Aplicações do Compilador Estelle (*)

Wanderley LOPES DE SOUZA

GRC/DSC/Universidade Federal da Paraíba

Edilson FERNEDA

DSC/Universidade Federal da Paraíba

Av. Aprígio Veloso, s/n
58.100 - Campina Grande - PB

Sumário:

Em [FeLo 86] é descrita a arquitetura de um compilador, escrito na linguagem de programação lógica Prolog [ClMe 81], para a técnica de especificação de protocolos "Extended state transition language (Estelle)" [ISO 85], que está sendo desenvolvida junto à "International Standards Organization (ISO)" pelo grupo de trabalho "ISO TC97/SC16/WG1 subgroup B". No presente trabalho apresentamos a estrutura de dados gerada pelo compilador à partir da especificação de um protocolo em Estelle, buscando com esse exemplo enfatizar algumas aplicações para o compilador em questão. Em particular, sugerimos a sua inserção em possíveis ambientes de simulação, que visam a verificação ou a análise de desempenho parciais de sistemas descritos em Estelle.

1. Introdução

Os objetivos da "Open System Interconnection (OSI)", no que diz respeito àcompatibilidade entre protocolos de comunicação, que são empregados em sistemas computacionais heterogêneos, só poderão ser atingidos se tais protocolos forem conformes aos padrões.

O emprego de técnicas formais para a descrição de serviços e protocolos de comunicação, além de ser uma referência não ambígua para o trabalho em equipe, facilita as tarefas subsequentes de verificação, implementação e teste desses protocolos.

(*) trabalho realizado com auxílio fornecido pelo CNPq

Um compilador para a técnica de descrição formal permite a obtenção semi-automática de protótipos de implementações, cujas especificações são realizadas nessa técnica, facilitando a transportabilidade e o desenvolvimento de implementações definitivas, já que elimina boa parte da codificação personalizada.

Uma implementação semi-automática pode ser uma referência comfiável para a normalização de implementações pertencentes a ambientes computacionais homogêneos e para a comparação com o comportamento de uma implementação desenvolvida manualmente (desde que ambas sejam derivadas da mesma especificação).

Finalmente esse compilador pode ser integrado a sistemas que tenham por objetivos fornecer

(a) um certo grau de verificação (ou validação da concepção) para uma especificação em relação à proposta informal do protocolo ou

(b) estudos de simulação visando a análise de desempenho do protocolo.

2. O Compilador Estelle

Estelle é uma técnica de descrição, cujo modelo é baseado num autômato de estados finito, complementando com variáveis de estado adicionais e procedimentos. Em Estelle um sistema é definido através de um conjunto de módulos, cujas portas comunicam-se através de canais. Nesse tipo de estrutura a comunicação é indireta, o que permite especificar módulos e conexões separadamente. A especificação de um módulo pode ser realizada através da definição do seu comportamento ou do seu refinamento em submódulos. Maiores detalhes relativos a tais conceitos são encontrados em [Lopes 85].

Uma vez que Estelle é uma linguagem ainda em desenvolvimento, a versão [ISO 83] e o conjunto de regras gramaticais escolhidos para a construção do compilador, apresentam algumas diferenças em relação à versão atual. Entre elas a mais importante está relacionada à definição da arquitetura do sistema. Na nossa versão as estruturas utilizadas são estáticas, enquanto na atual os módulos são criados e liberados dinamicamente (portanto um trabalho de atualização constante do compilador se faz necessário).

O tratamento de uma especificação Estelle pelo compilador pode ser realizado fragmentando-se a especificação. A análise léxica e sintática de cada fragmento é feita por um tradutor E_telle/Pascal, que verifica também a semântica relativa aos conceitos próprios de Estelle, a sintaxe e a semântica de base do Estelle são provenientes da linguagem Pascal [JeWi 74]). O compilador, á partir de uma especificação Estelle, gera:

(a) declarações de tipos de dados representando os processos, canais e interações e as declarações de variáveis e tipos de dados usados durante a execução;

(b) procedimentos que implementam os processos capazes de manipular as transições definidas e

(c) procedimentos que inicializam o sistema através da criação e ligação das estruturas de dados.

Aos segmentos traduzidos acrescenta-se um núcleo de exploração (no caso de uma implementação semi-automática) ou um módulo de construção das interconexões do sistema e escalonamento de primitivas (no caso de uma simulação), escritos em Pascal. Cada elemento desse conjunto é submetido ao compilador Pascal, que realiza as demais verificações semânticas e fornece um conjunto de códigos objeto apropriado. Adicionando-se os códigos objeto das primitivas de comunicação (escritas em "assembly") e das rotinas de suporte, realiza-se a fusão que produzirá um arquivo executável.

