**GERAÇÃO DE CÓDIGO
A PARTIR DE STATECHARTS**

Rosemeire Shibuya

Orientação: Prof. Dr. Paulo Cesar Masiero

*Dissertação apresentada ao Instituto de Ciências Matemáticas e de
Computação - USP, como parte dos requisitos para a obtenção do título
de Mestre em Ciências - Área de Ciências de Computação e Matemática
Computacional.*

USP - São Carlos
Maio de 1998

AGRADECIMENTOS

Ao Professor Dr. Paulo Cesar Masiero pela orientação, atenção, incentivo e dedicação durante o desenvolvimento deste trabalho.

Aos meus pais pelo apoio, incentivo e carinho.

Aos professores, amigos e funcionários do ICMC-SC, que direta ou indiretamente colaboraram para o desenvolvimento deste trabalho, em especial a Laura e Marília da seção de pós-graduação, pela atenção e dedicação.

Ao Sadao e Maristela, pelo apoio e amizade.

Ao Carlos pelo carinho, amizade e companheirismo.

RESUMO

A proposta deste trabalho é o desenvolvimento de um Gerador de Códigos baseado em Statechart integrado ao ambiente gráfico StatSim, desenvolvido pelo grupo de Engenharia de Software do ICMC-USP. Para isso, estudos sobre ferramentas e ambientes de sistemas baseados em Statecharts foram realizados. Durante o desenvolvimento do projeto, a partir da análise dos métodos existentes, foram desenvolvidos dois protótipos de código a serem gerado pela ferramenta pretendida, com o intuito de avaliar o algoritmo a ser empregado. Um protótipo foi implementado em Occam e o outro em C++.

Paralelamente foi desenvolvida uma ferramenta gráfica denominada HySChart (Hyperdocument System Based on stateCharts) para apoio ao modelo HMBS (Hypertext Model Based on Statechart), e validação de hiperdocumentos. Com base nesse ambiente, foi desenvolvido um gerador de código que transforma o hiperdocumento criado pelo HySChart em um hiperdocumento em HTML, que pode ser navegado com auxílio de um navegador HTML qualquer. Duas implementações foram feitas para esse gerador: uma que preserva a semântica de navegação do HMBS e outra que não preserva.

ABSTRACT

The proposal of this dissertation is the development of a Code Generator based on Statechart and integrated to the tool StatSim, developed by the Software Engineering group of the ICMC-USP with this aim, tools and environments that support statecharts were reviewed. During the development this project, starting from the analysis of the existent methods, two prototypes of code to be generated by the intended tool, aiming at evaluating the algorithm to be used. The two prototypes were implemented one in Occam and the another in C++.

Simultaneously a denominated graphic tool called HySCharts (Hyperdocument System Based on stateCharts) was developed to HMBS (Hypertext Model Based on Statechart). Based on this environment, a a code generator was developed that transforms the hiperdocument created with HySCharts in a HTML document, that can be browsed by any HTML browser. Two implementations were done: one that preserves the browsing semantics of HMBS and another that doesn't preserve.

Conteúdo

1. INTRODUÇÃO	1
1.1. Considerações Iniciais	1
1.2. Motivação e Objetivos do Trabalho	2
1.3. Organização do Trabalho.....	3
2. INTRODUÇÃO AOS STATECHARTS, AO HMBS, E ÀS FERRAMENTAS DE APOIO DESENVOLVIDAS NO ICMC.....	5
2.1. Cosiderações Inicias	5
2.2. Statecharts.....	5
2.2.1. Diagrama de Estados	5
2.2.2. Hierarquia e Ortogonalidade.....	6
2.2.3. Comunicação	8
2.2.4. História.....	8
2.3. Árvore de Alcançabilidade para Statecharts.....	9
2.4. StatSim.....	11
2.5. HMBS	13
2.5.1. Definição Formal	14
2.5.2. Definição Informal.....	15
2.5.3. Obejtos Estruturais.....	15
2.5.4. Objetos Navegacionais	15
2.5.5. Objetos de Apresentação	16
2.5.6. Características do HMBS.....	16
2.6. HySCharts	17
2.7. Considerações Finais	20
3. GERAÇÃO DE CÓDIGO PARA SISTEMAS REATIVOS ESPECIFICADOS COM BASE EM STATECHARTS	21
3.1. Considerações Iniciais	21
3.2. Abordagens para Métodos e Projetos de Sistemas Reativos	21
3.2.1. - DARTS	22

3.2.2. Esquema de Transformação	25
3.2.3. STATEMATE.....	28
3.2.4. ESTEREL	30
3.2.5. STATEXT.....	35
3.3. Algoritmos para Geração de Código.....	40
3.3.1. Especificação e Síntese de Hardware	41
3.3.2. Método de Rumbaugh.....	44
3.3.3. Outros Algoritmos	48
3.4. Avaliação das Alternativas	49
3.5. Considerações Finais	51
4. GERADOR DE CÓDIGO PARA LINGUAGENS PROCEDURAIS	53
4.1. Considerações Iniciais	53
4.2. Construção de um gerador de código a partir do ambiente STATSIM	53
4.3. Desenvolvimento de um protótipo para o Gerador de Códigos em Occam	55
4.3.1. Construção de um gerador de código a partir do ambiente STATSIM	55
4.3.2. Organização e Interface dos Processos Gerados.....	56
4.3.3. Semântica do Processamento dos Processos-Estado	57
4.3.4. Um exemplo para o algoritmo de execução de Statecharts.....	61
4.3.5. Uma avaliação preliminar.....	66
4.4. Desenvolvimento de um protótipo para o Gerador de Códigos em C++	68
4.4.1. Estruturas de Dados	69
4.4.2. Construção de classes representando estados.....	70
4.4.3. Um exemplo de implementação	74
4.5. Considerações Finais	75
5. GERAÇÃO HTML COM BASE NO HMBS	77
5.1. Considerações Iniciais	77
5.2. Arquitetura do Gerador de Códigos.....	77
5.3. Especificação do HCGBH	79
5.3.1. Conversões de Páginas Texto em HTML	80
5.3.2. Formas de Navegação das Páginas HTML	81
5.3.3. Armazenamento de Informações	85
5.4. Características Adicionais do HCGBH.....	86
5.5. Um Exemplo de Execução do HCGBH.....	87
5.6. Considerações Finais	93

6. CONCLUSÃO	95
6.1. Considerações Iniciais	95
6.2. Contribuições deste trabalho	95
6.3. Cosiderações Finais	96
BIBLIOGRAFIA.....	97

Lista de Figuras

Figura 2-1: Exemplo de Hierarquia de Statecharts	7
Figura 2-2: Exemplo de um Statechart Ortogonal	7
Figura 2-3: Exemplo de História em um Statechart.....	8
Figura 2-4: Exemplo de Statechart	10
Figura 2-5: Árvore de Alcançabilidade para o Statechart da Figura 2-4	11
Figura 2-6: O Ambiente StatSim.....	12
Figura 2-7: O Simulador do Ambiente StatSim.....	13
Figura 2-8: Estrutura de Organização de Informação no HMBS.....	17
Figura 2-9: Editor Gráfico do HySCharts	18
Figura 2-10: O Navegador do Ambiente HySCharts	19
Figura 3-1: Notações do Esquema de Transformação	25
Figura 3-2 - Gráfico de Mapeamento de Combinações de E/S	27
Figura 3-3: Especificação de duas Máquinas com um Buffer	39
Figura 3-4: Um Exemplo de Statechart	41
Figura 3-5: As Sete Máquinas de Estados Finitos.....	42
Figura 3-6: Esquema de Conexão dos Processos.....	43
Figura 3-7: Rede de Caixas Eletrônicos (CE).....	44
Figura 3-8: Diagrama de fluxo de eventos para o exemplo CE.....	45
Figura 3-9: O diagrama de estados para classe CE.....	46
Figura 3-10: Controle do CE.....	47
Figura 3-11: Avaliação dos Possíveis Métodos para Geração de Código.....	50
Figura 4-1: Módulo de Geração de Código do StatSim.....	54
Figura 4-2: Arquitetura do Código Gerado	56
Figura 4-3: Representação Geral em Occam de um Processo	57
Figura 4-4: Especificação Genérica de um PE Básico.....	58
Figura 4-5: Casos Considerados na Geração de Código para os PEs.	60
Figura 4-6: Exemplo de Statecharts.....	62
Figura 4-7: As sete MEFs da Figura 4-6	63

Figura 4-8: Esquema de Conexão das Figuras 3.6 e 3.7	64
Figura 4-9: Representação de Sincronização	66
Figura 4-10: Sincronização entre Ativação/Desativação	67
Figura 4-11: Casos Considerados na Geração de Código	69
Figura 4-12: Descrição do Problema	71
Figura 4-13: Descrição da Classe Pai e suas Derivadas	71
Figura 4-14: As Classes e seus Atributos	73
Figura 4-15: Operação: Gerar Código	74
Figura 4-16: Exemplo de um Statechart	75
Figura 5-1: Arquitetura do Gerador de Códigos	79
Figura 5-2: Estrutura de Classes do HCGBH	80
Figura 5-3: Visão Planar	82
Figura 5-4: Visão de Hierárquica	84
Figura 5-5: Exemplo de Statechart	87
Figura 5-6: Visão Planar do Exemplo da Figura 5-5	89
Figura 5-7: Exemplo de Código Gerado para Visão Planar	89
Figura 5-8: Visão Hierárquica do Exemplo Figura 5-5	90
Figura 5-9: Exemplo de Código Gerado para Visão Hierárquica	91
Figura 5-10: Exemplo de Código Gerado para Visão Hierárquica com N=1	91
Figura 5-11: Exemplo de Código Gerado para Visão por Índice com Evento	92
Figura 5-12: Exemplo de Código Gerado para Visão por Índice sem Eventos	92
Figura 5-13: Índice Geral para Navegação	93

Lista de Tabelas

Tabela 3-1: Combinação de Estado/Entrada dentro de Combinações Estado/Saída..	27
Tabela 4-1: Descrição dos Canais e suas ligações.....	57
Tabela 4-2: Exemplo de Codificação do Estado A11 em Occam	65
Tabela 4-3: Dicionário de dados do Modelo de Objetos	73
Tabela 5-1: Exemplo de Conversão de Página Texto em Página HTML.....	81
Tabela 5-2: Exemplo de Geração de Código para a Visão Planar.....	83
Tabela 5-3: Nomenclatura das Páginas HTML Geradas pelo HCGBH.....	86
Tabela 5-4: Associação de páginas aos estados.....	88

Capítulo 1

1. INTRODUÇÃO

1.1. Considerações Iniciais

A geração automática de código tem se mostrado muito eficiente e é muito importante para o desenvolvimento de sistemas sob dois aspectos:

- auxilia na verificação e validação de sistemas específicos porque erros podem ser encontrados durante a simulação do programa fonte gerado e podem ser retirados antes que o projeto possa ser concluído,
- apresenta um protótipo do sistema que está bem próximo do projeto final, economizando tempo e custo de desenvolvimento.

Um gerador de código recebe uma entrada de dados do usuário na forma de algumas abstrações, desenhos ou interação direta com o sistema e produz um programa em linguagem de mais baixo nível que implementa os detalhes da aplicação. Dessa forma, as atividades rotineiras envolvidas no desenvolvimento de software podem ser automatizadas, reduzindo os custos, aumentando a produtividade e melhorando a qualidade dos sistemas produzidos.

Um modelo baseado em Statechart é geralmente utilizado para especificar o núcleo reativo de um sistema, sendo composto por várias máquinas de estados finitos cooperando concorrente ou hierarquicamente. Essas máquinas podem também ser vistas como máquinas do tipo Moore ou Mealy, pois podem ser associadas ações às transições e atividades aos estados. As ações são executadas instantaneamente e as atividades têm uma duração perceptível. Ambas formam as ações semânticas que caracterizam cada aplicação. Portanto, ao se executar o núcleo reativo em conjunto com as ações e atividades têm-se a aplicação em seu pleno funcionamento.

Há então um grande interesse em algoritmos eficientes para executar núcleos reativos, seja a partir do próprio Statechart (um interpretador, por exemplo) ou então código em uma outra linguagem (um gerador de código/compilador). Além do interesse em eficiência, o desenvolvimento de um gerador de código para Statecharts é por si só complexo devido à complexidade semântica da notação [HAR87b].

Foi desenvolvida no Instituto de Ciências Matemáticas e de Computação (ICMC) uma ferramenta que visa a edição e simulação gráfica de Statecharts, denominada StatSim (Statechart Simulator) este ambiente gráfico auxilia no desenvolvimento e na validação de projetos [FOR91b, BAT91a, BOA92, CAN93a, MEI91c] especificados por Statecharts. Este ambiente gráfico permite atualmente a edição e simulação de sistemas, e a visualização dos resultados será possível através da geração de código para os modelos implementados por esta ferramenta, fechando-se o ciclo de desenvolvimento do projeto.

1.2. Motivação e Objetivos do Trabalho

As ferramentas disponíveis atualmente para codificação são apoiadas por várias outras ferramentas que auxiliam na geração de código final, criando-se um protótipo do sistema. Prototipagem é largamente utilizada e envolve a aplicação de técnicas para estabelecer um conjunto de requisitos final, completo, consistente e correto, antes que o sistema seja projetado, construído, visto ou experimentado pelo usuário.

A ferramenta StatSim não possui atualmente um gerador de códigos, a transformação em código da parte comportamental já foi estudada e apresentada em [MAS93], mas a solução proposta é relativamente ineficiente pois não tira proveito dos componentes hierárquicos e ortogonais dos Statecharts.

A proposta inicial do trabalho era o desenvolvimento de um Gerador de Códigos completo, abstrato e genérico, atendendo a uma grande diversidade de tipos de especificação de projetos implementados através da ferramenta StatSim. O código final seria uma proposta de execução com todos os dados necessários e suficientes para execução. Assim, o usuário teria após a modelagem um resultado mais próximo do sistema a ser desenvolvido. Esse objetivo não chegou a ser atingido plenamente. Fo

desenvolvidos dois protótipos para geração de código, um em Occam e outro em C++, que implementam especificamente um exemplo de Statechart e esse protótipo serviu como base para a definição do algoritmo de transformação a ser usado por um gerador.

Durante o desenvolvimento deste trabalho, paralelamente, foi desenvolvido o HMBS (Hypertext Model Based on Statechart) que usa a estrutura e execução semântica de Statechart para especificar a organização estrutural e a semântica de navegação de hiperdocumentos. A partir deste modelo foi desenvolvida a ferramenta HySCharts (Hyperdocument based System on Statechart) apresentada como um protótipo de sistemas hiperdocumento que tem como objetivo principal validar a proposta do modelo HMBS [OLI96, TUR97] e foi originado a partir do ambiente StatSim, como uma tese de doutorado [TUR98].

Como parte desta dissertação de mestrado foi estudado também a geração de texto HTML a partir da especificação HMBS. Essa geração considera a estrutura do hiperdocumento, que é um Statechart, e também os fragmentos do conteúdo do hiperdocumento. A partir da estrutura hierárquica de Statecharts duas soluções foram implementadas: preservando a mesma semântica de navegação do HMBS e a outra não preservando a semântica.

Assim, este trabalho tem como objetivo permitir a Geração de Código a partir de Statecharts e estudar especificamente a geração em dois domínios diferentes: controle de sistemas reativos e sistemas hiperdocumento, ambos tendo como ponto comum a especificação baseada em Statecharts.

1.3. Organização do Trabalho

O presente trabalho está dividido em seis capítulos. O Capítulo 1 apresenta uma introdução de forma a situar o problema e descrever os principais pontos que levaram ao desenvolvimento deste trabalho. O Capítulo 2 apresenta alguns temas de estudo abordados durante a fase de desenvolvimento do trabalho para a montagem do gerador de códigos. São apresentados neste capítulo os modelos de dados Statecharts e HMBS, a árvore de alcançabilidade e as ferramentas StatSim e HySCharts. O Capítulo 3 faz algumas considerações sobre implementações de sistemas reativos e algoritmos para geração de código, apresentando uma avaliação das alternativas abordadas. O Capítulo 4 apresenta os protótipos desenvolvidos em Occam e C++ a partir dos dados avaliados no Capítulo 3. O Capítulo 5 apresenta de forma detalhada o Gerador de Código desenvolvido para o ambiente HySCharts. O

Capítulo 6 apresenta as conclusões desta dissertação e trabalhos futuros que podem ser desenvolvidos.

Capítulo 2

2. Introdução aos Statecharts, ao HMBS, e às ferramentas de apoio desenvolvidas no ICMC

2.1. Cosiderações Inicias

A base de apoio ao desenvolvimento da ferramenta de Geração de Código foi a utilização de modelagem de sistemas baseados em Statecharts. Os ambientes gráficos StatSim e HySCharts ambos são provenientes da implementação de modelos baseados em Statecharts, sendo assim, são apresentados neste capítulo os modelos de dados e as ferramentas desenvolvidas para executa-los.

As seções seguintes apresentam uma introdução aos Statecharts, a árvore de alcançabilidade, o ambiente gráfico StatSim, o modelo HMBS e o HySCharts, respectivamente. A idéia básica é apresentar de forma resumida estes conceitos de forma a situar o problema e o desenvolvimento da ferramenta de apoio a estes sistemas. No final são apresentadas as conclusões.

2.2. Statecharts

Os Statecharts surgiram em 1987, propostos por Harel, em [HAR87a], para especificar sistemas reativos de forma concisa e rigorosa. Um Statechart pode ser definido como uma extensão de Máquinas de Estados Finitos (MEF), incluindo os conceitos de hierarquia, paralelismo e comunicação. Alguns destes conceitos são abordados de maneira informal nas próximas seções.

2.2.1. Diagrama de Estados

Em um diagrama representado por Statecharts os estados são apresentados graficamente por retângulos com bordas arredondadas usando encapsulamento para expressar a relação de hierarquia em qualquer nível, e as transições por setas direcionadas caracterizando as ligações entre os estados. As transições são rotuladas com eventos e opcionalmente com a condição parentizada representando as variáveis e as expressões de evento (time-out, true, false ...).

