Introdução aos Estadogramas

Fábio Nogueira de Lucena
fabio@dcc.unicamp.br
Hans K.E. Liesenberg
hans@dcc.unicamp.br

Relatório Técnico DCC–94-10

Novembro de 1994

Introdução aos Estadogramas

Fábio Nogueira de Lucena
fabio@dcc.unicamp.br

Hans K.E. Liesenberg
hans@dcc.unicamp.br

Projeto Xchart*
DCC/IMECC/UNICAMP
Caixa Postal 6065
13081-970 Campinas/SP, Brazil
e-mail: *xchart@dcc.unicamp.br*
WWW: *http://www.dcc.unicamp.br/projects/Xchart*

Sumário

Estadograma é uma notação voltada para a modelagem do comportamento complexo de sistemas reativos. Este texto descreve as suas características como linguagem (o seu léxico, a sua sintaxe e, informalmente, a sua semântica), suas origens (sistemas reativos e objetivos), empregos e ferramentas que a utilizam. Procurou-se fornecer uma visão abrangente e didática, inclusive àqueles já familiarizados com esta notação. Há ainda uma extensa lista de referências pertinentes.

1 Introdução

Estadogramas (*statecharts* [11]) surgiram para descrever sistemas reativos devido à insatisfação com técnicas até então empregadas para esta finalidade. Vários trabalhos têm utilizado esta recente extensão dos diagramas de transição de estados (DTEs) ou uma variação dos mesmos, inclusive em produtos comerciais. Este emprego já bastante difundido dos estadogramas é uma das motivações para o presente trabalho. Ao longo do texto fica claro o papel relevante desta notação para descrever o controle de sistemas complexos.

Esta seção define os sistemas reativos, identifica o problema de tais sistemas (comportamento complexo, que é difícil de ser descrito) e fornece uma taxonomia para eles. A Seção 2 fornece detalhes sobre a origem dos estadogramas. A Seção 3 define os estadogramas informalmente, a Seção 4 as áreas onde estão sendo utilizados e a Seção 5 exemplifica várias ferramentas que apóiam a construção de sistemas reativos usando esta notação. A Seção 6 contém uma visão geral e informal das características desta notação. A Seção 7 compara informalmente os DTEs com os estadogramas, ressaltando a economia do número de estados para especificação de comportamentos complexos usando estadogramas. Estadogramas é uma linguagem de especificação executável, i.e., possui semântica formalmente definida. A semântica formal é sucintamente comentada na Seção 8.

*Projeto que se concentra no desenvolvimento (especificação e implementação) de interfaces homem-computador. Por curiosidade: a 22ª letra do alfabeto grego (**X** e χ) é identificada em Inglês pela palavra CHI, que coincide com o acrônimo de *Computer-Human Interface.*

Sistemas reativos

Existem dicotomias aplicadas a sistemas (não exclusivamente de software) que separam os sistemas "difíceis" ou "problemáticos" dos demais. Sistemas síncronos/assíncronos, *off-line*/*on-line*, e seq:uenciais/concorrentes são alguns exemplos. Respectivamente, os sistemas assíncronos, *on-line* e concorrentes seriam aqueles que necessitam de tratamento especial, assim como os não-determinísticos. Em contrapartida, os sistemas determinísticos, *off-line* e seq:uenciais fornecem menos dificuldades de desenvolvimento do que aqueles das categorias complementares. Em [14] é apresentada outra dicotomia: sistema transformacional/reativo.[1]

Sistema transformacional (orientado para dados): obtém a saída através de transformações (funções) aplicadas à entrada (p.ex., compiladores, sistemas para confecção de folhas de pagamento, previsão meteorológica, contabilidade e outros).

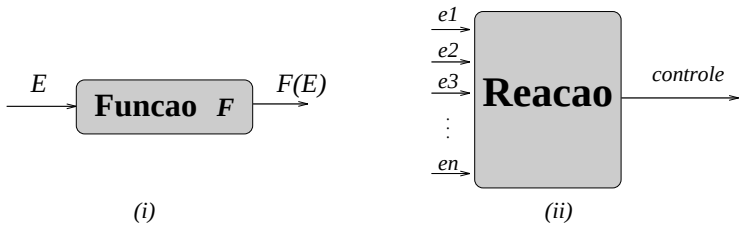

Figura 1: Sistema transformacional (i) e reativo (ii).

Um sistema transformacional, Figura 1 (i), obtém a saída S em função (transformação) da entrada E. Por exemplo, em um sistema de previsão meteorológica (processamento intensivo de dados), E compreenderia todas as informações necessárias para a previsão e $F(E)$ seria a saída obtida após o processamento da entrada (aplicação da função **Função** F à entrada E).

[1] Do inglês *transformational*. Aqui utiliza-se uma outra acepção normalmente atribuída ao termo transformacional – sistema obtido por sucessivas transformações que conservam uma equivalência funcional entre elas. Aqui, transformacional qualifica o sistema em termos de sua finalidade e não o seu método de construção.

Sistema reativo (controlado por eventos): continuamente responde (reage) a estímulos externos e internos. Em geral não computa uma função, mas mantém ininterrupto relacionamento com o ambiente. A reação (saída) é conseq:uência do tratamento da entrada (eventos) e compreende o controle do sistema, i.e., seq:uência de operações (p.ex., sistemas de comunicação, interfaces homem-computador, sistemas embutidos,[2] software de tempo real, relógio digital, caixa automático (banco), televisão e outros).

