GERAÇÃO DE GERENCIADORES DE SISTEMAS REATIVOS

Antonio G. Figueiredo Filho

Hans K. E. Liesenberg

RELATÓRIO TÉCNICO Nº 20/90

Sumário. Este artigo apresenta um gerador de programas adequados para implementar o controle de sistemas reativos complexos. A ferramenta utiliza *estadogramas*, uma extensão de diagramas de estados convencionais suportando conceitos de *hierarquia, concorrência* e *comunicação*. A sua entrada se constitue de uma descrição textual de um estadograma e produz como saída um programa funcionalmente equivalente em C.

Abstract. This paper presents a program generator appropriate to implement complex reactive systems. The tool uses statecharts, an extension of conventional state diagrams supporting the notion of hierarchy, concurrency and communication. Its input consists of a textual description of a statechart and it outputs a functionally equivalent program in C.

Instituto de Matemática, Estatística e Ciência da Computação
Universidade Estadual de Campinas
13.081, Campinas, S.P.
BRASIL

Maio – 1990

Geração de Gerenciadores de Sistemas Reativos*

Antonio G. Figueiredo Filho†

Hans K. E. Liesenberg‡

Departamento de Ciência da Computação
IMECC - UNICAMP
Caixa Postal 6065
13081 - Campinas, SP
Fone: (0192) 391301 r. 8442
Fax: (0192) 395808
e-Mail: antonio@bruc.ansp.br

Sumário

Este artigo apresenta um gerador de programas adequados para implementar o controle de sistemas reativos complexos. A ferramenta utiliza *estadogramas*, uma extensão de diagramas de estados convencionais suportando conceitos de *hierarquia*, *concorrência* e *comunicação*. A sua entrada se constitue de uma descrição textual de um estadograma e produz como saída um programa funcionalmente equivalente em C.

Abstract

This paper presents a program generator appropriate to implement complex reactive systems. The tool uses statecharts, an extension of conventional state diagrams supporting the notion of hierarchy, concurrency and communication. Its input consists of a textual description of a statechart and it outputs a functionally equivalent program in C.

*Este trabalho teve apoio financeiro do CNPq.

†Mestrando em Ciência da Computação (UNICAMP, Campinas-SP); Eng. Civil (UEMA, São Luís-MA); Áreas de Interesse: Interface com o usuário, computação gráfica e ambientes de programação.

‡Ph.D. (University of Newcastle upon Tyne, Inglaterra); Áreas de Interesse: Ambientes de programação, ferramentas de apoio a projeto de circuito integrados e impressos.

1 Introdução

O presente trabalho consiste do desenvolvimento de um Gerador de Gerenciadores de Sistemas de Controle e baseia-se na notação de *"estadogramas"*[1] originalmente proposta por D. Harel [HAR 87] para representar especificações funcionais de sistemas de *hardware* ([DRU 85], [DRU 89]) em termos de diagramas de estados suportando definições hierárquicas. O gerenciador, descrito a seguir, recebe como entrada uma especificação definida através de estadogramas e gera um programa em C funcionalmente equivalente a esta especificação. O sistema completo será composto por um editor gráfico de *bolhas*[2], parte constituintes de um estadograma, que produz uma descrição textual intermediária. Esta descrição serve, então, de entrada ao gerador propriamente dito que, a partir da descrição fornecida, produz o programa em C equivalente ao estadograma originalmente definido via editor gráfico. O presente trabalho aborda a segunda parte do sistema conforme ilustrado na figura 1.

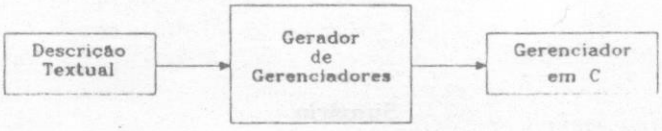

Figura 1: Descrição esquemática da ferramenta proposta

É possível observar na literatura que o enfoque estado/evento tem sido utilizado cada vez mais no contexto de Sistemas de Desenvolvimento de Interfaces Homem-Máquina adotando-se, geralmente, diagramas de transição de estados ou máquinas de estados finitos para a especificação de sistemas reativos ([MEY 89], [JAC 85], [JAC 83], [SUN 82]). O comportamento de um diagrama de estado está associado, basicamente, à mudança de um certo estado para outro após a ocorrência de um determinado evento ter sido disparado (ou acionado). Os diagramas de transição de estados, podem ser representados por "grafos dirigidos", onde os nós representam os estados e as arestas representam as transições.

