UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA

UM MÉTODO PARA ENGENHARIA REVERSA ORIENTADA A OBJETOS

Rosângela Aparecida Dellosso Penteado

OK

Tese apresentada ao Instituto de Física de São Carlos da Universidade de São Paulo para a obtenção do título de Doutor em Ciências (Física Aplicada - opção Física Computacional)

Orientador: Prof. Dr. Paulo Cesar Masiero

São Carlos
1996

MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE ROSÂNGELA APARECIDA DELLOSSO PENTEADO APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, UNIVERSIDADE DE SÃO PAULO, EM 19/01/1996.

COMISSÃO JULGADORA:

Prof. Dr. Paulo Cesar Masiero

Prof. Dr. Caetano Traina Junior

Prof. Dr. Julio Cesar Sampaio do Prado Leite

Prof. Dr. Antonio Francisco do Prado

Prof. Dr. Luciano da Fontoura Costa

Penteado, Rosângela A. D.

Um método para engenharia reversa orientada a objetos/Rosângela A. D. Penteado. São Carlos, 1996. 251 p.

Tese (Doutorado) - Instituto de Física de São Carlos, 1996.

Orientador: Prof. Dr. Paulo Cesar Masiero

**1. Engenharia Reversa. 2. Statecharts. 3. Método Fusion
4. Orientação a Objetos. I. Título**

Ao Dirceu, Eduardo e Lucas;

Aos meus pais

Noélia e João (in memoriam)

Agradecimentos

A Deus pela vida e tudo o que tenho.

Ao Prof. Dr. Paulo Cesar Masiero pela orientação durante todo o decorrer deste trabalho, pela sua amizade renovada a cada dia, pela sua atenção em todos os momentos, por ter me mostrado algumas pistas a serem seguidas no caminho da pesquisa. Pela confiança em mim depositada e pela disponibilidade em me ceder sua sala e os seus recursos computacionais.

Ao Prof. Dr. Fernão S. de R. Germano pela dedicação com que acompanhou o desenvolvimento deste trabalho, pelas valiosas sugestões e por tudo que me ensinou desde o tempo de mestrado.

Aos meus amigos Regina e Marcos Santana pelo empenho para que esse doutorado tivesse início, pelas injeções de ânimo e sobretudo, pelo carinho e amizade.

Ao Prof. Dr. José Carlos Maldonado pelo constante incentivo, apoio e disponibilidade para discussões.

Ao Dirceu, companheiro de todas as horas, que jamais mediu esforços para que eu pudesse realizar este trabalho, deixando muitas vezes o seu para “mais tarde”. Pela sua dedicação e seu amor em todos os momentos, principalmente naqueles em que foi pai e mãe. Eduardo e Lucas: finalmente o doutorado chegou ao fim, obrigado por vocês compreenderem todo o tempo que me ausentei e pelo muito que aprendi com a vinda de vocês.

À Sandra Fabbri pela disponibilidade constante em auxiliar, pelo “dividir de sala” no DC-UFSCar, pela estima e amizade.

Aos amigos conquistados durante essa etapa e que com certeza juntos continuaremos: João Cangussu, Rosemeire, Simone e Paulo, Ines, Rejane, Lairce, Elisa, Carmo, Tchelo, Mauro, Lucia Rino, Sandra, Renata. Agradeço

especialmente aos meus amigos-alunos Fabiano Borges Paulo e Márcia Cristina dos Santos pelos trabalhos desenvolvidos durante a iniciação científica e que muito contrubuíram para a elaboração deste.

À minha mãe Noélia, Maria Teresa e Mário, Tias Tita e Alice que estão sempre prontos a me auxiliar e compreender.

À Equipe 15, à Dete e ao Tide “pelo peso com que os sobrecarreguei” durante esses anos.

Às secretárias , bibliotecárias e funcionários do Instituto de Ciências Matemáticas de São Carlos, pelo carinho com que sempre me trataram, abrindo as portas para que esse período se tornasse agradável.

Às secretárias da pós-graduação e do Instituto de Física de São Carlos, Wladerez e Ivani, pela atenção e disponibilidade com que sempre me atenderam.

Aos docentes e funcionários do Departamento de Computação da UFSCar pela oportunidade para que eu pudesse realizar este trabalho.

Ao CNPq, CAPES e FAPESP pelos auxílios concedidos.

Sumário

Lista de Figuras	i
-------------------------------	----------

Lista de Tabelas	v
-------------------------------	----------

Resumo.....	vii
--------------------	------------

Abstract	viii
-----------------------	-------------

Capítulo 1

Introdução	1
1.1. Considerações Iniciais	1
1.2. Trabalhos Anteriores	2
1.3. Motivação	3
1.4. O Problema a Ser Resolvido	5
1.5. Organização da Tese	5

Capítulo 2

Resenha Bibliográfica	8
2.1. Considerações Iniciais	8
2.2. Orientação a Objetos.....	9
2.3. Sistemas Reativos e Abordagens para o seu Desenvolvimento.....	12
2.4. Métodos Orientados a Objetos Para Desenvolvimento de Sistemas.....	24
2.5. Reengenharia de Sistemas	33
2.6. Engenharia Reversa de Sistemas	39

2.7. Considerações Finais	50
---------------------------------	----

Capítulo 3

O Ambiente StatSim	52
3.1. Considerações Iniciais	52
3.2. Statecharts	53
3.3. O Ambiente StatSim	60
3.3.1. Versão Inicial	60
3.3.2. Versão Acrescida de Outros Modos de Simulação	64
3.4. Evolução do Ambiente StatSim - Novas Funcionalidades	69
3.5. Uso do Ambiente Statsim	81
3.5.1. Exemplo: um despertador	81
3.5.2. O Método	82
3.5.2.1. Preparar um Modelo de Aplicação	84
3.5.2.2. Transformar os Vários Statecharts em um Único e os Vários Diagramas de Atividades em um Único	89
3.5.2.3. Validar o Statechart Usando o StatSim	92
3.5.2.4. Projetar	112
3.5.2.5. Implementar	114
3.5.2.6. Testar	118
3.6. Considerações Finais	118

Capítulo 4

Engenharia Reversa Orientada a Objetos	120
4.1. Considerações Iniciais	120
4.2 Engenharia Reversa no Contexto de Reengenharia	121
4.3. O Método Fusion	127
4.4 Método de engenharia reversa orientada a Objetos	139

4.4.1. Revitalizar a Arquitetura do Sistema com Base na Documentação Existente	139
4.4.2 Recuperar o Modelo de Análise da Solução Atual.....	142
4.4.2.1. Definir Temas	142
4.4.2.2. Desenvolver o Modelo de Objetos do Sistema Existente	143
4.4.2.3. Definir o Ciclo de Vida.....	148
4.4.2.4. Abstrair e Desenvolver as Operações	149
4.4.3. Abstrair para o Modelo de Análise do Sistema	150
4.4.3.1. Desenvolver o Modelo de Objetos	151
4.4.3.2. Elaborar o Modelo de Ciclo de Vida	153
4.4.3.3. Especificar as Operações	154
4.4.4. Mapear o Modelo de Análise do Sistema para o Modelo de Análise do Sistema Atual.....	154
4.5. Interligação dos Documentos Produzidos.....	155
4.6. Métricas Para Avaliação do Esforço de Conversão	157
4.7. Reengenharia de Software	162
4.7.1. Com Aperfeiçoamento da Funcionalidade do Sistema	162
4.7.1.1. Identificar as Partes a Alterar	163
4.7.1.2. Desenvolver o Modelo de Objetos Aperfeiçoado	165
4.7.1.3. Elaborar o Modelo de Ciclo de Vida Aperfeiçoado.....	166
4.7.1.4. Especificar o Modelo de Operações Aperfeiçoado	166
4.7.1.5. Mapeamento do MASAp para o MAS	167
4.7.1.6. Continuação dos Passos de Reengenharia	167
4.7.2. Com Troca Parcial da Implementação	170
4.7.3. Com Troca Total da Implementação	171
4.7.4. Outras Combinações	172
4.8. Considerações Finais	173

Capítulo 5

Estudo de Caso: Engenharia Reversa do Ambiente Statsim	175
5.1. Considerações Iniciais	175
5.2. Revitalização da Arquitetura	176
5.3 Recuperação do Modelo de Análise da Solução Atual (MASA).....	179
5.3.1 Definição dos Temas	180
5.3.2 Desenvolvimento do Modelo de Objetos	182
5.3.3 Definição do Ciclo de Vida.....	189
5.3.4 Abstração e Desenvolvimento das Operações	190
5.4. Abstração para o Modelo de Análise do Sistema (MAS)	192
5.4.1. Desenvolvimento do Modelo de Objetos	192
5.4.2. Elaboração do Modelo de Ciclo de Vida	203
5.4.3 Elaboração do Modelo de Operações.....	203
5.5. Mapeamento do MAS para o MASA	205
5.6. Avaliação	212
5.6.1. Avaliação do Sistema Atual	212
a) Índice de Esforço de Conversão (IEC).....	213
b) Índice de Influência do Tipo de Implementação Utilizada (IITIU)	213
c) Influência do Código nos Procedimentos de Implementação (ICPI).....	214
d) Checagem dos Métodos Considerados (CMC)	214
5.6.2. Do Modelo Orientado a Objetos Produzido	215
5.6.3. Do Método de Engenharia Reversa Proposto	215
5.7. Considerações Finais	216

Capítulo 6

Conclusões e Sugestões para Futuras Pesquisas	218
6.1. Considerações Iniciais	218

6.2. Conclusões Gerais	218
6.3. Contribuições deste Trabalho	220
6.4. Sugestões para Futuras Pesquisas	222
Referências Bibliográficas.....	225
 Apêndice 1	
Gramática Utilizada no Ambiente StatSim	237
A1.1 Para definição de statechart.....	237
A1.2 Para definição de estado e atividades	237
A1.3 Para definição de transições	238
A1.4 Definições e abreviações	239
 Apêndice 2	
Modelos de Objetos do MASA Para os Temas Considerados	241
 Apêndice 3	
Modelo de Objetos do MAS Para os Temas Considerados	245
 Apêndice 4	
Opções Existentes na Interface do Ambiente StatSim e os Procedimentos que as Implementam	248

Lista de Figuras

Figura 2.1 - Notação do Diagrama de Estrutura Usada no Statemate.....	16
Figura 2.2 - Representação Gráfica para Diagrama de Atividades.....	16
Figura 2.3 - Decomposição Funcional.....	19
Figura 2.4 - Decomposição Funcional Orientada a Objeto.....	19
Figura 2.5 - Decomposição Comportamental.....	22
Figura 2.6 - Decomposição Comportamental Orientada a Objetos.....	23
Figura 2.7 - Influências que o Método Fusion Sofreu de Outros Métodos.....	27
Figura 3.1 - Exemplo de Estados.....	54
Figura 3.2 - Exemplo de um Statechart.....	56
Figura 3.3. -História.....	58
Figura 3.4 - Características do Ambiente StatSim Antes deste Trabalho.....	68
Figura 3.5 - Tela de Edição do Ambiente StatSim.....	70
Figura 3.6 - Tela de Simulação do Ambiente StatSim.....	70
Figura 3.7 - Exemplo não Permitido de “default” com Ação.....	71
Figura 3.8(a) -Criação de Transição “Default” com Ação.....	72
Figura 3.8(b) - Valor da Variável <i>com</i> Antes de Iniciar a Simulação.....	73
Figura 3.9 - Início da Simulação com Configuração Default.....	73
Figura 3.10 - Simulação da Transição “default” com Ação.....	74
Figura 3.11 - Evento Nulo.....	75
Figura 3.12 - Trecho da Gramática Modificada para Edição de Atividades.....	76
Figura 3.13 - Edição de Atividades quando da Criação do Estado.....	77
Figura 3.14 - Edição de Atividades posterior à Criação do Estado.....	78
Figura 3.15 - Características do Ambiente StatSim após este Trabalho.....	80
Figura 3.16 - Esboço do Relógio Despertador.....	82
Figura 3.17 - Diagrama de Atividades e Statechart do Relógio Despertador.....	85
Figura 3.18 - Diagrama de Atividades e Statechart Correspondente à Parte que Controla a Hora do Relógio Despertador.....	86
Figura 3.19 - Diagrama de Atividades e Statecharts Correspondente ao Alarme.....	87
Figura 3.20 - Diagrama de Atividade e Statecharts Correspondente ao	

Alarme Ligado.....	88
Figura 3.21 - Diagrama de Atividades Único e Statechart Único do Relógio	
Despertador.....	90
Figura 3.22 - Statechart Com Especificação não Correta para o Relógio	
Despertador.....	94
Figura 3.23 - Diagrama de Atividades e Statechart Corrigidos Correspondentes	
ao Alarme.....	95
Figura 3.24 - Diagrama de Atividades e Statechart Correspondentes ao	
Alarme Ligado.....	96
Figura 3.25 - Diagrama de Atividades e Statechart Após as Correções Realizadas..	98
Figura 3.26 - Statechart do Relógio Despertador Editado no Ambiente StatSim.....	99
Figura 3.27 - Início da Simulação Interativa do Relógio Despertador	
(Default Configuration)	101
Figura 3.28 - Acertando a Hora em que o Relógio Deve Despertar.....	102
Figura 3.29 - Alarme Soando.....	103
Figura 3.30 - Evento Soneca Ocorreu.....	104
Figura 3.31 - Evento Parado Ocorreu.....	105
Figura 3.32 - Evento Desligado Ocorreu.....	105
Figura 3.33 - Programa LCE para o Exemplo do Relógio Despertador.....	106
Figura 3.34 - Relatório Estatístico Resultante da Simulação Programada para o	
Exemplo do Relógio Despertador.....	108
Figura 3.35 - Trecho do Relatório de Passos da Simulação Programada para o	
Exemplo do Relógio Despertador.....	112
Figura 3.36 - Esboço dos Processos do Sistema.....	113
Figura 3.37 - Esboço Inicial do Relógio.....	115
Figura 3.38 -Horário de Despertar Acertado.....	116
Figura 3.39 - Botão Confirma Pressionado.....	116
Figura 3.40 - Relógio com Alarme Soando.....	117
Figura 3.41 - Botão Pare Pressionado.....	117
Figura 4.1 - Visão Geral do Método Fusion.....	128
Figura 4.2 - Representação de Classes e Relacionamentos.....	129
Figura 4.3 (a) - Representação de Agregação.....	130
Figura 4.3 (b) - Representação de Especialização/Generalização de Classes	
(Classes Subtipos Não Disjuntas)	130

Figura 4.3 (c) - Representação de Especialização/Generalização de Classes	
(Classes Subtipos Disjuntas)	131
Figura 4.4 - Modelo de Ciclo de Vida	132
Figura 4.5 - Modelo de Operações	133
Figura 4.6 - Notação do Grafo de Interação de Objetos	135
Figura 4.7 - Notação para o Grafo de Visibilidade	136
Figura 4.8 - Notação de Descrição de Classe	137
Figura 4.9 - Palavras Chaves para Descrição das Características dos Atributos	
na Descrição de Classe	138
Figura 4.10 - Procedimentos de um Módulo	141
Figura 4.11 - Cenários Possíveis da Implementação Atual	144
Figura 4.12 - Descrição da Classe, Atributos e Procedimentos	145
Figura 4.13 - Classificação dos Procedimentos de Determinada Classe	148
Figura 4.14 - Opções existentes na interface do ambiente	149
Figura 4.15 - Fontes de informações para o Modelo de Operação	150
Figura 4.16 - Tabela com o Mapeamento entre MAS e o MASA	155
Figura 4.17 - Interligação dos Documentos Existentes e Produzidos	156
Figura 4.18 - Métrica para Calcular o Índice do Esforço de Conversão	158
Figura 4.19 - Métrica para Calcular o Índice de Influência do Tipo de	
Implementação Utilizada	159
Figura 4.20 - Métrica para Calcular a Influência do Código nos Procedimentos	
de Implementação	160
Figura 4.21 - Métrica para Checagem do Métodos Considerados	161
Figura 4.22 - Resultado da Fase de Identificar as Partes a Alterar	154
Figura 4.23 - Interligação dos Documentos utilizados para Reengenharia com	
Aperfeiçoamento da Funcionalidade do Sistema	168
Figura 5.1 - Código Fonte parcial do Módulo Bolha.h	177
Figura 5.2 - Trecho da documentação produzida pela Revitalização da	
Arquitetura	178
Figura 5.3 - Temas	180
Figura 5.4 - Código fonte das Estruturas de Dados do StatSim	181
Figura 5.5 - Modelo de Objetos do MASA para o Tema Statechart Temporário	183
Figura 5.6 - Classe Nó_Estado	185
Figura 5.7 - Classificação de Parte dos Procedimentos Associados à Classe	186

Figura 5.8 - Modelo de Ciclo de Vida.....	189
Figura 5.9 - Modelo de Operação: cria_bolha.....	191
Figura 5.10 - Modelo de Objetos de Statechart com Alto Nível de Abstração.....	193
Figura 5.11 Tema Statecharts: Modelo de Objetos para Estado e Transição com seus Relacionamentos.....	194
Figura 5.12- Tema Statechart: Modelo de Objetos para as Expressões Executáveis.....	195
Figura 5.13 - Relacionamentos da Classe Estado.....	196
Figura 5.14 - Combinações das Duas Abstrações da Figura 5.13.....	197
Figura 5.15 - Modelo de Objetos do Sistema.....	198
Figura 5.16 - Lista de atributos e métodos (parcial) da Classe Estado do MAS.....	201
Figura 5.17 - Dicionário de Dados (Parcial)	202
Figura 5.18 - Modelo de Ciclo de Vida do MAS.....	203
Figura 5.19 - Modelo de Operação: criar_estado.....	204
Figura A21 - Modelo de Objetos Para o Tema Diagrama (MASA)	241
Figura A22 - Modelo de Objetos do Tema Diagrama (MASA) com Refinamento de Bolha e Arco.....	242
Figura A23 - Modelo de Objetos para o Tema Relatórios (MASA)	243
Figura A24 - Modelo de Objetos para o Tema Statechart Permanente.....	244
Figura A31 - Modelo de Objetos para o Tema Diagrama (MAS)	245
Figura A32 - Modelo de Objetos para o Tema Diagrama (MAS) com o Refinamento de Bolha e Arco.....	246
Figura A33 - Modelo de Objetos para o Tema Relatórios (MAS)	247

Lista de Tabelas

Tabela 2.1 - Características dos Métodos/Ambiente Citados para Especificação de Sistemas Reativos.....	18
Tabela 2.2 - Abstração de Estrutura	32
Tabela 2.3 - Abstração de Funções	32
Tabela 2.4 - Abstração de Comportamento	33
Tabela 3.1 - Conceitos Adicionais de Statecharts.....	59
Tabela 3.2(a) - Comportamento do Editor e Simulador de Statecharts na 1ª Versão para os Conceitos de Estado e Transição	61
Tabela 3.2(b) - Comportamento do Editor e Simulador de Statecharts na 1ª Versão para os Conceitos de Expressões e Condições	62
Tabela 3.2(c) - Comportamento do Editor e Simulador de Statecharts na 1ª Versão para os Conceitos de Eventos.....	63
Tabela 3.2(d) - Comportamento do Editor e Simulador de Statecharts na 1ª Versão para os Conceitos de Ações e Atividades.....	64
Tabela 3.3(a) - Comportamento do Editor e Simulador de Statecharts na 2ª Versão para os Conceitos de Estado e Transição.....	66
Tabela 3.3(b) - Comportamento do Editor e Simulador de Statecharts na 2ª Versão para os Conceitos de Expressões e Condições.....	67
Tabela 3.3(c) - Comportamento do Editor e Simulador de Statecharts na 2ª Versão para os Conceitos de Evento.....	67
Tabela 3.3(d) - Comportamento do Editor e Simulador de Statecharts na 2ª Versão para os Conceitos de Ações e Atividades.....	68
Tabela 3.4 - Ações Associadas às Transições.....	100
Tabela 3.5 - Atividades e Ações Associadas aos Estados	100
Tabela 4.1 - Operadores das Expressões Regulares do Modelo de Ciclo de Vida.....	132
Tabela 4.2 - Método de Engenharia Reversa Orientada a Objetos.....	140
Tabela 4.3 - Correspondência entre os Procedimentos e Métodos.....	152
Tabela 4.4 - Métrica para Avaliar a Conversão de um Sistema não Orientado a Objetos para um Sistema Orientado a Objetos.....	158

Tabela 4.5 - Métrica para Avaliar a Percentagem de Procedimentos de Implementação em um Sistema.....	159
Tabela 4.6 - Resumo dos Passos da Aplicação de Reengenharia com Aperfeiçoamento da Funcionalidade.....	169
Tabela 5.1 - Resumo da Classificação dos Procedimentos do Ambiente StatSim.....	187
Tabela 5.2 - Algumas Opções Existentes na Interface do Ambiente para o Modo de Edição.....	188
Tabela 5.3 - Atributos MASA x MAS da Classe que Representa as Informações de ESTADO.....	199
Tabela 5.4 - Procedimentos do MASA e Métodos Correspondentes do MAS.....	200
Tabela 5.5 - MAS X MASA para a Classe Nó_Estado.....	205
Tabela 5.6 - Correspondência entre as Operações do MAS e MASA para a Classe Estado	206
Tabela 5.7 - MAS x MASA para a Classe Transição.....	207
Tabela 5.8 - Correspondência entre as Operações do MAS e MASA para a Classe Transição.....	208
Tabela 5.9 - MAS x MASA para Evento.....	209
Tabela 5.10 - MAS x MASA para Variáveis.....	210
Tabela 5.11 - MAS x MASA para Atividades.....	211
Tabela 5.12 - MASA x MAS	211
Tabela A41 - Opções da Interface para o Modo de Edição (Files, Activities, Variables, Hide/Closure.....	249
Tabela A42 - Opções da Interface para o Modo de Simulação (Files, Decompose, Informations).....	250
Tabela A43 - Opções da Interface para o Modo de Simulação (Variables, Default Configuration, Simulation, Miscellaneous).....	251

RESUMO

Um método composto de quatro passos para conduzir engenharia reversa orientada a objetos de sistemas desenvolvidos sem usar a tecnologia de orientação a objetos é apresentado. Este método abrangente e detalhado mostrando como fazer engenharia reversa constitui um avanço para o estado da arte, pois não há métodos publicados nesse nível de detalhe até o momento. A documentação produzida pela engenharia reversa baseada no Método Fusion para desenvolvimento de sistemas orientados a objetos é composta de um Modelo de Objetos, um Modelo de Operação e um Modelo de Ciclo de Vida. Um conjunto de modelos intermediário extraído diretamente do código e dos documentos de interação processo/dados é também incluído com o nome de Modelo de Análise do Sistema Atual. A engenharia reversa de um ambiente de software para projetar sistemas reativos é mostrada em detalhes. Esse software tem 30.000 linhas de código C, divididas em 495 procedimentos agrupados em 36 módulos. Um documento semelhante a um dicionário de dados foi preparado com base na documentação contida em diversas dissertações, artigos e principalmente em seu código fonte. O ambiente de desenvolvimento de sistemas reativos é descrito, bem como sua evolução através de diversas versões e sua aplicação é ilustrada pelo projeto do statechart correspondente a um relógio despertador. Facilidades para especificar atividades associadas aos statecharts foram incorporadas ao ambiente StatSim como parte dos trabalhos desenvolvidos nesta tese e são também discutidas e aplicadas a um exemplo. A simulação de statecharts de controle sujeitos a diferentes eventos possíveis é incluída através de diagramas e "logs". Esse exemplo é também usado para ilustrar um procedimento relacionado ao acoplamento de statecharts e diagramas de atividades na especificação de sistemas reativos e para mostrar o uso do ambiente de software para verificar essa especificação. O Método Fusion é comparado a outros métodos de desenvolvimento de sistemas orientados a objetos. O ambiente de desenvolvimento de sistemas reativos baseado em statecharts é posicionado entre outras ferramentas para o desenvolvimento de tais sistemas.

ABSTRACT

A method composed of four steps to conduct object oriented reverse engineering in systems developed using non object oriented technology is presented. This comprehensive, detailed method showing how to do reverse engineering is an advance to the state of the art as there are no published methods hitherto at this level of detail. The documentation produced by the reverse engineering based on the Fusion Method for object oriented systems development is composed by an Object Model, an Operation Model and a Life Cycle Model. An intermediary set of models directly extracted from the code and from a process/data interaction document is also included with the name of Current Systems Analysis Model. The reverse engineering of a software environment for designing reactive systems is shown in detail. This software comprises 30.000 lines of code in C, spread for 495 procedures grouped in 36 modules. This data dictionary-like document was prepared based on documentation contained in several dissertations and papers and mainly on the code itself. The reactive systems software development environment is described as well as its evolution through several versions, and its application is illustrated by designing the statecharts corresponding to an alarm clock. Facilities to specify activities triggered by the statecharts incorporated to the StatSim environment as part of this thesis are also discussed and applied to the example. The simulation of the controlling statecharts subjected to different possible events is included through diagrams and logs. This example is also used to illustrate a procedure devised for coupling statecharts and activity diagrams in the specification of reactive systems and for using the software environment to verify this specification. The Fusion Method is compared to other methods for object oriented systems development. The reactive systems development environment based on statecharts is situated among other tools for the development of such systems.

Capítulo 1

Introdução

1.1. Considerações Iniciais

O desenvolvimento de sistemas exige do projetista, além do conhecimento das técnicas que possam apoiá-lo em diferentes fases do ciclo de desenvolvimento, que ele as empregue corretamente. Dessa forma, a elicitación dos requisitos que um sistema deve satisfazer é uma das tarefas mais árduas do engenheiro de software. Após a especificação, desenvolvimento e implantação do sistema requerido, sua manutenção deve ser realizada convenientemente. Nesse momento começam a surgir as dificuldades relacionadas com a implementação e documentação do sistema existente. Caso as pessoas que o desenvolveram estejam próximas, boa parte do tempo despendido o será em consultas e entrevistas com elas; caso contrário, a recuperação dessas informações torna-se um processo cansativo, enfadonho e propenso a erros.

A engenharia reversa de sistemas é uma área dentro da engenharia de software que apoia a recuperação das informações sobre sistemas existentes a fim de conduzi-los a uma reengenharia ou somente para que um modelo de análise desse sistema seja obtido, visando a tornar mais amena a tarefa de manutenção. A grande dificuldade nessa tarefa é o entendimento do sistema que deve ser ou está sendo mantido, pois envolve, normalmente, a pesquisa direta ao código fonte, entendimento de possíveis "truques" usados por parte dos programadores quando da implementação do sistema e possíveis problemas que possam surgir quando se deseja realizar alterações.

Sistemas desenvolvidos há algum tempo, mas que ainda atendem aos requisitos de seus usuários, podem ser convertidos para outras linguagens utilizando o recurso de um processo de engenharia reversa. A maioria dos sistemas desenvolvidos há longo tempo não o foi com o apoio de métodos existentes para especificação de sistemas e, dessa forma o trabalho de manutenção é maior. Os métodos de especificação de sistemas podem servir de apoio ao processo de engenharia reversa e mesmo à fase de engenharia avante quando a decisão for por uma reengenharia do sistema.

Este trabalho visa a dar condições para que sistemas herdados possam ser modelados através da abordagem de orientação a objetos, propondo um método de engenharia reversa que a partir da implementação de sistemas desenvolvidos sem a abordagem de orientação a objetos possibilite a construção de um modelo orientado a objetos para esse sistema.

1.2. Trabalhos Anteriores

A especificação de sistemas através de técnicas e métodos existentes na literatura técnica especializada tem sido objeto de interesse da autora desde os anos 80. Vários trabalhos foram desenvolvidos ao longo desses anos tanto no nível de estudos de casos de sistemas exemplos quanto na participação em pequenos projetos envolvendo problemas reais que atingem a comunidade universitária. Esses trabalhos, comentados sucintamente a seguir, ilustram a experiência anterior da autora e o interesse na área de engenharia de software que culminará com este trabalho.

Inicialmente, uma comparação entre algumas técnicas/métodos pôde ser realizada, [28], partindo-se da especificação de um mesmo sistema exemplo em cada uma delas [65, 66, 67 68]. O sistema foi especificado segundo Gane [26], HIPO [40], SOP [38, 39] e PSL/PSA [91], entre outros métodos e a comparação constou tanto dos aspectos conceituais envolvidos em cada uma das técnicas como também dos aspectos de representação gráfica dos conceitos utilizados em cada uma delas.

Com o surgimento da abordagem operacional para o desenvolvimento de sistemas, o método JSD [44] foi objeto de estudo e da elaboração de alguns trabalhos. Entre esses destaca-se a dissertação de mestrado [69], que apresenta um exemplo desenvolvido passo a passo nesse método, elucidando os detalhes desde a especificação até a implementação do sistema. O sistema PSL/PSA foi utilizado para documentação dos métodos JSP e JSD, possibilitando através de seus diferentes relatórios a visualização de diversos aspectos do método [71]. Em [70] foram utilizados os conceitos existentes no método JSP [43] para a especificação dos passos do método JSD [44].

Alguns sistemas foram desenvolvidos utilizando-se o método de Análise e Projeto Estruturado para atender às necessidades existentes em diversos setores da Universidade Federal de São Carlos, quer de atendimento a alunos, em problemas específicos existentes na coordenação de cursos [53, 54, 78] quer de âmbito mais geral, como a organização de normas, regimentos, etc. [52]. Esses sistemas foram implementados com apoio de gerenciadores de banco de dados relacionais e alguns estão ainda em operação.

Considerando-se a evolução das técnicas para especificação de sistemas e o grande interesse em bem especificar sistemas de tempo real, foram estudados métodos consagrados na literatura técnica especializada como DARTS [27], Esquemas de Transformações [94] e SA/RT [37]. Tomando-se como exemplo um sistema de controle de velocidade de cruzeiro e realizando sua especificação nos métodos acima citados e em JSD, elaborou-se uma tabela comparativa entre eles apresentando-se seus pontos fracos e fortes [83].

1.3. Motivação

O ambiente StatSim [22, 3, 6, 9] necessita de expansão e manutenção, porém esse trabalho estava se tornando impraticável devido à inexistência de uma documentação completa e integrada, contendo todas as informações necessárias para essas atividades. Seu desenvolvimento ocorreu de 1989 a 1993 e a documentação existente encontra-se espalhada em diversas dissertações de mestrado, relatórios de iniciação científica e artigos publicados.

Na tentativa de dar manutenção ao sistema e realizar pequenas alterações relacionadas com a sua funcionalidade [63, 64] pôde-se observar que a obtenção de informações resumia-se à pesquisa ao código fonte, documentos citados e quando possível às pessoas que escreveram determinadas partes do código.

Após a pesquisa realizada na literatura especializada pôde-se notar a ausência de diretrizes seguras para a realização do processo de engenharia reversa em sistemas herdados. Alguns pesquisadores citam superficialmente como deve ser o procedimento para a obtenção de informações a partir de códigos herdados existentes, mas não são claros quanto ao procedimento concreto e detalhado que deve ser realizado.

Coleman quando apresenta o método Fusion, dedica uma seção à reengenharia de sistemas e afirma que através de seu método pode-se realizar a reengenharia de sistemas desenvolvidos sem o uso de orientação a objetos para obter modelos de análise baseados na orientação a objetos [17]. Não apresenta, entretanto, nenhuma evidência que corrobore essa afirmação nem apresenta uma abordagem específica para isso. Jacobson, por outro lado, apresenta passos que podem ser seguidos para que a reengenharia seja realizada, mas não entra nos detalhes de como o processo de engenharia reversa deve ser realizado [45].

Paralelamente, notou-se o interesse por parte da comunidade científica e profissional na recuperação de sistemas desenvolvidos há alguns anos, quer simplesmente para troca da linguagem de implementação, quer pela dificuldade de manutenção, quer para ampliação da sua funcionalidade de acordo com paradigmas atualmente em uso. Esse interesse pôde ser observado tanto em artigos na literatura especializada como através de congressos nacionais e internacionais, tais como: Simpósio Brasileiro de Engenharia de Software, International Workshop on Computer-Aided Software Engineering - Working Conference on Reverse Engineering e International Conference on Software Maintenance.

Considerando-se os aspectos anteriormente mencionados, o interesse do grupo de Engenharia de Software do ICMSC - USP em expandir o ambiente StatSim e as dificuldades encontradas para sua manutenção, despertou o interesse em aprofundar as pesquisas nessa linha. Aliado a tudo isso há

também o interesse da autora pelo tema, uma vez que atua na área de engenharia de software em seu ambiente profissional.

1.4. O Problema a Ser Resolvido

A investigação de diretrizes que orientem o processo de engenharia reversa é o enfoque principal deste trabalho. Em especial diretrizes para a engenharia reversa de sistemas herdados, usando os conceitos e a notação do método Fusion, para análise e projeto orientados a objetos.

Assim, este trabalho preocupa-se em investigar e propor passos seguros e bem definidos para que um sistema desenvolvido sem a abordagem de orientação a objetos possa ser recuperado no nível de modelagem da análise, usando um modelo orientado a objetos. Dessa forma, utiliza para apoio ao método de engenharia reversa aqui proposto o método Fusion, desenvolvido originalmente para o projeto de sistemas e que é baseado na abordagem orientada a objetos.

Este trabalho tem por objetivo, adicionalmente, contribuir para a expansão do ambiente StatSim a fim de que seja possível a edição e simulação de: transição "default" com ação, evento nulo e atividades. Pretende-se também mostrar como o ambiente StatSim pode ser integrado a um processo sistemático de desenvolvimento de sistemas reativos, contribuindo para a validação de especificações comportamentais baseadas em statecharts.

1.5. Organização da Tese

Esta tese está organizada da forma descrita a seguir. No Capítulo 2 é realizada uma revisão de artigos existentes na literatura especializada, cujos temas são relacionados com sistemas reativos e métodos para especificação de tais sistemas, uma vez que o ambiente StatSim é voltado para edição e simulação de statecharts, que é uma técnica para especificação comportamental de sistemas

reativos. Nesse capítulo também são revisados os artigos que tratam de orientação a objetos e alguns métodos para o desenvolvimento de sistemas segundo essa abordagem. Algumas das técnicas existentes para reengenharia e engenharia reversa são analisadas, para embasar este trabalho, que propõe um método de engenharia reversa orientada a objetos.

No capítulo 3 apresentam-se os conceitos da técnica statecharts através de alguns exemplos, o ambiente StatSim e a evolução por ele sofrida ao longo dos anos. Versão é a denominação utilizada para cada extensão realizada. Um exemplo proposto por Coleman [16] é utilizado a fim de mostrar a utilidade do ambiente StatSim para validação de sistemas reativos. O sistema exemplo é especificado tanto funcional como comportamentalmente, editado e simulado no ambiente StatSim. Isso tornou-se possível devido à inserção de facilidades para edição e simulação de atividades no ambiente. É proposto um processo sistemático para que a especificação realizada possa ser validada com o apoio do ambiente StatSim.

Os principais aspectos do método Fusion para desenvolvimento de sistemas com a abordagem orientada a objetos e o método de engenharia reversa orientada a objetos proposto são apresentados no Capítulo 4. O método de engenharia reversa é completamente descrito, detalhando-se as tarefas que devem ser realizadas em cada um dos passos. Algumas métricas que permitem a verificação do esforço de conversão de um sistema não orientado a objetos para um sistema orientado a objetos são também propostas.

Um estudo de caso utilizando o ambiente StatSim e o método de engenharia reversa orientada a objetos proposto são o enfoque do Capítulo 5. Cada um dos passos do método é amplamente discutido e aplicado ao exemplo, fornecendo ao leitor uma visão detalhada das tarefas realizadas. As métricas definidas anteriormente são aplicadas ao sistema exemplo e as conclusões apresentadas.

O Capítulo 6 apresenta as conclusões gerais, as contribuições desta tese e sugestões para a realização de trabalhos futuros.

O Apêndice 1 apresenta a gramática utilizada no ambiente StatSim; os Apêndice 2 e 3 apresentam os Modelos de Objetos para os temas considerados e que não foram exibidos no corpo da tese, respectivamente, para o Modelo de Análise do Sistema Atual (MASA) e para o Modelo de Análise do Sistema (MAS). O Apêndice 4 apresenta as opções existentes na interface do ambiente StatSim, para os modos de edição e simulação, com os procedimentos que as implementam.

Capítulo 2

Resenha Bibliográfica

2.1. Considerações Iniciais

Neste capítulo são discutidos os recursos conceituais e instrumentais utilizados na pesquisa descrita nesta tese. Na verdade, essa pesquisa refere-se ao trabalho desenvolvido visando a estabelecer diretrizes detalhadas para a realização de engenharia reversa orientada a objetos em um sistema desenvolvido sem usar tecnologia orientada a objetos.

Os conceitos relacionados com sistemas reativos e os principais métodos que apoiam a especificação de tais sistemas são apresentados, bem com uma breve comparação entre os métodos mais utilizados. No Capítulo 3 utiliza-se statecharts para especificar um sistema reativo e o ambiente StatSim para verificar a especificação realizada.

Com o alto custo no desenvolvimento de novos sistemas tem-se optado por usar a técnica de reengenharia para recuperar as informações existentes nos sistemas mais antigos para, a partir dessa recuperação, expandir o sistema. Algumas vezes esse processo é interrompido na fase de engenharia reversa onde recuperam-se as informações do modelo, sem ocorrer o processo de alteração das funções realizadas pelo sistema.

A pesquisa bibliográfica realizada consta de artigos existentes na literatura técnica especializada. Esses foram divididos por assuntos e dentro de

cada assunto, procurou-se comentá-los segundo a ordem cronológica que foram publicados.

Assim, na seção 2.2 são apresentados conceitos de orientação a objetos e na seção 2.3 conceitos de sistemas reativos e abordagens existentes para desenvolvimento de sistemas reativos em termos de especificação, de ambientes de apoio por computador e métodos que se originaram já com apoio de ferramentas computadorizadas. Na seção 2.4 discutem-se alguns métodos para desenvolvimento de sistemas com orientação a objetos. Na seção 2.5 aborda-se a reengenharia de sistemas e na seção 2.6 a engenharia reversa de sistemas. Na seção 2.7 são apresentadas as considerações finais.

2.2. Orientação a Objetos

Na tentativa de se buscar maior facilidade na programação dos sistemas, as linguagens orientadas a objetos têm atraído grande atenção, uma vez que oferecem vantagens sobre as linguagens convencionais de programação, tais como: encapsulamento, abstração de dados e herança. Pressman comenta que no fim da década de 80 o “paradigma orientado a objetos” começava a surgir [79]. Nessa época alguns programas foram elaborados utilizando linguagens e técnicas de programação orientadas a objetos. Porém, a análise orientada a objetos surgiu mais recentemente. Coad e Yourdon justificam que essa demora em se aplicar os conceitos de análise orientada a objetos ocorreu devido ao fato de se estar demais ocupado em “seguir o fluxo” durante o apogeu da Análise Estruturada, uma vez que os conceitos de objetos, classes, atributos e métodos são inerentes ao conhecimento desde a idade pré escolar [79].

A abordagem orientada a objetos busca facilitar a manutenção, a reusabilidade e a extensibilidade do código/sistema. Programas desenvolvidos segundo a orientação a objetos são mais fáceis de serem lidos e entendidos do que os desenvolvidos segundo a abordagem tradicional. O controle da complexidade do programa pode ser conseguido através da hierarquia funcional

sobre os detalhes do programa e a ocultação de detalhes que não são necessários ao programador em determinado momento.

Define-se por objeto algo sobre o qual se tem interesse em modelar, algo que contém informação e tem um determinado comportamento em um sistema. Podem-se citar alguns exemplos de objetos: uma conta bancária e um cliente são objetos relevantes para um sistema de contas correntes em um banco; um carro é um objeto em um sistema de controle de locadoras de veículos, etc.. Em resumo, os objetos são encontrados de acordo com o ambiente que se deseja modelar e eles são caracterizados pelas suas identificações e informações associadas.

As informações sobre um objeto são dadas pelos seus atributos e o uso de um atributo é dado pelo seu comportamento ou pelos serviços que ele pode oferecer a outros objetos em seu ambiente. Os serviços podem ser caracterizados como as "ferramentas" que outros objetos usam quando desejam algo de um objeto. Um objeto utiliza algo de outro através das mensagens que são enviadas de um para outro. Os objetos encapsulam tanto atributos quanto serviços na sua própria estrutura, isto significa que toda informação é empacotada sob um nome e pode ser reusada como uma especificação ou componente do programa. Esses serviços recebem o nome de métodos na terminologia de orientação a objetos.

Uma classe é formada por um conjunto de objetos similares. A classe descreve os métodos que o objeto pode oferecer, as diversas atividades associadas a cada método e os diferentes atributos que cada objeto tem. Um objeto executa um método de acordo com a definição desse em sua classe. Uma classe pode herdar uma ou mais classes, o conceito de herdar significa que uma classe pode reusar os conceitos da outra, em termos de métodos e atributos. A reusabilidade permite que programadores descrevam em termos de objetos partes do sistema que posteriormente podem ser utilizadas em outras aplicações. O conceito de herança permite que objetos sejam facilmente estendidos com novas características sem a necessidade de duplicar o código. Um objeto filho é declarado herdeiro de todas as propriedades de seu objeto pai, sendo necessário

definir somente aquelas propriedades que diferem de seu objeto pai, ou seja, que lhes são próprias.

Um Modelo de Objetos de um sistema consiste de um conjunto de classes e um conjunto de associações. Pela associação de conhecimento é que um objeto conhece o outro e, pela associação de comunicação, um objeto de uma classe pode usar um objeto de outra, isto é, enviar mensagens para outro objeto.

O conceito de encapsulamento é importante em orientação a objetos significando que os atributos de um objeto estão contidos dentro dele. As informações armazenadas em um objeto são obtidas através da utilização de seus métodos. Os valores dos atributos são completamente escondidos do ambiente e dos outros objetos do sistema.

Jacobson [47] enumera os seguintes fatores positivos da análise orientada a objetos:

1) compreensão: pois torna-se possível quebrar hierarquicamente as classes, obtendo-se um entendimento completo do sistema que está sendo modelado;

2) entendimento: o sistema é descrito em termos de objetos os quais frequentemente têm uma ligação direta com o mundo real;

3) alterações: mudanças são feitas localmente para uma dada classe. Não há necessidade de afetar outras classes do modelo;

4) adaptabilidade: é possível especializar, com a ajuda dos mecanismos de herança existentes nas classes. Um modelo pode servir a diferentes situações através de adaptação para classes mais abstratas;

5) reusabilidade: classes podem ser construídas e manuseadas como componentes. Quando novas classes são criadas, propriedades de classes já definidas podem ser reusadas.

Muitos pesquisadores têm desenvolvido técnicas/métodos para apoiar a especificação de sistemas sob a abordagem de orientação a objetos para diferentes tipos de sistemas, alguns resultados dessas pesquisas são comentados na seção 2.4, bem como os trabalhos relacionados à aplicação dessas técnicas a diferentes tipos de sistemas.

2.3. Sistemas Reativos e Abordagens para o seu Desenvolvimento

Um sistema reativo é caracterizado por ser orientado a eventos, isto é, estar continuamente respondendo a estímulos do meio externo e interno. Há dificuldade em se descrever o comportamento reativo de maneira clara e realística e ao mesmo tempo formal e rigorosa [19, 34].

Sistemas reativos contrapõem-se aos outros sistemas devido à necessidade de que a resposta aos estímulos vindos do meio externo seja dada dentro de um intervalo de tempo preestabelecido. Caso isso não ocorra danos drásticos podem resultar, uma vez que esse tipo de sistema está fortemente acoplado ao ambiente externo. Em um sistema bancário, quando um cliente deseja saber o seu saldo e o sistema apresenta algum problema fazendo com que a resposta ao cliente demore mais do que o habitual, nada acontece diretamente com a vida do cliente, a não ser um atraso na realização de suas atividades já programadas. Em um sistema reativo, por exemplo, no engarrafamento de substâncias altamente tóxicas, a demora na resposta a um evento vindo do ambiente externo pode resultar em uma explosão da garrafa e/ou espalhamento do líquido, afetando vidas humanas. Alguns exemplos de sistemas reativos são: controle de aquecimento residencial, gerenciamento de atividades de robôs, controle de relógios, controle de caldeiras, redes de comunicação, sistemas de telefonias, etc..

Um sistema reativo tanto reage quanto afeta o ambiente em que está operando [2]. Assim, pode-se falar em sistema controlado, formado pelo hardware das máquinas que irão compor a parte do sistema reativo que se

comunica com o ambiente externo, e sistema controlador, formado pelo software juntamente com seu hardware.

Acredita-se que o modo mais natural para representar o comportamento reativo de um sistema complexo seja através da utilização de estados e eventos, que são elementos básicos de Máquinas de Estados Finitos [8]. Muitas vezes, porém, o uso dessas máquinas é inviável por serem planas, não permitirem visão de profundidade, hierarquia e modularidade. Nesse sentido outras técnicas foram estudadas a fim de suprir as dificuldades encontradas quando da utilização de máquinas de estados finitos.

O desenvolvimento de sistemas tempo real, um classe especial de sistemas reativos, deve ser realizado com o apoio de técnicas que especifiquem suas características. Entre as diversas técnicas existentes algumas são derivadas da abordagem funcional, tendo o método de Análise/Projeto Estruturado [100] ou Análise Essencial [59] como sua base. Dentro dessa classe podem ser citadas as técnicas de Gomaa, Ward e Hatley.

Em seu artigo [27], Gomaa, antes de apresentar o método DARTS aborda alguns métodos clássicos existentes na literatura, como JSP [43] e Warnier [61] que são baseados na estrutura dos dados. Diz que não se aplicam a sistemas reativos por não suportarem a especificação do fator tempo, relevante em tais sistemas. Structured Design [100] utiliza a decomposição em módulos, pode conduzir a um projeto modular altamente funcional mas não fornece ajuda na estruturação do sistema em tarefas e é frequentemente utilizado juntamente com Análise Estruturada [26]. HOS [31] usa decomposição funcional para modelar o sistema em módulos, mas também é falho quanto à divisão de um sistema em tarefas concorrentes. Information Hiding [62], apresenta um conceito poderoso de projeto que conduz a sistemas altamente modulares com módulos fracamente acoplados, porém não possui recursos para especificar o fator tempo. MASCOT, [86], é bem adaptado para Sistemas de Tempo Real por tratar especificamente da estruturação do sistema em tarefas e definir as interfaces entre elas. Esse método não será tratado nesta tese por não ser derivado dos conceitos de Análise e Projeto Estruturado e não ser amplamente utilizado como os demais aqui citados.

O método DARTS - Design Approach for Real Time Systems, tem cinco passos a serem seguidos para o desenvolvimento de um sistema de tempo real. A partir dos requisitos funcionais do sistema, o Diagrama de Fluxo de Dados é analisado e determinam-se as principais funcionalidades do sistema [26]. Os Diagramas de Fluxo de Dados são desenvolvidos e decompostos até que se tenha um detalhamento suficiente do problema. No próximo passo deve-se decidir se um dado processo deve ser estruturado como uma tarefa separada ou agrupado com outros processos em uma única tarefa. Ou seja, nesse passo identificam-se as tarefas do sistema segundo determinados critérios. O terceiro passo cuida da interface entre as tarefas, que pode ser de dois modos: Módulo de Comunicação entre Tarefas (TCM) e Módulo de Sincronização de Tarefas (TSM). A comunicação entre as tarefas é de dois tipos: com a tarefa produtora enviando mensagem e não recebendo confirmação ou recebendo confirmação da tarefa receptora. No quarto passo, projeta-se o Diagrama de Fluxo de Dados para cada tarefa. O último passo, consiste no desenvolvimento do Diagrama de Estrutura Modular [100] para cada tarefa com as interfaces existentes entre elas, uma vez que dentro da tarefa os processos são executados sequencialmente.

Um outro método para modelar os aspectos comportamentais das transformações de dados em função do tempo e de controle é conhecido por Esquema de Transformação. Foi criado por Ward, [94] e é uma extensão do método de Análise Estruturada/Projeto Estruturado [26], cobrindo as lacunas nele existentes, tais como: a representação de controles e de tempo. Esse método possui notação e regras de formação bem definidas, o que permite representar os sistemas como uma rede de transformações potencialmente concorrentes e de depósitos de dados interligados pelos fluxos de comunicação. Além disso, tem regras de execução que descrevem as entradas aceitas e as saídas produzidas pelas transformações.