Existem diversos métodos de projeto, linguagens de programação e ambientes apropriados para o desenvolvimento de sistemas transformacionais. No entanto, isto não ocorre com sistemas reativos, que são mais problemáticos devido à carência de métodos adequados para desenvolvimento.[2] Um sistema reativo, Figura 1 (*ii*), gera sinais de controle para o sistema conforme a ocorrência de eventos de entrada gerados pelo ambiente. A função do sistema reativo é apenas gerar tal seq:uência de sinais de controle sem se preocupar para que finalidade serão utilizados. A especificação do comportamento de um sistema reativo compreende um conjunto de seq:uências legais de eventos de entrada e a saída correspondente. Uma televisão, por exemplo, apresenta um conjunto de dispositivos (botões, teclas e outros) que podem ser manipulados. Neste caso os eventos de entrada correspondem às ações do telespectador sobre tais dispositivos. Os eventos de saída correspondem às reações que provocam aumento de volume, contraste (cor), mudança de canal e assim por diante. Note que uma mesma entrada pode provocar reações distintas dependendo do contexto onde ocorre. Isto ocorre devido à noção de contexto, modo ou estado. Por exemplo, há televisores que utilizam um mesma tecla ora para aumentar contraste, ora para o brilho, ora para o volume. Isto provoca uma economia de dispositivos, que agora têm múltiplas funções. Esta situação é similar ao que ocorre com os caixas automáticos de bancos. Geralmente eles utilizam as mesmas teclas numéricas como entrada da senha e do valor a ser sacado. Em uma interface homem-computador ora a tecla **A** escolhe uma opção de menu ora é parte de uma seq:uência de caracteres fornecida pelo usuário. No entanto, isto nada mais é que a interface (de um sistema reativo) responsável por captar o que ocorre no meio ambiente e transferir para o componente responsável pela reação. Enfim, o que estes sistemas possuem em comum é o caráter reativo, a saída (sinais de controle) como reação à seq:uência de eventos produzidos como entrada. *Reagir (diferente de transformar) representa a principal função destes sistemas.*

Outro exemplo: um caixa automático continuamente espera por eventos provocados por clientes de um banco. Um cliente primeiramente escolhe (provoca um evento) uma operação entre uma série de opções (extrato, saldo e outras). Como reação à sua escolha,

---

[1] Do inglês *embedded*. Sistemas em que o software é um entre vários componentes e freq:uentemente não apresenta interface com o usuário. Sua interface interage com outros componentes do sistema em questão. Geralmente controla outros componentes.

[2] Sistemas requerem métodos particulares para desenvolvimento de acordo com a sua natureza. A categoria dos sistemas transformacionais é bem estudada e para ela já existem teorias sólidas acerca do processo de desenvolvimento. Esta situação é contrastante, se a compararmos com a de outras categorias.

o sistema pergunta o número da conta ao cliente. Se o número existir (condição) pergunta a senha e assim por diante. Em qualquer momento o cliente pode interromper e reiniciar o processo, i.e., gerar um evento que provoca um retorno ao estado inicial. Se o cliente demorar demasiadamente para fornecer qualquer informação o processo é interrompido e o sistema retorna ao estado inicial, ou seja, o término de um intervalo de tempo é suficiente para provocar um evento. Vê-se que se trata de um sistema controlado por eventos. Embora essa simples descrição seja incompleta, o que é importante ressaltar é sua finalidade de relacionar eventos de entrada e saída ao longo do tempo.

Taxonomia para sistemas reativos

Neste trabalho segue-se a taxonomia apresentada por Berry e Gonthier [1]. Segundo esta taxonomia, um sistema reativo pode ser visto como uma composição de três camadas:

- Uma **interface** com o ambiente responsável por "perceber" a ocorrência de eventos (entrada) no ambiente externo e traduzir eventos físicos (externos) em lógicos (internos) e vice-versa para estabelecer a comunicação do ambiente externo com o núcleo reativo.
- Um **núcleo reativo** que contém a lógica ou o comportamento do sistema. Gerencia as entradas e saídas lógicas. Discrimina qual a reação a ser realizada em resposta a uma entrada específica. Ferramentas que apóiam a confecção de sistemas reativos geralmente fazem uso de linguagens de propósito específico para descrever este componente e da descrição geram automaticamente código em outra linguagem, por exemplo C, funcionalmente correspondente à descrição feita.
- **Aplicação** é a camada responsável pela funcionalidade do sistema, i.e., as transformações de dados. Ela recebe os sinais de controle gerados pelo núcleo reativo, processa-os, possivelmente gera informações para serem apresentados na interface.

A especificação e implementação do núcleo reativo são fundamentais, pois trata-se da camada de maior complexidade e importância dos sistemas reativos. O núcleo reativo também qualifica um sistema reativo, i.e., diz-se que um sistema é reativo quando o seu componente preponderante desempenha as funções pertinentes ao núcleo reativo.

2 Origem dos estadogramas

Alguns sistemas precisam de linguagens apropriadas para a descrição do comportamento complexo que apresentam. Um caixa automático (exemplo da seção anterior) oferece dificuldades quando se tenta especificar o seu comportamento informalmente em linguagem natural. Ainda há casos bem mais complexos onde uma linguagem informal ou imprecisa é de pouca utilidade, como por exemplo, a descrição do comportamento de um aparelho de som (tente primeiro sem o controle remoto), de um automóvel.

