileOffset: número de caracteres que indicam o diretório, no campo lpstrFile;

.nFileExtension: número de caracteres que indicam o diretório mais o nome do arquivo, excluindo a extensão, do campo lpstrFile;

.lpstrDefExt: extensão "default". Definido nas funções de iniciar arquivo;

.lCustData: campo de dados, nao utilizado;

.lpfnHook: aponta para uma função que processa mensagens, não utilizada;

.lpTemplateName: nome do arquivo de recursos, caso não sejam utilizadas as caixas de diálogo padrão. Este campo recebe um valor nulo, pois o EdGraf utiliza as caixas de diálogo.

Rotinas:

void InicializaArqModelo(HWND hjan)

Entradas:

.hjan: handle da janela.

Inicia a estrutra OPENFILENAME ofnArqModelo, com dados necessários para criação e manipulação do arquivo necessário para armazenar os dados do modelo definido pelo usuário [MIC92b].

Campos da estrutura alterados:

.hwndOwner: recebe hjan;

.lpstrFilter: recebe um ponteiro para um filtro definido localmente;

.lpstrdefExt: recebe a extensão "default" ".mod".

void InicializaArqRecursos(HWND hjan)

Entradas:

.hjan: handle da janela.

Inicializa a estrutra OPENFILENAME ofnArqRecursos, com dados necessários para criação e manipulação de um arquivo para armazenar os dados dos recursos definido pelo usuário [MIC92b]. Os campos são alterados como em InicializaArqModelo, porém a extensão "default" é ".rec".

void InicializaArqEspecGrafica(HWND hjan)

Entradas:

hjan: handle da janela.

Inicializa a estrutra OPENFILENAME ofnArqEspecGrafica, com dados necessários para criação e manipulação de um arquivo para armazenar a especificação gráfica definida pelo usuário [MIC92b]. Os campos são alterados como em InicializaArqModelo, porém a extensão "default" é ".esp".

void InicializaArqLigacoes(HWND hjan)

Entradas:

.hjan: handle da janela.

Inicia a estrutra OPENFILENAME ofnArqLigacoes, com dados necessários para criação e manipulação de um arquivo para armazenar dados relativos a ciclos definidos no modelo [MIC92b]. Os campos são alterados como em InicializaArqModelo, porém a extensão "default" é ".lig".

void InicializaArqLoop(HWND hjan)

Entradas:

.hjan: handle da janela.

Inicia a estrutra OPENFILENAME ofnArqLoop, com dados necessários para criação e manipulação de um arquivo para armazenar dados relativos a ciclos definidos no modelo [MIC92b]. Os campos são alterados como em InicializaArqModelo, porém a extensão "default" é ".lop".

void InicializaArqDist(HWND hjan)

Entradas:

.hjan: handle da janela.

Inicia a estrutra OPENFILENAME ofnArqDist, com dados necessários para criação e manipulação de um arquivo para armazenar os dados referentes à distribuição tabelada definida [MIC92b]. Os campos são alterados como em InicializaArqModelo, porém a extensão "default" é ".dis".

void InicializaArqEventos(HWND hjan)

Entradas:

.hjan: handle da janela.

Inicia a estrutra OPENFILENAME ofnArqEventos, com dados necessários para criação e manipulação de um arquivo para armazenar os dados referentes a eventos definidos [MIC92b]. Os campos são alterados como em InicializaArqModelo, porém a extensão "default" é ".eve".

void InicializaArqtudo(HWND hjan)

Entradas:

hjan: handle da janela.

Inicia a estrutura OPENFILENAME ofnArqTudo, utilizada para apresentar as caixas de diálogos para abrir e salvar todos os arquivos, utilizando o mesmo nome com extensões diferentes.

BOOL DlgAbrirArqModelo(HWND hjan, LPSTR lpstrNomeArq)

Entradas:

.hjan: handle da janela;

.lpstrNomeArq: nome do arquivo a ser aberto.

Campos da estrutura alterados:

.Flags: OFN_CREATEPROMPT, a caixa Abrir gera uma caixa de mensagem avisando ao usuário que o arquivo não existe.

Em seguida chama a função GetOpenFileName, a qual apresenta uma caixa de mensagem perguntado ao usuário se o arquivo deve ser criado, caso não exista um com o nome selecionado [MIC92a]. Essa função cria uma caixa de diálogo (padrão do Windows) que possibilita ao usuário selecionar um arquivo para abertura.

Retorno:

.FALSE: ocorreu um erro na abertura do arquivo;

.TRUE: o arquivo foi aberto corretamente.

BOOL DlgAbrirArqRecursos(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgAbrirArqModelo.

BOOL DlgAbrirArqEspecificacaoGrafica(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgAbrirArqModelo.

BOOL DlgAbrirArqDistribuicao(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgAbrirArqModelo.

BOOL DlgAbrirArqEventos(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgAbrirArqModelo.

BOOL DlgAbrirArqTudo(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgAbrirArqModelo, mas a estrutura ofnArqTudo apenas tem função para apresentar as caixas de diálogo e obter um nome para abrir todos os arquivos (com extensões diferentes).

BOOL DlgSalvarArqModelo(HWND hjan, LPSTR lpstrNomeArq)

Entradas:

.hjan: handle da janela;

.lpstrNomeArq: nome do arquivo a ser salvo.

Preenche os campos da estrutura OPENFILENAME ofnArqModelo com os parâmtros de entrada:

.hwndOwner: hjan;

.lpstrFile: lpstrNomeArq;

.lpstrFileTitle: lpstrNomeArq;

.Flags: OFN_OVERWRITEPROMPT, a caixa Salvar Como gera uma caixa de mensagem avisando ao usuário caso o arquivo já exista, e pergunta se sobrepõe os dados ou então pede um novo nome.

Chama a função GetSaveFileName, que cria uma caixa de diálogo (padrão do Windows), possibilitando ao usuário selecionar um arquivo a ser salvo.

Retorno:

.TRUE: arquivo salvo corretamente;

.FALSE: erro na gravação do arquivo.

BOOL DlgSalvarArqRecursos(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgSalvarArqModelo.

BOOL DlgSalvarArqEspecificacaoGrafica(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgSalvarArqModelo.

BOOL DlgSalvarArqDistribuicao(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgSalvarArqModelo.

BOOL DlgSalvarArqEventos(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgSalvarArqModelo.

BOOL DlgSalvarArqTudo(HWND hjan, LPSTR lpstrNomeArq)

Como em DlgSalvarArqModelo, a estrutura ofnArqTudo obtém o nome comum para gravar todos os arquivos.

BOOL LerArqModelo(LPSTR lpstrNomeArqModelo)

Entradas:

.lpstrNomeArqModelo: nome do arquivo de dados do modelo.

Abre o arquivo lpstrNomeArqModelo, que contém os dados gerais do modelo.Os dados do arquivo são lidos em um "buffer" auxiliar alocado na memória global (exigência da função _lread [MIC92a, PET93]), e armazenados em variáveis globais definidas em Eddef.h.

O arquivo é fechado através da função _lclose, e o "buffer" desalocado.

Retorno:

.TRUE: o arquivo foi lido corretamente;

.FALSE: erro na leitura do arquivo ou na alocação de variáveis auxiliares na leitura.

BOOL LerArqRecursos(HWND hjan, LPSTR lpstrNomeArqRecursos)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqRecursos: nome do arquivo de dados dos recursos.

Abre o arquivo de dados dos recursos lpstrNomeArqRecursos, lê (com a função_lread) os mesmos em uma estrutura auxiliar alocada na memória global e os armazena em um nó da lista encadeada de recursos (cujas funções estão definidas no módulo Listrec.c). Repete essa operação até encontrar fim de arquivo. A memória é desalocada e o arquivo fechado com _lclose [MIC92a, PET93].

Retorno:

.TRUE: o arquivo foi lido corretamente;

.FALSE: erro na leitura do arquivo ou na alocação de variáveis auxiliares na leitura.

BOOL LerArqEspecGrafica(HWND hjan, LPSTR lpstrNomeArqEspecGrafica)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqEspecGrafica: nome do arquivo.

Abre o arquivo lpstrNomeArqEspecGrafica, que contém a especificação gráfica de um modelo. Lê (com a função_lread) os mesmos em uma estrutura auxiliar alocada na memória global e os armazena em um nó da lista encadeada de símbolos (cujas funções estão definidas no módulo Lista.c). Repete essa operação até encontrar fim de arquivo. A memória é desalocada e o arquivo fechado com _lclose [MIC92a, PET93].

Retorno:

.TRUE: o arquivo foi lido corretamente;

.FALSE: erro na leitura do arquivo ou na alocação de variáveis auxiliares na leitura.

BOOL LerArqDistribuicao(HWND hjan, LPSTR lpstrNomeArqDistribuicao)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqDistribuicao: nome do arquivo.

Abre o arquivo lpstrNomeArqDistribuicao, que contém as entradas referentes à distribuição tabelada. Lê (com a função_lread) os mesmos em uma estrutura auxiliar alocada na memória global e os armazena em um nó da lista encadeada de distribuição tabelada (cujas funções estão definidas no módulo Listdist.c). Repete essa operação até encontrar fim de arquivo. A memória é desalocada e o arquivo fechado com _lclose [MIC92a, PET93].

Retorno:

.TRUE: o arquivo foi lido corretamente;

.FALSE: erro na leitura do arquivo ou na alocação de variáveis auxiliares na leitura.

