Um Editor Gráfico Para Statecharts

João do E.S. Batista Neto

Orientação:

Prof. Dr. Paulo Cesar Masiero

*Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos,
da Universidade de São Paulo, como parte dos requisitos para obtenção do título
de Mestre em Ciências de Computação e Matemática Computacional.*

**USP - São Carlos
Abril de 1991**

Aos Meus pais pelo incentivo.

Ao Pacheco e ao Ricardo pela amizade.

Agradecimentos

Ao Professor Dr. Paulo Cesar Masiero pela orientação sempre objetiva durante o desenvolvimento deste trabalho.

Ao Professor Edson Moreira pelas fotos.

À Professora Renata Pontin pela ajuda constante.

Ao Renato Tutumi pelo auxílio na programação.

À Beth e à Laura da secretaria da pós-graduação pela eficiência.

Ao CNPq que criou condições para a realização do Mestrado.

Conteúdo

1	Introdução	1
1.1	Situação do Problema e Motivação	1
1.2	Objetivos	3
1.3	Organização da Dissertação	3
2	Conceitos Básicos	5
2.1	Considerações Iniciais	5
2.2	Teoria dos Conjuntos - Diagramas de Venn-Euler	6
2.3	Grafos	6
2.4	Hipergrafos	7
2.5	Higraphs	8
2.6	Statechart	11
2.7	Statechart : um exemplo	13
2.7.1	História	15
2.7.2	Ortogonalidade : concorrência	18
2.7.3	Eventos Condicionados	20
2.7.4	A Ação “Limpa-História”	21
2.8	Eventos, Ações e Atividades	21

2.9	Considerações Finais	23
3	Revisão Bibliográfica	24
3.1	Considerações Iniciais	24
3.2	Técnicas para Representação do Aspecto Comportamental dos Sis- temas Reativos	25
3.2.1	Máquinas de Estados Finitos	25
3.2.2	Redes de Petri	28
3.2.3	F-Nets e R-Nets	29
3.2.4	Esquema de Transformação	31
3.2.5	Statechart	33
3.2.6	Tabelas e Árvores de Decisão	33
3.3	Ferramentas Automatizadas para Descrição de Sistemas Reativos	34
3.3.1	ProMod	34
3.3.2	TAGS	36
3.3.3	Excelerator	37
3.3.4	Statemate	38
3.3.5	IEW	40
3.3.6	Teamwork/ES	40
3.3.7	Argonauta	41
3.3.8	Protob	42
3.4	Avaliação das Ferramentas	42
3.5	Considerações finais	46
4	A Ferramenta Desenvolvida: um Editor Gráfico para Statecharts (EGS)	47

4.1	Considerações Iniciais	47
4.2	Requisitos do EGS	48
4.3	A Arquitetura do EGS	56
4.4	Decisões de Projeto	61
4.4.1	A Interface Gráfica do EGS	66
4.4.2	Restrições	68
4.5	Aspectos relacionados ao desenvolvimento do EGS	69
4.6	Considerações Finais	71
5	Exemplo de uso do EGS	73
5.1	Considerações Iniciais	73
5.2	A Interface do EGS	73
5.3	Exemplo de um Diagrama Criado Usando o EGS	77
5.3.1	Manipulação de Bolhas	79
5.3.2	Manipulação de Arcos	83
5.3.3	Arquivos: operações sobre a Base de Dados de Statechart .	89
5.3.4	Variáveis	91
5.4	Discussão sobre a generalização do EGS para um editor de Higraphs	92
5.5	Considerações Finais	96
6	Conclusões	97
6.1	Considerações Iniciais	97
6.2	Resultados Obtidos	98
6.3	Sugestões para Futuras Pesquisas	99
	Bibliografia	101

A	Tipos e Estruturas de Dados da Parte Gráfica do EGS	105
B	A Definição Formal dos Higraphs	113
C	Gramática da Linguagem de Especificação de Statecharts	116

RESUMO

Uma técnica para especificação de Sistemas Reativos são os Statecharts, que são uma extensão dos Diagramas de Transição de Estados (DTEs), incorporando aspectos de hierarquia, paralelismo e comunicação entre processos. O aspecto visual da técnica baseia-se no conceito de Higraphs.

Nesta dissertação apresenta-se um tutorial da técnica Statechart, mostrando-se com detalhes as suas principais características visuais, bem como o seu aspecto semântico. Apresenta-se, também, uma revisão bibliográfica de várias técnicas desenvolvidas com a mesma finalidade dos statecharts, isto é, representar o aspecto comportamental de Sistemas Reativos. Complementa-se a revisão bibliográfica com uma avaliação de algumas ferramentas de apoio ao uso dessas técnicas, com ênfase no aspecto gráfico e formal.

Apresentam-se os requisitos para um Editor Gráfico de Statecharts (EGS) e discutem-se aspectos da implementação realizada. Ilustra-se o uso da ferramenta, destacando-se a sua interface gráfica com manipulação direta de objetos na tela, através de um exemplo abrangente. Discute-se, também, o que deveria ser feito para generalizar esse editor, tornando-o um editor de Higraphs, permitindo o apoio à edição de outros tipos de diagramas.

ABSTRACT

The Statechart are a technique for specifying Reactive Systems. They are an extension of the State Transition Diagrams and embody the concepts of hierarchy, parallelism and process communication. The visual aspects of this technique is based on the Higraphs.

In this dissertation a Statechart tutorial is shown, focusing in its main visual characteristics. It is also shown a bibliographic revision of several techniques developed for reactive systems specification. This revision is completed with an evaluation of some tools available to support these techniques. The graphical and formal aspects are emphasized.

The requirements for a Statechart Graphical Editor are presented and the implementation done is discussed. The tool's use is demonstrated through an example, emphasizing the graphical interface with direct object manipulation. The necessary steps to generalize the editor is also discussed. This would allow the editor to support the creation of other diagrams.

Capítulo 1

Introdução

1.1 Situação do Problema e Motivação

Um Sistema Reativo, em contraste com um Sistema Transformacional, é caracterizado por ser orientado a eventos, ou seja, ele está continuamente respondendo a estímulos do meio externo e interno. Redes de Comunicação, Sistemas de Telefonia, Sistema Operacionais, Sistemas para Aviação e Interfaces Homem-Máquina são alguns exemplos de Sistemas Reativos.

No caso de Sistemas Transformacionais (Sistemas de Processamento de Dados, em geral), existem várias técnicas, métodos e ferramentas voltadas para a descrição e especificação desses sistemas que, na prática, funcionam satisfatoriamente. A disponibilidade de técnicas, métodos e ferramentas para Sistemas Reativos é bem menor, devido à complexidade do tratamento do fator tempo e porque essas aplicações geralmente são restritas a áreas de acesso limitado, como por exemplo os sistemas militares.

De um modo geral, existe um consenso na Engenharia de Software de que há um problema central a ser considerado na especificação e projeto de Sistemas Reativos grandes e complexos: a dificuldade em se descrever o comportamento reativo de maneira clara e realística e ao mesmo tempo, formal e rigorosa [Dav 88, Har 88].

Acredita-se, como pode ser visto em [Jac 83], [Sun 82], [Tan 81], que a maneira mais natural para representar o comportamento reativo de um sistema complexo seja através da utilização de estados e eventos, que são os elementos básicos das Máquinas de Estados Finitos (MEFs) [Bra 77]. Essa afirmação pode

ser reforçada, verificando-se que alguns métodos incorporam as MEFs, para possibilitar a especificação do comportamento reativo de Sistemas em Tempo Real, como pode ser visto em [War 86]. Nesse artigo, Ward apresenta uma extensão dos Diagramas de Fluxo de Dados capaz de representar controles e aspectos temporais.

Porém, o uso das MEFs convencionais na representação de Sistemas Reativos realmente complexos, muitas vezes se torna inviável por diversas razões:

a) As Matrizes de Transição de Estados(MTEs) e os Diagramas de Transição de Estados(DTEs), que são formas de representação gráfica das MEFs, são planos ("flats"). Não permitem uma visão da profundidade, hierarquia e modularidade, bem como não suportam desenvolvimento do tipo ascendente (bottom-up) ou descendente (top-down).

b) Os DTEs são muito dispendiosos com relação à representação de estados e transições: um evento que cause sempre a mesma transição a partir de um certo número de estados, deve ser conectado a cada um desses estados separadamente, resultando em um número muito grande de arcos. Dessa forma, na medida que a complexidade do sistema descrito cresce linearmente, o número de estados e transições cresce exponencialmente. Isso pode levar à geração de diagramas muito grandes e confusos.

c) Os DTEs são sequenciais por natureza, inviabilizando a representação de processos concorrentes de uma maneira natural.

Um outro fator a ser considerado é a crescente utilização de linguagens, técnicas e metodologias visuais que exploram de forma racional as vantagens da interação gráfica homem-máquina como pode ser visto nas interfaces gráficas existente em muitas ferramentas CASE, como por exemplo: Promod, TAGS, Excelsator, Statemate, etc., todas elas ilustradas no decorrer desta dissertação.

É nesse contexto que se apresentam os STATECHARTS¹ [Har 87, Har 87b, Har 88], constituindo-se numa maneira vantajosa para a representação de Sistemas Reativos, uma vez que faz uso do formalismo clássico das Máquinas de Estados Finitos e tenta adequar essas características para uso em aplicações grandes e complexas. Deve-se considerar, ainda, que a representação gráfica adotada pelos statecharts baseia-se em um formalismo visual denominado Higraphs, o qual será definido no decorrer desta dissertação.

---

¹Do inglês STATE = Estado e CHARTS = Diagramas. Uma possível tradução para esse termo é ESTADOGRAMA, encontrada em alguns trabalhos sobre statecharts publicados no Brasil. Nesta dissertação, será utilizado o nome original.

No entanto, por ser uma técnica recente (apresentada primeiramente por Harel, em [Har 87]), não existe para os statecharts um número grande de ferramentas e, conseqüentemente, uma interface gráfica para apoiar a técnica citada. Das ferramentas pesquisadas, apenas uma: Statemate [Har 88a], possui um editor de statecharts.

1.2 Objetivos

O objetivo desta dissertação é apresentar uma técnica para especificação e projeto de Sistemas Reativos conhecida como Statechart e propor o desenvolvimento de um editor gráfico para essa técnica.

Para justificar a escolha dos Statecharts são apresentadas algumas técnicas e ferramentas utilizadas na especificação dos sistemas reativos, ressaltando o aspecto visual e ao mesmo tempo formal em que estas técnicas e ferramentas se baseiam a fim de atingir os seus objetivos.

A razão desse enfoque se deve à crença do autor na eficiência e atratividade apresentadas por técnicas que possuem apelo visual (sem que por isso, sejam menos formais) para representação do comportamento de Sistemas Reativos, como é o caso dos Statecharts.

Baseado numa sintaxe própria, o editor gráfico permite a criação de diagramas de statecharts, faz a consistência das informações e armazena o resultado em uma base de dados.

Por ser implementado sob a definição dos Higraphs, o editor pode ser estendido e utilizado para dar apoio a outras técnicas que também sejam baseadas nos higraphs. Além disso, o editor gráfico deve poder ser integrado a outros sistemas, pois ele é uma das ferramentas de um ambiente de apoio ao desenvolvimento de Sistemas em Tempo Real que se encontra em construção no ICMSC-USP.

1.3 Organização da Dissertação

Esta dissertação é composta por seis capítulos logicamente organizados da seguinte maneira:

Neste primeiro capítulo foi apresentada a motivação ao trabalho pro-

posto. Foram levantadas as diferenças básicas entre os sistemas transformacionais e reativos e a necessidade de uma técnica mais adequada para especificação e projeto dessa última categoria de sistemas.

No capítulo dois apresenta-se um tutorial da técnica Statechart que é organizado em duas partes: descrição dos conceitos básicos que compõem a técnica e apresentação das suas principais características. Essa apresentação é feita utilizando-se os Statecharts na especificação de um relógio de pulso digital. Para o leitor familiarizado com a técnica statechart, esse capítulo não é imprescindível para o entendimento dos demais, podendo ser saltado.

O capítulo três contém a revisão bibliográfica. São expostas várias técnicas relevantes para a representação do aspecto comportamental dos Sistemas Reativos. Além disso, descrevem-se algumas ferramentas de apoio atualmente disponíveis para a especificação desse tipo de sistemas. Ao final do capítulo é feita uma avaliação das ferramentas com base no aspecto visual e formal que elas apresentam.

No capítulo quatro serão apresentados alguns aspectos relacionados à análise e ao projeto da ferramenta implementada. Foi definido um conjunto de requisitos baseado no formalismo dos Higraphs e dos Statecharts, mostrando-se como a ferramenta incorporou esses requisitos.

No capítulo cinco serão mostradas as principais funções desempenhadas pela ferramenta e as principais características de sua interface gráfica. Os diagramas apresentados nas figuras desse capítulo foram gerados pela própria ferramenta.

Finalizando, no capítulo seis encontram-se as conclusões finais sobre o projeto desenvolvido e uma avaliação da ferramenta implementada, bem como sugestões para futuras pesquisas e extensões que possam contribuir para enriquecer o editor.

Capítulo 2

Conceitos Básicos

2.1 Considerações Iniciais

Os Statecharts podem ser caracterizados como uma linguagem de especificação, como um formalismo e como uma tecnologia [Bru 86].

Como linguagem, os Statecharts podem ser descritos como uma abordagem visual, baseada em estados e voltada para a especificação de Sistemas Reativos. Uma especificação baseada em estados descreve o comportamento de um sistema em função dos estados que ele possui e dos eventos que causam a mudança entre estados. É uma abordagem visual porque os estados são representados visualmente como bolhas, conectadas por linhas que representam os eventos.

Como formalismo, os Statecharts são uma extensão das Máquinas de Estados Finitos, incluindo decomposição AND/OR, noção de história, variáveis de estados e lógica temporal.

Como tecnologia, os Statecharts são suportados por uma ferramenta chamada Statemate, que possibilita a criação/modificação, análise e execução dos diagramas de Statecharts.

O objetivo deste capítulo é mostrar os principais conceitos nos quais os Statecharts se baseiam e por que eles podem ser definidos como uma linguagem de especificação visual e ao mesmo tempo formal. A ferramenta Statemate é apresentada no capítulo seguinte.

Figura 2.1:

2.2 Teoria dos Conjuntos - Diagramas de Venn-Euler

Conjunto é um conceito fundamental em diversos ramos da matemática. Intuitivamente, um conjunto é uma lista, coleção ou classe de objetos bem definidos. Os objetos em um conjunto podem representar qualquer coisa: números, pessoas, letras, etc. Esses objetos são chamados de elementos ou membros do conjunto [Lip 64].

Pode-se representar os conjuntos de forma textual, onde os conjuntos em si são designados por letras maiúsculas e os elementos por letras minúsculas. Exemplo : $A = \{a, b, c, d, e, f\}$

Uma outra forma simples e instrutiva de representar as relações existentes entre conjuntos é por meio dos chamados diagramas de Venn-Euler ou simplesmente, diagramas de Venn. O conjunto será representado por uma área plana limitada geralmente por círculos. Como exemplo, sejam os conjuntos $A = \{a, b, c, d\}$ e $B = \{c, d, e, f\}$. Pode-se representá-los utilizando os diagramas de Venn, como mostra a figura 2.1.

2.3 Grafos

Um grafo é uma das formas mais conhecidas de representação topo-visual formal e foi proposto pelo matemático suíço Leonhard Euler.

Segundo Beineke e Wilson, em [Bei 83], um grafo G é um par $(V(G), E(G))$ onde $V(G)$ é um conjunto finito não vazio de elementos chamados vértices e $E(G)$, um conjunto finito de pares não ordenados de elementos distintos de $V(G)$, chamados de arestas (edges). Veja a figura 2.2, onde $V = \{A, B, C, D\}$ e $E = \{\{A, B\}, \{B, C\}, \{A, C\}, \{C, D\}\}$.

Figura 2.2:

Tomando-se a definição acima, pode-se ainda considerar grafos direcionados nos quais, ao invés de arestas (pares não ordenados), existem arcos (pares ordenados). Nesse caso, tem-se um dígrafo ou grafo dirigido. Ao se retirar a restrição de que as arestas devam ser distintas, tem-se os multigrafos, onde duas ou mais arestas conectam o mesmo par de vértices.

Segundo Berge, em [Ber 73], um grafo é formado por um conjunto de pontos e um conjunto de setas, com cada uma dessas setas unindo um ponto a outro. Os pontos são chamados de vértices e as setas, de arcos. Formalmente, um grafo G é definido como sendo um par (X, U) , onde:

1 - X é um conjunto $\{x_1, x_2, \dots, x_n\}$ de elementos chamados vértices.

2 - U é uma família $(u_1, u_2, \dots, u_m)$ de elementos do Produto Cartesiano $X \times X$, chamada arcos. Normalmente, esta família será representada pelo conjunto $U = \{1, 2, \dots, m\}$ de seus índices. Em um grafo $G = (X, U)$, cada arco $u_i = (x, y)$ determina uma linha contínua unindo x a y . Tal linha, se não houver nenhuma indicação de sua direção, é chamada de aresta (edge) e é denotada por $e_i = [x, y]$.

Neste trabalho, será utilizada a definição proposta por Berge, que ao contrário de Beineke e Wilson, não diferencia grafos direcionados dos não direcionados.

Os grafos e os multigrafos aparecem, com frequência, ocultos sob outros nomes. Quando se fala em sociograma (Psicologia), simplex (Topologia), redes elétricas, redes de comunicação e árvores, está-se falando também de grafos e/ou multigrafos. Todas estas disciplinas utilizam-se, no fundo, dos mesmos teoremas.

2.4 Hipergrafos

Segundo Beineke e Wilson, em [Bei 83], um hipergrafo G é um par $(V(G), E(G))$ onde $V(G)$ é um conjunto finito e não vazio de vértices e $E(G)$, um conjunto

Figura 2.3:

finito de conjuntos não ordenados de elementos distintos de $V(G)$, chamados de arestas (edges). Na figura 2.3 tem-se: $V = \{A, B, C, D, E, F, G, H\}$ e $E = \{\{A, B, C\}, \{B, E, F\}, \{D, G, H\}, \{G, H, F\}\}$.

Um hipergrafo, é um grafo no qual as relações a serem especificadas não são necessariamente binárias, ou seja, um arco não conecta um par de vértices e sim um subconjunto deles. Isso faz com que os hipergrafos possam assumir uma configuração difícil de ser representada visualmente.

2.5 Higraphs

Do que foi exposto em 2.2, 2.3 e 2.4, deve-se extrair a idéia de que, enquanto grafos e hipergrafos constituem-se num modo adequado de representação de um determinado conjunto de elementos e relações sobre eles, os Diagramas de Venn/Euler são uma boa forma de representação das relações estruturais entre conjuntos (Um certo conjunto pode estar contido em um outro, por exemplo). No entanto, existem no campo da computação, aplicações cuja complexidade dos objetos, sistemas ou situações sob consideração, requerem a utilização conjunta de ambas as capacidades acima descritas: o formalismo de grafos e hipergrafos e aspectos relacionados à Teoria dos Conjuntos presentes nos diagramas de Venn/Euler. Além dessa capacidade, seria relevante identificar, em diversas situações, o Produto Cartesiano de alguns conjuntos a fim de prevenir o crescimento exponencial de determinadas aplicações, tornando a representação gráfica mais concisa. Será visto mais adiante que o Produto Cartesiano é uma forma de representar concorrência.

Baseado nesses fatos e como uma solução para a composição dos conceitos mostrados em 2.2, 2.3 e 2.4, foram desenvolvidos os HIGRAPHS ¹ [Har 88], que

¹do inglês, Hierarchical Graphs.

Figura 2.4:

são uma combinação de:

- a) Diagramas de Venn/Euler,
- b) Grafos ou Hipergrafos, e
- c) Capacidade de representação do Produto Cartesiano.

A figura 2.4 representa um higraph no qual pode-se verificar os conceitos a, b e c descritos acima e que serão explicados a seguir.

Como pode ser notado, o higraph da figura 2.4, representa um diagrama de Venn/Euler modificado, onde ao invés de círculos, utilizam-se retângulos com bordas arredondadas, os quais serão denominados de bolhas (*blobs*, do termo em inglês). Considera-se, ainda, que cada bolha representa uma certa espécie de conjunto, no qual as bolhas internas representam uma inclusão de conjuntos e não membros deste conjunto. Assim, pode-se ter casos de inclusão, disjunção e intersecção de conjuntos.

Uma diferença a se considerar com relação aos diagramas de Venn/Euler é que nos higraphs, todo conjunto de interesse deve ser representado por uma bolha com contornos próprios. Um dos motivos para que todo conjunto possua uma área delimitada por uma bolha, é a necessidade de estabelecer níveis hierárquicos e nomenclatura adequada. Dessa maneira, o fato de *T* intersectar *R* no higraph

acima, não significa, necessariamente, que os conjuntos representados por T e R têm uma intersecção ou que o conjunto diferença de T por R ($T-R$) não é vazio. De fato, no formalismo adotado, a intersecção de duas bolhas não significa nada, a não ser que apareçam bolhas internas no local da intersecção. Dessa forma, da maneira como está na figura 2.4, a bolha T poderia ser desenhada completamente separada de R .

Utilizando esta convenção, é possível notar que os únicos conjuntos que podem realmente ser identificados, são os conjuntos chamados atômicos, ou seja, aqueles representados por bolhas residindo no nível mais interno do diagrama e que não contém nenhuma outra bolha interna a si mesmos. Assim, as bolhas (conjuntos) atômicas da figura 2.4 são: $B, C, E, G, H, I, K, L, M, N, O, Q, S$ e T . Pode-se concluir do exposto, que um espaço vazio em um higraph não representa nada. A não ser que seja a área interna de uma bolha atômica.

A figura 2.4 mostra ainda como um Produto Cartesiano pode ser representado em um higraph: a linha pontilhada presente na bolha J , separando as bolhas internas K, N e I, L, M , representa o produto da união das duas bolhas à direita (K e N) com a união das 3 bolhas à esquerda (I, L e M). Simbolicamente, tem-se :

$$J = W \times X = (K \cup N) \times (I \cup L \cup M)$$

Os operandos do produto W e X são chamados de componentes ortogonais da bolha J . Como será visto mais adiante, as linhas pontilhadas constituem a forma de representação de uma das mais importantes características de um higraph : a representação de concorrência.

Agora que já se definiu um formalismo para a representação dos conjuntos (bolhas) e as suas relações tanto estruturais quanto as pertinentes à teoria dos conjuntos (união, intersecção, produto cartesiano, etc.), é hora de se adicionar os arcos, ou seja, deve-se introduzir o conceito de grafos e hipergrafos dentro dos higraphs. Um higraph é obtido, permitindo-se que arcos sejam adicionados ao contorno de qualquer bolha. Assim como nos grafos, os arcos podem ser direcionados ou não, rotulados ou não, de apenas um tipo ou vários, etc.