Os sistemas reativos apresentam um comportamento complexo, são recheados de eventos condicionais, a noção de contexto (estado) está presente e, muitas vezes, conjuntos de

estados encontram-se ativos simultaneamente refletindo conceitualmente a concorrência no sistema modelado.

O comportamento complexo de um sistema reativo não é devidamente descrito por um simples relacionamento que especifica a saída como função da entrada, mas requer a relação entre saída e entrada através de combinações permitidas ao longo do tempo. Conforme Harel [14], métodos existentes para o desenvolvimento de sistemas passo a passo e bem-estruturados são predominantemente transformacionais. Draper [6] faz a mesma afirmação destacando a aplicação imprópria de métodos tradicionais no desenvolvimento de interfaces homem-computador (exemplo de uma subclasse de sistemas reativos). Ainda convém ressaltar que sistemas reativos não podem ser vistos naturalmente como sistemas transformacionais. Isto, porque a saída não depende exclusivamente da entrada corrente mas também de estados anteriores.

Os métodos empregados para descrição de sistemas transformacionais são inadequados para a descrição de sistemas reativos, e os métodos existentes atualmente para esta finalidade apresentam dificuldades [11]. Neste contexto surgem os estadogramas. Em [11] são destacados inconvenientes dos DTEs e de outras notações para a descrição do comportamento de sistemas reativos. As características principais enfatizadas são: a natureza gráfica (muitas notações são na forma de texto) que é mais intuitiva e facilita a compreensão, a inexistência de suporte à hierarquia e concorrência e a dificuldade de manutenção das especificações. Existem várias linguagens para a especificação de sistemas reativos. Cada uma delas, naturalmente, apresenta qualidades e inconvenientes.

Estadograma é considerada "boa candidata" para especificar e programar sistemas reativos [17]. Outros trabalhos citados ao longo deste texto mostram o grande interesse e atenção que estudiosos têm despendido com esta notação.

3 Estadogramas

Vimos que sistemas reativos são considerados complexos em relação ao comportamento e que estadogramas foram propostos para especificá-los. Esta seção define, em poucas palavras, a notação através de seus elementos principais.

Estadogramas: diagramas propostos para descrição do comportamento de sistemas reativos [11]. Retém a natureza gráfica dos DTEs e acrescenta concorrência, comunicação e hierarquia.

Harel em [12] enfatiza duas características importantes dos estadogramas:

- São de natureza **bidimensional** – podem ser representados graficamente, o que facilita a compreensão.
- São **formais** – permitem automatizar a sua manipulação, manutenção e análise.

Estadogramas são construídos sobre alguns conceitos básicos como estados, eventos e condições. Eventos e condições podem ser combinados para causar transições entre estados.

Ações são as respostas dos estadogramas às transições. Ações podem controlar atividades. Atividades fornecem semântica a uma aplicação, i.e., correspondem ao código pertinente à funcionalidade. Dessa forma, a entrada correspondente a uma especificação em estadogramas compreende estímulos externos (possivelmente associados a condições de guarda que eventualmente chegam a inibir o disparo de transições), enquanto a saída são ações que controlam atividades ou geram outros eventos. A união das saídas e entradas permitidas ao longo do tempo compreendem o controle de um sistema, que pode ser descrito em termos de estadogramas. Eventos, estados, atividades, ações e outros termos são definidos na Seção 6.

4 Empregos

A especificação de sistemas reativos é o motivo da existência dos estadogramas. Neste contexto há evidências favoráveis quanto ao seu emprego [17]. Em particular, apresentam qualidades para a especificação de hardware [7]. Em [7] também é cogitada a hipótese do uso de estadogramas como linguagem de programação para sistemas concorrentes. Harel [11] sugere várias aplicações dos estadogramas: especificação de diálogo de interfaces, descrição de hardware, protocolos de comunicação, sistemas de tempo real e outras.

5 Ferramentas que suportam estadogramas

Nesta seção são vistos alguns ambientes que apóiam a construção de sistemas reativos. Estes ambientes exemplificam os esforços e as atenções despendidos com os estadogramas. Todos os sistemas abaixo citados possuem em comum o emprego de estadogramas como linguagem central, apesar da sua "infância" e pouca maturação.

O uso da notação de estadogramas para a especificação de sistemas que pertencem a subclasses bem caracterizadas de sistemas reativos (por exemplo, interfaces homem-computador) induz a construção de dialetos da linguagem "oficial" para adequá-la às situações típicas do domínio considerado. Por exemplo, a ferramenta Reacto, descrita abaixo, utiliza uma variante da notação de estadogramas. Em [5] são enumeradas algumas dificuldades para especificação de interfaces homem-computador e proposta uma notação derivada de estadogramas para eliminar tais inconvenientes. Convém ressaltar que [11, pág. 256] não apresenta recomendações conclusivas para a sintaxe e semântica dos estadogramas.

Statemaster

Statemaster [26] é um UIMS[3] utilizado na construção de protótipos e implementação de interfaces homem-computador. Fornece o *run-time* de uma interface correspondente ao controle de diálogo.[4]

[3] Sistema que provê suporte ao desenvolvimento e execução de interfaces homem-computador.

[4] Equivale ao núcleo reativo de um sistema reativo. O diálogo da interface é o que se denomina comumente de sintaxe da interação.

O responsável pela especificação deve converter os estadogramas gerados para objetos compreendidos pelo *run-time* do Statemaster. (Não existe ferramenta que automatize este processo.) O paradigma de objetos e C++ devem ser utilizados para este propósito.

