**Uma proposta de Reengenharia
para o Ambiente de
Simulação Automático (ASiA)**

Ana Elisa Tozetto Piekarski

**Orientação:
Prof. Dr. Marcos José Santana**

*Dissertação apresentada ao Instituto de Ciências Matemáticas
e de Computação (ICMC-USP) como parte dos requisitos para
a obtenção do título de mestre em Ciências - Área: Ciências
de Computação e Matemática Computacional.*

**São Carlos
1998**

"A felicidade não está em viver, mas em saber viver.

Não vive mais o que mais vive, mas o que melhor vive,
porque a vida se não mede pelo tempo, mas pelo emprego que dela fazemos."

Dedicatória

À minha Mãe, que me mostrou o sentido das coisas mais valiosas da vida.

Ao meu Pai, que me ensinou a ser uma pessoa íntegra e batalhadora.

Amo vocês!

Agradecimentos

É complicado lembrar de todo mundo que contribuiu para a realização deste mestrado, pois isso não é apenas uma conquista desses últimos dois anos e quatro meses, mas sim consequência das experiências vividas até o momento...

Em primeiro lugar, agradeço a Deus pelas bênçãos a mim concedidas, pelo maravilhoso dom da vida e toda a sua intensidade!

Aos meus pais, Edmundo e Eunice, que não só me deram a vida, mas com muito amor me ensinaram e permitiram vivê-la!

Aos meus irmãos Eduardo e Eliane, meus segundos pais!

À Vó Lúcia e meus cunhados Silmara e Gilmar, pelo apoio e carinho!

À memória de meus avós Marcos, Magdalena e Estanislau, por seus exemplos sempre vivos.

Aos meus sobrinhos Amanda, Leandro e Marina: só quem tem crianças por perto sabe o quanto se pode aprender com eles...

Aos meus tios e primos, naturais e postigos, pelo afeto e incentivo em cada passo vivido.

Ao pessoal da Castrolanda, pelo apoio no iniciar o mestrado, em especial Angelo, Janaína, Ivone, Marcelo, Adriana e Carlos.

Ao Paulo e à Simone, que me abriram as portas de São Carlos e da USP, pela amizade, carinho e incentivo.

Ao Prof. Marcos e à Profa. Regina, pela acolhida, orientação, paciência, incentivo, amizade e disposição em ensinar!

À Profa. Rosângela, pela amizade e auxílio significativo no projeto.

À Jaqueline, minha companheira de residência, pela amizade, carinho, incentivo e partilha em todos os momentos!

À turminha toda especial formada em São Carlos (como diria a Mirla, "os mais legais"): Marisa, Cris, Luciano, Jaque, Fabiano, Mirla, Rudinei, Garga, Paulo, Simone (e o Felipe também!), Luis Carlos, Ernesto, Sandra, Mário, Márcia e Karina:

"Um amigo é um presente que você dá a si mesmo." (Robert Louis Stevenson)

Ao pessoal do LaSDPC, pela amizade, auxílio nos trabalhos e na dissertação, churrascos e piadinhas que tornaram a rotina muito mais agradável: Adriana, Sarita, Célia, Paulo, Luciano, Silmara, Aletéia, Omar, Kalinka, Márcio, Flávio, Rita, Renato, Regiane, Helder, Renata e Roberta. Em especial, agradeço a Sarita e a Célia as valiosas contribuições quanto ao ASiA.

Ao pessoal do "convívio informal", nas festas e corredores, do LABIC, LABES, Intermídia, LCAD, NILC e também da Matemática, Elétrica, Transportes, etc: Maristela, Patrícia, Juliana, Augusto (da Jaque), Alexandre Barra, Alessandra, Kutova, Sissy, Luciano Esteves, Lauro, Andréa (Cabelinha), Nilda, Telma, Luciana, Valéria, João Caldas, Tatiana, Andréia Simoni, Renata, Larissa, Chandler, Gawa, Júlio, Andréia, Cláudia, Daniel, e mais um montão de gente, que se fosse colocar o nome aqui, encheria uma página... Embora esses nomes não estejam aqui, estão no meu coração!

Aos professores do ICMC, pela atenção, disposição e carinho com que sempre me trataram. Aos funcionários do Instituto, da secretaria da Pós (Marília, Beth e Laura), biblioteca, LDCC (Sônia, Luciano e Franz), xerox (Jaques), zeladoras, Seu Zé e todos os que contribuem para a manutenção desse ambiente e bom desenvolvimento dos trabalhos.

Aos colegas da UNICENTRO, pelo apoio e incentivo na fase final do mestrado: Marcos Quinaia, Heródoto, Josiane, Luciane, Sandra, Lucélia e Marcos Bronoski.

Aos meus alunos, que em pouco tempo conquistaram meu carinho e me despertam para novos desafios.

Ao CNPq, pelo apoio financeiro.

Resumo

Esta dissertação descreve o processo de engenharia reversa do ASiA (Ambiente de Simulação Automático), desenvolvida usando a Fusion-RE, que é uma abordagem para a engenharia reversa de sistemas implementados sem orientação a objetos, produzindo modelos de análise orientados a objetos. A dissertação também apresenta uma revisão bibliográfica cobrindo: modelagem e simulação de sistemas, o ASiA (que é o ambiente simulação focado no trabalho) e manutenção de software (para justificar a escolha da engenharia reversa).

A documentação produzida para a versão atual do ambiente e os modelos gerados para o futuro processo de reengenharia são incluídos. O processo utilizado para a aplicação da Fusion-RE ao ASiA, a proposta para sua reengenharia e sugestões de melhorias a serem implementadas também são discutidos.

Abstract

This MSc dissertation describes the reverse engineering process for ASiA (Ambiente de Simulação Automático) developed by using the Fusion-RE, which is an approach for the reverse engineering of systems implemented without the object-oriented paradigm, producing object-oriented analysis models. This dissertation also presents a literature review covering: systems modeling and simulation techniques; the ASiA (the simulation environment focused in the work) and software maintenance (to justify the choice of the reverse engineering).

The documentation produced for the current environment version and the models generated for the future reengineering process are included. The process used for the Fusion-RE application to ASiA, the proposal for its reengineering and suggestion for improvements to be implemented are also discussed.

Índice

1. Introdução	1
2. Modelagem e Simulação de Sistemas Computacionais	4
2.1. Introdução	4
2.2. Modelagem	4
2.2.1. Classificação dos modelos	5
2.2.2. Técnicas de modelagem	5
2.2.3. Redes de Filas	6
2.2.4. Solucionando um modelo	10
2.3. Simulação	11
2.4. Processo de modelagem e análise de resultados	12
2.4.1. Fase de desenvolvimento	13
2.4.2. Fase de testes	14
2.4.3. Fase de análise de resultados	14
2.5. Ferramentas para simulação	15
2.5.1. Linguagens convencionais	16
2.5.2. Extensões funcionais	17
2.5.3. Linguagens de simulação	17
2.5.4. Pacotes específicos	17
2.6. Considerações finais	17
3. Ambiente de Simulação Automático (ASiA)	19
3.1. Introdução	19
3.2. Proposta inicial do Ambiente de Simulação Automático	20
3.2.1. Estrutura geral	20
3.2.2. Componentes básicos	21
3.3. Estado atual	24
3.3.1. Editor Gráfico	25
3.3.2. Gerador de Aplicações	28
3.3.3. Módulo Visual	30
3.3.4. Módulo Arquitetura	32
3.3.5. Módulo de Análise de Saídas	33
3.3.6. Projetos em desenvolvimento	34
3.4. Considerações finais	34
4. Manutenção de Software	36
4.1. Introdução	36
4.2. Processo de desenvolvimento de software e abstrações	37
4.3. O método Fusion	38
4.3.1. Fase de Análise	39
4.3.2. Fase de projeto	39
4.3.3. Fase de implementação	40
4.3.4. Características adicionais	40
4.4. Tecnologias usadas na manutenção de software	40

Índice	ii
4.4.1. Reestruturação	41
4.4.2. Engenharia reversa	41
4.4.3. Reengenharia	45
4.5. A abordagem Fusion-RE	45
4.5.1. Revitalizar a arquitetura do sistema com base na documentação existente	46
4.5.2. Recuperar o Modelo de Análise do Sistema Atual	47
4.5.3. Abstrair o Modelo de Análise do Sistema	50
4.5.4. Mapear o Modelo de Análise do Sistema para o Modelo de Análise do Sistema Atual	52
4.6. Considerações finais	53
5. Engenharia Reversa do ASiA	54
5.1. Introdução	54
5.2. Base de dados de apoio à abordagem Fusion-RE	55
5.2.1. Utilização da abordagem	55
5.2.2. Importância	57
5.2.3. Tabelas modeladas	57
5.2.4. Relatórios disponíveis	59
5.2.5. Estado do protótipo	61
5.3. Aplicação da Fusion-RE ao ASiA	62
5.3.1. Informações provenientes do sistema	62
5.3.2. Abstrações das características do sistema	64
5.3.3. Processos de execução	65
5.3.4. Documentos gerados	71
5.4. Considerações finais	73
6. Proposta para a Reengenharia do ASiA	74
6.1. Introdução	74
6.2. Processo para a reengenharia do ASiA	75
6.3. Alterações a serem implementadas	75
6.3.1. Sistema de simulação	75
6.3.2. Editor Gráfico	76
6.3.3. Módulo Visual	77
6.3.4. Demais módulos	77
6.4. Considerações finais	77
7. Conclusões	78
7.1. Contribuições	78
7.1.1. Contribuições para o ASiA	78
7.1.2. Contribuições para a Fusion-RE	79
7.2. Trabalhos futuros	80
7.2.1. Reengenharia do ASiA	80
7.2.2. Ferramenta de apoio à Fusion-RE	80
7.2.3. Aprimoramento da documentação	81
7.2.4. Linguagem adotada	82
7.2.5. Outras mudanças	82
7.3. Dificuldades encontradas	83
7.4. Considerações finais	83
Referências Bibliográficas	85
Bibliografia Auxiliar	89

Índice	iii
Anexos	90
A.1. Documento de Revitalização do Sistema	91
A.2. Procedimentos do ASiA (relatório alfabético)	124
A.3. Diagrama de Temas	129
A.4. Relatório de estruturas e elementos de dados	130
A.5. Relatório de classificação dos procedimentos por pseudo-classes	133
A.6. Modelo de Objetos do MASA (Tema temporário)	136
A.7. Modelo de Objetos do MASA (Tema permanente)	137
A.8. Modelo de Ciclo de Vida do MASA	138
A.9. Modelo de Operações do MASA	139
A.10. Procedimentos utilizados nas operações do ASiA	149
B.1. Modelo de Objetos do MAS	153
B.2. Especificação dos atributos para as classes do MAS	154
B.3. Especificação dos métodos das classes do MAS	155
B.4. Modelo de Ciclo de Vida do MAS	156
B.5. Mapeamento entre o MAS e o MASA	157

Índice de Figuras

Figura 2.1: Centro de serviço com uma fila e um servidor [Spo94, Spo94a]	6
Figura 2.2: Centro de serviço com uma fila e vários servidores [Spo94, Spo94a]	7
Figura 2.3: Centro de serviço com várias filas e um servidor [Spo94, Spo94a]	7
Figura 2.4: Centro de serviço com várias filas e vários servidores [Spo94, Spo94a]	8
Figura 2.5: Programa de simulação de uma fila M/M/1 em smpl [Mac87]	12
Figura 2.6: Processo de Modelagem e Análise de Resultados [Mac87]	14
Figura 3.1: Estrutura geral do Ambiente de Simulação Automático [San94a]	21
Figura 3.2: Visão geral do Editor Gráfico do ASiA [Spo94a]	26
Figura 3.3: Tela principal do Editor Gráfico [Bru97]	27
Figura 3.4: Visão geral do Gerador de Aplicações	29
Figura 3.5: Estrutura do ASiA com o Módulo Visual	32
Figura 3.6: Estrutura do Módulo Arquitetura	33
Figura 4.1: Relação entre os termos utilizados na manutenção de software [Chi90]	39
Figura 4.2: Documento de Revitalização da Arquitetura do Sistema [Pen96]	47
Figura 4.3: Classificação dos procedimentos de determinada classe	49
Figura 4.4: Modelo de Operações	50
Figura 4.5: Mapeamento entre o MAS e o MASA	53
Figura 5.1: Tela do protótipo da ferramenta de apoio à Fusion-RE	56
Figura 5.2: Tela para cadastro dos módulos e procedimentos	58
Figura 5.3: Formulário Classificação	59
Figura 5.4: Relatório de totais dos procedimentos classificados do ASiA	61
Figura 5.5: Diagrama da aplicação da Fusion-RE ao ASiA	63
Figura 5.6: Módulos do projeto do ASiA (asia.ide)	63
Figura 5.7: Trecho de código do arquivo tipos.h	64
Figura 5.8: Tela para cadastro dos temas (com os dados do ASiA)	66
Figura 5.9: Cadastro de estruturas de dados	67
Figura 5.10: Tela de associação entre estruturas e temas	67
Figura A.1: Tela do ASiA para a parametrização de um recurso	74
Figura A.2: Modelo da Operação Parametrizar recurso do MASA do ASiA	74

Índice de Tabelas

<i>Tabela 4.1: Método de engenharia reversa orientada a objetos [Pen96]</i>	46
<i>Tabela 4.2: Tabela para coleta de informações para a definição do Ciclo de Vida</i>	49
<i>Tabela 4.3: Correspondência entre procedimentos e métodos</i>	51
<i>Tabela 5.1: Temas definidos para o ASiA</i>	65
<i>Tabela 5.2: Classes do ASiA</i>	69

Introdução

O estudo de sistemas em geral dispõe de um conjunto variado de técnicas e ferramentas. Um exemplo é a técnica de modelagem, que constitui uma forma de representar com fidelidade o sistema em estudo, possibilitando sua análise.

Um modelo requer que sejam extraídas as características fundamentais para o funcionamento do sistema, adequadas para representar seu comportamento. Dentre as diversas técnicas de modelagem existentes, destacam-se Redes de Filas, Redes de Petri e Statecharts, que podem ser utilizadas tanto para a modelagem de sistemas computacionais quanto para sistemas de outras áreas.

Independente da técnica que é utilizada para a construção do modelo, sua resolução pode ser analítica ou por simulação. A solução analítica de um modelo corresponde à resolução de equações matemáticas que quantificam o sistema, e a solução por simulação requer que um programa de simulação seja construído para que, quando executado, reproduza o funcionamento do sistema real, gerando dados para análise.

Implementar um programa de simulação é uma tarefa simples para usuários experientes, mas extremamente complexa para usuários que não detêm conhecimentos tanto em simulação quanto em programação. No sentido de reduzir os problemas oriundos dessa complexidade, surgiram diversas ferramentas de apoio ao desenvolvimento e análise de programas de simulação, dentre os quais destacam-se os ambientes de simulação automáticos. Esses ambientes geram o código do programa de simulação a partir de um modelo fornecido pelo usuário e, normalmente, trazem ferramentas de apoio embutidas no sistema, visando facilitar a análise estatística dos dados produzidos pela execução do programa de simulação. Além de aumentar a produtividade para os usuários especializados, permitem aos demais a possibilidade de utilizar adequadamente os recursos básicos e avançados da área de simulação [Spo96].

Considerando as vantagens inerentes aos ambientes de simulação, o Grupo de Sistemas Distribuídos e Programação Concorrente, do Departamento de Computação e Estatística do

ICMC-USP, iniciou o desenvolvimento do Ambiente de Simulação Automático (ASiA). Atualmente em sua segunda versão, o ASiA é o resultado de diversos projetos, constituindo uma ferramenta de apoio para modelagem e simulação de sistemas. O objetivo principal do ASiA é facilitar a tarefa de transcrição dos modelos em programas de simulação, fornecendo uma interface gráfica para a construção de modelos em Redes de Filas e um gerador de aplicações para, a partir do modelo, ser produzido o código do programa de simulação em SMPL (que é uma extensão funcional da linguagem C para programas de simulação).

Após algum tempo de utilização, todo sistema necessita de manutenção. Da mesma forma, o ASiA requer um estudo minucioso, visando alguns ajustes, tanto em nível funcional quanto operacional, produzindo a redefinição de seus processos. Para auxiliar esse estudo, busca-se o embasamento da Engenharia de Software, especificamente manutenção.

As tecnologias utilizadas para a manutenção de software têm por objetivos corrigir erros do sistema, implementar modificações propostas pelos usuários, migrar o sistema para outro ambiente operacional e/ou melhorar seu desempenho.

Como previsto, durante a fase de estudo das tecnologias de manutenção existentes, optou-se por utilizar a engenharia reversa, a fim de gerar uma documentação abrangente da versão operacional do ASiA e resgatar as características propostas no projeto inicial. Para realizar a engenharia reversa, foi adotada a abordagem Fusion-RE, pelo fato de ser indicada para a engenharia reversa de sistemas que foram construídos sem a utilização do paradigma de orientação a objetos, produzindo os modelos de análise orientados a objetos, para posteriormente realizar a reengenharia do sistema. Essas características são adequadas ao ASiA e aos objetivos deste projeto.

Para abordar os principais temas relacionados a este projeto, incluindo o embasamento para a compreensão do ASiA e das tecnologias de manutenção de software, esta dissertação está organizada em sete capítulos, especificados a seguir.

No Capítulo 2, intitulado Modelagem e Simulação de Sistemas Computacionais, são apresentadas as definições e técnicas de modelagem de sistemas (especificamente Redes de Filas), além de discutir as etapas do processo de desenvolvimento que um programa de simulação deve seguir. São citadas também as ferramentas disponíveis para a construção desses programas.

O Capítulo 3 - Ambiente de Simulação Automático - descreve o projeto inicial do ASiA, discute os módulos que constituem sua versão atual e apresenta os projetos em andamento, bem como os projetos futuros.

Objetivando a busca de uma ferramenta adequada para o processo de manutenção do ASiA, o Capítulo 4 trata de Manutenção de Software, definindo o processo de desenvolvimento de software e explicando as principais técnicas de manutenção, tais como engenharia reversa, reengenharia e reestruturação. Cita como exemplo de método de desenvolvimento o Fusion, e descreve a abordagem de engenharia reversa Fusion-RE.

O Capítulo 5 apresenta o desenvolvimento do projeto, incluindo a descrição da base de dados de apoio utilizada e a forma como a abordagem Fusion-RE foi aplicada ao ASiA, detalhando as etapas de acordo com um diagrama de execução.

No Capítulo 6 é apresentada a proposta para a reengenharia do ASiA, resultante dos modelos apresentados no Capítulo 5, além de características a serem adicionadas ao ambiente.

No último capítulo são apresentadas as conclusões, incluindo as dificuldades encontradas na elaboração do projeto e as propostas para trabalhos futuros, em continuidade a essa dissertação.

O material utilizado em referências para embasamento deste trabalho está relacionado ao fim da dissertação.

Modelagem e Simulação de Sistemas Computacionais

2.1. Introdução

O estudo de sistemas computacionais, sob a ótica da avaliação de desempenho, requer que os problemas que estão sendo considerados sejam bem definidos, para depois serem solucionados. Nesse contexto, definir o problema pode significar estabelecer um modelo que represente o sistema que está sendo abordado para, a partir desse modelo, estudar o sistema.

A construção de modelos requer que o interessado detenha conhecimentos em relação tanto ao sistema em estudo quanto à técnica de modelagem aplicada. Considerando-se sistemas computacionais, algumas das técnicas de modelagem mais utilizadas são Redes de Filas [Soa92], Redes de Petri [Heu90] e *Statecharts* [Har87].

Para a análise do sistema a partir do modelo, uma solução deve ser construída, podendo ser analítica (levando a um modelo analítico) ou por simulação (resultando em um modelo de simulação).

Este capítulo é dedicado às definições gerais sobre técnicas de modelagem (em especial Redes de Filas) e simulação.

2.2. Modelagem

Um modelo é uma abstração¹ de um sistema, que representa uma visão particular do mesmo, podendo também ser visto como um conjunto de informações sobre o sistema, suficiente para estudá-lo seguindo determinada abordagem [Edw92, Mac87].

Tendo como base essa definição, modelagem corresponde à fase em que as características fundamentais do sistema são abstraídas, segundo a metodologia oferecida pela técnica

¹ Abstração é a habilidade de ignorar os aspectos não relevantes para o propósito em questão, dedicando maior concentração aos aspectos principais.

considerada, compondo o modelo. A solução do modelo poderá ser obtida analiticamente ou por simulação.

2.2.1. Classificação dos modelos

Para iniciar a construção de um modelo, é necessário abstrair do sistema real os componentes e suas interações, que são considerados suficientemente importantes para serem incluídos no modelo. Os componentes selecionados são referidos como entidades. Associados a cada entidade, podem existir atributos, que são valores que descrevem o estado da entidade e quais valores podem ou não variar com o tempo. A coleção de todos os atributos em um determinado tempo define o estado do sistema. A manipulação desses atributos de acordo com uma função de tempo é um caminho de operação para o sistema sobre o período de observação. O estado do sistema muda e essas mudanças são causadas por eventos [Mit92]. Evento é a ocorrência aleatória de um fato predefinido, que faz parte de um contexto geral determinado anteriormente.

As manipulações dos valores podem ser contínuas ou discretas em função do tempo. Em modelos de eventos discretos, as mudanças de valores ocorrem em instantes bem estabelecidos (referidos como tempo de evento) de forma determinística ou de forma aleatória, dependendo da natureza das entradas no modelo. Em modelos de eventos contínuos, as mudanças ocorrem continuamente ao longo do tempo de simulação. Existem ainda modelos de mudança combinada, onde as variáveis podem mudar de forma discreta, contínua ou contínua com saltos discretos. A forma como as mudanças ocorrem não altera as definições básicas de mudanças de estados [Soa92].

2.2.2. Técnicas de modelagem

A rigor, a maior parte dos estudos envolvendo modelos segue as características gerais de um sistema modelado anteriormente, isto é, utilizam modelos previamente construídos de acordo com as características em comum. Sendo esses modelos abstratos, eles são classificados de acordo com as estruturas usadas na sua elaboração, podendo ser construídos por Redes de Filas [Soa92], Redes de Petri [Heu90] e *Statecharts* [Har87], entre outros. O Ambiente de Simulação Automático (ASiA), assunto do próximo capítulo, apesar de conceitualmente poder adotar qualquer técnica de modelagem existente, no momento implementa modelos apenas em Redes de Filas [Oga97]. Devido a isso, Redes de Filas é a técnica de modelagem descrita a seguir.

2.2.3. Redes de Filas

Não é fácil prever a operação de sistemas complexos, onde há disputa pelo uso de serviços, devido a fatores como limitação das áreas de espera, existência de operações em paralelo e atividades simultâneas, múltiplas interações entre dispositivos e mecanismos de decisão complexos. Para que o funcionamento desses sistemas possa ser entendido, é necessária a utilização de métodos e procedimentos [Soa92].

Sistemas a serem modelados e simulados são compostos por recursos (por exemplo, a CPU de um computador) que são utilizados por clientes. Enquanto um recurso está sendo utilizado, ele pode ser requisitado por outros clientes, causando disputa por sua utilização e gerando filas de espera. O nível de disputa e a quantidade de recursos requisitados afetam o comportamento do sistema, sendo pontos que merecem atenção na análise. Se não há disputa, a análise de desempenho é simples [Soa92].

Uma Rede de Filas básica é formada por entidades chamadas centros de serviço e um conjunto de clientes.

Um centro de serviço é formado por um ou mais servidores, que são os recursos do sistema modelado, e por uma ou mais filas, que são as áreas de espera para os clientes que estão requisitando o serviço.

A Rede de Filas mais simples corresponde a um só centro de serviço, composto apenas por uma fila e um servidor, é representada na Figura 2.1. O sistema pode ser muito complexo, mas o modelo é simples. Um único centro de serviço, em um modelo de Redes de Filas, possui vários mecanismos relacionando os clientes com a área de espera e o servidor, e, embora o modelo não englobe toda a complexidade do sistema, possibilita que informações úteis sejam fornecidas [Soa92]. Para que os modelos sejam especificados mais detalhadamente, devem ser definidos alguns parâmetros. Em modelos com uma fila e um servidor, esses parâmetros são a disciplina da fila, a taxa de chegada de novos clientes e o tempo necessário para a execução do serviço [San94].

Figura 2.1: Centro de serviço com uma fila e um servidor [Spo94, Spo94a]

Um centro de serviços pode conter vários servidores (Figuras 2.2 e 2.4). Um servidor atende um cliente a uma taxa constante, ou a taxa de serviço pode ser dependente do número de clientes na fila de espera, que pode ter capacidade finita ou não [Soa92]. Uma variação do modelo básico contém um centro de serviço constituído por uma fila única, responsável por alimentar vários servidores, como ilustrado na Figura 2.2. Além da definição dos parâmetros do modelo anterior, precisa-se de uma forma de selecionar um servidor quando mais do que um está desocupado [San94].

Figura 2.2: Centro de serviço com uma fila e vários servidores [Spo94, Spo94a]

A Figura 2.3 exibe um modelo constituído por várias filas e um servidor. Nesse tipo de centro de serviço, é possível implementar uma fila para atendimento especial, com maior prioridade. É necessário definir os parâmetros como no modelo com uma só fila e um só servidor e, além disso, políticas para escolher uma das filas para retirar o próximo cliente a ser atendido e para definir que fila um cliente deverá ocupar ao entrar no sistema.

Figura 2.3: Centro de serviço com várias filas e um servidor [Spo94, Spo94a]

O modelo mais complexo constitui-se de um centro de serviço formado por múltiplas filas e múltiplos servidores (de acordo com a Figura 2.4). Também implementa a idéia de fila especial, com a variante que existem vários servidores disponíveis para o atendimento. Para detalhar o sistema, devem ser definidos todos os parâmetros citados para os outros modelos, inclusive políticas sobre escolha de filas e servidores.

Figura 2.4: Centro de serviço com várias filas e vários servidores [Spo94, Spo94a]

Os parâmetros básicos dos modelos em Redes de Filas são discutidos a seguir.

- **Tempo de serviço**

Está relacionado com o tempo necessário para que um servidor execute uma tarefa. É usualmente especificado por uma distribuição de probabilidade, embora, em alguns casos, possam ser utilizados valores médios, o que facilita a implementação, mas traz problemas de precisão. Por isso, na maioria dos casos, são utilizados valores obtidos por números aleatórios [San94].

- **Algoritmos de escalonamento**

A disputa por recursos acaba gerando filas de espera, onde os clientes aguardam a vez para serem atendidos. Existem vários algoritmos de escalonamento, também chamados disciplina da fila, responsáveis por manipular as filas de espera, decidindo qual cliente deverá entrar em serviço quando um servidor estiver disponível. Assim, uma disciplina de fila é uma regra para adicionar e remover clientes de uma fila de espera, cujo tamanho pode ser limitado ou não. Para simulação, os algoritmos mais usados são [Soa92]:

- ⇒ FCFS (First Come First Served): em uma fila de espera padrão, o primeiro cliente a chegar é o primeiro a ser atendido;
- ⇒ LCFS (Last Come First Served): implementado com a estrutura de uma pilha, o último cliente a chegar é o primeiro a ser atendido se o servidor estiver disponível;
- ⇒ RR (Round Robin): cada cliente é atendido durante um intervalo de tempo (chamado quantum). Se o serviço não for completado, ele entra novamente na fila de espera, para continuar sua execução posteriormente. O próximo cliente recebe um quantum para ser executado. Esse processo se repete até que o serviço requisitado pelo cliente seja completado;
- ⇒ Tempo de serviço: o cliente com maior ou menor tempo de serviço é retirado da fila. Para isso, é necessário que a fila se mantenha ordenada pelo tempo de serviço;

⇒ Uso de prioridades: os algoritmos com prioridades são utilizados para dar preferência a alguns clientes, podendo ser preemptivos (cliente com maior prioridade retira o que está sendo executado do servidor) ou não. Quando um cliente é retirado do servidor, ele só retorna quando todos os outros de maior prioridade já tiverem sido servidos.

- **Tempo entre chegadas**

É definido por uma função que determina o tempo da próxima chegada ao sistema. No caso mais simples, essa função pode ter valores constantes, mas não representa um sistema de modo realista. A função que descreve chegadas de clientes ao sistema deve produzir valores aleatórios que correspondam aos tempos entre chegadas no sistema real que está sendo modelado.

- **Algoritmos para escolha do servidor**

Quando um cliente requisita um serviço, e dois ou mais servidores estão disponíveis, é necessário escolher um deles para realizar o serviço. Para isso, utiliza-se um algoritmo de seleção (semelhante a uma disciplina de fila), que poderá escolher o servidor aleatoriamente, ou o que está a mais ou a menos tempo desocupado, ou de acordo com alguma prioridade [Mar80].

- **Algoritmos para escolha de qual fila retirar o próximo cliente a ser atendido**

Quando um servidor fica disponível, é preciso definir de qual fila o próximo cliente deverá ser retirado. Essa escolha pode ser feita segundo prioridades ou aleatoriamente.

- **Algoritmos para escolha de qual fila o novo cliente deverá ocupar**

Ao chegar a um centro de serviço, o cliente deverá optar por uma fila de espera, de acordo com alguma prioridade ou aleatoriamente [San94].

Notação

Um sistema de filas é representado por um descritor da forma $A/S/c/k/m$; onde:

- A distribuição de probabilidade de tempo entre chegadas
- S distribuição de probabilidade de tempo de serviço
- c número de servidores
- k número máximo de clientes permitido no sistema
- m número de clientes disponíveis na origem

Quando a população de clientes é assumida como infinita, m é usualmente omitido; da mesma forma, quando a capacidade do sistema é assumida como infinita, k é omitido [Mac87].

Os símbolos usados para descrever as distribuições de probabilidades são:

D tempo de serviço ou entre chegadas constante

M distribuição exponencial

Ek distribuição k-Erlang

Hk distribuição exponencial k-estágios

G distribuição geral (ou arbitrária)

A disciplina das filas pode ser adicionada ao descritor; caso contrário, assume-se como FCFS.

Com base nas definições anteriores, uma fila M/M/1 indica uma fila com um único servidor, permitindo uma população infinita de clientes, com disciplina FCFS, e o tempo entre chegadas e de serviço tem distribuição exponencial. Se o servidor estiver livre quando da chegada de um cliente, o serviço é iniciado; senão, o cliente vai para a fila de espera [Mac87].

2.2.4. Solucionando um modelo

Existem modelos distintos para resolução analítica ou por simulação [Edw92, Soa92]. Contudo, tendo-se um modelo construído, sua resolução pode ser obtida tanto por métodos analíticos, quanto através da execução de um programa de simulação.

Na resolução analítica, os resultados pretendidos são obtidos através de fórmulas matemáticas [Edw92, Soa92].

Para uma fila M/M/1, ilustrada na Figura 2.2, pode-se calcular a intensidade do tráfego (taxa de atendimento dos clientes) através da equação [Bha84]

$$\rho = \frac{\lambda}{\mu} \quad (1)$$

Onde:

λ = Tempo médio de serviço

μ = Tempo médio entre chegadas

Outro valor freqüentemente desejado é o comprimento médio da fila, que pode ser obtido por

$$E(Q|Q > 0) = \frac{E(Q)}{P(Q > 0)} \quad (2)$$

Onde:

$$E(Q) = \lambda * E(W + S) \quad (3)$$

Sendo $E(W + S)$ o tempo total do cliente no sistema, incluindo espera e atendimento, que é dado por

$$E(W + S) = \frac{1}{\mu * (1 - \rho)} \quad (4)$$

E $P(Q > 0)$ é a probabilidade da fila não estar vazia, o que é resultante da distribuição exponencial. Assim, o comprimento médio da fila será

$$E(Q|Q > 0) = \frac{\rho / (1 - \rho)}{\rho} = \frac{1}{1 - \rho} \quad (5)$$

A Figura 2.5 [Mac87] mostra um programa de simulação para a fila M/M/1 ilustrada na Figura 2.2.

Assim, pode-se dizer que, no caso da fila M/M/1, um modelo analítico é constituído pela representação da fila ilustrada na Figura 2.1 e pelas equações (1) e (5); enquanto um modelo para simulação é composto pela Figura 2.1 e pelo programa mostrado na Figura 2.5.

Ao finalizar a execução do programa da Figura 2.5, o relatório gerado pelo comando report() conterá valores sobre o período da simulação, utilização do servidor, média de tempo de ocupação do servidor, comprimento médio da fila, número de clientes atendidos e número de vezes que um cliente é colocado na fila [Mac87].

2.3. Simulação

Um estudo de simulação é essencialmente um experimento estatístico que requer ferramentas para a coleta e análise dos dados [Mit92].

Simular significa imitar. Portanto, para simular um sistema, é necessário descrevê-lo como um problema físico, iniciando sua resolução pela construção de um modelo [Meu92]. A resolução analítica pode ser utilizada quando é possível definir uma relação funcional entre os parâmetros do sistema e os critérios de desempenho escolhidos por meio de equações que podem

ser resolvidas analiticamente. Uma simulação pode ser definida pela execução de uma representação matemática/lógica, com o objetivo de imitar o comportamento do sistema [Soa92].

```
#include <smpl.h>

main()
{
    real Ta=200.0, Ts=100.0, te=200000.0;
    int customer=1, event, server;
    smpl(1, "M/M/1 Queue");
    server=facility("server", 1);
    schedule(1, 0.0, customer);
    while(time()<te)
    {
        cause(&event, &customer);
        switch(event)
        {
            case 1: /*arrival*/
                schedule(2, 0.0, customer);
                schedule(1, expntl(Ta), customer);
                break;
            case 2: /*request server*/
                if (request(server, customer, 0)==0)
                    schedule(3, expntl(Ts), customer);
                break;
            case 3: /*completion*/
                release(server, customer);
                break;
        }
    }
    report();
}
```

Figura 2.5: Programa de simulação de uma fila M/M/1 em smpl [Mac87]

2.4. Processo de modelagem e análise de resultados

O passo principal do processo de construção de um programa de simulação é o da abstração do projeto do sistema em um projeto do modelo, sendo que a experiência de quem constrói o modelo é muito valiosa [Mac87].

O processo de construção de um programa de simulação, envolvendo a análise dos resultados, proposto por Mac Dougall [Mac87], consiste nas fases de desenvolvimento, testes e análise de resultados, como ilustrado na Figura 2.6 e descrito a seguir. As etapas do processo, porém, não ocorrem sequencialmente como ilustrado, mas têm suas execuções sobrepostas, pois há interação entre elas.

2.4.1. Fase de desenvolvimento

A fase de desenvolvimento consiste em descrever a funcionalidade do sistema, abstraindo-a, de acordo com os objetivos da análise, em uma descrição do modelo. Essa descrição especifica os recursos a serem representados e suas interconexões, o trabalho a ser representado, seus atributos e as operações envolvidas para realizá-lo. A coleta dos dados dessa descrição determina o nível de detalhes do modelo. Na próxima etapa, é escolhido um método de análise apropriado e é desenvolvida uma implementação do modelo, que é o programa de simulação. Uma alternativa para substituir a continuidade do processo é fornecer o projeto do modelo e os parâmetros de entrada de dados para um pacote de análise. As etapas dessa fase são:

- ⇒ Descrição do sistema: aprendizado sobre o funcionamento do sistema e descrição das suas operações, fornecendo as bases para o desenvolvimento do modelo;
- ⇒ Abstração do sistema e descrição do modelo: representação do sistema contendo suas características essenciais, através de síntese (descrição em alto nível tem origem na combinação dos elementos do sistema e das atividades associadas) ou decomposição (sistema é visto como uma única entidade, trabalhando em mais alto nível, e o trabalho é decomposto nas principais atividades). Essa representação não deve ser construída com base em nenhum método de análise específico, pois a escolha do método de análise compõe uma etapa posterior do processo;
- ⇒ Coleta de dados: quando a descrição do modelo é finalizada, definem-se os parâmetros do modelo que devem ser especificados numericamente. Esses parâmetros (que serão os pontos de coleta) podem ser classificados como parâmetros de carga de trabalho (tempo entre chegadas, tempo de execução, requerimentos de armazenamento, tipo e tamanho de registros) ou como parâmetros do sistema (tempos de operação, tal como tempo de ciclo de memória). Um parâmetro deve ter um valor fixo, ou ser determinado de acordo com a distribuição estatística adotada;
- ⇒ Seleção do método de análise: baseada em parte no sistema enfocado e representação de dados, e em parte nas ferramentas e habilidades disponíveis. A resolução analítica sempre será escolhida quando já existir uma adequada à descrição do modelo. Simulação depende da forma como as considerações, a respeito de qual modelagem usar, são analisadas e do esforço necessário para o desenvolvimento. Se o esforço não é significativo, o programa de simulação é desenvolvido, sendo usado para checar os resultados analíticos;
- ⇒ Desenvolvimento do programa de simulação: similar ao desenvolvimento de qualquer outro programa computacional, sendo as principais fases: projeto do modelo de simulação, organização do programa, gerenciamento de parâmetros, suporte a depuração e instrumentação.

Figura 2.6: Processo de Modelagem e Análise de Resultados [Mac87]

2.4.2. Fase de testes

A fase de testes compreende três etapas: depuração, verificação e validação. Muito frequentemente, o programa de simulação é considerado depurado quando executa sem erros e produz resultados condizentes aos esperados. A verificação assegura que o programa de simulação é certamente uma implementação do modelo. A validação assegura que o modelo é uma representação razoável do sistema real.

2.4.3. Fase de análise de resultados

A fase de análise consiste nas considerações sobre estimativa e controle da precisão dos valores de saída da simulação, que são funções dos valores de entrada fornecidos através de amostras aleatórias ou de distribuições de probabilidades.

De modo geral, uma simulação pode ser transitória ou de regime permanente. Na simulação transitória, o tempo de simulação é quase sempre determinado pelo problema, assim como as condições iniciais; as medidas de interesse são definidas em termos do tempo requerido

para executar um conjunto de atividades específico, ou em termos do número de atividades necessárias para alcançar um estado específico.