O higraph da figura 2.4 possui arcos direcionados e sem rótulos, com aridade variando entre 2 e 3. A maioria das setas representa arcos binários tais como os que conectam E e A, Q e O . Outros são direcionadas em três rotas como os que unem E, T, J e R, M, D .

Pode-se encontrar na literatura da Ciência da Computação diversas apli-

cações que utilizam grafos. No entanto, algumas dessas aplicações poderiam se beneficiar das extensões oferecidas pelos Higraphs.

Harel, em [Har 88], utilizou um higraph no lugar de um diagrama de Entidade-Relacionamento para especificar um modelo de dados. Esses diagramas são, na realidade, hipergrafos onde os retângulos representam os nós (no caso, as entidades) e os arcos (cujos rótulos são representados por losangos) são as relações entre as entidades. O resultado final foi uma representação com maior grau de detalhes entre entidades e relacionamentos, e ao mesmo tempo, mais simples de se visualizar. Harel sugere, ainda, a utilização de higraphs na especificação de Redes Semânticas [Nil 80], que são estruturas baseadas em grafos, largamente usadas na área de Inteligência Artificial para processamento de linguagem natural e representação de conhecimento. O formalismo de Redes Semânticas é parecido com o formalismo dos diagramas de Entidade-Relacionamento, onde as relações, se usadas em grande escala, podem resultar em diagramas grandes e às vezes incompreensíveis.

Higraphs foi recentemente a base de uma publicação [Tyg 87] na qual apresentou-se uma linguagem visual para especificação de requisitos de segurança em Sistemas Operacionais. Direitos de acesso e exceções foram representados como arcos. As bolhas podiam ser grupos de usuários, arquivos e outras entidades relevantes. O produto cartesiano foi usado para representar a decomposição dos arquivos em seus componentes básicos. Uma outra idéia utilizando higraphs, aparece em [Man 87, Owi 82] na forma de diagramas para verificação de programas concorrentes.

No apêndice B desta dissertação são mostradas a sintaxe e a semântica formal dos Higraphs, definida por Harel em [Har 88].

2.6 Statechart

Statechart é uma técnica que combina Diagramas Convencionais de Transição de Estados com três elementos diretamente relacionados às noções de hierarquia, concorrência e comunicação. Esses elementos (também presentes nos higraphs) transformam a linguagem de diagramas de estados em uma linguagem altamente estruturada e de descrição econômica, que pode ser de grande utilidade nas etapas de especificação e projeto de Sistemas de Tempo Real, Protocolos de Comunicação e Sistemas Reativos, descritos no início deste trabalho.

Pode-se também definir Statechart como sendo um formalismo visual,

Figura 2.5:

onde é possível representar estados e transições de forma modular, possibilitando abstrações (*clusterização*), ortogonalidade (concorrência) e refinamento de processos, além do que se utiliza das técnicas de decomposição (zoom) para representar graficamente os diversos níveis hierárquicos e/ou abstratos que possam existir no diagrama.

Harel define, em [Har 87,Har 88], os Statecharts como diagramas de estados convencionais decompostos em estados do tipo AND/OR juntamente com níveis de transição e um mecanismo de comunicação entre os processos. A razão básica para se estender os diagramas de estados convencionais, foi a necessidade de representação gráfica e semântica de profundidade e a noção de ortogonalidade, ou seja, concorrência. Esquemáticamente tem-se:

STATECHART = Diagrama de Estados + profundidade + concorrência
+ mecanismo de comunicação.

Vale notar que os três últimos componentes (profundidade, concorrência e mecanismo de comunicação) são características presentes em um higraph, no qual os Statecharts estão fortemente embasados.

Representação dos Estados: Abstração e Refinamento

A representação gráfica dos Statecharts é semelhante à definida para os higraphs. Pode-se notar que, através desse tipo de simbologia gráfica, é possível reproduzir as noções de profundidade e hierarquia. Os retângulos de bordas arredondadas, representam estados em níveis variados e o encapsulamento expressa a relação hierárquica entre eles. Todo arco une um ou mais estados iniciais (origens) a um ou mais estados finais (destinos) e será rotulado por um evento, que por sua vez pode estar associado a uma condição representada entre parênteses. Veja a figura 2.5.

Semânticamente, o estado D da figura acima, representa um OU exclusivo

Figura 2.6:

(XOR) de A e C, ou seja, estar no estado D, significa estar em A e não estar em C ou estar em C e não estar em A. Percebe-se, assim, que D é uma abstração ou agregação de A e C. O estado D e o arco “a”, denotam uma propriedade comum de A e C; isto é, “a” tem origem tanto em A quanto em C e um destino comum que é o estado B.

Por outro lado, os estados A, B e C da mesma figura, podem ser decompostos em outros sub estados, isto é, eles podem ser refinados. Conclui-se, assim, que enquanto a abstração ou agregação é um processo do tipo ascendente (bottom-up), o refinamento é do tipo descendente (top-down).

A decisão de se permitir que transições partam de um superestado ao invés de partir de todos os subestados que o compõe, tal como visto na figura 2.5, é a principal forma de se economizar arcos num Statechart, tornando os diagramas graficamente mais claros.

Estados Defaults

Como já foi mencionado, o conceito de Máquinas de Estados Finitos é parte integrante dos Statecharts. Por isso, é necessário que se possua uma forma gráfica de representação dos “estados defaults”, ou seja, estados iniciais que o sistema deve assumir assim que for ativado. Suponha-se que na figura 2.5, A seja o estado default. Pode-se representar essa situação de duas maneiras diferentes, utilizando-se arcos diferenciados, como mostra a figura 2.6. À esquerda, uma representação em um passo. À direita, uma representação em dois passos, ou seja, D é o estado default entre D e B, e A também o é entre A e C.

2.7 Statechart : um exemplo

O exemplo que a partir de agora será usado para descrever a técnica Statechart consiste de um relógio digital de pulso. Apesar de ser um sistema relativamente simples, ele é suficientemente complexo para ilustrar as principais características

que envolvem a técnica. Esse exemplo foi retirado de [Har 87].

O relógio possui 5 áreas distintas no visor e 4 botões de controle a, b, c e d. Ele é composto por:

- Hora (Modo 24 horas ou do tipo am/pm)
- Data (Dia do mês, mês e dia da semana).
- Alerta (*beep* a cada hora, se habilitado).
- Dois alarmes independentes (Alarme1 e Alarme2).
- Cronômetro (Com indicativo de volta (*lap*)).
- Luz.
- Indicador luminoso intermitente (Se bateria ficar fraca).
- Teste de *beep*.

Os eventos externos são: pressionar e liberar os botões, denotados respectivamente por (tomando, como exemplo, o evento “a”) “a” e “â”. Existem também alguns eventos internos, os quais serão apresentados no decorrer deste capítulo.

A figura 2.7, mostra as transições entre o visor normal e os três estados de *beeps* existentes (*alarme1-beep*, *alarme2-beep* e *ambos*). T1 e T2 representam, respectivamente, as horas de ativação dos alarmes 1 e 2. T indica a hora corrente. P1 é a abreviação da condição: (*alarme1 – habilitado* $\wedge$ (*alarme2 – desabilitado* $\vee$ $T1 \neq T2$)). Isto significa que o beep de alarme1 só será ativado quando a hora em que foi programado (T1) coincidir com a hora corrente (T). Além disso, a condição P1 exige que, ou alarme2 esteja desabilitado ou a hora programada de ambos os alarmes não seja a mesma. O alarme2 funciona analogamente. P é a abreviação da condição (*alarme1 – habilitado* $\wedge$ *alarme2 – habilitado* $\wedge$ $T1 = T2$), ou seja, os dois alarmes serão ativados simultaneamente, somente se estiverem ambos programados na mesma hora e além disso, habilitados.

Note ainda que o estado *visor* é, na realidade, composto de outros estados (existem 5 áreas distintas no visor). Portanto, tem-se na figura 2.7, uma abstração deste estado. O refinamento do estado *visor* pode ser visto na figura 2.8. Essa figura mostra que, estando no estado *hora*, pressionando o botão “a” (evento externo a), ocorrerá uma transição para o estado *alarme1*. Novas ocorrências do evento a, provocarão novas transições de estados, a continuar por *alarme2*,

Figura 2.7:

alerta, cronômetro, retornando novamente a *hora*. Os estados *hora* e *data*, estão associados pelo evento “d”, ou seja, pressionando-se d, vai-se de *hora* para *data* e vice-versa. Porém, estando em *data* por dois minutos, retorna-se a *hora* devido ao evento interno 2-min-em-data.

2.7.1 História

Uma outra maneira de se atingir um grupo de estados é através da história (H) deste grupo, graficamente representada pela letra H inserida em um pequeno círculo. Ativar um determinado estado, dentro de um grupo de estados que possui uma história, significa considerar aquele estado que foi mais recentemente visitado.

A figura 2.9 corresponde ao refinamento do estado *alarme1* e mostra que ou entra-se no estado mais recentemente visitado ou entra-se no estado default *desligado*, se for a primeira vez. Isso significa que se o alarme estiver habilitado (estado *ligado*) e por ação de um outro evento abandonar-se esse estado e, posteriormente, retornar-se a ele (através da transição “a”), o estado a ser assumido será o próprio estado *ligado*.

Um fator importante a considerar nesta situação é que a história H se aplica apenas ao nível no qual ela aparece, ou seja, a ativação de um eventual sub-estado dos estados *ligado* e /ou *desligado*, não levará em consideração a história do estado *alarme1* pois este suposto sub-estado pertenceria a um nível mais interno, fugindo da ação da história do estado *alarme1*.

Figura 2.8:

Figura 2.9:

Figura 2.10:

Figura 2.11:

No entanto, pode-se fazer com que a história se aplique aos níveis mais internos no diagrama. Graficamente, isso é representado pelo símbolo “*” associado ao símbolo de história. Assim, na figura 2.10, o sistema ativará dentre A,B ,... ou E o estado que fora mais recentemente visitado, considerando os estados do nível mais interno ou estados atômicos.

O conceito de história também aparece no estado *atualização* refinado, como mostra a figura 2.11.

Estando no estado *atualização*, isto é, estando em algum dos 9 estados que o compõem, ao se pressionar “d” ocorrerá um abandono deste estado e uma entrada ao estado mais recentemente visitado. Porém, o estado mais recentemente visitado é aquele que acabara de ser abandonado. Então, pode-se concluir que a

Figura 2.12:

ocorrência do evento “d” em *atualização*, corresponde a 9 setas que tem origem e destino nos 9 sub estados que compõem o estado *atualização*. A figura 2.12, equivalente à figura 2.11, é uma representação alternativa para tal fato.

O conceito de história pode estar, ainda, associado ao conceito de estados defaults como mostra a figura 2.13. Numa situação como aquela, o sistema se comporta da seguinte maneira: ou assume o estado default *zero* (se for a primeira “visita” ao estado *cronômetro*) ou assume o estado mais recentemente visitado (caso já tenha estado em *cronômetro* anteriormente).

2.7.2 Ortogonalidade : concorrência

As características até aqui descritas apresentaram somente uma parte dos fatos: a decomposição exclusiva (XOR) de estados, isto é, dois estados nunca serão ativados ao mesmo tempo. Nesta seção será apresentada a decomposição AND, na qual estando em um estado, o sistema deve estar também em todos os seus componentes ANDs. Da mesma forma que nos higraphs, a ortogonalidade ou concorrência é representada por uma bolha na qual os componentes AND estão separados por uma linha pontilhada.

A figura 2.13 é um refinamento do estado *cronômetro* da figura 2.8. Os estados *normal* e *volta* são duas formas de exibição do visor e *zero* é um estado especial no qual o cronômetro está desativado, mas no seu estado inicial. Repare que a figura é auto explicativa: normalmente quando se seleciona o cronômetro de um relógio de pulso, o mostrador apresenta dígitos com valor zero (estado *zero*).

Figura 2.13:

Figura 2.14:

Ao acioná-lo (evento "b"), o tempo passa a ser contado e mostrado simultaneamente no visor (*Normal*). No entanto, ao se mudar a forma de exibição no visor (*Volta*), tem-se a impressão de que o cronômetro parou. Note, porém, que pela representação da concorrência, o tempo continua a ser contado, ou seja, o relógio está no estado *Volta* e ao mesmo tempo em *ligado*.

A concorrência aparece também nos níveis mais externos do relógio. Inicialmente o relógio pode ser visto como composto por dois grandes estados: *morto* (antes de se colocar a bateria) e *vivo* (em funcionamento), como mostra a figura 2.14.

Note que o estado *vivo* é composto por seis sub-estados ortogonais:

- *principal*: controla o estado *visor* e o tempo de disparo dos alarmes

Figura 2.15:

- *alarme1-status* : habilita/desabilita disparo do alarme1
- *alarme2-status*: habilita/desabilita disparo do alarme2
- *alerta-status*: habilita/desabilita o beep a cada hora cheia
- *luz* : permite acender a luz
- *bateria* : avisa caso a bateria esteja fraca.

Ativar o estado *vivo* significa ativar os seis sub estados simultaneamente nos seus respectivos estados defaults, ou seja : exibir as horas no visor; desabilitar alarme1, alarme2 e alerta; manter luz apagada e indicar que a bateria está "Ok". Veja figura 2.15.

2.7.3 Eventos Condicionados

A ortogonalidade dos componentes ilustrados na figura 2.15, pode levar a resultados inesperados. Suponha que o sistema esteja em qualquer sub estado do estado *atualização* (figuras 2.8 e 2.11). A eventual ocorrência do evento "b" vai provocar, além do abandono do estado *atualização*, o acendimento da luz.

Para evitar resultados indesejáveis, pode-se associar a um determinado

evento, uma certa condição, como é o caso do estado *alarme1-status*. Note que os estados de habilitado ou desabilitado de *alarme1-status* estão diretamente associados aos sub-estados *ligado/desligado* do estado *alarme1* da figura 2.9, isto é, o *alarme1* só é desabilitado se o sub-estado *ligado* de *alarme1* for o estado corrente, satisfazendo a condição *alarme1-ligado*.

2.7.4 A Ação “Limpa-História”

Como visto anteriormente, numa entrada do tipo H, deve-se assumir o estado que fora mais recentemente visitado. Suponha, no entanto, que no exemplo do relógio digital ocorra o evento “bat-remov” ou “bat-fraca”, como mostra a figura 2.15. Numa situação possivelmente real, pode-se esperar que ao recolocar a bateria ou substituí-la por uma nova, o processo seja reinicializado como se fora a primeira vez, ou seja, o sistema deve “esquecer” a história H dos estados (caso estes a possuïrem) e ativar os estados defaults.

Esta situação é representada utilizando-se (associada a um evento) as ações *clh(estado)* e *clh(estado *)* que provocam, respectivamente, o “esquecimento” dos estados mais recentemente visitados num primeiro nível e em todos os níveis. Uma vez “esquecidas”, as entradas do tipo H não se aplicam e, após a reinicialização, o sistema assume os estados defaults. O termo *clh* é a abreviação da expressão inglesa, *clear-history*, aqui designada por “limpa-historia”.

2.8 Eventos, Ações e Atividades

Statechart é uma técnica de representação do controle de um sistema, ou seja, uma forma de representar como um sistema toma decisões dependentes do tempo e que influenciam todo o seu comportamento.

No entanto, a reatividade do statechart descrito na seção 2.7 foi expressa em função de simples mudanças nos estados internos do sistema, em resposta aos eventos e condições existentes. Não foi descrito, por exemplo, como o sistema responde a esses eventos.

Falta, então, incorporar a essa técnica a habilidade de gerar eventos e mudar os valores das condições. Isso pode ser feito através de ações que normalmente vêm associadas a um evento, constituindo-se no que se denomina rótulo de uma transição.

Figura 2.16:

Figura 2.17:

Graficamente, tem-se a figura 2.16, onde E é um evento e A uma ação de ocorrência instantânea a ser executada pelo sistema. No rótulo (E/A), a ocorrência de E, além de disparar a transição de um estado para outro, provoca a execução da ação A, que pode ser, tanto um evento de saída do sistema, quanto a causa de uma nova transição dentro do sistema. Suponha, para exemplificar esse segundo caso, que uma determinada transição rotulada por um evento E, queira provocar uma outra transição (figura 2.17) sem que haja, necessariamente, efeitos externos imediatos. Ao “sentir” E, o sistema provoca a primeira transição e a ação A será executada, gerando A como um novo evento que possa ser “sentido” em qualquer lugar do sistema. Dessa forma, a transição A do componente do lado direito da figura 2.17, ocorre simultaneamente à transição do lado esquerdo. Vê-se, portanto, que eventos e ações estão fortemente relacionados. A essa característica, dá-se o nome de *Broadcasting* ou reação em cadeia.

No entanto, ações não são suficientes para representar como o sistema responde a determinados eventos. É preciso que existam atividades associadas a estas ações. Uma atividade está para uma ação, assim como uma condição está para um evento. Ao contrário das ações, as atividades são ocorrências que demandam tempo maior que zero. Pode-se dizer que as atividades são duráveis, enquanto que as ações são instantâneas. Uma ação pode ser executada ao se entrar e/ou sair de um determinado estado e uma atividade é caracterizada pela sua execução enquanto o sistema permanecer na bolha associada.

O fator Tempo

Nos parágrafos anteriores, foram introduzidas as idéias de ações e atividades aos statecharts e com elas, a necessidade de considerar o tempo como um

fator relevante dentro do processo de especificação dos Sistemas Reativos.

No entanto, o tempo é irrelevante para efeito de descrição do comportamento de um sistema reativo, uma vez que a análise do comportamento considera, apenas, o estado inicial e o escopo de um processo do sistema [Jan 86]. O tempo passa a ser importante a partir do momento que se quer simular o funcionamento de um sistema.

Portanto, o fator tempo será considerado aqui, apenas como elemento parte da especificação de um sistema, do mesmo modo como serão especificados os estados, transições, ações, etc., pois não faz parte do escopo desse trabalho simular o funcionamento do statechart gerado.

Fortes, em [For 91], mostra uma possibilidade de simulação para os Statecharts, definindo para isso, uma forma de modelagem do tempo.

2.9 Considerações Finais

Nesse capítulo descreveu-se a técnica denominada STATECHART, para especificação de Sistemas Reativos ressaltando-se três aspectos importantes:

- A necessidade de uma descrição comportamental clara e rigorosa como elemento principal no processo de especificação de Sistemas Reativos.
- Os Statecharts como uma técnica formal para especificação do comportamento reativo.
- A importância de linguagens visuais que possam explorar as vantagens da interação gráfica homem-máquina.

Resumindo: sistemas de natureza complexa, como os Sistemas Reativos, podem ser representados por um formalismo visual. O termo visual significa que estes sistemas serão gerados, compreendidos e farão comunicação com seres humanos e o termo formal significa que serão manipulados, mantidos e analisados por computador.

Harel, em [Har 87] vai mais longe e sugere que o formalismo visual seja utilizado como principal ferramenta para a construção de sistemas, fazendo com que as representações textuais sejam utilizadas como complemento.

Capítulo 3

Revisão Bibliográfica

3.1 Considerações Iniciais

Na literatura da Engenharia de Software pode-se encontrar vários estudos sobre técnicas destinadas a auxiliar no processo de desenvolvimento de softwares dos mais variados tipos. Da mesma maneira, existe uma grande quantidade de ferramentas automatizadas que, incorporando uma ou mais técnicas, contribuem para o processo de desenvolvimento de software.

O objetivo deste capítulo é fazer um levantamento das principais técnicas e ferramentas existentes, levando-se em consideração os seguintes aspectos:

- O tema central deste trabalho é o processo de representação dos Sistemas Reativos, logo, as técnicas e ferramentas apresentadas são voltadas para esse fim.

- A importância do aspecto visual e formal como maneira de facilitar a descrição e manter o rigor dos Sistemas Reativos. Assim, todas as técnicas apresentadas têm, na sua essência, elementos gráficos como principal arma de realização das tarefas a que se propõem.

Ao final deste capítulo é feita uma avaliação das ferramentas apresentadas, mostrando-se através de uma tabela, as principais características que elas apresentam, sobretudo no aspecto gráfico e de facilidade de operação por parte do usuário.

3.2 Técnicas para Representação do Aspecto Comportamental dos Sistemas Reativos

3.2.1 Máquinas de Estados Finitos

Segundo Brady [Bra 77], o termo algoritmo ou “procedimento mecânico” deixa transparecer a idéia de que qualquer computação pode ser especificada em detalhes suficientes para ser executada automaticamente por alguma máquina.

Um primeiro passo no sentido de se desenvolver uma teoria de máquinas de computação, segundo Brady, é considerar a máquina como uma caixa preta levando em consideração apenas o comportamento da mesma, diante das possíveis entradas e saídas (inputs e outputs, respectivamente). Esse tipo de máquina é conhecido como Máquina Básica.

Porém, existem situações em que as máquinas básicas não se aplicam. Considere um interruptor de luz que possui uma entrada (press-botão) e duas saídas (ligado e desligado). Ao se pressionar o botão, o que determina a saída (ligado ou desligado) é a noção de estado que a situação oferece. Isso quer dizer que ao se pressionar o botão e a luz estiver no estado claro, a saída será: desligado. Caso esteja no estado escuro, a saída será: ligado. Portanto, uma máquina para representar tal situação deve, também, incluir o conjunto de estados possíveis.

Brady em [Bra 77], define uma Máquina de Estado Finito(MEF) como sendo um par de funções do tipo:

$$mFn : I \times S \Rightarrow O ; sFn : I \times S \Rightarrow S$$

onde I , O e S são conjuntos finitos de entrada, saída e estados, respectivamente. O termo mFn representa a função máquina (machine function) e sFn , a função estado (state function).

Para o exemplo da máquina interruptor descrita anteriormente, tem-se:

$I \times S \Rightarrow O$		$I \times S \Rightarrow S$	
press-botão, claro	desligado	press-botão, claro	escuro
press-botão, escuro	ligado	press-botão, escuro	claro

Ainda em [Bra 77], Diagramas de Transição de Estados (DTEs) foram utilizados para representar graficamente as MEFs. Cada estado corresponde a

Figura 3.1:

Figura 3.2:

um círculo. Uma seta conecta os círculos correspondentes a dois estados $s1$ e $s2$, se houver alguma entrada i , tal que $sFn(i,s1) = s2$. A entrada i e a saída associada a ela, $O = mFn(i,s1)$ são representadas junto à seta que interconecta os círculos. Veja figura 3.1.

A figura 3.2, mostra o DTE para a máquina interruptor.

