UNIVERSIDADE DE SÃO PAULO

Introdução a Statecharts: Conceitos Básicos, Teste e Validação, Extensões e Ferramentas de Apoio

Tatiana Sugeta
José Carlos Maldonado
Sandra C.P.F. Fabbri
Paulo Cesar Masiero

Nº 41

NOTAS DIDÁTICAS

Instituto de Ciências Matemáticas de São Carlos

Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2585

Introdução a Statecharts: Conceitos Básicos, Teste e Validação, Extensões e Ferramentas de Apoio

Tatiana Sugeta
José Carlos Maldonado
Sandra C.P.F. Fabbri
Paulo Cesar Masiero

Nº 41

NOTAS DIDÁTICAS DO ICMC

São Carlos
Dez./1999

Resumo

Statecharts são uma extensão das Máquinas de Estados Finitos e baseiam-se em três aspectos: hierarquia, ortogonalidade e *broadcasting*. Esses três aspectos são essenciais para a especificação de sistemas complexos, permitindo um maior poder de representação. Nesta nota didática, apresentam-se as principais características da técnica Statecharts, bem como sua sintaxe e semântica formalmente definidas. Hypercharts são uma extensão de Statecharts que podem ser aplicadas na especificação de sistemas hipermídia. A técnica Hypercharts também é brevemente introduzida neste trabalho. Os ambientes StatSim e HySCharts, que apóiam a utilização de Statecharts em diferentes contextos, são também apresentados. Aspectos de teste e validação de Statecharts também são discutidos, e a aplicação da Análise de Mutantes nesse contexto é ilustrada com mais detalhes, assim como a ferramenta Proteum-RS/ST, que apóia o teste de especificações Statecharts baseada na Análise de Mutantes.

Índice

Índice de Figuras

Índice de Tabelas

CAPÍTULO 1

INTRODUÇÃO

Sistemas baseados em computação estão sendo utilizados em praticamente todas as áreas da atividade humana, provocando uma crescente demanda por qualidade e produtividade. A Engenharia de Software é uma disciplina que evoluiu nas últimas décadas procurando estabelecer técnicas, critérios, métodos e ferramentas para a produção de software. No processo de desenvolvimento tem-se a Garantia de Qualidade de Software, que é um conjunto de atividades de apoio, entre as quais tem-se as atividades de VV&T - Verificação, Validação e Teste, que visam a assegurar alta qualidade a cada passo do processo.

A atividade de Teste de Software visa a aprimorar a produtividade e a fornecer evidências de confiabilidade e qualidade de um produto de software, em complemento a outras atividades, como por exemplo, o uso de revisões e de técnicas formais e rigorosas de especificação e de verificação (Maldonado, 1997).

O software pode ser classificado em diversas categorias: Software Básico, Sistemas de Informação, Sistemas Científicos, Sistemas Embutidos, Sistemas Pessoais, Sistemas de Inteligência Artificial e Sistemas Reativos. Os Sistemas Reativos caracterizam-se por interagir continuamente com o ambiente, reagindo a eventos externos gerados pelo processo controlado. Incluem-se nessa classe os Sistemas de Tempo Real, Sistemas Embutidos e Sistemas Críticos com relação à segurança (Harel, 1987a; Allworth, 1981; Davis, 1988). Como exemplos desses sistemas, podem-se citar controle de tráfego aéreo, controle metroviário, controle de monitoramento hospitalar, dentre outros. Portanto, os Sistemas Reativos controlam algumas atividades humanas essenciais e por isso a atividade de teste no desenvolvimento dos mesmos é ainda mais crucial, dado que a ocorrência de falhas nesses sistemas podem colocar em risco vidas humanas ou determinar elevados prejuízos materiais. Desse modo, o desenvolvimento desses sistemas merece atenção especial, sendo necessária a utilização de técnicas especializadas, seguras e eficientes para a produção desse tipo de software.

De acordo com a Engenharia de Software, métodos devem ser utilizados para apoiar o trabalho do projetista durante todo o processo de desenvolvimento do software, visando assim ao aumento da qualidade e à minimização da inserção de erros no software (Fabbri, 1996). Classificam-se esses métodos como informais ou formais, sendo que a principal diferença está no fato de que os métodos informais não possuem base matemática, podendo gerar ambigüidade, enquanto que os métodos formais possuem embasamento matemático, dado geralmente por uma linguagem de especificação formal.

Os métodos formais permitem que se especifique, desenvolva e verifique o software ou parte dele, de modo sistemático. A sua base matemática fornece meios de se definirem com precisão noções de consistência, corretitude e completitude, além de possibilitar a verificação de propriedades do software, como ambigüidade, sem a necessidade de executá-lo para analisar seu comportamento. Esses métodos são utilizados geralmente para especificar o comportamento e propriedades estruturais do software. Se aplicados nas fases iniciais do desenvolvimento do software, eles podem evitar erros que só seriam descobertos nas fases de teste e depuração, e quando utilizados em fases mais

adiantadas do desenvolvimento, auxiliam na determinação da corretitude da implementação e da equivalência de diferentes implementações. Exemplos de técnicas utilizadas pelos métodos formais são: Máquinas de Estados Finitos (MEFs) (Gill, 1962), Redes de Petri (Peterson, 1981) e Statecharts (Harel, 1987a).

Os Sistemas Reativos requerem a especificação do relacionamento de entradas e saídas no tempo. Tais descrições envolvem seqüências complexas de eventos, ações, condições e fluxo de informações, freqüentemente com restrições de tempo explícitas que se combinam para formar o comportamento do sistema. A Análise Estruturada e Métodos de Projeto Estruturado não lidam adequadamente com a dinâmica dos Sistemas Reativos, pois foram propostos para lidar com aplicações não-reativas, dirigidas a dados, em que uma boa decomposição funcional e uma descrição de fluxo de dados são suficientes (Harel et al., 1990).

Uma técnica que pode ser utilizada durante a fase de especificação dos requisitos para apoiar a especificação do aspecto comportamental dos Sistemas Reativos é a linguagem natural, detalhando o comportamento esperado do sistema. Porém, a linguagem natural é ambígua, resultando em uma documentação também ambígua e inconsistente (Fabbri, 1996). Assim, devido à sua base matemática, os métodos formais podem ser utilizados na especificação do aspecto comportamental dos Sistemas Reativos, auxiliando na redução de possíveis erros. Segundo Fabbri (1996), as Máquinas de Estados Finitos, Statecharts e Redes de Petri são intensamente utilizadas na especificação do aspecto comportamental desses sistemas.

Em muitos casos, os métodos oferecem apoio para a execução das especificações (Fabbri, 1996); assim, a especificação pode ser usada pelo projetista para fornecer um feedback da própria especificação e permitir a elaboração de protótipos. Dentre esses métodos estão Statecharts, que são uma extensão das Máquinas de Estados Finitos e baseiam-se em três aspectos: hierarquia, ortogonalidade e *broadcasting*. Esses três aspectos são essenciais para a especificação de sistemas complexos, permitindo um maior poder de representação. A sintaxe e a semântica de Statecharts (Harel, 1987b; Harel & Naamad, 1996) também são bem definidas e possibilitam a análise dos aspectos dinâmicos do sistema.

Normalmente, as especificações feitas em Statecharts são validadas por meio dos vários tipos de simulação: *batch*, exaustiva, interativa e programada. Porém, essas abordagens não consideram uma questão relevante para a atividade de teste, o aspecto de cobertura, que é a análise da satisfação dos requisitos estabelecidos pelos critérios de teste e que devem ser considerados. Com a análise de cobertura a qualidade da atividade de teste pode ser quantificada. Nesse contexto, tem-se explorado a aplicação do critério de teste Análise de Mutantes no teste de especificações de Sistemas Reativos (Fabbri, 1996), considerando principalmente as especificações baseadas em Máquinas de Estados Finitos, Statecharts e Redes de Petri.

O objetivo deste trabalho é apresentar as principais características da técnica Statecharts, bem como sua sintaxe e semântica formalmente definidas, destacando-se as vantagens que se obtêm ao utilizar essa técnica na especificação de sistemas. Sua extensão Hypercharts, que pode ser aplicada na especificação de sistemas hipermídia, também é brevemente caracterizada. Os ambientes StatSim (Masiero et al., 1991) e HySCharts (Turine et al., 1998), que apóiam a utilização de Statecharts, são apresentados. O ambiente StatSim permite a edição de Statecharts e sua simulação nas mais diferentes formas. Já o HySCharts é um ambiente de autoria e navegação que permite criar e validar especificações de hiperdocumentos de acordo com o modelo HMBS (Hyperdocument Model Based on Statecharts), que utiliza a estrutura e a semântica operacional de Statecharts na especificação da estrutura organizacional e a semântica de navegação de

hiperdocumentos grandes e complexos (Turine et al., 1998). Aspectos de teste e validação de Statecharts também são discutidos e a aplicação da Análise de Mutantes nesse contexto é apresentada. A ferramenta de teste de especificações Statecharts baseada na Análise de Mutantes, Proteum-RS/ST, também é apresentada.

Este trabalho está organizado da seguinte forma: no Capítulo 2 são apresentadas a sintaxe e a semântica da técnica Statecharts, algumas de suas características e aspectos de teste e validação dessa técnica. Também nesse capítulo é apresentada a extensão de Statecharts, os Hypercharts, utilizados na especificação de aplicações hipermídia. No Capítulo 3 apresentam-se a ferramenta StatSim, com todas as formas de simulação de Statecharts apoiadas por esse ambiente, e o ambiente HySCharts, com a descrição de suas características principais. A aplicação do critério Análise de Mutantes no teste de especificações, enfatizando a técnica Statecharts, é apresentada no Capítulo 4, assim como a ferramenta de teste de especificações baseadas em Statecharts, Proteum-RS/ST (Fabbri, 1996; Sugeta, 1999; Sugeta et al., 1999). No Capítulo 5 encontram-se as considerações finais deste trabalho. No Apêndice A apresenta-se a sintaxe da Linguagem de Especificação de Statecharts (LES) utilizada para representar os statecharts na forma textual no ambiente StatSim e na ferramenta Proteum-RS/ST. No Apêndice B apresenta-se a Linguagem de Controle de Execução (LCE) utilizada para gerar programas de controle da execução programada no StatSim.

CAPÍTULO 2

A TÉCNICA STATECHARTS

O problema de se especificar e projetar Sistemas Reativos grandes e complexos, segundo Harel (Harel, 1987a), está na dificuldade de se descrever seu comportamento reativo de maneira clara e real e, ao mesmo tempo, formal e rigorosa o suficiente para permitir sua simulação detalhada. O comportamento de um Sistema Reativo é ,na realidade, um conjunto de seqüências permitidas de eventos de entrada e saída, condições e ações, além de algumas informações adicionais, tais como restrições de tempo.

Harel (1987a) afirma que os diagramas de estados convencionais são impróprios para a descrição comportamental de sistemas complexos, pois:

- diagramas de estados são planos, não possuindo recursos para a descrição de hierarquia e, dessa forma, não permitem um desenvolvimento top-down ou bottom-up;
- diagramas de estados não facilitam a especificação de transições que são disparadas pelo mesmo evento, mas que se originam de diferentes estados e essas transições devem ser denotadas de forma explícita em Máquinas de Estados Finitos;
- diagramas de estados são seqüenciais e não representam aspectos de concorrência de forma natural;
- é quase impossível utilizar diagramas de estados à medida que o sistema cresce, pois esse crescimento é exponencial.

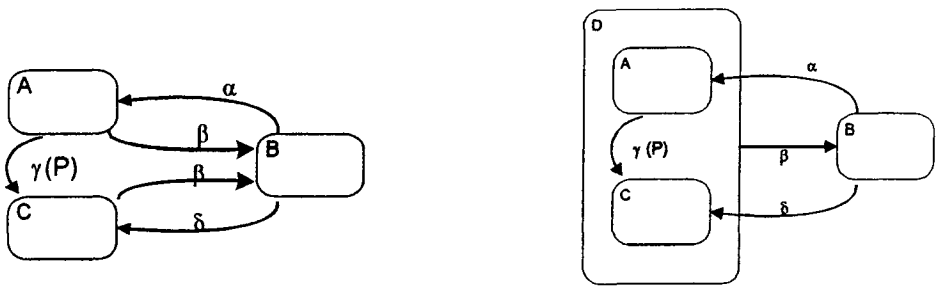
Harel (1987a) propôs a técnica Statecharts, um formalismo visual para a especificação do aspecto comportamental de Sistemas Reativos. Essa técnica é uma extensão das Máquinas de Estados Finitos convencionais e dos diagramas de transição de estado e baseia-se em três aspectos essenciais para a especificação e projeto de sistemas complexos: hierarquia (decomposição), concorrência (ortogonalidade) e comunicação (*broadcasting*). Resumidamente, pode-se dizer que:

Statecharts = diagramas de estados + decomposição + ortogonalidade + broadcasting

Inicialmente, far-se-á uma introdução informal às principais características de Statecharts e posteriormente serão apresentadas sua sintaxe e semântica.

O aspecto de decomposição permite que o sistema seja desenvolvido de forma modularizada, além de facilitar a representação das transições quando estas partem de estados diferentes, mas chegam a um mesmo estado. Essa característica pode ser exemplificada nas Figuras 2.1a e 2.1b (Harel, 1987a). Na Figura 2.1a, há três estados *A*, *B* e *C* e se, por exemplo, o evento γ ocorrer e o sistema se encontrar no estado *A*, o sistema é transferido para o estado *C*, mas somente se a condição P for verdadeira nesse instante. Por outro lado, se o evento β leva o sistema para o estado *B* de *A* ou de *C*, pode-se fazer um "*cluster*" nesses dois estados, criando um novo super-estado *D* e substituindo os dois arcos com o evento β por um único, como na Figura 2.1b. A semântica de *D* é o *exclusive-or* (XOR) de *A* e *C*, ou seja, estando em *D*, ou o estado *A* ou o estado *C* está ativo e nunca

ambos. Assim sendo, *D* é uma abstração de *A* e *C*. O estado *D* e seu arco de saída β capturam uma propriedade comum de *A* e *C*, desde que β os leva para o estado *B*. O fato de permitir que transições saiam de um super-estado, como β na Figura 2.1b, é a principal maneira de Statecharts diminuir a quantidade de arcos da especificação.



(a) (b)

Figura 2.1 - Exemplo de Decomposição de Estados (Harel, 1987a)

É possível especificar de maneira gráfica as partes de um statechart fora de sua vizinhança natural, isto é, apresenta-se um estado sem seus subestados e separadamente, ilustra-se este mesmo estado com seus subestados, como mostra a Figura 2.2. Essa notação de descrição hierárquica de Statecharts tem a vantagem de permitir que a vizinhança de estados mantenha-se pequena, mesmo que as partes envolvidas sejam grandes. Esta característica é importante e necessária quando o sistema que está sendo especificado é grande e complexo.

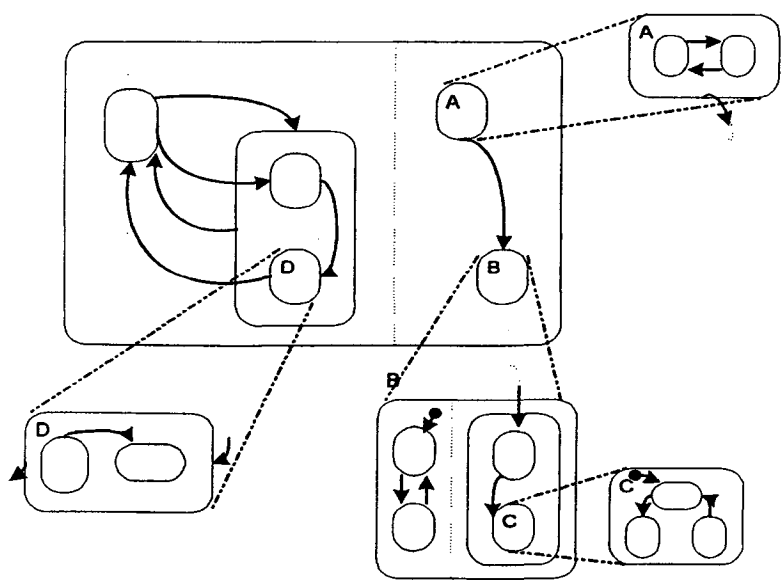


Figura 2.2 - Exemplo de "clusters" em Statecharts (Harel, 1987a)

A ortogonalidade corresponde à especificação do tipo AND de decomposição de estado, sendo que os Statecharts também permitem decomposição do tipo OR. Estados do tipo AND são estados compostos de subestados paralelos que são ativados simultaneamente. Por outro lado, em estados do tipo OR somente um subestado pode estar ativo a cada instante, da mesma forma que nas MEFs. Quando há estados do tipo AND, os aspectos de sincronização entre eles são representados por algumas condições e eventos que são avaliados no momento em que uma transição associada é habilitada. As condições e eventos têm por objetivo avaliar ou confirmar a ocorrência de uma dada situação num componente paralelo que, se confirmada, possibilita o disparo da transição. Na Figura 2.3 (Harel, 1987b), o estado *Y* exemplifica um estado do tipo AND, com os subestados *A*, *D* e *H*. Graficamente, os estados paralelos são separados por linhas tracejadas. Na Figura 2.4, um exemplo dos aspectos de sincronização representados por algumas condições como *in(s)* ou *ny(e)*, e por alguns eventos como *ex(s)* ou *en(s)* pode ser dado pela condição *(in Alarm)* da transição que tem como origem o subestado *Disab* do estado *Alarm_st*, e como destino o estado *Enab*. Essa condição faz com que essa transição seja disparada somente se, além da ocorrência do evento D, no estado paralelo *Main* estiver ativo o estado *Alarm*; caso contrário, essa transição não é disparada.

A concorrência pode ser modelada nos statecharts através de eventos de saída, as ações. Normalmente essas ações afetam o comportamento do próprio statechart nos componentes ortogonais através de um mecanismo de *broadcasting* (reação em cadeia). Assim, a ação é considerada um evento interno que pode, possivelmente, causar o disparo de outras transições em outros componentes.

A Figura 2.3 (Harel, 1987b) mostra um exemplo do mecanismo de *broadcasting*. A configuração de estados de um statechart é o conjunto de estados que estão ativos num determinado instante. Se a configuração atual for *(B,F,J)* e um evento externo *m* ocorrer, a próxima configuração será *(C,G,I)* pela ocorrência de *e* gerada em *H* que dispara as duas transições nos componentes *A* e *D*. Do mesmo modo, se um evento externo *n* ocorrer e a configuração for *(C,G,I)*, a nova configuração será *(B,E,J)*. Adicionalmente, duas restrições aparentemente contraditórias devem ser obedecidas nas reações em cadeia: elas devem ter um tempo de duração zero, porém a ordem das transições dentro das reações em cadeia é muito importante.

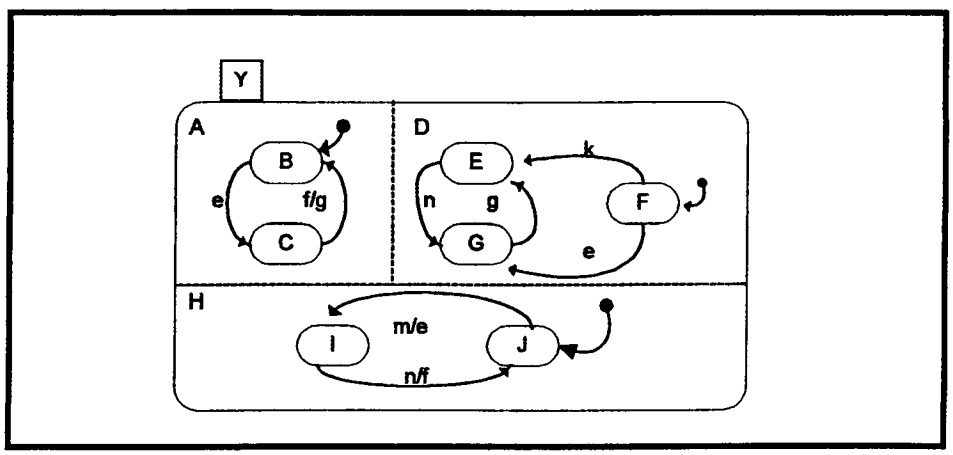


Figura 2.3 - Exemplo do Mecanismo de Broadcasting (Harel, 1987b)

A Figura 2.4, editada no ambiente StatSim, é a especificação de um relógio digital (Pressman, 1997) e será utilizada para ilustrar vários conceitos que serão apresentados. Uma breve descrição desse exemplo pode ser dada da seguinte maneira: o relógio *Watch* possui um visor, uma campainha de dois tons e quatro botões de controle denotados por *A*, *B*, *C* e *D*. Ele pode mostrar a hora (com am/pm ou no modo 24 horas), tem uma campainha que soa a toda hora se estiver habilitada, um cronômetro de 1/100 de segundos, uma luz para iluminação, um indicador da carga da bateria e um dispositivo de teste da campainha. Algumas observações relevantes para o entendimento do exemplo são:

- existe um ciclo que alterna os mostradores e que ocorre ao apertar-se repetidamente o botão *A*;
- os mostradores de hora e data estão relacionados pelo botão *D*;
- o estado que representa os mostradores (*Display*) possui capacidade para atualização do alarme, da data e da hora;
- os modos (subestados) de atualização são alcançados pelo botão *C*; e apertando-se o botão *B* volta-se ao mostrador anterior;
- para os modos de atualização existe uma saída adicional através do botão *C*, que não se aplica ao modo de atualização como um todo.

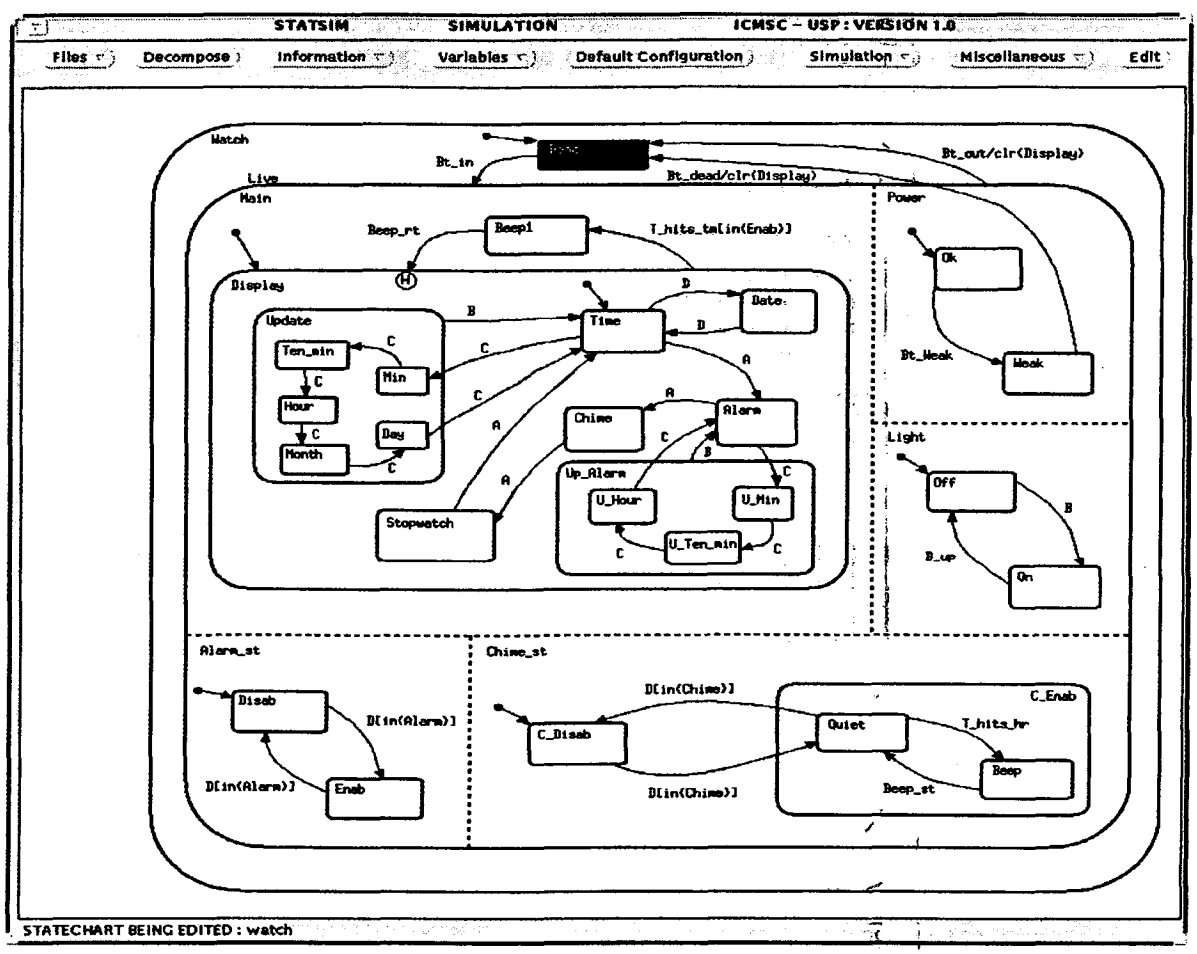


Figura 2.4 - Especificação de um Relógio Digital pela Técnica Statecharts (Pressman, 1997)

A configuração de estados de um statechart não tem tamanho fixo uma vez que está sempre mudando conforme o sistema entra ou sai dos estados ortogonais. Suponha-se que o statechart da Figura 2.4 esteja na configuração inicial de estados: (Dead), isto é, o estado ativo no instante inicial é *Dead*, tendo, portanto, tamanho 1. Se o evento *Bt_in* ocorrer, a nova configuração será (Time, Ok, Off, Disab, C_Disab), estando agora ativos os estados *Time, Ok, Off, Disab, C_Disab*, o que torna o tamanho da configuração igual a 5.

Statecharts possuem um outro aspecto importante que é a capacidade de "lembrar" estados visitados anteriormente, denominado *história*. Quando se ativa um estado, o seu subestado que se torna ativo é, normalmente, o subestado *default*, denotado por uma seta direcionada ao estado, a não ser que um símbolo de *história* (*H*) esteja associado à transição que leva a esse estado. Harel denomina esse caso de "entrada por história". Com isso, ao invés de ativar o subestado *default*, ativa-se o último subestado visitado anteriormente. Na Figura 2.4, o estado *Display* exemplifica essa característica. Se for a primeira vez que este estado é ativado então o subestado ativado é *Time*, que é o subestado *default*. Caso contrário, se *Display* for ativado pela ocorrência do evento *Beep_rt*, o subestado a ser ativado será o último que esteve ativo quando o sistema esteve em *Display* pela última vez.

Um outro símbolo também pode ser utilizado, *H**, indicando que, se um estado é decomposto em vários níveis, o aspecto de *história* é válido para todos os níveis da hierarquia.

Tendo todas essas características que possibilitam um grande poder de representação, a técnica Statecharts oferece mais recursos para apoiar a especificação de sistemas críticos, tendo uma linguagem de especificação formal com sintaxe e semântica bem definidas, o que permite uma avaliação mais rigorosa do sistema por meio da análise dos aspectos dinâmicos do mesmo.

Segundo Harel (1992), a verificação da consistência e completitude de um modelo não previne a ocorrência de erros lógicos. Sendo assim, é necessária a execução do modelo, cujo pré-requisito se resume na disponibilidade de uma semântica formal que contenha informação suficiente para definir com precisão o estado do sistema a cada passo da execução. Dessa forma, como ressaltado anteriormente, Statecharts são validados freqüentemente através das diversas formas de simulação que foram propostas baseadas na necessidade de apoio às atividades de teste de especificações de Sistemas Reativos: interativa, *batch*, programada e exaustiva.

Pode-se citar como uma ferramenta de apoio à utilização da técnica Statecharts a ferramenta comercial Statemate (Harel et al., 1990). Essa ferramenta possibilita a edição, simulação e análise de especificações baseadas em Statecharts. Similar ao Statemate, há o ambiente StatSim (Masiero et al., 1991), desenvolvido no Instituto de Ciências Matemáticas e de Computação da USP/São Carlos e que também permite a edição e simulação de Statecharts. O ambiente StatSim é descrito no Capítulo 3.