As transições **default** não têm estado origem, somente estado destino, e são denotadas por um círculo cheio iniciando a seta. A cada transição **default** poderá estar associado um evento, mas essa associação não é obrigatória. A inicialização permite o disparo do evento, expressão, etc. que estiver associado a essa transição (desde que seja possível); esse tipo de inicialização denomina-se **default com ação**.

Existem transições que têm o evento associado com uma ação, denotadas por **evento/ação**. Nesses casos quando o evento ocorrer, simultaneamente, a ação ocorrerá se o seu estado origem estiver ativo. Uma transição é disparada quando o seu estado origem estiver ativo e ocorrer o evento a ela associada. Esse evento pode ser um evento primitivo ou uma expressão contendo eventos primitivos.

Eventos externos são ocorrências externas ao sistema. Pode acontecer que uma transição tenha uma ação associada, gerando um evento interno ao sistema. Esse evento só poderá ocorrer em decorrência de algum outro evento. As ações são instantâneas, não consomem tempo.

As atividades, em contraposição às ações, quando forem executadas, requerem algum tempo. Elas podem estar associadas a estados e representam alguma função que o sistema deve realizar quando o estado for ativado.

2.2.2. Hierarquia e Ortogonalidade

Hierarquia representa a decomposição de um estado em um ou mais subestados, isso implica que mais de um estado pode estar ativo num mesmo passo. A Figura 2-1 ilustra a representação de hierarquia, os símbolos *e*, *f*, *g* e *h* representam os eventos associados às transições e o símbolo *c* representa uma condição. A transição de *A* para *C*, representada pelo símbolo *g(c)*, apenas ocorrerá

caso g ocorra e a condição c for verdadeira. Os símbolos A , B , C e D representam os estados.

Figura 2-1: Exemplo de Hierarquia de Statecharts

Como pode ser visualizado, os estados A e C são disjuntos e estão dentro de D . Isso significa que D é um estado com decomposição do tipo OR-exclusivo (XOR). Dessa forma, estar em D é equivalente a estar em A e C , mas não em ambos, diz-se que A e C são subestados de D .

Figura 2-2: Exemplo de um Statechart Ortogonal

A ortogonalidade é representada pelos Statecharts através de linhas tracejadas delimitando os estados concorrentes conforme a Figura 2-2. Os estados A e D representam o produto ortogonal de Y , ou seja, iniciar o estado Y , diferentemente da composição XOR, é ativar os estado A e D

simultaneamente.

2.2.3. Comunicação

A comunicação entre os estados ocorre quando um dado evento permite que um estado (origem) seja desativado e um estado (destino) seja ativado. Para que isso ocorra, é avaliada a expressão associada à transição, se essa expressão for verdadeira então a transição ocorrerá. Por exemplo, na Figura 2-2, estando os estados $\{A, D, B, F\}$ ativos, ao ser disparado o evento μ o resultado é $\{A, D, B, E\}$.

Uma outra forma de comunicação entre estados pode ser visualizado na Figura 2-2 onde estando os estados $\{A, D, B, F\}$ ativos, ao ser disparado o evento α o resultado é $\{A, D, C, G\}$, ou seja a transição ocorrerá nos dois estado XOR A e D por representarem uma componente ortogonal e ambos os estados B e F possuírem uma transição associada ao evento α .

2.2.4. História

História é uma característica de Statecharts que permite lembrar de uma visita feita anteriormente a um estado. Existem dois tipos de representação de história que são representadas pelos símbolos H e H^* . A história associada a um estado, representada pelo símbolo H , é aplicada no nível na qual ela aparece, e pelo símbolo H^* é aplicada recursivamente a todos os níveis inferiores.

Figura 2-3: Exemplo de História em um Statechart

Dessa forma, a Figura 2-3 mostra uma entrada por história em K associada ao símbolo H . O estado default G neste caso é irrelevante e o último estado recentemente visitado será iniciado. As considerações referentes ao estado default de G e F são efetuadas e iniciadas de acordo com o estado ativado. Se ao invés do símbolo H fosse atribuído o símbolo H^* , as considerações sobre os estados default de G e F seriam ignoradas.

2.3. Árvore de Alcançabilidade para Statecharts

A árvore de alcançabilidade para Statecharts é semelhante à árvore de alcançabilidade de Redes de Petri, com algumas características próprias, tais como: a notação utilizada, a maneira de representar hierarquia dos estados, o tratamento de história e a forma de utilizá-la [BOA92].

A árvore de alcançabilidade é construída considerando-se todas as possíveis seqüências de eventos do sistema sob desenvolvimento. Em cada passo todos os eventos são considerados como verdadeiro, dessa forma todas as transições habilitadas em cada passo levam a uma nova configuração do sistema. As condições são incluídas nas expressões dos eventos e também são consideradas quando uma transição é tomada. A execução do sistema sob desenvolvimento é uma seqüência representada pelo conjunto $\{(SC_i, \gamma_i, \Pi_i^*), i \geq 0\}$ [MAS94], onde:

- (1) $SC_0 = (X_0, \Pi_0, \theta_0)$, onde X_0 é a configuração inicial, Π_0 é o conjunto de eventos externos gerados pelo ambiente e θ_0 é o conjunto de condições primitivas cujo valor é verdadeiro;
- (2) γ_i é um passo tomado em SC_i , no instante σ_{i+1} para $i \geq 0$.
- (3) Π_i^* é o conjunto de eventos gerados por γ_i , para $i \geq 0$.

Dessa forma, tem-se uma seqüência de configurações de estado onde SC_{i+1} é a configuração do sistema alcançada de SC_i , quando as transições em γ_i , $i \geq 0$ são tomadas. Para cada um dos elementos dessa seqüência toma-se o conjunto $SR = \{(\gamma_i, \Pi_i^*), i \geq 0\}$ de possíveis reações do sistema.

Na árvore de alcançabilidade cada nó é associado a uma configuração de estados, essas

configurações são classificadas de acordo com a sua situação atual. São classificadas em 4 tipos: nova, velha, terminal e história. Configuração nova é aquela que foi gerada e não inserida na árvore; configuração velha é aquela que foi gerada e inserida na árvore; configuração terminal é aquela que não possui nenhuma transição habilitada; e configuração história é representada pelo conjunto de configurações alcançáveis a partir de transições ou estados associados ao símbolo de história. Os símbolos H e h aparecem nestes nós história em substituição às configurações alcançáveis, naqueles estados que são ativados por história H^* e H , respectivamente.

A árvore de alcançabilidade é construída tomando-se a configuração de estado inicial (ou default) e fazendo-a raiz da árvore. A raiz da árvore é uma configuração nova a ser processada. Para essa configuração consideram-se todos os eventos que podem ocorrer. Para cada um deles uma nova configuração é gerada, classificada e inserida na árvore. Esse procedimento se repete até que não existam mais configurações novas a serem processadas, ou seja, todas as configurações alcançáveis foram classificadas.

Figura 2-4: Exemplo de Statechart

A representação de um nó na árvore de alcançabilidade é realizada pela utilização de parênteses

para representar os super-estados do tipo XOR, colchetes para representar os super-estados do tipo AND e os valores 0 e 1 para representar os estados atômicos, sendo 0 para os estados desligados e 1 para os estados ligados.

Um nó da árvore de alcançabilidade para o Statechart da Figura 2-4 teria a configuração $[(0,0)([0,(0,0)(0,0)])]$ representada pelos estados $[(D,C)[E,[(B12,B11),(B22,B21)]]$ e a representação da árvore de alcançabilidade pode ser visualizada na Figura 2-5.

Figura 2-5: Árvore de Alcançabilidade para o Statechart da Figura 2-4

2.4. StatSim

Para utilizar todos os recursos de Statecharts foi desenvolvida uma ferramenta gráfica denominada StatSim (Statechart Simulator) no Instituto de Ciências Matemáticas e de Computação. Este ambiente gráfico possui um editor gráfico de Statecharts que verifica e valida os modelos projetados e vários módulos de simulação, inclusive a simulação gráfica, que permite o disparo de eventos de forma interativa com o usuário. Existe também um editor de diagramas de atividades [TUT91] e um gerador genérico de aplicações [MAS93].

Figura 2-6: O Ambiente StatSim

O sistema StatSim foi desenvolvido originalmente em C padrão ANSI e utilizou os recursos gráficos do Xview para visualização em Estações de trabalho Sun Sparc Station (sistema operacional Solaris).

O editor gráfico permite a especificação do aspecto comportamental de sistemas em forma de diagramas, usando a sintaxe de Statecharts. A sintaxe do Statecharts em edição é analisada a cada elemento inserido no editor, e caso seja detectada alguma incompatibilidade, são emitidas mensagens de erro, e a inserção desse novo elemento é recusada pelo editor.

O editor possui (além das inserções de estados e transições) opções para declaração e consulta de variáveis; consulta de estados e transições criados; modificação na base de dados para esconder uma determinada transição; e manipulação de arquivos em disco (gravação da base de dados).

Uma outra forma de se especificar o Statechart é usando a LES (Linguagem de Especificação de Statecharts) desenvolvida por [FOR91b]. A LES permite a especificação de forma textual e possui a

O modelo HMBS (Hyperdocument Model Based on Statecharts) utiliza a estrutura e execução semântica de Statechart para especificar a estrutura organizacional e a semântica de navegação para a simulação de hiperdocumentos baseados em Statecharts [TUR98]. É um método comportamental devido a capacidade e flexibilidade em modelar a estrutura organizacional e a semântica de navegação associada a hiperdocumentos. O HMBS oferece um mecanismo para efetivar e separar o hiperdocumento em estruturas físicas e lógicas. Um hiperdocumento neste modelo consiste de uma estrutura de Statechart representando a estrutura lógica da documentação em termos de nós e ligações..

2.5.1. Definição Formal

O modelo HMBS oferece mecanismos para efetivamente separar o hiperdocumento físico e as estruturas lógicas. De acordo com o modelo HMBS, um hiperdocumento H é definido como uma 7-tupla.

$$H = \langle ST, P, m, L, pl, ae, N \rangle$$

- ST representa o Statechart que especifica a estrutura organizacional e navegacional subjacente ao hiperdocumento. O Statechart é definido como uma 11-tupla, $ST = \{S, \rho, \psi, \gamma, \delta, V, C, E, A, R, T\}$ representando respectivamente estados, função de hierarquia, função tipo de decomposição, função história, função default, conjunto de expressões, conjunto de condições, conjunto de eventos, conjunto de ações, conjunto de rótulos e conjunto de transições;
- $P = \{p_1, p_2, \dots, p_n\}$, $n > 0$ é o conjunto finito de páginas de informação que define o conteúdo do hiperdocumento;
- $m: S_s \rightarrow P$ denota uma função que mapeia estados “s” de um subconjunto S_s ($S_s \subset S$) para páginas $p \in P$ do hiperdocumento. O subconjunto S_s é definido por

$$S_s: \{ x \in S \mid \psi(x) = OR \vee \rho(x) = \phi \}$$
, isto é “ S_s ” é o conjunto formado pelos estados compostos do tipo OR e pelos estados atômicos do Statechart. Estados do tipo AND não são mapeados pelas páginas;
- $L = \{l_1, l_2, \dots, l_k\}$ $k > 0$, define o conjunto de canais de apresentação, que são abstrações de dispositivos pelos quais é possível especificar os requisitos relativos

à apresentação da informação contida nas páginas do hiperdocumento;

- $pl: P \rightarrow L$ define a função de visualização que associa cada página $p \in P$ a um único canal $l \in L$ capaz de interpretá-la;
- $ae: Anc_p \rightarrow E$ define a função que associa elementos “ a_p ” ($a_p \in Anc_p$) a eventos do Statechart, que por sua vez definem as transições a serem disparadas.
- N ($N \geq 0$) define o nível de visibilidade ou de navegação do hiperdocumento. O autor do hiperdocumento pode utilizar o valor de N para definir a profundidade da hierarquia na estrutura especificada pelo Statechart subjacente a ser apresentada durante a navegação.

2.5.2. Definição Informal

O HMBS possui três tipos de objetos: estruturais, navegacionais, e de apresentação. Os objetos estruturais podem ser especificados como os estados, transições e eventos representando a estrutura organizacional do Statechart. Os objetos navegacionais definem a forma como o modelo será percorrido sendo composto pelas páginas, ligações e âncoras. E por fim, os canais definem os objetos relativos à apresentação do hiperdocumento.

2.5.3. Obejtos Estruturais

Os objetos estruturais podem ser definidos como a estrutura organizacional dos Statecharts. A forma como os estados, eventos e transições são tratados neste contexto são equivalentes àqueles definidos na seção 2.2.

2.5.4. Objetos Navegacionais

A estrutura navegacional de um hiperdocumento é especificada por objetos estruturais formados pelos estados, transições e eventos; pelos objetos navegacionais representado pelas páginas, âncoras e ligações; e pelos mecanismos de apresentação da informação, os canais. A semântica de navegação do HMBS define o comportamento das páginas, âncoras e as transformações navegacionais que ocorrem quando o leitor interage, assim, o comportamento dinâmico desta semântica é baseado na

semântica operacional de Statecharts a partir de uma configuração de estados qualquer.

As páginas são especificadas por um conjunto de atributos formando um objeto estrutural “estado”. Possuem título, conteúdo e âncoras. As ligações também referenciadas como “links”, são representadas pelo conjunto de transições definidas pelos Statecharts, sendo que a distinção entre eles pode ser visualizados, uma vez que os objetos estruturais “transição” representam o inter-relacionamento entre os estados na estrutura organizacional e os objetos estruturais “ligações” representam o inter-relacionamento entre os objetos “página” durante a navegação, pelas âncoras definidas no contexto.

O objeto navegacional “âncora” pode ser representado como sendo os pontos de ancoragem de um hiperdocumento, representa a ligação entre as páginas, e a sua ativação faz com que todos os eventos associados àquela âncora sejam disparados de acordo com a semântica operacional de Statecharts.

2.5.5. Objetos de Apresentação

Os objetos de apresentação são definidos pelos canais. Os canais são classificados de acordo com o tipo de mídia que manipulam, pelos atributos de apresentação de alto nível e pela coordenação espacial. Os tipos de mídia podem ser classificados em: texto, imagem, áudio ou vídeo; os atributos de apresentação de alto nível são representados pelo conjunto de características, tais como o tipo e tamanho da fonte, a cor de fundo do documento, o volume de áudio, entre outros; e a coordenação espacial define a formatação das janelas de apresentação tais como a largura e a altura de uma janela lógica.

2.5.6. Características do HMBS

O HMBS permite que a informação seja estruturada em três tipos: hierárquica, linear e rede (grafos); podendo inclusive combiná-los. A Figura 2-8 mostra os três tipos de estrutura organizacional mencionado anteriormente.

A separação entre a estrutura e o conteúdo do hiperdocumento é possível pelos mapeamentos definidos no modelo. O mecanismo de definição dos mapeamentos no HMBS garante

flexibilidade na especificação, possibilitando que objetos, tanto da estrutura quanto do conteúdo, sejam reutilizados de maneiras distintas.

Figura 2-8: Estrutura de Organização de Informação no HMBS

Os contextos de informação existentes no hiperdocumento podem ser definidos utilizando o conceito de super-estados de Statecharts. Um estado-pai determina o contexto dos seus subestados e os inter-relacionamentos entre eles, e define assim os nós ou páginas de composição que permitem aninhamento.

A possibilidade de decomposição de estados inerentes aos Statecharts permite abordar o processo de especificação de hiperdocumentos em vários níveis de abstração e granularidade. A capacidade de especificar concorrência entre estados no Statecharts possibilitou ao HMBS especificar a apresentação simultânea de páginas, permitindo a definição de caminhos concorrentes de navegação, ou seja, ligações independentes podem ser definidas para causar a modificação de apenas uma parte da apresentação.

O HMBS permite ainda duas formas de acesso à informação: a forma tradicional, seguindo as ligações habilitadas por âncoras, e o acesso direto à informação implementado por índices, que são estruturas de acesso alternativas apoiadas pelo modelo.

2.6. HySCharts

O HySCharts (*Hyperdocument System based on Statecharts*) é um protótipo de sistema

hiperdocumento cujo objetivo principal é validar a proposta do modelo HMBS [OLI96, TUR97], o qual utiliza Statecharts para especificar a estrutura e a semântica de navegação de um hiperdocumento. O sistema permite a criação de um hiperdocumento, por intermédio do módulo de autoria, e uma conseqüente exploração do hiperdocumento por intermédio do módulo de navegação.

O sistema HySCharts foi desenvolvido originalmente em C++ padrão ANSI, e X-View para as interfaces gráficas e de manipulação de janelas em ambiente Unix, utilizando-se sistema operacional Solaris.

Figura 2-9: Editor Gráfico do HySCharts

A arquitetura do sistema HySCharts é composta por três camadas principais: aplicação, estruturação e de armazenamento. A camada de aplicação é subdividida nos módulos autoria e de navegação. No módulo de autoria o autor modela o hiperdocumento e no módulo navegação o autor visualiza a navegação do modelo gerado. A camada de estruturação é composta pela estrutura interna subjacente à formalização do hiperdocumento em HMBS com base nos objetos estruturais, navegacionais e de

apresentação, e que por sua vez são armazenados em bases de dados gerenciadas na camada de armazenamento.

A ambiente gráfico de autoria permite a modelagem dos Statecharts e a definição dos objetos navegacionais definidos no HMBS Na Figura 2-9 pode ser vista a tela de autoria do ambiente HySCharts, onde é apresentada uma lista de páginas criadas dentro do ambiente gráfico e a interface utilizada para isso na janela “create page”.