Jacob em [Jac 83], utiliza as MEFs como uma técnica para especificação formal do módulo de interface com o usuário de um Sistema Militar de Mensagens (SMM), desenvolvido no Naval Research Laboratory, Washington. Jacob mostra, pela figura 3.3, como uma tarefa de acesso ao SMM (verificação de nome, senha, etc.) pode ser formalmente especificada utilizando uma MEF.

Os círculos e os nomes neles contidos, representam os estados. Os arcos representam as transições que são rotuladas por um nome que corresponde a um comando de entrada definido pela Linguagem de Comandos do Usuário (LCU). A esse comando pode estar associado uma ficha (token) que contém uma condição booleana e/ou uma resposta do sistema e/ou ações. Uma transição ocorre quando um comando de entrada é recebido e a condição for satisfeita. Nesse ponto, o sistema mostra a resposta (se existir) e realiza a ação que é uma sentença da Linguagem de Comandos Intermediários (LCI) controlada pelo gerenciador de dados do sistema SMM.

Figura 3.3:

- (1) resp : "Entre com o nome"
- (2) cond : not EXISTE_USUÁRIO(\$USUÁRIO)
resp : "Nome incorreto, entre com novo nome"
- (3) cond : EXISTE_USUÁRIO(\$USUÁRIO)
resp : "Entre com a senha"
- (4) cond : \$SENHA ≠ SENHA_DO_USUÁRIO
resp : "Senha incorreta, entre com uma nova senha"
- (5) cond : \$SENHA ≠ SENHA_DO_USUÁRIO
resp : "Senha incorreta, re-inicie o processo"

Sunshine, em [Sun 82], mostra como o sistema AFFIRM, utilizado para especificação e verificação de tipos abstratos de dados, pode ser utilizado para especificar e verificar protocolos de comunicação quando associados às MEFs.

Segundo Sunshine, as MEFs podem expressar a maioria dos eventos e ações que ocorrem em um sistema de protocolo através da utilização dos seguintes elementos :

- . Retângulos - Os retângulos e os nomes neles contidos, correspondem aos estados (status) que o serviço pode assumir, como por exemplo, ProntoParaEnviar, Enviando, ProntoParaReceber, etc..

- . Arcos - Representam as transições e são rotulados por uma variável de estado, como por exemplo: buffer que corresponde a uma fila de mensagens enviadas e ainda não recebidas e/ou uma transição (envio-completado, inicializar-serviço, etc.).

Uma vez definido, o diagrama pode ser convertido em uma linguagem interpretada pelo sistema AFFIRM. Em seguida, o núcleo do sistema inicia o processo de verificação do que foi especificado pelo usuário.

Figura 3.4:

3.2.2 Redes de Petri

Segundo Peterson, em [Pet 77], uma Rede de Petri (RP) é um modelo abstrato e formal de fluxo de informações. O objetivo principal ao se pesquisar as RPs é o de se estabelecer métodos simples, naturais e ao mesmo tempo poderosos para descrição e análise do fluxo de controle e da informação de sistemas, em particular os que apresentam características concorrentes e assíncronas.

Peterson define, graficamente, uma RP da seguinte maneira: Uma RP é um grafo que contém dois tipos de nós. Os lugares (places) e as transições (transitions) representados, respectivamente, por círculos e barras. Estes nós são conectados por arcos dirigidos com origem nos lugares e destinos nas transições e vice-versa. Se um arco é dirigido de um nó i para um nó j , diz-se que i é uma entrada (input) para j e que j é uma saída (output) para i . Na figura 3.4, por exemplo, o lugar $P1$ é uma entrada para a transição $t2$ enquanto os lugares $P2$ e $P3$ são saídas para a transição $t2$.

Uma RP pode ser executada, daí a sua característica dinâmica. A execução de uma RP é controlada pela posição e movimento das marcas (tokens), que são pontos residentes nos nós lugares. Para se executar uma RP deve-se obedecer a certas regras:

- 1 - Verificar se uma transição T está habilitada (enabled), ou seja, se todos os seus lugares de entrada têm pelos menos uma marca.
- 2 - Estando habilitada, qualquer transição pode ser disparada (fire).
- 3 - Disparar uma transição T , significa remover as marcas dos lugares de entrada de T e colocar novas marcas nos lugares de saída de T .
- 4 - As transições podem ser disparadas enquanto houver transições habilitadas. Caso contrário, a execução da RP termina.

As Redes de Petri têm sido utilizadas de maneira muito variada. No caso deste trabalho, a inclusão de RP, como objeto de estudos, se deve ao fato delas se constituírem numa forma eficiente para a representação dos aspectos comportamentais dos Sistemas Reativos.

Baer, em [Bae 86], mostra a utilidade das RPs na modelagem de arquiteturas avançadas de hardware, como por exemplo: os multi-processadores. Segundo Baer, o fato dessas arquiteturas apresentarem paralelismo e características de "pipeline", fazem com que as RPs sejam particularmente úteis para representá-las. Para tanto, é mostrado como paralelismo e "pipeline" podem ser representados por RPs.

Em [Rei 86], é proposta uma metodologia para desenvolvimento de sistemas baseada em RPs. Reisig mostra como as RPs possibilitam uma transição sistemática de especificações informais e imprecisas em especificações formais e precisas. O resultado dessa especificação é uma representação do comportamento dinâmico do sistema sob observação.

É mostrado, ainda, como esta metodologia abrange praticamente todo o ciclo de vida do sistema, dispensando o uso de técnicas adicionais de representação.

3.2.3 F-Nets e R-Nets

Em 1981, O U.S. Ballistic Missile Defence Advanced Technology Center (BM-DATC) patrocinou a criação de um sistema para especificação e projeto aplicável a sistemas de tempo real, conhecido como DCDS ou Distributed Computing Design System [Alf 85].

O sistema inclui uma metodologia de especificação de requisitos de sistemas (SYSREM - System Requirements Engineering Methodology) e uma metodologia de especificação de requisitos de software (SREM - Software Requirements Engineering Methodology).

SYSREM é baseada em um modelo gráfico do tipo estímulo-resposta chamado de F-Nets. Esse modelo é composto por nós e arcos que representam, respectivamente, funções do sistema e eventos ou fluxos. O sistema "aceita" uma sequência de entradas (inputs) e produz uma sequência de saídas (outputs). Num primeiro nível, esse modelo gráfico é equivalente a um conjunto de máquinas de estados que recebem entradas e produzem saídas. Algumas dessas máquinas correspondem a operações realizadas por software e outras, por hardware.

Uma vez definidos os processos do sistema como um todo, a metodologia SREM é utilizada para detalhar os processos relativos ao software. O comportamento de cada processo pode ser mostrado através de diagramas chamados de R-Nets (Requirement Networks).

Uma R-Net é constituída por :

- Uma única entrada: representa uma interface para o ambiente.
- Saídas: interface para o ambiente ou símbolos terminais.
- Subredes: as R-Nets são compostas por subredes (subnets) similares. Qualquer R-Net pode ser decomposta em subredes, possibilitando decomposição hierárquica.
- Funções: as funções em uma R-Net não podem ser decompostas e são conhecidas por ALPHAS.
- Nós do tipo OR: especificam condições de processamento (seleção).
- Nós do tipo AND: representam concorrência.
- Eventos.

Davis, em [Dav 88], mostra como uma R-Net pode ser utilizada para especificar o comportamento de um sistema de discagem telefônica (figura 3.5). Os pentágonos ao início e ao final do diagrama correspondem, respectivamente, aos símbolos de início e finalização da R-Net. A mensagem dentro do hexaedro representa um estímulo ou interface do ambiente, enquanto as mensagens dentro dos retângulos podem ser ações, geração de saídas ou seleção de dados. Os círculos podem ser pontos de validação, seleção ou concorrência, sempre que contenham um "V", "+" ou "&", respectivamente.

Em [Whi 87] tem-se um exemplo do uso de SYSREM para especificação dos requisitos de um sistema de aquecimento residencial. É apresentada uma F-Net na qual cada retângulo especifica uma função realizada pelo sistema. As setas representam o fluxo de informação entre uma função e outra. Alguns fluxos são rotulados por eventos. Os subsistemas de software são especificados através de R-Nets.

Figura 3.5:

3.2.4 Esquema de Transformação

Ward, em [War 85], apresenta uma extensão dos Diagramas de Fluxo de Dados (DFD) [Gan 79], chamada de Esquema de Transformação (Transformation Schema), capaz de representar aspectos temporais e de controle existentes em sistemas de tempo real. Segundo Ward, esta extensão se fez necessária pelos seguintes motivos:

- Em muitas aplicações em tempo real, o sistema deve ser capaz de distinguir entre um fluxo de dados discreto (por exemplo, o valor de uma variável requisitada em uma determinada unidade de tempo) e um fluxo de dados contínuo (variáveis continuamente requisitadas durante um certo intervalo de tempo, por exemplo).

- Necessidade de representar fluxos de eventos. Eventos traduzem um acontecimento ou um comando discreto no tempo que não possui valor como variável. Os eventos podem ser de três tipos: de sinal, ativação e desativação. O primeiro mostra a intenção do remetente em dizer que algo aconteceu. O segundo, a intenção do remetente em fazer com que o receptor produza alguma saída (output). E o terceiro, a intenção do remetente em evitar que o receptor produza alguma saída.

- Necessidade de representar controles. Num DFD convencional, um círculo representa uma transformação de dados, isto é, centros que transformam fluxos de entrada em fluxos de saída. Os controles ou centros de transformação

Figura 3.6:

de controle, controlam o comportamento dos centros de transformação de dados, ativando-os e desativando-os. Em outras palavras, um centro de transformação de controle é uma abstração de uma parte da lógica do sistema. Essa lógica é representada através de uma máquina de estados.

- Necessidade de buffers, dispositivos de armazenamento temporário. Os buffers representam, na realidade, pilhas ou filas.

Em [War 86], Ward aplica o Esquema de transformação para descrever o comportamento de um sistema de controle de processo, responsável pela monitoração da temperatura e Ph de um reservatório de água. É definido um esquema de transformação onde existem centros de transformação de dados e controles interrelacionados (figura 3.6). A lógica do centro de transformação de controle é representada por uma máquina de estados, na qual pode-se verificar como ele é capaz de ativar e desativar centros de transformação de dados.

Ward mostra, ainda, como executar um Esquema de Transformação. Essa execução é semelhante à realizada nas Redes de Petri e é visualizada em termos da colocação de marcas (tokens) nas transformações, fluxos e buffers.

3.2.5 Statechart

Statecharts são extensões de Máquinas de Estados Finitos (MEFs) propostas por Harel ([Har 87,Har 88]). Os Statecharts possuem uma notação e um conjunto de regras que facilitam a decomposição hierárquica das MEFs. Além disso, possuem mecanismos de comunicação e representação de concorrência entre estados. Maiores detalhes sobre a técnica statechart, podem ser encontrados no capítulo 2, seção 2.6 a 2.7 deste trabalho.

Davis em [Dav 88], utiliza Statecharts para descrever o comportamento de um sistema de discagem telefônica e as vantagens que essa técnica possui em relação às Máquinas de Estados Finitos (MEFs). Pela comparação apresentada por Davis, pode-se notar, claramente, que os Statecharts possibilitam uma representação mais clara, sucinta e natural do que a oferecida pelas MEFs.

Por ser baseado em um formalismo visual, os Statecharts apresentam sintaxe e semântica formalmente definidas. Maiores detalhes podem ser encontrados em [Har 87b].

3.2.6 Tabelas e Árvores de Decisão

Segundo Moret, em [Mor 82], uma tabela de decisão pode ser vista como uma matriz onde as linhas superiores especificam um conjunto de condições e as linhas restantes, um conjunto de ações a serem realizadas quando as condições forem satisfeitas. Cada coluna, chamada de regra, descreve um procedimento do tipo “Se condição, Então ação”. Como exemplo, tem-se:

Chovendo? Condicao do Vento	Sim	Não Brisa	Não Calmo	Forte
Assistir TV	X			X
Cuidar Jardim			X	
Soltar Pipa		X		

A tabela acima mostra como passar uma tarde de sábado. Têm-se duas condições: *chovendo?* e *condição do vento* que podem assumir, respectivamente, os valores *sim-nao* e *brisa-calmo-forte*; três ações (descritas nas três últimas linhas da tabela) e quatro regras. A presença de um símbolo “X” na mesma linha de uma ação, significa que esta deve ser executada.

Nesse mesmo artigo, Moret define uma árvore de decisão como um algoritmo determinístico que decide qual a próxima variável a ser testada, baseado em

variáveis previamente testadas e nos resultados de sua avaliação. As árvore e as tabelas de decisão capturam a mesma informação. A diferença é que a primeira é uma representação gráfica e a segunda, tabular.

Em [Dav 88], são mostradas tabelas e árvores de decisão de um sistema controlador de portas de um elevador. Segundo Davis, a utilização dessas técnicas nesse sistema é motivada pela presença de procedimentos do tipo “Se condição, então ação”, como por exemplo:

- Se botão fecha-porta for pressionado, então fechar porta.
- Se tempo de espera excedido, então fechar porta.
- Se elevador parou e botão fecha-porta pressionado, então abrir porta.

As árvores de decisão têm uma aplicação muito vasta. Em [Won 76], por exemplo, utilizam-se árvores de decisão para descrever o funcionamento do algoritmo de processamento de *queries* do sistema gerenciador de banco de dados INGRES.

3.3 Ferramentas Automatizadas para Descrição de Sistemas Reativos

A seguir, serão mostradas oito ferramentas, sendo que cinco delas foram desenvolvidas especificamente para especificação e simulação de Sistemas Reativos: Statemate, Argonauta, Teamwork/ES, Protob e TAGS. As outras três (ProMod, Excelsator e IEW), apesar de não terem sido projetadas especialmente para este fim, podem ser utilizadas no desenvolvimento de Sistemas Reativos devido ao alto grau de sofisticação e confiabilidade que fornecem ao usuário.

Apesar de cada ferramenta apresentar características particulares no que diz respeito às funções e às técnicas suportadas, elas possuem dois fatores em comum: o formalismo e o grande apelo visual. O fator formal se deve ao fato dessas ferramentas permitirem análise e simulação dos sistemas gerados e o visual porque, em todas elas, a principal forma de alimentação é feita graficamente, através de editores gráficos.

3.3.1 ProMod

ProMod (Project Model) [Hru 83] é um ambiente de desenvolvimento de software baseado no paradigma do ciclo de vida dos sistemas [Pre 87]. ProMod é um ambiente integrado, onde cada fase do ciclo de vida suportado pelo ambiente é baseada em um método (tabela abaixo). Por sua vez, cada método é suportado por um conjunto de ferramentas que possibilitam a automatização de um grande número de atividades.

Fase	Método
Análise de Requisitos	Análise Estruturada
Projeto de Arquitetura	Projeto Modular
Projeto de Programa	Pseudo Código
Implementação	Programação Estruturada

A seguir serão descritas as ferramentas existentes em cada fase do ProMod.

Análise de Requisitos

O ProMod considera três objetos básicos para análise: Diagramas de Fluxo de Dados (DFDs), Dicionários de Dados e Mini-especificações (mini-specs) que são pequenas especificações derivadas dos DFDs. As ferramentas são:

- Editor de texto interativo, orientado a sintaxe. Utilizado na definição de elementos do dicionário de dados e das mini-especificações.
- Editor gráfico interativo para DFD.
- Analisador de requisitos: checa a consistência e completeza dos objetos e prepara relatórios requisitados pelo analista de sistema.

Ambos os editores citados acima fazem checagem das regras sintáticas e semânticas existentes no método e somente os objetos corretos são armazenados pelo ProMod.

Projeto da Arquitetura

- Transformador: programa que transforma as informações geradas na fase de análise em um módulo de estruturas hierárquicas.
- Editor de diagramas de estruturas interativo e orientado a sintaxe: utilizado para completar e modificar os diagramas de estruturas construídos pelo

transformador.

Projeto de programa e implementação

O ProMod permite o uso de pseudo-código na geração dos programas. As ferramentas presentes nesta etapa realizam a checagem da sintaxe e a manutenção automática desse pseudo-código. A partir do pseudo-código, outras ferramentas de apoio propiciam a criação automática de tipos e estruturas de dados, chamadas de funções, laços e estruturas de decisão.

A interface com o usuário

O usuário “vê” o ambiente Promod através de uma única interface. Independente da módulo em que estiver trabalhando, existe um conjunto de comandos que permite a comunicação entre os vários módulos do sistema, especialmente com a base de dados central.

Todo comando obedece a uma sintaxe pré-definida e pode ser executado através de uma chamada abreviada de seu nome. Além disso, o ProMod é capaz de “lembrar” estados internos de modo que, ao se trabalhar por algum tempo com um mesmo objeto, o ProMod “memoriza” o nome desse objeto evitando que, ao ser requisitado, o usuário tenha que repetir o nome por várias vezes.

3.3.2 TAGS

O TAGS (Technology for the Automated Generation of Systems) é uma ferramenta CASE destinada a aplicações em tempo real [Fis 89], baseada no Paradigma Centrado em Automação (Automation-based Paradigm) proposto por Balzer, em [Bal 83].

O ambiente TAGS é composto por três elementos: a linguagem IORL (Input/Output Requirements Language), um conjunto de ferramentas automatizadas e a metodologia TAGS que trabalham juntas no sentido de apoiar o processo de construção de software.

Para se construir um software com essa ferramenta, os projetistas geram a especificação do mesmo em IORL baseados na metodologia TAGS. Tendo a IORL armazenada em uma base de dados, o TAGS é capaz de fazer a análise sintática e semântica da IORL, simulação e o controle de versões. Estas funções são realizadas, respectivamente, pelas ferramentas: analisador de diagnóstico, compilador de simulação e gerenciador de configuração.

A linguagem IORL

A linguagem IORL é uma linguagem de especificação com recursos gráficos e textuais que permite aos projetistas identificar cada componente do sistema durante a fase de projeto. Ela é uma linguagem de fácil manipulação, aplicável a todo tipo de sistemas (não apenas aos sistemas computacionais) que permite a utilização de notação matemática para expressar algoritmos e aspectos de desempenho dos sistemas projetados.

O nível mais alto da IORL consiste dos Diagramas Esquemáticos de Blocos ou SBD (Schematic Blocs Diagrams). Esses diagramas são compostos por retângulo e arcos representando, respectivamente, processamentos e fluxos de dados entre os processos. Os processamentos podem ser refinados, isto é, o TAGS permite que para cada processamento seja criado um novo SBD.

Uma vez construídos os SBDs, podem-se utilizar os Diagramas de Temporização e Relacionamento de Entrada e Saída ou IORTD (Input/Output Relationships and Timing Diagrams) para representar o fluxo de controle lógico e temporal dos processos descritos pelos SBDs. Nesse nível, a IORL oferece recursos para representação de processos concorrentes.

Os IORTDs, normalmente, vêm associados aos Diagramas de Processos Pré-definidos ou PPDs (Predefined-process Diagrams) que são similares aos IORTD em estrutura. O objetivo desses diagramas é o de aumentar a legibilidade da especificação, permitindo que ela possa ser definida hierarquicamente até o nível de detalhamento desejado.

Tendo-se a especificação completa (através dos diagramas acima citados), o projeto pode ser analisado, simulado e atualizado, utilizando-se as ferramentas do TAGS citadas anteriormente. O TAGS, neste ponto, é capaz de gerar, a partir destas especificações em IORL, o código fonte em ADA.

A importância do TAGS como ferramenta de desenvolvimento de software deve-se, sobretudo, à maneira automatizada pela qual uma especificação pode ser traduzida em código fonte e pelo fato da manutenção ser automaticamente feita sobre a especificação e não sobre o código fonte. Porém, deve ser ressaltada a maneira pela qual esta especificação é obtida: a utilização de recursos gráficos sob a forma de diagramas (SBD, IORTD e PPD) formalmente gerados, obedecendo a sintaxe e semântica pré-definidas pela linguagem IORL. Em [Fis 89] são mostrados detalhes da interface gráfica do TAGS com o usuário.

3.3.3 Excelerator

O Excelerator é um software desenvolvido com o objetivo de projetar e documentar sistemas altamente complexos [Aar 85]. A metodologia de projeto adotada pelo Excelerator é a Análise Estruturada de Sistemas proposta por Gane e Sarson em [Gan 79].

O Excelerator é composto por seis seções, dentre as quais uma seção gráfica responsável pela edição dos DFDs. Nessa seção, a tela é dividida verticalmente em duas partes: a primeira contém os símbolos utilizados na criação de DFDs e a segunda é a área de trabalho, utilizada para construção do DFD desejado. Através de um mouse, o usuário pode selecionar da lista de símbolos (na primeira parte da tela), aqueles que deseja para compor o DFD (na segunda parte). Além disso, o usuário pode decompor o Diagrama em até dez níveis hierárquicos. Graficamente, o Excelerator realiza essa tarefa, utilizando técnicas de *zoom*, tanto para visualizar os níveis mais internos, quanto para retroceder a níveis mais externos.

Ao lado dos recursos gráficos, o Excelerator possui um dicionário de dados que funciona como um depósito de informações sobre o sistema que está sendo projetado. Os relatórios emitidos a partir desse dicionário serão úteis na avaliação do projeto. O acesso a essas informações é feito através do próprio editor gráfico via mouse.

3.3.4 Statemate

O Statemate é um ambiente gráfico automatizado constituído por pacotes de consultas, teste e simulação para sistemas reativos, que permite a geração automática de código e protomodelagem de sistemas [Har 88a].

O Statemate realiza essas tarefas especificando e avaliando o Sistema Sob Desenvolvimento (SSD) a partir de três visões: estrutural, funcional e comportamental. A visão estrutural corresponde à decomposição hierárquica do SSD em seus componentes físicos chamados módulos e à identificação dos dados e dos sinais de controle que fluem entre eles. A visão funcional corresponde à decomposição funcional do SSD, ou seja, a hierarquização das atividades do SSD com a descrição dos detalhes dos dados e dos sinais de controle existentes entre os módulos. A visão comportamental se encarrega da especificação do controle e relações temporais existentes no SSD.

Para possibilitar a especificação e análise do SSD sob essas três visões,

Figura 3.7:

o Statemate dispõe, para cada uma delas, de uma linguagem gráfica manipulada por meio de um editor gráfico que checa a validade sintática das especificações à medida que estas vão sendo introduzidas. Basicamente, a maior parte dos dados introduzidos no Statemate são provenientes desses três editores gráficos, além de um outro editor para informações não gráficas.

As três linguagens gráficas ou Diagramas de Módulos (Module-charts), Diagramas de Atividades (Activity-charts) e Statecharts que correspondem, respectivamente, às visões estrutural, funcional e comportamental, são todas baseadas em um conjunto de convenções gráficas simples, onde cada qual possui uma semântica formal embutida no Statemate. Pode-se dizer que o Statemate é um exemplo real do formalismo visual proposto por Harel em [Har 87, Har 88]. A figura 3.7, representa a estrutura geral do Statemate.