Na simulação de regime permanente, as condições iniciais e a duração da simulação são determinadas pelo modelo. Como a simulação é executada infinitamente, as medidas de interesse são definidas por meio de um valor limitante especificado. É o tipo de simulação mais utilizado.

O ASiA utiliza o método de análise *batch means* (método para estimar um intervalo de confiança² para a média dos valores de saída da simulação), que é fornecido pelo SMPL [Spo94]. O método *batch means* trata o problema de *warmup*³ pela divisão de uma execução longa em um conjunto de execuções mais curtas, chamadas *batches*, calculando uma média separada para cada execução, e usando essas médias para calcular a média total e o intervalo de confiança.

2.5. Ferramentas para simulação

O uso generalizado de simulação como ferramenta de análise de sistemas fez com que surgisse uma série de linguagens, especificamente projetadas para esse propósito. Tanto as linguagens como os pacotes de modelagem determinam que os modelos sigam uma certa estrutura; esse fator faz com que as soluções desses modelos sejam simplificadas [Soa92].

Em geral, as ferramentas para simulação são linguagens convencionais, extensões funcionais, linguagens de simulação e pacotes específicos.

Qualquer linguagem algorítmica de propósito geral (Fortran, Basic, Pascal e C, por exemplo) é capaz de implementar o modelo desejado. Entretanto, linguagens de simulação especializadas tem muitas vantagens em termos de facilidade, eficiência e efetividade de uso, tais como [Peg91]:

- ⇒ redução da tarefa de programação;
- ⇒ orientação conceitual;
- ⇒ flexibilidade de atualização quando o modelo sofre mudanças;
- ⇒ redução de erros de programação;
- ⇒ obtenção automatizada de estatísticas.

Para as linguagens de simulação, um fator importante é a presença de mecanismos para representação de tempo. Enquanto o sistema que está sendo modelado executa em tempo real, a simulação trabalha com um relógio próprio, responsável por marcar a passagem de tempo no

² Tolerância especificada para a variância dos valores obtidos.

³ Forma de iniciar a simulação quando o sistema está em estado inativo.

programa de simulação [Mac87]. Outro fator importante é fornecer recursos para coleta de estatísticas e emissão de relatórios.

O desenvolvimento de novas linguagens deve enfatizar redução do tempo de desenvolvimento do modelo, maior precisão e melhor comunicação, enquanto o desenvolvimento de software para simulação enfatiza a facilidade de uso e um ambiente de simulação integrado, além dos benefícios que as linguagens de alto nível oferecem [Peg91].

A utilização de linguagens para simulação implica em falta de portabilidade, exigindo que o programador tenha um amplo domínio da linguagem. Para minimizar esses problemas, surgem as extensões funcionais, que apresentam os aspectos gerais de uma linguagem convencional associados ao caráter específico de uma linguagem destinada exclusivamente à simulação [Mac87].

Para a escolha de uma ferramenta para a construção de um programa de simulação, devem ser analisados vários fatores, tais como [Soa92]:

- ⇒ Portabilidade: disponibilidade da linguagem em outros computadores (desde microcomputadores até supercomputadores);
- ⇒ Flexibilidade: suporte oferecido a diferentes enfoques de simulação e diferentes conceitos de modelagem;
- ⇒ Capacidade de crescimento: disponibilidade de novos recursos, incluindo capacidade gráfica e de banco de dados;
- ⇒ Tempo de execução: velocidade de compilação e execução;
- ⇒ Capacidade de memória: gerenciamento adequado do espaço de armazenamento;
- ⇒ Depuração: facilidade de depuração e documentação adequada;
- ⇒ Aprendizado: facilidade no aprendizado da linguagem e treinamento disponível;
- ⇒ Considerações de processamento: disponibilidade de recursos para coleta de dados estatísticos, processamento de listas e geração de relatórios padrão e personalizados.

2.5.1. Linguagens convencionais

Para desenvolver um programa de simulação em uma linguagem convencional, o programador deve criar todo um ambiente necessário para a simulação, pois essas linguagens não oferecem as ferramentas necessárias a um ambiente de simulação, tais como estrutura de dados, facilidades para manipulação de listas, abstração de dados, coleta de estatísticas e emissão de relatórios.

A vantagem é que o programador não necessita aprender uma nova linguagem, podendo optar por uma que já é de seu conhecimento, desfrutando de todos os recursos oferecidos por ela. Porém, exige-se conhecimentos sobre simulação, para que um ambiente de simulação possa ser criado [Peg91].

2.5.2. Extensões funcionais

Utilizando-se dos recursos de uma linguagem convencional, as extensões funcionais eliminam as dificuldades relacionadas à criação de um ambiente de simulação, fornecendo bibliotecas (conjunto de rotinas) às linguagens convencionais (também chamadas hospedeiras). Com isso, tem-se um ambiente completo de simulação [Mac87].

2.5.3. Linguagens de simulação

As linguagens de simulação são projetadas para a modelagem de sistemas de vários tipos [Soa92]. Essas linguagens contêm todas as estruturas necessárias para a criação de um ambiente de simulação, livrando o programador da execução dessa tarefa. São classificadas em linguagens orientadas a evento, a atividade ou a processo, sendo que a escolha dependerá da orientação sobre a qual o modelo foi construído, sendo as mais utilizadas as orientadas a evento e a processo [Mac87].

2.5.4. Pacotes específicos

Os pacotes de simulação específicos são pouco flexíveis a mudanças, pois são destinados apenas à avaliação de sistemas para os quais foram projetados [San94].

Nesses pacotes, a modelagem é realizada na própria ferramenta, sendo os parâmetros do modelo especificados através de uma linguagem relacionada com o sistema modelado. Quando o sistema se adapta à linguagem de modelagem, é fácil construir o modelo e analisar o sistema. Assim, há pacotes específicos para modelagem de sistemas computacionais e seus sistemas operacionais, de redes de computadores, de sistemas aeroespaciais, etc [Soa92].

2.6. Considerações finais

A modelagem é uma técnica para estudo e avaliação de sistemas, sendo bastante utilizada quando sistemas computacionais são o foco da avaliação. A solução por simulação possibilita que sistemas onde a solução analítica não pode ser aplicada sejam modelados.

Para que a simulação seja usada como ferramenta para a solução de modelos, é necessário construir o programa de simulação. Isso requer que o usuário possua conhecimentos específicos

tanto do sistema a ser modelado quanto de programação, tornando-se uma tarefa complexa. Essa dificuldade é solucionada com a utilização dos ambientes de simulação automáticos, que permitem que usuários que não conhecem linguagens de simulação gerem o programa através do modelo construído, além de facilitar o trabalho de usuários mais experientes.

A importância de incluir modelagem e simulação nesta dissertação é fornecer embasamento para a compreensão do ASiA, que é o foco do projeto, descrito no próximo capítulo.

Ambiente de Simulação Automático (ASiA)

3.1. Introdução

Para que a simulação seja de fato uma ferramenta poderosa, permitindo a obtenção dos resultados finais pretendidos, existe um processo a ser seguido, como descrito no capítulo anterior (seção 2.4). Quando se deseja estudar um sistema real por meio de um programa computacional, a tarefa mais complexa pode ser a representação do sistema em um modelo, que servirá de base para a implementação do programa. Esse programa pode ser elaborado através de linguagens de programação de uso geral, linguagens específicas para simulação, ou extensões funcionais das linguagens de uso geral [Soa92, Mac87], citadas na seção 2.5.

Independente da ferramenta adotada, a tarefa de compor um programa de simulação não é tão complexa para pessoas especializadas, com conhecimentos tanto sobre simulação como programação em geral. Para tornar essa tarefa acessível para quem não detém tais conhecimentos e facilitá-la para os demais, a disponibilidade de um sistema que permita a geração automática de programas de simulação a partir da descrição do modelo em estudo é um fator significativo [San94a].

O Ambiente de Simulação Automático (ASiA) constitui um ambiente para a geração automática de programas de simulação, fornecendo suporte à geração de programas a partir de modelos construídos com base em diferentes metodologias. Potencialmente, o ASiA pode ser instanciado para gerar programas em linguagens ou sistemas de simulação diferentes. Com isso, apesar do ASiA ser destinado basicamente à avaliação de sistemas computacionais distribuídos, também pode ser utilizado em simulação de modo geral [San94a].

Primeiramente, este capítulo descreverá a concepção inicial do ASiA, incluindo as etapas definidas para o seu desenvolvimento. Depois, serão descritos os módulos implementados e as

melhorias adicionadas para alcançar a operacionalidade do sistema, fornecendo uma visão geral da versão atual do ambiente.

3.2. Proposta inicial do Ambiente de Simulação Automático

A proposta inicial do ASiA, descrita a seguir de acordo com [San94a], definiu diversos projetos de mestrado e iniciação científica para sua implementação. De modo geral, cada projeto desenvolvido está relacionado a um ou mais módulos do ASiA.

3.2.1. Estrutura geral

O objetivo básico de um ambiente de simulação automático é afastar o usuário da tarefa de transcrição do modelo em um programa de simulação. Em geral, o usuário fornece o modelo do sistema considerado, os parâmetros requeridos para a simulação e os pontos de coleta para análise por meio de um editor gráfico. As demais tarefas são executadas automaticamente pelo ambiente proposto.

Por outro lado, espera-se que o ambiente seja flexível o suficiente para permitir que os usuários mais experientes possam personalizar seus programas de simulação, permitindo alterações no código gerado, a fim de que características específicas sejam adicionadas.

Com base nessas definições, o ASiA deve permitir que os usuários descrevam os modelos (de acordo com a metodologia escolhida) através de um editor gráfico (usando primitivas gráficas amigáveis), definam os pontos de coleta de dados para análise estatística, selecionem os tipos de gráficos a serem apresentados no relatório final e escolham qual o sistema de simulação a ser utilizado (entre os disponíveis no ambiente). A partir disso, o ambiente se encarrega de compilar automaticamente todas as informações do modelo, incluindo as opções definidas pelo usuário, e gerar o programa de simulação correspondente (através de um sistema gerador de aplicações). Após gerado, o programa de simulação, que pode ser alterado pelo usuário (através de edição do código fonte), deve ser compilado e executado, o que também ocorre dentro do ambiente. Além disso, o ASiA possibilita aos usuários a análise detalhada dos dados coletados durante a execução do programa de simulação. Com isso, todas as etapas fundamentais do processo de elaboração e análise de uma simulação são feitas em um mesmo ambiente.

A estrutura básica do ambiente, ilustrada na Figura 3.1, é constituída pelos seguintes componentes:

⇒ Editor Gráfico: interface amigável que captura o modelo;

- ⇒ Gerador de Aplicações: módulo que produz automaticamente o programa de simulação;
- ⇒ Sistema de Simulação: ferramenta básica para a simulação;
- ⇒ Análise de Resultados: método para verificação e validação dos resultados;
- ⇒ Saída Gráfica: visualização gráfica dos resultados.

Os blocos da Figura 3.1 referidos como Programa de Simulação e como Execução, diferentes dos demais, ilustram estágios da utilização do ambiente e não módulos. Por exemplo, através da utilização do Editor Gráfico (para a captura do modelo) e das informações sobre o sistema de simulação (gabarito para o programa), o Gerador de Aplicações constrói o Programa de Simulação. Obtido esse programa, a próxima etapa é a Execução, onde são incorporadas as fases de compilação e geração do programa executável.

Figura 3.1: Estrutura geral do Ambiente de Simulação Automático [San94a]

3.2.2. Componentes básicos

A seguir, são especificadas, em linhas gerais, as características de cada módulo que deve compor o ASiA.

- **Editor Gráfico**

Para afastar o usuário da etapa de transcrição do modelo, o módulo de edição constitui uma interface gráfica amigável, oferecendo em um monitor de vídeo os elementos do modelo que representam o sistema a ser simulado. Esse editor oferece recursos para o desenho do modelo, permitindo modificações imediatas. Concluído o modelo, os parâmetros necessários para a simulação são requisitados, bem como os pontos de coleta de dados. O ambiente se encarrega de realizar as demais tarefas automaticamente.

O Editor Gráfico é responsável por uma série de funções, desde a interface entre o usuário e o ambiente de simulação automático até a geração das tabelas que contêm os dados necessários para que o Gerador de Aplicações construa o programa final. O Editor Gráfico é assim constituído:

- ⇒ definição de uma interface com o usuário: a interface gráfica oferecida pelo ambiente deve seguir os requisitos básicos para o projeto de interfaces (facilidade de aprendizagem, velocidade de utilização, baixa taxa de erros, etc). Para isso, segue a abordagem WYSIWYG (*What You See Is What You Get*), que se traduz como a visualização do modelo que está sendo definido passo a passo, através de menus *pop-down* e manipulação direta da tela [Fol87, Mar91];
- ⇒ definição do modelo a ser simulado: deve ser feita através da seleção de blocos que representam os elementos básicos existentes em modelos de redes de filas (que são filas, recursos, recurso com uma fila, recurso com várias filas, etc.) [Mar80, Soa92];
- ⇒ definição e verificação dos parâmetros do modelo: o usuário deve fornecer os parâmetros necessários para a caracterização do sistema em estudo, tais como política de escalonamento das filas, tempo de serviço, distribuição de chegadas, etc. [Peg91, Soa92];
- ⇒ formação da tabela de especificações: última tarefa executada pelo Editor Gráfico, implementa uma tabela contendo todas as informações necessárias para que o Gerador de Aplicações produza o programa de simulação. Dentre essas informações, estão a identificação dos recursos e eventos, inter-relacionamento dos recursos, parâmetros básicos, etc. [San94a].

• Gerador de Aplicações

Componente básico do ambiente, o Gerador de Aplicações não sofre interferências diretas do usuário. É alimentado pelas informações coletadas e organizadas em estruturas previamente definidas pelo Editor Gráfico, gerando através desses dados o programa de simulação, que é escrito na linguagem do sistema de simulação adotado, seguindo suas especificações.

Geradores de aplicações auxiliam usuários que não possuem conhecimentos razoáveis em programação, facilitam o trabalho de programadores experientes e ainda reduzem o custo de desenvolvimento de software [Spo94, Spo94b]. De modo geral, é um utilitário que, a partir de uma especificação em alto nível de um problema, transforma automaticamente essa especificação na implementação do problema, acrescentando qualidade nos sistemas produzidos, aumentando a produtividade e facilitando a manutenção e a escrita.

Genericamente, os geradores são compostos por 3 módulos [Mei91]:

- ⇒ módulo de interface com o usuário: torna mais amigável o uso do gerador, escondendo dificuldades da linguagem de especificação. Caso não exista, a especificação de entrada é textual;
- ⇒ analisador de especificações: responsável pela análise sintática e semântica das entradas, produzindo estruturas de dados intermediárias, que serão utilizadas pelo próximo módulo;
- ⇒ módulo gerador de produtos: gera o produto desejado pelo usuário através de um conjunto de rotinas, que acessam as estruturas de dados geradas pelo analisador de especificações e reconhecem descrições de produtos em uma linguagem própria, que deve indicar as manipulações necessárias na estrutura de dados intermediária e o formato final do produto.

Independente das vantagens proporcionadas pelos geradores de aplicações, sua adoção tem valor significativo para o ASiA, ainda mais considerando-se o suporte à utilização de diversos sistemas de simulação, baseados em diferentes linguagens. Com isso, através da mesma interface gráfica, poder-se-á obter códigos para execução em diversos ambientes de simulação, sem a necessidade de conhecer as particularidades de cada um [San94a].

• Sistema de Simulação

Um sistema de simulação deve ser adicionado ao ambiente para cuidar das características referentes ao programa de simulação que será gerado, sendo a linguagem a principal delas. Nesse caso, o usuário escolhe um dentre os sistemas de simulação disponíveis no ambiente, o que define em que linguagem o programa de simulação será gerado. Em princípio, qualquer sistema pode ser utilizado. Restrições provêm da plataforma de hardware adotada, que pode limitar os sistemas de simulação e os compiladores disponíveis.

O que mais tem sido utilizado para implementação de simulações são as extensões funcionais das linguagens de programação de uso geral (citadas na seção 2.5.2). O ASiA tem potencial para adotar vários sistemas de simulação, porém, inicialmente, será considerado apenas o sistema SMPL [Mac87], que é uma extensão funcional da linguagens C.

• Análise de Resultados

O módulo de Análise dos Resultados é responsável pelo controle da precisão dos resultados obtidos na simulação, através da execução de uma série de testes estatísticos [Mac87, Peg91, Soa92]. Como os resultados de uma simulação estocástica baseiam-se em um conjunto de

números aleatórios, fornecendo relatórios não exatos, essa é uma etapa importante. Para tanto, deve indicar se alguns parâmetros (tais como tempo total de simulação, número de execuções da simulação e utilização de *Warmup*⁴) são suficientes para que a precisão adequada seja alcançada. Esses testes são executados através de alguma ferramenta estatística que envolve a utilização de intervalos de confiança, replicações, análise por *batch means*⁵, etc.

Conceitualmente, a simulação deve ser executada até atingir a precisão estipulada. Se for alcançada, os resultados são considerados válidos; caso contrário, os parâmetros pertinentes devem ser alterados para que a precisão desejada possa ser obtida [San94a].

- **Saída Gráfica**

A Saída Gráfica é o módulo do ambiente que possibilita ao usuário uma saída onde é possível visualizar, através de gráficos, os pontos de coleta estabelecidos na fase de definição do modelo.

Basicamente, este módulo de saída deve ser bastante flexível, de modo a oferecer ao usuário diversas opções, incluindo visualização na tela (acompanhada ou não de impressão), ajuste de curvas, comparação entre resultados, cálculos estatísticos gerais, etc [San94a].

3.3. Estado atual

O ASiA está em desenvolvimento de acordo com as diretrizes especificadas na seção 3.2. Diversas melhorias têm sido acrescentadas, tanto em relação à funcionalidade quanto à interface.

Atualmente, a versão operacional do ASiA é constituída pelos módulos Editor Gráfico, Gerador de Aplicações e Módulo Arquitetura (módulo que fornece facilidades para a modelagem de arquiteturas de computadores), compilados e integrados com o Borland C++ 5.01, em ambiente de 32 bits (Windows 95).

O Módulo Visual, apesar de ter sido desenvolvido com sucesso, não faz parte da versão atual devido às dificuldades encontradas durante a fase de integração com os demais módulos. Como foi desenvolvido de acordo com o paradigma de orientação a objetos, diferente dos demais módulos, existem barreiras no que concerne às interações entre eles.

⁴ Modo de iniciar uma simulação com o sistema inativo, citado na seção 2.4.3.

⁵ Método de análise dos resultados de simulação, explicado na seção 2.4.3.

3.3.1. Editor Gráfico

A primeira versão do Editor Gráfico [Spo94a] foi construída no ambiente Windows 3.1, em linguagem C, utilizando o compilador Borland C++ versão 3.1, constituindo o módulo de interface entre o usuário e o Gerador de Aplicações do ASiA. A escolha do compilador deve-se às rotinas internas que implementam menus, caixas de diálogo, barras de rolagem e outros componentes para uma interface amigável.

O Editor Gráfico é responsável por [Spo94a]:

- ⇒ proporcionar ao usuário a interação com o ambiente ASiA;
- ⇒ captar todos os parâmetros do modelo;
- ⇒ dar manutenção à especificação gráfica (modelo e parâmetros);
- ⇒ estruturar as tabelas que serão utilizadas pelo Gerador de Aplicações.

O Editor Gráfico oferece aos usuários um conjunto de primitivas (barra de ferramentas, barra de menus, janelas, ícones, etc) para a descrição gráfica e parametrização dos modelos considerados. Apesar de poder dar suporte a diversos tipos de modelos, a versão atual implementa apenas modelos em redes de filas.

Durante a modelagem do sistema, o Editor Gráfico vai armazenando todos os símbolos gráficos e os parâmetros fornecidos pelo usuário (através das caixas de diálogo). Finalizado o modelo, o Editor Gráfico analisa a especificação gráfica, agrupando todas as informações necessárias para a geração automática de um programa de simulação em tabelas, que posteriormente constituirão arquivos que alimentarão o Gerador de Aplicações [Bru97].

Diversas melhorias foram acrescentadas ao Editor Gráfico [Oga97, Bru97]:

- ⇒ macros para personalizar o ambiente com ferramentas específicas para modelos definidos, tais como arquiteturas de computadores, redes de computadores, etc;
- ⇒ macros para implementar facilidades tornando o ambiente mais flexível ao usuário, tais como parametrização de um recurso através de duplo clique do botão esquerdo do mouse e possibilidade de movimentação dos recursos pela tela de edição;
- ⇒ Sistema de Ajuda implementado no modo padrão (*help on-line*) utilizando hipertexto⁶;
- ⇒ migração para Borland C++ 4.5, a fim de que os recursos fornecidos por essa linguagem fossem utilizados. Nessa migração, todos os módulos do ASiA foram

⁶ Texto não linear, onde se pode navegar entre os diversos tópicos do documento.

- remodelados e agrupados para constituírem apenas um projeto (isso porque os módulos eram incompatíveis, devido à inclusão, modificação e exclusão de funções);
- ⇒ substituição das caixas de diálogo para parametrização do modelo (recurso, chegada, fila e serviço), que eram chamadas cada qual de uma vez, por uma única caixa que possui botões que fazem chamadas às demais. Isso também acarretou mudanças no menu;
 - ⇒ nova migração para o Borland C++ 5.01, por ser um compilador feito para Windows 95. Foram necessárias algumas mudanças em nível de caixas de diálogos, menus e barra de atalho.

A Figura 3.2 apresenta a estrutura geral do Editor Gráfico. Todas as operações que o Editor Gráfico executa entre o modelo do usuário e o fornecimento dos arquivos para o Gerador de Aplicações visam manipular as informações obtidas da especificação gráfica e da parametrização do modelo, transformando-as em um conjunto de dados consistente e organizado que será utilizado pelo Gerador de Aplicações.

Figura 3.2: Visão geral do Editor Gráfico do ASiA [Spo94a]

O submódulo Interface com Usuário é responsável por fornecer ao usuário uma interface de fácil aprendizado e utilização, e também pela manutenção da parte gráfica, possibilitando

desenhar (com verificação automática de consistência), apagar elementos de um modelo, gravar arquivos, recuperar arquivos previamente definidos, etc.

A tela principal do Editor Gráfico, apresentada na Figura 3.3, mostra as barras de ferramentas e de menus.

A barra de ferramentas possibilita ao usuário:

- ⇒ desenhar os recursos do sistema que está sendo modelado (centros de serviços com uma ou várias filas e um ou vários servidores);
- ⇒ desenhar as entradas, as saídas e as ligações (inclusive ramificações) entre os centros de serviços, indicando qual o caminho a ser seguido por um cliente;
- ⇒ selecionar um recurso para parametrização;
- ⇒ apagar qualquer símbolo desenhado;
- ⇒ mover qualquer recurso na tela de edição.

Figura 3.3: Tela principal do Editor Gráfico [Bru97]

Com a barra de menus, é possível [Bru97]:

- ⇒ executar operações relacionadas com arquivos (criar um novo arquivo de especificação, abrir um arquivo de especificação existente completo, salvar a especificação de um modelo, salvar com outro nome a especificação que está sendo editada, e sair do ASiA);

- ⇒ fornecer os parâmetros do modelo e dos recursos, além de definir distribuições de probabilidade (na forma de tabelas);
- ⇒ gerar o código do programa fonte ou do executável para o modelo;
- ⇒ visualizar o código gerado e os resultados da simulação em um relatório;
- ⇒ ativar a barra de ferramentas do Módulo Arquitetura ou configurá-la;
- ⇒ ativar a ajuda on-line.

Na barra de ferramentas horizontal, localizada abaixo da de menus, existem botões que otimizam as chamadas às funções do menu: criar um novo arquivo para especificação, abrir um arquivo de especificação, salvar a especificação que está sendo construída, parametrizar o modelo, parametrizar o recurso selecionado, e esconder/mostrar a barra de ferramentas do Módulo Arquitetura.

3.3.2. Gerador de Aplicações

Simultaneamente ao desenvolvimento do Editor Gráfico, foi construído o Gerador de Aplicações [Spo94], em ambiente MS-DOS 5.0 e utilizando o compilador Turbo C. O Gerador de Aplicações toma a descrição gráfica capturada pela interface (e transformada em tabelas estruturadas pelo Editor Gráfico, contendo as especificações para a simulação) e, considerando o ambiente de simulação alvo (no caso, o SMPL), gera o programa de simulação final, baseando-se em um gabarito previamente definido (de acordo com as regras do ambiente de simulação).

O Gerador de Aplicações do ASiA é constituído pelos seguintes módulos [Spo94], como ilustrado na Figura 3.4:

- ⇒ Núcleo do Gerador: possibilita a geração dos principais comandos da extensão SMPL, responsáveis pelas atividades de preparação do modelo, definição e controle dos recursos, escalonamento e geração de eventos, geração de variáveis aleatórias, depuração do código, coleta de dados e emissão de relatório com estatísticas sobre a simulação;
- ⇒ Ferramentas: agrupadas em um conjunto, são utilizadas pelos diversos módulos do ambiente. São responsáveis pela execução de operações sobre as tabelas, manipulação de cadeias de caracteres (tornam a execução do Gerador de Aplicações mais eficiente), tratamento de mensagens de erros e auxílio na geração de eventos específicos;
- ⇒ Descrição de Produtos: através das tabelas fornecidas, o Gerador de Aplicações obtém todas as informações genéricas sobre o modelo e as particularidades de cada

um dos recursos que o compõem, além de informações sobre o escalonamento de eventos e definições de ciclos. A Descrição de Produtos é um tipo de gabarito, parecido com o produto final, onde trechos de código na linguagem alvo (SMPL) estão misturados com comandos que indicam ao Gerador de Aplicações como buscar as informações necessárias nas várias tabelas que serão utilizadas. Os trechos de código são copiados para o produto final, enquanto os comandos precedidos pelo caracter % determinam que primitivas do Gerador de Aplicações devem ser executadas. Dependendo do identificador que sucede o caracter %, um grupo de primitivas será considerado. Todo o processo de geração de código é controlado por um procedimento especial que identifica os comandos no gabarito e decide qual primitiva do Gerador de Aplicações deve ser executada. Cada uma dessas primitivas é responsável por criar um trecho específico no código final, utilizando as informações contidas nas diversas tabelas.

Figura 3.4: Visão geral do Gerador de Aplicações

Melhorias foram introduzidas ao Gerador de Aplicações [Pri96], tornando possível a representação de *tokens* nos programas gerados pelo ASiA, e aperfeiçoando a parte estatística.

Tokens representam as entidades ativas do sistema no SMPL [Mac87]. O comportamento dinâmico do sistema é modelado pelo movimento dos *tokens* através de um conjunto de recursos. Para que os programas de simulação gerados representassem *tokens*, fornecendo aos usuários a possibilidade de representação de tarefas de várias classes, com diferentes prioridades, foram

realizadas modificações tanto na estrutura do gabarito como nos arquivos do Gerador de Aplicações [Pri96].

Quanto à parte estatística, a implementação do Gerador de Aplicações permite o cálculo de duas probabilidades, que fornecem a porcentagem de vezes que o cliente não encontra fila no recurso e o número máximo e mínimo de clientes nas filas associadas aos recursos especificados [Spo94b].

Como melhoria, foi adicionada ao Gerador de Aplicações uma função do Módulo Visual, que permite que estatísticas sobre a utilização média de um recurso, o tempo médio de ocupação de um recurso e o tamanho médio da fila de um recurso sejam calculadas em três pontos da simulação: 1/3, 1/2 e 2/3, seja do tempo da simulação, do número de ciclos ou outro parâmetro definido pelo usuário. Para que seja possível o usuário do ambiente optar pelas estatísticas desejadas, devem ser realizadas alterações no Editor Gráfico, permitindo o acesso a essa função estatística [Pri96]. Portanto, esse recurso não é parte efetiva da versão atual do ASiA.

3.3.3. Módulo Visual

Como citado anteriormente, o Módulo Visual não faz parte da versão operacional do ASiA. A implementação do Módulo Visual foi realizada em ambiente Windows 3.1, usando o compilador Borland C++ 4.02 e o paradigma de programação orientada a objetos.

O Módulo Visual [Ped96] adiciona ao código produzido pelo Gerador de Aplicações funções de controle interligadas com a linguagem de simulação. Essas funções fornecem ao usuário, em tempo de execução, recursos para manipulação de parâmetros, eventos e *tokens*, e para geração de relatórios e gráficos.

Para que o Gerador de Aplicações utilize o Módulo Visual, é necessário que o usuário especifique parâmetros no Editor Gráfico, durante a criação do modelo.

O Módulo Visual tem o objetivo de facilitar a coleta e análise dos resultados do sistema simulado, de tal forma que o usuário do sistema possa finalizar a construção, verificação e validação do modelo e visualizar a evolução do processo de simulação no próprio ambiente gráfico. Devido a fatores como o elevado tempo de simulação da maioria dos modelos simulados, a possibilidade de observar o andamento da simulação em diversos níveis (tais como *tokens* e eventos gerados) e os valores de parâmetros, o Módulo Visual reduz o tempo de verificação, aumenta a produtividade e antecipa a análise de resultados do modelo simulado. A análise gráfica do modelo fornece ao usuário maior clareza nas informações [Ped96].

O paradigma de orientação a objetos foi adotado por fornecer:

- ⇒ propriedade de herança, que possibilita reescrever apenas o código necessário para a aplicação desejada;
- ⇒ suporte ao trabalho básico de programação em Windows, fornecido pelo OWL⁷.

A estrutura do Módulo Visual é composta, basicamente, por um manipulador de eventos, pela implementação de um modelo e pelo tratamento de visualização dos resultados.

Cada modelo implementado através do Módulo Visual recebe uma barra de menu, por meio da qual o usuário pode ter acesso às funções do Módulo Visual, tais como [Ped96]:

- ⇒ Rastrear: o usuário pode verificar o que está acontecendo internamente com o modelo simulado. Os eventos e *tokens* gerados são apresentados em uma caixa de diálogo, mediante o tempo de ocorrência;
- ⇒ Gráficos: o usuário pode requisitar uma quantidade ilimitada de gráficos, sendo que cada um é executado em uma janela distinta. O usuário, através de uma caixa de diálogo, opta pelos parâmetros que deseja utilizar na geração do gráfico;
- ⇒ Parâmetros: permite que o usuário altere valores definidos no início da execução do modelo e analise o comportamento do modelo com o novo valor configurado;
- ⇒ Utilitários: possibilita que o usuário tenha acesso aos relatórios, manipule a sequência de números aleatórios, determine o *batch means*, reinicie o modelo, etc;
- ⇒ *Stream*: o usuário pode mudar a semente de números aleatórios que está sendo usada pelo modelo para a geração de números aleatórios;
- ⇒ Relatório: opção através da qual o usuário pode verificar o relatório sobre o andamento das filas de recursos definidas no modelo simulado;
- ⇒ *Batch means*: procedimento sequencial, baseado no método *batch means*, que controla o tamanho da execução do modelo de simulação, além de estimar o valor médio de uma variável de simulação;
- ⇒ *Reset*: permite que o usuário reinicie a simulação do modelo, sem interromper a sua execução.

O Módulo Visual é parte integrante do programa fornecido pelo Gerador de Aplicações, ou seja, no programa fonte são incluídas chamadas a funções que tratam da análise das informações geradas pelo modelo e valores de parâmetros em tempo de simulação. O Módulo Visual permite o rastreamento de *tokens* e eventos ocorridos no modelo, a verificação dos valores dos parâmetros do modelo, a geração de gráficos dos parâmetros e recursos e a alteração

dos valores dos parâmetros, dentre outras funções. Quando um programa de simulação entra em execução, uma janela Windows é aberta, apresentando todas as opções do Módulo Visual, sendo essa interação mostrada na Figura 3.5. Nesse momento, a simulação é iniciada e só é interrompida se a condição de parada for atingida ou se o usuário selecionar a parada do programa.

A utilização do Módulo Visual não é exclusividade do Gerador de Aplicações do ASiA. Para utilizá-lo em outras aplicações, basta que o usuário conheça o compilador e o paradigma de programação orientada a objetos, e siga a estrutura por ele definida [Ped96a].

Figura 3.5: Estrutura do ASiA com o Módulo Visual

3.3.4. Módulo Arquitetura

O Módulo Arquitetura [Bru97], ilustrado na Figura 3.6, foi implementado com o objetivo de facilitar a construção de modelos que representem arquiteturas de computadores, buscando avaliar e validar o desempenho das arquiteturas através da simulação. Poderá, ainda, ser utilizado como uma ferramenta de auxílio ao ensino de arquiteturas de computadores.

Esse módulo é uma extensão do ASiA, que fornece recursos básicos (processadores, disco, memória) para que o usuário construa o modelo da arquitetura desejada. Além disso, oferece quatro modelos de arquiteturas pré-definidos e já parametrizados: von Neumann, von Neumann com *pipeline*, MIMD com memória compartilhada e MIMD com memória distribuída.

⁷ Object Windows Library: fornece os métodos básicos para a criação de uma aplicação Windows.

Os modelos pré-definidos podem ter seus parâmetros modificados, tendo em vista a influência de diferentes elementos no desempenho de cada arquitetura. Esses modelos podem ainda servir de base para a construção de modelos mais complexos.

Figura 3.6: Estrutura do Módulo Arquitetura

Assim, através do Módulo Arquitetura, o usuário pode:

- ⇒ utilizar uma arquitetura já definida;
- ⇒ utilizar como base uma arquitetura definida, alterando os parâmetros e a configuração do modelo;
- ⇒ definir uma nova arquitetura.

As alterações implementadas no Editor Gráfico, descritas na seção 3.3.1, possibilitaram que o ASiA fornecesse suporte ao Módulo Arquitetura.

3.3.5. Módulo de Análise de Saídas

O Módulo de Análise de Saídas [Oli96] foi desenvolvido com o objetivo de fornecer um conjunto de ferramentas para a análise e validação dos resultados da simulação gerados no ASiA.

A implementação desse módulo seguiu as seguintes etapas:

- ⇒ definição dos métodos estatísticos a serem adotados;
- ⇒ implementação e análise dos métodos selecionados;
- ⇒ determinação das alterações a serem efetuadas no ASiA.

A partir do modelo fornecido pelo usuário, o ASiA produz o programa de simulação. Para a Análise de Saídas, o modelo deverá conter o método de análise desejado, alterar os parâmetros de análise, transferir os valores observados da variável de interesse para o método, chamar o relatório que deverá mostrar o intervalo de confiança estimado e outros resultados. Dessa forma, quando o programa de simulação for gerado, essas informações farão parte do código produzido, contendo todas as chamadas necessárias às funções, chamando o método de análise escolhido ao ser executado.

Esse módulo deverá, ainda, passar pelas etapas de implementação e teste das alterações que deverão ser efetuadas no ASiA.

3.3.6. Projetos em desenvolvimento

Visando aperfeiçoar o ASiA, os seguintes módulos estão sendo desenvolvidos:

- ⇒ Suporte para Modelagem de Redes de Petri [Non97]: o usuário poderá optar entre modelar seus sistemas através de Redes de Filas (como na versão atual) ou através de Redes de Petri. O suporte para edição dessas redes já foi implementado, restando definir uma forma de ativá-lo no Editor Gráfico;
- ⇒ Extensão para Modelagem de Redes de Computadores [Oga97a]: possibilitará que redes de computadores sejam facilmente modeladas, incluindo recursos para representação dos meios de interconexão.

Futuros projetos previstos para o ASiA visam suporte para simulação distribuída, animação da execução da simulação, ferramentas para apoio à validação de modelos e a implementação de uma versão para ambiente UNIX (possivelmente LINUX).

3.4. Considerações finais

Em constante desenvolvimento, o ASiA constitui-se em um ambiente de simulação automático característico, que fornece ao usuário recursos gráficos para a construção de modelos e a codificação automática do programa de simulação correspondente, permitindo a visualização dos resultados através do relatório padrão gerado pelo SMPL.

Além disso, o ASiA possui extensão para modelagem de arquiteturas de computadores com alguns modelos previamente definidos. A versão atual (ASiA 1.2) utiliza o compilador Borland C++ 5.01. O módulo de Análise de Saídas foi desenvolvido, porém não integrado ao ASiA.

Vários projetos, alguns já em desenvolvimento, implementam novos módulos que serão adicionados ao ASiA, tais como o suporte à modelagem em Redes de Petri e a extensão para modelagem de redes de computadores.

Esse capítulo fornece uma visão aprofundada do ASiA, seguindo o estudo proposto neste projeto. A seguir, é incluída uma revisão sobre manutenção de software, para que seja escolhida a tecnologia mais adequada ao ASiA, visando os objetivos esperados.

Manutenção de Software

4.1. Introdução

Segundo Osborne e Chikofsky [Osb90], a variedade de problemas que envolve manutenção de software cresce constantemente, sendo que as soluções não acompanham essa evolução. Esses problemas são resultantes de código fonte e documentação mal elaborados, além da falta de compreensão do sistema.

A partir do momento em que um sistema começa a ser utilizado, ele entra em um estado contínuo de mudança. Mesmo que tenha sido construído aplicando as melhores técnicas de projeto e codificação existentes, os sistemas vão se tornando obsoletos em vista dos recursos que são disponibilizados.

As mudanças mais comuns que os sistemas sofrem são migrações para novas plataformas, ajustes para mudanças de tecnologia de hardware ou sistema operacional e extensões em sua funcionalidade para atender os usuários. Em geral, essas mudanças são realizadas sem que haja preocupação com a arquitetura geral do sistema, produzindo estruturas mal projetadas, documentação insuficiente, lógica e codificação ruins, sendo esses os focos que exigem manutenção em um sistema [Osb90].

Assim, manutenção de software é definida como o conjunto das atividades de correção de programas quando falhas são detectadas, de modificação quando os requisitos do usuário mudam (ou se deseja melhorar o desempenho), e de adaptação quando houver mudanças de ambiente [Pre92].