BOOL LerArqEventos(HWND hjan, LPSTR lpstrNomeArqEventos)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqEventos: nome do arquivo.

Abre o arquivo lpstrNomeArqEventos, que contém as entradas referentes ao eventos definidos no modelo. Lê (com a função_lread) os mesmos em uma estrutura auxiliar alocada na memória global e os armazena em um nó da lista encadeada de eventos (cujas funções estão definidas no módulo Listeven.c). Repete essa operação até encontrar fim de arquivo. A memória é desalocada e o arquivo fechado com _lclose [MIC92a, PET93].

Retorno:

.TRUE: o arquivo foi lido corretamente;

.FALSE: erro na leitura do arquivo ou na alocação de variáveis auxiliares na leitura.

BOOL LerArqLigacoes(HWND hjan, LPSTR lpstrNomeArqLigacoes)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLigacoes: nome do arquivo.

Abre o arquivo lpstrNomeArqLigacoes, que contém as ligações e ramificações entre os recursos definidos definidos no modelo. Lê (com a função_lread) os mesmos em uma estrutura auxiliar alocada na memória global e os armazena em um nó da lista encadeada de ligações (cujas funções estão definidas no módulo Listalig.c). Repete essa operação até encontrar fim de arquivo. A memória é desalocada e o arquivo fechado com _lclose [MIC92a, PET93].

Retorno:

.TRUE: o arquivo foi lido corretamente;

.FALSE: erro na leitura do arquivo ou na alocação de variáveis auxiliares na leitura.

BOOL LerArqLoop(HWND hjan, LPSTR lpstrNomeArqLoop)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLoop: nome do arquivo.

Abre o arquivo lpstrNomeArqLoop, que contém as informações referentes ao ciclos definidos no modelo. Lê (com a função_lread) os mesmos em uma estrutura auxiliar alocada na memória global e os armazena em um nó da lista encadeada de ciclos (cujas funções estão definidas no módulo Listloop.c). Repete essa operação até encontrar fim de arquivo. A memória é desalocada e o arquivo fechado com _lclose [MIC92a, PET93].

Retorno:

.TRUE: o arquivo foi lido corretamente;

.FALSE: erro na leitura do arquivo ou na alocação de variáveis auxiliares na leitura.

BOOL GravarArqModelo(LPSTR lpstrNomeArqModelo)

Entradas:

.lpstrNomeArqModelo: nome do arquivo de dados do modelo.

Grava os dados gerais do modelo, fornecidos pelo usuário, no arquivo lpstrNomeArqModelo. Esses dados são transferidos para uma estrutura auxiliar, alocada na memória global, e gravados com a função _lwrite [MIC92a, PET93]. A estrutura é desalocada e o arquivo fechado com _lclose.

Retorno:

.TRUE: dados gravados corretamente;

.FALSE: erro na gravação do arquivo.

BOOL GravarArqRecursos(HWND hjan, LPSTR lpstrNomeArqRecursos)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqRecursos: nome do arquivo de dados dos recursos.

Grava os dados fornecidos pelo usuário no arquivo lpstrNomeRecursos. Utiliza a função _lwrite [MIC92a]. Como esses dados estão armazenados em uma lista encadeada na memória, a lista é percorrida. Os dados do nó corrente são transferidos para uma estrutura auxiliar alocada na memória global, que é gravada no arquivo (com _lwrite) e desalocada. Essa operação se repete até o fim da lista. O arquivo é fechado com _lclose [PET93, MIC92a].

Retorno:

.TRUE: dados gravados corretamente;

.FALSE: erro na gravação do arquivo.

BOOL GravarArqEspecificacaoGrafica(HWND hjan, LPSTR lpstrNomeArqEspecGrafica)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqEspecGrafica: nome do arquivo de especificação gráfica.

Grava os dados fornecidos pelo usuário no arquivo lpstrNomeArqEspecGrafica. Utiliza a função _lwrite [MIC92a]. Como esses dados estão armazenados em uma lista encadeada na memória, a lista é percorrida. Os dados do nó corrente são transferidos para uma estrutura auxiliar alocada na memória global, que é gravada no arquivo (com _lwrite) e desalocada. Essa operação se repete até o fim da lista. O arquivo é fechado com _lclose [PET93, MIC92a].

Retorno:

.TRUE: dados gravados corretamente;

.FALSE: erro na gravação do arquivo.

BOOL GravarArqLigacoes(HWND hjan, LPSTR lpstrNomeArqLigacoes)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLigacoes: nome do arquivo de ligações.

Grava os dados referentes às ligações e ramificações definidas no modelo, no arquivo lpstrNomeArqLigacoes. Utiliza a função _lwrite [MIC92a]. Como esses dados estão armazenados em uma lista encadeada na memória, a lista é percorrida. Os dados do nó corrente são transferidos para uma estrutura auxiliar alocada na memória global, que é gravada no arquivo (com _lwrite) e desalocada. Essa operação se repete até o fim da lista. O arquivo é fechado com _lclose [PET93, MIC92a].

Retorno:

.TRUE: dados gravados corretamente;

.FALSE: erro na gravação do arquivo.

BOOL GravarArqLoop(HWND hjan, LPSTR lpstrNomeArqLoop)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLoop: nome do arquivo de ciclos.

Grava os dados referentes ao ciclos do modelo, no arquivo lpstrNomeArqLoop. Utiliza a função _lwrite [MIC92a]. Como esses dados estão armazenados em uma lista encadeada na memória, a lista é percorrida. Os dados do nó corrente são transferidos para uma estrutura auxiliar alocada na memória global, que é gravada no arquivo (com _lwrite) e desalocada. Essa operação se repete até o fim da lista. O arquivo é fechado com _lclose [PET93, MIC92a].

Retorno:

.TRUE: dados gravados corretamente;

.FALSE: erro na gravação do arquivo.

BOOL GravarArqDistribuicao(HWND hjan, LPSTR lpstrNomeArqDistribuicao)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqDistribuicao: nome do arquivo de distribuição.

Grava os dados referentes à distribuição tabelada, no arquivo lpstrNomeArqDistTabelada. Utiliza a função _lwrite [MIC92a]. Como esses dados estão armazenados em uma lista encadeada na memória, a lista é percorrida. Os dados do nó corrente são transferidos para uma estrutura auxiliar alocada na memória global, que é gravada no arquivo (com _lwrite) e desalocada. Essa operação se repete até o fim da lista. O arquivo é fechado com _lclose [PET93, MIC92a].

Retorno:

.TRUE: dados gravados corretamente;

.FALSE: erro na gravação do arquivo.

BOOL GravarArqEventos(HWND hjan, LPSTR lpstrNomeArqEventos)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqEventos: nome do arquivo de eventos definidos.

Grava os dados referentes aos eventos no arquivo lpstrNomeArqEventos. Utiliza a função _lwrite [MIC92a]. Como esses dados estão armazenados em uma lista encadeada na memória, a lista é percorrida. Os dados do nó corrente são transferidos para uma estrutura auxiliar alocada na memória global, que é gravada no arquivo (com _lwrite) e desalocada. Essa operação se repete até o fim da lista. O arquivo é fechado com _lclose [PET93, MIC92a].

Retorno:

.TRUE: dados gravados corretamente;

.FALSE: erro na gravação do arquivo.

void AuxAbrirArqLigacoes (HWND hjan, LPSTR lpstrNomeArqLigacoes)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLigacoes: nome do arquivo.

Preenche campos da estrutra ofnArqLigacoes, mas não apresenta a caixa de Abrir Arquivo, pois esse arquivo é lido juntamente com o arquivo de especificação gráfica.

Preenche os campos da estrutura OPENFILENAME ofnArqLigacoes com os parâmetros de entrada:

.hwnOwner: hjan;

.lpstrFile: lpstrNomeArqLigacoes;

void AuxAbrirArqLoop (HWND hjan, LPSTR lpstrNomeArqLoop)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLoop: nome do arquivo.

Preenche campos da estrutra ofnArqLoop, mas não apresenta a caixa de Abrir Arquivo, pois esse arquivo é lido juntamente com o arquivo de recursos.

Preenche os campos da estrutura OPENFILENAME ofnArqLigacoes com os parâmetros de entrada:

.hwnOwner: hjan;

.lpstrFile: lpstrNomeArqLoop;

void AuxSalvarArqLigacoes (HWND hjan, LPSTR lpstrNomeArqLigacoes)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLigacoes: nome do arquivo.

Preenche campos da estrutra ofnArqLigacoes, mas não apresenta a caixa de Salvar Arquivo, pois esse arquivo é gravado juntamente com o arquivo de especificação gráfica.

Preenche os campos da estrutura OPENFILENAME ofnArqLigacoes com os parâmetros de entrada:

.hwnOwner: hjan;

.lpstrFile: lpstrNomeArqLigacoes;

void AuxSalvarArqLoop (HWND hjan, LPSTR lpstrNomeArqLoop)

Entradas:

.hjan: "handle" da janela;

.lpstrNomeArqLoop: nome do arquivo.

Preenche campos da estrutra ofnArqLoop, mas não apresenta a caixa de Salvar Arquivo, pois esse arquivo é gravado juntamente com o arquivo de recursos.