Ao Diagrama de Fluxo de Dados convencional são adicionados os conceitos de fluxo de controle e transformação de controle. Uma Máquina de Estado Finito associada à transformação de controle também é incorporada ao Esquema de Transformação. Os estados dessa máquina representam períodos de tempo durante os quais o comportamento do sistema é fixo. São expostas as regras de execução do esquema de transformação baseadas na execução de uma

rede de Petri [77]. A execução do esquema é visualizada através da colocação de marcas, o que indica que uma atividade está ativa ou está potencialmente ativa. No artigo, um exemplo é apresentado ilustrando o método e algumas extensões são consideradas para a modelagem de certos problemas como a minimização do número de estados a ser utilizado.

Hatley propõe um método semelhante ao de Ward, conhecido por SA/RT (Structured Analysis/Real Time), [35]. A especificação de um sistema reativo é realizada tanto ao nível de fluxos de dados como de fluxos de controle. Dessa forma é proposto que para todo diagrama de fluxo de dados (DFD) seja também criado um diagrama de fluxo de controle (DFC). Os DFDs são criados até que se obtenha um completo entendimento do sistema, o mesmo ocorrendo com os DFCs. Máquinas de Estados Finitos são utilizadas para controlar o comportamento dos estados. Esse método abrange, também, a parte correspondente à arquitetura do sistema, apresentando um conjunto de técnicas para que essa modelagem possa ser realizada.

A linguagem Statecharts [32, 33, 34] permite a especificação de sistemas reativos, sendo uma extensão de Máquinas de Estados Finito. Os conceitos de paralelismo, propagação e hierarquia de estados são incorporados aos conceitos de estado e transição, possibilitando a representação comportamental de sistemas reativos complexos. Maiores detalhes da linguagem statecharts são fornecidos no Capítulo 3.

O ambiente Statemate [35] é um conjunto de ferramentas para especificação, análise, projeto e documentação de sistemas reativos complexos e grandes, com forte embasamento gráfico e que está disponível comercialmente desde 1987. Esse ambiente possibilita a especificação inter-relacionada da visão estrutural, funcional e comportamental, apoiada por editores gráficos. Cada uma dessas visões é baseada em linguagens com sintaxes e semânticas bem definidas. A visão estrutural é apoiada por diagramas de módulos ("module charts"); a visão funcional é apoiada por diagramas de atividades ("activity charts") e a visão comportamental por statecharts. O ambiente também possui ferramentas para teste, geração de código, geração de documentos, etc..

A visão estrutural fornece a decomposição hierárquica do sistema através de seus componentes físicos, chamados de módulos, e identifica a informação que flui entre eles, isto é, os dados e ou sinais de controle. A representação gráfica utilizada nessa visão é mostrada na Figura 2.1.

Figura 2.1 - Notação do Diagrama de Estrutura Usada no Statemate

Figura 2.2 - Representação Gráfica para Diagrama de Atividades

A visão funcional mostra a decomposição hierárquica das atividades (funções) que o sistema deve realizar, juntamente com detalhes de dados e sinais de controle que fluem entre as atividades e atividades de controle que especificam o comportamento temporal. O diagrama de atividades é utilizado para modelar funcionalmente o sistema porém sem especificar a sua parte dinâmica. Esse diagrama não se preocupa em dizer quando e porque uma atividade ocorre, se

ela termina ou não por conta própria ou se existem atividades sendo realizadas em paralelo. Os controladores é que dão a visão comportamental do sistema. A representação gráfica dos componentes dessa visão são os mostrados na Figura 2.2.

A visão comportamental descreve como e porque as coisas acontecem no sistema, especificando o controle das atividades. Os controladores podem aparecer em qualquer nível de hierarquia das atividades para controlar aquele nível em particular. A linguagem gráfica utilizada é statecharts, auxiliando na especificação do comportamento dos sistemas. O statechart de controle pode iniciar e parar atividades, pode gerar novos eventos e trocar os valores das variáveis. A representação gráfica utilizada nessa visão é mostrada em detalhes no Capítulo 3, Seção 3.2.

A Tabela 2.1 apresenta um resumo dos principais métodos para especificação de Sistemas Reativos e do ambiente Statemate.

Smith e Gerhart [88] discutem o uso do ambiente Statemate frente a quatro métodos apresentados em [87]. Esses métodos podem ser categorizados de acordo com duas abordagens distintas: orientação a objetos e decomposição funcional. Cada uma tem dois métodos de projeto: um concentrando-se em análise funcional do problema e outro em análise comportamental do sistema. Os passos para o desenvolvimento de um sistema seguindo cada uma das metodologias são apresentados. O uso de statecharts e diagramas de atividades existentes no ambiente Statemate permitem ilustrar como se comporta o sistema-exemplo. Dois exemplos são utilizados pelas autoras para ilustrar os métodos apresentados: Sistema de Controle de Velocidade Cruzeiro e um Sistema Bancário simplificado. O primeiro sistema foi desenvolvido utilizando o método de decomposição funcional, enquanto que o segundo, utilizou-se o método de decomposição funcional orientado a objetos.

Tabela 2.1 - Características dos Métodos/Ambiente Citados para Especificação de Sistemas Reativos

	DARTS	SA/RT (Ward)	SA/RT (Hatley)	STATEMATE
Fases do ciclo de vida que o método cobre	Da análise à implementação	Da análise à implementação	Da análise à implementação	Análise e Implementação Geração de código e realização de testes
Facilidade de entendimento pelo usuário	Necessário conhecimentos de Análise e Projeto Estruturado	Necessário conhecimentos de Análise, Projeto Estruturado, Máquinas de Estados Finitos	Necessário conhecimentos de Análise e Projeto Estruturado, Máquinas de Estado Finito, Máquinas Sequenciais	Necessário conhecimento de Máquinas de Estados Finitos
Ferramenta de Apoio	Não há	Existe, comercialmente	Existe, comercialmente	Existe, comercialmente
Notação Gráfica	DFD, Diagrama de Tarefas, Diagrama de Estrutura Modular. Formas de Comunicação	Esquemas de Transformação, Diagrama de Transição de Estado. Fluxos de Dados, Fluxos de Controle	DFD, DFC, Diagrama de Contexto de Dados, Diagrama de Contexto de Controle, Máquina de Estados Finitos, Máquina Sequencial, Modelo de Arquitetura	Diagrama de Módulos, Diagrama de Atividades, Statecharts
Representação do Tempo	Apenas implícita	Através de regras de execução	Máquinas Sequenciais	Nos statecharts através de eventos especiais, time-out
Visão Funcional	DFDs	DFDs, refinamentos sucessivos	DFDs, refinamentos sucessivos	Diagramas de Atividades (Activitycharts)
Visão Estrutural	Diagrama de Tarefas	Embutida no Esquema de Transformação	Comunicação entre os módulos do Diagrama de Controle	Diagrama de Módulos (Modulecharts)
Visão Comportamental	Diagrama de Transição de Estado	Durante a construção do Projeto Estruturado	Diagrama de Fluxo de Controle	Statecharts

Os quatro métodos disponíveis para se criar um diagrama de atividade e o statechart correspondente são: decomposição funcional,

decomposição comportamental, decomposição funcional orientada a objetos e decomposição comportamental orientada a objetos.

Diagrama de Atividade de A

Statechart C

onde: $E_i/st!(F_i)$ significa: evento E_i ocorreu e a atividade F_i teve início
 $sp(F_i)$ significa que a atividade F_i parou de ser executada

Figura 2.3 - Decomposição Funcional

A decomposição funcional identifica as principais funções do sistema. Nesse ponto o sistema é visto como uma atividade em um único nível. Projeta-se o statechart de controle para esse nível, caso ele não seja expressivo assume-se que todas as sub-atividades especificadas são concorrentes. As

principais funções são representadas como sub-atividades do sistema e devem ser especificadas e controladas por um statechart. As atividades têm o seu próprio statechart controlador e são decompostas em sub-atividades. A cada nível de decomposição deve-se especificar a interface e o fluxo de dados entre as atividades. A Figura 2.3 mostra um esboço do sistema especificado pela decomposição funcional. No diagrama de atividades, A representa o sistema, C o controlador do sistema, F_i , $i=1,2,\dots$, as atividades e C_i os controladores das atividades F_i . No statechart C os estados são denotados por S_i e as transições são rotuladas pelo evento/ação e têm a semântica descrita na Figura 2.3.

A decomposição funcional orientada a objetos identifica os principais objetos do sistema, projetando uma atividade para cada objeto físico ou abstrato. A atividade raiz corresponde ao sistema todo. São identificados os atributos de cada um dos principais objetos do sistema, as informações que podem ser importadas ou exportadas a partir de um objeto. A representação desses atributos pode ser feita de duas formas: colocando-se os depósitos de dados no diagrama de atividades ou colocando-se estados no statechart controlador da atividade referente àquele objeto. A seguir, identificam-se: os métodos sobre os objetos que são representados como sub-atividades de um particular objeto e, quais as atividades ou sub-atividades do sistema que estão recebendo ou enviando as informações identificadas anteriormente (atributos). Define-se um statechart controlador para cada atividade que representa um objeto. Isso irá especificar como os métodos são ativados dentro de um objeto. Os sub-objetos dos objetos existentes também devem ser identificados como atividades. Esses sub-objetos são dependentes do objeto gerador, aquele que os gerou. Dessa forma, o statechart do objeto gerador é que deve representar quando os sub-objetos estão ativos. Isso deve ser repetido até que cada um dos sub-objetos tenha sido detalhado suficientemente.

A Figura 2.4. mostra um esboço do diagrama de atividade do sistema quando especificado através da decomposição funcional orientada a objetos. O objeto B representa todo o sistema com dois objetos principais OB1 e OB2 que não têm statechart controlador pois considera-se que eles operem independentemente. OB_i , $i=1,2 \dots$, tem os métodos especificados por OPI e seu comportamento é representado pelo statechart controlador Controle- OPI . OB1 tem

sub-objetos OB1A e OB1B e dois métodos, OP1A e OP1B.; enquanto que OB2 tem um único sub-objeto OB2A e dois métodos OP2A e OP2B. Controle-OP1 e Controle-OP2 são os controladores dos objetos OB1 e OB2, respectivamente. Deve-se observar que os subobjetos podem ser refinados de modo que cada um tenha sub-objetos e sub-operações próprias.

Diagrama de Atividade de B

onde: OB_i representa o objeto i

OP_i representa o método do objeto i

Figura 2.4 - Decomposição Funcional Orientada a Objetos

Na decomposição comportamental as atividades básicas do sistema são listadas em um único nível e o statechart de controle contém todas as informações hierárquicas. As atividades devem ser, por essa abordagem, decompostas tanto quanto possível. O statechart deve ser organizado para refletir essa decomposição hierárquica. A Figura 2.5 mostra um esboço do sistema especificado pela decomposição comportamental. As subfunções do sistema denotadas por FiX , para $i=1,2$ e $X=A, B, C$, representam as atividades de mais alto nível do diagrama de atividade do sistema A. O statechart controlador é C-NOVO, que é análogo ao statechart C da Figura 2.3 (Decomposição Funcional), com a diferença de que aqui os estados S1 e S2 foram substituídos por C1-

Modificado e C2-Modificado. Assim, na Decomposição Comportamental o estado C1-Modificado é detalhado nesse nível enquanto que na Decomposição Funcional esse detalhamento ocorre no próximo nível, quando se constrói o controlador da atividade F1.

Diagrama de Atividade de A

Statechart Controlador C-NOVO

Figura 2.5 - Decomposição Comportamental

A decomposição comportamental orientada a objetos tem os mesmos princípios da decomposição funcional orientada a objetos. A diferença básica é que a decomposição hierárquica se manifesta somente no statechart. O diagrama de atividade tem um único nível e todos os objetos que compõem o sistema são especificados no nível-raiz. O único statechart controlador construído

é composto de vários estados ortogonais. Esses estados podem ser decompostos considerando os métodos dos objetos. A Figura 2.6 mostra um esboço do sistema especificado pela decomposição comportamental orientada a objetos. Nesta figura os objetos rotulados por OB_i e os métodos OPI , $i=1,2,\dots$, representam o diagrama de atividade de nível mais alto. O statechart CC captura todos os aspectos de controle. As atividades de controle OPI , $i=1,2$, são componentes ortogonais do estado controlador de todos os objetos.

Diagrama de Atividade de B

Statechart Controlador CC

Figura 2.6 - Decomposição Comportamental Orientada a Objetos

Smith e Gerhart afirmam que essas abordagens têm suas vantagens e que frequentemente é desejável combiná-las, citando algumas possibilidades: 1) uso casado: quando a simulação for importante, o método

comportamental pode ser usado depois de se encontrar um certo nível de detalhe com a utilização do método funcional; 2) usá-las de forma alternada: quando a característica mais importante do sistema é o controle do nível mais alto usa-se o método comportamental primeiro, sendo o método funcional usado para especificar os detalhes das funções que envolvem esse nível. 3) criar dois projetos separados, primeiro produzindo um projeto funcional usando as vantagens da abordagem “top-down” e a seguir o projeto comportamental pela transformação do projeto funcional. As autoras alertam que para sistemas grandes, a transformação pode ser extremamente difícil e o auxílio de mecanismos automatizados pode tornar-se necessário.

2.4. Métodos Orientados a Objetos Para Desenvolvimento de Sistemas

O método de Coad e Youdon [14] foi primeiramente desenvolvido para apoiar a fase de análise, sendo posteriormente estendido para a fase de projeto [15]. A identificação das abstrações é realizada através dos seguintes passos: identificar as classes, objetos, estruturas e assuntos; definir os atributos e definir os serviços. Esses passos não são necessariamente sequenciais e podem ser gradualmente ampliados através de mais e mais detalhes. No nível de projeto o Gerenciamento de Dados, Interações Humanas e Componentes de Gerenciamento de Tarefas também são considerados como atividades. Os resultados obtidos na fase de análise são utilizados para a fase de projeto sem adicionar novos elementos à notação. Novas entidades podem ser adicionadas durante o projeto como um refinamento das entidades existentes, tal como novas classes e objetos.

As atividades de projeto no método de Booch [7] seguem quatro questões: a) quais são as classes existentes e suas relações, b) quais os mecanismos usados para controlar a colaboração entre os objetos; c) onde as classes e objetos devem ser declarados, d) quais processos devem ser associados ao processador e como são planejados os múltiplos processos de um processador. Para responder a essas questões existem quatro diferentes

diagramas, que são: Diagrama de Classe, Diagrama de Objeto, Diagrama de Módulo e Diagrama de Processo. Os Diagramas de Classe e de Objeto fornecem a visão lógica do sistema e os outros diagramas correspondem à visão física do sistema.

O método OMT [81] é dividido em três fases: análise, em que se modela o mundo real; projeto, em que se decide sobre a arquitetura geral e dos subsistemas e, implementação, em que se codifica o projeto em uma linguagem de programação orientada a objetos. O modelo de análise é composto de três modelos e recomenda-se que eles sejam produzidos nessa ordem: Modelo de Objetos, Modelo Dinâmico e Modelo Funcional. O fato deles serem realizados nessa ordem não exclui a iteração natural para o desenvolvimento dos três modelos, seu refinamento e verificações entre eles.

A fase de projeto, como o próprio nome diz, constrói o projeto a partir do modelo de análise anteriormente elaborado [81]. Dois estágios compõem essa fase: Projeto do Sistema e Projeto do Objeto. O Projeto do Sistema quebra o sistema em subsistemas alocando processos para os processadores. O Projeto do Objeto cria os algoritmos para os métodos. Os modelos de análise são combinados para determinar os métodos de cada classe. As representações de dados são escolhidas e os grafos de herança são ajustados para refletir as decisões do projeto. O projeto é então particionado em módulos a serem convertidos para uma determinada linguagem de programação orientada a objetos.

A fase de implementação converte o projeto em código. São fornecidas diretrizes de como os diferentes aspectos do projeto podem ser codificados em diferentes linguagens orientadas a objetos. O método também recomenda como as classes devem ser projetadas visando a sua reutilização.

Outro método disponível para especificação de sistemas com orientação a objetos é o Objectory [46], com três fases: solicitações, análise e construção. O termo “*agente*” é empregado com a finalidade de cobrir todo usuário humano ou mecânico do sistema. Objectory é baseado no modelo de “*use-case*”, que é um diálogo realizado entre o sistema e o usuário com a finalidade de

capturar a funcionalidade do sistema, representando o que os agentes podem fazer. Todos os outros modelos utilizados nesse método são construídos considerando os “*use-cases*” e fornecem uma ligação entre as fases nele existentes.

O ponto de partida para a definição da funcionalidade do sistema é a obtenção dos requisitos através de linguagem natural que é utilizada para construir três modelos: Modelo de “*Use-Case*”, Modelo de Domínio de Objetos e Descrição da Interface do Usuário. O Modelo de “*Use-Case*” deve ser documentado por uma descrição textual que mostra a seqüência de eventos e explica como é a participação dos objetos do Modelo de Domínio de Objetos. Esse modelo é que define os conceitos com os quais o sistema trabalha. As Descrições da Interface do Usuário são esboços das janelas que os usuários irão ver na tela.

A fase de análise refina os modelos de requisitos a fim de produzir uma descrição ideal do sistema. O modelo de análise é baseado no modelo entidade-relacionamento. Ele é construído gradualmente considerando o quê os objetos e as classes precisam em cada “*use-case*”.

A fase de construção refina o modelo de análise. A comunicação entre objetos é mais precisamente definida e são consideradas as características do ambiente da implementação, produzindo-se quatro modelos: 1) Modelo do Blocos: exibe os blocos do sistema, que são formados por um ou mais objetos da fase de análise; 2) Diagramas de Interação: mostram como os blocos participam de cada “*use-case*”; 3) Interfaces de Bloco: mostra a interface operacional de cada bloco e 4) Especificação de Bloco: é um modelo opcional que mostra o comportamento interno do bloco através de máquinas de estado.

Os sistemas reativos precisam de métodos especiais para o seu desenvolvimento, como foi visto anteriormente, devido ao fator tempo. Dessa forma Coleman apresenta um método que denominou de Objectcharts [16], cuja proposta é a utilização de statecharts juntamente com o paradigma de orientação a objetos. Os conceitos de statecharts são estendidos de modo que cada estado representa o comportamento do objeto de uma determinada classe. As transições correspondem aos métodos que a classe fornece e aqueles que ela utiliza de

outras classes. Coleman e outros aplicam a proposta de Objectcharts para o exemplo de um relógio despertador, que é apresentado na seção 3.5.1. Os serviços (métodos) necessitam de argumentos que são considerados como parâmetros de entrada e de saída. Os parâmetros de entrada somente podem ser lidos pelo objeto que está fornecendo o serviço e os parâmetros de saída são os valores que retornam a esse objeto. Dentre as abordagens apresentadas na seção 2.4, sugeridas por Smith e Gerhart, Objectcharts enquadra-se na abordagem funcional orientada a objetos.

Figura 2.7 - Influências que o Método Fusion Sofreu de Outros Métodos

O método Fusion [17], criado por Coleman e outros, reúne o que segundo esses autores, os métodos OMT [81], CRC[95], Booch [7] e os métodos formais têm de melhor. A Figura 2.7 ilustra a contribuição de cada um dos métodos citados para a elaboração do método Fusion, que cobre a fase de análise, projeto e implementação de sistemas. Para cada uma dessas fases existe um conjunto de ferramentas facilitando a sua modelagem. A fase de análise, que tem o método OMT como base, oferece três modelos: o de Objetos, de Operação e de Ciclo de Vida para a especificação do sistema. A fase de projeto foi baseada em CRC e Booch e é composta dos seguintes modelos: Interação de Objetos, de Visibilidade, de Descrição de Classes e de Herança. A fase de implementação é realizada com apoio dos gráficos elaborados na fase de projeto. Maiores detalhes desse método podem ser encontrados no Capítulo 4, seção 4.2 desta tese.

Pode-se dizer que há algo em comum entre o método Fusion e Statecharts quanto a forma como foram elaborados, ou seja, ambos partem de resultados consagrados no desenvolvimento de sistemas: os pontos fortes dos métodos de desenvolvimento de sistemas segundo o paradigma de orientação a objetos e a extensão dos conceitos de máquinas de estados finitos, respectivamente. De acordo com as abordagens apresentadas na seção 2.4., o método Fusion pode ser classificado como uma abordagem funcional orientada a objetos devido aos passos bem delineados que apresenta para a especificação funcional de um sistema.

MOOD - Modelling approach to Object-Oriented Design [48], é um método criado por Lee e outros para o projeto de sistemas de software orientados a objetos que emprega as técnicas de refinamento de entidades, modelagem funcional, comportamental e dos objetos estáticos para modelar os objetos do domínio da solução. Esses modelos são integrados entre si. MOOD tem um conjunto de notações gráficas que descreve a estrutura, funções e relações cliente-servidor entre as entidades e cuja notação pode ser mapeada diretamente para a programação orientada a objetos na fase de implementação. O artigo apresenta uma tabela onde são listadas as principais características do método e uma tabela comparativa das técnicas de modelagem desse método e as de Booch, Rumbaugh, Coad e Yourdon, Shlear e Mellor, Wirfs-Brock, HOOD, GOOD e Análise Estruturada. Como MOOD é criado para a fase de projeto os modelos apresentados cobrem somente essa fase, não se preocupando com a fase de análise. Nessa tabela, os autores citam o apoio que cada um dos métodos, por eles considerados, oferece na fase de projeto da arquitetura, de objeto, dinâmico e funcional. MOOD oferece apoio em termos de projeto de arquitetura e funcional utilizando diagramas de fluxo refinado (RFDs). Em termos de projeto dinâmico nenhum diagrama é oferecido; para o projeto de objetos propõe o diagrama refinado do Modelo de Objetos (OMD). Apesar de Booch também não oferecer modelos para apoiar a fase de análise é o único método que cobre todos os quatro aspectos considerados na fase de projeto. Através dessa tabela pode-se observar também que o método de Coad e Yourdon oferece modelos de apoio para os quatro aspectos da fase de análise: arquitetura, objeto, dinâmico e funcional; porém, somente diagramas de modelo de objetos é que estão disponíveis para a

fase de projeto relacionada com os objetos. Rumbaugh, por sua vez, propõe para apoiar a fase de análise os modelos: de objetos, dinâmico e funcional.

Duntemann e Marinacci [21] discutem a conversão de uma aplicação existente para uma orientada a objetos. Salientam que um dos fatores a ser considerado deve ser o de custos versus benefícios, pois podem existir bibliotecas de códigos mais antigos que não podem ser ligados com os novos módulos desenvolvidos de acordo com a programação orientada a objetos. Alguns tipos de aplicação que a conversão para uma aplicações orientada a objetos é impossível ou muito trabalhosa são as não estruturadas, as com excesso de variáveis globais e as com "loops" implementados por GO TOs. Os autores comentam que a conversão deve ser realizada passo-a-passo e fornecem diretrizes para realizá-la: a) tratar dos aspectos da programação antiga que tem mais chance de se tornarem objetos, tais como, as estruturas de dados ou famílias de estruturas de dados, procedimentos e funções que atuam sobre essas estruturas; b) identificar o objeto central dentro de uma aplicação e estabelecer um plano hierárquico orientado a objetos, visando a futuras aplicações. Isso envolve o planejamento em alto nível da interface com o usuário, sistema de ajuda, tutoriais de acesso imediato e subsistemas reusáveis; c) com a hierarquia em mente, deve-se retornar à aplicação que está sendo convertida e identificar a estrutura de dados central.; d) desenvolver subsistemas orientados a objetos para futuras aplicações e tentar adicioná-las às aplicações que estão sendo convertidas. Os autores sugerem que antes de realizar o processo de conversão o engenheiro de software desenvolva uma aplicação completa seguindo a abordagem de orientação a objetos.

Losavio e outros apresentam uma comparação entre as notações semânticas dos métodos de Coad e Yourdon e de Booch [55]. Para tal utilizam um sistema de controle de estoque, onde a análise é realizada utilizando o método de Coad e Yourdon e o projeto é especificado utilizando o método de Booch. Uma tabela comparativa discutindo as notações utilizadas pelos métodos é apresentada, sendo o enfoque dado para: representação de classes e objetos, relacionamentos entre classes, instâncias das classes, mensagens de sincronização, linguagens de programação, gerenciamento de complexidade, atributos e métodos. Dessa comparação pode-se observar que Coad e Yourdon

representam juntos classes e diagramas de objetos enquanto que Booch o faz separadamente, desde que representem dois diferentes contextos semânticos. Entre os relacionamentos de classes, em Coad e Yourdon encontram-se os conceitos de herança, todo-parte e associação; em Booch, os conceitos são os mesmos com a terminologia: herança, tem, usa, metaclasses e instanciação. Com relação aos atributos e métodos, Coad e Yourdon especificam no Diagrama de Classe somente aqueles que são relevantes, sendo que todos aparecem explicitamente na especificação textual da classe. Booch, não especifica os atributos e métodos no Diagrama de Classe, eles aparecem explicitamente na especificação textual da classe; alguns métodos são mostrados no Diagrama de Objeto. Após o estudo realizado os autores concluem que o método de Booch oferece um conjunto mais rico de notação gráfica, sendo capaz de expressar com mais detalhes o Modelo de Objetos, o que permite também melhor refinamento do modelo. A semântica de Coad e Yourdon é mais dirigida à análise enquanto que Booch, é mais dirigido ao projeto. Justificam que o sistema-exemplo desenvolvido segundo a abordagem de Coad e Yourdon pôde ser facilmente adaptado para a abordagem de Booch, salientando que os dois métodos estão sendo enriquecidos com novos elementos.

livari comenta os trabalhos realizados por vários pesquisadores na comparação de métodos orientados a objetos, citando o enfoque dado em cada um dos artigos [41] Ele traça um paralelo entre esses trabalhos anteriormente desenvolvidos e o seu, salientando que está comparando seis métodos de análise orientada a objetos mostrando as diferenças entre eles para a modelagem funcional e comportamental. Justifica que a escolha dos métodos foi realizada levando em conta que cada método selecionado tem um livro que o descreve, cobre a fase de análise orientada a objetos e deve incluir uma abordagem razoavelmente específica para a fase de análise. Desse modo, os métodos selecionados e comparados foram: OOA/OD de Coad e Yourdon, OOSE de Jacobson e outros, OOAD de Martin e Odell, OMT de Rumbaugh e outros, OOSA de Shlaer e Mellor e DOOS de Wirfs-Brock e outros.

Antes de iniciar a comparação entre os métodos, livari comenta que existem duas abordagens para o entendimento da modelagem orientada a objetos: depende se os aspectos do sistema em desenvolvimento são estáticos ou

dinâmicos e se o ponto de vista é de um objeto ou de um conjunto de objetos. Ele apresenta uma tabela onde os aspectos estruturais, funcionais e comportamentais são listados para o ponto de vista de objeto individual e de um conjunto de objetos. Assim, para a visão estrutural tem-se objeto individual: classe, tipo e atributos do objeto enquanto que para o conjunto de objetos tem-se: relacionamento, herança, composição/agregação e subsistema. A visão funcional engloba os conceitos de método, operação e serviço somente para objetos individuais, enquanto que a comportamental é composta de transição de estado para objetos e mensagem/solicitação para conjuntos de objetos.

A comparação entre os métodos é realizada apresentando as características gerais de cada um quanto às abstrações funcionais, estruturais e comportamentais. O autor conclui que muitas similaridades e diferenças foram encontradas entre os métodos. Os métodos são similares quanto à modelagem estrutural mas diferem consideravelmente na modelagem funcional e comportamental. Após esse estudo foi possível identificar os pontos fortes e fracos e os problemas de cada um, que livari também apresenta em forma de tabela. Cita que o método de Coad e Yourdon é o mais simples, mais fácil de ser aprendido e aparentemente o mais fácil de ser aplicado. O ponto forte da análise orientada a objetos de Jacobson e outros é o conceito de “*use case*”, que permite uma melhor exploração da funcionalidade do sistema do que as classes individuais. A fase de análise do método OMT permite quase que uma completa especificação do sistema, cobrindo a modelagem estrutural e deixando a desejar em relação à funcional e comportamental.

O método Fusion é adicionado aos métodos apresentados por livari a fim de enriquecer a comparação exibida em [41]. Alguns aspectos, referentes à fase de análise, serão considerados para a comparação entre os métodos como as abstrações de estrutura, de funções e de comportamento. Por abstração de estrutura entende-se a descrição de objetos estáticos relevantes ao domínio em questão, seus relacionamentos, atributos e possíveis estados. A abstração de funções descreve as transformações de entrada-saída e a de comportamento descreve a instanciação e a dinâmica das transformações em relação ao tempo. Toda comparação é realizada sob o ponto de vista do objeto individual e da comunidade de objetos. A parte referente à descrição do método em comparação

aos demais é omitida para o método Fusion, uma vez que esse pode ser encontrado brevemente descrito nesta seção e com maiores detalhes no Capítulo 4 desta tese.

A Tabela 2.2 apresenta a abstração de estrutura na segunda linha para os *Objetos individuais* e na terceira linha para *Comunidades de objetos*.

Tabela 2.2 - Abstração de Estrutura

Coad and Yourdon	Jacobson	Martin and Odell	Rumbaugh	Shaler and Mellor	Wirfs-Brocks	Fusion
Objetos, classe, atributos	Objetos, atributos	Tipo de objeto	Objeto, classe, atributos	Objetos, instancia, atributos	Classe, atributos	Objetos, classe, atributos
Generalização Especialização Estrutura todo-parte assunto instancia	Associação, Subsistema	Relaciona- mento, Generaliza- ção, Composi- ção	Generaliza- ção, Agregação, Associação e ligação, Módulo Folha	Domínio, Subsistema Relaciona- mento Subtipo/ Supertipo	Subclasse, Superclasse Relação "é parte de", Subsistema	Generaliza- ção, Agregação, Herança, Relaciona- mento, Cardinalida de no relaciona- mento

A Tabela 2.3 apresenta abstração de funções, na segunda linha para os *Objetos Individuais* e na terceira linha para a *Comunidade de Objetos*.

Tabela 2.3 - Abstração de Funções

Coad and Yourdon	Jacobson	Martin and Odell	Rumbaugh	Shaler and Mellor	Wirfs-Brocks	Fusion
Serviço	Responsabi- lidade Operação	Operação	Operação Ação/ativida de Função	Ação Processo	Objetivo Responsa- bilidade Contrato Protocolo	
	Use-case			Responsabi- lidade do subsistema	Objetivo Responsa- bilidade Contrato (todos p/ subsistema)	Modelo de Operação

A Tabela 2.4 apresenta abstração de comportamento, na segunda linha para os *Objetos Individuais* e na terceira linha para a *Comunidade de Objetos*.

Tabela 2.4 - Abstração de Comportamento

Coad and Yourdon	Jacobson	Martin and Odell	Rumbaugh	Shaler and Mellor	Wirfs-Brocks	Fusion
Transição de Estado		Transição de Estado		Transição de Estado		
Conecção de mensagem	Use-case Assoc. comunicação	Evento Disparo Regra de Disparo Condição de controle	Cenário Evento Traço de evento/fluxo Valores de entrada/saída	Pontes Eventos Acessor	Colaboração Contrato Mensagem	Cenário, Modelo de Ciclo de Vida

O Método Fusion não tem diferenças significativas em relação aos demais quando comparado em relação à abstração de estrutura como pode ser visto na Tabela 2.2. Em relação à abstração de funções, esse método não oferece apoio para os objetos individuais, somente para a comunidade de objetos utilizando o Modelo de Operação. O mesmo ocorre em relação à abstração de comportamento, pois não cuida desse aspecto para os objetos individuais. Tanto o tratamento do comportamento como das funções consideram os relacionamentos e herança entre classes de objetos e não cada classe em separado. A parte comportamental dos objetos é tratada na fase de projeto através dos modelos existentes, mostrando a comunicação das mensagens entre os objetos e, portanto, os métodos.

2.5. Reengenharia de Sistemas

A reengenharia de sistemas antigos para a arquitetura orientada a objetos é a proposta de Jacobson e Lindström [45]. Eles têm por objetivo mostrar como um método orientado a objetos pode ser usado gradualmente para modernizar sistemas antigos não desenvolvidos segundo essa abordagem.

Mostram cenários de reengenharia relacionados com a orientação a objetos, sendo esses baseados em experiências reais das quais participaram.

Inicialmente os autores definem reengenharia como sendo composta de engenharia reversa + Δ + engenharia avante. Citam que o primeiro passo deve ser o da engenharia reversa é realizada uma definição mais abstrata e mais fácil de ser compreendida da representação do sistema. A segunda parte, a que chamaram de Δ , contém modificações tanto de funcionalidade do sistema como de implementação. A terceira e última fase refere-se à engenharia avante, que é o desenvolvimento normal de sistemas, criando-se uma representação que seja executável em alguma linguagem de programação, no caso, em uma linguagem orientada a objetos.

Para a fase de engenharia reversa são sugeridos três passos: 1) um grafo que descreva concretamente as componentes do sistema e suas interrelações; 2) um grafo abstrato, livre de detalhes da implementação, que mostre o comportamento e a estrutura do sistema; 3) um mapeamento entre os dois anteriores, isto é, como é o relacionamento de algo que existe no grafo abstrato com o grafo concreto e vice-versa.

Os cenários existentes para reengenharia são: a) troca completa da implementação sem troca da funcionalidade; b) troca parcial da implementação sem troca da funcionalidade e c) troca da funcionalidade.

Os passos para troca completa da implementação sem troca da funcionalidade são: 1- preparar um modelo da análise; 2- mapear cada objeto da análise para a implementação do sistema antigo; 3- reprojetar o sistema usando a abordagem orientada a objetos e 4- implementar o modelo da análise.

Os principais passos para realização da reengenharia parcial sem troca de funcionalidade são: 1- identificar as partes do sistema que serão reimplementadas usando técnica orientada a objetos; 2- Preparar um modelo de análise da parte a ser trocada e seu ambiente; 3- Mapear cada objeto para a implementação antiga do sistema; 4- Repetir os passos anteriores até que a interface entre a parte a ser trocada e a parte restante do sistema existente seja

aceitável; 5- Em paralelo: 5.1 - Projetar o novo subsistema e suas interfaces para o que falta do sistema antigo; 5.2 - Modificar o sistema antigo e adicionar uma interface ao novo subsistema e 6- Integrar e testar o novo subsistema e o sistema antigo modificado.

Para se realizar a reengenharia com troca de funcionalidade o cenário é o mesmo que para o processo de engenharia avante. Adicionam-se modificações da funcionalidade ao modelo de análise e implementa-se o modelo modificado usando técnicas de orientação a objetos. Os principais passos são: 1. Modificar o modelo de análise de acordo com os requisitos e 2- Projetar o sistema.

Wilkening e outros apresentam um método para reengenharia de sistemas de uma linguagem para outra, [97]. Inicialmente os estudos concentraram-se na reengenharia de sistemas aviônicos codificados em Fortran e que devem ser convertidos para Ada. O ambiente em desenvolvimento é projetado para apoiar outras linguagens de programação. Os autores têm investigado as técnicas de reengenharia e engenharia reversa existentes e baseado nesse estudo desenvolveram uma modelagem de projeto que define reengenharia em termos da construção de um novo programa, reutilizando o projeto e a implementação do programa original. Enfatizam que esse processo é consistente com modelo de reengenharia tradicional de Chikofsky e Cross [12]. A ferramenta de reengenharia chamada RET que construíram, pode desenvolver um programa em Ada reutilizando partes originais do projeto e da implementação escritas em Fortran. Eles consideram que as ferramentas de reengenharia existentes podem ser classificadas em duas categorias, abaixo descritas, enquanto que a RET pode ser considerada como uma reengenharia apoiada por computador:

a) Engenharia Reversa e Redocumentação - apresentam diferentes visões da estrutura de um programa, tal como gráficos de fluxo de controle e de fluxo de dados, para apoiar o entendimento do programa e a reengenharia manual. Entre elas estão: [18], [87] e [96].

b) Outras ferramentas - apoio automático para passar de uma linguagem para outra ou reestruturar automaticamente, por exemplo, eliminando "GO TOs". Essas ferramentas exigem um pouco de interação humana mas

fornece pouco apoio para a recuperação e melhoria do projeto. Entre elas pode-se citar: [84] e [99].

Semmel e Winkler apresentam um estudo sobre reengenharia de grandes sistemas de informação a partir de integração de bases de dados [85]. Os resultados da reengenharia relacionados ao projeto e fusão de bases de dados são discutidos e um sistema de consulta e projeto conhecido por QUICK é descrito. O exemplo utilizado refere-se ao processo de engenharia reversa aplicado a esquemas lógicos de duas bases de dados do Armamento dos Estados Unidos, gerando um modelo conceitual, que é posteriormente utilizado para geração de consulta automática e fusão de dados. Os autores afirmam que QUICK facilita a engenharia reversa a partir de esquemas lógicos em esquemas conceituais do modelo Entidade-Relacionamento Estendido [Che76]. Eles justificam que esse sistema tem habilidade para formular consultas eficientemente fornecendo mecanismos para avaliar tanto o esquema conceitual novo como o anterior, que passou pelo processo de engenharia reversa. No caso do sistema exemplo adotado, QUICK demonstrou que duas partes da base de dados logicamente disjuntas mas conceitualmente associadas podem ser fundidas e consultadas.

O artigo de Sage [82] apresenta uma visão geral e algumas perspectivas de sistemas e gerenciamento de sistemas que podem sofrer reengenharia, bem como abordagens relacionadas à revitalização organizacional e técnica envolvendo tecnologia de informação e engenharia de software. Considera-se que a reengenharia pode ser de três tipos: ao nível de produto, de processo e de gerenciamento de sistemas e que todos os três são necessários. Ele apresenta de forma gráfica algumas abordagens para reengenharia e propõe que toda entidade que passe pelo processo de reengenharia deve ser: definida, desenvolvida e avaliada constantemente.

Reengenharia de produto é definida como o exame, estudo, captação e modificação dos mecanismos internos ou da funcionalidade de um sistema existente ou de um produto a ser reconstituído sob nova forma ou com novas características. Frequentemente leva em consideração novas tecnologias mas não há maiores mudanças significativas na funcionalidade e no objetivo do

sistema. Como exemplo, pode-se considerar o esforço realizado para converter um código herdado escrito em Fortran ou Cobol para C++ ou Ada.

Por reengenharia de processos consideram-se os níveis do processo e o gerenciamento de sistemas. O esforço, nesse caso é quase que totalmente interno. Ele consiste de modificações para melhor acomodar o ciclo de processos que está em uso em uma organização, em termos de novas tecnologias ou novas exigências do usuário. Basicamente, a reengenharia de processo consiste da determinação de um processo eficaz para fornecer um produto que atenda às exigências genéricas do cliente, considerando-se os objetivos e capacidades críticas da engenharia de sistemas da organização.

A reengenharia para a área de gerenciamento de sistemas é o exame, estudo, captação e modificação dos mecanismos internos ou da funcionalidade dos processos de gerenciamento existentes e das práticas comuns na organização, a fim de reconstituí-los em uma nova forma e com novas características, levando-se em conta suas exigências competitivas.

Jacobson e outros [47] apresentam um livro sobre a reengenharia de negócios seguindo a abordagem de orientação a objetos. Os autores fazem considerações sobre a empresa e mostram um método para que o processo de reengenharia seja a ela aplicado. "Use-case" é a técnica utilizada para produzir e descrever os processos existentes dentro da empresa e que são candidatos à reengenharia. A modelagem de objetos reflete os objetos existentes no negócio. A iteração entre a construção do modelo de objetos e do modelo de "use-case" é chamada de engenharia reversa e o produto dessa fase é o modelo do negócio existente.

O modelo de "use-case" inicia-se com a idealização dos atores, identificados através dos diferentes tipos de clientes. Esse modelo pode ser somente referente a parte da empresa ou a ela como um todo. Para que um ator seja considerado deve-se levar em conta a sua participação no negócio. Após essa escolha deve-se estudar um conjunto de exemplos reais com quem a empresa tem contato, para se certificar de que todos aqueles que têm envolvimento com a empresa fazem parte do modelo, isto é, foram considerados

como atores. Em seguida deve-se preparar os “use-cases”. Deve-se verificar qual a ligação de cada ator com a empresa e modelar esse fato. Alguns comentários são feitos com relação ao número de “use-cases” encontrado, considerando que um detalhamento exagerado ocorre quando esse número ultrapassa trinta. Os autores sugerem algumas regras para que se priorize os “use-cases”

O modelo de objetos retrata um modelo do negócio em termos dos objetos de interação. Esses objetos são agrupados em subsistemas e os “use-cases” são realizados por esse objetos. Assim, após a identificação dos subsistemas deve-se novamente descrever os “use-cases” especificando como os objetos os realizam, para entender como os “use-cases” são manipulados pelo negócio. A análise desses resultados deve ser realizada para se ter a certeza de que os objetivos do sistema serão atingidos. Em seguida é sugerido um processo de engenharia avante composto de cinco passos: elaboração do modelo de “use-cases”, elaboração do modelo ideal de objetos, elaboração de um modelo real que consta da adaptação do modelo ideal às restrições encontradas no negócio, desenvolvimento de um sistema de informação e a verificação do novo negócio. Esses passos são dependentes uns dos outros e por isso devem ser realizados em paralelo.

O sucesso de grandes projetos que sofrem o processo de reengenharia está ligado ao seu planejamento, uma vez que não é fácil atualizar ou migrar centenas de programas e a base de dados sem perder o fluxo existente no serviço de processamento de dados [89]. Considerando os fatos acima, métricas são necessárias para garantir as melhorias a serem realizadas. A reengenharia em sistemas herdados requer muito esforço uma vez que abrange mudanças técnicas: transformação de linguagem, extração de objetos, realocação de funções e preservar a funcionalidade. Sneed propõe cinco passos para o processo de reengenharia começando com a análise do sistema herdado e terminando com um contrato de negócio. Esses passos devem justificar o projeto, priorizar as aplicações que são candidatas à reengenharia, fazer estimativa de custos, realizar análise custos versus benefícios e por último, gerar o contrato. Cada um dos passos é amplamente discutido e o autor encerra o artigo dizendo que não é fácil vender um sistema de reengenharia. Ele como gerente de uma pequena empresa de software depara-se constantemente com esse problema:

justificar o porque da reengenharia. Os clientes questionam o porquê em passar um sistema escrito em Cobol ou Fortran pelo processo de reengenharia se existem linguagens mais atraentes, tipo as de quarta geração ou orientadas a objetos.

2.6. Engenharia Reversa de Sistemas

A reengenharia de sistemas tem como primeiro passo a engenharia reversa, fato esse comentado pela maioria dos artigos sobre reengenharia. A engenharia reversa geralmente envolve a extração de artefatos e construções do projeto ou síntese de abstrações que são menos dependentes da implementação. Pode-se realizar engenharia reversa a partir de qualquer nível de abstração e para qualquer estágio do ciclo de vida. Ela mesma não envolve troca do sistema ou a criação de um novo sistema, é um processo de exame, não um processo de troca ou duplicação. Considerando as fases do ciclo de vida, a engenharia reversa começa da implementação existente, recapturando ou recriando o projeto e decifrando os requisitos atualmente implementados pelo sistema em estudo. Esta seção comenta alguns dos artigos existentes na literatura a respeito de engenharia reversa.

Os sistemas herdados são aqueles sistemas que estão em uso por muito tempo e são considerados insubstituíveis ou porque a reimplementação de suas funções tem um custo muito alto ou porque eles são imprescindíveis para seus usuários. Provavelmente esses sistemas foram implementados em uma linguagem procedimental convencional, com uso limitado de abstração de dados e encapsulamento. Dessa forma a falta de abstração complica a adição de novas aplicações para tais sistemas e a falta de encapsulamento dificulta modificações, pois as aplicações tornam-se dependentes do sistema interno. Dietrich e outros descrevem em [20] a experiência em utilizar e gerar uma interface orientada a objetos para um sistema herdado, GDP - Geometric Design Processor, para modelagem de sólidos, que foi escrito em PL/I.

Duas razões são dadas para implementar um sistema orientado a objetos usando um sistema herdado: 1- os sistemas herdados satisfazem a

maioria das necessidades dos usuários mas uma melhor programação de sua interface torna-se necessária para estender ou padronizar as suas funções e 2- o código desses sistemas pode ser reusado para implementar totalmente ou parcialmente as funções desejadas de um sistema orientado a objetos. O sistema TGMS - Tiered Geometric Modeling System é também descrito, fornecendo uma alternativa com interface orientada a objetos para a programação da interface do GDP. Eles afirmam que essa interface fornece um ambiente altamente produtivo para o desenvolvimento de novas aplicações. Assim, o TGMS foi construído por essas duas razões e uma importante melhora foi obtida: conseguiu-se isolar suas aplicações da modificação e substituir a base do modelador de sólidos. Os autores ilustram como um sistema herdado, específico para o tratamento de sólidos geométricos, pode ser usado como parte de um sistema orientado a objetos e concluem afirmando que vários sistemas legados podem ser integrados para construir um poderoso ambiente orientado a objetos.

Benedusi e outros descrevem um processo de engenharia reversa capaz de gerar automaticamente informações em nível de projeto e documentos diretos do código [4]. Eles mostram uma instanciação do paradigma organizacional de alto nível para reconstrução dos diagramas de estrutura de um sistema de software de acordo com o método de Yourdon e Constantine [100].

Os autores definem engenharia reversa como uma coleção de teorias, método e tecnologias capazes de apoiar: 1- o projeto e a implementação de um processo para extrair e abstrair informações a partir de software existente e produzir documentos consistentes com o código e, 2- a adição de conhecimento e experiência que não podem ser automaticamente reconstruídas a partir do código desses documentos. Os produtos obtidos do processo de engenharia reversa auxiliam na engenharia avante de forma a ajudar no desenvolvimento de reuso, teste e controle de qualidade de software. Eles citam que na Europa existem projetos que estão cuidando da definição, da fixação de metodologias e técnicas de engenharia reversa, pois consideram que o primeiro passo em um processo é a escolha, por parte da organização, sobre qual metodologia, técnica ou ferramentas são necessárias para o desenvolvimento do processo.

Eles acreditam que a definição e produção de um processo de engenharia reversa deve ser baseado no paradigma de objetivos/modelos/ferramentas. A fase correspondente aos objetivos inicia-se levantando-se quais as razões para aplicar o processo de engenharia reversa e termina com a definição dos documentos que devem ser produzidos. Devem ser elaborados nessa fase: Diagramas de Estruturas, Diagramas de Fluxo de Dados, Diagramas SADT, Diagramas JSP e JSD, Diagramas de Warnier-Orr, representações PDL, Grafos de Fluxo de Controle e Árvores Aninhadas.

A fase de modelos inclui: a) análise dos documentos produzidos, identificação da informação e relações que podem ser diretamente extraídas do código e que são direta ou indiretamente necessárias para produzir esses documentos, b) definição dos modelos de programa que representam a informação e as relações extraídas do código e c) a definição abstrata dos algoritmos que permitem a produção dos documentos desejados a partir desses modelos, via o processo de engenharia reversa. Os modelos considerados podem ser: Expressões Algébricas, Expressões Regulares, Matrizes de Chamadas, Matrizes de Variáveis/Procedimentos, Modelos Sintáticos e Semânticos.

Para o paradigma objetivos/modelos/ferramentas, os autores comentam que as ferramentas incluem: a) a definição, b) aquisição e/ou produção de ferramentas de software necessárias para produzir os documentos da engenharia reversa. Elas podem ser divididas em ferramentas extratoras, dependentes da linguagem, e ferramentas que abstraem informação, independentes da linguagem mas dependentes dos documentos e de métodos de projeto. Dentre as ferramentas são citadas: Analisadores Sintáticos e Dinâmicos (extratoras), Abstratores de Modelo, Manipuladores de Modelos e Ferramentas Gráficas.

Chikofsky e Cross II [12], comentam que existe confusão com relação à terminologia usada, tanto na área técnica como na área empresarial e propõem a definição de seis termos: *engenharia avante*, *engenharia reversa*, *redocumentação*, *recuperação do projeto*, *reestruturação* e *reengenharia*. O objetivo não é a criação de novos termos mas sim a racionalização e padronização dos termos que já vêm sendo usados. As definições resultantes são aplicadas em

um processo de engenharia em desenvolvimento sem considerar o grau de automação que foi aplicado. Citam que o termo “engenharia reversa” teve origem na análise de hardware por ser comum decifrar projetos a partir de produtos acabados. Alertam que a engenharia reversa está crescendo rapidamente e que tornar-se-á uma componente necessária e importante aos ambientes CASE. Pelo fato de todo o ciclo de vida ser uma atividade iterativa, as ferramentas de engenharia reversa podem fornecer uma maior ligação em todo o processo de desenvolvimento e manutenção. Afirmam que engenharia reversa, usada com tecnologias de software em desenvolvimento, pode fornecer ganhos significativos em termos de produtividade.