Statemate

Ambiente gráfico criado para a especificação, análise, projeto e documentação de sistemas reativos [13]. O ambiente fornece três linguagens visuais (gráficas) e ortogonais: *module-charts*, *activity-charts* e estadogramas. Cada uma representa uma visão diferente do sistema modelado. As linguagens capturam a estrutura, a funcionalidade e o comportamento reativo, respectivamente. As linguagens contam com um editor gráfico que facilita a construção de especificações. As especificações podem ser analisadas por uma ferramenta do ambiente que detecta, por exemplo, *deadlock* e não determinismo. Permite ainda animação (interativa), simulação (*batch*, especificada através de uma linguagem própria) e geração automática de código em Ada. Detalhes podem ser obtidos em [19] e [21].

StP

StP© — *Software through Pictures*, fevereiro/1994, é um produto da IDE (*Interactive Development Environments*) que dá suporte, entre outras, à metodologia OMT [25]. Esta metodologia usa uma variante de estadogramas para representar o "modelo dinâmico" de um sistema.

Reacto

Reacto é um ambiente de suporte à aquisição e implementação correta de especificações de software para sistemas reativos [23]. A modelagem de sistemas reativos neste ambiente faz uso de diagramas de transição de estados derivados de estadogramas. Nesta variante alguns recursos dos estadogramas como, por exemplo, o mecanismo de história, foram suprimidos. Reacto está voltado para o projeto confiável de sistemas reativos. A confiabilidade advém de invariantes fornecidas pelo especificador associadas a estados e através da prova de que a geração de código realizada pelo compilador é correta.

Tradutor Blob e Egrest

Tradutor Blob[5] compreende um "tradutor" e um *run-time* (código invariante). O "tradutor" gera código C a partir de especificações de estadogramas em forma de texto (linguagem LEG). O *run-time* interage com o código gerado para realizar as atividades de um núcleo reativo. O Tradutor Blob e o *run-time* são descritos detalhadamente em [10].

A Figura 2 mostra o processo de desenvolvimento de sistemas reativos utilizando estas ferramentas. O Egrest [9] permite a edição de estadogramas e gera automaticamente código LEG equivalente. O Egrest ainda permite a animação concorrente da especificação de acordo com eventos repassados pelo sistema executado paralelamente e gerado a partir

[5] *Blob* é uma outra denominação empregada em [12] para um estado de um estadograma.

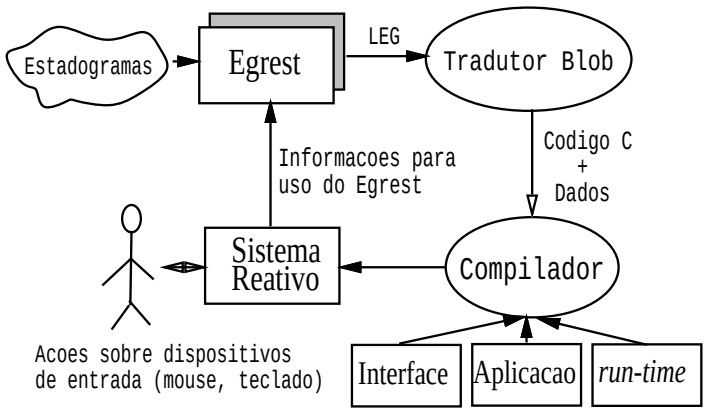

Figura 2: Relação entre Tradutor e Egrest.

do estadograma em questão. Desta forma o usuário pode interagir com o sistema reativo enquanto, simultaneamente, o Egrest exibe o progresso do sistema através de transições entre estados.

O *run-time* corresponde à semântica associada à notação de estadogramas e, portanto, é invariante. A grosso modo, a aplicação, a interface e o *run-time* (módulos no canto inferior direito da Figura 2) equivalem às camadas descritas na Seção 1 (taxonomia).

6 Descrição informal

Uma descrição informal e exaustiva dos estadogramas é fornecida em [11]. Outras fontes incluem [12, 14, 19, 20]. Nesta seção, as características e o comportamento básicos dos estadogramas são comentados informalmente. Cada elemento léxico está definido e destacado entre barras horizontais.

Estado: captura uma situação, contexto ou modo conceitual. É representado graficamente por um retângulo com cantos arredondados. Há três tipos de estados: **OR**, **AND** e **BASIC**.

Na especificação de uma interface homem-computador, por exemplo, um estado pode estar associado a um "modo" de operação do sistema. Na Figura 3 vemos dois estados identificados pelos rótulos **Or** e **And**. O estado **Or** (tipo **OR**) contém dois subestados:[6] **E1** e **E2**. Estar[7] no estado **Or** significa estar exclusivamente em apenas um de seus subestados, ou em **E1** ou em **E2**. Por outro lado, estar em **And** (estado tipo **AND**) significa estar em **E3**, **E4** e **E5** simultaneamente. Por conseguinte, os subestados de **And** são ditos

[6] Diz-se que um estado s é subestado de S quando S for ancestral direto ou indireto de s.

[7] "Estar" em um estado representa conceitualmente que o sistema modelado encontra-se em certa situação ou condição, representada por um estado ativo no instante considerado. Também é usado "atingir o estado **E**" com a acepção de "tornar corrente o estado **E**."