A sintaxe e a semântica de Statecharts (Harel, 1987b) são apresentadas nas seções a seguir. Em (Harel & Naamad, 1996), a semântica de Statecharts é revista de acordo com a implementação da ferramenta Statemate (Harel et al., 1990).

2.1 A Sintaxe de Statecharts

A sintaxe de Statecharts é definida sobre conjuntos de elementos básicos: estados, transições, eventos primitivos, condições primitivas e variáveis. Usando esses conjuntos básicos de elementos, definem-se os conjuntos estendidos de: eventos, condições, expressões e rótulos, e as conexões entre eles (Harel, 1987b).

1. **Estados**: o conjunto de estados, S, é definido junto com uma função hierárquica, ρ, uma função tipo, ψ, um conjunto de símbolos de história, H, e uma função *default*, δ.

   A ***Função Hierárquica*** $\rho : S \rightarrow 2^S$ define para cada estado os seus subestados. Tem-se que se $\rho(x) = \rho(y)$ então $x = y$.

   Existe um único estado $r \in S$ tal que $\forall s \in S \quad r \notin \rho(s)$; r é a raiz do statechart.

   ρ^* e ρ^+ são extensões de ρ definidas por: $\rho^*(s) = \cup \rho^i(s)$, $i \geq 0$
   $\rho^+(s) = \cup \rho^i(s)$, $i \geq 1$

   Para o estado *Main* da Figura 2.4, a função hierárquica é definida como:
   $\rho(Main)$ = {Beep1, Display}

   A ***Função Tipo*** $\psi : S \rightarrow \{AND, OR\}$ define, para cada estado, o seu tipo.

   Se $\rho(s) \neq \varnothing$ e $\psi(s) = OR$ então $\rho(s)$ é uma decomposição *XOR* de s. Isso significa que quando o sistema está no estado s, ele está em um e somente um dos subestados de s.

   Se $\rho(s) \neq \varnothing$ e $\psi(s) = AND$ então $\rho(s)$ é uma decomposição *AND* de s. Isso significa que quando o sistema está no estado s, ele está simultaneamente em todos os subestados de s.

   A função tipo do estado *Main* da Figura 2.4 é definida como: $\psi(Main) = OR$. E como $\rho(Main) \neq \varnothing$, então $\rho(Main)$ é uma decomposição *XOR* de *Main*, significando que quando o sistema está no estado *Main* ele está em um e somente um de seus subestados.

   O conjunto de ***Símbolos de História***, H, está relacionado ao conjunto de estados pela função $\gamma : H \rightarrow S$ tal que:

   $\gamma(h1) = \gamma(h2)$ implica $h1 = h2$ e $\gamma(H)$ é um subconjunto do conjunto de estados *OR*, ou seja, somente um estado *OR* pode ter um símbolo de história associado a ele.

   Para o símbolo de história do estado *Display*, da Figura 2.4, tem-se: $\gamma(H)$= {*Display*}

   Define-se $\omega : S \cup H \rightarrow S$ por $\omega(z)$ é: z se $z \in S$, e
   $\gamma(z)$ se $z \in H$

   A ***Função Default*** $\delta : S \rightarrow 2^{S \cup H}$ define para um estado s, um conjunto de estados e símbolos de história que estão contidos no estado.

   Se $x \in \delta(s)$ então: para $x \in S$, $x \in \rho^+(s)$, e
   para $x \in H$, $\gamma(x) \in \rho^*(s)$

   $\delta(s)$ é o conjunto *default* para s

   Na Figura 2.4, a função *default* para o estado *Main* é definida como:
   $\delta(Main)$ = {Display}

2. Expressões: o conjunto de variáveis é denotado por V_p. O conjunto de expressões, V, é definido indutivamente como segue:

1. Se k é um número então $k \in V$
2. Se $v \in V_p$ então $v \in V$
3. Se $v \in V$ então *current (v)* $\in V$
4. Se $v_1, v_2 \in V$ e *op* é uma operação algébrica então $op(v_1, v_2) \in V$

Obs: *current (v)* é abreviado por *cr(v)*

3. Condições: o conjunto de condições primitivas é denotado por C_p. O conjunto de condições, C, é definido indutivamente como segue:

1. $T, F \in C$; T, F correspondem a *true* e *false*, respectivamente
2. Se $c \in C_p$ então $c \in C$
3. Se $s \in S$ então *in(s)* $\in C$
4. Se $e \in E$ então *not_yet(e)* $\in C$
5. Se $u, v \in V$, $R \in \{=, >, <, \neq, \geq, \leq\}$ então $u R v \in C$
6. Se $c \in C$ então *current(c)* $\in C$
7. Se $c_1, c_2 \in C$ então $c_1 \vee c_2, c_1 \wedge c_2, \neg c_1 \in C$

Obs: *not_yet(e)* é abreviado por *ny(e)*
current (v) é abreviado por *cr(v)*

4. Eventos: o conjunto de eventos primitivos é denotado por E_p. O conjunto de eventos, E, é definido indutivamente como segue:

1. $\lambda \in E$; λ é o *evento nulo*
2. Se $e \in E_p$ então $e \in E$
3. Se $c \in C$ então *true(c), false(c)* $\in E$
4. Se $v \in V$ então *changed(v)* $\in E$
5. Se $s \in S$ então *exit(s), entered(s)* $\in E$
6. Se $e_1, e_2 \in E$ então $e_1 \vee e_2, e_1 \wedge e_2 \in E$
7. Se $e \in E, c \in C$ então $e[c] \in E$

Obs: *true(c), false(c)* são abreviados por *tr(c), fs(c)*, respectivamente;
changed(v) é abreviado por *ch(v)*;
exit(s), entered(s) são abreviados por *ex(s), en(s)*, respectivamente;
e é ***atômico*** se e é da forma 1-5.

5. Ações: o conjunto de ações, A, é definido indutivamente como segue:

1. $\mu \in A$, μ é a *ação nula*
2. Se $c \in C_p, d \in C$ então $c := d \in A$
3. Se $v \in V_p, u \in V$ então $v := u \in A$
4. Se $a_i \in A$, i = 0, ..., n então $a_0 ; \ldots ; a_n \in A$

$a \in A$ é ***atômica*** se ela é da forma 1-3.

6. **Rótulos**: o conjunto de *rótulos*, L, é o conjunto de pares $E \times A$, e para $l = (e, a)$, $e \in E$, $a \in A$, escreve-se e/a. Informalmente, se e/a é o rótulo de uma transição t, então t é ativada por e e a é executada quando t é disparada.

7. **Transições**: o conjunto de *transições*, T, é definido como o conjunto de triplas $T \subset 2^S \times L \times 2^{S \cup H}$. A transição $t = (X, l, Y)$ é composta de um conjunto *origem* X, um conjunto *destino* Y, denotados por *source(t)* e *target(t)*, respectivamente, e um rótulo l. Informalmente, se $l = e/a$, o sistema está em X e e ocorre, então t está habilitada e pode ser disparada. Se t é disparada, a é executada e o sistema vai então para Y.

2.2 A Semântica de Statecharts

Inicialmente, apresentam-se os termos e conceitos básicos usados na definição da semântica de Statecharts (Harel, 1987b). Alguns desses termos e conceitos são ilustrados com o statechart da Figura 2.4.

a) Um estado s é *básico* se $\rho(s) = \varnothing$.

Os estados básicos do statechart da Figura 2.4 são: {*Dead, Off, On, Ok, Weak, Disab, Enab, C_disab, Quiet, Beep1, Ten_min, Min, Hour, Month, Day, Alarm, Chime, Up_Min, Up_Ten_min, Up_Hour, Date, Stopwatch, Time, Beep* }

b) Para um conjunto de estados X, o *Menor Ancestral Comum de X*, denotado por *LCA(X)* é o estado x definido como segue: $LCA(X) = x$ se e somente se:

1. $X \subseteq \rho^*(x)$
2. $\forall s \in S,\ X \subseteq \rho^*(s) \Rightarrow x \in \rho^*(s)$

Exemplificando, para o conjunto de estados S = {*Date, Update, Beep1, Up_Hour*} da Figura 2.4, tem-se que o *Menor Ancestral Comum* de S é LCA(S) = {*Main*}

Para um conjunto de estados X, o *Menor Ancestral Estrito Comum OR de X*, denotado por $LCA^+(X)$ é o estado x definido como segue: $LCA^+(X) = x$ se e somente se:

1. $X \subseteq \rho^+(x)$
2. $\psi(x) = OR$
3. $\forall s \in S$ se $\psi(s) = OR$ então $X \subseteq \rho^*(s) \Rightarrow x \in \rho^*(s)$

c) Para uma transição $t \in T$, *LCA(t)* é definido como LCA^+ de seus conjuntos de estados origem e destino, definidos como: se $t = (X, l, Y)$ então $LCA(t) = LCA^+(X \cup \omega(Y))$.

d) Dois estados x,y são *ortogonais*, denotados por $x \perp y$, se $x=y$ ou se *LCA* de ambos é um estado AND, ou seja, $\psi(LCA(\{x,y\})) = AND$.

e) Um conjunto de estados X, é um *conjunto ortogonal* se $\forall x,y \in X$ então $x \perp y$. Observa-se que $\{x\}$ é um conjunto ortogonal para todo $x \in S$.

O conjunto de estados {*Main, Chime_st, Alarm_st, Light, Power*} da Figura 2.4 é um exemplo de conjunto ortogonal, pois todos os estados que pertencem a este conjunto são ortogonais entre si, ou seja, LCA desse conjunto é um estado AND, no caso, *Live*.

f) Um conjunto de estados X é um *conjunto ortogonal relativo a s* se:

1. $X \subset \rho^*(s)$
2. X é um conjunto ortogonal

O conjunto de estados {*Main, Chime_st, Alarm_st, Light, Power*} da Figura 2.4 é um conjunto ortogonal relativo ao estado *Watch*, pois é um conjunto ortogonal e pertence a $\rho^*(Watch)$.

g) Um conjunto X é um *conjunto ortogonal maximal relativo a s* se:

1. X é um conjunto ortogonal relativo a s
2. $\forall y \in \rho^*(s), y \notin X \Rightarrow X \cup \{y\}$ não é ortogonal

O conjunto de estados {*Main, Chime_st, Alarm_st, Light, Power*} da Figura 2.4 é um exemplo de conjunto ortogonal maximal relativo ao estado *Watch*, pois é um conjunto ortogonal relativo ao estado *Watch* e {*Main, Chime_st, Alarm_st, Light, Power*}$\cup$ {*Dead*} não é ortogonal.

h) Uma *configuração de estados de s* é um conjunto ortogonal relativo a s cujos membros são estados básicos.

i) Uma *configuração de estados maximal de s* é um conjunto ortogonal maximal relativo a s cujos membros são estados básicos.

j) Dada uma seqüência de configurações de estados maximal relativa à raiz, $(X_0,...,X_n)$, e um símbolo de história h com $\gamma(h)=s$, a função história $\tau(h, (X_0,...,X_n))$ define o subestado de s que foi o último visitado do sistema em s de acordo com $(X_0,...,X_n)$. Formalmente, $I=\{i/\rho^*(s)\cap X_i \neq \varnothing\}$. Se $I=\varnothing$ então $\tau(h, (X_0,...,X_n))= \delta(s)$. Considere por outro lado que j seja o número máximo em I. Então, desde que s é um estado OR, existe um único $s' \in \rho(s)$ tal que $\rho^*(s') \cap X_j \neq \varnothing$, define-se $\tau(h, (X_0,...,X_n)) = s'$.

k) O conjunto destino de uma transição pode conter estados não-básicos e símbolos de história. Cada passo do sistema é definido por um conjunto de transições que deve definir uma configuração de estados maximal do estado raiz. Assim, deve-se definir uma configuração de estados correspondente ao conjunto destino de uma transição. A função $C(s, X, (X_0,...,X_n))$, em que: X é um conjunto ortogonal relativo a s e $(X_0,...,X_n)$ é uma seqüência de configurações de estados maximal do estado raiz, é definida recursivamente. $C(s, X, (X_0,...,X_n))$ é uma configuração de estados maximal de s calculada aplicando-se repetidamente as funções *default* e história, e depois completando-se o conjunto ortogonal de forma a transformá-lo em um conjunto ortogonal maximal considerando os estados *default* nos componentes ortogonais.

l) A *configuração inicial de estados* X_0 é definida como $C(raiz,\{raiz\},\varnothing)$. A configuração inicial de estados é a configuração de estados maximal do estado raiz.

m) Um conjunto de transições Υ é *estruturalmente consistente* se $\forall\ t_1,t_2 \in \Upsilon$, $LCA\ (t_1) \perp LCA(t_2)$. Um conjunto consistente de transições pode disparar simultaneamente.

n) Uma transição $t = (Y,l,Z)$ é *estruturalmente relevante* para uma configuração de estados X se para todo $y \in Y$ há $x \in X$ tal que $x \in \rho^*(y)$. Se t é estruturalmente relevante a X e o sistema está em X então t pode ser disparada.

A semântica de Statecharts é baseada em uma seqüência de instantes de tempo $\{\sigma_i\}_{i \geq 0}$, correspondendo à taxa de execução do sistema. Os intervalos de tempo básicos são definidos por $I_i = (\sigma_i\ ,\ \sigma_{i+1})$. Em σ_{i+1}, o sistema reage a estímulos externos ocorridos no intervalo I_i. A semântica de Statecharts é dada por uma definição formal das mudanças que ocorrem no sistema como uma reação aos estímulos externos. Um estímulo externo associado a σ_{i+1} é uma tripla (Π,θ,ξ), onde: Π é um conjunto de eventos primitivos externos que ocorrem em I_i, θ é um conjunto de condições primitivas externas cujo valor é verdadeiro (*true*) em $(\sigma_i\ ,\ \sigma_{i+1})$ para algum $\sigma \geq \sigma_i$, e ξ é uma função determinada pelo ambiente externo tal que para uma variável v, $\xi(v) = x$ se o valor de v for x em $(\sigma_i\ ,\ \sigma_{i+1})$ para algum $\sigma \geq \sigma_i$. Uma *configuração de sistema* associada ao instante σ_{i+1} é uma tupla (X,Π,θ,ξ) onde X é a *configuração de estados maximal* do estado raiz e (Π,θ,ξ) é um estímulo externo associado a σ_{i+1}.

A reação do sistema em um instante é composta pelo conjunto de transições disparadas naquele instante e pelo conjunto de eventos gerados quando essas transições são disparadas. Assim, uma *reação do sistema* é um par (Υ,Π^*), onde Υ é um conjunto de transições denominado *passo* e Π^* é um conjunto de eventos atômicos gerados por Υ. Dada uma configuração de sistema $SC=(X,\Pi,\theta,\xi)$, pode-se definir o conjunto $\{SR=(\Upsilon,\Pi^*)\}$ das possíveis reações do sistema para SC.

Intuitivamente, um *passo* é um conjunto de transições consistentes que são estruturalmente relevantes para uma dada configuração de sistema e habilitadas por um dado estímulo externo. O conjunto de eventos gerados é o conjunto de eventos que ocorrem como um resultado do disparo das transições de um *passo* e da execução de ações associadas a essas transições. Todas as transições que participam de um *passo* Υ são disparadas simultaneamente. Um *passo* pode também ser definido como uma seqüência de micro-passos, sendo que cada micro-passo é um subconjunto de Υ. Embora os micro-passos sejam essenciais para a definição precisa e o entendimento da semântica, eles são considerados como mecanismos internos para o cálculo dos passos e devem ser ocultos dos usuários.

Definição: Dada uma configuração de sistema $SC=(X,\Pi,\theta,\xi)$, define-se uma extensão de ξ para V, e pode-se dizer que v é *avaliado para x numa determinada SC*, se $\xi(v)=x$. ξ é estendido para V como:

1. Se k é um número então $\xi(k)=k$
2. Se $v_1,v_2 \in V$ então $\xi(op(v_1,v_2)) = op(\xi(v_1), \xi(v_2))$
3. Se $v \in V$ então $\xi(cr(v))= \xi(v)$

Definição: Dada uma configuração de sistema $SC=(X,\Pi,\theta,\xi)$, define-se *SC satisfaz* $c \in C$, denotada por $SC \rightarrow c$, como:

1. $SC \rightarrow T$, $not(SC \rightarrow F)$
2. Se $c \in C_p$ então $SC \rightarrow c$ se e somente se $c \in \theta$
3. Se $s \in S$ então $SC \rightarrow in(s)$ se e somente se $X \cap \rho^*(s) \neq \varnothing$
4. Se $e \in E$ então $SC \rightarrow ny(e)$ se e somente se not $(SC \rightarrow e)$
5. Se $u,v \in V$, $R \in \{=,>,<,\neq,\geq,\leq\}$ então $SC \rightarrow u\ R\ v$ se e somente se $\xi(u)\ R\ \xi(v)$
6. Se $c \in C$ então $SC \rightarrow cr(c)$ se e somente se $SC \rightarrow c$
7. Se $c_1,c_2 \in C$ então:
   7.1. $SC \rightarrow c_1 \wedge c_2$ se e somente se $SC \rightarrow c_1$ e $SC \rightarrow c_2$
   7.2. $SC \rightarrow c_1 \vee c_2$ se e somente se $SC \rightarrow c_1$ ou $SC \rightarrow c_2$
   7.3. $SC \rightarrow \neg c_1$ se e somente se not$(SC \rightarrow c_1)$

Definição: Dada uma configuração de sistema $SC=(X,\Pi,\theta,\xi)$, define-se $e \in E$ *ocorre em SC*, denotado por $SC \rightarrow e$:

1. $SC \rightarrow \lambda$
2. Se $e \in E_p$ então $SC \rightarrow e$ se e somente se $e \in \Pi$
3. Se $c \in C$ então $not(SC \rightarrow tr(c))$, $not(SC \rightarrow fs(c))$
4. Se $v \in V$ então $not(SC \rightarrow ch(v))$
5. Se $s \in S$ então $not(SC \rightarrow ex(s))$, $not(SC \rightarrow en(s))$
6. Se $e_1,e_2 \in E$ então:
   6.1. $SC \rightarrow e_1 \wedge e_2$ se e somente se $SC \rightarrow e_1$ e $SC \rightarrow e_2$
   6.2. $SC \rightarrow e_1 \vee e_2$ se e somente se $SC \rightarrow e_1$ ou $SC \rightarrow e_2$
7. Se $e \in E$, $c \in C$ então $SC \rightarrow e[c]$ se e somente se $SC \rightarrow e$ e $SC \rightarrow c$

Dada uma configuração de sistema *SC*, a reação do sistema é composta por uma seqüência de micro-passos. O primeiro micro-passo é definido como um conjunto de transições cujo disparo ocorre em *SC*. Esse primeiro micro-passo resulta em uma *micro configuração de sistema* para a qual o sistema pode reagir por um outro micro-passo e assim por diante. Uma configuração de sistema é uma descrição completa do *status* do sistema em um instante de tempo. A micro configuração de sistema é um termo abstrato, pois não descreve um *status* atual do sistema. Define-se a seguir o termo *micro configuração de sistema* e os termos *satisfaz* e *ocorre* para as condições e eventos, em relação à micro configuração do sistema.

Definição: Uma *micro configuração do sistema* μSC, em relação a uma configuração do sistema $SC=(X,\Pi,\theta,\xi)$, é uma quíntupla $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$, em que:

1. μX é uma configuração de estados parcial, $\mu X \subseteq X$
2. $\mu\Pi \subset E_p \cup \{ ex(s), en(s) / s \in S\}$, $\Pi \subseteq \mu\Pi$
3. $\mu\theta \subseteq \{cr(c)/ c \in C_p\}$
4. $\mu\xi$ é uma função que atribui valores às variáveis atuais, ou seja, $\mu\xi$ é uma função de $\{cr(v)| v \in V_p\}$ para o conjunto de números
5. μY é uma configuração de estados parcial

Definição: Dada $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$ em relação a uma configuração do sistema $SC=(X,\Pi,\theta,\xi)$, define-se uma extensão de $\mu\xi$ para V, e diz-se que v é *avaliada como* x *numa determinada* μSC se $\mu\xi(v) = x$. $\mu\xi$ é estendido para V:

1. Se k é um número então $\mu\xi(k) = \mu\xi(cr(k))=k$
2. Se $v \in V_p$ então $\mu\xi(v) = \xi(v)$
3. Se $v_1,v_2 \in V$ então $\mu\xi(op(v_1,v_2)) = op(\mu\xi(v_1), \mu\xi(v_2))$
4. Se $v_1,v_2 \in V$ então $\mu\xi(cr(op(v_1,v_2))) = op(\mu\xi(cr(v_1)), \mu\xi(cr\ (v_2)))$

Note que para qualquer expressão v que não use cr, $\mu\xi(v) = \xi(v)$.

Definição: Dada $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$ em relação a uma configuração do sistema $SC=(X,\Pi,\theta,\xi)$, define-se μSC *satisfaz* $c \in C$, denotado por $\mu SC \rightarrow c$, como:

1. $\mu SC \rightarrow T$, $not(\mu SC \rightarrow F)$
2. Se $c \in C_p$ então:
   2.1. $\mu SC \rightarrow c$ se e somente se $SC \rightarrow c$
   2.2. $\mu SC \rightarrow cr(c)$ se e somente se $cr(c) \in \mu\theta$
3. Se $s \in S$ então:
   3.1. $\mu SC \rightarrow in(s)$ se e somente se $SC \rightarrow in(s)$
   3.2. $\mu SC \rightarrow cr(in(s))$ se e somente se $\mu X \cap \rho^*(s) \neq \varnothing$ ou $\mu Y \cap \rho^*(s) \neq \varnothing$
4. Se $e \in E$ então:
   4.1. $\mu SC \rightarrow ny(e)$ se e somente se $not(\mu SC \rightarrow e)$
   4.2. $\mu SC \rightarrow cr(ny(e))$ se e somente se $\mu SC \rightarrow ny(e)$
5. Se $u,v \in$ V, $R \in \{=,>,<,\neq,\geq,\leq\}$ então:
   5.1. $\mu SC \rightarrow u\ R\ v$ se e somente se $\mu\xi(u)\ R\ \mu\xi(v)$
   5.2. $\mu SC \rightarrow cr(u\ R\ v)$ se e somente se $\mu SC \rightarrow cr(u)\ R\ cr(v)$
6. Se $c_1,c_2 \in C$ então:
   6.1. para $c_1 \wedge c_2 \in C$:
      6.1.1. $\mu SC \rightarrow c_1 \wedge c_2$ se e somente se $\mu SC \rightarrow c_1$ e $\mu SC \rightarrow c_2$
      6.1.2. $\mu SC \rightarrow cr(c_1 \wedge c_2)$ se e somente se $\mu SC \rightarrow cr(c_1)$ e $\mu SC \rightarrow cr(c_2)$
   6.2. para $c_1 \vee c_2 \in C$:
      6.2.1. $\mu SC \rightarrow c_1 \vee c_2$ se e somente se $\mu SC \rightarrow c_1$ ou $\mu SC \rightarrow c_2$
      6.2.2. $\mu SC \rightarrow cr(c_1 \vee c_2)$ se e somente se $\mu SC \rightarrow cr(c_1)$ ou $\mu SC \rightarrow cr(c_2)$
   6.3. para $\neg c_1 \in C$:
      6.3.1. $\mu SC \rightarrow \neg c_1$ se e somente se $not(\mu SC \rightarrow c_1)$
      6.3.2. $\mu SC \rightarrow cr(\neg c_1)$ se e somente se $not(\mu SC \rightarrow cr(c_1))$

Note que para qualquer condição c que não use cr ou ny, $\mu SC \rightarrow c$ se e somente se $SC \rightarrow c$.

Definição: Dadas duas micro configurações de sistema $\mu SC_1=(\mu X_1, \mu\Pi_1, \mu\theta_1, \mu\xi_1, \mu Y_1)$, $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$, ambas em relação a uma configuração do sistema $SC=(X,\Pi,\theta,\xi)$, μSC é uma *possível sucessora* de μSC_1 se:

1. $\mu X \subseteq \mu X_1$
2. $\mu\Pi_1 \subseteq \mu\Pi$
3. $\mu Y_1 \subseteq \mu Y$

Definição: Dada $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$ que é uma possível sucessora de $\mu SC_1=(\mu X_1, \mu\Pi_1, \mu\theta_1, \mu\xi_1, \mu Y_1)$, ambas em relação a uma configuração de sistema $SC=(X,\Pi,\theta,\xi)$, define-se $e \in E$ *ocorre em* μSC, denotado por $\mu SC \rightarrow e$, como:

1. $\mu SC \rightarrow \lambda$
2. Se $e \in E_p$ então $\mu SC \rightarrow e$ se e somente se $e \in \mu\Pi$
3. Se $c \in C$ então
   3.1. $\mu SC \rightarrow tr(c)$ se e somente se $\mu SC_1 \rightarrow tr(c)$ ou $\mu SC_1 \rightarrow cr(\neg c)$ e $\mu SC \rightarrow cr(c)$
   3.2. $\mu SC \rightarrow fs(c)$ se e somente se $\mu SC_1 \rightarrow fs(c)$ ou $\mu SC_1 \rightarrow cr(c)$ e $\mu SC \rightarrow cr(\neg c)$
4. Se $v \in V$ então $\mu SC \rightarrow ch(v)$ se e somente se $\mu SC_1 \rightarrow ch(v)$ ou $\mu\xi_1(cr(v)) \neq \mu\xi(cr(v))$
5. Se $s \in S$ então:
   5.1. $SC \rightarrow ex(s)$ se e somente se $\mu SC_1 \rightarrow ex(s)$ ou $ex(s) \in \mu\Pi$
   5.2. $SC \rightarrow en(s)$ se e somente se $\mu SC_1 \rightarrow en(s)$ ou $en(s) \in \mu\Pi$
6. Se $e_1, e_2 \in E$ então:
   6.1. $\mu SC \rightarrow e_1 \wedge e_2$ se e somente se $\mu SC \rightarrow e_1$ e $\mu SC \rightarrow e_2$
   6.2. $\mu SC \rightarrow e_1 \vee e_2$ se e somente se $\mu SC \rightarrow e_1$ ou $\mu SC \rightarrow e_2$
7. Se $e \in E$, $c \in C$ então $\mu SC \rightarrow e[c]$ se e somente se $\mu SC \rightarrow e$ e $\mu SC \rightarrow c$

Note que, uma vez que $\lambda[c] \equiv c$, tem-se que $C \subseteq E$.

Lema: Para qualquer $e \in E$, se $\mu SC_1 \rightarrow e$ então $\mu SC \rightarrow e$

Informalmente, um micro-passo é um conjunto de transições cujos disparos ocorrem em uma dada micro configuração do sistema e que podem ser disparadas simultaneamente sem causar qualquer tipo de conflito estrutural ou nos valores atribuídos às variáveis e condições. A seguir, define-se micro-passo:

Lema: Se $SC=(X,\Pi,\theta,\xi)$ é uma configuração do sistema, então $\mu SC=(X, \Pi, \theta, \xi, \varnothing)$ é uma micro configuração do sistema em relação à SC. Além disso, μSC é uma possível sucessora dela mesma.

Definição: Dada uma micro configuração do sistema $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$, duas ações atômicas a, b são *inconsistentes* numa determinada μSC se:

ou $a \equiv c := d_1$ e $b \equiv c := d_2$ para algum $c \in C_p$

ou $a \equiv v := u_1$ e $b \equiv v := u_2$ para algum $v \in V_p$

Duas ações atômicas a, b são *consistentes* numa determinada μSC se elas não forem inconsistentes.

Uma ação $a = a_0;\ldots; a_n$ é *consistente* numa determinada μSC se $\forall\ i, j$ se $i \neq j$ então a_i, a_j são consistentes numa determinada μSC.

Note que uma ação atômica é sempre consistente.

