Instituto de Ciências Matemáticas de São Carlos

ISSN-0109-2569

Gerador de aplicação para o ASIA

Descrição da Implementação

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

Nº 21

RELATÓRIOS TÉCNICOS DO ICMSC

SÃO CARLOS
jul. / 1994

SYSNO 868595
DATA / /
ICMC - SBAB

A autora Renata Spolon expressa seus agradecimentos à Fapesp pelo apoio financeiro.

Gerador de Aplicação para o ASiA

Descrição da Implementação

Roberta Spolon

Renata Spolon

Marcos José Santana

Regina H. C. 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

Sumário

Lista de Figuras

1. Introdução 01

2. Interface entre Editor Gráfico e Gerador de Aplicação 02

2.1 Informações Genéricas Sobre o Modelo 02

2.2 Informações Específicas Sobre Cada Recurso 05

2.3 Informações Sobre os Ciclos Existentes no Modelo 11

2.4 Diversas Entradas no Sistema 11

2.5 Modelos Fechados 12

2.6 Recursos Sem Fila 12

2.7 Definição de Novas Distribuições 13

3. Utilização do Gerador de Aplicação 15

3.1 Visão Geral da Estrutura do Gerador 15

3.2 A Descrição de Produtos 17

3.3 A Interface Desenvolvida para o Gerador 19

4. Módulos e Primitivas Implementados 20

4.1 Módulo INTERF.C 20

4.2 Módulo DEF.H 21

4.3 Módulo PROTOT.H 21

4.4 Módulo PRINCIPAL.C 21

4.5 Módulo UTIL.C 29

4.6 Módulo DIST_TAB.C 32

4.7 Módulo REESC.C 34

5. Referências Bibliográficas 35

6. Bibliografia 36

Lista de Figuras

Figura 3.1: Fluxo de Dados no Gerador 16

Figura 3.2: Gabarito 18

1. Introdução

O Gerador de Aplicação constitui o módulo gerador de produtos 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 [SPO93b, SPO93a]. O gerador foi implementado na linguagem C, utilizando-se o compilador Borland C++ versão 3.1, sob o sistema operacional MS-DOS versão 5.0.

No ASiA, um Editor Gráfico é responsável pela interface entre o usuário e o ambiente, e pela geração de diversas estruturas de dados (tabelas de especificação). Estas especificações contêm os dados necessários para que o Gerador de Aplicação construa o programa de simulação correspondente ao modelo definido pelo usuário [SPO93a], basicamente um modelo de rede de filas. A seção 2 discute a definição da interface entre o Editor Gráfico e o Gerador de Aplicação.

O Gerador permite a implementação de programas de simulação em SMPL [MAC87], uma extensão funcional da linguagem C para simulação orientada a eventos. A seção 3 fornece uma visão geral da estrutura do Gerador.

Finalizando, a seção 4 apresenta os módulos e todas as funções implementadas, com descrição dos parâmetros e objetivos de cada uma.

2. Interface entre Editor Gráfico e Gerador de Aplicação

Como o Gerador utiliza na produção dos programas de simulação as informações fornecidas pelo usuário através do Editor Gráfico, tornou-se necessário determinar como armazenar essas informações e passá-las ao Gerador. Definiu-se então que os dados coletados pelo Editor Gráfico seriam armazenados em arquivos (a identificação dos arquivos também faz parte das informações que o Editor Gráfico deve fornecer ao Gerador, tornando-o mais flexível), numa estrutura de dados que facilite a utilização pelo Gerador.

Ressalta-se que a consistência de todas as informações contidas nas estruturas é atribuição do Editor Gráfico, responsável pelas análises sintática e semântica da especificação.

As estruturas de dados definidas são descritas a seguir. Todas as extensões para os nomes dos arquivos utilizados são padronizadas pelo Editor Gráfico. As identificações dos arquivos são determinadas pelo usuário, o que possibilita a reutilização da especificação fornecida. Estas identificações são fornecidas ao Gerador através de estruturas de dados definidas no módulo DEF.H (seção 4.2).

2.1 Informações Genéricas Sobre o Modelo

Dados referentes às principais características do modelo a ser implementado, são armazenados em um arquivo com a extensão padrão ".mod":

```
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],
```

```
                tempo_real;
        unsigned int valores_desprezados,
                     tamanho_batch,
                     distrib_tab,
                     reesc;
        double warm_up;
} especificacao;
```

nome_modelo : corresponde ao nome do modelo (necessário para a implementação em SMPL). Este nome é utilizado na geração do relatório padrão;

parametro_01 : indica se o usuário deseja ou não ativar o módulo mtr, uma interface interativa do SMPL para a execução do modelo de simulação [MAC87]. Assume os valores 0 ou 1;

tempo : limita o período da simulação por tempo de execução. Deve ser especificado como 0.0 se a simulação não for limitada por tempo;

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

num_voltas : limita a simulação pelo número de ciclos que os clientes devem executar no sistema (para o caso de um sistema fechado). Deve ser 0 se o modelo representar um sistema aberto;

trace : ativa o "trace" (ferramenta de depuração do SMPL) e pode assumir os seguintes valores [MAC87]:

0: "trace" desativado;

1: trace "free_running", as mensagens são mostradas continuamente no terminal de vídeo ou 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 que o relatório padrão do SMPL seja gerado. Este relatório apresenta informações sobre a utilização dos recursos, período médio ocupado e comprimento médio da fila associada a cada recurso, além de contadores para as operações realizadas e período de tempo utilizado pela simulação. Assume os valores 0 ou 1;

nome_arqsaida : nome do arquivo que contém o programa de simulação a ser gerado, normalmente com a extensão ".c";

tempo_prim_chegada : tempo para indicar quando irá ocorrer a chegada do primeiro cliente ao sistema;

valores_desprezados : quantidade de valores a serem desprezados antes de iniciar a coleta de dados para análise com o método "Batch Means" [MAC87] no modelo de simulação. Deve ser especificado com o valor 0 se o método não for utilizado;

tamanho_batch : tamanho do lote, quando o método "Batch Means" for utilizado. Deve ser 0 se o método não for utilizado;

distrib_tab : indica o número de novas distribuições definidas pelo usuário, a serem utilizadas no modelo. O Gerador considera as seguintes distribuições:

.distribuições pré-definidas pelo SMPL (exponencial, erlang, hiperexponencial, uniforme e triangular) e

. distribuições definidas pelo usuário.