A manutenção envolve alterações em diversos níveis do sistema e, para tanto, requer ferramentas adequadas [Osb90]. Nesse contexto, as ferramentas CASE (*Computer-Aided Software-Engineering*) estão se difundindo de tal forma que deixaram de se preocupar apenas em tornar acessível o desenvolvimento de sistemas, mas também em expandir seu potencial para solucionar problemas de manutenção e aperfeiçoar sistemas existentes [Chi90].

Este capítulo fornece uma visão geral de desenvolvimento e manutenção de software, define as técnicas mais utilizadas para essa atividade, cita o método Fusion como exemplo de método de desenvolvimento e a Fusion-RE como exemplo de abordagem para engenharia reversa de sistemas.

4.2. Processo de desenvolvimento de software e abstrações

A adoção de um processo de desenvolvimento de software é essencial para o gerenciamento e alterações em sistemas, pelo fato de focar a manutenção em todas as etapas do processo de desenvolvimento [Os90].

Para que essa visão seja alcançada, é necessária a utilização de [Os90]:

- ⇒ disciplinas e técnicas para assegurar a qualidade e gerenciar a configuração;
- ⇒ avaliação periódica dos efeitos produzidos pelas mudanças no projeto, na estrutura e na lógica do sistema;
- ⇒ reengenharia e reuso de produtos de software (requisitos, projeto e código);
- ⇒ programação modular eficiente para obter um sistema melhor, cobrindo falhas na estrutura e na lógica do programa;
- ⇒ técnicas de medição para avaliar as melhorias no software quando qualquer mudança for realizada.

Para abordar adequadamente as técnicas de manutenção de software, deve-se primeiramente considerar três conceitos dependentes: a existência de um processo de desenvolvimento de software, a presença de um sistema a ser analisado e a identificação de níveis de abstração.

O sistema focado pode ser um único programa ou um fragmento de código, ou ainda um conjunto complexo de programas que interagem, instruções de controle de *jobs*, interfaces de sinais e arquivos de dados.

Qualquer que seja o processo de desenvolvimento de software, espera-se que haja interação entre seus estágios e, talvez, recursão. Em um processo de desenvolvimento de software, os estágios iniciais envolvem conceitos mais gerais, independentes da implementação, enquanto os estágios finais enfatizam os detalhes de implementação. O aumento de detalhes durante o processo de desenvolvimento conceitua os níveis de abstração. Estágios iniciais do sistema planejam e definem requisitos de alto nível quando comparados com a própria implementação.

Essa comparação é importante para deixar claro que nível de abstração e grau de abstração são grandezas distintas. Enquanto o nível de abstração é um conceito que diferencia os estágios conceituais do projeto, o grau de abstração é intrínseco a cada estágio. A evolução através das fases do processo de desenvolvimento de software envolve transições dos níveis mais altos de abstração nos estágios iniciais para níveis mais baixos nos estágios posteriores. As informações podem ser representadas em qualquer estágio do desenvolvimento, seja de forma detalhada (baixo grau de abstração), seja de forma mais sucinta ou global (alto grau de abstração). Essa evolução enfatiza o conceito de níveis de abstração entre as fases do processo de desenvolvimento.

Para que as técnicas de manutenção de software sejam descritas de forma simplificada, serão tomadas como base apenas três fases do processo de desenvolvimento de software, com níveis de abstração bem diferenciados (como ilustrado na Figura 4.1):

- ⇒ Requisitos: especificação do problema a ser resolvido, incluindo objetivos, restrições e regras de negociação;
- ⇒ Projeto: especificação da solução;
- ⇒ Implementação: codificação, teste e adaptação ao sistema operacional.

A técnica tradicional, que avança progressivamente pelas fases do processo de desenvolvimento de software, é denominada engenharia progressiva. A execução dessa técnica consiste em partir de projetos independentes da implementação, que possuem altos níveis de abstração, indo em direção à implementação física do sistema. Em outras palavras, engenharia progressiva segue a seqüência de desenvolvimento estabelecida no projeto, visando a obtenção do sistema implementado [Chi90]. Exemplo de método de desenvolvimento de software é o Fusion [Col96], descrito brevemente na próxima seção.

Na Figura 4.1, com exceção da engenharia progressiva, as demais transições entre as fases de desenvolvimento são tecnologias utilizadas na manutenção de software, descritas na seção 4.4.

4.3. O método Fusion

O Fusion [Col96] é um método de desenvolvimento de software destinado à construção de programas com a utilização do paradigma de orientação a objetos.

O método Fusion é híbrido, e integra e estende as melhores abordagens existentes para o desenvolvimento orientado a objetos. É um método abrangente, que fornece todos os recursos para as fases de análise, projeto e implementação.

Figura 4.1: Relação entre os termos utilizados na manutenção de software [Chi90]

4.3.1. Fase de Análise

O Fusion não engloba a fase de levantamento dos requisitos. O usuário deve fornecer um documento inicial de requisitos.

O analista define o comportamento esperado do sistema. São produzidos os modelos do sistema (Modelo de Objetos, Modelo de Ciclo de Vida e Modelo de Operações), que descrevem os seguintes elementos:

- ⇒ Classes de objetos que existem no sistema;
- ⇒ Relacionamentos entre essas classes;
- ⇒ Operações que podem ser executadas no sistema;
- ⇒ Sequências permissíveis para essas operações.

No Fusion, a associação de métodos a classes particulares ocorre na fase posterior à de análise.

4.3.2. Fase de projeto

O projetista decide como as operações do sistema serão implementadas pelo comportamento, em tempo de execução, dos objetos em interação. Diversas formas para subdividir uma operação em interações de objetos podem ser tentadas. Durante esse processo, as operações são associadas a classes. O projetista também escolhe como os objetos farão referências mútuas e quais os relacionamentos de herança serão apropriados entre as classes.

A fase de projeto gera modelos que mostram:

- ⇒ Como as operações do sistema serão implementadas pelas interações entre os objetos;
- ⇒ Como as classes farão referências mútuas e como estarão relacionadas através da herança;
- ⇒ Os atributos das classes bem como suas operações.

Os responsáveis pelo projeto podem ter de investigar com maiores detalhes as subestruturas de algumas classes e suas operações. Isso será feito com a decomposição hierárquica - aplicando as técnicas de análise e projeto a essas classes, considerando-as como um subsistema.

4.3.3. Fase de implementação

O responsável pela implementação deve transformar o projeto em código, usando uma linguagem de programação específica. O Fusion orienta como isso deve ser feito:

- ⇒ Herança, referências e atributos das classes serão implementados nas classes da linguagem de programação;
- ⇒ As interações entre os objetos são codificadas na forma de métodos pertencentes a uma classe selecionada;
- ⇒ As seqüências permitidas para as operações são reconhecidas por máquinas de estado.

4.3.4. Características adicionais

Ao final de cada fase, o Fusion faz verificações de consistência para garantir que os modelos construídos estão em concordância mútua e também concordam com os resultados da fase anterior.

O Fusion mantém um dicionário de dados onde as diferentes entidades existentes no sistema podem ser identificadas e descritas. Esse dicionário servirá de referência durante todo o processo de desenvolvimento.

O método Fusion pode ser aplicado à reengenharia de sistemas desenvolvidos sem o uso da orientação a objetos para obter modelos de análise baseados em orientação a objetos. Outro fator proporcionado pelo método é a reutilização de software.

4.4. Tecnologias usadas na manutenção de software

A evolução das tecnologias de manutenção (ferramentas e métodos) deve fornecer meios adequados de esclarecer, ao próximo programador, usuário ou gerente, o porquê uma mudança

foi feita e como ela afeta o sistema. Além disso, ferramentas devem ajudar a assegurar que o software não forneça apenas resultados corretos, mas que também satisfaça os requisitos do usuário, estendendo essa verificação durante toda a sua vida útil [Os90].

Segundo Chikofsky e Osborne [Chi90, Os90], a evolução das tecnologias tem gerado conflitos quanto aos vários termos que a elas se referem. Para esclarecer isso, são definidas algumas tecnologias aplicadas à manutenção de software, como descrito a seguir.

4.4.1. Reestruturação

A reestruturação é a transformação de uma forma de representação para outra no mesmo nível relativo de abstração, enquanto preserva o comportamento externo do sistema que está sendo analisado (funcional e semanticamente) [Chi90]. É a tecnologia aplicada para construir um projeto com a mesma função que o anterior, porém com maior qualidade [Pre92].

Freqüentemente, a reestruturação é usada para melhorar visualmente o programa, tal como alterar seu código para melhorar a estrutura (quando programação estruturada é utilizada). O termo “reestruturação” originou-se da estruturação de um código desestruturado, tendo, ainda, um significado abrangente que reconhece a aplicação de uma transformação similar e técnicas de remodelagem para formatar novamente modelos de dados, planos de projetos e estruturas de requisitos.

A reestruturação cria novas versões que implementam ou propõem mudanças no sistema abordado, mas, normalmente, não envolve modificações devido a novos requisitos. Apesar disso, sugere mudanças para melhorar aspectos do sistema.

É mais comum usar reestruturação como forma de manutenção preventiva, visando melhorar o código do sistema no que concerne à preferência de alguns padrões (identação é um exemplo). Além disso, deve também considerar ajustes no sistema, procurando outras restrições de ambientes que não envolvam reavaliação nos níveis mais altos de abstração [Chi90].

4.4.2. Engenharia reversa

O termo “engenharia reversa” originou-se da análise de hardware, que extraía o projeto a partir do produto final. Em geral, é aplicada para melhorar um produto ou para analisar um produto concorrente ou de um adversário (principalmente em situação militar ou de segurança nacional).

Aplicando o conceito inicial de engenharia reversa a sistemas de software, muitas das técnicas utilizadas em hardware servem para obter uma compreensão básica do sistema e sua

estrutura. Entretanto, enquanto o objetivo básico para hardware é duplicar o sistema, os objetivos mais frequentes para software são obter uma compreensão suficiente em nível de projeto para auxiliar a manutenção, fortalecer o crescimento do sistema, e substituir o suporte [Chi90].

A engenharia reversa é o processo de analisar um sistema com a finalidade de criar sua representação de uma forma diferente ou em um nível mais alto de abstração do que o código fonte [Chi90, Pre92]. Além disso, a engenharia reversa analisa um sistema para identificar os seus componentes e inter-relacionamentos [Chi90].

A engenharia reversa geralmente envolve a extração de características do projeto e das abstrações de construção ou síntese que são menos dependentes da implementação. É mais comum que a engenharia reversa seja realizada em um sistema funcional existente; porém, pode ser aplicada em apenas um nível de abstração ou estado do processo de desenvolvimento [Chi90, Pre92].

Pode-se dizer que a engenharia reversa é uma ferramenta para verificação e não para mudança ou replicação.

A engenharia reversa cobre vários estágios do processo de desenvolvimento de um sistema: inicia na implementação existente, recaptura e recria o projeto, e decifra os requisitos atualmente implementados no sistema [Chi90].

A engenharia reversa requer que quatro propriedades sejam abstraídas de um sistema [Cho90]:

- ⇒ estrutural: propriedade descrita por meio da troca de recursos⁸ entre módulos e subsistemas através de interfaces de interconexão;
- ⇒ funcional: propriedade descrita pela semântica da troca de recursos. Por exemplo, para os recursos operacionais (aquele que executa uma operação) são abstraídas condições relativas à execução. Para recursos não operacionais (aqueles que armazenam um valor) são abstraídas definições de tipos de dados;
- ⇒ dinâmica: para identificar as propriedades dinâmicas de um módulo, é necessário converter, de forma clara, uma implementação de um algoritmo procedural para uma linguagem que utiliza maior nível de abstração; isso deve-se ao fato das propriedades dinâmicas serem intramodulares;
- ⇒ comportamental: propriedade descrita pelo comportamento dos seus objetos (módulos) em termos das relações entre eles, dos seus atributos, e das ações que os manipulam. As principais propriedades comportamentais são intramodulares.

⁸ Nesse contexto, recursos de um sistema podem ser, por exemplo, tipos de dados, procedimentos e variáveis.

O principal propósito de aplicar técnicas de engenharia reversa em um sistema de software é obter uma compreensão global do sistema para manutenção e novos desenvolvimentos. O desenvolvimento das técnicas de engenharia reversa é motivado por alguns fatores, tais como [Chi90]:

- ⇒ enfrentar a complexidade: o desenvolvimento de métodos deve reduzir o volume e a complexidade dos sistemas. Métodos e ferramentas de engenharia reversa combinados com ferramentas CASE fornecem meios de extrair informações relevantes, mas as decisões para controlar o processo e o produto na evolução do sistema dependem do usuário;
- ⇒ gerar visões alternativas: representações gráficas auxiliam na compreensão dos sistemas. Entretanto, a criação e a manutenção dessas representações continuam sendo tarefas complexas. As ferramentas de engenharia reversa facilitam a geração de representações gráficas de diversas formas, produzindo outros modelos a partir dos primários, e auxiliando o processo de revisão e verificação do sistema. Por exemplo, os diagramas de fluxo podem gerar esquemas de estrutura, diagramas de controle de fluxo e diagramas entidade-relacionamento. Também pode ser criada alguma forma de representação não-gráfica para fazer parte do sistema de documentação;
- ⇒ recuperar informações perdidas: sistemas grandes, de vida longa, que evoluem continuamente, tendem a descaracterizar seu projeto inicial. Frequentemente, as modificações não são refletidas na documentação, particularmente se tiverem sido realizadas em um alto nível do próprio código. Enquanto não existem formas eficientes de preservar o projeto, a engenharia reversa, em especial recuperação de projeto (subárea de engenharia reversa definida a seguir), é o meio de proteger tudo que se pode do sistema existente;
- ⇒ detectar efeitos de mudanças: casualmente, o projeto inicial e suas sucessivas modificações podem levar a ramificações não pretendidas e efeitos que impedem o bom desempenho do sistema devido a características sutis. A engenharia reversa fornece informações mais detalhadas do que a engenharia progressiva, ajudando a detectar anomalias e problemas antes que os usuários as percebam;
- ⇒ sintetizar a maior abstração: a engenharia reversa requer métodos e técnicas para criar visões alternativas que transcendam o mais alto nível de abstração. Existem discussões sobre a forma como esse processo pode ser completamente automatizado. A tecnologia de sistemas especialistas poderá ativar em maior quantidade o potencial de geração de abstrações de alto nível;

⇒ facilitar o reuso: um resultado significativo no avanço da capacidade de reutilização de software se deve à grande quantidade de softwares existentes. A engenharia reversa pode ajudar a detectar componentes de software que serão candidatos à reutilização no sistema abordado.

A engenharia reversa tem se tornado um componente importante para as ferramentas CASE. Devido ao fato do processo de desenvolvimento ser uma atividade interativa, as ferramentas de engenharia reversa podem fornecer uma maior integração nesse processo e na sua manutenção.

Existem muitas subáreas de engenharia reversa, dentre as quais destacam-se redocumentação e recuperação de projeto [Chi90].

- **Redocumentação**

A redocumentação é a criação ou revisão de uma representação alternativa de um programa semanticamente equivalente e com o mesmo nível de abstração. As formas resultantes dessa representação são usualmente consideradas opções de visões para uma apresentação, tais como fluxo de dados, estrutura de dados e fluxo de controle.

É a forma mais simples e antiga de engenharia reversa, sendo uma forma fraca de reestruturação. Basicamente, implica em substituir a documentação do sistema que existiu ou deveria existir [Chi90].

- **Recuperação de projeto**

A recuperação de projeto, sub-área da engenharia reversa, utiliza o conhecimento sobre o domínio da aplicação, as informações externas, e a dedução ou o raciocínio nebuloso como formas de melhor observar o sistema, identificando abstrações significativas, de alto nível, ao invés das obtidas pela análise direta do próprio sistema [Chi90].

De acordo com Biggerstaff [Big89], a recuperação de projeto recria abstrações do projeto a partir de uma combinação de código, documentação do projeto existente (se disponível), experiência pessoal e conhecimentos gerais sobre os domínios do problema e as aplicações. Deve reproduzir todas as informações necessárias para a completa compreensão da funcionalidade do programa, incluindo “o que” faz, “como” faz e “por que” faz. Isso explica a grande quantidade de informações encontradas nas representações e códigos da engenharia de software convencional.

4.4.3. Reengenharia

Também conhecida como renovação ou reaproveitamento, a reengenharia, além de recuperar informações do sistema, usa essas informações para alterá-lo ou reconstituí-lo em uma nova forma, procedendo à implementação dessa reconstituição [Chi90]. É executada objetivando um projeto de melhor qualidade, tenha ele a mesma funcionalidade ou implemente mudanças [Chi90, Pre92].

Dentre as mudanças executadas pela reengenharia, que não alteram a funcionalidade do sistema, estão a troca da linguagem, a mudança de orientação, a realocação de funções e o fornecimento de um sistema de funcionalidade equivalente [Sne95].

A reengenharia combina a obtenção de características de análise e projeto proveniente da engenharia reversa (para efetuar uma descrição mais abstrata), com algumas técnicas de reestruturação (a serem aplicadas na lógica, na arquitetura e nos dados do programa). Embora envolva diversas técnicas, não é um conjunto delas [Chi90, Pre92].

Os objetivos da reengenharia são [Sne95]:

- ⇒ facilitar a manutenção: consiste em alterar o sistema de forma a reduzir as manutenções nele realizadas. Se esse for o único objetivo, a reengenharia de um sistema não é justificada, pois seus benefícios são muito difíceis de avaliar;
- ⇒ migrar o sistema: consiste em reescrever o sistema para substituição do ambiente operacional. Esse objetivo demonstra claramente que a funcionalidade do sistema não é alterada;
- ⇒ reduzir os erros: testes exaustivos exigidos para demonstrar a equivalência da funcionalidade poderão revelar erros até então escondidos no software. A reestruturação tem maior potencial para trazer os defeitos à tona, e a análise de falhas mede facilmente os resultados dessa redução;
- ⇒ preparar para o aperfeiçoamento funcional: a decomposição de programas em módulos menores com a finalidade de melhorar a estrutura e os isolar, facilitando a alteração ou adição de funções sem afetar os demais módulos.

4.5. A abordagem Fusion-RE

A ausência de diretrizes concretas para a realização de engenharia reversa orientada a objetos de sistemas desenvolvidos sem seguir essa abordagem foi a motivação para a elaboração da abordagem Fusion-RE [Pen96]. O embasamento no método Fusion, desenvolvido inicialmente para o projeto de sistemas e baseado na abordagem de orientação a objetos, se deve à estruturação de suas etapas de desenvolvimento, bem como o relacionamento entre elas. A

relação existente entre a análise e o projeto é de grande valia no processo retroativo (parte do projeto em direção a análise), que caracteriza a engenharia reversa. Além disso, o Fusion utiliza técnicas bastante conhecidas no que diz respeito à facilidade de utilização.

A Fusion-RE preocupa-se em investigar e propor passos seguros e bem definidos, mostrados na Tabela 4.1 e descritos no decorrer desta seção, para que um sistema desenvolvido sem a abordagem de orientação a objetos possa ser recuperado no nível de modelagem da análise, usando um método orientado a objetos.

Tabela 4.1: Método de engenharia reversa orientada a objetos [Pen96]

Passo	Objetivo
1. Revitalizar a arquitetura do sistema com base na documentação existente	Obter informações relacionadas à arquitetura do sistema para o seu entendimento
2. Recuperar o Modelo de Análise do Sistema Atual	Obter um modelo de análise considerando somente os aspectos físicos
2.1. Definir temas	Modelar em temas as informações armazenadas relativas às entradas, saídas, armazenamento permanente e temporário
2.2. Desenvolver o Modelo de Objetos	Elaborar um modelo com as classes e seus relacionamentos, extraídos dos tipos abstratos de dados que compõem a base de dados do sistema
2.3. Definir o Ciclo de Vida	Mostrar o comportamento global do sistema
2.4. Abstrair e desenvolver as operações	Obter as operações realizadas pelo sistema
3. Abstrair o Modelo de Análise do Sistema	Obter um modelo de análise do sistema, considerando os aspectos do domínio de aplicação
3.1. Desenvolver o Modelo de Objetos	Elaborar um Modelo de Objetos, considerando as classes e seus relacionamentos, que devem ser tratados pelo sistema
3.2. Elaborar o Modelo de Ciclo de Vida	Fornecer uma visão global do comportamento do sistema a partir da abstração realizada
3.3. Especificar o Modelo de Operações	Descrever como as operações devem ser realizadas
4. Mapear o Modelo de Análise do Sistema para o Modelo de Análise do Sistema Atual	Descrever a relação entre o Modelo de Análise do Sistema Atual e o Modelo de Análise do Sistema

4.5.1. Revitalizar a arquitetura do sistema com base na documentação existente

A revitalização do sistema consiste em listar todos os módulos que compõem o sistema a partir do código fonte existente, incluindo os procedimentos de cada um. Deve-se incluir uma descrição de cada módulo e de cada procedimento. Além disso, para cada procedimento devem ser relacionados os procedimentos que efetuam chamadas a ele e os que ele chama.

Como resultado dessa etapa, obtém-se o Documento de Revitalização da Arquitetura do Sistema, ilustrado na Figura 4.2.

<u>Módulo</u>	<u>Descrição</u>
Módulo1.h	Este módulo refere-se às operações realizadas sobre Módulo1.h
<u>Procedimentos</u>	
Proc1	Procedimento referente às operações realizadas por proc1.
	<u>Chama:</u> proc2 proc3 (Módulo2.h) proc4
	<u>Chamado por:</u> proc3 (Módulo2.h) proc5

Figura 4.2: Documento de Revitalização da Arquitetura do Sistema [Pen96]

4.5.2. Recuperar o Modelo de Análise do Sistema Atual

Com base no código fonte que implementa o sistema, a execução desse passo deve produzir o Modelo de Análise do Sistema Atual (MASA), que é constituído pelos modelos de Objetos, de Operações e do Ciclo de Vida, por meio dos sub-passos citados a seguir. Esses três modelos provêm do método Fusion.

- **Definir temas**

Os temas são constituídos pelas informações que o sistema manipula. São determinados por: entrada (interface), processamento (incluindo informações temporárias e permanentes armazenadas) e saída. O termo “tema” equivale à definição de cenários⁹ da orientação a objetos.

- **Desenvolver o Modelo de Objetos**

O Modelo de Objetos é construído a partir de abstrações da implementação tradicional. Cada tipo de implementação deve seguir um método de abstração distinto.

Se a implementação utiliza tipos estruturados de dados, para produzir a documentação de apoio a essa fase do método, deve-se:

- ⇒ consultar o código para descobrir quais são as estruturas de dados relevantes, tanto globais quanto locais. As estruturas de dados, em geral, contém ponteiros que são responsáveis pelas ligações existentes entre elas, podendo haver duplicação de chaves;
- ⇒ construir um diagrama simples com as estruturas de dados lógicas recuperadas e os relacionamentos entre elas;

⁹ Sequência de eventos entre agentes e o sistema criado com algum propósito; são representados em diagramas de tempo.

- ⇒ converter esse diagrama para a notação do método Fusion e acrescentar as demais características semânticas (cardinalidade¹⁰, papéis¹¹, etc);
- ⇒ recuperar, a partir desse diagrama, quais procedimentos atuam sobre essas estruturas de dados lógicas.

As estruturas de dados são definidas como pseudo-classes no MASA. No decorrer do texto, serão referenciadas apenas como classes.

Os procedimentos devem ser classificados de acordo com a tarefa que realizam em cada classe. Dessa forma, um procedimento será **construtor** de uma classe (identificado como **c**) quando realizar alterações em uma classe. Se apenas consultar, será **observador** da classe (identificado por **o**). Caso um procedimento não se refira a classe alguma, seja relacionado apenas à interface ou ao módulo escalonador, será classificado como **i**.

Os procedimentos poderão ser classificados da seguinte forma:

- oc** observador de uma classe e construtor de uma classe;
- o+c** observador de mais de uma classe e construtor de uma classe;
- oc+** observador de uma classe e construtor de mais de uma classe;
- o+** observador de mais de uma classe;
- c+** construtor de mais de uma classe;
- o+c+** observador de mais de uma classe e construtor de mais de uma classe.

Se um procedimento for observador e construtor de uma determinada classe, será classificado apenas como construtor. Se um procedimento executar tarefas em mais de uma classe, ele será associado a apenas uma classe, seguindo as prioridades:

- ⇒ será relacionado com a classe à qual desempenha a função de construtor se for do tipo **oc**;
- ⇒ será relacionado com a primeira classe analisada à qual desempenha a função de construtor se for do tipo **oc+**;
- ⇒ será relacionado com a primeira classe analisada à qual desempenha a função de observador se for do tipo **o+c**.

Essa classificação é realizada por classe de objetos, como ilustrado na Figura 4.3. A notação utilizada para a classificação dos procedimentos é:

`<nome_proc> <[tipo/módulo]>`

¹⁰ Número de classes que podem estar mutuamente associadas a um relacionamento.

¹¹ Nome que qualifica a classe participante de um relacionamento.

onde:

<nome_proc>: nome do procedimento no código fonte analisado

tipo: o ou c, de acordo com a função exercida pelo procedimento na classe

módulo: módulo ao qual o procedimento pertence

No caso do procedimento ser associado a mais de uma classe, a notação utilizada é:

<nome_proc> <[tipo/módulo]> <[tipo_1/classe_1/tema_1]> ... <[tipo_i/classe_i/tema_i]>

onde:

classe_i: refere-se à classe adicional a que o procedimento se relaciona

tema_i: tema ao qual a classe_i pertence

Classe:			
Procedimentos:			
A1	[o/m1]		o
A2	[c/m2]	[o/c1/t1]	oc

Figura 4.3: Classificação dos procedimentos de determinada classe

• Definir o Ciclo de Vida

O ciclo de vida é definido a partir dos eventos de entrada e saída do sistema, identificados por meio da observação da interface durante a utilização do sistema. As seqüências de operações permitidas são registradas em cenários, que reunidos e generalizados, formam a base para a definição do ciclo de vida.

Essa análise das operações pode ser auxiliada e validada com o apoio dos Manuais de Utilização do sistema (se existirem).

Para auxiliar a coleta de informações dessa etapa, que são relacionadas às chamadas de procedimentos quando o sistema atual é ativado, pode-se utilizar a Tabela 4.2.

Tabela 4.2: Tabela para coleta de informações para a definição do Ciclo de Vida

Opções de interface	Operações	Hierarquia de chamadas

A primeira coluna é preenchida com as opções fornecidas na interface do sistema e, na segunda, são listadas as operações relacionadas às opções. A terceira coluna consiste dos nomes dos procedimentos que implementam as operações. Se alguma operação for chamada por um procedimento do tipo i (não realiza operação específica, apenas tarefas relacionadas à interface), devem-se listar os procedimentos até o segundo nível de chamada.

Essa tabela serve de apoio para a construção de expressões regulares que compõem o ciclo de vida, de acordo com o seguinte gabarito:

life cycle [Nome:] Expressão-Regular

(nome-local = Expressão-Regular)*

- **Abstrair e desenvolver as operações**

Para cada operação identificada anteriormente (Tabela 4.2), é construído um Modelo de Operações, como ilustrado na Figura 4.4, tendo como base as informações obtidas no código fonte.

Operação:	nome da operação
Descrição:	descrição da operação
Lê:	[<i>supplied</i>] itens de dados
Modifica:	[<i>new</i>] itens de dados
Envia:	agentes e eventos
Assume:	condição (pré-condição)
Resultado:	estado (pós-condição)

Figura 4.4: Modelo de Operações

Por definição do método Fusion, quando um item de dado Lê vem precedido do termo *supplied*, ele é um parâmetro da operação; quando em Modifica o item (identificador de objeto) vem precedido por *new*, a execução do procedimento cria um novo objeto na base do sistema.

4.5.3. Abstrair o Modelo de Análise do Sistema

A execução das etapas anteriores, que tem como base as informações obtidas no código existente, produzem o MASA. A partir dessa fase, será criada a abstração da visão física para a visão lógica, denominado Modelo de Análise do Sistema (MAS). A visão lógica abstrai características que já deviam ter sido especificadas e não foram.

Os documentos gerados são semelhantes aos do MASA, mas com enfoque na aplicação, e não na implementação como anteriormente. As informações sobre o MAS podem ser armazenadas em um Dicionário de Dados (de acordo com a proposta do Fusion). No MAS, a denominação de elementos de dados e procedimentos é substituída, respectivamente, por atributos e métodos.

Da mesma forma que o MASA, para se obter o MAS é necessário executar os sub-passos descritos na sequência.

• Desenvolver o Modelo de Objetos

As classes do MASA são abstraídas para originar as classes do MAS. Um conjunto de classes e relacionamentos no MASA pode dar origem a uma classe no MAS, pois os objetos de implementação escolhidos (tais como listas, vetores de ponteiros, etc) devem ser abstraídos para um conceito do sistema. O contrário também pode ocorrer, uma classe do MASA gerar mais de uma no MAS, pelo fato de ter mais de uma instanciação concreta que corresponde aos conceitos do sistema.

No MAS, os atributos recebem nomes mais significativos, de acordo com os conceitos do mundo real que representam, e são analisados, pois alguns são desnecessários (são dependentes da implementação) e outros apontam para classes, sendo substituídos por relacionamentos. Os atributos também podem ser transferidos para a superclasse ou subclasse, de acordo com as relações de generalização/especificação respectivamente.

São considerados procedimentos anômalos os que agregam funções de observador ou construtor a mais de uma classe. Nessa etapa da abordagem, esses procedimentos podem ser acrescentados ao Modelo de Objetos, eliminando-se as anomalias. Os procedimentos classificados como *o* e *c* se transformam em métodos do MAS; os procedimentos com anomalias originam tantos métodos quantas são as classes por eles manipuladas. Os procedimentos *i* não dão origem a métodos. A correspondência na transformação é mostrada na Tabela 4.3.

Tabela 4.3: Correspondência entre procedimentos e métodos

Classificação dos procedimentos no MASA	Classificação dos métodos no MAS
<i>o</i>	<i>o</i>
<i>c</i>	<i>c</i>
<i>o+</i>	<i>o</i> /classe 1/tema <i>k</i> ; <i>o</i> /classe 2/tema <i>j</i> ; ...
<i>c+</i>	<i>c</i> /classe 1/tema <i>k</i> ; <i>c</i> /classe 2/tema <i>j</i> ; ...
<i>oc</i>	<i>o</i> /classe 1/tema <i>n</i> <i>c</i> /classe 2/tema <i>m</i>
<i>oc+</i>	<i>o</i> /classe 1/tema <i>i</i> <i>c</i> /classe 2/tema <i>k</i> ; <i>c</i> /classe 3/tema <i>y</i> ; ...
<i>o+c</i>	<i>o</i> /classe 1/tema <i>k</i> ; <i>o</i> /classe 2/tema <i>n</i> ; ... <i>c</i> /classe <i>m</i> /tema <i>t</i>
<i>o+c+</i>	<i>o</i> /classe 1/tema <i>n</i> ; <i>o</i> /classe 2/tema <i>f</i> ; ... <i>c</i> /classe <i>j</i> /tema <i>p</i> ; <i>c</i> /classe <i>k</i> /tema <i>s</i> ; ...

O Modelo de Objetos produzido contém as classes e relacionamentos, de acordo com a notação proposta no método Fusion. Também é obtida a listagem das classes pertencentes ao

modelo, com seus atributos e métodos (da mesma forma que os procedimentos dos módulos e elementos de dados no MASA). Ainda, seguindo o Fusion, podem ser acrescentados os agentes¹² como classes, embora os limites do sistema já estejam bem definidos pela implementação atual.

- **Elaborar o Modelo de Ciclo de Vida**

O Modelo de Ciclo de Vida do MAS constitui-se do Modelo de Ciclo de Vida do MASA, apenas alterando os nomes dos eventos de entrada e saída para nomes mais significativos; por consequência, os nomes das operações também devem ser alterados. Se a funcionalidade do sistema enfocado não for alterada, o Modelo de Ciclo de Vida aqui gerado será praticamente o mesmo do MASA.

Com base nas informações cadastradas na Tabela 4.2, no Documento de Revitalização do Sistema e no conhecimento do engenheiro de software, é elaborado o Modelo de Ciclo de Vida do MAS, com o mesmo *layout* do Modelo produzido no MASA.

- **Especificar o Modelo de Operações**

As operações identificadas anteriormente, que têm seus nomes substituídos por outros mais significativos, devem ser reespecificadas de acordo com o nível de abstração do MAS. Essa reespecificação consiste em traduzir o Modelo para a notação do Fusion sem alterar a funcionalidade das operações, considerando as classes, atributos e relacionamentos do Modelo de Objetos do MAS. O Modelo deve ser especificado como ilustrado na Figura 4.4. As operações passam a ser os métodos dos objetos.

4.5.4. Mapear o Modelo de Análise do Sistema para o Modelo de Análise do Sistema Atual

A importância de correlacionar o MAS e o MASA é facilitar o entendimento e a implementação futura (identifica os elementos do MAS que já estão implementados quando se deseja aplicar a reutilização, principalmente se o paradigma de programação for mantido).

Para cada uma das classes do MAS é elaborada uma tabela como a da Figura 4.5, relacionando seus equivalentes no MASA. Se a funcionalidade do sistema for alterada, as classes que implementam essas novas funções não terão equivalência no MASA. Para cada classe são listados os elementos de dados que correspondem aos atributos e os procedimentos que

¹² Agente é uma entidade do ambiente do sistema que ativa operações (através do envio de eventos) e recebe resultados (transportados pelos eventos).

correspondem aos métodos associados. Se um procedimento originou mais de um método, ele será repetido na correspondência ao MASA tantas vezes quanto o número de métodos gerados.

MAS	MASA
Modelo de Objetos	
Classe: <i>nome dado no MAS</i>	<i>nome dado no MASA</i>
Atributos	Elementos de dados
Métodos	Procedimentos

Figura 4.5: Mapeamento entre o MAS e o MASA

4.6. Considerações finais

Considerada a etapa mais dispendiosa do ciclo de vida de software, a manutenção dispõe de diversas tecnologias, metodologias e ferramentas.

Neste capítulo, foram abordadas definições sobre manutenção de software, situando-a no processo de desenvolvimento. Como exemplo, foi descrito o método de desenvolvimento de software orientado a objetos Fusion. Além disso, o Fusion serve como embasamento para a abordagem de engenharia reversa Fusion-RE, escolhida para ser aplicada ao ASiA. O processo de engenharia reversa do ASiA é o assunto do próximo capítulo.

Engenharia Reversa do ASiA

5.1. Introdução

A etapa do projeto destinada à seleção de um método ou ferramenta para manutenção de software resultou na escolha da engenharia reversa que, como descrito na seção 4.4.2, tem como objetivo principal a obtenção de uma compreensão global do sistema, tanto para sua manutenção como para novos desenvolvimentos, o que é adequado à meta estabelecida para esta dissertação. Isso é feito, basicamente, através de quatro propriedades abstraídas do sistema: estrutural, funcional, dinâmica e comportamental.

Iniciou-se, então, a busca por ferramentas e/ou métodos para engenharia reversa. A escolha pela abordagem Fusion-RE, desenvolvida no ICMC-USP, pelo grupo de Engenharia de Software, foi motivada pelo fato de que fornece, detalhada e sistematicamente, diretrizes seguras que conduzem à engenharia reversa de sistemas desenvolvidos sem orientação a objetos para modelos de análise que seguem o paradigma de orientação a objetos. Isso se encaixa perfeitamente às características do trabalho descrito nesta dissertação.

Segundo Dietrich [Die89], há duas razões para implementar sistemas orientados a objetos usando sistemas existentes que não seguem esse paradigma, o que reforça essa escolha:

- ⇒ os sistemas satisfazem às necessidades da maioria dos seus usuários; havendo necessidade de melhor programação de sua interface para ampliação ou atualização de suas funções;
- ⇒ possibilidade de reuso de partes do código para a implementação de funções do sistema.

Dessa forma, tendo-se optado pela adoção da abordagem Fusion-RE para a engenharia reversa do ASiA, este capítulo descreve a base de dados desenvolvida para apoio à Fusion-RE e a aplicação, passo a passo, da abordagem escolhida ao ASiA.

5.2. Base de dados de apoio à abordagem Fusion-RE

A necessidade da base de dados de apoio à Fusion-RE, descrita na seção 4.5, surgiu para minimizar a tarefa de documentação envolvida no processo, pois as aplicações anteriores que a utilizaram foram apoiadas por editores de texto.

Objetivando facilitar o processo, foi adotado o Microsoft® Access. Utilizando-se os recursos de bases de dados, é possível dispor de facilidades tais como integridade, confiabilidade e segurança das informações, além de reduzir digitação e otimizar buscas.

Estudando a abordagem Fusion-RE, pode-se perceber que as informações coletadas durante a execução da engenharia reversa são repetidas em diversos pontos da documentação gerada. Além disso, essas informações são altamente relacionadas. Esses fatores induzem ao projeto de uma base de dados, onde essas informações possam ser facilmente armazenadas, relacionadas e recuperadas.

A definição das tabelas e seus relacionamentos tem como base a descrição dos passos da Fusion-RE e os relatórios são construídos seguindo os padrões propostos na abordagem.

Embora o protótipo desse apoio computacional tenha sido desenvolvido para auxiliar a aplicação da Fusion-RE ao ASiA, é importante ressaltar que ele pode ser utilizado para a engenharia reversa de qualquer sistema similar ao ambiente em questão, além de poder ser estendido a sistemas com características diferentes, sofrendo algumas alterações pouco significativas.

5.2.1. Utilização da abordagem

Um fator a ser considerado é o tempo gasto entre a escolha da abordagem e o início da sua aplicação. Para se utilizar a Fusion-RE, é necessário estudar o seu processo de execução, identificando a importância de cada passo no resultado final esperado.