Definição: Dado um conjunto de transições $\mu\Upsilon = \{t_0, ..., t_n\}$, onde $t_i = (X_i,(e_i, a_i),Y_i)$, $a = a_0;...; a_n$ é a ação *causada* por $\mu\Upsilon$.

Definição: Dada uma micro configuração do sistema μSC, um conjunto de transições $\mu\Upsilon$ é *consistente* numa determinada μSC se:

1. $\mu\Upsilon$ é estruturalmente consistente
2. a ação causada por $\mu\Upsilon$ é consistente numa determinada μSC

Definição: Dadas duas micro configurações do sistema μSC_1 e $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$, ambas em relação à SC, onde μSC é uma possível sucessora de μSC_1, e uma transição t rotulada por $l = e/a$, t é *habilitada* numa determinada μSC se:

1. t é estruturalmente relevante para μSC
2. $\mu SC \rightarrow e$

Definição: Dados duas micro configurações de sistema μSC_1 e $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$, ambas em relação à SC, onde μSC é uma possível sucessora de μSC_1, e um conjunto não vazio de transições $\mu\Upsilon$, $\mu\Upsilon$ é um micro-passo a partir de μSC se:

1. $\mu\Upsilon$ é consistente numa determinada μSC
2. $t \in \mu\Upsilon$, t é habilitada numa determinada μSC

Um micro-passo causa mudanças nos valores atuais das condições e variáveis e a ocorrência de novos eventos. Essas mudanças são formalmente definidas como:

Definição: Dados duas micro configurações do sistema μSC_1 e $\mu SC=(\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$, ambas em relação à SC, onde μSC é uma possível sucessora de μSC_1, e um micro-passo $\mu\Upsilon$ de μSC, onde $a = a_0;...; a_n$ é a ação causada por $\mu\Upsilon$, com a_i sendo ações atômicas:

1. $\mu\Upsilon$ *gera* $e \in E_p$ se e somente se $a_i = e$ para algum i
2. $\mu\Upsilon$ *gera* $ex(s)$ se e somente se $\rho^*(s) \cap \mu X \neq \varnothing$ e existe $t \in \mu\Upsilon$ tal que $s \in \rho^+(LCA(t))$
3. Dada uma seqüência de configurações de estados $\mathbf{X} = (X_0, ..., X_n)$ onde $X_n = X$, $\mu\Upsilon$ *gera* $en(s)$ se e somente se existe $t \in \mu\Upsilon$ tal que $\rho^*(s) \cap C(LCA(t), target(t), \mathbf{X}) \neq \varnothing$ e $s \in \rho^+(LCA(t))$
4. $\mu\Upsilon$ *atribui* T para $c \in C_p$ se e somente se $a_i \equiv c := d$ e $\mu SC \rightarrow d$ para algum i
5. $\mu\Upsilon$ *atribui* F para $c \in C_p$ se e somente se $a_i \equiv c := d$ e $\mu SC \rightarrow \neg d$ para algum i
6. $\mu\Upsilon$ *atribui* x para $v \in V_p$ se e somente se $a_i \equiv v := u$ e $\mu\xi(u) = x$ para algum i

Lema: Se $\mu\Upsilon$ é um micro-passo a partir de μSC, então:

1. Se $\mu\Upsilon$ atribui T (F) para $c \in C_p$, então não atribui F (T) para c
2. Se $x \neq y$ e $\mu\Upsilon$ *atribui* x para $v \in V_p$, então não atribui y para v

Um micro-passo também resulta em uma nova micro configuração do sistema:

Definição: Sejam X = (X_0, ..., X_n) uma seqüência de configurações de estados, SC uma configuração de sistema cuja configuração de estados é X_n, μSC_1 uma micro configuração de sistema em relação à SC e $\mu\Upsilon$ um micro-passo a partir de μSC_1. Então μSC=(μX, $\mu\Pi$, $\mu\theta$, $\mu\xi$, μY) é a micro configuração de sistema *alcançada* por $\mu\Upsilon$ a partir de μSC_1, se:

1. $\mu X = \mu X_1 - \mu X_1 \cap \{\cup \rho^*(LCA(t))| \, t \in \mu\Upsilon\}$
2. $\mu\Pi = \mu\Pi_1 \cup \{e| \, \mu\Upsilon \text{ gera } e\}$
3. $\mu\theta = \mu\theta_1 \cap \{cr(c)| \, \mu\Upsilon \text{ não atribui } F \text{ para } c\} \cup \{cr(c)| \, \mu\Upsilon \text{ atribui } T \text{ para } c\}$
4. $\mu\xi(v) = x$ se e somente se $\mu\xi_1(v) = s$ e $\mu\Upsilon$ não atribui nenhum valor para v ou $\mu\Upsilon$ atribui x para v
5. $\mu\Upsilon = \mu\Upsilon_1 \cup \{\cup C(LCA(t), target(t), \mathbf{X})| \, t \in \mu\Upsilon\}$

Lema: Se μSC é alcançada por $\mu\Upsilon$ a partir de μSC_1, então:

1. para $e \in E_p \cup \{ex(s), en(s)| \, s \in S\}$, $\mu SC \rightarrow e$ se e somente se $\mu SC_1 \rightarrow e$ ou $\mu\Upsilon$ gera e
2. para $c \in C_p$
   2.1. $\mu SC \rightarrow cr(c)$ se e somente se $\mu SC_1 \rightarrow cr(c)$ e $\mu\Upsilon$ não atribui F a c ou $\mu SC_1 \rightarrow cr(\neg c)$ e $\mu\Upsilon$ atribui T a c
   2.2. $\mu SC \rightarrow tr(c)(fs(c))$ se e somente se $\mu SC_1 \rightarrow tr(c)(fs(c))$ ou $\mu SC_1 \rightarrow cr(\neg c)(cr(c))$ e $\mu\Upsilon$ atribui $T(F)$ a c
3. para $v \in V_p$
   3.1. $\mu\xi(cr(v)) = x$ se e somente se $\mu\xi_1(cr(v)) = x$ e $\mu\Upsilon$ não atribui nenhum valor a v ou $\mu\Upsilon$ atribui x a v
   3.2. $\mu SC \rightarrow ch(v)$ se e somente se $\mu SC_1 \rightarrow ch(v)$ ou $\mu\Upsilon$ atribui x a v e $\mu\xi_1(cr(v)) \neq x$

Um passo é uma seqüência maximal de micro-passos formalmente definido como:

Definição: Seja SC uma configuração do sistema, um *passo* Υ a partir de SC é uma seqüência ($\mu\Upsilon_0$, ..., $\mu\Upsilon_m$), em que:

1. $\mu SC_0 = SC$
2. $\mu\Upsilon_i$ é um micro-passo a partir de μSC_i; para i = 0, ..., m
3. μSC_{i+1} é a micro configuração do sistema alcançada por $\mu\Upsilon_i$ para i = 0, ..., m
4. o conjunto $\mu\Upsilon_0 \cup ... \cup \mu\Upsilon_n$ é um conjunto estruturalmente consistente de transições
5. Se t está habilitada numa determinada μSC_{m+1}, então $\{t\} \cup \mu\Upsilon_0 \cup ... \cup \mu\Upsilon_n$ não é estruturalmente consistente, ou seja, a seqüência é maximal

Um passo resulta em uma nova configuração do sistema, que é similar à última micro configuração do sistema alcançada pela seqüência de micro-passos que compõem o passo. Desde que não haja transições a partir desta última micro

configuração do sistema, o sistema "espera" durante um intervalo de tempo por um novo estímulo externo. Um passo também causa a ocorrência de alguns eventos. Esses eventos juntamente com os novos valores atribuídos às condições e variáveis são a saída do sistema para o ambiente externo.

Definição: Seja *SC'* uma configuração do sistema e $\Upsilon = (\mu\Upsilon_0, ..., \mu\Upsilon_m)$ um passo a partir de *SC'* realizado no instante de tempo σ_i . $SC=(X, \Pi, \theta, \xi)$ é a configuração do sistema *alcançada* por Υ se:

1. $X = \mu X_{m+1} \cup \mu Y_{m+1}$
2. Π é o conjunto de eventos primitivos que ocorrem no intervalo de tempo i
3. Para $c \in C_p$, c se mantém em σ_i se e somente se $\mu SC_{m+1} \rightarrow cr(c)$
   θ é o conjunto de condições primitivas que se mantém em (σ,σ_{i+1}), para algum $\sigma \geq \sigma_i$
4. Para $v \in V_p$, o valor de v em σ_i é x se e somente se $\mu\xi_{m+1}(cr(v)) = x$
   $\xi(v) = x$ se o valor de v em (σ,σ_{i+1}) é x, para algum $\sigma \geq \sigma_i$

Definição: Seja *SC* uma configuração do sistema no instante i e $\Upsilon = (\mu\Upsilon_0, ..., \mu\Upsilon_m)$ um passo a partir de *SC* realizado em σ_i. O conjunto de *eventos gerados* por Υ é $\Pi^* = \{e|$ *e* gerado por $\mu\Upsilon_i$ para algum i$\} \cup \{e|\ \mu SC_{m+1} \rightarrow e$, *e* atômico $\}$.

Definição: Uma execução do sistema é uma seqüência $\{(SC_i, \Upsilon_i, \Pi_i^*)\}_{i\geq 0}$, em que:

1. $SC_0=(X_0, \Pi_0, \theta_0, \xi_0)$, X_0 é a configuração inicial de estados, e (Π_0, θ_0, ξ_0) são os estímulos externos que ocorrem no primeiro intervalo de tempo I_0
2. Υ_i é um passo realizado a partir de SC_i no instante de tempo σ_{i+1} , para $i\geq 0$
3. SC_{i+1} é a configuração de sistema alcançada a partir de SC_i por Υ_i, para $i\geq 0$
4. Π_i^* é o conjunto de eventos gerados por Υ_i, para $i\geq 0$

Definição: Seja *SC* uma configuração do sistema. O sistema é *não-determinístico* em *SC* se existem duas reações diferentes (Υ_1, Π_1^*) e (Υ_2, Π_2^*) tal que $\Pi_1^* \neq \Pi_2^*$ ou $\Upsilon_1 \neq \Upsilon_2$.

2.3 Teste e Validação de Statecharts

Como ressaltado anteriormente, o poder de descrição de Statecharts dificulta a validação dos modelos especificados nessa técnica em relação ao comportamento desejado, sendo então necessária a execução do modelo. Dessa forma, Statecharts são validados freqüentemente através das diversas formas de simulação propostas: *simulação interativa*, que com o apoio de uma ferramenta automatizada, possibilita que o usuário represente o meio ambiente do sistema gerando, passo a passo, eventos que provocam alteração no mesmo; *simulação em batch*, em que a simulação é realizada de modo iterativo e não mais interativo, com base em um conjunto de eventos pré-determinados; *simulação programada*, que permite analisar o modelo sob condições geradas randomicamente, sendo que podem ser definidos pontos de parada de modo que a ferramenta execute determinadas ações quando ocorrem situações específicas; *simulação exaustiva*, que teoricamente

consiste em executar o modelo gerando-se todos os possíveis eventos externos e todas as alterações nos valores das condições e variáveis.

A maioria dos ambientes para especificação de Sistemas Reativos contém ferramentas para a execução interativa de modelos, tanto na forma textual quanto na forma gráfica. A execução interativa é útil para a validação e verificação de modelos pois permite que o usuário interaja diretamente com o sistema e permite a execução do sistema livremente, de acordo com o desejo do usuário. Porém, também é importante observar o sistema sob condições randômicas, não em cenários previamente projetados. Podem-se alcançar essas condições usando-se um programa de controle de simulação que gera os eventos randomicamente, inicia e modifica os valores das variáveis e possibilita a captura de informações estatísticas sobre a execução. Isso é feito na execução programada, que complementa as outras técnicas de simulação pois pode produzir resultados que são muito difíceis de se obter com as outras técnicas. Já a simulação exaustiva produz um número muito grande de diferentes possibilidades, tornando-se uma busca impraticável. Uma possível solução para este problema é simular exaustivamente somente as partes mais críticas do sistema. E essa tarefa é simplificada pela natureza hierárquica de Statecharts. Além dessas várias formas de simulação citadas, há uma outra maneira de se validar sistemas especificados em Statecharts. Trata-se da Árvore de Alcançabilidade, proposta por Masiero et al. (1994). O conceito de Árvore de Alcançabilidade foi proposto inicialmente para validar especificações feitas em Redes de Petri, simulando diferentes configurações de uma Rede de Petri, reduzindo assim o tamanho da especificação. A Árvore de Alcançabilidade proposta para Statecharts será abordada na Seção 2.3.1.

No contexto de teste do aspecto comportamental de Sistemas Reativos, observa-se o estabelecimento de métodos/critérios de geração de seqüências de teste baseada em Máquinas de Estados Finitos (Chow, 1978; Fujiwara et al., 1991), entre outros. A atividade de teste de produtos especificados através de MEFs é bastante intensa e relevante em diversos domínios de aplicação, como por exemplo, na área de protocolos de comunicação Bochmann et al., 1992; Bochmann & Petrenko, 1994; Chow, 1978; Petrenko & Bochmann, 1996; Yao et al., 1994). A MGASET é uma ferramenta desenvolvida no Instituto de Ciências Matemáticas e de Computação - USP/São Carlos que gera seqüências de testes baseadas em Máquinas de Estados Finitos (MEFs) (Gill, 1962; Sabnani & Dahbura, 1988; Naito & Tsunoyama, 1981; Fujiwara et al., 1991) e apóia a aplicação dos métodos de geração de seqüências de testes mais tradicionais, como o método W (Chow, 1978). A disponibilidade dessa ferramenta viabiliza a realização de estudos empíricos para avaliar o custo e eficácia de técnicas e critérios alternativos, a exemplo dos trabalhos conduzidos por Probert & Guo (1991) e Fabbri et al. (1993).

Métodos como o método W, de Chow (Chow, 1978), para garantir a cobertura de todos os estados e transições no teste de especificações baseadas em MEFs requerem que a máquina satisfaça certas propriedades como, por exemplo, ser minimal. Como na prática raramente as especificações satisfazem essas propriedades, há necessidade de se complementar a validação de MEFs de outras formas, motivando-se assim o uso de outras técnicas e critérios, como a Análise de Mutantes (DeMillo, 1980). E como Statecharts são uma extensão de MEFs com características intrínsecas como história, paralelismo e *broadcasting*, tornando seus modelos mais complexos, a validação de especificações baseadas nessa técnica através desses métodos torna-se uma atividade ainda mais difícil.

As diversas abordagens de simulação citadas anteriormente são consideradas formas de validação de Sistemas Reativos em geral e necessitam de ferramentas para que possam ser utilizadas (Harel, 1992). Uma questão relevante para a atividade de teste que não é considerada por essas formas de validação, é o aspecto de cobertura. Esse aspecto foi abordado em relação às especificações baseadas em MEFs por Petrenko e Bochmann

(1996), que salientam a relevância de pesquisas nessa direção, já que com a análise de cobertura a qualidade da atividade de teste pode ser quantificada.

No trabalho de Fabbri (1996), explorou-se a adequação do uso do conceito de mutação, especificamente do critério Análise de Mutantes, no contexto de teste e validação de aspectos comportamentais de Sistemas Reativos, considerando-se as técnicas baseadas em modelos de máquinas de transição de estados: Máquinas de Estados Finitos (MEFs) (Gill, 1962); Statecharts (Harel, 1987a) e Redes de Petri (Peterson, 1981). A Análise de Mutantes é um critério de teste proposto para o teste de programas, mas que, segundo Fabbri (1996), pode facilitar a atividade de teste de especificações de Sistemas Reativos. Para apoiar a aplicação do critério neste contexto, Fabbri (1996) especificou a ferramenta Proteum-RS, que possui três instanciações: Proteum-RS/FSM (Fabbri, 1996), Proteum-RS/PN (Simão, 2000) e Proteum-RS/ST (Sugeta, 1999), para o teste de especificações baseadas em MEFs, Redes de Petri e Statecharts. O critério Análise de Mutantes, sua aplicabilidade no teste de especificações e a ferramenta Proteum-RS/ST são descritos com mais detalhe no Capítulo 4.

Para complementar as formas de validação de especificações Statecharts, Souza et al. (2000) propuseram a SCCF (Statechart Coverage Criteria Family), uma família de critérios que fornecem mecanismos para avaliar seqüências de teste geradas por simulação e mecanismos para guiar a geração de seqüências de teste adequadas (a um determinado critério). Esses critérios fornecem uma medida de cobertura para quantificar a atividade de teste e assim também contribuem para a melhoria da qualidade desta atividade no contexto de especificações Statecharts. A aplicação desses critérios de cobertura é feita usando uma representação comportamental dos statecharts, a Árvore de Alcançabilidade, apresentada na Seção 2.3.1, a seguir.

2.3.1 A Árvore de Alcançabilidade para Statecharts

A Árvore de Alcançabilidade foi proposta por Masiero et al. (1994) para a validação de especificações baseadas em Statecharts. Esta proposta foi baseada em algumas extensões do conceito de Árvore de Alcançabilidade desenvolvidas para simular Redes de Petri. A Árvore de Alcançabilidade pode ser tratada formalmente e suporta a análise de várias propriedades tais como: alcançabilidade, reiniciabilidade, seqüência de eventos válidos, *deadlock* e uso de transições.

A simulação exaustiva do ambiente StatSim é baseada na Árvore de Alcançabilidade, como poderá ser observado no Capítulo 3.

Construção da Árvore de Alcançabilidade (Boaventura, 1992)

A árvore de alcançabilidade é construída considerando-se todas as possíveis seqüências de eventos do sistema especificado. Em cada passo, consideram-se todos os eventos verdadeiros, fazendo com que todas as transições habilitadas num passo levem a uma nova configuração do sistema. As condições também são consideradas quando uma transição é habilitada e são incluídas nas expressões dos eventos.

De acordo com Masiero et al. (1994), as regras semânticas para calcular cada passo são as mesmas definidas em (Harel, 1987b). Os cálculos se desenvolvem em uma seqüência de passos representada pelo conjunto $\{(SC_i, \Upsilon_i, \Pi_i^*), i \geq 0\}$, em que:

(1) $SC_0 = (X_0, \Pi_0, \theta_0)$, onde X_0 é a configuração inicial e (Π_0, θ_0) representam os estímulos externos que ocorrem no primeiro intervalo de tempo I_0. Π_0 é o conjunto

de eventos externos gerados pelo ambiente e θ_0 é o conjunto de condições primitivas cujo valor é verdadeiro.
(2) Υ_i é um passo realizado no SC_i no instante de tempo σ_{i+1}, $i \geq 0$.
(3) Π_i^* é o conjunto de eventos gerados por Υ_i, $i \geq 0$.

Com isso, tem-se então uma seqüência de configurações de estados, onde SC_{i+1} é a configuração do sistema alcançada a partir de SC_i quando as transições em Υ_i, $i \geq 0$, são disparadas. Porém, variáveis contidas em condições não são admitidas, pois introduzem indecidibilidade. Outras condições e eventos especiais são considerados quando a árvore é construída.

Para cada elemento na seqüência de configurações de estados, o conjunto $SR=\{(\Upsilon_j, \Pi_j^*)\}$ de todas as possíveis reações do sistema é considerado. Na árvore, cada nó representa uma configuração do statechart. Essas configurações podem ser de quatro tipos: configuração nova, configuração velha, configuração terminal e configuração história. Uma configuração é considerada nova se for gerada e ainda não existir na árvore, caso contrário, ela é denominada configuração velha. Já uma configuração é terminal se não tem nenhuma transição habilitada, isto é, $SR=\emptyset$. E por fim, uma configuração história representa todas as configurações alcançáveis de transições que têm o símbolo de história associado. Os símbolos *H* e *h* aparecem nesses nós em substituição às configurações alcançáveis naqueles estados que são ativados por *H** e *H*, respectivamente. Quando se faz uma busca na árvore, os símbolos de história são substituídos pela última configuração daqueles estados no escopo do símbolo de história em questão.

A configuração inicial (ou *default*) é o nó raiz da árvore de alcançabilidade. A partir dela considera-se o conjunto de todas as transições que podem deixá-la implícita ou explicitamente. Para cada transição encontrada, um novo passo é realizado, gerando a próxima configuração que é classificada e apropriadamente inserida na árvore. Esse procedimento é repetido para cada nova configuração até que não existam mais possíveis configurações.

A hierarquia de estados de um statechart pode ser mostrada através de uma árvore E/OU. A notação utilizada para a representação de um nó da árvore de alcançabilidade é:

- parênteses "(" e ")" são utilizados para representar os superestados tipo XOR.
- colchetes "[" e "]" são utilizados para representar os superestados tipo AND.
- os estados átomos são representados por "1" ou "0", conforme estejam ligados ou desligados, respectivamente. Eventualmente, podem ser representados pelo seu próprio nome.

A semântica dos Statecharts também permite que muitos eventos externos aconteçam ao mesmo tempo. Na árvore de alcançabilidade considera-se cada simples transição ao invés de se combinar diferentes eventos que poderiam ocorrer no mesmo passo, habilitando dessa maneira transições em componentes ortogonais.

A árvore de alcançabilidade também pode ser usada na análise de algumas propriedades de um statechart: alcançabilidade, reiniciabilidade, *deadlock*, uso de transições e seqüência de eventos válidos:

- Alcançabilidade

Alcançabilidade em Statecharts é definida como sendo a busca de configurações de estados que o sistema pode alcançar. Essa propriedade pode ser útil quando um sistema está sendo especificado ao fornecer meios de verificar se todos os estados especificados são ativados em algum momento, ou se, sob certas condições, alcança uma determinada

configuração. Consideram-se dois tipos de alcançabilidade: alcançabilidade de uma configuração a partir da configuração inicial e alcançabilidade de uma configuração a partir de uma outra configuração qualquer.

Alcançabilidade de uma configuração a partir da configuração inicial
O objetivo é verificar, a partir da configuração inicial, se uma determinada configuração de estados pode ser alcançada. Uma configuração é alcançável se for um nó da árvore. Portanto, o algoritmo que verifica essa propriedade percorre a árvore em busca da configuração dada. Se ela estiver na árvore, é fornecida a seqüência de eventos que leva da configuração inicial de estados até essa configuração de estados. Podem existir mais de uma seqüência de eventos que leva à configuração desejada. Caso contrário, a configuração dada não é alcançável.

Alcançabilidade de uma configuração a partir de uma configuração qualquer
Dadas duas configurações de estados SC_1 e SC_2, essa propriedade verifica a existência de um caminho que leva de SC_1 a SC_2. Caso exista um caminho, mostra-se a seqüência de eventos necessária para o sistema alcançar SC_2 a partir de SC_1. Essa propriedade é verificada procurando-se as duas configurações na árvore. Se elas existirem, inicia-se a busca com base em SC_2 a partir de SC_1. Se SC_2 for um descendente de SC_1, essa busca é trivial. Senão, um caminho de ancestrais de SC_2 deve ser construído e uma busca é realizada, a partir de SC_1, verificando se existe algum descendente de SC_1 cujo ponteiro de retorno aponta para algum ancestral de SC_2. Uma outra possibilidade é que SC_2 seja alcançada a partir de SC_1 por história. Nesse caso, verifica-se se SC_1 possui algum descendente que seja um nó história e, se houver, deve-se verificar se a configuração SC_2 é alguma instância desse nó história encontrado.

- Reiniciabilidade

Para verificar se um statechart é reinicializável basta verificar se para toda configuração SC, alcançada a partir de SC_0, existe uma seqüência de eventos que a faça retornar à configuração SC_0.

- *Deadlock*

Um sistema modelado em Statecharts pode ter possíveis *deadlocks* se atingir configurações de estados das quais nenhuma transição pode disparar e, portanto, em sua árvore de alcançabilidade só aparecerão nós terminais. Diz-se "possíveis *deadlocks*" pois há sistemas que modelam algumas tarefas que possuem início e fim de processamento bem definido e a árvore de alcançabilidade neste caso também possui nós terminais que não indicam *deadlocks* e sim o final do processamento dessas tarefas. Dessa forma, cabe ao responsável pela especificação do sistema analisar as respostas e verificar se realmente existe *deadlock*.

- Uso de Transições

Esta propriedade garante que o sistema foi especificado corretamente de modo que não haja especificações de transições que nunca disparam, ou então, que nenhuma transição foi esquecida no statechart modelado. Caso uma transição nunca dispare ou não tenha

sido especificada no statechart, ela não aparece como um arco da árvore de alcançabilidade.

- Seqüência de Eventos Válida

Uma seqüência de eventos é válida se cada um dos eventos dessa seqüência causar uma mudança na configuração de estados do sistema modelado. O objetivo dessa propriedade é validar uma seqüência de eventos através de um algoritmo que percorre a árvore de alcançabilidade em busca de cada elemento dessa seqüência. A busca é feita em duas etapas: na primeira delas obtém-se o primeiro elemento da seqüência fazendo-se uma busca recursiva na árvore e, caso esse elemento seja encontrado, a segunda etapa procurará os outros elementos na árvore. A segunda etapa é bem sucedida se, a partir do nó em que incide o arco cujo rótulo é o nome do primeiro elemento da seqüência, existe um caminho no qual os eventos correspondem aos elementos restantes da seqüência dada. Caso a segunda etapa venha a falhar, continua-se a buscar recursivamente o primeiro elemento da seqüência, passando então para a segunda etapa se for encontrado um novo arco com o seu nome. Esse processo termina se conseguir validar a seqüência de eventos ou se o processamento recursivo chegar ao final.

O tempo de execução do algoritmo responsável por esse processo tende a aumentar consideravelmente com a complexidade do sistema e o tamanho da árvore de alcançabilidade gerada.

Como as características intrínsecas da técnica Statecharts facilitam a especificação de aspectos comportamentais dos sistemas, Turine et al. (1998) afirmam que essa técnica também pode fornecer suporte adequado à especificação de hiperdocumentos, minimizando diversos problemas relacionados a essa atividade. Porém, como às vezes é necessário definir o comportamento temporal da aplicação hipermídia, que envolve aspectos de sincronização, a utilização da técnica Hypercharts, uma extensão da técnica Statecharts que incorpora a descrição de requisitos de seqüências de tempo e sincronização, se torna mais adequada. Assim, na Seção 2.4, a técnica Hypercharts é brevemente descrita.

2.4 Hypercharts

Aplicações hipermídia integram os conceitos de hipertexto e multimídia em um modelo simples para organização e recuperação de informações. Elas herdam dos hipertextos a organização não-linear de informação e permitem aos usuários navegar e recuperar informações pela ativação de *links* estabelecidos no documento. A incorporação de recursos multimídia, com a integração simultânea de diferentes mídias estáticas e dinâmicas (dependentes de tempo) também melhora consideravelmente a habilidade da aplicação em tratar a informação. A especificação de sistemas hipermídia pode ser uma tarefa complexa e métodos e modelos de especificação podem auxiliar o processo de desenvolvimento desses sistemas e de hiperdocumentos, que são aplicações por eles gerenciadas. Porém, métodos e técnicas tradicionais de desenvolvimento de software não são adequados para o projeto de hiperdocumentos, que apresentam requisitos próprios, tais como a necessidade de gerenciar um grande volume de informações e de combinar a navegação controlada pelo leitor com a própria natureza das informações multimídia. Além

disso, outros problemas associados ao projeto de hiperdocumentos são a captura e organização da estrutura do domínio de informação, muitas vezes complexa, de maneira a torná-la clara e acessível aos leitores; a criação de hiperdocumentos para plataformas diferentes a partir de uma única especificação/projeto; e o estabelecimento de abordagens sistemáticas para definir a estrutura organizacional e a semântica de navegação de hiperdocumentos (Turine et al., 1998).

Assim, buscam-se modelos e métodos que forneçam diretrizes para gerenciar o projeto de hiperdocumentos estruturados, ajudando a disciplinar a atividade de autoria. O uso de modelos não somente melhora o processo de projeto e desenvolvimento de hiperdocumentos, como também facilita sua análise e avaliação.