O módulo de navegação definido pelo HyCharts permite que os dados modelados durante a autoria possam ser visualizados e navegados como hiperdocumentos. A simulação pode ser analisada visualmente com as páginas de hiperdocumentos sendo apresentadas na tela de acordo com os módulos definidos durante a autoria para visualização dos hiperdocumentos (forma de acesso tradicional ou por índice).

Figura 2-10: O Navegador do Ambiente HySCharts

O HySCharts define um conjunto de janelas para navegação do hiperdocumento, como foi definido

anteriormente e, além dessas janelas lógicas utilizadas para apresentação do hiperdocumento, uma outra janela “Show_Hview” é apresentada no ambiente de navegação (Figura 2-10). Essa janela está associada à apresentação dos níveis hierárquicos definidos no Statecharts, e uma vez pressionada a opção Hview, os botões “UP”, “DOWN” e “DISMISS” são ativados e uma nova apresentação é realizada. Como o próprio nome diz, as configurações “UP”, “DOWN” e “DISMISS”, representam a visualização das configurações de páginas associadas aos estados no nível acima ou abaixo do correntemente visualizado (determinado pelo nível de visibilidade, N), ou na desativação da opção Hview respectivamente.

Existe também um outro botão de navegação denominado “BACK” o qual permite o retorno à configuração imediatamente anterior àquela atualmente visitada. É importante ressaltar que essa operação não está associada às características de contexto definidas no Statechart para os níveis hierárquicos e foi implementada na ferramenta como um recurso adicional de navegação.

2.7. Considerações Finais

Este capítulo apresentou sucintamente as técnicas Statecharts e HMBS, apresentou a árvore de alcançabilidade e os ambientes gráficos StatSim e HySCharts desenvolvidos a partir dessas técnicas. Assim, alguns conceitos e conhecimentos importantes para o entendimento dos geradores de códigos propostos nos dois capítulos seguintes foram aqui apresentados.

Capítulo 3

3. Geração de Código para Sistemas Reativos Especificados com base em Statecharts

3.1. Considerações Iniciais

A especificação e o desenvolvimento de um gerador de código dependem do grau de granularidade requerido, da eficiência desejada e da linguagem alvo a ser gerada. A linguagem alvo deve conter alguns aspectos essenciais para representação de tempo, pois sem isso o código gerado pode ser ineficiente.

Assim, algumas métodos e algoritmos são apresentados neste capítulo visando a situar o problema frente as opções existentes hoje. Os tópicos estudados aqui são:

- métodos, ferramentas e linguagens para sistemas reativos,
- algoritmos de geração de código existente na literatura,

Na seção 3.2 são apresentados alguns métodos e projetos de Sistemas Reativos, na seção 3.3 serão apresentados alguns algoritmos para geração de código e na seção 3.4 é apresentada uma avaliação das alternativas discutidas para geração de código.

3.2. Abordagens para Métodos e Projetos de Sistemas Reativos

Um sistema reativo é caracterizado por ser orientado a eventos, ou seja, está continuamente respondendo a estímulos do meio externo e interno. Estes estímulos devem produzir os resultados

corretos dentro de um intervalo de tempo previamente especificado. Se o sistema atrasa em responder a um destes estímulos, então o sistema falha. A interação constante como ambiente e as restrições de tempo impostas tornam a especificação de sistemas reativos uma tarefa bastante complexa.

Esta complexidade se dá devido à dificuldade em descrever comportamentos reativos de um modo claro e realista e ao mesmo tempo formal e rigoroso, para que se obtenha uma simulação detalhada. Vários métodos, técnicas e algoritmos foram propostos para:

- identificar e representar atividades paralelas;
- representar interrupção e mudanças de contexto;
- permitir concorrência entre processos/tarefas;
- permitir comunicação e sincronismo entre tarefas;
- representar extensa variação de dados e taxas de comunicação;
- representar restrições de tempo;
- manusear erros e descobrir falhas;
- permitir processamento assíncrono;
- ter acoplamento necessário com elementos externos do sistema.

Dentre estes métodos, alguns dos mais importantes são: DARTS [GOM84], o Esquema de Transformação [WAR89], STATEMATE [HAR90].

3.2.1. - DARTS

DARTS (Design Method for Real-Time System) é um método de projeto para sistemas de tempo real que visa ao desenvolvimento de um sistema estruturado, altamente modular, com interfaces bem definidas e redução de acoplamento entre tarefas [GOM84]. É uma extensão do projeto orientado a fluxo de dados, ou seja, os dados são considerados como fluxos de entrada e saída e podem ser transformados por tarefas de software.

O método DARTS é composto dos seguintes passos:

- Análise do Fluxo de Dados
- Decomposição em tarefas
- Definição da Interface das Tarefas
- Projeto de Tarefas

Análise do Fluxo de Dados

O método começa com a aplicação de fundamentos de análise de sistemas utilizando a notação de fluxo de dados. Diagramas de fluxo de dados são criados; um dicionário de dados correspondente é definido e interfaces entre as principais funções do sistema são estabelecidas [PRE92].

Decomposição em Tarefas

Depois de se ter definido todas as funções do sistema e o fluxo de dados entre elas, definem-se em seguida os requisitos de concorrências. As transformações em diagramas de fluxo de dados são analisadas para identificar quais podem rodar concorrentemente e quais são sequenciais.

Os critérios para decidir se uma transformação deve ser uma tarefa separada ou agrupada em uma única tarefa são os seguintes:

- *Dependência de E/S*: o processo depende do dispositivo de E/S com o qual interage. Neste caso, a transformação precisa ser uma tarefa separada;
- *Função de tempo crítico*: deve rodar com alta prioridade e portanto é bom que seja uma tarefa separada;
- *Requisitos comportamentais*: uma função (ou conjunto delas) que usa intensivamente o computador deve rodar com baixa prioridade, consumindo poucos ciclos de CPU, a cada ativação;
- *Coesão funcional*: transformações que efetuam um conjunto de funções altamente relacionadas podem ser agrupadas juntas como uma tarefa. Se o tráfego de dados for grande e as funções consideradas como tarefas separadas, a coesão funcional é garantida tanto no nível de módulo (função) como no nível da tarefa;
- *Coesão temporal*: certas transformações permitem que funções sejam carregadas para

execução ao mesmo tempo. Estas funções podem ser agrupadas dentro de uma tarefa até que sejam executadas a cada vez que a tarefa recebe um estímulo;

- *Execução periódica*: uma transformação que necessita ser executada periodicamente pode ser estruturada como uma tarefa separada, que é ativada em intervalos regulares.

Definição da Interface das Tarefas

Em DARTS, interfaces de tarefas são especificadas pela definição de duas classes de módulos de interface, que são:

- *Módulo de Comunicação de Tarefa (TCM)*: manuseia todos os casos de comunicação entre tarefas, contém estrutura de dados e define os procedimentos de acesso ao módulo. É criado pela comunicação entre tarefas e usa a sincronização do sistema operacional para garantir acessos aos dados. São divididos em duas categorias:
 - ⇒ *Módulos de Comunicação por Mensagens (MCMs)*: suporta comunicação por mensagens e implementa mecanismos apropriados para gerenciar fila de mensagens e primitivas de sincronização;
 - ⇒ *Módulos de Ocultamento de Informação (IHM)*: oferece acesso a porções de dados ou armazenamento de dados. O IHM implementa a estrutura de dados, bem como o método de acesso que habilita outras tarefas a ter acesso à estrutura de dados.
- *Módulo de Sincronização de Tarefas (TSM)*: é o módulo principal ou supervisor de uma tarefa. Há somente um TSM por tarefa e é requisitado apenas por tarefas que tenham muitas tarefas sincronizadas. Neste módulo, a tarefa espera que um ou mais eventos ocorram, estes podem ser eventos sincronizados ou eventos na fila de mensagens. Dependendo da circunstância a tarefa pode esperar por diferentes eventos em diferentes instantes de tempo.

Projeto da Tarefa

Uma tarefa em um sistema de tempo-real é um programa que pode ser projetado usando métodos convencionais. Neste estágio DARTS permite projetar cada tarefa individual como um programa sequencial. Esta tarefa pode ser vista como um processo sequencial, implementado como uma

estrutura de programa organizado hierarquicamente. Cada módulo no programa estruturado pode ser projetado usando a filosofia de programação estruturada.

Portanto, o DFD que representa o fluxo de dados dentro da tarefa pode ser mapeado para um programa estruturado usando transformação convencional ou a técnica de mapeamento de transação. Porém, sistemas de tempo-real são dependentes de ambos: a entrada de dados da tarefa e o estado corrente do sistema. Por esta razão controle é dependente do estado.

3.2.2. Esquema de Transformação

Diagramas de fluxo de dados têm sido extensivamente usados para modelar aspectos de transformação de sistemas propostos. Uma extensão do diagrama de fluxo de dados é chamado esquema transformacional e tem capacidade generalizada para representar os aspectos de tempo e controle do sistema.

Figura 3-1: Notações do Esquema de Transformação

O Esquema de Transformação [WAR89] oferece uma notação e formação de regras para se construir um modelo abrangente de sistemas e um conjunto de regras de execução para permitir compreender o comportamento do sistema modelado desse modo.

A notação do esquema básico de transformação é ilustrada pela Figura 3-1 e pode ser interpretada da

seguinte forma:

- *Transformações*: são divididas em transformações de dados e transformações de controle.
 - ⇒ *Transformações de dados* permitem a alteração de valores de variáveis, modificação e criação de dados armazenados, etc. Pode-se dizer então que é uma abstração das funções de baixo nível do sistema onde os dados podem ser acessados ou controlados externamente.
 - ⇒ *Transformações de controle* gerenciam o comportamento das transformações de dados através de suas ativações e desativações, ou seja, são abstrações de controle lógico do sistema
- *Fluxos*: são divididos em fluxo de dados e fluxo de eventos.
 - ⇒ *Fluxo de dados*: podem representar dados discretos ou contínuos. Fluxos de dados discretos são associados a valores de variáveis que são definidos em pontos discretos no tempo. Fluxos de dados contínuos são associados a um valor ou um conjunto de valores definidos continuamente sobre um intervalo de tempo;
 - ⇒ *Fluxo de eventos*: representam o que está acontecendo no sistema através de mensagens dentro do sistema. Há três tipos de fluxos de eventos, que são:
 - * *Fluxo de sinal*: um sinal é enviado ao emissor para relatar que alguma coisa aconteceu, ou seja, um sinal foi enviado mas não foi reconhecido;
 - * *Fluxo de ativação*: uma fluxo de ativação diz que uma mensagem deve ser enviada. O emissor manda o receptor produzir alguma saída;
 - * *Fluxo de desativação*: um fluxo de desativação não permite que uma mensagem seja enviada. Os emissores não permitem que uma saída seja produzida pelo receptor.
- *Armazenamento*: são divididos em armazenamento de dados e buffer.
 - ⇒ *Armazenamento de dados*: age como um depósito de dados para armazenamento. Pode ser visto como uma abstração de arquivos.
 - ⇒ *Buffer*: é um tipo especial de armazenamento no qual fluxos produzidos por uma ou mais transformações estão sujeitos a atraso de armazenamento antes de serem usados. O buffer é uma abstração de pilha ou fila.
- *Máquinas de Estado*: A notação inclui a representação de autômatos finitos representados

como máquinas de Mealy. O autômato pode ser representado por uma tabela mapeando combinação de estado/entrada para combinações estado/saída (Tabela 3-1) ou por um gráfico equivalente, como ilustrado pela Figura 3-2

Estados	Entradas			
	I1	I2	I3	I4
S1 {inicial}	01;S2	N/C	N/C	N/C
S2	N/C	02;S1	01;S3	N/C
S3	N/C	04;S1	N/C	03;S3

N/C sem combinação

Tabela 3-1: Combinação de Estado/Entrada dentro de Combinações Estado/Saída

Figura 3-2 - Gráfico de Mapeamento de Combinações de E/S

A notação e o formalismo de regras permite a descrição de um sistema como uma rede concorrente de processos (transformação) e dados armazenados, ligados por caminhos de comunicação (fluxos de

dados).

O esquema de transformação permite, ainda, a criação e avaliação de dois diferentes tipos de modelos de sistemas [WAR89]:

- *modelo essencial* (requisitos): o esquema é usado para representar uma máquina virtual com recursos infinitos. Os elementos do esquema descrito idealizam componentes de processamento e memória;
- *modelo de implementação*: o esquema é usado para representar a máquina real com recursos limitados e os resultados da execução predizem o comportamento de uma implementação dos requisitos.

3.2.3. STATEMATE

STATEMATE é uma ambiente de desenvolvimento de software construído pela Ad Cad Ltda subordinada à i-Logix Inc. É um conjunto de ferramentas gráficas orientadas para especificação, análise, projeto e documentação de sistemas reativos grandes e complexos. É baseado na preparação e análise de um modelo conceitual rigoroso que capta não apenas funcionalidade, mas também o comportamento dinâmico.

STATEMATE é composto de três diferentes tipos de linguagens gráficas: diagrama de módulos, que mostra a visão estrutural do sistema; diagrama de atividades, que mostra a visão funcional; e diagrama de estados, que mostra o comportamento do sistema. As três linguagens são especificadas usando formalismo visual claro e intuitivo, e ao mesmo tempo preciso e rigoroso, podendo portanto, serem analisadas automaticamente.

Cada linguagem permite preparar, analisar e depurar diagramas com o apoio de editores gráficos com sintaxe e semântica bem definidas, permitindo descrever o sistema sob desenvolvimento, inter-relacionando as três visões.

Visões em STATEMATE

A visão estrutural oferece uma decomposição hierárquica dentro de seus componentes físicos chamados módulos e identifica a informação que flui entre eles; que é “a parte” de dados e sinais de controle que fluem sem interrupção, por mais que ligações físicas existam entre os módulos.

A visão funcional não permite especificação dinâmica; não diz *quando* e *porque* as atividades são ativadas, se ou não elas terminam em si mesmas, e se elas podem ser carregadas em paralelo. O mesmo é verdade para o fluxo de dados; na visão funcional é apenas especificado que dados podem fluir e não *como* e *quando* fluirão. Em outras palavras, a visão funcional oferece a decomposição dentro de atividades e a possibilidade de fluxo de informação, mas não diz nada sobre como essas atividades e suas entradas e saídas associadas são controladas.

A visão comportamental é usada para especificar o controle de atividades. Estas podem ser apresentadas em qualquer nível de atividade hierárquica, controlando aquele nível particular. São estes controladores que são responsáveis por especificar *como* e *porque* “coisas” acontecem. Entre outros recursos, um controlador de Statechart pode começar e parar atividades, pode gerar novos eventos e pode mudar os valores das variáveis. Ele pode sentir também se atividades estão ativas ou dados fluírem, e pode responder a eventos e testes de valores de condições e variáveis.

Geração de Código

Um aspecto adicional de STATEMATE é a capacidade de geração de código. Em STATEMATE, isto é feito usando o pacote prototipador, o qual pode ser instruído para traduzir automaticamente um diagrama de atividades com todos os seus descendentes, incluindo seu controlador de Statecharts, para uma linguagem de alto nível como ADA. O principal uso do código resultante está na possibilidade de observar o desempenho do sistema sob circunstâncias do mundo real. Dessa forma, ao invés de visualizar diagramas de estados e diagramas de atividades executando cenários de um modo animado, o usuário pode interagir com um protótipo do sistema.

A abordagem sobre como foi efetuada a tradução dos diagramas em código fonte não está disponível, ou seja, não existe publicação sobre como esta ferramenta foi projetada, mas apenas sabe-se que ela existe e gera código em ADA.

Outras Características

STATEMATE oferece ainda ferramentas que recuperam e formatam informações do modelo. O usuário pode primeiro listar um conjunto de atividades de interesse particular e então requisitar o relatório; ele será aplicado apenas para as atividades da lista. Entre os relatórios correntemente implementados estão: dicionário de dados de vários tipos, protocolos textuais de estados ou atividades que contém todas as informações relevantes a eles, diagramas da interface e versão de árvores de várias hierarquias chamadas de diagramas- N^2 . Adicionalmente ao relatório, STATEMATE oferece uma linguagem de geração de documento com a qual os usuários podem criar seus próprios documentos.

Uma das grandes vantagens de STATEMATE é a execução e análise dinâmica de um sistema especificado. O sistema pode ser executado das seguintes formas:

- **modo interativo:** passo a passo, onde em cada passo o usuário gera eventos externos, muda condições e carrega outras ações (como mudança de valores de variáveis), emulando o ambiente do sistema;
- **execução programada:** execução não interativa, executando condições aleatórias e situações típicas e não típicas. É realizada com base em uma Linguagem de Controle de Simulação (SCL), capacitando o usuário a manter controle geral sobre como a execução será processada.
- **interativo/programado:** a execução pode ser parada, retornando o controle ao usuário, que pode gerar eventos interativamente e depois voltar à execução programada, e assim por diante.
- **execução exaustiva:** é permitida para testar a confiabilidade, não determinismo, deadlock e uso de transições. É muito importante pois permite esgotar as possíveis situações de erro em tempo de execução e eliminá-los.

3.2.4. ESTEREL

ESTEREL é uma linguagem síncrona, originária da união dos projetos INRIA-ENSMP sobre a semântica do paralelismo. Os princípios básicos de ESTEREL são apresentados aqui, considerando-

se: o modelo básico; a semântica, a implementação e o ambiente de Esterel [BOU91].

Modelo Básico

Seu modelo básico pode ser resumido pelos seguintes aspectos:

- *reatividade*: a vida de um sistema reativo é dividida em instantes que são os momentos nos quais ele reage. Declarações reativas são aquelas definidas para referenciar instantes. Assim, pode-se dizer que reatividade é um conjunto de instantes lógicos e programas que utilizam declarações reativas.
- *atomicidade de reações*: não permite a ativação de um sistema enquanto ele ainda reage para uma ativação corrente. Ele permite considerar programas reativos com história: um programa reativo é uma função que produz uma sequência de eventos de saída a partir de uma sequência de eventos de entrada.
- *transmissão instantânea*: a comunicação é análoga à comunicação de rádio, em que há muitos receptores que recebem a mesma informação. Em ESTEREL, a comunicação é feita usando sinais que podem ser emitidos, testados para verificar se estão presentes e que podem ter um valor. Transmissão em ESTEREL tem uma característica importante: emissões e recepções não finalizam o instante corrente. Em outras palavras, pode haver várias emissões e recepções de sinais sequencialmente no mesmo instante, isto é, o passo atual da execução.
- *determinismo*: é um outro modo de simplificar programação reativa. Com programas determinísticos, comportamentos são reproduzidos, simplificando validações e testes de programas. Em ESTEREL, paralelismo é determinístico.