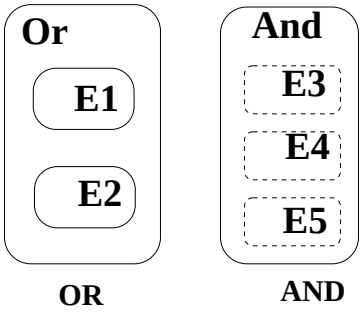

Figura 3: Tipos de estados em estadograma.

concorrentes ou ortogonais, pois jamais um deles estará ativo sem que os demais não estejam. Ainda, na Figura 3, tem-se **E1**, **E2**, **E3**, **E4**, e **E5** como estados básicos, pois não possuem subestados.

Especificações em estadogramas interagem com o "mundo exterior" através de **eventos** (entrada) e **ações** (saída).

Evento: modela a entrada fornecida a um sistema reativo. Uma especificação na forma de um estadograma deve reagir à ocorrência de um evento conforme o controle desejado para o sistema reativo descrito. Cada evento é discreto, não possui duração e vários eventos podem ser produzidos em um mesmo instante de tempo.

O agente que provoca um evento varia com a natureza de sistema reativo modelado. Se for uma interface homem-computador, o usuário gera eventos ao manipular dispositivos de entrada; se for um sistema distribuído, uma mensagem enviada por um processo equivale a um evento, e assim por diante.

Ação: expressa a saída para o "ambiente externo" provocada pela reação a uma dada entrada. Uma ação pode estar associada a uma transição ou uma ativação/desativação de um estado. Geralmente são utilizadas para controlar atividades.

Conceitualmente uma ação não consome tempo e pode gerar eventos que eventualmente afetam componentes ortogonais. Ações são semelhantes às saídas das máquinas de Mealy e Moore [2]. São instantâneas e fornecem um meio para controlar atividades (responsáveis pela funcionalidade do sistema modelado). Para esta finalidade existem: `start(X)` e `stop(X)` para iniciar e interromper a atividade `X`. A expressão `active(X)` é uma condição, que é verdadeira quando `X` está em execução.

Atividade: processamento responsável pela funcionalidade de um sistema reativo e que consome tempo. Geralmente transforma dados.

As atividades de um sistema reativo são controladas por meio de ações. Atividade modela a funcionalidade. Pode representar um processamento longo (intensivo em termos de processamento) ou o soar de um apito, por exemplo. Atividades, contudo, não têm as suas especificações descritas em termos de construções da notação de estadogramas. Relacionam-se com a parte funcional e são geralmente descritas com notações pertinentes. Nas metodologias OMT [25] e Statemate [13], por exemplo, uma atividade corresponde à funcionalidade de uma bolha de um diagrama de fluxo de dados do sistema modelado.

Atividades podem ser vistas como um conjunto de procedimentos agrupados em uma biblioteca que fornece a funcionalidade do sistema modelado. A cada instante que um estado torna-se ativo ou é desativado, conforme a especificação, uma função desta biblioteca pode conceitualmente vir a ser executada. De forma análoga podemos ter uma função associada a uma transição, que uma vez percorrida, provoca a execução da função. Estas chamadas são realizadas por ações especificadas nos estados ou transições. Esta semântica permite a execução de estadogramas como em [4].

Aresta: construção utilizada para descrever uma transição (relações) entre estados. Uma aresta é representada por um arco orientado.

Uma aresta constitui-se no único meio através do qual um estado pode ser atingido, explícita ou implicitamente. A Figura 4 exibe duas transições. Uma com origem nos estados (**A**,**B**,**C**) e destino (**D**,**E**); outra com origem **F** e destino **C**.

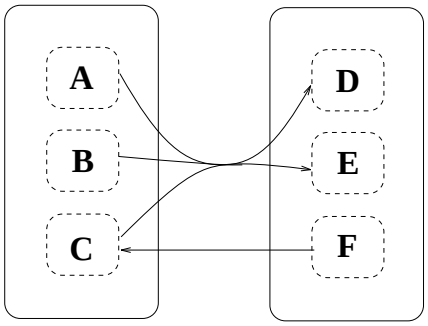

Figura 4: Arestas típicas em estadogramas.

Rótulo: combinação de eventos, condições e ações. Um rótulo estabelece a forma de interpretação de uma aresta.

Genericamente, uma aresta é rotulada por $\alpha[\delta]/\beta$ onde: α é o evento necessário para que uma transição ocorra; δ é uma condição de guarda que eventualmente inibe a transição, caso seja falsa; e β é uma ação disparada quando da ocorrência da transição. Os elementos de um rótulo são opcionais. A Figura 5 fornece exemplos de arestas. No exemplo (i), se o evento α ocorre e o seu estado origem estiver ativo, então a transição correspondente é percorrida. Quando a "origem" de uma aresta é representada por um ponto hachurado (ii) não temos ligação entre estados e o rótulo deste tipo de aresta sempre é vazio. Esta aresta é denominada aresta *default*. Arestas *default* são percorridas durante um processo de ativação quando nenhuma outra alternativa é imposta. O estado destino é dito estado *default*. Em (iii), se e somente se a origem da aresta estiver ativa, ocorre α e a condição δ for verdadeira neste instante, então a transição é percorrida e a ação β é disparada.

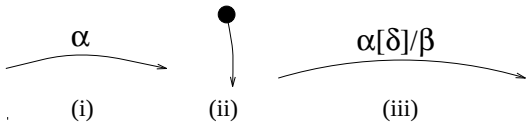

Figura 5: Arestas em estadograma.