Para que o enfoque estado/evento seja útil na especificação de sistemas reativos (veja em [HAR 85] e [PNU 86] discursões sobre sistemas reativos e seus comportamentos), a representação deverá permitir a descrição de estruturas hierárquicas e modulares. O estadograma é uma representação bastante flexível que consiste de uma extensão do diagrama de estado convencional abrangendo os conceitos de *hierarquia*, *concorrência* e *comunicação*. Com esta extensão permitiu-se ter novas características em diagramas de estados como, principalmente, a noção de profundidade e ortogonalidade.

[1] *Statecharts*
[2] *blobs*

Na figura 2 é apresentado um estadograma que consiste de um encaixe sucessivo de bolhas com transições entre as mesmas. O estado global do estadograma é definido pela lista de bolhas no qual o sistema se encontra em um determinado instante.

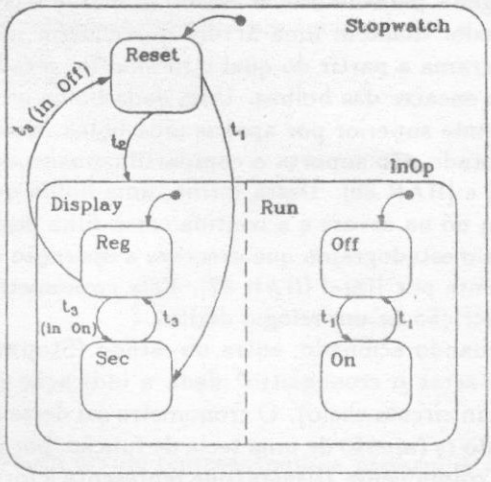

Figura 2: Estadograma de Cronômetro

No estadograma, a reatividade do sistema não é expressa simplesmente por uma mudança de seu estado, por uma ocorrência de eventos e pelo atendimento a possíveis condições associadas a estes. Também é possível associar às bolhas um *script* de ações e atividades. Uma ação pode ser executada ao se entrar e/ou sair de uma determinada bolha e uma atividade é caracterizada pela sua execução enquanto o sistema permanecer na bolha associada.

Algumas extensões e aplicações de estadogramas são abordadas em [HAR 88], [HAR 85] e [DRU 86].

2 Estrutura dos Programas Gerados

Um gerenciador construído pelo gerador, objeto deste trabalho, constitui-se de uma ferramenta que efetua mudanças do estado global de um estadograma em função da ocorrência de eventos e executa as atividades e ações associadas às bolhas envolvidas nas mudanças efetuadas. A ferramenta é gerada na linguagem C e, a seguir, são introduzidas alguns aspectos sobre a estrutura de dados do gerenciador e sua estrutura de controle.

O gerenciador mantém uma lista duplamente ligada de identificadores de bolhas, nas quais o sistema se encontra, representando esta lista, portanto, o estado global do gerenciador em um determinado instante. Esta lista é ma-

nipulada sempre que ocorre alguma mudança no estado global retirando-se da lista os identificadores das bolhas dos quais o sistema "sai" em função de um dado evento e são adicionados os identificadores das bolhas nas quais o sistema "entra", sendo executado em cada caso as ações associadas.

Além da lista ligada para manter o estado global do sistema, o programa construído pelo gerador mantém uma árvore que contém informações sobre a estrutura do estadograma a partir do qual o mesmo foi gerado. A estrutura da árvore é isomorfa ao encaixe das bolhas. Uma dada bolha sempre é contida em um nível imediatamente superior por apenas uma bolha exceto a mais externa. O gerador implementado não suporta o compartilhamento de bolhas como sugerido em [HAR 89] e [HAR 88]. Desta forma, uma bolha que contém outra é representada por um nó na árvore e a contida como filho deste nó.