Semântica ESTEREL

A semântica intuitiva de ESTEREL descreve o comportamento de um módulo sobre uma entrada dada. Denomina-se *evento de entrada* à ocorrência de um ou possivelmente vários sinais de entrada simultâneos vindos do ambiente. ESTEREL reage a cada *evento de entrada* pela atualização das variáveis locais e pela emissão de sinais locais e de saída. A emissão de sinais de saída fará com que o ambiente ESTEREL envie o *evento de saída* para um módulo do ambiente. Este processo completo

é chamado de “reação” de um evento de entrada. A reação deve ser simultânea: o evento de saída está sincronizado com o evento de entrada.

A idéia básica da semântica intuitiva, é descrever formalmente não apenas as ações efetuadas por cada declaração (atualização na memória, emissão de sinal ou testes), mas também o aspecto temporal, isto é, em qual instante eles foram efetuados.

Além da semântica intuitiva, ESTEREL possui uma semântica de mais alto nível, baseada em notação que pode ser vista como semântica comportamental e a semântica computacional. A semântica comportamental define cada reação globalmente. Dado um programa qualquer e um evento de entrada, ela determina o evento de saída gerado pela reação do programa, e este novo estado do programa torna-se habilitado para manusear eventos de entradas subsequentes. A dedução semântica natural usual é “executável”: suas regras podem ser diretamente usadas para construir interpretadores.

Já a semântica computacional, ao invés de definir comportamentos de um modo global, permite computar os resultados de seqüências de ações de uma execução. Sinais são implementados usando memória compartilhada, com a seguinte disciplina de leitura/escrita de compartilhamento: um sinal não pode ser lido até que ele não possa mais ser escrito.

Geração de Código em ESTEREL

Embora manipulem dados, programas ESTEREL podem ser traduzidos em autômatos de forma tão simples quanto aquela usada para programas com sincronização pura. Um programa que considera sincronização pura é aquele que apenas contém sinais puros, sem nenhuma variável, constante ou valor de qualquer tipo. A idéia básica é guardar na memória ações formais em tempo de compilação desde que eles possam apenas ser efetuados em tempo de execução. Uma transição no autômato consiste de uma seqüência de ações em tempo de execução (mais precisamente de uma árvore de ações devidas às condições).

As ações são operações efetuadas sobre variáveis alocadas globalmente. Uma variável o código objeto é reservada para cada variável alocada explicitamente ou pelo valor do sinal de um código fonte. Outras variáveis implícitas são alocadas a cada ocorrência, como contadoras de loop de

repetição, indicando presença de sinal de entrada boolean. As seguintes ações podem aparecer em transições: associações, chamadas de procedimentos externos, testes e emissão de sinal de saída.

Um exemplo utilizando a compilação de um programa fonte em ESTEREL, gerando um autômato, é mostrado a seguir. Note que este exemplo considera sincronização pura, ou seja, não são consideradas variáveis.

```
module P:
  input I;
  output O;
  signal S in
    emit O;
  loop
    await I;
    await I do emit S end
  ||
    await S do emit O end
  end
end
```

Neste exemplo foram utilizadas algumas declarações ESTEREL, cujo significados são os seguinte:

- **Declarações de interface**

input: é um sinal que vem do ambiente e que não pode ser emitido internamente no Esterel Básico. Sua forma geral é:

input S (**type**)

onde S é um sinal de entrada e **type** é o tipo da informação que vai ser recebida.

output: é um sinal emitido para o ambiente e possui a forma geral:

output S (**combine type with comb**)

onde **type** e **comb** já devem ter sido declarados e **comb** deve ser declarado como uma função **function comb (type,type): type;**

- **Declarações básicas**

signal: é uma declaração de sinal local. Sua forma geral é:

signal S (combine type with comb) in stat end

onde **type** e **comb** são declarados como em **output** e **stat** são as declarações que podem ser executadas.

emit: é uma declaração de emissão de sinal. Sua forma geral é:

emit S(exp)

onde **S** é um sinal com o valor da expressão **exp**.

loop: é um loop infinito. Sua forma geral é:

loop stat end

onde **stat** é uma declaração opcional.

||: representa o paralelismo, ou seja, indica que as instruções devem ser executadas em paralelo.

- **Outras:**

await} é uma declaração que espera a chegada de um sinal. Sua forma geral é:

await S do stat end

onde **S** é o sinal esperado e **stat** representa as declarações que devem ser executadas com a chegada do sinal **S**.

o autômato produzido para o exemplo acima é:

state 0:

e -> output O; goto 1

l -> output O; goto 1

state 1:

e -> goto 1

l -> goto 2

state 2:

e -> goto 2

l -> output O; goto 1

onde “output O” significa “emita o sinal de saída O” e “goto n” significa “a reação corrente acabou,

tratar a próxima reação no estado n ".

Note que o algoritmo traduz um programa concorrente em um sequencial. Concorrência é tratada em tempo de compilação e não em tempo de execução. Note também que o sinal local S desaparece completamente no código compilado. Um sinal local age como um auxiliar não terminal no gerador.

Ambiente ESTEREL

O ambiente é composto por dois conjuntos distintos de ferramentas:

- *Ferramenta de Verificação e Validação*: são aquelas que representam a estrutura interna do autômato compilado. Permitem comparações com especificações, bem como a visualização total ou parcial de estados e transições.
- *Ferramenta de Simulação e Desenvolvimento*: são aquelas que consideram programas ESTEREL como caixas reativas. Permitem ao usuário ativá-las interativamente e registrar as reações correspondentes.

3.2.5. STATEXT

STATEXT é uma representação textual dos Statecharts [KES92] criada para permitir o estudo dos aspectos independentes da sintaxe de Statecharts e apresentar uma representação alternativa para as pessoas que preferem representações textuais de programas e especificações de linguagem. Como linguagem textual STATEXT é construída sem declarações, manipulando dados e recebendo mensagens.

A linguagem STATEXT é baseada em variáveis compartilhadas e canais de comunicação. Ambos, variáveis e sinais, são tipados. É exigido que o domínio de cada canal contenha um valor especial $\perp$, indicando que não há mensagem corrente (um canal vazio). Expressões são formadas pela aplicação de operações a constantes, variáveis e canais. E finalmente, uma especificação de intervalo pode ser usada para especificar aspectos de tempo real. Esta delimitado por:

$[l,u], [l,u), (l,u], (l,u)$

onde l e u são números reais, $0 \leq l \leq u \leq \infty$, chamados de limites inferior e superior respectivamente. Para especificar qualquer um dos casos acima é escrito $\{l, u\}$, e para qualquer t pertencente ao intervalo I , denotado por $t \in I$, de acordo com os 4 casos anteriores tem-se:

$$t \in [l, u] \text{ iff } l \leq t \leq u$$

$$t \in (l, u] \text{ iff } l < t \leq u$$

$$t \in [l, u) \text{ iff } l \leq t < u$$

$$t \in (l, u) \text{ iff } l < t < u$$

Diz-se que t não excede I , denotado por $t \notin I$, se existe um $t' \in I$ tal que $t \leq t'$.

Comandos em STATEXT

Os comandos simples em STATEXT são os seguintes:

- *skip*: esta declaração serve como um filtro. Ela não faz nada e termina em um único passo de execução.
- *idle*: como a declaração *skip*, a declaração *idle* não altera o estado dos dados. Entretanto, diferentemente da declaração *skip*, a declaração *idle* nunca termina. O único modo de sair da declaração *idle* é pela preempção, a qual é a principal característica de STATEXT (e dos Statecharts)
- *ascertain*: possui a forma geral,

ascertain c for I

onde c é uma expressão boolean (condição) e I é um intervalo especificado. A declaração espera até que a condição c seja continuamente verdadeira para o tempo $t \in I$ e quando isso ocorre, ela termina.

- *Associação*: para uma lista $\bar{y}$ de variáveis e uma lista $\bar{e}$ de expressões do mesmo tamanho e tipos correspondentes,

$$\bar{y} := \bar{e}$$

é uma declaração de associação. Sua execução avalia a lista de expressões $\bar{e} = e_1, \dots, e_m$

e então as associa às variáveis $\bar{y} = y_1, \dots, y_m$ respectivamente. Uma associação é habilitada, portanto, está pronta, isto é, o controle está a frente da associação, e termina em um único passo.

- *Send*: dado um canal α e uma expressão e ,

$$\alpha ! e$$

é uma declaração de envio.

- *Receive*: dado um canal α e uma variável y de tipo compatível

$$\alpha ? y$$

é uma declaração de recebimento.

Declarações compostas:

- *Condicional*: para as declarações S_1 e S_2 e a expressão boolean c

$$\text{if } c \text{ then } S_1 \text{ else } S_2$$

é uma declaração condicional. A expressão c é avaliada e testada. Se a condição for verdadeira então a declaração S_1 é selecionada para execução subsequente; caso contrário S_2 é selecionada.

- *while*: para uma expressão booleana c e uma declaração S ,

$$\text{while } c \text{ do } S$$

é uma declaração **while**. A execução começa com a avaliação de c . Se c é falso, a execução da declaração termina; caso contrário, passos subsequentes ocorrem para executar S . Se S termina, c é testado novamente, e assim por diante, até c tornar-se falso.

- *Concatenação*: para as declarações S_1 e S_2

$$S_1; S_2$$

é uma declaração de concatenação. É uma composição sequencial. Primeiro S_1 é executado e quando termina, S_2 é executado.

- *Seleção*: para as declarações S_1 e S_2

$$S_1 \cup S_2$$

é uma declaração de seleção. Como primeiro passo, um de S_1 e S_2 , o que estiver habilitado, é selecionado, e o primeiro passo na declaração de seleção é executado. Passos subsequentes ocorrem para executar o resto de subdeclarações de seleções. Se ambos S_1 e S_2 estão habilitados, a seleção é não-determinística. Se ambos S_1 e S_2 estão

desabilitados a declaração aguarda um dos dois ser habilitado.

- *Cooperação*: para as declarações S_1 e S_2

$$S_1 \parallel S_2$$

é uma declaração de cooperação. Este comando faz a execução paralela de S_1 e S_2 . Execução paralela é modelada por inter-relacionamentos. Passos em $[S_1 \parallel S_2]$ são inter-relacionados por passos de S_1 e S_2 . A declaração de cooperação termina quando ambos S_1 e S_2 terminam.

- *Preempção*: para as declarações S_1 e S_2

$$S_1 \cup S_2$$

é uma declaração de preempção. Passos na execução desta declaração são ou passos da execução de S_1 ou qualquer *primeiro* passo na execução de S_2 . Qualquer passo que conduz ao término de S_1 também causa o término de $S_1 \cup S_2$. Execução de um passo de S_2 causa o descarte do resto de S_1 , e procede a execução de passos de S_2 até terminar. Ou seja, executar S_1 até que um primeiro passo de S_2 seja executado.

A Figura 3-3 mostra como exemplo um Statechart especificando duas máquinas que se comunicam por um buffer que pode pegar apenas um item de cada vez. Este buffer pode representar um receptáculo que viaja para trás e para diante entre as duas máquinas.

A especificação consiste de 3 autômatos: M_1 , M_2 e Buffer, que agem concorrentemente. Esses componentes podem representar uma máquina que faz um processamento inicial, uma segunda máquina mais avançada processando esse fragmento e um carregador que move o fragmento de uma máquina para outra.

A representação geral de uma transição na especificação da Figura 3-3 é uma das seguintes formas:

name: e[c]/a

name: ([c] for I)/a

onde **name** é o nome da transição (sem significado semântico), **e** é um evento-expressão (conjunção de evento ou negação de evento), **[c]** é uma condição (expressão boolean sobre variáveis compartilhadas), **a** é uma ação que pode gerar eventos ou associar valores às variáveis compartilhadas quando a transição é analisada, e **I** é um intervalo de tempo. Todos os componentes

do rótulo da transição são opcionais.

Figura 3-3: Especificação de duas Máquinas com um Buffer

A representação textual deste Statechart usando a linguagem STATEXTE seria:

spec :: [M1 || Buffer || M2]

M_1 , $Buffer$, and M_2 são processos definidos como segue:

$$M_1 :: \left[\begin{array}{l} \text{loop forever do} \\ \quad \text{Livre: skip} \\ \quad \text{inicia : delay } [l_3, u_3] \\ \quad \text{Ocupado : } \left[\begin{array}{l} \text{skip; bloco-ruim : delay } [l_1, u_1] \\ \quad \sqcup \\ \text{skip; bloco-bom:delay } [l_2, u_2]; \text{colocar !} \end{array} \right] \end{array} \right]$$

$$Buffer :: \left[\begin{array}{l} \text{loop forever do} \\ \quad @1: \text{await colocar; primeiro!} \\ \quad \left[\begin{array}{l} m2: \text{delay } [l_4, u_4]; \text{pronto!} \\ @2: \text{await pegar!} \\ m1: \text{delay } [l_5, u_5]; \text{pronto!} \\ m3: \text{idle} \end{array} \right] \\ \quad \cup \\ \quad \left[\begin{array}{l} \text{await colocar} \wedge \neg \text{primeiro; Erro: idle} \\ \quad \sqcup \\ \text{await saída} \end{array} \right] \end{array} \right]$$

O processo M_2 é dado por:

$$M_2 :: \left[\begin{array}{l} \text{loop forever do} \\ \quad \left[\begin{array}{l} \text{Livre: await pronto; pegar !} \\ \text{Ocupado: delay } [l_6, u_6] \end{array} \right] \end{array} \right]$$

3.3. Algoritmos para Geração de Código

Várias ferramentas e métodos têm sido propostos na literatura para obter uma especificação mais

eficiente de sistemas de tempo real. Representações gráficas e várias formas de simulação têm ajudado na detecção de erros e na diminuição dos mesmos. Juntamente com os editores e simuladores, os geradores de código, além de auxiliarem na simulação de especificações, ajudam também na obtenção do código final. Dessa forma, importantes métodos têm surgido apresentando essa característica; alguns deles são descritos nesta seção.

3.3.1. Especificação e Síntese de Hardware

Drusinsky e Harel propõem a utilização de Statecharts como uma linguagem de descrição de hardware e mostram como essa especificação pode ser implementada usando PLAs (Programmable Logic Array).

Figura 3-4: Um Exemplo de Statechart

A idéia básica do modelo é tratar cada estado não atômico de um Statechart como uma máquina simples implementando uma MEF. Essas seriam interconectadas como uma árvore de máquinas, implementando um Statechart. A execução seria concorrente através da comunicação entre elas e alguns eventos seriam criados para representar as conexões entre os estados.

Figura 3-5: As Sete Máquinas de Estados Finitos

Dessa forma, o Statechart da Figura 3-4, apresenta 15 pequenas máquinas implementando as 7 máquinas de estados finitos da Figura 3-5 e mais oito máquinas para os estados atômicos. Os eventos que são enviados na representação da Figura 3-5 têm as seguintes características:

- sinal **entered X**: é criado por uma máquina de mais alto nível na hierarquia, dizendo para entrar no estado X, ou seja, inicia os estados filhos de X até chegar no estados básicos default;
- sinal **left X**: notifica a máquina de nível mais baixo, para entrar em seu estado *Idle*; ou seja, envia o sinal de saída direta do estado X;
- sinal **leave X**: é criado por um estado de nível mais baixo, para notificar seus predecessores sobre seu término; ou seja, notifica a saída do estado X apenas. Por exemplo, A1 notifica C Figura 3-5 (c) pelo sinal leave A1_a;
- sinal **enter X**: é criado por um estado de nível mais alto quando um de seus subestados (que não é o default) é requisitado para começar a operar no estado X. Por exemplo C notifica A2 na Figura 3-5(a).

A comunicação entre os estados é ilustrada na Figura 3-6 representando o esquema de conexão do Statechart com as MEF apresentadas na Figura 3-5. Note que os eventos são recebidos sempre pelos estados de nível maior, nunca pelos atômicos e utilizam a representação para cada máquina individual como um máquina de Moore.

Figura 3-6: Esquema de Conexão dos Processos

Com isso, o esquema de implementação baseado em Statechart apresentado nesse artigo, oferece vantagens, incluindo redução de área ocupada da PLA e esquema de implementação natural de concorrência e sincronização.

3.3.2. Método de Rumbaugh

Figura 3-7: Rede de Caixas Eletrônicos (CE)

Considere uma rede de Caixas Eletrônicos (CE) como mostra a Figura 3-7. O exemplo apresenta o modelo de um software para apoiar uma rede de bancos computadorizada, incluindo caixas humanos e caixas eletrônicos (CE) para serem compartilhados por um consórcio de bancos. Cada banco fornece seu próprio computador para manter seus próprios clientes (contas) e processar suas transações. Cada banco possui um posto de caixa que faz comunicação direta com o seu próprio computador. Caixas humanas digitam a conta e a data da transação. Caixas eletrônicas comunicam-se com um computador central, o qual libera a transação com o banco apropriado. Um caixa eletrônico aceita um cartão, interage com o usuário, comunica-se com o sistema central para completar a transação, entrega o dinheiro e imprime recibos. O sistema requer armazenamento apropriado de registro e garantia de segurança. O sistema deve tratar corretamente acessos concorrentes para a mesma conta. Os bancos fornecerão seus próprios softwares para seus próprios

computadores; deve-se projetar o software para os CE e a rede. O custo para o sistema compartilhado será distribuído entre os bancos, de acordo com o número de clientes com cartão.