Se nenhuma distribuição for definida pelo usuário, esse parâmetro assume valor 0 (seção 4.6);

reesc : define se existem recursos sem fila no modelo (para utilização de tabelas para reescalonamento de eventos). Esse parâmetro deve ser

especificado pois a tabela é gerada em um arquivo separado a ser incluído no programa final em SMPL. Assume os valores 0 (quando não é necessário gerar a tabela de reescalonamento) ou 1 (caso contrário) (seção 4.7).

warm_up : caso o usuário não queira utilizar o método de análise "Batch Means", mas queira identificar um tempo na simulação para zerar todos os contadores e medidas obtidas no SMPL, deve utilizar esse parâmetro. O tempo especificado passa a ser o início do intervalo de medida, e as medidas relatadas representam a atividade do modelo de simulação somente daquele ponto no tempo, até o ponto em que o relatório das medidas obtidas for produzido. Pode ser utilizado para descartar dados obtidos enquanto o modelo não atingiu o estado estacionário. Se não for utilizado, deve receber o valor 0.0.

2.2 Informações Específicas Sobre Cada Recurso

Para um dos recursos definidos no modelo existe um conjunto de informações que orientam o Gerador na criação do código no produto final, sendo armazenados em um arquivo, com extensão padrão ".rec". O conjunto de informações é estruturado pelo Gerador em uma lista encadeada na memória:

```
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;
} lista_recursos;
```

ident_rec : identificação do recurso, utilizada pelo Editor Gráfico para verificar se um recurso já foi ou não parametrizado. Não utilizado pelo Gerador;

nome_recurso : identificação do recurso, utilizado pelo SMPL para emissão de relatórios, mensagens de erro e depuração;

num_servidores : número de servidores do recurso;

distribuicao_servico : distribuição do tempo de serviço; pode assumir os seguintes valores:

0: exponencial;

1: erlang;

2: hiperexponencial;

3: uniforme;

4: triangular.

Para distribuições definidas pelo usuário, a identificação é feita mantendo-se a seqüência estabelecida, ou seja, a próxima distribuição será identificada como 5, depois 6 e assim por diante, até o máximo de definições;

est_max_min : indica se o usuário deseja incluir no programa de simulação gerado estatísticas sobre o tamanho máximo e mínimo da fila. Deve ser utilizado com recursos com uma fila, pois SMPL não implementa recursos com várias filas. Os dados são apresentados ao final da execução da simulação em forma de relatório, logo após o relatório padrão do SMPL (na criação dos relatórios é obedecida a seguinte seqüência: relatório padrão, relatório com estatísticas sobre o tamanho máximo e mínimo das filas e relatório com informações sobre o estado da fila (em seqüência). Assume os valores 0 (não deseja que a estatística seja gerada) ou 1 (caso contrário);

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 recurso com apenas uma fila. Segue as mesmas regras do parâmetro est_max_min;

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 distinta da triangular seja utilizada, esse parâmetro deve assumir o valor 0.0.

seq_num_aleat_servico : seqüência para a geração de números aleatórios para o tempo de serviço. O valor 0 indica que a escolha será feita internamente pelo SMPL, 1 a 15 indicam diferentes seqüências.

disciplina : indica o algoritmo de escalonamento para a formação de filas e atendimento dos clientes. SMPL utiliza a disciplina FCFS ("First Come First Served"), onde os clientes são atendidos na sequência em que chegam ao recurso). Esse parâmetro permanece como opção para futuras expansões do ASiA a outras extensões e linguagens de simulação. Assume o valor 0 durante o processo de execução. Poderão também ser consideradas as seguintes disciplinas [SOA90, SPO93b]:

LCFS : ("Last Come First Served") : essa disciplina tem a estrutura de uma pilha, onde o último cliente 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 cliente com maior (ou menor) tempo de serviço esperado é atendido primeiro;

Uso de prioridades : as formas de escalonamento com prioridades são utilizadas para dar preferência a alguns clientes, e podem ou não utilizar preempção.

escolha_fila : algoritmo para determinar de qual fila retirar o próximo cliente a ser atendido quando o recurso apresenta mais de uma fila. Como SMPL implementa apenas uma fila para um mesmo recurso, esse parâmetro não é utilizado devendo assumir o valor 0, mas deve ser mantido para futuras expansões do ASiA;

escolha_cliente : algoritmo para determinar em qual fila o cliente irá entrar, caso o recurso apresente 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 recurso [MAC87]). Esta informação também foi mantida prevendo-se expansões do ASiA e deve assumir o valor 0;

escolha_prox_rec : determina o modo de decisão para escolha do próximo recurso. Se a saída de um recurso está ligada a vários outros, deve hever uma decisão para escolher para qual recurso um cliente do sistema deve se dirigir. Pode assumir os seguintes valores:

0: utiliza intervalos de probabilidade. No caso do recurso estar ligado a apenas um, este campo também deve ser assumir o valor 0;

1: por meio de ciclos, isto é, um cliente entra na fila do recurso, é atendido e volta para a fila, repetindo este procedimento até atingir o número especificado de vezes;

2: utilizado para sistemas fechados [MAC87, SOA90] e deve ser especificado para o primeiro recurso definido no modelo.

distribuicao_chegada : distribuição do tempo entre as chegadas dos clientes, podendo ser:

0: exponencial;

1: erlang;

2: hiperexponencial;

3: uniforme,

4: triangular.