Graficamente, os Diagramas de Módulos e os Diagramas de Atividades são similares, ou seja, tanto retângulos quanto arcos são utilizados para representar módulos e fluxos de dados (nos Diagramas de Módulos) bem como atividades e fluxos de controles (Diagramas de Atividades). Para que não haja confusões, o Statemate emprega convenções gráficas distintas, utilizando cores e traçados de arcos diferentes para ambos os diagramas de forma que o usuário possa facilmente distinguir, no vídeo, um diagrama do outro.

Além de realizar análise sintática e atualização da base de dados de forma *on-line*, os três editores gráficos do Statemate possuem opções de cópia, movi-

mentação, *zoom* e refinamento, aplicáveis a todos os elementos dos diagramas, quer sejam módulos, atividades, estados, fluxos de dados, etc. As informações não gráficas podem ser atualizadas e visualizadas via editor de formulários. A interação com o usuário é feita via menus com a utilização de mouse.

3.3.5 IEW

Em [Jun 88] descreve-se a ferramenta CASE IEW (Information Engineering Workbench), que proporciona um ambiente integrado dirigido para as fases de especificação, análise e projeto estruturado de sistemas que, sob certas limitações, pode ser utilizado no desenvolvimento de Sistemas Reativos.

A principal ferramenta do IEW é a IEW/AW (IEW/Analysis Workstation) responsável pela criação, manutenção e consistência das informações produzidas, sendo que para realizar essa última tarefa, a IEW/AW utiliza-se de técnicas de inteligência artificial.

A principal forma de entrada de dados no IEW/AW se dá através de quatro tipos de diagramas diferentes: decomposição, entidade, fluxo de dados e ações, responsáveis, respectivamente, pela representação das estruturas hierárquicas em diferentes níveis, identificação dos tipos de entidades envolvidas na execução de uma tarefa, criação de DFDs correspondentes às atividades e, por último, descrição dos procedimentos e atividades.

A maior parte das operações é realizada via mouse, sob comando do Gerenciador do Ambiente Gráfico. Através dessas operações (que compõem o coração da interface do IEW/AW com o usuário) é possível visualizar, simultaneamente, diagramas, detalhes da descrição de processos e outras informações dentro do contexto. O uso de cores ajuda a distinguir os diversos conjuntos de informações.

3.3.6 Teamwork/ES

Em [Blu 88], Blumofe apresenta o Sistema Teamwork/ES (Teamwork/Executable Specification) destinado a executar (simular), de forma gráfica e interativa, especificações de análise estruturada, segundo a extensão proposta por Ward em [War 86], conhecida por Esquema de Transformação e já comentada neste trabalho.

O Teamwork/ES é um sistema de simulação, integrado ao sistema de

Figura 3.8:

modelagem para tempo real Teamwork/RT. A execução do sistema é visual e corresponde a movimentação de marcas (tokens) ao longo dos Diagramas de Fluxos de Dados estendidos (Esquemas de Transformação) e ao longo de Diagramas de Transição de Estados. A interface entre o usuário e o sistema é feita por meio de um Painel de Execução, através do qual o usuário pode parametrizar e controlar a execução, utilizando-se de operações como: iniciar, terminar, congelar, retroceder, avançar, etc. Paralelamente à execução visual do sistema é possível fazer alterações nos diagramas, isto é, adicionar, apagar e mover fluxos, estados e centros de transformações. O Teamwork/ES ainda é capaz de realizar detecção de impasse (deadlock).

3.3.7 Argonauta

Em [Mar 89] descreve-se o sistema Argonauta projetado para descrever, especificar e verificar sistemas reativos, tais como protocolos de comunicação, sistemas em tempo real e interfaces homem-máquina. A descrição geral do sistema é dada pela figura 3.8. Ao conjunto de ferramentas disponíveis é dado o nome de sistema Argonauta.

A linguagem Argos é baseada nos Higraphs e tem sua semântica definida por uma álgebra de processos. A sintaxe visual da linguagem é bastante parecida com a dos Statecharts. O ambiente gráfico do sistema Argonauta permite a descrição gráfica de Sistemas Reativos e a especificação das propriedades dos

sistemas através de um editor de fórmulas. A partir dessas entradas, o Sistema Argonauta permite a simulação (inclusive visual) do sistema descrito, junto com a avaliação das fórmulas.

O Editor de Descrição de Sistemas permite que o usuário crie e manipule estruturas gráficas através de menus, de ícones e de funções básicas existentes no gerenciador de janelas. O Editor de Fórmulas funciona interativamente com o Editor de Descrição de Sistemas, de maneira que, ao se definir graficamente um estado qualquer de um determinado sistema, pode-se atribuir a ele uma propriedade (fórmula), através do Editor de Fórmulas.

3.3.8 Protob

Em [Bld 88], descreve-se a ferramenta CASE orientada a objeto Protob, baseada em Redes de Petri de alto nível chamadas de Redes de Prot (Prot nets) que se destina a apoiar atividades de especificação, modelagem e prototipagem de sistemas distribuídos, tais como Sistemas em Tempo Real Embutidos e Protocolos de Comunicação.

A grande força da ferramenta Protob é a sua linguagem de modelagem, que possui uma representação gráfica e semântica própria. A representação gráfica, baseada em estados, transições e mensagens, permite que o usuário modele facilmente o comportamento do sistema sob especificação. Essa modelagem é baseada no conceito de objetos, cada qual representado por uma Rede de Prot que é uma linguagem de descrição gráfica composta de elementos como lugares(places), portas de entrada/saída, arcos e transições, cada qual representado por um símbolo gráfico diferente.

O Protob permite que os objetos ou Redes de Prot sejam interconectados de forma a gerar a modelagem final do sistema. Essa interconecção pode ser feita hierarquicamente através do conceito de sub-objetos (objetos parte de um outro objeto hierarquicamente superior). O Protob é capaz de representar graficamente essas hierarquias, utilizando-se do conceito de Visões (Views), isto é, quando um gráfico é muito complexo (não cabe inteiro no vídeo), pode-se decompô-lo em várias visões de modo que todo o sistema possa ser representado visualmente.

Além de sua natureza gráfica, o Protob pode ser caracterizado como um ambiente formal pois realiza tarefas como: análise sintática durante a construção das Redes de Prot, tradução do modelo em programa executável, geração de relatórios de documentação e execução visual do sistema modelado.

3.4 Avaliação das Ferramentas

Hatley, em [Hat 88], mostra um processo de avaliação de ferramentas CASE para o método RTRS (Real Time Requirements Specification) proposto por ele próprio em [Hat 84]. Segundo Hatley, muitas das ferramentas disponíveis de suporte ao RTRS não o fazem de modo completo. Em virtude desse fato e para evitar que o usuário dessas ferramentas façam uma escolha inadequada às suas necessidades, Hatley sugere uma série de fatores que ele considera desejáveis em ferramentas e que devem ser considerados durante o processo de escolha.

Considerando-se os fatores propostos por Hatley e a importância dos aspectos visual e formal presentes nas oito ferramentas apresentadas na seção anterior, é mostrada uma tabela (dividida em duas partes) de avaliação dessas ferramentas, segundo os seguintes fatores:

- Paradigma, Método e Linguagem: segundo Hatley, antes de selecionar uma ferramenta, o usuário deve estabelecer claramente o processo de desenvolvimento que deseja seguir, então decidir qual método suporta o processo escolhido e finalmente, selecionar as ferramentas que suportam o método. Portanto, esses fatores visam estabelecer quais paradigmas, métodos ou linguagens, são suportados pelas ferramentas descritas.

- Forma Principal de Entrada de Dados: indica de que forma é introduzida a maior parte das informações manipuladas pelas ferramentas.

- Interface com o usuário: mostra como é feita a comunicação usuário-ferramenta.

- Consistência das Informações: mostra o grau de confiabilidade e precisão das informações armazenadas pelas ferramentas. Descreve o aspecto formal das ferramentas.

- Características Gráficas Adicionais: Particularidades gráficas existentes em cada ferramenta como, por exemplo, técnicas utilizadas na edição e simulação dos diagramas.

Fatores	ProMod	Excelerator	IEW	TeamWork/ES
- Paradigma - Método - Linguagem	- Ciclo de Vida - Análise Estruturada - Projeto Estruturado	- Ciclo de Vida - Análise Estruturada	- Ciclo de Vida - Análise Estruturada	- Extensão da Análise Estruturada (Ward)
Principal Forma de Entrada de Dados	- Editor Gráfico para DFDs, Diagramas de Estruturas - Editor Texto para Minispecs e Dicionário de Dados	- Editor Gráfico para DFD, Modelo de Dados e Diagramas de Estruturas	- Editor Gráfico para Ações, DFDs, Entidades e Decomposição	- Editor Gráfico para Diagramas Contexto, DFDs, DTEs e Controle Dados - Editor de Formulários
Interface com Usuário	- Única para todas fases do ProMod - Interativa e independente da fase	- Mouse - Menus de Funções - Ícones	- Mouse - Menus de Funções	- Mouse - Menus de Funções
Consistência das Informações	- Análise Sintática durante Edição - Geração de documentos e Pseudo-código	- Integridade, completeza de dados por Dicionário de Dados - Relatório de Avaliação	- Análise Sintática - Técnicas de I.A. para consistir informações	- Detecção de Deadlock - Testes de Alcançabilidade
Características Gráficas Adicionais	- Navegação entre níveis hierárquicos nos DFDs	- Técnicas de zoom para visualizar níveis hierárquicos diferentes	- Visualização simultânea de várias janelas - Geração automática de interfaces - Algoritmos para reposição de fluxos	- Execução e simulação interativas - Simulação por painel e colocação de marcas

Fatores	TAGS	Protob	Argonauta	Statemate
- Paradigma - Método - Linguagem	- Paradigma da Automação - Linguagem IORL	- Ciclo de Vida com Especificação Operacional - Programação Orientada Objeto - Prot Nets	- Higraphs - Álgebra de Processos - Linguagem Argos	- Statecharts
Principal Forma de Entrada de Dados	- Editor Gráfico de Diagramas - SBD, IORTDs e PPDs	- Editor Gráfico de Redes de Prot - Descrição textual por Scripts	- Editor de Descrição de Diagramas - Editor de Fórmulas	- Editor Gráfico de Diagramas de Módulos, Atividades e Statechart - Editor de Formulários
Interface com Usuário	- Menus de Funções - Conjunto de símbolos da IORL (ícones) - Mouse	- Não é fornecido na literatura	- Mouse - Menus de Funções - Ícones	- Mouse - Menus de Funções - Ícones
Consistência das Informações	- Análise Sintática - Referência cruzada - Consistência fluxo de controle	- Análise Sintática durante edição - Geração de código executável - Simulação - Relatório de documentação	- Linguagem Argos com semântica formal - Simulação	- Análise Sintática - Simulação - Geração de código em Ada
Características Gráficas Adicionais	- Ícones + Mouse = Facilidade de utilização	- Simulação por mailboxes - Visualização de diversos níveis hierárquicos	- Gerenciamento de janelas com Zoom - Movimentação e ocultamento de processos	- Utilização de cores e fluxos diferentes nos diagramas. - Simulação por "acendimento" de estados

Confrontando-se as ferramentas (os quatro últimos fatores, principalmente), pode-se concluir que:

- Fator 2 (Entrada de Dados): por possuírem forte apelo visual, as ferramentas utilizam, como principal forma de entrada de dados, editores gráficos baseados nas técnicas ou linguagens que suportam.

- Fator 3 (Interface com Usuário): há um consenso na utilização de menus de funções, ícones, janelas e interação com mouse e/ou teclado.

- Fator 4 (Consistência das Informações): para se garantir a integridade das informações armazenadas, normalmente são feitos análise sintática, testes, checagem da consistência das informações. Isso possibilita a execução ou simulação visual dos diagramas construídos e, em alguns casos, a geração automática de código fonte em uma determinada linguagem.

- Fator 5 (Características Gráficas Adicionais): existem mecanismos para remover, deslocar e redimensionar os elementos gráficos suportados e, na existência de estruturas hierárquicas, a possibilidade de visualização de diversos níveis.

3.5 Considerações finais

Neste capítulo foram apresentadas seis técnicas (Máquinas de Estados Finitos, Redes de Petri, F-nets e R-nets, Esquemas de Transformação, Statecharts e Tabelas e Árvores de Decisão) e oito ferramentas (ProMod, TAGS, Excelerator, IEW, Teamwork/ES, Argonauta e Protob) destinadas à especificação de Sistemas Reativos, destacando-se o aspecto formal e visual que apresentam.

A seguir, fez-se uma avaliação das ferramentas apresentadas, na qual pôde-se verificar que elas procuram aliar o embasamento teórico com características gráficas, muitas vezes poderosas, que tornam o seu uso mais agradável.

Dessa forma, justifica-se a conveniência de se desenvolver ferramentas semelhantes para apoiar o uso de statecharts.

Capítulo 4

A Ferramenta Desenvolvida: um Editor Gráfico para Statecharts (EGS)

4.1 Considerações Iniciais

Como foi visto no capítulo dois desta dissertação, os statecharts são uma técnica para especificação e projeto de sistemas reativos baseada no que Harel define como formalismo visual, isto é, existe uma linguagem que define formalmente os statecharts e que é representável através de símbolos gráficos, constituindo um tipo especial de diagramas de estados.

Neste capítulo são apresentados alguns aspectos relacionados à análise e ao projeto da ferramenta desenvolvida, procurando mostrar como, e em até que nível, esse formalismo visual foi incorporado ao EGS.

São apresentados na seção 4.2 os requisitos funcionais do sistema, com base na definição formal dos statecharts. A seguir, na seção 4.3, são apresentadas as arquiteturas do EGS e da ferramenta desenvolvida por Fortes [For 91], mostrando a relação existente entre ambas. Finalmente, são relacionadas algumas restrições e decisões que foram tomadas para possibilitar a implementação dessa primeira versão do EGS.

4.2 Requisitos do EGS

Como foi mostrado no capítulo dois, os statecharts são um tipo particular dos Higraphs propostos por Harel, em [Har 88]. Portanto, pode-se dizer que os elementos que compõem um higraph estão, de alguma maneira, presentes nos Statecharts.

Harel, em [Har 87b, Har 88], define a semântica formal dos statecharts e dos higraphs, respectivamente. Baseado nesses artigos, serão relacionadas a seguir as principais características dessas técnicas e, a partir delas, serão apresentados os requisitos que serviram de base para a implementação da ferramenta.

Um Higraph é uma quadrupla (B, A, π, σ) onde :

- 1 - B é um conjunto finito de elementos chamados **Bolhas**.
- 2 - A é um conjunto de arestas que correspondem a **hiperarcos direcionados**.
- 3 - Função σ das **sub-bolhas**: associa a uma bolha de B, um conjunto de sub-bolhas. Dessa função, pode-se derivar os seguintes requisitos:
  - 3.1 - Uma bolha pertencente a B pode ser decomposta hierarquicamente em sub-bolhas.
  - 3.2 - Bolha atômica é aquela para a qual o conjunto das sub-bolhas é vazio.
  - 3.3 - A ausência de ciclos no relacionamento hierárquico entre um conjunto de bolhas permite derivar uma estrutura de árvore para representação desse relacionamento.
- 4 - Função π de partição: define a decomposição de uma bolha pertencente a um conjunto de bolhas B em seus **componentes ortogonais**. Como foi visto na seção 2.7.2, a ortogonalidade é a forma de representação de concorrência nos Statecharts. Desse elemento, pode-se derivar os seguintes requisitos:
  - 4.1 - As bolhas existentes em componentes ortogonais distintos são disjuntas, ou seja, não se intersectam.
  - 4.2 - Pode haver intersecção entre bolhas que pertençam a um mesmo componente ortogonal.

Os requisitos apresentados abaixo foram levantados por Fortes em [For 91] e estendidos nesta dissertação para mostrar o inter-relacionamento entre os statecharts e os higraphs.

- 5 - **Estados:** os estados estão para os statecharts, assim como as bolhas estão para os higraphs. Pode-se definir os seguintes requisitos com relação a eles:
- 5.1 - A função **Hierarquia** que define a decomposição dos estados em subestados. Essa função corresponde à função σ de sub-bolhas definida nos higraphs.
  - 5.2 - A existência de um único estado **raiz** do Statechart, que não é subestado de nenhum outro estado.
  - 5.3 - A função **Tipo** que define para cada estado o seu tipo: **AND** ou **OR**. Essa função determina que, se um estado é do tipo **OR**, a decomposição dos seus subestados significa que quando o sistema estiver nesse estado, ele estará em somente um dos subestado desse estado. Portanto, o tipo **OR** possui a lógica de ou-exclusivo (xor) entre os subestados. Para o estado do tipo **AND** o sistema estará em todos os subestados (componentes ortogonais) desse estado, em paralelo. Essa função corresponde à função π dos higraphs.
É importante notar que nos higraphs, existem situações em que as bolhas podem se intersectar. Nos statecharts isso nunca acontece.
  - 5.4 - Os símbolos de **História**, que definem a memorização de uma determinada configuração de estados do sistema de forma a possibilitar sua restauração.
    - 5.4.1 - Para que somente um nível de restauração seja memorizado o símbolo H é utilizado.
    - 5.4.2 - Quando a restauração for requerida em todos os níveis, a partir de onde se encontra o símbolo de *história*, o símbolo utilizado é H^* .
    - 5.4.3 - Num diagrama de statechart, a *história* (tanto H como H^*) é considerada como sendo um peso de um arco, isto é, sempre que se desenha um arco pode-se associar a ele um símbolo de *história*.
  - 5.5 - A função **default** que define um conjunto de estados e símbolos de *história* de forma a configurar um estado do sistema a partir do estado raiz. Essa função possui as seguintes possibilidades de utilização:
    - 5.5.1 - **def** : indica que o estado é um estado default do sistema e sua representação gráfica é uma pequena seta, apontando para o estado. Esse símbolo deve estar associado a um único estado, no caso de subestados de um estado do tipo **OR**. Num diagrama de statecharts, **def** é um arco que não tem estado origem.
    - 5.5.2 - **def/H** : indica que o estado é um estado default do sistema, respeitando-se a *história* do sistema naquele estado. Sua representação gráfica é uma pequena seta apontando para o estado,

juntamente com o símbolo *H*. Sua utilização significa que, se o sistema já possuir uma memorização de algum estado de mesmo nível de *hierarquia*, esse estado prevalece sobre o estado onde se encontra o símbolo de default. O símbolo *H*, nesse caso, também é considerado um peso do arco default.

5.5.3 - def/H* : indica que o estado é um estado default do sistema, respeitando-se a *história* do sistema a partir daquele estado, em todos os níveis de Hierarquia de subestados. Sua representação gráfica é uma pequena seta apontando para o estado, juntamente com o símbolo *H**. Sua utilização significa que, caso o sistema já possua uma memorização de algum estado de mesmo nível de Hierarquia, esse estado prevalece sobre o estado onde se encontra o símbolo de default, bem como toda a *história* dos subestados a partir dele. O símbolo *H**, assim como o símbolo *H*, é um peso do arco default.

6 - Expressões: o conjunto de *Expressões* é definido como a combinação de operações algébricas sobre números e variáveis, além de um operador especial: *current*.

6.1 - O operador *current* aplica-se a variáveis, retornando o valor atual da variável e é denotado por *cr(var)*.

6.2 - Os operadores algébricos são requeridos na formação de expressões.

7 - Condições Primitivas: o conjunto de *Condições Primitivas* define, basicamente, os valores *true* e *false* para serem avaliados durante uma transição no sistema. Essas condições fazem com que os estímulos percebidos pelo sistema sejam neutralizados em função de uma avaliação das condições especificadas na transição.

As *condições* são combinadas entre si de forma a compor novas condições com os seguintes operadores:

7.1 - in(estado_x) : retorna *true* quando o sistema está no estado_x.

7.2 - not_yet(evento_x) : retorna *true* quando o sistema avalia que o evento_x ainda não foi disparado.

7.3 - current(condição_x) : retorna o valor atual da condição_x.

7.4 - os operadores relacionais aplicados a expressões:

= (igual)

<> (diferente)

> (maior)

< (menor)
<= (menor ou igual)
>= (maior ou igual)

7.5 - os operadores lógicos aplicados a condições: **and**, **or** e **not**.

8 - **Eventos Primitivos**: o conjunto de *Eventos Primitivos* define os estímulos que são percebidos pelo sistema e que propiciarão o disparo das transições. Como *Eventos Primitivos*, entende-se intuitivamente qualquer ocorrência, independente de avaliação, que pode alterar o estado do sistema.

Os eventos podem ser de dois tipos :

8.1 - **eventos externos**: são acionados sob alguma intervenção externa à Simulação. São tidos como simples identificadores na especificação.

8.2 - **eventos internos**: são disparados pela própria lógica de sua especificação. Ou seja, a especificação do evento deve ser avaliada para que possa intervir na configuração dos estados do sistema.

Os eventos primitivos podem estar combinados entre si e com os eventos internos relacionados abaixo:

8.3 - **true(condição_x)** : determina um estímulo no instante em que a condição_x se tornou verdadeira.

8.4 - **false(condição_x)** : determina um estímulo no instante em que a condição_x se tornou falsa.

8.5 - **changed(variavel)** : determina um estímulo no instante em que a variável mudou de valor.

8.6 - **exit(estado_x)** : determina um estímulo no instante em que o sistema deixou de estar no estado_x.

8.7 - **entered(estado_x)** : determina um estímulo no instante em que o sistema passou a estar no estado_x.

8.8 - Operadores **or**, **and** e **not**: utilizados na composição de eventos.

8.9 - **evento_x [condicao_y]** : determina um estímulo no instante em que ocorreu o evento_x, desde que a condição_y seja verdadeira nesse mesmo instante.

Finalmente, um evento especial bastante utilizado nos exemplos de Harel pode ser enunciado:

8.10 - **time_out (n unid_tempo)**: determina um estímulo no instante em que “n” unidades de tempo (unid_tempo) transcorreram desde o momento em que o sistema passou a estar no estado-origem da transição que possui esse evento.

- 9 - **Ações:** o conjunto de *Ações* é definido como o das reações instantâneas simultâneas às transições em que elas são especificadas. As ações aparecem especificadas após o símbolo “/” e são disparadas pelos eventos que as antecedem.

Uma ação requer o operador de atribuição “:=” nas seguintes formas:

9.1 - **condição_primitiva := condição_x :** onde a ação consiste da atribuição do valor da avaliação da condição_x à condição_primitiva.

