GERAÇÃO DE CÓDIGO A PARTIR DE MODELOS COMPORTAMENTAIS ESPECIFICADOS POR STATECHARTS

Rosemeire Shibuya(*), Rosângela D. Penteado(**), Paulo Cesar Masiero(*)

(*) Instituto de Ciências Matemáticas de São Carlos - USP
Caixa Postal 668 - 13560-970 São Carlos - SP
e-mail: shibuya@icmsc.sc.usp.br
e-mail: masiero@icmsc.sc.usp.br
(**) Universidade Federal de São Carlos, IFQSC-USP
Caixa Postal 676 - São Carlos - SP
e-mail: rosangel@icmsc.sc.usp.br

Resumo

Um algoritmo eficiente para execução de statecharts, adaptado de um proposto por Drusinsky e Harel para síntese de hardware é apresentado. Esse algoritmo decompõe um statechart em um conjunto de máquinas de estados finito executando concorrentemente e comunicando-se entre si. Mostra-se também como o algoritmo é usado genericamente para, dentro do ambiente StatSim, gerar código em Occam para núcleos de sistemas reativos especificados por statecharts. Diferenças, vantagens e desvantagens deste algoritmo em relação ao de Drusinsky e Harel são discutidas e um exemplo é apresentado para ilustrar o seu uso.

Abstract

An efficient algorithm for statecharts execution, adapted from one proposed by Drusinsky and Harel for hardware synthesis is shown. This algorithm decomposes a statechart in a set of finite state machines running concurrently and communicating among themselves. It is also shown how the algorithm is used generically to generate code in Occam for reactive systems' kernels within the StatSim environment. Differences, pros and cons of this algorithm compared to the one presented by Drusinsky and Harel are discussed and an example is shown to illustrate its use.

Palavras-chaves: Sistemas Reativos, Statecharts, Occam, Gerador de Código

1. Introdução

A classe dos Sistemas Reativos tem a peculiaridade de interagir continuamente com o ambiente, reagindo a eventos externos gerados pelo processo controlado. Subclasses de sistemas desse tipo incluem os Sistemas de Tempo Real, Sistemas Embutidos e Sistemas Críticos com Relação a Segurança [AL81,DA88]. Os sistemas reativos normalmente executam concorrentemente processos paralelos e suas entradas geralmente não são previsíveis, o que torna difícil a sua modelagem e certificação de que o modelo satisfaz aos requisitos do sistema.

Algumas técnicas para especificação do aspecto comportamental (dinâmico) dos Sistemas Reativos estão baseadas em máquinas de estados finito (MEF), onde os requisitos dinâmicos são modelados através de eventos discretos no tempo, isto é, evoluindo através de passos de tempo. Dentre essas técnicas podem-se destacar os statecharts [HA87a], as máquinas hierárquicas multi-estados [GA91], as Redes de Petri [PE81] e diversos cálculos

baseados em lógica temporal, além das máquinas de estados finito convencionais. Adicionalmente, essas técnicas estão sendo incorporadas a métodos projetados especialmente para a especificação e projeto de sistemas desse tipo [WA86].

O objetivo deste trabalho é propor um algoritmo, baseado naquele apresentado em [DR89] para síntese de hardware, para a execução lógica eficiente de statecharts. Discutem-se também vantagens e desvantagens desse algoritmo e aspectos de um gerador de código que seja capaz de, dada uma descrição do statechart dentro do ambiente StatSim [MA91], criar um programa em Occam capaz de executá-lo. Dessa forma, dota-se esse ambiente de capacidade para executar protótipos de sistemas reativos. Considera-se neste trabalho que o leitor esteja familiarizado com a notação dos statecharts. Descrições informais podem ser encontradas em [HA87a] e [MA91 e a semântica formal em [HA87b].

Este trabalho está organizado da seguinte forma: na Seção 2 introduz-se o problema da geração de código a partir de modelos baseados em máquinas de estados finito e revisam-se alguns trabalhos correlatos; na Seção 3 discutem-se a arquitetura e outros aspectos do gerador que está sendo desenvolvido dentro do ambiente StatSim; um exemplo do algoritmo proposto é então ilustrado na Seção 4, usando um exemplo hipotético; na Seção 5 compara-se o algoritmo de Drusinsky e a implementação da versão proposta neste trabalho; e , finalmente, na seção 6 comenta-se sobre a situação atual deste trabalho e a sua evolução futura.