Segue as mesmas considerações para o parâmetro distribuição_serviço;

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

desvio_padrao_chegada : desvio padrão do tempo entre as chegadas dos clientes, utilizado pelas distribuições erlang, hyperexponencial e uniforme (para esta distribuição o campo armaazena o limite superior do intervalo);

seq_num_aleat_chegada : seqüência para a geração de números aleatórios para o tempo entre as chegadas dos clientes no sistema. O valor 0 indica que a escolha será feita internamente pelo SMPL, 1 a 15 indicam diferentes seqüências.

entradas_saídas : determina o número do próximo recurso que será utilizado pelo cliente, o intervalo de probabilidade ou o número de vezes que um cliente deve passar pelo mesmo recurso. Assim, para cada ligação ou ramificação de um recurso para outro, este campo recebe um grupo de 4 caracteres: os dois primeiros identificam o próximo recurso (para indicar saída do sistema, adotou-se a identificação "00"), e os próximos dois dígitos

dependem do valor assumido pelo campo escolha_prox_rec; um caracter '$' indica o término da seqüência:

esc_prox_rec = 0: nesse caso, vai haver uma escolha de qual recurso utilizar através de um intervalo de probabilidade. Os dois dígitos seguintes representam um intervalo para a probabilidade, isto é, para representar um intervalo com 10% de probabilidade do cliente seguir para o recurso especificado os dígitos devem conter "11", para 20% "12" etc;

esc_prox_rec = 1: os dois dígitos seguintes podem ser:

"00": quando cria um ciclo entre os recursos;

"10": quando uma entrada deve ser gerada obrigatoriamente 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 [MAC87, SOA90]. Deve ser determinado para o primeiro recurso do modelo. Segue a mesma sintaxe do caso 1, e devem ser escalonados eventos para ocorrer antes do início da simulação (seção 2.5).

sem_fila : indica se o recurso possui ou não fila, pode assumir os seguintes valores:

. 0: recurso possui fila;

. 1: o recurso não tem fila e utilizará uma tabela de tempos para o reescalonamento de eventos (seção 4.7) ou

. 2: o recurso não possui fila e utilizará um algoritmo para o reescalonamento de novas tentativas de utilização do recurso [MAC87];

num_filas : indica o número de filas do recurso (um recurso tem pelo menos uma fila);

flag_sinc : deve indicar com quantos recursos o recurso passivo [SOA90] está sincronizado. Se não houver sincronismo, especificar 0 para esse parâmetro;

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 utilizando os serviços de outro(s) recurso(s)). Deve receber o valor **0** quando o recurso passivo não é liberado ou 1 em caso contrário.

2.3 Informações sobre Ciclos Existentes no Modelo

Esta estrutura armazena os dados relativos aos ciclos em um modelo quando, por exemplo, um cliente deve utilizar um recurso específico ou percorrer um caminho por um número determinado de vezes no sistema.

n_rec : identificação do contador do número de vezes que um "token" (SMPL) ou cliente deve passar por um ciclo;

n_loop : número de voltas que um "token" (cliente) deve executar em um ciclo.

2.4 Diversas Entradas no Sistema

Existem modelos onde podem ocorrer mais de uma entrada para o sistema. Nesse caso, é necessário especificar os recursos para os quais uma entrada no sistema vai ser gerada. As identificações dos recursos são armazenadas em arquivo com extensão padrão ".ent".

```
struct entradas_separadas
        {
        unsigned int num_recurso;
        } filas_separadas;
```

num_recurso : identificação (número) do recurso para o qual deve ser gerada uma entrada no sistema.

2.5 Modelos Fechados

Para modelos de sistemas fechados [MAC87, SOA90] normalmente alguns eventos são escalonados para ocorrência antes do início da simulação, e a chegada de novos clientes ao sistema não ocorre. Para que o programa de simulação seja gerado corretamente, o usuário deve especificar quais os eventos que devem ocorrer. Essas informações são armazenadas em um arquivo com extensão padrão ".eve".

```
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.

2.6 Recursos Sem Fila

Quando um recurso não possui fila, é necessário estabelecer uma política para reescalonar uma nova tentativa de utilização quando não há servidores disponíveis para atender ao cliente.

O Gerador implementa todos os comandos necessários para a utilização de uma tabela circular para reescalonamentos. A cada tentativa mal sucedida de utilização de um recurso, um novo tempo é utilizado para escalonar a próxima tentativa. Ressalta-se que SMPL não implementa recursos sem fila, sendo feita uma "simulação" neste caso.