Utilizando-se um apoio computacional especializado, é possível dispensar essa etapa através de um acompanhamento para a aplicação da abordagem. Esse acompanhamento deve fornecer um método bem definido para a aplicação da abordagem, permitindo a obtenção dos resultados pretendidos através do fornecimento de informações suficientemente detalhadas de cada etapa. Com isso, espera-se que a abordagem seja compreendida em seu contexto geral por intermédio da compreensão de cada uma de suas etapas, o que vai acontecendo à medida que a aplicação da abordagem vai sendo realizada.

A existência de formulários explicativos (em forma de *helps*), contendo a forma como as informações devem ser cadastradas e quais suas relações com os demais dados, é significativa para que o estudo prévio da abordagem seja descartado.

Outro fator a ser considerado é que alguns sub-passos, que constituem passos da abordagem, são formados por um conjunto de tarefas mais amplo. Nesses sub-passos, a especificação é maior, pois resultam de conjuntos de informações bem delimitados que devem ser fornecidos ou por relatórios/diagramas que podem ser obtidos. A Figura 5.1 é uma tela do protótipo desenvolvido, que exemplifica a subdivisão dos passos que constituem a segunda fase da abordagem.

Dispondo-se dessas facilidades, o maior conhecimento necessário para iniciar a aplicação da abordagem é sobre o sistema que sofrerá a engenharia reversa. Conhecimento básico do paradigma de orientação a objetos é necessário para a obtenção dos modelos de análise envolvidos.

Figura 5.1: Tela do protótipo da ferramenta de apoio à Fusion-RE

5.2.2. Importância

A base de dados de apoio à abordagem é uma forma de difundir a utilização da Fusion-RE, pois pretende facilitar significativamente sua aplicação, fornecendo tarefas bem determinadas com o objetivo final de produzir o Modelo de Análise do Sistema que servirá de base para a reengenharia. Pretende, ainda, aumentar a confiabilidade dos modelos produzidos, através da consistência das informações envolvidas. A utilização de referências cruzadas entre as tabelas aponta para informações mais bem estruturadas em relação ao processo de execução manual.

Além disso, é possível validar cada um dos passos (e sub-passos) da abordagem em vista de um sistema relativamente complexo, detalhando minuciosamente os passos que a compõem.

As informações do sistema que estejam cadastradas nessa base de dados podem ser obtidas facilmente, além de combinadas, resultando em relatórios auxiliares não previstos na abordagem, mas que podem ser úteis para a compreensão do sistema e agilidade da execução da engenharia reversa.

5.2.3. Tabelas modeladas

Para cada grupo de informações dos passos da Fusion-RE foram criadas no MS-Access as tabelas para armazenamento.

- **Passo 1**

Para obter o documento de revitalização do sistema, foram modeladas as tabelas Módulo, Procedimento e Chama, e estabelecidos os relacionamentos entre elas. A tabela Módulo armazena os nomes de todos os módulos em linguagem C (os arquivos fonte .c e .cpp) que compõem o sistema em questão, com uma descrição sucinta do conteúdo de cada um.

A tabela Procedimento mantém as funções (ou procedimentos) do sistema, incluindo uma breve descrição da funcionalidade de cada um e o módulo a que pertence.

A associação "chama/chamado por" é armazenada na tabela Chama. É necessário cadastrar as informações referentes aos procedimentos que uma determinada função chama. As informações associadas aos procedimentos chamados por, que são mostrados no formulário Módulos e procedimentos (ilustrado na Figura 5.2), resultam de uma consulta à tabela Chama. Isso é possível com o uso dos recursos de bases de dados.

Para as tabelas Módulo e Procedimento, foram definidas chaves sequenciais fornecidas pelo Access como identificadores únicos. Na tabela Chama, a chave é composta, limitando a

apenas um registro para cada relação procedimento/procedimento chamado. Para a inclusão dos dados nessas tabelas é necessário primeiro cadastrar as funções. Após isso, são incluídos os procedimentos que são chamados pelas funções (tabela Chama).

- **Passo 2**

O Passo 2 da abordagem constitui-se em uma série de outras etapas. A primeira delas é Definir temas (de acordo com a Tabela 4.1). Para os temas, foi modelada a tabela Tema, que utiliza um código sequencial como chave e os campos para cadastro do tema (entrada, saída ou processamento), nome genérico pelo qual será referenciado e descrição do tema.

Figura 5.2: Tela para cadastro dos módulos e procedimentos

Para cadastrar as informações referentes às estruturas de dados e seus elementos foram construídas as tabelas Estrutura de dados (com campos para o código e o nome da estrutura) e Elemento de dados (que inclui código, nome do elemento e código da estrutura a que pertence). As informações são incluídas nessas tabelas através do formulário Estruturas de dados. Para cada estrutura, são cadastrados seus elementos.

A tabela Classificação, com os campos Estrutura, Procedimento e Classificação permite que cada procedimento tenha associado a si as estruturas de dados que manipula, bem como a classificação da forma como a manipula (observador ou construtor). Para que a tabela receba essas informações, é utilizado o formulário Classificação (ilustrado na Figura 5.3).

A tabela Classes e temas armazena o relacionamento entre as estruturas de dados (pseudo-classes) e os temas definidos para o sistema, ou seja, em qual(is) tema(s) a estrutura é utilizada. O formulário Classes e temas permite que essas informações sejam adicionadas à tabela.

Para auxiliar a elaboração do Modelo de Ciclo de Vida, foram modeladas as tabelas Opções da interface e Operações. Nessas tabelas, através do Formulário Opções da interface, são cadastradas as opções do menu do sistema e as operações por elas disparadas.

Figura 5.3: Formulário Classificação

5.2.4. Relatórios disponíveis

À medida que as tabelas para armazenamento dos dados de determinada etapa da abordagem foram modeladas, os relatórios referentes a essas informações foram construídos, de acordo com os padrões da Fusion-RE. Alguns deles são auxiliares, não fazem parte da documentação gerada (portanto, não têm padrões a seguir).

Os relatórios que podem ser obtidos por meio do protótipo são comentados a seguir.

- **Relatório de Módulos e procedimentos (Documento de Revitalização do Sistema)**

O Documento de Revitalização do Sistema é produzido com base nas informações coletadas no Passo 1 (tabelas Módulo, Procedimento e Chama), seguindo o formato especificado pela Fusion-RE. O Anexo A.1 contém esse documento para o ASiA.

- **Relatório de Procedimentos (alfabético)**

Para que as funções sejam localizadas mais facilmente (no Documento de Revitalização, os procedimentos e funções são agrupados por módulo), foi construído um relatório alfabético, que consiste em listar todos os procedimentos que compõem o sistema em questão, ordenando-os alfabeticamente pelo nome. Ao lado de cada procedimento, aparece o módulo a que pertence. O Anexo A.2 apresenta esse relatório para os procedimentos do ASiA.

- **Relatório de Temas**

As informações abstraídas, referentes aos temas do sistema, que são cadastradas na tabela Tema, devem ser apresentadas graficamente. Esse relatório é apresentado no Anexo A.3.

- **Relatório de Estruturas e Elementos de dados**

Os elementos de dados, cadastrados na tabela Elemento de dados, são listados em um relatório, agrupados pelas estruturas às quais pertencem (cadastradas na tabela Estrutura de dados). Esse relatório auxilia na elaboração do Modelo de Objetos do MASA. O Anexo A.4 contém esse relatório como os dados do ASiA.

- **Relatório de Classificação dos Procedimentos por Pseudo-classes**

O relatório de classificação contém as informações das tabelas Estrutura de dados, Procedimentos e Classificação. No MASA, as estruturas de dados são chamadas pseudo-classes. Para cada pseudo-classe, são listados os procedimento que a manipulam, o módulo a que esse procedimento pertence e a classificação de seu comportamento na classe (construtor ou observador). O Anexo A.5 mostra esse relatório com os dados do ASiA.

- **Relatório de totais de procedimentos classificados**

O relatório de totais permite visualizar com clareza o estado da implementação atual do sistema. Procedimentos anômalos são aqueles que possuem funções diversas em estruturas de dados diferentes. Quanto maior o número desses procedimentos, mais complexo se torna modelar o MAS a partir do MASA. Outra quantidade importante a ser analisada é o número de procedimentos do tipo i (são aqueles implementados para operacionalizar o sistema, dizem respeito à interface ou detalhes da implementação que não constituem procedimentos com ações definidas em alguma estrutura de dados). A Figura 5.4 mostra os dados dos procedimentos do ASiA.

<i>Procedimentos classificados (totais)</i>	
<i>Classificação</i>	<i>Procedimentos</i>
c	25
c+	1
i	163
o	47
o+	2
oc	6
oc+	1
Total:	245

Figura 5.4: Relatório de totais dos procedimentos classificados do ASiA

• **Procedimentos utilizados nas operações**

Embora esse relatório não faça parte da abordagem, auxilia a execução da construção do Modelo de Objetos do MAS, pois permite identificar os métodos de cada classe do MAS a partir das operações existentes no sistema atual. Consiste em uma listagem dos procedimentos que são utilizados em cada operação do sistema. Esse relatório com os dados do ASiA está no Anexo A.10.

5.2.5. Estado do protótipo

A necessidade dessa base de dados surgiu para auxiliar o desenvolvimento do projeto, não sendo foco deste. Portanto, o protótipo evoluiu apenas nas etapas iniciais da abordagem, não fornecendo acompanhamento a todo o processo de engenharia reversa de um sistema através da Fusion-RE.

O protótipo permite o acompanhamento completo do Passo 1 da abordagem, que é a Revitalização da arquitetura do sistema com base na documentação existente, e algumas etapas do Passo 2, tais como a obtenção das estruturas de dados, cadastro de temas, relacionamento entre estruturas e procedimentos e classificação desse relacionamento, entre outras.

Os relatórios fornecidos no Passo 2 são de grande utilidade na elaboração dos modelos do MAS, o que constitui o Passo 3. Isso deve-se ao fato de que permitem identificar as anomalias em relação aos procedimentos e estruturas de dados, além de dimensionar o sistema que está sofrendo a engenharia reversa.

Muitas outras funções podem ser adicionadas a esse protótipo, sendo que algumas são sugeridas no Capítulo 7, como trabalhos futuros.

5.3. Aplicação da Fusion-RE ao ASiA

De acordo com as motivações descritas anteriormente, esta seção é destinada à descrição da engenharia reversa realizada no ASiA, que é um sistema implementado em C, utilizando tipos estruturados de dados.

O diagrama da Figura 5.5 mostra como foi realizada a engenharia reversa do ASiA, tendo como base quatro tipos de componentes: informações do sistema, abstrações das características, processos de execução e documentos gerados. Cada componente é identificado por uma letra acompanhada de um número, onde a letra indica o tipo do componente (I = informações; A = abstrações; P = processos; D = documentos) e o número é sequencial em relação ao tipo do componente. A representação utilizada é:

As fronteiras pontilhadas identificam a correspondência com os passos da Fusion-RE, mostrados na Tabela 4.1. Os componentes são descritos no decorrer da seção.

5.3.1. Informações provenientes do sistema

As informações do sistema obtidas tanto a partir do código fonte quanto de sua utilização são distribuídas em componentes distintos, identificados por I no diagrama e descritos a seguir.

- **Módulos e Procedimentos (I1)**

O componente Módulos e Procedimentos é formado por uma relação de todos os programas fonte que compõem o sistema e os procedimentos de cada um. A origem dessas informações são os arquivos com extensão .c e .cpp que fazem parte do projeto Asia.ide. Os módulos que compõem o Asia.ide são mostrados na Figura 5.6.

Figura 5.5: Diagrama da aplicação da Fusion-RE ao ASiA

```

Project : g:\asiasace\bc501\bin\asia.ide
• -[ ] asia [ .exe ]
•   [ ] asia [ .cpp ] code size=46238 lines=5906 data size=14396
•   [ ] gerador [ .cpp ] code size=25197 lines=3365 data size=10475
•   [ ] listarq.cpp [ .cpp ] code size=9991 lines=1534 data size=1176
•   [ ] listdist.cpp [ .cpp ] code size=483 lines=197 data size=652
•   [ ] listeven [ .cpp ] code size=455 lines=168 data size=624
•   [ ] inforec [ .cpp ] code size=205 lines=194 data size=24
•   [ ] listloop [ .cpp ] code size=1940 lines=603 data size=1192
•   [ ] consist [ .cpp ] code size=3252 lines=1212 data size=4228
•   [ ] auxiliar [ .cpp ] code size=1704 lines=611 data size=368
•   [ ] desenha [ .cpp ] code size=1844 lines=201 data size=364
•   [ ] editarq [ .cpp ] code size=7026 lines=1967 data size=2500
•   [ ] lista [ .cpp ] code size=8741 lines=1717 data size=1260
•   [ ] listalig [ .cpp ] code size=12313 lines=2912 data size=2312
•   [ ] listrec [ .cpp ] code size=5907 lines=1475 data size=3152
•   [ ] asia [ .rc ]
•   [ ] asia [ .def ]
  
```

Figura 5.6: Módulos do projeto do ASiA (asia.ide)

- **Estruturas de Dados (I2)**

O componente I2 possui a identificação de todas as estruturas de armazenamento que o sistema utiliza. Essas estruturas são obtidas nos arquivos de cabeçalho (.h) necessários ao sistema. No ASiA, esse arquivo é o tipos.h. Na Figura 5.7, é possível identificar as estruturas auxsimb e lista_ligações. Essas informações também são mostradas no Relatório de estruturas e elementos de dados (Anexo A.4).

```
typedef struct simb{           // Tipo auxiliar para gravar e ler
    int x,                     // a estrutura de símbolos em arquivo
        y;
    char tipo;
    int identrec;
} auxsimb;
typedef struct lista_de_ligacoes{
    int recursol,
        recurso2,
        x1, y1,
        x2, y2;
    char tipo;
    LOCALHANDLE prox;
} lista_ligacoes;
```

Figura 5.7: Trecho de código do arquivo tipos.h

- **Código Fonte (I3)**

O I3 é o componente responsável pelo fornecimento de informações detalhadas da implementação. É constituído pelo conteúdo dos arquivos .c e .cpp que compõem o ASiA, mostrados na Figura 5.6.

- **Opções da Interface (I4)**

Esse componente consiste em uma lista de todas as opções do sistema, disponíveis por meio de menus ou barra de ferramentas a partir da tela principal do ASiA, ilustrada na Figura 3.3, e demais características do sistema, descritas na seção 3.3. As opções são identificadas durante a utilização do sistema.

5.3.2. Abstrações das características do sistema

As abstrações das características do sistema são realizadas com base nas informações disponíveis (componentes especificados no item anterior), sendo focadas nos componentes identificados no diagrama da Figura 5.5 por A e descritos a seguir.

- **Entrada/Processamento/Saída (A1)**

Analisando-se o sistema (por meio do código fonte, interface, utilização da interface e conhecimento prévio), identificam-se os principais assuntos tratados, de acordo com a etapa 2.1

(Definir temas - Tabela 4.1) da abordagem Fusion-RE, descrita na seção 4.5.2. Basicamente, esses assuntos relacionam-se às entradas, processamento e saídas (denominados temas). O tema processamento pode ser dividido para melhor especificação. A cada um desses temas, é associado um nome genérico, facilitando referências posteriores.

Para o ASiA, foram abstraídos quatro temas, pois o tema processamento foi subdividido em "informações temporárias" e "informações permanentes". Os temas definidos são apresentados na Tabela 5.1.

Tabela 5.1: Temas definidos para o ASiA

Tema	Elementos	Nome genérico
Entrada	Modelos em Redes de Filas parametrizados, fornecidos através do Editor Gráfico	Editor Gráfico
Saída	Relatório com os resultados da execução do programa de simulação gerado a partir do modelo fornecido	Resultados da simulação
Informações temporárias	Modelo, parâmetros e programa de simulação gerado	Modelo em construção
Informações permanentes	Modelo e parâmetros em arquivos, programa de simulação (fonte e executável) e relatório de resultados de execução	Modelo arquivado

- **Objetos para o MAS (A2)**

A partir dos Modelos de Objetos do MASA, obtidos em D3, e com base em todo o conhecimento adquirido sobre o sistema (resultante em D1, D2, D4 e D5), identificam-se os objetos essenciais para o MAS.

Os relatórios auxiliares, fornecidos pelo protótipo, também auxiliam na identificação dos objetos que devem fazer parte do modelo de análise para a reengenharia do sistema.

5.3.3. Processos de execução

Os componentes que se referem aos processos de execução constituem, basicamente, as etapas para a obtenção da engenharia reversa propostas pela abordagem Fusion-RE. O modo como essas etapas foram aplicadas ao ASiA é descrito nos componentes abaixo, indicados por P no diagrama (Figura 5.5).

- **Cadastro de Módulos (P1)**

O Cadastro de módulos refere-se à primeira etapa da abordagem, que é a Revitalização da arquitetura do sistema com base na documentação existente, descrita na seção 4.5.1. As informações provêm do componente I1 (Módulos e procedimentos).

A interface do protótipo utilizada para o cadastro dessas informações é ilustrada na Figura 5.2. A partir dos módulos, foram cadastradas as funções (ou procedimentos) a eles associadas (na tabela Procedimento), incluindo uma breve descrição de sua funcionalidade.

• Cadastro de Temas (P2)

Os temas, identificados nas abstrações para Entrada/Processamento/Saída (A1), são cadastrados através do componente P2. O formulário do protótipo utilizado é mostrado na Figura 5.8. Deve ser incluída uma descrição, identificando o(s) elemento(s) de cada tema, e o nome genérico para referências posteriores. A partir dessas informações será produzido o D2.

Temas : Formulário	
Tema: Entrada	Nome: Editor Gráfico
Descrição: Modelo em Rede de Filas	
Tema: Saída	Nome: Resultados da simulação
Descrição: Relatório dos resultados da simulação	
Tema: Informações temporárias	Nome: Modelo em construção
Descrição: Modelo, parâmetros e programa de simulação	
Tema: Informações permanentes	Nome: Modelo arquivado
Descrição: Modelo, parâmetros, programa de simulação e relatório de resultados	

Figura 5.8: Tela para cadastro dos temas (com os dados do ASiA)

• Cadastro de Estruturas de Dados (P3)

Para executar o Cadastro de estruturas de dados, são utilizadas as informações do componente I2. As estruturas de dados servirão de base para que o Modelo de Objetos do MASA seja desenhado. As estruturas e os elementos de dados que as compõem são cadastradas com o auxílio do protótipo, através do formulário mostrado na Figura 5.9.

• Relacionar Estruturas aos Temas (P4)

As estruturas cadastradas anteriormente (P3) são agora associadas aos temas a que pertencem, através do formulário ilustrado na Figura 5.10. Esse componente permite a identificação das anomalias quanto às estruturas que se relacionam a mais de um tema. Essas anomalias devem ser tratadas quando da construção do MAS.

Os temas da Tabela 5.1 valem para o MASA. Analisando-se as estruturas de dados implementadas, quando da construção do Modelo de Objetos do MAS, verifica-se que é desnecessária a existência de temas distintos para as informações temporárias e as permanentes,

pois isso é característica da implementação atual (possui estruturas de dados diferentes para armazenamento temporário e permanente, mas com os mesmos elementos de dados). Dessa forma, o MAS contará apenas com os temas Entrada, Saída e Processamento.

Figura 5.9: Cadastro de estruturas de dados

Figura 5.10: Tela de associação entre estruturas e temas

- **Relacionar Procedimentos e Estruturas (P5)**

Relacionar quais estruturas de dados cada procedimento manipula não é uma tarefa trivial. O componente P5 é responsável para que uma classificação detalhada da relação entre estruturas e procedimentos seja obtida, a partir da qual são identificadas as anomalias do sistema em relação aos procedimentos que manipulam mais de uma estrutura.

A Figura 5.3 ilustra o formulário do protótipo que permite que seja feita a associação entre os procedimentos e as estruturas por eles manipuladas, além de classificar essa manipulação (o procedimento pode ser observador ou construtor da estrutura).

A classificação do procedimento na estrutura também é uma etapa complexa. A forma como o ASiA foi implementado dificulta a identificação da funcionalidade dos procedimentos. Há variáveis declaradas, comuns para os procedimentos pertinentes, que são utilizadas temporariamente, para depois terem seus conteúdos gravados nas estruturas. Uma solução é que cada variável declarada fosse mapeada para a respectiva estrutura, para então poder classificar esses procedimentos adequadamente. No entanto, para minimizar a complexidade dessa etapa, foi adotado um nível de abstração mais alto, classificando apenas os procedimentos que manipulam realmente as estruturas como observadores e/ou construtores e os demais como procedimentos responsáveis pela implementação (ou interface - tipo i).

- **Cadastro de Opções (P6)**

A interface do ASiA é constituída pela barra de menus, pela barra de acesso rápido (com botões de atalho para as funções do menu) e pela barra de ferramentas (com as opções fornecidas para desenhar o modelo desejado), além da barra de ferramentas específica do módulo Arquitetura (com recursos para desenhar a arquitetura que se quer modelar). Todos esses elementos constituem opções do sistema, que devem ser consideradas para a construção do Modelo de Ciclo de Vida do MASA. Para o ASiA, foi elaborada uma lista com essas opções, a partir da qual foi construído o Modelo de Ciclo de Vida.

- **Identificar Operações (P7)**

No componente Identificar Operações, tendo como base as opções da interface e o Modelo de Ciclo de Vida, são listadas as operações, que são opções que executam funcionalidades do sistema. É uma etapa complexa, pois o ASiA não dispõem de manuais atualizados (os manuais do usuário existentes se referem à primeira versão operacional do ambiente). Assim, é necessário testar cada opção e, com base em conhecimentos adquiridos ao longo das etapas anteriores, construir o Modelo de Operações de todas as funções do sistema em questão. O resultado desse componente é obtido no componente D5.

- **Cadastro de Objetos do MAS (P8)**

A supressão de um dos temas de processamento acarreta na redução do número de objetos, facilitando a compreensão e manutenção do sistema, características desejáveis para o

ASiA. Essa redução implica em direcionar o MAS para os objetos essenciais ao funcionamento do ambiente, desprezando características de implementação e interface gráfica.

Os objetos do MAS foram identificados a partir dos objetos do MASA. As diferenças se referem a anomalias encontradas na modelagem do sistema atual. A equivalência entre os objetos do MASA e do MAS é mostrada na Tabela 5.2.

Tabela 5.2: Classes do ASiA

Objetos do MASA (tema temporário)	Objetos do MASA (tema permanente)	Objetos do MAS
	auxspecifi	Modelo
lista_recursos	auxrec	Recurso
lista_ligacoes	auxlig	Ligação
lista_simbolos	auxsimb	
lista_temp_eventos	auxtempeventos	Evento
lista_tot_loop	auxtotloop	Loop
lista_dist_tab	auxdisttab	Distribuição
lista_arquitet	auxarquitet	Arquitetura
		Probabilidade

• Identificar Classes e Relacionamentos (P9)

Os objetos identificados anteriormente originam as classes a serem implementadas na versão produzida pela reengenharia do ASiA. As estruturas auxsimb e lista_simbolos, que armazenam informações referentes ao recurso com uma associação um para um, foram eliminadas do MAS, sendo que seus elementos de dados foram adicionados como atributos à classe Recurso. As informações referentes a probabilidade de uma entidade seguir determinada ramificação (quando uma ligação possuir ramificações) são armazenadas no elemento de dados entradas_saida da estrutura auxrec, de forma bastante complexa (é construída uma *string* com essas informações). Para simplificar essa estrutura, foi criada a classe Probabilidade.

A redução do número de objetos em relação ao número de estruturas de dados do MASA contribui para a solução de algumas anomalias em relação aos procedimentos. Existem procedimentos dedicados à manipulação de dados entre estruturas de dados semelhantes em temas distintos, que se tornam obsoletos quando essas estruturas resultam em apenas uma classe (a funcionalidade desses procedimentos torna-se desnecessária).

Para identificar os futuros métodos do ASiA, foi estabelecida uma associação entre procedimentos e operações, da forma mostrada no relatório Procedimentos utilizados nas operações (Anexo A.10). Através desse relatório, foram identificadas anomalias na implementação do ASiA. É muito difícil estabelecer o relacionamento entre os procedimentos e

os métodos que devem ser gerados. Isso não acontece se um nível de abstração mais alto for utilizado, associando apenas as operações, apesar dessas serem constituídas por procedimentos.

Os Anexos B.2 e B.3 contém as especificações dos atributos e métodos, respectivamente, para as classes do MAS.

Algumas operações do MASA, tais como Ok, Cancel, Personalizar, Sair e Sobre o ASiA, não constituem métodos das classes do MAS. Posteriormente, farão parte das classes utilizadas para implementar a interface, que irão compor o tema de entrada (as classes apresentadas até o momento constituem o tema de processamento do MAS, que substitui os temas temporário e permanente do MASA).

- **Atualizar Ciclo de Vida (P10)**

Para efetuar o componente P10, utiliza-se como base o Modelo de Ciclo de Vida do MASA, tendo como objetivo a atualização dos nomes (utilizando nomes mais significativos).

No ASiA, a diferença entre o Modelo de Ciclo de Vida do MAS e o do MASA é resultante das alterações realizadas nos nomes das operações.

- **Reespecificar Operações (P11)**

Esse componente trata de alterar os Modelos de Operações do MASA, incluindo os nomes dos relacionamentos, atributos e classes identificados para o MAS.

Nos modelos propostos para a reengenharia do ASiA, a funcionalidade das operações é mantida. A maior parte das informações que constituem o Modelo de Operações diz respeito a características de implementação e interface, o que não é adequado ao nível adotado para a elaboração dos modelos do MAS.

- **Mapear Objetos do MAS e do MASA (P12)**

O mapeamento entre os objetos do MAS e do MASA trata de construir uma correspondência entre as classes do MAS e as pseudo-classes do MASA, incluindo atributos (estruturas de dados do MASA) e métodos (procedimentos do MASA).

A execução desse componente resulta no documento de Mapeamento entre o MAS e o MASA (D9).

5.3.4. Documentos gerados

O conteúdo dos documentos produzidos na engenharia reversa do ASiA e a forma como eles são elaborados são descritos a seguir. Os documentos gerados para o ASiA fazem parte dos Anexos desta dissertação.

- **Documento de Revitalização do Sistema (D1)**

O Documento de Revitalização do Sistema é produzido a partir do componente P1 e segue o padrão proposto pela Fusion-RE (Figura 4.2).

O D1, que constitui o Anexo A.1, fornece todos os procedimentos que compõem o ASiA, separados nos módulos a que pertencem, incluindo uma descrição de sua funcionalidade e uma listagem dos procedimentos que chama e dos procedimentos pelos quais é chamado.

- **Diagrama de Temas (D2)**

O D2 representa graficamente as informações obtidas no componente P2, contidas na Tabela 5.1 (temas abstraídos com seus nomes genéricos). O Diagrama de Temas do ASiA é apresentado no Anexo A.3.

- **Modelo de Objetos do MASA (D3)**

O Modelo de Objetos segue a representação do método Fusion e inclui as características de cardinalidade, papéis, agregação, generalização e especialização às estruturas previamente identificadas.

A construção do Modelo de Objetos a partir das estruturas exige conhecimentos mais aprofundados a respeito de como o sistema foi construído e sua funcionalidade.

Para auxiliar na execução do componente D3, optou-se pela utilização da Fusion CASE [Fus94], que é uma ferramenta para acompanhamento do desenvolvimento de um projeto seguindo o método Fusion, fornecendo os recursos gráficos necessários à construção dos modelos envolvidos no método e, por consequência, do modelo em questão.

Para o MASA do ASiA, foi necessário desenvolver dois Modelos de Objetos distintos, um para o tema temporário e outro para o tema permanente, de acordo com as estruturas de dados manipuladas pelo sistema. O Anexo A.6 mostra o Modelo de Objetos do MASA para o tema temporário e o A.7 para o tema permanente.

- **Modelo de Ciclo de Vida do MASA (D4)**

A partir das opções incluídas no componente Cadastro de opções, é construído o Modelo de Ciclo de Vida, seguindo a notação do Fusion proposta pela abordagem utilizada, como descrito na seção 4.5.2.

A notação utilizada é composta pelos operadores:

- () precedência
- [] opcionalidade
- | ou
- . e
- * zero ou mais vezes
- + uma ou mais vezes
- # evento de saída

O Anexo A.8 contém o Modelo de Ciclo de Vida do ASiA. As informações para a construção desse modelo são provenientes da interface do sistema, sendo a tela principal do ASiA mostrada na Figura 3.3.

- **Modelo de Operações do MASA (D5)**

As operações identificadas anteriormente são, nesse componente, descritas de acordo com o modelo definido pelo Fusion, ilustrado na Figura 4.4.

Para se obter o D5, é necessário conhecer, além da interface, o modo como o sistema é implementado, para poder identificar os dados que são manipulados. Os Modelos de Operações do ASiA são incluídos no Anexo A.9.

- **Modelo de Objetos do MAS (D6)**

De acordo com os objetos cadastrados em P8 e as classes e relacionamentos identificados em P9, é construído o Modelo de Objetos, muito semelhante aos do MASA, mas com nomes mais significativos (de acordo com os nomes mostrados na Tabela 5.2).

O Modelo de Objetos do MAS do ASiA está no Anexo B.1.

- **Modelo de Ciclo de Vida do MAS (D7)**

Mostrado no Anexo B.4, o Modelo de Ciclo de Vida do MAS não apresenta alterações na interface atual do ASiA, com exceção das funções sem funcionalidade que foram retiradas (tais como Help e Animação).

- **Modelo de Operações do MAS (D8)**

Em consequência do componente P11 não ter sido executado, o Modelo de Operações para o MAS não foi produzido.

- **Mapeamento entre o MAS e o MASA (D9)**

O componente D9 produz um modelo que facilita a identificação dos elementos no momento de efetuar a reengenharia do sistema, bem como facilitar sua compreensão.

O Anexo B.5 ilustra o relacionamento entre as estruturas de dados e seus elementos da versão atual do ASiA e as classes e seus respectivos atributos do modelo proposto para a reengenharia do ambiente.

5.4. Considerações finais

A complexidade envolvida no estudo de um sistema pode ser minimizada com o auxílio de etapas bem definidas. A abordagem Fusion-RE tem essa função, auxiliando na construção dos modelos do sistema em estudo (engenharia reversa) e produção dos modelos para a reconstrução (reengenharia) desse sistema.

Há ainda muitas evoluções que a Fusion-RE pode implementar para tornar-se mais funcional. Uma ferramenta, como o protótipo aqui apresentado, é uma contribuição significativa à aplicabilidade da abordagem.

Os modelos produzidos pela aplicação da Fusion-RE ao ASiA, tanto da versão atual (MASA) quanto para os modelos para sua reconstrução estão contidos, em sua íntegra, como Anexos.

Proposta para a Reengenharia do ASiA

6.1. Introdução

A reengenharia de sistemas trata, basicamente, de reescrever o código, sem alterar a funcionalidade, com os seguintes objetivos (como descrito na seção 4.4.3):

- ⇒ facilitar a manutenção;
- ⇒ migrar o sistema;
- ⇒ reduzir erros;
- ⇒ preparar para aperfeiçoamento funcional.

Além disso, os modelos obtidos pela engenharia reversa podem ser acrescidos de novas funcionalidades. Uma análise crítica desses modelos é fortemente recomendada para que características indesejáveis do sistema atual não sejam transcritas para a nova versão.

A reengenharia do ASiA terá como ponto de partida os modelos produzidos pela engenharia reversa (inseridos em anexo), utilizando a linguagem C++ e seguindo o paradigma de orientação a objetos. A aplicação da orientação a objetos suprirá as características desejáveis para a nova versão do ASiA: um sistema simples, conciso, com a compreensão e manutenção facilitadas.

Segundo Sneed [Sne95], a reengenharia técnica não deve ser executada concomitantemente à estrutural. Portanto, o objetivo de se ter uma nova versão do ASiA é facilitar o crescimento do sistema, para que posteriormente possam ser adicionados novos módulos, funcionalidades e características propostas no projeto inicial que não fazem parte da versão operacional.

6.2. Processo para a reengenharia do ASiA

Tendo-se utilizado a abordagem Fusion-RE para produzir os modelos de análise do ASiA, é indicado que o método de desenvolvimento Fusion seja utilizado para a reengenharia do ambiente.

Com certeza, a viabilidade da engenharia progressiva do ASiA com base nos modelos produzidos pela engenharia reversa é efetiva. Todas as inconsistências verificadas na versão atual e suas anteriores podem ser tratadas (como algumas já o foram), resultando em um sistema muito mais estável, integrado, conciso e claro, facilitando a adição de novos módulos.

Para não comprometer o trabalho apresentado nesta dissertação, os módulos posteriores deverão seguir a forma de documentação aqui adotada, estando prontos para a integração quando a nova versão estiver disponível.

Outro fator a ser analisado é a possibilidade de executar a reengenharia do sistema em etapas, permitindo que o sistema funcione com uma parte da codificação na linguagem atual e outra na linguagem proposta (no ASiA, C e C++ respectivamente). A característica essencial para isso é a compatibilidade entre as linguagens envolvidas [Pen96].

6.3. Alterações a serem implementadas

Para o aperfeiçoamento funcional do ASiA, são sugeridas a seguir alterações que se vêm necessárias em vista do estudo realizado durante a engenharia reversa e das características desejáveis ao ambiente.

6.3.1. Sistema de simulação

A versão atual do ASiA utiliza o SMPL [Mac87] como sistema de simulação (fornece as funções específicas de simulação em C para que o gerador de aplicações produza os programas).

Em substituição ao SMPL, poderá ser utilizado o SimPack [Fis92], que embora tenha sido desenvolvido com base no SMPL, possui um grande número de rotinas estendidas e funções adicionais. O SimPack é uma biblioteca com as funções de simulação em C++, compatível com a versão produzida pela reengenharia do ASiA, sendo que as características que favorecem a utilização do SMPL permanecem, a exemplo da programação em C (agora C++), que elimina a necessidade de se aprender uma nova linguagem de programação.

Um dos objetivos pretendidos para o ASiA é a disponibilidade de diversas técnicas de modelagem. O SimPack oferece características adequadas para que isso seja possível, pois

fornece suporte a diversos tipos de modelagem, possibilitando que o modelo seja criado utilizando a técnica mais adequada, incluindo Autômatos Finitos, Cadeias de Markov, Redes de Filas, Redes de Petri e equações diferenciais. Além disso, o SimPack permite a implementação de diferentes algoritmos de simulação, incluindo simulação de eventos discretos, simulação contínua e simulação combinada (multimodelos¹³). Os multimodelos admitidos são uniões de diversos modelos conectados por um grafo ou rede, utilizados em problemas de simulação combinados em múltiplos níveis de abstração.

Além disso, é possível vislumbrar a utilização de uma nova versão do SimPack, o OOPM/MOOSE (Multimodeling Object-Oriented Simulation Environment) [Cub97], que consiste em uma implementação para um sistema de simulação baseado em uma extensão para projeto orientado a objetos.

O propósito do MOOSE é permitir a simulação de processos físicos através da construção de multimodelos. A linguagem para modelos de blocos fornece uma linguagem montadora para os diferentes tipos de modelos disponíveis no MOOSE, que são traduzidos em modelos de blocos e então simulados usando o SimPack.

6.3.2. Editor Gráfico

A interface do Editor Gráfico deve mudar, com o objetivo de facilitar a parametrização dos modelos. Janelas de parametrização suspensas, sensíveis ao contexto, possibilitarão que os parâmetros de determinado recurso sejam visualizados durante toda a edição do modelo (basta selecionar um recurso para visualizar seus parâmetros, ou clicar em um ponto vazio da janela de edição gráfica para que os parâmetros do modelo sejam mostrados).

Quanto às funções mover e excluir, o ideal é que sejam padronizadas, a exemplo dos softwares gráficos. Uma boa opção é ativá-las pelo click do botão direito do mouse sobre o recurso ou ligação. A seleção de diversos recursos (através de funções do tipo clicar e arrastar) também é desejável para a função mover. Essas funções podem ser encontradas prontas em APIs.

Outro fator é a interface para as distribuições de probabilidade geradas pelo usuário, que na versão atual é confusa. A forma mais adequada seria uma janela que permitisse a visualização das distribuições geradas, bem como os valores cadastrados para cada distribuição.

¹³ Um multimodelo é uma hierarquia heterogênea de modelos, onde um componente em um nível de abstração é refinado em um modelo, possivelmente de um tipo diferente, para o próximo nível mais baixo.

O desenvolvimento dos *helps*, que tem por objetivo auxiliar a utilização do sistema, irá permitir que a interface seja melhor utilizada.

6.3.3. Módulo Visual

Para integrar o Módulo Visual ao projeto de reengenharia do ASiA, deve-se realizar a construção dos seus modelos de análise, integrando-os com os modelos apresentados para a nova versão do ambiente. Como esse módulo foi desenvolvido seguindo o paradigma de orientação a objetos, essa etapa não deve envolver muita complexidade.

6.3.4. Demais módulos

Tendo em vista o desenvolvimento da extensão do ASiA para modelagem em Redes de Petri, um passo posterior é a implementação de modelos híbridos, para que partes mais complexas possam utilizar técnicas de modelagem mais adequadas, aplicando os recursos fornecidos pelo SimPack.

Para os módulos que estão em desenvolvimento, é aconselhável que sejam produzidos os modelos, para que também possam ser integrados à nova versão do ASiA.

Enquanto a reengenharia do ASiA não acontece, os módulos que possam vir a ser desenvolvidos devem seguir as especificações da nova versão, ou seja, ser construídos em C++, aplicando o paradigma de orientação a objetos. Poderão ser desenvolvidos *gateways*, que são interfaces para o mapeamento de funções entre módulos distintos, para integrá-los, temporariamente, à versão atual. Segundo Olsem [Ols97], essa tecnologia é denominada reengenharia incremental não-reintegrada.

6.4. Considerações finais

O Fusion está descrito na seção 4.3. A Fase de Análise para a reengenharia do ASiA está concluída; futuramente, deverá ser acrescida com as novas funcionalidades.