2. Geração de código a partir de modelos baseados em máquinas de estados finito

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

Esse é o mesmo princípio por trás de vários métodos atuais para especificação de Sistemas de Tempo-Real, como o de Ward [WA86], baseado na abordagem de fluxo de dados ou o de Rumbaugh [RU91], orientado a objetos. A ferramenta STATEMATE [HA90], por outro lado, além de dar apoio ao uso de um método similar ao de Ward, permite a execução do modelo, combinando statecharts e diagramas de atividades para obter um protótipo do sistema.

Há então um grande interesse em algoritmos eficientes para executar núcleos reativos, seja a partir do próprio statechart (um interpretador, por exemplo) ou então código em uma outra linguagem (um gerador de código/compilador). O próprio STATEMATE possui um módulo proprietário que gera código para ADA [HA90]. Rumbaugh oferece diretrizes simples para a criação de procedimentos a partir de modelos dinâmicos (máquinas de estados finito) e Faison oferece uma solução baseada em orientação a objetos [FA93]. No Brasil, Masiero [MA93] e Figueiredo [FI91] já estudaram esse problema e propuseram soluções que, entretanto, são relativamente ineficientes pois não tiram proveito dos componentes hierárquicos e ortogonais dos statecharts. Além do interesse em eficiência, o desenvolvimento de um gerador de código para statecharts é por si só complexo devido à complexidade semântica da notação [HA87b].

Drusinsky e Harel propõem em [DR89] os statecharts para a especificação de hardware e mostram como o modelo criado pode ser implementado como uma PLA (Programmable Logic Array). Esse algoritmo propõe a decomposição dos statecharts em máquinas de estados

finito convencionais, dispostas hierarquicamente e comunicando-se entre si, de tal forma que possam ser executadas concorrentemente, obtendo-se então grande eficiência tanto em velocidade de execução quanto em área ocupada pela PLA. Este algoritmo implementa muitos dos aspectos básicos dos statecharts, mas deixa de lado algumas características mais delicadas, como o uso de variáveis e o tratamento de inconsistências estruturais.

3. Construção de um gerador de código a partir do ambiente StatSim

O ambiente StatSim vem sendo desenvolvido pelo grupo de engenharia de software do ICMSC-USP e conta com um editor gráfico de statecharts e vários módulos para simulação do modelo: simulação interativa, simulação programada e simulação exaustiva [MA91, MA94]. Existe também um editor de diagramas de atividades [TU92] e um gerador de aplicações do tipo "Múltiplos Domínios - Múltiplas Aplicações [MA93]. A partir desse ambiente, pretende-se agora juntar o núcleo reativo especificado como um statechart e a semântica das aplicações, definidas em diagramas de atividades, de forma a ter um protótipo executável. Observe-se também que as ações semânticas podem ser omitidas e têm-se então apenas a simulação do núcleo reativo do sistema.

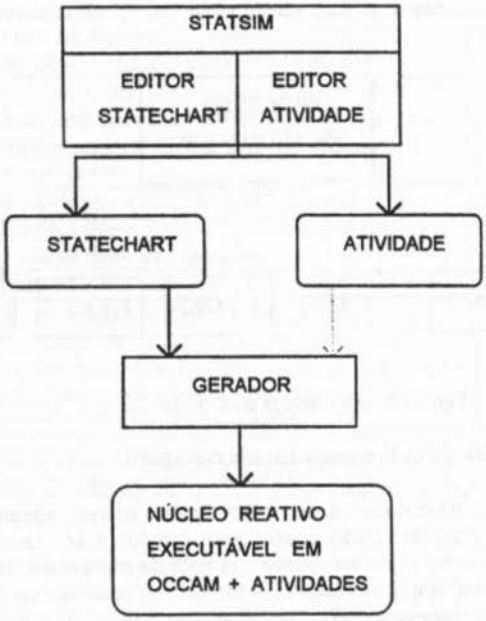

Figura 1: Módulo de Geração de Código do StatSim

A figura 1 mostra a arquitetura parcial desse módulo de geração de código. A partir de um statechart armazenado na base de dados do StatSim o gerador gera um programa em Occam que executa o statechart, de acordo com um algoritmo cujas idéias básicas são

delineadas nesta seção. Caso as ações/atividades estejam também disponíveis como código em Occam, o gerador as integra ao núcleo reativo, criando um protótipo do sistema.