```
struct reesc
        {
        char tempo_reesc[5];
        } Reescalona;
```

tempo_reesc : tempo para o qual deve ser escalonado uma nova tentativa de utilização do recurso.

2.7 Definição de Novas Distribuições

A execução da simulação requer a geração de amostras aleatórias para indicar, por exemplo, o tempo da próxima chegada de um cliente ao sistema ou tempo de serviço em um recurso determinado. Podem-se obter tais amostras a partir de uma distribuição teórica (se os dados se ajustam a uma determinada distribuição) ou diretamente das tabelas fornecidas pelo usuário.

Além das distribuições de probabilidade pré-definidas, (seção 2.2), o ambiente ASiA possibilita ao usuário a definição de novas distribuições, através de tabelas fornecidas ao Editor Gráfico. O usuário fornece ao editor os valores e números de ocorrência observados e este produz a distribuição de freqüência correspondente. A partir dessa distribuição o Gerador obtém a distribuição de freqüência acumulada [MAC87].

Os dados são armazenados em um arquivo com a extensão padrão ".dis". Com esses dados a(s) distribuição(ões) é(são) gerada(s) e armazenada(s) no arquivo DTAB.C, a ser incluído no programa de simulação em SMPL.

O Gerador limita o número de novas distribuições (no máximo 10) pois é necessário definir uma estrutura de dados estática (módulo DEF.H) para armazenar a identificação dos arquivos utilizados. Essa estrutura pode ser ampliada, caso seja necessário.

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

valor : valor cujo número de ocorrências foi observado;

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

3. Utilização do Gerador de Aplicação

3.1 Visão Geral da Estrutura do Gerador

Antes de iniciar o processo de geração de código, o Gerador prepara as tabelas de especificações do Modelo e Recursos, cujas estruturas de dados estão descritas na seção 2. As tabelas seguintes são preparadas apenas quando é necessário a utilização de suas informações:

- Ciclos, cuja estrutura de dados é descrita na seção 2.3;
- Entradas, sua estrutura de dados é descrita na seção 2.4;
- Fechado, a seção 2.5 relata sua estrutura;
- Distrib_Tab, cuja estrutura de dados é relatada na seção 2.6 e
- Reesc, tem sua estrutura de dados especificada na seção 2.7.

O Gerador utiliza paralelamente as informações contidas no Gabarito do programa de simulação (descrito no próximo item) e as tabelas de especificações para a geração do produto final. A figura 3.1 demonstra o fluxo de dados no Gerador.

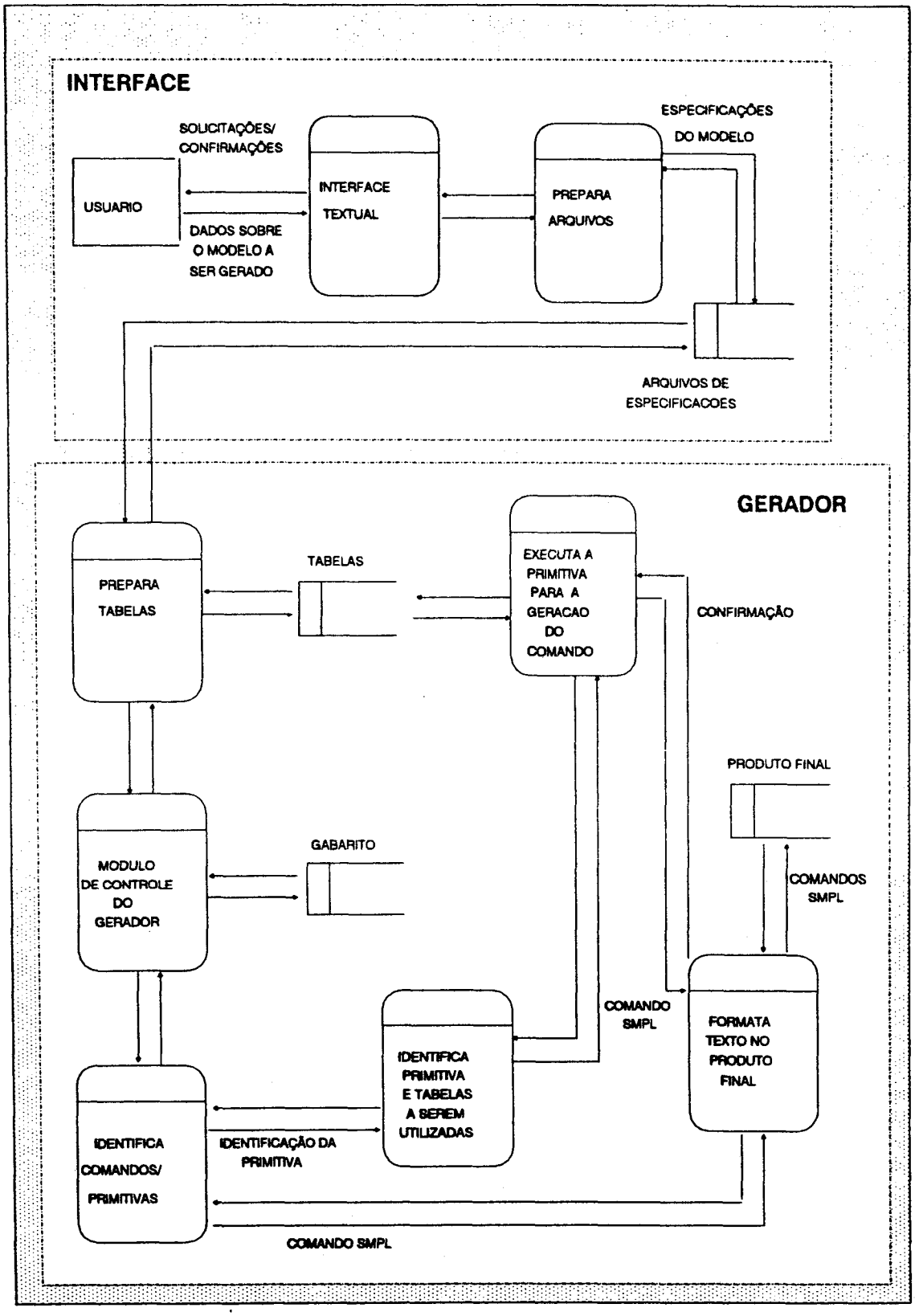

Figura 3.1: Fluxo de Dados no Gerador

No Gerador todo o processo de geração de código é controlado por um procedimento principal (Le_Gabarito) que identifica os comandos do gabarito (precedidos pelo caracter '%') decidindo qual primitiva deve ser executada. Identificada, a primitiva é executada, e o controle retorna ao procedimento Le_Gabarito, para continuar o processo de geração.

Cada uma das primitivas que compõem o Gerador é responsável por criar um trecho específico no código final, utilizando as informações contidas nas tabelas (cujas estruturas de dados estão descritas na seção 2). Todas as primitivas implementadas são detalhadas na seção 4. Podem ser gerados também alguns arquivos que serão incluídos no programa de simulação.

3.2 A Descrição de Produtos

Além das tabelas que especificam o modelo a ser implementado (cujas estruturas de dados foram descritas na seção 2), o Gerador utiliza no processo de geração de código outra tabela, denominada descrição de produtos.

A descrição de produtos é um tipo de gabarito, parecido com o produto final, onde trechos de códigos na linguagem alvo (SMPL) estão misturados com comandos que indicam ao Gerador como buscar as informações necessárias nas várias tabelas que serão utilizadas. Os trechos de código são copiados para o produto final, enquanto os demais comandos (precedidos por um caracter especial), determinam quais as primitivas do Gerador devem executadas. A descrição utilizada será referenciada neste trabalho como Gabarito.

A estrutura do Gabarito reflete exatamente o produto final a ser gerado, no caso um programa de simulação em SMPL. Esse Gabarito é específico para o tipo de produto a ser criado, juntamente com o núcleo do gerador.