Na Figura 6 vemos os estados **A** e **B** ligados pela aresta rotulada apenas com o evento e. Neste caso, a ocorrência do evento e é a condição necessária e suficiente para causar a transição do estado **A** para o **B**, caso **A** esteja ativo. Também deve ser destacado que o estado *default* é o **A**. Na animação do comportamento descrito por este estadograma o primeiro estado visitado é o estado **A**, por *default*.

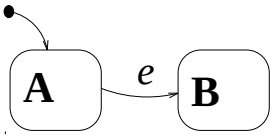

Figura 6: Transição entre estados.

História: mecanismo de ativação de subestados. Se a entrada em um estado se der por história, o subestado mais recentemente visitado é escolhido, em detrimento do estado *default*.

Se o mecanismo de história estiver prevalecendo e for do tipo H*, então considera-se como candidato a ser ativado o subestado mais recentemente visitado, independente do nível hierárquico em que se encontra. Se a história for simples (H), considera-se apenas subestados (descendentes diretos) do estado que contém a indicação de história. No último caso, a ativação destes subestados prossegue com arestas *default* para os seus subestados indiretos subseq:uentes.

Na Figura 7 temos uma entrada no estado **X** por história simples. Caso o estado corrente seja K e ocorra o evento e, então o subestado de **X** a ser ativado será o mais recentemente

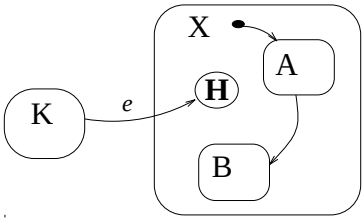

Figura 7: Exemplo de história simples.

visitado entre **A** e **B**. Caso seja a primeira vez que o estado **X** é visitado então segue-se a aresta *default* que conduz ao estado **A**. Este tipo mais simples de história, contudo, não se aplica mais aos eventuais subestados de **A** e **B**. Desse modo, se **A** e **B** possuem subestados, a entrada em **X** pelo evento e irá conduzir ao subestado *default* de **A** ou **B**. A Figura 8 também exemplifica o emprego de história. A ocorrência do evento e na Figura 8 (i) irá conduzir ao estado **B** ou **D** (subestados *default* de **K1** e **K2**, respectivamente). A Figura 8 (ii) mostra outro tipo de história, a história $\mathbf{H}^*$. Neste caso, com a ocorrência de e, a entrada em **K** tornará ativo o estado mais recentemente visitado de **K**, independente do nível em que se encontrar. Desse modo, o novo estado poderá ser **A**, **B**, **C** ou **D**, conforme o passado mais recente de ativação.

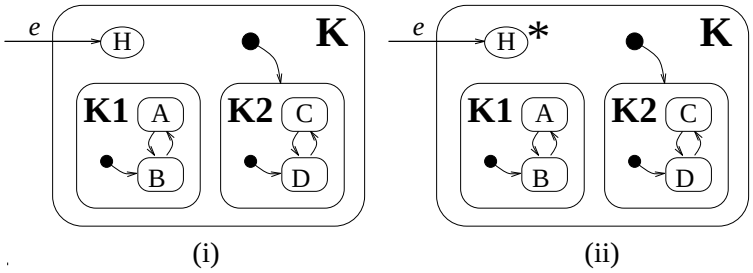

Figura 8: Exemplo de história $\mathbf{H}^*$.

Durante o processo de especificação estados podem ser agrupados formando um novo estado (processo *bottom-up*), ou refinados através da definição de seus subestados (*top-down*).

Na Figura 8, os estados **A** e **B**, durante o processo de especificação do estado **K**, podem ter sido agrupados formando o estado **K1**; **C** e **D** formando o estado **K2** e, por último, **K1** e **K2** formando **K**. Ou seja, durante a especificação foi utilizado o processo *bottom-up*. A ordem inversa também é possível, i.e., processo *top-down*. O estado **K** é refinado produzindo os estados **K1** e **K2**. Estes passam, então, a ser refinados e assim por diante. Ainda pode-se utilizar um processo de confecção de estadogramas misto, mesclando ambos os processos.

Hierarquia: estados podem estar agrupados em outros estados formando "níveis." Estes níveis formam uma hierarquia

---

A hierarquia de estados de uma especificação em estadogramas pode ser representada por

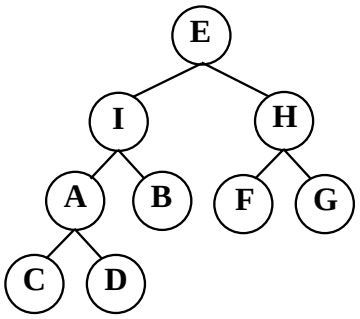

Figura 9: Representação explícita da hierarquia presente na Figura 11 (i).

uma árvore já que subestados não são compartilhados. Por exemplo, a Figura 9 ressalta a hierarquia da especificação apresentada na Figura 11 (i). Sem hierarquia não há um processo de construção *top-down* ou *bottom-up*. Hierarquia ainda permite "fatorar" transições (veja a Figura 11) e facilita interação com especificações complexas. Neste último caso, níveis de abstrações distintos podem ser empregados, quando conveniente.

Comunicação: estados concorrentes podem utilizar eventos gerados em ações ou primitivas como `en(s)`, `ex(s)` e outras para comunicação (sincronização) com outros estados.