A Fase de Projeto, em se tratando de reescrever o sistema sem alterar suas funcionalidades, também está pronta. Com isso, o tempo de reconstrução do ASiA será significativamente reduzido, assim como sua complexidade. A utilização da Fusion CASE poderá otimizar ainda mais esse processo, sendo necessária a construção de alguns modelos para que as classes em C++ sejam geradas.

Conclusões

7.1. Contribuições

Este projeto, além de produzir os modelos de análise para o ASiA, fornece também especificações mais precisas em algumas das etapas da abordagem Fusion-RE. Nesta seção, são apresentadas as principais contribuições deste trabalho em relação ao ASiA e à Fusion-RE.

7.1.1. Contribuições para o ASiA

A realização da engenharia reversa do ASiA constitui a principal contribuição desta dissertação. Através dela foram produzidos os modelos de análise que retratam sua implementação atual, bem como os modelos para reengenharia do ambiente seguindo o paradigma de orientação a objetos.

Os modelos do sistema atual, que constituem o MASA, permitem que o sistema seja compreendido mais facilmente, pois a documentação existente estava desatualizada e se dispunha apenas do código fonte para seu estudo.

Apesar da reengenharia do ASiA não ter sido realizada, são fornecidos todos os modelos de análise e o processo de desenvolvimento que deve ser seguido para que uma nova versão seja implementada.

Com o estudo e utilização do ASiA durante a engenharia reversa, diversas melhorias foram identificadas, auxiliando a integração de futuros módulos. Exemplo dessas melhorias são as mudanças entre estruturas de dados na versão atual e as classes para a reengenharia do ASiA e a troca do sistema de simulação (de SMPL para uma linguagem de simulação orientada a objetos).

Os modelos de análise orientados a objetos para a reengenharia do ASiA constituem a especificação completa de um sistema de simulação de Redes de Filas, sendo que a literatura dessa área não apresenta especificações desse tipo. Essa documentação, portanto, pode ser vista como uma contribuição para que outros sistemas de simulação sejam desenvolvidos.

O ASiA foi desenvolvido originalmente em ambiente Windows, mas desde o início do projeto é desejada a implementação de uma versão em UNIX (Linux). Os modelos de análise produzidos poderão ser utilizados para a implementação desse sistema em qualquer plataforma, desde que siga o paradigma orientado a objetos.

7.1.2. Contribuições para a Fusion-RE

A adoção da abordagem Fusion-RE como ferramenta para a engenharia reversa do ASiA possibilitou a validação da abordagem em relação a sistemas com as mesmas características de implementação do ASiA.

A utilização da base de dados de apoio à execução da engenharia reversa de acordo com a Fusion-RE (na forma de um protótipo) contribuiu para simplificar a aplicação da abordagem, encorajando o desenvolvimento de uma ferramenta que forneça suporte detalhado a todas as etapas da Fusion-RE.

Especificações mais precisas em relação a algumas etapas da Fusion-RE foram desenvolvidas. O processo para aplicação da abordagem adotado, representado no diagrama da Figura 5.5 e descrito na seção 5.3, torna a utilização da abordagem em questão mais simples, pois consiste em passos bem definidos e explicados.

Em geral, os documentos que descrevem métodos ou abordagens de desenvolvimento de projetos incluem exemplos dos documentos gerados, o que não é suficiente quando se deseja aplicar o método ou abordagem. Nesse aspecto, este trabalho é significativo por apresentar, além do processo de execução, todos os modelos gerados em cada etapa da engenharia reversa (em anexo).

Outra contribuição à aplicação da Fusion-RE são os relatórios auxiliares fornecidos pelo protótipo. Embora não façam parte dos requisitos para a obtenção dos modelos da abordagem, facilitam significativamente a execução de algumas etapas e contribuem para a elaboração dos relatórios necessários.

Este projeto constitui um estudo de caso importante para a abordagem Fusion-RE que, embora seja ainda pouco disseminada, é uma abordagem com potencial a ser explorado, sendo que as contribuições aqui apresentadas colaboram para sua melhor especificação e difusão.

7.2. Trabalhos futuros

Nesta seção, serão propostos alguns trabalhos como continuidade deste projeto. Dentre eles, estão a reengenharia do ASiA e o desenvolvimento da ferramenta de apoio à Fusion-RE a partir da base de dados utilizada como apoio neste projeto.

7.2.1. Reengenharia do ASiA

A reengenharia do ASiA deve seguir o processo descrito no Capítulo 6, produzindo a nova versão. Após isso, os novos módulos e funcionalidades poderão ser adicionados muito mais facilmente.

7.2.2. Ferramenta de apoio à Fusion-RE

Em se tratando do acompanhamento da execução da engenharia reversa, a ferramenta pode monitorar a conclusão dos passos (ou sub-passos) da abordagem. Isso pode ser feito através de uma lista de verificação, que dependerá do engenheiro de software para informar cada etapa concluída. Com o decorrer do tempo, através da utilização da ferramenta, pode-se obter uma base de dados sobre a relação entre a quantidade de informações manipuladas em cada passo, permitindo o cálculo de estatísticas a respeito do percentual de execução a que cada etapa corresponde.

A abordagem Fusion-RE fornece métricas [Pen96] para calcular os esforços necessários para a conversão do sistema. A ferramenta deverá, tão logo quanto possível, fornecer os valores dessas métricas (basta que os dados necessários para o cálculo estejam disponíveis). Das quatro métricas fornecidas pela Fusion-RE, três (IEC, IITIU e CMC) são calculadas a partir das informações necessárias à documentação do sistema. Para a métrica ICPI, é necessário cadastrar o número de linhas de código em cada procedimento. Nesse caso, vale analisar a relação custo/benefício, considerando-se a dificuldade de coletar as informações adicionais que devem ser cadastradas e os resultados por ela fornecidos.

Como descrito anteriormente, espera-se, com a utilização do sistema, formar uma base de dados com informações a respeito do tempo de execução de cada passo. Com base nisso, poderão ser produzidas novas métricas, mais avançadas e precisas, tanto em relação à execução da abordagem quanto da reengenharia do sistema enfocado. Em geral, essas métricas são aplicadas para a melhoria do processo de execução da abordagem e, por consequência, da ferramenta [Pre95].

Outra característica a ser acrescida é a leitura do código fonte. Como boa parte das informações para o MASA tem origem direta no código fonte do sistema atual, a captura automática dessas informações reduziria consideravelmente o tempo de execução da engenharia reversa e interação com o engenheiro de software, aumentando a corretitude das informações devido à redução do fornecimento de dados manual na ferramenta.

A construção dos modelos e diagramas, seguindo a notação gráfica do método Fusion, também deverá ser fornecida pela ferramenta. A construção desse tipo de recurso não é tarefa trivial, mas é significativamente auxiliada pela disponibilidade de ferramentas gráficas.

Outro fator a ser analisado é a viabilidade de gerar arquivos para alimentar a Fusion CASE [Fus94], para que ela forneça os modelos utilizando informações já existentes. Isso implica em identificar o formato dos arquivos que a Fusion CASE manipula, para fornecer as informações necessárias a cada modelo desejado. Além disso, deverão ser obtidos os modelos gerados e armazenados em formulários ou relatórios na base de dados utilizada. Isso também auxilia na segurança e coerência das informações provenientes dos modelos, que poderão ser armazenadas juntamente às demais informações.

7.2.3. Aprimoramento da documentação

Os métodos de desenvolvimento são constituídos, em geral, por uma linguagem para o projeto dos modelos de análise e de um processo para o desenvolvimento (construção) do sistema. Via de regra, apenas a linguagem é utilizada, pois o processo não se adapta às necessidades do desenvolvedor ou já existem processos estabelecidos. Dessa forma, a UML (Unified Model Language) é uma linguagem para especificação, construção, visualização e documentação dos artefatos de sistemas de software, que não engloba um processo de desenvolvimento [Sco98].

A forma como um sistema computacional é desenvolvido não é de importância relevante quando se pretende entendê-lo, ao contrário dos modelos que o representam. Os modelos devem permitir que qualquer pessoa que necessite obter informações sobre o sistema o compreenda.

Sendo proposta a reengenharia do ASiA com a utilização do paradigma de orientação a objetos, sugere-se, portanto, que os modelos apresentados neste projeto sejam traduzidos para UML, proporcionando maior facilidade para o crescimento do sistema.

É importante que, depois de realizar a reengenharia de um sistema (realizada a partir do modelo de análise fornecido pela engenharia reversa), continue-se a utilizar o mesmo método adotado para o desenvolvimento de novos módulos, mantendo a documentação do sistema com o

objetivo de facilitar sua manutenção e extensibilidade. Como o ASiA é um sistema dinâmico (está em constante desenvolvimento), essa documentação também deverá ser adotada no desenvolvimento dos futuros módulos, tendo a situação ideal de existir a documentação do sistema na sua íntegra sempre disponível.

A UML está sendo validada no que concerne ao fornecimento de bases para a construção de ferramentas, tais como modelagem visual, simulação e ambientes de desenvolvimento [Uml98]. Uma proposta para a Fusion-RE é que sejam estabelecidas diretrizes para a conversão dos modelos produzidos em seus moldes para UML, possibilitando que a abordagem continue a ser utilizada, resultando em modelos padrão em UML, sem mudar o objetivo a que se destina.

Outra proposta, que foge ao escopo deste trabalho, mas visa auxiliar o Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC, é o estudo de UML para a modelagem de sistemas distribuídos e concorrentes, pois esses recursos já existem. Exemplo disso é o *Deployment Diagram*, que pode mostrar como os componentes e objetos são roteados e se movem em um sistema distribuído [Fow97]. O estabelecimento de critérios para que o grupo venha a utilizar a UML para a especificação de seus sistemas poderia ser originado pela conversão da modelagem do ASiA.

7.2.4. Linguagem adotada

Uma solução para a obtenção de dois objetivos simultaneamente – reengenharia do ASiA e construção de uma nova versão para Linux –seria a adoção de Java, devido à sua portabilidade, em substituição a atual linguagem C ou C++ proposta para a sua reengenharia.

Entretanto, a linguagem Java ainda não é recomendada para aplicações complexas, que envolvem um grande número de cálculos (o que engloba o ASiA), pois seu desempenho deixa a desejar mesmo quando tecnologias de compilação ou ferramentas para incrementar desempenho são utilizadas [Gon98].

A questão do desempenho poderia ser solucionada com a utilização de Java Nativo, mas a portabilidade, que é a característica chave desejada, seria perdida [Shi98].

7.2.5. Outras mudanças

Para a obtenção de contribuições para a melhoria do ASiA, é interessante disponibilizar o ambiente para que seja utilizado fora do domínio onde é desenvolvido. A forma mais simples de fazer isso é disponibilizar o ASiA em um site na Internet; para que isso seja possível, são necessárias algumas alterações (durante ou após a reengenharia do ASiA), tais como:

- ⇒ especificação em UML: para possibilitar o intercâmbio com sistemas similares que possam estar sendo desenvolvidos;
- ⇒ construção de helps: para facilitar a utilização do ambiente, incluindo auxílio à modelagem e parametrização dos modelos, que não são tarefas simples;
- ⇒ confecção de manuais: para auxiliar a instalação e utilização do ambiente, além de descrever os objetivos e domínio do ASiA;
- ⇒ adição de bibliotecas da linguagem C: para possibilitar a geração dos programas de simulação, caso contrário não se consegue utilizar o sistema efetivamente. Outra opção é especificar essas bibliotecas como requisitos;
- ⇒ construção de um tutorial: para ensinar as funções do sistema a quem não o conhece, de uma forma mais simples e rápida do que a leitura dos manuais. Um modelo bem construído, parametrizado e simulado é interessante para que possam ser analisados os valores a serem fornecidos e os resultados produzidos pelo programa de simulação;
- ⇒ relação de links para sites sobre modelagem, Redes de Filas e simulação: para auxiliar o processo de aprendizagem do domínio do ASiA;
- ⇒ tradução para a língua inglesa: para que seja possível disponibilizar essa página e o ambiente, interfaces e helps devem estar em inglês, para que o ASiA seja efetivamente utilizado e receba contribuições significativas.

7.3. Dificuldades encontradas

Todo o conhecimento envolvido neste projeto foi adquirido durante sua realização. As áreas envolvidas (modelagem, simulação e manutenção de software) não são simples, mas a literatura disponível e as discussões com os pesquisadores da área, em particular os envolvidos no projeto do ASiA e com a Fusion-RE. A multidisciplinariedade envolvida na execução desta dissertação permitiu a compreensão mais abrangente de algumas das áreas que a computação engloba.

7.4. Considerações finais

A manutenção de software é uma área de destaque, pois envolve maior parte dos recursos gastos em um sistema. Das metodologias de manutenção existentes, destacam-se a engenharia reversa e a reengenharia. A ausência de métodos para a execução da engenharia reversa motivou a elaboração da abordagem Fusion-RE, que utiliza como embasamento o conhecido método de desenvolvimento orientado a objetos Fusion. Outro fator que torna a Fusion-RE uma abordagem

atraente é o fato de produzir o modelo de análise orientado a objetos a partir de um sistema implementado sem a utilização desse paradigma.

Com relação ao ASiA, a obtenção da documentação produzida através da aplicação da abordagem auxiliará a implementação e integração dos módulos adicionais, até que seja executada a reengenharia do ambiente. A proposta inicial deste projeto era um estudo do ASiA, realizando uma comparação da implementação atual com a especificação inicial do ambiente, visando a produção de um software mais simples e melhor estruturado, com todos os seus módulos seguindo o mesmo paradigma de implementação.

Embora a nova versão não tenha sido produzida, este projeto fornece todos os recursos para tal, através do processo de reengenharia descrito no Capítulo 7 e dos modelos em anexo. Muitas características desejadas inicialmente foram limitadas na primeira versão pelo fato do SMPL não fornecer recursos para implementá-las. Dessa forma, se o sistema de simulação for substituído na reengenharia do ASiA, será possível resgatar tais características.

Referências Bibliográficas

- [Big89] BIGGERSTAFF, T.J. *Design Recovery for Maintenance and Reuse*. IEEE Computer, v. 22, n. 7, pp. 36-49, Julho, 1989.
- [Bha84] BHAT, U.N. *Element of Applied Stochastic Processes*. John Wiley & Sons, 1984.
- [Bru97] BRUSCHI, S.M. *Extensão do ASiA para Simulação de Arquiteturas de Computadores*. Dissertação de Mestrado, ICMSC-USP, Setembro, 1997.
- [Chi90] CHIKOFSKY, J.E.; CROSS, J.H. *Reverse Engineering and Design Recovery: A Taxonomy*. IEEE Software, v. 7, n. 1, pp. 13-17, Janeiro, 1990.
- [Cho90] CHOY, S.C.; SCACHI, W. *Extracting and Restructuring the Design of Large Systems*. IEEE Software, v. 7, n. 1, pp. 66-71, Janeiro, 1990.
- [Col96] COLEMAN, D.; ARNOLD, P.; BODOFF, S.; DOLLIN, C.; GILCHRIST, H.; HAYES, F.; JEREMAES, P. *Desenvolvimento Orientado a Objetos - O método Fusion*. Campus, 1996.
- [Cub97] CUBERT, R. M.; Goktekin, T.; Fishwick, P. A. *MOOSE: architecture of an object-oriented multimodeling simulation system*. Proceedings of Enabling Technology for Simulation Science, SPIE AeroSense '97 Conference, Orlando, FL, Abril, 1997.
- [Die89] DIETRICH, W. C.; NACKMAN, L. R.; GRACER, F. *Saving a Legacy with Objects*. OOPSLA, 89, New Orleans, Proceedings ACM, pp. 77-83, 1989.
- [Edw92] EDWARDS, G.; SANKAR, R. *Modeling and Simulation of Networks Using CSIM*. Simulation, v. 58, n. 2, pp. 131-136, Fevereiro, 1992.
- [Fis92] FISHWICK, P.A. *SimPack: getting started with simulation programming in C and C++*. Winter Simulation Conference, Arlington, VA, Dezembro, 1992.
- [Fol87] FOLEY, J.D.; DAM, A. van; FEINER, S.K.; HUGHES, J.F. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, 1987.
- [Fow97] FOWLER, M.; SCOTT, K. *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley Longman Inc, 1997.
- [Fus94] Fusion CASE – User Guide. Software Consulting Ltd, 1994.

- [Gon98] GONSALVES, A. *Java performance still disappointing*. PC Week Online. Available from World Wide Web: <http://www.zdnet.com/pcweek/news/0525/25java>, Maio, 1998.
- [Har87] HAREL, D. *Statecharts: a Visual Formalism for Complex Systems*. Science of Computer Programming, pp. 231-234, 1987.
- [Heu90] HEUSER, C.A. *Modelagem Conceitual de Sistemas - Rede de Petri*. V EBAI, 1990.
- [Mac87] MACDOUGALL, M.H. *Simulating Computer Systems: Techniques and Tools*. The MIT Press, 1987.
- [Mar80] MARYANSKY, F.J. *Digital Computer Simulation*. Hayden Book Company Inc., 1980.
- [Mar91] MARCUS, A.; DAM, A. van. *User Interface Developments for the Nineties*. IEEE Computer, v. 24, n. 9, pp. 49-57, Setembro, 1991.
- [Mei91] MEIRA, C.A.A. *Sobre Geradores de Aplicação*. Dissertação de Mestrado, ICMSC-USP, 1991.
- [Meu92] MEURANT, G. *Os Computadores Paralelos e o Futuro da Computação Científica*. I Escola de Computação Científica de Alto Desempenho, LNCC, Agosto, 1992.
- [Mit92] MITRANI, I. *Simulation Techniques for Discrete Event Systems*. Cambridge University Press, 1992.
- [Non97] NONATO, S.A. *Utilização de Redes de Petri em Sistemas Computacionais*. Minidissertação, ICMSC-USP, Abril, 1997.
- [Oga97] OGAWA, C.L. *Aprimoramento e Expansão do Ambiente de Simulação Automático ASiA*. Monografia do Projeto de Graduação, ICMSC-USP, Junho, 1997.
- [Oga97a] OGAWA, C.L. *Extensão do Ambiente de Simulação Automático (ASiA) para Simulação de Redes de Computadores*. Plano de Estudo e Pesquisa, ICMSC-USP, Agosto, 1997.
- [Oli96] OLIVEIRA, L.L. *Um Módulo de Análise de Resultados para o ASiA*. Minidissertação, ICMSC-USP, Maio, 1996.
- [Ols97] OLSEM, M. R. *Incremental Reengineering: a better approach to software engineering*. Online. Available from World Wide Web: <http://stsc.hill.af.mil/crosstalk/1997/jun/incremental.html>, Junho, 1997.

- [Osb90] OSBORNE, W.M.; CHIKOFFSKY, E.J. *Fitting Pieces to the Maintenance Puzzle*. IEEE Software, v. 7, n. 1, pp. 11-12, Janeiro, 1990.
- [Ped96] PEDROSO Jr., W. *Módulo Visual: Uma Interface Gráfica para o Ambiente de Simulação Automático*. Dissertação de Mestrado, ICMSC-USP, Abril, 1996.
- [Ped96a] PEDROSO Jr., W.; SANTANA, M.J.; CALÔNEGO Jr., N.; SANTANA, R.H.C. *ASiAModV - Módulo Visual para o ASiA: Descrição de Implementação*. Notas Didáticas do ICMSC-USP, n. 23, Abril, 1996.
- [Peg91] PEGDEN, C.D.; SHANNON, R.E.; SADOWSKI, R.P. *Introduction to Simulation Using SIMAN*. McGraw-Hill Internacional Editions, 1991.
- [Pen96] PENTEADO, R.A.D. Um método para engenharia reversa orientada a objetos. Tese de Doutorado, IFSC-USP, 1996.
- [Pre92] PRESSMAN, R.S. *Software Engineering: Practitioner's Approach*. McGraw-Hill Book Company Europe, 3ª Edição, 1992.
- [Pri96] PRIOLI, M.C. *Integração de Módulos e Expansão do Gerador de Aplicação do Ambiente de Simulação Automático ASiA*. Relatório de Iniciação Científica, ICMSC-USP, Setembro, 1996.
- [San94] SANTANA, R.H.C.; SANTANA, M.J.; ORLANDI, R.C.G.S.; SPOLON, R.; CALÔNEGO Jr., N. *Técnicas para Avaliação de Desempenho de Sistemas Computacionais*. Notas Didáticas do ICMSC-USP, n. 14, Setembro, 1994.
- [San94a] SANTANA, R.H.C. *Ambiente de Simulação Automático para Modelamento e Análise de Sistemas Distribuídos*. Projeto para Concurso de Professor Assistente do ICMSC-USP, 1994.
- [Sco98] *Scope of the UML*. Online. Available from World Wide Web: <<http://www.rational.com/www/docs/uml/html/summary/summary4.html>>. Outubro, 1998.
- [Shi98] SHIFFMAN, H. *Boosting Java Performance: Native Code & JIT Compilation*. Online. Available from World Wide Web: <<http://www.disordered.org/Java-JIT.html>>, Outubro, 1998.
- [Sne95] SNEED, H.M. *Planning the Reengineering of Legacy Systems*. IEEE Software, v. 12, n. 1, pp. 24-34, Janeiro, 1995.

- [Soa92] SOARES, L.F.G. *Modelagem e Simulação Discreta de Sistemas*. Editora Campus Ltda., 1992.
- [Spo94] SPOLON, R. *Um Gerador de Aplicação para um Ambiente de Simulação Automático*. Dissertação de Mestrado, ICMSC-USP, Agosto, 1994.
- [Spo94a] SPOLON, R. *Um Editor Gráfico para um Ambiente de Simulação Automático*. Dissertação de Mestrado, ICMSC-USP, Agosto, 1994.
- [Spo94b] SPOLON, R.; SPOLON, R.; SANTANA, M.J.; SANTANA, R.H.C. *Gerador de Aplicação para o ASiA: Descrição da Implementação*. Relatórios Técnicos do ICMSC-USP, n. 21, Julho, 1994.
- [Spo96] SPOLON, R.; SPOLON, R.; SANTANA, M.J.; SANTANA, R.H.C. *Automatic Generation of Discrete-System Simulation Programs*. Proceedings of the 1996 Summer Computer Simulation Conference. Portland, Oregon, 1996.
- [Uml98] *UML past, present, and future*. Online. Available from World Wide Web: <<http://www.rational.com/uml/html/summary/summary5.html>>, Outubro, 1998.

Bibliografia Auxiliar

- BRAND, M.G.J. van den; KLINT, P.; VERHOEF, C. *Reverse Engineering and System Renovation - An Annotated Bibliography*. Software Engineering Notes, v. 22, n. 1, Janeiro, 1997.
- COSTA, R.M. *FUSION-RE/I: Um Método de Engenharia Reversa para Auxiliar Manutenção de Software*. Dissertação de Mestrado, ICMSC-USP, Maio, 1997.
- MACDOUGALL, M.H. *System Level Simulation*. Digital System Design Automation: Languages, Simulation & Database, Computer Science Press Inc., pp. 1-115, 1975.
- QUINAIA, M.A. *Diretrizes para reengenharia de software com características de software legado*. Dissertação de Mestrado, ICMC-USP, Junho, 1998.
- SCACCHI, W. *Process Models in Software Engineering*. Encyclopedia of Software Engineering, Wiley Interscience Publications, v. 2, pp. 860-869, 1994.
- SPOLON, R.; SPOLON, R.; SANTANA, M.J.; SANTANA, R.H.C. *Desenvolvimento de um Gerador de Aplicação para Simulação de Sistemas Discretos*. Notas do ICMSC-USP, n. 11, Julho, 1994.
- WONG, K.; TILLEY, S.R.; MÜLLER, H.A.; STOREY, M.A.D. *Structural Redocumentation: A Case Study*. IEEE Software, v. 12, n. 1, pp. 46-54, Janeiro, 1995.

Anexos

Todos os modelos gerados durante a aplicação da Fusion-RE ao ASiA estão nas páginas a seguir, inclusive os relatórios não propostos na abordagem mas úteis à engenharia reversa.

O conteúdo do Anexo A compõem a documentação da versão atual do ASiA, incluindo os seguintes documentos:

- ⇒ Documento de Revitalização do Sistema;
- ⇒ Procedimentos do ASiA (relatório alfabético);
- ⇒ Diagrama de Temas;
- ⇒ Relatório de estruturas e elementos de dados;
- ⇒ Relatório de classificação dos procedimentos por pseudo-classes;
- ⇒ Modelo de Objetos do MASA para o tema temporário;
- ⇒ Modelo de Objetos do MASA para o tema permanente;
- ⇒ Modelo de Ciclo de Vida do MASA;
- ⇒ Modelo de Operações do MASA;
- ⇒ Procedimentos utilizados nas operações.

O Anexo B contém os modelos de análise para a reengenharia do ASiA, proposta no Capítulo 6. Esses modelos são:

- ⇒ Modelo de Objetos do MAS;
- ⇒ Especificação dos atributos para as classes do MAS;
- ⇒ Operações relacionadas as classes do MAS;
- ⇒ Modelo de Ciclo de Vida do MAS.

A.1. Documento de Revitalização do Sistema

Módulo	Descrição
Asia.cpp	É o módulo principal do ASiA, contém as funções que definem a interface gráfica com o usuário.
Procedimentos	
AskAboutSave	Exibe uma janela perguntado se quer salvar as mudanças num
	<i>Chamado por</i>
	WndProc
ChegadaDigProc	Função que trata da janela para a entrada dos dados da chegada.
	<i>Chama</i>
	DigAbrirArqDistribuicao (Editarq.cpp)
	DigSalvarArqDistribuicao (Editarq.cpp)
	GravarArqDistribuicao (Editarq.cpp)
	LerArqDistribuicao (Editarq.cpp)
	OkMessage
	<i>Chamado por</i>
	RecursoDigProc
ChildWndProc	Função que trata as mensagens enviadas pela janela filha (de
	<i>Chama</i>
	ApagaListaArq (Listarq.cpp)
	ApagaListaRecursos (Listrec.cpp)
	ApagaRamificacao (Listalg.cpp)
	ApagaSimbolo (Lista.cpp)
	ApagaUmaLigacao (Listalg.cpp)
	AtualizaNumFiliaServidor (Auxiliar.cpp)
	DesenhaLigacoes (Listalg.cpp)
	DesenhaRec (Lista.cpp)
	DesenhaRec2 (Lista.cpp)
	DesenhaSimbolos (Lista.cpp)
	DesLigacao (Listalg.cpp)
	DrawIconShow (Desenha.cpp)
	identRecSimboio (Lista.cpp)
	InicCondicoesLigacao (Listalg.cpp)
	InsereLista (Lista.cpp)
	insListaArq (Listarq.cpp)
	insListeLig (Listalg.cpp)
	JaTem10Saidas (Listalg.cpp)
	JaTemLigacao (Listalg.cpp)
	MoveSimboio (Lista.cpp)
	NaoPodeSerFim (Listalg.cpp)
	NaoPodeSerInicio (Listalg.cpp)
	NumBlocoDigProc
	ObtemRecursoLigado (Listalg.cpp)
	PontoEmLigacao (Listalg.cpp)
	PtoFinal (Lista.cpp)
	PtoInicial (Lista.cpp)
	RecursoJaParametrizado (Listrec.cpp)
	RemoveUltimoListeLigacoes (Listalg.cpp)
	RemoveUltimoSimboioLista (Lista.cpp)
	TemLigacoes (Listalg.cpp)
	TemListaRecursos (Listrec.cpp)
	TemQueParametrizarRecurso (Inforec.cpp)
	TemSimbolo (Lista.cpp)

Módulo	Descrição
	<i>Chamado por</i> WinMain
DistribuicaoDigProc	Função que trata da janela para a entrada dos dados da distribuição.
	<i>Chamado por</i> WndProc
DoCaption	Seta o nome de uma janela.
	<i>Chamado por</i> WndProc
EventosDigProc	Função que trata da janela para a entrada dos dados dos eventos.
	<i>Chamado por</i> RecursoDigProc
FilaDigProc	Função que trata da janela para a entrada dos dados da fila.
	<i>Chamado por</i> RecursoDigProc
IconWndProc	Função que trata as mensagens enviadas pela janela filha (de <i>Chama</i> DesinicCondiicoesLigacao (Listalg.cpp) Determinalcone (Desenha.cpp) DrawiconBar (Desenha.cpp) inicCondiicoesLigacao (Listalg.cpp) <i>Chamado por</i> WinMain
LigacaoPrincipalDigProc	Função que trata da janela para a entrada dos dados da ligação.
	<i>Chamado por</i> RecursoDigProc
LoopDigProc	Função que trata da janela para a entrada dos dados do loop.
	<i>Chamado por</i> RecursoDigProc
ModeloDigProc	Função que trata da janela para a entrada dos dados do modelo.
	<i>Chamado por</i> WndProc
NumBlocoDigProc	Função que trata da janela para a entrada do número de blocos.
	<i>Chamado por</i> ChildWndProc
OkMessage	Mostra uma janela informando um problema em algum arquivo.
	<i>Chamado por</i> ChegadaDigProc RecursoDigProc ServicoDigProc WndProc
RamificacaoDigProc	Função que trata da janela para a entrada dos dados da ramificação.
	<i>Chamado por</i> RecursoDigProc
RecursoDigProc	Função que trata da janela para a entrada dos dados do recurso. <i>Chama</i> ChegadaDigProc EFim (Listalg.cpp) Inicio (Listalg.cpp) EventosDigProc FilaDigProc InicializaDadosEventos (Auxiliar.cpp) InsListaEventos (Listeven.cpp)

Módulo**Descrição**

	InsListaLoop (Listloop.cpp) LigacaoPrincipalDigProc LoopDigProc MontaStringEntradaSalda (Auxiliar.cpp) MontaStringEntraSalSemRamificacao MontaStringEntSal1 (Auxiliar.cpp) ObtemNumeroRamificacoes (Listalig.cpp) ObtemRecursoComRamificacao (Listalig.cpp) ObtemRecursoLigado (Listalig.cpp) OkMessage ParametrizouChegada (Inforec.cpp) ParametrizouFila (Inforec.cpp) ParametrizouServico (Inforec.cpp) RamificacaoDigProc ServicoDigProc TransformaNumero (Auxiliar.cpp) TransformaString (Auxiliar.cpp) VerificaConsistenciaEventos (Consist.cpp) VerificaConsistenciaLoop (Consist.cpp) VerificaConsistenciaRecChegada VerificaConsistenciaRecFila (Consist.cpp) VerificaConsistenciaRecLigacaoPrincipal VerificaConsistenciaRecRamificacao VerificaConsistenciaRecServico (Consist.cpp) VerificaTotal (Consist.cpp)
	Chamado por WndProc
ServicoDigProc	Função que trata da janela para a entrada dos dados do serviço. Chama DigAbrirArqDistribuicao (Editarq.cpp) DigSaivarArqDistribuicao (Editarq.cpp) GravarArqDistribuicao (Editarq.cpp) LerArqDistribuicao (Editarq.cpp) OkMessage
	Chamado por RecursoDigProc
TipoArquiteturaDigProc	Função que trata da janela para a entrada do tipo de arquitetura. Chamado por WndProc
WinMain	Função principal de um programa Windows. Chama ChildWndProc IconWndProc WndProc
WndProc	Função do Windows que trata as mensagens enviadas pelo sistema. Chama ApagaLista (Lista.cpp) ApagaListaArq (Listarq.cpp) ApagaListaDist (Listdist.cpp) ApagaListaEventos (Listeven.cpp) ApagaListaLigacoes (Listalig.cpp) ApagaListaLoop (Listloop.cpp) ApagaListaRecursos (Listrec.cpp) ApenasUmaSalda (Listalig.cpp) AskAboutSave AuxAbrirArqLigacoes (Editarq.cpp)

<i>Módulo</i>	<i>Descrição</i>
	ConverteExtNomeArq (Auxiliar.cpp) DistribuicaoDigProc DigAbrirArqEspecGrafica (Editarq.cpp) DigSalvarArqEspecGrafica (Editarq.cpp) DoCaption EInicio (Listalig.cpp) ExisteLoop (Listalig.cpp) Gerador (Gerador.cpp) GravarArqArquitet (Editarq.cpp) GravarArqEspecGrafica (Editarq.cpp) GravarArqEventos (Editarq.cpp) GravarArqLigacoes (Editarq.cpp) GravarArqLoop (Editarq.cpp) GravarArqModelo (Editarq.cpp) GravarArqRecursos (Editarq.cpp) GuardaDadosLoop (Listloop.cpp) GuardaDadosRecurso (Listrec.cpp) InicializaArqDistribuicao (Editarq.cpp) InicializaArqEspecGrafica (Editarq.cpp) InicializaDadosDistribuicao (Auxiliar.cpp) InicializaDadosEventos (Auxiliar.cpp) InicializaDadosLoop (Auxiliar.cpp) InicializaDadosModelo (Auxiliar.cpp) InicializaDadosRecursos (Auxiliar.cpp) InicializaNumFliaServidor (Auxiliar.cpp) InsListaDist (Listdist.cpp) InsListaRec (Listrec.cpp) InsListaRecChegada (Listrec.cpp) InsListaRecFila (Listrec.cpp) InsListaRecRamificacao (Listrec.cpp) InsListaRecServico (Listrec.cpp) JaTemFim (Listalig.cpp) JaTemInicio (Listalig.cpp) LerArqArquitet (Editarq.cpp) LerArqEspecGrafica (Editarq.cpp) LerArqEventos (Editarq.cpp) LerArqLigacoes (Editarq.cpp) LerArqLoop (Editarq.cpp) LerArqModelo (Editarq.cpp) LerArqRecursos (Editarq.cpp) LoopEProb (Listalig.cpp) ModeloDigProc OkMessage Parametriza (Listarq.cpp) RecursoDigProc RecursoJaParametrizado (Listrec.cpp) TemQueParametrizarRecurso (Inforec.cpp) TipoArquiteturaDigProc TiraCaminho (Auxiliar.cpp) TodosRecTemLigParaOutro (Listalig.cpp) TransfereDadosAnteriores (Listrec.cpp) VerificaConsistenciaDistribulcao (Consist.cpp) VerificaConsistenciaModelo (Consist.cpp) VerificaConsistenciaRec (Consist.cpp) <i>Chamado por</i> WinMain
Auxiliar.cpp	Contém rotinas que inicializam e atualizam dados e manipulam cadeias de caracteres.