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

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

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

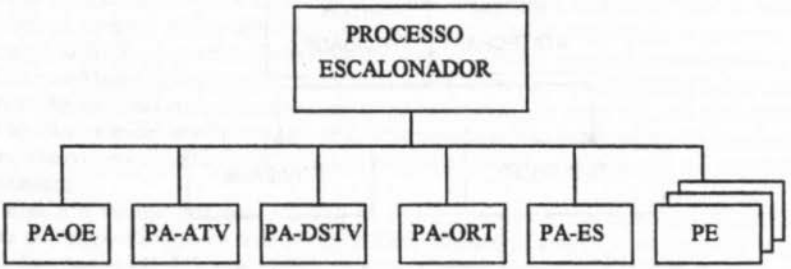

Figura 2 : Arquitetura do Código Gerado

3.1 Organização e Interface dos Processos-Estado Gerados

A figura 3 mostra um processo genérico em Occam, correspondente a um processo-estado, com todos os canais de ligação criados pelo gerador. Cada canal é descrito com detalhes na Tabela 1. A coluna "Ligações" indica que tipos de processos o canal interliga. PA-AA → PE, por exemplo, indica que o canal é utilizado por um processo auxiliar qualquer para enviar mensagens para um processo-estado.

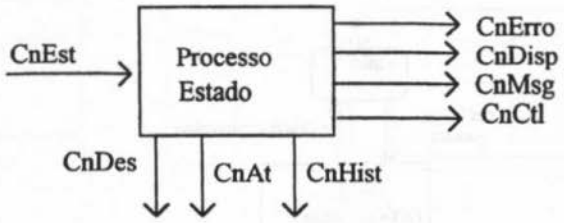

Figura 3: Representação geral em Occam de um processo

Tabela 1 - Descrição dos Canais e suas ligações.

Nome	Descrição	Ligações
CnEst	Recebe e envia eventos externos. São definidos vários canais, cada um deles representando um estado definido no statechart.	PA-ES → PA-OE PA-OE → PE PE ↔ PE
CnErro	Envia mensagens de erro para o usuário. Entre estes erros tem-se: disparo de eventos não habilitados, estado não habilitado e eventos inexistentes.	PE → PA-ES PA-OE → PA-ES
CnDisp	Envia informações para que o processo auxiliar possa disparar os eventos fornecidos pelo usuário através dos canais CnEst. É utilizado somente em componentes OR.	PE ↔ PA-ES
CnMsg	Envia dados para serem apresentados na tela. São informações a respeito do estado ativo em um determinado passo da execução.	PE → PA-ES
CnCtl	Utilizado para controlar ativação de estados.	PE → PA-ATV
CnAt	Ativa outros estados. Tem a mesma função que o canal CnDisp, mas o disparo de eventos ocorre independentemente do PA-OE, isso é, um dos canais CnEst é quem envia o evento, podendo ocorrer o disparo simultâneo de vários eventos.	PE ↔ PE
CnDes	Controla a desativação de estados.	PE → PA-DSTV
CnHist	Controla entradas e saídas de um estado, por história.	PE → PE-H

3.2 Semântica do Processamento dos Processos-Estado

Os super-estados são implementados por PEs correspondentes a cada MEF especificada ao nível de seus subestados. Os estados básicos do statechart são implementados de forma bastante simples: uma MEF composta de apenas dois estados: *Idle* e *Ativo*, sendo que no estado *Ativo*, a atividade correspondente da aplicação é executada (figura 4). Dessa forma, tem-se uma estrutura hierárquica de MEFs. Além disso, cada PE comunica-se com seu ancestral e seus descendentes via o canal CnEst, para indicar a ativação ou desativação de estados, de acordo com a semântica dos statecharts.

Figura 4: Especificação genérica de um PE básico.

O gerador produz um gabarito para cada PE da seguinte forma:

```
PROC <estado> (CHAN OF INT Cn<Est>, CnMsg,
               CnDisp, CnCtl, CnErro)
  INT x:
  SEQ
    x:=0
    WHILE x <> EVQUIT
      SEQ
        Cn<Est> ? x              (I)
        IF
          condição 1
            Trecho a ser gerado para cada
            caso da Figura 5
            .
            .
            .
          condição n
            Trecho a ser gerado para cada
            caso da Figura 5
          TRUE
            SEQ
              CnMsg ! EVINV
              CnDisp ! OK
:
```