A Figura 10 apresenta um estado **A** e seus vários subestados concorrentes (**X**, **Y** e **Z**). Quando o evento e é gerado, por exemplo, todos os estados ativos que possuírem transições rotuladas com e serão desativados e os estados destinos das transições ativados. Neste caso particular, isto é válido para os estados **B** e **D**. A ocorrência desta situação exemplifica o mecanismo de *broadcast*. Se, contudo, os estados **D** e **F** estiverem ativos, a ocorrência do evento e faz com que **D** e **F** sejam desativados e **E** e **G** ativados. A transição de **F** para **G** ocorre em conseq:uência da ação (evento a) ocasionada pela transição de **D** para **E**.

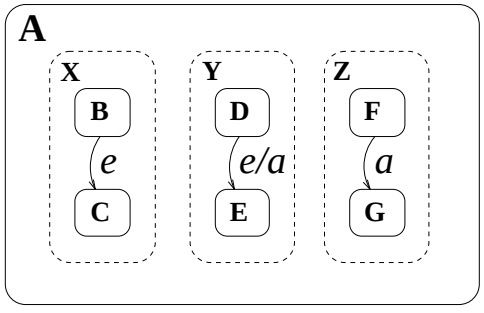

Figura 10: Estados concorrentes.

Concorrência: estados concorrentes (ortogonais) modelam concorrência em estadogramas. Todos os "filhos" de um estado tipo **AND** tornam-se ativos simultaneamente quando este for ativado, e são desativados quando este estado deixar de ser um estado corrente. Os comportamentos definidos por estados concorrentes são assíncronos entre si, no sentido de que uma transição em um subestado concorrente ocorre de forma independente das transições nos demais subestados concorrentes de um estado do tipo **AND**.

Estados concorrentes são aqueles descendentes imediatos do mesmo ancestral do tipo **AND**. Todos os descendentes diretos deste ancestral tipo **AND** são ativados quando este ancestral é ativado. De forma análoga, todos os subestados são desativados quando este ancestral é desativado. Estados concorrentes possibilitam a modelagem natural de concorrência.

7 Poder de expressão

Em [8] é mostrado que utilizar estadogramas significa uma economia exponencial quanto a densidade de construções utilizadas em uma especificação comparado com o empregado de outra notação como a de DTEs para o mesmo problema. Esta seção ressalta as extensões propostas nos estadogramas feitas sobre os DTEs. A Figura 11 exibe um estadograma (i) e seu equivalente em diagramas de transição de estados convencionais (ii).

Dificuldades com os DTEs:

- Não fornecem noção de hierarquia ou de modularidade. Conseq:uentemente não apoiam um estilo de desenvolvimento *top-down* ou *bottom-up*. A hierarquia de estados ainda permite o uso do mecanismo de história.

- Um evento que provoca a mesma transição de um grande número de estados deve ser explicitamente especificada para cada estado. Não há como "fatorar" transições semelhantes (ocasionadas por um mesmo evento) de um grupo de estados. Por exemplo, a Figura 11 (i) mostra a transição do estado **A** para o estado **B** rotulada com o evento a. Uma outra forma de especificar este comportamento é substituir esta transição por duas outras saindo dos estados **C** e **D** para o estado **B**.

- O número de estados cresce de forma exponencial nos DTEs a medida que o sistema modelado cresce linearmente quando da adição de novos estados concorrentes (independentes), já que estes estados novos precisam ser combinados com os já existentes.

- DTEs são intrinsecamente seq:uenciais, não representam concorrência naturalmente. Modelar um sistema concorrente pelo seu estado global não é considerado natural, neste trabalho.

Em contrapartida, a semântica das construções da notação dos DTEs é trivial, comparada com o sofisticado mecanismo de transições dos estadogramas. A semântica operacional dos estadogramas é apresentada na Seção 8.

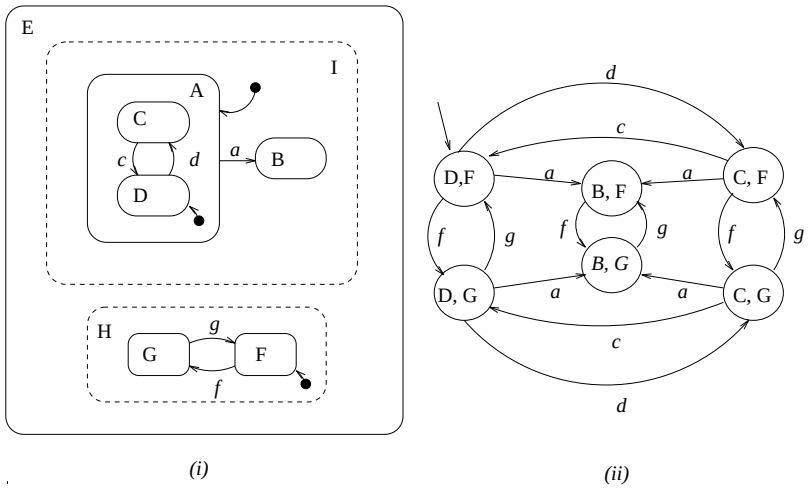

Figura 11: Um estadograma (i) e o DTEs (ii) convencional equivalente.

Baseado no problema modelado na Figura 11(i) podemos inicialmente especificar os estados **A**, **B** e a transição entre eles. Após isto podemos refinar o estado **A**, levando à identificação dos estados **C** e **D** (*top-down*). Pode-se também fazer o inverso, ou seja, identificar os subestados **C** e **D** e encapsulá-los no estado **A** (*bottom-up*). Diagramas convencionais (ii), por outro lado, não permitem tais processos de construção.

A transição rotulada com o evento a em (i) é substituída por quatro outras transições em (ii) induzidas pela hierarquia e concorrência existente nos estadogramas.