Preenche os campos da estrutura OPENFILENAME ofnArqLigacoes com os parâmetros de entrada:

.hwnOwner: hjan;

.lpstrFile: lpstrNomeArqLoop;

5.3 Módulo LISTA.C

Este módulo contém as rotinas que manipulam a lista de símbolos gráficos na memória. Os símbolos que compõem a especificação gráfica do modelo de simulação são armazenados em uma lista encadeada, à medida em que são desenhados pelo usuário.

Neste módulo também são implementadas as rotinas de desenho dos centros de serviço (que representam recursos definidos no modelo de simulação). Tais rotinas

utilizam as primitivas da GDI ("Graphics Device Interface"), basicamente LineTo, MoveTo e Ellipse [PET93].

Também estão neste módulo as rotinas que verificam se o botão do "mouse" foi clicado sobre o símbolo que define um recurso, para identificar o início ou fim de uma ligação ou ramificação, ou então apagar um símbolo. Para apagar, um retângulo inválido que abrange o símbolo é determinado, para que somente essa região seja repintada (assim a imagem na janela de edição gráfica não parece "piscar").

A lista é mantida pelos "handles" hLLista e hLFimLista, que indicam respectivamente o primeiro e o último nó da mesma.

Os nós da lista de símbolos armazenam, para cada símbolo:

```
typedef struct lista_de_simbolos{
        int x,
            y;
        char tipo;
        int identrec;
        LOCALHANDLE prox;
} lista_simbolos;
```

.x, y: coordenadas x e y do canto superior esquerdo de um retângulo imaginário, que envolve o símbolo e serve para delimitar o espaço no caso de identificar um símbolo através do "mouse" (seção 3);

.tipo: o tipo do símbolo (definidos em Editor.h). A seção 4.1 apresentou os tipos possíveis;

.identrec: a identificação, que indica a ordem em que foram desenhados e vai ser utilizada para identificar o recurso na parametrização. É esse valor que preenche o campo ident_rec, da lista de recursos, quando o recurso é parametrizado;

.prox: um "handle" para o próximo nó da lista. A alocação dinâmica armazena "handles", ao invés de ponteiros, devido ao gerenciamento de memória feito pelo Windows.

Rotinas:

short InsereLista(HWND hjan, int x, int y, char tipo, int identrec)

Entradas:

.hjan: handle da janela;

.x, y: coordenadas do canto superior esquerdo do retângulo imaginário que envolve o desenho;

.tipo: tipo do símbolo;

.identrec: identificação.

Insere um símbolo na lista de símbolos desenhados pelo usuário, sempre no final da lista.

Retorno:

.1: inserção feita corretamente;

.0: erro ao acessar a lista.

short ApagaLista(HWND hjan)

Entradas:

.hjan: handle da janela.

Apaga da memória a lista de símbolos. Chamada quando o usuário encerra o sistema ou quando escolhe a opção Limpar no menu Editar.

Retorno:

.1: a lista foi apagada corretamente;

.0: erro ao apagar a lista.

BOOL TemSimbolo()

Retorna TRUE se a lista de símbolos possui algum elemento (isto é, se o usuário ja' desenhou algum símbolo) ou FALSE em caso contrário (o usuário não especificou nenhum símbolo do modelo).

int IdentRecSimbolo(HWND hjan, int x, int y, int *mx, int *my, char *tipo)

Entradas:

.hjan: "handle" da janela;

.x, y: posição definida com o "mouse".

Saídas:

.mx, my: posição onde vai ser desenhada a seta que aponta para o símbolo de um recurso selecionado;

.tipo: tipo do símbolo do recurso selecionado.

Percorre a lista de símbolos, chamando ClicRecurso para verificar se o ponto definido com o "mouse" está sobre algum símbolo de recurso. Em caso positivo, calcula o ponto para desenhar a seta que aponta para o símbolo.

Retorno:

.identificação do recurso sobre o qual o usuário pressionou o botão esquerdo do "mouse";

.0 caso contrário.

BOOL ClicRecurso(int x, int y, int xs, int ys, char tipo)

Entradas:

.x, y: ponto na janela de edição ser pesquisado;

.xs, ys: ponto superior esquerdo do símbolo de recurso sendo analisado;

.tipo: tipo do símbolo (REC1, REC2, REC3 OU REC4).

Verifica se o usuário pressionou o botão esquerdo do "mouse" (ponto (x,y)) sobre o desenho cujo canto superior esquerdo é o ponto (xs, ys) e se o mesmo é o símbolo de um recurso. O parâmetro tipo auxilia para a verificação, pois cada tipo tem diferentes extensões tanto no eixo x quanto no eixo y.

Retorno:

.TRUE: é um recurso;

.FALSE: não é um recurso.

short DesenhaSimbolos(HDC hdc, HWND hjan)

Entradas:

.hdc : handle do dispositivo de contexto;

.hjan: handle da janela.

Chama os procedimentos necessários para desenhar os símbolos da lista na janela-filha de edição gráfica. Utilizada no processamento da mensagem WM_PAINT, no procedimento de janela de edição gráfica (quando o tamanho da janela é alterado, ou quando a janela é criada).

Retorno:

.0: erro ao acessar a lista de símbolos;

.1: operação realizada corretamente.

short DesenhaRec(HDC hdc, HWND hjan, char tipodorecurso)

Entradas:

.hdc: "handle" do dispositivo de contexto;

.hjan" "handle" da janela;

.tipodorecurso: tipo do símbolo de recurso a ser desenhado.

Chama a função adequada para desenhar o símbolo do recurso, que está no fim da lista encadeada.

Retorno:

.0: erro ao acessar a lista;

.1: operação realizada corretamente.

void DesRec1(HDC hdc, int x, int y)

Entradas:

.hdc: handle do dispositivo de contexto;

.x,y: coordenadas em pontos da tela.

Desenha na janela de edição o símbolo correspondente a um centro de serviço constituído de uma fila e um servidor (REC1). Inicia o desenho na posição (x,y).

void DesRec2(HDC hdc, int x, int y)

Entradas:

.hdc: handle do dispositivo de contexto;

.x,y: coordenadas em pontos da tela.

Desenha na janela de edição o símbolo correspondente a um centro de serviço, constituído de uma fila e vários servidores (REC2). Inicia o desenho na posição (x,y).

void DesRec3(HDC hdc, int x, int y)

Entradas:

.hdc: handle do dispositivo de contexto;

.x,y: coordenadas em pontos da tela.

Desenha na janela de edição o símbolo correspondente a um centro de serviço, constituído de várias filas e um servidor (REC3). Inicia o desenho na posição (x,y).

void DesRec4(HDC hdc, int x, int y)

Entradas:

.hdc: handle do dispositivo de contexto;

.x,y: coordenadas em pontos da tela.

Desenha na janela de edição o símbolo correspondente a um centro de serviço, constituído de várias filas e vários servidores (REC4). Inicia o desenho na posição (x,y).

BOOL PtoInicial(HWND hjan, int mousex, int mousey, int *x, int *y, int *identrec)

Entradas:

.hjan: "handle" da janela

.mousex, mousey: coordenadas do ponto identificado pelo "mouse"

Saídas:

.x, y: coordenadas para desenhar o início da ligação;

.identrec: identificação do símbolo de recurso início da ligação.

Verifica se o botão do "mouse" foi pressionado sobre o símbolo de um recurso (pesquisa por todos os nós da lista de símbolos), e em caso afirmativo retorna em (x,y) o ponto de início para desenhar uma ligação e em identrec a identificação do recurso. É utilizada quando o usuário vai desenhar uma ligação entre dois símbolos de recursos.

Retorno:

.TRUE: o símbolo é de um recurso;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL PtoFinal(HWND hjan, int mousex, int mousey, int *x, int *y, int *identrec)

Entradas:

.hjan: "handle" da janela

.mousex, mousey: coordenadas do ponto identificado pelo "mouse"

Saídas:

.x, y: coordenadas para desenhar o fim da ligação;

.identrec: identificação do símbolo do recurso fim da ligação.

Verifica se o botão do "mouse" foi pressionado sobre o símbolo de um recurso (pesquisa todos os nós da lista de símbolos), e em caso afirmativo retorna em (x,y) o ponto final para desenhar uma ligação e em identrec a identificação do recurso. É

utilizada quando o usuário vai desenhar uma ligação entre dois símbolos de recursos.

Retorno:

.TRUE: o símbolo é um recurso;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL ApagaSimbolo(HWND hjan, int x, int y, int *x1, int *y1, int *x2, int *y2, BOOL bPrecisaApagarLig, int *isimboloapagado)

Entradas:

.hjan: "handle" da janela;

.x, y: ponto a ser verificado se está sobre um símbolo de recurso.

Saídas:

.x1, y1, x2, y2: coordenadas do retângulo inválido para apagar o símbolo;

.bPrecisaApagarLig: indica se precisa apagar ligação que chega ou sai do recurso apagado;

.isimboloapagado: identificação do recurso cujo símbolo foi apagado.

Apaga o símbolo do recurso indicado pelo usuário, através do "mouse" (verifica se (x,y) faz parte de um desenho). O nó correspondente é removido da lista, e o desenho apagado da janela de edição (obtendo as coordenadas do retângulo inválido, para que somente aquela região seja repintada). Também indica se precisa apagar ligação que chega ou sai do mesmo, e também retorna a identificação do recurso para retirar seus dados da lista de recursos e de ciclos.

Retorno:

.TRUE: o usuário indicou um símbolo que foi apagado;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL RemoveUltimoSimboloLista(HWND hjan)

Entrada:

.hjan: "handle" da janela.