A figura 2 ilustra o estadograma que descreve a operação de um cronômetro descrito funcionalmente por Harel [HAR 87]. Este cronômetro, no artigo original, faz parte da descrição de um relógio digital.

O cronômetro, quando acionado, entra no estado (Stopwatch, Reset)[3] com a ação associada de zerar o cronômetro[4] dada a indicação da entrada *default* (aresta com origem em círculo cheio). O cronômetro sai deste estado caso ocorra o evento t_1 ou o evento t_2 (aperto de uma tecla de função, por exemplo). No caso de t_1, a bolha *Sec* da componente *Display* (que representa a forma de visualização de tempo cronometrado – *Sec*: em segundos; *Reg*: em termos de horas, minutos e segundos) da bolha *InOp* é atingida concomitantemente com a bolha *Off* da componente *Run* (que representa o acionamento e a suspensão da contagem de tempo pelo cronômetro) da bolha *InOp* de acordo com a especificação da entrada *default* desta última componente. Após a ocorrência do evento t_1, o estado global transforma-se em (*Stopwatch, InOp, Display, Sec, Run, Off*). Na mudança são executadas as eventuais ações associadas à saída da bolha *Reset* e às entradas nas bolhas *InOp, Sec* e *Off*. A eventual atividade associada a *Reset* é desativada e as eventuais atividades associadas a *InOp, Sec* e *Off* são iniciadas. Caso o sistema se encontrar em *Reset* e ocorrer o evento t_2 então o estado global muda de (*Stopwatch, Reset*) para (*Stopwatch, InOp, Display, Reg, Run, Off*) de acordo com a especificação das entradas *default* das componentes de *InOp*.

A componente *Run* opera de modo independente da componente *Display*. A mudança de uma de suas bolhas para outra se dá na ocorrência do evento t_1. A componente *Display* tem o seu comportamento sujeito à ocorrência do evento t_3 e, quando em *Reg*, além da ocorrência de t_3, também depende do estado da componente *Run*. Caso a componente *Display* se encontre em *Reg* e ocorrer o evento t_3 então ocorre uma mudança para *Sec* se a componente *Run* se encontrar em *On*. Se, por outro lado, a componente *Run* estiver em *Off*, o sistema abandona *InOp* saindo de ambas as componentes e entra em *Reset* passando do estado (*Stopwatch, InOp, Display, Reg, Run, Off*) para (*Stopwatch,*

[3] O estado global é representado no texto por uma lista.

[4] As ações e atividades associadas a bolhas não são representadas na figura.

Reset).

A estrutura da árvore que representa o encaixe de bolhas do estadograma da figura 2 é apresentado na figura 3. Os nós 1, 2, 3, 4, 5, 6, 7, 8 e 9 correspondem, respectivamente, as bolhas e aos componentes *Stopwatch, Reset, InOp, Display, Reg, Sec, Run, Off* e *On* no estadograma. Os nós 4 e 7 são representados de forma distinta dos demais pois representam componentes e não bolhas.

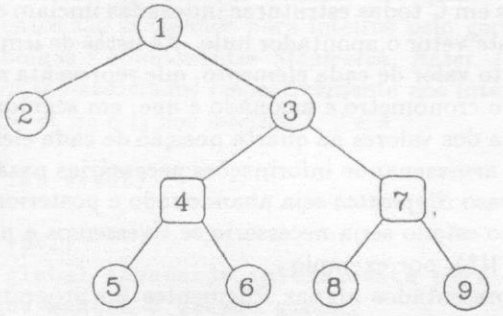

Figura 3: Estrutura da árvore representando a hierarquia do estadograma

A estrutura da árvore que representa uma hierarquia de bolhas não se altera durante a execução do programa gerado. Assim sendo uma árvore deste tipo é implementada por um vetor de vetores sendo o primeiro composto por $n+1$ elementos, onde n representa o número de nós da árvore, e o segundo com 4 elementos contendo informações que facilitam a navegação pela árvore. O primeiro elemento deste último vetor corresponde ao primeiro filho do nó representado, o segundo aponta para o próximo irmão do nó em questão e o terceiro ao seu pai. O quarto campo é utilizado para memorizar a mais recente descida de um pai para um determinado filho. Esta informação é necessária para implementar o mecanismo de *history* que consiste da especificação de retorno ao estado imediatamente anterior à saída de uma bolha, quando revisitada. Este retorno ao estado anterior pode ser limitado ao nível imediatamente inferior (H) ou a todos os níveis inferiores (H*) na hierarquia de bolhas definida a partir da bolha com este mecanismo de retorno associado.