Segundo Turine et al. (1998), a técnica Statecharts pode fornecer um suporte adequado para minimizar diversos problemas associados à especificação de hiperdocumentos. Foi proposto então o modelo HMBS - ***Hyperdocument Model Based on Statecharts*** - ou Modelo de Hiperdocumentos Baseado em Statecharts, que permite separar as informações referentes à estrutura organizacional e navegacional das representações físicas do hiperdocumento (Turine et al., 1998). E, para oferecer suporte automatizado ao desenvolvimento de aplicações hipermídia, foi desenvolvido o HySCharts (***Hyperdocument System based on StateCharts***), que oferece ambientes de autoria e de navegação que permitem criar e validar especificações de hiperdocumentos segundo o modelo HMBS. Este ambiente e o modelo HMBS são apresentados no Capítulo 3.

Muitas vezes porém, é necessário definir o comportamento temporal da aplicação hipermídia. Esse comportamento é responsável por impor a seqüência de relacionamentos entre os diferentes dados e os requisitos de sincronização relacionados à apresentação de dados dinâmica, tais como áudio e vídeo, durante a navegação. Sua especificação exige modelos adequados capazes de descrever políticas de sequenciamento e sincronização. Muitos modelos foram propostos para fornecer suporte à especificação e desenvolvimento multimídia, e muitos deles são baseados em Redes de Petri, e se concentram na especificação de requisitos de sincronização. Porém, como são poucos os modelos válidos para a especificação de aplicações hipermídia em geral, foi proposto um novo modelo, o XHMBS (eXtended Hyperdocument Model Based on Statecharts) (Paulo et al., 1998), que fornece um formalismo visual para a especificação de sistemas hipermídia usando Hypercharts como uma alternativa às Redes de Petri como seu modelo básico. Como no contexto deste trabalho é relevante somente a compreensão dos Hypercharts, informações sobre o XHMBS podem ser obtidas em (Paulo et al., 1998). As informações que seguem sobre Hypercharts são retiradas de (Paulo et al., 1998).

Hypercharts são Statecharts estendidos que, além das características de Statecharts, incorporam a descrição de requisitos de seqüências de tempos e sincronização típicos de aplicações multimídia. Isso é obtido com três novos conjuntos de notações: história temporal, transições temporais e mecanismos de sincronização. Além disso, são introduzidas notações para parametrização de estados e abstração de transição para permitir a descrição compacta de aplicações de tamanho real. Uma estrutura hyperchart (HYP) é uma 14-tupla:

$$HYP=<\mathbf{S},\rho,\psi,\delta,\gamma,\mathbf{V},\mathbf{C},\mathbf{E},\mathbf{T},\mathbf{Ac},\tau,LSC,T\_hist,T_S>, \text{ em que:}$$

-**S** é o conjunto de estados

-ρ: $\mathbf{S}\rightarrow 2^{\mathbf{S}}$ é a função hierárquica que define os subestados de cada estado (um estado s é básico se $\rho(s) = \varnothing$)

-ψ: $\mathbf{S}\rightarrow\{AND,OR\}$ é uma função que define o tipo de cada estado

-δ: $S \rightarrow 2^S$ é a função *default* que define o conjunto de estados iniciais contidos em um estado *s*
-γ: **H**→**S** é a função história, responsável por mapear símbolos de história aos estados para que $\gamma(\mathbf{H}) = \{y\}$ se $y \in \mathbf{S}$ e $\psi(y) = \{OR\}$
-**V** é o conjunto de expressões contendo identificadores de variáveis lógicas
-**C** é o conjunto de condições, que podem ser T (*true*), F (*false*) or expressões lógicas
-**E** é o conjunto de expressões de eventos, também chamadas de rótulos
-$\mathbf{T} \subset 2^S \times \mathbf{E} \times 2^S$ é o conjunto de transições
-**Ac** é o conjunto de ações

Como pode ser observado, as definições acima são comuns aos Statecharts e Hypercharts. A semântica dos Hypercharts segue a semântica operacional dos Statecharts, caracterizada pela seqüência de passos de execução, cada um definindo uma configuração de estado válida. Um relógio global controla os passos de execução, e cada passo de execução pode consistir de alguns sub-passos, em que uma simples transição é ativada em um momento.

A seguir, descreve-se a sintaxe das extensões introduzidas nos Hypercharts. As semânticas associadas também são discutidas.

-τ: $\mathbf{T} \rightarrow \{true, false\}$ é uma função de história temporal. $\tau(t) = true$ implica que a transição *t* tem um símbolo de história temporal associado. τ^*: $\mathbf{T} \rightarrow \{true, false\} \mid \tau^*(t) = true$ implica que a transição *t* tem um símbolo de história temporal recursivo associado. A representação gráfica de história temporal é um ícone de relógio colocado perto da transição *t*.

-*LSC*: **S**→**N**, sendo **N** o conjunto de números naturais, é uma função de Contador de Passo Local (**L**ocal **S**tep **C**ounter) que age como um contador de passos de execução para um estado *s*, registrando o tempo de ativação total de *s* em termos de passos de execução. Esse contador é incrementado a cada passo de execução do statechart enquanto o estado permanecer ativo. O relógio global do sistema é o mecanismo que permite a atualização incremental dos LSC*s*.

-*T_hist*: **S**→**N** é uma função de registro do tempo responsável por guardar o valor de LSC de um estado *s* no momento em que *s* foi desabilitado por último.

-$T_S \subseteq (2^{S \times L}) \times 2^{S \times H} \times sync \times (\mathbf{S} \cup \varnothing)$, em que *sync* é um tipo de sincronização, é o conjunto de *M:N* transições sincronizadas. Tais transições, além de terem um tipo que define uma política de sincronização, são rotuladas com eventos temporais.

Hypercharts fornecem notações definidas em termos da semântica de Statecharts, então qualquer hyperchart pode ser visto como uma descrição sintática que pode ser transformada em um statechart semanticamente equivalente. Portanto, é sempre possível gerar um statechart que se comporta como um dado hyperchart (Paulo, 1997).

História Temporal

A atribuição do símbolo de história temporal - o ícone de relógio - a uma transição *t* com um estado destino *s*, como ilustrado na Figura 2.5, implica que, se *s* esteve ativo no passado, o disparo de *t* recuperará o valor do registro de história temporal de *s*, além de recuperar a última configuração de *s* (em termos de seus subestados). A versão recursiva do símbolo de história temporal implica que os registros de história temporal de *s* e de

todos os seus subestados, recuperados por história recursiva convencional, também serão recuperados.

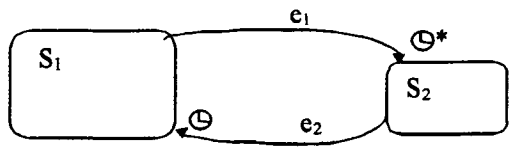


Figura 2.5 - Notação de história Temporal (Paulo et al., 1998)

Para todo estado $s \in S$ para o qual a história temporal será registrada, há uma ação "*On exit A_Thist(s):=LSC(s)*" atribuída, assegurando que seu registro de história temporal será atualizado sempre que o sistema sair de *s*. Se há uma transição *t* com história temporal (recursiva), sempre que *t* é disparada, se os estados no conjunto destino de *t* já estiveram ativos pelo menos uma vez, a operação "*LSC(s):=Thist(s)*" é executada para todo estado *s* ativado pelo símbolo de história convencional (recursivo). Se a transição não tem um símbolo de história temporal atribuído, a operação "*LSC(s):=0*" é executada para cada estado *s* recuperado pelo símbolo de história convencional (recursivo) ou ativado por *default*.

História temporal permite que as atividades associadas aos estados sejam retomadas do ponto onde foram anteriormente interrompidas. Tão logo o sistema inicia sua execução, a operação "*A_LSC(s):=0*" é executada para todo estado *s*. Também, para cada passo de execução *p*, a operação "*LSC(s):=LSC(s)+1*" é executada para todos os estados que estavam ativos no passo anterior.

Transições Temporais

Transições temporais têm seus disparos controlados pelo progresso de tempo durante o período em que seus estados origens estiverem ativos. Elas fornecem uma capacidade de especificação temporal poderosa e útil para especificação de comportamentos funcionais dependentes de tempo, suportando a especificação de requisitos de apresentação multimídia, tais como atrasos e *jitter*. Elas têm eventos temporais atribuídos da forma "[α,η,β]", e devem ter cardinalidade 1:1. Além disso, um estado pode ser origem de somente uma transição temporal.

A Figura 2.6 apresenta a forma genérica de uma transição temporal e seu evento temporal, em que *a* denota uma ação que pode ser executada depois do disparo da transição, de acordo com a semântica convencional de Statecharts. Uma transição temporal neste exemplo, implica que há uma atividade principal *A* sendo controlada pelo estado *X* (conhecida como atividade de apresentação de *X*), que gera um evento associado *end_X* indicando seu término.

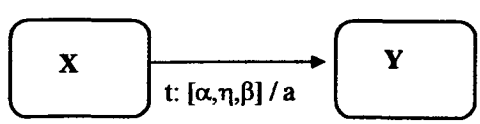


Figura 2.6 - Transição Temporal (Paulo et al., 1998)

A notação [α,η,β] impõe duas restrições no disparo da transição *t*. A primeira é que, mesmo que a atividade *A* tenha acabado, ou seja, *end_X* ocorreu, *t* não disparará a

menos que o limite temporal mínimo α tenha sido alcançado, isto é, a menos que a aplicação tenha permanecido pelo menos α passos no estado *X*. Se o evento de término *end_X* ocorrer em qualquer instante entre α e β, *t* disparará imediatamente. A segunda restrição é relacionada ao limite máximo temporal β. Se a aplicação permanecer no estado *X* até o momento β e o evento *end_X* não ocorrer, *t* disparará de qualquer forma, abortando a execução da atividade *A*.

Para modelar aplicações hipermídia, é suficiente considerar que algum estado básico *X* controla a execução de uma atividade durante seu tempo de ativação. Semanticamente, os estados básicos que são origens de uma transição temporal podem ser decompostos em três subestados exclusivos, responsáveis pelo controle de possíveis condições de disparo de transições, e três variáveis (*min_X_t,end_X_t,max_X_t*) são usadas para indicar a razão exata do disparo. O término da atividade de apresentação é indicado pelo evento interno *end_X*, definido para cada estado *X*. A Figura 2.7 mostra a expansão do statechart equivalente para a transição temporal exemplificada pelo hyperchart da Figura 2.6.

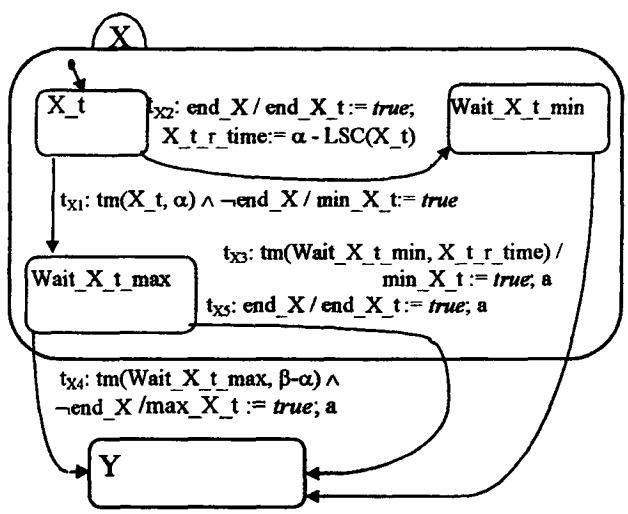


Figura 2.7 - Exemplo de transformação para estados básicos (Paulo et al., 1998)

Se *X* não é um estado básico, uma transformação equivalente é obtida com a criação de um subestado de *X* que controla o término da atividade de apresentação de *X*. Este subestado deve ser concorrente a quaisquer outros subestados de *X*. Um tipo especial de *time-out* é também definido em Hypercharts, denotado por "*tm(s,n)*", em que *s* é um estado e *n* é o número de passos em que *s* permaneceu ativo.

Mecanismos de Sincronização

A sincronização é especificada em Hypercharts por meio de *M:N* transições sincronizadas, que têm cinco componentes: estados origens, arcos origens, estados destinos, rótulos (um para cada arco origem) e um tipo de sincronização. O tipo de sincronização indica como as restrições impostas pelos eventos temporais associados serão consideradas na geração de uma única restrição temporal para todas as transições. São válidos nove tipos de sincronizações, sendo cinco deles básicos, denominados *strong-or (so)*; *weak-and (wa)*; *master (m)*; *or* e *and;* e os outros quatro compostos, derivados dos

básicos, denominados *or-master*; *strong-master*; *weak-master* e *and-master*. A Figura 2.8 ilustra um exemplo de representação visual de uma transição sincronizada com M = 3 e N = 1, em que T indica um dos tipos de sincronização. Arcos origens não devem ter rótulos com condições ou ações associadas - a única expressão permitida no rótulo de um arco origem é um evento temporal.

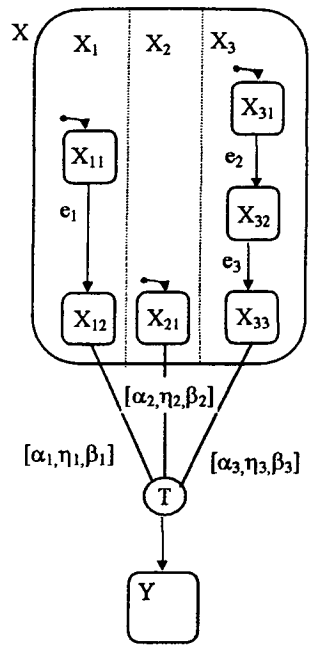


Figura 2.8 - Notação genérica para M:N transições sincronizadas (com N =1) (Paulo et al., 1998)

Uma transição M:N sincronizada com M estados origens cujos respectivos arcos origens são rotulados com expressões do tipo $[\alpha_i,\eta_i,\beta_i]$ $(1 \leq i \leq n)$, tem seu momento de disparo θ definido em um intervalo de tempo da transição. Exemplificando, para uma sincronização do tipo *OR*, o intervalo é dado por $\min_i(\alpha_i) \leq \theta \leq \max_i(\beta_i)$, pois a política de sincronização OR considera a corretitude temporal do primeiro de seus dados que terminar com sucesso sua atividade de apresentação. Assim, a transição temporal dispara tão logo qualquer estado declare o término de sua atividade de apresentação, desde que todas elas já tenham iniciado ou tão logo todos os estados tenham alcançado seu limite temporal máximo.

A corretitude temporal de qualquer atividade atribuída a um estado s é garantida se as restrições impostas pela transição temporal ou pelo arco origem que parte de s, são satisfeitas. Se um fluxo de dado é representado por um conjunto de estados mapeados para atividades que exibem os dados e são relacionados por transições temporais, sua corretitude temporal é garantida se é garantida para todo estado que compõe o fluxo de dado. Um cenário de sincronização é o conjunto de componentes concorrentes de um hyperchart que são parte de uma transição sincronizada M:N. Assim, ele sempre contém dois ou mais fluxos de dados a serem sincronizados em algum ponto. Cada um desses fluxos de dados é representado por um componente OR do hyperchart. Geralmente, dois pontos de sincronização, denominados pontos de início e de fim, definem um cenário de sincronização.

A semântica do processo de transformação para derivar um statechart equivalente de um hyperchart é baseada em regras de disparo definidas pela política de sincronização. Por exemplo, se o hyperchart da Figura 2.8 ilustra um tipo de sincronização OR, a representação de seu statechart equivalente é ilustrada na Figura 2.9.

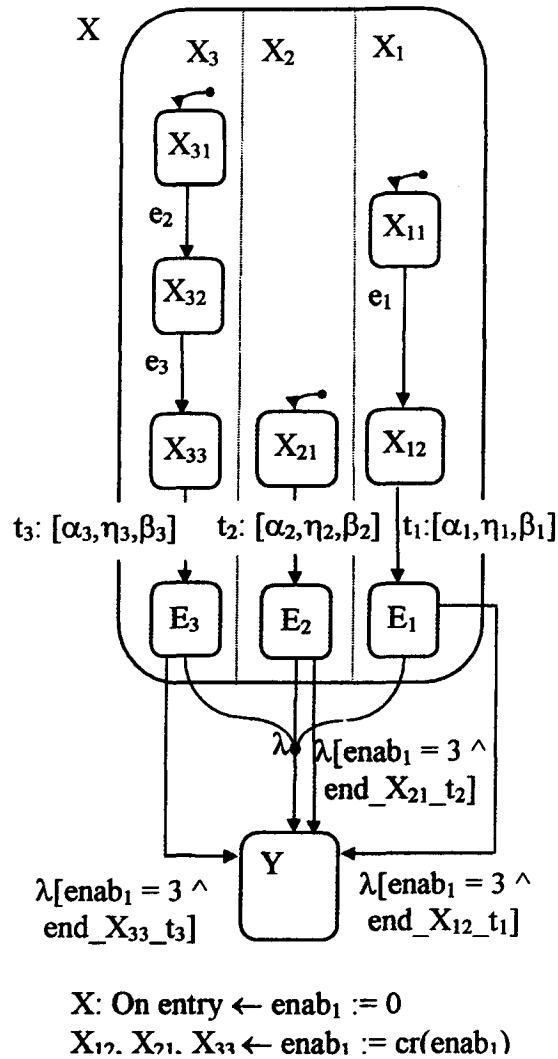


Figura 2.9 - Transformação para uma transição OR (Paulo et al., 1998)

Para a utilização da técnica Statecharts na especificação de sistemas, é importante o apoio automatizado de ferramentas. O ambiente StatSim e o ambiente HySCharts, que apóiam a aplicação de Statecharts, são apresentados no Capítulo 3.

CAPÍTULO 3

FERRAMENTAS DE APOIO À TÉCNICA STATECHARTS

Neste capítulo são apresentadas duas ferramentas que apóiam a utilização da técnica Statecharts, o ambiente StatSim e o ambiente HySCharts. O ambiente StatSim permite a edição e simulação de Statecharts, e o ambiente HySCharts permite a criação, interpretação e execução de especificações formais de aplicações hipermídia de acordo com o modelo HMBS (Hyperdocument Model Based on Statecharts), que utiliza a estrutura e a semântica operacional de Statecharts para especificar a estrutura organizacional e a semântica de navegação de hiperdocumentos grandes e complexos.

3.1 Ambiente StatSim

O ambiente StatSim (Statecharts Simulator) (Masiero et al., 1991) foi desenvolvido pelo grupo de Engenharia de Software do Instituto de Ciências Matemáticas e de Computação da USP/São Carlos. Esse ambiente é similar à ferramenta comercial STATEMATE (Harel et al., 1990), permitindo a edição e simulação de Statecharts, na forma gráfica ou textual, sendo esta última, através da LES (Linguagem de Especificação de Statecharts) (Fortes, 1991). Seu desenvolvimento foi baseado na sintaxe e semântica dos Statecharts, da maneira como definida em (Harel, 1987b). Dois módulos principais compõem esse ambiente: um editor, que permite a inserção e edição de estados e transições, além da especificação de variáveis, e outro módulo simulador de Statecharts, que possibilita todas as formas de simulação de statecharts. Assim, após a edição de um statechart através do módulo de edição do ambiente, pode-se realizar sua simulação. A janela principal do ambiente permite a manipulação dos dois módulos disponíveis e é mostrada na Figura 3.1. As figuras que seguem ilustrando os módulos de edição e simulação são de um statechart que especifica o problema de dois processos compartilhando dois recursos. Os dois processos executam concorrentemente (Masiero et al., 1994). Informações referentes ao modo de utilização do ambiente podem ser encontradas no Manual de Uso do Sistema StatSim (Penteado et al., 1995).

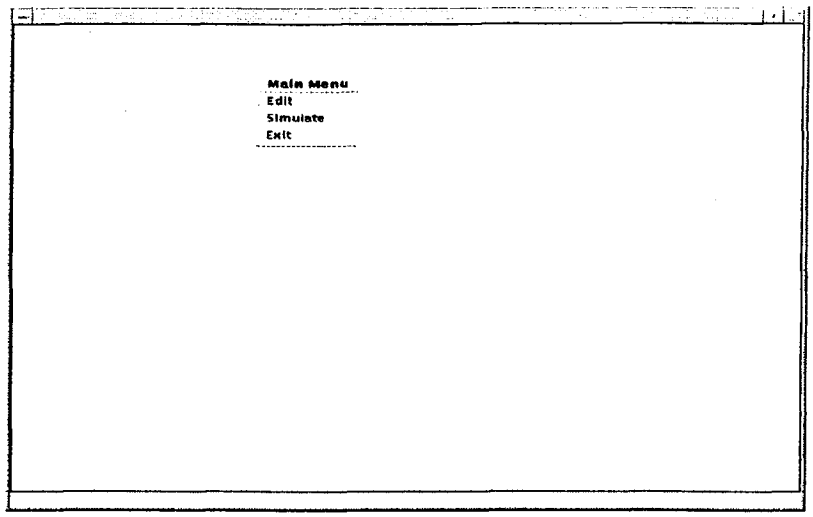


Figura 3.1 - Opções existentes no ambiente StatSim

Módulo de Edição

O módulo de edição permite operações de manipulação de arquivos, estados, transições, atividades, variáveis, entre outros. A tela do editor gráfico é mostrada na Figura 3.2.

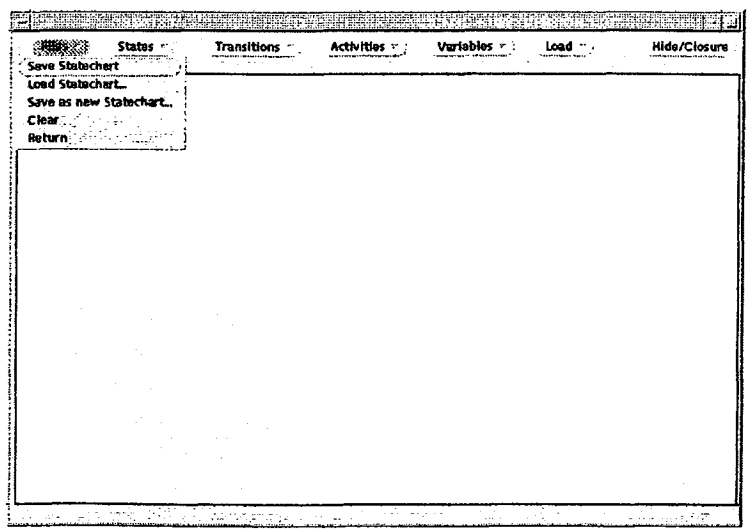


Figura 3.2 - Editor Gráfico de Statecharts com as opções do botão Files

O botão *States* permite que estados sejam criados de modo que seja informado o nome do novo estado e o seu tipo (AND ou OR), sendo que o tipo OR é o *default*. Caso o estado seja AND, o número de filhos que ele possui também deve ser informado. No ambiente StatSim, estados podem ter atividades associadas. Essas atividades estão sujeitas a três tipos de ativação: DO - quando elas têm duração igual ao período que o estado está ativo; ENTRY- a ativação inicia quando o estado é ativado e pára por conta própria; e EXIT - a ativação tem início quando o estado é desativado e pára por conta própria. Também pode-se remover, consultar e decompor estados através do botão *States*. A criação

do estado do tipo AND, com dois subestados, por exemplo, PxR, é apresentada na Figura 3.3.

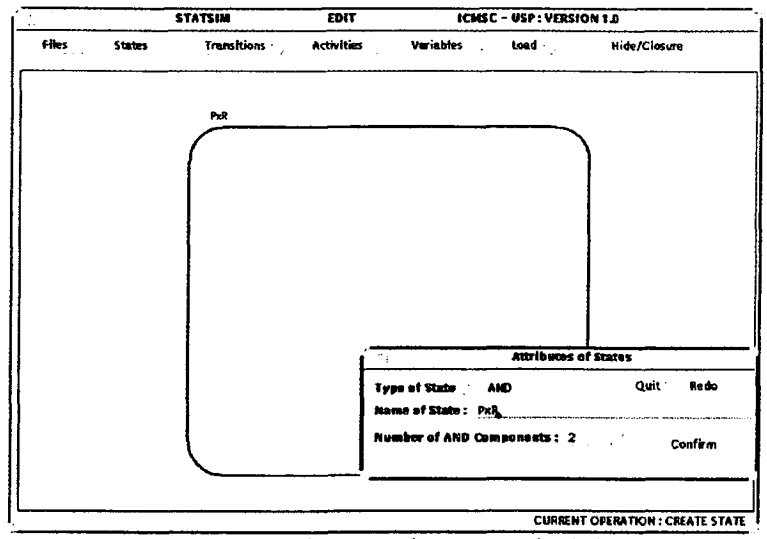


Figura 3.3 - Criação de um estado AND

Variáveis podem ser criadas com o botão *Variables*. Podem-se declarar e consultar variáveis. Elas terão um tipo e, quando consultadas, seu valor atual será indicado.

As transições podem ser criadas pelo botão *Transitions*, e podem ser de três tipos: 1:1- a transição tem um estado origem e um estado destino; M:N- a transição tem M estados origens e N estados destinos; *Default*- a transição tem somente estado destino, não tem rótulo e pode ter ou não ações associadas a ela. A opção *default* é 1:1. As transições 1:1 e M:N requerem rótulos e eventos. Ações são ativadas simultaneamente quando suas transições associadas são disparadas. Uma transição também pode ter *história* associada a ela. Também é possível remover e consultar transições. A Figura 3.4 mostra a criação de uma transição 1:1.

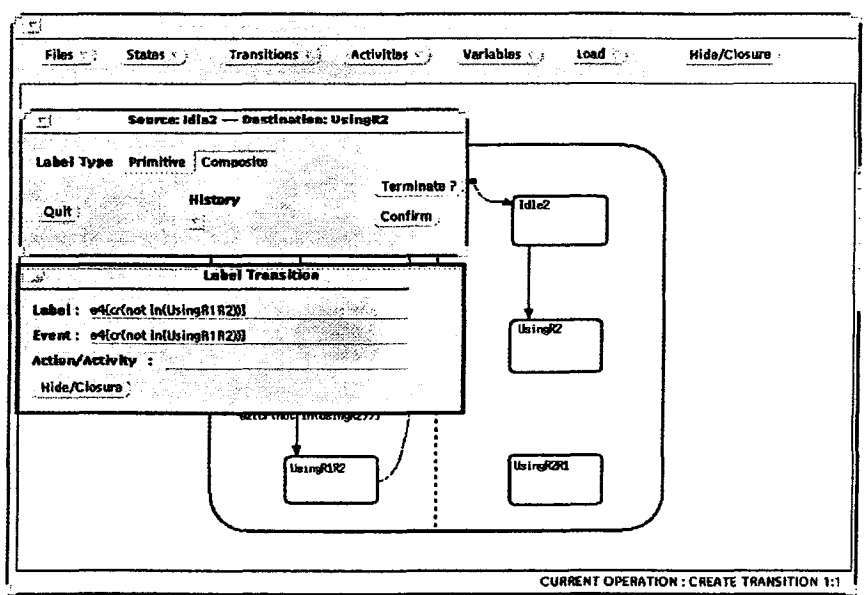


Figura 3.4 - Criação de uma transição 1:1

A Linguagem de Especificação de Statecharts (LES) (Fortes, 1991) é utilizada nas especificações feitas no modo textual do ambiente StatSim e tem uma gramática definida. Porém, apesar de utilizá-la para validar as especificações, o editor gráfico normalmente dispensa essa notação textual, com exceção de algumas situações em que a sintaxe deve ser seguida pelo usuário, ou seja, na definição de eventos, condições e ações associadas às transições. A gramática completa da LES é apresentada no Apêndice A. Um exemplo de especificação em LES também é apresentado neste apêndice.

Módulo de Simulação

Após a edição de um statechart, pode-se usar o módulo de simulação, que permite a manipulação de arquivos, a exibição da simulação de estados decompostos, a obtenção de informações a respeito dos componentes do statechart, a manipulação de variáveis, a inicialização do statechart, operações sobre os modos de simulação e diversas opções como ocultamento de variáveis, visualização do arquivo de log, visualização dos eventos, entre outras.