Figura 3-8: Diagrama de fluxo de eventos para o exemplo CE

Dada as características do caixa eletrônico (CE) o primeiro passo da análise de requisitos é a construção de um modelo de objeto seguindo os seguintes passos: identificação dos objetos das classes, preparação do dicionário de dados, identificação das associações entre objetos, identificação de atributos de objetos e ligações, organização e simplificação de classes de objetos usando herança, refinamento do modelo e agrupamento de classes dentro dos módulos.

Depois de definidas as classes e os objetos, o comportamento do sistema deve ser representado considerando-se os aspectos dependentes do tempo para identificar os eventos entre objetos e construir o diagrama de estados. Para este exemplo, quatro classes são definidas, como mostra a

- normalmente esperada. Escrever o nome dos estados desse caminho como uma sequência linear. Isto representa a sequência de declarações no programa;
2. Identificar caminhos alternativos, ramificações laterais do caminho principal, e reuni-los mais tarde. Estas tornam-se declarações condicionais do programa;
 3. Identificar caminhos inversos da ramificação lateral do laço principal e reuni-los antes. Estes tornam-se laços no programa. Se há múltiplos caminhos inversos que não se cruzam, eles tornam-se laços aninhados no programa. Caminhos inversos que se cruzam não se aninham e não podem ser implementados com goto's se todos falharem, mas isso é raro.
 4. Os estados e transições que permanecem correspondem às condições de exceção. Eles podem ser manuseados por várias técnicas, incluindo: subrotinas de erro, erros suportados pela linguagem, ou iniciando e testando o status dos flags.

Pseudo-código

```

do forever
  display janela principal
  read cartão
  repeat
    pergunte pela password
    verifica conta

  until verificação da conta = ok
  repeat
    pergunte o tipo de transação
    read tipo
    pergunte a quantia
    começa transação
    espere completar transação

  until transação = ok
  entregue cartão
  espere o cliente pegá-lo
  pergunte se quer continuar

until usuário = não continuar
ejetar cartão
espere o cliente pegar o cartão
  
```

Figura 3-10: Controle do CE

Para o exemplo apresentado anteriormente, o modelo de estado e o pseudo-código derivado dele é mostrado na Figura 3-10. Primeiro identificou-se o controle do caminho principal, o qual corresponde à leitura do cartão, indagando o usuário sobre a informação da transação, processando a transação, imprimindo o recibo e ejetando o cartão. Fluxos alternativos de controle ocorrem se o cliente quer processar mais que uma transação ou se a senha é inválida e pergunta-se ao cliente se ele quer tentar novamente. O evento *cancel* poderia ser incluído no fluxo de controle e implementado como um código de exceção *goto*.

3.3.3. Outros Algoritmos

Outros algoritmos também foram propostos para a conversão automática de Statecharts em código. Dentre eles, dois estão embutidos nas ferramentas desenvolvidas por Figueiredo [FIL91] e [MEI91b].

A ferramenta apresentada em [FIL91] é um gerador de programas que implementa o controle de sistemas reativos complexos. A sua entrada é uma descrição textual de um Statechart (diagramas de estado) e produz como saída um programa funcionalmente equivalente em C.

A ferramenta apresentada em [MEI91b] é um gerador de aplicação para sistemas reativos que implementa a especificação de sistemas utilizando Statecharts. Estruturas de dados, rotinas de manipulação, etc, podem ser geradas automaticamente, em diferentes linguagens de programação, a partir de uma especificação de alto nível. O gerador recebe como entrada uma especificação em Statechart de um sistema reativo e produz como saída um produto relacionado com o sistema especificado. Um possível produto a ser gerado é o programa que simula o sistema, realizando mudanças de estado causadas pelos eventos e executando as ações e atividades associadas a esse evento.

O algoritmo usado em ambas as ferramentas é essencialmente o mesmo: há um laço externo que obtém os eventos do ambiente e, usando um comando CASE, escolhe os estados ativos. Dentro de cada estado há um outro CASE para escolher a transição a ser efetuada e assim por diante. Esses algoritmos não são muito eficiente porque produzem um programa muito extenso, que fica continuamente ativo na memória.

Além dessas uma outra ferramenta que auxilia na geração de código automático é o TALISMAN, que é uma ferramenta voltada para sistemas em que o tempo não é relevante e que permite especificar, projetar e implementar software em geral [AMB92]. TALISMAN usa a estratégia de linearizar diagramas de estruturas de função (tipo diagramas de estrutura modular).

O STATEMATE descrito anteriormente também possui um gerador de código automático eficiente, mas não é de domínio público e suas características não estão disponíveis para serem analisadas.

3.4. Avaliação das Alternativas

Nas seções 3.2 e 3.3 foram apresentados alguns métodos, linguagens e ferramentas utilizadas para especificação de sistemas reativos, suas principais características e uma avaliação das diferentes formas de geração de código é apresentado agora. As diferenças são significativas e a forma como eles são representados varia muito.

Pôde-se observar que os métodos (DARTS, Esquema de Transformação, e o Método de Rumbaugh) apresentados não são formais, possuindo um conjunto de regras para definir a especificação.

Essas especificações exigem uma transformação do diagrama em código fonte que depende fortemente do conhecimento e experiência do projetista/programador. Logo, a geração de código também torna-se complexa.

As linguagens apresentadas são formais e permitem a representação das especificações de forma genérica. Elas são específicas para a representação de sistemas de tempo real, permitindo sincronização, comunicação e concorrência, que são aspectos importantes e tornam a geração de código menos complexa e mais imediata. A linguagem ESTEREL já possui um gerador de código que gera programas em forma de autômatos; mas a linguagem STATEXT possui apenas a especificação, e ainda não foi implementada.

Finalmente, as ferramentas mostram que o processo de geração de código é possível, desde que a especificação tenha uma representação formal e bem definida. STATEMATE, por exemplo, é

baseado em Statecharts, que possui semântica e sintaxe bem definida e através dessa formalidade permite gerar código. Existem outras propostas como as do método de Rumbaugh, de Meira e de Figueiredo, todas baseados em Statecharts e que apresentam formas simples para geração de código. Já o método de Drusinsky propõe a decomposição dos Statecharts em máquinas de estados finitos, sendo um método mais complexo mas muito eficiente tanto em velocidade como em área ocupada pela PLA [DRU89].

Dentre as alternativas estudadas pode-se apresentar um esquema de possíveis formas de geração de código como mostra a Figura 3-11.

Figura 3-11: Avaliação dos Possíveis Métodos para Geração de Código

Partindo-se do princípio que Statecharts é uma especificação formal e muito bem definida, são propostas quatro possibilidades para se desenvolver o gerador de código automático a partir das alternativas estudadas anteriormente. Analisando a Figura 3-11 tem-se:

- *específico*: seria a criação de um método para geração de código usando uma linguagem

qualquer como C, Fortran, etc. Nesse caso, cada diagrama de estados seria implementado diretamente a partir da sua representação visual sem ter regras de transformação genéricas, isto é, cada implementação é um novo caso. Dessa forma, cada programa gerado não teria necessariamente o mesmo tipo de configuração e ainda, perder-se-ia tempo em seu desenvolvimento.

- *genérico*: os geradores existentes apresentam uma forma simples de analisar e desenvolver o Statechart. Métodos como os de Drusinsky, Figueiredo e Meira possuem essa característica, apresentando regras de transformação e facilitando a geração de código. O método de Rumbaugh é bastante simples e sem base formal para uma transformação automática.

- *linguagens intermediárias*

STATEXT: a geração de código utilizando a linguagem STATEXT passaria por dois passos. O primeiro seria muito simples, quase direto, consistindo de transformações dos Statecharts na linguagem STATEXT (representação textual de Statecharts). Já o segundo processo é mais complexo, pois ter-se-ia que desenvolver um compilador (ou interpretador) para gerar código a partir da linguagem STATEXT e dessa forma o problema da alternativa direta persistiria.

ESTEREL: como a geração de código utilizando a linguagem STATEXT, utilizando a linguagem ESTEREL, a geração poderia ser feita em dois passos. O primeiro é complexo, pois a transformação de Statecharts para ESTEREL pode ser tão complexa quanto a tradução direta para um programa fonte. Mas, se esta transformação fosse feita, o segundo passo seria simples, uma vez que já existe um gerador de código ESTEREL.

Assim conclui-se que o processo de geração de código desejado para o ambiente StatSim pode ser implementado de várias maneiras, excluída a forma específica, que não é genérica. Analisando-se as propostas possíveis, o método mais eficiente é aquele apresentado por Drusinsky, por basear-se em Statechart (que é formal), ser muito eficiente e por ser uma transformação única, sem passar por linguagens intermediárias.

3.5. Considerações Finais

Neste capítulo foram discutidas diversas formas de implementação de um gerador de código baseado em Statecharts, abordando métodos e ambientes de especificação de sistemas reativos. Pôde-se avaliar, através das abordagens apresentadas, as possíveis alternativas para a geração de código

dentro do ambiente StatSim. Os métodos estudados proporcionaram uma análise objetiva, e permitiram a análise e comparação de diversas formas de geração.

A partir dos estudos efetuados foram desenvolvidos os protótipos para Occam e C++ apresentados no Capítulo 4 e o Gerador de Código como ferramenta de apoio ao HySCharts apresentado no Capítulo 5.

Capítulo 4

4. Gerador de Código para Linguagens Procedurais

4.1. Considerações Iniciais

Drusinsky e Harel propõem, em [DRU89], os Statecharts para a especificação de hardware e mostram como o modelo criado pode ser implementado como uma PLA (Programmable Logic Array). Esse algoritmo propõe a decomposição dos Statecharts em máquinas de estados finitos convencionais, dispostas hierarquicamente e comunicando-se entre si, de tal forma que possam ser executadas concorrentemente, obtendo-se então grande eficiência tanto em velocidade de execução quanto em área ocupada pela PLA, conforme descrito no capítulo anterior. Esse algoritmo implementa muitos dos aspectos básicos dos Statecharts, mas deixa de lado algumas características mais delicadas, como o uso de variáveis e o tratamento de inconsistências estruturais. Este algoritmo será implementado aqui, adaptado às características do software a ser desenvolvido.

A partir do algoritmo proposto por Drusinsky [DRU89] foi projetada a implementação do Gerador de Código pretendido para o ambiente StatSim. A descrição proposta é apresentada no item 3.2, e a seguir, nas seções 3.3 e 3.4 são apresentados os dois protótipos do código a ser gerado pela ferramenta, desenvolvidos em OCCAM e C++, respectivamente. É apresentada também uma análise dos resultados obtidos com essas implementações na seção 3.5.

4.2. Construção de um gerador de código a partir do ambiente STATSIM

A partir do ambiente gráfico do StatSim, pretende-se agora juntar o núcleo reativo especificado como

um Statechart e a semântica das aplicações, definidas em diagramas de atividades, de forma a ter um protótipo executável. Para isso, o gerador de aplicações está sendo usado como ferramenta básica. Observe-se também que as ações semânticas podem ser omitidas e têm-se então apenas a simulação do núcleo reativo do sistema.

Figura 4-1: Módulo de Geração de Código do StatSim

A Figura 4-1 mostra a arquitetura parcial desse módulo de geração de código. A partir de um Statechart armazenado na base de dados do StatSim o Gerador de Código gera um programa em Occam ou C++ que executa o Statechart, de acordo com um determinado algoritmo cujas idéias básicas são delineadas neste Capítulo. Caso as ações/atividades estejam também disponíveis como código em Occam ou C++, o gerador as integra ao núcleo reativo, criando um protótipo do sistema.

A visão inicial desenvolvida em Occam visava a utilização de multi-processadores e a execução concorrente para que fosse possível visualizar a execução dos módulos projetados no ambiente

StatSim em tempo real e observar os resultados obtidos. Devido a pouca disseminação de recursos envolvendo transputers e a linguagem de programação ficar restrita a este tipo de equipamento foi desenvolvido um outro protótipo que pudesse ser utilizado em outros ambientes de trabalho.

Visto que o ambiente gráfico do StatSim foi desenvolvido em C, e a ferramenta é executável em estações de trabalho Sun, elaborou-se um projeto para que o sistema pudesse visualizar os resultados obtidos juntamente com a execução da ferramenta, ou seja, permitir que os programas gerados pudessem ser testados sem a necessidade de um equipamento específico como era o caso descrito utilizando-se a linguagem Occam.

Para isso, foram observados os resultados obtidos inicialmente em Occam e foi desenvolvido um novo protótipo utilizando nesta etapa a linguagem C++. Dessa forma, apresentamos aqui os dois protótipos desenvolvidos, um utilizando a linguagem Occam e o outro utilizando-se a linguagem C++.

4.3. Desenvolvimento de um protótipo para o Gerador de Códigos em Occam

Um protótipo de um Gerador de Código baseado naquele apresentado em [DRU89] para síntese de hardware, foi desenvolvido para a execução lógica eficiente de Statecharts.

Problemas de utilização deste protótipo foram encontrados visto que a execução estava restrita a máquinas com transputers e compiladores Occam. Apresentaremos aqui o protótipo desenvolvido e suas principais características.

4.3.1. Construção de um gerador de código a partir do ambiente STATSIM

A idéia básica do algoritmo, visando obter maior eficiência na sua execução, é decompor o Statechart em um conjunto de processos (procedimentos Occam), executando concorrentemente, onde cada processo representa um estado. Estes processos serão denominados processos-estado (PE). Um PE especial, PE-H, é criado para a representação de história. O gerador analisa primeiramente os estados básicos do Statechart a ser executado e posteriormente os níveis acima da hierarquia, até atingir o estado de mais alto nível.

Para completar o programa executável em Occam, é necessária a criação de alguns processos auxiliares (PA) e um processo escalonador que ative para execução em paralelo, ambos os conjuntos: de processos auxiliares e de processos-estado, conforme pode ser visto na arquitetura mostrada na Figura 4-2. A comunicação entre os processos é feita através de canais lógicos que são criados automaticamente pelo gerador para cada processo PA e PE criado.

Figura 4-2: Arquitetura do Código Gerado

São cinco os processos auxiliares gerados: PA-OE, que obtém os eventos de entrada da interface, valida-os e os despacha para os PEs correspondentes; PA-ATV que controla a ativação dos estados; PA-DSTV, que controla a desativação de estados; PA-ORT, que controla a entrada de eventos em componentes ortogonais; e, PA-ES, que emite todas as mensagens de entrada e saída do sistema, como por exemplo: mensagem de erro, de comunicação com os usuários, leitura de dados, etc. Para ativar e desativar estados, PA-ATV e PA-DSTV compartilham um vetor de estados que contém uma posição para cada super-estado, excluindo-se os ortogonais, mapeando cada estado em seu componente ativo.

4.3.2. Organização e Interface dos Processos Gerados

A Figura 4-3 mostra um processo genérico em Occam, correspondente a um processo-estado, com todos os canais de ligação criados pelo gerador. Cada canal é descrito com detalhes na Tabela 4.1. A coluna “Ligações” indica que tipos de processos o canal interliga. Por exemplo, PA → PE indica que o canal é utilizado por um processo auxiliar para

enviar mensagens para um processo-estado.

Figura 4-3: Representação Geral em Occam de um Processo

Nome	Descrição	Ligações
CnEst	recebe e envia eventos externos. São definidos vários canais, cada um deles representando um estado definido no Statechart.	Interface → PA PA → PE PE ↔ PE
CnErro	envia mensagens de erro para o usuário. Entre estes erros tem-se: disparo de eventos não habilitados, estado não habilitado e eventos inexistentes.	PE → impr
CnDisp	envia informações para o processo auxiliar para enviar eventos disparados pelo usuário através dos canais CnEst. É utilizado somente em componentes OR.	PE ↔ PA
CnMsg	envia dados para serem apresentados na tela. São informações a respeito do estado ativo naquele determinado instante.	PE → impr
CnCtl	utilizado para controlar ativação de estados. O controle é realizado por meio de um vetor que contém as informações sobre os estados ativos.	PE → PA
CnAt	ativa outros estados. Tem a mesma função que o canal CnDisp, mas o disparo de eventos ocorre independente do PA, isso é, o canal CnEst dispara os eventos.	PE ↔ PE
CnDes	controla a desativação de estados. O controle é realizado por meio do mesmo vetor manipulado por CnCtl.	PE → PA
CnHist	controla entradas e saídas de um estado, por história.	PE → proc. história

Tabela 4-1: Descrição dos Canais e suas ligações

4.3.3. Semântica do Processamento dos Processos-Estado

Cada PE implementa a máquina de estados finitos correspondente a um estado do Statechart. Os processos correspondentes aos estados básicos do Statechart são compostos de apenas dois estados: idle e ativo, sendo que no estado ativo, a atividade correspondente é executada (Figura 4-4). Além disso, cada PE comunica-se com seu ancestral ou seus descendentes, para indicar a ativação ou desativação de estados, de acordo com a semântica dos Statecharts.

Figura 4-4: Especificação Genérica de um PE Básico.

O gerador produz um gabarito para PE da seguinte forma:

```

PROC <estado> (CHAN OF INT Cn<estado>, ... )
  INT x:
  SEQ
    x:=0
    while x <> EVQUIT
      SEQ
        Cn<estado> ? x                                     (I)
        IF
          condição 1
            Trecho a ser gerado para cada
              caso da Figura 4-4
            .
            .
            .
          condição 1
  
```

Trecho a ser gerado para cada
caso da Figura 4-4

```
TRUE
SEQ
  CnMsg ! EVINV
  CnDisp ! Ok
```

Este gabarito mostra o código que implementa genericamente uma MEF tal como aquelas da Figura 4-7. Mostra-se que há iteração e que o processo espera em seu estado *Idle* por um evento (I) e, dependendo de qual evento é recebido, uma certa transição é executada. Caso o evento não seja válido retorna-se uma mensagem de erro e o processo PA-ES é informado de que novo evento pode ser processado.