9.2 - **variável := expressão_x :** onde a ação consiste da atribuição do valor da avaliação da expressão_x à variável.

Dessa forma, pode-se observar que uma ação pode vir a ser um evento para o sistema, uma vez que seu acionamento é instantâneo e simultâneo à transição. Esse efeito é chamado de propagação de eventos (Broadcasting). Para esse efeito deve ser prevista uma distinção entre os eventos ocorridos externamente dos que foram criados por propagação. Harel define as unidades de passo, que são os intervalos entre eventos ocorridos, e os micro-passos, que reconfiguram os estados do sistema devido a eventos gerados por ações das transições e que são intervalos contidos nas unidades de passo.

Uma ação especial pode ser enunciada devido a sua utilização nos exemplos apresentados por Harel:

9.3 - **clear(estado[*]) :** equivale a *clear history* e consiste em eliminar a história para a configuração de um estado do sistema (apenas ele ou ele e todos os seus subestados), fazendo com que nas próximas transições chegando a esse estado, mesmo as que possuem o símbolo de história, o sistema vá para os estados defaults.

Os itens 6,7,8 e 9 compõem um outro peso de um arco: o nome. Um nome, em um diagrama de statechart, é uma simples cadeia de caracteres na qual podem estar presentes desde um evento primitivo até uma composição dos itens citados acima. A identificação de cada item da cadeia é feita por um analisador que será discutido ainda neste capítulo.

- 10 - **Transições:** dado um conjunto de estados atuais do sistema, se ocorrer um estímulo (evento) no sistema, ele então passará para um outro conjunto de estados, configurando a resposta ao estímulo recebido.

As *Transições*, que graficamente constituem os arcos no Statecharts, possuem os rótulos que as acompanham da seguinte forma: **evento_x / ação_y**. Esse rótulo determina que a transição seja disparada no instante em que recebe o estímulo do evento_x e a ação_y seja executada.

As *Transições* podem ser:

- 10.1 - 1:1 (um para um) quando afetam estados do tipo OR da seguinte maneira: deixam de estar no estado-origem da transição e passam para o estado-destino da transição. Uma transição 1:1 corresponde a um arco binário, ou seja, aquele que tem apenas um ponto de origem e outro de destino. Esses pontos podem ser referentes a uma única bolha ou, então, a duas bolhas distintas.
- 10.2 - M:N (M para N) quando afetam estados do tipo AND. Para esse tipo de transição, os M estados-origem em que o sistema deixa de estar devem ser o de todos os estados ortogonais de onde parte a transição. Equivalentemente, os N estados-destino da transição para os quais o sistema configura, devem ser todos os estados ortogonais a ele. Uma transição M:N corresponde a um hiperarco, ou seja, aquele que tem M pontos origem e N pontos destinos ($M > 1$ e $N > 1$).
Um outro requisito importante no conjunto de *Transições* pode ser descrito:
- 10.3 - As *transições* são relacionamentos entre estados do tipo OU do sistema especificado. A partir desse fato pode-se ter o caso de transições paralelas, por estarem contidas em subestados do tipo AND. Esse é o caso, por exemplo, quando várias transições, contidas em subestados ortogonais diferentes, possuírem o mesmo rótulo. Portanto, elas devem ser efetivadas simultaneamente.

A tabela apresentada a seguir tem dois objetivos básicos:

a) Mostrar como os requisitos enumerados acima foram considerados e traduzidos para a implementação do Editor. Em vários casos, um grupo de requisitos pôde ser agrupado em um único aspecto no que diz respeito à implementação. Como exemplo, pode-se citar: uma expressão, uma condição e/ou um evento são para o EGS apenas uma cadeia de caracteres que será definida de forma textual pelo usuário durante a especificação gráfica de um diagrama de Statecharts. O Analisador desenvolvido por Fortes em [For 91] se encarrega (dada essa cadeia de caracteres como entrada) de identificar, separar e armazenar na estrutura de dados, as partes identificadas.

b) Mostrar algumas das características da interface do EGS com o usuário e justificar essa interface em função dos requisitos levantados anteriormente.

Item	Requisito	Aspectos da Implementação	Interface com Usuário
1 5 5.2	Bolhas Estados Raiz	Ao se desenhar uma bolha, dois aspectos são considerados: a bolha não pode intersectar nenhuma outra bolha irmã e não pode ultrapassar os limites da bolha pai. A maior bolha é delimitada pelo tamanho da janela de edição.	A bolha é desenhada pressionando-se o botão esquerdo e arrastando-se o mouse, simultaneamente. Utilização de efeito "rubber-band" para visualização da bolha em edição. Nome da bolha é especificado em um <i>panel</i> ¹ tipo texto. Operações: Criar, remover e consultar bolhas.
3 5.1	Função σ Hierarquia	A Hierarquia no EGS se dá de duas maneiras: desenhar uma bolha no interior de uma outra bolha já desenhada ou expandir uma bolha em novo diagrama.	O mesmo processo do item anterior. Caso não haja espaço para se desenhar uma nova bolha no interior de uma outra, cria-se nova janela de edição e reinicia-se o processo de edição. Esse processo é conhecido como refinamento ou expansão. Operações: Criar e expandir bolha.
4 5.3	Função π Função tipo	O usuário seleciona o tipo da bolha: AND ou OR. Se for AND, especifica o número de componentes ortogonais que deseja para a bolha.	Os componentes ortogonais são escolhidos por <i>panels</i> e delimitados por linhas tracejadas no interior da bolha a ser dividida. As linhas são traçadas a partir de pontos escolhidos pelo usuário junto às bordas da bolha a ser dividida. Operações: Criar bolha tipo AND.

1 - Objeto visual do sistema XVIEW. Maiores detalhes no capítulo 5, seção 5.2.

Item	Requisito	Aspectos da Implementação	Interface com Usuário
5.4.1 5.4.2 5.4.3	H H* H e H*	Símbolos considerados atributos de um arco: sempre que se desenhar um arco pode-se associar a ele estes símbolos, permitindo que eles sejam desenhados junto ao arco.	Seleção através de <i>panels</i> . Uma vez selecionados, são desenhados automaticamente e posicionados no local escolhido pelo usuário. O posicionamento é feito pressionando-se o botão esquerdo e arrastando-se o mouse até a posição desejada. Operações: Criar arcos.
5.5 5.5.1 5.5.2 5.5.3	Função default def def/H def/H*	Considerada atributo de uma bolha. Gráficamente, corresponde à tarefa de se traçar um arco. Um arco é concebido como uma sucessão de pontos determinados pelo usuário. Uma função Spline se encarrega de suavizar a curva. A um arco default, também pode-se associar símbolos de história.	Ponto inicial diferenciado é livre. O ponto final é identificado automaticamente pelo EGS quando reconhece as bordas ou interior de uma bolha. Entre os pontos intermediários são traçadas retas que vão compondo o traçado do arco, até serem substituídas por uma curva suavizada. Operações: Criar e remover arcos default.
6 7 8 9	Expressões Condições Eventos Ações	Atributos de um arco. São combinados em uma única cadeia de caracteres. Essa cadeia é analisada sintaticamente por um Analisador. A identificação visual de um arco é armazenada em um rótulo.	Visualmente, um arco é um conjunto de curva, símbolo de História e um rótulo que é o identificador visual de um arco. Um arco pode ser: primitivo (apenas um evento primitivo) ou complexo (evento + expressão + ação). Para cada arco existe um conjunto de <i>panels</i> textos que coletam essas informações. Operações: Criar e remover arcos 1:1 e M:N.

Item	Requisito	Aspectos da Implementação	Interface com Usuário
2 10.1	Hiperarcos Transições 1:1	Operação de traçar um arco binário, ou seja, aquele com um ponto inicial e outro final. Os pontos origem e destino podem ser relativos a uma única bolha.	O EGS reconhece automaticamente os pontos inicial e final durante a operação. Entre os pontos intermediários são traçadas retas que vão compondo o traçado do arco, até serem substituídas por uma curva suavizada. Operações: Criar e remover arco 1:1.
2 10.2 10.3	Hiperarcos Transições M:N transições paralelas	Operação de se traçar um hiperarco (arco M:N), ou seja, desenhar M+N sub-arcos com origem em M bolhas e destino em N bolhas. Cada sub-arco é um conjunto de pontos definidos a critério do usuário. Uma transição paralela, associa um conjunto de bolhas fontes a um conjunto de bolhas destinos, portanto é um arco M:N.	Processo similar ao anterior. Nesse caso, além de desenhar os sub-arcos, deve-se definir uma área de intersecção que é o ponto de ligação entre os sub-arcos M e N. Ao traçar os sub-arcos M os pontos origens são as bolhas origens da transição e os pontos destinos, a área de intersecção. Para os arcos N, os pontos de origem estão na área de intersecção e os pontos destinos são as bolhas destinos da transição. Operações: Criar e remover arcos M:N.

4.3 A Arquitetura do EGS

No capítulo 3 os Statecharts foram apresentados como uma linguagem de especificação que combina formalismo com um forte apelo visual na representação da especificação.

O aspecto formal é apresentado na tese de mestrado de Fortes [For 91] e dela resultou uma proposta e posterior implementação de uma ferramenta composta de um Analisador e um Simulador para Statecharts, cuja arquitetura é mostrada na figura 4.1. O Analisador recebe como entrada a especificação de um sistema escrita em LES (Linguagem de Especificação de Statechart - Apêndice C), faz a análise (léxica, sintática e de contexto) e gera uma Base de Dados da Especificação. Nesse ponto, a simulação pode ser feita: o Simulador, aceitando

um conjunto de eventos como entrada, consulta e modifica a Base de Dados, indicando de forma textual as alterações no comportamento do sistema em função da ocorrência de eventos.

O EGS é uma ferramenta que tem por objetivo possibilitar ao usuário, a utilização de recursos gráficos para a geração de diagramas de Statecharts e, em função do diagrama gerado, criar a mesma Base de Dados que é criada pela ferramenta de Fortes, acrescida, naturalmente, das informações gráficas.

A figura 4.2, corresponde à arquitetura do EGS. É importante notar que o formalismo está presente no EGS, pois durante a edição gráfica utiliza-se o analisador (apresentado com maiores detalhes ainda nesta seção) para manter consistente as informações que vão sendo armazenadas na Base.

O Analisador

A sintaxe dos Statecharts é definida em função de cinco elementos primitivos: estados, transições, eventos primitivos, condições e variáveis. A partir desses elementos, definem-se conjuntos estendidos de: eventos, condições, expressões e rótulos. Essa sintaxe é formalmente definida por Harel em [Har 87b] e em [For 91] encontra-se a descrição do analisador que é utilizado pelo EGS. A seguir será dado um exemplo da utilização do analisador dentro do EGS.

Como foi mostrado na tabela da seção 4.2, o conjunto formado por expressões, condições, eventos e ações é considerado um atributo ou peso de um arco. Dessa forma, sempre que se for traçar um arco pode-se associar a ele eventos, condições, etc. Nesse ponto, o analisador é solicitado para verificar se esse atributo está correto sintática e semanticamente. Caso não esteja, retorna uma indicação de erro ao EGS, que recusa o atributo indicando a causa do erro.

Esse tipo de procedimento garante a integridade de parte das informações armazenadas na Base de Dados de Statecharts. Garantir a concordância sintática e semântica das informações durante a edição é o primeiro passo no sentido de assegurar o sucesso nas operações realizadas sobre o diagrama, como por exemplo a tarefa de simulação.

A Base de Dados Statechart

Para recuperar um statechart da memória secundária, o EGS consulta a Base de Dados, carrega as informações na memória e “desenha” o diagrama no vídeo tal como fora construído durante a tarefa de edição. A partir de então, pode-se re-editar o diagrama e/ou simulá-lo. Pode-se dizer que essa Base de Dados é composta por dois grupos de informações distintas: sintáticas e gráficas.

Figura 4.1: Arquitetura do Analisador e Simulador de Fortes

Figura 4.2: Arquitetura do EGS

Informações Sintáticas: são as informações que estabelecem a relação lógica entre os elementos de um statechart. Por exemplo:

- Estado A ligado ao estado B, via transição t.
- Estado A, B e C são concorrentes.
- A transição t acontece se for satisfeita a condição c.
- Estado A é pai do estado B, etc.

Informações Gráficas: são as informações que definem o formato visual dos elementos de um statechart. Por exemplo:

- Tamanho e posição de desenho de uma bolha.
- Curvatura, ponto de origem e destino de um arco.
- Posição de escrita de um rótulo de uma transição.
- Tamanho e posição de desenho de uma bolha (caso tenha sido expandida), etc.

Na verdade, o termo Base de Dados não é usado nesta dissertação, no sentido de que existe um Sistema Gerenciador de Banco de Dados para manipular a Base de Dados de Statechart. O processo de edição, bem como o de simulação, é feito manipulando-se apenas estruturas residentes em memória principal. A Base de Dados criada para o EGS, resume-se a um conjunto de arquivos gerados pela própria linguagem C. Esses arquivos são lidos no início e atualizados durante e/ou após uma seção de edição de um diagrama de Statechart. As operações de ler e gravar em disco são realizadas por um conjunto de rotinas escritas em C.

Uma representação da estrutura de dados do EGS

Nas figuras 4.3 a 4.7 são representados esquemas da estrutura de dados das bolhas (estados) e dos arcos (transições) existentes na Base. O objetivo dessa representação é mostrar como essas estruturas foram projetadas a fim de permitir a simulação (manipulação da estrutura sintática) integrada à edição gráfica dos diagramas (manipulação das estruturas gráficas). A figura 4.3 corresponde à estrutura do statechart da figura 2.13 do capítulo 2.

No apêndice A encontram-se detalhadas as estruturas de dados das informações gráficas utilizadas para a implementação do modelo descrito nas figuras 4.3 a 4.7. Maiores detalhes sobre as informações sintáticas, podem ser obtidas em [For 91].

A Estrutura Funcional do EGS

Segundo Pressman, em [Pre 87], a arquitetura de um software está relacionada com a representação de duas importantes características de um programa:

Figura 4.3: Estrutura de Árvore do Statechart

a estrutura de dados (detalhes no Apêndice A) e a estrutura hierárquica dos seus componentes procedimentais ou módulos. O diagrama estruturado de Blocos da figura 4.8 representa a organização hierárquica dos módulos que compõem o EGS. Os módulos Bolhas, Arcos, Variáveis, Arquivos e Diretório são módulos de controle e os demais, abaixo na hierarquia, correspondem às principais tarefas realizadas pelo EGS. A nível de interface, os cinco módulos de controle correspondem a cinco menus distintos na tela principal de edição e cada item desses menus representa um módulo funcional hierarquicamente inferior aos módulos de controle.

O Funcionamento de cada módulo, com ênfase para a interface gráfica com o usuário, será apresentado no capítulo seguinte.

4.4 Decisões de Projeto

Para concretizar a implementação da ferramenta, dois aspectos foram previamente definidos: um padrão de interface com o usuário e um conjunto, embora pequeno, de restrições à técnica apoiada pela ferramenta.

Figura 4.4: Modelo da Estrutura de uma Bolha (Estado)

Figura 4.5: Modelo da Estrutura GRAF (Inf. Gráficas de uma Bolha)

Figura 4.6: Estrutura de uma Transição

Figura 4.7: Modelo da estrutura de um ARCO

Figura 4.8: Decomposição Hierarquica dos módulos do EGS

4.4.1 A Interface Gráfica do EGS

Para a implementação do EGS, adotou-se a seguinte linha de conduta: explorar os recursos de hardware e software de uma estação gráfica, de forma a se construir uma interface amigável que tornasse a edição dos diagramas de Statechart uma tarefa simples e agradável para o usuário e que, além disso, reproduzisse com fidelidade, no terminal de vídeo, o aspecto visual da técnica proposta por Harel.

Para auxiliar o processo de implementação da interface foi utilizado o padrão de programação XVIEW [Hel 90]. As razões dessa escolha são explicadas a seguir.

O Sistema X Window

O X Window é um sistema padrão adotado por vários fabricantes de hardware e software, que dá ao programador recursos de software para o desenvolvimento de interfaces com o usuário baseadas em janelas e utilização de mouse [Sch 86].

O que torna o X Window um sistema padrão para o gerenciamento de janelas é o fato de ser baseado em protocolo de rede - um conjunto pré-definido de pedidos e respostas - ao invés de chamadas de procedimentos, como ocorre com outros gerenciadores de janelas. A função desse protocolo é permitir a comunicação entre dois processos: o Cliente e o Servidor X. Um cliente é qualquer programa de aplicação que opera em um terminal gráfico através do servidor X (no caso deste trabalho, o cliente seria o sistema EGS). O Servidor X controla os recursos do usuário (terminal de vídeo, mouse, teclado) e torna esses recursos disponíveis à aplicação (Cliente). Em outras palavras, o servidor isola estes recursos do Cliente, conferindo ao Sistema X Window uma grande independência com relação ao hardware e tornando a aplicação bastante portátil.

Ao contrário de outros sistemas de gerenciamento de janelas, o X Window não estabelece regras fixas para o controle dos eventos gerados por uma aplicação (tecla e/ou mouse pressionado, movimentação e sobreposição de janelas, etc). Existe um cliente especial chamado Gerente de Janelas que controla, segundo diretrizes estabelecidas pela aplicação, o comportamento do sistema com relação à ocorrência de eventos.

Outro componente importante do X Window é a biblioteca de funções XLib. Ao se compilar e adicionar um programa cliente à XLib, pode-se garantir que ele funcionará (sem necessidade de mudanças) com qualquer Servidor X, de forma transparente ao longo da rede.

Figura 4.9: Inter-relacionamento dos Sistemas X Window, XVIEW e EGS

O Modelo de Programação XVIEW

A grande desvantagem do X Window é o baixo nível de programação ao qual o programador tem que se submeter durante a implementação do software. A Sun Microsystems desenvolveu o XVIEW para simplificar a programação de software utilizando o X Window. O XVIEW consiste de um conjunto pré-definido de macros que permite ao projetista desenvolver um software utilizando o X Window, com a vantagem de ter um código fonte mais simples e escrito em mais alto nível. Quando necessário, é possível incluir, dentro do XVIEW, chamadas a funções do X Window. A Programação em XVIEW é baseada nos princípios da Programação Orientada a Objetos. Os objetos:

- São representados em uma hierarquia de classes.
- São tipos opacos de dados.
- Têm atributos que podem ser manipulados através de funções.

Por exemplo: um *frame* é uma subclasse de uma classe mais geral chamada Window. Ao criar um *frame*, o programador o faz sem ter noção clara da estrutura que está manipulando e, para modificar, por exemplo o tamanho desse *frame*, é chamada uma função que altera os valores dos atributos “altura” e “comprimento” do objeto *frame*.

A figura 4.9 mostra o inter-relacionamento entre os Sistemas X Window, XVIEW e a ferramenta EGS.

4.4.2 Restrições

As restrições apresentadas a seguir correspondem a certas limitações do EGS na edição gráfica dos Statecharts, mas que no entanto, não afetam e nem violam as características da técnica original proposta por Harel.

Representação de Arcos *Default*

Na seção 2.6 (figura 2.6), foram mostradas duas formas de representação visual para os arcos defaults: a primeira onde com apenas um arco, definiam-se duas ou mais bolhas defaults; e uma segunda, onde cada bolha default possuía seu próprio arco default. A primeira representação, apesar de mais econômica visualmente, implicaria num algoritmo muito complexo, cuja interpretação para definir o conjunto de bolhas defaults, dependeria do tamanho e posição do arco traçado. Assim, optou-se por implementar a segunda representação que tem o mesmo significado da primeira.

Refinamento de uma Bolha

O EGS permite que uma bolha seja refinada de duas formas:

a) Desenhando-se, numa mesma janela de edição, uma bolha no interior de uma outra bolha. Nesse caso, diz-se que essas bolhas têm o mesmo nível visual, pois são todas visíveis simultaneamente.

b) Criando-se uma nova janela de edição caso a bolha que se deseja refinar não possua um espaço físico interno suficientemente grande para se desenhar novas bolha no seu interior. Nesse caso, diz-se que as bolhas desenhadas nessa nova janela possuem nível de visualização maior que o da bolha refinada.

Neste caso, o EGS não permite que arcos sejam traçados unindo bolhas de diferentes níveis visuais, ou seja, arcos que tenham origem numa janela de edição e destino em outra janela e vice-versa. Tome-se, como exemplo, as figuras 2.14 e 2.15 do capítulo 2: o arco “bat.fraca” da figura 2.14 sugere que o ponto inicial está relacionado a alguma bolha filha da bolha “bateria”, ou seja, a bolha bateria foi refinada e possui bolhas filhas que não são visíveis naquele nível.

O que acontece, a nível de interface, é que o EGS cria uma nova janela de edição para mostrar a bolha “bateria” refinada, e nela são desenhadas novas

bolhas. Nesta primeira versão do EGS, não é permitido traçar arcos deste tipo, ou seja, arcos que interligam bolhas pertencentes a janelas de edição diferentes (bolhas com níveis visuais diferentes).

Abstração de Bolhas

Ao traçar uma bolha, o EGS considera dois aspectos com relação a seus limites físicos:

- a) A bolha não deve intersectar as suas bolhas irmãs (caso existam).
- b) A bolha não deve exceder os limites das bordas de sua bolha pai.

Conclui-se que a construção de um diagrama pelo EGS é feita de forma descendente (top-down), não permitindo o encapsulamento de um conjunto de bolhas em uma nova bolha de nível hierárquico maior (construção ascendente ou bottom-up). Portanto, o usuário deve definir com antecedência os possíveis casos de abstração que possam existir no diagrama e então, iniciar o processo de edição.

A Movimentação de Bolhas dentro do Diagrama

Uma bolha pode ser o ponto de origem e/ou destino de vários arcos. Graficamente, mover uma bolha dentro de um diagrama de statechart, poderia implicar num processo de reposicionamento desses arcos, o que exigiria um algoritmo complexo se for considerado o problema da sobreposição entre os diversos elementos existentes no diagrama.

Porém, a falta desse procedimento não chega a ser uma limitação considerável, devido a grande facilidade existente em se criar e remover bolhas e arcos. A utilização desses dois processos, alternadamente, substitui a operação de mover bolhas.

4.5 Aspectos relacionados ao desenvolvimento do EGS

O Editor foi desenvolvido em linguagem C, utilizando o modelo de programação XVIEW, baseado nos princípios da programação orientada a objetos e no Sistema Gerenciador de Janelas X Window.

O equipamento utilizado foi uma Estação Gráfica Sun Spark 1. O programa fonte gerado consta atualmente de 7000 linhas e 300 Kbytes de código

executável.