Módulo**Descrição**

AtualizaNumFilaServidor	<p>Seta as variáveis referentes ao número de filas e servidores dependendo do tipo do recurso.</p> <p><i>Chamado por</i> ChildWndProc (Asia.cpp)</p>
ConverteExtNomeArq	<p>Retorna o nome de um arquivo com a extensão fornecida como parâmetro.</p> <p><i>Chamado por</i> Abre_Arquivo_Saida (Util181.c) Define_Arq_Saida (Gerador.cpp) Distrib_Tabela (Dist_tab.c) Gera_Primeiro_Evento (Gerador.cpp) WndProc (Asia.cpp)</p>
InicializaDadosDistribuição	<p>Inicializa atributos da distribuição para o modelo.</p> <p><i>Chamado por</i> WndProc (Asia.cpp)</p>
InicializaDadosEventos	<p>Inicializa variável para ocorrência de eventos.</p> <p><i>Chamado por</i> RecursoDigProc (Asia.cpp) WndProc (Asia.cpp)</p>
InicializaDadosLoop	<p>Inicializa os dados para os ciclos de repetição.</p> <p><i>Chamado por</i> WndProc (Asia.cpp)</p>
InicializaDadosModelo	<p>Inicializa todos os parâmetros referentes ao modelo.</p> <p><i>Chamado por</i> WndProc (Asia.cpp)</p>
InicializaDadosRecursos	<p>Inicializa todos os atributos de um recurso.</p> <p><i>Chamado por</i> WndProc (Asia.cpp)</p>
InicializaNumFilaServidor	<p>Inicializa as variáveis referentes ao número de filas e de servidores.</p> <p><i>Chamado por</i> WndProc (Asia.cpp)</p>
MontaStringEntradaSaida	<p>Condensa parâmetros fornecidos em uma cadeia. Utilizado quando o recurso possui ramificação.</p> <p><i>Chamado por</i> Probabilidade (Listarq.cpp) RecursoDigProc (Asia.cpp)</p>
MontaStringEntraSaiSemRamificacao	<p>Monta cadeia padrão para recurso sem ramificação.</p> <p><i>Chamado por</i> Disco (Listarq.cpp) Entrada (Listarq.cpp) Memoria (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) RecursoDigProc (Asia.cpp) Saida (Listarq.cpp)</p>
MontaStringEntSai1	<p>Monta cadeia padrão para recurso com loop.</p> <p><i>Chamado por</i> Loop (Listarq.cpp) RecursoDigProc (Asia.cpp)</p>
TiraCaminho	<p>Extrai nome do arquivo do path fornecido.</p> <p><i>Chamado por</i></p>

Módulo	Descrição
TransformaNumero	<p>WndProc (Asia.cpp)</p> <p>Retira a porcentagem da cadeia construída por</p> <p><i>Chamado por</i></p> <p>Probabilidade (Listarq.cpp)</p> <p>RecursoDigProc (Asia.cpp)</p>
TransformaString	<p>Transforma um valor de porcentagem (probabilidade de uma ramificação) em string.</p> <p><i>Chamado por</i></p> <p>Probabilidade (Listarq.cpp)</p> <p>RecursoDigProc (Asia.cpp)</p>
Consist.cpp	<p>Contém as funções que verificam a consistência dos dados fornecidos pelo usuário.</p>
Procedimentos	
VerificaConsistenciaDistribuicao	<p>Verifica a consistência dos dados da Distribuição Tabelaada digitada pelo usuário.</p> <p><i>Chama</i></p> <p>VerificaInteiro</p> <p><i>Chamado por</i></p> <p>WndProc (Asia.cpp)</p>
VerificaConsistenciaEventos	<p>Verifica a consistência dos dados dos eventos chegada digitados pelo usuário.</p> <p><i>Chama</i></p> <p>VerificaInteiro</p> <p>VerificaReal</p> <p><i>Chamado por</i></p> <p>RecursoDigProc (Asia.cpp)</p>
VerificaConsistenciaLoop	<p>Verifica a consistência do número de voltas do loop - Controle de</p> <p><i>Chama</i></p> <p>VerificaInteiro</p> <p><i>Chamado por</i></p> <p>RecursoDigProc (Asia.cpp)</p>
VerificaConsistenciaModelo	<p>Verifica a consistência dos dados da caixa Modelo.</p> <p><i>Chama</i></p> <p>VerificaInteiro</p> <p>VerificaReal</p> <p><i>Chamado por</i></p> <p>WndProc (Asia.cpp)</p>
VerificaConsistenciaRec	<p>Verifica consistência dos dados dos recursos.</p> <p><i>Chama</i></p> <p>TemMais1Ciclo1Ligacao (Listalig.cpp)</p> <p>VerificaInteiro</p> <p><i>Chamado por</i></p> <p>WndProc (Asia.cpp)</p>
VerificaConsistenciaRecChegada	<p>Verifica a consistência dos dados dos recursos - chegada.</p> <p><i>Chama</i></p> <p>VerificaInteiro</p> <p>VerificaReal</p> <p><i>Chamado por</i></p> <p>RecursoDigProc (Asia.cpp)</p>
VerificaConsistenciaRecFila	<p>Verifica a consistência dos dados dos recursos - fila.</p> <p><i>Chama</i></p> <p>VerificaInteiro</p>

<i>Módulo</i>	<i>Descrição</i>
	<i>Chamado por</i> RecursoDigProc (Asia.cpp)
VerificaConsistenciaRecLigacaoPrincipal	Verifica a consistência dos dados dos recursos - Ligação Principal.
	<i>Chama</i> VerificaInteiro VerificaReal
	<i>Chamado por</i> RecursoDigProc (Asia.cpp)
VerificaConsistenciaRecRamificacao	Verifica a consistência da porcentagem da realimentação.
	<i>Chama</i> VerificaInteiro VerificaReal
	<i>Chamado por</i> RecursoDigProc (Asia.cpp)
VerificaConsistenciaRecServico	Verifica a consistência dos dados dos recursos - serviço.
	<i>Chama</i> VerificaInteiro VerificaReal
	<i>Chamado por</i> RecursoDigProc (Asia.cpp)
VerificaInteiro	Verifica se uma string corresponde a representação correta de um número inteiro.
	<i>Chamado por</i> VerificaConsistenciaDistribulcao VerificaConsistenciaEventos VerificaConsistenciaLoop VerificaConsistenciaModelo VerificaConsistenciaRec VerificaConsistenciaRecChegada VerificaConsistenciaRecFila VerificaConsistenciaRecLigacaoPrincipal VerificaConsistenciaRecRamificacao VerificaConsistenciaRecServico
VerificaReal	Verifica se uma string corresponde a representação correta de um número real, da forma x.y.
	<i>Chamado por</i> VerificaConsistenciaEventos VerificaConsistenciaModelo VerificaConsistenciaRecChegada VerificaConsistenciaRecLigacaoPrincipal VerificaConsistenciaRecRamificacao VerificaConsistenciaRecServico
VerificaTotal	Verifica se a soma das porcentagens passou de 100%.
	<i>Chamado por</i> RecursoDigProc (Asia.cpp)
Desenha.cpp	Manipula a barra de ícones da interface gráfica.
<i>Procedimentos</i>	
Determinaicone	Determina sobre que ícone o usuário clicou.
	<i>Chamado por</i> IconWndProc (Asia.cpp)
DrawIconBar	Desenha a barra de ícones.
	<i>Chamado por</i>

Módulo	Descrição
DrawIconShow	IconWndProc (Asia.cpp) Desenha a seta indicando qual recurso está selecionado. <i>Chamado por</i> ChildWndProc (Asia.cpp)
Dist_tab.c	Módulo com as funções que geram a implementação da distribuição tabelada definida pelo usuário (o nome do arquivo é especificado pelo próprio usuário). A saída é armazenada no arquivo DTAB.C, que vai ser incluído no arquivo saída em smpi.
Procedimentos	
Convert	Converte um número do tipo float para uma string.
Cop	Copia uma string para a string Buf. <i>Chama</i> Limpa_Buf <i>Chamado por</i> Monta_Array_Acumulado
Distrib_Tabela	Verifica se usuário definiu novas distribuições; em caso afirmativo, gera o include e a distribuição. <i>Chama</i> ConverteExtNomeArq (Auxiliar.cpp) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito (Gerador.cpp)
Grava_Buf	Grava a string Buf no arquivo de saída. <i>Chamado por</i> Escalona_Release (Gerador.cpp) Gera_Proxima_Chegada (Gerador.cpp) Monta_Array_Acumulado
ItOA_Defi	Converte um número do tipo inteiro sem sinal para string. <i>Chama</i> Reverte_String <i>Chamado por</i> Monta_Array_Acumulado
Limpa_Buf	Atribui ' ' para a variável Buf. <i>Chamado por</i> Cop
Monta_Array_Acumulado	Verificado o número de entradas no arquivo TAB.DAT, gera o vetor contendo a probabilidade acumulada. <i>Chama</i> Cop Grava_Buf ItOA_Defi Verifica_Tabela <i>Chamado por</i> Escalona_Release (Gerador.cpp) Gera_Proxima_Chegada (Gerador.cpp)
Reverte_String	Inverte uma string, utilizada pela rotina ItOA_Defi. <i>Chamado por</i> ItOA_Defi
Verifica_Tabela	Verificar o número de entradas no arquivo TAB.DAT, para gerar as estruturas de dados e primitivas necessárias a implementação da

Módulo	Descrição
	distribuição definida pelo usuário. <i>Chamado por</i> Monta_Array_Acumulado
Editarq.cpp	Módulo com as funções para manipulação dos arquivos do modelo.
Procedimentos	
AuxAbrirArqLigacoes	Auxilia na abertura do arquivo de ligações. <i>Chamado por</i> WndProc (Asia.cpp)
AuxAbrirArqLoop	Auxilia na abertura do arquivo de loop.
AuxSalvarArqLigacoes	Auxilia na armazenagem do arquivo de ligações.
AuxSalvarArqLoop	Auxilia na armazenagem do arquivo de loop.
DlgAbrirArqDistribuicao	Exibe a caixa de diálogo para entrada do nome do arquivo de distribuição tabelada a abrir. <i>Chamado por</i> ChegadaDlgProc (Asia.cpp) ServicoDlgProc (Asia.cpp)
DlgAbrirArqEspecGrafica	Exibe a caixa de diálogo para entrada do nome do arquivo a abrir. <i>Chamado por</i> WndProc (Asia.cpp)
DlgSalvarArqDistribuicao	Exibe a caixa de diálogo para entrada do nome do arquivo de distribuição tabelada a salvar. <i>Chamado por</i> ChegadaDlgProc (Asia.cpp) ServicoDlgProc (Asia.cpp)
DlgSalvarArqEspecGrafica	Exibe a caixa de diálogo para entrada do nome do arquivo a salvar. <i>Chamado por</i> WndProc (Asia.cpp)
GravarArqArquitet	Grava os dados da lista de arquitetura no arquivo de arquitetura. <i>Chamado por</i> WndProc (Asia.cpp)
GravarArqDistribuicao	Grava os dados da lista de distribuição no arquivo de distribuição. <i>Chamado por</i> ChegadaDlgProc (Asia.cpp) ServicoDlgProc (Asia.cpp)
GravarArqEspecGrafica	Grava os dados da lista de especificação gráfica no arquivo de especificação gráfica. <i>Chamado por</i> WndProc (Asia.cpp)
GravarArqEventos	Grava os dados da lista de eventos no arquivo de eventos. <i>Chamado por</i> WndProc (Asia.cpp)
GravarArqLigacoes	Grava os dados da lista de ligações no arquivo de ligações. <i>Chamado por</i> WndProc (Asia.cpp)
GravarArqLoop	Grava os dados da lista de loop no arquivo de loop. <i>Chamado por</i> WndProc (Asia.cpp)

Módulo	Descrição
GravarArqModelo	Grava os dados das variáveis de parametrização do modelo no arquivo de especificação gráfica. <i>Chamado por</i> WndProc (Asia.cpp)
GravarArqRecursos	Grava os dados da lista de recursos no arquivo de recursos. <i>Chamado por</i> WndProc (Asia.cpp)
InicializaArqDistribuicao	Seta as variáveis para exibir a caixa de abrir arquivos de distribuição tabelada. <i>Chamado por</i> WndProc (Asia.cpp)
InicializaArqEspecGrafica	Seta as variáveis para exibir a caixa de abrir arquivos de especificação gráfica. <i>Chamado por</i> WndProc (Asia.cpp)
LerArqArquitet	Lê os dados do arquivo de arquitetura e insere na lista de arquitetura. <i>Chama</i> InsListaArq (Listarq.cpp) <i>Chamado por</i> WndProc (Asia.cpp)
LerArqDistribuicao	Lê os dados do arquivo de distribuição tabelada e insere na lista de distribuição. <i>Chama</i> InsListaDist (Listdist.cpp) <i>Chamado por</i> ChegadaDlgProc (Asia.cpp) ServicoDlgProc (Asia.cpp)
LerArqEspecGrafica	Lê os dados do arquivo de especificação gráfica e insere na lista de especificação gráfica. <i>Chama</i> InsereLista (Lista.cpp) <i>Chamado por</i> WndProc (Asia.cpp)
LerArqEvantos	Lê os dados do arquivo de eventos e insere na lista de eventos. <i>Chama</i> InsListaEventos (Listeven.cpp) <i>Chamado por</i> WndProc (Asia.cpp)
LerArqLigacoes	Lê os dados do arquivo de ligações e insere na lista de ligações. <i>Chama</i> InsListaLig (Listalig.cpp) <i>Chamado por</i> WndProc (Asia.cpp)
LerArqLoop	Lê os dados do arquivo de loop e insere na lista de loop. <i>Chama</i> InsListaLoop (Listloop.cpp) <i>Chamado por</i> WndProc (Asia.cpp)
LerArqModelo	Lê os dados do arquivo de modelo e os atribui às variáveis globais. <i>Chamado por</i> WndProc (Asia.cpp)

Módulo	Descrição
LerArqRecursos	Lê os dados do arquivo de recursos e insere na lista de recursos. <i>Chama</i> InsListaRec (Listrec.cpp) InsListaRecChegada (Listrec.cpp) InsListaRecFila (Listrec.cpp) InsListaRecRamificacao (Listrec.cpp) InsListaRecServico (Listrec.cpp) <i>Chamado por</i> WndProc (Asia.cpp)
Gerador.cpp	Módulo com as funções para gerar o programa de simulação a partir da especificação gráfica.
<i>Procedimentos</i> Define_Arq_Seida	Declara e abre para escrita o arquivo de saída no código do programa de simulação. Este arquivo de saída conterá o relatório da simulação. <i>Chama</i> ConverteExtNomeArq (Auxiliar.cpp) Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Define_BMeans	Define as variáveis necessárias para utilizar o método de análise Batch Means. <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Define_Estat_Fila_Vazia	Define as variáveis que serão utilizadas para os cálculos estatísticos: porcentagem de vezes em que o cliente encontra fila vazia, associada ao recurso especificado. <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Define_Estat_Max_Min	Define as variáveis que serão utilizadas para os cálculos estatísticos: tamanho máximo e mínimo da fila associada ao recursos <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Define_Reesc	Define as variáveis e tabela necessárias para o reescalonamento de um evento quando o recurso não tem fila. O campo reesc no arquivo MODELO.DAT determina se existem recursos sem fila no modelo implementado. <i>Chama</i> Define_Var_Reescalona Monta_Tabela_Reescalona (Reesc.c) <i>Chamado por</i> Le_Gabarito
Define_Tempo_Max	Gera o comando para limitar a execução da simulação por tempo, por número de clientes que deixam o sistema, ou pelo número de ciclos (tours) do cliente no sistema, para a declaração da variável a ser utilizada.

<i>Módulo</i>	<i>Descrição</i>
	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Define_Var_Backoff	Gera as variáveis para o algoritmo de Backoff (recurso sem fila) - (Algoritmo ethernet). <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) ltoa_Def (Util181.c) <i>Chamado por</i> Le_Gabarito
Define_Var_Reescalona	Gera os contadores para as diversas tentativas de utilização de um recurso sem fila. São os índices para a tabela de reescalonamento para cada um dos recursos do modelo. <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Define_Reesc
Define_Var_Server	Gera as variáveis para a definição dos recursos que compõem o modelo a ser implementado. <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Define_Var_Tempos	Gera as variáveis para os tempos médios entre chegadas e serviço. <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Escalona_Release	Implementa a primitiva para a liberação do recurso. <i>Chama</i> Copy (Util181.c) Gera_Estat_Fila_Vazia Gera_Estat_Max_Min Gera_Stream Grava_Buf (Dist_tab.c) Grava_Buffer (Util181.c) ltoa_Def (Util181.c) Monta_Array_Acumulado (Dist_tab.c) <i>Chamado por</i> Gera_Eventos Gera_Release
Gera_Break	Gera a primitiva break. <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Gera_Eventos Gera_Release
Gera_Case	Gera a primitiva case.

Módulo	Descrição
	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Gera_Eventos Gera_Release
Gera_Cause	Implementa o comando cause.
	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Gera_Contadores	Gera as variáveis (contadores) para o caso de número de voltas definidas para os clientes em um determinado servidor.
	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) Itoa_Def (Util181.c) <i>Chamado por</i> Le_Gabarito
Gera_Def_Server	Gera o comando FACILITY para a definição dos recursos que compõem o sistema (um recurso corresponde a uma única fila e n servidores paralelos).
	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Le_Gabarito
Gera_Delay	Gera um período de espera na simulação.
	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) Itoa_Def (Util181.c) <i>Chamado por</i> Gera_Eventos
Gera_Entradas_Saidas	Gera a saída de um recurso (que pode ser a entrada para um outro recurso ou a saída do sistema).
	<i>Chama</i> Atoi_Def (Util181.c) Copy (Util181.c) Copy_SubString (Util181.c) Grava_Buffer (Util181.c) Itoa_Def (Util181.c) VerificaIqual VerificaProb <i>Chamado por</i> Gera_Release
Gera_Estat_Fila_Vazia	Gera as estatísticas para o cálculo da porcentagem de vezes em que o cliente encontra fila vazia.
	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) <i>Chamado por</i> Escalona_Release
Gera_Estat_Max_Min	Gera as estatísticas para o cálculo do número de clientes em cada fila determinada pelo usuário.

Módulo**Descrição**

Gera_Eventos	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c)
	<i>Chamado por</i> Escalona_Release
	Gera os eventos que constituem a simulação.
Gera_Filas_Separadas	<i>Chama</i> Escalona_Release Gera_Break Gera_Case Gera_Delay Gera_Filas_Separadas Gera_Proxima_Chegada Gera_Release
	<i>Chamado por</i> Le_Gabarito
	Gera a entrada para sistemas com mais de uma fila de entrada.
Gera_Loop	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) Itoa_Def (Util181.c)
	<i>Chamado por</i> Gera_Eventos
	Gera o comando de repetição, limitando a simulação por tempo, número de clientes que passam pelo sistema, número de voltas no sistema (sistema fechado - modelo do servidor central), ou método de análise Batch Means.
Gera_Matodo_Analise_Saida	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c)
	<i>Chamado por</i> Le_Gabarito
	Gera a chamada ao módulo init_bm(m0,m1) do smpl, para a utilização do Método Batch Means (estimar um intervalo de confiança para a média de uma variável de saída da simulação).
Gera_Primeiro_Evento	<i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c)
	<i>Chamado por</i> Le_Gabarito
	Escalona o primeiro evento a ocorrer no caso de sistema com uma única entrada, ou todos os eventos que devem ser escalonados antes do início da simulação para modelos de sistemas fechados.
Gera_Proxima_Chegada	<i>Chama</i> ConverteExtNomeArq (Auxiliar.cpp) Copy (Util181.c) Grava_Buffer (Util181.c) Itoa_Def (Util181.c)
	<i>Chamado por</i> Le_Gabarito
	Para modelos fechados, escalona a próxima chegada.
	<i>Chama</i> Copy (Util181.c) Gera_Stream

Módulo	Descrição
	Grava_Buf (Dist_tab.c) Grava_Buffer (Util181.c) Itoa_Def (Util181.c) Monta_Array_Acumulado (Dist_tab.c) Chamado por Gera_Eventos
Gera_Relat_Est_Max_Min	Gera relatório com as estatísticas das filas dos recursos especificados pelo usuário. Chama Copy (Util181.c) Grava_Buffer (Util181.c) Chamado por Le_Gabarito
Gera_Relat_Fila_Vazia	Gera relatório com as estatísticas das filas dos recursos especificados pelo usuário: porcentagem de vezes na qual o cliente Chama Copy (Util181.c) Grava_Buffer (Util181.c) Chamado por Le_Gabarito
Gera_Relatorio_Final	Gera o relatório final (padrão do smpl) no programa de simulação. Chama Copy (Util181.c) Grava_Buffer (Util181.c) Chamado por Le_Gabarito
Gera_Release	Gera o comando RELEASE para a liberação do recurso. Chama Auxilia_Gera_Release (Util181.c) Copy (Util181.c) Escalona_Release Gera_Break Gera_Case Gera_Entradas_Saidas Grava_Buffer (Util181.c) Itoa_Def (Util181.c) Chamado por Gera_Eventos
Gera_Stream	Gera o comando para determinar qual a sequência a ser utilizada na geração do número aleatório. Se o usuário não especificar nenhuma semente (colocar zero na opção seq_num_aleat do arquivo RECURSOS.DAT), smpl utiliza a sequência corrente. Quando o comando smpl é executado, a sequência é setada para 1 e incrementada a medida em que é utilizada. Chama Copy (Util181.c) Grava_Buffer (Util181.c) Chamado por Auxilia_Gera_Release (Util181.c) Escalona_Release Gera_Proxima_Chegada
Gera_Switch	Gera o comando switch(). Chama Copy (Util181.c) Grava_Buffer (Util181.c) Chamado por

Módulo	Descrição
Gerador	<p>Le_Gabarito</p> <p>Função principal do gerador. Abre o arquivo de gabarito, chama a função Le_Gabarito e fecha todos os arquivos utilizados.</p> <p><i>Chama</i></p> <p>Abre_Arquivo_Salda (Util181.c)</p> <p>Fecha_arquivos (Util181.c)</p> <p>Le_Gabarito</p> <p>Prepara_Estruturas</p> <p><i>Chamado por</i></p> <p>WndProc (Asia.cpp)</p>
Le_Gabarito	<p>Através dos comandos do arquivo GABARITO.DAT, determina a próxima primitiva a ser executada.</p> <p><i>Chama</i></p> <p>Copy (Util181.c)</p> <p>Define_Arq_Salda</p> <p>Define_BMeans</p> <p>Define_Estat_Fila_Vazia</p> <p>Define_Estat_Max_Min</p> <p>Define_Reesc</p> <p>Define_Tempo_Max</p> <p>Define_Var_Backoff</p> <p>Define_Var_Server</p> <p>Define_Var_Tempos</p> <p>Distrib_Tabala (Dist_tab.c)</p> <p>Gera_Causa</p> <p>Gera_Contadores</p> <p>Gera_Def_Server</p> <p>Gera_Eventos</p> <p>Gera_Loop</p> <p>Gera_Metodo_Analise_Salda</p> <p>Gera_Primeiro_Evento</p> <p>Gera_Relat_Est_Max_Min</p> <p>Gera_Relat_Fila_Vazia</p> <p>Gera_Relatorio_Final</p> <p>Gera_Switch</p> <p>Noma_Parametro</p> <p>Warmup</p> <p><i>Chamado por</i></p> <p>Gerador</p>
Nome_Paramatro	<p>Gera o comando SMPL para a inicialização do modelo de simulação.</p> <p><i>Chama</i></p> <p>Copy (Util181.c)</p> <p>Grava_Buffer (Util181.c)</p> <p><i>Chamado por</i></p> <p>Le_Gabarito</p>
Prepara_Estruturas	<p>Cria uma lista ligada com os recursos que fazem parte da especificação do modelo, e estão armazenadas no arquivo de especificação (.esp). A lista é removida após a execução.</p> <p><i>Chamado por</i></p> <p>Gerador</p>
VerificaIgual	<p>Verifica se os valores que estão em string a partir da posição pos1 até a posição pos1+tam são iguais aos valores a partir da posição pos2.</p> <p><i>Chamado por</i></p> <p>Gera_Entradas_Saldas</p>
VerificaProb	<p>Verifica se os valores que estão em ent_sai a partir da posição pos são iguais aos valores de prob.</p>

Módulo**Descrição**

Função utilizada na função Gera_Entradas_Saidas.

Chamado por

Gera_Entradas_Saidas

Warmup

Executa o warmup (período inicial para que o sistema entre em equilíbrio), para que o mesmo possa ser representado de maneira mais realística. Após um tempo estabelecido, todas as estatísticas são zeradas.

Chama

Copy (Util181.c)

Grava_Buffer (Util181.c)

Chamado por

Le_Gabarito

Inforec.cpp

Contém as funções que retornam informações a respeito da parametrização do recurso selecionado.

Procedimentos

ParametrizouChegada

Indica que os dados da chegada já foram fornecidos.

Chamado por

RecursoDigProc (Asia.cpp)

ParametrizouFila

Indica que os dados da fila já foram fornecidos.

Chamado por

RecursoDigProc (Asia.cpp)

ParametrizouServico

Indica que os dados do serviço já foram fornecidos.

Chamado por

RecursoDigProc (Asia.cpp)

TemQueParametrizarRecurso

Indica que os dados dos recursos devem ser fornecidos.

Chamado por

ChildWndProc (Asia.cpp)

WndProc (Asia.cpp)

Lista.cpp

Módulo que contém as funções para manipulação da lista de especificação gráfica.

Procedimentos

ApagaLista

Apaga a lista de primitivas da memória.

Chamado por

ApagaSimbolo

WndProc (Asia.cpp)

ApagaSimbolo

Apaga o símbolo de recurso indicado pelo usuário. A identificação do mesmo é removida da lista de símbolos. Não faz nada se o usuário não clicou sobre algum símbolo. Obtém as coordenadas do retângulo inválido para não desenhar toda a tela novamente. Caso o símbolo tenha ligações de ou para outro recurso, elas também são apagadas.

Chama

ApagaLigacao (Listalig.cpp)

ApagaLista

ApagaRamificacao (Listalig.cpp)

ApagaUmRegistro (Listarq.cpp)

BuscaRamificacaoQueChega (Listalig.cpp)

ClicRecurso

RetornaPosicaoRamificacao (Listalig.cpp)

VerificaSeTemLigacao (Listalig.cpp)

Chamado por

ChildWndProc (Asia.cpp)

Módulo	Descrição
ClicRecurso	Verifica se o ponto clicado está sobre um desenho e se o mesmo é um recurso (centro de serviço). <i>Chamado por</i> ApagaSimbolo IdentRecSimbolo
DesBlocoDisco	Desenha bloco de discos. <i>Chama</i> DesPonto DesRec1 <i>Chamado por</i> DesenhaRec2 DesenhaSimbolos
DesBlocoMem	Desenha bloco de memórias. <i>Chama</i> DesPonto DesRec1 <i>Chamado por</i> DesenhaRec2 DesenhaSimbolos
DesBlocoProc	Desenha bloco de processadores. <i>Chama</i> DesPonto DesRec1 <i>Chamado por</i> DesenhaRec2 DesenhaSimbolos
DesenhaRec	Chama a função para desenhar um recurso, de acordo com o tipo especificado. <i>Chama</i> DesRec1 DesRec2 DesRec3 DesRec4 <i>Chamado por</i> ChildWndProc (Asia.cpp)
DesenhaRec2	Desenha os recursos do Módulo Arquitetura. <i>Chama</i> DesBlocoDisco DesBlocoMem DesBlocoProc DesPipeline <i>Chamado por</i> ChildWndProc (Asia.cpp)
DesenhaSimbolos	Desenha os símbolos na tela - usado na mensagem WM_PAINT. Percorre toda a lista de símbolos, desenhando cada um deles. <i>Chama</i> AjustaVariaveis (Listarq.cpp) DesBlocoDisco DesBlocoMem DesBlocoProc DesPipeline DesRec1 DesRec2 DesRec3

<i>Módulo</i>	<i>Descrição</i>
	DesRec4 PertenceLista (Listarq.cpp) ProcuraNome (Listrec.cpp) <i>Chamado por</i> ChildWndProc (Asla.cpp)
DesFila	Função que desenha uma fila na tela. <i>Chamado por</i> DesRec1 DesRec2 DesRec3 DesRec4
DesLinha	Desenha uma linha (em recursos do tipo 2). <i>Chamado por</i> DesRec3 DesRec4
DesPipeline	Desenha um pipeline. <i>Chama</i> DesPonto DesRec1 <i>Chamado por</i> DesenhaRec2 DesenhaSímbolos
DesPonto	Desenha um ponto na tela. <i>Chamado por</i> DesBlocoDisco DesBlocoMem DesBlocoProc DesPipeline DesRec2 DesRec3 DesRec4
DesRec1	Desenha um recurso do tipo 1 (uma fila, um servidor). Utilizada na função DesenhaSímbolos, em WM_PAINT. <i>Chama</i> DesFila DesServ <i>Chamado por</i> DesBlocoDisco DesBlocoMem DesBlocoProc DesenhaRec DesenhaSímbolos DesPipeline
DesRec2	Desenha um recurso do tipo 2 (uma fila, vários servidores). Utilizada na função DesenhaSímbolos, em WM_PAINT. <i>Chama</i> DesFila DesPonto DesServ <i>Chamado por</i> DesenhaRec DesenhaSímbolos
DesRec3	Desenha um recurso do tipo 3 (várias filas, um servidor). Utilizada na função DesenhaSímbolos, em WM_PAINT. <i>Chama</i>

Módulo	Descrição
	DesFila DesLinha DesPonto DesServ <i>Chamado por</i> DesenhaRec DesenhaSimbolos
DesRec4	Desenha um recurso do tipo 4 (várias filias, vários servidores). Utilizada na função DesenhaSimbolos, em WM_PAINT. <i>Chama</i> DesFila DesLinha DesPonto DesServ <i>Chamado por</i> DesenhaRec DesenhaSimbolos
DesServ	Função que desenha um servidor na tela. <i>Chamado por</i> DesRec1 DesRec2 DesRec3 DesRec4
IdentRecSimbolo	Verifica o tipo do recurso ao qual pertence o símbolo clicado - usado quando o usuário clica num recurso para parametrizá-lo. Também retorna a posição em que deve ser desenhada a seta que mostra qual o recurso a ser selecionado. <i>Chama</i> ClicRecurso <i>Chamado por</i> ChildWndProc (Asia.cpp)
InsererLista	Insere item de desenho na lista dinâmica. <i>Chamado por</i> ChildWndProc (Asia.cpp) LerArqEspecGrafica (Editarq.cpp)
MoveSimbolo	Move o recurso de uma posição para outra. A posição é modificada na lista de símbolos. Não faz nada se o usuário não clicou sobre algum símbolo. <i>Chama</i> PertenceLista (Listarq.cpp) VerificaLigacao (Listalig.cpp) VerificaRamificacao (Listalig.cpp) <i>Chamado por</i> ChildWndProc (Asia.cpp)
PtoFinal	Verifica se o usuário clicou o mouse sobre um recurso, e em caso afirmativo, retorna a posição final para a ligação e a identificação do recurso. <i>Chamado por</i> ChildWndProc (Asia.cpp)
PtoInicial	Verifica se o usuário clicou o mouse sobre um recurso, e em caso afirmativo, retorna a posição de início da ligação e a identificação do recurso. <i>Chamado por</i> ChildWndProc (Asia.cpp)
RemoveUltimoSimboloLista	Remove o último símbolo da lista.

Módulo	Descrição
	<i>Chamado por</i> ChildWndProc (Asia.cpp)
TemSimbolo	Verifica se existe algum desenho do usuário - usado em WM_PAINT do procedimento de Janela ProcJan.
	<i>Chamado por</i> ChildWndProc (Asia.cpp)
Listalig.cpp	Contém as funções que manipulam a lista de ligações.
<i>Procedimentos</i>	
ApagaLigacao	Apaga a ligação indicada pelo usuário. A identificação da mesma é removida da lista de ligações da memória.
	<i>Chama</i> ApagaListaLigacoes
	<i>Chamado por</i> ApagaSimbolo (Lista.cpp)
ApagaListaLigacoes	Apaga a lista de ligações da memória.
	<i>Chamado por</i> ApagaLigacao ApagaRamificacao ApagaUmaLigacao WndProc (Asia.cpp)
ApagaRamificacao	Apaga a ramificação indicada pelo usuário. A identificação da mesma é removida da lista de ligações da memória.
	<i>Chama</i> ApagaListaLigacoes PertenceRamificacao
	<i>Chamado por</i> ApagaSimbolo (Lista.cpp) ChildWndProc (Asia.cpp)
ApagaUmaLigacao	Apaga a ligação indicada pelo usuário. A identificação da mesma é removida da lista de ligações da memória.
	<i>Chama</i> ApagaListaLigacoes PertenceLigacao
	<i>Chamado por</i> ChildWndProc (Asia.cpp)
ApenasUmaSaida	Verifica se o recurso tem apenas uma saída para outro ou para o fim.
	<i>Chamado por</i> AjustaCondicoes (Lstarq.cpp) WndProc (Asia.cpp)
BuscaRamificacaoQueChega	Busca a ramificação que chega ao recurso identre.
	<i>Chamado por</i> ApagaSimbolo (Lista.cpp)
DesenhaLigacoes	Desenha todas as ligações e ramificações da lista de ligações.
	<i>Chama</i> DesLig DesRamificacao DesSeta
	<i>Chamado por</i> ChildWndProc (Asia.cpp)
DesinicCondicoesLigacao	Desinicializa variáveis necessárias para a captação dos pontos para ligar dois recursos. Utilizada depois que o ASIA captou os pontos para fazer a ligação entre os recursos.

Módulo**Descrição**

	<p><i>Chamado por</i> IconWndProc (Asia.cpp)</p>
DesLig	Desenha a ligação entre dois recursos, do ponto (x1,y1) (saída do recurso) até o ponto (x2,y2) (entrada do recurso).
	<p><i>Chamado por</i> DesenhaLigacoes DesLigacao</p>
DesLigacao	Desenha a ligação entre dois recursos, cujo ponto inicial e final foi especificado pelo usuário, com a utilização do mouse.
	<p><i>Chama</i> DesLig DesRamificacao DesSeta</p>
	<p><i>Chamado por</i> ChildWndProc (Asia.cpp)</p>
DesRamificacao	Desenha no ponto (x,y) o símbolo que indica que vai uma ramificação para outro recurso.
	<p><i>Chamado por</i> DesenhaLigacoes DesLigacao</p>
DesSeta	Desenha no ponto (x,y) a seta que indica que o recurso é o último do sistema, ou seja, não tem ligação com nenhum outro; ou desenha no mesmo ponto a seta que indica que o recurso é ponto de entrada no sistema.
	<p><i>Chamado por</i> DesenhaLigacoes DesLigacao</p>
EFim	Verifica se o recurso é saída do modelo.
	<p><i>Chamado por</i> Disco (Listarq.cpp) Entrada (Listarq.cpp) Loop (Listarq.cpp) Memória (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) RecursoDigProc (Asia.cpp) Saida (Listarq.cpp)</p>
EInicio	Verifica se o recurso é a entrada do modelo.
	<p><i>Chamado por</i> Disco (Listarq.cpp) Entrada (Listarq.cpp) Loop (Listarq.cpp) Memória (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) RecursoDigProc (Asia.cpp) Saida (Listarq.cpp) WndProc (Asia.cpp)</p>
ExisteLoop	Verifica se o recurso tem loop.
	<p><i>Chamado por</i> AjustaCondicoes (Listarq.cpp) WndProc (Asia.cpp)</p>
InicCondicoesLigacao	Inicializa variáveis necessárias para a captação dos pontos para ligar dois recursos.
	<p><i>Chamado por</i></p>

Módulo	Descrição
	ChildWndProc (Asia.cpp) IconWndProc (Asia.cpp)
InsListaLig	insere item de ligação na lista dinâmica. <i>Chamado por</i> ChildWndProc (Asia.cpp) LerArqLigacoes (Editarq.cpp)
JaTem10Saidas	Verifica se o recurso tem 10 ligações, ramificações ou seta para o <i>Chamado por</i> ChildWndProc (Asia.cpp)
JaTemFim	Verifica se o usuário já definiu ao menos uma saída do modelo, isto é, se já utilizou a seta de fim. <i>Chamado por</i> WndProc (Asia.cpp)
JaTemInicio	Verifica se o usuário já definiu a entrada do modelo, isto é, se já utilizou a seta de início. Essa seta somente pode ser usada uma vez. O ASIA não implementa modelos com várias entradas. <i>Chamado por</i> WndProc (Asia.cpp)
JaTemLigacao	Verifica se o usuário já definiu uma ligação, ramificação entre os recursos rec1 e rec2, ou então seta de início ou fim para um recurso. <i>Chamado por</i> ChildWndProc (Asia.cpp)
LoopEProb	Verifica se o recurso tem loop e mais de uma ramificação. <i>Chamado por</i> AjustaCondicoes (Listarq.cpp) WndProc (Asia.cpp)
NaoPodeSerFim	Verifica se o usuário já definiu uma ligação partindo do recurso rec, ou seja, se rec é o início de ligação para algum recurso do modelo. <i>Chamado por</i> ChildWndProc (Asia.cpp)
NaoPodeSerInicio	Verifica se o usuário já definiu uma ligação para o recurso rec, ou seja, se rec é entrada para a saída de algum outro recurso do modelo. <i>Chamado por</i> ChildWndProc (Asia.cpp)
ObtemNumeroRamificacoes	Contar o número de ramificações que um determinado recurso possui. <i>Chamado por</i> Probabilidade (Listarq.cpp) RecursoDlgProc (Asia.cpp)
ObtemRecursoComRamificacao	Procura recurso ligado a Idrec1, via ramificação. Retoma a identificação do recurso, e busca sempre a partir da i-ésima ramificação, identificada por contador. <i>Chamado por</i> Loop (Listarq.cpp) Probabilidade (Listarq.cpp) RecursoDlgProc (Asia.cpp)
ObtemRecursoLigado	Obtém o recurso ao qual o recurso rec1 está ligado. <i>Chamado por</i> ChildWndProc (Asia.cpp) Disco (Listarq.cpp) Entrada (Listarq.cpp) Loop (Listarq.cpp) Memoria (Listarq.cpp)

Módulo	Descrição
	Pipeline (Lstarq.cpp) Probabilidade (Lstarq.cpp) Processador (Lstarq.cpp) RecursoDigProc (Asia.cpp) Saida (Lstarq.cpp)
PertenceLigacao	Verifica se o usuário clicou o botão esquerdo do mouse sobre a ligação do ponto (x1,y1) ao ponto (x2,y2). <i>Chamado por</i> ApagaUmaLigacao PontoEmLigacao
PertenceRamificacao	Verifica se o usuário clicou o botão esquerdo do mouse sobre a ramificação do ponto (x1,y1) ao ponto (x2,y2). <i>Chamado por</i> ApagaRamificacao
PontoEmLigacao	Verifica se o usuário clicou o botão esquerdo do mouse sobre uma ligação, e em caso afirmativo retorna o ponto correto para desenhar uma ramificação, e o recurso inicial da mesma. <i>Chama</i> PertenceLigacao <i>Chamado por</i> ChildWndProc (Asia.cpp)
RecursoTemRamificacao	Verifica se o recurso tem ramificação para outro.
RemoveUltimoListaLigacoes	Remove o último elemento da lista. <i>Chamado por</i> ChildWndProc (Asia.cpp)
RetornaPosicaoRamificacao	Verifica se o recurso tem ramificação para outro, e retorna a posição inicial dessa ramificação. <i>Chamado por</i> ApagaSimbolo (Lista.cpp)
TemLigacoes	Verifica se existe alguma ligação ou ramificação. <i>Chamado por</i> ChildWndProc (Asia.cpp)
TemMais1Ciclo1Ligacao	Verifica se o recurso tem mais de 1 ciclo, ou ramificação para qq outro. Se tiver, não pode ser escproxrec=1 ciclo. Usada dentro do consist, no VerificaConsistenciaRec.
TodosRecTemLigParaOutro	VerificaConsistenciaRec (Consist.cpp) Verificam, para cada recurso definido no modelo, se tem a sua saída ligada a algum outro, ou para fim. <i>Chamado por</i> WndProc (Asia.cpp)
VerificaLigacao	Verifica se o recurso rec1 tem uma ligação do tipo início ou então uma ligação com outro recurso ou fim e move. <i>Chamado por</i> MoveSimbolo (Lista.cpp)
VerificaRamificacao	Verifica se o recurso tem ramificação para outro, e retorna a posição inicial dessa ramificação e move. <i>Chamado por</i> MoveSimbolo (Lista.cpp)
VerificaSeTemLigacao	Verifica se o recurso rec1 tem uma ligação do tipo início, ou uma ligação com outro recurso, ou fim. <i>Chamado por</i> ApagaSimbolo (Lista.cpp)

Módulo

Listarq.cpp

Descrição

Módulo que contém as funções para a manipulação da lista do Módulo Arquitetura.

Procedimentos

AjustaCondicoes

Função principal que chama as funções auxiliares para setar variáveis dependendo do recurso.

Chama

ApenasUmaSaida (Listalg.cpp)

ExisteLoop (Listalg.cpp)

LoopEProb (Listalg.cpp)

Chamado por

Parametriza

AjustaVarlaveis

Seta as variáveis booleanas globais para determinado tipo de componente.