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

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

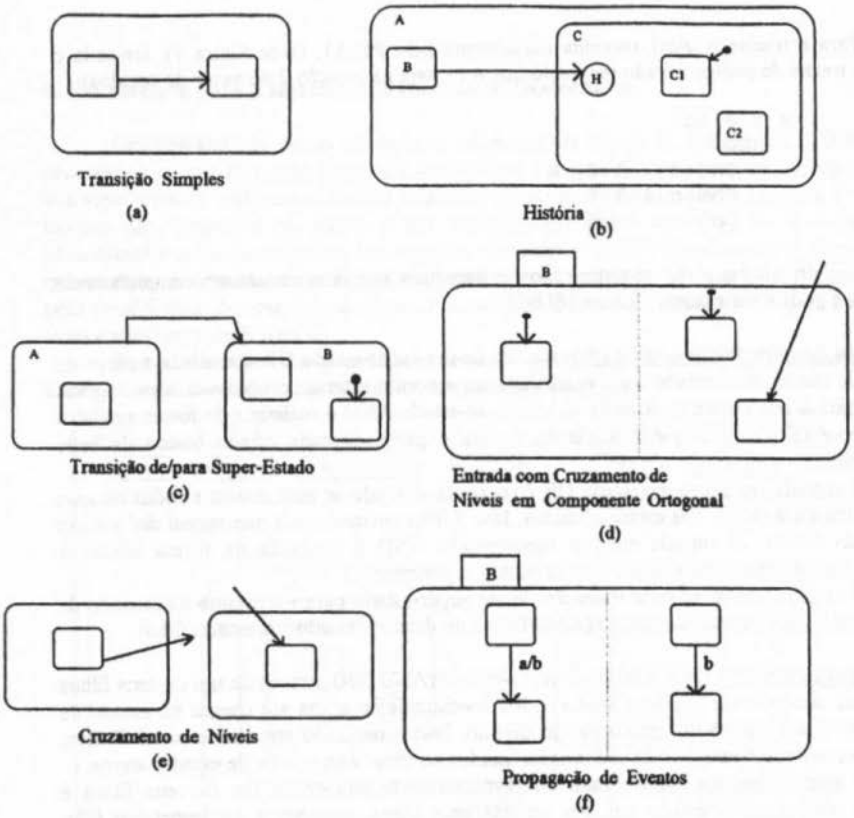

Figura 5: Casos considerados na geração de código para os PEs.

Transição Simples (5a): Analisa-se o estado origem desativando-o (se estiver ativo) através do canal CnDes e ativando-se o estado destino, através do canal CnCtl. Isso dá origem ao seguinte gabarito:

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

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

Para a transição *alfa1* recebida inicialmente pelo PE-A1, (vide figura 7), ter-se-ia o seguinte trecho de código gerado (supondo que A1 esteja na posição 2 do vetor de estados):

```
x = alfa1
  SEQ
    CnCtl ! A12; 2
    CnMsg ! A12
    CnDisp ! OK
```

Nos demais casos não são mostrados os gabaritos, que já se encontram completamente definidos e podem ser encontrados em [SH94].

Transição de/para Super-estado (5c): a saída de um super-estado OR é avaliada a partir de seu estado básico. Esse estado sente primeiramente o evento externo, processa-o, e em seguida envia-o para o seu ancestral. A saída de um super-estado AND é realizada de forma similar à componente OR, exceto que a avaliação é feita a partir de cada estado básico de cada componente.

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

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

Cruzamento de Níveis (5e): a saída de super-estado (AND/OR) através de um de seus filhos é realizada saindo-se de todos os seus estados imediatamente acima até chegar no estado de mesmo nível hierárquico do estado pai do destino. Isso é realizado através do canal CnDes, que envia mensagem desativando todos esses estados no vetor de controle de estados ativos.

A entrada em um super-estado OR explicitamente através de um de seus filhos é realizada ativando-se o estado pai, em seguida seus filhos, até chegar ao respectivo filho destino.

Cruzamento de Níveis em Componente Ortogonal (5d): a análise é feita da mesma forma que aquela mencionada anteriormente, mas neste caso, todas as outras componentes serão ativadas por default enviando-se a mensagem *def* via CnEst para os PE correspondentes às componentes ortogonais.

Propagação de Eventos (5f) : a sincronização é realizada utilizando-se o PA-OE que recebe a mensagem e a envia por um dos canais que ativa outros estados (representadas aqui pelos canais CnEst) de acordo com a ação associada.

Eventos simultâneos podem ser disparados em componentes ortogonais. Isso pode ser feito verificando se os estados origem de cada evento de entrada do sistema estão ativos naquele passo, se a resposta for verdadeira o evento é disparado. A ordem de disparo é aleatória sendo vinculada à chegada de mensagens através de um dos canais CnEst que ativam os PE.

História (5b) : é avaliada através do canal CnHist. Um processo é criado para avaliar as entradas e saídas por história. Esse processo fica aguardando a chegada de informação de um estado e, quando essa chega, envia o evento de saída para o PE destino e aguarda o retorno para esse estado.