A implementação foi baseada no conceito de Tipos Abstratos de Dados (TAD): um arco, por exemplo, pode ser visto como um vetor de coordenadas (x,y) associado a um rótulo (uma cadeia de caracteres que identifica o arco) e uma coordenada que indica a posição de escrita desse rótulo. Assim foi implementado um TAD chamado ARCO e um conjunto de rotinas que operam sobre ele. Sempre que se quer desenhar ou remover arcos, recorre-se a essas rotinas.

Com relação ao aspecto metodológico de desenvolvimento, foram seguidos os seguintes passos:

- Análise e especificação dos requisitos do EGS: baseado na definição proposta por Harel, definiu-se um conjunto de requisitos funcionais (veja seção 4.2) que serviu de base para a implementação.

- Projeto Estruturado do EGS: decomposição funcional do EGS em módulos hierárquicos. Definição da estrutura de dados do sistema (projetos das estruturas gráficas).

- Implementação dos módulos: codificação em linguagem C dos módulos definidos na etapa anterior.

- Testes: Dois tipos de testes foram realizados:

- Teste de Unidade: Foram testadas a integridade das estruturas gráficas criadas durante a edição e verificada a robustez da interface gráfica do EGS. A técnica funcional foi aplicada e consistiu da validação do conteúdo e da estrutura de dados gerados pelo EGS para cada elemento gráfico primitivo criado durante a edição. Além disso, testou-se o aspecto visual de cada objeto gráfico, comparando-se o resultado obtido no vídeo com os requisitos estabelecidos na especificação do projeto. Nesse sentido, foram editados vários diagramas executando-se todas as operações apresentadas no Diagrama Hierárquico de Blocos. Muitas vezes, a edição era feita por pessoas que não participaram diretamente do projeto.
- Teste de Integração: consistiu da validação da interface entre o EGS e a ferramenta projetada por Fortes. Como foi mostrado na seção 4.3, a Base de Dados de Statechart é única. Quer seja gerada pelo Analisador, quer seja pelo EGS. Foram editados, através do EGS, vários diagramas de statecharts e em seguida, simularam-se os diagramas obtidos utilizando a ferramenta de Fortes.

4.6 Considerações Finais

Neste capítulo foram apresentados alguns aspectos da ferramenta desenvolvida. Baseando-se na definição formal de statecharts proposta por Harel, definiu-se um conjunto de requisitos que foi utilizado como base para a implementação.

Foi apresentada a arquitetura do EGS, mostrando-se sua relação com a ferramenta desenvolvida por Fortes. Alguns detalhes do modelo da estrutura de dados e da decomposição funcional dos módulos do EGS também foram apresentados.

Finalizando, justificou-se a escolha do sistema XVIEW como modelo de programação e foram apresentadas as restrições de projeto do sistema implementado.

A conclusão a que se chegou, ao final da implementação do EGS, é a seguinte:

a) As restrições existentes no EGS não impedem que todas as características dos statecharts propostos por Harel sejam representadas graficamente.

b) Avaliando-se o EGS, conforme mostra a tabela a seguir, e comparando-se com a avaliação (segundo os mesmos critérios dessa tabela) das ferramentas apresentadas no capítulo três, pode-se dizer que o EGS segue os mesmos padrões e tendências das ferramentas destinadas à especificação e projeto de sistemas reativos e que têm como principais características a combinação do aspecto formal com uma forte capacidade de representação visual da técnica apoiada.

Ferramenta	Linguagem	Principal Forma de Entrada de Dados	Interface com Usuário	Consistência das Informações	Características Gráficas Adicionais
EGS	Statecharts	Manipulação direta de objetos na tela, possibilitando a edição dos elementos gráficos dos state-chart.	- Mouse - Menus de Funções - Sobreposição de janelas	- Análise Sintática e Semântica on line - Consistência no desenho de arcos e bolhas e respectivos atributos.	- Técnicas de zoom para visualização do refinamento de uma bolha - Visualização de vários níveis hierárquicos simultaneamente.

Capítulo 5

Exemplo de uso do EGS

5.1 Considerações Iniciais

Neste capítulo são mostradas as principais características da ferramenta desenvolvida. Na seção 5.2 é ilustrada a interface do EGS, descrevendo-se os seus principais elementos.

Na seção 5.3, mostra-se o EGS em operação e são descritas as principais funções desempenhadas pelo editor. As figuras mostradas foram geradas pelo próprio EGS.

Finalizando, discute-se a possibilidade da generalização do EGS para apoio a outras técnicas baseadas em grafos, além dos statecharts.

5.2 A Interface do EGS

Uma preocupação ao se implementar o EGS foi a construção de uma interface homogênea e de fácil utilização, que possibilitasse a construção de diagramas de forma interativa e que, ao mesmo tempo, utilizasse de forma racional os recursos de software e hardware de uma estação gráfica.

Nesse sentido, a implementação do editor em C, utilizando o sistema XVIEW, contribuiu para que esse objetivo fosse atingido ao possibilitar que fossem introduzidos na interface, de maneira natural, elementos como:

a) Manipulação Direta de Objetos na Tela: praticamente, toda a edição é feita com a utilização do *mouse* para manipulação direta de objetos, quer seja a seleção de operações a executar, quer seja a construção dos elementos gráficos que compõem o diagrama, como por exemplo: arcos, bolhas, etc.

b) Teclado: utilização restrita a declarações de informações textuais, tais como: rótulo de um arco, ações, eventos e variáveis.

c) Janelas: a edição de um diagrama de statechart pode implicar na utilização simultânea de várias janelas que podem necessitar ser sobrepostas, deslocadas ou removidas. Portanto, a manipulação de janelas constitui-se numa das principais características da interface do EGS. Entenda-se por janela o conjunto formado pelos vários elementos visuais da interface. Esses elementos serão apresentados no tópico Elementos Visuais da Interface do EGS, ainda neste capítulo.

d) Troca de mensagens com o usuário: sempre que alguma informação sintática ou semanticamente incorreta for inserida, o EGS as recusa, informando o provável motivo do erro. Todas as mensagens obedecem a um padrão pré-estabelecido pela interface.

O EGS, na atual versão, não inclui menus de ajuda do tipo *help*. Isso poderá ser feito facilmente se versões futuras mais profissionais, forem desenvolvidas. Foi possível notar que, mesmo para usuários ainda não familiarizados com a técnica statechart e com o editor em si, a edição dos diagramas se tornou, em pouco tempo, uma tarefa relativamente simples. Isso se deve ao processo de edição, que é intuitivo (desenhar bolhas e depois ligá-las com arcos) e também ao fato de que o EGS permite uma certa flexibilidade na ordem de edição dos elementos gráficos.

Os elementos visuais da interface do EGS

Como visto no capítulo anterior, o sistema XVIEW é baseado em objetos que podem ser visuais e não visuais. Pode-se dizer que as aplicações implementadas sob esse sistema tendem a se constituir, a nível de interface, em um conjunto de objetos visuais.

Para melhor compreender o exemplo que será apresentado na seção seguinte, serão relacionados abaixo os principais objetos da interface, mostrando o papel que desempenham no EGS.

a) *Canvas*: um dos objetos mais importantes do XVIEW. É a área de uma janela na qual são apresentados os gráficos e captados os eventos vindos do teclado e do *mouse*.

b) *Frame*: é o objeto que delimita a área de uma determinada janela. Ele controla a geometria e o posicionamento de outros objetos existentes em uma janela. Visualmente, corresponde à moldura de uma janela.

c) *Panels*: a nível de interface, um *panel*, pode ser visto como um botão através do qual se aciona alguma outra tarefa. No EGS os *panels* são utilizados para apresentar menus e seus respectivos itens (esses *panels* são conhecidos como botões de menu), habilitar a captação de eventos no *canvas*, armazenar informações textuais (*panel* texto) e disparar alguma tarefa do editor como, por exemplo, salvar um diagrama de statechart.

d) *Notice*: é um tipo especial de janela utilizado como padrão para reportar mensagens de erro.

e) *Menus*: é uma pequena lista na qual estão relacionadas as operações do sistema. A seleção das operações realizadas pelo EGS é feita com o auxílio do *mouse*. Na interface implementada, o acesso a um menu e seus respectivos itens pode ser feito de duas maneiras:

- Pressionando-se o botão direito do *mouse* em qualquer região do *canvas*. Esse menu é conhecido como menu *pop-up*.
- Pressionando-se o botão direito do *mouse* na região delimitada por um *panel* “botão de menu”. Esse menu é chamado de menu *push-down*.

A figura 5.1 representa uma árvore que corresponde à organização hierárquica das funções realizadas pelo EGS. Os nomes representados dentro de retângulos correspondem aos botões de menus existentes na tela principal de edição.

O menu principal do EGS

Ao ser carregado, o EGS coloca no vídeo uma janela de apresentação do sistema. Ao se pressionar o botão de menu (botão esquerdo do *mouse*) é mostrado um menu pop-up com as seguintes opções:

- Edição.
- Simulação.
- Sai do Sistema.

A tarefa de edição e as demais tarefas subsequentes a ela serão apresentadas na seção 5.3.

Figura 5.1: As opções de Menu do EGS

A tarefa de simulação, apesar de não pertencer ao escopo deste trabalho, foi incorporada ao menu principal do EGS por dois motivos:

a) Como foi mostrado no decorrer desta dissertação, os statecharts especificam o aspecto comportamental de um sistema reativo. Esse comportamento pode ser avaliado através de simulações sobre o diagrama construído.

b) Já existe no Instituto de Ciências Matemáticas de São Carlos (ICMSC) uma ferramenta desenvolvida por Fortes [For 91] que realiza a tarefa de simulação, embora de maneira não gráfica, de um diagrama de statechart. Como a Base de Dados é única em ambos os projetos, a integração entre as duas ferramentas é um passo natural.

5.3 Exemplo de um Diagrama Criado Usando o EGS

A fim de melhor explicar as principais funções da tarefa de edição, será mostrado nesta seção um exemplo da construção de um diagrama de statechart através do EGS. O exemplo utilizado é o mesmo que foi empregado no capítulo 2 e foi extraído de [Har 87]. Apesar de não ser um exemplo muito complexo, ele incorpora todas as características de um statechart apresentadas nesta dissertação. As figuras mostradas a seguir foram geradas e extraídas do próprio EGS. Na ausência de uma impressora com linguagem post-script, que pudesse reproduzir as figuras, optou-se por tirar algumas fotos.

A Tela Principal de Edição

Ao selecionar a opção de Edição no menu principal, o EGS cria a janela da figura 5.2, na qual existem cinco funções bem definidas e que são seleccionadas através de cinco *panels* (botões de menus) distintos:

- Bolhas.
- Arcos.
- Arquivos.
- Variáveis.
- Diretório.

Figura 5.2: A Janela Principal de Edição

Nas subseções seguintes serão mostrados alguns detalhes do funcionamento de cada uma das cinco opções, com especial atenção para as tarefas realizadas sobre as bolhas e os arcos. Considerações sobre o *panel* Diretório são feitas na subseção que descreve o *panel* Arquivos.

5.3.1 Manipulação de Bolhas

O menu associado ao *panel* Bolha é composto dos seguintes itens:

- Criar
- Apagar
- Expandir
- Informações

A seguir será mostrado o significado de cada tarefa e a maneira como o EGS executa cada uma delas. As tarefas realizadas nesse módulo do sistema incorporam os requisitos mostrados na primeira tabela da seção 4.2.

A criação de Bolhas

A figura 5.3 ilustra uma tarefa de criação de bolha. Uma janela de atributos da bolha é criada no canto superior direito da janela principal de edição sempre que uma bolha válida é traçada. Para o EGS, uma bolha válida é aquela que não intersecta as suas bolhas irmãs e não ultrapassa os limites das bordas de sua bolha pai. Isso é prevenido pelo próprio sistema.

A bolha é traçada com auxílio do *mouse*, utilizando-se a técnica de *rubber-band*, ou seja, à medida que o botão direito do *mouse* é pressionado e “arrastado” simultaneamente, os lados da bolha vão sendo esticados e /ou comprimidos a partir de um ponto inicial estabelecido pelo usuário no *canvas* da tela de edição. Essa técnica é útil no sentido de que o formato da bolha pode ser moldado antes mesmo que sua edição seja concluída. Caso a bolha seja considerada inválida, o EGS a recusará, emitindo uma mensagem de erro.

A janela de atributos é, na realidade, um misto de atributos e operações realizadas sobre a bolha que está sendo desenhada:

- Operação Refaz: apaga do *canvas* a bolha recém desenhada e reinicializa o processo de desenho. Essa opção é útil quando o usuário não aprovou o formato da bolha obtida e deseja redesenhá-la logo a seguir.

Figura 5.3: Criação de uma bolha.

- Operação Abandona: apaga a bolha, abandonando o processo de criação de bolhas.

- Operação Confirma: abandona a criação de bolha, armazenando-a na estrutura de dados.

- Atributo Nome: o nome que identifica uma bolha é digitado em um *panel* texto e a seguir transferido para o *canvas* com o auxílio do *mouse*. O posicionamento do nome no *canvas* fica a cargo do usuário.

- Tipo da bolha: por *default*, toda bolha criada é do tipo OR. Caso o usuário selecione, através desse *panel*, uma bolha do tipo AND, o EGS cria um outro *panel* que solicita o número de filhos (componentes ortogonais). Confirmado o número de filhos, o usuário deve repartir a bolha. Essa divisão é identificada por linhas pontilhadas, traçadas pelo usuário a partir de pontos próximos às bordas da bolha. O EGS é capaz de reconhecer se o ponto está suficientemente próximo à bolha para poder traçar a linha pontilhada. A seguir, para cada componente ortogonal (filhos) é solicitado um nome. Esse processo é controlado pelo EGS, isto é, não são aceitos nomes iguais para os filhos e, além disso, esses nomes devem ser posicionados, obrigatoriamente, no interior de uma das áreas divididas para a qual ainda não foi associado um nome.

Expansão de uma Bolha

Como foi dito no capítulo anterior, a tarefa de expansão corresponde ao refinamento de uma bolha. Na figura 5.4 é mostrado um processo de refinamento sucessivo de bolhas. A primeira janela, a partir da janela principal de edição, corresponde ao refinamento da bolha VISOR e a janela em primeiro plano visual mostra a bolha CRONÔMETRO refinada.

O processo de Expansão

Uma bolha somente pode ser refinada se for do tipo OR e se for folha (não possuir filhos). Ao selecionar, via *mouse*, uma bolha com essas características, o EGS cria uma nova janela de edição no canto superior esquerdo do vídeo, cujos tamanho e posição podem ser remodelados pelo usuário. A janela recém-criada passa, então, a ser a nova janela de edição na qual podem ser desenhadas novas bolhas e arcos, constituindo um sub-diagrama de statechart.

A janela que foi sobreposta permanece inibida enquanto existir uma outra janela sobre ela, ou seja, os eventos somente serão percebidos, e conseqüentemente os desenhos só poderão ser feitos, na ultima janela que foi criada (Na figura 5.4, essa janela é a de expansão da bolha cronômetro).

Figura 5.4: A expansão de uma bolha

O EGS permite que várias janelas de refinamento sejam criadas simultaneamente. O retorno a uma janela de nível visual imediatamente anterior é feito através do *panel* "Volta ao Nível Anterior". Por exemplo: na figura 5.4, pressionando-se esse *panel* da janela de expansão da bolha cronômetro, remove-se essa janela e ativa-se a edição na janela de expansão da bolha visor.

O XVIEW mostrou-se um sistema bastante poderoso no que diz respeito à manipulação de janelas sobrepostas. Por esse motivo, e considerando a utilização de alocação dinâmica de memória na criação das estruturas de dados, pode-se dizer que o EGS suporta um número de bolhas expandidas que não representa um fator limitante na criação de diagramas complexos.

Apagar Bolhas

Qualquer bolha pode ser removida de um diagrama. A seleção é feita pressionando-se o botão esquerdo do *mouse* no interior da bolha que se deseja remover.

Apagar uma bolha significa remover da estrutura de dados toda a sub-árvore do diagrama (constituída pela bolha e seus filhos) e todos os arcos (transições) que tenham origem e destino em alguma das bolhas dessa sub-árvore. Por esse motivo, antes de remover a bolha o EGS emite uma mensagem identificando a bolha a ser removida e pede que a operação seja confirmada.

Informações

Corresponde a uma consulta aos atributos da bolha selecionada. Essa seleção é feita da mesma maneira que na operação de apagar a bolha. Os atributos listados a seguir têm uma relação com os requisitos apresentados na seção 4.2:

- Tipo da Bolha: AND ou OR (Requisito 4 e 5.3 - Função *pi* e *Tipo*).
- Nível Hierárquico (Requisito 5.1 - Função *hierarquia*)
- Refinamento: sim ou não (Requisito 5.1).
- Número de filhos: se for bolha AND (Requisito 5.3).
- etc.

5.3.2 Manipulação de Arcos

O menu associado ao *panel* arco é composto dos seguintes itens:

- Criar

- Apagar
- Informações

A seguir será mostrado o significado e a maneira como o EGS executa cada uma dessas tarefas. Os requisitos mostrados nas duas últimas tabelas da seção 4.2, estão incorporados nas tarefas realizadas nesse módulo do sistema.

A Criação de Arcos

Identificam-se nos statecharts três tipos de arcos: *defaults*, 1:1 (um para um) e M:N (M para N). No EGS o acesso à criação de cada um deles é feito a partir de um outro menu, como é mostrado na figura 5.5.

O Processo de Desenho de um Arco

Apesar de existirem três tipos de arcos distintos, o processo de desenho de cada um deles, no *canvas*, é semelhante em todos os casos.

Um arco é uma sequência de pontos cujas posições são determinadas pelo usuário. O aspecto visual de um arco corresponde à aplicação de uma função Spline a esses pontos. Desse conjunto de pontos, dois têm significados especiais: o ponto inicial e o final.

Nesse aspecto, a interface do EGS é bastante amigável, uma vez que ela identifica, automaticamente, os pontos inicial e final assim que reconhece quais são as bolhas origem e destino da transição. Isso significa que o usuário não necessita informar ao sistema quando começará ou quando terminará de traçar o arco. Basta pressionar o botão esquerdo do *mouse* nas proximidades das bordas de uma bolha que o EGS identificará como sendo bolha origem da transição (se for o primeiro ponto) ou bolha destino (caso já exista uma bolha origem identificada).

O processo de construção é dirigido passo-a-passo (o formato do arco vai sendo pré-estabelecido à medida que pequenos segmentos de retas, unindo os pontos intermediários, vão sendo traçados). Para auxiliar o usuário, o EGS possibilita o uso da técnica rubber-band. Isso significa que o segmento de reta traçado entre dois pontos pode ser re-direcionado (pressionando-se o botão esquerdo do *mouse* e arrastando-o, simultaneamente). Esse processo permite que o usuário tenha um maior controle sobre o formato de um arco, possibilitando um traçado mais uniforme e evitando sobreposições com outros elementos do diagrama. Ao identificar um ponto final para o arco, o EGS substitui todas os segmentos de reta por uma curva suavizada.

Na figura 5.5, mostra-se um exemplo, ainda não concluído, da criação

Figura 5.5: O Processo de Criação de um Arco

de um arco. A bolha “alarme.beep” foi reconhecida como origem da transição e como ainda não foi determinado um ponto nas proximidades da borda de uma outra bolha, o EGS considera esses pontos como intermediários e traça retas entre eles. O efeito de *rubber-band* permite que as retas sejam direcionadas livremente.

História: atributo comum aos arcos *default*, 1:1 e M:N

Como foi visto na descrição dos requisitos do capítulo anterior, a história e sus variações (H e H*) são um atributo de um arco, ou seja, ao traçar um arco o usuário pode associar a ele um símbolo de história. Essa associação é feita selecionando-se no *panel* de história, via *mouse*, uma dentre as três opções existentes (veja a figura 5.6). O símbolo selecionado é gerado automaticamente e pode ser posicionado ao longo do arco, “arrastando-se” o *mouse* no *canvas*.

A terceira opção (branco) é utilizada para remover um símbolo de história sem que seja necessário apagar todo o arco.

Tipo do Arco: atributo comum aos arcos 1:1 e M:N

Todo arco 1:1 e M:N deve ser identificado por um nome. Essa rotulação pode ser classificada como primitiva ou complexa. Em uma rotulação complexa, um arco pode ser descrito em função de três *panels* (Rótulo, Eventos e Ações, ilustrados nos parágrafos abaixo). Já em uma rotulação primitiva, um arco é descrito por apenas um *panel* (Rótulo). A figura 5.6 ilustra a criação de um arco 1:2 dentro da janela de expansão da bolha cronômetro e para o qual foi atribuído um rótulo do tipo complexo.

O *panel* rótulo, cujo valor nesse caso é Ev1, não tem significado semântico. Ele é usado como identificador visual do arco e seu posicionamento é determinado pelo usuário com auxílio do *mouse*.

Os *panels* evento e ação correspondem à combinação dos requisitos 6, 7, 8 e 9 apresentados na seção 4.2 do capítulo anterior. Nesses *panels* o usuário digita cadeias de caracteres, que serão examinadas pelo analisador descrito na seção 4.3. Caso haja alguma inconsistência, o EGS recusa a expressão, emitindo uma mensagem de erro.

Se o tipo do arco escolhido for primitivo, o EGS cria apenas o *panel* rótulo. Nesse caso, além de ter um significado visual, o rótulo tem significado semântico, pois corresponde a um evento primitivo (requisito 8 da seção 4.2).

As Operações Abandona, Finaliza e Confirma

Existem três formas para se terminar a criação de um arco *default*, 1:1 e

M:N:

- Abandona: interrompe a criação em qualquer ponto, apagando do *canvas* o arco recém traçado.

- Finaliza: o EGS identifica automaticamente o ponto inicial e final de um arco. Isso pressupõe que todo arco deve ter início e destino nas bordas das bolhas. No entanto, em um diagrama de statechart é possível que existam arcos que terminem em algum ponto que não seja a borda de uma bolha. O objetivo dessa operação é a de forçar a finalização do traçado de um arco, de forma que o último ponto intermediário especificado pelo usuário se torne o ponto final do arco. Na figura 5.6 o sub-arco que termina dentro da bolha "ligado", foi criado dessa maneira.

- Confirma: abandona a operação, criando e armazenando as informações na estrutura de dados.

Particularidades da criação de um arco M:N

Ao iniciar o traçado de um arco M:N, o usuário deve estabelecer quais as quantidades de arcos M e N deseja ter. A seguir, deve posicionar no *canvas*, a área de intersecção representada por um pequeno quadrado (veja na figura 5.6). A função dessa área é servir de ponto de referência para a chegada dos sub-arcos M e saída dos sub-arcos N.

Os sub-arcos M são aqueles cujos pontos de origem são as bolhas origens da transição e o ponto destino, a área de intersecção. Os sub-arcos N são aqueles cujo ponto de origem é a área de intersecção e os pontos destinos, as bolhas destinos da transição.