Durante a especificação de um sistema pode ser identificada a necessidade de um estado concorrente adicional, não previsto anteriormente. Usando estadogramas seria suficiente adicionar um estado concorrente ao conjunto de estados já especificados para modelar o sistema desejado. Quando são empregados diagramas convencionais, é preciso duplicar toda a especificação! É simples imaginar a especificação em estadogramas antes do acréscimo do estado **H** como subestado de **E**, o que já não é tão simples com os DTEs em (ii).

A situação anterior é conseq:uência da dificuldade de se expressar concorrência nos diagramas convencionais. Apesar de estados como **H** e **I** serem independentes, nos diagramas convencionais não há como representar explicitamente esta independência.

Neste exemplo ainda não foram explorados o mecanismo de comunicação entre estados concorrentes e transições condicionais (com "guarda"), bem como o recurso de história.

8 Semântica formal

Em [11] apenas são comentadas algumas questões que a semântica formal dos estadogramas deve resolver. Em [17] são comentadas sutilezas relacionadas à semântica de estadogramas, por exemplo: necessidade de uma reação em cadeia não consumir tempo (conceitualmente),

micro-passos e outras. Há várias propostas para a semântica formal dos estadogramas e algumas delas podem ser obtidas em [15, 18, 22, 16] e [24]. A descrição informal apresentada na seção anterior equivale à semântica formal (operacional) apresentada em [15]. Esta semântica diz como um sistema reativo descrito em estadogramas deve reagir quando da ocorrência de eventos. Esta semântica é comentada em outro trabalho [3].

Agradecimentos

Este texto contou com leituras e sugestões de várias pessoas. Entre elas Mário Massato Harada, Thierson Couto Rosa, Osvaldo Severino Junior, Luciana de Paula Brito e Jorge E. S. Jambeiro Filho.

Referências

[1] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[2] John Carroll and Darrell Long. *Theory of Finite Automata*. Prentice Hall, Inc., 1989.

[3] Fábio N. de Lucena, Mário Massato Harada, and Hans K.E. Liesenberg. Semântica Operacional de Statecharts. *Em preparação*, 1994.

[4] Fábio N. de Lucena and Hans K. E. Liesenberg. A Statechart Engine to Support Implementations of Complex Behaviour. *Proceedings of XXI Semish*, 1994. Caxambu/MG, Brazil.

[5] Fábio N. de Lucena and Hans K.E. Liesenberg. Reflections on Using Statecharts to Capture User Interface Behaviour. *Proceedings of XIV Int. Conf. of the Chilean CSS*, october 1994.

[6] Stephen W. Draper and Donald A. Norman. Software Engineering for User Interfaces. *IEEE Transactions on Software Engineering*, SE-11(3):252–258, March 1985.

[7] Doran Drusinsky and David Harel. Using Statecharts for Hardware Description and Synthesis. *IEEE Transactions on Computer Aided Design*, 8(7):798–807, July 1989.

[8] Doron Drusinsky and David Harel. On the Power of Cooperative Concurrence. *LNCS*, 335, 1988.

[9] Valéria Gonçalves Soares Elias. Editor Gráfico de Estadogramas, Dezembro 1992. DCC - IMECC - UNICAMP. Tese de Mestrado.

[10] Antonio Gonçalves Figueiredo Filho and Hans Liesenberg. Um Processo de Síntese de Sistemas Reativos, Dezembro 1991. DCC - IMECC - UNICAMP. Tese de Mestrado.

[11] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[12] D. Harel. On Visual Formalism. *Communications of the ACM*, 31(5):514–530, May 1988.

[13] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, April 1990.

[14] D. Harel and A. Pnueli. On the Development of Reactive Systems. Technical Report, The Weizmann Institute of Science — Department of Applied Mathematics, Rehovot 76100, Israel, 1985.

[15] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. *Proceedings of 2nd. IEEE Symposium on Logic in Computer Science*, pages 54–64, 1987.

[16] J.J.M. Hooman, S. Ramesh, and W.P. de Roever. A Compositional Axiomatization of Statecharts. *Theoretical Computer Science*, 101(2):289–335, July 1992.

[17] C. Huizing and W. P. de Roever. Introduction to Design Choices in the Semantics of Statecharts. *Information Processing Letters*, 37:205–213, 1991.

[18] C. Huizing, R. Gerth, and W. P. de Roever. Modelling Statecharts Behaviour in a Fully Abstract Way. In *Lecture Notes in Computer Science*, pages 271–294. Springer-Verlag, 1988. Number 299.

[19] i Logix Inc. The Languages of STATEMATE, 1987. Burlington, MA.

[20] i Logix Inc. STATEMATE: Semantics of statecharts, August 1989. Burlington, MA.

[21] i Logix Inc. The Statemate Approach to Complex Systems, August 1989. Burlington, MA.

[22] Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. *Lecture Notes in Computer Science*, 571:591–620, 1992.

[23] Limei Gilham, Allen Goldberg, and T. C. Wang. Toward Reliable Reactive Systems. *ACM SIGSOFT Engineering Notes*, 14(3):68–74, May 1989.

[24] A. Pnueli and M. Shalev. What is in a Step: On the Semantics of Statecharts. In *Lecture Notes in Computer Science*, pages 244–264. Springer-Verlag, 1991. Number 526.

[25] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, N.J., 1991.

[26] Pierre D. Wellner. Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation. In *Proceedings SIGCHI'89*, pages 177–182, Austin,TX, April 1989.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the DNA Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br