Para o estadograma da figura 2 o vetor gerado que representa sua hierarquia é dado abaixo.

```
int hierarchy[10][4] = {{  0,  0,  0,  0},     /*  NULL  */
                        {  2,  1,  0,  0},     /*    1   */
                        {  0,  3,  1,  0},     /*    2   */
                        { -4,  2,  1,  0},     /*    3   */
                        {  5, -7,  3,  0},     /*    4   */
                        {  0,  6, -4,  0},     /*    5   */
```

```
    {  0,  5, -4,  0},      /*  6   */
    {  8, -4,  3,  0},      /*  7   */
    {  0,  9, -7,  0},      /*  8   */
    {  0,  8, -7,  0}       /*  9   */
};
```

O sinal "-" indica que o nó sendo apontado representa uma componente e não uma bolha. O primeiro elemento do vetor acima é composto por valores 0 e não é utilizado, pois em C todas estruturas indexadas iniciam com índice 0 e este valor representa neste vetor o apontador nulo. As listas de irmãos são circulares. Inicialmente o quarto valor de cada elemento, que representa os nós da árvore, é 0. Supondo-se que o cronômetro é acionado e que, em seguida, ocorra o evento t_1, então a seqüência dos valores na quarta posição de cada elemento ficaria 0, 3, 0, -4, 6, 0, 0, 8, 0, 0 armazenando informações necessárias para efetuar o retorno ao mesmo estado, caso *Stopwatch* seja abandonado e posteriormente revisitado. O retorno ao mesmo estado seria necessário se tivessemos a nível de Stopwatch uma especificação (H*), por exemplo.

A seguir são apresentados alguns fragmentos do programa equivalente ao estadograma da figura 2 para ilustrar como as transições são especificadas no código gerado. O primeiro fragmento dado abaixo corresponde ao programa principal.

```
main()
{
    current_state = InitState();
    do {
        traverse(event = GetEvent());
    } while(current_state != NULL);
}
```

O programa principal indica que, primeiramente, é constituído o estado global inicial obtido descendo-se pela hierarquia pelas entradas *default*. A seguir é executado um comando repetitivo onde, em cada iteração, é obtido o "próximo" evento e, em função do mesmo, é analisado o estado corrente para ver se este é afetado. Em caso afirmativo a árvore é percorrida de tal forma a encontrar o primeiro "ancestral" comum[5] da bolha a ser abandonada e da bolha a ser atingida. Enquanto o percurso for na direção do ancestral comum, as bolhas representadas pelos nós deste percurso são abandonadas. Uma vez alcançado o ancestral, o percurso é feito em direção ao nó que representa a bolha a ser atingida. O sistema entra nas bolhas na mesma ordem em que são visitados os nós correspondentes na árvore nesta segunda parte do percurso. O estado global é alterado em função da direção do percurso, isto é, são removidos os identificadores de bolhas da lista ligada correspondentes aos nós da árvore, quando

[5] A construção do vetor representando a árvore facilita a busca do ancestral.

o percurso é em direção ao ancestral comum, e são adicionados identificadores, quando o percurso é na outra direção.

No código da função *traverse()* são incorporadas as transições entre bolhas em função dos eventos e condições definidas no estadograma que serviu de especificação para o gerenciador gerado.