Chamado por

DesenhaSimbolos (Lista.cpp)

ApagaListaArq

Apaga toda a lista de arquitetura.

Chamado por

ApagaUmRegistro

ChildWndProc (Asia.cpp)

WndProc (Asia.cpp)

ApagaUmRegistro

Apaga um único registro da lista de arquitetura.

Chama

ApagaListaArq

Chamado por

ApagaSimbolo (Lista.cpp)

Disco

Seta as variáveis e insere na lista de recursos um componente disco.

Chama

EFim (Listalg.cpp)

Einicio (Listalg.cpp)

InsListaRec (Listrec.cpp)

InsListaRecChegada (Listrec.cpp)

InsListaRecFila (Listrec.cpp)

InsListaRecRamificacao (Listrec.cpp)

InsListaRecServico (Listrec.cpp)

Loop

MontaStringEntraSaizSemRamificacao

ObtemRecursoLigado (Listalg.cpp)

Probabilidade

Chamado por

Parametriza

Entrada

Seta as variáveis e insere na lista de recursos um componente

Chama

EFim (Listalg.cpp)

Einicio (Listalg.cpp)

InsListaRec (Listrec.cpp)

InsListaRecChegada (Listrec.cpp)

InsListaRecFila (Listrec.cpp)

InsListaRecRamificacao (Listrec.cpp)

InsListaRecServico (Listrec.cpp)

Loop

MontaStringEntraSaizSemRamificacao

ObtemRecursoLigado (Listalg.cpp)

Probabilidade

Chamado por

Módulo	Descrição
InsListaArq	<p>entrada. Parametriza Insere na lista de arquitetura.</p> <p><i>Chamado por</i> ChildWndProc (Asia.cpp) LerArqArquitet (EditArq.cpp)</p>
LimpaTudo	<p>Zera todas as variáveis utilizadas na lista de arquitetura.</p> <p><i>Chamado por</i> Parametriza</p>
Loop	<p>Monta a string EntradaSaida quando o recurso tem loop.</p> <p><i>Chama</i> EFim (Listalg.cpp) Inicio (Listalg.cpp) InsListaLoop (Listloop.cpp) MontaStringEntSai1 (Auxiliar.cpp) ObtemRecursoComRamificacao (Listalg.cpp) ObtemRecursoLigado (Listalg.cpp)</p> <p><i>Chamado por</i> Disco Entrada Memoria Pipeline Processador Saida</p>
Memoria	<p>Seta as variáveis e insere na lista de recursos um componente</p> <p><i>Chama</i> EFim (Listalg.cpp) Inicio (Listalg.cpp) insListaRec (Listrec.cpp) insListaRecChegada (Listrec.cpp) insListaRecFila (Listrec.cpp) insListaRecRamificacao (Listrec.cpp) insListaRecServico (Listrec.cpp) Loop MontaStringEntraSaiSemRamificacao ObtemRecursoLigado (Listalg.cpp) Probabilidade</p> <p><i>Chamado por</i> Parametriza</p>
Parametriza	<p>Percorre a lista de arquitetura e realiza a parametrização automática.</p> <p><i>Chama</i> AjustaCondicoes Disco Entrada LimpaTudo Memoria Pipeline Processador Saida</p> <p><i>Chamado por</i> WndProc (Asia.cpp)</p>
PertenceLista	<p>Verifica se um determinado recurso pertence a lista de arquitetura.</p> <p><i>Chamado por</i> DesenhaSimbolos (Lista.cpp) MoveSimbolo (Lista.cpp)</p>

Módulo	Descrição
Pipeline	<p>Seta as variáveis e insere na lista de recursos um componente</p> <p><i>Chama</i></p> <p>EFim (Listalig.cpp) Einicio (Listalig.cpp) InsListaRec (Listrec.cpp) InsListaRecChegada (Listrec.cpp) InsListaRecFila (Listrec.cpp) InsListaRecRamificacao (Listrec.cpp) InsListaRecServico (Listrec.cpp) Loop MontaStringEntraSaiSemRamificacao ObtemRecursoLigado (Listalig.cpp) Probabilidade</p> <p><i>Chamado por</i></p> <p>Parametriza</p>
Probabilidade	<p>Monta a string EntradaSalda quando o recurso tem ramificação.</p> <p><i>Chama</i></p> <p>MontaStringEntradaSalda (Auxiliar.cpp) ObtemNumeroRamificacoes (Listalig.cpp) ObtemRecursoComRamificacao (Listalig.cpp) ObtemRecursoLigado (Listalig.cpp) TransformaNumero (Auxiliar.cpp) TransformaString (Auxiliar.cpp)</p> <p><i>Chamado por</i></p> <p>Disco Entrada Memoria Pipeline Processador Salda</p>
Processador	<p>Seta as variáveis e insere na lista de recursos um componente processador.</p> <p><i>Chama</i></p> <p>EFim (Listalig.cpp) Einicio (Listalig.cpp) InsListaRec (Listrec.cpp) InsListaRecChegada (Listrec.cpp) InsListaRecFila (Listrec.cpp) InsListaRecRamificacao (Listrec.cpp) InsListaRecServico (Listrec.cpp) Loop MontaStringEntraSaiSemRamificacao ObtemRecursoLigado (Listalig.cpp) Probabilidade</p> <p><i>Chamado por</i></p> <p>Parametriza</p>
Salda	<p>Seta as variáveis e insere na lista de recursos um componente saída.</p> <p><i>Chama</i></p> <p>EFim (Listalig.cpp) Einicio (Listalig.cpp) InsListaRec (Listrec.cpp) InsListaRecChegada (Listrec.cpp) InsListaRecFila (Listrec.cpp) InsListaRecRamificacao (Listrec.cpp) InsListaRecServico (Listrec.cpp) Loop MontaStringEntraSaiSemRamificacao</p>

Módulo	Descrição
	ObtemRecursoLigado (Listalig.cpp) Probabilidade <i>Chamado por</i> Parametriza
Listdist.cpp	Contém as funções que manipulam a lista de dados da distribuição tabelada. Usada para armazenar temporariamente a distribuição em memória, para depois ser armazenada em arquivo.
Procedimentos	
ApagaListaDist	Apaga a lista de distribuição tabelada da memória. <i>Chamado por</i> WndProc (Asia.cpp)
InsListaDist	Insere item de distribuição tabelada na lista dinâmica. <i>Chamado por</i> LerArqDistribuicao (Editarq.cpp) WndProc (Asia.cpp)
Listeven.cpp	Contém as funções que manipulam a lista de dados dos eventos chegada definidos pelo usuário.
Procedimentos	
ApagaListaEventos	Apaga a lista de eventos da memória. <i>Chamado por</i> WndProc (Asia.cpp)
InsListaEventos	Insere item de evento na lista dinâmica. <i>Chamado por</i> LerArqEventos (Editarq.cpp) RecursoDigProc (Asia.cpp)
Listloop.cpp	Contém as funções que manipulam a lista de loops.
Procedimentos	
ApagaListaLoop	Apaga a lista de loop da memória. <i>Chamado por</i> CancelaParLoopAtual WndProc (Asia.cpp)
CancelaParLoopAtual	Cancela a parametrização corrente, removendo os dados do loop do recurso da lista encadeada (se houver). <i>Chama</i> ApagaListaLoop
GuardaDadosLoop	Armazena os dados do ciclo (loop) referente ao recurso na estrutura auxtoioop. <i>Chamado por</i> WndProc (Asia.cpp)
InsListaLoop	Insere item de loop na lista dinâmica. <i>Chamado por</i> LerArqLoop (Editarq.cpp) Loop (Listarq.cpp) RecursoDigProc (Asia.cpp)
TransfereDadoLoopAnterior	Transfere os dados da parametrização que ia ser alterada para
Listrec.cpp	Contém as funções que manipulam a lista de recursos.
Procedimentos	

Módulo	Descrição
ApagaListaRecursos	Apaga a lista de recursos da memória. <i>Chamado por</i> ApagaRecurso CancelaParRecAtual ChildWndProc (Asia.cpp) WndProc (Asia.cpp)
ApagaRecurso	Retira da lista de dados dos recursos os parâmetros fornecidos para o recurso cujo símbolo é Identrec. <i>Chama</i> ApagaListaRecursos
CancelaParRecAtual	Cancela a parametrização corrente, removendo os dados dos recursos da lista encadeada. <i>Chama</i> ApagaListaRecursos
GuardaDadosRecurso	Armazena os dados do recurso na estrutura auxrec. <i>Chamado por</i> WndProc (Asia.cpp)
InsListaRec	Inserir dados do recurso na lista de recurso - nome, nº parâmetros e estatísticas. Caso binsereposatual seja TRUE, o recurso está sendo alterado e, portanto, procura na lista para inserir na posição correta. <i>Chamado por</i> Disco (Listarq.cpp) Entrada (Listarq.cpp) LerArqRecursos (Editarq.cpp) Memoria (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) Saida (Listarq.cpp) WndProc (Asia.cpp)
InsListaRecChegada	Inserir dados do recurso na lista de recurso - chegada. Caso binsereposatual=TRUE, procura na lista a posição correta para inserir. <i>Chamado por</i> Disco (Listarq.cpp) Entrada (Listarq.cpp) LerArqRecursos (Editarq.cpp) Memoria (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) Saida (Listarq.cpp) WndProc (Asia.cpp)
InsListaRecFila	Inserir dados do recurso na lista de recurso - fila. <i>Chamado por</i> Disco (Listarq.cpp) Entrada (Listarq.cpp) LerArqRecursos (Editarq.cpp) Memoria (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) Saida (Listarq.cpp) WndProc (Asia.cpp)
InsListaRecRamificacao	Inserir dados do recurso na lista de recurso - entrada/saída. <i>Chamado por</i> Disco (Listarq.cpp)

Módulo	Descrição
	Entrada (Listarq.cpp) LerArqRecursos (Editarq.cpp) Memória (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) Saída (Listarq.cpp) WndProc (Asia.cpp)
InsListaRecServico	Insere dados do recurso na lista de recursos - serviço. Caso binSereposatual=TRUE, procura na lista a posição correta para inserir. <i>Chamado por</i> Disco (Listarq.cpp) Entrada (Listarq.cpp) LerArqRecursos (Editarq.cpp) Memória (Listarq.cpp) Pipeline (Listarq.cpp) Processador (Listarq.cpp) Saída (Listarq.cpp) WndProc (Asia.cpp)
ProcuraNome	Pega nome do recurso para indentificá-lo na especificação gráfica. <i>Chamado por</i> DesenhaSimbolos (Lista.cpp)
RecursoJaParametrizado	Verifica se o recurso já foi parametrizado ou não. <i>Chamado por</i> ChildWndProc (Asia.cpp) WndProc (Asia.cpp)
TemListaRecursos	Verifica se existe recurso parametrizado. <i>Chamado por</i> ChildWndProc (Asia.cpp)
TransfereDadosAnteriores	Transfere os dados da parametrização que ia ser alterada para <i>Chamado por</i> WndProc (Asia.cpp)
Reesc.c	Gera a tabela de tempos para reescalamento de eventos, para o caso de recursos sem fila.
Procedimentos	
Monta_Tabela_Reescalona	Verificado o número de entradas no arquivo REESCALO.DAT, gera a tabela contendo o tempo para reescalamento de eventos. <i>Chama</i> Copy (Util181.c) Grava_Buffer (Util181.c) Itoa_Def (Util181.c) Verifica_Tabela_Reescalona <i>Chamado por</i> Define_Reesc (Gerador.cpp)
Verifica_Tabela_Reescaiona	Verifica o número de entradas no arquivo TAB.DAT para gerar as estruturas de dados e primitivas necessárias à implementação da distribuição definida pelo usuário. <i>Chamado por</i> Monta_Tabela_Reescaiona
Util181.c	Rotinas utilitárias para o Gerador de Aplicações do ASIA.
Procedimentos	
Abre_Arquivo_Saida	Abre o arquivo saída (implementação do modelo em smpl). Este arquivo é aberto após os demais, pois o nome do arquivo de saída

Módulo**Descrição**

está definido pelo usuário no arquivo MODELO.DAT.

Chama

ConverteExtNomeArq (Auxiliar.cpp)

Chamado por

Gerador (Gerador.cpp)

Atoi_Def

Converte uma string para um número do tipo inteiro.

Chamado por

Gera_Entradas_Saidas (Gerador.cpp)

Auxilia_Gera_Release

Utilitário para a rotina Gera_Release.

Chama

Copy

Gera_Stream (Gerador.cpp)

Grava_Buffer

Itoa_Def

Chamado por

Gera_Release (Gerador.cpp)

Converte

Converte um número do tipo float para uma string.

Chama

Itoa_Def

Copy

Cópia para a string Buffer a string Literal.

Chamado por

Auxilia_Gera_Release

Define_Arq_Saida (Gerador.cpp)

Define_BMeans (Gerador.cpp)

Define_Estat_Fila_Vazia (Gerador.cpp)

Define_Estat_Max_Min (Gerador.cpp)

Define_Tempo_Max (Gerador.cpp)

Define_Var_Backoff (Gerador.cpp)

Define_Var_Reescalona (Gerador.cpp)

Define_Var_Server (Gerador.cpp)

Define_Var_Tempos (Gerador.cpp)

Escalona_Release (Gerador.cpp)

Gera_Break (Gerador.cpp)

Gera_Case (Gerador.cpp)

Gera_Cause (Gerador.cpp)

Gera_Contadores (Gerador.cpp)

Gera_Def_Server (Gerador.cpp)

Gera_Delay (Gerador.cpp)

Gera_Entradas_Saidas (Gerador.cpp)

Gera_Estat_Fila_Vazia (Gerador.cpp)

Gera_Estat_Max_Min (Gerador.cpp)

Gera_Filas_Separadas (Gerador.cpp)

Gera_Loop (Gerador.cpp)

Gera_Metodo_Analise_Saida (Gerador.cpp)

Gera_Primeiro_Evento (Gerador.cpp)

Gera_Proxima_Chegada (Gerador.cpp)

Gera_Relat_Est_Max_Min (Gerador.cpp)

Gera_Relat_Fila_Vazia (Gerador.cpp)

Gera_Relatorio_Final (Gerador.cpp)

Gera_Release (Gerador.cpp)

Gera_Stream (Gerador.cpp)

Gera_Switch (Gerador.cpp)

Le_Gabarito (Gerador.cpp)

Monta_Tabela_Reescalona (Reesc.c)

Nome_Parametro (Gerador.cpp)

Módulo	Descrição
Copy_SubString	Warmup (Gerador.cpp) Copia subcadeia a partir de uma cadeia, sendo especificados o tamanho e a posição inicial.
Erro	<i>Chamado por</i> Gera_Entradas_Saldas (Gerador.cpp) Retorna as mensagens de erro correspondentes.
Fecha_arquivos	<i>Chama</i> MensagemErro Fechar os arquivos utilizados pelo Gerador.
Grava_Buffer	<i>Chamado por</i> Gerador (Gerador.cpp) Grava o conteúdo da variável Buffer no arquivo de saída do gerador (implementação em smpl).
	<i>Chamado por</i> Auxilia_Gera_Release Define_Arq_Salda (Gerador.cpp) Define_BMeans (Gerador.cpp) Define_Estat_Fila_Vazia (Gerador.cpp) Define_Estat_Max_Min (Gerador.cpp) Define_Tempo_Max (Gerador.cpp) Define_Var_Backoff (Gerador.cpp) Define_Var_Reescalona (Gerador.cpp) Define_Var_Server (Gerador.cpp) Define_Var_Tempos (Gerador.cpp) Distrib_Tabela (Dist_tab.c) Escalona_Release (Gerador.cpp) Gera_Break (Gerador.cpp) Gera_Case (Gerador.cpp) Gera_Cause (Gerador.cpp) Gera_Contadores (Gerador.cpp) Gera_Def_Server (Gerador.cpp) Gera_Delay (Gerador.cpp) Gera_Entradas_Saldas (Gerador.cpp) Gera_Estat_Fila_Vazia (Gerador.cpp) Gera_Estat_Max_Min (Gerador.cpp) Gera_Filas_Separadas (Gerador.cpp) Gera_Loop (Gerador.cpp) Gera_Metodo_Analise_Salda (Gerador.cpp) Gera_Primeiro_Evento (Gerador.cpp) Gera_Proxima_Chegada (Gerador.cpp) Gera_Relat_Est_Max_Min (Gerador.cpp) Gera_Relat_Fila_Vazia (Gerador.cpp) Gera_Relatorio_Final (Gerador.cpp) Gera_Release (Gerador.cpp) Gera_Stream (Gerador.cpp) Gera_Switch (Gerador.cpp) Monta_Tabela_Reescalona (Reesc.c) Nome_Parametro (Gerador.cpp) Warmup (Gerador.cpp)
Ittoa_Def	Converte um número do tipo inteiro sem sinal para string, porém na ordem inversa. <i>Chama</i> Reverte <i>Chamado por</i> Auxilia_Gera_Release Converte

<i>Módulo</i>	<i>Descrição</i>
	Define_Var_Backoff (Gerador.cpp) Escalona_Release (Gerador.cpp) Gera_Contadores (Gerador.cpp) Gera_Delay (Gerador.cpp) Gera_Entradas_Saldas (Gerador.cpp) Gera_Filas_Separadas (Gerador.cpp) Gera_Primeiro_Evento (Gerador.cpp) Gera_Proxima_Chegada (Gerador.cpp) Gera_Release (Gerador.cpp) Monta_Tabela_Reescalona (Reesc.c)
Limpa_Buffer	Limpa o buffer, para evitar sujeiras no arquivo de saída do gerador.
MensagemErro	Auxilia o sistema de mensagens de erros. <i>Chamado por</i> Erro
Reverte	Reverte uma cadeia de caracteres. <i>Chamado por</i> Itca_Def
Strcat_Def	Concatena um caracter em uma string.

A.2. Procedimentos do ASiA (relatório alfabético)

<i>Procedimento</i>	<i>Módulo</i>
Abre_Arquivo_Saida	Util181.c
AjustaCondições	Listarq.cpp
AjustaVariáveis	Listarq.cpp
ApagaLigacao	Listalig.cpp
ApagaLista	Lista.cpp
ApagaListaArq	Listarq.cpp
ApagaListaDist	Listdist.cpp
ApagaListaEventos	Listeven.cpp
ApagaListaLigacoes	Listalig.cpp
ApagaListaLoop	Listloop.cpp
ApagaListaRecursos	Listrec.cpp
ApagaRamificacao	Listalig.cpp
ApagaRecurso	Listrec.cpp
ApagaSimbolo	Lista.cpp
ApagaUmaLigacao	Listalig.cpp
ApagaUmRegistro	Listarq.cpp
ApenasUmaSaida	Listalig.cpp
AskAboutSave	Asia.cpp
Atoi_Def	Util181.c
AtualizaNumFilaServidor	Auxiliar.cpp
AuxAbrirArqLigacoes	EditArq.cpp
AuxAbrirArqLoop	EditArq.cpp
Auxilia_Gera_Release	Util181.c
AuxSalvarArqLigacoes	EditArq.cpp
AuxSalvarArqLoop	EditArq.cpp
BuscaRamificacaoQueChega	Listalig.cpp
CancelaParLoopAtual	Listloop.cpp
CancelaParRecAtual	Listrec.cpp
ChegadaDigProc	Asia.cpp
ChildWndProc	Asia.cpp
ClicRecurso	Lista.cpp
Convert	Dist_tab.c
Converte	Util181.c
ConverteExtNomeArq	Auxiliar.cpp
Cop	Dist_tab.c
Copy	Util181.c
Copy_SubString	Util181.c
Define_Arq_Saida	Gerador.cpp
Define_BMeans	Gerador.cpp
Define_Estat_Fila_Vazia	Gerador.cpp
Define_Estat_Max_Min	Gerador.cpp
Define_Reesc	Gerador.cpp
Define_Tempo_Max	Gerador.cpp
Define_Var_Backoff	Gerador.cpp
Define_Var_Reescalona	Gerador.cpp
Define_Var_Server	Gerador.cpp
Define_Var_Tempos	Gerador.cpp

<i>Procedimento</i>	<i>Módulo</i>
DesBlocoDisco	Lista.cpp
DesBlocoMem	Lista.cpp
DesBlocoProc	Lista.cpp
DesenhaLigacoes	Listalig.cpp
DesenhaRec	Lista.cpp
DesenhaRec2	Lista.cpp
DesenhaSimbolos	Lista.cpp
DesFila	Lista.cpp
DesInicCondicoesLigacao	Listalig.cpp
DesLig	Listalig.cpp
DesLigacao	Listalig.cpp
DesLinha	Lista.cpp
DesPipeline	Lista.cpp
DesPonto	Lista.cpp
DesRamificacao	Listalig.cpp
DesRec1	Lista.cpp
DesRec2	Lista.cpp
DesRec3	Lista.cpp
DesRec4	Lista.cpp
DesServ	Lista.cpp
DesSeta	Listalig.cpp
DeterminaIcone	Desenha.cpp
Disco	Listarq.cpp
Distrib_Tabela	Dist_tab.c
DistribuicaoDigProc	Asia.cpp
DigAbrirArqDistribuicao	Editrarq.cpp
DigAbrirArqEspecGrafica	Editrarq.cpp
DigSalvarArqDistribuicao	Editrarq.cpp
DigSalvarArqEspecGrafica	Editrarq.cpp
DoCaption	Asia.cpp
DrawiconBar	Desenha.cpp
DrawiconShow	Desenha.cpp
EFim	Listalig.cpp
Einicio	Listalig.cpp
Entrada	Listarq.cpp
Erro	Util181.c
Escalona_Release	Gerador.cpp
EventosDigProc	Asia.cpp
ExisteLoop	Listalig.cpp
Fecha_arquivos	Util181.c
FilaDigProc	Asia.cpp
Gera_Break	Gerador.cpp
Gera_Case	Gerador.cpp
Gera_Cause	Gerador.cpp
Gera_Contadores	Gerador.cpp
Gera_Def_Server	Gerador.cpp
Gera_Delay	Gerador.cpp
Gera_Entradas_Saidas	Gerador.cpp
Gera_Estat_Fila_Vazia	Gerador.cpp

Procedimento**Módulo**

Gera_Estat_Max_Min	Gerador.cpp
Gera_Eventos	Gerador.cpp
Gera_Filas_Separadas	Gerador.cpp
Gera_Loop	Gerador.cpp
Gera_Metodo_Analise_Saida	Gerador.cpp
Gera_Primeiro_Evento	Gerador.cpp
Gera_Proxima_Chegada	Gerador.cpp
Gera_Relat_Est_Max_Min	Gerador.cpp
Gera_Relat_Fila_Vazia	Gerador.cpp
Gera_Relatorio_Final	Gerador.cpp
Gera_Release	Gerador.cpp
Gera_Stream	Gerador.cpp
Gera_Switch	Gerador.cpp
Gerador	Gerador.cpp
Grava_Buf	Dist_tab.c
Grava_Buffer	Util181.c
GravarArqArquitet	Editarg.cpp
GravarArqDistribuicao	Editarg.cpp
GravarArqEspecGrafica	Editarg.cpp
GravarArqEventos	Editarg.cpp
GravarArqLigacoes	Editarg.cpp
GravarArqLoop	Editarg.cpp
GravarArqModelo	Editarg.cpp
GravarArqRecursos	Editarg.cpp
GuardaDadosLoop	Listloop.cpp
GuardaDadosRecurso	Listrec.cpp
IconWndProc	Asia.cpp
IdentRecSimbolo	Lista.cpp
InicCondicoesLigacao	Listalig.cpp
InicializaArqDistribuicao	Editarg.cpp
InicializaArqEspecGrafica	Editarg.cpp
InicializaDadosDistribuicao	Auxiliar.cpp
InicializaDadosEventos	Auxiliar.cpp
InicializaDadosLoop	Auxiliar.cpp
InicializaDadosModelo	Auxiliar.cpp
InicializaDadosRecursos	Auxiliar.cpp
InicializaNumFilaServidor	Auxiliar.cpp
InsererLista	Lista.cpp
InsListaArq	Listarg.cpp
InsListaDist	Listdist.cpp
InsListaEventos	Listeven.cpp
InsListaLig	Listalig.cpp
InsListaLoop	Listloop.cpp
InsListaRec	Listrec.cpp
InsListaRecChegada	Listrec.cpp
InsListaRecFila	Listrec.cpp
InsListaRecRamificacao	Listrec.cpp
InsListaRecServico	Listrec.cpp
ltoa_Def	Util181.c

Procedimento**Módulo**

Itoa_Defi	Dist_tab.c
JaTem10Saidas	Listalig.cpp
JaTemFim	Listalig.cpp
JaTemInicio	Listalig.cpp
JaTemLigacao	Listalig.cpp
Le_Gabarito	Gerador.cpp
LerArqArquitet	EditArq.cpp
LerArqDistribuicao	EditArq.cpp
LerArqEspecGrafica	EditArq.cpp
LerArqEventos	EditArq.cpp
LerArqLigacoes	EditArq.cpp
LerArqLoop	EditArq.cpp
LerArqModelo	EditArq.cpp
LerArqRecursos	EditArq.cpp
LigacaoPrincipalDigProc	Asia.cpp
Limpa_Buf	Dist_tab.c
Limpa_Buffer	Util181.c
LimpaTudo	Listarq.cpp
Loop	Listarq.cpp
LoopDigProc	Asia.cpp
LoopEProb	Listalig.cpp
Memoria	Listarq.cpp
MensagemErro	Util181.c
ModeloDigProc	Asia.cpp
Monta_Array_Acumulado	Dist_tab.c
Monta_Tabela_Reescalona	Reesc.c
MontaStringEntradaSalda	Auxiliar.cpp
MontaStringEntraSaiSemRamificacao	Auxiliar.cpp
MontaStringEntSai1	Auxiliar.cpp
MoveSimbolo	Lista.cpp
NaoPodeSerFim	Listalig.cpp
NaoPodeSerInicio	Listalig.cpp
Nome_Parametro	Gerador.cpp
NumBlocoDigProc	Asia.cpp
ObtemNumeroRamificacoes	Listalig.cpp
ObtemRecursoComRamificacao	Listalig.cpp
ObtemRecursoLigado	Listalig.cpp
OkMessage	Asia.cpp
Parametriza	Listarq.cpp
ParametrizouChegada	Inforec.cpp
ParametrizouFila	Inforec.cpp
ParametrizouServico	Inforec.cpp
PertenceLigacao	Listalig.cpp
PertenceLista	Listarq.cpp
PertenceRamificacao	Listalig.cpp
Pipeline	Listarq.cpp
PontoEmLigacao	Listalig.cpp
Prepara_Estruturas	Gerador.cpp
Probabilidade	Listarq.cpp
Processador	Listarq.cpp

Procedimento	Módulo
ProcuraNome	Listrec.cpp
PtoFinal	Lista.cpp
PtoInicial	Lista.cpp
RamificacaoDigProc	Asia.cpp
RecursoDigProc	Asia.cpp
RecursoJaParametrizado	Listrec.cpp
RecursoTemRamificacao	Listalig.cpp
RemoveUltimoListaLigacoes	Listalig.cpp
RemoveUltimoSimboloLista	Lista.cpp
RetomaPosicaoRamificacao	Listalig.cpp
Reverte	Util181.c
Reverte_String	Dist_tab.c
Saida	Listarq.cpp
ServicoDigProc	Asia.cpp
Strcat_Def	Util181.c
TemLigacoes	Listalig.cpp
TemListaRecursos	Listrec.cpp
TemMais1Ciclo1Ligacao	Listalig.cpp
TemQueParametrizarRecurso	inforec.cpp
TemSimbolo	Lista.cpp
TipoArquiteturaDigProc	Asia.cpp
TiraCaminho	Auxiliar.cpp
TodosRecTemLigParaOutro	Listalig.cpp
TransfereDadoLoopAnterior	Listloop.cpp
TransfereDadosAnteriores	Listrec.cpp
TransformaNumero	Auxiliar.cpp
TransformaString	Auxiliar.cpp
Verifica_Tabela	Dist_tab.c
Verifica_Tabela_Reescalona	Reesc.c
VerificaConsistenciaDistribuicao	Consist.cpp
VerificaConsistenciaEventos	Consist.cpp
VerificaConsistenciaLoop	Consist.cpp
VerificaConsistenciaModelo	Consist.cpp
VerificaConsistenciaRec	Consist.cpp
VerificaConsistenciaRecChegada	Consist.cpp
VerificaConsistenciaRecFila	Consist.cpp
VerificaConsistenciaRecLigacaoPrincipal	Consist.cpp
VerificaConsistenciaRecRamificacao	Consist.cpp
VerificaConsistenciaRecServico	Consist.cpp
VerificaIgual	Gerador.cpp
VerificaInteiro	Consist.cpp
VerificaLigacao	Listalig.cpp
VerificaProb	Gerador.cpp
VerificaRamificacao	Listalig.cpp
VerificaReal	Consist.cpp
VerificaSeTemLigacao	Listalig.cpp
VerificaTotal	Consist.cpp
Warmup	Gerador.cpp
WinMain	Asia.cpp
WndProc	Asia.cpp

A.3. Diagrama de Temas

A.4. Relatório de estruturas e elementos de dados

<i>Estrutura</i>	<i>Elemento</i>
auxarquitet	ident_rec num_bloco tipo
auxdisttab	prob valor
auxlig	recurso1 recurso2 tipo x1 x2 y1 y2
auxrec	desvio_padrao_chegada desvio_padrao_servico disciplina distribuicao_chegada distribuicao_servico entradas_saidas escolha_cliente escolha_fila escolha_prox_rec escolha_servidor est_fila_vazia est_max_min flag_slnc ident_rec libera_recurso media_chegada media_servico nome_distr_chegada nome_distr_servico nome_recurso num_filas num_servidores pto_triang_chegada pto_triang_servico sem_fila semente_chegada semente_servico
auxsimb	identrec tipo x y
auxspecifi	distrib_tab modelo_fechado nome_modelo num_max_entidades num_voltas reesc relatorio_final

Estrutura	Elemento
	tamanho_batch
	tempo
	tempo_prim_chegada
	valores_desprezados
	warm_up
auxtempeventos	num_recurso
	tempo_ocorrendia
auxtotloop	n_joop
	n_rec
lista_arquitet	ident_rec
	num_bloco
	prox
	tipo
lista_dist_tab	prob
	prox
	valor
lista_ligacoes	prox
	recurso1
	recurso2
	tipo
	x1
	x2
	y1
	y2
lista_recursos	desvio_padrao_chegada
	desvio_padrao_servico
	disciplina
	distribuicao_chegada
	distribuicao_servico
	entradas_saidas
	escolha_cliente
	escolha_fila
	escolha_prox_rec
	escolha_servidor
	est_fila_vazia
	est_max_min
	flag_sinc
	ident_rec
	libera_recurso
	media_chegada
	media_servico
	nome_distr_chegada
	nome_distr_servico
	nome_recurso
	num_filas
	num_servidores
	prox
	pto_triag_chegada
	pto_triag_servico
	sem_fila
	semente_chegada
	semente_servico

<i>Estrutura</i>	<i>Elemento</i>
lista_simbolos	identrec
	prox
	tipo
	x
	y
lista_temp_eventos	num_recurso
	prox
	tempo_ocorrenda
lista_tot_loop	n_loop
	n_rec
	prox

A.5. Relatório de classificação dos procedimentos por pseudo-classes

Pseudo-classe: auxarquitet		
Procedimentos:		
GravarArqArquitet	[c/Editarq.cpp]	oc
LerArqArquitet	[o/Editarq.cpp]	o

Pseudo-classe: auxdisttab		
Procedimentos:		
GravarArqDistribuicao	[c/Editarq.cpp]	oc
LerArqDistribuicao	[o/Editarq.cpp]	o+

Pseudo-classe: auxlig		
Procedimentos:		
GravarArqLigacoes	[c/Editarq.cpp]	c
LerArqLigacoes	[o/Editarq.cpp]	o

Pseudo-classe: auxrec		
Procedimentos:		
GravarArqRecursos	[c/Editarq.cpp]	oc
LerArqRecursos	[o/Editarq.cpp]	o

Pseudo-classe: auxsimb		
Procedimentos:		
GravarArqEspecGrafica	[c/Editarq.cpp]	oc+
LerArqEspecGrafica	[o/Editarq.cpp]	o

Pseudo-classe: auxspecifi		
Procedimentos:		
GravarArqModelo	[c/Editarq.cpp]	c
LerArqModelo	[o/Editarq.cpp]	o

Pseudo-classe: auxtempeventos		
Procedimentos:		
GravarArqEventos	[c/Editarq.cpp]	oc
LerArqDistribuicao	[o/Editarq.cpp]	o+
LerArqEventos	[c/Editarq.cpp]	o

Pseudo-classe: auxtotloop		
Procedimentos:		
GravarArqLoop	[c/Editarq.cpp]	oc
GuardaDadosLoop	[c/Listloop.cpp]	oc
LerArqLoop	[o/Editarq.cpp]	o
TransfereDadoLoopAnterior	[o/Listloop.cpp]	o

Pseudo-classe: lista_arquitet		
Procedimentos:		
ApagaListaArq	[o/Listarq.cpp]	o
ApagaUmRegistro	[c/Listarq.cpp]	c
GravarArqArquitet	[o/Editarq.cpp]	oc
InsListaArq	[c/Listarq.cpp]	c
Parametriza	[o/Listarq.cpp]	o
PertenceLista	[o/Listarq.cpp]	o

Pseudo-classe: lista_dist_tab		
Procedimentos:		
ApagaListaDist	[o/Listdist.cpp]	o
GravarArqDistribuicao	[o/Editarq.cpp]	oc
InsListaDist	[c/Listdist.cpp]	c

Pseudo-classe: lista_ligacoes		
Procedimentos:		
ApagaLigacao	[c/Listalig.cpp]	c
ApagaListaLigacoes	[o/Listalig.cpp]	o
ApagaRamificacao	[c/Listalig.cpp]	c
ApagaUmaLigacao	[c/Listalig.cpp]	c
ApenasUmaSaida	[o/Listalig.cpp]	o
BuscaRamificacaoQueChega	[o/Listalig.cpp]	o
DesenhaLigacoes	[o/Listalig.cpp]	o
DesLigacao	[o/Listalig.cpp]	o
EFim	[o/Listalig.cpp]	o
Einicio	[o/Listalig.cpp]	o
ExisteLoop	[o/Listalig.cpp]	o
GravarArqEspecGrafica	[c/Editarq.cpp]	oc+
InsListaLig	[c/Listalig.cpp]	c
JaTem10Saidas	[o/Listalig.cpp]	o
JaTemFim	[o/Listalig.cpp]	o
JaTemInicio	[o/Listalig.cpp]	o
JaTemLigacao	[o/Listalig.cpp]	o
LoopEProb	[o/Listalig.cpp]	o
NaoPodeSerFim	[o/Listalig.cpp]	o
NaoPodeSerInicio	[o/Listalig.cpp]	o
ObtemNumeroRamificacoes	[o/Listalig.cpp]	o
ObtemRecursoComRamificacao	[o/Listalig.cpp]	o
ObtemRecursoLigado	[o/Listalig.cpp]	o
PontoEmLigacao	[o/Listalig.cpp]	o
RecursoTemRamificacao	[o/Listalig.cpp]	o
RemoveUltimoListaLigacoes	[c/Listalig.cpp]	c
RetomaPosicaoRamificacao	[o/Listalig.cpp]	o
TemMais1Ciclo1Ligacao	[o/Listalig.cpp]	o
TodosRecTemLigParaOutro	[o/Listalig.cpp]	o+
VerificaLigacao	[c/Listalig.cpp]	c
VerificaRamificacao	[c/Listalig.cpp]	c
VerificaSeTemLigacao	[o/Listalig.cpp]	o

Pseudo-classe: lista_recursos		
Procedimentos:		
ApagaListaRecursos	[c/Listrec.cpp]	c
ApagaRecurso	[c/Listrec.cpp]	c
CancelaParRecAtual	[c/Listrec.cpp]	c+
GravarArqRecursos	[o/Editarq.cpp]	oc
GuardaDadosRecurso	[o/Listrec.cpp]	o
InsListaRec	[c/Listrec.cpp]	c
InsListaRecChegada	[c/Listrec.cpp]	c
InsListaRecFila	[c/Listrec.cpp]	c
InsListaRecRamificacao	[c/Listrec.cpp]	c
InsListaRecServico	[c/Listrec.cpp]	c
ProcuraNome	[o/Listrec.cpp]	o
RecursoJaParametrizado	[c/Listrec.cpp]	c

Pseudo-classe: lista_simbolos		
Procedimentos:		
ApagaLista	[o/Lista.cpp]	o
ApagaSimbolo	[o/Lista.cpp]	o
DesenhaRec	[o/Lista.cpp]	o
DesenhaRec2	[o/Lista.cpp]	o
DesenhaSimbolos	[o/Lista.cpp]	o
GravarArqEspecGrafica	[o/Editarq.cpp]	oc+
IdentRecSimbolo	[o/Lista.cpp]	o
InsererLista	[c/Lista.cpp]	c
MoveSimbolo	[c/Lista.cpp]	c
PtoFinal	[o/Lista.cpp]	o
PtoInicial	[o/Lista.cpp]	o
RemoveUltimoSimboloLista	[c/Lista.cpp]	c
TodosRecTemLigParaOutro	[o/Listalig.cpp]	o+

Pseudo-classe: lista_temp_eventos		
Procedimentos:		
ApagaListaEventos	[o/Listeven.cpp]	o
GravarArqEventos	[o/Editarq.cpp]	oc
InsListaEventos	[c/Listeven.cpp]	c

Pseudo-classe: lista_tot_loop		
Procedimentos:		
ApagaListaLoop	[o/Listloop.cpp]	o
CancelaParRecAtual	[c/Listrec.cpp]	c+
GravarArqLoop	[o/Editarq.cpp]	oc
GuardaDadosLoop	[o/Listloop.cpp]	oc
InsListaLoop	[c/Listloop.cpp]	c

A.6. Modelo de Objetos do MASA (Tema temporário)

A.7. Modelo de Objetos do MASA (Tema permanente)

A.8. Modelo de Ciclo de Vida do MASA

life cycle ASiA : (Arquivo | Módulo Arquitetura | ? | Barra de ferramentas)+ . (Arquivo | Parametrizar | Gerador | Visualizar | Módulo Arquitetura | ? | Barra de ferramentas)*

Arquivo = ((Novo . (Salvar Como . #Nome_arquivo)) . (Novo | Abrir . #Nome_arquivo | Salvar | Salvar Como . #Nome_arquivo | Sair)*) | Sair

Parametrizar = (Modelo . #Janela_parametrização_modelo | Recurso)

Recurso = #Janela_parametrização_recurso . [(Ligação_Principal . Ramificação) . Loop] . (Chegada . Fila . Serviço . Eventos)* . (Ok | Cancel | Help*)

Chegada = #Janela_parametrizar_chegada . [Gerar . #Janela_Distribuição . (Ok | Cancel | Fim | Help*) | Ler . #Janela_abrir_distribuição]

Fila = #Janela_parametrizar_fila . (Ok | Cancel | Help*)

Serviço = #Janela_parametrizar_serviço . [Gerar . #Janela_Distribuição . (Ok | Cancel | Fim | Help*) | Ler . #Janela_abrir_distribuição]

Eventos = #Janela_parametrizar_eventos . (Ok | Cancel | Help*)

Ligação_Principal = #Janela_ligação_principal . (Ok | Cancel | Help*)

Ramificação = #Janela_ramificação . (Ok | Cancel | Help*)

Loop = #Janela_loop . (Ok | Cancel | Help*)

Gerador = (Gerar_o_código . #Janela_de_confirmação_de_geração_de_código | Gerar_o_executável . #Janela_de_confirmação_de_geração_do_executável)

Visualizar = (Código . #Código_Gerado | Animação | Relatórios . #Resultados_da_simulação)

Animação = (NULO)

Módulo Arquitetura = (Ativar . #Janela_Tipos . ((von_Neumann | von_Neumann_com_pipeline | Multiprocessador_com_memória_compartilhada | Multiprocessador_com_memória_distribuída) . (Ok | Cancel | Help*)) | Personalizar . #Janela_Personalizar_Barra_de_ferramentas)

Ativar = (Arquivo | Módulo Arquitetura | ? | Barra de ferramentas)+ . (Arquivo | Parametrizar | Gerador | Visualizar | Módulo Arquitetura | ? | Barra de ferramentas)*

Parametrizar = (Tudo | Modelo . #Janela_parametrização_modelo | Recurso)

? = (Ajuda | Sobre_o_ASiA)

Ajuda = (NULO)

Barra_de_ferramentas = ((Inserir_um_recurso_com_uma_fila_e_um_servidor | Inserir_um_serviço_com_uma_fila_e_vários_servidores | Inserir_um_serviço_com_várias_filas_e_um_servidor | Inserir_um_serviço_com_várias_filas_e_vários_servidores | Inserir_uma_ligação | Inserir_uma_ramificação)+ . Inserir_uma_sete_de_início . Inserir_uma_sete_de_fim . Selecionar_um_recurso . (Apagar_símbolos_e_ligações | Mover_recursos))*

Barra_de_ferramentas_módulo_arquitetura = ((Inserir_um_processador | Inserir_uma_memória | Inserir_um_disco | Inserir_uma_entrada | Inserir_um_bloco_pipeline | Inserir_um_bloco_de_processadores | Inserir_um_bloco_de_memórias | Inserir_um_bloco_de_discos | Inserir_uma_saída | Inserir_uma_ligação | Inserir_uma_ramificação)+ . Inserir_uma_sete_de_início . Inserir_uma_sete_de_fim . Selecionar_um_recurso . (Apagar_símbolos_e_ligações | Mover_recursos))*

A.9. Modelo de Operações do MASA

A Figura A.1 ilustra a tela do ASiA para a parametrização de um recurso, sendo o Modelo de Operações equivalente apresentado na Figura A.2. Essas figuras foram adicionadas para exemplificar a construção do Modelo de Operações a partir da interface do sistema.