```
#include "stdio.h"
#include "conio.h"

%Idistribuicao_tabela

main()
{
%0define_tempo_num_max
int Event = 1, Customer = 1, Aleatorio;
%Agera_contadores_loop
%1define_variaveis

%2define_smpl
%Hmodulo_estat
%3define_server
%4escalona_primeiro_evento
%Bgera_trace

%5gera_loop
{
%Jwarm_up
%6gera_cause
%7gera_switch
{
%8gera_eventos
}

%9gera_relatorio_padrao
%Cgera_relatorio_estatisticas_max_min
%Dgera_relatorio_estatisticas_fila_vazia
```

Figura 3.2: Gabarito

3.3 A Interface Desenvolvida para o Gerador

Como a atividade de desenvolvimento dos módulos que compõem o ASiA (incluindo o Gerador) está sendo executada paralelamente, foi implementada uma interface simples e independente para a criação das diversas tabelas necessárias. Esta interface é utilizada para a realização de testes no Gerador.

A interface é textual, sendo composta por diversos módulos e o usuário se limita a responder questões do tipo Sim ou Não ou a fornecer os parâmetros especificados. Como a interface foi desenvolvida apenas para a execução de testes no Gerador e deve ser retirada quando os módulos do ASiA (Editor Gráfico e Gerador de Aplicação) forem integrados, não é feita nenhuma consistência dos dados fornecidos.

4. Módulos e Primitivas Implementados

4.1 Módulo INTERF.C

Este é um módulo independente, sendo responsável pela interface com o usuário do sistema. Foi desenvolvido com a finalidade de possibilitar a realização de testes com o gerador. A sua implementação não está associada ao Gerador pois, quando o ambiente ASiA for integrado (módulos Gerador de Aplicação e Editor Gráfico), esta interface será retirada do sistema.

A interface desenvolvida é constituída pelos seguintes módulos para a criação das tabelas de especificações do modelo a ser implementado (as estruturas de dados utilizadas estão descritas na seção 2):

- **criamod.c** : contém as primitivas necessárias para possibilitar que o usuário especifique as principais características do modelo a ser implementado;

- **criarec.c.c** : permite ao usuário especificar cada um dos recursos que compõem o modelo, definindo suas características particulares;

- **criaent.c** : possibilita a definição de várias entradas no modelo;

- **criatab.c** : permite ao usuário a definição de novas distribuições de probabilidade;

- **criaf.c** : possibilita a definição dos recursos para os quais um determinado evento deva ser escalonado antes do início da simulação;

- **criarees.c** : se o usuário definiu recursos sem fila no seu modelo, este módulo permite a especificação de uma tabela contendo os tempos para uma nova tentativa de utilização do recurso caso encontre este ocupado.

Cada um desses módulos contém primitivas para abrir/fechar arquivos, leitura/gravação de registros e mensagens de erros durante sua execução.

4.2 Módulo DEF.H

Contém as definições das estruturas de dados utilizadas pelo Gerador (descritas na seção 2), além de variáveis e constantes globais necessárias durante a execução do sistema. Este módulo é incluído no módulo PRINCIPAL.C.

4.3 Módulo PROTOT.H

Contém os protótipos de todas as primitivas definidas nos módulos PRINCIPAL, UTIL, REESC e DIST_TAB. Este módulo é incluído no módulo PRINCIPAL.C.

4.4 Módulo PRINCIPAL.C

Este módulo implementa o núcleo do Gerador, ou seja, define as principais primitivas utilizadas no processo de geração de código. A seguir são descritas as primitivas desenvolvidas.

Primitivas do Módulo PRINCIPAL

void Define_Tempo_Max(void)

Gera o comando de repetição, com a finalidade de limitar a execução da simulação:

- por tempo;
- por número de clientes que deixam o sistema;
- pelo número de ciclos que o cliente executa no sistema ou
- através do método de análise "Batch Means".

void Define_Var_Tempos(void)

Responsável pela geração no programa final (em SMPL) das diversas variáveis associadas aos tempos médios entre chegadas de clientes e tempo médio de serviço para cada recurso. Como todas as informações referentes aos recursos estão mantidas em uma lista encadeada (Recursos), a primitiva Define_Var_Tempos() percorre a lista identificando e gerando no produto final as variáveis necessárias.

void Define_Var_Reescalona(void)

Gera no produto final os índices para as tabelas de tempo de reescalonamento de eventos (para recursos sem fila). As informações necessárias são obtidas da lista Recursos.

void Define_Var_Server(void)

Implementa as estruturas de dados para a definição dos recursos que compõem o modelo, no programa de simulação. Sua execução é semelhante às primitivas anteriores.

void Define_Estat_Max_Min(void)

Define no programa de simulação as estruturas de dados que serão utilizadas para calcular o tamanho máximo e mínimo da fila associada ao recurso especificado. Identifica na lista Recursos para quais recursos a estatística deve ser implementada, gerando no programa de simulação as variáveis necessárias.

void Define_Estat_Fila_Vazia(void)

Define no produto final as estruturas de dados utilizadas para o cálculo da porcentagem de vezes em que um cliente encontra, em um determinado recurso, fila vazia. Sua execução é semelhante à primitiva Define_Estat_Max_Min().

void Define_BMeans(void)

Gera as estruturas de dados necessárias para a utilização do método de análise "Batch Means" [MAC87].