A opção *Default Configuration* inicia a simulação do statechart, marcando seus estados *default*.

Para que um statechart seja simulado, pode-se usar o botão *Simulation* que permite a escolha do modo de simulação desejado, que pode ser interativo, *batch*, programado ou exaustivo.

Simulação Interativa

A simulação interativa permite que o usuário especifique em cada passo os eventos que devem ser disparados, criando cenários da execução no modo gráfico, ou um "log" no modo textual. Esta é a opção *default*. A tela do ambiente na simulação interativa do statechart exemplo é apresentada na Figura 3.5. Os estados em destaque são aqueles que estão ativos no momento.

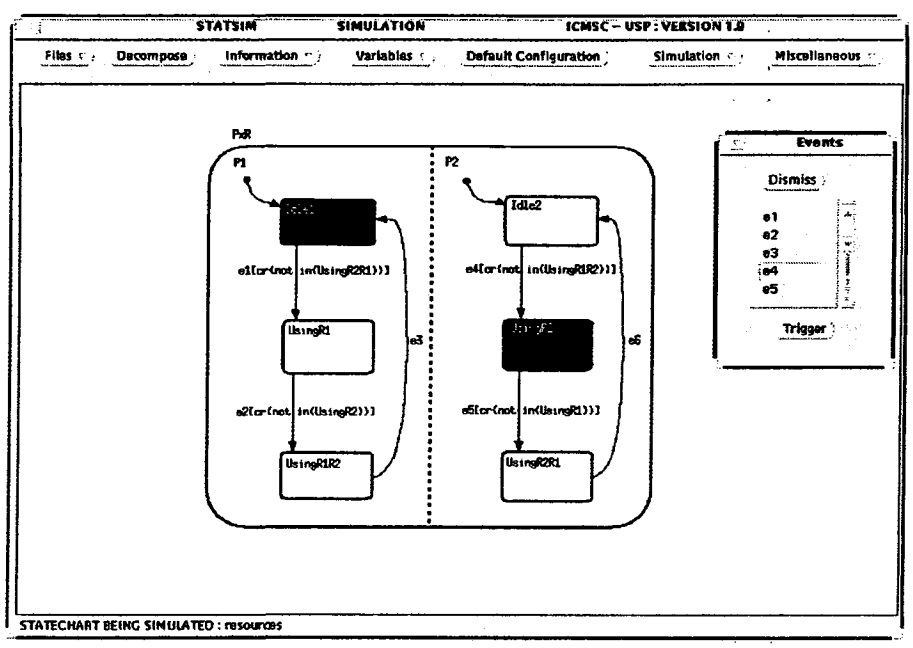


Figura 3.5 - Simulação Interativa no ambiente StatSim

Simulação Batch

Na simulação *batch* define-se uma seqüência de eventos que deve ser executada pelo ambiente de forma animada ou passo a passo, sob controle do usuário. Essa seqüência de eventos deve ser declarada em um arquivo que será executado, como na Figura 3.6.

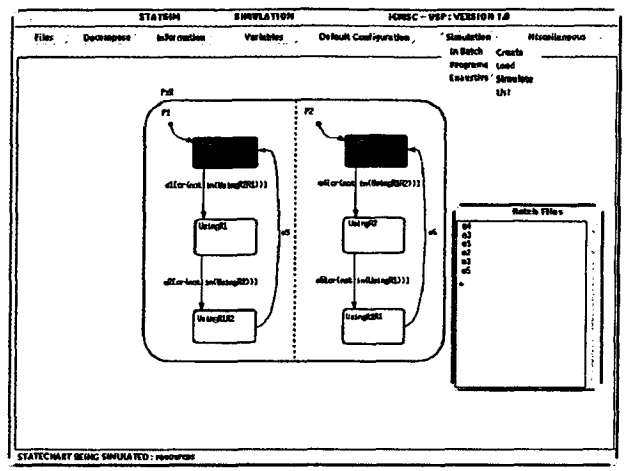


Figura 3.6 - Simulação Batch

Simulação Programada e a Linguagem de Controle de Execução (LCE)

A simulação programada, Figura 3.7, é realizada de acordo com um roteiro previamente especificado pelo programa de controle e com as ocorrências de eventos especificados através de distribuições probabilísticas que modelam seus comportamentos no mundo real (Cangussu, 1995). No StatSim, a linguagem de controle é a Linguagem de Controle de Execução - LCE (Execution Control Language). Os aspectos gerais e a sintaxe da LCE são apresentados no Apêndice B.

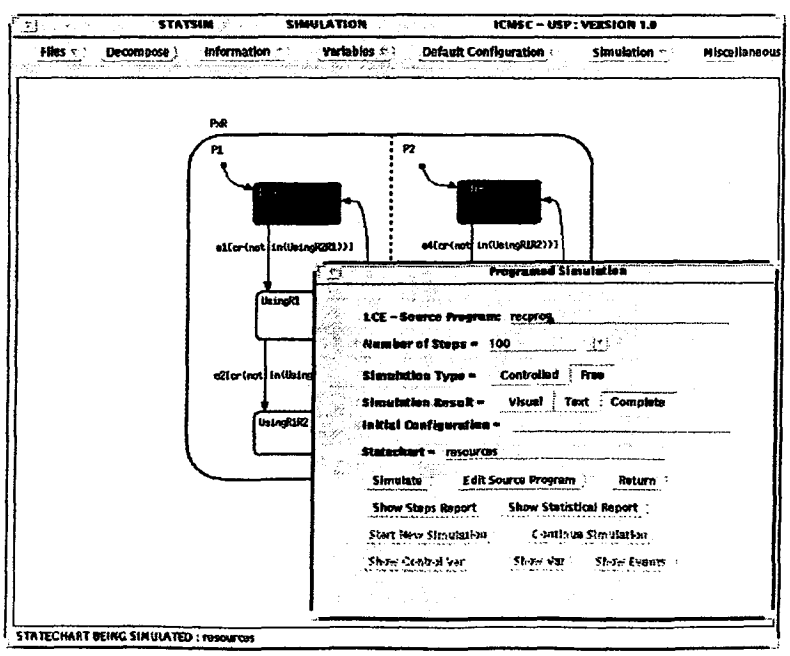


Figura 3.7 - Simulação Programada no ambiente StatSim

A LCE é composta de três partes, sendo que na primeira parte são definidos os parâmetros globais, como o número de passos na simulação, a configuração inicial, se a simulação deve ser controlada pelo usuário ou livre, isto é, automaticamente sem o controle do usuário. O usuário pode especificar esses parâmetros em um programa fonte escrito em LCE ou preencher um formulário fornecido pelo StatSim. Valores *default* são válidos para todos os parâmetros globais.

A segunda parte da LCE permite a declaração de variáveis de controle e a especificação da distribuição dos eventos. Variáveis de controle são do tipo inteiro e são usadas no programa de simulação para registrar certos aspectos da simulação ou para tomar decisões que afetam o fluxo da simulação. Não se deve confundi-las com as variáveis dos Statecharts que são definidas nos próprios statecharts e podem ser manuseadas dentro de qualquer programa LCE.

As ocorrências de eventos externos são especificadas através do uso de distribuições probabilísticas: normal, exponencial e uniforme. Também podem-se especificar eventos que ocorram em intervalos fixos ou usá-los em distribuição linear. Além disso, é possível simplesmente criar um arquivo seqüencial contendo a seqüência de eventos desejada. Novos eventos que não pertencem ao statechart sendo simulado não podem ser criados.

A terceira parte da LCE especifica o procedimento de simulação (ou roteiro), basicamente definindo certos estados desejados em alguns passos específicos e o que fazer quando a simulação alcança aqueles estados ou passos. A Tabela 3.1 tem um resumo dos comandos válidos na LCE. O usuário pode programar interrupções na execução automática do statechart e intervir usando facilidades fornecidas pela execução interativa.

Tabela 3.1 - Resumo dos comandos em LCE (Cangussu, 1995)

Declaração	Significado
AT STEP s1, ..., sn	Especifica as declarações que serão executadas nos passos s1, ..., sn
ALL STEPS	Especifica as declarações que serão executadas em todos os passos
STEP FOR	Especifica as declarações que serão executadas nos passos determinados pela declaração FOR
IF	Declaração condicional
FOR	Declaração de Loop
WHILE	Declaração de Loop
SHOW_VAR	Mostra as variáveis do statechart
SHOW_CONTROL_VAR	Mostra as variáveis de controle
SHOW_EVENTS	Mostra os eventos disparados pelo passo atual
INTERACTIVE	Retorna para a simulação interativa
EVENTS	Dispara os eventos passados como parâmetros
AVAIL	Torna os eventos passados como parâmetros válidos
UNAVAIL	Torna os eventos passados como parâmetros inválidos
MESSAGE	Imprime mensagens/valores
SET_CONFIGURATION	A configuração especificada torna-se a configuração do statechart
ACTIVE	Verifica se os estados especificados estão ativos
CONFIGURATION	Verifica se a configuração especificada está ativa

O StatSim tem um Módulo de Execução Programada (PEM) que possui facilidades para editar e executar programas escritos em LCE. Durante uma simulação, o interpretador da LCE acessa a base de dados do statechart sendo simulado para fazer verificação de consistência. O mesmo procedimento de simulação usado na simulação interativa é utilizado. Assim, pode-se editar um programa escrito em LCE (Cangussu, 1995), como

ilustrado na Figura 3.8. Esse programa realiza a simulação do statechart em 100 passos e uma distribuição probabilística normal com parâmetros 2.5 e 0.5 para os eventos *e1*, *e3*, *e4*, *e6*, uma distribuição probabilística exponencial para o evento *e2* e uma distribuição uniforme para o evento *e5*. Quando o statechart entra em *deadlock*, o programa faz com que o statechart volte para sua configuração *default* e continua a simulação. Também é verificado o número de vezes que cada recurso foi utilizado e quantas vezes o statechart entrou em *deadlock*.

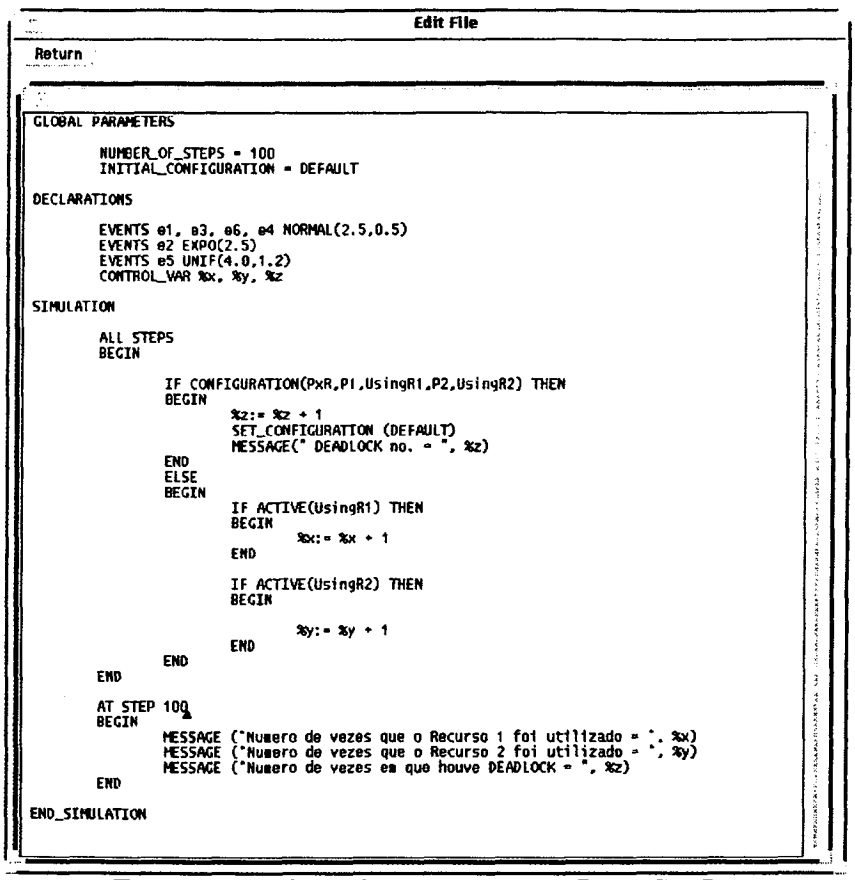


Figura 3.8 - Edição de um programa LCE no StatSim

Durante a simulação, os eventos gerados pelas funções probabilísticas são dados como entrada para o interpretador que chama o procedimento de simulação. Esse procedimento faz a simulação e retorna a nova configuração. Ele também manuseia todas as mudanças na tela e no arquivo de log. Um relatório que mostra os passos da simulação é preenchido com as novas informações a cada passo, e um relatório estatístico é gerado no fim da execução baseado na informação coletada e armazenada a cada passo. A Figura 3.9 ilustra a situação do statechart nos passos 99 e 100 do relatório de passos gerado após a simulação do statechart, e a Figura 3.10 ilustra o relatório estatístico da simulação, com informações sobre os estados básicos, eventos, transições e variáveis. O usuário pode criar outros relatórios baseados nas variáveis de controle e no comando "MESSAGE" (Cangussu, 1995).

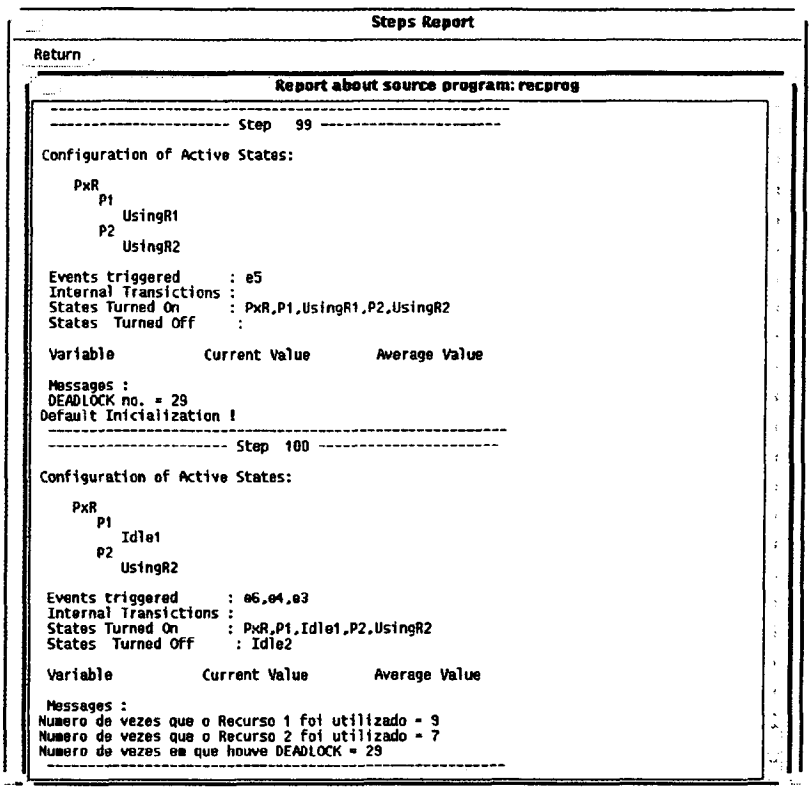


Figura 3.9 - Relatório dos passos da Simulação Programada

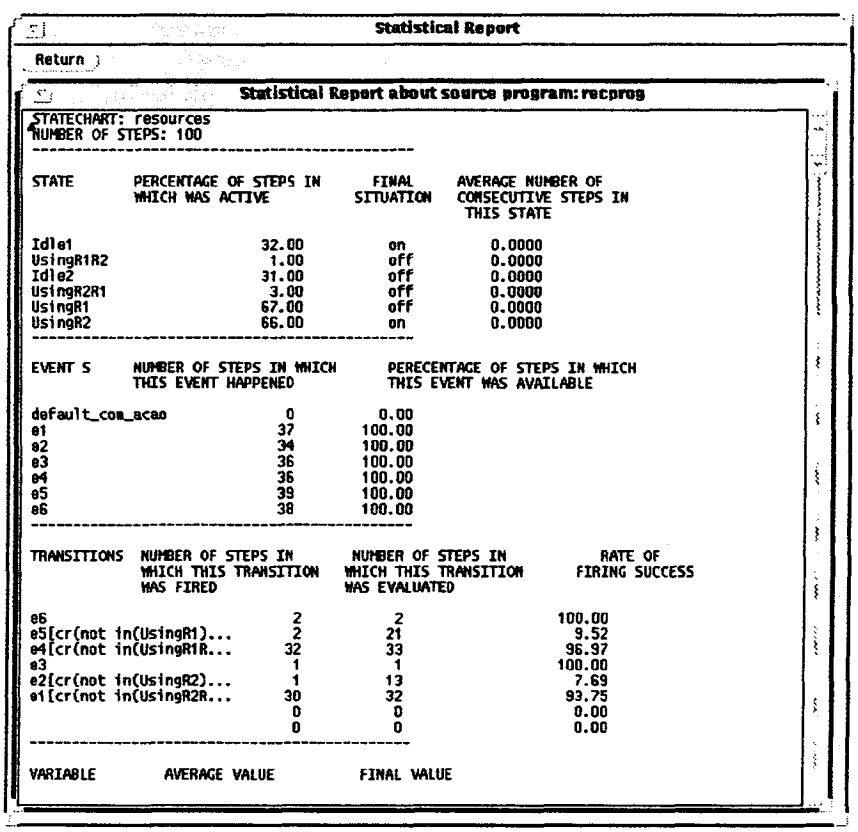


Figura 3.10 - Relatório estatístico da Simulação Programada

Simulação Exaustiva

A simulação exaustiva no ambiente StatSim é baseada na Árvore de Alcançabilidade (Masiero et al., 1994), já apresentada no Capítulo 2. Como dito anteriormente, a Árvore de Alcançabilidade permite a verificação de algumas propriedades de um statechart, como alcançabilidade, reiniciabilidade, *deadlock*, uso de transições e seqüência de eventos válida. O módulo de Simulação Exaustiva ainda não está acoplado ao ambiente StatSim, mas este acoplamento deverá ser feito de forma que o ambiente possibilite a verificação de todas as propriedades por meio da árvore de alcançabilidade, como mostra o menu apresentado na Figura 3.11.

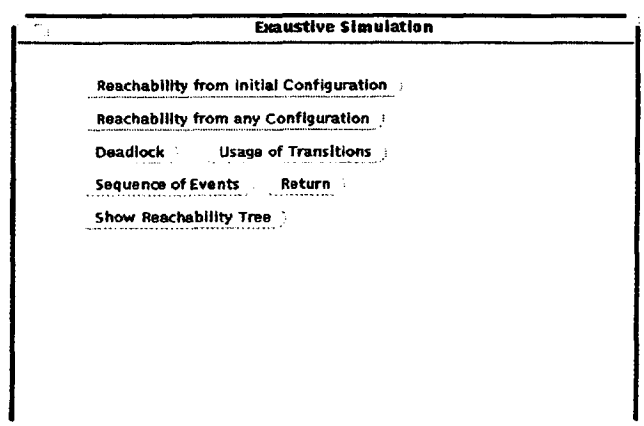


Figura 3.11 - Simulação Exaustiva no ambiente StatSim

Pelas opções da Figura 3.11, nota-se que é possível gerar a árvore de alcançabilidade de um statechart e visualizá-la como mostra a Figura 3.12. Para cada nó da árvore, pode-se saber quem é o seu pai, seu primeiro filho, seu irmão mais à direita, sua configuração correspondente e os possíveis arcos que partem dele. Por exemplo, o nó de endereço <571088> da árvore da Figura 3.12 que representa a configuração [(100)(001)], correspondente a [(UsingR1, UsingR1R2, Idle1)(UsingR2, UsingR2R1, Idle2)], tem o nó <413128> como pai, o nó <571248> como primeiro filho e seu irmão é o nó <571168>. Além disso, dois arcos partem desse nó, <e2[cr(not in(UsingR2))]> e <e4[cr(not in(UsingR1R2))]>. Analisando-se algumas informações pode-se saber também quem são as folhas da árvore, isto é, a partir de quais configurações o sistema não tem mais transições habilitadas. Por exemplo, na Figura 3.12, o nó <571488> é um nó folha pois não tem arcos partindo dele.

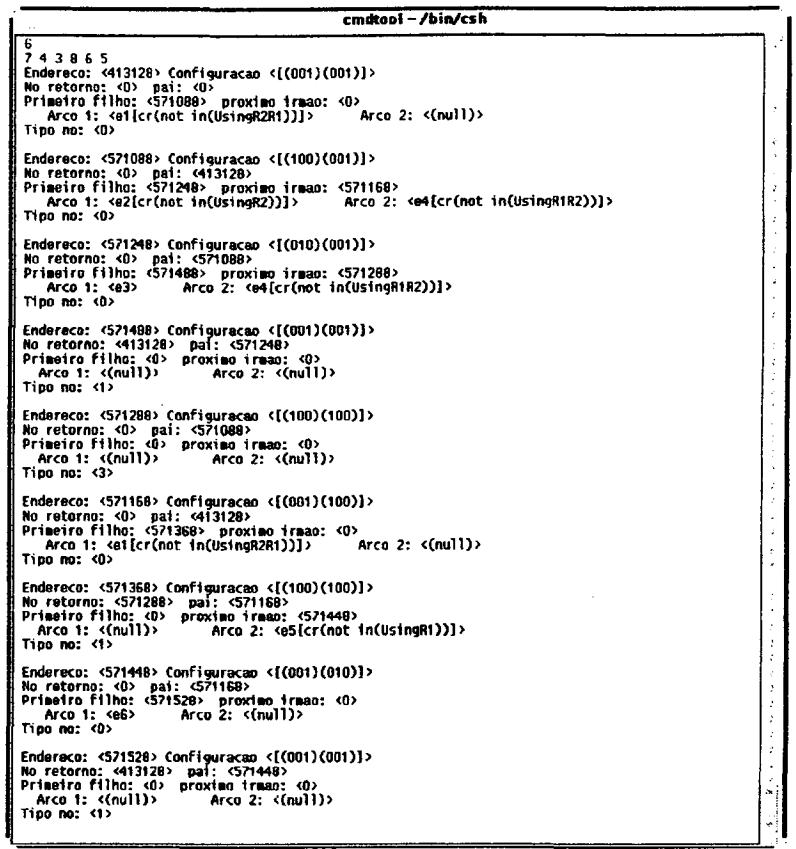


Figura 3.12 - Árvore de Alcançabilidade

O teste da alcançabilidade de uma configuração de estados a partir da configuração inicial pode ser realizado como na Figura 3.13. Nesta figura, pede-se para verificar se a configuração <(010)(001)>, em que os estados UsingR1R2 e Idle2 estão ativos, é alcançável a partir da configuração inicial. E a resposta é sim, através da seqüência de eventos: (e1[cr(not in(UsingR2R1)), e2[cr(not in(UsingR2))).

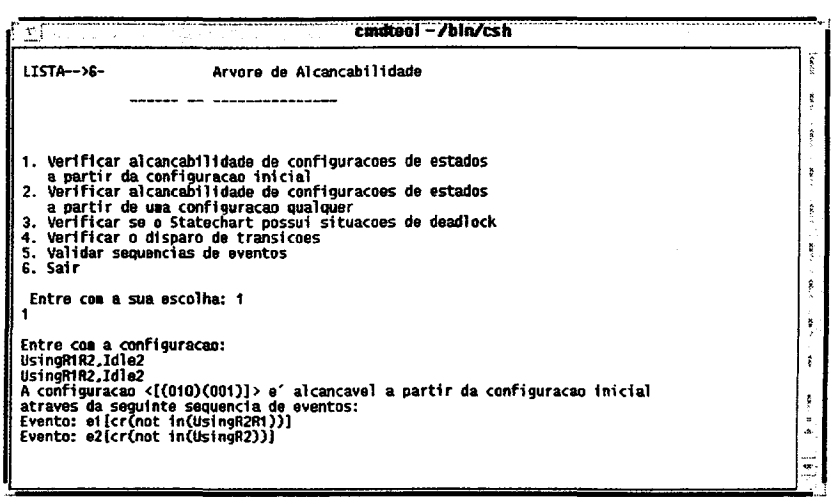


Figura 3.13 - Verificação da alcançabilidade de uma configuração de estados

Pode-se também verificar se existe *deadlock* no statechart, como ilustrado na Figura 3.14. A verificação indica que há um possível *deadlock* no statechart quando este atinge a configuração [(100)(100)], cujos estados ativos são UsingR1 e UsingR2. Este *deadlock* é atingido a partir da seguinte seqüência de eventos: (e1[cr(not in(UsingR2R1)), e4[cr(not in(UsingR1R2))).

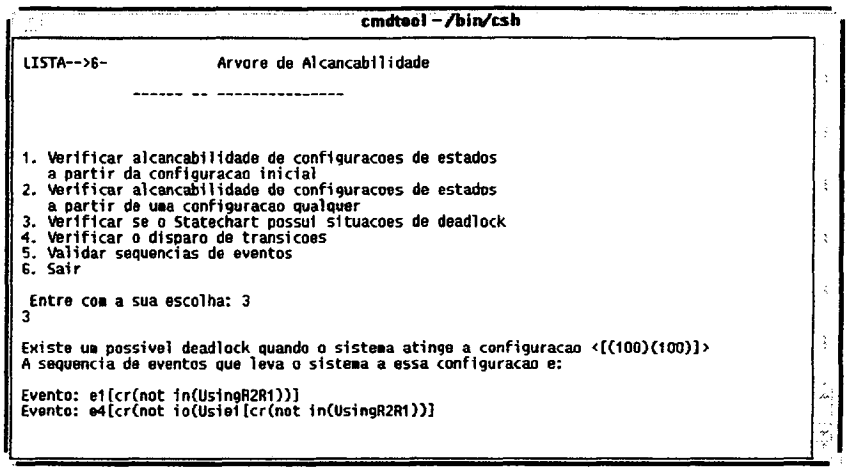


Figura 3.14 - Verificação da existência de deadlocks

3.2 Ambiente HySCharts

O ambiente HySCharts (***Hyperdocument System based on StateCharts***) foi desenvolvido pelo grupo de Engenharia de Software do Instituto de Ciências Matemáticas e de Computação da USP/São Carlos para oferecer apoio automatizado ao desenvolvimento de aplicações hipermídia, apresentando ambientes de autoria e de navegação que permitem criar e validar especificações de hiperdocumentos segundo o modelo HMBS (Turine et al., 1998). O modelo HMBS utiliza a estrutura e a semântica operacional de Statecharts na especificação da estrutura organizacional e da semântica de navegação de hiperdocumentos grandes e complexos. No HMBS, um hiperdocumento é composto por três tipos de objetos básicos: estruturais, navegacionais e de apresentação. A estrutura organizacional do hiperdocumento definida pelo statechart subjacente permite especificar os objetos estruturais, que são os estados, as transições e os eventos. Os objetos navegacionais (páginas), as ligações (ainda que não explicitamente formalizadas no modelo) e as âncoras definem a estrutura navegacional do hiperdocumento. Os canais definem os objetos relativos à apresentação, isto é, são invocados para apresentar as informações contidas nas páginas durante a navegação. Desse modo, os estados do statechart são associados às porções de informação ou páginas. Os eventos contidos nos rótulos das transições representam, respectivamente, as âncoras que disparam as possíveis ligações entre as páginas, definindo assim os caminhos de navegação disponíveis para o leitor.