A engenharia reversa é definida em [13] como envolvendo primeiro a geração da descrição do projeto a partir da implementação e daí então a descrição da especificação. Choi e Scacchi propõem que sejam abstraídas quatro propriedades do sistema: 1) Estrutural: descrita pela troca de recursos entre módulos e submódulos através de interfaces interconectadas; 2) Funcional: descrita pela semântica dos recursos trocados; 3) Dinâmica: descrita pelos algoritmos procedurais que transformam os recursos importados em exportados (é uma propriedade intramodular); 4) comportamental: descrita pelo comportamento de seus objetos em termos de relações entre objetos, seus atributos e as ações que o manipulam (também é intramodular). O objetivo do artigo é extrair a propriedade estrutural e em segundo plano as propriedades dinâmica e funcional de grandes sistemas, compostos de módulos e subsistemas. Para chegar ao objetivo proposto utilizam uma linguagem de interconecção modular chamada de NuMIL, que vê as dependências entre os módulos com um caráter funcional e estrutural. Isso pode ser verificado através da análise estática. Este artigo é comentado com mais detalhes no Capítulo 4.

Leite e Franco propõe uma estratégia de reengenharia baseada no método JSD. Utilizam o sistema HiperLex como caso de estudo [49]. Os componentes do método JSD foram utilizados como forma de registrar a recuperação do projeto. Como o processo é de reengenharia, o primeiro passo é a construção do Diagrama de Implementação do Sistema (DIS). Para isso foram consideradas algumas heurísticas e suas justificativas, sendo que o código fonte foi utilizado como texto estruturado. Para a fase de modelagem primeiro foram

construídos os textos estruturados e a partir desses o Diagrama de Especificação do Sistema (DES). As entidades puderam ser facilmente encontradas, segundo o comentário dos autores, depois que essas fases anteriores foram realizadas. Para cada uma das fases são mostrados os diagramas para o sistema HiperLex.

Leite e outros apresentam a reengenharia do ambiente DRACO-PUC, enfocando seu analisador sintático. O método SADT é utilizado para apresentar a estratégia de reengenharia e também para a especificação de novas funcionalidades do sistema. Diagramas do método JSD são utilizados para representar a estrutura de procedimentos recuperados [50].

McCabe [58] afirma que a reengenharia de software tem crescido muito nos últimos anos tendo-se tornado comum entre os sistemas de informação nos anos 90, pois as empresas tentam conter os custos para manter os sistemas de software e maximizar seus investimentos em softwares existentes. Assim as empresas que precisam ampliar suas aplicações estão optando pelas técnicas de reengenharia a fim de recuperar seus investimentos no processo lógico de software existente. Comenta que engenharia reversa é o processo usado para construir um sistema utilizando técnicas estruturadas e que tem como ponto de partida a recuperação da lógica do sistema existente. Salaria que as aplicações que podem passar pelo processo de engenharia reversa são aquelas que foram criadas sem utilizar técnicas de modelagem como análise ou projeto estruturado ou aplicações em que o modelo original não foi mantido de acordo com o código fonte. As ferramentas automatizadas podem ajudar o engenheiro de software a entender como um sistema está construído fornecendo a documentação que está faltando para o nível de projeto. Têm a facilidade de representar centenas de linhas de código através de uma simples imagem gráfica. Engenharia reversa pode ser usada também para apresentar um modelo de análise que mostra o que a aplicação faz e, na prática, é o único modo de garantir que o sistema substituto terá toda a capacidade lógica do sistema original.

Chikofsky, dentro do artigo de McCabe, comenta sobre algumas das ferramentas que apoiam a fase de engenharia reversa: Bachman/Analyst e Bachman/Database Administrator que fornecem um conjunto eficiente de ferramentas para apoiar o entendimento e a transição de modelos de dados

implementados em DB2, IMS ou Cobol. Teamwork/DB Designer da Cadre Technologies Inc, fornece um conjunto de ferramentas para modelagem de dados. Para a leitura de código Cobol há uma série de ferramentas de apoio, com uma vasta gama de recursos disponíveis entre a leitura de código e a habilidade dos usuários em fazer algo útil com esse código lido. Application Browser e HyperCode Management System apoiam o usuário com uma coleção de ferramentas integradas para que o código possa ser entendido e aprendido para que se realizem trabalhos com ele. Afirma que o produto mais poderoso para linguagem Cobol é ESW (Existing Systems Workbench), que integra numerosas ferramentas de análise Cobol através da arquitetura de depósitos de dados. Battlemap e ACT da McCabe and Associates Inc. of Columbia e Logicscope Verilog Inc. devem ser consultadas quando o interesse está em visualização e análise de complexidade de código, uma vez que os “displays” e as métricas dessa ferramenta podem auxiliar tornando o código menos complexo através de seu particionamento. O principal objetivo das ferramentas da McCabe é também gerar caminhos de teste para software em várias linguagens, as quais permitem a integração dos esforços de engenharia reversa e estratégias de teste de software.

Guedes e Staa apresentam a reengenharia de sistemas sob o enfoque econômico, propondo mecanismos de redução de custos e aumento de qualidade. Todo o processo tem apoio de ferramentas computadorizadas sendo que a ferramenta CASE Talisman é utilizada para a reengenharia do sistema [30].

O entendimento de um programa que não é familiar ao engenheiro de software, ocorre de maneira informal através da reconstrução do processo de análise, descobrindo-se os conceitos utilizados e associando-os às estruturas de implementação existentes no programa. O problema em descobrir esses conceitos orientados para o entendimento humano e atribuir suas realizações dentro de um programa específico ou de seu contexto é chamado de “*problema de atribuição de conceito*”, Biggerstaff e outros, [5]. Existem, na prática, diversas ferramentas e estratégias para resolver esse problema com sucesso. Os autores ilustram algumas dessas estratégias através de exemplos de cenários e do sistema DESIRE (Design Information Recovery Environment), por eles criado. Esse sistema foi projetado para apoiar o entendimento do programa, tendo como idéia central que um modelo de reconhecimento orientado ao parsing baseado em

formalismo é necessário mas não suficiente para resolver o “*problema da atribuição de conceitos*”. São mostrados alguns cenários de como as ferramentas de apoio podem ser utilizadas com a finalidade de simplificar e acelerar a atribuição de conceitos.

O Departamento de Defesa dos Estados Unidos (DOD) tem muitos sistemas heterogêneos, formando desse modo bilhões de linhas de código. Frequentemente esses sistemas são duplicados devido a falta de organização e com isso dois problemas surgem, como relatados em [1]: a) o elevado custo operacional e b) a incapacidade de se obter informação correta a partir dos dados armazenados devido ao grande número de bases de dados sem padronização e sem referência. Em face do problema, acenou-se com a possibilidade dos sistemas herdados sofrerem o processo de engenharia reversa.

A engenharia reversa foi aplicada em diversas etapas, sendo cada uma delas discutida. O primeiro passo foi a extração dos modelos de dados lógicos de múltiplos sistemas de migração, bases de dados e arquivos para reduzir a redundância, manutenção e inconsistências em seus sistemas atuais. O produto resultante dessa etapa foi uma representação de dados necessária ao DOD para integrar os requisitos de dados. Em seguida estudou-se o impacto da engenharia reversa ao nível da dificuldades encontradas para a sua realização e no nível de complexidade tecnológica e administrativa. A obtenção de informações não documentadas ou de posse de pessoas não disponíveis para fornecê-las pode ser considerado como o maior problema para a realização da engenharia reversa. As complexidades tecnológicas envolvem os projetos de sistema, tecnologia e eletrônica obsoleta e na parte administrativa, não existe disciplina organizacional.

Um roteiro, incluindo quinze passos, que deve ser seguido e os produtos resultantes, em cada um deles, para que a engenharia reversa seja realizada é fornecido pelos autores. Sugerem que a análise da engenharia reversa deve ser utilizada após as modificações terem sido realizadas. Encerram o artigo dizendo que entre as lições aprendidas estão a dificuldade na obtenção de informações, pois as pessoas que as possuem estão constantemente sendo requisitadas para o desenvolvimento de outros sistemas e os custos da engenharia reversa, que nem sempre são bem dimensionados pelos gerentes. Os

autores têm sido constantemente questionados sobre os produtos da engenharia reversa: o quê foi produzido, como esses produtos estão sendo usados, como a engenharia reversa se relaciona com outras atividades de desenvolvimento, etc.. Para responder a essas questões apresentam uma tabela mostrando os produtos usados nas atividades de desenvolvimento da arquitetura do sistema.

Muitas vezes nas empresas os sistemas existentes inibem o crescimento do negócio e a capacidade de mudança. Um modo de resolver esse problema é substituir o sistema herdado por outro com um novo desenvolvimento que seja condizente com as novas necessidades. Porém essa alternativa é raramente realizada porque o sistema contém regras críticas do negócio que precisam ser usadas diariamente. Uma outra alternativa é encapsular o sistema herdado de modo que ele possa ser utilizado totalmente em um novo ambiente de execução ou dentro de um novo sistema. Essa alternativa tem uma série de atrativos mas também apresenta desvantagens, como a demora de adaptação do sistema à nova plataforma, pouca flexibilidade, o sistema herdado pode ser reusado como um todo e é impossível reusá-lo em partes. Ning e outros citam que o centro de tecnologia de pesquisas estratégicas da Andersen Consultoria construiu um conjunto de ferramentas voltados para o problema de identificar e extrair componentes de grandes sistemas herdados em Cobol [60]. Esse conjunto de ferramentas chama-se Cobol/SRE - Cobol System Renovation Environment e é o resultado de vários anos de pesquisa nessa área. Cada um dos componentes da ferramenta é apresentada e é mostrado como o usuário divide o sistema em segmentos, nome dado a trechos de código formado por um conjunto de programas.

Wong e outros afirmam que sistemas de software antigos exigem uma documentação diferente da tradicionalmente utilizada ou seja, algoritmo e estruturas de dados [98]. Em sistemas grandes a redocumentação estrutural através da engenharia reversa está se tornando a palavra chave. Reconstruir e efetivamente redocumentar o projeto do sistema de software existente é mais difícil do que o projeto inicial. Reconhecer as abstrações do sistema do mundo real é tarefa crucial tanto quanto projetar adequadamente abstrações para os novos projetos.

A engenharia reversa reconstrói modelos estruturais pela identificação dos componentes do sistema atual, descobrindo suas dependências e gerando abstrações para gerenciar complexidade; fornecendo “insights” que podem então melhorar o desenvolvimento subsequente, facilitar a manutenção e reengenharia e ajudar o projeto de gerenciamento. A redocumentação estrutural é engenharia reversa dos aspectos arquiteturais do software. Como um dos resultados dessa redocumentação, recuperam-se o objetivo geral do sistema e algumas das informações do projeto da arquitetura. Além disso, a redocumentação estrutural não envolve reestruturação do código fisicamente, embora isso pode ser uma saída desejável da fase subsequente da reengenharia.

Wong cita que na universidade de Vitoria foi desenvolvido um ambiente chamado Rigi, que enfoca os aspectos arquiteturais para entendimento de programas. O ambiente Rigi é composto por: 1- *Rigireverse* - um sistema de “parsing” que apoia linguagens de programação tipo C e Cobol e um parser para LaTeX, para analisar a documentação; 2- *Rigiserver* - um depósito para armazenar as informações extraídas do código fonte; 3- *Rigiedit* - um editor gráfico com interface baseada em janelas, interativo, para manipular as representações do programa. O ambiente apoia um método para identificar, construir e documentar subsistemas hierárquicos. Críticas ao uso do ambiente Rigi surgem quanto a sua facilidade para armazenar e recuperar as visões da engenharia reversa. Essas visões são usadas para transferir informações sobre abstrações para os engenheiros de software e os autores comprovaram que essas visões são compatíveis com os modelos mentais usados pelo engenheiros de software. Assim a documentação produzida por Rigi traz benefícios por: a) tornar concreta a estrutura de software previamente existente somente na mente dos projetistas; b) acentuar as áreas críticas da estrutura de software que necessitam de mais atenção; c) fornecer uma base objetiva para discussão e manutenção de software porque as visões são baseadas em código fonte atual, e não da forma como é a documentação de sistemas que, frequentemente, torna-se desatualizada quando os códigos evoluem, e d) verificar que as estruturas de software do sistema puderam ser, no mínimo, entendidas por um analista experiente externo ao sistema.

A abordagem de engenharia reversa desenvolvida pelo projeto Rigi foi aplicada com sucesso em vários projetos de software de tempo real: sistema de registro de informações médicas, escrito em Cobol; um programa controlador de um acelerador, escrito em C; e vários utilitários Unix. Antes do exemplo relatado no artigo, o ambiente Rigi foi aplicado a um programa contendo 120.000 linhas de código.

RESCUE tem como núcleo a Linguagem de Representação RESCUE. As expressões nessa linguagem podem ser geradas tanto pelo processo de engenharia reversa de sistemas herdados ou pelo uso de ferramentas de programação manual ou automática que possibilitam a inclusão tanto de novos projetos como de requisitos de implementação para sistemas que passam pelo processo de engenharia reversa. Zoufaly e outros mostram os resultados obtidos no processo de engenharia reversa [101]. Os autores assumem que é possível construir um sistema geral de tradução automática baseada em dedução. Demonstram essa hipótese através da conversão de programas IBM RPG II para programas Oracle V7 PL/SQL2 e Oracle FORMS3, que empregam as regras de dedução existentes em Logistica (linguagem utilizada em inteligência artificial que implementa o Princípio Fundamental da Dedução, o qual fixa que dedução é o processo de reescrever expressões preservando a equivalência). São mostrados o algoritmo geral de tradução e exemplos do código gerado no protótipo RESCUE. O programa de entrada é considerado como sendo sintática e semanticamente correto, embora o tradutor realize algumas checagens para assegurar a integridade do código. No futuro, programas RPG incorretos poderão ser aceitos. Os autores concluem que o projeto tem atingido os seus objetivos iniciais, com resultados promissores como demonstrado através da conversão automática de RPG para Oracle. Estudos estão sendo realizados a fim de possibilitar a expansão tanto das linguagens fontes como as destinos.

DECODE [80] é um protótipo de um ambiente em que o programador e o sistema auxiliam a obtenção do projeto orientado a objetos a partir de sistema herdado. Suas três componentes são: um programa automatizado (APU) que reconhece alguns elementos iniciais do projeto orientados a objetos; uma base de conhecimento para registrar as informações de projeto extraídas e um memorando estruturado para editar e consultar esse

projeto. A partir de um programa fonte, o programa automatizado (APU) cria uma base inicial de conhecimento. O programador usa o memorando estruturado para inspecionar o programa fonte e estender o entendimento parcial do sistema. Enquanto o programador reconhece a finalidade de algum segmento arbitrário de código, esse código pode ser selecionado e mapeado em algum objeto ou operação de acordo com a hierarquia orientada a objetos que esse programador criou. São mostrados como exemplos os resultados obtidos com a utilização do ambiente DECODE, sendo o programa fonte escrito em COBOL. Os autores justificam que a idéia principal do ambiente DECODE é que o programador e o sistema trabalhem juntos para o entendimento do software.

Gall e Klösch [25] comentam a dificuldade de manutenção de sistemas herdados e as vantagens existentes quando os sistemas são desenvolvidos com orientação a objetos. Afirmam que o sucesso da manutenção requer um conhecimento detalhado das partes do programa. Assim propõem que os programas procedurais herdados existentes possam ser transformados em orientados a objetos. A identificação de objetos e suas dependências auxilia o entendimento do projeto do sistema, permite voltar ao projeto original durante a manutenção e tem facilidade de reuso. Esses objetos são encontrados a partir do código fonte e do conhecimento do usuário. Citam que a partir dos diagramas de estruturas e diagramas de fluxo de dados os objetos são identificados e que o processo apresentado por Benedusi [4], pode ser utilizado. Mostram como os elementos que compõem o programa podem ser classificados em objetos e dois modelos que devem ser produzidos: a) RooAM - Reverse generated object-oriented Application Model, que é o modelo com os candidatos a objetos e seus relacionamentos e b) FooAM - Forward generated Application Model, que é baseado no programa procedural e é gerado independente da implementação, podendo ser desenvolvido utilizando qualquer um dos métodos de análise orientada a objetos existentes. São mostrados exemplos dessas transformações e é citado que esse método foi aplicado a sistemas industriais.

Sneed e Nyáry [89] dizem que a tendência atual é a transição de mainframes para workstations e que o pré-requisito para que isso aconteça é a adequação e documentação atualizada dos programas, o que é difícil em se tratando de sistemas herdados. O processo de transição pode ser feito de três

formas: a) início de uma nova análise e o desenvolvimento do projeto, o que acarreta altos custos e a dificuldade de se obter consenso entre os usuários; b) conversão direta dos programas a partir do código existente, como o projeto atual nem sempre é adequado para o novo ambiente, isso implica na alteração das estruturas; c) extraindo as estruturas do código herdado e criando uma nova especificação do projeto do programa que pode ser reimplementada em outra linguagem com outra arquitetura, porém preservando a sua funcionalidade. Com base nessa terceira hipótese o artigo apresenta uma abordagem para que se extraia automaticamente a documentação orientada a objetos do projeto a partir de programas COBOL existentes para mainframes. São apresentadas as diretrizes para a escolha dos objetos, considerando-se o modelo de base de dados utilizado na implementação, o tipo de linguagem, a interface, etc..Em seguida são extraídas as operações associadas aos objetos avaliando-se os procedimentos existentes. Object-Redoc é a ferramenta projetada para analisar os programas COBOL para produzir projeto orientado a objetos.

2.7. Considerações Finais

Conforme foi visto nas diferentes seções deste capítulo, procurou-se abordar os principais trabalhos existentes na literatura sobre o assunto da pesquisa apresentada nesta tese, a fim de relacionar os conceitos e instrumentos aqui utilizados, permitindo situar seus resultados junto aos de outros trabalhos já publicados.

Na seção 2.2 foram apresentados os conceitos de orientação a objetos necessários ao entendimento do método para a realização de engenharia reversa orientada a objetos em sistemas não desenvolvidos orientados a objetos como mostrado no Capítulo 4.

Assim, na seção 2.3 foram apresentados os principais conceitos sobre sistemas reativos necessários para a melhor compreensão do ambiente StatSim bem como as principais abordagens para especificação desse tipo de sistemas.

A seção 2.4 apresentou alguns métodos para desenvolvimento de sistemas orientado a objetos posicionando os conceitos do método Fusion frente aos de outros métodos existentes e que são os mais utilizados para desenvolvimento de sistemas sob essa abordagem.

Na seção 2.5 apresentaram-se os conceitos de reengenharia, úteis ao posicionamento da engenharia reversa que foi detalhada na seção 2.6, de modo a melhor se poder apreciar o método de engenharia reversa de que trata esta tese.

A maioria dos artigos preocupa-se com a reengenharia de sistemas de negócios. Pode-se observar que a sequência de passos a ser seguida na engenharia reversa de sistemas independe do tipo do sistema. A partir dessa revisão foi possível maior integração com os objetivos do processo de engenharia reversa para que o método de engenharia reversa orientada a objetos pudesse ser elaborado conforme mostrado no Capítulo 4 deste trabalho.

Capítulo 3

O Ambiente StatSim

3.1. Considerações Iniciais

A especificação de Sistemas Reativos requer técnicas capazes de expressar, principalmente, as características de tempo. Statecharts [32] é uma das técnicas existentes que apoia o desenvolvimento e validação de tais sistemas, possuindo sintaxe e semântica bem definidas [33].

Se a especificação puder contar com ferramentas computadorizadas de apoio tem-se enormes vantagens sobre a técnica manual. Um dos ambientes de apoio ao desenvolvimento de sistemas reativos é STATEMATE [35]. O grupo de Engenharia de Software do ICMSC-USP desenvolveu um sistema de software similar em ambiente Unix para edição e simulação de Statecharts chamado StatSim (Statechart Simulator). Esse sistema foi desenvolvido como um protótipo ao longo de cerca de cinco anos (1989-1993) a partir de projetos de mestrado e iniciação científica. A versão atual, que se encontra em forma de protótipo operacional desde 1993, possui cerca de 30000 linhas de código C e, além do código fonte, conta com descrições sobre o projeto espalhadas por várias dissertações, relatórios e artigos [22, 3, 56, 23, 24, 92, 93, 57, 10]. Há interesse do grupo em evoluir o StatSim de forma mais organizada e foi eleita a abordagem orientada a objetos tanto para o desenvolvimento de novos módulos quanto para uma eventual mudança para ambiente DOS/Windows.

A especificação de sistemas utilizando o ambiente StatSim fica facilitada, pois ele permite que statecharts sejam editados e simulados. Com a

simulação de um statechart pode-se observar se o modelo especificado reflete os requisitos do sistema e caso isso não ocorra o ambiente de edição permite que se corrija os problemas existentes. Depois de realizadas essas correções, o modelo pode ser novamente simulado e assim sucessivamente até ter-se confiança de que a especificação satisfaz os requisitos do sistema.

O ambiente StatSim possuía inicialmente somente recursos textuais para a especificação de sistemas. Depois foi ampliado de modo a possibilitar a representação gráfica na edição e simulação de statecharts. O modo de simulação permitia somente a simulação interativa [56]. Nessa simulação o usuário fornece, passo a passo, os eventos que devem ser disparados e através da mudança de cor dos estados na tela pode-se visualizar o comportamento do sistema. Alguns trabalhos foram desenvolvidos permitindo-se que outros tipos de simulação pudessem ser realizadas. Assim, ambiente foi estendido para permitir simulação batch [93], programada [9] e exaustiva [57].

Atualmente o ambiente Statsim fornece a opção de que atividades sejam associadas aos estados, permitindo que seja possível observar a funcionalidade do sistema através da opção de Log existente. Dessa forma há a possibilidade de validar o sistema não só sob a visão comportamental como também sob a visão funcional.

Os principais conceitos sobre statecharts são apresentados na seção 3.2; na seção 3.3 a primeira versão do ambiente StatSim é discutida, bem como as evoluções que o ambiente sofreu até se chegar à execução programada. Na seção 3.4 é apresentada a terceira e última versão em que é introduzida a visão funcional nos sistemas reativos especificados por statechart. Na seção 3.5 é apresentado como usar o StatSim para a validação de um modelo e o protótipo resultante desse modelo. Na seção 3.6 são feitas as considerações finais.

3.2. Statecharts

Harel propôs uma linguagem visual para especificação da parte comportamental de sistemas reativos, chamada Statecharts [32, 33, 34]. Eles podem ser considerados como extensão das Máquinas de Estados Finitos [8]

utilizando os conceitos de estados e transições e possuindo as características adicionais de: propagação de eventos, estados paralelos e hierarquia de estados. Eles possuem semântica e sintaxe formalmente definidas, o que possibilita a verificação de uma especificação [34]:

Um statechart se comporta como uma Máquinas de Estados Finitos, ou seja, a cada ocorrência de um evento se o estado origem da transição associada ao evento estiver ativo, a transição dispara. O estado origem é desativado e o estado destino da transição é ativado. Diz-se que a transição de um estado para outro ocorreu em um passo.

Em statecharts os estados são representados por retângulos com os cantos arredondados e as transições são representadas por setas rotuladas.

Figura 3.1 - Exemplo de Estados

Uma característica adicional dos statecharts é que um estado pode ser composto por outros estados, sendo chamado de superestado, ou pode ser um estado atômico, não composto de outros estados. Tem-se assim uma hierarquia de estados. A Figura 3.1 ilustra os casos de estados atômicos e superestados. Por exemplo, o estado A é um estado composto de B, X e Y, enquanto B e X são também superestados e Y é estado atômico. O estado pode ser composto tanto por estados atômicos como por superestados. Pode-se usar, também, a terminologia pai e filho; assim, diz-se que B, X e Y são filhos de A e, reciprocamente, que A é pai de B,X e Y.

Os superestados podem ser classificados em dois tipos: XOR e AND. O tipo XOR, ou exclusivo, significa que somente um dos sub-estado está ativo a cada passo. O superestado tipo AND representa o paralelismo entre estados, significando que dois ou mais sub-estados estão ativos no mesmo passo. A diferença gráfica entre o estado XOR e AND é que o AND tem segmentos de reta pontilhados separando seus filhos. Na Figura 3.1, o estado C3 é do tipo AND, com filhos C31 e C32. Os estados C, B, X e A são do tipo XOR.

As transições são definidas como uma tripla (o , l , d) onde: o significa o conjunto de estados origem, l o rótulo (o nome de um evento ou de uma expressão envolvendo eventos e condições) e d o conjunto de estados destino. Uma transição é disparada se estiver habilitada, isto é, se o estado origem ao qual ela está associada estiver ativo. A cada transição tem-se uma nova configuração de estados para o statechart e essa é fornecida por uma tupla composta somente dos estados básicos do sistema que estão ativos simultaneamente. Os ancestrais desses estado também estão ativos nesse passo, embora não façam parte da tupla.

Como foi dito anteriormente, o conceito de Máquina de Estado Finito é parte integrante dos statecharts. Dessa forma os estados “defaults”, isto é, os estados iniciais que o sistema deve assumir quando é ativado, são representados pelos estados que possuem uma transição “default”. Além desse tipo, statecharts permitem representar transições um para um (1:1), que têm somente um estado origem e um estado destino e transições M para N (M:N), que podem ter vários estados origem e/ou vários estados destino.

Quando o statechart da Figura 3.2 é ativado, tem-se inicialmente (ou por “default”) o estado Y ativo. Desse modo, a transição rotulada por $y\bar{b}$ está habilitada e será disparada no momento que o evento externo associado a ela ocorrer. Qualquer evento diferente de $y\bar{b}$ que ocorra no sistema, nesse passo, não causará mudança de estado.

Considera-se que os rótulos das transições exibidas na Figura 3.2 são eventos externos recebidos pelo sistema. A transição rotulada por $y\bar{b}$ sendo disparada, causa a desativação do estado Y e a ativação do estado C, por

“default”. Também por “default” ocorre outra ativação, agora referente ao estado C1. A configuração de estados desse statechart, nesse passo, é (C1). Como o estado C1 está ativo, a transição rotulada por k pode disparar quando o evento externo k for recebido pelo sistema. Isso ocorrendo, tem-se a desativação do estado C1 e a ativação dos estados “defaults” de C31 e C32. A ocorrência do evento bx faz com que a transição com esse rótulo dispare, desativando o estado B e seus filhos e o sub-estado que estava ativo, para ativar o estado X e consequentemente X2.

Figura 3.2 - Exemplo de um Statechart

São permitidas transições caracterizadas por um evento ou expressão seguida de uma ação. O rótulo normalmente é composto da seguinte forma: $ne[c]/a$, onde ne é a expressão contendo eventos, c é a condição sob a qual o evento pode ocorrer e a é uma ação. Quando essa transição for disparada, ou seja, quando a expressão associada a ela tornar-se verdadeira, em consequência da ocorrência dos eventos que compõem a expressão, a ação será executada. Essa ação pode gerar eventos internos ou externos ao sistema ou pode ser a atribuição de uma expressão a uma variável. A ação, por definição,

ocorre instantaneamente, não consumindo tempo. Na Figura 3.2, exemplos de transição com ação são as rotuladas por “ a/m ” e “ s/n ”.

Quando C3 é ativado, a configuração de estados desse statechart é (C311, C322), pois os estados C311 e C322 são estados “default”. Quando ocorre o evento externo s , a transição rotulada por s/n dispara, ocorrendo a transição do estado C322 para o estado C321. Simultaneamente, ocorre a transição do estado C311 para C312 rotulada pelo evento n . Assim, após esse passo a configuração de estado é: (C312, C321). O fato de ter ocorrido um evento e esse ter sido propagado é um conceito adicional de statecharts, conhecido como “broadcasting” ou reação em cadeia.

As expressões do rótulo das transições podem ter variáveis associadas, bem como combinações de eventos usando os operadores **and**, **or**, **false (fs)**, **true (tr)**, **changed (ch)**, **current (cr)** e outros, que são descritos na Tabela 3.1. Esses operadores enriquecem os statecharts, possibilitando a representação de diversas situações. As expressões compostas por esses operadores são tratadas como expressões booleanas e após sua avaliação, quando são verdadeiras, possibilitam o disparo da transição. O evento nulo é representado por “lambda”. É um tipo especial de evento que permite a mudança de um estado para outro sem que um evento externo ocorra. Utiliza-se o evento nulo, por exemplo, quando se deseja que em um certo passo um determinado estado sempre transicione para um outro estado, independentemente da ocorrência de qualquer evento.

Um conceito adicional dos statecharts é o de História (H), que é uma outra maneira de se atingir um grupo de estados. Ativar um determinado grupo de estados, dentro de um grupo que possui história, significa considerar como ativo aquele estado que foi mais recentemente visitado. História é graficamente representado pela letra H inserida em um pequeno círculo.

A Figura 3.3 (a) exhibe um exemplo de história simples, H. Quando o evento c ocorre ou entra-se no estado mais recentemente visitado ou entra-se no estado “default”, se for a primeira vez. Isso significa que se o estado X estiver ativo (X2 está ativo) e pela ocorrência de algum outro evento abandonar-se esse

estado, posteriormente, ao retornar a ele (através da transição rotulada por c) o estado a ser assumido é o próprio X2. História H se aplica apenas ao nível no qual ela aparece, ou seja, a ativação de um eventual sub-estado de X1 ou X2, não levará em consideração a história do estado X. Já no exemplo da Figura 3.3 (b), a história (H^*), se aplica ao nível dos eventuais sub-estados dos estados de B. Suponha que os estados ativos, nesse passo, sejam (B,B3) e o evento x ocorra. Posteriormente, quando o evento a ocorrer, os estados ativos voltam a ser (B,B3) e se B3 tiver sub-estados, esses também serão ativados/desativos de acordo com a situação em que se encontravam quando x ocorreu.

Figura 3.3 - História

Em contraposição com ações que não consomem tempo, isto é, são instantâneas, existem as atividades que para serem realizadas consomem tempo. Elas podem ser associadas aos estados e referem-se à realização de alguma função que o estado está representando. Quando associadas às transições elas se comportam como ações. Na realidade, essa é uma decisão deixada para o usuário, que pode associar atividades a estados/transições, conforme a semântica da aplicação.

Além de estados, transições e história, existem outros conceitos que enriquecem os statecharts e esses são apresentados através da Tabela 3.1. Essa tabela mostra para cada um desses conceitos a sua notação e sua semântica.

Tabela 3.1 - Conceitos Adicionais de Statecharts

Conceito	Notação	Semântica
Condições Especiais		
current	cr(condição)	retorna o valor atual de uma condição
in	in(estado-X)	retorna o valor "true" quando o sistema está no estado X.
not_yet	not_yet(evento_x) ou ny(evento_x)	retorna "true" quando o sistema avalia que o evento_x ainda não foi disparado
Operadores Relacionais (aplicados a condições)		
igual	a=b	retorna "true" se a igual a b
diferente	<>	retorna "true" se a diferente de b
maior	>	retorna "true" se a maior que b
menor	<	retorna "true" se a menor que b
maior ou igual	>=	retorna "true" se a maior ou igual a b
menor ou igual	<=	retorna "true" se a menor ou igual a b
Operadores Lógicos (aplicados às condições)		
and	exp1 and exp2	retorna "true" se exp1 e exp2 forem "true"
or	exp1 or exp2	retorna "true" se exp1 ou exp2 forem "true"
not	not exp1	atribui o valor "true" a uma variável se exp1 for "false" ou vice-versa
Eventos Especiais		
true	true(condição) ou tr(condição)	determina um evento no instante que a condição se tornou verdadeira
false	false(condição) ou fs(condição)	determina um evento no instante que a condição se tornou falsa
changed	changed(variável) ou ch(variável)	determina um evento no instante em que a variável mudou de valor
exit	exit(estado) ou ex(estado)	determina um evento no instante em que o sistema deixou de estar no estado
entered	entered(estado) ou en(estado)	determina um evento no instante em que o sistema passou a estar no estado
time_out	t_out(evento,m)	determina um evento no instante em que "m" passos transcorreram desde o momento em que o sistema passou a estar no estado origem da transição
hide/closure		hide: oculta eventos closure: confirma a situação do evento
nulo (lambda)	evento correspondente: true(true)	permite disparar a transição em todo passo em que o seu estado origem está ativo
Ações		
condição,variável atividade	rótulo/ação rótulo/atividade	ao disparar o evento ocorre a ação ao disparar o evento a atividade deve ser realizada
clear history	clh	limpa história associada ao estado,
Atividade em estado	nome da atividade	executada quando de acordo com o seu tipo

3.3. O Ambiente StatSim

O ambiente StatSim, Statecharts Simulator, foi desenvolvido pelo grupo de engenharia de software do ICMSC-USP, permitindo a edição e simulação de statecharts utilizando a sintaxe e a semântica formal especificada em [33], linguagem C para implementação de procedimentos, X-View para interfaces gráficas e manipulação de janelas em ambiente Unix e em estações de trabalho Sun.

Vários trabalhos foram desenvolvidos desde 1989 para que o ambiente StatSim chegasse ao estágio em que se encontra. Nas subseções seguintes comenta-se cada uma das principais etapas realizadas. A seção 3.3.1. apresenta a fase inicial do ambiente e as decisões tomadas para construí-lo. Na seção 3.3.2. apresenta-se uma evolução do ambiente, reunindo diversos trabalhos e, posteriormente, na seção 3.4. relatam-se algumas contribuições deste trabalho para a evolução do ambiente.

3.3.1. Versão Inicial

O ambiente foi projetado, inicialmente, definindo-se as estruturas de dados dos elementos que compõem a base de dados, isto é, a representação de estados e transições. Para a elaboração dessas estruturas utilizou-se o conceito de tipos abstratos de dados. A interface com o usuário foi definida, inicialmente, com diálogos do tipo pergunta/resposta. Elaborou-se a Linguagem de Especificação de Statecharts (LES) que permite a descrição textual de um statecharts para inserí-lo na base de dados do StatSim. Quando da simulação de um statecharts já editado, a cada ocorrência de evento o simulador consultava a base de dados e fornecia, também de modo textual, o resultado da simulação, atualizando a base de dados [22].

A Linguagem de Especificação de Statecharts foi criada para permitir a descrição textual de um statecharts a qual pode ser analisada. O analisador recebe a especificação em LES e faz a análise, que pode ser dividida em três partes: análise léxica, análise sintática e análise de contexto. A partir daí

ocorre a geração de código intermediário das estruturas básicas da base de dados do statechart para possibilitar a interpretação dos dados para a fase de simulação.

Posteriormente, a interface foi modificada, sendo apresentado ao usuário um menu com as seguintes opções : “Analisar LES , Inspeccionar, Simular, Salvar, Recuperar, Terminar”. A partir da escolha do usuário, sub-janelas eram apresentadas sequencialmente até a realização da opção escolhida. A edição de um statechart era então processada, terminando com o seu armazenamento. A fase de simulação era realizada apresentando a janela dos eventos inseridos para o statechart que estava sendo simulado, uma janela com a configuração dos estados ativados e uma outra exibindo passo a passo os estados ligados/desligados e os eventos disparados chamada de “Log” da simulação.

**Tabela 3.2 (a) Comportamento do Editor e Simulador de Statecharts na 1ª
Versão para os Conceitos de Estado e Transição**

Conceito	Editor Gráfico	Algoritmo de Simulação
Estado:		
AND,OR superestado, estado atômico	Permite a edição de superestados e estados atômicos, AND e OR	Interpreta os diferentes tipos de estados
História (H, H*)	Permite transição com história	Interpreta fazendo com que a transição recupere a última configuração de estados ativos
“default”	Permite estado “default”	Interpreta como o estado inicial de um superestado
def/H , def/H*	Permite estado “default” com história	Interpreta como o estado inicial de um superestado e considera história
Transição:		
um para um	Permite a edição	Interpreta realizando a transição de um estado se o evento associado à transição ocorreu e se a transição está habilitada
M para N	Permite a edição	Interpreta realizando a transição de diversos estados se o evento associado à transição ocorreu e se a transição está habilitada
transição simultânea	Permite a edição	Interpreta. Em estados AND quando há transições com o mesmo evento associado, estados origem diferentes, as transições disparam simultaneamente se o evento ocorreu e se os estados origem estiverem habilitados
propagação de eventos	Não reconhece	Não interpreta

Paralelamente à construção do analisador sintático, foi desenvolvido um Editor Gráfico para Statecharts (EGS), utilizando-se a mesma estrutura de base de dados descrita anteriormente acrescida de informações gráficas. As informações gráficas são as que definem o formato visual dos elementos de um statechart. As rotinas básicas para edição de statecharts foram então incorporadas ao ambiente e integradas ao algoritmo para simulação de statecharts já existente textualmente.

Nessa etapa, que será chamada de primeira versão (1ª), existiam algumas restrições, pois nem todas as características dos statecharts eram suportadas pelo ambiente. O único tipo de simulação existente era a interativa, na qual o usuário fornece passo a passo os eventos desejados.

Tabela 3.2 (b) Comportamento do Editor e Simulador de Statecharts na 1ª Versão para os Conceitos de Expressões e Condições

Conceito	Editor Gráfico	Algoritmo de Simulação
Expressões		
cr (var)	Reconhece como operador de expressão	Interpreta sempre como verdadeiro
Operadores Algébricos +, -, /, *	Reconhece como operador de expressão	Interpreta sempre como verdadeiro
Condições :		
in	Reconhece como operador de condição	Interpreta verificando se o estado está ativado
current	Reconhece como operador de condição	Interpreta sempre como verdadeiro
not_yet	Reconhece como operador de condição	Interpreta sempre como verdadeiro
Operadores relacionais: =, <>, , <, >=, <=	Reconhece como operadores de condição	Interpreta sempre como verdadeiro,
and, or, not	Reconhece como operadores de condição	Interpreta sempre como verdadeiro

As Tabelas 3.2(a) a 3.2 (d), baseadas em [22], apresentam para a primeira versão como os Conceitos existentes em statecharts são tratados pelo Editor Gráfico e pelo Simulador. Utiliza-se a frase: "Interpreta sempre como verdadeiro" para indicar que o simulador não dá o tratamento semântico adequado e sim que interpreta sempre como verdadeira a expressão qualquer que seja a

situação. A frase “Não Interpreta” é usada quando o simulador não está apto a fazer a interpretação. “Não Reconhece” é usada para expressar que o editor gráfico não suporta o objeto especificado. Os conceitos apresentados nessas tabelas serão repetidos para as outras versões, a fim de possibilitar ao leitor a comparação entre as versões, evidenciando as expansões realizadas no ambiente.

**Tabela 3.2 (c) Comportamento do Editor e Simulador de Statecharts na 1ª
Versão para os Conceitos de Eventos**

Conceito	Editor Gráfico	Algoritmo de Simulação
Evento		
evento externo	Reconhece como identificador	Aceita como entrada em cada passo da simulação
evento interno	Reconhece como operador de condição, de expressão e palavras chaves	Interpreta avaliando-o a cada passo da simulação
true, false	Reconhece como operador aplicado à condição pelo símbolo true ou false	Avalia o valor da condição
changed (v)	Reconhece como operador aplicado à variável pelo símbolo ch	Interpreta sempre como verdadeiro
exit(e), entered(e)	Reconhece como operador aplicado a estado pelo símbolo ex e en, respectivamente	Interpreta sempre como verdadeiro
time_out	Reconhece como evento interno de controle de tempo	Interpreta sempre como verdadeiro
hide/closure	Não reconhece	Não interpreta
evento nulo	Não reconhece	Não interpreta
or, and	Reconhece como operadores lógicos de eventos	Avaliados de acordo com o seu significado convencional
evento[condição]	Reconhece	Interpreta avaliando a condição

O conjunto de expressões permitidas é definido como a combinação de operações algébricas sobre números e variáveis, além do operador especial “current”. O conjunto de condições primitivas define basicamente os valores “true” e “false”, que são avaliados durante uma transição no sistema. Essas condições fazem com que o evento percebido pelo sistema seja neutralizado em função de uma avaliação das condições especificadas na transição. A Tabela 3.2 (b) mostra o comportamento do editor gráfico e do simulador para esses conceitos.

**Tabela 3.2(d)- Comportamento do Editor e Simulador de Statecharts na 1ª
Versão para os Conceitos de Ações e Atividades**

Conceito	Editor Gráfico	Algoritmo de Simulação
Ação		
evento	Reconhece, atribui o valor do evento à ação	Não interpreta
expressão	Reconhece, atribui o valor da expressão a uma variável	Não interpreta
clear history	Reconhece como ação especial	Não interpreta
clear(est)	Reconhece, como ação especial	Não interpreta
Atividade/estado:		
chamada de procedimento	Não reconhece	Não interpreta

3.3.2. Versão Acrescida de Outros Modos de Simulação

A próxima expansão realizada foi relativa à simulação dentro do ambiente EGS. Batista [3] afirma que esse passo é natural uma vez que a base de dados utilizada é a mesma que Fortes usa para realizar a simulação textual de statecharts [92]. A partir do carregamento na tela de um statechart já editado inicia-se a sua simulação da simulação interativa, que é a opção “default” quando se está no modo de simulação. A partir da ocorrência de eventos, as transições habilitadas disparam. Quando isso acontece o estado origem da transição é pintado de branco, representando que está desativado, e o estado destino é pintado de preto, representando que o estado está ativo. Pode-se então acompanhar a simulação de um statechart de modo gráfico, o que facilita muito a validação de sistemas.

A simulação batch foi uma outra expansão realizada. Nesse tipo de simulação deve-se elaborar um arquivo batch, contendo a sequência dos eventos que devem ocorrer. Quando é iniciada essa simulação o arquivo batch é lido e as transições são disparadas, podendo-se observar a animação dos estados. Como ocorre na simulação interativa, os estados ativos são os pintados de preto e os não ativos são pintados de branco.

Em seguida foi incorporada ao ambiente a simulação exaustiva, que engloba algoritmos para detecção de “deadlock”, não determinismo e alcançabilidade. Os algoritmos foram desenvolvidos baseados em uma árvore de alcançabilidade para statecharts [6], que é similar à árvore de alcançabilidade para Redes de Petri. A árvore de alcançabilidade é construída considerando-se todas as sequências possíveis de configurações de um statechart. A cada passo todos os eventos são considerados verdadeiros (as condições são incluídas nas expressões de evento e também são consideradas) e as transições que estão habilitadas, num passo, levam a uma nova configuração do sistema. Na árvore, cada nó é associado a uma configuração de estados do sistema em desenvolvimento. Os nós história representam todas as configurações alcançáveis a partir de transições ou estados associados ao símbolo história. Boaventura justifica a utilidade da alcançabilidade de uma configuração para verificar se todos os estados especificados são ativados em algum momento, ou se sob certas condições, uma determinada configuração é alcançada.

A reiniciabilidade de um statechart é verificada se para toda configuração alcançada a partir de uma anterior, existe uma sequência de eventos que a faça retornar à configuração anterior. Um sistema modelado em statecharts poderá ter possíveis “deadlocks” se atingir configurações de estados das quais nenhuma transição pode disparar e, portanto, em sua árvore de alcançabilidade aparecerão nós terminais. Maiores detalhes sobre essas características podem ser encontrados em [6].

Dando continuidade aos trabalhos de expansão do ambiente, mais um tipo de simulação foi incorporada aos já existentes: a simulação programada [9]. Ela é realizada através do Módulo de Execução Programada, MEP, que é formado de um editor de texto para edição de programas e de um interpretador de programas, que realiza a execução dos modelos de acordo com os programas previamente escritos. Os programas devem ser escritos de acordo com a Linguagem de Controle de Execução, LCE. Nesses programas especifica-se como deve ser a ocorrência de eventos, a distribuição probabilística que eles devem obedecer, quantos passos de simulação devem ser realizados, etc. de acordo com a sintaxe da LCE. São possíveis dois tipos de relatórios como produtos finais do MEP: relatório estatístico e relatório de passos. As informações que constam do relatório de passos são provenientes do interpretador da LCE, que a cada passo

da simulação armazena essas informações. O relatório estatístico é gerado cada vez que um programa escrito em LCE é executado, independentemente das especificações do usuário. Maiores detalhes sobre a LCE podem ser encontrados em [9]. Com a inserção desse modo de simulação aos já existentes, o ambiente foi enriquecido, [10].

Foram incorporados ao ambiente por [92] o operador “*current*” que retorna o valor atual da variável e é denotado por “*cr(var)*”; o evento “*time-out*” que determina um evento no instante em que *n* unidades de tempo transcorreram desde o momento em que o sistema passou a estar no estado origem da transição que possui esse evento. O estado origem da transição que tem o evento “*time-out*” associado tornar-se-á desativado após a ocorrência de tantos passos quanto for o número estipulado como unidade de tempo. “*not yet(evento x)*” que retorna “true” quando o sistema avalia que o *evento x* ainda não foi disparado, também foi implementado ao ambiente. Além dessas, foram também incorporados ao ambiente: “*changed(v)*”, que determina um estímulo no instante em que a variável mudou de valor, “*exit(f)*” que determina um estímulo no instante em que o sistema deixou de estar no estado *f*, “*hide/closure*” que oculta um evento interno não o tornando disponível na tabela de eventos que podem ser selecionados pelo usuário.

**Tabela 3.3(a) -Comportamento do Editor e Simulador de Statecharts na 2ª
Versão para os Conceitos de Estado e Transição**

Conceito	Editor Gráfico	Algoritmo de Simulação
Estado:		
AND,OR, superestado,estado atômico	(*)	(*)
História (H, H*)	(*)	(*)
“default”	(*)	(*)
def/H, def/H*	(*)	(*)
Transição:		
um para um	(*)	(*)
M para N	(*)	(*)
transição simultânea	(*)	(*)
propagação de eventos	Reconhece a ocorrência de eventos interno	Interpreta, disparando a transição associada ao evento interno, permitindo a reação em cadeia

As Tabelas 3.3(a) a 3.3(d) apresentam a segunda (2ª) versão do ambiente StatSim uma reprodução das Tabelas 3.2(a) a (d), exibidas na seção anterior, evidenciando somente as extensões realizadas no ambiente StatSim. Os conceitos tratados pela primeira versão aparecem com asterisco (*) nas colunas "Editor Gráfico" e "Algoritmo de Simulação". As demais convenções continuam as mesmas adotadas para as Tabelas 3.2.

**Tabela 3.3(b)-Comportamento do Editor e Simulador de Statecharts na 2ª
Versão para os Conceitos de Expressões e Condições**

Expressões	Editor Gráfico	Algoritmo de Simulação
cr (var)	Reconhece como operador de expressão	De acordo com a semântica
Operadores Algébricos +, -, /, *	(*)	(*)
Condições :		
in	(*)	(*)
current	Reconhece como operador de condição	De acordo com a semântica
not_yet	Reconhece como operador de condição	De acordo com a semântica
Operadores relacionais: =, <>, , <, >=, <=	Reconhece como operadores de condição	De acordo com a semântica
and, or, not	Reconhece como operadores de condição	De acordo com a semântica

**Tabela 3.3(c) -Comportamento do Editor e Simulador de Statecharts na 2ª
Versão para o Conceito de Evento**

Eventos	Editor Gráfico	Algoritmo de Simulação
evento externo	(*)	(*)
true, false	(*)	(*)
changed (v), exit(e), entered(e)	Reconhece como operador aplicado à condição	De acordo com a semântica
time_out	Reconhece pelo símbolo t_out	De acordo com a semântica
hide closure	Reconhece ocultando os eventos selecionados.	Não permite que eles sejam disparados pelo usuário
evento nulo	Não reconhece	Não interpreta
or, and	(*)	(*)
evento[condição]	(*)	(*)

**Tabela 3.3(d) -Comportamento do Editor e Simulador de Statecharts na 2ª
Versão para o Conceito de Ações e Atividades**

Ação	Editor Gráfico	Algoritmo de Simulação
evento	(*)	Interpreta de acordo com a semântica
expressão	(*)	
clear(est)	(*)	Limpa história quando abandona o estado.
clear history	(*)	Interpreta limpando história se existir
Atividade:		
chamada de procedimento	Não reconhece	Não interpreta

Figura 3.4 - Características do Ambiente StatSim Antes deste Trabalho

A finalidade da Figura 3.4 é a de apresentar ao leitor um esboço de como o sistema se comporta em cada uma das versões comentadas em termos de edição, simulação e algoritmos de simulação. Esses algoritmos referem-se à forma como foi tratado cada um dos conceitos quando aceitos/reconhecidos pelo editor gráfico aceita. Dessa forma, espera-se que o leitor tenha uma visão completa da evolução do ambiente ao longo desses anos.

3.4. Evolução do Ambiente StatSim - Novas Funcionalidades

Dando continuidade à expansão do ambiente StatSim este trabalho cuidou da implementação de novas funcionalidades, que são detalhadas a seguir, permitindo a edição/simulação de transição “default” com ação associada, a implementação de evento nulo, a edição de atividades associadas aos estados e transições e sua simulação através da forma textual pela opção de Log existente.