4. Um exemplo para o algoritmo de execução de statecharts

Considerando-se como exemplo o statechart da figura 6, retirado de [DR89], são necessários quinze PEs, oito dos quais são básicos. Os sete PEs não básicos correspondentes aos super-estados estão especificados pelas sete máquinas de estados finito da figura 7. Os PEs básicos não aparecem na figura e são especificados como na figura 4. A notação de comunicação entre as máquinas é a seguinte: *<evento>/<MEF>!<mensagem>*, significando que ao disparar uma certa transição a *mensagem* correspondente é enviada à *MEF* especificada pelo canal CnEst. No exemplo da figura 7, a máquina D envia a mensagem *def* para a máquina B ao transicionar de B para C.

Para o statechart em questão, são necessários também 15 canais CnEst devido aos 15 estados existentes, cada canal ativando seu respectivo processo.

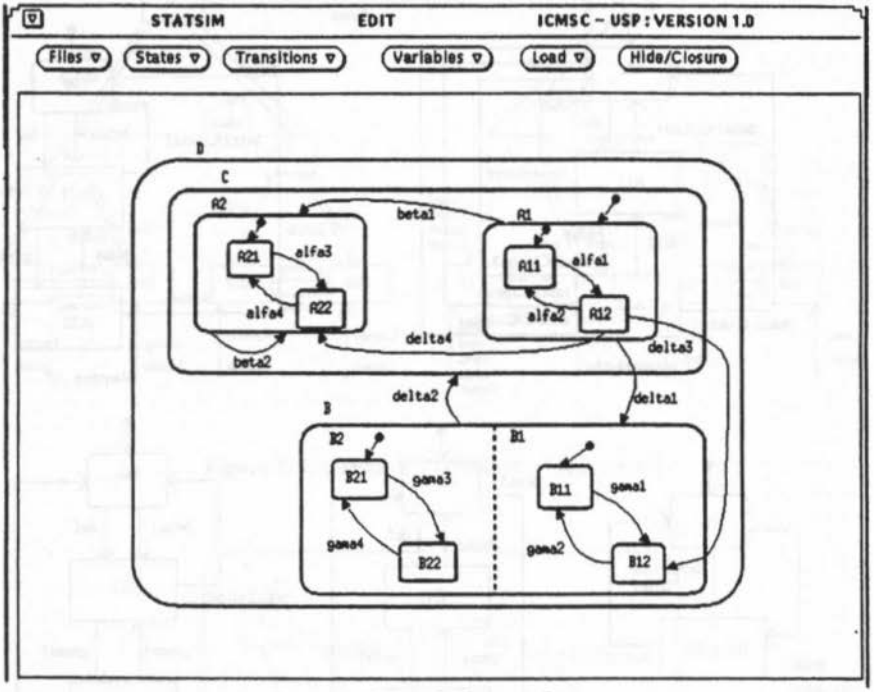

Figura 6: Exemplo de statechart

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

Outros pontos importantes do algoritmo são referentes ao tratamento de disparo de eventos simultâneos e sincronização. Eventos simultâneos podem ser visualizados através do