void Define_Reesc(void)

Implementa as estruturas de dados necessárias para o reescalonamento de um evento quando o recurso não tem fila. O parâmetro reesc na estrutura Modelo (seção 2.1) determina se existem recursos sem fila no modelo. Em caso afirmativo, a primitiva procura na lista Recursos quais os recursos sem fila para a geração das estruturas de dados no programa de simulação. A primitiva Define_Tabela_Reescalona() é responsável pela criação da tabela de reescalonamento (módulo REESC.C), enquanto a primitiva Define_Var_Reescalona() (módulo PRINCIPAL) gera os índices para a tabela.

void Gera_Contadores(void)

Gera no programa de simulação as estruturas de dados para os ciclos de repetição no qual um cliente permanece dentro do sistema, utilizando as informações da tabela Ciclos.

void Nome_Parametro(void)

Implementa o comando *smpl()* para iniciar o ambiente de simulação, utilizando a tabela Modelo.

void Gera_Trace(void)

Implementa o comando *trace()* (ferramenta de depuração do SMPL), com informações da tabela Modelo.

void Gera_Def_Server(void)

Gera no programa de simulação o comando *facility()* para a definição dos recursos do sistema (um recurso, em SMPL, corresponde a uma única fila e a *n* servidores paralelos), através da lista Recursos.

void Gera_Primeiro_Evento(void)

Gera as primitivas necessárias para escalonar o primeiro evento (modelos com uma única entrada) antes do início da simulação, ou todos os eventos a serem escalonados para modelos fechados. Utiliza as tabelas Modelo e Fechado.

void Gera_Loop(void)

Implementa o comando de repetição, limitando a simulação por tempo, número de clientes que deixam o sistema, número de ciclos executados ou pelo método de análise "Batch Means". Utiliza as tabelas Modelo e Ciclos.

void Gera_Cause(void)

Gera no produto final o comando *cause()* do SMPL.

void Gera_Switch(void)

Cria no programa de simulação o comando *switch*.

void Gera_Entradas_Saidas(void)

Um cliente, ao ser atendido em um recurso, pode requisitar os serviços de um outro recurso ou mesmo deixar o sistema. A primitiva Gera_Entradas_Saidas implementa os comandos necessários para a executar os comandos que representem o caminho percorrido pelo cliente no sistema, utilizando as tabelas Recursos, Ciclos e Fechado. Assume os valores especificados na seção 2.2. Também gera os comandos para totalizar o número de clientes que deixam o sistema.

void Gera_Filas_Separadas(void)

Responsável por gerar os comandos para diversas entradas no modelo, utilizando a tabela Entradas.

void Gera_Release(int numero_recurso,int *Numero_Evento,int *total_recurso)

Parâmetros:

numero_recurso: identificador associado ao recurso

Numero_Evento: número do próximo evento a ser gerado

total_recurso: armazena o número de recursos que constituem o modelo.

Gera no produto final o comando release para a liberação do recurso indicado por numero_recurso. Se existe no modelo a posse simultânea de recursos, implementa também todos os comandos necessários para representar o sincronismo entre esses recursos.

void Gera_Estat_Fila_Vazia(int Numero_Recurso)

Numero_Recurso: identifica o recurso para o qual deve ser gerada a estatística.

Cria no produto final os comandos necessários para o cálculo da porcentagem de vezes em que o cliente não encontra fila ao utilizar o recurso identificado por Numero_Recurso. No programa de simulação, a cada passagem de um cliente pelo recurso especificado, verifica-se se o mesmo tem servidor(es) livre(s), através do comando *status()* do SMPL, e os contadores associados são atualizados. Esses contadores são definidos pela primitiva Define_Estat_Fila_Vazia().

void Gera_Estat_Max_Min(int Numero_Recurso)

Parâmetro:

Numero_Recurso: identifica o recurso para o qual deve ser gerada a estatística.

Cria no programa de simulação os comandos necessários para o cálculo do número máximo e mínimo de clientes na fila do recurso identificado por Numero_Recurso. No programa de simulação, a cada passagem de um cliente pelo recurso especificado, o número total de clientes na fila do recurso é verificado através do comando *inq()* do SMPL, atualizando os contadores de número máximo e mínimo de clientes. Esses contadores são definidos pela primitiva Define_Estat_Max_Min().

void Gera_Stream(int Sequência, char Espaco[])

Parâmetros:

Seqüência: número da sequência a ser utilizada para a geração de números aleatórios, variando de 1 a 15.

Espaco: deslocamento do comando gerado no produto final.

Gera no programa de simulação o comando *stream()* do SMPL para determinar qual seqüência será utilizada na geração de números aleatórios. Se Sequência for igual a 0, SMPL utiliza a seqüência 1 e incrementa a cada execução do comando smpl().

void Escalona_Release(int Numero_Evento, int numero_recurso)

Parâmetros:

Numero_Evento: número do próximo evento a ser gerado

numero_recurso: identificador do recurso

Implementa no programa de simulação os comandos para fazer a requisição de um recurso (comando *request()*) e escalonar o término da utilização do mesmo (comando *schedule()*). Se o recurso não tem fila, uma nova tentativa é escalonada de acordo com a política estabelecida (através de uma tabela circular de tempos para reescalonamento (módulo REESC.C)).