Cada trecho do gabarito geral é produzido pelo gerador a partir das construções dos Statecharts cujos principais casos são mostrados na Figura 4-5. A seguir, com base nesses casos, descreve-se as decisões a serem tomadas pelo gerador.

Transição Simples (3.5a): Analisa-se o estado origem desativando-o (se estiver ativo) através do canal CnDes e ativando-se o estado destino, através do canal CnCtl. Isso dá origem ao seguinte gabarito:

```
x = <evento>
SEQ
  CnCtl ! <estado destino>; <estado destino>.pos
  CnMsg ! <estado destino>
  CnDisp ! Ok
```

ou seja, se o evento recebido em x é aquele que determina a transição simples, uma mensagem é enviada pelo canal CnCtl atualizando a posição do vetor correspondente a esse estado que deve ser ativado; uma informação é enviada pelo canal CnMsg registrando a ativação do estado destino; e, finalmente, pelo canal CnDisp o processo volta ao seu estado *Idle* esperando outras informações serem recebidas.

Figura 4-5: Casos Considerados na Geração de Código para os PEs.

Transição de/para Super-estado (3.5c): a saída de um super-estado OR é avaliada a partir de seu estado básico. Esse estado sente primeiramente o evento externo, processa-o, e em seguida envia-o para o seu ancestral. A saída de um super-estado AND é realizada de forma similar à componente OR, exceto que a avaliação é feita a partir de cada estado básico de cada componente.

A entrada em um super-estado OR é realizada ativando-se esse estado e todos os seus estados filhos até chegar aos estados básicos. Isso é feito enviando-se a mensagem *def* por um dos canais *CnEst*. A entrada em um super-estado AND é realizada de forma similar à componente OR, mas estendida a todos os seus componentes.

O caso particular de uma transição de um super-estado para ele mesmo é realizada de forma similar, dependendo do estado (AND/OR) e do destino (estado-pai/estado filho).

Cruzamento de Níveis (3.5e): a saída de super-estado (AND/OR) por um de seus filhos é realizada saindo-se de todos os seus estados imediatamente acima até chegar ao estado de mesmo nível hierárquico do estado pai do destino. Isso é realizado pelo canal *CnDes*, que desativa todos esses estados no vetor de controle de estados ativos.

A entrada em um super-estado OR explicitamente por um de seus filhos é realizada ativando-se o estado pai, em seguida seus filhos até se chegar ao respectivo filho destino. A entrada em um super-estado AND por um de seus filhos é realizada de forma similar à componente OR, mas todas as outras componentes são ativadas por default até chegar em seu estado básico.

Cruzamento de Níveis em Componente Ortogonal (3.5d): a análise é feita da mesma forma que aquela mencionada anteriormente, mas neste caso todas as outras componentes serão ativadas por default enviando-se a mensagem *def* via *CnEst* para os PE correspondentes às componetes ortogonais.

Propagação de Eventos (3.5b) : a sincronização é realizada utilizando-se os canais que ativam outros estado (representados aqui por *CnAt*) que enviam as mensagens ativando os processos de acordo com a ação associada. O evento é disparado pelo canal adequado, ou seja, o canal que ativa o estado destino daquela ação.

Eventos simultâneos podem ser disparados em componentes ortogonais. Isso pode ser feito verificando se os estados origem de cada evento selecionado pelo usuário estão ativos naquele passo. Se a resposta for verdadeira o evento é disparado. A ordem de disparo é aleatória, e vinculada à chegada de mensagens por um dos canais *CnEst* que ativam os processos-estados.

História (3.5f) : é avaliada pelo canal *CnHist*. Um processo é criado para avaliar as entradas e saídas por história, esse processo aguarda que uma informação de saída de um estado chegue, envia o evento de saída para o processo destino e aguarda o retorno desse disparo de evento.

4.3.4. Um exemplo para o algoritmo de execução de Statecharts

Considerando-se como exemplo o Statechart da Figura 4-6, retirado de [DRU89], são necessários quinze PEs, oito dos quais são básicos. Os sete PEs não básicos correspondentes aos super-estados estão especificados pelas sete máquinas de estados finitos da Figura 4-7. Os PEs básicos não aparecem na figura e são especificados como na Figura 4-4. A notação de comunicação entre as máquinas é a seguinte: *<evento>/<MEF> ! <mensagem>*, significando que ao disparar uma certa transição a *mensagem* correspondente é enviada à *MEF* especificada pelo canal CnEst. No exemplo da Figura 4-7, a máquina D envia a mensagem *def* para a máquina B ao transicionar de B para C.

Figura 4-6: Exemplo de Statecharts

Para o Statechart em questão, são necessários 15 canais CnEst devido aos 15 estados existentes, cada canal ativando seu respectivo processo. Para controlar a ativação desses processos-estados

Figura 4-7: As sete MEFs da Figura 4-6

alguns processos monitores foram criados. Um processo monitor é aquele que gerencia canais que estão sendo manipulados concorrentemente, isto é, evita que um canal envie mais de um evento (o que provocaria um deadlock de recepção). Entre estes processos, os mais importantes são aqueles que ativam os estados, e em segundo plano ficam os outros: de controle de disparos simultâneos, de controle de ativação de estados e de controle de desativação de estados.

Os processos que monitoram a ativação de estados passam por duas fases. A primeira, é representada pela coleta de dados pelo usuário e é feita uma verificação para saber se aquele evento existe para ser disparado; a segunda, e a mais importante, é a validação do disparo, ou seja, é nesse momento que é verificado se aquele evento é considerado válido para ser disparado.

Figura 4-8: Esquema de Conexão das Figuras 3.6 e 3.7

A arquitetura dos PEs, ligados pelos canais de comunicação CnEst e o esquema de troca de mensagens entre eles é mostrada na Figura 4-8, para o exemplo utilizado. As setas que aparecem na extremidade inferior de um estado básico especificam os eventos que podem ser recebidos por esse estado. Assim, o estado A11 pode receber os eventos alfa1, beta1 e delta1 conforme exemplo da Tabela 4-2. A comunicação com os demais processos ocorre através da passagem de eventos pelos canais.

O vetor de estados ativos controlado pelos canais CnDes e CnCtl, neste exemplo, é um vetor com 6 posições, cada posição referenciando um estado existente no Statechart, excluindo os estados básicos e o primeiro super-estado ortogonal de cada componente da raiz OR ou AND, ou a própria raiz, se esta for AND.

```

PROC A11(CHAN OF INT CnA11,CnAt,CnErro,CnDisp,
        CHAN OF entrada CnCtl)

  INT x: -- variavel que recebe a informacao de qual evento
        -- foi disparado pelo estado A11

  SEQ
    x:= 0
    WHILE x <> EVQUIT
      SEQ
        CnA11 ? x
        IF
          x = ALFA1
            SEQ
              CnAt ! EstA12
              CnCtl ! EstA12; 2
              CnDisp ! DISPARO
          x = BETA1
            SEQ
              CnA1 ! x
          x = DELTA1
            SEQ
              CnA1 ! x
          x = EVQUIT
            SKIP
          TRUE
            SEQ
              CnErro ! 3
              CnDisp ! DISPARO
:

```

Tabela 4-2: Exemplo de Codificação do Estado A11 em Occam

Outros pontos importantes do algoritmo são referentes ao tratamento de disparo de eventos simultâneos e sincronização. Eventos simultâneos podem ser visualizados através do exemplo da Figura 4-6. Suponha que a configuração ativa seja {D, B, B1, B2, B11, B21} e que os eventos *gama3* e *gama1* foram disparados simultaneamente pelo usuário. O processo auxiliar irá enviar esses eventos simultaneamente para os processos B11 e B21. Considerando apenas B11, este recebe *gama1*, desativa sua atividade, vai para o estado Idle e envia *gama1* para B12 que é ativado. O mesmo ocorre simetricamente para B21. A configuração resultante será {D, B, B1 B2, B12, B22}.

Quando há necessidade de ativar ou desativar super-estados, as mensagens são passadas para os ancestrais. O leitor pode acompanhar por exemplo, partindo da mesma configuração, a trajetória do evento *delta2*, que se propaga para **B1** e **B2**, **B**, **C** e desce por default para os descendentes de **C**.

Figura 4-9: Representação de Sincronização

A sincronização pode ser verificada no exemplo da Figura 4-8. Considerando que os estados **B21** e **B11** estejam ativos, ou seja, que a configuração inicial seja **{B, B1, B2, B11, B21}**, ao disparar o evento *gama3* **B21** transicionará para **B22** e a ação *gama1* associada também será disparada. Esse disparo será sentido pelo processo **B22** nesse mesmo intervalo de tempo, ativando o processo auxiliar. Se **B11** estiver ativo a transição para **B12** ocorrerá, caso contrário uma mensagem de erro será enviada para o usuário.

4.3.5. Uma avaliação preliminar

O algoritmo apresentado aqui, apesar de seguir a mesma filosofia de decomposição do Statechart em uma hierarquia de MEFs concorrentes, difere em alguns aspectos daquele proposto por Harel e Drusinsky em [DRU89]. O primeiro ponto a ser comentado se refere ao processo que recebe os eventos externos. No algoritmo original eles são recebidos em qualquer nível, exatamente na máquina correspondente ao estado cujo filho é origem da transição e na versão apresentada aqui eles são recebidos pelos estados básicos e daí são comunicados para cima até o ancestral comum relativo à transição que está sendo disparada.

Figura 4-10: Sincronização entre Ativação/Desativação

Isso tem a vantagem que eliminam-se possíveis problemas de demora que podem ocorrer na versão de Drusinsky e Harel, quando numa situação como aquela da Figura 4-10 pode-se entrar primeiro no estado destino antes que se tenha saído de toda a hierarquia dos estados fontes ativos. Por outro lado, a ordem no qual as atividades vão sendo desativadas também é bottom-up e isso precisa ser considerado na especificação da aplicação.

Por causa dessa mesma decisão quanto à recepção dos eventos externos, em [DRU89] criam-se eventos internos artificiais (leave, left, enter) com o propósito de comunicar ao ancestral ou aos descendentes a decisão a ser tomada. Na versão apresentada aqui, não há necessidade de se criar novos eventos. Tudo se passa como se o Statechart sofresse uma projeção para o nível mais baixo.

No nível mais baixo isso significa a necessidade de ligações entre os estados básicos (entre B11 e B12, por exemplo), o que não ocorre em [DRU89]. Considerando que a maior parte das transações ocorre nesse nível, tem-se uma maior eficiência à custa de mais ligações. Como neste caso as ligações são lógicas e não físicas essa desvantagem é insignificante. A principal vantagem para adoção deste esquema é tornar mais fácil a geração do código em Occam. Futuramente esta decisão poderá ser parametrizada, deixando a cargo do usuário a decisão se quer gerar o algoritmo original, a variação apresentada neste trabalho ou mesmo outras, em função de suas necessidades e da linguagem alvo.

Na versão atual do gerador não se encontram implementados o tratamento de inconsistências estruturais e o uso de variáveis, assim como em [DRU89]. Na realidade, não há necessidade de se preocupar com inconsistências estruturais, pois elas podem ser verificadas pelo ambiente StatSim, antes da geração de código. Com relação às variáveis é perfeitamente possível fazer esta extensão apesar de ser relativamente complexa, pois as variáveis não podem ser declaradas globalmente em Occam. Seria necessário um processo encarregado de controlar as variáveis (como se fosse um servidor de variáveis), em paralelo com os demais, que seria informado de qualquer alteração ou consulta aos valores das variáveis.

Nesta versão também não há qualquer restrição quanto ao número de subestados que um estado pode ter. No exemplo da Figura 4-6 observe que cada estado tem apenas dois filhos. Isso é conveniente na implementação em hardware para controlar a quantidade de ligações físicas entre as máquinas, mas na versão apresentada aqui os canais são lógicos e são criados de acordo com a necessidade. Se o estado A1, por exemplo, tivesse mais um filho A13 seria necessário ter a ligação entre eles e também de A13 para com os irmão com os quais tivesse ligação através de transições.

4.4. Desenvolvimento de um protótipo para o Gerador de Códigos em C++

Apresentaremos o protótipo desenvolvido em C++ com as características básicas implementadas.

Para este protótipo, como no exemplo descrito na seção 4.3 foi escolhido o exemplo apresentado por Drusinsky em [DRU89]. Segundo o algoritmo de Drusinsky e Harel o Statechart deve ser decomposto em um conjunto de máquinas de estados finito executando concorrentemente entre si. A decomposição em máquinas de estados finito pode ser facilmente visualizada considerando o Statechart como uma classe e um estado finito como um objeto desta classe, mas a concorrência fica difícil de visualizar considerando que será utilizado uma linguagem como C++, que não é concorrente. A partir disso serão apresentadas aqui as principais considerações sobre alguns aspectos analisados para o reconhecimento do Statechart.

A idéia básica do algoritmo é decompor os Statecharts em um conjunto de objetos. Cada objeto com um conjunto de características próprias diferenciando-se e comunicando-se dependendo do tipo de

estado a que ele pertença. Como objetos de tipos de estados diferentes (AND, OR, atômico) podem ter informações em comum, será utilizado aqui o conceitos de HERANÇA e POLIMORFISMO.

4.4.1. Estruturas de Dados

O primeiro passo para definir o modelo dos dados é definir as classes e os métodos pertencentes às classes. Neste caso será necessário a existência de uma classe pai com características comuns de todos estados para armazenagem de dados, e classes específicas representando cada tipo de estado que pode existir em um Statechart.

Figura 4-11: Casos Considerados na Geração de Código

Após definidas as classes é necessário definir um conjunto de regras para que os objetos possam se comunicar. Os objetos receberão informações da base de dados do StatSim e serão considerados da seguinte forma:

- o conjunto de informações do estado deve ser passado para o **objeto** representando esse **estado**, através de nome do estado, nível hierárquico, história, etc;
- os eventos associados a transições pertencerão ao estado origem da transição;
- quando uma transição simples ocorre, o estado origem é desativado e o destino é imediatamente ativado;
- quando uma transição de/para super-estado ocorre, o primeiro objeto a ser avaliado e sentir que uma transição ocorreu é o “objeto atômico” (do estado atômico) que vai repassando a desativação até chegar ao estado pai. Quando o objeto pai é desativado o próximo estado é ativado, e, se ele não for atômico o estado destino é ativado por default, disparado sucessivamente o evento DEFAULT até chegar a seu estado atômico;
- quando uma transição implica em cruzamento de níveis de um estado para outro é considerada a saída de um estado da mesma forma que no item anterior e a ativação para o próximo estado é efetuada disparando o evento da transição em questão sucessivamente, até chegar ao estado atômico do destino;
- para os eventos que possuem ação associada, estas serão disparadas após o ciclo de disparo do evento ser completado e não simultaneamente;
- para entrar em um estado de nível hierárquico diferente deve existir na base de dados do protótipo uma estrutura fornecendo todos os estados que devem ser desativados quando aquele evento ocorrer.
- em estados que devem ser desativados ou ativados por história deve ser mantida uma estrutura de dados contendo informações da última visita. Estas informações devem ser mantidas junto ao “objeto” que possui esta característica. A lista será avaliada quando este “objeto” for ativado novamente.

4.4.2. Construção de classes representando estados

Figura 4-12: Descrição do Problema

Figura 4-13: Descrição da Classe Pai e suas Derivadas

Considerando que a base de dados fornecida pelo Statechart foi analisada e não possui erros, basta que seja capturada e transformada em código fonte pelo gerador. Avaliando os dados e os Statecharts foram definidas quatro classes para o protótipo, são elas: ESTADO, ATOMICO, NATOMICO, ORTOG. Embora um estado não atômico possa ser do tipo OR ou AND, para a nomenclatura deste trabalho foi considerado que o estado OR na definição da classe seria NATOMICO. As três últimas classes representam o estado atômico, o estado OR e o estado AND,

respectivamente. A classe ESTADO é a principal e é a classe pai das outras três.

Como representação gráfica de como pode ser possível visualizar o processo PRODUTO apresentam-se na Figura 4-13 as classes definidas no protótipo e como elas se interrelacionam.

Os componentes básicos de um Statechart são os estados, que podem ser embutidos em superestados criando uma hierarquia de estados. Dessa forma as informações comuns destinadas aos estados foram armazenadas na classe ESTADO, enquanto suas especializações foram separadas em suas classes derivadas.

Nome	Tipo	Descrição
ESTADO	classe	Conjunto de informações que são lidas da base de dados e passadas para o protótipo
ATOMICO	classe	Tipos de informações específicas sobre os estados atômicos e as formas como eles podem mudar de estado (transicionar).
NATOMICO	classe	Informações para transição de dados e entrada por história
ORTOG	classe	Informações para entrada de dados de forma diferenciada (componentes ortogonais), transição de estados
nome_estado	atributo	nome do estado, seja OR, AND ou atômico
trans	atributo	dados das transições associadas a um dado objeto. Atributo multivalorado, estrutura.
trans.evento	atributo	nome do evento associada a transição
trans.prox_estado	atributo	dado o evento “trans.evento” o próximo estado relacionado com este evento
trans.tipo	atributo	classificação do proximo estado em AND, OR ou atômico.
trans.acao	atributo	ação associada ao evento “trans.evento”
saida	atributo	dados sobre objetos que devem ser desativados quando uma transição ocorre e outro “objeto” deve ser ativado. Atributo multivalorado
saida.est	atributo	estado que deve ser abandonado
saida.tipoest	atributo	classificação do estado que deve ser abandonado. AND, OR ou atômico
ativo	atributo	verifica se um estado esta ativo=1 ou nao=0
nev	atributo	número de eventos que um estado possui, de acordo com o numero de transições
nest	atributo	número de estados que devem ser abandonados ao entrar em um objeto

ehist	atributo	estado que deve ser entrado por história
thist	atributo	classificação do estado que deve ser entrado por história em AND, OR ou atômico
nhist	atributo	número de estado que devem ser entrados por história caso seja um superestado
nivel	atributo	nível hierarquico do estado no Statechart
pai	atributo	estado pai do objeto
classepai	atributo	análise da classe do estado pai em AND, OR ou atômico
trans_and	atributo	dados das transições associadas a um dado objeto. em um superestado ortogonal. Atributo multivalorado, estrutura. Possui as mesmas variáveis que a estrutura "trans", mas as variáveis aqui são matrizes.
nro_comp	atributo	número de componentes ortogonais de um dado objeto
hist	atributo	informação sobre história (-1= não possui história e 1 = possui história)
nevl	atributo	contador de número de eventos existente em cada componente ortogonal

Tabela 4-3: Dicionário de dados do Modelo de Objetos

Figura 4-14: As Classes e seus Atributos

Foram avaliadas alguns recursos de como podemos considerar um determinado estado em uma visão orientada a objetos. Aqui os estados são objetos, as transições são métodos, e as informações de entrada e saída de um determinado objeto representam a base de dados de um objeto. As classes e seus atributos foram descritas na tabela 1, descreveremos agora os métodos associados a estas

classes.