A função, portanto, é reconfigurada para cada estadograma apresentado ao gerador. No caso particular do estadograma da figura 2, o código da função *traverse()* corresponde ao apresentado abaixo. Os identificadores de bolhas, componentes e eventos são mapeados para inteiros pelo gerador. No caso particular abaixo as bolhas e componentes *Stopwatch*, *Reset*, *InOp*, *Display*, *Reg*, *Sec*, *Run*, *Off* e *On* são associados respectivamente aos inteiros 1, 2, 3, 4, 5, 6, 7, 8 e 9 e os eventos t_1, t_2 e t_3 aos inteiros 1, 2 e 3.

```
void traverse( int event)

{ lista_global *pt1;
  /* pt - var. global apontando inicialmente para o comeco
     da lista que mantem o estado global                    */
  do {pt1 = pt;
     switch(pt -> d){
        case 2: switch(event){
                    case 1: pt = fromBlobtoBlob(2,6);
                            break;
                    case 2: pt = fromBlobtoBlob(2,3);
                            break;
                }
                break;
        case 5: switch(event){
                    case 3: if (in_Blob(8))
                               pt = fromBlobtoBlob(5,2);
                            if (in_Blob(9))
                               pt = fromBlobtoBlob(5,6);
                            break;
                }
                break;
        case 6: switch(event){
                    case 3: pt = fromBlobtoBlob(6,5);
                            break;
                }
                break;
        case 8: switch(event){
                    case 1: pt = fromBlobtoBlob(8,9);
                            break;
                }
```

```
                break;
        case 9: switch(event){
                        case 1: pt = fromBlobtoBlob(9,8);
                                break;
                }
                break;
    }
    if (pt == pt1)
        pt = pt -> next;
  }while(pt != NULL);
```

A função *traverse()* percorre a lista ligada que mantém o estado global do sistema para verificar se uma dada bolha, cujo identificador se encontra na lista ligada representando este estado, é afetada pelo evento corrente que está sendo analisado. O percurso da lista é controlada pelo comando repetitivo mais externo e a verificação, se uma dada bolha é afetada ou não pelo evento corrente, é feita por uma estrutura de comando *switch* em dois níveis. No primeiro é selecionado o código correspondente à bolha, cuja sensibilidade ao evento corrente está sendo verificada, e o segundo nível corresponde ao código específico relacionado ao evento corrente associado a esta bolha. Neste nível também são verificadas as condições estabelecidas para determinadas transições como, por exemplo, as transições que têm origem na bolha *Reg* (identificada no código pelo inteiro 5) da figura 2. O percurso da árvore, que representa a estrutura do estadograma em que se baseia o programa gerado, e o controle da execução das ações e atividades associadas às bolhas envolvidas neste percurso são efetuadas pela função *fromBlobtoBlob()*.

3 Conclusão

O trabalho aqui descrito ainda não está concluído. Até o momento foi especificada a sintaxe da linguagem intermediária descritiva de estadogramas e foram elaborados diversos exemplos com geração "manual" dos programas equivalentes em linguagem C visando posterior automação deste processo pelo gerador atualmente em construção.

O gerenciador em desenvolvimento pretende ser uma ferramenta que facilite a especificação de gerenciadores de sistemas complexos privilegiando o aspecto de controle. A definição das ações e atividades é da responsabilidade do usuário e são definidas em termos de funções e processos invocadas e acionadas pelos gerenciadores gerados. Provavelmente é interessante manter para cada domínio de aplicações bibliotecas de ações e atividades específicas. Espera-se que a ferramenta em questão seja particularmente útil para desenvolver gerenciadores de diálogos em interfaces homem-máquina.

Referências

[DRU 85] Drunsinsky, D. & Harel, D., *Using Statecharts for Hardware Description*, Technical Report CS85-06, Dept. of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, December 1985.

[DRU 86] Drunsinsky, D. & Harel, D., *Statecharts as an Abstract Model for Digital control Units* , Technical Report CS86-12, Dept. of Applied Mathematics & Computer Science, The Weizmann Institute of Science, Rehovot, Israel, May 1986.

[DRU 89] Drunsinsky, D. & Harel, D., *Using Statecharts for Hardware Description and Systhesis*, IEEE Trans. on Computer-Aided Design, Vol. 8, No. 7, July 1989, pp. 798-807.

[HAR 85] Harel, D. & Pnueli, A., *On the Development of Reactive Systems*, in: K. R. Apt., Ed., Logics and Models of Concurrent Systems (Springer, New York, 1985), pp. 477-498.

[HAR 87] Harel, D., *STATECHARTS: A Visual Formalism for Complex Systems*, Science of Computer Programming, vol. 8, no. 3, june 1987, pp. 231-274.