void Gera_Proxima_Chegada(int Numero_Evento, int numero_recurso)

Parâmetros:

Numero_Evento: número do próximo evento a ser gerado

numero_recurso: identificador do recurso

Implementa os comandos necessários para determinar a chegada de um novo cliente ao sistema, utilizando a lista Recursos.

void Gera_Case(int Numero_Evento)

Parâmetro:

Numero_Evento: próximo evento a ocorrer

Gera o comando *case* no programa de simulação.

void Gera_Eventos(void)

Controla a seqüência de eventos que devem ser gerados para constituir o programa de simulação. Baseando-se na estrutura comum dos programas a serem gerados (definida no Gabarito), é possível determinar quais eventos devem ser criados no produto final.

void Gera_Relatorio_Final(void)

Gera no programa de simulação o comando *report()* do SMPL para a emissão do relatório padrão. Utiliza a tabela Modelo.

void Gera_Relat_Estat_Max_Min(void)

Implementa os comandos necessários para a emissão de relatório contendo o número máximo e mínimo de clientes nas filas associadas aos recursos especificados. O relatório é apresentado no monitor de vídeo, dispositivo padrão para saída de dados. As estruturas de dados necessárias são definidas pela primitiva Define_Estat_Max_Min().

void Gera_Relat_Fila_Vazia(void)

Gera no produto final os comandos necessários para a emissão de relatório contendo a porcentagem de vezes em que o cliente não encontra fila em um determinado recurso. Este relatório é apresentado no dispositivo padrão de saída, o monitor de vídeo. As estruturas de dados necessárias são definidas pela primitiva Define_Estat_Fila_Vazia().

void Warmup(void)

Gera o comando *reset()* do SMPL no programa de simulação, além de comandos adicionais para zerar as estatísticas definidas pelo Gerador. Com este comando é possível desprezar os dados coletados no sistema antes que o mesmo entre em equilíbrio. Utiliza a tabela Modelo e a lista Recursos.

void Gera_Metodo_Analise_Saida(void)

Implementa a chamada ao módulo *init_bm()* do SMPL, para a utilização do método "Batch Means" (que estima um intervalo de confiança para a medida de uma variável de saída da simulação). O método é aplicado sobre os tempos de serviços no sistema. Utiliza a tabela Modelo.

void Le_Gabarito(void)

Controla a seqüência de execução das primitivas que compõem o Gerador, de acordo com os comandos do Gabarito (seção 3.2). Esta primitiva distingue os trechos de código dos comandos através do caracter '%'. A primitiva correspondente ao comando encontrado no Gabarito é executada e o controle da execução volta para a primitiva Le_Gabarito().

void Prepara_Estruturas(void)

Cria uma lista encadeada com as informações relativas aos recursos, de acordo com as estruturas de dados descritas na seção 2.2. A lista é removida após a execução do gerador. Esta primitiva também prepara uma tabela com as especificações genéricas do modelo (descritas na seção 2.1).

4.5 Módulo UTIL.C

Contém as primitivas (ferramentas) comuns utilizadas pelos diversos módulos do sistema para abertura/fechamento de arquivos, leitura/gravação de registros, manipulação de cadeias de caracteres, mensagens de erros e auxílio na geração de eventos específicos. Algumas das primitivas para a manipulação de cadeias de caracteres foram implementadas para dar maior eficiência à execução do Gerador.

Primitivas do Módulo UTIL.C

Algumas das primitivas desse módulo utilizam uma estrutura de dados ("buffer"), definido no módulo DEF.H. O "buffer" armazena os comandos a serem gravados no programa de simulação.

void Limpa_Buffer(void)

Limpa o "buffer" utilizado para armazenar os comandos gerados para o produto final (programa de simulação).

void Grava_Buffer(void)

Armazena o conteúdo do "buffer" no arquivo destino, ou seja, o programa de simulação.

void Copy(char Literal[])

Parâmetro:

Literal: variável do tipo caadeia de caracteres a ser copiada para o "buffer"

Copia para o "buffer" as informações contidas em Literal.

void Erro(Char Num_Erro)

Parâmetro:

Num_Erro: número da mensagem de erro a ser exibida

Retorna mensagens de erros ocorridos durante a execução do Gerador.

void Abre_Arquivos(void)

Abre os seguintes arquivos utilizados pelo Gerador:

Gabarito - contém as informações sobre o produto a ser gerado;

Modelo - contém as informações sobre o Modelo a ser implementado;

Recursos - contém informações dos Recursos que constituem o Modelo;

Ciclos - contém o número máximo de ciclos para um caminho;

Ev_Fechado - indica quais eventos devem ser escalonados para ocorrência no caso de sistemas fechados;

Entradas - indica o número dos servidores quando há mais de uma fila para entrada no modelo (filas paralelas);

Reesc - tabela de tempos para reescalonamento de eventos e

Tab - definição de novas distribuições.

void Fecha_Arquivos(void)

Fecha todos os arquivos utilizados pelo Gerador.

void Itoa_Def(unsigned int Numero, char Literal[])

Parâmetros:

Numero: número a ser convertido para o tipo string

Literal: armazena o resultado da conversão

Converte um número do tipo inteiro sem sinal para o tipo cadeia de caracteres.

void Strcat_Def(char Literal[], char Caracter)

Parâmetros:

Literal: variável do tipo cadeia de caracteres

Caracter: caracter a ser concatenado

Concatena o caracter Caracter em Literal.