A extensão realizada iniciou-se com a familiarização, da autora, com o ambiente, através da especificação e simulação de statecharts de problemas já especificados por outras técnicas. A partir daí pôde-se observar que a utilização do ambiente não era possível somente através da solicitação das informações, já que no ambiente não existem menus de ajuda ao usuário. Necessitava-se do monitoramento de outra pessoa que o tivesse utilizado ou então recorrer a diversos documentos, como as dissertações de mestrado e relatórios de iniciação científica que tratassem como utilizar o editor e simulador de statecharts. Para facilitar o uso, era preciso a elaboração de um documento que reunisse todas as informações sobre a utilização do ambiente. Assim, elaborou-se o “Manual de Uso do Sistema StatSim”, [73], contendo exemplos de todas as opções e mensagens existentes no ambiente, mostrando ao usuário quais procedimentos devem ser realizados quando da escolha dessas opções.

Depois do aprendizado de utilização do ambiente como usuário passou-se à fase de entendimento de sua arquitetura interna de forma que alterações pudessem ser realizadas, visando à implementação das expansões pretendidas. As telas iniciais de edição e simulação são as apresentadas nas Figuras 3.5 e 3.6 respectivamente. A tela de edição, nessa época, diferia da atual pois não apresentava o botão “Activities”, uma vez que, como descrito no

Figura 3.5 - Tela de Edição do Ambiente StatSim

Figura 3.6 - Tela de Simulação do Ambiente StatSim

parágrafo anterior, a parte relacionada a atividades foi uma das tarefas aqui realizadas.

Nos artigos em que Harel define informalmente os statecharts não fica claro que a transição “default” pode ter ação associada, [32, 33, 35]. Esse ponto também não é tratado em [33], que descreve a sintaxe/semântica formal dos statecharts. Nos estudos realizados pôde-se observar que a sua existência não viola as regras de sintaxe e semântica propostas por ele. De forma que algumas restrições tiveram que ser respeitadas para que não houvesse violação de tais regras. As ações podem ser operações com variáveis ou eventos. Uma das restrições existente quanto aos eventos é que uma transição “default” não pode gerar um evento que dispara uma transição cujo estado origem é o próprio estado “default”, conforme mostra a Figura 3.7. Se isso fosse possível, em um único passo ocorreria mais de uma transição de estados dentro de um mesmo superestado do tipo XOR burlando a semântica proposta para statecharts. Nesse exemplo, quando o statechart for iniciado por “default”, tem-se a configuração de estado (A1, A2). Em A1 a ação associada à transição “default” é a ocorrência do evento “início” o que causaria em um só passo a ativação e desativação do estado A1 e a ativação do estado CA1. Desse modo ocorreriam duas transições de estado em um único passo.

Figura 3.7 - Exemplo não Permitido de “default” com Ação

Para que o ambiente pudesse aceitar transição “default” com ação, a interface com o usuário foi alterada. Primeiramente, o usuário escolhe a opção de criar uma transição “default” pelo menu de opções existente no botão *Transitions* do modo de edição. Em seguida deve-se desenhar a transição para daí selecionar a opção “Default/Action” no canvas referente aos atributos da transição. Após essa seleção, deve-se fornecer a ação associada a essa transição, como mostra a Figura 3.8. A transição “default” com ou sem ação tem a mesma representação gráfica na edição de um statechart, ou seja, não aparece no diagrama a parte relativa ao rótulo nem à ação, embora a expressão esteja armazenada na base de dados.

Figura 3.8(a) -Criação de Transição “Default” com Ação

A transição “default” associada ao estado A1 de ParteA tem como ação a expressão $com := com + 10$. A variável *com* deve ser previamente definida. Toda vez que esse estado for iniciado por “default” há o incremento da variável *com* de 10 unidades. Esse procedimento é notado quando o statechart é simulado. A ação pode ser também o ocorrência de um evento. Como exemplo, a Figura 3.8(a) mostra que a transição “default”, associada ao estado A221, tem como ação a propagação do evento “*muda_def*”.

Figura 3.8(b) - Valor da Variável *com* Antes de Iniciar a Simulação

Figura 3.9 - Início da Simulação com Configuração Default

No statechart da Figura 3.8(b) antes de iniciar a simulação por “Default Configuration” a variável “com” está com o valor zero. Após o início por “default” obtêm-se a configuração mostrada na Figura 3.9 e a variável “com” teve o seu valor inicial acrescido de 10 unidades. Quando o evento “início” ocorre tem-se a configuração exibida na Figura 3.10. No momento em que o estado A221 de A2 foi ativado por “default” houve a propagação do evento “muda_def”, pois ele está associado à transição default, ocorrendo a transição de B2 para B1 no estado ParteB. O Log da Figura 3.10 mostra passo a passo os eventos que ocorreram e a relação de todos os estados ativos.

Figura 3.10 - Simulação da Transição “default” com Ação

Outra característica implementada foi a edição e simulação do evento nulo. A interface não sofreu alterações nem no modo de edição nem no de simulação. Após selecionar o tipo de transição desejada são solicitados os atributos dessa transição: tipo (primitiva/composta), o rótulo e o evento

associados. Nesse caso o rótulo pode ser qualquer combinação de caracteres permitidos pela gramática, mas normalmente utiliza-se a palavra “*lambda*” para representar graficamente que se refere a um evento nulo. Porém, internamente o evento associado ao rótulo “*lambda*” é representado pela expressão *true(true)*. Isso permite que essa transição dispare a todo passo. A Figura 3.11 mostra a criação de um evento nulo. Nesse exemplo o rótulo “*toda_vez*”, é usado para um evento nulo, tendo como origem e destino o estado A221.

Figura 3.11 - Evento Nulo

O ambiente StatSim foi também ampliado, neste trabalho, de modo a permitir que as atividades fossem editadas e posteriormente simuladas. Para que isso fosse possível alterou-se a gramática em vigor acrescentando-se as atividades aos estados e transições, como mostra a Figura 3.12. As alterações realizadas são as marcadas com o símbolo (**). A gramática completa do ambiente StatSim encontra-se no Apêndice 1. Os módulos do ambiente foram alterados para suportar a nova funcionalidade. A representação de um estado passou a

contar com um campo ponteiro que aponta para uma lista de atividades a ele associada. Uma tabela para armazenar as atividades foi declarada. Os procedimentos de manipulação de estados foram alterados, assim como os procedimentos de gravação e recuperação de statecharts.

```

...
<estados> ::= <descrição estado> { <descrição estado> }
<descrição estado> ::= ESTADO <nome-estado> <tipo> [<atividades>]      (**)
<atividades> ::= ATIVIDADE <função>                                     (**)
<função> ::= DO <chamada-atividade> {','<chamada-atividade>}          (**)
           | ENTRY <chamada-atividade> {','<chamada-atividade>}        (**)
           | EXIT <chamada-atividade> {','<chamada-atividade>}        (**)
<chamada-atividade> ::= <identificador> '(' <lista-parametro> ')'      (**)
<lista-parametro> ::= <identificador> {',' <identificador>}
...
<transições> ::= <transição> { <transição> }
<transição> ::= EVENTO <evento>
                ORIGEM <estado-origem> {','<estado-origem> }
                DESTINO <estado-destino> [(/H | /H*)]
                {','<estado-destino> [(/H | /H*)] }
                [ AÇÃO '{' <ações> '}' ]
...
<ações> ::= <ação> {','<ação> }
<ação> ::= <identificador> | <variavel> ':=' <exp>
           | clear '(' <nome-estado> [ * ] ')' | <chamada-atividade>    (**)
...

```

Figura 3.12 - Trecho da Gramática Modificada para Edição de Atividades

As palavras chave **DO**, **ENTRY** e **EXIT** foram usadas para indicar a ativação das ações/atividades, fazendo a ligação entre o modelo dinâmico (statecharts) e o modelo funcional (diagrama de atividades), de acordo com vários métodos de análise e projeto de sistemas [81, 88, 35]. Os significados dessas chamadas são:

DO significa que a atividade começa a ser executada quando um estado for ativado e pára sua execução quando o

estado for desativado, isto é, executa enquanto o estado estiver ativo.

ENTER significa que a atividade começa a ser executada a partir da ativação do estado e pára a execução por conta própria.

EXIT significa que a atividade começa a ser executada quando o estado for desativado e pára a execução por conta própria.

Figura 3.13 - Edição de Atividades quando da Criação do Estado

De dois modos pode-se associar atividades aos estados: quando de sua criação ou posteriormente à sua criação. Para essa ampliação foi adicionado à interface do ambiente StatSim o botão “Activities” na tela do modo de edição, para permitir que estados que foram criados e não tiveram atividades associadas no

momento da sua criação, possam tê-las posteriormente. A Figura 3.13 mostra a edição de atividades quando da criação de estados. Uma janela é apresentada solicitando o tipo e o nome da atividade a ser associada ao estado que está sendo criado. A cada atividade fornecida o usuário deve confirmá-la e para encerrar o processo de edição de atividades o botão rotulado por "End" deve ser pressionado. Nesse exemplo o estado B3 está sendo criado e a atividade "INSERIR_ATIVIDADE_CRIANDO_EST" é associada ao estado e deve ser realizada enquanto o estado estiver ativo (DO).

Figura 3.14 - Edição de Atividades posterior à Criação do Estado

A Figura 3.14 mostra a edição de atividades posteriormente à criação do estado. Nesse caso, quando o botão "Activities" for selecionado, deve-se escolher o estado que deve receber as atividades, clicando o seu nome com o mouse. Em seguida é apresentada uma tela contendo as informações sobre esse estado: nome, tipo, sub-estado, atividades (se já existirem), etc.. Junto com essa tela, uma outra é apresentada para que sejam fornecidas as atividades associadas ao estado e seu tipo, do mesmo modo que o descrito para a edição de

atividades quando da criação de estados. Quando todas as atividades necessárias tiverem sido inseridas deve-se encerrar essa operação através da seleção de botões apropriados e salvar esse statechart para que a sua base de dados seja atualizada. Nesse exemplo, o estado B3 possuía uma atividade e a agora foi inserida uma nova atividade ENTRY "INSERIR_ATIVIDADE_APOS_CRIAÇÃO_E". Assim, quando o estado B3 se tornar ativo essa atividade tem início e pára por conta própria.

Existem dois tipos de classificação para as máquinas de estados finitos: Máquinas de Mealy ou Máquinas de Moore. As máquinas de Mealy são aquelas onde as ações ou atividades estão associadas às transições. As máquinas de Moore são aquelas onde as ações ou atividades estão associadas ao estado em que a máquina se encontra. Os statecharts são uma junção dessas duas máquinas, pois ações podem ser geradas quando transições são executadas e atividades podem ser realizadas quando o estado ao qual elas estão associadas estiver ativo.

As atividades associadas às transições são tratadas de modo análogo às ações, ou seja, são realizadas de modo "instantâneo" quando ocorre o disparo da transição. Para efeitos de implementação considerou-se, nesse caso, as atividades como sendo do tipo ENTRY. As atividades associadas aos estados não são instantâneas, consomem algum tempo para sua realização .

Quando um statechart for simulado no modo interativo, pode-se observar a simulação das atividades selecionando-se a opção "Show-Log" existente nesse modo. Com essa opção, um Log é criado permitindo que se visualize para cada passo, textualmente, os eventos que foram disparados, os estados ativos, as atividades que estão sendo realizadas. Exemplos com Log podem ser encontrados na seção 3.5.2.3.

A Figura 3.15 mostra as características do ambiente StatSim englobando as três versões, como chamadas aqui, em termos de edição e simulação. As novas funcionalidades inseridas ao ambiente que fazem parte deste trabalho foram incorporadas à terceira (3ª) versão. Aqui também são colocados para cada um dos aspectos relacionados com editor gráfico, simulação e algoritmos de simulação, as expansões realizadas, como foi feito na Figura 3.4. A

terceira versão relacionada com os tipos de simulação sofreu alteração com relação ao Log, onde foi acrescida a representação das atividades que estão sendo realizadas a cada passo. Com relação ao editor gráfico, a mudança realizada foi permitir a edição de atividades e quanto ao algoritmo de simulação as alterações foram relacionadas à simulação de atividades, de transições default com ação e do evento nulo.

Figura 3.15 - Características do Ambiente StatSim após este Trabalho

3.5. Uso do Ambiente Statsim

Como foi dito anteriormente, o ambiente StatSim foi utilizado, pela autora, primeiramente com a edição e simulação de sistemas exemplos existentes na literatura, depois para sistemas exemplos adaptados de outras técnicas que foram especificados em statecharts. O ambiente foi também utilizado com a finalidade de validar o modelo exemplo de um relógio despertador, apresentado por Coleman em [16]. Para que a validação fosse realizada tanto sob o aspecto comportamental como funcional, os diagramas de atividades foram também modelados. Pôde-se então produzir um protótipo desse sistema apoiado nos diagramas de atividades e statecharts modelados, que foi implementado em linguagem C e Xview.

Para ilustrar o uso do ambiente StatSim utiliza-se o exemplo descrito na seção 3.5.1. A partir desse exemplo uma especificação e sua validação são realizadas seguindo o método descrito na seção 3.5.2. Tendo-se o modelo validado com o apoio do ambiente StatSim é possível gerar um protótipo do sistema baseado na especificação realizada, a qual é mostrada na seção 3.5.3.

3.5.1. Exemplo: um despertador

O exemplo refere-se a um relógio despertador com um visor, onde é mostrada ininterruptamente a hora do dia, a partir do momento em que o relógio for ligado. Em seu painel há os seguintes botões: **“Liga”**, que faz com que o relógio inicie o seu funcionamento; **“Desliga”** que faz o relógio deixar de funcionar e **“Acerto”**, que permite o acerto da hora em que se deseja despertar, ou seja, quando o alarme deve começar a soar; **“Confirma”** para indicar o término do ajuste da hora em que se deseja despertar; **“Pare”** que faz com que o alarme pare de soar e **“Soneca”**, que faz com que o alarme pare de soar por um tempo preestabelecido quando da construção do relógio e depois volte a soar novamente. A partir do momento em que o botão **“Acerto”** foi pressionado pode-se acionar o botão **“Cancela”**, caso o desejo seja o cancelamento da operação em curso. Um esboço do painel desse relógio é exibido na Figura 3.16.

Figura 3.16 - Esboço do Relógio Despertador

O funcionamento do relógio ocorre da seguinte forma: a partir do momento em que ele é ligado e acertado com a hora atual do dia ele pode ser preparado para despertar em uma hora predeterminada. Estando no modo de despertar há uma comparação constante entre a hora do dia com a hora estipulada para que ele desperte. No momento em que elas coincidem um alarme soa. Nesse momento existem três opções: a) pode-se acionar o botão **"Pare"**, fazendo com que o alarme pare de soar; b) pode-se deixar o alarme tocando por um tempo predeterminado, após o qual ele é desligado por conta própria ou c) pode-se acionar o botão **"Soneca"** fazendo com que o alarme deixe de soar por um intervalo de tempo especificado pelo próprio relógio, por exemplo, dez minutos, voltando a soar em seguida. Após a hora para despertar ter sido escolhida pode-se cancelar a tarefa que está sendo realizada acionando-se o botão **"Cancela"**.

3.5.2. O Método

O uso do ambiente StatSim para validação desse exemplo foi realizado seguindo a combinação das abordagens de decomposição funcional e

comportamental, como sugerem Smith e Gerhart [88] e que foram apresentadas no Capítulo 2, seção 2.4. Essa escolha ocorreu tendo em vista que o sistema exemplo é um sistema reativo e há interesse, além da modelagem de suas funções, também na modelagem de seu comportamento. Esse método tem o seu primeiro passo baseado na proposta de [88] e os seguintes foram elaborados a partir da utilização do ambiente StatSim para validação da especificação de alguns exemplos.

- 1- Preparar um modelo da aplicação [88]
 - 1.1. identificar as principais funções do sistema
 - 1.2. projetar o statechart do nível controlador
 - 1.3. especificar as relações de interface entre as atividades de um mesmo nível
 - 1.4. especificar os statecharts e as interfaces de cada sub-nível
 - 1.5. repetir o processo até que se obtenha o grau de detalhe desejado
- 2- Transformar os vários statecharts num único statechart e os vários diagramas de atividades em um único diagrama de atividades.
- 3- Validar o statechart usando o StatSim
 - 3.1. adaptar as notações utilizadas no modelo da aplicação para as notações existentes no ambiente StatSim
 - 3.2. utilizar os diversos tipos de simulação para validação do statechart construído
 - 3.3. checar o resultado da simulação com os resultados esperados. Caso não sejam encontrados modificar o statechart/atividades e repetir o passo 3
- 4- Projetar (a partir da especificação validada)
- 5- Implementar
- 6- Testar

Esse método será aplicado no exemplo do relógio despertador nas subseções seguintes.

3.5.2.1. Preparar um Modelo de Aplicação

Seguindo o método proposto anteriormente, deve-se identificar as funções do sistema, isto é, deve-se construir um Diagrama de Atividades para representar as funções que o sistema realiza. A notação usada para o Diagrama de Atividades segue a sintaxe proposta por Harel [35] com algumas diferenças relacionadas aos fluxos de controle que ativam/desativam as atividades. Os fluxos de dados são representados por setas com linhas contínuas enquanto que os fluxos de controle são representados por linhas tracejadas. A notação usada para ativação/desativação de atividades é a mostrada a seguir:

.....►► **ativação:** origem no controlador da atividade composta e destino na atividade a ser refinada.

.....◄◄ **desativação:** origem na atividade refinada e destino no controlador da atividade composta.

A Figura 3.17 (a) apresenta o Diagrama de Atividades correspondente às funções realizadas pelo relógio despertador. Nesse diagrama o controlador é representado por um statechart que recebe o nome de “Controlador de Despertador”, as atividades que devem ser realizadas são: “Monitorar Hora” e “Monitorar Alarme”. A hora do dia fica armazenada no depósito de dados com o mesmo nome, disponível para as consultas que são realizadas sistematicamente.

Os fluxos de controle existentes na Figura 3.17 (a) correspondem aos botões que podem ser pressionados para que o relógio inicie o seu funcionamento. O fluxo de dado, “t”, possui o valor da hora do sistema que é armazenada no depósito de dados, através da atividade “Monitorar Hora” e esse é consultado periodicamente pela atividade “Monitorar Alarme”.

Após a identificação das principais funções, passa-se a projetar o statechart controlador desse nível. Esse statechart é chamado de “Controlador de Despertador” e sua especificação é a exibida na Figura 3.17 (b). Ele possui dois estados: “Aguardando” que representa o sistema quando este não está em funcionamento e “Trabalhando” que representa o sistema em funcionamento. Algumas das funções do relógio são descritas através das atividades associadas

aos estados do statechart, enquanto outras são descritas através de ações associadas às transições. O rótulo da transição é composto de evento/ação. Quando do disparo de uma transição, a ação associada é realizada de acordo com a semântica do comando ENTRY.

Diagrama de Atividades: Relógio Despertador

(a)

Statechart Correspondente ao Controlador do Despertador

(b)

Figura 3.17 - Diagrama de Atividades e Statechart do Relógio Despertador

As relações de interface entre as atividades de um mesmo nível podem ser observadas na Figura 3.17(a). Quando os fluxos de controle (eventos) são recebidos pelo controlador, as atividades associadas recebem o sinal de

ativação para serem executadas, como sugerido no sub-passo 1.3. O quarto sub-passo do passo 1, a elaboração do statechart correspondente, segue os conceitos propostos por Harel, que foram apresentados na seção 3.2.

A Figura 3.18 mostra o detalhamento da atividade “Monitorar Hora”. Quando o evento “ligado” é recebido pelo “Controlador de Despertador” esse sinal ativa “Monitorar Hora”. A partir daí, como pode ser observado na Figura 3.18 (a) as atividades realizadas são a leitura constante da hora do sistema com a sua colocação no depósito de dados e a exibição da hora do sistema. O statechart da atividade “Monitorar_Hora”, chamado de “Controlador de Hora”, não tem estados pois as atividades que ele controla são atômicas, ou seja, não necessitam de outro controlador para serem realizadas.

Diagrama de Atividade: Monitorar Hora
(a)

Statechart Correspondente ao Controlador de Hora
(b)

Figura 3.18 - Diagrama de Atividades e Statechart Correspondente à Parte que Controla a Hora do Relógio Despertador

Diagrama de Atividade: Monitorar Alarme

(a)

Statechart: Controlador de Alarme Ligado

(b)

Figura 3.19 - Diagrama de Atividades e Statecharts Correspondente ao Alarme

Na Figura 3.19 mostra-se o refinamento da atividade “Monitorar Alarme”, isto é, detalham-se as atividades que devem ser realizadas para o que o relógio soe o alarme na hora preestabelecida. O Diagrama de Atividade de “Monitorar Alarme”, exibido na Figura 3.19 (a), recebe o fluxo de controle “*acertado*” e ativa “Acertar Hora do Alarme”, que corresponde ao acerto da hora em que o relógio deve despertar. Essa atividade, por sua vez, coloca a hora escolhida para o relógio despertar, após o recebimento de “*confirmado*”, em um

Alarme Ligado”, a fim de realizar a comparação entre essa hora e a hora do sistema.

O Statechart exibido na Figura 3.19 (b) mostra o comportamento do sistema quando os fluxos de controle são recebidos e quais atividades o sistema realiza. A partir do momento em que o evento “*acertado*” foi recebido o evento “*confirmado*” ou “*cancelado*” é esperado e então ou ocorre a confirmação da hora escolhida ou o seu cancelamento.

Diagrama de Atividade: Monitorar Alarme Ligado

(a)

Statechart: Controlador de Alarme Ligado

(b)

Figura 3.20 - Diagrama de Atividade e Statecharts Correspondente ao Alarme Ligado

Mais um nível de detalhe é necessário para que o sistema seja completamente especificado. Ele refere-se à atividade "Monitorar Alarme Ligado" e o statechart "Controlador de Alarme Ligado". Eles são mostrados na Figura 3.20 (a) e (b), respectivamente. O controlador da Figura 3.20 (a) recebe dois fluxos de controle: "*parado*" e "*soneca*" e os fluxos de dados "*t*" e "*h*" que representam a hora do dia e a hora que o alarme deve soar, respectivamente. A Figura 3.20 (b) apresenta o comportamento do sistema quando está sendo realizado o controle do alarme ligado.

3.5.2.2. Transformar os Vários Statecharts em um Único e os Vários Diagramas de Atividades em um Único

Sendo este método a combinação das abordagens funcional e comportamental, após a especificação do sistema utilizando os recursos que a abordagem funcional oferece, passa-se à utilização dos recursos existentes na parte comportamental. Dessa forma, como foi mostrado no Capítulo 2, seção 2.3., a abordagem comportamental utiliza um diagrama de atividade para englobar todas as atividades realizadas pelo sistema e um statechart que contém todos os controladores de nível mais baixo.

A transformação para um único statechart que engloba todos os statecharts controladores das atividades de nível mais baixo segue as regras descritas a seguir:

- O statechart controlador de nível zero passa a ser o statechart do sistema e tem como filhos todos os statecharts de nível mais baixo;
- O tipo do estado do nível i , $i=1, 2, \dots$ é dado de acordo com as atividades que devem ser realizadas e que constam do diagrama de atividades. Se mais do que uma atividade do tipo DO, não

(a)

(b)

Figura 3.21 - Diagrama de Atividades Único e Statechart Único do Relógio Despertador

- atômica, for realizada o estado será do tipo AND, caso contrário, será do tipo XOR;
- Caso o statechart a ser representado em um estado seja grande demais, o recurso de decomposição de estados pode ser utilizado. Dessa forma o estado do statechart controlador será decomposto em um outro statechart contendo todos os detalhes inerentes ao statechart de nível i , $1 \leq i \leq n$, onde n é o último nível de atividades decompostas pela abordagem funcional;
- Quando uma transição cruza níveis do statechart, o seu estado origem permanece inalterado e o seu estado destino é obtido verificando-se entre os estados de níveis mais altos qual recebe o evento a ele associado;
- Esses passos devem ser aplicados repetidamente até o último nível especificado pela abordagem funcional.

Para transformar os vários diagramas de atividades em um único deve-se partir do diagrama de mais alto nível e expandir as atividades sucessivamente, preservando a hierarquia já existente nos diagramas construídos no passo 1. Em resumo, pode-se dizer que tanto os statecharts como os diagramas de atividades são englobados em um único, conservando a mesma semântica.

A Figura 3.21 (a) e (b) mostra o diagrama de atividade e o statechart reunidos em um único, respectivamente, para o sistema exemplo, o relógio despertador, sem mostrar os detalhes de ativação de ações e atividades, que em nada mudaram.

O controlador do sistema é formado por um estado tipo "XOR" chamado **Controlador_de_Despertador**, que representa o relógio despertador. Este é composto por outros dois estados, representando o relógio em seus estados ligado ou desligado, chamados de **Aguardando** e **Trabalhando**, respectivamente. A partir do momento em que o relógio é ligado, isto é, ocorreu o evento "*ligado*", o estado **Aguardando** é abandonado passando para o estado **Trabalhando**, que é um estado tipo AND, onde permanece, paralelamente, tanto lendo e mostrando ininterruptamente a hora do relógio como controlando o alarme.

O estado **Lendo_Hora** foi criado para representar a periodicidade com que as leituras ocorrem. O statechart foi construído conservando-se os nomes de estados/transições utilizados no passo anterior quando da elaboração dos statecharts controladores do diagrama de atividades. Outros comentários sobre a construção dos statecharts constam da subseção seguinte quando eles são validados utilizando-se o ambiente StatSim.

3.5.2.3. Validar o Statechart Usando o StatSim

Os nomes de eventos, rótulos de estados e transições do statechart gerado no passo anterior (Figura 3.21b) devem ser adaptados de acordo com a sintaxe existente no ambiente StatSim. Os nomes de estados sofreram poucas alterações, como a substituição dos caracteres em branco por “sublinhado” e a retirada de preposições existente nos nomes compostos. As transições precisaram sofrer algumas modificações, de acordo com a sintaxe existente no ambiente, que serão descritas a seguir. O operador lógico “v” que aparece nos rótulos das transições foi substituído, no ambiente StatSim, por “ou”. O operador lógico “^” foi substituído por “;” quando da declaração das ações associadas às transições.

O estado **Lendo_Hora**, tem a função de representar as leituras que ocorrem no mundo real e exibir a hora atual. Como no StatSim não há possibilidade de se obter a hora atual isso será simulado através do incremento da variável t de uma unidade a todo passo, isto corresponde a $t := t + 1$, que é uma ação associada à transição (**Lendo_Hora**, *passo decorrido*, **Lendo_Hora**). Essa transição é implementada pelo evento nulo “*lambda*” ou seja, dispara em todo passo, simulando a mudança constante da hora.

A transição (**Esperando_Tocar**, após 2 minutos, **Despertando**) é a transição (**Esperando_Tocar**, *após_2min*, **Despertando**) e foi implementada pelo evento interno “ $t_out(soneca, 2)$ ”. Após decorridos dois passos que o evento “*soneca*” ocorreu, a transição é disparada com o alarme soando novamente quando o estado **Esperando_Tocar** estiver ativo.

A transição (**Aguardando_Despertar**, $t=h$, **Despertando**) é implementada pelo evento " $\text{lambda}[cr(t=h)]$ ". O operador "cr" é usado para que a transição dispare levando em consideração o valor atual das variáveis " t " e " h ".

O evento interno t_out foi utilizado para implementar a transição (**Despertando**, *após_3min_ou_parado*, **Aguardando_Despertar**). Após três passos da ocorrência do evento "*soneca*" ou da coincidência das horas, ou ainda se o botão "*pare*" for pressionado, o alarme deve ser desligado. Isso é representado através do evento " $\text{parado or } t_out(\text{lambda}[cr(t=h) \text{ or } \text{soneca}, 3])$ ". No statechart da Figura 3.21 (b) essa transição é a correspondente à (**Despertando**, após 3 minutos ou parado, **Aguardando_Despertar**). As transições que envolvem tempo são representadas pelo evento interno "time_out" sendo o tempo expresso em número de passos.

As atividades/ações realizadas foram inseridas quando da edição do statechart. Exemplos desses casos são as atividades que habilitam/desabilitam botões. Quando esse statechart for simulado, pode-se observar o seu comportamento e textualmente, através do Log existente, pode-se observar as atividades que estão sendo realizadas. Alguns estados têm ações associadas, são aquelas tarefas que devem ser realizadas instantaneamente, essas foram inseridas utilizando-se "ENTRY". Quando um estado é destino de diversas transições com a mesma ação, essa ação pode ser associada ao estado com a cláusula "ENTRY" ao invés de ficar associada a cada transição. O mesmo ocorre para quando o estado é origem, nesse caso, deve-se utilizar "EXIT" pois as ações devem ser realizadas a partir do momento que o estado é desativado. O statechart da Figura 3.21 (b) não apresenta as atividades nem junto às transições nem junto aos estados para não sobrecarregar o diagrama.

As atividades que têm refinamento são realizadas nos estados decompostos. Por exemplo, o estado **Controlador_Alarme_Ligado** do statechart da Figura 3.19 tem como atividade *Monitorar_Alarme_Ligado*, essa atividade é diluída em outras que serão realizadas nos estados que o compõem.

Após a especificação do sistema-exemplo e sua edição no ambiente StatSim, tem-se o statechart mostrado na Figura 3.22. Esse será agora simulado a

fim de validar o modelo criado. Foram utilizadas as simulações interativa e programada para validá-lo.

Figura 3.22 - Statechart Com Especificação não Correta para o Relógio Despertador

Foram efetuadas várias simulações, com início por “default” e várias seqüências de eventos supondo operações típicas do despertador e operações com erros em alguns pontos. O modelo não correspondeu ao esperado quando a seguinte seqüência de eventos ocorreu: *ligado*, *acertado*, *confirmado*, $\lambda[cr(t=h)]$, *parado* e *acertado*. A especificação apresentada não permite que o relógio seja novamente acertado, para que desperte em outra hora desejada, quando o alarme disparou e o botão *pare* foi pressionado ou quando o relógio pára depois de soar por três minutos. Um novo acerto só é possível se o botão *cancela* for pressionado. Isso pode ser observado pelo statechart editado no ambiente StatSim com a simulação mostrada na Figura 3.22. Quando o evento *parado* ocorre o estado **Aguardando_Despertar** é ativado, o que não permite que

a transição (Aguardando_Alarme_Desligado, *acertado*, Acertando_Hora_Alarme) seja disparada.

Diagrama de Atividade: Monitorar Alarme

(a)

Statechart: Controlador de Alarme Ligado

(b)

Figura 3.23 - Diagrama de Atividades e Statechart Corrigidos Correspondentes ao Alarme

Diagrama de Atividade: Monitorar Alarme Ligado

(a)

Statechart: Controlador de Alarme Ligado

(b)

Figura 3.24 - Diagrama de Atividades e Statechart Correspondentes ao Alarme Ligado

Os diagramas de atividades e os statecharts da Figura 3.19 e 3.20 devem ser alterados, corrigindo o erro detectado. Essas figuras são apresentadas a seguir. O **Controlador de Alarme** da atividade Monitorar Alarme (Figura 3.19) passa a receber o fluxo de controle *parado* e o sinal de que o alarme está soando a três minutos (*após 3 minutos*) da atividade Monitorar Alarme Ligado, como mostra a Figura 3.23.

A Figura 3.24 mostra o diagrama de atividades e o respectivo statechart quando o alarme é ligado. Quando o botão *pare* (evento "*parado*") for pressionado o controle do statechart é passado para um nível acima permitindo desse modo que um novo acerto seja realizado. O mesmo acontece quando o alarme fica tocando por mais de três minutos consecutivos. Isso é representado no statechart pela seta que representa graficamente a transição, cruzando o limite do estado. Essa transição é (*Despertando, após 3minutos v parado / hab_botão_cancela ^ hab_botão_acerto*).

A Figura 3.25 mostra o diagrama de atividades e o statechart com a nova especificação realizada se adequando à realidade.

Essa alteração realizada no statechart editado no ambiente StatSim correspondeu a remover a transição anterior e criar um nova de mesmo rótulo que a anterior, *após_3min_ou_parado*, porém o estado destino foi alterado, ou seja, deixou de ser **Aguardando Despertar** e passou a ser **Aguardando com Alarme Desligado** (**Aguardando_Alarme_Desligado**, no StatSim). O statechart editado no ambiente StatSim é o mostrado na Figura 3.26.

Figura 3.25 - Diagrama de Atividades e Statechart Após as Correções Realizadas

Figura 3.26 - Statechart do Relógio Despertador Editado no Ambiente StatSim

A Tabela 3.4 mostra as ações associadas às transições especificadas para o exemplo do relógio despertador, quando editado e simulado no ambiente StatSim. Utilizam-se os prefixos “des” significando desabilitar e “hab” significando habilitar.

Na Tabela 3.5 são apresentadas as atividades/ações associadas aos estados do statechart editado e simulado no ambiente StatSim para o exemplo do relógio despertador.

Tabela 3.4 - Ações Associadas às Transições

Ações	Transição
des_botão_acerto	(Acertando_Hora_Alarme, <i>confirmado</i> , Controlador_Alarme_Ligado)
des_botão_cancela	(Despertando,, <i>parado_ou_soando_3min</i> ,,Aguardando_Alarme_Desligado) (Controlador_Alarme_Ligado, <i>cancelado</i> , Aguardando_Alarme_Desligado)
des_botão_confirma	(Acertando_Hora_Alarme, <i>confirmado</i> ,Controlador_Alarme Ligado)
des_botão_liga	(Aguardando, <i>ligado</i> , Trabalhando)
des_botões_ligados	(Trabalhando, <i>desligado</i> , Aguardando)
hab_botão_acerto	(Aguardando, <i>ligado</i> , Trabalhando) (Despertando,, <i>parado_ou_soando_3min</i> ,,Aguardando_Alarme_Desligado) (Despertando,, <i>cancelado</i> , Aguardando_Alarme_Desligado)
hab_botão_liga	(- , default, Aguardando)
hab_botão_cancela	(Acertando_Hora_Alarme, <i>confirmado</i> ,Controlador_Alarme Ligado)
hab_botão_confirma	(Aguardando_Alarme_Desligado, <i>acertado</i> ,Acertando_Hora Alarme)
hab_botão_desliga	(Aguardando, <i>ligado</i> , Trabalhando)
hab_mostrador_hora	(Aguardando, <i>ligado</i> , Trabalhando)

Tabela 3.5. - Atividades e Ações Associadas aos Estados

Atividade/Ação	Estados
des_botão_pare	Despertando
des_botão_soneca	Despertando
hab_botão_pare	Despertando
hab_botão_soneca	Despertando
Ler_Hora	Lendo_Hora
Mostrar_Hora	Lendo_Hora
Soar_Alarme	Despertando

Novamente foram utilizadas as simulações interativas e programada para validar a especificação. Várias seqüências de eventos foram utilizadas para teste sendo que o modelo correspondeu ao esperado. A Figura 3.27 mostra o passo da simulação correspondente ao início por "default" (Default Configuration). Através do Log existente no ambiente pode-se observar o comportamento funcional em cada estado, a cada passo de simulação, através das atividades que devem ser realizadas no mundo real.

Figura 3.27 - Início da Simulação Interativa do Relógio Despertador (Default Configuration)

A Figura 3.28 mostra a configuração do sistema-exemplo após a ocorrência dos eventos "ligado" e "acertado", como pode ser visto através da opção LOG. Nesse momento deve-se alterar o valor da variável h que representa o horário no qual o alarme deve soar. Para esse exemplo, h recebeu o valor 4. Pode-se observar pelo LOG que as atividades realizadas são as descritas na

Figura 3.19 (b) relacionadas ao acerto do alarme e à atividade “Ler Hora” que é realizada periodicamente (a cada minuto). No ambiente Statsim esse período é representado por um passo, assim, a hora é lida a todo passo.

Figura 3.28 - Acertando a Hora em que o Relógio Deve Despertar

A Figura 3.29 mostra o comportamento do sistema especificado através do statechart no momento em que a hora do dia (t) e a hora ajustada para o relógio despertar (h) coincidem, isto é, o estado **Despertando** fica ativo. Em seguida, o evento “soneca” ocorre, causando a mudança do estado **Despertando** para o estado **Esperando_Tocar**, como mostra a Figura 3.30. Esse estado fica ativo por dois minutos, por exemplo, sendo que depois desse tempo é desativado e o estado **Despertando** é novamente ativado. Na especificação em statecharts isso é representado pela transição rotulada por (**Esperando_Tocar**, *após_2min*, **Despertando**). Pode ocorrer que o alarme fique soando por três minutos sem parar, por exemplo, desligando por conta própria. Foi assumido, para esse exemplo, que um minuto é representado por um passo no ambiente StatSim. A

transição rotulada por (Despertando, após_3min_ou_parado, Aguardando_Alarme_Ligado) representa esse fato.

Figura 3.29 - Alarme Soando

A qualquer momento em que o estado **Controlador_Alarme_Ligado** estiver ativo pode ocorrer o evento “cancelado”, que como próprio nome diz cancela todas as operações selecionadas. O LOG da Figura 3.31 exibe as atividades que são realizadas quando os evento “parado” ocorre. Essas são as mesmas quando o relógio permanece tocando por três minutos e depois pára, o que é representado pelo evento interno *t_out(soneca or despertar_iniciado, 3)*.

A Figura 3.32 mostra o comportamento do sistema quando os eventos “cancelado” e “desligado” ocorrem. Através do LOG pode-se observar as atividades que devem ser realizadas correspondentes a esses eventos. Quando o

evento “desligado” ocorre o estado **Aguardando**, em **Controlador do Despertador**, é o único que está ativo e assim permanece até que o relógio recomece o seu funcionamento.

Figura 3.30 - Evento Soneca Ocorreu

A simulação programada também foi utilizada como forma de validação do modelo construído. O programa LCE mostrado na Figura 3.33 tem 40 passos de execução sendo que o evento “*ligado*” ocorre no primeiro passo. A partir do passo 4, a cada 12 passos, ocorre o evento “*acertado*” onde *h* recebe o valor atual de *t* acrescido de 3 unidades. A partir do quinto passo o evento “*confirmado*” ocorre, sendo repetido, também, a cada doze passos. Os eventos “*soneca*, *cancela* e *desligado*” ocorrem com percentagens diferentes, ou seja, para “*soneca*” tem-se a probabilidade de 20% de ocorrência em cada passo enquanto que “*desligado* e *cancela*” tem a probabilidade de 5% de ocorrência em cada passo.

Figura 3.31 - Evento Parado Ocorreu

Figura 3.32 - Evento Desligado Ocorreu

Após a execução desse programa o relatório de passos é obtido, bem como o relatório estatístico que é o mostrado na Figura 3.34. O relatório estatístico mostra informações relacionadas com os estados, eventos, transições e variáveis do statechart. Pode-se observar nesse relatório que no término da simulação, ao fim dos 40 passos, o relógio estava trabalhando (**Aguardando** - "off"), sendo realizada a atividade de leitura da hora atual (**Lendo_Hora** - "on") e, paralelamente, o acerto de nova hora a despertar (**Acertando_Hora_Alarme** - "on"). O estado que realiza a leitura da hora do sistema foi o que esteve ativo por maior parte do tempo. Pode-se observar que o evento "*parado*" não ocorreu nessa simulação, já o evento interno que simula a leitura da hora atual ocorreu em quase todos os passos. Essas informações são dadas no relatório estatístico, para todos os eventos e transições. Este tipo de sistema não é muito indicado para a simulação programada pois os eventos ocorrerem de forma sequencial, embora tenha sido possível observar que o sistema se comportou de acordo com o esperado. A simulação programada se aplica melhor aos sistemas em que há o disparo de vários eventos ao mesmo tempo e, pela simulação interativa, nem sempre todas as possibilidades são vislumbradas.

```
GLOBAL PARAMETERS
NUMBER_OF_STEPS = 40

DECLARATIONS
EVENTS cancela, desligado 5%
EVENTS soneca 20%

SIMULATION

    FOR STEP 1 EACH 10 STEPS DO
    BEGIN
        EVENTS(ligado)
    END

    FOR STEP 4 EACH 12 STEPS DO
    BEGIN
        EVENTS(acertado)
        h:=t+3
    END

    FOR STEP 5 EACH 12 STEPS DO
    BEGIN
        EVENTS(confirmado)
    END

END SIMULATION
```

Figura 3.33 - Programa LCE para o Exemplo do Relógio Despertador

STATECHART: clock2
NUMBER OF STEPS: 40

STATE	PERCENTAGE OF STEPS IN WHICH WAS ACTIVE	FINAL SITUATION	AVERAGE NUMBER OF CONSECUTIVE STEPS IN THIS STATE
Aguardando	2.50	off	0.5000
Acertando_Hora_Alarme	5.00	on	1.0000
Aguardando_Despertar	22.50	off	3.0000
Lendo_Hora	97.50	on	1.0000
Esperando_Tocar	12.50	off	2.5000
Despertando	22.50	off	2.2500
Aguardando_Alarme_Desligado	35.00	off	4.6667

EVENTS	NUMBER OF STEPS IN WHICH THIS EVENT HAPPENED	PERCENTAGE OF STEPS IN WHICH THIS EVENT WAS AVAILABLE
default_com_acao	1	100.00
parado	0	100.00
cancelado	2	100.00
confirmado	3	100.00
soneca	7	100.00
ligado	4	100.00
desligado	1	100.00
acertado	4	100.00

TRANSITIONS SUCCESS	NUMBER OF STEPS IN WHICH THIS TRANSITION WAS FIRED	NUMBER OF STEPS IN WHICH THIS TRANSITION WAS EVALUATED	RATE OF FIRING
apos_3min_ou_parado	2	11	18.18
t=h	2	9	22.22
cancelado	1	7	14.29
apos_2min	2	6	33.33
ligado	2	4	50.00
soneca	2	4	50.00
acertado	2	2	100.00
passo_decorrido	37	37	100.00
desligado	1	11	9.09
confirmado	1	1	100.00

VARIABLE	AVERAGE VALUE	FINAL VALUE
t	18.9750	37
h	10.9000	39

Figura 3.34 - Relatório Estatístico Resultante da Simulação Programada para o Exemplo do Relógio Despertador.

O relatório de passos exibe, como o próprio nome diz, todas as informações relacionadas a: eventos internos e externos disparados, estados ativos, mensagens (se existirem), o estado ativo e qual deixou de estar e os valores das variáveis naquele passo. A Figura 3.35 apresenta um trecho do relatório de passos. Reticências (. . .) são colocadas para representar as informações não significativas que foram omitidas desse relatório.

Statechart in Simulation: clock2
Number of Simulated Steps: 40

----- Step 1 -----

Configuration of Active States:

Controlador_de_Despertador

Trabalhando

Controlador_Hora

Lendo_Hora

Controlador_Alarme

Aguardando_Alarme_Desligado

Events triggered : ligado,soneca

Internal Transitions :

States Turned On :

Trabalhando,Controlador_Hora,Lendo_Hora,Controlador_Alarme,Aguardando_Alarme_Desligado

States Turned Off : Aguardando

Variable	Current Value	Average Value
t	0	0.0000
h	0	0.0000

Messages :

----- Step 2 -----

. . .

```

Events triggered      :
Internal Transitions : passo_decorrido
States Turned On     :
States Turned Off    :

```

. . .

----- Step 4 -----

Configuration of Active States:

```

Controlador_de_Despertador
  Trabalhando
    Controlador_Hora
      Lendo_Hora
        Controlador_Alarme
          Acertando_Hora_Alarme

```

```

Events triggered      : acertado
Internal Transitions : passo_decorrido
States Turned On     : Acertando_Hora_Alarme
States Turned Off    : Aguardando_Alarme_Desligado

```

Variable	Current Value	Average Value
t	3	1.5000
h	5	1.2500

Messages :

----- Step 5 -----

Configuration of Active States:

```

Controlador_de_Despertador
  Trabalhando
    Controlador_Hora
      Lendo_Hora
        Controlador_Alarme
          Controlador_Alarme_Ligado
            Aguardando_Despertar

```

```

Events triggered      : confirmado
Internal Transitions : passo_decorrido
States Turned On     : Controlador_Alarme_Ligado,Aguardando_Despertar
States Turned Off    : Acertando_Hora_Alarme

```

Variable	Current Value	Average Value
t	4	2.0000
h	5	2.0000

Messages :

----- Step 7 -----

Configuration of Active States:

Controlador_de_Despertador

Trabalhando

Controlador_Hora

Lendo_Hora

Controlador_Alarme

Controlador_Alarme_Ligado

Despertando

Events triggered :

Internal Transitions : t=h,passo_decorrido

States Turned On : Despertando

States Turned Off : Aguardando_Despertar

Variable	Current Value	Average Value
t	6	3.0000
h	0	2.1429

Messages :

----- Step 9 -----

. . .

Events triggered : soneca

Internal Transitions : passo_decorrido

States Turned On : Esperando_Tocar

States Turned Off : Despertando

Variable	Current Value	Average Value
t	8	4.0000
h	0	1.6667

Messages :

----- Step 11 -----

. . .

Events triggered : ligado

Internal Transitions : apos_2min,passo_decorrido

States Turned On : Despertando

States Turned Off : Esperando_Tocar

Variable	Current Value	Average Value
t	10	5.0000
h	0	1.3636

Messages :

----- Step 14 -----

Configuration of Active States:

```

Controlador_de_Despertador
  Trabalhando
    Controlador_Hora
      Lendo_Hora
        Controlador_Alarme
          Controlador_Alarme_Ligado
            Aguardando_Despertar

```

Events triggered :

Internal Transitions : passo_decorrido, apos_3min_ou_parado

States Turned On : Aguardando_Despertar

States Turned Off : Despertando

Variable	Current Value	Average Value
t	13	6.5000
h	0	1.0714

Messages :

----- Step 16 -----

Configuration of Active States:

```

Controlador_de_Despertador
  Trabalhando
    Controlador_Hora
      Lendo_Hora
        Controlador_Alarme
          Controlador_Alarme_Ligado
            Aguardando_Despertar

```

Events triggered : acertado

Internal Transitions : passo_decorrido

States Turned On :

States Turned Off :

Variable	Current Value	Average Value
t	15	7.5000
h	17	2.0000

Messages :

----- Step 28 -----

Configuration of Active States:

```

Controlador_de_Despertador
  Trabalhando
    Controlador_Hora
      Lendo_Hora

```

```

Controlador_Alarme
  Aguardando_Alarme_Desligado

Events triggered      : acertado, cancelado
Internal Transitions : passo_decorrido
States Turned On     : Aguardando_Alarme_Desligado
States Turned Off    : Controlador_Alarme_Ligado, Aguardando_Despertar

Variable      Current Value      Average Value
t              27                 13.5000
h              29                 2.7857

Messages :
-----
----- Step 40 -----
Configuration of Active States:
  Controlador_de_Despertador
    Trabalhando
      Controlador_Hora
        Lendo_Hora
          Controlador_Alarme
            Acertando_Hora_Alarme

Events triggered      : acertado
Internal Transitions : passo_decorrido
States Turned On     : Acertando_Hora_Alarme
States Turned Off    : Aguardando_Alarme_Desligado

Variable      Current Value      Average Value
t              37                 18.9750
h              39                 10.9000

Messages :
-----

```

Figura 3.35 - Trecho do Relatório de Passos da Simulação Programada para o Exemplo do Relógio Despertador.

3.5.2.4. Projetar

A partir do modelo obtido nos passos anteriores passa-se a construir um projeto do sistema visando sua futura implementação. A partir do diagrama de

atividades e do statechart apresentados na Figura 3.25, dois módulos devem existir para que a simulação do relógio possa ser realizada. Um módulo faz o controle da hora, lendo-a constantemente e enviando-a para o outro módulo, que é o que controla o alarme.

Módulo Monitora Alarma (a)

Módulo Monitora Hora (b)

Figura 3.36 - Esboço dos Processos do Sistema

A Figura 3.36 apresenta um esboço da arquitetura do sistema, listando os procedimentos que irão compor cada um dos módulos. Para o módulo que controla o alarme, optou-se por construir um procedimento para desempenhar

as funções de cada um dos botões existentes no relógio, um procedimento principal que cuida da interface com o usuário e procedimentos auxiliares que viabilizem a realização das funções do relógio. O módulo que controla a hora tem um procedimento que faz a leitura constante da hora do sistema (computador) e outros auxiliares para as funções de comunicação.

Esses processos estão constantemente se comunicando entre si. Um processo servidor (**Controlador_Hora**) e um processo cliente (**Controlador_Alarme**) realizam as atividades descritas anteriormente.

3.5.2.5. Implementar

Utilizando-se do modelo produzido pôde-se realizar a implementação do sistema exemplo utilizando linguagem C, em estações de trabalho Sun e usando também o Sistema de Gerenciamento de Interface do Usuário (SGIU) Xview.