Remove o último nó da lista de símbolos.

5.4 Módulo LISTREC.C

Este módulo contém as rotinas que manipulam a lista de parâmetros dos recursos. Os recursos definidos no modelo têm seus dados armazenados em uma lista encadeada, cuja definição se encontra no arquivo Tipos.h. Cada nó da lista possui os seguintes campos (cujo significado é o mesmo dos campos descritos na seção 2.2):

```
typedef struct lista_de_recursos{
        int ident_rec;
        char nome_recurso[10];
        int num_servidores,
          distribuição_serviço;
        char est_max_min,
           est_fila_vazia,
           media_servico[10],
           pto_triang_servico[10];
        int seq_num_aleat_servico,
           disciplina,
           escolha_fila,
           escolha_cliente,
           escolha_servidor,
           escolha_prox_rec,
           distribuição_chegada;
        char media_chegada[10],
           desvio_padrao_chegada[10],
           pto_triang_chegada[10];
        int seq_num_aleat_chegada;
        char entradas_saidas[41],
           sem_fila;
        unsigned int num_filas,
           flag_sinc;
           char libera_recurso;
        LOCALHANDLE prox;
} lista_recursos;
```

A lista é manipulada por meio de três apontadores, hListaRec (que mantém a cabeça da lista), hMeioListaRec (indica meio da lista) hFimListaRec (que mantém o último nó da lista). Como para as demais listas, os nós guardam "handles" e não ponteiros (o campo prox).

Rotinas:

short InsListaRec(HWND hjan, char nomerec[10], int numserv, char estmaxmin, char estfilavazia, int escproxrec, unsigned int numfilas, BOOL binsereposatual, int condicao)

Entradas:

.hjan: "handle" da janela;

.nomerec: nome do recurso;

.numserv: número de servidores do recurso;

.estmaxmin: opção do usuário: deseja ou não estatísticas sobre o comprimento da(s) fila(s) de espera;

.esfilavazia: opção do usuário: deseja ou não estatísticas sobre fila(s) vazia(s);

.escproxrec: identifica o modo pelo qual o próximo recurso será acessado;

.numfilas: número de filas;

.binsereposatual: indica que a parametrização está sendo alterada, deve inserir no nó que existe;

.condicao: indica onde deve inserir (cabeça, meio ou fim).

Insere os dados de identificação do recurso na lista de recursos. Das funções de inclusão nesta lista, é a única que cria um novo nó (se binsereposatual for FALSE, senão insere em nó já alocado). As demais sempre inserem dados de um nó já existente (já alocado), isto é, de um recurso já identificado. A inserção ocorre na cabeça, se for a entrada do modelo; no fim, se for saída, e no meio, se for outro (após a cabeça, depois da última inserção).

Retorno:

.0: erro ao acessar a lista;

.1: operação realizada corretamente.

short InsListaRecChegada(HWND hjan, int dist, char media[10], char dpadrao[10], int semente, char ptotriang[10], BOOL binsereposatual, int identrec)

Entradas:

.hjan: "handle" da janela;

.dist: distribuição do tempo entre-chegadas;

.media: média do tempo entre-chegadas;

.dpadrao: desvio-padrão do tempo entre-chegadas;

.semente: sequência a ser utilizada para a geração do tempo da próxima chegada ao sistema;

.char ptotriang: ponto de maior probabilidade de ocorrência da distribuição triangular;

.binsereposatual: indica se deve inserir em nó de recurso que está sendo alterado;

.identrec: identificação do recurso.

Insere os dados da chegada na lista de recursos. Caso binsereposatual seja TRUE, procura o nó correto para inserção, senão utiliza o nó corrente, isto é, identificado quando da chamada mais recente a InsListaRec.

Retorno:

.0: inserção efetuada corretamente;

.1: erro ao acessar a lista.

short InsListaRecServico(HWND hjan, int dist, char media[10], char dpadrao[10], int semente, char ptotriang[10], BOOL binsereposatual, int identrec)

Entradas:

.hjan: handle da janela;

.dist: distribuição do tempo de serviço;

.media: media do tempo de serviço;

.dpadrao: desvio-padrão do tempo de serviço;

.semente: semente a ser utilizada para a geração do tempo de serviço;

.ptotriang: ponto de maior probabilidade de ocorrência da distribuição triangular;

.binsereposatual: indica se deve inserir em nó de recurso que está sendo alterado;

.identrec: identificação do recurso.

Insere os dados do serviço na lista de recursos. Caso binsereposatual seja TRUE, procura o nó correto para inserção, senão utiliza o nó corrente, isto é, identificado quando da chamada mais recente a InsListaRec.

Retorno:

.0: inserção efetuada corretamente;

.1: erro ao acessar a lista.

short InsListaRecFila(HWND hjan, int discfila, int escfila, int esccliente, int escservidor, BOOL binsererposatual, int identrec)

Entradas:

.hjan: handle da janela;

.discfila: disciplina da fila;

.esccliente: algoritmo para o cliente escolher em qual fila irá entrar (quando existe mais de uma fila);

.escfila: algoritmo para escolha de qual fila retirar o próximo cliente a ser atendido (no caso de recurso-centro de serviço-com mais de uma fila);

.escservidor: algoritmo para escolha de um servidor quando mais de um está disponível;

.binsereposatual: indica se deve inserir em nó de recurso que está sendo alterado;

.identrec: identificação do recurso.

Insere os dados da(s) fila(s) na lista de recursos. Caso binsererposatual seja TRUE procura o nó correto para inserção, senão utiliza o nó corrente, isto é, identificado quando da chamada mais recente a InsListaRec.

Retorno:

.0: erro ao acessar a lista;

.1: inserção efetuada corretamente.

short InsListaRecRealimentacao(HWND hjan, char entradasaida[41], BOOL binsereposatual, int identrec)

Entradas:

.hjan: handle da janela;

.entradasaida: descreve as ligações e ramificações do modelo, assim como as probabilidades para escolha do próximo recurso;

.binsereposatual: indica se deve inserir em nó de recurso que está sendo alterado;

.identrec: identificação do recurso.

Insere o dado das ligações e ramificações na lista de recursos. Caso binsereposatual seja TRUE, procura a posição correta para inserção, senão utiliza o nó corrente, isto é, identificado quando da chamada mais recente a InsListaRec.

Retorno:

.0: erro ao acessar a lista;

.1: inserção efetuada corretamente.

BOOL TemListaRecursos()

Retorna TRUE se algum recurso foi parametrizado, FALSE caso contrário.

short ApagaListaRecursos(HWND hjan);

Entradas:

.hjan: handle da janela.

Apaga a lista de dados dos recursos da memória. Chamada quando o usuário encerr ao EdGraf, cancela toda a parametrização efetuada, ou escolhe os ítens Limpar (menun Sair) ou Nova (menu Especificação Gráfica).

Retorno:

.0: erro ao acessar a lista;

.1: operação efetuada corretamente.

BOOL CancelaParRecAtual(HWND hjan, int identrec)

Entradas:

.hjan: "handle" da janela;

.identrec: identificação do recurso.

Cancela a parametrização do recurso selecionado, removendo o último nó da lista de recursos.

Retorno:

.TRUE: operação realizada corretamente;

.FALSE: caso contrário, erro ao acessar a lista.

BOOL RecursoJaParametrizado(HWND hjan, int identrec)

Entradas:

.hjan: "handle" da janela;

.identrec: identificação do recurso.

Verifica se o recurso foi ou não parametrizado.

Retorno:

.TRUE: caso afirmativo;

.FALSE: caso contrário.

BOOL FaltaRecursoParaParametrizar(HWND hjan)

Entrada:

.hjan: "handle" da janela.

Verifica se o usuário parametrizou todos os recursos definidos.

Retorno:

.TRUE: todos os recursos foram parametrizados;

.FALSE: caso contrário, ou erro ao acessar a lista.

5.5 Módulo LISTALIG.C

Este módulo contém as rotinas que manipulam a lista encadeada que armazena os dados das ligações e ramificações definidas entre os símbolos que representam recursos do modelo.

Como as demais listas, os "handles" hListaLig e hFimListaLig marcam, respectivamente, o primeiro e o último elemento da lista.

Cada nó da lista apresenta a estrutura:

```
typedef struct lista_de_ligacoes{
        int recurso1,
            recurso2,
            x1, y1,
            x2, y2;
        char tipo;
        LOCALHANDLE prox;
} lista_ligacoes;
```

.recurso1: identificação do recurso inicial;

.recurso2: identificação do recurso final;

.x1, y1: ponto inicial da ligação ou ramificação;

.x2, y2: ponto final da ligação ou ramificação;

.tipo: indica se é ligação (LIGACAO) ou ramificação (RAMIFICACAO);

.prox: "handle para o próximo nó da lista

Rotinas:

short InsListaLig(HWND hjan, int rec1, int rec rec2, int x1, int y1, int x2, int y2, char tipo)

Entradas:

.hjan: "handle" da janela;

.rec1: recurso inicial da ligação;

.rec2: recurso final da ligação;

.x1, y1: coordenadas do ponto inicial para desenhar a ligação ou ramificação;

.x2, y2: coordenadas do ponto final da ligação ou ramificação;

.tipo: identifica se é uma ramificação ou ligação.

Armazena os dados referentes a ligação ou ramificação na lista encadeada. A inserção é sempre feita no final da lista.