[HAR 89] Harel, D. & Kahana, C.-A., *On Statecharts with Overlapping*, Technical Report CS89-05, Dept. of Applied Mathematics & Computer Science, The Weizmann Institute of Science, Rehovot, Israel, April 1989.

[HAR 88] Harel, D., *On Visual Formalisms*, Comm. ACM, vol. 31, no. 5, may 1987, pp. 514-530.

[JAC 83] Jacob, R. J. K., *Using Formal Specifications in the Design of a Human-Computer Interface*, Comm. ACM, 26 (1983), pp. 259-264.

[JAC 85] Jacob, R. J. K., *A State Transition Diagram Language for Visual Programming* , Computer, August 1985, pp. 51-59.

[MEY 89] Meyers, B. A., *User-Interface Tools: Introduction and Surveys*, IEEE Software, January 1989, pp. 15-23.

[PNU 86] Pnueli, A., *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends* , In Current Trends in Concurrency, Bakker, J. W. et al., Lecture Notes in Computer Sciences, vol. 224, Springer-Verlag, New York, (1986), pp. 510-584.

[SUN 82] Sunshine, C. A., et al., *Specification and Verification of Communication Protocols in AFFIRM Using State Transition Models*, IEEE Trans. Soft. Eng., 8 (1982), pp. 460-489.

RELATÓRIOS TÉCNICOS — 1990

01/90 — Harmonic Maps Into Periodic Flag Manifolds and Into Loop Groups — *Caio J. C. Negreiros.*

02/90 — On Jacobi Expansions — *E. Capelas de Oliveira.*

03/90 — On a Superlinear Sturm–Liouville Equatíon and a Related Bouncing Problem — *D. G. Figueiredo* and *B. Ruf.*

04/90 — F– Quotients and Envelope of F–Holomorphy — *Luiza A. Moraes, Otília W. Paques* and *M. Carmelina F. Zaine.*

05/90 — S-Rationally Convex Domains and The Approximation of Silva-Holomorphic Functions by S–Rational Functions — *Otília W. Paques* and *M. Carmelina F. Zaine.*

06/90 — Linearization of Holomorphic Mappings On Locally Convex Spaces — *Jorge Mujica* and *Leopoldo Nachbin.*

07/90 — On Kummer Expansions — *E. Capelas de Oliveira.*

08/90 — On the Convergence of SOR and JOR Type Methods for Convex Linear Complementarity Problems — *Alvaro R. De Pierro* and *Alfredo N. Iusem.*

09/90 — A Curvilinear Search Using Tridiagonal Secant Updates for Unconstrained Optimization — *J. E. Dennis Jr., N. Echebest, M. T. Guardarucci, J. M. Martínez, H. D. Scolnik* and *C. Vacchino.*

10/90 — The Hypebolic Model of the Mean × Standard Deviation "Plane" — *Sueli I. R. Costa* and *Sandra A. Santos.*

11/90 — A Condition for Positivity of Curvature — *A. Derdzinski* and *A. Rigas.*

12/90 — On Generating Functions — *E. Capelas de Oliveira.*

13/90 — An Introduction to the Conceptual Difficulties in the Foundations of Quantum Mechanics a Personal View — *V. Buonomano.*

14/90 — Quasi-Invariance of product measures Under Lie Group Perturbations: Fisher Information And L^2-Differentiability — *Mauro S. de F. Marques* and *Luiz San Martin.*

15/90 — On Cyclic Quartic Extensions with Normal Basis — *Miguel Ferrero, Antonio Paques* and *Andrzej Solecki.*

16/90 — Semilinear Elliptic Equations with the Primitive of the Nonlinearity Away from the Spectrum — *Djairo G. de Figueiredo* and *Olimpio H. Miyagaki.*

17/90 — On a Conjugate Orbit of G_2 — *Lucas M. Chaves* and *A. Rigas.*

18/90 — Convergence Properties of Iterative Methods for Symmetric Positive Semidefinite Linear Complementarity Problems — *Álvaro R. de Pierro* and *Alfredo N. Iusem.*

19/90 — The Status of the Principle of Relativity — *W. A. Rodrigues Jr.* and *Q. A. Gomes de Souza.*