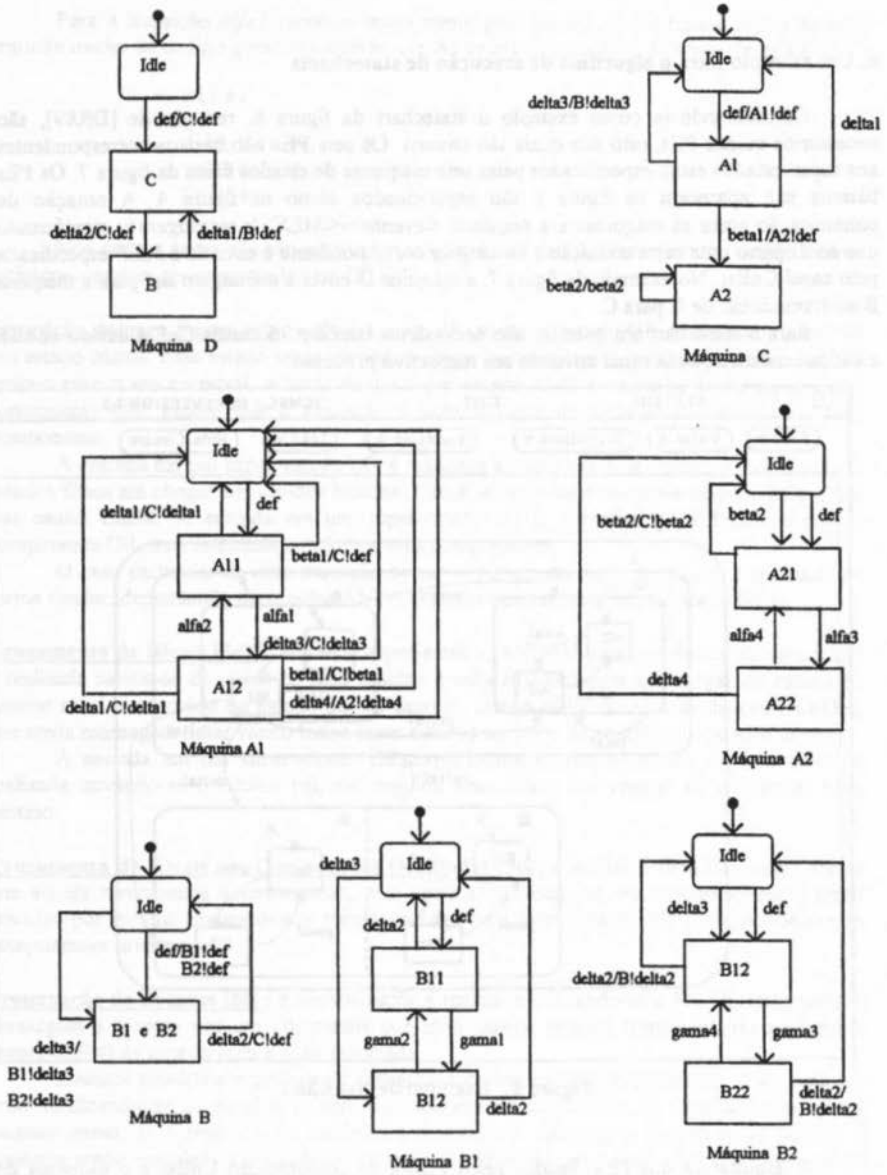

Figura 7 - As sete MEFs da figura 6

exemplo da figura 6. Suponha que a configuração ativa seja **{D, B, B1, B2, B11, B21}** e que os eventos *gama3* e *gama1* sejam recebidos simultaneamente pelo sistema em um certo

intervalo de tempo. PA-OE irá então enviar esses eventos simultaneamente para os processos **B11** e **B21**. Considerando apenas **B11**, este recebe *gama1*, desativa sua atividade, vai para o estado *Idle* e envia *gama1* para **B12** que é ativado. O mesmo ocorre simetricamente para **B21**. A configuração resultante será {**D, B, B1 B2, B12, B22**}.

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

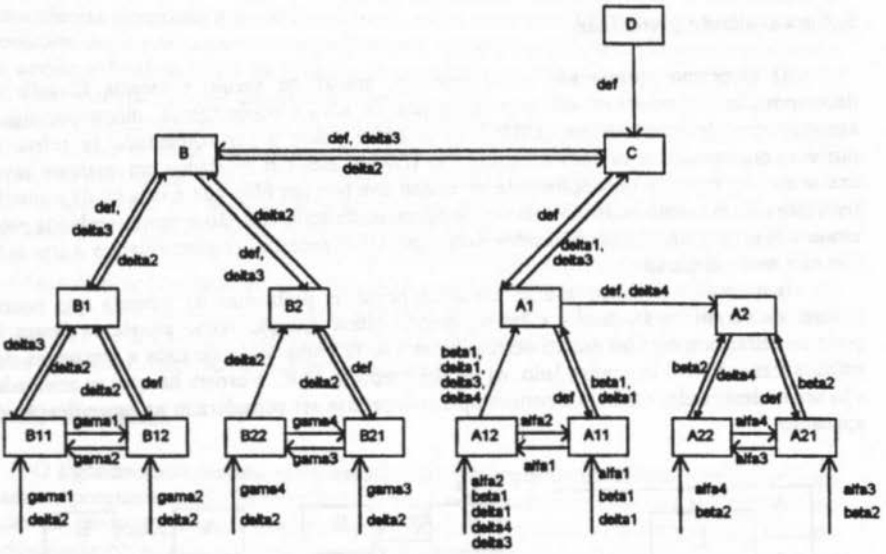

Figura 8: Esquema de Conexão das Figuras 6 e 7

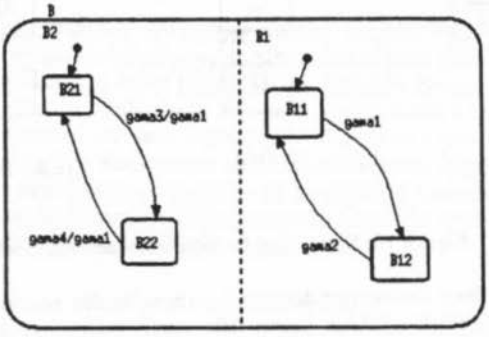

Figura 9: Representação de sincronização

A sincronização pode ser verificada através do exemplo da figura 9. Considerando que os estados **B21** e **B11** estejam ativos, ou seja, que a configuração inicial seja {**B, B1, B2, B11, B21**} ao disparar o evento *gama3* **B21** transicionará para **B22** e a ação *gama1* associada também será disparada. Esse disparo será sentido pelo processo **B22** nesse mesmo intervalo de tempo, ativando o processo auxiliar. Se **B11** estiver ativo a transição para **B12** ocorrerá, caso contrário uma mensagem de erro será enviada para o usuário.