Para representar o paralelismo existente no statechart da Figura 3.25, são utilizadas as primitivas do sistema operacional Unix para comunicação de processos. Os dois processos implementados correspondem aos estados do statechart chamados de **Controlador_Hora** e **Controlador_Alarme** e são denominados de HORA.C e RELOGIO.C, respectivamente.

Os processos se comunicam através de “sockets”, manipulados pelas primitivas especiais do Unix. Cada processo pode escrever e ler do seu socket por ele ser um porto de comunicação. O “frame” transmitido entre os dois processos é composto da hora do sistema e de mensagens de controle. Cada um dos processos fica num “loop” esperando que chegue uma mensagem para ser lida. Os processos são avisados da chegada de um “frame” pelo sinal (interrupção) do sistema.

O sistema também utiliza as primitivas gráficas do Xview para implementar as interfaces gráficas. A Figura 3.37 mostra o esboço do relógio quando de sua ativação de acordo com a especificação feita. Pode-se observar

que o “layout” segue o descrito na seção 3.5.1. O hora e data atual obtidas do sistema aparecem no relógio exibido na parte superior das Figuras 3.37 a 3.41.

Figura 3.37 - Esboço Inicial do Relógio

A Figura 3.38 mostra que o botão “Acerto” foi pressionado e a hora desejada para que o alarme soe foi escolhida. Quando o botão “Confirma” é selecionado o relógio tem a configuração mostrada na Figura 3.39. No momento da coincidência da hora selecionada para despertar com a hora do dia (sistema) a mensagem “**ACORDE**” é escrita na região do visor do relógio, como mostra a Figura 3.40. Quando o botão “soneca” é selecionado o alarme pára de soar e recomeça após um minuto. Quando o “Pare” é acionado o alarme pára de soar e o relógio fica mostrando a hora do dia, somente os botões “Cancela” e “Desliga” estão habilitados, como pode ser visto na Figura 3.41. O botão “Cancela” quando pressionado permite que uma nova hora seja escolhida para que o relógio desperte, ou seja, os botões rotulados por “Acerto” e “Desliga” estão habilitados. A simulação do relógio termina quando o botão “Desliga” é acionado e o “frame” correspondente ao relógio é retirado da tela.

Figura 3.38 - Horário de Despertar Acertado

Figura 3.39 - Botão Confirma Pressionado

Figura 3.40 - Relógio com Alarme Soando

Figura 3.41 - Botão Pare Pressionado

3.5.2.6. Testar

Após a implementação, a fase de testes deve ser realizada. As seqüências de eventos utilizadas nas simulações do ambiente StatSim devem ser utilizadas para que se verifique se o protótipo implementado se comporta da forma desejada.

No caso do relógio despertador foram selecionadas as mesmas seqüências de eventos utilizadas quando da simulação interativa e programada e aplicada ao protótipo gerado. O seu comportamento produziu os mesmos resultados que a especificação realizada. Dessa forma, pode-se dizer que a especificação está atendendo aos requisitos solicitados.

3.6. Considerações Finais

Neste capítulo mostrou-se o uso do ambiente StatSim e as extensões realizadas por este trabalho e outros anteriores.

Ao se permitir que o ambiente StatSim tenha a edição e a simulação de atividades possibilita-se duas visões distintas do sistema ao engenheiro de software. Uma visão, a comportamental, que pode ser observada através dos diversos tipos de simulação depois que o sistema foi especificado em Statechart e editado no ambiente e a funcional, observada através da opção Log existente no modo de simulação. As atividades, quando executadas, mostram as tarefas que o sistema deve realizar.

Conforme observado, principalmente nas seções 3.4 e 3.5, teve-se a oportunidade de interagir bastante com o ambiente StatSim. Não só usando o ambiente mas também efetuando alterações, como a introdução de edição e simulação de transição "default" com ação, transição com evento nulo. Pôde-se também expandir os recursos do ambiente, com a introdução da visão funcional nos sistemas reativos especificados por statechart. Isso é salientado principalmente para frisar que a condução da engenharia reversa descrita no capítulo 5, nesse ambiente, só é possível a alguém que tenha alguma familiaridade com o software em questão.

O protótipo gerado para o exemplo do despertador valida a especificação realizada e sugere caminhos de pesquisa para a geração de protótipos a partir da simulação de statecharts. A partir da definição de requisitos de um sistema pode-se desenvolvê-lo seguindo as diretrizes apresentadas na seção 3.5.2, o que possibilita que as visões funcional e comportamental sejam integradas, além da utilização do ambiente Statsim como forma de validação da especificação realizada e não somente como ferramenta de apoio à especificação de sistemas.

Capítulo 4

Engenharia Reversa Orientada a Objetos

4.1. Considerações Iniciais

A reengenharia de software tem crescido muito nos anos 90 pois, segundo McCabe [58], as companhias tentam conter os custos para a manutenção dos sistemas de software e maximizar seus investimentos nos softwares existentes ao invés de os abandonar e criar novos softwares. É comum que grande parte desses sistemas satisfaça razoavelmente aos requisitos dos usuários, mas não disponha de documentação de análise e projeto ou que estas estejam desatualizadas, dificultando manutenções de qualquer tipo. Tais sistemas são denominados de sistemas herdados (legacy code). O processo de reengenharia é composto de diversas fases, sendo uma delas a engenharia reversa. Um dos objetivos para que se realize engenharia reversa em sistemas herdados refere-se aos ganhos que podem ser obtidos com as melhorias realizadas no sistema e com as futuras manutenções. Após a engenharia reversa de um sistema as alterações tornam-se mais fáceis, reduzindo, dessa forma, os gastos na fase de manutenção.

A Engenharia Reversa vem sendo aplicada a sistemas herdados visando futura reengenharia desses sistemas ou somente para representá-los de outra forma ou com nível mais alto de abstração. As diversas etapas da engenharia reversa permitem que se identifique quais partes do sistema devem ser re-implementadas, prepara um modelo de análise e faz um mapeamento entre o modelo de análise preparado e o sistema existente.

A abordagem de orientação a objetos é atualmente a tecnologia mais adequada para especificação de sistemas. Tem entre as vantagens a reutilização de código, linguagens de apoio, etc.. Assim essa abordagem foi a escolhida para servir de apoio para a engenharia reversa de sistemas desenvolvidos sem essa abordagem.

A ausência de diretrizes concretas para a realização de engenharia reversa orientada a objetos de sistemas desenvolvidos sem seguir essa abordagem foi o estímulo para o início das pesquisas, escolhendo-se o método Fusion por produzir modelos consistentes entre si e com uma ligação clara e objetiva entre as fases de análise e projeto. Essa afirmação foi totalmente corroborada pelos estudos que se realizou sobre o referido método. É importante notar que a ligação entre a análise e o projeto é de extrema importância na retroação do projeto para a análise, que caracteriza qualquer abordagem de engenharia reversa.

Na seção 4.2 apresenta-se o contexto da engenharia reversa dentro da atividade de reengenharia, na seção 4.3 é resumido o método Fusion, mostrando-se na seção 4.4 as diretrizes detalhadas a que se chegou para a engenharia reversa. Na seção 4.5 apresenta-se a interligação entre os passos do método e os documentos produzidos. Na seção 4.6 apresentam-se algumas métricas que possibilitam uma avaliação das características de orientação a objetos do sistema existente e da facilidade para reengenharia. Na seção 4.7 é realizado um estudo quando em uma reengenharia tem-se ampliação de funcionalidade e na seção 4.8 são apresentadas as considerações finais deste capítulo.

4.2 Engenharia Reversa no Contexto de Reengenharia

Chikofsky, em [12], comenta a confusão existente na terminologia de reengenharia usada tanto na área técnica como na área de marketing. Em seu artigo ele define seis termos: *engenharia avante*, *engenharia reversa*, *redocumentação*, *recuperação do projeto*, *reestruturação* e *reengenharia*, tendo como objetivo não a criação de novos termos mas sim a racionalização dos termos

que já vêm sendo usados. As definições resultantes são aplicadas em um processo de engenharia em desenvolvimento sem considerar o grau de automação que foi aplicado.

Engenharia avante é definida como o processo tradicional que se inicia com abstrações de alto nível, lógicas e projetos independentes da implementação para se chegar à implementação física do sistema, ou seja, o processo de ir do projeto lógico para o projeto físico do sistema.

Engenharia Reversa é definida como o processo de análise de um sistema para identificar as componentes e suas interrelações de modo a criar representação do sistema em uma outra forma ou em um nível mais alto de abstração. Engenharia Reversa geralmente envolve a extração de artefatos e construções do projeto ou síntese de abstrações que são menos dependentes da implementação. Pode-se realizar engenharia reversa a partir de qualquer nível de abstração e para qualquer estágio do ciclo de vida. Ela é um processo de exame, não um processo de troca ou duplicação. Durante a fase do ciclo de vida, engenharia reversa cobre um grande domínio, começando da implementação existente, capturando ou recriando o projeto e decifrando os requisitos atualmente implementados pelo sistema em estudo. Há várias sub-áreas de engenharia reversa, sendo que as duas mais fortes são redocumentação e recuperação do projeto.

Redocumentação é a criação ou revisão de uma representação semanticamente equivalente com o mesmo nível de abstração que está sendo tratado. As formas resultantes da representação são usualmente consideradas como visões alternativas (por exemplo, fluxo de dados, estrutura de dados e fluxo de controle) necessárias para uma exposição às pessoas. É a mais simples e a mais antiga forma de engenharia reversa e muitos a consideram como a que menos interfere, a forma mais fraca de reestruturação. O prefixo “re” implica a intenção de descobrir documentação sobre o sistema que existiu ou que ainda existe. Alguns documentos consultados para a redocumentação são listagem de código, diagramas gerados (diagramas criados a partir do código, refletindo o fluxo de controle ou a estrutura do código), geração de listas de referência cruzada. O objetivo principal desses documentos é fornecer um modo mais fácil para

visualizar as relações entre os componentes do programa de modo que se possa reconhecer e seguir por caminhos mais claros.

Recuperação do projeto é um subconjunto da engenharia reversa no qual o domínio do conhecimento, informações externas e dedução são adicionadas às observações do sistema alvo para identificar as abstrações de nível mais alto em torno daquelas obtidas diretamente do exame do sistema em si. Biggerstaff declara (de acordo com [12]), que a recuperação do projeto “recria abstrações de projeto com base numa combinação de código, documentação de projeto, experiência pessoal e conhecimentos sobre o domínio do problema. Para o completo entendimento do que um programa faz, como faz e porque faz, deve-se reproduzir todas as informações obtidas a partir da solicitação feita a uma pessoa. Assim tem-se um domínio mais restrito do que aquele que se têm a partir de solicitação livre de um usuário.

Reestruturação - é a transformação de uma representação para uma outra no mesmo nível de abstração, preservando o comportamento externo do sistema atual (funcionalidade e semântica). Reestruturação é frequentemente uma forma aparente, tal como código alternativo, para melhorar a estrutura do sistema no sentido tradicional de projeto estruturado. Tornou-se um termo conhecido devido a transformação de código para código, isto é, um código não estruturado para um estruturado. Muitos tipos de reestruturação podem ser realizados a partir do conhecimento da forma estrutural sem que se entenda o seu significado. Por exemplo, pode-se transformar um conjunto de comandos “if” em um “case” ou vice versa sem que se conheça o objetivo do programa ou algo sobre o domínio do problema. Enquanto a reestruturação cria novas versões que implementam ou propõem troca do sistema em estudo, ela não envolve novos requisitos. É normalmente usada como uma forma de manutenção preventiva para melhorar o estado físico do sistema com relação a algum padrão escolhido, melhor que o anterior. Ela também envolve o ajuste do sistema para que se encontrem novas restrições ambientais sem que ocorra a sua redefinição em níveis mais altos de abstração.

Reengenharia - também conhecida como “renovação” e “reclamation”, é quando se examina e se altera um sistema para reconstitui-lo de uma nova forma e a partir dessa reconstrução realiza-se a sua implementação.

Reengenharia geralmente inclui alguma forma de engenharia reversa (para encontrar um maior nível de abstração) seguido pela engenharia avante ou reestruturação. Uma forma de reengenharia é a mudança da técnica de implementação sem mudança na funcionalidade do sistema. Outra é a mudança da técnica da implementação com mudança na funcionalidade do sistema. Por exemplo, durante a reengenharia de sistemas de informação gerenciais, uma organização tem acesso a como o sistema implementa as regras do negócio em alto nível e faz modificações de acordo com as modificações dos negócios para o futuro.

Chikofsky afirma que há muita confusão entre os termos reengenharia e reestruturação. Um grupo de usuários da IBM, *Guide*, define *aplicação de reengenharia* como “o processo de modificação dos mecanismos internos de um sistema, ou programa ou estrutura de dados do sistema, onde não há troca da funcionalidade (alteram o *como* sem afetar o *o quê*). Para [12] isso é reestruturação. Porém, alerta que no mesmo texto da IBM, em um parágrafo anterior ao citado, os autores dizem que é “raro que a aplicação que sofre reengenharia não tenha troca de funcionalidade”.

Três etapas para reengenharia são definidas pelos autores em [58]: engenharia reversa, revisão do modelo de especificação e engenharia avante. A etapa de engenharia reversa, como o próprio nome diz, é o reverso do processo natural e tem como objetivo recuperar a capacidade funcional do sistema, ou seja, recuperar o modelo lógico do sistema existente em uma forma mais simplificada que pode ser revisada e atualizada para formar a base do novo sistema substituto. Os sistemas que devem passar por uma engenharia reversa são aqueles que foram desenvolvidos sem uma técnica de modelagem apropriada ou aplicações nas quais o modelo original não foi mantido compatibilizado em sincronia com o código fonte. Engenharia reversa é também usada para apresentar o modelo lógico de análise que mostra o que a aplicação faz sem se preocupar com como ela faz. A etapa da engenharia reversa fornece a garantia de que a substituição do sistema fornecerá toda a capacidade funcional que ele possuía anteriormente.

Jacobson, em [45], mostra três cenários possíveis para reengenharia: troca completa da implementação sem troca da funcionalidade; troca parcial da implementação sem troca de funcionalidade; troca da

funcionalidade. Para cada um dos cenários é fornecida a linha geral a ser seguida, mostrando como os dados são obtidos para que se realize sua análise e o mapeamento de cada elemento da análise para a implementação. Aí também sente-se falta de diretrizes para a engenharia reversa que deve preceder a re-engenharia.

Coleman, em [17], utiliza os mesmos conceitos que Chikofsky e cita as três possibilidades de reengenharia apresentadas por Jacobson [45], salientando que reengenharia é considerada quando há troca de funcionalidade. Apresenta um roteiro para que um sistema com mudança de funcionalidade seja reimplementado parcialmente. Esse roteiro consiste de sete passos, sendo que os três primeiros referem-se à fase de engenharia reversa e, os outros referem-se à engenharia avante. Na fase de engenharia reversa deve-se:

1- *Identificar parte do sistema*: deve-se identificar a parte do sistema que deve ser reimplementada bem como os agentes que interagem com ela.

2- *Preparar o modelo de análise*: com base na documentação obtida; manuais, código fonte, requisitos de especificação, modelos de projeto, esquema de base de dados, deve-se produzir o modelo de análise das componentes que devem passar por uma reengenharia. Isso deve enfatizar “o *quê*” o sistema faz, sem enfatizar o “*como*” será implementado.

3- *Mapear o modelo de análise para o sistema existente*: esse mapeamento deve refletir que para cada um dos componentes da análise e cada um dos relacionamentos no modelo de análise, deve existir um elemento na documentação do sistema que é consistente com o código fonte.

A seguir são apresentadas as diretrizes para engenharia avante: incorporação dos novos requisitos, avaliação da interface de novas componentes, projeto do novo subsistema, modificação do sistema antigo. Os autores [17] afirmam que os projetos de desenvolvimento de software em que se decidem por introduzir a tecnologia orientada a objetos em muitos casos vão se defrontar com um problema de reengenharia. Desenvolvimentos perfeitos que não envolvam códigos herdados são raros na prática. Como também é pouco usual ou pouco

provável que exista um produto que passe completamente por reengenharia usando uma tecnologia nova de uma única vez. Logo, a reengenharia conduzida por componentes fornece um caminho de migração com baixo risco.

A afirmação também de Coleman e outros, de que o processo de reengenharia descrito anteriormente é baseado na premissa de que um modelo de análise orientado a objetos pode ser usado para caracterizar um sistema que tenha sido projetado sem usar a técnica de orientação a objetos. Dependendo da qualidade e do nível da documentação do sistema legado, esforços significativos podem ser dispendidos sem que se consiga uma abstração apropriada da análise. Muito esforço poderá ser necessário para estabelecer uma interface apropriada entre o sistema antigo e o as novas componentes. Afirmam que o método Fusion fornece diretrizes seguras para a engenharia avante e que algumas experiências com reengenharia foram publicadas, porém somente tentativas de arcabouços de processos tem sido propostos.

Dietrich, em [20], salienta que existem duas razões para implementar sistemas orientados a objetos usando sistemas herdados, que são sistemas existentes os quais não foram desenvolvidos orientados a objetos. Primeiramente, os sistemas herdados satisfazem às necessidades da maioria de seus usuários, mas há necessidade de melhor programação de sua interface para estender ou atualizar suas funções. Em segundo lugar para verificar se tais sistemas têm partes do código que podem ser reusadas para implementar funções do sistema especificado segundo a abordagem de orientação a objetos. O autor comenta alguns pontos de uma implementação realizada com um sistema exemplo, principalmente para construção da interface, mas não são fornecidas diretrizes detalhadas para realizar a engenharia reversa que deve preceder a implementação.

Wong, em [98], afirma que nos sistemas grandes a redocumentação estrutural através de engenharia reversa torna-se a palavra chave. Reconstruir e efetivamente redocumentar o projeto do sistema de software existente é mais difícil do que o projeto inicial. Reconhecer as abstrações do sistema do mundo real é tarefa crucial tanto quanto projetar adequadamente as abstrações para novos projetos. Os autores comentam que a engenharia reversa reconstrói modelos estruturais pela identificação de componentes do sistema atual, descobrindo suas

dependências e gerando abstrações para gerenciar a complexidade, fornecendo “insights” que podem melhorar o desenvolvimento subsequente, facilitar a re-engenharia, etc. Afirmam, também, que a redocumentação estrutural é a engenharia reversa dos aspectos arquiteturais do sistema. Um sistema chamado Rigi está sendo por ele desenvolvido, enfocando os aspectos arquiteturais para o entendimento de programas. Também não apresenta diretrizes para a realização da engenharia reversa.

Como ponto principal da reengenharia de sistema tem-se a engenharia reversa para a obtenção das informações até então desconhecidas ou mal documentadas nos sistemas existentes. Uma forma de recuperar essas informações é a utilização de técnicas que interessem à maioria dos projetistas quer pela facilidade de utilização, quer por serem comprovadas como as mais completas da atualidade. Dessa forma, o método aqui proposto de engenharia reversa tem o apoio da abordagem de orientação a objetos proposta por Coleman e outros em [17] e conhecida como método Fusion, que será apresentado na seção seguinte.

4.3. O Método Fusion

O método Fusion foi criado por Coleman e outros, reunindo as que, em sua opinião, são as melhores técnicas propostas por outros métodos existentes [17]. Três fases distintas compreendem o método: análise, projeto e implementação. Os autores comentam que para a fase de análise a maior inspiração foi o método OMT [81] e propõem três modelos de análise: Modelo de Objetos, Modelo de Ciclo de Vida e Modelo de Operações.

A Figura 4.1, extraída de [17], mostra a interação existente entre as três fases do método e os documentos produzidos em cada uma delas. O levantamento dos requisitos necessários produz um Documento de Requisitos, que é informal e utilizado como fonte de informações para a **fase de análise**. Esta pode ser descrita através dos quatro passos seguintes:

Figura 4.1 - Visão Geral do Método Fusion

Valores possíveis de C	Significado
branco	"default", podem existir zero ou mais objetos
*	podem existir zero ou mais objetos relacionados
+	podem existir um ou mais objetos relacionados
N	número exato de objetos da classe relacionados
M..N	número mínimo e máximo de objetos relacionados entre as classes
■	marcador de totalidade. Todos os objetos participam do relacionamento

Figura 4.2 - Representação de Classes e Relacionamentos

Primeiro Passo - Desenvolver um Modelo de Objetos para o Domínio do Problema: tem por objetivo representar os conceitos existentes no domínio do problema e suas relações. Permite representar classes, atributos e relacionamentos entre classes, incluindo agregação e especialização/generalização. A notação é bastante semelhante à do Modelo de Objetos do método OMT. A análise deve iniciar em um alto nível de abstração, sendo que os detalhes devem ser inseridos só depois que a estrutura geral for considerada satisfatória.

A representação de Classe e Relacionamento é a mostrada na Figura 4.2 onde as classes são representadas por retângulos e os relacionamentos por losângos. *Regra i* representa o papel da classe na relação C e a cardinalidade do relacionamento.

Figura 4.3 (a) - Representação de Agregação

Figura 4.3 (b) - Representação de Especialização/Generalização de Classes (Classes Subtipos Não Disjuntas)

Há dois tipos especiais de relacionamentos que podem ser representados no Modelo de Objetos: agregação e especialização/generalização. A agregação de classes é mostrada na Figura 4.3(a), com as classes componentes contidas dentro da classe agregada. Uma classe agregada pode ser decomposta em outro Modelo de Objetos facilitando a definição dos níveis de abstração em modelos complexos. A especialização/generalização de classes é representada por um triângulo conectando uma classe supertipo aos seus subtipos. Quando o triângulo é cheio (Figura 4.3 (c)), significa que as classes

subtipos são disjuntas e, caso contrário, diz-se que as classes subtipos são não disjuntas (Figura 4.3 (b)).

Figura 4.3 (c) - Representação de Especialização/Generalização de Classes (Classes Subtipos Disjuntas)

Segundo Passo - Determinar a Interface do Sistema: consiste na identificação dos agentes, operações e eventos e em refinar o Modelo de Objetos do sistema construído no passo anterior adicionando-lhe limites. Para estabelecer os limites do sistema pode-se usar de cenários. A partir do Modelo de Objetos construído no primeiro passo, pode-se delinear o limite do sistema através de uma linha tracejada englobando o que deve ser considerado.

Terceiro Passo - Determinar um Modelo de Interface: consiste na elaboração do Modelo de Ciclo de Vida e do Modelo de Operações, como pode ser visto na figura 4.1.

O Modelo de Ciclo de Vida especifica através de expressões regulares as seqüências permitidas de interações com o ambiente em que o sistema pode participar. Para facilitar a elaboração do Modelo de Ciclo de Vida são construídos cenários do sistema mostrando o seu comportamento de acordo com os eventos gerados pelos agentes do ambiente e as respostas (eventos de saída) dadas pelo sistema a esses eventos de entrada. O ciclo é então generalizado a partir dos cenários. O ciclo de vida é semelhante ao usado por Jackson em JSD [44], mas ao invés de ser definido para cada classe, é definido globalmente para o sistema.

life cycle [Nome:] Expressão-Regular
 (nome-local = Expressão-Regular)*

Figura 4.4 - Modelo de Ciclo de Vida

O Modelo de Ciclo de Vida tem a sintaxe geral exibida na Figura 4.4., onde Nome especifica o Nome do sistema. As expressões regulares definem a seqüência de eventos em que o sistema pode participar durante toda a sua vida. Do mesmo modo que nas gramáticas elas definem as seqüências de símbolos da linguagem que podem ser aceitas pelo compilador.

Tabela 4.1 Operadores das Expressões Regulares do Modelo de Ciclo de Vida

Operadores	Expressão Resultante	Significado
. concatenação	$x . y$	x seguido de y
escolha	$x y$	ou x ou y
* repete 0 ou mais	x^*	repete x zero ou mais vezes
+ repete 1 ou mais	x^+	repete x uma ou mais vezes
[] opcional	$[x]$	x é realizado opcionalmente
concorrência	$x y$	x e y são realizados concorrentemente
() agrupamento para alterar a precedência	$(x y)^* . z$	x e y são intercalados e finalizados com z. Exemplo: xxyxyyz

As expressões regulares podem ser combinadas com operadores de forma a gerar todo o ciclo de vida do sistema. Considere x e y como expressões regulares, os operadores têm o significado mostrado na Tabela 4.1. A precedência de operadores em ordem decrescente é a seguinte: [], *, +, ., |, ||.

Enquanto o Modelo de Operações descreve o comportamento de uma operação individual do sistema, o Modelo de Ciclo de Vida descreve o comportamento completo de como sistema se comunica com o ambiente, desde a sua criação até o seu término.

O Modelo de Operações é decorrente do Modelo de Ciclo de Vida. A cada evento de entrada corresponde uma operação que é especificada por um gabarito textual. Nesse gabarito há entradas para descrever informalmente a operação e para especificar:

- a) os dados que são fornecidos junto com os eventos de entrada ou estão disponíveis globalmente;
- b) os objetos a que a operação pode ter acesso, mudando ou não seu estado;
- c) os eventos de saída que a operação gera e os agentes que os recebem;
- d) as pré-condições que devem ser satisfeitas e os resultados da operação, descritos como uma pós-condição.

O Modelo de Operações tem o gabarito exibido na Figura 4.5.

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

Figura 4.5 - Modelo de Operações

A palavra **supplied** precedendo um item, em **Lê**, indica que o identificador é um parâmetro da operação. A palavra **new** precedendo um identificador de objeto (item), em **Modifica**, indica que a operação cria um novo objeto na base de dados do sistema.

Quarto Passo - Checar os modelos da análise: verificar se o modelo é consistente e completo. Um modelo é dito consistente quando não existem contradições implícita ou explicitamente entre as partes do modelo. Um modelo é dito completo se todas as abstrações significativas do domínio foram captadas e

se o modelo de análise é condizente com as expectativas do usuário descritas no Documento de Requisitos.

A **fase de projeto** do método Fusion é baseada nos métodos CRC [95] e de Booch [7]. Durante o projeto estruturas de software são introduzidas para satisfazer as abstrações produzidas na fase anterior, a da análise. Cinco passos representam essa fase e são descritos a seguir.

Primeiro Passo: Desenvolver um Grafo de Interação de Objetos para cada uma das operações do Modelo de Operações. Este grafo mostra os objetos que participam da execução de uma operação e a sequência de trocas de mensagens entre eles. Com isso pode-se definir o objeto controlador da operação (os demais são colaboradores) e os métodos que uma classe deve ter.

A notação usada no Grafo de Interação de Objetos é a mostrada na Figura 4.6 (a) e um Grafo de Interação de Objetos hipotético é mostrado na Figura 4.6 (b).

Os retângulos com linhas sólidas representam os objetos e os com linhas tracejadas representam coleções de objetos. As setas indicam a mensagem que está sendo passada entre os objetos/classes. A seta rotulada por **create** indica que houve a criação dinâmica do obj2:Classe2. A seta rotulada por **nome mens a (lista param)** indica o envio de uma mensagem de obj2:Classe2 para obj5:Classe5. Quando além do nome aparece um número entre parênteses próximo da mensagem, esse número indica a ordem seqüencial de envio de mensagens. Assim, primeiro é passada a mensagem de obj2:Classe2 para obj5:Classe5 e depois de obj2:Classe2 para obj4:Classe4. A mensagem **nome mens c()** é passada de um objeto para uma coleção de objetos. O rótulo **n=nome mens ():Type** indica que um valor retornou da mensagem.obj1:Classe 1 é chamado de objeto controlador pois controla uma operação do sistema. Graficamente isso é representado por uma seta sem origem, com destino no objeto controlador e com o nome da operação, nesse caso **op1**. Os demais objetos são colaboradores.

4.6 (a)

4.6 (b)

Figura 4.6 - Notação do Grafo de Interação de Objetos

Segundo Passo: Construir um Grafo de Visibilidade para cada uma das classes existentes. Ele permite que se estude a necessidade de referências entre classes, a partir da troca de mensagens entre os objetos explicitada nos Grafos de Interação. As referências devem ser classificadas segundo quatro aspectos: duração, visibilidade do servidor, dependência do servidor e mutabilidade. A duração indica se a referência deve ser dinâmica ou permanente. Quando a referência deve continuar existindo entre várias chamadas de um método, especifica-se uma referência permanente, caso contrario ela é dinâmica. A visibilidade do servidor indica se o acesso a um objeto servidor pode ser feito apenas por um certo objeto cliente — referência exclusiva — ou se pode ser feita por qualquer objeto — referência compartilhada. A dependência do servidor indica se a vida do objeto servidor depende de um objeto cliente, isto é, se o servidor desaparece com o desaparecimento do cliente. A mutabilidade indica se

a referência pode ser alterada após ter sido criada. A Figura 4.7 mostra a notação utilizada para o Grafo de Visibilidade.

Figura 4.7 - Notação para o Grafo de Visibilidade

Terceiro Passo: Elaborar a Descrição de Classes. Essa descrição, usando uma notação textual abstrata, especifica os atributos de cada objeto e a sua interface externa, isto é seus métodos. Os atributos de dados, atributos de referência a objetos e métodos são definidos para todas as classes do sistema. As informações do Modelo de Objetos do Sistema, dos Grafos de Interação de

Objetos e dos Grafos de Visibilidade são utilizadas para a derivação das descrições de classe.

Quarto Passo: Construir os Grafos de Herança: eles usam a mesma notação do Modelo de Objetos (Figura 4.3 (b) e (c)) para herança e a representação simples de classe. São usados para complementar as descrições de classes. Novas abstrações podem ser acrescentadas, diferentes daquelas usadas no Modelo de Objetos do sistema, refletindo decisões de projeto.

Quinto Passo: Atualizar a Descrição de Classe com as novas informações de herança. Checar se as relações de subtipos são preservadas no Modelo de Objetos do Sistema; se todas as classes estão representadas no Grafo de Interação; se todas as classes estão representadas no Grafo de Herança e se a Descrição de Classes atualizada implementa todas as funcionalidades das anteriores.

A Descrição de Classes utiliza a notação exibida na Figura 4.8. Ela inicia com a palavra chave **class** seguida pelo nome da classe. Cada um dos atributos dessa classe é descrito com a palavra chave **attribute** opcionalmente seguido de nome e tipo. Existem algumas palavras chave para descrever cada uma das características dos atributos que são as mostradas na Figura 4.9.

```
class<Nome da Classe> [is a <Nome da super Classe>]

  // para cada atributo

    [attribute] [Mutabilidade]<a_nome>:[Compartilhamento][Dependência]<Type>
    ...
  // para cada método

    [method] <m_nome> <lista de argumentos>[:<tipo>]
    ...
endclass
```

Figura 4.8 - Notação de Descrição de Classe

Mutabilidade:	variable ou constant , sendo variable, "default", para descrever a mutabilidade do atributo;
Compartilhamento:	shared ou exclusive , sendo shared, "default", para descrever a forma da informação compartilhada;
Dependência:	bound ou unbound , sendo unbound, "default", para descrever o tempo de vida de um objeto do Grafo de Visibilidade.

Figura 4.9 - Palavras Chaves para Descrição das Características dos Atributos na Descrição de Classe

Pode-se observar pela Figura 4.1 que as informações constantes do Modelo de Objetos são fonte de alimentação para os Grafos de Herança e as Descrições de Classes, enquanto que as informações do Modelo de Interface alimentam o Grafo de Interação de Objetos.

A última fase consiste na **implementação** e é composta pelos três passos descritos a seguir, sem uma ordem fixa de execução:

Primeiro Passo - Codificação: traduz-se cada classe descrita na fase de projeto para a sintaxe e semântica de uma linguagem de programação em particular. O método oferece diretrizes para essa tradução, visando principalmente as linguagens C++, Eiffel e Smalltalk. Entre os aspectos importantes nessa etapa pode-se destacar a tradução das classes, o desenvolvimento do corpo dos métodos, o tratamento de erros e a satisfação de requisitos de desempenho.

As seqüências de operações permitidas descritas pelo ciclo de vida definem a interface do sistema e são implementadas por um objeto controlador da interface. O controlador é desenvolvido através da transformação da expressão regular do ciclo de vida em uma máquina de estados finitos (ou um statechart, se houver concorrência ou intercalação de eventos) e desta para o código.

Segundo Passo - Desempenho: deve-se considerar o desempenho durante as três fases: análise, projeto e implementação.

Terceiro Passo - Revisão: o código produzido deve ser revisado. Deve-se garantir que o código produzido possa ser entendido por outras pessoas além do próprio autor. Os testes devem ser realizados para verificar o comportamento atual do sistema ou parte do sistema, em relação aos requisitos inicialmente solicitados. Os autores do método sugerem que as técnicas padrão para teste de software devem ser re-examinadas à luz dos conceitos de orientação a objetos: herança, níveis de abstração e estado do objeto, para auxiliar nessa fase.

O método Fusion recomenda o uso de um Dicionário de Dados durante as três fases, registrando os diversos componentes de cada modelo desenvolvido. Há também uma grande preocupação com os aspectos de qualidade e são recomendadas revisões ao final de cada fase, que visam a, principalmente, checar a completitude da especificação e a consistência interna entre os modelos.

4.4 Método de engenharia reversa orientada a Objetos

O método de engenharia reversa orientada a objetos proposto é resumido na Tabela 4.2. Cada passo é descrito com mais detalhes no restante desta Seção. A seqüência ideal para execução dos passos é a apresentada na tabela, sendo que os sub-passos dos passos 1 e 2, com exceção do 2.1, são conduzidos com bastante interseção e interação.

4.4.1. Revitalizar a Arquitetura do Sistema com Base na Documentação Existente

O objetivo é recuperar as informações relacionadas à arquitetura do sistema para que ele possa ser entendido/estendido pelo engenheiro de software, facilitando o desenvolvimento dos passos posteriores. Caso o sistema possua documentação adequada, basta apenas identificá-la e organizá-la de forma a permitir que a funcionalidade do sistema seja completamente entendida.

Tabela 4.2 - Método de engenharia reversa orientada a Objetos

Passo	Objetivo	Produto
1. Revitalizar a Arquitetura do Sistema com base na documentação existente	Obter informações relacionadas à arquitetura do sistema para o seu entendimento	Lista de todos os procedimentos, sua descrição e a relação chama/chamado por
2. Recuperar o Modelo de Análise da Solução Atual	Obter um modelo de análise considerando somente os aspectos físicos	Documentos descritos nos passos 2.1., 2.2., 2.3., 2.4.
2.1. Definir Temas	Modelar em temas as informações armazenadas relativas às entradas, saídas, armazenamento permanente e temporário	Lista de Temas
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	Modelo de Objetos, lista de atributos, procedimentos associados às classes, lista das anomalias existentes
2.3. Definir o Ciclo de Vida	Mostrar o comportamento global do sistema	Modelo de Ciclo de Vida
2.4. Abstrair e Desenvolver as operações	Obter as operações realizadas pelo sistema	Modelo de Operações
3. Abstrair o Modelo de Análise do Sistema	Obter um modelo de análise do sistema considerando os aspectos do domínio da aplicação	Documentos descritos nos passos 3.1., 3.2., 3.3.
3.1. Desenvolver o Modelo de Objetos	Elaborar um Modelo de Objetos considerando as classes e seus relacionamentos que devem ser tratados pelo sistema	Modelo de Objetos. Para cada classe: lista dos atributos e métodos
3.2. Elaborar o Modelo de Ciclo de Vida	Fornecer uma visão global do comportamento do sistema a partir da abstração realizada	Modelo de Ciclo de Vida
3.3. Especificar o Modelo de Operações	Descrever como as operações devem ser realizadas	Modelo de Operações
4. Mapear o Modelo de Análise do Sistema para o Modelo de Análise do Sistema Atual	Descrever a Relação entre os Modelos de Análise do Sistema atual e novo.	Mapeamento das Classes e dos Métodos do MAS para o MASA

A documentação pode ser recuperada semi-automaticamente, com apoio de uma ferramenta, como aquela descrita em [98], ou manualmente. De ambas as formas, deve-se descrever a função de cada um dos procedimentos utilizados para implementação do sistema, fazendo-se referência cruzada entre eles: os procedimentos utilizados (CHAMA) e os procedimentos utilizadores (CHAMADO POR). Essas informações são obtidas a partir do código fonte existente. Os parâmetros das chamadas dos procedimentos não são especificados aqui, embora eles possam ser obtidos a partir do código existente. A Figura 4.10 mostra um trecho da documentação produzida nesta fase do método.

Módulo =====	Descrição =====
Módulo1.h	Este módulo refere-se às operações realizadas sobre módulo 1.h
<u>Procedimentos</u>	
proc1	Procedimento referente às operações realizadas por proc1.
	CHAMA: =====
	proc2 proc3 (Módulo 2.h) proc4
	CHAMADO POR: =====
	proc3 (Módulo 3.h) proc5

Figura 4.10 - Procedimentos de um Módulo

De qualquer modo que se recupere as informações para revitalizar a arquitetura do sistema é importante que o engenheiro de software tenha conhecimento dos métodos e técnicas usadas para projeto e implementação do sistema atual.

Caso não existam informações sobre a forma de implementação realizada para o sistema, o engenheiro de software pode procurar obtê-las com analistas que utilizem ou tenham utilizado o sistema exhaustivamente e que tenham condições de explicar o seu funcionamento. Caso isso não seja possível deve-se consultar o código fonte para recuperar as informações relacionadas com a base

de dados, para posteriormente passar ao levantamento dos procedimentos utilizados. A consulta ao código fonte pode ser apoiada por ferramentas extradoras [4, 98]. É comum também que os profissionais envolvidos num projeto de engenharia reversa não desconheçam completamente o sistema, isto é, já tenham tido alguma experiência anterior como mantenedores do sistema.

O produto deste passo é uma lista com as informações de todos procedimentos do sistema, ordenados alfabeticamente para facilitar a recuperação manual, funcionando como um índice para o código fonte e as relações CHAMA/CHAMADO POR como mostra a Figura 4.10. Adotou-se a convenção de colocar entre parênteses o nome do módulo que o procedimento pertence quando esse for distinto do módulo corrente que está sendo analisado. Os procedimentos que somente apresentam o seu nome são os que pertencem ao módulo corrente.

4.4.2 Recuperar o Modelo de Análise da Solução Atual

Deve-se desenvolver neste passo um modelo de análise do sistema atual considerando-se somente os seus aspectos físicos, ou seja, deve-se considerar somente a implementação atual. Os resultados deste passo são o Modelo de Objeto, de Operações e de Ciclo de Vida, seguindo os modelos propostos no método Fusion, descritos na seção anterior. Esses modelos utilizam a notação exibida nas Figuras 4.3, 4.4 e 4.5, respectivamente, que compõe o Modelo de Análise do Sistema Atual (MASA).

4.4.2.1. Definir Temas

O objetivo é modelar os grandes temas relativos às informações que o sistema manipula. Geralmente eles estão relacionados às entradas (interface), saídas e às informações armazenadas em caráter temporário e permanente. As informações de caráter temporário são aquelas em que o acesso é permitido somente durante o processamento, ou seja, relativas às operações do sistema executadas na memória RAM. Já o termo permanente é utilizado para caracterizar aquelas informações que ficam armazenadas permanentemente em

algum dispositivo de armazenamento não volátil como por exemplo, discos. O resultado deste sub-passo é uma lista de temas abrangendo todas as informações relacionadas ao sistema.

4.4.2.2. Desenvolver o Modelo de Objetos do Sistema Existente

Para cada um dos assuntos considerados no item anterior deve-se listar inicialmente todas as classes existentes, identificadas a partir das estruturas de dados implementadas. Os Modelos de Objetos criados neste sub-passo são abstrações realizadas visando a facilitar o entendimento do sistema existente, uma vez que a implementação atual não segue a abordagem de orientação a objetos. O termo Modelo de Objetos é utilizado pois sua construção é baseada nos conceitos preconizados por Coleman [17], mas deve ser entendido como um modelo intermediário que representa uma abstração de um código não implementado usando os conceitos e técnicas orientados a objetos. É construído e usado para a criação do Modelo de Análise do Sistema (passo3).

A seguir são sugeridos os passos para a recuperação do Modelo de Objetos para três cenários possíveis de implementação do sistema atual resumidos na Figura 4.11.

Se a implementação foi realizada utilizando base de dados relacional o engenheiro de software deve consultá-la a fim de obter todas as entidades, relacionamentos e chaves das relações para projetar o modelo de entidade-relacionamento (MER). Como no modelo relacional os dados são armazenados através de tabelas e a duplicação de dado em tabelas indica que ele é chave de um relacionamento, obtêm-se as informações necessárias para recuperar as entidades e seus relacionamentos. A partir do MER pode-se facilmente adaptá-lo para a notação Fusion e outras características adicionais podem ser acrescentadas ao modelo, tais como cardinalidade e papéis.

Se a técnica utilizada na implementação for Tipos Abstratos de Dados (TAD) deve-se consultar o código para descobrir quais são as estruturas de dados relevantes, de uso global ou local. Nessa técnica as estruturas de dados

geralmente contém ponteiros que são os responsáveis pelas ligações existentes entre elas, mas pode haver duplicação de chaves. Como passo intermediário pode-se construir um diagrama simples com as estruturas de dados lógicas recuperadas e os relacionamentos entre elas. Esse diagrama pode então ser convertido para a notação do Fusion e as demais características semânticas (exemplo: cardinalidade, papéis, etc.) podem ser acrescentadas. A partir daí tem-se condições de se recuperar quais são os procedimentos que atuam sobre essas estruturas de dados lógicas e construir uma documentação de apoio a essa fase do método.

Figura 4.11 - Cenários Possíveis da Implementação Atual

Caso o sistema tenha somente código Cobol e nada se saiba sobre a técnica de projeto utilizada, deve-se recorrer às estruturas de dados definidas na Data Division e a partir daí começar a reconstruir as estruturas de dados. Os dados definidos na Working Storage Section não devem ser priorizados uma vez que eles são somente dados auxiliares. Na Procedure Division encontram-se os prováveis procedimentos que devem ser considerados, isso se o programa foi desenvolvido com programação modular.

A partir dessas informações preliminares são definidos os registros lógicos. Em seguida verificam-se os relacionamentos entre eles, que podem ter sido implementados por índices, chaves duplicadas, contiguidade física, etc. Esse diagrama intermediário pode então ser convertido para a notação do Fusion e, assim como nos casos anteriores, outras características semânticas são acrescentadas.

É importante notar que para a construção do MASA não é necessário dar ênfase às estruturas de agregação e especialização/generalização, uma vez que provavelmente elas não fazem parte da implementação atual. Estruturas simples de agregação, como as que relacionam um arquivo e seus registros podem ser usadas sempre que relacionamentos nos dois níveis existirem.

O método Fusion propõe que a classe seja descrita através de seus atributos e métodos. Como este método de engenharia reversa aplica-se a sistemas desenvolvidos sem utilizar linguagens orientadas a objetos, optou-se pela terminologia “elementos de dados” ao invés de “atributos” e “procedimentos” ao invés de “métodos”. Os termos “atributos” e “métodos” serão utilizados somente no modelo generalizado (MAS), que é baseado na análise orientada a objetos..

O relacionamento entre essas classes é obtido através da própria relação expressa entre os tipos de dados, isso é, os ponteiros existentes entre as diversas estruturas ou o esquema da base de dados. A partir das classes de objetos deve-se listar todos os atributos da classe. Os atributos são os mesmos que os existentes em cada uma das estruturas de dados selecionadas.

Classe: xxxxx
Atributos: _____ _____
Procedimentos: _____ _____

Figura 4.12 - Descrição da Classe, Atributos e Procedimentos

Os procedimentos são obtidos com base no conhecimento do engenheiro de software do sistema e dos documentos obtidos no passo 1. Deve-se analisar cada um dos procedimentos e verificar qual estrutura de dados está associada a ele. Além disso deve-se analisar a forma de associação entre a classe e o procedimento. O procedimento será associado à classe que ele consulta ou constrói. Adotou-se a convenção (c) para construtor, quando o procedimento altera a estrutura de dados, e (o) para observador, quando o procedimento somente consulta a estrutura. A Figura 4.12 mostra um diagrama esquemático para representação de classe.

Como já foi dito anteriormente, quando o sistema não tem uma implementação orientada a objetos, a análise dos procedimentos é complexa. Elaborou-se uma classificação de anomalias, sendo que as mais comuns se referem a procedimentos que são:

- observador de uma classe e construtor de outra (oc)
- observador de uma classe e construtor de mais de uma classe (oc^+)
- observador de mais de uma classe e construtor de outra (o+c)
- observador de mais de uma classe e construtor de mais de uma classe (o+c+)
- construtor de duas classes (ou mais) (c+)
- observador de duas classes (ou mais) (o+)

Existem procedimentos que não se referem a classe alguma, sendo dependentes da implementação, relacionados à interface ou ao módulo escalonador, etc., Esses procedimentos são classificados como (i).

Quando um procedimento é construtor de uma classe e também observador dessa mesma classe, este deve ser classificado somente como construtor. Por exemplo, no caso da estrutura ser uma lista, existe um procedimento que verifica em qual posição da lista esse elemento deve ficar e coloca-o. Assim quando o procedimento consulta a lista para saber em qual posição o elemento deve ficar, ele é observador da lista; já quando ele coloca efetivamente o elemento nessa posição, ele é construtor. Para esses casos o

procedimento foi classificado somente como construtor (c), como ocorre normalmente com especificações algébricas..

Se mais de uma classe está associada a um procedimento então este será associado a uma única classe, seguindo a seguinte ordem de prioridade:

- À classe construída se for do tipo **oc**;
- À primeira classe construída analisada se for do tipo **oc+**;
- À primeira classe consultada analisada se for do tipo **o+c**.

A classificação é realizada por classe de objetos, como mostra a Figura 4.13. A notação utilizada para a classificação dos procedimentos é a seguinte:

<nome do procedimento> <[tipo/módulo]>

onde:

nome do procedimento: nome do procedimento no código fonte

tipo: é o (observador) ou c (construtor) de acordo com a atuação do procedimento sobre essa classe

módulo: refere-se ao módulo ao qual o procedimento está alocado.

Se um procedimento está associado a mais de uma classe utiliza-se a seguinte notação:

**<nome_procedimento> <[tipo/ módulo]> <[tipo-1/ classe-1/ tema-1]>
<[tipo-2/ classe-2/ tema-2]> . . .**

onde:

nome do procedimento: nome do procedimento no código fonte

[tipo, módulo] segue a definição acima;

[tipo-i/ classe-i/ tema-i] para i = 1,2, ...,n

tipo-i conforme descrito anteriormente

classe-i refere-se à classe adicional relacionada ao procedimento

tema-i é o nome do tema ao qual a classe-i pertence.

Por exemplo, considere a classe X1 como a que está sendo descrita. O procedimento a1 é observador para a classe X1 e está no módulo m1.

O procedimento a2 é construtor para a classe X1, inserido no módulo m2 e observador para a classe c1 do tema t1. Portanto esse procedimento é classificado com anomalia **oc**.

Quando não há anomalia o procedimento é classificado simplesmente como **(o)** ou **(c)**.

Como produto deste passo tem-se: a) Modelo de Objetos descrevendo as classes e relacionamentos utilizando a notação do método Fusion como foi mostrado nas Figuras 4.2 e 4.3; b) as classes, atributos e procedimentos identificados como mostra a Figura 4.12 e c) a classificação dos procedimentos com ou sem anomalias, como mostra a Figura 4.13.

Classe: X1			
Procedimentos			
a1	[o/m1]		o
a2	[c/m2]	[o/c1/t1]	oc
a3	[c/m1]		c
...			
a4	[c/m3]	[c/c2/t2]	c+

Figura 4.13 - Classificação dos Procedimentos de Determinada Classe

4.4.2.3. Definir o Ciclo de Vida

O objetivo deste sub-passo é definir o ciclo de vida do sistema. A observação em uso da interface do sistema atual é geralmente suficiente para definir todos os eventos de entrada e de saída que o sistema aceita. As seqüências de operações permitidas podem ser registradas em cenários que, reunidos e generalizados, são a base para a definição do ciclo de vida. Manuais de uso do sistema, quando existentes, também são úteis para auxiliar a definir as operações do sistema e a assegurar a completude da análise.

OPÇÕES DE INTERFACE	OPERAÇÕES	HIERARQUIA DE CHAMADAS
_____	_____	_____
_____	_____	_____

Figura 4.14 - Opções existentes na interface do ambiente

Para auxiliar a construção do Modelo de Ciclo de Vida pode-se construir uma tabela, do tipo da Figura 4.14, contendo todas as informações existentes relacionadas com as chamadas de procedimentos quando o sistema atual é ativado. Devem ser listadas as opções da interface, as operações que estão associadas a essas opções e a hierarquia de chamadas, isto é, os nomes dos procedimentos que implementam as operações. Deve-se listar os procedimentos até o segundo nível de chamada sempre que o primeiro for um procedimento tipo (i), já que esses são inerentes à forma utilizada e portanto não realizam operação específica. As informações contidas nessa tabela dão apoio para a construção de expressões regulares que compõem o ciclo de vida com gabarito mostrado na Figura 4.4.