void Libera_Lista_Recurso(void)

Libera a memória ocupada pelos nós que compõem a lista encadeada com as informações sobre os recursos, criada no início da execução do Gerador.

int Atoi_Def(char Literal[])

Parâmetro:

Literal: variável do tipo string a ser convertida

Retorno: número inteiro resultante da conversão

Converte a variável Literal para seu valor correspondente no tipo inteiro.

void Auxilia_Gera_Release(int Numero_Evento, int numero_recurso)

Parâmetros:

Numero_Evento: número do próximo evento a ser gerado

numero_recurso: identificador do recurso

Auxilia no processo de geração dos eventos quando há a posse simultânea de recursos. É utilizada pela primitiva Gera_Release() do módulo PRINCIPAL.C.

4.6 Módulo DIST_TAB.C

Contém as primitivas necessárias para a implementação das distribuições de freqüência acumulada. As distribuições geradas são armazenadas no arquivo DTAB.C, a ser incluído no produto final (programa de simulação).

Primitivas do Módulo DIST_TAB.C

Algumas das primitivas desenvolvidas utilizam uma estrutura de dados ("buffer"), definida no próprio módulo. Esse "buffer" armazena os comandos a serem gravados no arquivo DTAB.C.

void Limpa_Buf(void)

Libera o "buffer" utilizado para a criação do arquivo de saída.

void Grava_Buf(void)

Grava o "buffer" no arquivo de saída.

void Cop(char Literal[])

Parâmetro:

Literal: comandos a serem gravados no arquivo saída

Copia para o "buffer" os comandos a serem gravados no arquivo de saída.

void Itoa_Defi(unsigned int Numero, char Literal[]);

Parâmetros:

Numero: número a ser convertido

Literal: armazena o resultado da conversão

Converte um número do tipo inteiro sem sinal para o tipo cadeia de caracteres.

void Convert(float Numero, char Literal[])

Parâmetros:

Numero: número a ser convertido

Literal: armazena o resultado da conversão

Converte um número do tipo ponto flutuante para o tipo cadeia de caracteres.

unsigned int Verifica_Tabela(unsigned int Cont_Dist)

Parâmetro:

Cont_Dist: identificador da distribuição a ser gerada

Retorno: tamanho da tabela de distribuição de freqüência

Verifica o número de entradas na tabela que armazena os tempos de reescalonamento, para gerar as estruturas de dados e comandos necessários para o reescalonamento de eventos.

void Monta_Array_Acumulado(unsigned int Cont_Dist)

Parâmetro:

Cont_Dist: identificador da distribuição a ser gerada

Através da tabela Distrib_Tab, que contém o número de ocorrência de um determinado valor e a freqüência de ocorrência deste no conjunto observado, esta primitiva obtém uma distribuição de freqüência acumulada, a ser armazenada no arquivo DTAB.C. Esse arquivo é incluído no programa de simulação gerado. São gerados também as estruturas de dados e comandos necessários para a obtenção de números aleatórios através da distribuição obtida [MAC87].

4.7 Módulo REESC.C

Contém as primitivas necessárias para a geração da tabela de tempos e comandos necessários para a sua utilização (seção 2.6). Essa tabela é empregada no reescalonamento de eventos para recursos sem fila.

Primitivas do Módulo REESC.C

unsigned int Verifica_Tabela_Reescalona(void)

Retorno: tamanho da tabela de tempos para reescalonamento de eventos

Verifica o número de entradas na tabela que armazena os tempos de reescalonamento (Reesc), a fim de gerar as estruturas de dados e comandos necessários para o reescalonamento de eventos.

void Monta_Tabela_Reescalona(void)

Verificado o número de entradas da tabela de tempos para reescalonamento de eventos (através da primitiva Verifica_Tabela_Reescalona()), gera no programa de simulação uma estrutura de dados, contendo os tempos para o reescalonamento das tentativas de utilização dos recursos sem fila.

5. Referências Bibliográficas

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

[SOA90] SOARES, L. F. G. **Modelagem e Simulação Discreta de Sistemas.** VII Escola de Computação, São Paulo, 1990.

[SPO93a] SPOLON, Roberta. **Um Gerador de Aplicação para um Ambiente de Simulação Automático.** São Carlos, 1993. Mini- Dissertação (mestrado) - Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo.

[SPO93b] SPOLON, Renata. **Um Editor Gráfico para um Ambiente de Simulação Automático.** São Carlos, 1993. Mini-Dissertação (mestrado) - Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo.

6. 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.

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

LEWIS, T. Code Generators. **IEEE Software**, v. 7, n. 3, p. 67-70, May 1990.

LUKER, P. A.; BURNS, A. Program Generator and Generation Software. **The Computer Journal.** v. 29, n. 4, p. 315-321, 1986.

MACDOUGALL, M. H. System Level Simulation - em **Digital System Design Automation: Languages, Simulation and Data Base.** Computer Science Press, Inc., p. 1-115, 1975.

MEIRA, C. A. A. **Sobre Geradores de Aplicação.** São Carlos, 1991. Dissertação (mestrado) - Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo.

MEIRA, C. A. A.; MASIERO, P. C. Um Gerador de Aplicação para Sistemas Reativos. In: V SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, Ouro Preto-MG, 1991. **Anais.** p. 45-59.

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

WALRAND, J.. **An Introduction to Queueing Networks.** Prentice-Hall International, Inc., 1988.