Figura 4-15: Operação: Gerar Código

4.4.3. Um exemplo de implementação

Considerando o exemplo dado em [DRU89] e apresentado na Figura 4-16, há 15 objetos, sendo 8 atômicos, 4 não atômico (OR) e 3 objetos ortogonais, um estado pai e duas componentes. Uma lista de dados relacionando objetos e transições deve ser fornecida para a base de dados do objeto. Neste exemplo temos 16 tipos de eventos (considerando o DEFAULT) que devem ser avaliados pelos objetos.

Módulos para verificação e análise de disparo de eventos efetuados pelo usuários são criados por rotinas definidas em bibliotecas. Estas rotinas são responsáveis pela comunicação entre objetos.

Como foi dito anteriormente, os eventos associados a transições pertencem ao estado origem da transição, dessa forma, de acordo com o exemplo da Figura 4-16 o objeto B21, por exemplo, armazenaria informações sobre as transições *gama3* e *delta2*, A21 armazenaria informações sobre as transições *alfa3*, *beta2* e *beta1*, e assim por diante. No exemplo do objeto A21 temos duas transições eventos *beta2* e *beta1* que não são transições que tem origem em A21, mas foram definidas em A21. Isso se deve ao fato do objeto ser um estado atômico e qualquer transição ser inicialmente sentida

(observada) no estado atômico e só depois repassada para os estados imediatamente acima até chegar no estado que pode realmente disparar aquele evento. Embora o estado atômico não possa transicionar para um próximo estado ele pode reconhecer aquele evento como sendo válido.

Figura 4-16: Exemplo de um Statechart

4.5. Considerações Finais

O algoritmo apresentado na seção 4.3 foi primeiramente codificado manualmente em Occam e o programa criado foi testado de forma a se ter segurança quanto à sua correção em relação ao Statechart (isto é, a especificação) original. Depois foram feitos vários estudos, principalmente em relação às possíveis variações de comunicação entre os processos. Com base nesses estudos delineou-se o algoritmo de geração de código do Statechart para Occam.

Já o protótipo desenvolvido em C++ permitiu a visualização dos processo através da definição das classes. A linearização fez com que os processos concorrentes e, consequentemente, a execução em

tempo real fossem prejudicados, uma vez que estas características inerentes a Occam não são intrínsecas da linguagem C++. Se os sistemas a serem desenvolvidos primam pela utilização destas características o código gerado não será adequado para os testes da modelagem do projeto.

Assim, os protótipos definidos neste capítulo serviram como base para o desenvolvimento do Gerador de Código, mas não foi implementado nenhuma ferramenta para a codificação destes protótipos neste ponto. Por outro lado, os dados obtidos aqui representaram a visão na qual o Gerador poderia ser implementado e uma nova linguagem alvo foi definida, a linguagem HTML, que é apresentada no capítulo seguinte.

Capítulo 5

5. Geração HTML com base no HMBS

5.1. Considerações Iniciais

O Gerador de Códigos baseado no HMBS pode ser implementado segundo as características intrínsecas dos Statecharts, ou de acordo com os dados fornecidos pelo usuário com base no HySCharts como hiperdocumentos de navegação.

No primeiro caso tem-se a utilização dos dados fornecidos pelos modelos implementados sobre a ferramenta e a execução conforme a simulação interativa implementada no HySCharts; no segundo caso tem-se a execução através da navegação das home-pages individualmente, conforme especificado no HySCharts pelo usuário.

As duas visões são apresentadas neste capítulo da seguinte forma: a seção 5.2 apresenta a arquitetura do gerador de códigos, as seções 5.3 e 5.4 apresentam de forma detalhada o projeto e a especificação e, finalmente, na seção 5.5 é apresentado um exemplo de execução da ferramenta e na seção 5.6 são feitas as considerações finais sobre a implementação.

5.2. Arquitetura do Gerador de Códigos

A característica principal da ferramenta apresentada é agrupar informações definidas no modelo de dados HMBS (Hyperdocument Model Based Statechart) representado pela ferramenta HySChart e desenvolver uma ferramenta de trabalho que permita ao usuário validar a sua modelagem através de um protótipo do código a ser gerado a partir de suas especificações. As informações de páginas e textos definidos no HMBS deverão gerar código HTML para serem visíveis, a priori, em qualquer

navegador existente, como por exemplo o NETSCAPE e o EXPLORER.

O código HTML gerado é composto de vários comando que são interpretados pelo navegador e são apresentados de acordo com a característica específica de cada um. Um hiperdocumento básico, utilizando-se a linguagem HTML, é composto de cabeçalho, texto, video, sons e imagens. A estrutura básica de um documento HTML apresenta os componentes apresentados a seguir.

```
<HTML>
<HEAD> <TITLE>cabecalho</TITLE></HEAD>
<BODY>
texto
video
som
imagem
</BODY>
</HTML>
```

A Geração de Código pode ser dividida de acordo com sua organização estrutural em três etapas. Na primeira etapa o Gerador cria as estruturas de conversão das páginas de textos definidos no HySCharts em Linguagem HTML. Os ambientes pré-definidos na ferramentas e as peculiaridades da linguagem alvo são abordados e um analisador léxico é implementado. Nessa fase são questionados também as âncoras definidas e as associações às suas configurações destino (um pseudo analisador sintático). Na segunda etapa são desenvolvidos formas navegacionais que permitem ao usuário validar os resultados obtidos. E a terceira etapa representa o armazenamento das informações para serem manipulados pelos desenvolvedores. Um vez validados, a Geração de Código está completada e o aplicativo final da especificação implementado. A ferramenta desenvolvida é denominada então HCGBH (*Hípertext Code Generator Based on HMBS*), e as três etapas descritas aqui serão detalhadas na seção seguinte.

A obtenção e implementação dessas três etapas estruturais requer que um conjunto de informações seja coletado formando a estrutura básica do HCGBH, essas informações são formadas pela: estrutura de Statecharts implementado no StatSim, pela estrutura de contexto e navegação implementados no HySCharts e pela estrutura de configurações alcançáveis definida na Árvore de Alcançabilidade. Obtemos assim, na Figura 5-1 a visualização da arquitetura projetada para o desenvolvimento do Gerador de Códigos.

Figura 5-1: Arquitetura do Gerador de Códigos

5.3. Especificação do HCGBH

Para implementar o Gerador de Códigos descrito na seção 5.2 um conjunto de classes foi definido. Essas classes representam as informações estruturais obtidas dos ambiente gráficos (StatSim e HMBS) e as informações criadas pelo gerador. A Figura 5-2 mostra a representação da estrutura de classes definida para o HCGBH.

Observa-se que a base de dados referente à Árvore de Alcançabilidade não faz parte da estrutura, isso ocorre porque a árvore é executada em um módulo independente e gera uma base de dados que pode ser armazenada temporariamente. Ou seja, os dados coletados não precisam ficar estruturados em classes, otimizando a estrutura do Gerador.

As três etapas definidas para a implementação do HCGBH e descritas na seção anterior como sendo as formas organizacionais da estrutura são descritas nas próximas seções.

Figura 5-2: Estrutura de Classes do HCGBH

5.3.1. Conversões de Páginas Texto em HTML

A primeira etapa do desenvolvimento do HCGBH é a conversão de páginas de texto (inseridas pelo autor no HySCharts) em documentos HTML. A conversão pode ser obtida através do conjunto de dados $\{E, H\}$ coletados pelo Gerador, onde E representa as configurações de estado dos statecharts e H representa as configurações do hiperdocumento.

Nessa fase um analisador léxico é implementado para capturar os dados armazenados nas páginas de texto e converter para a linguagem alvo HTML. As referências de controle existentes nas páginas de texto são analisadas e a correspondente codificação HTML é inserida. Dentre os dados codificados, as informações referentes à manipulação de âncoras no hiperdocumento são as mais importantes.

Um exemplo de codificação de uma página simples, criado no módulo de autoria do HySCharts, pode ser visualizada na Tabela 5-1. Na coluna 1 é apresentado o arquivo texto a ser codificado e na coluna 2 o arquivo HTML gerado pelo HCGBH.

A referência de controle mencionada anteriormente é apresentada na coluna 1 através dos marcadores de início “<<” e fim “>>”, que representam a definição de uma âncora em uma

página. O gerador define as ligações associadas às âncoras, os conjuntos de atributos de visualização do documento, e as “tags” necessárias para gerar o código HTML.

<i>Arquivo: secao2_2.txt</i>	<i>Arquivo: secao2_2tb.htm</i>
<p>Esta e a pagina da Secao 2 do Capitulo 2 associada com o estado E.</p> <p>Clique <<aqui>> para voltar para a secao anterior</p>	<pre> <HTML> <HEAD> <TITLE> secao2_2 </TITLE> </HEAD> <BODY TEXT= #FF7F00 BGCOLOR= #C0C0C0> <H1> secao2_2 </H1>
 <HR> <ID> <P style="margin-left:2cm; margin-right:2 cm "> Esta e a pagina da Secao 2 do Capitulo 2 associada com o estado E. </P> <P style="margin-left:2cm; margin-right:2 cm "> Clique aqui para voltar para a secao anterior </P> <P style="margin-left:2cm; margin-right:2 cm "> </P>

<HR> </BODY> </HTML> </pre>

Tabela 5-1: Exemplo de Conversão de Página Texto em Página HTML

Este exemplo apresenta a codificação de uma página de texto, mas outros tipos de páginas também são possíveis. O HCGBH permite a geração de código para documentos dos tipos: texto, áudio, video e imagem. Cada tipo de documento possui um conjunto de atributos específicos para codificação.

5.3.2. Formas de Navegação das Páginas HTML

A segunda etapa do desenvolvimento do HCGBH é a definição de formas de navegação do hiperdocumento de forma a verificar os resultados obtidos para possíveis modificações no modelo implementado. A implementação do algoritmo permite a visualização dos hiperdocumentos através de 3 métodos de visualização: visão planar, visão hierárquica e visão por índice.

Considerações sobre o tipo de estados que devem estar ativos, a forma com que os páginas irão se comunicar e o nível hierárquico entre o estado origem e destino não foram analisados nesta visão, pois a forma de visualização planar não se preocupa com as características inerentes dos Statecharts e sim na visualização dos hiperdocumentos e na configuração da árvore de estados gerada pelos Statecharts. Ou seja, a semântica e hierarquia de Statecharts não são preservados e a visualização das páginas em níveis, conseqüentemente, não é possível.

<i>Arquivo: secao2_2.txt</i>	<i>Arquivo: secao2_2t.htm</i>
<p>Esta e a pagina da Secao 2 do Capitulo 2 associada com o estado E.</p> <p>Clique <<aqui>> para voltar para a secao anterior</p>	<pre> <HTML> <HEAD> <TITLE> secao2_2 </TITLE> </HEAD> <BODY TEXT= #FF7F00 BGCOLOR= #C0C0C0> <H1> secao2_2 </H1>
 <HR> <I> <P style="margin-left:2cm; margin-right:2 cm "> Esta e a pagina da Secao 2 do Capitulo 2 associada com o estado E. </P> <P style="margin-left:2cm; margin-right:2 cm "> Clique aqui para voltar para a secao anterior </P> <P style="margin-left:2cm; margin-right:2 cm "> </P> <HR>
 <TABLE> <CENTER> <TR> <TD> up
 </TD> <TD> down
 </TD> <TD> left
 </TD> <TD> right
 </TD> </TR> </TABLE> </CENTER> </BODY> </HTML> </pre>

Tabela 5-2: Exemplo de Geração de Código para a Visão Planar

A função principal da visão planar é apresentar página a página, individualmente, todas as características implementadas pelo HCGBS. A Figura 5-3 apresenta uma visualização gráfica desta forma de visualização. Observe que as setas referenciando o “link” de controle *down* apenas são

associadas ao primeiro filho, não é permitido a definição de duas configurações de controle *down* para um mesmo estado. As setas relacionadas com os “links” de controle *right* e *left* são associadas de acordo com a referência ao primeiro filho e próximo irmão, caso existam. A Tabela 5-2 apresenta o código gerado pela visão planar para o página referenciada na Tabela 5-1, em que é possível notar as inserções de referência aos “links” de controle *up*, *down*, *left* e *right*, de acordo com as especificações definidas, sendo que nem todas referências de controle são associadas a uma página.

Visão Hierárquica

Uma outra forma de visualização especificada é aquela em que todos os estados ativos são visualizados simultaneamente conforme as especificações dos Statecharts. Pode ser interpretado como uma simulação do Statechart em que ao invés da visualização dos estados tem-se a configuração das páginas de hipertexto.

Figura 5-4: Visão de Hierárquica

A Figura 5-4 apresenta de forma gráfica a estrutura definida para essa visão. É possível observar que os eventos são apresentados ao usuário a cada configuração. O conjunto de eventos apresentados

são distintos uns em relação aos outros, dependendo exclusivamente das configurações ativas naquela configuração apresentada.

Este tipo de *visão* preserva a semântica de navegação de Statecharts e permite a codificação de páginas HTML de acordo com um valor N especificado, onde N representa o nível hierárquico do Statechart sendo utilizado. O usuário define antes da geração de código com qual nível hierárquico devem ser geradas as páginas HTML e a visualização é alterada permitindo a inclusão das páginas definidas N níveis acima. O HCGBH gera as páginas de controle associando as configurações de eventos e as configurações de estados simultaneamente utilizando “frames”, que é a notação na linguagem HTML para esta representação.

Visão por Índices

A forma de visualização por índices é aquela onde é apresentado um lista contendo os Títulos de todos as páginas codificadas pelo gerador. Há dois tipos de configuração para esta visão, uma é aquela onde as páginas apresentadas apresentam os eventos associados a ela, caso existam, e a outra é aquela onde as páginas são apresentadas sem quaisquer “links” de controle ou de eventos. Apenas as âncoras definidas nos arquivos textos são visualizadas.

Esse tipo de visão também não mantém a semântica e a hierarquia de Statechart. É apenas uma forma de navegação definida para auxiliar o usuário na verificação das páginas. O índice fica disponível e as configurações das páginas HTML é mostrada em um frame ao lado.

5.3.3. Armazenamento de Informações

O armazenamento das informações geradas pelo HCGBH é efetuados por intermédio de gravação em arquivos de dados em formato binário. As informações podem ser obtidas utilizando-se o arquivo de controle criado para auxiliar o usuário na navegação dos diversos tipos de visões implementadas pela ferramenta. Caso o usuário necessite alterar qualquer informação apresentada nas páginas HTML faz-se necessário a utilização da Tabela 5-3 que apresenta as principais nomenclaturas utilizadas para a criação das páginas.

<i>Nome</i>	<i>Descrição</i>
<i>index.htm</i>	índice que contém as configurações de navegação definidas através das três visões apresentadas (visão planar, hierárquica e por índice)
<i>index2a.htm</i>	contém a configuração de navegação baseada na visão por índice com a presença de eventos.
<i>index2b.htm</i>	contém a configuração de navegação baseada na visão por índice sem a presença de eventos
<i>StchM.htm</i>	contém a configuração de navegação baseada na visão hierárquica, onde <i>M</i> representa o número da configuração solicitada.
<i>confN_M.htm</i>	páginas de controle utilizadas na visão hierárquica, contém as configurações de estados e <i>M</i> representa um número de 2 algarismos.
<i>eventoM.htm</i>	páginas de controle utilizadas na visão hierárquica, contém os eventos associados às configurações de estado. <i>M</i> representa um número de 2 algarismos.
	Para as páginas de <i>NomeArqX.htm</i> são codificados mais dois arquivos de dados <i>NomeArqXa.htm</i> , que representa a visão por índice com eventos, e <i>NomeArqXb.htm</i> , que representa a visão por índice sem eventos. E <i>X</i> pode ser {t, i, a, v} conforme descrito abaixo.
<i>NomeArqt.htm</i>	página texto gerada de acordo com a visão planar, onde <i>NomeArq</i> representa o nome da pagina original e <i>t</i> e <i>a</i> representação indicando que é do tipo texto
<i>NomeArqi.htm</i>	página de imagem gerada de acordo com a visão planar, onde <i>NomeArq</i> representa o nome da pagina original e <i>i</i> e <i>a</i> representação indicando que é do tipo imagem
<i>NomeArqa.htm</i>	página de áudio gerada de acordo com a visão planar, onde <i>NomeArq</i> representa o nome da pagina original e <i>a</i> é a representação indicando que é do tipo áudio
<i>NomeArqv.htm</i>	página de vídeo gerada de acordo com a visão planar, onde <i>NomeArq</i> representa o nome da pagina original e <i>v</i> é a representação indicando que é do tipo vídeo.
<i>EstNM.htm</i>	páginas de controle para estados ortogonais, <i>M</i> representa um número de dois algarismos. Possui a mesma descrição dos arquivos <i>NomeArqXa.htm</i> e <i>NomeArqXb.htm</i> para os arquivos <i>EstNM.a.htm</i> e <i>EstNM.b.htm</i>

Tabela 5-3: Nomenclatura das Páginas HTML Geradas pelo HCGBH

5.4. Características Adicionais do HCGBH

O HCGBH implementado foi desenvolvido utilizando-se a linguagem C++ e as bibliotecas gráficas XVIEW. É uma ferramenta que pode ser utilizada independentemente do ambiente gráfico HySCharts ou que pode ser acoplado.