Retorno:

.0: Erro ao acessar a lista;

.1: Operação realizada corretamente.

short ApagaListaLigacoes(HWND hjan)

Entradas:

.hjan: "handle" da janela.

Remove da memória a lista encadeada.

Retorno:

.0: erro ao acessar a lista;

.1: operação realizada corretamente.

BOOL TemLigacoes()

Retorna TRUE se o usuário definiu ligação ou ramificação entre símbolos, FALSE caso contrário.

void InicCondicoesLigacao(int vet[6])

Entradas:

.vet: armazena os pontos inicial e final da ligação, e os dois recursos unidos.

Inicia variáveis (definidas em Eddef.h) necessárias à captação dos pontos necessários para desenhar uma ligação entre dois recursos. Indica que o usuário deve indicar dois recursos definidos no modelo.

void DesInicCondicoesLigacao(int vet[6])

Entradas:

.vet: armazena os pontos inicial e final da ligação, e os dois recursos unidos.

Coloca valores inválidos nas variáveis (definidas em Eddef.h) necessárias à captação dos pontos necessários para desenhar uma ligação entre dois recursos. Utilizada depois que o EdGraf captou os pontos.

short DesenhaLigacoes(HDC hdc, HWND hjan)

Entradas:

.hdc: "handle" do dispositivo de contexto;

.hjan: "handle" da janela.

Percorre a lista encadeada, chamando as rotinas para desenhar todas as ligações e ramificações.

Retorno:

.0: Erro ao acessar a lista;

.1: Operação realizada corretamente.

short DesLigacao(HDC hdc, HWND hjan)

Entradas:

.hdc: "handle" do dispositivo de contexto;

.hjan: "handle" da janela.

Chama a rotina para desenhar a ligação ou raificação cuja entrada está armazenada na última posição da lista.

Retorno:

.0: Erro ao acessar a lista;

.1: Operação realizada corretamente.

void DesLig(HDC hdc, int x1, int y1, int x2, int y2)

Entradas:

.hdc: "handle" do dispositivo de contexto;

.x1, y1: ponto inicial da ligação ou ramificação;

.x2, y2: ponto final.

Chama as primitvas da GDI para desenhar uma linha entre os pontos (x1,y1) e (x2,y2). Verifica todas as combinações: x1 < = x2 ou x1 > x2, e y1 < -y2 ou y1 > y2. Também desenha uma pequena seta que aponta para o recurso final da ligação ou ramificação. O desenho não é inteligente, isto é, não verifica se vai passar sobre outro desenho. Sugere-se ao usuário deixar os símbolos bem espaçados uns dos outros.

void DesRealimentacao(HDC hdc, int x, int y)

Entradas:

.hdc: "handle" do dispositivo de contexto;

.x,y: ponto que identifica o início da ramificação.

Desenha uma pequena oval no ponto inicial da ramificação, que serve para distinguir das ligações. Utiliza a primitiva Ellipse, da GDI.

BOOL ObtemRecursoLigado(HWND hjan, int rec1, int *rec2)

Entradas:

.hjan: "handle" da janela;

.rec1: recurso sendo pesquisado.

Saídas:

.rec2: identificação do recurso ao qual rec1 está ligado.

Pesquisa a lista de ligações para verificar se o recurso identificado por rec1 está ligado a algum outro (EdGraf permite que somente uma ligação saia de um recurso, as demais devem ser ramificações). Em caso positivo, rec2 retorna a identificação desse outro.

Retorno:

.TRUE: está ligado a outro;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL PontoEmLigacao(HWND hjan, int mousex, int mousey, int *x, int *y, int *rec1)

Entradas:

.hjan: "handle" da janela;

.mousex, mousey: ponto indicado pelo "mouse".

Saídas:

.x, y: ponto para desenhar a elipse que indica ramificação;

.rec1: recurso originário da ramificação.

Pesquisa a lista de ligações para verificar se o ponto indicado pelo "mouse" está sobre uma ligação entre dois recursos (chama a função PertenceLigacao). Em caso afirmativo o ponto (x,y) armazena as coordenadas corretas para desenhar a elipse que indica ramificação, e rec1 indica o símbolo do recurso em questão.

Retorno:

.TRUE: o ponto está sobre uma ligação;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL PertenceLigacao(int mousex, int mousey, int x1, int y1, int x2, int y2, int *posx, int *posy)

Entradas:

.mousex, mousey: coordenadas indicadas pelo "mouse";

.x1, y1: coordenadas do ponto inicial da ligação em estudo;

.x2, y2: coordenadas do ponto final da ligação em estudo.

Saídas:

.posx, posy: coordenadas do ponto para desenhar a elipse que indica ramificação e o seu início.

Verifica se o usuário pressionou o botão esquerdo do "mouse" sobre a ligação do ponto (x1,y1) até (x2,y2). Verifica se mousey pertence ao intervalo [y1 - 4,y1 + 4].

Retorno:

.TRUE: se é ligação;

.FALSE: caso contrário.

BOOL PertenceRamificação(int mousex, int mousey, int x1, int y1)

Entradas:

.mousex, mousey: coordenadas do ponto indicado pelo "mouse";

.x1, y1: coordenadas iniciais da ramificação em estudo.

Verifica se o usuário pressionou o botão do "mouse" sobre a elipse da ramificação que inicia no ponto (x1, y1).

Retorno:

.TRUE: caso seja ramificação;

.FALSE: caso contrário.

BOOL RecursoTemRealimentacao(HWND hjan, int idrec)

Entradas:

.hjan: "handle" da janela;

.idrec: identificação do recurso pesquisado.

Pesquisa a lista de ligações, para ver se o recurso idrec tem ramificação.

Retorno:

.TRUE: tem ramificação;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL ObtemRecursoComRealimentacao(HWND hjan, int idrec1, int *contador, int *idrec2)

Entradas:

.hjan: "handle" da janela;

.idrec1: identificação do recurso sendo pesquisado.

Saídas:

.contador: identifica a i-ésima ramificação pesquisada;

.idrec2: identificação do recurso ligado a idrec1.

Pesquisa a lista de ligações para encontrar recursos aos quais idrec1 está ligado, através de ramificação. Busca sempre a partir da i-ésima ramificação, identificada por contador.

Retorno:

.TRUE: idrec1 é ligado a outro, através de ramificação;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL ApagaLigacao(HWND hjan, int x, int y)

Entradas:

.hjan: "handle" da janela;

.x, y: coordenada indicada pelo "mouse".

Percorre a lista de ligações, para verificar se o ponto (x,y) pertence a uma ligação (PertenceLigacao). Em caso afirmativo, remove o nó correspondente da lista.

Retorno:

.TRUE: o ponto indica uma ligação que foi apagada;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL ApagaUmaLigacao(HWND hjan, int x, int y)

Entradas:

.hjan: "handle" da janela;

.x, y: ponto indicado pelo usuário.

Apaga a ligação indicada pelo usuário, removendo o nó da lista de ligações.

Retorno:

.TRUE: ligação apagada;

.FALSE: o ponto não estava sobre uma ligação, ou erro ao acessar a lista.

BOOL ApagaRamificacao(HWND hjan, int x, int y)

Entradas:

.hjan: "handle" da janela;

.x, y: coordenada indicada pelo "mouse".

Como a ApagaLigacao, utiliza a rotina PertenceRamificacao.

Retorno:

.TRUE: o ponto indica uma ligação que foi apagada;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL JaTemLigacao(HWND hjan, int rec1, int rec2, char tipo)

Entradas:

.hjan: "handle" da janela;

.rec1: recurso inicial da ligacão;

.rec2: recurso final da ligação;

.tipo: tipo da ligação: LIGACAO, RAMIFICACAO, INICIO OU FIM.

Percorre a lista de ligações para verificar se rec1 tem ligação ou ramificação para rec2, ou ainda se rec1 já tem uma seta de início ou de fim. Usado para não permitir a definição de mais de uma união entre dois recursos, ou setas de início ou fim.

Retorno:

.TRUE: ligação repetida;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL NaoPodeSerInicio(HWND hjan, int rec)

Entradas:

.hjan: "handle" da janela;

.rec: identificação do recurso.

Verifica se rec é o fim de alguma ligação vinda de outro recurso do modelo, ou início. Nesse caso, rec não pode ser o fim da seta de início.

Retorno:

.TRUE: em caso afirmativo;

.FALSE: caso contrário.

BOOL NaoPodeSerFim(HWND hjan, int rec)

Entradas:

.hjan: "handle" da janela;

.rec: identificação do recurso.

Verifica se rec é o início de alguma ligação para outro recurso do modelo, ou para fim. Nesse caso, rec não pode ser a origem da seta de fim.

Retorno:

.TRUE: em caso afirmativo;

.FALSE: caso contrário.

BOOL TemMais1Ciclo1Ligacao(int identrec)

Entrada:

.identrec: identificação do recurso.

Verifica se o recurso tem ramificação ou mais de um ciclo para outro, nesse caso o botão Ciclo de Escolha do Próximo Recurso (caixa Dados do Modelo) não pode ser selecionado.

Retorno:

.TRUE: tem mais de um ciclo ligação;

.FALSE: caso contrário.

BOOL JaTemInicio(HWND hjan)

Entrada:

.hjan: "handle" da janela.

Verifica se o usuário já definiu a entrada do modelo.

Retorno:

.TRUE: em caso afirmativo;

.FALSE: caso contrário.