4.4.2.4. Abstrair e Desenvolver as Operações

Para cada operação identificada constrói-se o Modelo de Operações seguindo o gabarito descrito na Figura 4.5. A operação é normalmente implementada pelo procedimento chamado pelo módulo ou programa controlador da interface a partir da ocorrência de um evento de entrada. Muitas vezes há uma cadeia de chamadas até se chegar ao procedimento que realmente executa a operação, geralmente auxiliado por outros procedimentos subordinados. Esse deve ser o procedimento base para a especificação da operação, pois os demais cuidam de aspectos da interface e são dependentes da implementação. Consistências efetuadas podem dar origem a pré-condições.

As abstrações relevantes para a definição das operações são encontradas geralmente até no penúltimo nível da hierarquia de chamada de procedimentos a partir da interface. Essas operações são parte daquelas

classificadas como (i), na seção 4.4.2.2. O nível mais baixo contém os procedimentos que constróem ou observam objetos, dando origem aos demais tipos da classificação apresentada na seção 4.4.2.2, mostrado na Figura 4.13.

A Figura 4.15 fornece um esquema de onde podem ser obtidas as informações para a elaboração do Modelo de Operações. Os componentes desse modelo, como mostrado na Figura 4.5, são escritos em **negrito** e as fontes onde eles podem ser obtidos são escritas com o tipo *itálico*.

Operação	<i>Modelo de Ciclo de Vida</i>
Descrição	<i>Documento de Revitalização da Arquitetura do Sistema (descrição do procedimento com o mesmo nome que o da operação que está sendo descrita)</i>
Lê	<i>Código fonte</i>
Modifica	<i>Código fonte</i>
Envia	<i>Código fonte</i>
Assume	<i>Código fonte juntamente com os conhecimentos do engenheiro de Software e do Documento de Revitalização da Arquitetura do Sistema</i>
Resultado	<i>Código fonte</i>

Figura 4.15 - Fontes de informações para o Modelo de Operação

4.4.3. Abstrair para o Modelo de Análise do Sistema

Até o momento trabalhou-se com informações disponíveis na implementação existente, construindo-se o Modelo de Análise do Sistema Atual (MASA). A partir do passo 3 realiza-se a abstração da visão física criando-se a visão lógica do sistema, denominada de Modelo de Análise do Sistema (MAS). A visão lógica abstrai da visão física, sempre que possível, aspectos que deveriam ter sido especificados anteriormente e que não foram. O produto gerado é semelhante aos documentos gerados no passo 2, com exceção da lista de temas, dando-se enfoque no domínio da aplicação e não na implementação (domínio da solução). Um Dicionário de Dados, manual ou automatizado, deve ser usado para armazenar informações sobre o MAS, como recomendado em Fusion. Neste

passo são utilizados os termos “atributos” e “métodos” para identificar os dados e os algoritmos associados a uma certa classe.

4.4.3.1. Desenvolver o Modelo de Objetos

As classes do MAS são abstraídas das classes do MASA. É comum que um conjunto de classes e relações do MASA dê origem a uma classe do MAS, pois os objetos de implementação escolhidos (listas, vetores de ponteiros, etc.) devem ser abstraídos para um conceito do sistema. Pode ocorrer também que uma classe (tipo) do MASA com mais de uma instanciação concreta (instâncias) dê origem a mais de uma classe no MAS, correspondentes aos conceitos do sistema. Por exemplo, uma pilha pode armazenar informações relacionadas a mais de um tipo de objeto. Quando ela for especificada no MAS dará origem a tantas classes de objetos quantas as diferentes informações que ela instancia.

Os atributos são analisados e normalmente têm seu nome modificado para um mnemônico com significado mais próximo do conceito que ele representa no mundo real e com maior legibilidade (sem abreviações, por exemplo). Alguns atributos são desnecessários porque são dependentes da implementação e os que apontam para outras classes (referências) são substituídos por relacionamentos. Os atributos podem também se deslocar para a superclasse ou subclasse, conforme se identifiquem relações de generalização/especialização, respectivamente. Alguns métodos canônicos, como aqueles que criam a classe e permitem acesso a atributos da classe (para atualizar ou consultar) podem ser incorporados ao MAS neste passo, mas o método Fusion não recomenda que isso seja feito, podendo ser deixado para as fases de projeto ou de implementação.

É importante notar que os métodos não canônicos, em Fusion, são incorporados ao Modelo de Objetos na fase de projeto, a partir dos Grafos de Interação de Objetos. Como na engenharia reversa os procedimentos são recuperados a partir da implementação atual, eles também podem ser acrescentados ao Modelo de Objetos neste ponto, eliminando-se as anomalias identificadas. Por exemplo, procedimentos sem anomalias (classificados no sub-

passo 2.2. como (o) ou (c)) geralmente serão métodos do MAS, a menos de possível adaptação do nome; procedimentos com anomalia (oc) devem dar origem a dois métodos: um relacionado à classe da qual o procedimento é observador e outro relacionado à classe da qual o procedimento é construtor e assim por diante para as demais anomalias. Os procedimentos classificados como (i) não dão origem a métodos. Nesse passo também não são apresentados os parâmetros que acompanham os procedimentos, embora esses sejam facilmente encontrados no código fonte existente. A tabela 4.3 mostra a correspondência entre os procedimentos do MASA e os métodos do MAS.

Tabela 4.3 - Correspondência entre os Procedimentos e Métodos

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

O Modelo de Objetos do sistema produzido neste sub-passo contém classes e seus respectivos relacionamentos, como o proposto no método Fusion

utilizando a notação exibida na Figura 4.3. Tem-se também para cada uma das classes pertencentes ao modelo, a listagem dos atributos e métodos associados, como nas Figuras 4.10 e 4.12.

De acordo com o método Fusion depois da construção do Modelo de Objetos deve-se verificar a interface do sistema, ou seja, adicionar limites ao modelo. Não há preocupação com esse passo no método de engenharia reversa orientada a objetos uma vez que o próprio MAS define o limite da implementação em função do estudo realizado pelo MASA anteriormente, devendo-se cuidar do Modelo de Ciclo de Vida do Sistema. Entretanto os agentes podem ser acrescentados ao modelo como classes, da maneira utilizada no Fusion.

4.4.3.2. Elaborar o Modelo de Ciclo de Vida

O Modelo de Ciclo de Vida para o MAS é praticamente o mesmo que o produzido no passo 2.3, mas tomando-se o cuidado de alterar os nomes dos eventos de entrada e de saída para nomes mais mnemônicos. Cada alteração realizada implica também em mudança do nome da operação correspondente.

Interfaces de comandos exigem, além da experimentação com o sistema, alguma documentação escrita com a especificação dos comandos. Interfaces guiadas por menus ou por manipulação direta de ícones na tela são mais simples de serem analisadas. Com base na tabela criada no sub-passo 4.4.2.3, como mostra a Figura 4.14, na documentação produzida quando da revitalização do sistema e do conhecimento do engenheiro de software pode-se elaborar o Modelo de Ciclo de Vida do MAS, com o mesmo “layout” apresentado quando da elaboração do MASA.

Se o objetivo principal do projeto de reengenharia dentro do qual está inserida a engenharia reversa for não alterar a funcionalidade do sistema o Modelo de Ciclo de Vida para o MAS é praticamente o mesmo que o produzido no passo 2. 3, a menos de alterações dos nomes dos eventos de entrada e de saída. A ampliação da funcionalidade implicaria em outras mudanças que fogem ao escopo deste trabalho mas poderiam ser integradas a esta abordagem. Por

exemplo: seria necessário a verificação de outros relacionamentos entre as classes, a adição de novas operações e a elaboração do Modelo de Ciclo de Vida seguindo essas modificações realizadas.

4.4.3.3. Especificar as Operações

Para cada uma das operações do Modelo de Operações desenvolvido no sub-passo 2.4, possivelmente com seu nome alterado no sub-passo 3.2, Tabela 4.2., deve-se reespecificar a operação para o nível de abstração do MAS. Deve-se respeitar a funcionalidade especificada mas atualizar a notação para o estilo do Fusion e considerar agora as classes, relacionamentos e atributos do Modelo de Objetos do MAS. O Modelo de Operações segue o exemplo mostrado na Figura 4.5. Essas operações descritas passam a ser consideradas como métodos dos objetos que são seus controladores.

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

Este mapeamento é fundamental para o desenvolvimento futuro de todo ou de substituição de partes do sistema atual, facilitando o reuso. O próprio desenvolvimento do sub-passo 3.1. conduz para o mapeamento entre os documentos do MAS e correspondentes do MASA, que deve ser cuidadosamente documentado e checado em termos de completitude neste passo. Esse mapeamento facilita tanto o entendimento como a implementação futura do sistema, pois mostra facilmente como os elementos do MAS estão atualmente implementados facilitando a reutilização. Pode-se avaliar neste ponto quanto da implementação pode ser reutilizada caso se decida continuar com a mesma abordagem atualmente utilizada e não passar a adotar uma nova abordagem como a de orientação a objetos. Em termos de notação a correspondência pode ser feita por tabelas, mas a informação pode também estar armazenada no Dicionário de Dados. As tabelas construídas são obtidas com as informações

produzidas nas seções 4.4.3.1, 4.4.3.2 e 4.4.3.3 e são como a mostrada na Figura 4.16.

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

Figura 4.16 - Tabela com o Mapeamento entre MAS e o MASA

Para cada uma das classes descritas no MAS elabora-se uma tabela conforme mostrada na Figura 4.16, relacionando-as com o seu correspondente no MASA. Essa correspondência deixa de existir para os objetos necessários quando o sistema sofre mudança de funcionalidade. Para cada classe são listados os atributos e seus elementos de dados correspondentes bem como os métodos associados e os correspondentes procedimentos. Pode ocorrer que um procedimento dê origem a mais de método e, nesse caso, o procedimento aparecerá repetidamente na coluna correspondente ao MASA.

4.5. Interligação dos Documentos Produzidos

A Figura 4.17 exhibe o relacionamento existente entre os documentos atuais existentes do sistema e os documentos produzidos pelo método de engenharia reversa orientada a objetos. Pode-se notar que os procedimentos, estruturas de dados e o conhecimento dos usuários e profissionais são o subsídio para a recuperação da arquitetura do sistema. Se o sistema possuir alguma documentação, ela, juntamente com os documentos anteriormente citados é que dá apoio à geração dessa nova documentação. A partir dessa

Figura 4.17 - Interligação dos Documentos Existentes e Produzidos

documentação, recuperando-se a arquitetura do sistema, elaboram-se os documentos do MASA e posteriormente os do MAS.

Pode-se observar pela Figura 4.17 que os documentos produzidos pela fase de revitalização da arquitetura são fonte de alimentação tanto para os documentos do MASA como do MAS. Os **Temas** resultam das consultas às Estruturas de Dados e a partir deles e da consulta aos **Procedimentos** o **Modelo de Objetos** é criado. Com as informações armazenadas no **Modelo de Objetos** gera-se o **Modelo de Ciclo de Vida**. Por sua vez, o **Modelo de Operação** tem origem a partir das informações encontradas no **Modelo Ciclo de Vida**, juntamente com o **Modelo de Objetos**. A partir dos documentos criados no MASA passa-se à elaboração do MAS. Para a criação dos documentos do MAS utilizam-se todos os modelos criados anteriormente e os documentos correspondentes à **Revitalização da Arquitetura do Sistema**. O último passo do método consta do **Mapeamento** entre os documentos produzidos no MASA e no MAS.

4.6. Métricas Para Avaliação do Esforço de Conversão

Algumas métricas podem ser utilizadas para a avaliação do esforço de conversão. Consideram-se para essa avaliação as informações existentes quanto aos procedimentos levantados, métodos correspondentes a esses procedimentos e linhas de código fonte existentes.

A primeira métrica proposta é denominada de “**Índice do Esforço de Conversão**” ou “**Índice de Adaptação do Modelo Existente para Orientado a Objetos**”. Ela é definida pela expressão que compõe a Figura 4.18.

Essa métrica permite que se avalie a facilidade em converter um sistema não orientado a objetos em um sistema orientado a objetos. A tabela 4.4 resume os principais valores encontrados para essa métrica.

$$IEC = \frac{NP_o + NP_c + 0,9NP_{oc} + 0,8NP_{o^+} + 0,7NP_{o^+c} + 0,5NP_{c^+} + 0,4NP_{oc^+} + 0,2NP_{o^+c^+}}{(NProc - NPi)}$$

onde :

- NP_o representa o número de procedimentos classificados como o;
- NP_c representa o número de procedimentos classificados como c;
- NP_{oc} representa o número de procedimentos classificados como oc;
- NP_{o^+} representa o número de procedimentos classificados como o^+ ;
- NP_{c^+} representa o número de procedimentos classificados como c^+ ;
- NP_{o^+c} representa o número de procedimentos classificados como o^+c ;
- NP_{oc^+} representa o número de procedimentos classificados como oc^+ ;
- $NP_{o^+c^+}$ representa o número de procedimentos classificados como o^+c^+ ;
- $NProc$ representa o número de procedimentos existentes
- NP_i . representa o número de procedimentos relacionados à implementação

Figura 4.18 - Métrica para Calcular o Índice do Esforço de Conversão

Tabela 4.4 - Métrica para Avaliar a Conversão de um Sistema não Orientado a Objetos para um Sistema Orientado a Objetos

Valor de IEC	Significado
$1 \leq IEC \leq 0,8$	o sistema pode ser facilmente convertido para um sistema seguindo a abordagem orientada a objetos
$0,8 < IEC \leq 0,5$	o sistema pode ser convertido para um sistema utilizando a abordagem orientada a objetos dispendendo médio esforço
$0,5 < IEC \leq 0,2$	o sistema pode ser convertido utilizando a abordagem orientada a objetos dispendendo-se grande esforço

Uma segunda métrica proposta denominada “Índice da Influência do Tipo de Implementação Utilizada”, IITIU, exibida na Figura 4.19 se refere à percentagem dos procedimentos de implementação em relação a todos os

procedimentos existentes no sistema. A Tabela 4.5 apresenta alguns resultados e seus significados para IITIU, levando em consideração pesquisas empíricas que indicam como se comporta a quantidade de código relativa à aplicação e relativa à interface.

$$IITIU = \frac{NP_i}{NProc}$$

onde:

- NP_i representa o número de procedimentos classificados como tipo i ;
- $NProc$ representa o número total de processos existentes.

Figura 4.19 - Métrica para Calcular o Índice de Influência do Tipo de Implementação Utilizada

Tabela 4.5 - Métrica para a Avaliar a Percentagem de Procedimentos de Implementação em um Sistema

Valor de IITIU	Significado
$IITIU \geq 30\%$	grande a influência dos procedimentos de implementação no sistema.
$30\% < IITIU \leq 15\%$	normal a influência dos procedimentos de implementação no sistema
$15\% < IITIU$	baixa influência dos procedimentos de implementação no sistema

Uma métrica alternativa à anteriormente apresentada é denominada “Influência do Código nos Procedimentos de Implementação”, ICPI (Figura 4.20). Ela permite que se avalie qual a influência do número de linhas de código dos procedimentos i em relação aos demais procedimentos implementados atualmente.

$$ICPI = \frac{\sum_j KLOC_{PROC_i}}{\sum_j KLOC_{PROC} - \sum_j KLOC_{PROC_i}}$$

onde:

- $KLOC_{PROC_i}$ é o total de linhas de código existentes para is procedimentos i
- $KLOC_{PROC}$ é o total de linhas de código existentes no sistema

Figura 4.20 - Métrica para Calcular a Influência do Código nos Procedimentos de Implementação

A métrica ICPI serve para complementar e checar a métrica IITIU. O valor esperado de ambas deve ficar próximo, indicando que há um certo equilíbrio entre os tamanhos dos procedimentos relativos à interface/implementação e os relativos à aplicação. Pode haver, entretanto, situações patológicas, como poucos procedimentos (i) de tamanho muito grande ou muitos procedimentos (i) de tamanho muito pequeno. É necessário estudar, caso a caso como essas patologias podem influenciar o esforço de conversão.

Não se tem uma idéia precisa do esforço de conversão necessário para os métodos i. Entretanto, para sistemas implementados usando software de apoio como o Xview (Motiff, ou outros), conjectura-se que esse esforço seria relativamente pequeno, por duas razões principais. Em primeiro lugar porque esses softwares já utilizam conceitos de orientação a objetos e segundo porque grande parte das janelas, formulários, "prompts", etc. que compõe a interface do sistema atual seria reutilizada.

Se entretanto, outro software de gerência de interface com o usuário for utilizado, se novas funcionalidades forem acrescentadas ou se a interface do sistema atual for completamente alterada, então o esforço de conversão pode ser bem maior.

No caso de interfaces convencionais, isto é, implementadas de forma entrelaçada com o código de aplicação, sem utilização de Sistemas de

Gerenciamento de Interface do Usuário (S.G.I.U.), o esforço de conversão pode ser grande. Um valor percentual muito baixo para esta métrica pode indicar que o código relativo à interface está misturado com os procedimentos da aplicação, o que dificulta os esforços de conversão.

Uma outra métrica proposta é denominada CMC: “**Checação dos Métodos Considerados**” (Figura 4.21). Ela permite avaliar os métodos que foram considerados em relação ao número de total de procedimentos existentes no sistema exceto os procedimentos relativos à implementação.

$$CMC = \frac{N_{Met}}{N_{Proc} - N_{Proc_i}}$$

onde:

- N_{Met} é o número de métodos existentes no MAS
- N_{Proc} é o número total de procedimentos existentes no MASA
- N_{Proc_i} é o número de procedimentos relativos à implementação

Figura 4.21 - Métrica para Checagem do Métodos Considerados

Se esse valor for 1 isso significa que todos os procedimentos estão relacionados a uma só classe como construtor ou observador e portanto não havia anomalias, o que costuma não ocorrer em sistemas herdados. Por outro lado o valor sendo maior do que, por exemplo, 2, pode significar que o número de anomalias era muito grande. Nesse caso provavelmente o objetivo da engenharia reversa orientada a objetos era a mudança de funcionalidade do sistema. CMC dando um valor entre de 1 e 2, por exemplo, significa que existiam anomalias e que essas geraram novos métodos que atendem às necessidades do sistema, caracterizando que a engenharia reversa deveria mesmo ter sido aplicada para melhorar o entendimento e facilitar alterações no sistema. O valor 2 foi utilizado para facilitar o raciocínio, considerando-se que se, em média, cada procedimento gerar mais de dois métodos então o trabalho e a complexidade da reengenharia começará a aumentar muito. Esse ponto de corte precisa ser validado na prática. É importante notar que a parte relativa aos dados (atributos e referências) das

classes não é levada em consideração nas métricas propostas. Elas, portanto, referem-se somente ao esforço de conversão dos procedimentos.

Há certamente um esforço de conversão relacionado aos dados, que é dependente da quantidade de classes do modelo, da quantidade e complexidade dos relacionamentos e da quantidade de atributos. Nesta tese não está sendo sugerida nenhuma medida para medir esse aspecto, mas este é um ponto que vale a pena ser mais investigado no futuro. De qualquer forma, como não se está mudando a funcionalidade do sistema e o novo Modelo de Objetos é criado a partir do que existe pode-se conjecturar que esse esforço tem o mesmo nível de complexidade para qualquer tipo de sistema e é proporcional à quantidade de elementos do modelo.

4.7. Reengenharia de Software

Como foi discutido na seção 4.2, um sistema pode passar por um processo de reengenharia por três motivos principais: ampliação de funcionalidade, troca parcial ou troca total da implementação. Algumas diretrizes para que se realize esse tipo de reengenharia são dadas por Jacobson em [45] e reforçadas por Coleman [17] como foi mostrado, também, na referida seção. A partir do método de engenharia reversa orientada a objetos descrito na seção 4.4 recuperam-se todas as informações necessárias para que se construa um modelo de análise do sistema atual, porém sem a alteração da funcionalidade do sistema. Nesta seção serão discutidas algumas formas para a realização da reengenharia do sistema: aperfeiçoamento da funcionalidade, seção 4.7.1.; troca parcial da implementação, seção 4.7.2.; troca total da implementação, 4.7.3. e na seção 4.7.4. algumas combinações que podem ser feitas com as acima citadas.

4.7.1. Com Aperfeiçoamento da Funcionalidade do Sistema

O processo de reengenharia com aperfeiçoamento da funcionalidade do sistema toma por base o conjunto de informações produzidas a

partir do MAS, isto é, o método de engenharia reversa é desenvolvido normalmente e o MAS será o principal instrumento de apoio para essa atividade. Esse modelo será alterado para que em seguida possa ser projetado e implementado de acordo com os novos requisitos do sistema. O termo aperfeiçoamento é usado nesta seção com o mesmo significado utilizado no contexto de manutenções para o aperfeiçoamento [79], isto é, o sistema é modificado visando à ampliação de sua funcionalidade, com a possível renovação ou substituição de funcionalidades existentes.

Os passos que devem ser seguidos para a realização da reengenharia a partir do método de engenharia reversa apresentado serão descritos nas próximas seções e estão relacionados: 1) identificação de quais partes do sistema deve-se alterar; 2) desenvolvimento do Modelo de Objetos Aperfeiçoado; 3) elaboração do Modelo de Ciclo de Vida Aperfeiçoado; 4) especificação do Modelo de Operações Aperfeiçoado; 5) mapeamento do Modelo de Análise do Sistema Aperfeiçoado (MASAp) para o MAS e 6) continuação dos passos de reengenharia.

4.7.1.1. Identificar as Partes a Alterar

Neste ponto deve-se identificar quais são as partes do sistema que devem ser alteradas a fim de realizar o seu aperfeiçoamento. O modelo a ser desenvolvido nessa etapa recebe o nome de Modelo de Análise do Sistema Aperfeiçoado, cuja sigla é MASAp, e todos os modelos gerados para compô-lo terão sempre o sufixo “aperfeiçoado” acrescido ao seu nome. Caso se chegue a conclusão que o sistema deve ser alterado totalmente deve-se utilizar a abordagem usual para desenvolvimento de sistemas orientados a objetos, ou seja, seguir os passos naturais do ciclo de vida de desenvolvimento de um software, não sendo portanto o caso de uma reengenharia. Assim desde já a fase de análise deve ser apoiada por um método para especificação de sistemas seguindo a abordagem de orientação a objetos.

A Figura 4.22 esboça as alternativas existentes relacionadas com o sistema que passa pelo processo de engenharia reversa. Para cada um dos temas

existentes no sistema deve-se identificar quais as partes que permanecem inalteradas, quais sofrem alterações e quais devem ser retiradas.

Figura 4.22 - Resultado da Fase de Identificar as Partes a Alterar

O aperfeiçoamento do modelo pode se dar ao nível de ampliação de sua funcionalidade sendo necessário incluir novos temas ou somente incluir novas classes de objetos. Os temas selecionados anteriormente quando da construção do MASA, podem ser alterados para a elaboração do MASAp. Esse é o caso quando se deseja modelar informações que não estejam relacionadas aos temas já existentes, tornando-se necessária a inclusão de novo tema aos já existentes. O estudo do relacionamento desse novo tema com os demais deve ser realizado para que possa ser especificado quer na fase de análise, se necessário, mas principalmente na fase de projeto quando é construído o Grafo de Interação de Objetos.

Caso o aperfeiçoamento seja necessário somente através da modelagem de novas classes de objetos dentro dos temas já considerados, deve-se identificar quais são as partes do sistema que serão afetadas com a modificação a ser realizada. Deve ser feito um estudo para que se possa conhecer

quais são os atributos que devem ser considerados para essas novas classes e como essas classes passam a se relacionar com as já existentes.

Pode-se também chegar à conclusão que algumas partes do sistema devem ser retiradas. Isso acarreta mudanças no mapeamento entre o MASAp e o MAS que serão discutidas na seção 4.7.1.6.

O passo seguinte a esse seja qual for a modificação a ser realizada é o de desenvolvimento do Modelo de Objetos Aperfeiçoado.

4.7.1.2. Desenvolver o Modelo de Objetos Aperfeiçoado

Quando há expansão dos temas existentes o Modelo de Objetos Aperfeiçoado deve ser desenvolvido a partir do Modelo de Objetos do MAS, incluindo todas as classes de objetos dos novos temas e os seus respectivos relacionamentos, seguindo os conceitos já apresentados na seção 4.4.2.2. Os atributos que compõem essas novas classes também devem ser listados para que o Modelo de Objetos Aperfeiçoado possa ser concluído.

Quando não há inserção de novos temas os modelos produzidos no passo 3.1 do método, Desenvolver o Modelo de Objetos, devem ser revisados para que as novas classes de objetos identificadas, por exigências dos novos requisitos, sejam inseridas ao Modelo de Objetos e relacionadas com os demais existentes através de novos relacionamentos. Novas classes/objetos que passam a compor esse modelo são aquelas referentes às informações que se tornam necessárias, a partir da expansão do sistema, e que as classes que compõem o Modelo de Objetos do MAS não têm condições de fornecer. Deve-se também identificar os atributos de cada objeto inserido. Os novos métodos ainda não são selecionados, pois ainda não foram identificadas todas as operações que devem ser realizadas com esse novo objeto. O Modelo de Objetos é concluído com a identificação dos relacionamentos entre os objetos novos com os já existentes e os relacionamentos entre objetos novos.

O produto resultante deste passo é o Modelo de Objetos Aperfeiçoado, ou seja, completo, incluindo a adaptação feita aos objetos existentes no sistema atual e os objetos que foram inseridos devido a mudanças de funcionalidade do sistema. Embora com frequência menor, pode ocorrer que objetos e relacionamentos sejam removidos do modelo, substituídos por novas abstrações ou simplesmente por não atenderem aos novos requisitos. Nesse caso todos os relacionamentos com as classes que permanecem devem ser removidos, tomando-se o cuidado de evitar inconsistências internas no modelo.

4.7.1.3. Elaborar o Modelo de Ciclo de Vida Aperfeiçoado

O Modelo de Ciclo de Vida do sistema deve ser modificado com expressões regulares relativas às partes expandidas ou modificadas do sistema. Se um novo tema for incluído, provavelmente, devem existir novas atividades que devem ser realizadas pelo sistema. Essas são agora colocadas na forma de expressões regulares compondo o Modelo de Ciclo de Vida. Em quaisquer dos casos, novos temas ou somente novas classes de objetos, as diretrizes a serem seguidas são as mesmas apresentadas na seção 4.4.2. Este modelo deve refletir o que o sistema deve realizar, atribuindo-se nomes sugestivos para as novas operações. Como é sugerido no método Fusion pode-se utilizar de Cenários para modelar as operações que ocorrem no sistema e esses auxiliam na elaboração do Modelo de Ciclo de Vida Aperfeiçoado. O produto resultante desse passo é o conjunto de expressões regulares que possibilita a visão completa da interface do sistema.

4.7.1.4. Especificar o Modelo de Operações Aperfeiçoado

As operações que representam as extensões realizadas no sistema, quer a inclusão de novos temas ou somente de novas classes de objetos, devem ser especificadas seguindo o mesmo padrão que o apresentado para as operações quando não há alteração de funcionalidade, seção 4.4.3. As operações devem ser descritas e para cada uma delas deve haver um objeto controlador. Quando todas as operações já tiverem sido descritas deve-se completar a

descrição de classes realizada parcialmente quando do desenvolvimento do Modelo de Objetos, seção 4.6.1, uma vez que os métodos são agora conhecidos. Aqui, também na reengenharia do sistema os objetos controladores das operações passam a tê-las descritas como métodos, da mesma forma que ocorreu na engenharia reversa. Deve-se observar se as operações já existentes no sistema sofrem alguma alteração em razão das novas inclusões realizadas. Se isso ocorrer o Modelo de Operações Aperfeiçoado deve retratar tais alterações.

4.7.1.5. Mapeamento do MASAp para o MAS

Este passo é semelhante ao descrito na seção 4.4.4, porém a correspondência existe somente entre os objetos e métodos que compõem o MAS, uma vez que somente ele é utilizado para reengenharia com aperfeiçoamento do sistema. Não existe correspondência entre as classes de objetos dos novos temas, as novas classes inseridas bem como os métodos a elas associados, pois não existiam anteriormente no MAS. Se a mudança for com a retirada de alguma classe ocorre a situação inversa à anterior, ou seja, somente o MAS possui as informações e o MASAp não. Esse mapeamento entre os dois modelos expõe claramente quais as modificações sofridas pelo sistema que passou por reengenharia, quer na inserção ou retirada de requisitos. Esse mapeamento visa facilitar a fase de implementação quer se decida pela sua troca parcial ou por uma combinação, por exemplo, em não alterar a implementação existente mas ampliar a funcionalidade do sistema projetando essa nova função de acordo com a abordagem de orientação a objetos.

4.7.1.6. Continuação dos Passos de Reengenharia

Coleman esboçou, conforme descrito na seção 4.2, um roteiro para reengenharia de sistemas composto da fase de engenharia reversa e da fase de engenharia avante, [17]. Pode-se utilizar esse mesmo roteiro aqui, pois para engenharia reversa foram utilizados os passos do método proposto na seção 4.4 e para engenharia avante, seguem-se os passos de um desenvolvimento usual, ou seja, as fases de projeto e implementação de um método orientado a objetos.

Figura 4.23 - Interligação dos Documentos utilizados para Reengenharia com Aperfeiçoamento da Funcionalidade do Sistema

Tabela 4.6 - Resumo dos Passos da Aplicação de Reengenharia com Aperfeiçoamento da Funcionalidade

Reengenharia com Aperfeiçoamento de Funcionalidade	Alterações que devem ser realizadas no MASAp em relação ao MAS
Passos do método de engenharia reversa 1. Revitalizar a Arquitetura do Sistema com base na documentação existente 2. Recuperar o MASA 3. Abstrair o MAS 4. Mapear o MAS para o MASA	Não são alterados
5- Desenvolver o MASAp	
5.1. identificar as partes a alterar	Selecionar quais as partes que devem ser aperfeiçoadas
5.2. Desenvolver o Modelo de Objetos Aperfeiçoado	O Modelo de Objetos deve ser elaborado contendo: <ul style="list-style-type: none"> • as classes necessárias para representar os novos requisitos do sistema; • os relacionamentos entre as novas classes e destas com as já existentes; • os atributos das novas classes para as classes antigas
5.3. Elaborar o Modelo de Ciclo de Vida Aperfeiçoado	O Modelo de Ciclo de Vida deve ser refeito: <ul style="list-style-type: none"> • incorporando todas as operações que representam as ampliações requeridas ao sistema. • retirando as operações correspondentes às partes removidas.
5.4. Especificar o Modelo de Operações Aperfeiçoado	Modelo de Operações: <ul style="list-style-type: none"> • deve ser especificado para cada uma das operações que correspondem às aperfeiçoamento realizado. • não deve conter as operações correspondentes às partes removidas do modelo.
5.5. Mapear o MASAp para o MAS	<ul style="list-style-type: none"> • Se novas partes foram inseridas no MASAp elas não tem correspondência no MAS • Se partes foram removidas elas devem ser removidas do mapeamento do MAS para o MASA. Servindo como guia para a modificação do código
8- Passos da engenharia avante	
8.1.- Fase de Projeto	<ul style="list-style-type: none"> • Diretrizes para a fase de projeto seguindo o método Fusion.
8.2. - Fase de Implementação	<ul style="list-style-type: none"> • Diretrizes para implementação seguindo o método Fusion.

A partir dos modelos construídos, na seção anterior, pode-se passar à fase de projeto do sistema e esta pode ser apoiada inteiramente no método Fusion, apresentado na seção 4.3. Quando essa fase estiver concluída passa-se então à implementação, optando-se por uma linguagem apropriada ao desenvolvimento de sistemas com orientação a objetos. Uma decisão a ser tomada nesta fase é quanto ao código que pode ser reutilizado sendo que o mapeamento descrito na seção 4.7.1.5. pode apoiar essa atividade. Algumas considerações quanto à fase de implementação são abordadas nas seções de 4.7.2. a 4.7.4.

A Figura 4.23 mostra a interligação existente entre os documentos para a realização da reengenharia de um sistema com aperfeiçoamento de funcionalidade. Comparando-se essa figura com a Figura 4.17, pode-se observar que quando a reengenharia é realizada o MASA não é utilizado, as informações são provenientes do MAS e há necessidade de consulta a documentos relacionados com os novos requisitos solicitados ao sistema. Os temas são necessários quando ocorrer a inclusão de algum novo, caso contrário eles são dispensáveis. Por essa razão, na Figura 4.23, eles aparecem em uma caixa tracejada e sua ligação com o Modelo de Objetos também se dá por seta tracejada. A Tabela 4.6 resume os principais passos realizados para a reengenharia de um sistema com aperfeiçoamento de funcionalidade.

4.7.2. Com Troca Parcial da Implementação

Quando após o processo de engenharia reversa orientada a objetos decide-se pela troca parcial da implementação deve-se considerar quais são as partes do sistema que deverão ser implementadas com orientação a objetos e traçar um mapeamento entre elas e as que irão permanecer. Isso visando ao completo funcionamento do sistema, isto é, de modo que o sistema possa se considerado como todo ele sendo formado por objetos, [45]. Esse mapeamento deve ser feito entre cada objeto e a implementação existente.

O projeto do sistema dessas partes que irão sofrer alterações deve ser feito seguindo a abordagem de orientação a objetos, por exemplo, como

proposto no método Fusion. Assim deve-se elaborar os diversos modelos conectando-se, quando existirem, as classes dos novos temas com as já existentes no sistema. Os modelos são os seguintes, conforme descritos na seção 4.3, para a **fase de projeto**: Grafo de Interação entre Objetos, Grafo de Visibilidade para cada uma das classes existentes, a Descrição de Classe; constrói-se os Grafos de Herança e atualiza-se a Descrição de Classes com as novas informações de herança. Deve haver integração entre as partes que permaneceram sem seguir a abordagem orientada a objetos e a projetada com orientação a objetos. A interface entre essas duas partes deve ser cuidadosamente projetada.

Em seguida deve-se passar à **fase de implementação**, também apoiada em Fusion, conectando-se a parte de código já existente, implementado pela abordagem não orientada a objetos, com essa parte que foi trocada e implementada pela abordagem orientada a objetos. A ligação entre as duas partes deve ser cuidadosamente realizada para que o funcionamento do sistema não seja prejudicado. Com o desenvolvimento do sistema ocorrendo dessa forma tem-se a oportunidade de converter o sistema por partes, não havendo a necessidade de realizar toda a implementação de uma só vez, o que é vantajoso para o usuário pois o sistema continua sendo utilizado e gradativamente vai sendo aperfeiçoado/expandido para a realização de suas tarefas.

4.7.3. Com Troca Total da Implementação

O método de engenharia reversa orientada a objetos proposto pode ser utilizado decidindo-se pela troca total da implementação do sistema. Nesse caso há necessidade de elaborar o projeto orientado a objetos a partir do MAS, ou seja, é o processo de engenharia avante.

O método Fusion pode ser utilizado pois ele apóia tanto a fase de projeto quanto a de implementação. Como comentado na seção 4.7.2. os modelos que compõem a **fase de projeto** devem ser elaborados só que aqui para todo o sistema. As informações obtidas na fase de engenharia reversa dão apoio para as fases de projeto e de implementação.

A próxima fase é a de implementação. Uma linguagem orientada a objetos deve ser escolhida e o código fonte deve ser gerado para que haja substituição do código atualmente existente pelo novo, seguindo a abordagem de orientação a objetos. Com esse tipo de implementação não há necessidade de conectá-la com a existente uma vez que todo o sistema foi reimplementado.

4.7.4. Outras Combinações

Podem existir outras combinações de modificações no sistema quando se realiza a reengenharia após o processo de engenharia reversa em um sistema. Uma delas, talvez a mais interessante, refere-se à ampliação da funcionalidade do sistema existente seguindo a abordagem de orientação a objetos sem que o sistema existente seja reimplementado com linguagem orientada a objetos.

Após a realização do processo de engenharia reversa pode-se optar por adicionar novas funções ao sistema existente, sendo essas projetadas já com a abordagem de orientação a objetos, porém preservando a implementação existente. Um estudo deve ser realizado de forma a possibilitar o acoplamento das novas funções (operações) com as que existem no ambiente. Os Modelos de Objetos, Modelo de Ciclo de Vida e de Operações devem ser compatibilizados de forma que não se perca a harmonia do sistema. Dessa forma a implementação da nova funcionalidade é realizada com o desenvolvimento de código seguindo a orientação a objetos e código herdado permanece inalterado. A interface deve sofrer readaptação para abrigar a nova funcionalidade sem prejuízo das que implementam o ambiente. A parte referente a expansão do sistema deve ser totalmente desenvolvida utilizando-se os conceitos e notações do método Fusion uma vez que esse método foi o que apoiou o método de engenharia reversa.

4.8. Considerações Finais

As diretrizes para a realização da engenharia reversa orientada a objetos a que se chegou suprem a ausência referida na seção 4.1. Mais que isso, o grau de detalhes com que são apresentadas permite que sejam consideradas um método para a realização da engenharia reversa orientada a objetos, usando Fusion como notação base. Outros métodos de orientação a objetos, como OMT [81], CRC [95], Booch [7] e outros poderiam ter sido adotados como base para a notação e para o processo de engenharia reversa. Fusion foi selecionado por ser mais recente e por cobrir e integrar as fases de análise, projeto e implementação

Não é de estranhar que se tenha um método para a aplicação de outro método pois o método Fusion é empregado para o desenvolvimento de sistemas orientados a objetos, enquanto que o método aqui apresentado visa à realização de engenharia reversa orientada a objetos em um sistema que foi desenvolvido sem usar técnicas de orientação a objetos. Para isso adotam-se as idéias, as convenções, os conceitos e notações do método Fusion mas acrescidas de atividades que devem ser consideradas para a realização da engenharia reversa. Essas atividades foram detalhadas de modo que se possa dispor de uma orientação firme e segura para a realização de engenharia reversa.

Foram propostas algumas métricas para avaliação do sistema atual e do esforço de conversão. Pode-se avaliar, então o quanto os procedimentos de implementação interferem no sistema existente; que proporção de linhas de código dos procedimentos relacionados à implementação representa em relação ao total de linhas de código existentes no sistema; se um sistema pode ou não ser facilmente convertido e a proporção do número de métodos gerados em relação aos procedimentos de aplicação atualmente implementados. Ao realizar esse estudo o engenheiro de software tem condições de avaliar quanto precisará investir em seu sistema para tê-lo de forma apropriada e facilitada para uso e alterações.

São também fornecidas diretrizes gerais para que o engenheiro de software possa realizar reengenharia com aperfeiçoamento da funcionalidade do sistema em estudo, seguindo o método de engenharia reversa orientada a objetos

proposto neste capítulo. As considerações para reengenharia foram feitas ao nível de troca de funcionalidade, troca parcial e troca total da implementação e foi comentado que essas podem ser combinadas de forma que se realize o aperfeiçoamento do sistema dentro de suas características e necessidades.

No próximo capítulo o método de engenharia reversa aqui apresentado é aplicado ao ambiente StatSim, constituindo um estudo de caso a partir do qual pôde ser feita sua validação em um experimento envolvendo um sistema complexo e com características muito próximas dos sistemas reais em operação (herdados).

Capítulo 5

Estudo de Caso: Engenharia Reversa do Ambiente Statsim

5.1. Considerações Iniciais

Neste Capítulo o método de engenharia reversa orientada a objetos apresentado no Capítulo 4 é aplicado ao ambiente StatSim, descrito no Capítulo 3. Este estudo de caso tem por objetivo validar o método proposto apresentado no Capítulo anterior e produzir um modelo orientado a objetos do ambiente StatSim, visando a futuras manutenções e mesmo a uma possível reengenharia. Esse estudo de caso pode ser classificado como uma engenharia reversa de um sistema sem alteração de funcionalidade, de acordo com a classificação de Jacobson [45].

As notações e ilustrações, tanto relacionadas aos conceitos de orientação a objetos do Método Fusion como aquelas relacionadas ao método de engenharia reversa desenvolvido, referem-se às apresentadas no Capítulo 4. No Apêndice 2 e 3, [72, 74, 75] são apresentados mais detalhes sobre o estudo de caso abordado neste Capítulo. Isso porque procurou-se evitar que o excesso de informação tirasse a visão global que deve ser transmitida no contexto do Capítulo. Assim, cada passo do método apresentado no Capítulo 4 é discutido a seguir e ilustrado com informações extraídas do trabalho de engenharia reversa realizado. O leitor encontrará no apêndice informações adicionais, a menos das que são muito volumosas, as quais podem ser encontradas em [72, 75, 76].

Este Capítulo apresenta na seção 5.2 o passo referente à revitalização da arquitetura do ambiente StatSim baseado nos documentos disponíveis; na seção 5.3 apresenta-se o Modelo de Análise do Sistema Atual (MASA). Com base nas informações adquiridas do MASA elabora-se uma abstração desse modelo construindo-se o Modelo de Análise do Sistema que é mostrado na seção 5.4. Na seção 5.5 é mostrado o mapeamento entre o MAS e o MASA e na seção 5.6 é realizada uma avaliação do método proposto e uma avaliação relacionando o modelo existente e o modelo orientado a objetos produzido. As considerações finais são apresentadas na seção 5.7.

5.2. Revitalização da Arquitetura

As informações disponíveis para o ambiente StatSim no início deste projeto eram o código fonte, as estruturas de dados, relatórios e dissertações relativas aos trabalhos realizados [6, 9, 3, 22, 23, 24, 92, 93]. O processo de engenharia reversa do StatSim foi conduzido manualmente, sendo que a maior parte das informações foi extraída do código fonte, utilizando-se os recursos disponíveis no sistema Unix, como a facilidade de editores de textos. Foram analisados todos os procedimentos que compõem o sistema, os quais foram agrupados em módulos de acordo com as funções que realizam. Os módulos *Bolha.h*, *Bolha1.h*, *Bolha2.h* e *Rot_Bolha.h* como se vê pelos seus próprios nomes, são os que contêm a maioria dos procedimentos que implementam/manipulam o conceito de “estado” de um statecharts, seja na forma gráfica seja semanticamente.

A Figura 5.1 exibe, parcialmente, o código fonte do módulo *Bolha.h* que é o responsável pela criação de um novo estado. Esse módulo tem 1091 linhas de código e se encontra no conjunto de programas do sistema [76]. As partes dos procedimentos aqui apresentados são as referentes à criação de um estado do statechart desenhado no diagrama, no modo de edição. O procedimento *cria_bolha* chama o procedimento *desenha_bolha* e após essa ser desenhada é solicitada confirmação da operação. Quando isso ocorre a estrutura de dados correspondente a essa nova bolha criada é armazenada. Os trechos de código não apresentados na Figura 5.1 são representados por reticências (. . .).

Bolha.h

```

/*Contem todas as rotinas pertinentes a manipulacao de folhas, tais
como criacao, alteracao, etc. A estrutura de dados tambem e'alterada
*/
. . .
/*-----
Confirmada a bolha, guarda na estrutura de dados as informacoes
sintáticas e graficas referentes a ela. E' o procedimento que cria
a arvore.
-----*/
Xv_opaque confirma_bolha(item,evento) /*void -> Xv_opaque */
. . .
/* A bolha foi aceita e vai ser colocada na Estrut. de Dados */
. . .
if (subest[0] == NULL)          /* Preencher raiz_tab */
{
    topo->bolha = &raiz_tab; /* Coloca no topo a raiz da arvore */
    raiz_tab.nivel_visual = topo->nivel; /* Guarda o nivel
visual*/
    raiz_tab.gf_exp.x = -1; /*A bolha expandida ainda nao existe*/
    raiz_tab.gf = bolha; /* Tam e posicao da bolha */
    raiz_tab.pai = NULL;
    raiz_tab.prim_filho = NULL;
    raiz_tab.prox_irmao = NULL;
    . . .
else
    aloca_bolha (&no_filho); /*aloca memoria p/ um novo filho*/
    no_filho->gf = bolha, /*Atualiza no com inf. graficas */
    . . .
    copia_nome(&raiz_tab.nome_est,buf); * Atualiza o nome do no
*/
    raiz_tab.gf.x_nome = C_NOM.x; /* Pos. de escrita */
    raiz_tab.gf.y_nome = C_NOM.y;
    nro_est = 1; /* Inicializa indice de subest */
    . . .
    WIN_EVENT_PROC, reparte_bolha,
    . . .
    indica_oper_corrente((topo->fr_can)->fr, OPER_CORRENTE);
. . .
    limpa_operacao((topo->fr_can)->fr, &OPER_CORRENTE);
. . .
    chamada_atividade(nro_est-1,231);
. . .
/*-----*/
Habilita o canvas para eventos apropriados e chama a rotina de
tratamento de eventos
-----*/
void cria_bolha(menu, menu_item)
. . .
    xv_set(canvas_paint_window((topo->fr_can)->can),
            WIN_CONSUME_EVENTS, WIN_MOUSE_BUTTONS,LOC_DRAG,NULL,
            WIN_EVENT_PROC, desenha_bolha,
            NULL);
    OPER_CORRENTE = 1;
    indica_oper_corrente((topo->fr_can)->fr,OPER_CORRENTE);
}

```

Figura 5.1 - Código Fonte parcial do Módulo Bolha.h

Módulo =====	Descrição =====
Bolha.h	<p>Contém todas as rotinas pertinentes a manipulação de bolhas, tais como criação e alteração.</p> <p>A estrutura de dados também é alterada.</p>
<u>Procedimentos:</u>	
confirma_bolha	<p>Confirmada a bolha, guarda na estrutura de dados as informações sintáticas e as gráficas referentes a ela. É o procedimento que cria a árvore de estados.</p> <p><u>CHAMA:</u> aloca_bolha (rot_bolha) chamada_atividade copia_nome (rot_bolha) indica_oper_corrente (rot_bas) limpa_operação (rot_bas) reparte_bolha</p> <p><u>CHAMADO POR:</u> solicita_atributos</p>
cria_bolha	<p>Habilita o canvas para eventos apropriados e chama a rotina de tratamento de eventos.</p> <p><u>CHAMA:</u> desenha_bolha indica_oper_corrente (rot_bas)</p>
solicita_atributos	<p>Criada a bolha, cria-se a janela que solicita os tipos de atributos da mesma: Nome, Tipo (AND, OR).</p> <p><u>CHAMA:</u> confirma_bolha cria_panel_nro_filhos mostra_nome refaz_abandona_bolha</p> <p><u>CHAMADO POR:</u> desenha_bolha</p>

Figura 5.2 - Trecho da documentação produzida pela Revitalização da Arquitetura

Para cada um dos módulos existentes no ambiente StatSim, semelhantes ao apresentado na Figura 5.1, foram listados todos os procedimentos que os compõem e construída a relação CHAMA/CHAMADO POR. A documentação completa produzida a partir dessa análise é apresentada em [72]. A Figura 5.2 ilustra a documentação obtida neste passo, apresentando parte do módulo Bolha.h, a partir das informações obtidas do código fonte que é parcialmente exibido na Figura 5.1. O procedimento **solicita_atributos** aparece apenas na Figura 5.2.; o procedimento **cria_bolha** chama dois outros procedimentos como pode ser visto na Figura 5.1 marcados em negrito. A Figura 5.1 mostra, em negrito, todos os procedimentos que **confirma_bolha** chama para realizar suas funções. Na Figura 5.2 pode-se encontrar esses procedimentos relacionados na cláusula CHAMA do procedimento **confirma_bolha**. O procedimento não é apresentado inteiramente porque se optou por fornecer apenas a idéia geral do mecanismo utilizado para obtenção das informações.

De acordo com a convenção adotada e exposta na seção 4.4.1, os procedimentos que constam de outros módulos apresentam o nome desse módulos entre parênteses. O procedimento **confirma_bolha** chama, entre outros, o procedimento **aloca_bolha** que pode ser encontrado no módulo Rot_Bolha e o procedimento **limpa_operação** que pertence ao módulo Rot_Bas. A fim de facilitar a pesquisa sobre a localização dos procedimentos dentro dos módulos foi criado um índice geral que contém todos os procedimentos que compõem o ambiente StatSim e na frente de seu nome, entre parênteses, o módulo onde estão alocados. Esse índice é ordenado por ordem alfabética dos procedimentos.