Desenvolvido inicialmente para ser uma ferramenta independente do ambiente gráfico HySCharts, o HCGBH possui uma interface gráfica própria que apenas auxilia o usuário na geração de código. A intervenção do usuário é realizada de duas formas, escolha do modelo especificado no HySCharts e escolha do nível hierárquico a ser utilizado pela ferramenta HCGBH para Geração de Código.

A Geração de Código é efetuada então em dois passos, no primeiro passo devem ser criadas as bases de dados referentes à Árvore de Alcançabilidade (recurso adicionado na ferramenta) e após a obtenção das configurações alcançáveis é possível codificar as páginas.

5.5. Um Exemplo de Execução do HCGBH

Figura 5-5: Exemplo de Statechart

Considerando o exemplo apresentado na Figura 5-5, tem-se a esquematização da parte administrativa de uma empresa definido no HySCharts, e a apresentação dos dados referentes à associação de estados e páginas é fornecido na Tabela 5-4. Com esses dados capturadas pelo HCGBH tem-se a geração de código das páginas da Tabela 5-4 e a geração de código das páginas de controle para navegação dos hiperdocumentos.

Ident.	Nóme do Estado	Página Associada
0	CETERP	apresent.txt
1	Conheca a Empresa	não há página associada
2	Forms apres	princ.txt
3	Logotipo	ctlogo13.gif
4	Administracao	cadm.txt
5	Informações	admin.txt
6	Financas	não há página associada
7	Inf fin	finan97.txt
8	Img fin	finan.jpg
9	Histórico	chist.txt
10	Início	não há página associada
11	Inf hist	hist.txt
12	Img hist	hist.jpg
13	Hoje	não há página associada
14	Inf emp	empr.txt
15	Img emp	empre.jpg
16	Novidades	nov.txt
17	Informativo	não há página associada
18	Cont Info	não há página associada
19	Img If	inform.jpg
20	Dados	não há página associada
21	629	629.jpg
22	noticia	noticia.txt
23	fale conosco	fcon.txt
24	CGIR	não há página associada
25	Inf CGIR	não há página associada
26	Img CGIR	não há página associada

Tabela 5-4: Associação de páginas aos estados

A configuração obtida da visão planar para este exemplo pode ser visualizada na Figura 5-6. A representação planar é feita através do número de identificação das páginas definidas na Tabela 5-4. No caso dos estados ortogonais que não permitem a associação de páginas, um hiperdocumento de controle foi criado automaticamente pelo gerador para permitir a navegação. No exemplo pode-se visualizar os estado ortogonais 1, 6, 10, 13, 17, 20 e 24 que não possuem páginas mas permitem a navegação através da estrutura planar devido a criação das páginas de controle. Os demais estados que não possuem página associada e não são ortogonais não aparecem na configuração da visão planar. A Figura 5-7 apresenta o código gerado para a visualização planar, apresentada no NETSCAPE. A página visualizada representa o estado 5, denominado Informações. Para este estado apenas duas configurações são possíveis *up* e *left*.

Figura 5-6: Visão Planar do Exemplo da Figura 5-5

Figura 5-7: Exemplo de Código Gerado para Visão Planar

A visão hierárquica corresponde à execução exaustiva do Statechart e à captura de todas as configurações possíveis, de acordo com o disparo de eventos associados a cada configuração

básica conforme descrito anteriormente. De acordo com o exemplo da Figura 5-5 pode-se ilustrar na Figura 5-8 quatro configurações possíveis considerando o nível 0 para a visualização, ou seja, a geração de códigos obtida referencia apenas os estados atômicos relacionados com cada configuração.

Figura 5-8: Visão Hierárquica do Exemplo Figura 5-5

De acordo com base de dados da Árvore de Alcançabilidade foram geradas 30 configurações possíveis. O HCGBH cria uma página HTML de controle para cada configuração definida e não ambígua. Um exemplo do código gerado para este tipo de navegação pode ser visualizada pela Figura 5-9 onde N (nível hierárquico é 0) e pela Figura 5-10 onde o nível hierárquico definido para visualização é 1, e é apresentado as páginas associadas aos estados *Infomacoes* nível 0 e *Administracao* nível 1. O código gerado para as configurações hierárquicas apresentam a página na forma de “frames”. Dividindo-se a página horizontalmente temos duas informações: a configuração de estados na parte superior e os eventos na parte inferior.

A outra forma de navegação, a visão por índice, pode ser visualizada através das Figura 5-11 representando a configuração das páginas com os eventos associados; e Figura 5-12 representando a configuração das páginas sem os eventos associado.

Figura 5-9: Exemplo de Código Gerado para Visão Hierárquica

Figura 5-10: Exemplo de Código Gerado para Visão Hierárquica com N=1

Figura 5-11: Exemplo de Código Gerado para Visão por Índice com Evento

Figura 5-12: Exemplo de Código Gerado para Visão por Índice sem Eventos

A navegação do hiperdocumento pode ser feita aleatoriamente a partir da abertura dos arquivos gerados pelo sistema, ou como mencionado anteriormente através do índice de navegação. O índice apresenta as várias formas de navegação implementadas para que o usuário decida por uma e percorra os hiperdocumentos. A Figura 5-13 apresenta esse índice de navegação conforme definido no sistema e visualizado no Netscape.

Figura 5-13: Índice Geral para Navegação

5.6. Considerações Finais

O tipo de código a ser gerado depende da aplicação a ser desenvolvida e as necessidades de uso decorrentes da modelagem. Se um sistema reativo for modelado dentro do ambiente onde a execução em tempo real é altamente relevante, um Gerador de Códigos deverá disponibilizar uma programação concorrente o mais próximo possível da codificação final solicitada pelo usuário.

Se o sistema a ser desenvolvido como protótipo não necessita de verificação real em suas simulações um outro tipo de código, que seja sequencial e menos complexo, pode ser gerado. Assim, escolher um tipo de aplicação e criar uma ferramenta que possa gerar código o mais próximo possível, é mais viável para um Gerador de Códigos.

Através dos dados fornecidos e do código gerado pelo HCGBH foi possível aplicar os conceitos desenvolvido durante o decorrer da pesquisa e utilizar os conhecimentos obtidos em uma aplicação específica, onde os resultados obtidos puderam ser visualizados e testados.

A forma com a qual foram definidas os “links” na Visão Planar pode ser comparada com aquela implementada no protótipo desenvolvido em Occam e proposto por Drusinsky [DRU89], a diferença entre elas é que uma preseva a sêmantica de Statechart (Drusinsky) e a outra está baseada na semântica de navegação do HMBS.

Capítulo 6

6. Conclusão

6.1. Considerações Iniciais

O desenvolvimento de ferramentas para geração de código a partir de modelos baseados em Statechart está restrito àquela desenvolvida juntamente com STATEMATE e cujas especificações não estão disponibilizadas.

A utilização do ambiente StatSim para o desenvolvimento de projetos baseados em Statecharts que já contém as especificação de um editor e um simulador necessitava desta ferramenta para complementar a prototipagem e a disponibilização de recursos para geração de código dos projetos modelados.

Com a definição do modelo HMBS (Hyperdocument Model Based on Statecharts) [TUR98] e o desenvolvimento do protótipo para essa ferramenta, o HySCharts, a base de informações e as aplicações definida para os hiperdocumentos possibilitaram definir uma aplicação mais próxima do usuário, através da implementação desta ferramenta.

6.2. Contribuições deste trabalho

A disponibilização de vários recursos para análise do modelo, dentro do mesmo ambiente de modelagem é importante para o projetista pois este pode escolher a forma mais adequada e eficiente para cada caso. Apresentando ao usuário uma primeira alternativa de trabalho através da geração de código, evita o desperdício de tempo e custo no desenvolvimento do software.

A geração de código integrando o ambiente HySCharts contribuirá para aumentar a funcionalidade do ambiente já que os requisitos de navegação também foram implementados na ferramenta, e permitirá que o usuário possa visualizar os testes efetuados durante a simulação através de uma codificação real em seu ambiente de trabalho natural.

6.3. Cosiderações Finais

Os recursos gerados pela ferramenta implementada utilizam outro algoritmos. Em uma primeira instância estes algoritmos poderiam ser atualizados a fim de aproveitar melhor os recursos do sistema.

Alguns recursos poderiam ser adicionados ao Gerador de Códigos para definir propriedades de utilização de tópicos para navegação avançada de hiperdocumento, através de informações interativa de formulários, “frames” entre outros. Criar um ambiente de trabalho onde o usuário pudesse decidir por alguns detalhes de implementação durante o processo de geração de código.

A idéia de permitir a construção de um gerador genérico para qualquer tipo de aplicação também poderá ser avaliada e implementada para o ambiente StatSim de acordo com as características definidas a partir deste trabalho.

BIBLIOGRAFIA

- [ALL81] ALLWORTH, S.T. - Introduction to Real-time Software Design. London, McMillan, 1981.
- [AMB92] Ambiente de Desenvolvimento de Software - TALISMAN; Resumo Técnico; Staa Informática; 1992.
- [BAT91a] BATISTA NETO, J. do E.S. - Um Editor Gráfico para Statecharts. Dissertação de Mestrado, Instituto de Ciências Matemáticas de São Carlos, março, 1991.
- [BAT91b] BATISTA NETO, J. do E.S.; TUTUMI, R.; MASIERO, P.C. - Um Editor Gráfico de Statecharts (EGS), Resumo de Ferramenta, V Simpósio Brasileiro de Engenharia de Software, Ouro Preto, MG, 23 a 25/10, p.11, 1991.
- [BOA92] BOAVENTURA, I.G. - Propriedades Dinâmicas de Statecharts, Dissertação de Mestrado, Instituto de Ciências Matemáticas de São Carlos - USP, março de 1992.
- [BOU91] BOUSSINOT, F. e SIMONE R. de - The ESTEREL Language, Proceedings of the IEEE, 79(9), setembro, 1991.
- [CAN92] CANGUSSU, J.W.L.; MASIERO, P.C. - Uma Linguagem para Execução Programada de Statecharts. Anais do XII Congresso Nacional da SBC, XIX SEMISH, Rio de Janeiro, 29/09 a 02/10, p.22-242, 1992.
- [CAN93a] CANGUSSU, J.W. de L. - Execução Programada de Statecharts, São Carlos, ICMSC-USP, Dissertação de Mestrado, Instituto de Ciências Matemáticas de São Carlos, 1993.
- [CAN93b] CANGUSSU, J.W. de L.; MASIERO, P.C.; MALDONADO, J.C. - Execução Programada de Statecharts, VII Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, 27 a 30 de Outubro, 1993.

- [COL92] COLEMAN, D.; HAYES, F.; BEARS, S. - Introducing Objectcharts or How to Use Objectcharts in Object-Oriented Design. IEEE Transactions on Software Engineering, Vol. 18 (1):9-18, 1992.
- [COL94] COLEMAN, D.; ARNOLD, P.; BODOFF, S.; DOLLIN, C.; GILCHRIST, H.; HAYES, F.; JEREMAES, P. *Object-Oriented Development: THE FUSION METHOD*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1994, 313p.
- [DRU89] DRUSINSKY, D.; HAREL, D. - Using Statecharts for Hardware Description and Synthesis, IEEE Transactions on Computer-Aided Design, Vol. 8 (7): 798:806, July 1989.
- [DAV88a] DAVIS, A. M. - A comparison of Techniques for the Specification of External System Behavior. Communications of the ACM, 31 (9):1098-115, 1988.
- [DAV88b] DAVIS, A.M. - A Comparison of Techniques for the Specification of External System Behavior. Communications of the ACM, 31 (9), p. 1098-115, 1989.
- [FAI93] FAISON, T. - Object-Oriented State Machines, Software Development, 1 (3) : 37-50, September 1993.
- [FIL91] FILHO, A.G.F. e LIESENBERG, H.K.E. - Geração de Gerenciadores de Sistemas Reativos - V Simpósio Brasileiro de Engenharia de Software, Ouro Preto, MG, p.6, 23 a 25/10, 1991.
- [FOR91a] FORTES, R.P. de M.; MASIERO, P.C.; Um Sistema para Simulação de Estadogramas (Statecharts). XXIV Congresso Nacional de Informática - SUCESU, p.153-161, São Paulo, setembro de 1991.
- [FOR91b] FORTES, R.P. de M. - Uma Ferramenta de Apoio à Utilização de Statecharts para a Especificação do Comportamento de Sistemas de Tempo-Real Complexos, Dissertação de Mestrado, Instituto de Ciências Matemáticas de São Carlos, Abril, 1991.
- [FOR91c] FORTES, R.P. de M. - Um Simulador de Statecharts, Resumo de Ferramenta, V Simpósio Brasileiro de Engenharia de Software, Ouro Preto, MG, p.7, 23 a 25/10, 1991.

- [FUR92] FURUUTI, R. - Extensões do Ambiente Statsim para Permitir a Simulação de Micro-Passos. São Carlos, ICMSC-USP, Relatório CNPq, IC, Partes I e II, 1992.
- [GAB91] GABRIELIAN, A.; FRANKLIN, M. K. - Multi-level Specification of Real-Time Software, Communications of the ACM, 33 (5):50-60, May 1991.
- [GOM84] GOMMA, H. - A Software Design Method for Real-Time Systems, Communications of the ACM, 27(3), p. 938-949, 1984.
- [HAR87a] HAREL, D. - STATECHARTS: A visual Formalism to Complex Systems. Science of Computer Programming, 8, p. 231-274, 1987.
- [HAR87b] HAREL, D. et al - On the Formal Semantics of Statecharts, Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, Ithaca, N.Y., 1987.
- [HAR88] HAREL, D. - On Visual Formalisms, Communications of the ACM, 31 (5), p. 514-530, 1988.
- [HAR90] HAREL, D - STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering, 16 (4), p. 403-414, 1990.
- [KES92] KESTEN, Y. e PNUELI, A. - Timed and Hybrid Statecharts and Their Textual Representation; In Formal Techniques in Real-Time and Fault-Tolerant Systems; Lecture Notes in Comp. Sci.; Springer-Verlag; Proceedings of a Symposium Nijmegen; 1992.
- [MAS91a] MASIERO, P.C; FORTES, R.P.M.; BATISTA, J.E.S. - Edição e Simulação do Aspecto Comportamental de Sistemas de Tempo Real. In: Seminário Integrado de Software e Hardware, 18, Santos 1991, Anais, Santos-SBC, p.45-61, 1991.
- [MAS93] MASIERO, P.C. ; MEIRA, C.A. - Development and Instantiation of a Generic Application Generator, The Journal of Systems and Software, 23 (1):27-38, October 1993.

- [MAS94] MASIERO, P.C.; MALDONADO, J.C.; BOAVENTURA, I.G. - A Reachability Tree for Statecharts and Analysis of Some Properties. *Information and Software Technology*, v.36(10), p615-24, 1994.
- [MEI91b] MEIRA, C.A.A.; MASIEIRO, P.C. - UmGerador de Aplicações para Sistemas Reativos. *Anais do V Simpósio Brasileiro de Engenharia de Software*, Ouro Preto, p. 45-60, 23 a 25/10, 1991.
- [MEI91c] MEIRA, C.A.A. - Sobre Geradores de Aplicação, Dissertação de Mestrado, Instituto de Ciências Matemáticas de São Carlos - USP, set, 1991.
- [OLI96] OLIVEIRA, M.C.F.; TURINE, M.A.S.; MASIERO, P.C. An overview of HMBS: a statechart-based model for hypertext. In: *II Workshop sobre Sistemas Hipermídias - II WOSH*, Fortaleza, CE, 1996. *Anais*. Fortaleza, CE, 18 e 19 de maio de 1996, p.11-20.
- [PET81] PETERSON, J. L. - *Petri Net Theory and the Modeling of Systems*, Englewood Cliffs, NJ, Prentice-Hall, 1991.
- [PRE92] PRESSMAN, R.S. - *Software Engineering: A Practitioner's Approach*, McGraw-Hill Inc, p.792, 3o. edição, 1992.
- [RUM91] RUMBAUGH, J. et all - *Object-Oriented Modeling and Design* - Prentice Hall Inc, p. 500, 1991.
- [TUR96] TURINE, M.A.S.; MASIERO, P.C. Especificação de Requisitos: Uma Introdução. In: *Relatório Técnico do ICMSC*, São Carlos, Março 1996, 25p.
- [TUR97] TURINE, M.A.S.; OLIVEIRA, M.C.F.; MASIERO, P.C. A Navigation-Oriented Hypertext Model Based on Statecharts. In: *Hypertext'97, 8th International Hypertext Conference*, ACM, Southampton, UK, April 6-11, 1997. *Proceedings*. University of Southampton, The Eighth ACM Conference on Hypertext, p.102-111, 1997.

- [TUR98] TURINE, M.A.S.; HMBS: Um Modelo Baseado em Statecharts ~~para a Especificação~~
Formal de Hiperdocumentos. Dissertação de Doutorado, IFSC-USP, São Carlos, 1998 (em
preparação).
- [TUT92] TUTUME, R.; YSHIY, S. - Implementação de um Editor Gráfico ~~para Diagrama de~~
Atividades. Relatório de IC- CNPq, ICMSC-USP, 1992.
- [WAR86] WARD, P. T. - The Transformation Schema: an Extension of the Data Flow Diagram to
Represent Control and Timing. IEEE Transactions on Software Engineering, 12 (2):198-210,
1986.
- [WAR89] WARD, P.T. - The Transformation Schema: An Extension of the Data Flow Diagram to
Represent Control and Timing, IEEE Transactions on Software Engineering, 12(3), p. 198-
210, 1986.