BOOL JaTemFim(HWND hjan)

Entrada:

.hjan: "handle" da janela.

Verifica se o usuário já definiu ao menos uma saída do modelo.

Retorno:

.TRUE: em caso afirmativo;

.FALSE: caso contrário.

BOOL TodosRecTemLigParaOutro(HWND hjan, int id_comp)

Entradas:

.hjan: "handle" da janela;

.id_comp: usado para comparação.

Verifica, para cada recurso definido, tem sem entrada ou saída definida para outro (dependendo do valor de id_comp).

Retorno:

.TRUE: falta definir chegada ou saída de recurso;

.FALSE: caso contrário, ou erro ao acessar a lista.

BOOL JaTem10Saidas(HWND hjan, int idrec)

Entradas:

.hjan: "handle" da janela;

.icrec: identificação do recurso.

Verifica se já foram definidas 10 saídas (entre ligação e ramificações) a partir do recurso.

Retorno:

.TRUE: em caso afirmativo;

.FALSE: caso contrário.

5.6 Módulo LISTDIST.C

Este módulo implementa as rotinas que manipulam a lista encadeada que armazena os dados da distribuição tabelada fornecida pelo usuário, já transformados para distribuição de frequência. Os dados são mantidos temporariamente em memória, até que sejam gravados em arquivo.

Os "handles" hListaDist e hFimListaDist são definidos nesse módulo, e apontam respectivamente para o início e fim da lista encadeada.

Ressalta-se que EdGraf somente permite 20 entradas, no máximo, para cada distribuição, pois utiliza um vetor estático para armazenar os valores digitados pelo usuário e assim obter a distribuição de frequência.

Cada nó da lista apresenta os campos:

```
typedef struct lista_de_dist_tab{
        char valor[5];
        char prob[6];
        LOCALHANDLE prox;
} lista_dist_tab;
```

.valor: valor cujo número de ocorrências foi observado;

.prob: número de ocorrêencias deste valor.

.prox: "handle" para o próximo nó da lista.

Rotinas:

short InsListaDist (HWND hjan, char valor[5], char prob[6])

Entradas:

.hjan: "handle" da janela;.valor: valor de ocorrência;

.prob: frequência da ocorrência dentre o total de ocorrências.

Insere os dados na lista de distribuição tabelada. A inserção é feita sempre no final da lista.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

short MostraListaDist (HWND hjan)

Entradas:

.hjan: "handle" da janela

Percorre a lista encadeada, apresentando para cada nó uma caixa de mensagem (rotina CxMostraDist) que mostra os valores da distribuição tabelada. As caixas são apresentadas uma após a outra, até o fim da lista.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

short ApagaListaDist (HWND hjan)

Entradas:

.hjan: "handle" da janela

Remove a lista de distribuição tabelada da memória.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

BOOL TemListaDist ()

Verifica se existe distribuição tabelada digitada pelo usuário, ou seja, se o "handle" inicial da lista é diferente de nulo.

Retorno:

.TRUE: em caso afirmativo

.FALSE: caso contrario

void CxMostraDist (char *szTitulo, char *szFormato, char valor[5], char prob[6])

Entradas:

. szTitulo: título da caixa de mensagem;

.szFormato: formato para apresenta os dados;

.valor[5]: valor de ocorrência a ser apresentado;

.prob[6]: frequência da ocorrência a ser apresentada na caixa.

Apresenta uma caixa de mensagem contendo os dados valor e prob, no formato especificado por szFormato. Utilizado pela rotina MostraListaDist.

5.7 Módulo LISTEVEN.C

Este módulo contém as rotinas que manipulam a lista encadeada que armazena os dados relativos a eventos definidos pelo usuário. A lista é mantida em memória até que o usuário inicie um novo conjunto de dados, quando os eventos são gravados em arquivo ou descartados.

Os "handles" hListaEventos e hFimListaEventos são definidos neste módulo e indicam, respectivamente, para o início e fim da lista encadeada.

Cada nó da lista apresenta os campos:

```
typedef struct lista_de_temp_eventos{
        int num_recurso;
```

```
    char tempo_ocorrencia[9];
    LOCALHANDLE prox;
} lista_temp_eventos;
```

.num_recurso: número do recurso para o qual um evento deve ser escalonado, como a chegada de um cliente ou a utilização deste recurso;

.tempo_ocorrencia: tempo para o qual o evento especificado será escalonado.

.prox: "handle" para o próximo nó da lista.

Rotinas:

short InsListaEventos (HWND hjan, int numrecurso, char tempoocorrencia[9])

Entradas:

.hjan: "handle" da janela;

.numrecurso: identificação do recurso ao qual está relacionado o evento chegada;

.tempoocorrencia: tempo de ocorrência do evento.

Insere na lista os dados relativos ao evento chegada definido. A inserção é feita sempre no final da lista.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

short MostraListaEventos (HWND hjan)

Entradas:

.hjan: "handle" da janela

Percorre a lista encadeada, apresentando para cada nó uma caixa de mensagem (rotina CxMostraEventos) que mostra os valores dos eventos definidos. As caixas são apresentadas uma após a outra, até o fim da lista.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

short ApagaListaEventos (HWND hjan)

Entradas:

.hjan: "handle" da janela

Apaga a lista de eventos da memória.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

void CxMostraEventos (char *szTitulo, char *szFormato, int numrecurso, char tempoocorrencia[9])

Entradas:

.szTitulo: título da caixa de mensagem;

.szFormato: formato para apresenta os dados;

.numrecurso: identificação do recurso;

.tempoocorrencia: tempo de ocorrência do evento.

Apresenta uma caixa de mensagem contendo os dados nomerecurso e tempoocorrencia, no formato especificado por szFormato. Utilizado pela rotina MostraListaEventos.

5.8 Módulo DESENHA.C

Este módulo contém as rotinas responsáveis pelo desenho dos ícones na janela de ícones, e funções para alteração do cursor quando o botão esquerdo do "mouse" é pressionado e liberado (quando o usuário escolhe um ícone, o cursor muda do formato

de uma seta para uma cruz, e quando o botão é liberado, o cursor volta ao formato de seta).

Também contém a rotina que desenha o ícone de uma seta, quando um recurso é selecionado para parametrização ou definir eventos.

Rotinas:

HCURSOR MudaCursor(HWND hjan)

Entradas:

.hjan: "handle" da janela.

Muda o formato atual do cursor (seta) para o formato de uma cruz, com as funções LoadCursor, SetCursor e ShowCursor [PET93].

Retorno:

."handle" do cursor anterior.

void VoltaCursor(HCURSOR hcrVelho)

Entradas:

.hcrVelho: "handle" do cursor anterior.

Retorna o formato do cursor para o formato indicado pelo "handle" hcrVelho.

short DeterminaIcone(int y, short cyIcone)

Entradas:

.y: reflete a coordenada, no eixo y, da posição em que o botão esquerdo do "mouse" foi pressionado (em relação à janela de ícones);

.cyIcone: dimensão (no eixo y) de um ícone.

Identifica em qual ícone o usuário pressionou o botão esquerdo do "mouse".

Retorno:

.a identificação do ícone escolhido.

void DesenhaIcones(HDC hdc, HICON hIcone[NUMICONES], short cxIcone, short cyIcone)

Entradas:

.hdc: "handle" do dispositivo de contexto;

.hIcone: vetor que armazena os "handles" dos ícones da janela de ícones;

.cxIcone: dimensão (no eixo x) de um ícone;

cyIcone: dimensão (no eixo y) de um ícone;

Desenha os ícones da barra de ferramentas na janela de ícones, utilizando as funções DrawIcon, MoveTo e LineTo da GDI [PET93].

void DesenhaIconeMostra(HDC hdc, int x, int y, HICON hIconeMostra)

Entradas:

.hdc: "handle" do dispositivo de contexto;

.x: posição no eixo x onde o ícone será desenhado;

.y: posição no eixo y onde o ícone será desenhado;

.hIconeMostra: "handle" do ícone que mostra o recurso selecionado

Desenha na janela de edição gráfica o ícone de uma seta, que aponta para o recurso selecionado para parametrização ou definição de um evento.

5.9 Módulo INFOREC.C

Este módulo contém as rotinas que retornam informações a respeito da parametrização do recurso selecionado.

Rotinas:

void ParametrizouChegada()

Indica que os parâmetros relacionados ao tempo entre-chegadas do recurso selecionado foram fornecidos.

void ParametrizouFila()

Indica que os parâmetros relacionados à(s) fila(s) do recurso selecionado foram fornecidos.

void ParametrizouServico()

Indica que os parâmetros relacionados ao tempo de serviço do recurso selecioando foram fornecidos.

BOOL JaParametrizouRecurso()

Caso as informações referentes à chegada, fila e serviço já tenham sido fornecidas, indica que o recurso foi parametrizado, isto é, não faltam informações. Essa rotina indica que pode ser apagada a seta que indica o recurso, depois que o usuário termina de parametrizá-lo.

void TemQueParametrizarRecurso()

Indica que todos os parâmetros para um determinado recurso devem ser fornecidos.

5.10 Módulo CONSIST.C

Este módulo contém as rotinas que verificam a consistência dos parâmetros fornecidos pelo usuário. Para todas as funções de consistência, quando ocorre algum erro ou o usuário deixa de fornecer algum dado, é apresentada uma caixa de mensagem que avisa ao usuário do erro (as mensagens estão definidas em Menserro.h).