5.3 Recuperação do Modelo de Análise da Solução Atual (MASA)

Com base nas informações obtidas anteriormente e com o conhecimento adquirido pelo uso e manutenção do ambiente pela autora, este modelo foi construído completamente. Os resultados de cada sub-passo são comentados a seguir. A Figura 4.17 mostra que a documentação produzida no passo de Revitalização da Arquitetura do Sistema é agora utilizada para a recuperação do Modelo de Análise do Sistema (MASA). Essa documentação, na

engenharia reversa, faz o papel do Documento de Requisitos, que é o primeiro passo do método Fusion como pode ser observado através da Figura 4.1.

Em um desenvolvimento normal de sistemas após a definição dos requisitos passa-se à elaboração do modelo de análise. Utilizando-se engenharia reversa, a partir da documentação obtida com a Revitalização da Arquitetura, inicia-se o processo de abstração do sistema existente elaborando-se a lista dos Temas revelantes do sistema.

5.3.1 Definição dos Temas

Seguindo as diretrizes do sub-passo 2.1 (Tabela 4.2), do método de engenharia reversa orientada a objetos apresentado no Capítulo 4, as entradas/saídas das informações para edição de statecharts no ambiente StatSim estão no tema **diagrama**; os **relatórios** produzidos pelos vários tipos de simulação, como um tema relacionado a saída; **statechart permanente**, como um tema correspondente às informações armazenadas em disco e que são carregadas para processamento, pois não se usou um Sistema de Gerenciamento de Banco de Dados (SGBD) na implementação atual e **statechart temporário**, como um tema correspondente aos dados armazenados na memória RAM quando o statechart é carregado do disco. A Figura 5.3 apresenta o esquema de relacionamento dos temas.

Figura 5.3 - Temas

```

/*-----
                                ESTRUT.h                                VER-2
Contem as estruturas basicas para simulacao de STATECHARTS
-----*/

. . .
typedef struct tab {
    CADEIA nome_est;          /* nome do estado          */
    int ind_sub_est;          /* Valor do Indice do subest p/ este no'
*/
    int tipo;                  /* tipo final(and/xor/fim) ( 0 / 1 / 2 ) */
    int defo;                  /* default/nao =(1/0) defH/ defH* =
(2/3)*/
    int nivel_visual;          /* Nivel visual da bolha */
    int nivel;                  /* altura-arvore (raiz=0) */
    int lig;                    /* indica "ligado" (1/0) */
    int clig;                   /* indica "ligado" (1/0) */
    int ult_lig;               /* indica hist "ligado" */
    int num_steps_active;      /* indica o numero de passos ativos */
    int num_steps_consec;      /*indica o numero de passos
consecutivos*/
    int num_consec;            /* indica o numero de passos consecutivos
*/
    int num_consec_average;    /*indica o numero medio de passos
consecutivos */
    struct tab *pai;           /* pont. para o pai */
    struct tab *prim_filho;     /* pont. para o 1o. filho */
    struct tab *prox_irmao;     /* pont. para prox irmao */
    struct tab *ant_irmao;      /* pont. para irmao ant. */
    PILHA_POL *ativ;           /* pont. para pilha polonesa p/ avaliacao da
atividade */ /* ATIVIDADE */
    GRAF gf;                   /* Inf. graficas da bolha nao exp. */
    GRAF gf_exp;               /* Inf. graficas da bolha expandida */
    int sentinela;             /* alerta = 1234 */
}NO_ESTADO;

. . .
typedef struct trans {
    PILHA_POL *ev_aval;        /* evento a ser avaliado para efetivacao
da transicao */
    PILHA_POL *ac_aval;        /* Pilha polonesa p/ avaliacao de um acao
*/
    int ind_est_fonte;          /* indice estado fonte */
    int ind_est_destino;        /* indice estado dest */
    int num_disp;
    int num_avalia;

    . . .
}TRANSICAO;

. . .
typedef struct sub_trans {
    . . .
}SUB_TRANSICAO;

typedef struct trans_ev {
    TRANSICAO *trans;           /* pont p/ trans relativa ao evento */
    struct trans_ev *prox_trans_ev; pont p/ proxima trans
relativa ao evento */
} TRANS_EV;

. . .

```

Figura 5.4 - Código fonte das Estruturas de Dados do StatSim

Os modelos a seguir são aqueles descritos no método Fusion e ilustrados através da Figura 4.1 como os que compõem a fase de análise: Modelo de Objetos e Modelo de Interface do Sistema.

5.3.2 Desenvolvimento do Modelo de Objetos

Para os temas selecionados foram recuperadas as informações que cada um deles modela e foram criadas as classes de objetos, como recomendado no sub-passo 2.2 da tabela 4.2. Nem sempre as informações contidas no documento de revitalização da arquitetura do sistema foram suficientes para a classificação dos procedimentos, sendo necessárias consultas diretamente ao código fonte. A Figura 4.15 mostra essa ligação. Quando do desenvolvimento do ambiente StatSim, o método adotado foi usar a linguagem C com a idéia de tipos abstratos de dados: definir estruturas de dados implementando entidades do problema e procedimentos agindo sobre essas estruturas. A Figura 5.4 mostra um trecho do código fonte que contém a descrição das estruturas de dados. A listagem completa do código fonte do ambiente StatSim encontra-se em [75]. Nessa figura é exemplificado completamente a estrutura de `NÓ_ESTADO` e de `TRANS_EV`; as outras estruturas são apresentadas parcialmente.

A notação utilizada no restante deste Capítulo é a seguinte:

- a) referente à estrutura de dados do código fonte atualmente existente: todas as letras em maiúsculas, por exemplo, `NÓ_ESTADO`.
- b) referente à classe do Modelo de Objetos: a primeira letra de cada nome da classe é maiúscula e as demais são minúsculas e em negrito. Por exemplo: **`Nó_Estado`**.
- c) referente a relacionamento: em itálico.
- d) referente a procedimentos: em negrito, o nome que consta da implementação.
- e) referente a operações/métodos: letras minúsculas, em negrito.

A consulta a essas estruturas de dados facilitou a identificação das estruturas relevantes para o Modelo de Objetos. As classes de objetos que compõem o Modelo de Objetos foram obtidas considerando-se as principais estruturas de dados para o armazenamento temporário de um statechart. Os nomes adotados para as classes são os mesmos que os usados na definição das estruturas de dados que as implementam no ambiente StatSim (Figura 5.4); **NÓ_ESTADO**, **TRANSIÇÃO**, **TRAN_EV**, **SUB_TRANSIÇÃO**, e assim por diante. O Modelo de Objetos produzido para o tema Statechart Temporário é mostrado na Figura 5.5. Nesse modelo somente a primeira letra de cada palavra que compõe o nome é escrita com maiúsculas e as demais com letras minúsculas, assim, as classes selecionadas são **Nó_Estado**, **Transição**, **Trans_Ev**, etc.. Os Modelos de Objetos para os demais temas encontram-se no Apêndice 2.

Figura 5.5 - Modelo de Objetos do MASA para o Tema Statechart Temporário

Após ter-se especificado as classes de objetos deve-se analisar quais os relacionamentos que existem entre elas. Para isso consideram-se as ligações existentes entre as estruturas de dados atualmente definidas. Por exemplo, existe um ponteiro, **pai*, que é transformado em um relacionamento de **Nó_Estado** com si mesmo, que também recebe o nome *Pai*. O mesmo procedimento é utilizado para todos os outros ponteiros declarados nas estruturas. No caso da classe **Trans_Ev**, nota-se pela Figura 5.5 que ela relaciona-se com a classe **Transição** através do relacionamento *Ev Pertence* e consigo mesma através do relacionamento *Próxima Trans_Ev*. Essas informações foram extraídas da declaração de **TRANS_EV**, que pode ser vista na Figura 5.4. Todos os elementos de dados não simples, isso é, que são ponteiros ou referências para outras estruturas de dados conforme implementação realizada, são transformados em relacionamento entre classes quando da construção do Modelo de Objetos. Os elementos de dados simples continuam sendo tratados como tais. Os nomes adotados para os relacionamentos são os mesmos nomes que os utilizados na declaração da estrutura correspondente. Para o caso de **Nó_Estado** esses nomes são *Pai*, *Prim Filho*, *Prox Irmão* e assim por diante. Notem que neste modelo, por se tratar de abstração de uma implementação, todas as cardinalidades foram simplificadas, não existindo relacionamento M:N. Os demais modelos de objetos relativos aos outros temas podem ser encontrados no Apêndice 2.

Como pôde ser visto no Capítulo 4, o método Fusion propõe que cada classe pertencente ao Modelo de Objetos seja descrita com seu nome e seus atributos. A descrição de **Nó_Estado** com seus elementos de dados e alguns procedimentos é exibida na Figura 5.6. Os elementos de dados são os que compõem a estrutura de dado que definem **NO_ESTADO**, conservando-se os mesmos nomes já apresentados na Figura 5.4. Os procedimentos estão relacionados com algumas das operações implementadas sobre **NO_ESTADO** como para apagar um estado, indicar que o estado tem um arco "default" associado, verificar se a bolha é válida, etc..

Os procedimentos associados são procedentes da documentação produzida na fase de revitalização da arquitetura do sistema (Figura 5.2) do conhecimento adquirido quando da utilização do ambiente e de consulta ao código fonte (Figura 5.1). Partindo-se do programa principal e seguindo-se a ordem de chamadas dos procedimentos foi possível recuperar os procedimentos relevantes

à classe. Os procedimentos são apresentados sem argumentos com a finalidade de simplificar, uma vez que essa informação é obtida facilmente através da implementação existente. A lista de procedimentos apresentados na Figura 5.6 é parcial.

Classe: Nó_Estado	
Elementos de Dados: nome_est ind_sub_est tipo defo nivel_visual num_steps_active num_consec sentinela lig clig ult_lig gf gf_exp nivel num_steps_consec num_consec_average *pai *prim_filho *ant_irmão *prox_irmão *ativ	
Procedimentos: Acha_anc_comum Acha_cam_independ Acha_indsub_arv aloca_bolha apagar_bolha armazena_filhos bolha_do_arco bolha_válida chamada_atividade Cria_no_arv desaloca_bolha Copia_clig_para_lig desenha_bolha acha_estado_nome Acha_ind_est acha_nro_filhos ancestral_direto armazena_arco_default atualiza_árvore bolha_normal busca_bolha confirma_bolha copia_nome draw_bolha desenha_bolha	

Figura 5.6 - Classe Nó_Estado

A partir da descrição das classes de objetos realizou-se a análise dos procedimentos com o objetivo de determinar qual o tipo de acesso às classes do Modelo de Objetos para consulta (observador) ou para modificação (construtor). Foram analisados e classificados 495 procedimentos neste subpasso, referentes a todas as classes de objetos que compõem o Modelo de Objetos. Esses procedimentos foram classificados de acordo com a nomenclatura apresentada na seção 4.3 - Desenvolver o Modelo de Objetos.

Procedimentos		
Acha_anc_comum	(o/rot_simula)	o
acha_estado_nome	(o/rot_sim)	o
Acha_ind_est	(l/rot_simula)	i
acha_nro_filhos	(o/bolha2)	o
Achar_ancest_dest	(o/rot_simula)(o/transição)(c/subtrans)	o+c
aloca_bolha	(c/rot_bolha)	c
apaga-statechart_memoria	(c/arquivo)	c
apagar_bolha	(o/bolha1)	o
armazena_arco_default	(c/rot_arco/G)	c
armazena_filhos	(c/rot_bolha)	c
atualiza_árvore	(o/atividades) (c/tab-atividdes)	oc
bolha_do_arco	(o/rot_arco) (o/trans, subtrans)	o+
bolha_normal	(o/rot_bolha/G)	o
bolha_valida	(o/rot_consist/G)	o
busca_bolha	(o/rot_bolha) (o/bolha-diagramas)	o+
busca_bolha_pelo_nome	(o/rot_bolha) (o/bolha-diagramas)	o+
busca_indice_bolha	(o/rot_arco)	o
busca_no	(o/atividades) (o/no-estado)	o+
busca_pai	(o/rot_bolha/G)	o
captura_bolha	(o/zoom)	o+
chamada_atividade	(c/bolha) (c/pilha_pol)	c+
Checa_estrut	(o/rot_simula)	o
Clear_arvore	(c/rot_avalia)	c
confirma_bolha	(c/bolha)	c
Constroi_arvEOU	(o/rot_semantic)	o
Copia_clig_para_lig	(c/rot_lig)	c
Cria_no_arv	(c/rot_semantic)	c
Cria_subs_arvEOI	(c/rot_semantic)	c
desaloca_bolha	(c/rot_bolha) (o/atividade)	oc
desaloca_nos_de_estados	(o/rot-bolha) (c/atividade)	oc
desaloca_nos_e_transicoes	(c/bolha1/2c) (o/subtransicao;c/ativ)	oc+
desenha_bolha	(c/bolha) (c/bolha/diagramas)	c+
draw_bolha	(o/arquivo) (c/bolha)	oc
Efetua_sim	(c/rot_simula) (o/subtrans) (c/transição)	oc+
refaz_abandona_bolha	(c/bolha/2c) (c/bolha)	c+

Figura 5.7 - Classificação de Parte dos Procedimentos Associados à Classe

A Figura 5.7 mostra parte dos procedimentos associados à classe **Nó_Estado** classificados de acordo com a convenção adotada na seção 4.4.2.2, Capítulo 4. Por exemplo, para a classe **Nó_Estado**

busca_bolha_pelo_nome (o/rot_bolha) (o/bolha/diagrama)

significa que o procedimento está no módulo **rot_bolha**, é observador da classe **Nó_Estado** e também observador da classe **bolha** do tema **diagrama**. Assim sua classificação é **(o+)**. Já o procedimento:

Acha_anc_comum (o/rot_simula)

está no módulo **rot_simula** e é um observador somente da classe **Nó_Estado**, **(o)**. Dessa forma, esses procedimentos aparecem na Figura 5.7 contendo em negrito a sua classificação.

A Tabela 5.1 exibe os resultados obtidos a partir da análise realizada para todos os procedimentos associados a todas as classes de objetos descritas nos temas considerados, ou seja todos os procedimentos do ambiente StaStim. Na seção 5.6 é realizada a análise dos resultados obtidos.

Tabela 5.1 - Resumo da Classificação dos Procedimentos do Ambiente StatSim

Classificação	Quantidade
observadores (o)	141
construtores (c)	188
anomalia (oc)	17
anomalia (oc+)	09
anomalia (o+c)	03
anomalia (o+c+)	03
anomalia (c+)	13
anomalia (o+)	16
anomalia (i)	105
Total	495

**Tabela 5.2 - Algumas Opções Existentes na Interface do Ambiente para o
Modo de Edição**

OPÇÕES DA INTERFACE	OPERAÇÕES	OPERAÇÕES IMPLEMENTADAS
States	Create	cria_bolha desenha_bolha aloca_bolha busca_pai contido default_gc determina_raio intersecta preenche_bolha rubber_band_gc traça_bolha traça_retangulo solicita_atributos indica_oper_corrente
	Remove	deleta_bolha abandonar_processo apagar_bolha indica_oper_corrente
	Decompose	expande_bolha captura_bolha indica_oper_corrente
	Query	info_bolha identifica_bolha indica_oper_corrente
Transitions		
	Create 1:1	cria_arc_def_1p1 cria_jan_rotulo cria_jan_rotulo_def confirma_def_1p1 desenha_def_1p1 finaliza_def_1p1 hide_closure inicializa_variaveis_arco refaz_def_1p1 sel_hist_def_1p1
	Create Default	cria_arc_def_1p1 cria_jan_rotulo cria_jan_rotulo_def confirma_def_1p1 desenha_def_1p1 finaliza_def_1p1 hide_closure inicializa_variaveis_arco refaz_def_1p1 sel_hist_def_1p1
	Create M:N	cria_arc_MparaN abandona_valores_MN confirma_valores_MN marca_ponto_intersecção
	Remove Default, 1:1, M:N	remove_arco abandona_remoção apagar_arco
	Query	info_arc identifica_arco

Após elaborar o Modelo de Objetos, o próximo passo é o desenvolvimento do Modelo de Interface do Sistema compreendido pelo Modelo de Ciclo de Vida do Sistema e pelo Modelo de Operações.

5.3.3 Definição do Ciclo de Vida

Reportando-se à Figura 4.17 pode-se notar que o Modelo de Ciclo de Vida é obtido a partir das consultas aos documentos correspondentes à Revitalização do Sistema e à execução do sistema. O ambiente StatSim é composto de um modo de edição e outro de simulação. A edição é realizada sobre o diagrama e corresponde a operações para manipular estados e transições, declaração de variáveis e carregamento e armazenamento de statecharts. Na simulação pode-se escolher o tipo de simulação, interagir de acordo com esse tipo e obter relatórios da simulação.

```

life cycle STATSIM: (Edit | Simulate)*

Edit = (Files . carregar_arq. nome_stchart ) | (States . cria_bolha) .
      (States | Transitions | Files | Activities | Variables | Load_Statechart | Hide/Closure)*

Simulate = Files . carregar_arq . nome_stchart . (Files | Decompose | Variables |
      Default_Configuration | Simulation | Miscellaneous)

States = Create . (Create | Remove | Query | Decompose)*

Create = cria_bolha . #confirmação_bolha_desenhada.

cria_bolha = desenha_bolha . solicita_atributos. (refaz_bolha | abandonar_processo)

Remove = #escolha_da_bolha_a_delir . #confirmação_bolha_a_delir .
      (apagar_bolha | abandonar_processo)

Query = #escolha_da_bolha .identifica_bolha. #informações_estado

Decompose = captura_bolha. #escolha_da_bolha. Edit
...

```

Figura 5.8 - Modelo de Ciclo de Vida

O Modelo de Ciclo de Vida foi um resultado relativamente simples de ser produzido devido à interface guiada por menus e janelas do StatSim e também à disponibilidade do manual do usuário [73], produzido como parte das

Operação:	cria_bolha
Descrição:	cria uma bolha no canvas, obtém todas as informações a ela relacionadas e armazena essas informações em uma estrutura de dados chamada nó-estado.
Lê:	supplied ponto inicial, ponto final, nome-estado, tipo, nome_atividade1, nome_atividade2, ...; Nó_Estado
Modifica:	new bolha, new estado
Envia	
Assume:	a bolha está dentro dos limites; a bolha não intercepta irmão.
Resultados:	<ul style="list-style-type: none"> - uma nova bolha foi criada e desenhada no canvas - um novo Nó_Estado foi criado - foi criada e gravada a estrutura de Nó_Estado - o Nó_Estado tem um único nome - o Nó_Estado tem um único tipo. - o Nó_Estado está dentro dos limites - se o Nó_Estado criado foi o primeiro, então nível recebeu 0 senão nível recebeu initial nível + 1. - num_steps_active recebeu 0 - num_steps_consec recebeu 0 - num_consec_average recebeu 0 - clig, lig e ult_lig receberam "NÃO LIGADO" - sentinela recebeu 1234 - *pai, *prim_filho, *prox_irmao, *ant_irmao foram ligados e atualizados apropriadamente - se tipo.atributo = AND cada estado_filho tem um único nome, fornecido por nome_estado - se bolha normal, gf. recebeu as posições gráficas , recebeu ponto inicial e ponto final - se bolha expandida, gf_exp recebeu as posições gráficas da bolha expandida, recebeu ponto inicial e ponto final - se existir atividade *ativ, os nomes-atividade foram fornecidos e *ativ foi atualizado apropriadamente - a lista de subestado (subest) foi atualizada

Figura 5.9 - Modelo de Operação: cria_bolha

Com todos os modelos produzidos, todas as informações relevantes do sistema atual para esta fase foram obtidas, finalizando-se a recuperação do MASA. A seguir passa-se à elaboração do modelo de análise do sistema (MAS) que é abstraído do modelo de análise do sistema atual (MASA).

5.4. Abstração para o Modelo de Análise do Sistema (MAS)

Para elaborar o Modelo de Análise do Sistema (MAS) são considerados os documentos produzidos na seção 5.3 e o conhecimento do engenheiro de software para que sejam descritos os aspectos relevantes ao sistema.

A Figura 4.17 mostra que os documentos produzidos no MASA são utilizados na elaboração do MAS. Há correspondência entre os elementos do MASA e do MAS, exceto temas que somente aparecem no MASA pois as informações que o sistema manipula continuam sendo as mesmas, ou seja, a funcionalidade do ambiente não será alterada durante a engenharia reversa orientada a objetos. Caso a funcionalidade do sistema fosse alterada, dever-se-ia tratá-la neste passo. Após a recuperação do MASA a nova funcionalidade deveria ser incluída no MAS.

Para construir os modelos sugeridos pelo método Fusion para o MAS, consideram-se não mais os aspectos de implementação mas somente o modelo de análise. Os modelos descritos aqui devem ser tratados de modo natural, como quando se desenvolve sistemas seguindo a abordagem orientada a objetos.

5.4.1. Desenvolvimento do Modelo de Objetos

Analisando-se o sistema torna-se evidente que deve haver a representação de estados e transições. Dessa forma criam-se as classes **CJ-Estado** e **CJ-Transição** para representar esses conceitos de todos os estados e todas as transições do statechart. Continuando essa análise nota-se a necessidade de modelar expressões que são executadas pelo ambiente StatSim quando da simulação de um statechart. Cria-se então a classe denominada **CJ-Expr. Executável**. Variáveis, eventos e procedimentos relacionados com as expressões executáveis, estados e transições devem também ser representadas. Dessa forma, pôde-se elaborar o Modelo de Objetos de **Statechart** representado somente através da agregação das classes acima descritas, como mostra a Figura

5.10. O nome é apenas **Statechart** porque não há mais necessidade de distinção entre armazenamento temporário e permanente no modelo orientado a objetos. No Apêndice 3 são encontrados os Modelos de Objetos para os demais temas considerados.

Figura 5.10 - Modelo de Objetos de Statechart com Alto Nível de Abstração

Na Figura 5.11 o Modelo de Objetos é mostrado parcialmente, focando como assunto os relacionamentos entre as classes **Estado** e **Transição**. Nessa figura elas são expandidas sendo que cada uma pode ser representada através de relacionamentos de generalização. Pode-se observar, aqui, que as cardinalidades não são só simples, como no Modelo de Objetos do MASA, existindo relações “muitos para muitos”. Quando a relação acontece com uma mesma classe nos dois sentidos, é declarado qual o papel de cada classe no relacionamento para evidenciar o seu sentido.

No Modelo de Objetos do MASA, Figura 5.5, considerou-se **Nó_Estado** como um objeto que armazena as informações de todos os tipos de estado. Segundo Harel, [32], como foi visto na seção 3.2 Capítulo 3, um estado pode ser classificado como “atômico” ou “superestado”. Neste passo **Nó_Estado** é representado por **Estado** como mostra a Figura 5.11. Os relacionamentos da classe **Nó_Estado**, *Prim_Filho*, *Prox_Irmão*, *Ant_Irmão* não existem para a classe

Estado, pois não há mais a restrição de implementação através de árvore binária. O relacionamento “*Decomposto*” de **Estado** especifica a relação que todos os estados têm um pai, a menos do estado raiz, o que é expresso através da cardinalidade 0..1 e que um estado pode ter zero ou mais filhos, expresso por *.

Figura 5.11 Tema Statecharts: Modelo de Objetos para Estado e Transição com seus Relacionamentos

De acordo com Harel [32], as transições podem ser “default”, um para um e M para N, como foi visto na seção 3.2 do Capítulo 3. Desse modo a classe **Transição** é transformada em uma generalização de **Trans M:N**, **Trans 1:1** e **Trans Default**. Essa classe relaciona-se com a classe **Estado** através dos relacionamentos denominados de acordo com o tipo da transição. Por exemplo, o relacionamento “*Destino Default*” especifica que um estado é destino para uma transição “default”. Os relacionamentos “*Tem Origem*” e “*Tem Destino*” referem-se às transições do tipo um para um. As transições do tipo M-para-N, como o próprio nome diz, podem ter múltiplas origens e múltiplos destinos e isso é expresso pelo

relacionamento “*Tem Origem Múltipla*” e “*Tem Destino Múltiplo*”. No Modelo de Objetos do MASA, Figura 5.5 **Nó_Estado** e **Transição** relacionavam-se através dos mesmos relacionamentos citados só que aqui, no MAS, esse relacionamento acontece com a subclasse específica de **Transição** e com a classe **Estado**, uma vez que a origem e/ou destino da transição podem ser estados que tem decomposição ou estados atômicos.

Figura 5.12- Tema Statechart: Modelo de Objetos para as Expressões Executáveis

A Figura 5.12 exibe o Modelo de Objetos para a classe **Expressão Executável** que é uma generalização das classes **E-Atividade**, **E-Evento** e **E-Ação**. Ela representa a parte do ambiente Statsim que modela as ações, eventos e atividades que são avaliadas quando um statechart é simulado. A figura mostra também o relacionamento de **Expressão Executável** com as classes **Estado** e

Transição através de “*executa*”. O relacionamento “*executa*” entre **Estado** e **E-Atividade** representa as atividades que estão associadas aos estados. O relacionamento “*executa*” entre **Transição** e **Expressão Executável** está representando os eventos, ações e atividades que estão associados às transições e que são tratados como expressões executáveis quando o ambiente StatSim está sendo simulado.

Considere a classe **Nó_Estado** com os elementos de dados listados na figura 5.6. Para cada um dos elementos de dados deve-se investigar qual o seu papel no sistema atual, isto é, se será ou não necessário no MAS. Os elementos de dados que não são necessários são aqueles relacionados com a forma de implementação utilizada ou que devido à generalização de uma classe o elemento passe a pertencer a uma das suas classes.

Figura 5.13 - Relacionamentos da Classe Estado

A classe **Estado** tem alguns relacionamentos de generalização/especialização bastante intuitivos, que à primeira vista deveriam constar do Modelo de Objetos. Um deles é determinado pelo atributo “tipo” e o outro pela diferenciação entre estados atômicos e compostos, como mostra a figura 5.13. Dependendo da semântica utilizada, o estado atômico é considerado

por exemplo, e mesmo outros métodos orientados a objetos, como o OMT é que as entradas e saídas são modeladas com as classes no Modelo de Objetos do sistema. Como há muitos relacionamentos desse tipo e todos eles são relativamente simples, a Figura 5.15 dá uma idéia de como seria o modelo completo, que não é mostrado nesta tese.

Figura 5.15 - Modelo de Objetos do Sistema

Posteriormente, após a definição do Ciclo de Vida do sistema definem-se os seus limites e a classe Engenheiro de Software é definida como externa ao sistema, tornando-se um agente.

A Tabela 5.3 mostra os atributos da classe **Estado** (MAS) abstraídos da classe **Nó_Estado** (MASA). Alguns elementos de dados não têm correspondência por serem específicos do tipo de implementação utilizada, como por exemplo, sentinela. No MASA a informação de que um estado possui atividade associada é implementada por um ponteiro, *ativ, entre as duas estruturas. No MAS a representação ocorre através de *E_Atividade/executa por ser um relacionamento entre duas classes distintas. Quando o relacionamento ocorre com a mesma classe e nos dois sentidos, coloca-se, também, o papel que a relação desempenha. Por exemplo, *filho é representado por col *Estado/Decomposto/filho, significando que um estado é filho de outro estado.

A lista de métodos associados à classe **Estado** é composta por métodos relacionados com os procedimentos classificados sem anomalias (ver

como sendo do tipo AND [36] ou como não tendo tipo [32]. Neste último caso as duas abstrações da Figura 5.13 poderiam ser combinadas da forma como mostra a Figura 5.14.

Entretanto, nenhum ganho significativo em concisão do diagrama, economia do número de relacionamentos ou particionamento dos relacionamentos existentes, acrescentando mais semântica ao diagrama pôde ser observado com essas abstrações. Dessa forma, preferiu-se manter a classe **Estado** sem especialização, como mostrado na Figura 5.11. Não fica descartada a hipótese de que, em ocorrendo a fase de projeto, essas abstrações se tornem relevantes e sejam incorporadas ao projeto. Uma razão para isso poderia ser a existência de métodos diferenciados para certos tipos de estados.

Figura 5.14 - Combinações das Duas Abstrações da Figura 5.13

Uma diferença entre a modelagem de objetos preconizada pelo Fusion e a modelagem convencional usando o Modelo Entidade-Relacionamento,

Figura 5.7), ou seja, todos aqueles construtores ou observadores da classe **Nó_Estado** que realizam operações relevantes sobre as classes agora existentes. Os procedimentos com anomalias devem ser analisados de forma que sejam inseridos nessa lista com a função específica que realizam sobre a classe, dando origem a novos métodos.

Tabela 5.3 - Atributos MASA x MAS da classe que representa as informações de ESTADO

Atributos da Classe Nó_Estado (MASA)	Atributos da Classe Estado (MAS)
nome_est	nome
ind_sub_est	_____
tipo	tipo
defo	default
nível_visual	nível_visual
nível	nível
lig	ativação
clig	cópia ativação
ult_lig	última ativação
num_steps_active	passos ativos
num_steps_consec	passos consecutivos
num_consec	num médio ativ consecutivas
num_consec_average	média passos consecutivos
gf	informações gráficas normais
gf_exp	informações gráficas expandidas
sentinela	_____
*pai	col *ESTADO/Decomposto /pai
*filho	col *ESTADO/Decomposto /filho
*ant_irmão	_____
*prox_irmão	_____
*ativ	*E_Atividade/executa

A Tabela 5.4 apresenta o mapeamento de alguns procedimentos do MASA para seus respectivos correspondentes no MAS. Os procedimentos foram analisados quanto às anomalias e os métodos resultantes foram acrescentados ao modelo de objeto seguindo-se as diretrizes recomendadas na Seção 4.4. Para simplificar, não são mostrados os parâmetros dos métodos e procedimentos nesta tabela. Esses parâmetros podem ser obtidos considerando-se os parâmetros dos procedimentos implementados desde que não se leve em conta os detalhes de implementação.

O procedimento **desaloca_nos_de_estados** tem como parâmetro nó que é do tipo **NÓ_ESTADO**, referente ao estado que deve ser retirado do statechart que está sendo editado e na Tabela 5.4 esse procedimento é listado sem parâmetro. Outro exemplo é o caso do procedimento **confirma_bolha**, exibida na Figura 5.1, que tem como parâmetros item e evento. Item é um parâmetro referente a implementação utilizada para criação do canvas relacionado com esse procedimento, enquanto que evento refere-se ao botão do mouse que será pressionado confirmando ou não a criação da nova bolha.

Tabela 5.4 - Procedimentos do MASA e métodos correspondentes do MAS

Procedimentos do MASA	Métodos do MAS	Classe / Tema
Acha_ind_est (i)		
aloca_bolha (c)	criar_estado (c)	Estado / Statechart
bolha_normal (o)	verificar_estado_normal (o)	Estado/ Statechart
confirma_bolha (c)	confirmar_estado (c)	Estado / Statechart
desaloca_nos_de_estados (i)		
cria_panel_nro_filhos (o)	receber_nro_filhos (o)	Estado/ Statechart
desaloca_nós_e_transição (oc+)	remover_estado (c) remover_transição_estado(c) remover_ativids_estado (c) consultar_bolha (o)	Estado/Statechart Transição/Statechart E_Atividade/Statechart Bolha/Diagrama
desenha_bolha (c+)	desenhar_estado (c) colocar_infs_gráficas_estado (c)	Bolha / Diagrama Estado / Statechart
identifica_bolha (o+)	obter_infs_estado identificar_bolha	Estado/Statechart Bolha/Diagrama
...	...	...

Pode-se observar, por exemplo, que o procedimento **aloca_bolha** do MASA corresponde a um método de criação de um novo objeto Estado no MAS (método canônico). Alguns procedimentos sofrem apenas alteração (adaptação) de nome como é o caso de **confirma_bolha** (MASA) que passa a ser denominado **confirmar_estado**. O procedimento **desaloca_nós_e_transição**,

com anomalia (oc+) deu origem a quatro métodos associados a diferentes classes. Esse é um exemplo típico dessa anomalia, onde um procedimento engloba todas as atividades envolvidas em uma operação.

Estado
Atributos: nome tipo default nível nível_visual ativação cópia_ativação última_ativação passos_ativos passos_consecutivos num_médio_ativ_consecutivas informações_gráficas informações_gráficas_expandidas
Métodos: criar_estado colocar_infs_gráficas_estado confirmar_estado obter_infs_estado remover_estado verificar_estado_normal ...

Figura 5.16 - Lista de atributos e métodos (parcial) da Classe Estado do MAS

Outro tipo de procedimento encontrado foi aquele típico da programação XView, que somente ativa/chama outros procedimentos sem realizar a função específica que seu nome sugere ou pela ordem em que é chamado. Como exemplo tem-se **desaloca_nos_de_estado**. Recorrendo-se ao código fonte para entendê-lo, observa-se que esse é o procedimento chamado para desalocar os espaços reservados para estado. Ao se obter o trecho de código referente a esse procedimento nota-se que sua função é ativar **desaloca_nós_e_transição** que é o procedimento que efetivamente remove estado, transição e atividades, classificado como **oc+**. Os procedimentos que foram criados devido ao tipo de estrutura de dados alocada para cada um dos objetos e da forma como a implementação foi realizada existem no MASA e não tem correspondente no MAS. O procedimento **Acha_ind_est** é um exemplo daqueles procedimentos que foram criados visando a otimizar a implementação, nesse caso específico, para a localização dos estados ser realizada de modo mais rápido e fácil utiliza-se de

5.4.2. Elaboração do Modelo de Ciclo de Vida

Nova versão foi produzida, apenas com alterações quanto aos nomes dos eventos de entrada e de saída, sem incluir ou eliminar eventos de entrada e saída na seqüenciação de eventos. Fez-se uma adaptação aos nomes das operações listadas no Modelo de Ciclo de Vida do MASA, na Figura 5.8, para o modelo do MAS. Por exemplo, a troca da palavra bolha por estado, uma vez que essa classe estado é a que foi especificada no Modelo de Objetos para o tema **statechart**; já bolha é uma classe de objetos existentes no tema **diagrama**.

Um trecho do Modelo de Ciclo de Vida, com as alterações citadas é o mostrado na Figura 5.18. As operações e os eventos de saída estão agora totalmente em português, o que não impede que a interface do sistema atual continue a ser em inglês, pois esse são nomes internos ao sistema.

```

life cycle STATSIM = (Edit | Simulate)*

Edit = (Editar_Estados | Carregar_Statechart | Editar_Arquivo. Carregar_Statechart).
      (Editar_Arquivos | Editar_Estados | Editar_Transições | Editar_Atividades |
      Editar_Variáveis | Carregar_Statechart | Ocultar_Eventos)* . #statechart_editado

Simulate = Carregar_Statechart. (Simular_Statechart | Decompor_Statechart |
      Informacoes_Componentes | Informações_Variaveis |
      Configuração_Default | tipos_simulação | variedades) . #statechart_simulado

Editar_Estados = Criar_Estado.(Criar_Estado | Remover_Estado | Consultar_Estado |
      Decompor_Estado)*

Criar_Estado = ...criar_estado. # bolha_desenhada. ...

```

Figura 5.18 - Modelo de Ciclo de Vida do MAS

5.4.3 Elaboração do Modelo de Operações

Todas as operações identificadas na seção 3.4 deste Capítulo, foram re-especificadas, de forma a se adequarem aos novos nomes dos eventos de entrada e saída e ao MAS, criando para cada uma um Modelo de Operação. Por exemplo, a operação **criar_estado**, exibida na Figura 5.19, é a correspondente à *cria_bolha* exibida na Figura 5.9. Essa mudança de nome

uma tabela auxiliar que contém o índice dos subestados. Esse tipo de procedimento refere-se à forma de implementação realizada, sendo classificados como (i) e sem método correspondente no MAS.

A Figura 5.16 mostra a classe **Estado** com os seus atributos e alguns de seus métodos. Essa descrição é produzida tendo-se por base as informações constantes das Tabelas 5.3 e 5.4. Análise posterior dessas subclasses podem determinar alguma alteração desses atributos e métodos.

Deve-se utilizar um Dicionário de Dados, preferencialmente automatizado, para complementar as informações relacionadas aos componentes do modelo. A Figura 5.17 mostra um exemplo de Dicionário de Dados.

Nome	Tipo	Descrição
ESTADO	Classe	Representa as informações semânticas de estados na linguagem statecharts.
TRANSIÇÃO	Classe	Representa a ligação entre dois estados e pode ter um evento ou uma expressão de eventos associadas.
Tem Origem	Relacionamento	Relacionamento para representar que uma transição origina-se em um estado.
Referencia	Relacionamento	Relacionamento para representar que uma ação (E-Ação) pode ser formada por variáveis.
tipo	atributo	Um estado pode ser do tipo AND ou OR
cria_estado	operação	Operação que realiza a criação de uma nova classe chamada ESTADO
bolha está dentro dos limites	invariante	A bolha não ultrapassa os limites de seu pai
engenheiro de software	agente	Responsável pelas entradas do sistema

Figura 5.17 - Dicionário de Dados (Parcial)

ocorreu devido à classe bolha pertencer ao tema **diagrama**, ou seja, bolha é a representação gráfica de estado. Muitos dos resultados especificados na operação *cria_bolha* deixam de existir agora pois são referentes à forma de implementação utilizada, porém a maioria deles é conservada. A elaboração desse modelo se dá levando em conta as informações existentes uma vez que a mudança de funcionalidade não ocorre para esse sistema. Se houvesse necessidade de mudança de funcionalidade do sistema, as operações aqui descritas deveriam evidenciar os pontos alterados em relação aos descritos no Modelo de Operações do MASA, alterando consideravelmente as pré e pós condições.

Operação	criar_estado
Descrição:	Cria um estado, desenha uma bolha e armazena todas as informações relacionadas a este estado.
Lê:	supplied ponto_inicial, ponto_final, nome-estado, tipo
Modifica:	new Estado, new Bolha CJ-Estado, Estado
Envia:	Engenheiro de Software: {bolha_desenhada}
Assume:	bolha está dentro dos limites, a bolha não intercepta irmão
Resultado:	<ul style="list-style-type: none"> - foi criada uma new Bolha - o evento bolha_desenhada foi enviado ao Engenheiro de Software - foi criado um new Estado - o estado tem um único nome - o estado tem um único tipo. - o estado está dentro dos limites - se o Estado criado foi o primeiro, então nível recebeu 0 senão nível recebeu initial nível + 1. - ativação recebeu 0 - cópia_ativação recebeu 0. - última_ativação recebeu 0 - se bolha normal, informações_gráficas_normais recebeu as posições gráficas , ou seja, ponto inicial e ponto final - se bolha expandida, informações_gráficas_expandidas recebeu as posições gráficas da bolha expandida, ou seja, ponto inicial e ponto final.

Figura 5.19 - Modelo de Operação: criar_estado

5.5. Mapeamento do MAS para o MASA

Este passo documenta o relacionamento entre o Modelo de Análise do Sistema e o Modelo de Análise do Sistema Atual, exibindo para todas as classes de objetos no Modelo de Objetos do MAS, qual a sua origem do MASA.

A Tabela 5.5 apresenta a correspondência entre o MAS e MASA para a classe **Estado** do Statechart. Os modelos descritos anteriormente são apresentados facilitando ao leitor a associação entre o sistema existente e as abstrações realizadas.

Tabela 5.5 MAS x MASA para Classe Nó_Estado

MAS	MASA
Modelo de Objetos: Classe: Estado:	Nó_Estado
Atributos	Elementos de Dados
nome default tipo nível_visual nível ativação cópia_ativação última_ativação passos_ativos passos_consecutivos número_consecutivo média_passos_consecutivos informações_gráficas_normais informações_gráficas_expandidas col *ESTADO/Decomposto /pai col *ESTADO/Decomposto / filho *E_Atividade/Associada	nome_est defo tipo nível_visual nível lig clig ult_lig num_steps_active num_steps_consec num_consec num_consec_average gf gf_exp *pai *filho *ativ
Métodos	Procedimentos
criar_estado (c) verificar_estado_normal (o) confirmar_estado (c) receber_nro_filhos (o) remover_estado (c) colocar_infs_gráficas_estado (c) obter_infs_estado (o) ...	aloca_bolha (c) bolha_normal (o) confirma_bolha (c) cria_panel_nro_filhos (o) desaloca_nos_e_transição (oc+) desenha_bolha (c+) identifica_bolha (o+) ...

Os métodos **criar_estado** e **remover_estado** são operações descritas no Modelo de Operações. O objeto controlador dessas operações é **Estado**. Embora sejam classificadas como operações elas são especificadas como métodos quando da descrição da classe na fase de projeto do método Fusion. Como aqui está sendo realizada a engenharia reversa do sistema, as operações e o controlador já são identificados. Conforme foi mostrado na Figura 4.6, o Grafo de Interação de Objetos tem a representação do objeto controlador das operações, assim, quando esse grafo for construído na fase do projeto, o objeto **Estado** é que conterà uma seta sem origem com o nome da mensagem sendo o correspondente às operações citadas. Optou-se por construir tabelas mostrando a correspondência existente entre as operações do MAS e do MASA. A Tabela 5.6 exibe essa correspondência para o objeto **Estado** do MASA. Deve-se notar que embora o nome da operação no MAS tenha sido alterado, a implementação dela no sistema atual continua sendo feita pelo procedimento *cria_bolha*.

Tabela 5.6 - Correspondência entre as Operações do MAS e MASA para a Classe Estado

MAS	MASA
criar_estado	cria_bolha
remover_estado	deleta_bolha
expandir_bolha	expande_bolha
informar_sobre_estado	info_bolha
ativar_estados_iniciais	inicializa_ligados
repintar_statechart	repinta_statechart

Na Tabela 5.7 é apresentada a correspondência entre o MAS e o MASA para a classe **Transição** do Statechart. São listados os atributos e os elementos de dados correspondentes obtidos a partir do código fonte, como pode ser visto parcialmente na Figura 5.4, e as operações obtidas dos documentos de Revitalização da Arquitetura do Sistema e do uso do ambiente, conforme descrito no Capítulo 3. Os elementos de dados que não têm atributos correspondentes são

aqueles relativos à forma de implementação realizada, como por exemplo: sentinela e ind_gravação. Os elementos de dados *est_f e *est_d não têm correspondentes na coluna do MAS pois agora são atributos específicos de **Trans_M:N**. No MASA a diferenciação entre uma transição um para um e M para N é realizada com est_f e est_d apontando para a subtransição que faz a ligação com a transição. No MAS isso é realizado através da classe **Trans_M:N** e dos relacionamentos, *Tem_Origem_MN* e *Tem_Destino_MN*, com **Estado**. Os relacionamentos entre **Transição** e **Trans_EV** no MASA, Figura 5.5, são agora representados pelo relacionamento “*executa*” entre **Transição** e **Expressão Executável**, Figura 5.12. Esse relacionamento ocorre entre essas classes pois transições têm eventos, ações e atividades associadas.

Tabela 5.7 - MAS x MASA para Transição

MAS	MASA
Modelo de Objetos Classe: Transição	Transição
Atributos	Elementos de Dados
estado_fonte estado_destino número_disparos número_p/_avaliação controle_time_out história infs_arco	ind_est_fonte ind_est_destino num_disp num_avalia ind_gravação h arco
Métodos	Procedimentos
obter_transição (o) criar_transição (c) remover_transição_estado (c) eliminar_transição (c) armazenar_transição_temporariamente (c) gerar_transição (c) ...	busca_transição (o) cria_transição desaloca_nós_e_transições (c+o) elimina_transição_e_polonesa (c+) Recupera_trans (c+o) Gera_sub_tr (c+) ...

As operações exibidas na Tabela 5.8, coluna MAS, têm como objeto controlador **Transição**. Pode-se notar nessa tabela que as operações do MASA

relacionadas com a criação de sub-transição não aparecem no MAS conforme comentado anteriormente.

Tabela 5.8 - Correspondência entre as Operações do MAS e MASA para a Classe Transição

MAS	MASA
criar_transição-default criar_transição_1p1	cria_arc_def_1p1
criar_transição_M:N	cria_arc_MparaN
informar_sobre_transição	info_arc
remover_transição	remove_arco

Alguns procedimentos e os métodos correspondentes são também listados. Esses procedimentos foram selecionados como nos casos anteriores. Os métodos foram elaborados após a análise do que o sistema deve realizar. Pode-se notar que no modelo atual, MASA, existiam as classes **Trans_Ev** e **Sub_Transição** que deixam de existir no MAS. Essas classes não são mais necessárias uma vez que se referiam à forma como as transições M:N foram implementadas. Muitos dos procedimentos analisados no MASA que eram do tipo **c+o** ou **o+c** e que envolviam as classes de **Transição** e **Sub_Transição** ou **Trans_Ev** se tornam métodos somente da classe **Transição** ou de suas subclasses: **Trans_1:1**, **Trans_M:N** ou **Trans_Default**. O procedimento **Gera_sub_tr**, por exemplo, é classificado como **c+**, sendo construtor de duas classes: **Transição** e **Sub_Transição** (MASA), no MAS ele é construtor da classe **Trans_M:N**.

Do mesmo modo que as classes apresentadas anteriormente, os dados aqui exibidos são os provenientes do código fonte que contém a estrutura de dados para os eventos do statecharts, do documento de revitalização do sistema e de conhecimentos adquiridos com o uso do sistema. A Tabela 5.9 exhibe o MAS e o MASA para eventos do statechart.

A classe **Evento_Atômico** relaciona-se com a classe **E_Evento** que é especialização de **Expressão Executável** através da relação "Avalia",

descrevendo assim que uma expressão de evento é composta de eventos atômicos. Os procedimentos associados à classe *Nó_Evento* são adaptados em sua maioria diretamente para os métodos conforme pode ser visto na Tabela 5.9.

Tabela 5.9 - MAS x MASA para Evento

MAS	MASA
Modelo de Objetos	
Classe <i>Evento_Atômico</i>	<i>Nó_Evento</i>
Atributos	Elementos de Dados
nome_evento valor_evento evento_oculto numeros_gerados número-avaliado avaliação contador_time_out Associada/Expressão_Executável	nome_ev valor_ev hide num_gerados num_avail avail time_out *trans
Métodos	Procedimentos
remover_eventos (c) iniciar_evento (c) armazenar_evento_permanentemente (o) consultar_evento (o) consultar_evento_simulação (o) iniciar_time_out (c) iniciar_evento_simulação (c) salvar_evento (c) zerar_eventos (c) ...	desaloca_eventos (c) Inicia_ev (c) Salva_ev (o) Verifica_Evento (oc) Obten_eventos (o) Inic_time_out (c) Inic_time_out (c) Salva_eve (o+c) Zerar_estatísticas (c+) ...

A Tabela 5.10 exibe como as variáveis de statechart se comportam no MASA e no MAS. Elas se relacionam com **E-Ação**, como mostra a Figura 5.12, através da relação "*Referencia*", isto é, uma ação pode ser composta de variáveis. Os elementos de dados descritos na estrutura de dados têm os correspondentes atributos, a menos de acerto quanto aos seus nomes. O elemento de dados *ind_var* não tem atributo correspondente por ser um dado relativo à forma de implementação utilizada no sistema atual. O procedimento *declara_variáveis* tem o método correspondente **declarar_variáveis**, que é somente construtor da classe **Variável** do MAS, pois sua função é alterar o valor das variáveis declaradas.

Tabela 5.10 MAS x MASA para variáveis

MAS	MASA
Modelo de Objetos Classe :Variável	Vars
Atributos	Elementos de Dados
nome_da_variável tipo_da_variável valor_atual valor_anterior	nome_var tipo_var valor valor_ant
Métodos	Procedimentos
alterar_variáveis (o) atribuir_valor_variavel (c) limpar_variáveis (c) declarar_variaveis (c) iniciar_valores_variáveis (c) verificar_se_e'_variavel (o) ...	altera_variáveis (o) atribui_valor (c) Clear_var (c) declara_variáveis (c) Inic_val_vars (c) is_var (o) ...

A Tabela 5.11 exibe informações do MAS e do MASA para atividades do statechart. Essas atividades foram armazenadas na forma de tabela no MASA, Figura 5.5., **Tab_Ativ**. No MAS, como pode ser visto na Figura 5.12, a classe **Procedimento** está relacionada com **E_Atividade** em um relacionamento cuja cardinalidade é representada por *, mostrando que **E-Atividade** pode executar vários **Procedimentos** e que um **Procedimento** pode ser executado por várias **E_Atividade**.

Pilha_Pol, no MASA, é uma estrutura de dados que implementa todos os tipos de expressões executáveis. Os procedimentos associados a **Pilha_Pol** são agora separados em métodos de **E-Atividade**, **E_Evento** e **E-Ação** de acordo com a operação realizada.

A tabela 5.12 resume os modelos construídos no passo 2 e os modelos correspondentes construídos no passo 3 do método descrito no Capítulo 4. O Modelo de Objetos para o tema Statechart Permanente não foi construído para o MAS, porque não há necessidade dessa preocupação na visão do sistema.