A partir desse momento, desenhar um arco M:N corresponde ao processo de se traçar $M + N$ sub-arcos. O EGS controla automaticamente a quantidade de sub-arcos já traçados e "sabe" quando mudar o traçado de sub-arcos M para sub-arcos N. Além disso, o EGS permite que cada sub-arco seja apagado e refeito (*panel Refaz Sub Arco*). Isso evita ter que se apagar e refazer todo um arco por causa de um sub-arco incorreto ou de traçado muito irregular.

A Consistência dos Arcos

Os arcos de um statechart não podem ser traçados de forma aleatória, ou seja, existe um conjunto de regras a ser seguido no que diz respeito ao grupo de bolhas associado através de um arco. Portanto, ao finalizar um arco, o EGS dispara um processo de consistência do arco e caso encontre algum erro, este é reportado ao usuário na forma de uma mensagem que pode ser, por exemplo:

Figura 5.6: Um arco 1:2 e seus Atributos

- Bolha é filha de uma bolha AND.
- Existe um irmão como *default*.
- Ancestral comum é do tipo AND.
- Bolha raiz não pode ser *default*.
- Este arco já foi traçado, etc.

Apagar Arcos

Um arco *default*, ao contrário dos arcos 1:1 e M:N, não possui rótulo e o ponto de início, por não estar associado a uma bolha, é diferente dos demais. Visualmente é representado por um ponto preto de diâmetro maior. Por esse motivo, o processo de localizar arcos para serem apagados é subdividido da seguinte maneira:

- apagar arcos *default*: o arco é identificado pelo ponto inicial diferenciado.
- apagar arcos 1:1 e M:N: os arcos são identificados através da área ocupada pelo rótulo no *canvas* da janela de edição.

Em ambos os casos, antes de efetivar a remoção, o EGS identifica por meio de uma janela o arco a remover e pede que a operação seja confirmada ou então cancelada.

Informações

O EGS cria uma janela contendo informações referentes ao arco consultado. Essas informações são baseadas nos requisitos apresentados na seção 4.2. Como exemplo, pode-se citar:

- Bolha origem e bolha destino da transição (Requisito 10).
- Rótulo, evento e ação (Requisitos 6,7,8 e 9).

5.3.3 Arquivos: operações sobre a Base de Dados de Statechart

O menu principal associado ao *panel* arquivos é composto dos seguintes itens:

- Salva Statechart.
- Carrega Statechart.
- Salva como novo Statechart.
- Abandona Statechart.
- Sai do Sistema.

O padrão adotado pelo EGS na manipulação de arquivos é o mesmo utilizado pelo editor de texto do sistema OpenWindows2, que foi utilizado como ambiente de desenvolvimento do EGS.

- Salva Statechart: percorre a estrutura de Dados que está na memória principal gravando as informações no disco. Essa operação pode ser feita durante todo o processo de edição.

- Carrega Statechart: busca no disco o statechart especificado pelo usuário. Caso exista algum outro diagrama em edição, o EGS verifica, antes de carregar o novo statechart, se o diagrama antigo foi modificado. Se foi, o EGS emite mensagem de erro indicando que vai carregar um novo diagrama e que as alterações feitas no diagrama antigo serão perdidas. Nesse ponto, o usuário pode confirmar ou cancelar a operação.

É importante notar que no *panel* Diretório existente no canto superior direito da tela principal de edição está disponível essa mesma função. A vantagem em utilizá-lo, em vez da opção Carrega Statechart, é que o usuário tem uma lista dos diagramas já construídos e pode carregá-los apenas com o auxílio do *mouse*.

- Salva como novo Statechart: Salva statechart no disco. É utilizado em duas situações:

- O diagrama a salvar ainda não existe na base de dados.
- O usuário deseja fazer uma cópia de algum diagrama já existente na Base de Dados.

- Abandona Statechart: abandona a edição de um diagrama permanecendo no modo de edição. É útil quando são feitas algumas modificações no diagrama e se deseja que as mesmas sejam desprezadas. Também nesse caso, o EGS reporta ao usuário que as mudanças serão perdidas. Este pode, então, confirmar ou cancelar a operação.

- Sai do Sistema: abandona a edição do diagrama saindo do sistema. O EGS avisará, através de uma mensagem, caso haja algum diagrama em edição sendo modificado. O usuário pode, então, confirmar ou cancelar a operação.

5.3.4 Variáveis

O Menu associado ao *panel* Variáveis é composto de duas funções:

- Declaração
- Consulta

Formalmente, um statechart é uma linguagem para a qual são definidas expressões, condições, ações e eventos, que podem ser compostos por operadores, variáveis, etc. A descrição formal de uma linguagem para especificação de statecharts, encontra-se no apêndice C desta dissertação.

A opção “declaração” de variáveis permite que o usuário declare um conjunto de variáveis e, com isso, possa referenciá-lo na especificação de algum evento e/ou ação de um arco. Isso é necessário pois, como dito anteriormente, os eventos e ações são analisados sintática e semanticamente. Caso existam variáveis nesses eventos ou ações, elas precisam estar declaradas com antecedência.

As variáveis a declarar são especificadas em um *panel* texto. O tipo dessas variáveis é selecionado através de um outro *panel* que representa cinco tipos diferentes de variáveis:

- INT: variáveis inteiras.
- BOOL: variáveis booleanas.
- REAL: variáveis reais.
- STRING: cadeias de caracteres.
- COND: condições primitivas.

Ao selecionar a opção de Consulta, o EGS cria uma janela e lista todas as variáveis declaradas para aquele statechart.

5.4 Discussão sobre a generalização do EGS para um editor de Higraphs

Harrel, em [Har 88], mostra como certas técnicas de diagramação baseadas em grafos como, por exemplo, os Diagramas Entidade-Relacionamento, Redes Semânticas e Diagramas de Fluxos de Dados podem se beneficiar das características oferecidas pelos higraphs.

Nesta seção serão levantadas as possibilidades de utilização do EGS como ferramenta para representação de técnicas e metodologias baseadas em grafos, que não sejam os statecharts. Essa discussão se fundamenta no fato de que os statecharts são um tipo particular de higraphs. Portanto, o EGS (sob certas restrições e mediante algumas alterações que serão propostas) poderia ser generalizado e, assim, ser capaz de representar, não apenas uma, mas um conjunto de técnicas e metodologias baseadas em grafos.

Os Diagramas Entidade-Relacionamento (E-R)

Os diagramas E-R utilizados na especificação de base de dados podem ser considerados hipergrafos [Har 88], onde os nós, envolvidos em um retângulo, representam uma entidade. Os hiperarcos, cujos rótulos são escritos em pequenos losangos, representam a relação entre as entidades.

No entanto, os Diagramas E-R são uma forma de representação plana, o que pode levar à construção de diagramas confusos, especialmente quando existirem várias entidades e relacionamentos envolvidos.

Uma forma vantajosa e diagramaticamente econômica de se representar um diagrama E-R é utilizar um outro diagrama, baseado em higraphs, como mostra a figura 5.7. Nela descreve-se um pequeno modelo de dados de uma companhia aérea.

O encapsulamento das entidades *secretária*, *gerente* e *pilotos*, traduz a seguinte relação: *secretária* é um empregado, *gerente* também é um empregado, etc. Num diagrama E-R convencional essas relações teriam que ser especificadas explicitamente para cada uma das três entidades, o que pode tornar o diagrama complexo e até confuso.

O EGS poderia ser empregado como editor dessa notação estendida de diagramas E-R, observando-se dois aspectos:

- a) No EGS sempre existe uma bolha raiz. Portanto, seria necessário criar

Figura 5.7: Um Diagrama de E-R estendido com Higraphs

uma bolha de nível hierárquico superior (que poderia ser um identificador genérico para o diagrama) e a partir dela desenhar as entidades e relacionamentos.

b) Os rótulos deveriam vir envolvidos em um losango, como estabelece a notação.

Assim como os Diagramas E-R, as Redes Semânticas (que possui formalismo semelhante) podem se beneficiar da extensão baseada em higraphs mostrada na figura 5.7. Harel, em [Har 88], mostra um exemplo de como isso é possível.

Os Diagramas de Fluxo de Dados (DFDs)

Um outro formalismo baseado em grafos no qual os higraphs podem ser utilizados são os DFDs. Uma versão baseada em higraphs de tais diagramas chama-se Diagrama de Atividades. Esse diagrama corresponde a uma das três linguagens visuais da ferramenta Statemate apresentada no capítulo três desta dissertação. Num Diagrama de Atividades, as bolhas representam as funções ou atividades e as bolhas em nível hierárquico inferior, as sub-atividades. Os arcos correspondem aos fluxos de dados existentes.

Graficamente, a única restrição ao editar Diagramas de Atividades através do EGS é a necessidade de uma bolha raiz, a partir da qual são desenhadas as atividades e os fluxos. Já a aplicação do EGS como um editor de DFD convencional (que também é um hipergrafo) não seria possível diretamente, pois nesse caso existe mais do que um tipo de nó (bolha), como é o caso dos depósitos de dados, entidades externas, etc., que têm formato geométrico e significados diferentes.

A Metodologia JSD (Jackson Structured Development)

A metodologia JSD é um conjunto de técnicas aplicado passo a passo destinado a apoiar as fases de análise e projeto de sistemas. Uma dessas técnicas consiste da descrição de uma entidade do sistema através dos chamados Diagramas de Estruturas que mostram o relacionamento hierárquico dessas entidades com as ações. Graficamente as entidade e ações são representadas por retângulos que podem conter atributos gráficos. Assim seria necessário que um editor genérico de higraphs também pudesse ser capaz de associar atributos aos nós, e não apenas aos arcos e transições.

Os Diagramas de Estruturas podem ser vistos como higraphs, onde todas os nós (ações e entidades) têm um mesmo nível hierárquico, isto é, não existe a noção de agregação e refinamento entre os nós. O EGS pode ser usado para descrevê-los, contanto que se crie uma bolha raiz e a partir dela, se desenha o

diagrama propriamente dito.

Um outro diagrama existente na metodologia JSD são os Diagramas de Especificação de Sistemas (DES) mas, assim como os DFDs convencionais, esses possuem mais do que um tipo de nó. Portanto, os DES não podem ser representados diretamente através do EGS.

Conclusões

Do que foi exposto nesta seção, pode-se concluir o seguinte:

a) O EGS, por ser baseado no formalismo dos higraphs, pode ser prontamente utilizado com algumas restrições (além de seu propósito original que é a edição de diagramas de statechart), quando:

- Existem técnicas e formalismos baseados em grafos que possam se beneficiar da aplicação de higraphs como é o caso dos diagramas estendidos de E-R, Redes Semânticas e Diagramas de Atividades.
- Existem técnicas baseadas em grafos que mesmo não possuindo uma representação conveniente em higraphs, tenham uma estrutura baseada em arcos e apenas um tipo de nó. Como exemplo foram citados os Diagramas de Estruturas da metodologia JSD.

2) O EGS, por ser baseado no formalismo dos higraphs, pode ser generalizado de forma a dar apoio a técnicas e formalismos que sejam baseadas em grafos. Essa generalização pode ser feita, por exemplo, dotando o EGS com a capacidade de definir e manipular símbolos visuais, além dos que já existem para os statecharts. Uma outra possibilidade é parametrizar o editor de forma que o usuário possa: selecionar o formato geométrico dos elementos que compõem um higraph (que são, basicamente nós e arcos), escolher alguns atributos para esses elementos e definir regras de sintaxe para a técnica a apoiar. Em função desses três pontos, sugere-se a parametrização dos seguintes elementos:

a) Nós:

- Formatos geométricos diferentes, tais como: retângulos convencionais e losangos (para JSD), círculos e símbolos de depósitos de dados (para DFDs), etc.
- Atributos tais como: asteriscos (para JSD), formas variadas de identificação textual dos nós, etc.

b) Arcos:

- Atributos: terminações como setas (para DFD), “pés-de-galinha” e valores numéricos (para Diagramas E-R), etc.
- Rótulos: encapsulamento dos rótulos em símbolos geométricos (rótulos envoltos em losangos, para Diagramas E-R), visualização do rótulo em mais de uma linha, etc.

c) Regras de Sintaxe para as Técnicas:

- Cada técnica apoiada tem um conjunto pré estabelecido de regras sintáticas que devem ser consideradas durante a edição, a fim de que seja mantida a consistência do diagrama criado. Deve ser estabelecida alguma forma de generalização dessas regras.

5.5 Considerações Finais

Neste capítulo foi descrita a interface do Editor Gráfico de Statecharts desenvolvido, ilustrando-se com um exemplo os aspectos mais importantes da interface. Devido ao aspecto interativo de construção dos diagramas, não se mostrou com detalhes o exemplo, registrando-se apenas alguns “instantâneos” do processo de construção do diagrama.

Também foi discutida a possibilidade de generalização do EGS, no sentido de apoiar outras técnicas que sejam baseadas em grafos, como os diagramas JSD, E-R, etc.

Mostrou-se que o editor permite, de forma relativamente fácil, a construção fiel de diagramas de Statecharts, seguindo a definição formal de sua sintaxe estabelecida por Harel em [Har 87b] (Statecharts) e [Har 88] (Higraphs).

Capítulo 6

Conclusões

6.1 Considerações Iniciais

No capítulo 2 deste trabalho apresentou-se um tutorial detalhado dos Statecharts, no qual foi justificado porque essa técnica pode ser considerado um formalismo visual e como ela se aplica à descrição do comportamento de Sistemas Reativos. Justificou-se o formalismo dos statecharts apresentando-se os conceitos matemáticos e as principais características de um outro formalismo, que é a base dos statecharts: os HIGRAPHS.

No capítulo 3, apresentou-se a revisão bibliográfica, na qual foi dada ênfase à apresentação de técnicas e ferramentas destinadas à especificação de Sistemas Reativos. Pôde ser notado, em todas as ferramentas analisadas, a importância das interfaces gráficas na descrição do comportamento reativo de sistemas. Além disso, a maioria das ferramentas possui dispositivos para consistência das informações e mecanismos para simulação e geração de código, o que implica na existência de algum formalismo subjacente a essas ferramentas.

A revisão bibliográfica confirmou que: os Sistemas Reativos são diferentes dos Sistemas Transformacionais e, portanto, requerem uma abordagem diferente durante a fase de especificação e, ainda, que um elemento essencial na especificação de Sistemas Reativos é a necessidade de uma descrição comportamental clara e rigorosa.

Assim, foi desenvolvido um editor gráfico, cujo objetivo é permitir a edição de diagramas de statecharts e, em função do diagrama criado, gerar uma Base de Dados contendo informações gráficas (representação visual de cada ele-

mento do diagrama) e informações sintáticas. A existência deste editor gráfico representa um passo valioso para o ambiente de apoio ao desenvolvimento de Sistemas em Tempo Real, que está sendo desenvolvido no ICMSC-USP.

No contexto desse ambiente, o EGS representa a interface gráfica, isto é, através dele pode-se especificar graficamente um statechart e, a partir disso, realizar outras tarefas como, por exemplo, a simulação visual (as bolhas ligadas e desligadas são enfatizadas através de cores e cada passo da simulação é refletido no diagrama através do “apagamento” e “acendimento” de bolhas na tela). Anteriormente à existência do EGS, tanto a tarefa de especificação quanto a simulação só podiam ser feitas textualmente.

6.2 Resultados Obtidos

No que diz respeito à utilização de dispositivos, objetos e técnicas adotadas como padrão nas interfaces gráficas de sistemas mais recentes, pode-se afirmar que o EGS segue as mesmas tendências que as ferramentas apresentadas na revisão bibliográfica: os diagramas são gerados basicamente com o auxílio de mouse, os níveis hierárquicos são apresentados através da manipulação de janelas sobrepostas, a seleção de tarefas é feita por meio de *panels* e *menus* e, finalmente, existe um alto *feed-back* com o usuário ao reportar, imediatamente, mensagens de erro em função de consistências feitas durante a edição.

O padrão de interface que foi obtido deve-se, também, à decisão de utilizar o sistema XVIEW (baseado no X Window) como modelo de programação, pois ele permite que os elementos destacados no parágrafo acima sejam colocados na interface com uma relativa facilidade. No entanto, O EGS foi o primeiro software desenvolvido no ICMSC-USP sob o X Window, o que exigiu um esforço adicional na implementação da ferramenta. Pelos resultados obtidos, a tendência (a partir de agora) é que as demais aplicações sejam implementadas, também, sob o sistema XVIEW.

O autor acredita que o EGS, pelas características, base formal e recursos de edição gráfica que possui (como, por exemplo, a facilidade na edição de arcos), não é apenas um protótipo, podendo, com algumas extensões e melhor documentação, ser utilizado por profissionais da área.

Um outro fator a ser considerado, é a integração entre o EGS e a ferramenta de simulação desenvolvida por Fortes. Como foi dito na seção 4.3, a Base de Dados dos Statecharts é única para ambas as ferramentas. Isso significa

que um diagrama especificado graficamente através do editor resultará em informações para a base de dados, que serão utilizadas pela ferramenta de Fortes, responsável pela simulação. Como essa simulação é feita textualmente (essa integração já está em funcionamento), o próximo passo será a implementação de rotinas que possibilitem a simulação visual do diagrama. A partir de então, as duas ferramentas estarão completamente integradas.

O EGS, integrado à ferramenta de simulação, pode facilitar a divulgação e o uso dos statecharts em nosso meio, uma vez que já se dispõe de uma maneira fácil e visualmente agradável para se representar o comportamento reativo de sistemas, aliada à capacidade de simulação desse comportamento. As organizações que usem métodos de especificação de Sistemas Reativos baseados em diagramas de transição de estados, tais como os de Ward e Mellor [War 86], Hatley [Hat 84], Gomaa [Pre 87], etc., podem contar com o apoio desta ferramenta.

Um outro ponto importante é que o EGS, apesar de ser um editor para statecharts, pode ser usado (sob certas restrições) para apoiar outros formalismos que sejam baseados em grafos (Diagramas E-R, Redes Semânticas, Diagramas de Estruturas do JSD, etc.), pois o editor foi implementado sob o formalismo dos higraphs.

6.3 Sugestões para Futuras Pesquisas

Muitas técnicas e metodologias possuem alguma forma de representação visual que correspondem a grafos ou hipergrafos (DFDs, Diagramas de E-R, Diagramas de Atividades, Diagramas de Estruturas de JSD, etc.). Por ser baseado no formalismo dos Higraphs, o EGS pode ser generalizado e, assim, dar apoio a essas técnicas e metodologias.

Um primeiro passo nesse sentido, seria a generalização do EGS em um editor de Higraphs convencional. Em um higraph, ao contrário dos statecharts, as bolhas podem se intersectar e não existe, necessariamente, a idéia de uma bolha raiz a partir da qual, o resto do diagrama é gerado.

Com um editor geral de Higraphs, uma aplicação natural seria particularizá-lo para um editor de Atvivigramas ("Activity-Charts") o que estenderia a abrangência das ferramentas em desenvolvimento pelo grupo de Engenharia de Software do ICMSC-USP para apoio à especificação e desenvolvimento de Sistemas em Tempo Real, porque assim os aspectos de especificação funcional também teriam apoio por computador, ao lado da descrição comportamental,

já apoiada pelo EGS, pelo simulador de Fortes e por outras ferramentas em desenvolvimento.

Finalmente, algumas sugestões de pequeno porte podem contribuir para enriquecer o editor desenvolvido, oferecendo mais opções ao usuário, como por exemplo: impressão dos diagramas editados (quando houver disponibilidade de uma impressora a laser junto à estação), funções gráficas adicionais tais como: mover bolha e permitir a abstração de um conjunto de bolhas em uma bolha de nível hierárquico superior, a diponibilidade de *helps*, seleção do tamanho e tipo da letra (fonte) para descrever uma bolha ou arco, etc.

Bibliografia

- [Aar 85] Aarons, D. - *Zooming in for a Closer Look*. PC Magazine, 1: 87-93, 1985.
- [Alf 85] Alford, M. - *SREM at the Age of Eight: Distributed Computing Design System*. Computer, 18(4): 36-46, 1985.
- [Bae 86] Baer, J. - *Modelling Architectural Features with Petri Nets*. Berlin, LNCS, 255:258-277, Springer-Verlag, 1986.
- [Bal 83] Balzer, R. ; Cheatham, T.E. - *Software technology in the 1990's: Using a New Paradigm*. Computer, 11: 39-45, 1983.
- [Bei 83] Beineke, J.M. ; Wilson, R.J. - *Selected Topics in Graphs Theory 2*. New York, Academic Press, 1983.
- [Ber 73] Berge, C. - *Graphs and Hypergraphs*. New York, North-Holland Publishing Company, 1973.
- [Bld 88] Baldassari, M. - *PROTOB: a Hierarchical Object-Oriented CASE Tool for Distributed Systems*. Berlin, LNCS, Springer-Verlag, 340:422-445, 1988.
- [Blu 88] Blumofe, R ; Hecht, A. - *Executing Real-Time Structured Analysis Specification*. ACM SIGSOFT, 13(3): 32-40, 1988.
- [Bra 77] Brady, J.M. - *The Theory of Computer Science: A Programming Approach*. London, Chapman and Hall, 1977.
- [Bru 86] Bruns, G.R. et al. - *Design Technology Assessment: the Statechart Approach*. MCC Technical Report Number STP-107-86. Microelectronics and Computer Technology Corporation. Austin, Texas, 1986.
- [Dav 88] Davis, A.M. - *A Comparison of Techniques for the Specification of External System Behavior*. Communications of the ACM, 31(9): 1098-1115, 1988.