Ressalta-se que todos os dados numéricos são lidos como cadeias de caracteres, para verificar sua consistência. Caso estejam corretos, são convertidos para números inteiros ou reais, para serem inseridos na lista de recursos e posteriormente gravados em arquivo.

Para cada caixa de diálogos que aceita dados do usuário, uma rotina faz a verificação da consistência.

Rotinas:

BOOL VerificaConsistenciaModelo(char nomemod[30], char tempoexe[10], char trace[2], char relat_final[2], char nomearqsaida[13], char primchegada[10], char interface[2], char num_max_entidades[6], char num_voltas[6], char warmup[10], char tamanho_batch[6], char valores_deletados[6], char disttabelada[6], char szmenserro[80])

.nomemod: nome do modelo;

.tempoexe: tempo de execução da simulação. Deve ser um número real da forma x.y;

.trace: opção do usuário para "trace" ou não. Deve ser 0, 1, 2, 3, 4 ou identificador de distribuição definida pelo usuário;

.relat_final: opção do usuário para relatório final ou não. Deve ser 's' ou 'n';

.nomearqsaida: nome do arquivo do programa de simulação, em SMPL;

.primchegada: tempo da primeira chegada ao sistema de simulação. Deve ser um número real da forma x.y;

.interface: ativa ou não a interface mtr do SMPL. Deve ser 's' ou 'n';

.num_max_entidades: número máximo de entidades no sistema. Deve ser um número inteiro;

.num_voltas: número de voltas em um sistema fechado. Deve ser um número inteiro;

.warmup: indica o tempo para "warmup";

.tamanho_batch: tamanho do lote, para utilizar o métdod "Batch Means";

.valores_deletados: indica o número de valores desprezados, para começar a utilizar "Batch Means";

.disttabelada: número de distribuições tabeladas a serem utilizadas;

szmenserro: armazena a mensagem de erro de consistência, se houver.

Verifica a consistência de todos os dados fornecidos através da opção Parametrizar, no menu Modelo. O usuário deve preencher todos os campos, senão uma caixa de mensagem o avisará do campo que ficou em branco e da necessidade de preenchê-lo. Só aceita os dados depois de todos estarem corretos.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaRec(char nomerecurso[10], char numservidores[6], char estmaxmin[2], char estfilavazia[2], char escproxrec[2], char numfilas[6], char szmenserro[80], int idrec, BOOL EModeloFechado, BOOL TemEsc2)

.nomerecurso: nome do recurso;

.numservidores: número de servidores do recurso. Deve ser um número inteiro;

.estmaxmin: opção do usuário: deseja ou não estatísticas de comprimento de filas. Deve ser 's' ou 'n';

.estfilavazia: opção do usuário: deseja ou não estatísticas de fila vazia. Deve ser 's' ou 'n';

.escproxrec: indica a escolha para o próximo recurso;

.numfilas: número de filas do recurso;

.szmenserro: armazena a mensagem de erro de consistência, se houver;

.idrec: identificação do recurso;

.EModeloFechado: indica se o modelo é fechado;

.TemEsc2: indica se já tem recurso com Escolha Proximo Recurso definido como Fechado.

Verifica a consistência dos dados que identificam um recurso, obtidos através da opção Parametrizar (Recurso), do menu Recursos. Todos os campos devem ser preenchidos, e uma caixa de mensagem avisa quando houver campo em branco. Só aceita os dados depois de todos estarem corretos. Caso Escolha do Prúximo Recurso

seja definido como Ciclo, e o recurso tem saída para outros, apresenta mensagem de erro. Também não permite escolher Fechado para modelos que não são deste tipo.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaRecChegada(char distribuicao[3], char media[10], char desviopadrao[10], char semente., char ptotriang[10], unsigned int numdist, char szmenserro[80], BOOL bdistusuario)

Entradas:

.distribuicao: distribuição de probabilidade para o tempo entre- chegadas;

.media: tempo médio entre-chegadas. Deve ser um número real da forma x.y;

.desviopadrao: desvio padrão do tempo entre-chegadas. Deve ser um número real da forma x.y;

.semente: sequência para geração de números aleatórios;

.ptotriang: ponto mais provável da distribuição triangular, se essa for utilizada;

.numdist: número de distribuições definidas pelo usuário, permite fazer a consistência do número da distribuição escolhida;

.bdistusuario: indica se o usuário indicou número de distribuições definidas a utilizar na parametrização do modelo.

Saídas:

.szmenserro: armazena mensagem de erro de consistência, se houver.

Verifica a consistência dos dados referentes à chegada de novos clientes em um recurso, através do item Chegada do submenu Parametrizar, menu Recursos. Todos os campos devem ser preenchidos, e uma caixa de mensagem avisa quando houver campo em branco. Só aceita os dados depois de todos estarem corretos.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaRecServico(char distribuicao[3], char media[10], char desviopadrao[10], char semente[3], char ptotriang[10], unsigned int numdist, char szmenserro[80], BOOL bdistusuario)

Como em VerificaConsistenciaRecChegada.

BOOL VerificaConsistenciaRecFila(char disciplina[2], char escfila[2], char esccliente[2], char escservidor[2], char szmenserro[80])

Entradas:

.disciplina: disciplina da fila;

.escfila: algoritmo de escolha de qual fila retirar próximo cliente;

.esccliente: algoritmo para o cliente escolher em qual fila entrar;

.escservidor: algoritmo para escolher servidor.

Saídas:

.szmenserro: armazena mensagem de erro de consistência, se houver.

Verifica a consistência dos parâmetros da caixa Fila, apresentada pelo item Fila do submenu Parametrizar, menu Recursos.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaRecRealimentacao(char porcentagem[3], char szmenserro[80])

Entradas:

.porcentagem: porcentagem da ramificacao.

Saídas:

.szmenserro: armazena mensagem de erro de consistência, se houver.

Faz a consistência do dado da caixa Ramificação, apresentada depois da caixa Dados do Recurso, quando Escolha do Próximo Recurso tem o botão Probabilidade selecionado.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaLigacaoPrincipal(char porligprincipal, char szmenserro[80])

Entradas:

.porcligprncipal: porcentagem da ligação.

Saídas:

.szmenserro: armazena mensagem de erro de consistência, se houver.

Verifica a consistência do dado da caixa Ligação Principal, apresentada depois da caixa Dados do Recurso, quando Escolha do Próximo Recurso tem o botão Probabilidade selecionado.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaLoop(char nloop[6], char szmenserro[80])

Entradas:

.nloop: número de voltas, num ciclo.

Saídas:

.szmenserro: armazena mensagem de erro de consistência, se houver.

Verifica a consistência do dado da caixa Ciclo, apresentada quando Escolha do Próximo Recurso tem o botão Ciclo selecionado.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaDistribuicao(char auxvalor[5], char auxprob[6], char szmenserro[80])

Entradas:

.auxvalor: valor de ocorrência;

.auxprob: número de ocorrências.

Saídas:

.szmenserro: armazena mensagem de erro de consistência, se houver.

Faz a consistência dos dados da caixa Distribuição Tabelada.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaConsistenciaEventos(char tempoocor[9], char szmenserro[80])

Entradas:

.tempoocor: tempo de ocorrência do evento.

Saídas:

.szmenserro: armazena mensagem de erro de consistência, se houver.

Verifica a consistência do dado da caixa Evento.

Retorno:

.TRUE: todos os dados estão corretos;

.FALSE: dado incorreto ou em branco.

BOOL VerificaReal(char str[])

Entradas:

.str: cadeia de caracteres a ser verificada.

Verifica se uma cadeia de caracteres corresponde a uma representação correta de um número real, da forma x.y.

Retorno:

.TRUE: a cadeia representa um número real;

.FALSE: a cadeia não representa um número real.

BOOL VerificaInteiro(char str[])

Entradas:

.str: cadeia de caracteres a ser verificada.

Verifica se uma cadeia de caracteres corresponde a uma representação correta de um número inteiro.

Retorno:

.TRUE: a cadeia representa um número inteiro;

.FALSE: a cadeia não representa um número inteiro.

5.11 Módulo AUXILIAR.C

Esse módulo contém as rotinas para iniciar as variáveis e estruturas globais utilizadas pelo EdGraf (definidas em Eddef.h).

Rotinas:

void InicializaDadosModelo()

Inicia todas as variáveis utilizadas para leitura e verificacão de consistência dos dados gerais do modelo, fornecidos pelo usuário.

void InicializaDadosRecursos()

Inicia todas as variáveis utilizadas para leitura e verificação de consistência dos parâmetros fornecidos pelo usuário.

void InicializaDadosLoop()

Inicia todas as variáveis utilizadas para leitura e verificação de consistência dos dados relativos a ciclos definidos no modelo.

void InicializaDadosDistribuição()

Inicia todas as variáveis utilizadas para leitura e verificação de consistência dos dados relativos a distribuição tabelada, fornecida pelo usuário.

void InicializaDadosEventos()

Inicia todas as variáveis utilizadas para leitura e verificação de consistência dos dados relativos a eventos definidos para recursos do modelo sendo simulado.

void MontaStringEntradaSaida(char porcentagem[3], int *porcant, int *vez, int rec, char auxentradasaida[5])

Entradas:

.porcentagem: porcentagem digitada pelo usuário;

.rec: recurso final da ramificação;

Saídas:

.porcant: guarda o primeiro dígito da porcentagem anterior;

.vez: utilizado para marcar se é a primeira porcentagem fornecida ou não. Necessário pois o Gerador entende que a primeira porcentgem inicia com

1, as demais com a anterior;

.auxentradasaidaa: string montada para ser concatenada a rRecursos.EntradaSaida;

Analisa a porcentagem digitada pelo usuário para ser concatenada a rRecursos.EntradaSaida, como necessário ao Gerador de Aplicação: os dois primeiros caracteres receber o número do próximo recurso, os dois seguintes recebem a faixa de porcentagem e o último recebe o caracter nulo.a

void MontaStringEntraSaiSemRealimentação(int rec, char auxentradasaida[5])

Entradas:

.rec: recurso final da ligação.