Segundo o HMBS, um hiperdocumento H é definido como uma 7-tupla:

$$H = <ST, P, m, L, pl, ae, N>, \text{ na qual:}$$

a) ST representa o statechart subjacente ao hiperdocumento, definido como uma 11-tupla ST = <S, ρ, ψ, γ, δ, V, C, E, A, R, T> representando, respectivamente, estados, função de hierarquia, função tipo de decomposição, função história, função *default*, conjunto de expressões, conjunto de condições, conjunto de eventos, conjunto de ações, conjunto de rótulos e conjunto de transições. A definição de Statecharts adotada é uma simplificação da apresentada por Harel (1987a).

b) P é o conjunto finito de páginas de informação que define o conteúdo do hiperdocumento. Cada página p $\in$ P, comumente denominada nó de informação, é definida conceitualmente pela tripla <c, t, Anc_p>, tal que "c" é a porção de informação, que pode ser composta por mídias estáticas (texto, gráfico ou imagem, por exemplo) ou dinâmicas (vídeo, áudio ou animação); "t" representa o título associado à página; e "Anc_p" define uma coleção de âncoras contidas na página. O conjunto de páginas pode incluir uma página nula especial (Pλ), sem qualquer conteúdo, título e/ou âncora, a qual pode ser associada a estados que não modelam a apresentação da informação;

c) m: $S_S \rightarrow$ P denota uma função que mapeia estados de um subconjunto S_S ($S_S \subset$ S) em páginas p $\in$ P do hiperdocumento. O subconjunto "S_S" é definido por S_S: {x $\in$ S | ψ(x)=OR $\vee$ ρ(x) = $\varnothing$}, isto é, "S_S" é o conjunto formado pelos estados compostos do tipo OR e pelos estados atômicos do statechart. Estados AND não são mapeados em páginas, sendo utilizados unicamente para especificar concorrência de informações na apresentação;

d) L define o conjunto de canais de apresentação (ou leitores) invocados para interpretar e visualizar as informações contidas nas páginas durante a navegação. Permitem ao autor especificar a coordenação espacial das informações contidas nas páginas, além de controlar atributos globais da apresentação do hiperdocumento;

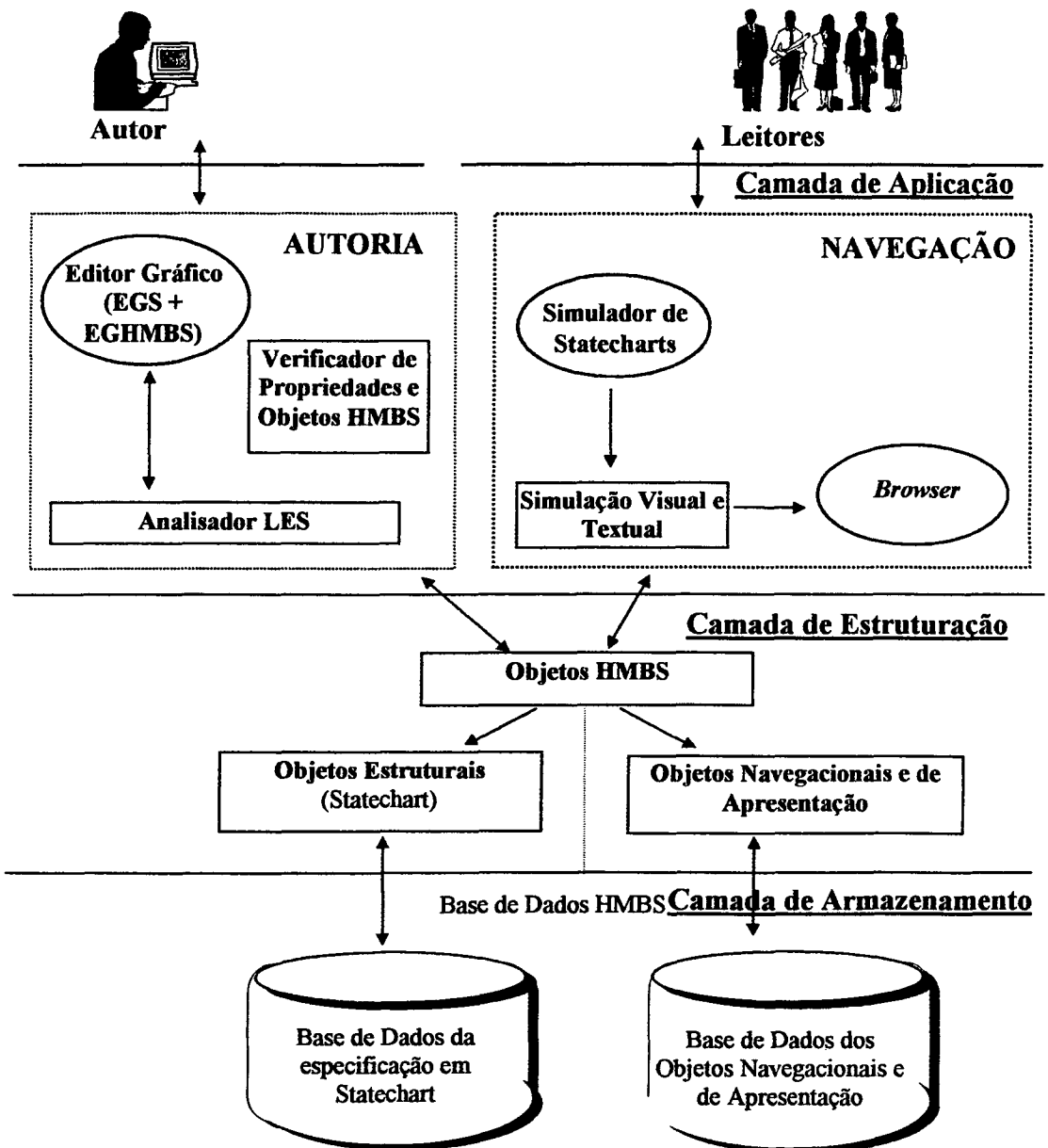
e) pl: P $\rightarrow$ L define a função de visualização, que associa cada página p $\in$ P a um único canal l $\in$ L capaz de interpretá-la;

f) ae: $Anc_p \rightarrow$ E define a função que associa elementos "a_p" ($a_p \in Anc_p$) a eventos do statechart que, por sua vez, definem as transições a serem disparadas. Segundo o HMBS, para que as âncoras possam ser associadas a um evento de uma transição é necessário que tal evento seja único no rótulo da transição. Adicionalmente, a página que contém a âncora deve ser mapeada para um estado pertencente ao conjunto de estados origem da transição especificada ou, ainda, pode ser mapeada para um subestado de algum estado pertencente a esse conjunto. As transições com seus respectivos eventos podem ser reutilizadas para definir âncoras em diferentes contextos de navegação; e

g) N (N $\geq$ 0) define o nível de visibilidade do hiperdocumento. O autor pode utilizar o valor de "N" para definir a profundidade hierárquica a ser apresentada durante a navegação.

O HySCharts foi desenvolvido como uma extensão do sistema StatSim (Masiero et al., 1991). A arquitetura do HySCharts é composta por três camadas principais, denominadas camadas de aplicação, de estruturação e de armazenamento, como pode ser observado na Figura 3.15. A camada de aplicação está subdividida nos módulos de autoria e de navegação, em adição a um ambiente de edição e simulação de Statecharts. A camada de estruturação é o núcleo do sistema, pois define a estrutura interna do hiperdocumento por intermédio dos objetos estruturais, navegacionais e de apresentação do HMBS. Todos

esses objetos são armazenados em bases de dados gerenciadas por funções da camada de armazenamento.



Legendas:
LES - Linguagem de Especificação de Statecharts
EGS - Editor Gráfico de Statecharts
EGHMBS - Editor Gráfico dos Objetos HMBS

Figura 3.15 - Arquitetura do ambiente HySCharts (Turine et al., 1998)

3.2.1 Módulo de Autoria

Durante a atividade de autoria, o autor especifica visualmente os objetos estruturais, os objetos navegacionais e de apresentação, além de analisar a estrutura navegacional. O statechart subjacente ao hiperdocumento é criado com um editor gráfico (EGS - Editor Gráfico de Statecharts), ilustrado na Figura 3.16. A estrutura organizacional apresentada na Figura 3.16 retrata a especificação de um hiperdocumento que apresenta o Parque Ecológico de São Carlos (PESC), e inclui informações gerais sobre o parque, os animais nele abrigados, descrições das atividades promovidas pelo parque, um histórico, um depoimento do diretor e dicas para os visitantes. A janela com título "*Query State: Animal*", sobreposta à janela principal na Figura 3.16, apresenta informações estruturais sobre o estado "*Animal*", um estado do tipo OR cuja decomposição é especificada em um outro nível hierárquico, não apresentado neste trabalho.

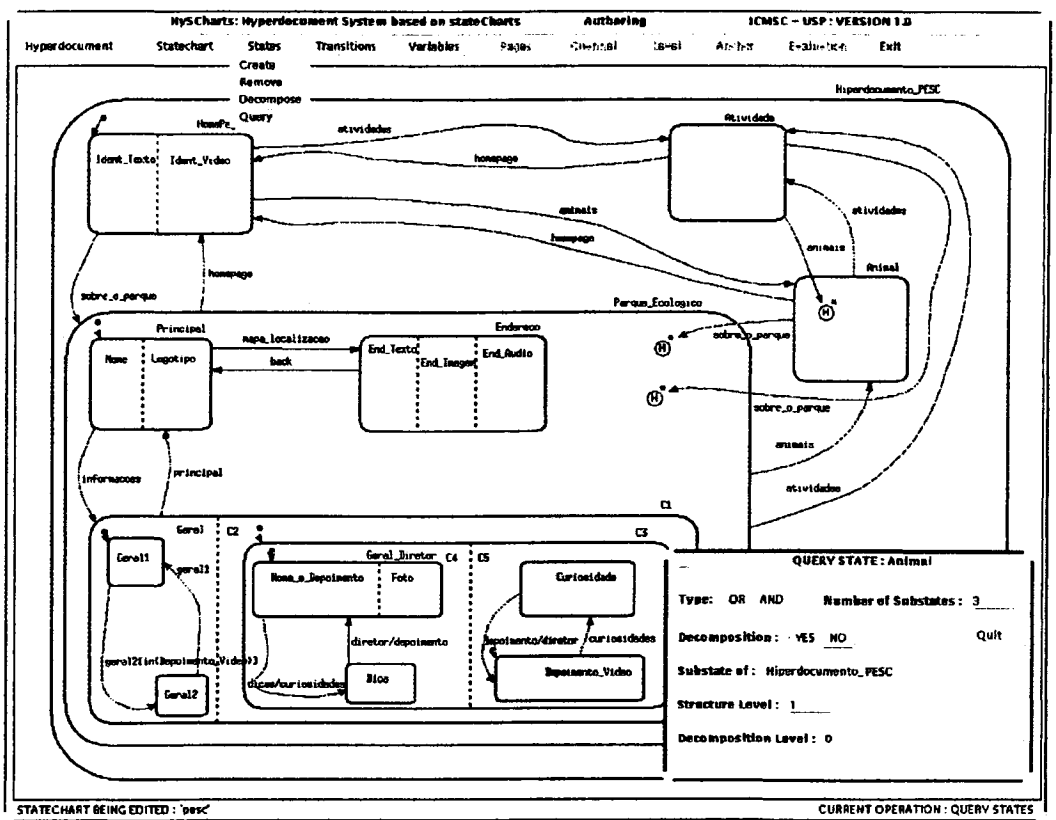


Figura 3.16 - EGS do módulo de autoria do ambiente HySCharts (Turine et al., 1998)

A janela do editor gráfico de objetos HMBS (EGHMBS), ilustrada na Figura 3.17, possui ferramentas para gerenciar páginas, âncoras e canais. Na Figura 3.17 são apresentadas três janelas relacionadas aos objetos página, com títulos "*Create Page*", "*Query Page of the State Ident_Texto*" e "*Pages Created*". A janela "*Create Page*" permite especificar uma página pelos atributos título, conteúdo, tipo da mídia e canal de apresentação que a reconhece. O conteúdo da página é identificado pelo nome de um arquivo ("*File*"). Para interpretar e visualizar o conteúdo de uma página é necessário associá-la a um único canal de apresentação, também especificado pelo autor. Basicamente, os canais definem o tipo de mídia (texto, imagem, áudio e vídeo) associado à página e as características de visualização. Após a especificação dessas informações, o autor pode associar tal página ao objeto estrutural (estado) correspondente, criando uma

instância do mapeamento "m" entre estados e páginas.

A Figura 3.17 também ilustra a operação de consulta às páginas associadas a estados. Ao selecionar um estado, caso este tenha uma página associada, as informações relativas à página e ao estado são apresentadas. Por exemplo, na janela "*Query Page of the State Ident_Texto*" são apresentadas as informações relativas ao estado "*Ident_Texto*" (subestado do estado "*HomePage*") e à página associada. A janela "*Pages Created*" contém informações (título, tipo da mídia e nome do estado associado) sobre todas as páginas criadas para o hiperdocumento instanciado.

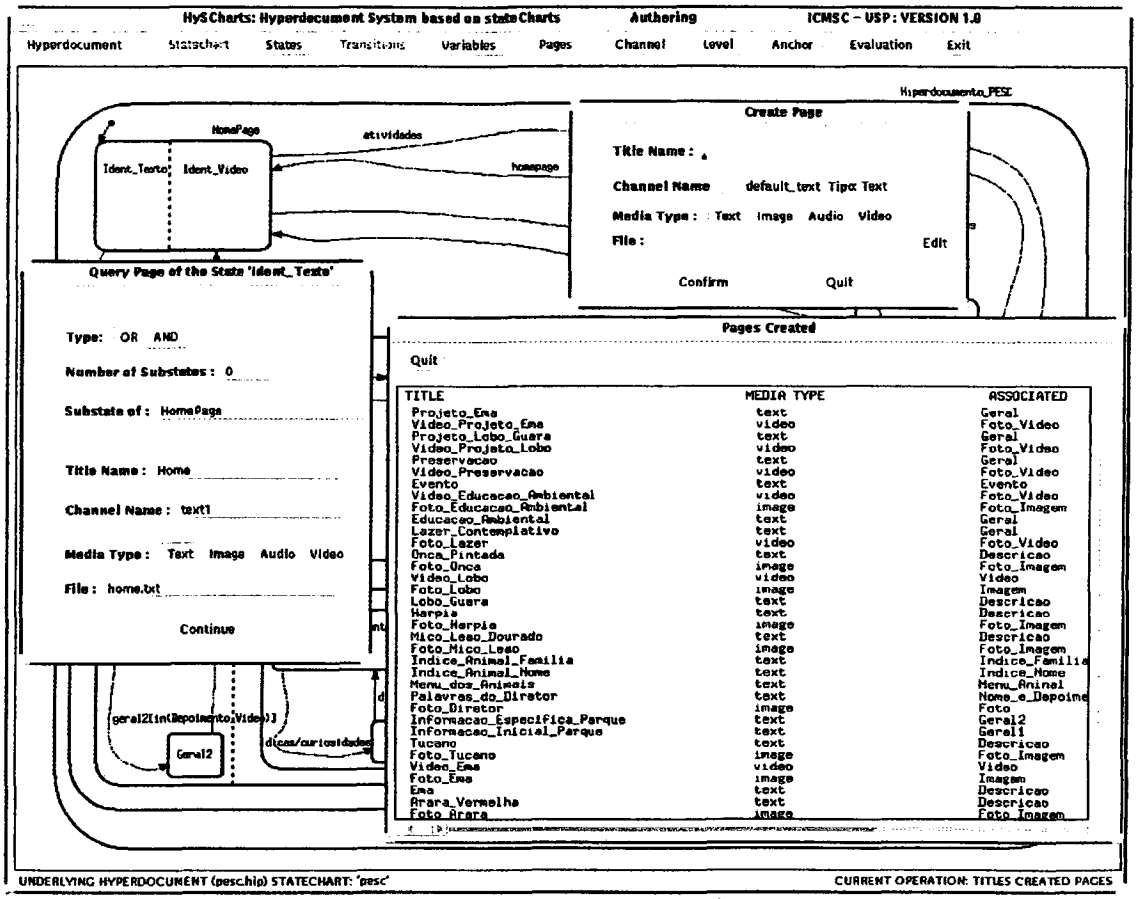


Figura 3.17 - Janelas relacionadas ao gerenciamento dos objetos página (Turine et al., 1998)

Tendo criado as páginas e associado-as aos estados correspondentes, o autor pode especificar os objetos navegacionais âncora que habilitam as ligações do hiperdocumento. Durante a navegação, as âncoras pertencentes às páginas do tipo texto são visualmente apresentadas sublinhadas e em negrito. Nas demais páginas (de tipo imagem, vídeo e áudio) as âncoras são apresentadas na forma de botões. Para criar um objeto âncora pertencente a uma página, o autor escolhe a palavra-chave desejada e, em seguida, seleciona o evento associado pertencente ao rótulo de uma transição. Dessa maneira, uma instância do relacionamento "ae" é criada e os objetos âncora e ligação são definidos.

Na Figura 3.18, o conteúdo da página texto "P_{Ident_Texto}" associado ao estado "*Ident_Texto*" é exibido na janela com o título "*Viewer of the page with title Home and state Ident_Texto*". Também é ilustrada na Figura 3.18 a janela "*Query Anchor-> Event*", que apresenta o resultado da operação de consulta sobre a âncora "Sobre_o_Parque" da página "P_{Ident_Texto}". Tal âncora está associada ao evento rotulado "sobre_o_parque", cuja

transição tem como nome de estado origem "*HomePage*" e como destino "Parque_Ecológico".

A camada de estruturação armazena também o objeto canal, uma abstração de um dispositivo que inclui atributos de apresentação de alto nível (por exemplo, o estilo e o tamanho da fonte, além da cor de fundo da janela para mídias tipo texto) e mecanismos de coordenação espacial das páginas (largura e altura das janelas apresentadas). O sistema HySCharts trata esses elementos de especificação segundo uma abordagem orientada a objetos, no sentido de que existem classes de canais *default* que podem ser instanciadas, conforme as necessidades do autor, usando os mecanismos de herança e especialização/generalização. Por exemplo, pode-se ter dois canais diferentes do tipo texto: um utilizando a fonte "*Times New Roman*" e o outro "*Arial*". O uso de canais permite que um mesmo documento seja apresentado de diferentes maneiras, reespecificando os atributos de apresentação ao invés de modificar a estrutura navegacional.

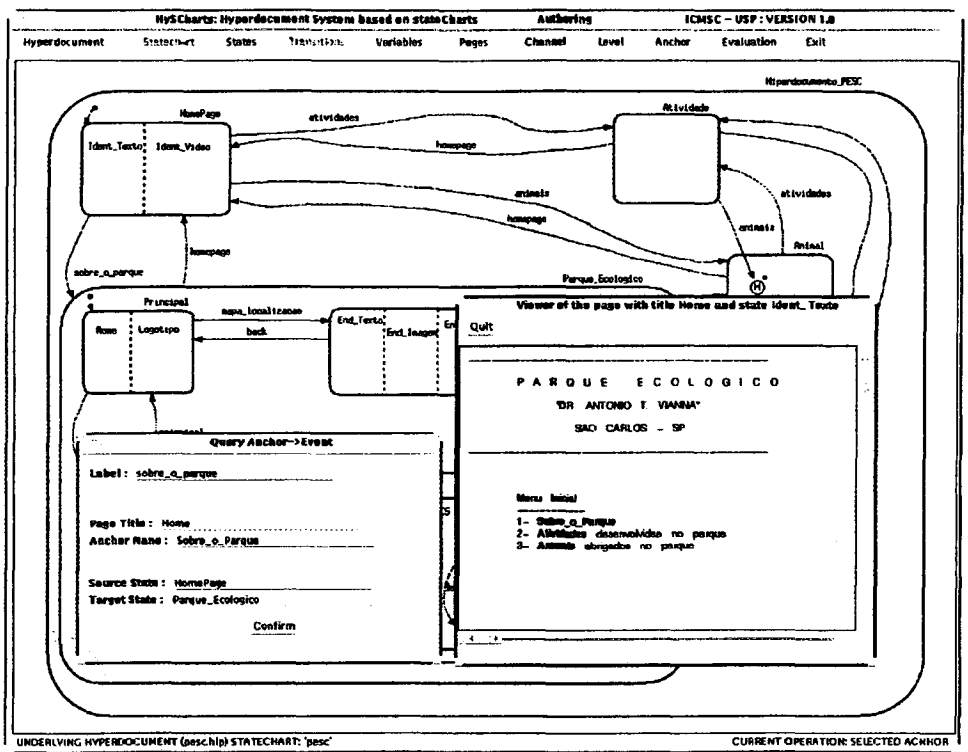


Figura 3.18 - Janelas que gerenciam os objetos âncora (Turine et al., 1998)

O módulo de autoria também oferece ferramentas para análise e validação das especificações de hiperdocumentos em HMBS (módulo "Verificador de Propriedades e Objetos HMBS" da Figura 3.15). O HMBS permite identificar inconsistências estruturais e problemas relacionados à visualização e navegação, bem como a análise de algumas propriedades. Essas análises são baseadas essencialmente na geração da árvore de alcançabilidade, que fornece subsídios para a detecção de anomalias na especificação. Algumas propriedades dinâmicas de Statecharts foram definidas e introduzidas no contexto de hiperdocumentos: alcançabilidade de uma página a partir de uma configuração de contexto qualquer, reiniciabilidade, *deadlock* durante a navegação, vivacidade de ligações navegacionais e seqüência válida de âncoras.

3.2.2 Módulo de Navegação

Para tornar disponível aos leitores o conteúdo do hiperdocumento, as especificações em HMBS devem ser interpretadas e "executadas" no módulo de navegação segundo a semântica de navegação do modelo. O HySCharts dispõe de uma máquina de Statecharts associada ao *Browser* (Figura 3.15) que recebe os eventos externos (ações do leitor) e os interpreta, modificando a apresentação.

A navegação pode ser iniciada pela configuração de contexto inicial "CC_0" ou pela escolha de um título associado a uma página específica (acesso direto à informação pelos títulos das páginas). Seguindo a primeira forma de navegação, a Figura 3.19 retrata o leiaute que corresponde à configuração de contexto $CC_0=\{P_{Ident\_Texto}, P_{Ident\_Video}\}$ do hiperdocumento especificado na Figura 3.16, considerando N = 0. O leitor visualiza duas janelas lógicas: uma janela do tipo texto (título "*State: Ident_Texto Title: Home*") e outra do tipo vídeo (título "*State: Ident_Video*"), à qual tem sempre associada uma janela de comandos de controle. A janela de título "*Show_Hview*" sempre é apresentada durante a navegação. A janela texto associada à página "P_{Ident_Texto}" contém as seguintes âncoras, sublinhadas e em negrito no seu conteúdo: "Sobre_o_Parque", "Atividades" e "Animais".

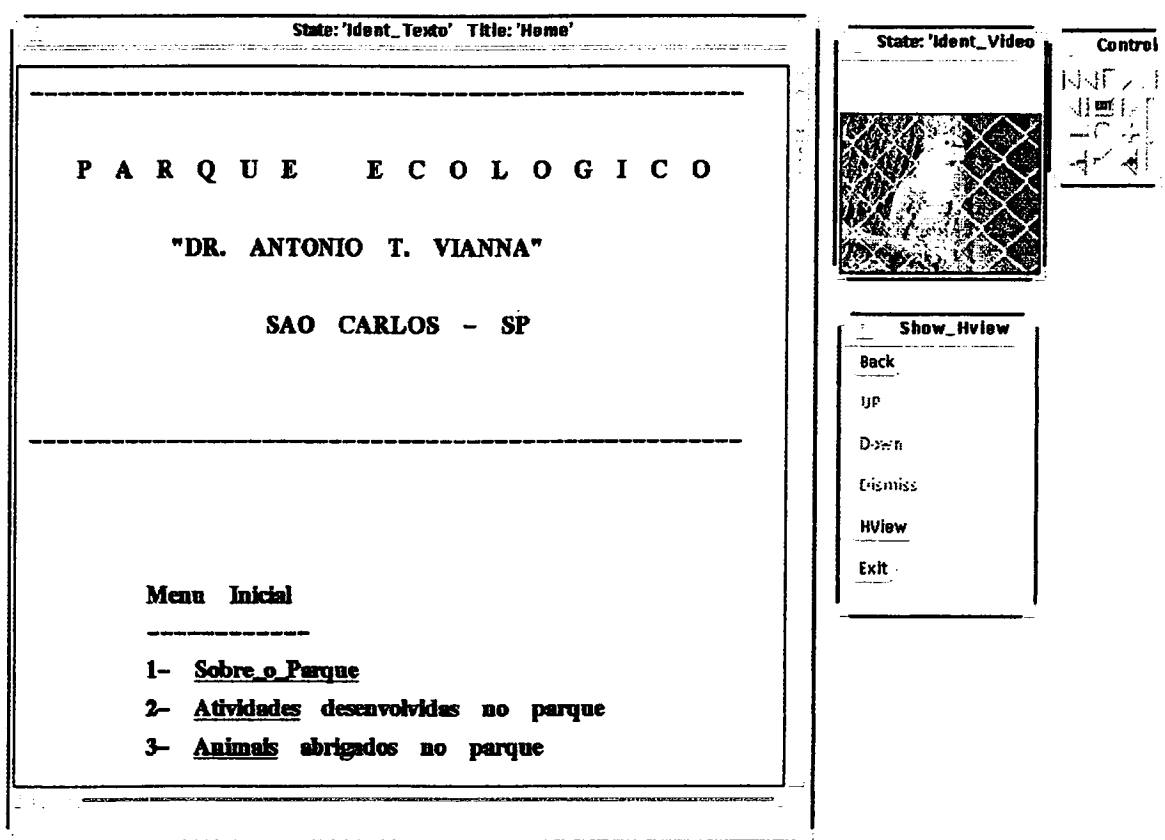


Figura 3.19 - Janela de navegação considerando o Statechart da Figura 3.16, N=0 e $CC_0=\{P_{Ident\_Texto}, P_{Ident\_Video}\}$ (Turine et al., 1998)

Mais informações sobre o ambiente HySCharts podem ser encontradas em (Turine et al., 1998). No Capítulo 4, apresenta-se o critério Análise de Mutantes e sua aplicação no contexto de teste de especificações de Sistemas Reativos, mais especificamente no teste de especificações baseadas em Statecharts. Apresenta-se também a ferramenta Proteum-RS/ST (Fabbri, 1996; Sugeta, 1999; Sugeta et al., 1999), que apóia a aplicação desse critério no teste de especificações Statecharts.

CAPÍTULO 4

ANÁLISE DE MUTANTES NO TESTE DE ESPECIFICAÇÕES STATECHARTS

O critério Análise de Mutantes (DeMillo et al., 1978) visa a testar a corretitude de um programa através da geração de programas mutantes, que são programas criados com pequenas alterações sintáticas, e da construção de casos de teste capazes de diferenciar o comportamento do programa original e dos seus mutantes. As alterações sintáticas feitas no programa para gerar os mutantes são baseadas em um conjunto de operadores, os operadores de mutação. Cada operador pode ter associado um tipo ou classe de erros que se pretende revelar no programa.

A Análise de Mutantes fornece uma medida do nível de confiança na adequação dos casos de teste utilizados. É possível avaliar, depois da execução dos mutantes, a adequação dos casos de teste através do *escore de mutação*, que relaciona o número de mutantes gerados com o número de mutantes mortos (aqueles que produzem resultados diferentes do programa original) fornecendo, dessa maneira, uma medida da confiabilidade do programa testado. Considerando P, o programa sendo testado e T, o conjunto de casos de teste, o escore de mutação $ms(P,T)$ é definido:

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

sendo,

DM(P,T): número de mutantes mortos pelos casos de teste em T
M(P): número total de mutantes gerados
EM(P): número de mutantes gerados equivalentes a P

Os mutantes são considerados equivalentes quando executam as mesmas funções de P, e apresentam um resultado idêntico a P quando executados com um mesmo conjunto de casos de teste. A determinação desses mutantes é obtida com a intervenção do testador, que decide se o mutante é realmente equivalente, ou então se decide, quando possível, através de heurísticas por um processo automatizado. E, se por acaso, a alteração realizada para gerar o mutante causar um erro sintático no programa mutante, então este mutante é considerado anômalo.