Tabela 5.11 MAS x MASA para atividades

MAS	MASA
Modelo de Objetos Classe: Procedimento	Tab_Ativ
Atributos nome_da_atividade endereço_atividade qtdade_estados_associados atividade	Elementos de Dados nome_ativ endereço_ativ qtd est ativ
Métodos atualizar_arvore (c) avaliar_argumento (o) confirmar_atividade (c) apagar_atividade (c) consultar_atividades descarregar_tabela_estado (o) ...	Procedimentos atualiza_arvore (oc) avalia_argumento (o) confirma_ativ (c) deleta_item (oc) janela_info_at (o+) Descarrega_tab_est (o+c) ...

Tabela 5.12 - MASA x MAS

	MASA	MAS
Modelo de Objetos Temas:		
Diagrama	X	X
Relatórios	X	X
Statechart Temporário	X	X
Statechart Permanente	X	
Modelo de Ciclo de Vida	X	X
Modelo de Operação	X	X

5.6.2. Do Modelo Orientado a Objetos Produzido

O modelo produzido é um modelo de Análise Orientada a Objetos completo, de acordo com o que propõe o método Fusion. A partir desse modelo pode-se projetar e implementar o ambiente baseado em orientação a objetos ou ele pode ser utilizado para realizar manutenções orientada a objetos.

Com a engenharia reversa realizada há possibilidade de uma reengenharia futura do ambiente, reimplementando-o em linguagem orientada a objetos tipo C++. A partir de agora há facilidade para a evolução do ambiente uma vez que, nos documentos produzidos pelo método, estão reunidas todas as informações necessárias.

O Modelo de Análise do Sistema, MAS, produzido pela engenharia reversa antecipa algumas atividades da fase de projeto, já que os métodos estão em sua grande maioria definidos. A fase de projetos pode, entretanto, refiná-los. O mapeamento entre o modelo produzido e o sistema atual permite que manutenções possam ser analisadas no modelo de análise do Fusion e depois concretizadas seguindo a implementação atual.

5.6.3. Do Método de Engenharia Reversa Proposto

No início deste projeto de engenharia reversa havia uma noção dos objetivos a serem atingidos e de como fazê-lo; mas durante o desenvolvimento, o método foi sendo delineado a partir da maior compreensão das atividades envolvidas e da racionalização do trabalho, com a descoberta de regras de relacionamento entre o código e o MASA e entre o MASA e o MAS. Esses "insights" podem agora ser reutilizados em outras situações e o método proposto pode ser adotado em situações análogas. Com isso, o objetivo de ter um método concreto, fundamentado e testado na prática foi atingido.

Embora não utilizada no experimento relatado neste Capítulo, porque não havia disponibilidade, uma ferramenta de apoio ao método Fusion poderia ser incorporada ao método para facilitar o desenvolvimento do passo 3. Idealmente, deveria ser uma ferramenta aberta, para que o mapeamento criado no passo 4 possa ser também alimentado para a ferramenta.

Benedusi e outros [4], comentam que no estágio atual, um processo de Engenharia Reversa deve ser refinado e projetado para cada uso, considerando as necessidades a satisfazer e o ambiente. O método proposto no Capítulo anterior e validado neste Capítulo mostrou constituir um passo adiante nesse estágio pois, sem dúvida, pode ser reusado em muitas outras situações, pois:

- A passagem de um passo para outro é bem definida.
- Em cada passo o processo é guiado por documentos a serem produzidos.
- Há uma articulação clara entre os documentos do sistema atual e os documentos a serem produzidos.

5.7. Considerações Finais

As diretrizes para a realização de engenharia reversa, referidas na seção 4.1, Considerações Iniciais, do Capítulo 4, foram testadas neste Capítulo. Pôde-se observar em detalhes todas as anomalias apresentadas pelo sistema e com o estudo realizado tem-se uma medida do esforço necessário para corrigi-las.

O objetivo de ter um modelo orientado a objetos do ambiente StatSim foi plenamente atingido. Com base na implementação atual, foram derivados os modelos de objeto, de ciclo de vida e de operações, que serão utilizados a partir de agora para evolução do ambiente. O experimento serviu também para reforçar a suposição já testada por alguns autores, de que é possível derivar um modelo orientado a objetos com base em uma implementação não orientada a objetos.

A motivação gerada pelos artigos de Jacobson, Wong, Dietrich e outros consolidou-se com a elaboração do método e cobre uma lacuna por eles deixada quando tratam da reengenharia de sistemas sem fornecer um método concreto para a etapa de engenharia reversa.

Comparando a sensação de falta absoluta de orientação quando dos primeiros contatos com o ambiente com o guia seguro constituído pela documentação produzida pela engenharia reversa, é possível afirmar que o trabalho empregado em executá-la traz benefícios que contribuem para a melhoria de qualidade do sistema atual e para futuras evoluções, orientadas a objeto ou não.

Aplicando-se as métricas apresentadas no Capítulo 4 observa-se que a implementação do ambiente StatSim pode ser convertida para uma implementação baseada em orientação a objetos com relativa facilidade. A influência do tipo de implementação utilizada é grande, uma vez que a anomalia tipo i corresponde a mais de 60% dos tipos de anomalias detectadas.

Capítulo 6

Conclusões e Sugestões para Futuras Pesquisas

6.1. Considerações Iniciais

Esta tese tem como objetivo apresentar um método de engenharia reversa orientado a objetos e a sua aplicação a um sistema-exemplo. Para se chegar ao método proposto alguns métodos de desenvolvimento de sistemas reativos e orientados a objetos são apresentados, comparando-se os pontos positivos e negativos entre eles. Apresenta-se, também, o ambiente StatSim e as suas evoluções, bem como a ampliação realizada como parte deste trabalho. Esse ambiente é utilizado como o sistema exemplo para validação do método de engenharia reversa orientado a objetos proposto. Além disso, mostra-se como o StatSim pode ser utilizado para validação da especificação de um sistema reativo em Statecharts.

As conclusões gerais deste trabalho são retomadas e discutidas na seção 6.2. As contribuições desta tese são apresentadas na seção 6.3 e as sugestões para trabalhos futuros estão na seção 6.4.

6.2. Conclusões Gerais

Como exposto no Capítulo 1 deste trabalho, a dificuldade encontrada em manter e ampliar as funções do ambiente StatSim foi um dos fatores que motivou esta pesquisa. Pode-se citar o entendimento de trechos do

código fonte como uma das dificuldades encontradas quando da ampliação da funcionalidade do ambiente: inclusão de atividades associadas aos estados e transição "default" com ação. A implementação de transição "default" com ação permite que um maior número de sistemas possa ser especificado e o ambiente StatSim possa ser utilizado para editá-los e simulá-los. Antes disso, todo sistema que possuía evento associado a estado "default", precisava ser editado no ambiente utilizando artifícios, uma vez que as ações estavam associadas somente às transições não "defaults". O mesmo ocorria com os sistemas que possuíam tarefas associadas aos estados, não podiam ser editados/simulados no ambiente StatSim pois somente a parte comportamental de um sistema podia ser especificada. Ao se inserir atividades aos estados permitiu-se que um sistema possa ser visualizado tanto do aspecto comportamental como funcional.

Após a pesquisa na literatura técnica especializada, foi notada a ausência de diretrizes seguras que conduzissem à engenharia reversa de sistemas desenvolvidos sem a orientação a objetos para um modelo orientado a objetos. Esta tese preenche esta lacuna, propondo, para esse fim, um método detalhado, sistemático e utilizável em outras situações semelhantes, que são comuns na realidade atual. O método de engenharia reversa orientado a objetos proposto neste trabalho apoia-se nos modelos existentes, para a fase de análise, no método de Fusion. A partir de um sistema implementado sem orientação a objetos é possível através desse método elaborar um modelo orientado a objetos. Vale ressaltar que os modelos orientados a objetos obtidos são abstrações realizadas, uma vez que o sistema implementado não segue a abordagem orientada a objetos. Eles são produzidos visando a facilitar a elaboração do modelo "real" orientado a objetos, que é o produto final após a aplicação do método de engenharia reversa.

A avaliação do método proposto foi efetuada aplicando-o ao próprio ambiente StatSim. Dessa forma, pode-se dizer que o objetivo foi totalmente alcançado, uma vez que esse método pode ser seguido passo a passo e produz os resultados desejados. As anomalias detectadas são provenientes de alguns fatores, tais como: a pouca familiaridade dos programadores com a linguagem C, a falta de diretrizes bem definidas para a realização do projeto e a não preocupação com as expansões que poderiam vir a ser realizadas com o ambiente. Após o levantamento de todas as anomalias encontradas, o modelo

orientado a objetos foi construído considerando-se somente os aspectos relevantes ao ambiente, sem a preocupação da forma como seria a implementação, caso um processo de reengenharia fosse a opção escolhida. O modelo produzido facilita as manutenções que devem ser realizadas no sistema quando necessário.

A afirmação de Coleman de que o método Fusion é adequado para engenharia reversa [17] pôde ser comprovada nesta tese. Uma das principais razões para isso é a existência de três fases distintas, articuladas entre si. Embora o objetivo final seja produzir um modelo de análise, o processo de engenharia reversa parte do código existente e há um caminho natural para seguir o processo inverso de modelagem. Por exemplo: da interface do sistema atual produzir o seu Modelo de Ciclo de Vida e daí produzir um modelo de análise; da mesma forma produzir um Modelo de Operação para cada operação da interface e a partir daí generalizá-lo em função do Modelo de Objetos.

6.3. Contribuições deste Trabalho

O trabalho acrescentou ao ambiente StatSim além de recursos para o tratamento dinâmico do sistema especificado também recursos para o tratamento funcional. A edição de atividades associadas aos estados permite que quando o statechart for simulado, se acompanhe quais são as tarefas realizadas no momento que o estado está ativo. As atividades associadas às transições se comportam como as ações. Dessa forma, o engenheiro de software tem a possibilidade de visualizar um sistema sob esses dois aspectos, comportamental e funcional, possibilitando a validação da especificação.

A redocumentação que foi feita [72, 73] ao ambiente StatSim contribui não só para facilitar o seu uso e manutenção como demonstra a familiaridade da autora com o ambiente enfocado.

A seqüência de passos proposta no Capítulo 3 para que um sistema reativo seja desenvolvido utilizando-se das duas visões, tem por objetivo fornecer diretrizes concretas para facilitar a utilização do ambiente StatSim como um mecanismo de validação de sistemas. Essa seqüência de passos visa dar ao leitor

condições para a especificação de um sistema tanto através de statecharts como através de diagramas de atividades. As simulações realizadas no ambiente StatSim para o sistema exemplo desenvolvido, seguindo as diretrizes expostas, permitiram verificar que a especificação está correta. As seqüências de eventos tomadas como exemplo tiveram o mesmo comportamento quando aplicadas ao exemplo tanto no ambiente StatSim quanto ao protótipo existente. Os tipos de simulações existentes no ambiente StatSim permitem, também, a validação das diferentes seqüências de eventos possíveis.

Este trabalho apresenta diretrizes detalhadas e articuladas na forma de método para engenharia reversa orientada a objetos para sistemas desenvolvidos sem essa orientação. A utilização do método de Fusion para apoio ao método de engenharia reversa criado é original e mostrou-se bastante viável. Na literatura especializada existem diversos trabalhos que citam a possibilidade de que a engenharia reversa seja apoiado por um método de desenvolvimento de sistemas orientados a objetos, porém não são fornecidas as diretrizes para que isso seja realizado, métodos detalhados ou casos elaborados. A escolha do método Fusion se deve ao fato dele ser um método recente para o desenvolvimento de sistemas com orientação a objetos, por cobrir as fases de análise, projeto, implementação e por conter, em seus modelos, o que existe de melhor nos outros métodos existentes de orientação a objetos e que são amplamente utilizados.

A aplicação desse método ao ambiente StatSim passo a passo, produziu uma documentação que pode ser considerada como um manual para a repetição dessa tarefa em relação a outros sistemas. Foi possível também detectar problemas importantes relacionados à implementação atual, que dificultam a manutenção e violam regras de modularização bem fundamentadas. As anomalias detectadas na implementação podem ser consideradas uma métrica de qualidade da implementação atual. As anomalias relacionadas ao tipo de implementação empregada não alteram o estado do sistema, sendo possível a correção de alguns procedimentos, como a longa hierarquia de chamadas. Dentre as demais anomalias encontradas as que consomem maior esforço são as do tipo `o+c+`, pois a manipulação de diversas estruturas de dados deve ser realizada através da elaboração de diversos métodos. O objetivo de ter um modelo orientado a objetos do ambiente StatSim foi plenamente atingido. Os modelos de objetos, de ciclo de

vida e de operações foram derivados a partir da implementação atual e serão a partir de agora utilizados para evolução do ambiente. O experimento serviu também para reforçar a suposição já testada por alguns autores, de que é possível derivar um modelo orientado a objetos com base em uma implementação não orientada a objetos. Dessa forma, a proposta de se ter um método sistemático, bem fundamentado e testado na prática foi concretizada.

São fornecidas métricas para avaliação do esforço de conversão de um sistema desenvolvido sem a orientação a objetos em um com orientação objetos. A elaboração dessas métricas ocorrer a partir das anomalias detectadas no sistema que passa pelo processo de engenharia reversa. Permite-se, dessa forma, que o engenheiro de software tenha condições de avaliar o esforço necessário que será dispendido para que um código orientado a objetos seja criado a partir do sistema implementado sem a abordagem de orientação a objetos.

6.4. Sugestões para Futuras Pesquisas

Existem alguns pontos que merecem atenção quanto à exploração para a obtenção de resultados que possam auxiliar no processo de engenharia reversa e reengenharia de sistemas sob o enfoque da orientação a objetos. Um deles é a aplicação do método de engenharia reversa orientada a objetos proposto neste trabalho a outros tipos de sistemas, como sistemas de informação e sistemas reativos. Considerando-se esses domínios específicos, talvez novos aspectos possam ser incorporados ao método proposto nesta tese.

Sistemas de informação desenvolvidos em linguagens de programação mais antigas e sob paradigmas não orientados a objetos podem ser recuperados, construindo-se um modelo orientado a objetos. Esses sistemas, na maioria das vezes, atendem a todas as necessidades de seus usuários, porém há o desejo de torná-los disponíveis em plataformas mais recentes de software/hardware sem que o desenvolvimento de um novo sistema seja necessário. Outras vezes há o desejo de se alterar somente a linguagem de programação e, em outras ainda, há a necessidade de ampliar a funcionalidade

desses sistemas. Em todos esses casos pode-se aplicar o método de engenharia reversa orientada a objetos aqui proposto.

Os sistemas reativos merecem um cuidado especial. Um trabalho interessante diz respeito à aplicação do método de engenharia reversa orientada a objetos a sistemas reativos quando um dos casos análogos aos citados acima ocorre. Os modelos a serem obtidos podem necessitar de algumas adaptações relacionadas ao método Fusion para que o fator tempo seja expresso e para que as respostas aos eventos sejam dadas de forma condizente com o tipo de sistema.

A ampliação do ambiente StatSim a partir do Modelo de Objetos gerado é uma das atividades que pode ser realizada. Novas funcionalidades podem ser adicionadas ao ambiente, com base no modelo criado por este trabalho, ou seja, nos modelos produzidos para o MAS. A reengenharia parcial do ambiente, ou seja, a alteração do código para uma linguagem orientada a objetos pode ser realizada e com isso obtem-se também a validação das métricas sugeridas no Capítulo 4. No caso de reengenharia do ambiente pode-se utilizar o método Fusion para a fase de engenharia avante. A partir do modelo MAS construído pode-se iniciar a codificação dos métodos relacionados às classes utilizando-se uma linguagem orientada a objetos.

As métricas propostas podem ser validadas por outros projetos de engenharia reversa ou reengenharia e possíveis extensões ou ajustes podem ser feitos a partir de novas pesquisas.

O estudo de ferramentas que possam apoiar o método de engenharia reversa orientada a objetos é um outro ponto a ser investigado. Ferramentas que possam auxiliar a fase de revitalização da arquitetura do sistema existente facilitam o trabalho de recuperação de informações e podem vir a agilizar o processo de engenharia reversa. Outro tipo de ferramenta a ser investigado é o que auxilia no processo de classificação de procedimentos quanto ao tipo de anomalia existente, ou seja, que investiga se um procedimento é construtor ou observador de uma ou mais classes.

As ferramentas CASE, como Fusion Case [17], dão apoio à construção de modelos do MAS. Já para os modelos do MASA, adaptações devem ser realizadas para que os modelos correspondentes ao sistema atual sejam obtidos. A adequação da ferramenta ao método, bem como a especificação dos requisitos dessas adaptações, serão objetos de estudos futuros.

O método de engenharia reversa pode ter outras versões apoiadas em outros métodos de desenvolvimento de sistemas orientado a objetos, tais como, Objectory, OMT, etc.. A partir daí, uma comparação entre essas versões pode ser realizada evidenciando os pontos positivos e negativos de cada um deles.

Referências Bibliográficas

- [1] **Aiken, P.; Muntz, A.; Richards, R.** - DOD Legacy Systems Reverse Engineering Data Requirements. *Communications of the ACM*, v.37, n. 5, p. 26-57, 1994.
- [2] **Allworth, S.T.** - *Introduction to Real-time Software Design*. London, McMillan, 1981.
- [3] **Batista, J. E. S.** - Um Editor Gráfico para Statecharts. São Carlos, 1991. *Dissertação (Mestrado)*, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo, 1001.
- [4] **Benedusi, P.; Cimitile, A.; De Carlini, U.** - Reverse Engineerign Processes, Design Document Production, and Structure Charts. *The Journal of Systems and Software* , n. 19, v. 3, p. 225-245, 1992.
- [5] **Biggerstaff, T.J.; Mitbander, B.G.; Webster, D. E.** - Program Understanding and the Concept Assignment Problem. *Communications of the ACM*, n. 37, v. 5, p. 72-83, 1994.
- [6] **Boaventura, I. A. G.** - Propriedades Dinâmicas de Statecharts. São Carlos, 1992. *Dissertação (Mestrado)*, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo, 1992.
- [7] **Booch, G.** - *Object-Oriented Design with Applications*. Benjamin Cummings, CA, 1991.

-
- [8] **Brady, J. M.** - *The Theory of Computer Science: A Programming Approach*. London, Chapman and Hall, 1977.
- [9] **Cangussu, J.W.L.** - Execução Programada de Statecharts. São Carlos, 1993. *Dissertação (Mestrado)*, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo, 1993.
- [10] **Cangussu, J. W. L. ; Penteado, R. D., Masiero, P.C., Maldonado, J.C.**- Validation of Statecharts Based on Programmed Execution. In: *International Conference On Computer And Information*, 7, 1995, Peterborough. *Anais*. Ontario. Canada, 1995.
- [11] **Chen, P. P.** - The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems* 1, p.9-36, 1976.
- [12] **Chikofsky, J. E., Cross, J. H.** - Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, v. 7, n.1, p. 13-7, 1990.
- [13] **Choi, S. C.; Scacchi, W.** - Extracting and Restructuring the Design of Large Systems. *IEEE Software*, v.7, n.1, p. 66-71, 1990.
- [14] **Coad,P.; Yourdon, E.** - *Object Oriented Analysis*. Prentice-Hall. Englewoof Cliffs, New Jersey, 1990.
- [15] **Coad, P. ; Yourdon, E.** - *Oject Oriented Design*. Prentice-Hall. Englewoof Cliffs, New Jersey, 1991.
- [16] **Coleman, D.; Hayes, F.; Bear, S.** - Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design. *IEEE Transactions on Sofwtare Engineering*, v. 18, n. 1, p. 9-18, 1992.
- [17] **Coleman, D. et al** - *Object-Oriented Development - The Fusion Method*. Prentice Hall, 1994.

-
- [18] **Cross, II; et al** - Reverse Engineering. *Advance Computer*, v. 35, p.199-254, 1992.
- [19] **Davis, A. M.** - A Comparision of Techniques for the Specification of External System Behavior. *Communications of the ACM*, v. 31, n. 9, p. 1098-115, 1988.
- [20] **Dietrich, W.C.; Nackman, L.R.; Gracer, F.** - Saving a Legacy with Objects. In: *OOPSLA, 89*. New Orleans, 1989. *Procedings*. Yorktown, ACM, 1989. p.77-83, 1989.
- [21] **Duntemann, J.; Marinacci, C.** - New Objects for Old Structures. *Byte*, v. 15, n. 4, p.261-6, 1990.
- [22] **Fortes, R. M. P.** - Uma Ferramenta de Apoio à Utilização de Statecharts para Especificação do Comportamento de Sistemas de Tempo-Real Complexos. São Carlos, 1991. *Dissertação (Mestrado)*, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo.
- [23] **Fuuruti, R.; Masiero, P.C.** - *Extensões ao Ambiente StatSim para Permitir a Simulação de Micro-Passos*. São Carlos, 1992. (Relatório CNPq, 1, ICMSC - USP), 1992.
- [24] **Fuuruti, R., Masiero, P.C.** - *Extensões ao Ambiente StatSim para Permitir a Simulação de Micro-Passos*. São Carlos.. (Relatório CNPq, 2, ICMSC - USP) , 1992.
- [25] **Gall, H.; Klösch, R.** - Finding Objects in Procedural Programs: An Alternative Approach. . In: *Working Conference on Reverse Engineering, 2*. Toronto, Ontario, Canada, 1995. *Anais*. IEEE, p.208 - 216, 1995.

-
- [26] **Gane,C.; Sarson,T.** - *Structured Systems Analysis: Tools and Techniques*. New Jersey, Prentice-Hall, 1979.
- [27] **Gomaa, H.** - A Software Design Method for Real-Time Systems. *Communications of ACM* , v. 27, n. 9, p.938-49, 1984.
- [29] **Germano, F.; Masiero, P.C.; Penteado, R.D** - Análise Comparativa dos Métodos SA/SD, HIPO, PSL/PSA, SOP, Systematics e PDL. In: *Encontro de Metodologias de Sistemas*. 1983, Belo Horizonte. Anais. Belo Horizonte, SUCESSU-MG, 1983.
- [30] **Guedes, L.C.; Staa, A.** -. Um Processo de Re-engenharia Econômico e Eficaz. In *Simpósio de Engenharia de Software*, 7, Rio de Janeiro-RJ.. Anais. p. 77 - 91, 1993.
- [31] **Hamilton, M. ; Zeldim, S.** - Higher Order Software - a methodology for defining software. *IEEE Transactions on Software Engineering SE* v. 2, n. 1, p.9-32, 1976.
- [32] **Harel, D.** - STATECHARTS: A Visual Formalism fo Complex Systems. *Science of Computer Programming*, v. 8, p. 231-74, 1987.
- [33] **Harel,D. et al** - On the Formal Semantics of Statecharts, In: *IEEE Symposium on Logic in Computer Science*, 2, Ithaca, 1987. *Proceedings*, New York, p. 54-64, 1987.
- [34] **Harel, D.** - On Visual Formalisms. *Communications of the ACM*, v. 31 n. 5, p. 514-30, 1988.
- [35] **Harel, D.** - STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, v. 16, n. 4, p. 403-14, 1990.

-
- [36] **Harel, D.; Kahana, C-A.** - On Statecharts with Overlapping. *ACM Transactions on Software Engineering and Methodology*, v.1 n. 4, pp. 399-421, 1992.
- [37] **Hatley, D. J.; Pirbhai, I. A.** - *Estratégias para Especificação de Sistemas em Tempo Real*. Makron Books do Brasil Editora Ltda, São Paulo, 1990.
- [38] **IBM Study Organization Plan - Documentation Techniques.** IBM Corporation, 1963. (C20-8075-0).
- [39] **IBM Study Organization Plan -The Approach.** IBM Corporation, 1963 . (C20-8075-0)
- [40] **IBM Corporation, Data Processing Division - HIPO: A Design Aid and Documentation Technique.** White Plains, New York, 1974.
- [41] **Iivari, J.** - Object-orientation as Structural, Functional and Behavioural Modelling: a Comparison of Six Methods for Object-Oriented Analysis. *Information and Software Technology* , v. 37, n. 3, p.155-163, 1995.
- [42] **The Language Statemate.** Burlington, i-Logix Inc. , 1987.
- [43] **Jackson, M.** - *Principles of Programming Design*, Academic Press, 1975.
- [44] **Jackson, M.** - *System Development*. Prentice Hall International, Englewood Cliffs, 1983.
- [45] **Jacobson, I.; Lindström, F.** - Re-engineering of Old Systems to an Object-Oriented Architecture. In: *OOPSLA, 91, 1991. Proceedings.* ACM, 1991, p. 340-350,.

- [46] **Jacobson, I.** - *Object-Oriented Software Engineering*. Addison-Wesley, Reading, MA, 1992.
- [47] **Jacobson, I. ; Ericsson, M. , Jacobson, A.** - *The Object Advantage - Business Process Reengineering with Object Technology*. Addison Wesley, 1995.
- [48] **Lee, P. J.; Chen, D. J., Chung, C. G.** - An Object-Oriented Modelling Approach to System Software Design. *Information and Software Technology* v. 36, n. 11, p.683-694, 1994.
- [49] **Leite, J.C.S.P.; Franco, A.P.M.** - Reengenharia baseada em JSD. In: *Simpósio de Engenharia de Software, 5. Anais*. Ouro Preto, UFMG. p. 177 -192, 1991.
- [50] **Leite, J.C.S.P.; Prado, A.F.; Sant'Anna, M.** - DRACO-PUC, Experiências e Resultados de Re-Engenharia de Software. In: *Simpósio de Engenharia de Software, 6, Gramado - RS. Anais*. p. 115 -128, 1992.
- [52] **Leiva, W. D.; Penteado, R. D.** - Desenvolvimento do Sistema de Indexação de Normas Internas da Universidade Federal de São Carlos. *Relatório Bolsa Trabalho*, Universidade Federal de São Carlos, São Carlos, 1991.
- [53] **Lira, L. O.; Prieto, G. A; et al** - Sistema de Equivalência de Disciplinas. *Relatório Bolsa Trabalho*, Universidade Federal de São Carlos, São Carlos, 1991.
- [54] **Lira, L. O.; Prieto, G. A. ; et al** - Geração de Questionário Para Demanda das Disciplinas. *Relatório Bolsa Trabalho*, Universidade Federal de São Carlos, São Carlos, 1991.

-
- [55] **Losavio, F.; Matteo, A., Schlienger, F.** - Object-oriented Methodologies of Coad and Yourdon and Booch: Comparison of Graphical Notations. *Information and Software Technology*, v. 36 n. 8, p. 503 - 14, 1994.
- [56] **Masiero, P.C.; Fortes, R.P.M.; Batista, J.E.S.** - Edição e Simulação de Aspectos Comportamentais de Sistemas de Tempo Real. In: *Seminário Integrado de Software e Hardware*, 18, Santos, 1991.. Anais. Santos, SBC, p. 45-61, 1991.
- [57] **Masiero, P.C.; Maldonado, J.C; Boaventura, I. A. G.** - A Reachability Tree for Statecharts and Analysis of Some Properties. *Information and Software Technology*, v. 36, n. 10, p. 615-24, 1994.
- [58] **McCabe, T.J.; Williamson, E. S.** - Tips On Reengineering Redundant Software. *Datamation* , v.38, n. 9, p.71-4. April-15, 1992.
- [59] **McMenamin, S. M.; Palmer, J. F.** - *Análise Essencial de Sistemas*. Makron Books Editora Ltda, 1991.
- [60] **Ning, J. Q.; Engberts, A.; Kozaczynski, W.** - Automated Support for Legacy Code Understanding. *Communications of the ACM*, v. 37, n. 5, p. 50-7, 1994.
- [61] **Orr , K. T.** - *Structured Systems Development*. Yourdon Press, New York, 1977.
- [62] **Parnas, D. L.** - On the Criteria to be used in decomposing systems into modules. *Communications of ACM* v. 15, n. 12, p. 1053 -58, 1972.
- [63] **Paulo, F. B.; Penteado, R. D.** - Incorporando a Edição de Atividades no Ambiente StatSim. In: *Congresso de Iniciação Científica da Universidade Federal de São Carlos*, 2. São Carlos, 1994. Anais, p. 22. São Carlos, UFSCar, 1994.

- [64] **Paulo, F.B.; Penteado, R.D.** - Editando Atividades no Ambiente StatSim. In: *Congresso de Iniciação Científica da Universidade Federal de São Carlos*, 3. São Carlos, 1995. Anais. São Carlos, UFSCar, 1995.
- [65] **Penteado, R. D.; Moriya, E.; Masiero, P.C.** - Alguns Métodos para Análise e Projeto de Sistemas de Informação vistos sob o Ângulo do Método HIPO. In: *Encontro de Metodologias de Sistemas*, Belo Horizonte, 1983. Anais. Belo Horizonte, SUCESU-MG, 1983.
- [66] **Penteado, R. D.; Moriya, E.; Costa, G.** - Alguns Métodos para Análise e Projeto de Sistemas de Informação vistos sob o Ângulo do Método SOP. In: *Encontro de Metodologias de Sistemas*, Belo Horizonte, 1983. Anais. Belo Horizonte, SUCESU-MG, 1983.
- [67] **Penteado, R. D.; Moriya, E.; Masiero, P.C.** - Alguns Métodos para Análise e Projeto de Sistemas de Informação vistos sob o Ângulo do Método Análise Estruturada. In: *Encontro de Metodologias de Sistemas*, Belo Horizonte, 1983. Anais. Belo Horizonte, SUCESU-MG, 1983.
- [68] **Penteado, R. D.; Fabbri, S. Chiossi, T.** - Alguns Métodos para Análise e Projeto de Sistemas de Informação vistos sob o Ângulo do Sistema PSL/PSA. In: *Encontro de Metodologias de Sistemas*, Belo Horizonte, 1983. Anais. Belo Horizonte, SUCESU-MG, 1983.
- [69] **Penteado, R. D.** - Sobre uma Abordagem Operacional para Desenvolvimento de Sistemas de Informação. São Carlos, 1984. *Dissertação (Mestrado)*, Instituto de Ciências Matemáticas de São Carlos, Universidade de São Paulo, 1984.
- [70] **Penteado, R.D.; Masiero, P.C.; Germano, F.** - Structure Charts for Jackson System Development. *Software Engineering Notes*, v. 10 n. 3, p. 40-6, 1985.

-
- [71] **Penteado, R.D.; Faria, L.; Germano, F.** - Uma Descrição Elaborada com Apoio de Computador do Método de Jackson para Projeto de Sistemas. In: *Congresso Nacional de Informática*, 18, São Paulo, 1985. Anais. São Paulo, Sucesu, 1985. v. 2, p. 912-26.
- [72] **Penteado, R. D.; Masiero, P.C.; Germano, F.** - *Dicionário de Dados Correspondente à Interação Processos/Dados do Ambiente StatSim*. (Documento de Trabalho), 1994.
- [73] **Penteado, R. D.; Germano, F. Masiero, P. C** - *Manual de Uso do Sistema StatSim*. São Carlos, ICMSC-USP, 1995. (Relatório Técnico, n. 32). Instituto de Ciências Matemáticas de São Carlos, USP, 1995.
- [74] **Penteado, R. D.; Germano, F. Masiero, P. C.** - Engenharia Reversa Orientada a objetos do Ambiente Statsim: método utilizado e resultados obtidos. In: *Simpósio Brasileiro de Engenharia de Software*, 9 Recife. Anais, 1995. Recife, UFPe, p. 345-360, 1995.
- [75] **Penteado, R. D.; Germano, F. Masiero, P. C.** - *Documentação da Engenharia Reversa do Ambiente Statsim - Versão 3*. São Carlos, 1995. (Documento de Trabalho).
- [76] **Penteado, R. D.; et al** - *Listagem do Código Fonte dos Módulos do Ambiente StatSim*. São Carlos, 1995. (Documento de Trabalho).
- [77] **Peterson, J.** - Petri-nets. *ACM Computing Surveys*, v. 9, n. 3, p.223-52, 1977.
- [78] **Portes, A. C. R. B; et al** - *Sistema de Acompanhamento de Aluno*. São Carlos, 1992. (Relatório de Bolsa Trabalho - UFSCar).
- [79] **Pressman, R.** - *Engenharia de Software*. São Paulo, Makron Books, 1995.

-
- [80] **Quilici, A.; Chin, D.** - DECODE: A Cooperative Environment for Reverse-Engineering Legacy Software. In *Working Conference on Reverse Engineering*, 2. Toronto, Ontario, Canada, 1995. *Anais. IEEE*. p. 156-165, 1995.
- [81] **Rumbaugh, J. et al** - *Object-Oriented Modeling and Design*. Englewood Cliffs, Prentice Hall International, 1991.
- [82] **Sage, A. P.** - Systems Engineering and Systems Management for Reengineering. *The Journal of Systems and Software*, v. 30, n. 1/2, p. 3-25, 1995.
- [83] **Santos, M. C.; Penteado, R.D.** - Estudando e Comparando Métodos Para Desenvolvimento de Sistemas de Tempo Real. In: *Congresso de Iniciação Científica da Universidade Federal de São Carlos*, 1. São Carlos, 1993. *Anais. São Carlos, UFSCar*, 1993.
- [84] **Scandura, J. M.** - Cognitive Approach to Systems Engineering and Re-Engineering: Integrating New Designs with Old Systems. *Software Maintenance Research Practice*, v. 2, p. 145-56, 1990.
- [85] **Semimel, R.D.; Winkler, R.P.** - Integrating Reengineered Databases to Support Data Fusion. *The Journal of Systems and Software*, v. 30, n. 1/2, p. 127-35, 1995.
- [86] **Simpson, H.R.; Jackson, K.L.** - Process Synchronisation in MASCOT. *The Computer Journal*, v. 22, n.4, p.332-45, 1979.
- [87] **Sittenaver, C.; Olsem, M. Murdock, D.** - *Re-Engineering Rools Report*. (Technical Report), Software Tecnology Support Center (STSC), Hill Air Force Base, Wtah, 1992.

-
- [88] **Smith, S. L.; Gerhart, S. L.** - *Statemate and Cruise Control: A Case Study*. MCC Technical Report, n. STP-275-88, Austin, Texas. (MCC Software Technology Program).1988.
- [89] **Sneed, H. M.** - Planning the Reengineering of Legacy Systems. *IEEE Software*, v.12, n. 1, p. 24-34, 1995.
- [90] **Sneed, H.; Nyáry, E.** - Extracting Object-Oriented Specification from Procedurally Oriented Programs. . In *Working Conference on Reverse Engineering*, 2. Toronto, Ontario, Canada,. *Anais. IEEE*. p. 217-226, 1995.
- [91] **Teichroew, D.; Hershey, E. A.** - PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems. *IEEE Transactions on Software Engineering*, v. 3, n. 1, p. 41-8, 1977.
- [92] **Tutumi, R.; Masiero, P.C.** - Simulação Visual de Statecharts. In: *Congresso de Iniciação Científica e Tecnológica em Engenharia*, 10, São Carlos. 1991. *Anais. São Carlos*, p.669, USP, 991.
- [93] **Tutumi, R.; Yshiy, S.; Masiero, P.C.** - *Implementação de um Editor Gráfico para Diagrama de Atividades*. São Carlos, 1992, (Relatório de IC- CNPq, ICMSC - USP), 1992.
- [94] **Ward, P.T.** - The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timming. *IEEE Transactions on Software Engineering*, v. SE-12, n. 2, 1986.
- [95] **Wirfs-Brock ; Wilkerson, V.; Wiener, L.** - *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, 1990.

-
- [96] **Wilkening, D. E.; Kreutzfeld, R. J.; Loyall, J. P.** - *Avionics Software Re-Engineering Technology (ASRET.. Software Re-Engineering Study Report, (Technical Report TR-6661), TASC, Reading, Massachusetts, 1993.*
- [97] **Wilkening, D. E.; et al** - A Reuse Approach to Software Reengineering. *The Journal of Systems and Software* , v. 30, n. 1-2, p. 117-25, 1995.
- [98] **Wong, K.; et al** - Structural Redocumentation: A Case Study. *IEEE Software*, v.12, n. 1, p. 46 - 54, 1995.
- [99] **Xinotech Research, Inc. ;** *The Design of the Xinotech Language Translator - Jovial to Ada*, second revised edition. (Technical Report XRI 8911-04), Xinotech Research, Inc.. Minneapolis, Minnesota, 1989.
- [100] **Yourdon, E.; Constantine, L.** - *Structured Design*. 2nd ed. Yourdon Press, New York, 1978.
- [101] **Zoufaly, P.; et al.** (1995). RESCUE: Legacy Systems Translator. . In *Working Conference on Reverse Engineering*, 2. Toronto, Ontario, Canada, 1995. *Anais. IEEE*, 1995. p. 39 - 50.

Gramática Utilizada no Ambiente StatSim

A1.1 Para definição de statechart

A1.2 Para definição de estado e atividades

<descrição estado> ::= ESTADO < nome-estado> <tipo> [<atividades>] ()**

<atividades> ::= ATIVIDADE <função> ()**

<função> ::= DO <chamada-atividade> {'<chamada-atividade>} ()**

| ENTRY <chamada-atividade> {'<chamada-atividade>} ()**

| EXIT <chamada-atividade> {'<chamada-atividade>} ()**

<chamada-atividade> ::= <identificador> '(' <lista-parametro> ')' ()**

<lista-parametro> ::= <identificador> {'<identificador>}

<tipo> ::= OU <subestado ou> | E <subestado e> | ATOMO

<subestados ou> ::= SUBESTADOS <nome-estado> {'<nome-estado>}

A1.3 Para definição de transições

<transições> ::= <transição> { <transição>}

<transição> ::= EVENTO <evento>

ORIGEM <estado-origem> {'<estado-origem> }

DESTINO <estado-destino> [(/H | /H*)]

{'<estado-destino> [(/H | /H*)] }

[AÇÃO {'<ações> '}]

<evento> ::= <ev_ou> { or <ev_ou> }

<ev_ou> ::= <ev_e> { and <ev_e> }

<ev_e> ::= <ev_f> ['[' <exp> ']']

<ev_f> ::= <evento primitivo> | ('<evento primitivo>') | <true> '(' <exp> ')'

| <false> '(' <exp> ')' | <changed> '(' <variavel> ')'

| <exit> '(' <nome-estado> ')' | <entered> '(' <nome-estado> ')'

| t_out '(' <evento> <número> ')'

$\langle \text{exp} \rangle ::= \langle \text{expS} \rangle [(= | > | < | < > | < = | > = | \langle \text{expS} \rangle]$

$\langle \text{expS} \rangle ::= [+ | -] \langle \text{termo} \rangle \{ (+ | - | \text{or}) \langle \text{termo} \rangle \}$

$\langle \text{termo} \rangle ::= \langle \text{fator} \rangle \{ (* | / | \text{and} \langle \text{fator} \rangle \}$

$\langle \text{fator} \rangle ::= \langle \text{true} \rangle | \langle \text{false} \rangle | \langle \text{condição primitiva} \rangle | \langle \text{numero} \rangle$
 $| \langle \text{variavel} \rangle | \text{in ' (} \langle \text{nome-estado} \rangle \text{ ') } | \langle \text{not-yet} \rangle \text{ ' (} \langle \text{evento} \rangle \text{ ') }$
 $| \langle \text{current} \rangle \text{ ' (} \langle \text{exp} \rangle \text{ ') } | \text{not } \langle \text{exp} \rangle | \text{ ' (} \langle \text{exp} \rangle \text{ ' }$

$\langle \text{ações} \rangle ::= \langle \text{ação} \rangle \{ \text{' ; ' } \langle \text{ação} \rangle \}$

$\langle \text{ação} \rangle ::= \langle \text{identificador} \rangle | \langle \text{variavel} \rangle \text{' := ' } \langle \text{exp} \rangle$
 $| \text{clear ' (} \langle \text{nome-estado} \rangle [*] \text{ ') } | \langle \text{chamada-atividade} \rangle \quad (**)$

A1.4 Definições e abreviações

$\langle \text{estado-origem} \rangle ::= \langle \text{nome-estado} \rangle$

$\langle \text{estado-destino} \rangle ::= \langle \text{nome-estado} \rangle$

$\langle \text{nome-estado} \rangle ::= \langle \text{identificador} \rangle [\text{' . ' } \langle \text{identificador} \rangle]$

$\langle \text{evento primitivo} \rangle ::= \langle \text{identificador} \rangle | \text{lambda} \quad (**)$

$\langle \text{condição primitiva} \rangle ::= \langle \text{identificador} \rangle$

$\langle \text{variável} \rangle ::= \langle \text{identificador} \rangle$

$\langle \text{true} \rangle ::= ([tT] [rR] [uU] [eE] | \text{tr})$

$\langle \text{false} \rangle ::= ([fF] [aA] [lL] [sS] [eE] | \text{fs})$

$\langle \text{changed} \rangle ::= (\text{change} | \text{ch})$

<exit> ::= (exit | ex)

<entered> ::= (entered | en)

<not_yet> ::= (not_yet | ny)

<current> ::= (current | cr)

<identificador> ::= [a-zA-Z] [a-zA-Z0-9]*

<numero> ::= [0-9]+

<unid_tempo> ::= (seg | min | hr)

<lambda> ::= (lambda | lamb)

Apêndice 2

Modelos de Objetos do MASA Para os Temas Considerados

Para cada um dos temas considerados foi elaborado um Modelo de Objetos para o Modelo de Análise do Sistema Existente. Esses modelos foram construídos a partir da consulta às estruturas de dados atualmente implementadas. As estruturas foram implementadas utilizando-se os conceitos de tipos abstratos de dados (TAD).

Figura A21 - Modelo de Objetos Para o Tema Diagrama (MASA)

A Figura A21 mostra o Modelo de Objetos para o Tema Diagrama. São consideradas todas as estruturas que representam os objetos relativos à inserção de um diagrama na tela. Os termos *Bolha* e *Arco* referem-se, respectivamente, aos estados e transições do statechart. A Figura A22 mostra o Modelo de Objetos para o Tema Diagrama com o refinamento de *Bolha* e *Arco*, exibindo todos os conectores utilizados para o desenho de um statechart, como

por exemplo, conector cheio (utilizado para o traçado de transição "default"), conector vazio (traçado de transição M:N), segmento de reta e de curva (utilizado para o traçado do estado), etc..

Figura A22 - Modelo de Objetos do Tema Diagrama (MASA) com Refinamento de Bolha e Arco

A Figura A23 mostra o Modelo de Objetos para o Tema Relatórios. Esse tema refere-se ao produto que pode ser obtido depois da simulação de um statechart. São representados, nesse diagrama, os diversos tipos de relatórios produzidos quando da simulação programada, a tabela de eventos que é apresentada quando a simulação interativa é a opção escolhida, etc..

Figura A23 - Modelo de Objetos para o Tema Relatórios (MASA)

A Figura A24 mostra o Modelo de Objetos para o Tema Statechart Permanente. Esse tema refere-se à forma como os statecharts depois de editados são armazenados em dispositivos não voláteis. Os nomes de classes seguem a regra sintática *<nome>.extensão* onde: *nome* representa o nome atribuído ao statechart e *extensão* refere-se ao tipo de dado armazenado: *est* para estado, *tr* para transição, *ativ* para atividades, etc..

Figura A24 - Modelo de Objetos para o Tema Statechart Permanente

Apêndice 3

Modelo de Objetos do MAS Para os Temas Considerados

Para cada um dos temas do Modelo de Análise do Sistema (MAS) foi construído o respectivo Modelo de Objetos. As informações utilizadas nessa construção são obtidas a partir da abstração realizada do Modelo de Objetos do MASA. São consideradas as informações necessárias para a modelagem das classes de acordo com os requisitos do sistema. Os detalhes relacionados com a implementação não são levados em conta.

A Figura A31 mostra o Modelo de Objetos para o Tema Diagrama. As classes consideradas são as relacionadas com as informações de entrada necessárias para a representação das transições e estados. As classes *Arco* e *Bolha* continuam sendo as representantes de transições e estados, respectivamente. A Figura A32 mostra o Modelo de Objetos para o Tema Diagrama com o refinamento de *Arco* e *Bolha*: segmento de reta, rótulo, seta, etc.. O relacionamento "pai representa o fato de estado ter ou não história e a hierarquia entre os estados.

Figura A31 - Modelo de Objetos para o Tema Diagrama (MAS)

Refinamento de Bolha e Arco

A Figura A33 mostra o Modelo de Objetos para o Tema Relatórios. São considerados nesse tema todas as informações relevantes sobre os tipos de simulação existentes no ambiente, relatórios por elas produzidos, Log e as intervenções do engenheiro de software.

Figura A33 - Modelo de Objetos para o Tema Relatórios (MAS)

Apêndice 4

Opções Existentes na Interface do Ambiente StatSim e os Procedimentos que as Implementam

As tabelas a seguir contém as opções de interface do ambiente StatSim e os procedimentos que as implementam. Essas tabelas complementam a Tabela 5.2., apresentada no Capítulo 5,

A Tabela A41 mostra as opções de interface para o modo de edição.

A Tabela A42 e A43 mostram as opções de interface para o modo de simulação.

Tabela A41 - Opções da Interface para o Modo de Edição (Files, Activities, Variables, Hide/Closure)

OPÇÕES DA INTERFACE (Edição)	OPERAÇÕES	OPERAÇÕES IMPLEMENTADAS
Files	Save Statchart	salva_arquivo indica_oper_corrente limpa_operacao mensagem_arq salva_statechart
	Load Statechart	carrega_arquivo arquivo_novo limpa_operacao mensagem_arq mostra_statechart recupera_statechart
	Save as new Statechart	salva_como_novo_arquivo abandona indica_oper_corrente salva_novo_arq
	Clear	limpa_edicao apaga_statechart_memoria aterra_ev coloca_na_pilha_janela indica_oper_corrente Inic_simb_espec Inic_tab_vars limpa_operacao
	Return	sai_Edicao apaga_statechart_memoria aterra_ev coloca_na_pilha_janela Inic_simb_espec Inici_tab_vars limpa_operacao
Activities	Insert	insere_atividades busca_no indica_oper_corrente
	Remove	remove_atividades busca_no indica_oper_corrente
Variables	Declaration	solicita_variaveis analisa_e_armazena_var declara_variaveis
	Query	lista_variaveis abandona_listagem imprime_variaveis indica_oper_corrente
Hide/Closure	Hide/Closure	Lista_hideclosure_e Abandona_hide_e Confirma_hide_e

**Tabela A42 - Opções da Interface - Modo de Simulação (Files,
Decompose, Informations)**

OPÇÕES DA INTERFACE (Simulação)	OPERAÇÕES	OPERAÇÕES IMPLEMENTADAS
Files	Load	carrega_arquivo arquivo_novo limpa_operação mensagem_arq mostra_statechart recupera_statechart
	Save overwrite	salva_arquivo indica_oper_corrente limpa_operação mensagem_arq salva_statechart
	Save as new statechart	salva_como_novo_arquivo abandona indica_oper_corrente salva_novo_arq
	Clear	limpa_edição apaga_statechart_memoria aterra_ev coloca_na_pilha_janela indica_oper_corrente Inic_simb_espec Inic_tab_vars limpa_operação
	Return	sai_Simulação apaga_statechart_memoria aterra_ev coloca_na_pilha_janela Inic_simb_espec Inic_tab_vars limpa_operação
Decompose	Decompose	expande_bolha captura_bolha indica_oper_corrente
Information	Transition	info_arc identifica_arco indica_oper_corrente
	State	info_bolha identifica_bolha indica_oper_corrente
	Activation	info_ativ identif_bolha

Tabela A43 - Opções da Interface -Modo de Simulação (Variables, Default Configuration, Simulation, Miscellaneous).

OPÇÕES DA INTERFACE (continuação Simulação)	OPERAÇÕES	OPERAÇÕES IMPLEMENTADAS
Variables	Query	lista_variaveis abandona_listagem imprime_variaveis indica_oper_corrente
	Change	altera_variaveis recupera_valor
Default Configuration	Default Configuration	inicializa_ligados default_gc Inic_ligados_defo compara1 refresh
Simulation	Batch	
	Create	Batch
	Load	Carrega_arq_lote
	Simulate	Lista_lote abandona_lote imprime_lote indica_oper_corrente
	List	
	Programed	Execução Programada Abandona_Simulation Clear_Var coleta_configuração coleta_nome_fonte coleta_number_stpes coleta_result coleta_type Edite_LCE Execução_Programada2 Show_Control_Var Show_Events Show_Steps_Report Show_Statistical_Report Show_Var Start_simulation
Miscellaneous	Refresh	repinta_stchart default_gc refresh
	Events	Eventos_S
	Show Log	Ver_Log_S
	Clear History	Limpa_historia
	Hide/Closure	Lista_hideclosure_s Abandona_hide_s Confirma_hide_s