Saídas:

.auxentradasaida: string montada para ser concatenada a rRecursos.EntradaSaida.

Monta a string auxentradasaida para ser concatenada a rRecursos.EntradaSaida, como necessário ao Gerador de Aplicação. utilizada quando o recurso não tem ramificação, ou seja, é ligado a apenas um recurso. A porcentagem, nesse caso, é igual a 1 (vai para o próximo recurso com uma porcentagem de 100%). Os dois primeiro caracteres recebem o número do próximo recurso, os dois seguintes recebem 1 e 0, e o último o caracter nulo.

void MontaStringEntraSai1(char parametro[3], int rec, char auxenmtradasaida[5])

Entradas:

.parametro[3]: indica se é ramificação ou ligação principal;

.int rec: identificação do próximo recurso.

Monta a string para ser concatenada a rRecursos.EntradaSaida. Utilizada quando Escolha do Próximo Recurso é igual a 1. Os dois primeiros caracteres recebem o número do próximo recurso, os dois seguintes indicam se é ramificação ou

ligação, e o último recebe o caracter nulo.

Saídas:

.auxentradasaida: string montada para ser concatenada a rRecursos.EntradaSaida.

void ConverteExtNomeArq(char ext[5], char *szNomeArq1, char *szNomeArq2)

Entradas:

.ext: nova extensão;

.szNomeArq1: nome do arquivo.

Saídas:

.szNomeArq2: nome igual ao primeiro, mas com a nova extensão.

Copia o nome de arquivo szNomeArq1 para szNomeArq2, e altera a extensão de szNomeArq2 para ext.

void AtualizaNumFilaServidor(char tipo)

Entrada:

.tipo: tipo do recurso.

Coloca nas variáveis que contém o número de filas e servidores o valor 1 (quando for o caso), para mostrar na caixa Dados do Recurso.

void InicializaNumFilaServidor()

Coloca valores inválidos nas variáveis da rotina anterior.

5.12 Módulo LISTLOOP.C

Este módulo contém as rotinas que manipulam a lista encadeada que armazena os dados relativos a ciclos definidos no modelo. A lista é mantida em memória até que o usuário grave os parâmetros fornecidos através do menu Recursos.

Os "handles" hListaLoop, hMeioListaLoop e hFimListaLoop são definidos neste módulo e indicam, respectivamente, início (para inserção da entrada do modelo), meio (posição para inserção de recursos que não são entrada nem saída do modelo) e fim da lista encadeada (para inserção dos recursos que sejam saída do modelo).

Cada nó da lista apresenta os campos:

```
typedef struct lista_de_tot_loop{
        char n_rec[10];
        unsigned int n_loop;
        LOCALHANDLE prox;
} lista_tot_loop;
```

.n_rec: identificação do recurso, é utilizado pelo Gerador de Aplicação para identificar uma variável contadora do número de vezes que um cliente deve passar pelo ciclo;

.n_loop: número de vezes que um cliente deve passar pelo ciclo;

.prox: "handle" para o próximo nó da lista.

Rotinas:

short InsListaLoop(HWND hjan, char nrec[10], unsigned int nloop, BOOL binsereposatual, int identrec, int condicao)

Entradas:

.hjan; "handle" da janela;

.nrec: identificação do recurso;

.nloop: número de voltas no ciclo;

.binsereposatual: se for TRUE, indica que a parametrização do recurso está sendo alterada, e portanto deve buscar na lista a posição (que já existe) para inserção;

.identrec: identificação do recurso;

.condicao: indica se deve inserir no início, meio ou fim da lista.

Insere na lista os dados relativos ao ciclo definido. A inserção é feita na cabeça da lista, quando se tratar de ponto de entrada do recurso do modelo. Quando for saída, sempre no final. E os demais recursos são inseridos no meio, logo após a cabeça. Se o recurso está sendo alterado, busca e insere no nó que existe.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

short ApagaListaLoop(HWND hjan)

Entradas:

.hjan: "handle" da janela

Apaga a lista de ciclos da memória.

Retorno:

.0: erro ao acessar a lista

.1: operação realizada corretamente

BOOL CancelaParLoopAtual (HWND hjan, ind identrec)

Entradas:

.hjan: "hande" da janela;

.identrec: identificação do recurso.

Remove da lista os dados referentes ao recurso.

Retorno:

.TRUE: operação realizada corretamente;

.FALSE: caso contrário.

5.13 Arquivos de Inclusão

5.13.1 MENSERRO.H

Contém as mensagens de erro apresentadas quando ocorre algum erro durante a verificação de consistência dos parâmetros fornecidos pelo usuário. Utilizado no módulo Consist.C.

5.13.2 TIPOS.H

Contém a definição de estruturas de dados utilizado nos módulos Listrec.c, Lista.c, Listdist.c, Listeven.c, Listalig.c, Listlop.c, Inforec.c, Auxiliar.c e Editarq.c.

5.13.3 EDDEF.H

Contém a definição das variáveis globais utilizadas no EdGraf. Utilizado no módulo principal, Editor.c.

5.13.4 EDITOR.H

Contém a definição de todas as constantes utilizadas pelo Editor Gráfico. Incluído nos módulos Editor.c e Editor.rc.

5.14 Módulo EDITOR.RC

Este módulo contém a definição dos recursos utilizados pelo Editor: menus, caixas de diálogos (todas as caixas para parametrizar o modelo) e ícones (os ícones que aparecem na janela de ícones e nas caixas de diálogo). Recursos são dados que definem a porção visível de um programa Windows, e portanto não deve ser confundido com os recursos definidos em um modelo de simulação. Por exemplo, quando o usuário abre uma caixa de diálogo, ele está interagindo com recursos do

sistema em execução.

Os recursos utilizados no Editor foram projetados com o auxílio do Resource Workshop, da Borland [RES91] (parte integrante do pacote Borland C++ 3.1). O Resource Workshop é uma ferramenta que integra o processo de projetar e compilar recursos para aplicações que rodam no Microsoft Windows, através de editores de menus, de ícones e caixas de diálogo. Permite também que se observe, durante o projeto, como o recurso vai aparecer na tela, facilitando o trabalho do projetista que não necessita compilar e executar todo o sistema para ver como uma caixa de diálogos ou um ícone vai aparecer na tela.

5.15 Arquivo de Definições: EDITOR.DEF

Em adição ao código fonte em C, um outro arquivo é requerido pelos programas Windows. É denominado "arquivo de definições do módulo", e tem a extensão ".def". Esse arquivo auxilia na criação do arquivo executável, informando as características dos segmentos de dados e de código, tamanho do "heap" (para alocação de memória) e o tamanho da pilha. Indica também que o programa "winstub.exe" deve ser executado, caso o usuário tente executar o EdGraf a partir da linha de comando do MS-DOS. O programa winstub está incluso no pacote Borland C++ 3.1, e simplesmente mostra a mensagem: "Este programa requer o Microsoft Windows" e se encerra.

6. Referências Bibliográficas

[MAC87] MacDougall, M.H. **Simulating Computer Systems, Techniques and Tools.** The MIT Press, 1987.

[MIC92] Microsoft Windows Software Development Kit - **Programmer's Reference, v.1: Overview.** Microsoft Corporation, 1992.

[MIC92a] Microsoft Windows SoftwareDevelopment Kit - **Programmer's Reference, v.2: Functions.** Microsoft Corporation, 1992.

[MIC92b] Microsoft Windows Software Development Kit - **Programmer's Reference, v.3: Messages, Structures and Macros.** Microsoft Corporation, 1992.

[PET93] Petzold, Charles. **Programando para Windows 3.1.** Makron Books, 1993.

[RES91] **Resource Workshop User's Guide.** Borland International, 1991.

[SPO93a] Spolon, Roberta. **Um Gerador de Aplicação para um Ambiente de Simulação Automático.** Minidissertação (Mestrado) - ICMSC-USP. Junho 1993.

[SPO93b] Spolon, Renata. **Um Editor Gráfico para Simulação de Sistemas Discretos.** Minidissertação (Mestrado) - ICMSC-USP. Julho 1993.

7. Bibliografia

BORLAND C++ **Tools and Utilities Guide**. Borland International, Inc. 1991.

BORLAND C++ **User's Guide**. Borland International, Inc. 1991.

BORLAND C++ **Programmer's Guide**. Borland International, Inc. 1991.

Microsoft Windows Software Development Kit - **Guide to Programming**. Microsoft Corporation, 1992.

KERNIGHAM, B.W. e RITCHIE, D.M.. **C a Linguagem de Programação, 3 ed., Rio de Janeiro, Campus**, 1987.

TENENBAUM, A.M.; LANGSAM, Y.; AUGENSTEIN, M.J.. **Data Structures using C**. Englewood Cliffs, Prentice-Hall, 1990.