A Análise de Mutantes caracteriza-se por quatro etapas principais (DeMillo, 1980):

1. Execução do programa original com base em um conjunto de casos de teste
2. Geração de mutantes
3. Execução dos mutantes
4. Análise dos mutantes

Uma ressalva quanto à aplicabilidade deste critério é quanto ao número de mutantes gerados. Mesmo em programas pequenos esse número pode ser proibitivo, uma vez que todos os mutantes devem ser compilados, executados e analisados pelo testador para que se

decida sobre a equivalência dos que permaneceram vivos, isto é, aqueles cujas saídas não diferenciam das saídas do programa original. Duas alternativas foram propostas por Wong et al. (1994) para a aplicação da Análise de Mutantes no teste de programas: Mutação Aleatória e Mutação Restrita.

Como já ressaltado, a especificação e o projeto de Sistemas Reativos grandes e complexos são considerados problemáticos no desenvolvimento dos mesmos. A dificuldade se encontra na descrição de seu comportamento de maneira clara, formal e rigorosa. Este formalismo permite que as especificações sejam simuladas e testadas. Atividades de teste e validação de Sistemas Reativos são aspectos relevantes que devem ser tratados com rigor e critério, uma vez que o desenvolvimento desses sistemas deve ser realizado de modo que se garanta produtos de alta qualidade. As formas mais comuns de testar especificações dos aspectos comportamentais desses sistemas são os vários tipos de simulação, que permitem ao usuário avaliar a especificação em relação ao comportamento esperado do sistema.

Considerando-se especificações baseadas em Statecharts, as formas de validação utilizadas mais freqüentemente são também os vários tipos de simulação, sendo que ainda não existem critérios para seleção e/ou avaliação de casos de teste. Os métodos de geração de seqüências de teste existentes para MEFs, com a devida adequação, podem ser aplicados somente nos componentes do tipo OR dos statecharts, que correspondem a MEFs. Além disso, Statecharts possuem alguns aspectos adicionais que tornam seus modelos mais complexos, pois envolvem paralelismo, concorrência e história. Uma técnica de análise de Statecharts é a árvore de alcançabilidade, proposta por Masiero et al. (1994) e já discutida nos capítulos anteriores.

No contexto das técnicas MEFs e Statecharts, segundo Fabbri (1996), a Análise de Mutantes pode facilitar a atividade de teste, pois esse critério é baseado em um modelo de erros que pode ser definido de forma adequada para retratar tipos de erros que explorem todas as características das duas técnicas. E, devido à flexibilidade do critério Análise de Mutantes, embora este seja um critério de teste desenvolvido para o teste de programas, sua aplicação em diferentes contextos pode ser uma alternativa bastante viável, mas que requer ferramentas que apóiem essa aplicação.

A exemplo do que foi proposto por Wong et al. (1994) para solucionar o problema do número de mutantes gerados, Fabbri (1996) propõe duas alternativas para a aplicação desse critério no teste de especificações: Mutação Aleatória e Mutação Restrita. Na Mutação Aleatória, examina-se uma pequena porcentagem de mutantes selecionados aleatoriamente de cada tipo de mutante, e o restante é ignorado. Já na Mutação Restrita, somente alguns tipos específicos de mutantes são examinados. Neste caso, Fabbri ressalta que a seleção dos operadores de mutação é um ponto crucial e pode levar a uma redução no custo de execução, sem sacrificar a força e a capacidade de detecção de falhas do teste de mutação, como observado em (Wong et al., 1994). Fabbri aplicou os critérios de Mutação Aleatória e Mutação Restrita com o objetivo de avaliar o uso dessas duas alternativas na validação de especificações baseadas em MEFs e Redes de Petri. Os resultados obtidos mostraram uma redução significativa no custo, sem se perder muito na eficácia do critério Análise de Mutantes, indicando que o uso desse critério pode ser uma alternativa também no teste de especificações.

De acordo com Fabbri (1996), é importante o desenvolvimento de ferramentas que apóiem a atividade de teste, pois isso permite aumentar a qualidade dessa atividade através da sua automatização e, como conseqüência, a qualidade do produto sendo desenvolvido. Particularmente, com relação ao critério Análise de Mutantes, como relatado em (Fabbri, 1996), sua aplicação de forma manual, como foi feito para Máquinas de Estados Finitos e Redes de Petri, a título de experimento piloto, é inviável. O número de mutantes gerado é

muito grande, o que consome muito tempo além dos vários erros que podem ocorrer durante a geração e execução dos mesmos. Assim, Fabbri especificou a ferramenta Proteum-RS que apóia o teste de especificações de Sistemas Reativos baseado no critério Análise de Mutantes. Essa ferramenta pode ser instanciada para apoiar o teste de especificações realizadas em Máquinas de Estados Finitos, Statecharts e Redes de Petri, Proteum-RS/FSM (Fabbri, 1996), Proteum-RS/ST (Fabbri, 1996; Sugeta, 1999; Sugeta et al., 1999) e Proteum-RS/PN (Simão, 2000), respectivamente.

A seguir, apresenta-se a Proteum-RS/ST, que apóia o teste de especificações Statecharts baseada na Análise de Mutantes.

4.1 Proteum-RS/ST

De acordo com (Delamaro & Maldonado, 1996a), uma ferramenta de teste baseada em mutação deve fornecer um conjunto mínimo de operações:

- *manipulação de casos de teste*, possibilitando a execução, inclusão/exclusão e desabilitação/habilitação dos casos de teste;
- *manipulação de mutantes*, permitindo a criação, seleção, execução e análise dos mutantes;
- *análise de adequação*, permitindo calcular o escore de mutação e fornecer relatórios estatísticos.

A ferramenta Proteum-RS/ST (Fabbri, 1996; Sugeta, 1999) satisfaz esse conjunto mínimo de requisitos, apoiando as atividades básicas para a aplicação do critério Análise de Mutantes e permitindo ao testador a execução das seguintes tarefas:

- definição de casos de teste;
- simulação da especificação em teste;
- geração de mutantes;
- execução dos mutantes;
- análise dos mutantes vivos;
- cálculo do escore de mutação.

A Proteum-RS/ST tem a mesma estrutura da Proteum-RS/FSM (Fabbri, 1996), que apóia o teste de especificações baseadas em Máquinas de Estados Finitos pelo critério Análise de Mutantes. No desenvolvimento da Proteum-RS/FSM, os módulos do ambiente StatSim (Masiero et al., 1991), que apóiam a edição e simulação de Statecharts fornecidos na forma textual, através da Linguagem de Especificação de Statecharts (LES), foram utilizados. É ressaltado ainda que a utilização do analisador e do simulador no contexto de Máquinas de Estados Finitos não exigiu nenhuma alteração na funcionalidade desses módulos. Eles apenas foram usados de forma restrita, pois para especificar uma MEF através da LES, é suficiente utilizar-se apenas um subconjunto dessa linguagem, uma vez que a técnica Statecharts é uma extensão das MEFs.

A estrutura básica da Proteum/C (Delamaro, 1993), que apóia o teste de programas C, também foi reutilizada no desenvolvimento da Proteum-RS/FSM, fazendo-se as adequações necessárias para possibilitar o teste de Máquinas de Estados Finitos, pois a seqüência de atividades que caracterizam o critério Análise de Mutantes é basicamente a mesma nas duas ferramentas. A diferença fundamental reside na definição de casos de teste e na geração de mutantes, uma vez que a ferramenta Proteum-RS/FSM atende à validação de especificações de Sistemas Reativos baseadas em Máquinas de Estados Finitos.

A ferramenta Proteum-RS/ST mantém uma base de dados com informação sobre casos de teste e sobre os mutantes e seus estados. O usuário atua sobre essa base de dados editando o conjunto de casos de teste a ser utilizado, gerando os mutantes, executando-os, eliminando os mutantes equivalentes e também solicitando relatórios sobre o estado do teste que está sendo conduzido. A Proteum-RS/ST foi desenvolvida de forma a permitir o trabalho com sessões de teste, possibilitando que o usuário inicie o teste de uma especificação, encerre a atividade quando desejado e depois a retome do ponto onde havia parado. Para isso, os estados intermediários da aplicação do teste são guardados na base de dados para que possam ser recuperados posteriormente.

A edição do statechart é realizada gerando-se uma descrição do mesmo através da LES. Essa tarefa deve ser realizada utilizando-se um editor de texto disponível no sistema, através do qual cria-se o arquivo com a especificação statechart. Esse arquivo é então utilizado na ferramenta, para a realização dos testes.

A Figura 4.1 ilustra a janela principal da ferramenta. Os aspectos operacionais e funcionais da Proteum-RS/ST podem ser encontrados em (Sugeta et al., 1999).

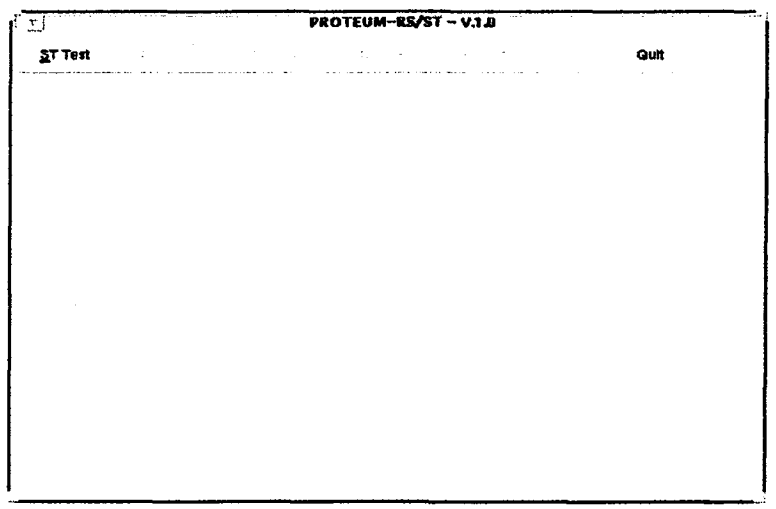


Figura 4.1 - Tela Principal da Ferramenta Proteum-RS/ST

Um dos pontos fundamentais para a aplicação do critério Análise de Mutantes é a definição de um conjunto de operadores de mutação. A geração dos mutantes é feita com base no conjunto de operadores de mutação definidos para Statecharts e apresentados na Tabela 4.1. Essa geração pode ser restrita escolhendo-se apenas alguns tipos de operadores a serem aplicados, como pode também ser aleatória, escolhendo-se apenas uma porcentagem de mutantes a serem gerados de cada um dos tipos de operadores escolhidos.

No contexto de Statecharts é também relevante o fornecimento de subsídios para lidar com conceitos de hierarquia e paralelismo, inerentes a esta técnica. Assim, Fabbri (1996) especifica em seu trabalho estratégias básicas para abstração das máquinas de estado subjacentes, fornecendo subsídios para o estabelecimento de estratégias de teste incremental neste contexto.

As três estratégias de abstração para a técnica Statecharts propostas por Fabbri (1996) são baseadas no aspecto de decomposição dessa técnica, pois através da abstração do statechart, selecionam-se, por nível de decomposição, os componentes pertencentes àquele nível. As estratégias, denominadas *Básica*, *Baseada em Ortogonalidade* e *Baseada em História*, diferenciam-se, essencialmente, no aspecto que exploram na atividade de

teste. No entanto, todas elas aplicam basicamente o mesmo procedimento para abstrair os componentes de cada nível do statechart.

Tabela 4.1 - Operadores de Mutação para Statecharts que Abordam Aspectos Relacionados a Máquinas de Estados Finitos, a Máquinas de Estados Finitos Estendidas e Aspectos Intrínsecos à Técnica Statecharts

FSM		EFSM	
Operador	*Descrição*	*Operador*	*Descrição*
TraDefStaAlt	Altera estado *default*	TraExpDel	Exclusão de expressão
TraArcDel	Arco faltando	TraNegBoolExp	Negação de expressão booleana
TraEveDel	Evento faltando	TraAssRelAlt	Troca associatividade dos termos
TraEveIns	Evento extra	TraAritOperArit	Troca op. Aritmético por op. Aritmético
TraEveAlt	Evento trocado	TraRelOperRel	Troca op. relacional por op. relacional
TraDesStaAlt	Destino trocado	TraLogOperLog	Troca op. lógico por op. lógico
TraActDel	Ação faltando	TraNegLogExp	Negação Lógica
TraActAlt	Ação trocada	TraVarAltVar	Troca variável por variável
		TraConsAlt	Troca de constantes por constantes requeridas
		TraConsAltVar	Troca constante por variável escalar

Aspectos Intrínsecos de Statecharts	
Operador	*Descrição*
HistDelTra	Desassocia história da transição
HistTraAltTra	Troca transição associada ao símbolo de história
HistDelSta	Exclui história do estado
HistHH*	Troca h por h*
HistH*H	Troca h* por h
HisInsHSta	Associa h ao estado
HisInsH*Sta	Associa h* ao estado
TraCondInDel	Exclui condição in(s)
TraCondInStaAlt	Troca estado da condição in(s)
TraCondNYDel	Exclui condição not-yet(e)
TraCondNYEveAlt	Troca evento da condição not-yet(e)
TraEveExDel	Exclui evento exit(s)
TraEveExStaAlt	Troca estado do evento exit(s)
TraEveEnDel	Exclui evento entered(s)
TraEveEnStaAlt	Troca estado do evento entered(s)
TraBrOrigAlt	Altera transição origem do broadcasting
TraBrDestAlt	Altera transição destino do broadcasting

A Estratégia Básica abstrai os componentes do tipo OR que correspondem essencialmente à estrutura básica de composição dos statecharts que é a Máquina de Estados Finitos. Esses componentes podem ser vistos como Máquinas de Estados Finitos Estendidas. O objetivo principal dessa estratégia é testar os componentes OR, abordando também aspectos de integração entre eles. Ela pode ser considerada como uma estratégia mais geral, através da qual não se exploram outros aspectos intrínsecos à técnica Statecharts, além da hierarquia.

A Estratégia Baseada em Ortogonalidade explora, na atividade de teste de um statechart, o aspecto de paralelismo. Nesta estratégia são selecionados como alvos de mutação, em cada nível de abstração do statechart, apenas os componentes tipo AND que compõem a especificação.

A Estratégia Baseada em História explora a característica de história, que é a capacidade da técnica Statecharts de memorizar a última configuração atingida por um estado. São selecionados como alvos de mutação para esta estratégia apenas os componentes (as MEFs Estendidas) que possuem símbolos de história associados.

Definidos esses componentes, o testador pode dar ênfase ao aspecto de história, para cada nível de abstração do statechart.

Identificados os componentes através das estratégias, podem-se aplicar a eles os operadores de mutação definidos para Statecharts, escolhendo-se aqueles que melhor explorem os aspectos abstraídos pelas estratégias e que melhor caracterizem os erros que se deseja explorar na especificação em teste. As estratégias de abstração e os operadores de mutação podem ser combinados de modo a definir estratégias de teste que conduzam o teste explorando as características de Statecharts de acordo com a vontade ou necessidade do testador.

Podem-se gerar mutantes aplicando os operadores de mutação apenas em um nível do statechart. No caso da Figura 4.2, foram gerados mutantes com a aplicação dos operadores nos componentes do nível 4 abstraídos pela Estratégia Básica do statechart do relógio digital da Figura 2.4, apresentada no Capítulo 2.

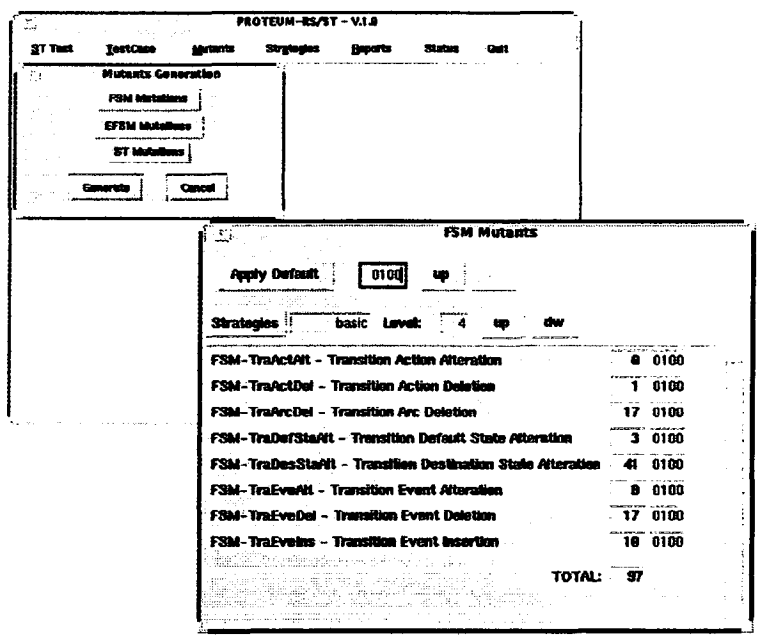


Figura 4.2 - Geração de Mutantes

Para aplicar os operadores em algum nível do statechart abstraído por uma das estratégias, é necessário selecionar uma estratégia, dependendo de qual aspecto (MEFs, história ou ortogonalidade) interessa ao testador ao aplicar tais operadores. Assim, a estratégia pode ser escolhida no botão *Strategies* da janela de geração de mutantes, sendo que uma estratégia só poderá ser utilizada se já tiver sido aplicada. Para se aplicar uma estratégia, basta selecionar o botão *Strategies* do menu principal e escolher qual estratégia se deseja aplicar. Após aplicada a estratégia, os estados de cada nível hierárquico do statechart em teste podem ser visualizados pela opção *View* do botão *Strategies*, como na Figura 4.3, que apresenta os componentes do nível 4 do statechart identificados pela Estratégia Básica.

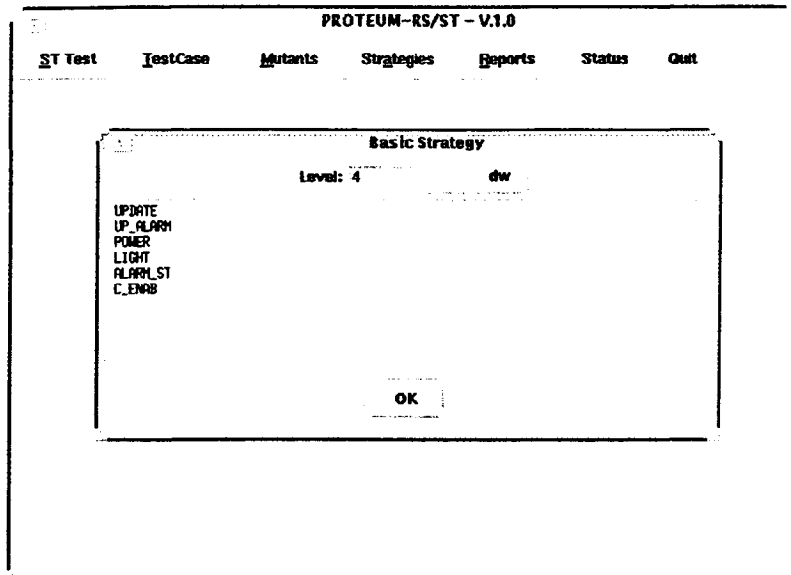


Figura 4.3 - Visualização dos Estados de um Nível Hierárquico

Todos os mutantes gerados são simulados com os casos de teste que estão habilitados no momento. Todas as informações produzidas após a execução dos mutantes devem ser armazenadas, como o estado do mutante (vivo ou morto), o caso de teste que o matou, etc. Realizada a simulação do mutante, compara-se seus estados finais ativos aos estados finais ativos do statechart original. Feita a comparação, o mutante é marcado como morto, caso os estados finais ativos sejam distintos, ou como vivo, caso contrário. Se por acaso, a alteração realizada para gerar o mutante causar um erro sintático na especificação mutante, então este mutante é considerado anômalo.

O estado da sessão de teste pode ser verificado através do botão *Status* do menu principal. Como pode ser observado na Figura 4.4, são fornecidas informações como o número de mutantes gerados, o número de mutantes equivalentes e anômalos, o número de mutantes vivos, o número de casos de teste e o escore de mutação.

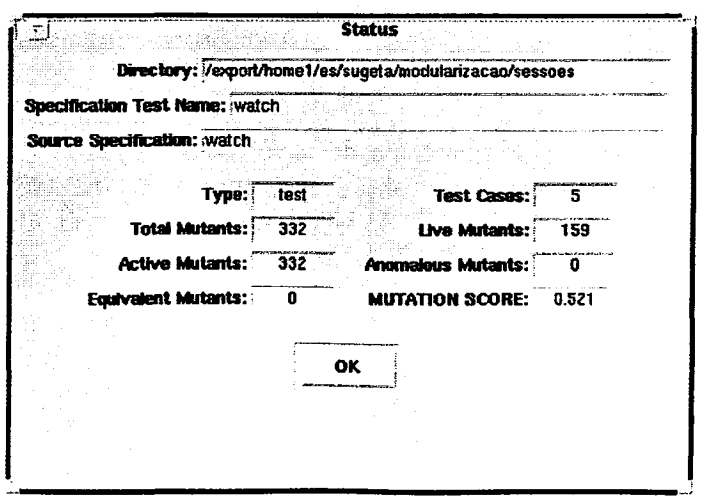


Figura 4.4 - Status da Sessão de Teste

O valor do escore de mutação varia de 0 a 1. O objetivo é conseguir casos de teste que obtenham um escore o mais próximo possível de 1, ou seja, que faça com que o número de mutantes vivos seja próximo de 1. Neste caso, o escore de mutação é 0,521, sendo que dos 332 mutantes gerados e executados, restaram 159 vivos depois da execução com o conjunto de 5 casos de teste inicial. Avaliando o escore de mutação, deve-se julgar se o teste pode ser finalizado, considerando que o conjunto de casos de teste é um bom conjunto para a especificação sendo testada. Caso contrário, pode-se então analisar os mutantes vivos e tentar encontrar casos de teste que os matem ou marcá-los como equivalentes.

Os mutantes podem ser visualizados e analisados através da opção *View* do botão *Mutants*. Uma janela é apresentada, como na Figura 4.5, e mostra a situação do mutante após a execução com o conjunto de casos de teste, o operador que o gerou e também a LES do statechart original e do mutante, de forma que o local onde foi feita a mutação pode ser visualizado. O mutante mostrado na Figura 4.5 é o de número 32, foi gerado pelo operador *TraCondInDel* e continua vivo. Além disso, o usuário pode escolher quais tipos de mutantes deseja visualizar, sejam eles vivos, mortos, equivalentes, anômalos ou inativos. No caso da Figura 4.5, apenas os mutantes vivos serão visualizados. Pode-se também marcar o mutante como anômalo ou equivalente nesta janela.

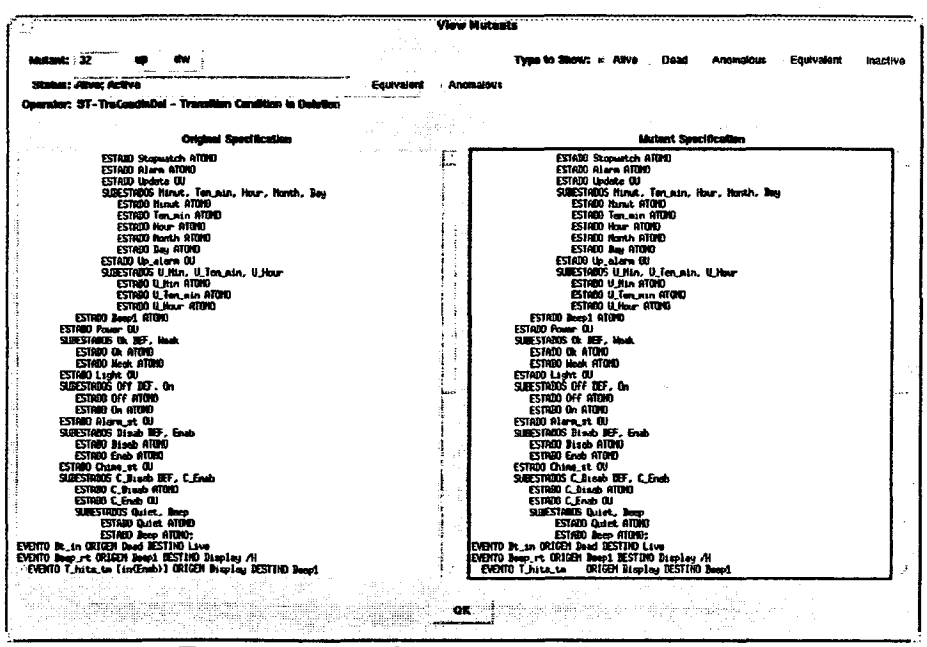


Figura 4.5 - Visualização de um Mutante

Também podem-se gerar relatórios de casos de teste, através do botão *Reports*. Uma janela como a mostrada na Figura 4.6 é apresentada e o usuário deve selecionar as informações que deseja ver no relatório. Se nenhum nome for atribuído no campo *Report Name*, o relatório terá o nome da sessão de teste.

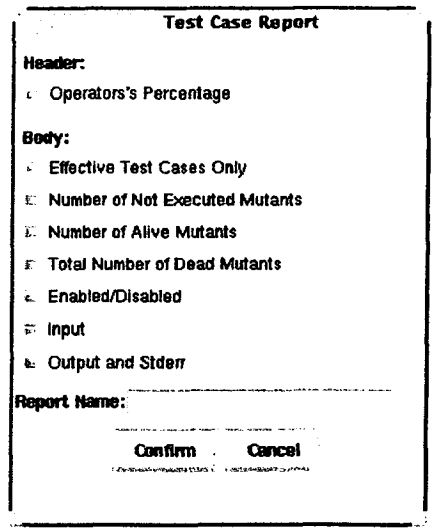


Figura 4.6 - Gerar Relatório sobre os Casos de Teste

Assim, a sessão de teste pode continuar sendo conduzida, com a adição de novos casos de teste, ou não.

CAPÍTULO 5

CONSIDERAÇÕES FINAIS

Este trabalho abordou as principais características da técnica formal Statecharts. Statecharts são uma extensão das Máquinas de Estados Finitos que baseia-se em três aspectos: hierarquia, comunicação via *broadcasting* e ortogonalidade. Como esses três aspectos são essenciais no desenvolvimento de sistemas complexos, a técnica Statecharts é muito empregada na especificação desse tipo de sistema. Além disso, Statecharts permitem a análise de aspectos dinâmicos do sistema, devido à sua sintaxe e à sua semântica serem bem definidas, como pode ser observado neste trabalho. Essa característica é relevante uma vez que é necessária a execução do modelo, pois apenas a verificação da consistência e completitude do mesmo não garante que não existam erros. Assim, comumente valida-se Statecharts através de simulação. Um ambiente que suporta os vários tipos de simulação é o StatSim, que também permite a edição dos Statecharts. Essas formas de simulação porém, não consideram uma questão relevante para a atividade de teste, que é o aspecto de cobertura. Esse aspecto é a análise da satisfação dos requisitos estabelecidos pelos critérios de teste e que devem ser considerados. Com a análise de cobertura a qualidade da atividade de teste pode ser quantificada. Neste contexto, tem-se explorado a aplicação da Análise de Mutantes na atividade de teste de especificações, pois esse critério é baseado em um modelo de erros que pode ser definido de forma adequada para retratar tipos de erros que explorem todas as características da técnica empregada na especificação do sistema.

Statecharts também podem fornecer apoio adequado para minimizar diversos problemas associados à especificação de hiperdocumentos, sendo então utilizados na especificação da estrutura organizacional e da semântica de navegação de hiperdocumentos grandes e complexos, como no modelo HMBS. Neste contexto, é importante o desenvolvimento de ferramentas que ofereçam suporte automatizado ao desenvolvimento de aplicações hipermídia, como é o caso do HySCharts (***Hyperdocument System based on StateCharts***), cujos ambientes de autoria e de navegação permitem criar e validar especificações de hiperdocumentos de acordo com o modelo HMBS. Uma extensão de Statecharts, os Hypercharts, pode ser mais adequada para aplicações hipermídia que necessitam da definição de seu comportamento temporal e da descrição de aspectos de sincronização. Os Hypercharts também foram brevemente apresentados nesse trabalho.