5. Uma avaliação preliminar

O algoritmo apresentado nesse trabalho, apesar de seguir a mesma filosofia de decomposição do statechart em uma hierarquia de MEFs concorrentes, difere em alguns aspectos daquele proposto em [DR89]. O primeiro ponto a ser comentado se refere ao processo que recebe os eventos externos. Em [DR89] eles são recebidos em qualquer nível, exatamente na máquina correspondente ao estado que tem um filho que é origem da transição relevante para o evento ocorrido e na versão apresentada aqui, eles são sempre recebidos pelos estados básicos e daí são comunicados para cima até o ancestral comum relativo à transição que está sendo disparada.

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

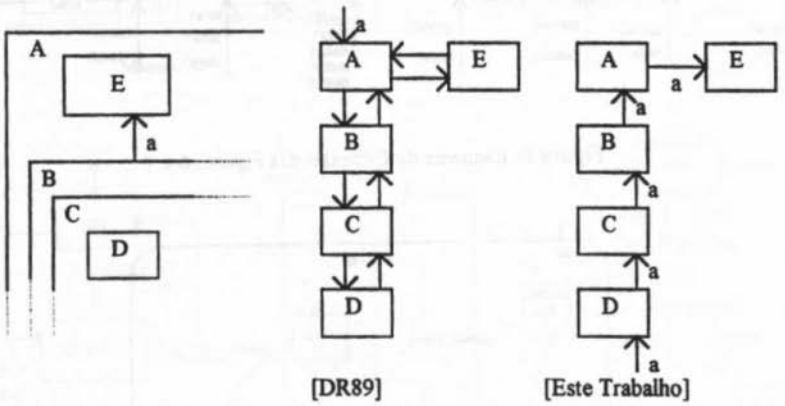

Figura 10: Sincronização entre ativação/desativação

Em função dessa mesma decisão quanto à recepção dos eventos externos, em [DR89] criam-se eventos internos artificiais (leave, leftt, enter) com o propósito de comunicar ao ancestral ou aos descendentes a decisão a ser tomada. Na versão apresentada aqui, não há necessidade de se criar novos eventos. Tudo se passa como se o statechart sofresse uma projeção para o nível mais baixo, tornando-se plano.

No nível mais baixo isso significa a necessidade de ligações entre os estados básicos (entre **B11** e **B12**, por exemplo), o que não ocorre em [DR89]. Considerando que a maior parte das transações ocorre nesse nível, tem-se uma maior eficiência a custa de mais ligações. Como neste caso as ligações são lógicas e não físicas essa desvantagem é insignificante. A principal vantagem para adoção deste esquema é tornar mais fácil a geração do código em Occam. Futuramente esta decisão poderá ser parametrizada, deixando a cargo do usuário a decisão se quer gerar o algoritmo original, a variação apresentada neste trabalho ou mesmo outras, em função de suas necessidades e da linguagem alvo.

Na versão atual do gerador não se encontram implementados o tratamento de inconsistências estruturais e o uso de variáveis, assim como em [DR89]. Na realidade, não há necessidade de se preocupar com inconsistências estruturais, pois elas podem ser verificadas pelo ambiente StatSim, antes da geração. Com relação às variáveis é perfeitamente possível fazer esta extensão, apesar de ser relativamente complexa, pois as variáveis não podem ser declaradas globalmente em Occam. Seria necessário um processo encarregado de controlar as variáveis (como se fosse um servidor de variáveis), em paralelo com os demais, que seria comunicado de qualquer alteração ou consulta aos valores das variáveis.

Nesta versão em software também não há qualquer restrição quanto ao número de subestados que um estado pode ter. No exemplo da figura 5 observe que cada estado tem apenas dois filhos. Isso é conveniente na implementação em hardware para controlar a quantidade de ligações físicas entre as máquinas, mas na versão apresentada aqui os canais são lógicos e são criados de acordo com a necessidade. Se o estado **A1** , por exemplo, tivesse mais um filho **A13** seria necessário ter a ligação entre eles e também de **A13** para com os irmãos com os quais tivesse ligação através de transições.