Figura A.1: Tela do ASiA para a parametrização de um recurso

Operação:	Recurso
Descrição:	Abre a janela para fornecimento dos valores dos parâmetros do modelo que está sendo editado se o modelo estiver completo (possuir entrada e saída).
Lê:	<i>supplied</i> nome, número de servidores, número de filas, estatística para fila vazia, estatística para comprimento máximo e mínimo das filas, Chegada, Fila, Serviço, Eventos
Modifica:	Parâmetros do recurso lidos
Envia:	Usuário: {valor_inválido}; mensagens de erro
Assume:	Assume que o modelo está completo (possui ligação de entrada e de saída)
Resultado:	Recurso parametrizado

Figura A.2: Modelo da Operação Parametrizar recurso do MASA do ASiA

A seguir, são apresentados os modelos para as demais operações do ASiA.

Operação:	Novo
Descrição:	Habilita ambiente para criação de um novo modelo. Se um modelo estiver sendo editado e não estiver salvo, pergunta se salva o modelo ativo. As variáveis são reiniciadas.
Lê:	Nome do arquivo, diretório
Modifica:	Variáveis do ambiente, listas de edição, parâmetros
Envia:	Usuário: (pergunta se salva); usuário: {janela salvar}; usuário: {editor inicializado}; mensagens de erro
Assume:	
Resultado:	Ambiente inicializado para a edição de um novo modelo

Operação:	Salvar como
Descrição:	Salva um modelo novo (especificação e parâmetros) ainda sem nome ou um modelo que está sendo editado com outro nome.
Lê:	<i>supplied</i> diretório, nome do arquivo
Modifica:	<i>new</i> arquivos com extensão .esp, .lig, .lop, .eve, .mod, .rec
Envia:	Usuário: {janela salvar}; mensagens de erro
Assume:	
Resultado:	Arquivos de especificação do modelo gravados com o nome fornecido pelo usuário

Operação:	Salvar
Descrição:	Grava o modelo em edição (especificação e parâmetros) quando já tem nome
Lê:	
Modifica:	Arquivos .esp, .lig, .lop, .eve do modelo
Envia:	Usuário: mensagens de erro
Assume:	nome do arquivo em edição
Resultado:	Arquivos de especificação do modelo gravados

Operação:	Abrir
Descrição:	Abre um modelo com todos os arquivos que o compõem. Se um modelo estiver sendo editado e não estiver salvo, pergunta se salva o modelo ativo.
Lê:	<i>supplied</i> diretório, nome do arquivo (.esp)
Modifica:	Modelo em edição
Envia:	Usuário: {janela_escolhe_arquivo}. mensagens de erro
Assume:	
Resultado:	Modelo aberto para edição

Operação:	Sair
Descrição:	Fecha o ASIA. Se o modelo que está sendo editado não estiver salvo, pergunta se salva.
Lê:	
Modifica:	Arquivos do modelo
Envia:	Usuário: {confirma_saida}; mensagens de erro
Assume:	
Resultado:	Arquivos de especificação salvos Ambiente fechado

Operação:	Modelo
Descrição:	Abre a janela para fornecimento dos valores dos parâmetros do modelo que está sendo editado se o modelo estiver completo (possuir entrada e saída).
Lê:	<i>supplied</i> nome, tempo de execução, tempo de warmup, primeira chegada, número de distribuições, número máximo de entidades, número de ciclos, tamanho batch, valores desprezados, relatório final
Modifica:	Parâmetros do modelo lidos
Envia:	Usuário: {valor_inválido}; mensagens de erro
Assume:	
Resultado:	Modelo parametrizado

Operação:	Gerar
Descrição:	Permite que o usuário produza uma distribuição estatística para ser usada na simulação do modelo
Lê:	<i>supplied</i> valor, número de ocorrências <i>supplied</i> diretório, nome do arquivo (.dis)
Modifica:	<i>new</i> arquivo .dis
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Nova distribuição do usuário definida

Operação:	Ler
Descrição:	Lê uma distribuição estatística criada pelo usuário
Lê:	, <i>supplied</i> diretório, nome do arquivo .dis
Modifica:	
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Utiliza uma distribuição criada pelo usuário na simulação do modelo

Operação:	Ok
Descrição:	Fecha a janela ativa e atualiza as variáveis
Lê:	Dados da janela ativa
Modifica:	Variáveis do ambiente
Envia:	Usuário: mensagens de erro
Assume:	Todos as informações estão completos
Resultado:	Função da janela é executada

Operação:	Cancel
Descrição:	Fecha a janela ativa e cancela as alterações que possam ter sido feitas
Lê:	
Modifica:	
Envia:	
Assume:	
Resultado:	Janela é fechada e alterações são ignoradas

Operação:	Fim
Descrição:	Finaliza a inserção de valores para uma distribuição gerada pelo usuário
Lê:	<i>supplied</i> diretório, nome do arquivo
Modifica:	<i>new</i> arquivo .dis
Envia:	Usuário: mensagens de erro, janela para gravar distribuição do usuário (arquivo .dis)
Assume:	Que todos os valores para a distribuição foram fornecidos
Resultado:	Envia valores da distribuição do usuário para o arquivo de distribuição

Operação:	Help
Descrição:	(Opção sem funcionalidade)
Lê:	
Modifica:	
Envia:	
Assume:	
Resultado:	

Operação:	Recurso
Descrição:	Abre a janela para fornecimento dos valores dos parâmetros do modelo que está sendo editado se o modelo estiver completo (possuir entrada e saída).
Lê:	<i>supplied</i> nome, número de servidores, número de filas, estatística para fila vazia, estatística para comprimento máximo e mínimo das filas, Chegada, Fila, Serviço, Eventos
Modifica:	Parâmetros do recurso lidos
Envia:	Usuário: {valor_inálido}; mensagens de erro
Assume:	Assume que o modelo está completo (possui ligação de entrada e de saída)
Resultado:	Recurso parametrizado

Operação:	Chegada
Descrição:	Permite o fornecimento de parâmetros para a chegada de clientes ao recurso
Lê:	<i>supplied</i> Distribuição, Média, Desvio Padrão, Seqüência, Ponto mais provável
Modifica:	
Envia:	usuários: mensagens de erro
Assume:	
Resultado:	informações para a chegada de clientes no recurso fornecidas

Operação:	Fila
Descrição:	Permite o fornecimento de parâmetros para a fila do recurso
Lê:	<i>supplied</i> Disciplina, Escolha Fila, Escolha Cliente, Escolha Servidor
Modifica:	variáveis do ambiente
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Dados para a fila do recurso fornecidos

Operação:	Serviço
Descrição:	Permite a inclusão de parâmetros para o serviço fornecido pelo recurso
Lê:	<i>supplied</i> Distribuição, Média, Desvio Padrão, Seqüência, Ponto mais provável
Modifica:	variáveis do ambiente
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Serviço do recurso parametrizado

Operação:	Eventos
Descrição:	Permite o cadastro de eventos (em quantidade indefinida) para os modelos fechados
Lê:	<i>supplied</i> Tempo de ocorrência
Modifica:	arquivo de eventos (.eve)
Envia:	
Assume:	Que o modelo é fechado
Resultado:	Tempos de eventos cadastrados

Operação:	Ligação principal
Descrição:	Insere probabilidade para escolha da ligação principal caso haja ramificação
Lê:	<i>supplied</i> Ligação principal
Modifica:	variáveis do ambiente
Envia:	Usuário: mensagem de erro
Assume:	Que há ramificação e não há ciclo
Resultado:	Probabilidade do cliente seguir a ligação principal para outro recurso é incluída

Operação:	Ramificação
Descrição:	Permite a Inclusão de probabilidade para o cliente seguir a ramificação indicada
Lê:	<i>supplied</i> Ramificação
Modifica:	variáveis do ambiente
Envia:	Usuário: mensagem de erro
Assume:	Que há ramificação Ramificação com valor igual a diferença entre a probabilidade para a ligação principal e 100% (no caso de apenas uma ramificação) ou a soma das probabilidades das demais ramificações (no caso de várias ramificações)
Resultado:	Probabilidade para utilização da ramificação incluída

Operação:	Loop
Descrição:	Permite o fornecimento do número de loops para recursos com ciclo e ramificação
Lê:	<i>supplied</i> Loop
Modifica:	
Envia:	Usuário: mensagem de erro
Assume:	Que há ciclo e ramificação
Resultado:	Número de voltas no ciclo é fornecido

Operação:	Gerar o código
Descrição:	Produz o código fonte do programa de simulação referente ao modelo em edição
Lê:	arquivos .esp .mod .rec .eve .lop. .arq .dis
Modifica:	<i>new</i> arquivo .cpp
Envia:	Usuário: janela de confirmação, mensagens de erro
Assume:	
Resultado:	Código do programa de simulação construído

Operação:	Gerar o executável
Descrição:	Produz o programa (arquivo .exe) de simulação referente ao modelo em edição
Lê:	arquivo .cpp
Modifica:	<i>new</i> arquivo .exe
Envia:	Usuário: janela de confirmação, mensagens de erro
Assume:	
Resultado:	Programa de simulação gerado

Operação:	Código
Descrição:	Permite a visualização do código fonte do programa de simulação gerado
Lê:	arquivo .cpp
Modifica:	
Envia:	Usuário: janela de edição do código do programa de simulação
Assume:	
Resultado:	Janela com o código fonte aberto

Operação:	Animação
Descrição:	(Opção sem funcionalidade)
Lê:	
Modifica:	
Envia:	
Assume:	
Resultado:	

Operação:	Relatórios
Descrição:	Permite visualizar os resultados da execução do programa de simulação
Lê:	arquivo de resultados (.out)
Modifica:	
Envia:	Usuário: janela com os resultados
Assume:	Que o programa foi executado com sucesso
Resultado:	Janela aberta com os resultados da execução da simulação

Operação:	Personalizar
Descrição:	Permite que o usuário personalize a barra de ferramentas
Lê:	
Modifica:	Barra de acesso rápido do Editor Gráfico
Envia:	Usuário: barra de acesso rápido modificada no Editor Gráfico
Assume:	
Resultado:	Barra de acesso rápido personalizada de acordo com as alterações do usuário

Operação:	Ativar
Descrição:	Permite que o Módulo Arquitetura seja ativado
Lê:	
Modifica:	Barra de acesso rápido e de ferramentas do Editor Gráfico com as opções do Módulo Arquitetura
Envia:	Usuário: janela com tipos de arquiteturas pré-definidas para escolha
Assume:	
Resultado:	Barras de ferramentas do Editor Gráfico assume funções do Módulo Arquitetura

Operação:	von Neumann
Descrição:	Permite a abertura do modelo já parametrizado da arquitetura de von Neumann
Lê:	Arquivos von_neum.esp von_neum.eve von_neum.lop von_neum.rec von_neum.lig von_neum.mod von_neum.arq
Modifica:	Tela do Editor Gráfico: mostra modelo da arquitetura de von Neumann
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Modelo da arquitetura de von Neumann aberto no Editor Gráfico

Operação:	von Neumann com pipeline
Descrição:	Permite a abertura do modelo já parametrizado da arquitetura de von Neumann com pipeline
Lê:	Arquivos pipeline.esp pipeline.eve pipeline.lop pipeline.rec pipeline.lig pipeline.mod pipeline.arq
Modifica:	Tela do Editor Gráfico: mostra modelo da arquitetura de von Neumann com pipeline
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Modelo da arquitetura de von Neumann com pipeline aberto no Editor Gráfico

Operação:	Multiprocessador com memória distribuída
Descrição:	Abre o modelo pré-definido para Multiprocessador com memória distribuída
Lê:	Arquivos mimd_md.esp mimd_md.eve mimd_md.lop mimd_md.rec mimd_md.lig mimd_md.mod mimd_md.arq
Modifica:	Tela do Editor Gráfico: mostra modelo da arquitetura de multiprocessador com memória distribuída
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Modelo de arquitetura multiprocessada com memória distribuída aberto no Editor Gráfico

Operação:	Multiprocessador com memória compartilhada
Descrição:	Abre o modelo pré-definido para Multiprocessador com memória compartilhada
Lê:	Arquivos mimd_mc.esp mimd_mc.eve mimd_mc.lop mimd_mc.rec mimd_mc.lig mimd_mc.mod mimd_mc.arq
Modifica:	Tela do Editor Gráfico: mostra modelo da arquitetura de multiprocessador com memória compartilhada
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Modelo de arquitetura multiprocessada com memória compartilhada aberto no Editor Gráfico

Operação:	Tudo (MA)
Descrição:	Permite a parametrização automática do modelo pré-definido no Módulo Arquitetura
Lê:	supplied arquivos .esp .lig .arq
Modifica:	Arquivos .rec .mod .eve .lop
Envia:	Usuário: mensagens de erro
Assume:	
Resultado:	Modelo em edição parametrizado

Operação:	Ajuda
Descrição:	(Opção sem funcionalidade)
Lê:	
Modifica:	
Envia:	
Assume:	
Resultado:	

Operação:	Sobre o ASIA
Descrição:	Fornecer informações sucintas sobre o sistema
Lê:	
Modifica:	
Envia:	Usuário: janela de informações
Assume:	
Resultado:	Janela de informações é aberta

Operação:	Insere um recurso com uma fila e um servidor
Descrição:	Permite selecionar o recurso com uma fila e um servidor a inseri-lo na janela de edição gráfica.
Lê:	
Modifica:	Modelo em edição
Envia:	Editor Gráfico: recurso com uma fila e um servidor
Assume:	
Resultado:	Recurso com uma fila e um servidor é desenhado na janela de edição gráfica

Operação:	Insere um recurso com uma fila e vários servidores
Descrição:	Permite selecionar o recurso com uma fila e vários servidores e inseri-lo na janela de edição gráfica.
Lê:	
Modifica:	Modelo em edição
Envia:	Editor Gráfico: recurso com uma fila e vários servidores
Assume:	
Resultado:	Recurso com uma fila e vários servidores desenhado na janela de edição gráfica

Operação:	Insere um recurso com várias filas e um servidor
Descrição:	Permite seleccionar o recurso com várias filas e um servidor e inseri-lo na janela de edição gráfica.
Lê:	
Modifica:	Modelo em edição
Envia:	Editor Gráfico: recurso com várias filas e um servidor
Assume:	
Resultado:	Recurso com várias filas e um servidor é desenhado na janela de edição gráfica

Operação:	Insere um recurso com várias filas e vários servidores
Descrição:	Permite seleccionar o recurso com várias filas e vários servidores e inseri-lo na janela de edição gráfica.
Lê:	
Modifica:	Modelo em edição
Envia:	Editor Gráfico: recurso com várias filas e vários servidores
Assume:	
Resultado:	Recurso com várias filas e vários servidores desenhado na janela de edição gráfica

Operação:	Insere uma seta de início
Descrição:	Insere uma seta que representa o início do modelo, possibilitando a definição da entrada do sistema que está sendo modelado
Lê:	
Modifica:	Modelo em edição
Envia:	Editor Gráfico: sinal de inicialização do modelo
Assume:	
Resultado:	Seta de início desenhada na janela de edição gráfica

Operação:	insere uma seta de fim
Descrição:	Insere uma seta que representa o fim do modelo, possibilitando a definição da saída do sistema que está sendo modelado
Lê:	
Modifica:	Modelo em edição
Envia:	Editor Gráfico: seta de finalização do modelo
Assume:	
Resultado:	Seta de fim desenhada na janela de edição gráfica

Operação:	Insere uma ligação
Descrição:	Seleciona a ligação para relacionar os recursos da janela de edição gráfica. Uma ligação deve ter como origem e destino recursos desenhados. O recurso de destino pode, inclusive, ser o mesmo que o de origem, no caso de ciclos.
Lê:	
Modifica:	
Envia:	Usuário: (já existe ligação, deve ser ramificação), mensagens de erro Editor Gráfico: ligação entre dois recursos
Assume:	
Resultado:	Desenho da ligação entre os recursos seleccionados como origem e destino.

Operação:	Insere uma ramificação
Descrição:	Permite unir a saída já existente de um recurso do modelo a outro diferente do que faz parte da ligação.
Lê:	
Modifica:	
Envia:	Editor Gráfico: ramificação em uma ligação
Assume:	Que existe uma ligação entre o recurso de origem e outro recurso do modelo
Resultado:	Desenho da ramificação, tendo como origem um recurso e como destino dois recursos ou mais do modelo em edição.

Operação:	Seleciona um recurso
Descrição:	Permite que um recurso seja selecionado para ser parametrizado
Lê:	Posição do cursor na tela de edição gráfica
Modifica:	
Envia:	
Assume:	
Resultado:	Recurso selecionado é indicado por seta vermelha no Editor Gráfico

Operação:	Apaga símbolos e ligações
Descrição:	Seleciona a função apagar, que removerá da janela de edição gráfica o recurso, a ligação, a entrada, a saída ou a ramificação indicada.
Lê:	Arquivos .esp .lig .rec
Modifica:	Arquivos .esp .lig .rec
Envia:	Editor Gráfico: remoção do elemento
Assume:	
Resultado:	Apaga símbolo ou ligação selecionado do modelo em edição e das listas, além das ligações associadas ao símbolo (quando ele for um recurso).

Operação:	Movimenta recursos
Descrição:	Permite que os elementos gráficos (recursos e ligações) sejam movidos para outra posição na janela de edição gráfica
Lê:	Arquivo .esp
Modifica:	Modelo em edição
Envia:	Editor Gráfico: elemento em outra posição
Assume:	
Resultado:	Recurso é redesenhado em outra posição da janela de edição gráfica e as ligações (se houverem) são arrastadas.

Operação:	Insere um processador (MA)
Descrição:	Desenha o modelo de um processador na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de um processador
Assume:	
Resultado:	Modelo de um processador é desenhado na janela de edição gráfica

Operação:	Insere uma memória (MA)
Descrição:	Desenha o modelo de uma memória na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de uma memória
Assume:	
Resultado:	Modelo de uma memória é desenhado na janela de edição gráfica

Operação:	Insere um disco (MA)
Descrição:	Desenha o recurso que representa um disco na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de um disco
Assume:	
Resultado:	Modelo de um disco é desenhado na janela de edição gráfica

Operação:	Insere uma entrada (MA)
Descrição:	Desenha uma entrada para o modelo em edição na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: entrada para o modelo
Assume:	
Resultado:	Entrada para o modelo de uma arquitetura é desenhada na janela de edição gráfica

Operação:	Insere um bloco pipeline (MA)
Descrição:	Desenha um bloco pipeline na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de um pipeline
Assume:	
Resultado:	Modelo de um bloco pipeline é desenhado na janela de edição gráfica

Operação:	Insere um bloco de processadores (MA)
Descrição:	Desenha um bloco de processadores na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de um bloco de processadores
Assume:	
Resultado:	Modelo de um bloco de processadores é desenhado na janela de edição gráfica

Operação:	Insere um bloco de memórias (MA)
Descrição:	Desenha o modelo de um bloco de memórias na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de um bloco de memórias
Assume:	
Resultado:	Modelo de um bloco de memórias é desenhado na janela de edição gráfica

Operação:	Insere um bloco de discos (MA)
Descrição:	Desenha o modelo de um bloco de discos na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de um bloco de discos
Assume:	
Resultado:	Modelo de um bloco de discos é desenhado na janela de edição gráfica

Operação:	Insere uma saída (MA)
Descrição:	Desenha o modelo de uma saída na janela de edição gráfica
Lê:	Arquivo .esp e .arq
Modifica:	Modelo em edição
Envia:	Editor Gráfico: modelo de uma saída
Assume:	
Resultado:	Modelo de uma saída é desenhado na janela de edição gráfica

A.10. Procedimentos utilizados nas operações do ASiA

<i>Operação</i>	<i>Procedimentos</i>
Abrir	DlgSalvarArqEspecGrafica GravarArqEspecGrafica OkMessage ConverteExtNomeArq GravarArqLigacoes GravarArqModelo GravarArqRecursos GravarArqLoop GravarArqEventos DoCaption
Ajuda	
Animação	
Apaga símbolos e ligações	DesInicCondicoesLigacao
Cancel	
Código	ConverteExtNomeArq
Editar código	DesInicCondicoesLigacao
Fim	ConverteExtNomeArq
Gerar	Gerador
Gerar o código	Gerador
Gerar o executável	ConverteExtNomeArq
Help	
Inserir um bloco de discos	DesInicCondicoesLigacao
Inserir um bloco de memórias	DesInicCondicoesLigacao
Inserir um bloco de processadores	DesInicCondicoesLigacao
Inserir um bloco pipeline	InicCondicoesLigacao
Inserir um disco	DesInicCondicoesLigacao
Inserir um processador	DesInicCondicoesLigacao
Inserir um recurso com uma fila e um servidor	DesInicCondicoesLigacao
Inserir um recurso com uma fila e vários servidores	DesInicCondicoesLigacao
Inserir um recurso com várias filas e um servidor	DesInicCondicoesLigacao
Inserir um recurso com várias filas e vários servidores	DesInicCondicoesLigacao
Inserir uma entrada	DesInicCondicoesLigacao
Inserir uma ligação	InicCondicoesLigacao

<i>Operação</i>	<i>Procedimentos</i>
Inserir uma memória	DesInicCondiçõesLigação
Inserir uma ramificação	InicCondiçõesLigação
Inserir uma saída	DesInicCondiçõesLigação
Inserir uma seta de fim	InicCondiçõesLigação
Inserir uma seta de início	InicCondiçõesLigação
Ler	ConverteExtNomeArq
Modelo	JaTemInicio
	JaTemFim
	TodosRecTemLigParaOutro
	TemSimbolo
	RecursoJaParametrizado
	GuardaDadosRecurso
	GuardaDadosLoop
	TemQueParametrizarRecurso
	ApenasUmaSaida
	ExisteLoop
	LoopEProb
	VerificaConsistenciaRec
	OkMessage
	EInicio
	InsListaRec
	EFim
	InsListaRecChegada
	InsListaRecFila
	InsListaRecServico
	InsListaRecRamificacao
	TransfereDadosAnteriores
Mover recursos	DesInicCondiçõesLigação
Multiprocessador com mem. Compartilhada	InicCondiçõesLigação
Multiprocessador com mem. Distribuída	InicCondiçõesLigação

Operação**Procedimentos**

Novo

AskAboutSave
 ApagaLista
 ApagaListaLigacoes
 ApagaListaRecursos
 ApagaListaLoop
 ApagaListaEventos
 ApagaListaDist
 ApagaListaArq
 DigAbrirArqEspecGrafica
 TiraCaminho
 LerArqEspecGrafica
 OkMessage
 ConverteExtNomeArq
 AuxAbrirArqLigacoes
 LerArqLigacoes
 LerArqModelo
 LerArqRecursos
 LerArqLoop
 LerArqEventos
 LerArqArquitet
 DoCaption
 InicializaDadosLoop
 InicializaDadosEventos
 InicializaDadosModelo
 InicializaDadosRecursos
 InicializaDadosDistribuicao

Ok

ConverteExtNomeArq

Personalizar

DesInicCondicoesLigacao

Relatórios

DesInicCondicoesLigacao

ConverteExtNomeArq

Sair

JaTemInicio

JaTemFim

TodosRecTemLigParaOutro

VerificaConsistenciaModelo

ModeloDigProc

OkMessage

Salvar

AskAboutSave

Operação

Salvar como

Procedimentos

GravarArqEspecGrafica
OkMessage
DlgAbrirArqEspecGrafica
ConverteExtNomeArq
GravarArqLigacoes
GravarArqModelo
GravarArqRecursos
GravarArqLoop
GravarArqEventos
GravarArqArquitet
DoCaption

Seleciona um recurso

DesInicCondicoesLigacao

Sobre o ASiA

DesInicCondicoesLigacao

Tudo

TemSimbolo
JaTemInicio
JaTemFim
TodosRecTemLigParaOutro
Parametriza

von Neumann

InicCondicoesLigacao

von Neumann com pipeline

InicCondicoesLigacao

B.1. Modelo de Objetos do MAS

B.2. Especificação dos atributos para as classes do MAS

Classe: Modelo
Atributos
Nome_modelo
Num_max_entidades
Tempo/Num_ciclos
Relatorio_final
Tamanho_batch_means
Tempo_prim_chegada
Qtde_valores_desprezados
Tempo_warm_up

Classe: Recurso
Atributos
Nome_recurso
Desvio_padrao_chegada
Distribuicao_chegada
Media_chegada
Nome_distr_chegada
Pto_triag_chegada
Semente_chegada
Desvio_padrao_servico
Distribuicao_servico
Media_servico
Nome_distr_servico
Pto_triag_servico
Semente_servico
Disciplina_fila
Escolha_cliente
Escolha_fila
Escolha_servidor
Escolha_prox_rec
Num_filas
Num_servidores
Est_fila_vazia
Est_max_min
Tipo_recurso
X_inicial
Y_inicial

Classe: Distribuição
Atributos
Probabilidade
Valor

Classe: Ligação
Atributos
Rec_origem
Rec_destino
Tipo_ligação
X_inicial
Y_inicial
X_final
Y_final

Classe: Evento
Atributos
Tempo_ocorrenda

Classe: Loop
Atributos
Num_voltas

Classe: Probabilidade
Atributos
Valor_prob_ramificação

Classe: Arquitetura
Atributos
Num_blocos
Tipo_recurso

B.3. Especificação dos métodos das classes do MAS

Classe: Modelo
Métodos
Abrir modelo
Editar o código
Gerar o código de simulação
Gerar o executável
Inserir um recurso com uma fila e um servidor
Inserir um recurso com várias filas e um servidor
Inserir um recurso com uma fila e vários servidores
Inserir um recurso com várias filas e vários servidores
Inserir uma ligação
Inserir uma ramificação
Inserir uma seta de início
Inserir uma seta de fim
Parametrizar modelo
Novo
Visualizar resultados da simulação
Salvar modelo
Salvar modelo como

Classe: Recurso
Métodos
Apaga recurso
Mover recurso
Parametrizar recurso
Seleciona um recurso

Classe: Distribuição
Métodos
Gerar uma distribuição de usuário
Ler uma distribuição gerada

Classe: Ligação
Métodos
Apaga ligação

Classe: Arquitetura
Métodos
Inserir um bloco de discos
Inserir um bloco de memórias
Inserir um bloco de processadores
Inserir um bloco pipeline
Inserir uma memória
Inserir um processador
Inserir um disco
Inserir uma entrada
Inserir uma saída
Inserir arquitetura de multiprocessador com memória compartilhada
Inserir arquitetura de multiprocessador com memória distribuída
Parametriza tudo
Inserir arquitetura de von Neumann
Inserir arquitetura de von Neumann com pipeline

B.4. Modelo de Ciclo de Vida do MAS

life cycle ASiA : (Arquivo | Módulo Arquitetura | ? | Barra de ferramentas)+ . (Arquivo | Parametrizar | Gerador | Visualizar | Módulo Arquitetura | ? | Barra de ferramentas)*

Arquivo = ((Novo . (Salvar Modelo Como . #Nome_arquivo)) . (Novo | Abrir modelo . #Nome_arquivo | Salvar Modelo | Salvar Modelo Como . #Nome_arquivo | Sair)*) | Sair

Parametrizar = (Parametrizar Modelo . #Janela_parametrização_modelo | Parametrizar Recurso)

Parametrizar Recurso = #Janela_parametrização_recurso . [(Ligação_Principal . Ramificação) . Loop] . (Chegada . Fila . Serviço . Eventos)* . (Ok | Cancel | Help*)

Chegada = #Janela_parametrizar_chegada . [Gerar uma distribuição de usuário . #Janela_Distribuição . (Ok | Cancel | Fim | Help*) | Ler uma distribuição de usuário . #Janela_abrir_distribuição]

Fila = #Janela_parametrizar_fila . (Ok | Cancel | Help*)

Serviço = #Janela_parametrizar_serviço . [Gerar uma distribuição de usuário . #Janela_Distribuição . (Ok | Cancel | Fim | Help*) | Ler uma distribuição de usuário . #Janela_abrir_distribuição]

Eventos = #Janela_parametrizar_eventos . (Ok | Cancel | Help*)

Ligação_Principal = #Janela_ligação_principal . (Ok | Cancel | Help*)

Ramificação = #Janela_ramificação . (Ok | Cancel | Help*)

Loop = #Janela_loop . (Ok | Cancel | Help*)

Gerador = (Gerar o código de simulação . #Janela_de_confirmação_de_geração_de_código | Gerar o executável . #Janela_de_confirmação_de_geração_do_executável)

Visualizar = (Editar o código . #Código_Gerado | Visualizar resultados da simulação . #Resultados_da_simulação)

Módulo Arquitetura = (Ativar . #Janela_Tipos . ((Insere arquitetura de von Neumann | Insere arquitetura de von Neumann com pipeline | Insere arquitetura de multiprocessador com memória compartilhada | Insere arquitetura de multiprocessador com memória distribuída) . (Ok | Cancel | Help*)) | Personalizar . #Janela_Personalizar_Barra_de_ferramentas)

Ativar = (Arquivo | Módulo Arquitetura | ? | Barra de ferramentas)+ . (Arquivo | Parametrizar | Gerador | Visualizar | Módulo Arquitetura | ? | Barra de ferramentas)*

Parametrizar = (Parametriza tudo | Parametriza modelo . #Janela_parametrização_modelo | Parametriza recurso)

? = (Ajuda | Sobre o ASiA)

Barra de ferramentas = ((Insere um recurso com uma fila e um servidor | Insere um recurso com uma fila e vários servidores | Insere um recurso com várias filas e um servidor | Insere um recurso com várias filas e vários servidores | Insere uma ligação | Insere uma ramificação)+ . Insere uma seta de início . Insere uma seta de fim . Seleciona um recurso . (Apaga símbolos e ligações | Move recurso))*

Barra de ferramentas módulo arquitetura = ((Insere um processador | Insere uma memória | Insere um disco | Insere uma entrada | Insere um bloco pipeline | Insere um bloco de processadores | Insere um bloco de memórias | Insere um bloco de discos | Insere uma saída | Insere uma ligação | Insere uma ramificação)+ . Insere uma seta de início . Insere uma seta de fim . Seleciona um recurso . (Apaga símbolos e ligações | Move recurso))*

B.5. Mapeamento entre o MAS e o MASA

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos		
Classe: Recurso	auxrec	lista_recursos
Atributos	Elementos de dados	Elementos de dados
Nome_recurso	nome_recurso	nome_recurso
Desvio_padrao_chegada	desvio_padrao_chegada	desvio_padrao_chegada
Distribuicao_chegada	distribuicao_chegada	distribuicao_chegada
Media_chegada	media_chegada	media_chegada
Nome_distr_chegada	nome_distr_chegada	nome_distr_chegada
Pto_triangu_chegada	pto_triangu_chegada	pto_triangu_chegada
Semente_chegada	semente_chegada	semente_chegada
Desvio_padrao_servico	desvio_padrao_servico	desvio_padrao_servico
Distribuicao_servico	distribuicao_servico	distribuicao_servico
Media_servico	media_servico	media_servico
Nome_distr_servico	nome_distr_servico	nome_distr_servico
Pto_triangu_servico	pto_triangu_servico	pto_triangu_servico
Semente_servico	semente_servico	semente_servico
Disciplina_fila	disciplina	disciplina
Escolha_cliente	escolha_cliente	escolha_cliente
Escolha_fila	escolha_fila	escolha_fila
Escolha_servidor	escolha_servidor	escolha_servidor
Escolha_prox_rec	escolha_prox_rec	escolha_prox_rec
Num_filas	num_filas	num_filas
Num_servidores	num_servidores	num_servidores
Est_fila_vazia	est_fila_vazia	est_fila_vazia
Est_max_min	est_max_min	est_max_min
Tipo_recurso	tipo (auxsimb)	
X_inicial	x (auxsimb)	
Y_inicial	y (auxsimb)	
	entradas_saidas	entradas_saidas
	sem_fila	sem_fila
	flag_sinc	flag_sinc
	libera_recurso	libera_recurso
	ident_rec	ident_rec
		prox

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos Classe: Modelo	auxspecifi	
Atributos	Elementos de dados	Elementos de dados
Nome_modelo Num_max_entidades Tempo/Num_ciclos Relatorio_final Tamanho_batch_means Tempo_prim_chegada Qtde_valores_desprezados Tempo_warm_up .	nome_modelo num_max_entidades tempo num_voltas relatorio_final tamanho_batch tempo_prim_chegada valores_desprezados warm_up distrib_tab reesc modelo_fechado	

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos Classe: Ligação	auxlig	lista_ligacoes
Atributos	Elementos de dados	Elementos de dados
Rec_origem Rec_destino Tipo_ligação X_inicial Y_inicial X_final Y_final	recurso1 recurso2 tipo x1 y1 x2 y2	recurso1 recurso2 tipo x1 y1 x2 y2 prox

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos Classe: Evento	auxtempeventos	lista_temp_eventos
Atributos	Elementos de dados	Elementos de dados
Tempo_ocorrendia	tempo_ocorrendia num_recurso	tempo_ocorrendia num_recurso prox

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos		
Classe: Loop	auxtotloop	lista_tot_loop
Atributos	Elementos de dados	Elementos de dados
Num_voltas	n_loop n_rec	n_loop n_rec prox

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos		
Classe: Distribuição	auxdisttab	lista_dist_tab
Atributos	Elementos de dados	Elementos de dados
Probabilidade Valor	prob valor	prob valor prox

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos		
Classe: Arquitetura	auxarquitet	lista_arquitet
Atributos	Elementos de dados	Elementos de dados
Num_blocos Tipo_recurso	num_bloco tipo ident_rec	num_bloco tipo ident_rec prox

MAS	MASA (tema permanente)	MASA (tema temporário)
Modelo de Objetos		
Classe: Probabilidade		
Atributos	Elementos de dados	Elementos de dados
Valor_prob_ramificação	entradas_saidas (auxrec)	