- [Fis 89] Fischer, A. - *Case: Utilização de Ferramentas para Desenvolvimento de Software*. Rio de Janeiro, Campos, 1989.
- [For 91] Fortes, R.P.M. - *Uma Ferramenta de Apoio à Utilização de Statecharts para Especificação do Comportamento de Sistemas de Tempo-Real Complexos*. Tese de Mestrado, ICMSC-USP, São Paulo, 1991.
- [Gan 79] Gane, C ; Sarson, T. - *Structured Systems Analysis: Tools and Techniques*. New Jersey, Prentice-Hall, 1979.
- [Har 87] Harel, D. - *A Visual Formalism for Complex Systems*. Sciences of Computer Programming, 8:231-74, 1987.
- [Har 87b] Harel,D., Pnueli,A., Schmidt,J.P. e Sheiman,R. - *On the formal semantics of statecharts*. IN Proceeding of the 2nd IEEE Symposium on Logic in Computer Science. Ithaca, N.Y., 1987.
- [Har 88] Harel, D. - *On Visual Formalisms*. Communications of the ACM, 31(5):514-530, 1988.
- [Har 88a] Harel, D. - *STATEMATE: a Working Environment for the Development of Complex Reactive systems*. IN Proceedings of the Tenth International conference on Software Engineering. (Singapore), Washington D.C., IEEE, 1988.
- [Hat 84] Hatley,D.J. - *The Use of Structured Methods in the Development of Large-Software Avionics Systems*. IN Proceedings of the AIAA/IEEE 6th Digital Avionics Systems Conference. (Baltimore), Washington D.C., IEEE, 1984.
- [Hat 88] Hatley,D.J. - *CASE Tool Evaluation: A Real-Time Example*. SLI Avionics Systems Corporation, april, 1988.
- [Hel 90] Heller,D. - *XView Programming Manual*. California, O'Reilly & Associates, Volume 7, July, 1990.
- [Hru 83] Hruschka, P. - *The Software Engineering Environment Promod*. Proc. ESA, ESTEC Software Engineering Seminar, Noorwijk, oct, 1983.
- [Jac 83] Jacob, R.J.K. - *Using Formal Specification in the Design of a Human-Computer Interface*. Communications of the ACM, 26(4):259-264, 1983.
- [Jan 86] Janson, A. ; Sutschet, G. - *A Process Oriented Approach for Qualitative Modelling and Analysis of Dynamical Systems*. Tex-b Memo Set, 1986.

- [Jun 88] Junk, W.S - *More than Just a Diagram Tool*. IEEE Software, (3): 97-99, 1988.
- [Lip 64] Lipschultz, S. - *Set Theory and Related Topics*. New York, Schaum Publishing, 1964.
- [Man 87] Manna, Z. ; Pnueli, A. - *Specification and Verification of Concurrent Programs by V-automata*. IN Proceedings of the 14th ACM Symposium on Principles of Programming Languages(Munich). New York, ACM, 1987.
- [Mar 89] Maraninchi, F. - *Argonaute: Graphical Description, Semantics and Verification of Reactive Systems by Using a Process Algebra*. Berlin, LNCS, Springer-Verlag, 407:38-53, 1989.
- [Mor 82] Moret, B. - *Decision Trees and Diagrams*. ACM Computing Surveys, 14(12): 593-623, 1982.
- [Nil 80] Nilsson, N.J. - *Principles of Artificial Intelligence*. Tioga, Palo Alto, California, 1980.
- [Owi 82] Owicki, S. e Lamport, I. - *Proving Liveness Properties of Concurrent Programs*. ACM Trans. Progr. Lang. Systems, 7:455-495, 1982.
- [Pre 87] Pressman, R. - *Software Engineering: a Practitioner's Approach*. Singapore, McGraw-Hill, 1987.
- [Pet 77] Peterson, J. - *Petri-nets*. ACM Computing Surveys, 9(3):223-252, 1977.
- [Rei 86] Reisig, W. - *Petri Nets in Software Engineering*. Berlin, LNCS, 255:63-96, 1986.
- [Sch 86] Scheifler, R.W. - *The X Window System*. New York, ACM Transactions on Graphics, 5(2):79-109, 1986.
- [Sun 82] Sunshine, C. A. et alli - *Specification and Verification of Communication Protocols in AFFIRM using State Transition Models*. IEEE Transactions on Software Engineering, 8:460-489, 1982.
- [Tan 81] Tanenbaum, A. S. - *Computer Networks*. New Jersey, Prentice Hall, Englewood Cliffs, 1981.
- [Tyg 87] Tygar, J.D. e Wing, J.M. - *Visual Specification of Security Constraints*. IN the IEEE Workshop on Visual Languages. New York, IEEE Press, 1987.

- [War 85] Ward, P. ; Mellor, S. - *Structured Development for Real-time Systems*. New Jersey, Prentice-Hall, 1985.
- [War 86] Ward, P. - *The Transformation Schema: an Extension of the Data Flow Diagram to Represent Control and Timing*. IEEE Transaction on Software Engineering, 12(2):198-210, 1986.
- [Whi 87] White, S.M. - *A Pragmatic Formal Method for Computer System Definition*. PhD Dissertation. Polytechnic University, N. York, 1987.
- [Won 76] Wong, E. e Youssefi, K. - *Decomposition: A Strategy for Query Processing*. ACM Transaction on Database Systems, 1(3):223-241, 1976.

Apêndice A

Tipos e Estruturas de Dados da Parte Gráfica do EGS

A listagem apresentada a seguir corresponde a parte relativa às informações gráficas do arquivo “estrut.h”, no qual estão declaradas todos tipo e estruturas de dados utilizada pelo EGS. Também são mostrados os arquivos que compõem a Base Dados dos Statecharts.

```
/*-----
                                estrut.h  (TAD)

Este arquivo contem as estruturas basicas para edicao e simulacao de um
diagrama de STATECHARTS
-----

/*****  CONSTANTES UTILIZADAS NO EDITOR GRAFICO  *****/

#define M_ERRO 7                /* Limite de erro p/ selecao de bolha com mouse
#define ARC_DEFAULT 0           /* Identifica um arco do TIPO DEFAULT
#define UM_PARA_UM 1           /* Identifica um arco do tipo 1 para 1
#define ARCO_M 2                /* Identifica um arco sem seta na ponta(M)
#define ARCO_N 3                /* Identifica um arco do tipo N
#define MAX_PTS 30              /* Nro max. de pontos de um arco p/ tracar SPLINE
#define NORMAL 1                /* Indica que esta editando uma bolha normal
#define EXPANDIDO 0             /* Indica que esta editando uma bolha expandida
```

```

/*-----

/* ----- TIPOS E ESTRUTURAS DA INTERFACE COM USUARIO -----

/* Par FRAME_CANVAS. Transfere para uma janela filho, qual o canvas
   e o frame da janela pai

typedef struct estr1
{
    Frame fr;
    Canvas can;
}PAR;

/* ----- TIPOS E ESTRUTURAS GRAFICAS -----

/***** Ponto na tela. x -> Eixo X; y -> Eixo Y *****/
typedef struct coorde
{
    int x,y;
}COORDENADA;

/***** COORD. (x,y) REAL. Para tracar arco c/ SPLINE *****/
typedef struct tipo_ponto
{
    float x;
    float y;
}COORD_REAL;

/***** INF. p/ Armazenar uma curva e traca-la c/ SPLINE *****/
typedef struct tipo_curva
{
    int ordem;
    int nro_pontos;           /* Quantidades de pontos validos na malha
    COORD_REAL malha[MAX_PTS]; /* Vetor de pontos relativos ao arco
}TIPO_CURVA;

/***** Estrut. de um ARCO *****/
typedef struct arco
{
    TIPO_CURVA arc;           /* Os pts. que geram o arco

```

```

        COORDENADA hist,          /* posicao da historia(Se existir)
            pos_rot;              /* Posicao do rotulo(nomeo)
        int valor_hist;          /* Tipo da historia(H,H*, "")
        int nivel_visual;       /* Nive de visualizacao do arco
        CADEIA rotulo;          /* Nome que aparece no video
        CADEIA acao;            /* Expressao de acao p/ arco complexo
        CADEIA evento;         /* Evento de um arco complexo
    }ARCO;

/***** declaracao da estrutura RETANGULO *****/

typedef struct list {
        COORDENADA c1,          /* Canto superior esquerdo do retangulo
            c2;                /* Canto inferior direito do retangulo
        struct list *prox;
    }RET;

/***** declaracao da estrutura GRAF *****/
/* Armazena todas as informacoes graficas necessarias para repre-
   sentar uma bolha e seus atributos */

typedef struct graf {
        int x,                 /* Coord. do canto superior esquerdo
            y,
            x_larg,            /* Coord. do canto inferior direito da bolha
            y_alt,
            x_nome,            /* Coord. posicao inicial de escrita do nome
            y_nome;
        COORDENADA cse;       /* Canto superior da janela que contem a bolha
        COORDENADA lh;        /* Largura (l) e altura (h) da janela
        ARCO arc_hist;        /* Se houver historia, armazenar arco
    }GRAF;

/***** Estrutura da pilha de janelas *****/
/* Informacoes referentes a hierarquias de janelas que vao sendo
   criadas a medida que se expande as bolhas */

typedef struct pilha_jan {
        PAR *fr_can;          /* Apontadores p/ canvas e frame da janela
        int nivel;            /* nivel de visualizacao da janela
        struct tab *bolha;    /* bolha a ser expandida

```



```

        struct pilha_jan *prox;
    }NO_PILHA_JANELA;

/* ----- TIPOS E ESTRUTURAS PARA SIMULACAO ----- */
/* Note que as estruturas declaradas a partir de agora incorporam
   as estruturas graficas definidas anteriormente. */

/***** declaracao da estrutura TAB-EST *****/

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

NO_ESTADO *subest[MAX_NRO_EST]; /* subest:vetor ponts p/ sub-estados

/*-----*/
/**** Estr. temporaria que guarda os sub arcos tracados que depois
    serao guardados nas sub transicoes(Usado em arcos M pa N) *****/

typedef struct lista_estados {
    NO_ESTADO *bolha; /* Ponteiro para a bolha corresp. */
    struct lista_estados *prox; /* pont. para proximo no da lista */
    ARCO arco; /* Inf. Graficas do sub arco */
}LISTA_ESTADOS;

/*-----*/
/***** declaracao da estrutura da Pilha Polonesa *****/

```

```

typedef struct polon {
    int tipo_item;                /* tipo do item a ser avaliado
    int ind_item;                 /* indice do item
}PILHA_POL[TAM_POL];

/*-----*/
/***** declaracao da estrutura da LISTA DE TRANSICOES *****/

typedef struct trans {
    PILHA_POL *ev_aval;           /* evento a ser avaliado
                                   para efetivacao da transicao
    PILHA_POL *ac_aval;           /* Pilha polonesa p/ avaliacao de um acao
    int ind_est_fonte;            /* indice estado fonte
    int ind_est_destino;          /* indice estado dest
    char h[4];                   /* indica historia: H / H*
    struct trans *prox_tr;        /* pont. para a prox trans
    struct sub_trans *est_f;      /* pont. para est_fonte da
                                   trans paralela
    struct sub_trans *est_d;      /* pont. para est_dest da
                                   trans paralela
    int ind_gravacao;            /* indice de gravacao
    int sentinela;               /* alerta = 4567
    ARCO arco;                   /* Inf. graficas do arco
}TRANSICAO;

TRANSICAO *tr_ok[MAX_NRO_TR];    /* vetor de ponts. p/ trans habilit

/*-----*/
/***** declaracao da estrutura DE TRANSICOES PARALELAS *****/

typedef struct sub_trans {
    int ind_est;                 /* indice estado
    char h[4];                   /* indica historia: H / H*
    struct sub_trans *prox_sub_tr; /* pont. p/ prox tr
    ARCO arco;                   /* Inf. graficas do sub arco
    int sentinela;               /* alerta = 4567
}SUB_TRANSICAO;

/*-----*/
/***** declaracao da estrutura de TRANS relativa ao EVento *****/

```

```

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

```

```

/*-----*/
/***** declaracao da estrutura de TAB-EV *****/

```

```

typedef struct evento {
    CADEIA nome_ev;           /* nome do evento
    int valor_ev;             /* valor do evento
    TRANS_EV *trans;          /* pont. p/ inic lista de
                                   transicoes deste evento
    int sentinela;            /* alerta = 9876
} NO_EVENTO;

```

```

NO_EVENTO *event[MAX_NRO_EV]; /* vetor de ponts. p/ eventos STTCHART */

```

```

/*-----*/
TIPOS E ESTRUTURAS DOS ARQUIVOS

```

Estas estruturas compoem os arquivos que representam a BASE DE DADOS DE
ESPECIFICACAO DE STATECHARTS.

```

-----*/

```

```

/***** Registro de um ARCO *****/

```

```

typedef struct regis_arco
{
    TIPO_CURVA arc;           /* Os pts. que geram o arco
    COORDENADA hist,          /* posicao da historia(Se existir)
        pos_rot;              /* Posicao do rotulo(nomeo)
    int valor_hist;           /* Tipo da historia(H,H*, "")
    int nivel_visual;         /* Nivel de visualizacao do arco
    char rotulo[40];          /* Nome que aparece no video
    char acao[500];           /* Expressao de acao p/ arco complexo
    char evento[500];         /* Evento de um aarco complexo
} REGISTRO_ARCO;

```

```

/***** registro a ser gravado para TABELA DE ESTADOS *****/

```

```

typedef struct regis_estado {
    int ind_est;                /* indice do estado
    char nome_est[30];          /* nome do estado
    int tipo;                   /* tipo final(fim/and/xor)
    int defo;                   /* default/ nao default / defH / defH*
    int nivel;                  /* altura da arvore (raiz=0)
    int nivel_visual;          /* Nivel de representacao visual da bolha
    int lig;                    /* indica estado ligado
    int ult_lig;                /* indica historia ligado
    GRAF info_grf;              /* Inform. graficas de uma bolha
    GRAF info_exp;              /* Informacoes de uma bolha expandida
}REGISTRO_EST;

```

```

/*-----*/
/***** registro a ser gravado para TABELA DE EVENTOS *****/

```

```

typedef struct regis_ev {
    int ind_ev;                /* indice do evento
    char nome_ev[30];          /* nome do evento
    int qtdd_trans;            /* quantas trans relativas ao evento
}REGISTRO_EV;

```

```

/*-----*/
/** registro a ser gravado para TABELA DE TRANSICOES *****/

```

```

typedef struct regis_tr {
    int ind_tr;                /* indice da transicao
    int qt_pol_ev;              /* quantas linhas-pol eventos
    int qt_pol_ac;              /* quantas linhas-pol acao
    int ind_estf;               /* ind do estado-fonte
    int ind_estd;               /* ind do estado-dest
    char h[4];                  /* historia: H / H*
    int nro_sub_tr_f;           /* nro de est_fontes paralelos
                                na sub_trans
    int nro_sub_tr_d;           /* nro de est_dest paralelos
                                na sub_trans
    REGISTRO_ARCO arc;          /* Inf. graficas do arco
}REGISTRO_TR;

```

```

/*-----*/
/**registro a ser gravado p/ as TABELAS DE SUB-TRANSICOES *****/
typedef struct regis_sub_tr {
    char ind_est;                /* indice do estado
    char h[4];                  /* historia: H / H*
    REGISTRO_ARCO arc;          /* Inf. graficas do sub arco
}REGISTRO_SUB_TR;

/*-----*/
/** registro a ser gravado para as tabelas PILHAS-POLONESAS *****/

typedef struct regis_polon {
    int tipo_item;              /* tipo do item a ser avaliado
    int ind_item;              /* indice do item *
}REGISTRO_POL;

/*-----*/
/***** registro a ser gravado para as TRANS_EVENTOS *****/

typedef struct regis_trans_ev {
    int ind_ev;                /* indice do evento
    int ind_tr;                /* indice da transicao
}REGISTRO_TRANS_EV;

/*-----*/
/***** registro a ser gravado para a TABELA DE VARIAVEIS *****/

typedef struct regis_vars {
    int ind_var;                /* indice da variavel
    int tipo_var;              /* tipo da variavel
    union {
        int valor_i;
        float valor_f;
        char valor_byte;
        char *valor_s;
    } u;                        /* valor da variavel
    char nome_var[30];          /* nome da variavel
}REGISTRO_VARS;

```

Apêndice B

A Definição Formal dos Higraphs

A seguir serão apresentadas, formalmente, a sintaxe e semântica dos higraphs, baseadas na definição apresentada por Harel em [Har 88].

Um higraph é uma quádrupla

$$H = (B, A, \sigma, \pi),$$

onde B é um conjunto finito de elementos chamados **bolhas**, e A , o conjunto de **arcos**, é uma relação binária em 2^B :

$$A \subseteq 2^B \times 2^B$$

A função sub-bolha σ é definida como

$$\sigma : B \rightarrow 2^B$$

Ela associa a cada bolha $x \in B$ seu conjunto $\sigma(x)$ de sub bolhas que é livre de ciclos. Assim, ao definir

$$\sigma^0(x) = x, \quad \sigma^{i+1}(x) = \bigcup_{y \in \sigma^i(x)} \sigma(y),$$

$$\text{and } \sigma^+(x) = \bigcup_{i=1}^{\infty} \sigma^i(x),$$

então σ é restrita de forma que $x \notin \sigma^+(x)$.

A função de partição π é definida como

$$\pi : B \rightarrow 2^{B \times B},$$

associando a cada bolha $x \in B$ alguma relação de equivalência $\pi(x)$ ao conjunto de sub-bolhas, $\sigma(x)$. Na realidade, isso é uma forma rigorosa de se especificar a quebra de x em seus componentes ortogonais, que agora são definidas como sendo as classes de equivalência induzidas pela relação $\pi(x)$. De fato, para $x \in B$ denota-se essas classes por $\pi_1(x), \dots, \pi_{k_x}(x)$. Para que a divisão ortogonal em componentes seja representável graficamente (e assim tornar a semântica mais clara) foi estabelecido que as bolhas em diferentes componentes ortogonais de x devem ser disjuntas. Formalmente, para cada x requer-se que qualquer par de elementos y e z de $\sigma(x)$ não possa se intersectar - isto é, possa satisfazer $\sigma^+(y) \cap \sigma^+(z) \neq \emptyset$ - a menos que esses elementos estejam no mesmo componente ortogonal. A representação $k_x = 1$, significa que x não possui componentes ortogonais.

A Semântica dos Higraphs

Dado um higraph H , define-se o conjunto de bolhas atômicas como sendo

$$A = \{x \in B \mid \sigma(x) = \emptyset\}.$$

Obviamente, o fato de B ser um conjunto finito e a ausência de ciclos sobre σ , implica que A é não vazio. O produto cartesiano não ordenado de dois conjuntos S e T é definido como sendo

$$S \otimes T = \{\{s, t\} \mid s \in S, t \in T\}.$$

Dado um higraph H , um modelo para H é o par $M = (D, \mu)$, onde D é um conjunto de elementos chamado domínio do modelo M , e μ associa subconjuntos disjuntos de D às bolhas atômicas de H . Então,

$$\mu : A \rightarrow 2^D,$$

onde, se $x \neq y$, então $\mu(x) \cap \mu(y) = \emptyset$. Agora será mosrtrado como estender a associação de bolhas atômicas com conjuntos sobre D a uma associação de todas as bolhas com objetos mais complexos sobre D . Isso pode ser feito ao se estender μ , definindo-se indutivamente, para cada $x \in B$,

$$\mu(x) = \bigotimes_{i=1}^{k_x} \left(\bigcup_{y \in \pi_i(x)} \mu(y) \right),$$

e que para calcular a semântica de uma bolha x , deve-se gerar o produto cartesiano não ordenado do significado de seus componentes ortogonais, onde cada qual, por sua vez, é simplesmente a união dos significados das bolhas que constituem o produto cartesiano. Se $k_x = 1$, não haverá nenhum produto e ter-se-á

$$\mu(x) = \bigcup_{y \in \sigma(x)} \mu(y),$$

como esperado.

Para completar a semântica, é importante notar que o conjunto de arcos A induz a relação semântica A_M em $\mu(x)$ s, definida por

$$(\mu(x), \mu(y)) \in A_M \quad \text{iff} \quad (x, y) \in A.$$

Apêndice C

Gramática da Linguagem de Especificação de Statecharts

Essa é a notação em BNF da Gramática de Statechart definida por Fortes, em [For 91]. Os nomes dos arcos (Rótulos, Eventos e ações) e as variáveis, devem ser declarados segundo a sintaxe apresentada a seguir.

```
<estadograma> ::= [ <declaracoes> ] ';'
                  <estados> ';'
                  <transicoes> '.'

<declaracoes> ::= <declaracao> { ';' <declaracao> }

<declaracao> ::= <tipo-id> <lista-identificadores>

<tipo-id> ::= ( int | bool | real | string | cond )

<lista-identificadores> ::= <identificador> { ';' <identificador> }

<estados> ::= <descricao estado> { ';' <descricao estado> }

<descricao estado> ::= estado <estado> <tipo>

<tipo> ::= ou <subestados ou> | e <subestados e> | atomo

<subestados ou> ::= subestados <estado>
                  [ ( def | def/H | def/H* ) ]
                  { ';' <estado> }

<subestados e> ::= subestados <estado> { ';' <estado> }
```

<transicoes> ::= <transicao> { ';' <transicao> }

<transicao> ::= evento <evento> { ';' <evento> }
origem <estado> { ';' <estado> }
destino <estado> [(/H | /H*)]
{ ';' <estado> [(/H | /H*)] }
[acao {' <acoes> '}]

<evento> ::= <identificador>
| <true> '(' <exp> ')'
| <false> '(' <exp> ')'
| <changed> '(' <exp> ')'
| <exit> '(' <estado> ')'
| <entered> '(' <estado> ')'
| <evento> or <evento>
| <evento> and <evento>
| <evento> '[' <exp> ']'
| t_out '(' <numero> <unid-tempo> ')'

<exp> ::= <expS> [(= | > | < | <> | <= | >=) <expS>]

<expS> ::= <termo> { (+ | - | or) <termo> }

<termo> ::= <fator> { (* | / | and) <fator> }

<fator> ::= <true>
| <false>
| <condicao primitiva>
| <numero>
| <variavel>
| in '(' <estado> ')'
| <not yet> '(' <evento> ')'
| <current> '(' <exp> ')'
| not <exp>
| '(' <exp> ')'

<acoes> ::= <acao> { ';' <acao> }

<acao> ::= <identificador>
| <variavel> ':=' <exp>
| clear '(' <estado> ['*'] ')'

$\langle \text{estado} \rangle ::= \langle \text{identificador} \rangle$
 $\quad | \langle \text{identificador} \rangle ' . ' \langle \text{identificador} \rangle$

$\langle \text{variavel} \rangle ::= \langle \text{identificador} \rangle$

Símbolos terminais :

$\langle \text{true} \rangle ::= ([tT]/[rR]/[uU]/[eE] | \text{tr})$

$\langle \text{false} \rangle ::= ([fF]/[aA]/[lL]/[sS]/[eE] | \text{fs})$

$\langle \text{changed} \rangle ::= (\text{changed} | \text{ch})$

$\langle \text{exit} \rangle ::= (\text{exit} | \text{ex})$

$\langle \text{entered} \rangle ::= (\text{entered} | \text{en})$

$\langle \text{not yet} \rangle ::= (\text{not\_yet} | \text{ny})$

$\langle \text{current} \rangle ::= (\text{current} | \text{cr})$

$\langle \text{identificador} \rangle ::= [a - zA - Z]$
 $\quad [a - zA - Z 0 - 9]^*$

$\langle \text{numero} \rangle ::= [0 - 9]^+$

$\langle \text{unid-tempo} \rangle ::= (\text{seg} | \text{min} | \text{hr})$