6. Observações finais

O algoritmo apresentado neste trabalho foi primeiramente codificado manualmente em Occam e o programa criado foi testado de forma a se ter segurança quanto à sua correção em relação ao statechart original (isto é, a especificação). Depois foram feitos vários estudos, principalmente em relação às possíveis variações de comunicação entre os processos, conforme foi discutido na seção anterior. Com base nesses estudos delineou-se o algoritmo de geração do statechart para Occam, que atualmente está sendo implementado dentro do ambiente StatSim, na linguagem C, e está funcionando parcialmente. Para execução, o programa gerado é transportado para um outro equipamento, onde tem-se um ambiente com transputers e um compilador Occam.

Como evolução deste trabalho, pretende-se estudar também uma variação deste algoritmo para geração de código em uma linguagem orientada a objetos, possivelmente C++. Mesmo num ambiente seqüencial, a idéia básica de um conjunto hierárquico de MEFs convencionais pode ser mantida, onde cada PE seria então visto como um objeto comunicando-se com seu ancestral ou seus descendentes. Vários recursos das linguagens orientadas a objetos, como a herança entre classes e a criação dinâmica de objetos deverão ser explorados, visando obter mais eficiência do algoritmo ou simplificá-lo. Uma abordagem deste tipo facilitaria no futuro o uso do ambiente StatSim para apoio a métodos orientados a objetos que utilizam statecharts, como é o caso de [CO92].

Bibliografia

[AL81] ALLWORTH, S.T. - Introduction to Real-time Software Design. London, McMillan, 1981.

[CO92] COLEMAN, D.; HAYES, F.; BEARS, S. - Introducing Objectcharts or How to Use Objectcharts in Object-Oriented Design. IEEE Transactions on Software Engineering, Vol. 18 (1):9-18,1992.

[DR89] DRUSINSKY, D.; HAREL, D. - Using Statechrts for Hardware Description and Syntesis, IEEE Transactions on Computer-Aided Design, Vol. 8 (7): 798:806, July 1989.

[DA88] DAVIS, A. M. - A comparison of Techniques for the Specification of External System Behavior. Communications of the ACM, 31 (9):1098-115, 1988.

[FA93] FAISON, T. - Object-Oriented State Machines, Software Development, 1 (3) : 37-50, September 1993.

[FI91] FIGUEIREDO Fo., A. G.; LIESENBERG, H. K. E. - Geração de Gerenciadores de Sistemas Reativos, V Simpósio Brasileiro de Engenharia de Software, Ouro Preto:31-44, 23 a 25 de Outubro de 1991.

[GA91] GABRIELIAN, A.; FRANKLIN, M. K. - Multi-level Specification of Real-Time Software, Communications of the ACM, 33 (5):50-60, May 1991.

[HA87a] HAREL, D. - STATECHARTS: A Visual Formalism for Complex Systems. Science of Computer Programming, 8:231-274, 1987.

[HA87b] HAREL,D. et all - On the Formal Semantics of Statecharts, Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, Ithaca, N.Y., 1987.

[HA90] HAREL, D. - STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering, 16 (4):403-14, 1990.

[MA91] MASIERO, P.C.; FORTES, R.P.de M.; BATISTA NETO, J. do E.S. - Edição e Simulação do Aspecto Comportamental de Sistemas de Tempo Real, XVIII SEMISH, Santos:45-61, 5 a 9 de Setembro de 1991.

[MA93] MASIERO,P.C. ; MEIRA, C.A. - Development and Instantiation of a Generic Application Generator, The Journal of Systems and Software, 23 (1):27-38, October 1993.

[MA94] MASIERO, P.C.; MALDONADO, J.C.; BOAVENTURA, I.G. - A Reachability Tree for Statecharts and Analysis of Some Properties. Information and Software Technology (aceito para publicação), 1994.

[RU91] RUMBAUGH, J. et alli - Object-Oriented Modeling and Design. Englewood Cliffs, NJ, Prentice-Hall, 1991.

[PE81] PETERSON, J. L. - Petri Net Theory and the Modeling of Systems, Englewood Cliffs, NJ, Prentice-Hall, 1991.

[SH94] SHIBUYA, R. - Geração de Código em Occam para Núcleos de Sistemas Reativos. Relatório Técnico, ICMSC-USP, 1994.

[TU92] TUTUME, R.; YSHIY, S. - Implementação de um Editor Gráfico para Diagrama de Atividades. Relatório de IC- CNPq, ICMSC-USP, 1992.

[WA86] WARD, P. T. - The Transformation Schema: an Extension of the Data Flow Diagram to Represent Control and Timing. IEEE Transactions on Software Engineering, 12 (2):198-210, 1986.