Assim como é importante o desenvolvimento de ferramentas automatizadas que apóiem a aplicação da técnica Statecharts em diferentes contextos, o desenvolvimento de ferramentas que apóiem o teste e a validação de especificações Statecharts também se torna relevante. Desse modo, também foi apresentada a ferramenta de teste Proteum-RS/ST, que apóia o teste de especificações Statecharts baseada no critério Análise de Mutantes.

REFERÊNCIAS BIBLIOGRÁFICAS

(Allworth, 1981) Allworth, S.T.; *Introduction to Real-time Software Design*, London, McMillan, 1981.

(Boaventura, 1992) Boaventura, I.A.G., *Propriedades Dinâmicas de Statecharts*, Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - USP/São Carlos, 1992.

(Bochmann et al., 1992) Bochmann, G.v.; Das, A.; Dssouli, R.; Dubuc, M.; Ghedamsi, A.; Luo, G.; *Fault Models in Testing*, IV Protocol Test Systems, Elsevier Science Publishers B. V., North-Holland, 1992.

(Bochmann & Petrenko, 1994) Bochmann, G.v.; Petrenko, A.; *Protocol Testing: Review of Methods and Relevance for Software Testing*, Proceedings of the ISSTA'1994 - International Symposium on Software Testing and Analysis, ACM - Software Engineering Notes, pp. 109-124, 1994.

(Cangussu, 1995) Cangussu, J.W.L., *Execução Programada de Statecharts*, Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - USP/São Carlos, 1995.

(Chow, 1978) Chow, T.S.; *Testing Software Design Modeled by Finite-State Machines*, IEEE Transactions on Software Engineering, SE(4(3)), pp. 178-187, 1978.

(Davis, 1988) Davis, A.M.; *A Comparison of Techniques for the Specification of External System Behavior*, Communications of the ACM, vol. 31, n. 9, setembro, 1988.

(Delamaro, 1993) Delamaro, M.E.; *Proteum - Um Ambiente de Teste Baseado na Análise de Mutantes*, Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação – USP/São Carlos, outubro, 1993.

(Delamaro & Maldonado, 1996a) Delamaro, M.E.; Maldonado, J.C.; *PROTEUM: A Tool for the Assessment of Test Adequacy for C Programs*, Conference on Performability in Computing System., New Jersey, julho, pp. 79-95, 1996.

(DeMillo et al., 1978) DeMillo, R.A.; *Software Testing and Evaluation*, The Benjamim/Commings Publishing Company, Inc, 1978.

(DeMillo, 1980) DeMillo, R.A.; *Mutation Analysis as a Tool for Software Quality*, Proceedings of COMPSAC80, Chicago-IL, outubro, 1980.

(Fabbri et al., 1993) Fabbri, S.C.P.F.; Maldonado, J.C.; Masiero, P.C.; Delamaro, M.E.; *Análise de Mutantes Baseada em Máquinas de Estado Finito*, 11° SBRC - Simpósio Brasileiro de Redes de Computadores, UNICAMP, Campinas, SP, maio, pp. 407-425, 1993.

(Fabbri, 1996) Fabbri, S.C.P.F.; *A Análise de Mutantes no Contexto de Sistemas Reativos: Uma Contribuição para o Estabelecimento de Estratégias de Teste e Validação*, Tese de Doutorado, IFSC/USP, São Carlos, SP, outubro, 1996.

(Fortes, 1991) Fortes, R.P.M.; *Uma Ferramenta de Apoio à Utilização de Statecharts para Especificação do Comportamento de Sistemas de Tempo Real Complexos*, Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação – USP/São Carlos, 1991.

(Fujiwara et al., 1991) Fujiwara, S.; Bochmann, G.V.; Khendek, F.; Amalou, M.; Ghedamsi, A.; *Test Selection Based on Finite State Models*, IEEE Transactions on Software Engineering, vol. 17, n. 6, junho, 1991.

(Gill, 1962) Gill, A.; *Introduction to the Theory of Finite-State Machines*, New York, McGraw-Hill, 1962.

(Harel, 1987a) Harel, D.; *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming, vol. 8, pp. 231-274, 1987.

(Harel, 1987b) Harel, D.; *Statecharts: On the Formal Semantics of Statecharts*, Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, Ithaca, New York, 1987.

(Harel et al., 1990) Harel, D.; Lachover, H.; Naamad, A.; Pnueli, A.; Politi, M.; Sherman, R.; Shtull-Trauring, A.; Trakhtenbrot, M.; *STATEMATE: A Working environment for the development o Complex Reactive Systems*, IEEE Transactions on Software Engineering, vol. 16, n. 4, pp. 403-414, 1990.

(Harel, 1992) Harel, D.; *Bitting the Silver Bullet - Toward a Brighter Future for Systems Development*, Computer IEEE, janeiro, pp. 8-20, 1992.

(Harel & Naamad, 1996) Harel, D., Naamad, A.; *The STATEMATE Semantics of Statecharts*, ACM Transactions on Software Engineering and Methodology, vol. 5, n. 4, pp. 293-333, 1996.

(Maldonado, 1997) Maldonado, J.C.; *Critérios de Teste de Software: Aspectos Teóricos, Empíricos e de Automação*, Trabalho de Livre Docência, maio, 1997.

(Masiero et al., 1991) Masiero, P.C.; Fortes, R.P.M.; Batista Neto, J.E.S.; *Edição e Simulação do Aspecto Comportamental de Sistemas de Tempo Real*, Anais do XI Congresso Nacional da SBC, XVIII SEMISH, Santos, agosto, pp. 45-61, 1991.

(Masiero et al., 1994) Masiero, P.C.; Maldonado, J.C.; Boaventura; *A Reachability Tree for Statecharts and Analysis of some Properties*, Information and Software Technology, 36(10), pp. 615-24, 1994.

(Naito & Tsunoyama, 1981) Naito, S.; Tsunoyama, M.; *Fault Detection for Sequential Machines by Transition-Tours*, Proceedings of the FTCS - Fault Tolerant Comput. Systems, pp. 238-243, 1981.

(Paulo, 1997) Paulo, F.B.; *Especificação de Aplicações Hipermídia baseada em Statecharts*, Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação – USP/São Carlos, 1997.

(Paulo et al., 1998) Paulo, F.B., Turine, M.A.S., Oliveira, M.C.F., Masiero, P.C.; *XHMBS: A Formal Model to Support Hypermedia Specification*, The Ninth ACM Conference on Hypertext and Hypermedia, Pittsburgh, junho, pp.161-170, 1998.

(Penteado et al., 1995) Penteado, R.A.D.; Masiero, P.C.; Germano, F.S.R.; *Manual de Uso do Sistema StatSim*, Relatórios Técnicos do ICMC/USP, São Carlos, n. 32, março, 1995.

(Peterson, 1981) Peterson, J.L.; *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1981.

(Petrenko & Bochmann, 1996) Petrenko, A.; Bochmann, G.v.; *On Fault Coverage of Tests for Finite State Specifications*, http://www.iro.umontreal.ca/pub/teleinfo/TRs/Petr96b.ps.gz, 1996.

(Pressman, 1997) Pressman, R.S.; *Software Engineering: A Practitioner's Approach*, McGRAW-HILL, 1997.

(Probert & Guo, 1991) Probert, R.L.; Guo, F.; *Mutation Testing of Protocols: Principles and Preliminary Experimental Results*, Proceedings of the IFIP TC6 Third International Workshop on Protocol Test Systems, North-Holland, pp. 57-76, 1991.

(Sabnani & Dahbura, 1988) Sabnani, K.K.; Dahbura, A.T.; *A Protocol Testing Procedure*, Computer Networks and ISDN Syst., vol. 15, n. 4, pp. 285-297, 1988.

(Simão, 2000) Simão, A.S.; *Proteum-RS/PN: Uma Ferramenta para a Validação de Redes de Petri baseada na Análise de Mutantes*, Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 2000. (em preparação)

(Souza et al., 2000) Souza, S.R.S.; Maldonado, J.C.; Masiero, P.C.; Fabbri, S.C.P.; *Coverage Testing for Specifications Based on Statecharts*, 22nd International Conference on Software Engineering, Limerick, junho, 2000. (submetido)

(Sugeta, 1999) Sugeta, T.; *Proteum-RS/ST: Uma Ferramenta para Apoiar a Validação de Especificações Statecharts baseada na Análise de Mutantes*, Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 1999.

(Sugeta et al., 1999) Sugeta, T.; Maldonado, J.C.; Fabbri, S.C.P.; *Proteum-RS/ST: Manual do Usuário*, Relatórios Técnicos do ICMC/USP, São Carlos, n. 98, dezembro, 1999.

(Turine et al., 1998) Turine, M.A.S.; Oliveira, M.C.F.; Masiero, P.C.; *HySCharts: Um Ambiente de Autoria e de Navegação Baseado no Modelo HMBS;* Anais do IV Simpósio Brasileiro de Sistemas Multimídia e Hipermídia, Rio de Janeiro, maio, pp. 27-38, 1998.

(Wong et al., 1994) Wong, W.E.; Maldonado, J.C.; Mathur, A.P.; *Mutation versus All-uses: An Empirical Evaluation of Cost, Strength and Effectiveness*, Proceedings of the First IFIP/SQI International Conference on Software Quality and Productivity, Hong Kong, novembro, 1994.

(Yao et al., 1994) Yao, M.; Petrenko, A.; Bochmann, G.v.; *Fault Coverage Analysis in Respect to a FSM Specification*, IEEE INFOCOM'94, Toronto, Canadá, pp. 768-775, 1994.

APÊNDICE A - LINGUAGEM DE ESPECIFICAÇÃO DE STATECHARTS (LES)

Neste apêndice, apresenta-se a gramática da Linguagem de Especificação de Statecharts (Fortes, 1991), com as expressões necessárias para a descrição de todos os aspectos da técnica Statecharts, como paralelismo, *broadcasting*, história (Fabbri, 1996). Essas expressões são utilizadas para descrever os statecharts a serem testados na ferramenta Proteum-RS/ST. A especificação em LES do statecharts do Relógio Digital da Figura 2.4, ilustrado no Capítulo 2, também é apresentada no final deste apêndice.

<estadograma> ::=
    [<declarações>';']
    <estados>';'
    <transições>';'

<declarações> ::=
    <declaração> {<declaração>}

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

<tipo-id> ::= (**INT** | **BOOL** | **REAL** | **STRING** | **COND**)

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

<estados> ::=
    <descrição estado> {<descrição estado>}

<descrição estado> ::=
    ESTADO <nome-estado> <tipo>

<tipo> ::=
    OU <subestados ou> | **E** <subestados e> | **ATOMO**

<subestados ou> ::=
    SUBESTADOS <nome-estado> [(**DEF** | **DEF/H** | **DEF/H***)]
        {',' <nome-estado>}

<subestados e> ::=
    SUBESTADOS <nome-estado>{',' <nome-estado>}

<transições> ::=
    <transição> {<transição>}

<transição> ::=
EVENTO <evento>
ORIGEM <estado-origem> {',' <estado-origem>}
DESTINO <estado-destino> [(/**H** | /**H***)]
{',' <estado-destino> [(/**H** | /**H***)]}
[**AÇÃO** '{'<ações> '}']

<evento> ::=
<ev-ou> {or <ev-ou>}

<ev-ou> ::=
<ev-e> {and <ev-e>}

<ev-e> ::=
<ev-f> ['[' <exp> ']']

<ev-f> ::=
<evento-primitivo>
| <true> '(' <exp> ')'
| <false> '(' <exp> ')'
| <changed> '(' <exp> ')'
| <exit> '(' <nome-estado> ')'
| <entered> '(' <nome-estado> ')'
| <t-out> '(' <numero> <unid-tempo> ')'

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

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

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

<fator> ::=
|<true>
|<false>
|<condição-primitiva>
|<número>
|<variável>
| in '(' <nome-estado> ')'
|<not-yet> '(' <evento> ')'
|<current> '(' <exp> ')'
| not <exp>
| '(' <exp> ')'

<ações> ::=
<ação> {';' <ação>}

<ação> ::=
    <identificador>
    | <variável> ':=' <exp>
    | clear '(' <nome-estado> [*] ')'

<estado-origem> ::= <nome-estado>

<estado-destino> ::= <nome-estado>

<nome-estado> ::=
    <identificador> ['.' <identificadores>]

<evento-primitivo> ::= <identificador>

<condição-primitiva> ::= <identificador>

<variável> ::= <identificador>

Símbolos Terminais:

<true> ::= ([t T] [r R] [u U] [e E] | tr)

<false> ::= ([f F] [a A] [l L] [s S] [e E] | fs)

<changed> ::= (changed | ch)

<exit> ::= (exit | ex)

<entered> ::= (entered | en)

<t-out> ::= (t_out)

<not-yet> ::= (not-yet | ny)

<current> ::= (current | cr)

<identificador> ::= [a - z A - Z][a - z A - Z 0 - 9]*

<número> ::= [0 - 9]+

<unid-tempo> ::= (seg | min | hr)

Observações:

[α] – a cadeia α é opcional
{α} – repetição da cadeia α zero ou mais vezes
α | β - α ou β deve ser escolhida

A seguir, apresenta-se um exemplo de especificação em LES do Relógio Digital (Pressman, 1997) ilustrado na Figura 2.4 do Capítulo 2.

```
ESTADO Watch OU
SUBESTADOS Dead DEF, Live
   ESTADO Dead ATOMO
   ESTADO Live E
   SUBESTADOS Main, Power, Light, Alarm_st, Chime_st
       ESTADO Main OU
       SUBESTADOS Display DEF, Beep1
          ESTADO Display OU
          SUBESTADOS Time DEF, Date, Chime, Stopwatch, Alarm, Update, Up_alarm
              ESTADO Time ATOMO
              ESTADO Date ATOMO
              ESTADO Chime ATOMO
              ESTADO Stopwatch ATOMO
              ESTADO Alarm ATOMO
              ESTADO Update OU
              SUBESTADOS Minut, Ten_min, Hour, Month, Day
                 ESTADO Minut ATOMO
                 ESTADO Ten_min ATOMO
                 ESTADO Hour ATOMO
                 ESTADO Month ATOMO
                 ESTADO Day ATOMO
              ESTADO Up_alarm     OU
              SUBESTADOS U_Min, U_Ten_min, U_Hour
                 ESTADO U_Min ATOMO
                 ESTADO U_Ten_min ATOMO
                 ESTADO U_Hour ATOMO
          ESTADO Beep1 ATOMO
       ESTADO Power OU
       SUBESTADOS Ok DEF, Weak
          ESTADO Ok ATOMO
          ESTADO Weak ATOMO
       ESTADO Light OU
       SUBESTADOS Off DEF, On
          ESTADO Off ATOMO
          ESTADO On ATOMO
       ESTADO Alarm_st OU
       SUBESTADOS Disab DEF, Enab
          ESTADO Disab ATOMO
          ESTADO Enab ATOMO
       ESTADO Chime_st OU
       SUBESTADOS C_Disab DEF, C_Enab
          ESTADO C_Disab ATOMO
          ESTADO C_Enab OU
          SUBESTADOS Quiet, Beep
              ESTADO Quiet ATOMO
              ESTADO Beep ATOMO;
EVENTO Bt_in ORIGEM Dead DESTINO Live
EVENTO Beep_rt ORIGEM Beep1 DESTINO Display /H
EVENTO T_hits_tm [in(Enab)] ORIGEM Display DESTINO Beep1
EVENTO C ORIGEM Time DESTINO Minut
EVENTO C ORIGEM Minut DESTINO Ten_min
EVENTO C ORIGEM Ten_min DESTINO Hour
EVENTO C ORIGEM Hour DESTINO Month
EVENTO C ORIGEM Month DESTINO Day
EVENTO C ORIGEM Day DESTINO Time
EVENTO B ORIGEM Update DESTINO Time
EVENTO A ORIGEM Stopwatch DESTINO Time
EVENTO A ORIGEM Chime DESTINO Stopwatch
EVENTO A ORIGEM Alarm DESTINO Chime
EVENTO C ORIGEM Alarm DESTINO U_Min
EVENTO C ORIGEM U_Min DESTINO U_Ten_min
EVENTO C ORIGEM U_Ten_min DESTINO U_Hour
```

```
EVENTO C ORIGEM U_Hour DESTINO Alarm
EVENTO B ORIGEM Up_alarm DESTINO Alarm
EVENTO A ORIGEM Time DESTINO Alarm
EVENTO D ORIGEM Date DESTINO Time
EVENTO D ORIGEM Time DESTINO Date
EVENTO D [in(Alarm)] ORIGEM Disab DESTINO Enab
EVENTO D [in(Alarm)] ORIGEM Enab DESTINO Disab
EVENTO D [in(Chime)] ORIGEM C_Disab DESTINO Quiet
EVENTO D [in(Chime)] ORIGEM Quiet DESTINO C_Disab
EVENTO T_hits_hr ORIGEM Quiet DESTINO Beep
EVENTO Beep_st ORIGEM Beep DESTINO Quiet
EVENTO B ORIGEM Off DESTINO On
EVENTO B_up ORIGEM On DESTINO Off
EVENTO Bt_weak ORIGEM Ok DESTINO Weak
EVENTO Bt_dead ORIGEM Weak DESTINO Dead ACAO {clear(Display)}
EVENTO Bt_out ORIGEM Live DESTINO Dead ACAO {clear(Display)};
```

APÊNDICE B - LINGUAGEM DE CONTROLE DE EXECUÇÃO

O programa de controle pelo qual é realizada a simulação programada no ambiente StatSim deve ser fornecido na Linguagem de Controle de Execução (LCE). Essa linguagem é composta por três partes e sua sintaxe permite, entre outras coisas, que a simulação pare quando se atinge um certo estado, um certo valor de uma variável, ou então um certo passo, passando o controle do sistema para o usuário, que por sua vez pode acionar eventos e modificar variáveis e condições e devolver o controle à "Execução Programada" quando desejar. A seguir, apresentam-se aspectos gerais da LCE, assim como sua sintaxe, ressaltando-se que todas as informações que seguem foram retiradas de (Cangussu, 1995).

A LCE permite que se estenda o controle dos passos de simulação para que o usuário especifique determinadas ações em determinados passos. Por exemplo, se no passo 100 o usuário deseja que o controle da simulação passe a ser interativo, deve-se especificar um comando como: "AT STEP 100 INTERACTIVE". Caso ele desejar ver o valor das variáveis a cada 30 passos, basta que especifique o comando: "FOR STEP 1 EACH 30 STEPS DO SHOW_VAR", juntamente com a sentença anterior.

Um dos aspectos gerais da LCE é que o usuário tem a possibilidade de escolher entre a execução animada ou a textual. Na execução animada ou gráfica, pode-se ver o comportamento do modelo diretamente na tela, enquanto que na textual, um arquivo de saída dos passos de simulação é gerado. Os comandos para especificar o tipo de execução desejado são "VISUAL" ou "TEXT".

A LCE possui características que facilitam a depuração de modelos. A possibilidade da simulação passo a passo é uma delas e é expressa através do comando "CONTROLLED". Esse mecanismo de execução porém, só é disponível no modo gráfico, pois no textual o usuário pode analisar toda a execução passo a passo através do arquivo gerado. Se este comando for omitido, a simulação é feita continuamente por *default*.

Sintaxe da LCE

A LCE é dividida em três partes. A primeira diz respeito aos parâmetros globais da execução, a segunda parte trata da declaração das variáveis de controle e das declarações para geração de eventos e a terceira parte executa o controle da simulação através dos laços, sentenças condicionais e controle dos passos, entre outros (Cangussu, 1995).

Parâmetros Globais

Essa parte é opcional, sendo que os parâmetros omitidos serão inicializados com valores *default* pré-determinados. Os parâmetros possíveis são:

NUMBER_OF_STEPS = valor
Especifica o número de passos que serão executados, sendo que o valor deve ser um número natural. O valor *default* é 100.

SIMULATION_RESULT = modo
Especifica o modo de execução, que pode ser:
VISUAL - execução apenas visual (animada), não sendo gerado o relatório de passos.
TEXT - execução apenas textual, não havendo animação e sendo gerado o relatório de passos da simulação.
COMPLETE - é o modo *default* e indica que a execução será tanto visual quanto textual, através da animação na tela e da geração do relatório de passos, respectivamente.

SIMULATION_TYPE = tipo
A simulação animada pode ser executada de duas formas, sendo que a primeira corresponde ao tipo "CONTROLLED", em que a cada passo executado o ambiente fica esperando que o usuário forneça uma confirmação para prosseguir, e a segunda forma corresponde ao tipo "FREE", no qual os passos são executados continuamente, sem a intervenção do usuário a cada passo. O tipo *default* é "FREE".

INITIAL_CONFIGURATION = configuração
Parâmetro que indica a configuração de estados ativos na qual a execução será iniciada. "DEFAULT" é o valor *default* em que o modelo será inicializado pelos estados *default* do statechart. Uma configuração específica deve ser constituída de uma lista dos nomes dos estados atômicos, os quais serão ligados juntamente com seus ancestrais.

STATECHART = modelo
Parâmetro que indica qual statechart será executado, sendo que o valor *default* é o statechart que já estiver carregado na memória.

Declarações

As declarações podem ser de dois tipos: as primeiras são as declarações das variáveis de controle, que são variáveis inteiras e possuem o símbolo "%" precedendo seus nomes identificadores. A declaração tem a forma:

CONTROL_VAR %nome1, %nome2, ..., %nomeN

As declarações de eventos determinam a forma como os eventos serão gerados nos passos de simulação, mas não especificam novos eventos. Todos os eventos citados nas declarações devem corresponder aos eventos existentes no statechart a ser executado. Podem-se gerar os eventos pelas seguintes formas:

READ_FILE(arquivo)
Cada linha do arquivo dado representa os eventos que serão disparados em um passo da simulação. Dessa forma, esse arquivo deve obedecer a um determinado formato, como no exemplo:

```
a,b,c
c,a
d,b
b,g
```

Esse exemplo indica que no primeiro passo dispara-se os eventos "a", "b" e "c", no segundo passo dispara-se "c" e "a", e assim sucessivamente, até chegar ao fim do arquivo.

EVENTS e1,...,en distribuição
Especifica que os eventos e1,...,en serão gerados seguindo uma distribuição probabilística, que pode ser exponencial, normal ou uniforme. Um exemplo pode ser:

```
EVENTS e1 EXPO(1.4)
EVENTS e2, e3 NORMAL(1.2,0.5)
EVENTS e4 UNIF(3,0.4)
```

EVENTS e1,...,en X%
Essa declaração especifica que os eventos possuem uma probabilidade X de ocorrer. No exemplo abaixo, os eventos e1 e e2 têm 30% de chance de ocorrerem em um passo e o evento e3 tem 80% de chance de ser disparado em um passo.

```
EVENTS e1,e2 30%
EVENTS e3 80%
```

EVENTS e1,...,en FROM STEP s1 TO s2 EACH s3 STEPS
Indica que os eventos e1,...,en serão disparados a partir do passo s1 até o passo s2 a cada s3 passos. A parte "TO s2" pode ser omitida e significa que o fim corresponderá ao número de passos. Se a parte omitida for "FROM STEP s1", o passo inicial será o passo 1 (um).

Controle de Simulação

Essa terceira parte é iniciada com a palavra reservada "SIMULATION" e deve terminar com "END_SIMULATION". Essa parte da linguagem especifica as atividades que serão executadas nos passos. As sentenças que controlam a simulação e auxiliam nas tarefas de verificação e validação do modelo são:

```
AT STEP s1
BEGIN
        comandos
END
```

Esta sentença determina que no passo s1 os comandos delimitados por BEGIN...END sejam executados. Se houver apenas um comando a ser executado, então pode-se omitir BEGIN...END.

```
ALL STEPS
BEGIN
        comandos
END
```

Esta sentença determina que os comandos especificados serão executados em todos os passos da simulação. Se o comando for único, também pode-se omitir BEGIN...END. Essa omissão é válida na LCE sempre que houver um único comando em qualquer sentença.

```
FOR STEP s1 TO s2 EACH s3 STEPS DO
BEGIN
        comandos
END
```

Os comandos delimitados por BEGIN e END serão executados, iniciando do passo s1 até o passo s2, a cada s3 passos. Caso a parte "TO s2" da sentença seja omitida, o limite será estabelecido pelo número de passos a ser simulado.

Observa-se que as sentenças especificadas acima não podem ser aninhadas, pois determinam o controle dos passos. Dessa forma, a composição do exemplo a seguir não é permitida.

```
SIMULATION
        AT STEP 5
        BEGIN
                ALL STEPS SHOW_VAR
                SHOW_EVENTS
        END
END_SIMULATION
```

Contudo, se forem colocadas seqüencialmente como a seguir, a composição torna-se correta.

```
SIMULATION
        AT STEP 5 SHOW_EVENTS
        ALL STEPS SHOW_VAR
END_SIMULATION
```

Os comandos que podem ser aninhados são:

```
IF condição THEN
BEGIN
        comandos
END
ELSE BEGIN
        comandos
END
```

Se a condição desta sentença for verdadeira, executam-se os comandos da parte "THEN", caso contrário, executam-se os comandos da parte "ELSE". A parte "ELSE" pode ser omitida caso não existam comandos a serem executados.

```
FOR %vc1 = expressão1 TO expressão2 DO
BEGIN
        comandos
END
```

Esta sentença determina que os comandos internos ao laço sejam executados n vezes, onde n = expressão2 – expressão1. Os contadores do laço, no caso %vc1, devem ser especificados somente por variáveis de controle.

WHILE condição DO
BEGIN
comandos
END
A sentença especifica que os comandos internos a ela sejam executados enquanto a condição especificada tiver o valor "true".

v := expressão2
%vc := expressão1
Esta sentença determina que uma variável ou uma variável de controle receberá o valor determinado pela expressão.

As expressões citadas nos comandos acima podem ser compostas de operadores (+, -, /, *), variáveis, variáveis de controle e números, enquanto que as condições são compostas de operadores relacionais, variáveis, variáveis de controle, números, além das funções:

ACTIVE(estado)
Retorna "true" se o estado estiver ativo. Caso contrário, retorna "false".

CONFIGURATION(configuração)
Retorna "true" se a configuração no passo atual for igual à especificada. Caso contrário, retorna "false".

Funções disponíveis

A seguir, apresenta-se uma breve descrição das funções disponíveis ao usuário, bem como suas sintaxes.

SHOW_VAR
Essa função é usada para exibir o valor instantâneo e médio das variáveis do statechart.

SHOW_CONTROL_VAR
Essa função é usada para exibir o valor instantâneo e médio das variáveis de controle declaradas pelo usuário.

SHOW_EVENTS
Essa função é usada para exibir os eventos disparados no passo atual de simulação.

INTERACTIVE
Função que retorna a execução para o modo interativo.

EVENTS(e1,...,en,parâmetro)
Essa função determina que os eventos pertencentes à lista de eventos especificada sejam disparados de acordo com o valor de "parâmetro", que por sua vez pode ser:

- omitido, fato que determina que os eventos sejam disparados no passo atual;
- um número natural n, determinando que os eventos sejam disparados a n passos do passo atual. Exemplo: EVENTS(a,b,5);
- uma distribuição probabilística, fazendo com que os eventos sejam disparados a n passos do passo atual, sendo n determinado para cada evento da lista de acordo com a distribuição especificada. Exemplo: EVENTS(a,b,EXPO(1)).

AVAIL(e1,...,en)
Determina que os eventos passados como parâmetro tornem-se disponíveis para serem disparados.

UNAVAIL(e1,...,en)
Determina que os eventos passados como parâmetro tornem-se indisponíveis para serem disparados.

MESSAGE(mensagem,%vc)
Função que imprime a mensagem que foi passada como parâmetro no relatório de passos em todo passo em que ela for executada. A mensagem pode ser composta de cadeias de caracteres e valores das variáveis de controle.

SET_CONFIGURATION = DEFAULT ou
SET_CONFIGURATION = configuração
Função que reinicializa o statechart por *default*, ou então de acordo com a configuração especificada.