Consideremos o exemplo de especificação de um protocolo de re-sincronização (Fig. 1). O sistema todo é composto de um módulo ESTACAO, cujo comportamento é descrito pelo processo EX_SIMPLES, e de um canal RE_SINCRONISMO que permite o acesso dos usuários ao serviço de re-sincronização.

As declarações geradas à partir desta especificação é mostrada na Fig. 2. Nas linhas 7-24 está declarada a estrutura S0TYPE que representa as primitivas de interação da especificação (10-18) e as sub-estruturas para a manipulação de transições espontâneas (19-23). As linhas 28-40 mostram a estrutura P0TYPE que representa todos os processos envolvidos no sistema. Para cada processo representado, estão associadas suas variáveis locais (36-39). Finalmente, nas linhas 44-47 são definidas as variáveis globais P0VAR (que contêm o endereço do processo corrente), C0VAR (canal corrente) e S0VAR (interação sendo tratada).

A especificação de um processo pode gerar 2 tipos de procedimentos: um para estabelecer a estrutura de dados associada ao processo (28-40 da Fig. 2) e outra para representar o próprio autômato (Fig. 3). Para cada uma das transições existentes na especificação, um conjunto de instruções é gerado.

Inicialmente, para prevenir-se de uma possível chamada do procedimento out, as estruturas de dados são salvas (13-14). Em seguida, a estrutura

```
with P0VAR->.D0EX_SIMPLES do begin ... end;
```

é usada para limitar o procedimento ao ambiente que contêm as variáveis do processo envolvido. Dentro dessa estrutura é que se define as transições propriamente ditas (17-42). A tradução da cláusula provided é feita pela geração de uma instrução if (com a mesma expressão booleana desta cláusula - linhas 17 e 22) acompanhada de um begin ... end, que conterá o resto da especificação da transição. Deve-se, então, identificar o canal

(18,24 e 36) e o sinal a ser tratado (17,25 e 37). Caso exista uma cláusula do tipo **when** uma estrutura **with** é gerada, indicando que os atributos da interação recebidos devem ser usados diretamente no código que implementa a transição.

Uma cláusula **out** gera um bloco de instruções (27-33), onde: é realizada uma chamada à rotina externa **PØPROCEDURE**, que localiza o canal que será usado; é criado e inicializado o sinal a ser emitido e é chamada a rotina **OUT**, que coloca o sinal na fila de recepção pertinente.

Ao final de cada uma das transições há um desvio para uma parte do código, onde é feita a liberação dos registros de dados usados na transição (45-54). Nesse ponto não há diferença entre o código usado para transições normais e espontâneas. O que as distingue é o uso do canal 0, pelas transições espontâneas, e o atributo criado pelo compilador.

O nosso exemplo é bem simples e não usa todos os recursos oferecidos pelo Estelle. Em casos mais próximos da realidade, esse código Pascal gerado pode se tornar bem mais complexo. Maiores detalhes sobre essa tradução e as regras gramaticais usadas neste trabalho podem ser encontradas em [Gerb 83].

O emprego de Prolog na construção do tradutor, além de economizar o tempo e o esforço que seriam necessários se outro tipo de linguagem tivesse sido escolhido, proporcionou um código preciso e legível, facilitando futuras modificações e/ou extensões do compilador. Em contrapartida a sua eficiência deixa a desejar, mesmo considerando a não utilização frequente de um tal compilador. O desenvolvimento de uma interface Pascal/Prolog permitiria realizar o analisador léxico através de um predicado Prolog pré-definido, escrito em Pascal, aumentando assim consideravelmente a velocidade de compilação.

3. Verificação de especificações através de simulação

Os métodos utilizados para a validação de protocolos estão normalmente relacionados às técnicas empregadas para a especificação dos mesmos. Tais métodos podem ser classificados em 3 categorias: análise lógica, simulação e teste. [Boch 86]

Na primeira categoria encontram-se métodos (análise de alcançabilidade, derivação de invariantes, provas indutivas, execução simbólica, etc.) que utilizam um certo raciocínio lógico para verificar determinadas propriedades requeridas aos protocolos. Embora produzam resultados conclusivos, a aplicação exclusiva de tais métodos em protocolos complexos tem-se mostrado impraticável.

Em contrapartida, simulação e teste não fornecem resultados definitivos, já que não podem ser aplicados exaustivamente. Entretanto eles podem ser utilizados em protocolos reais, dando-lhes uma certa credibilidade, que pode ser aumentada com análise

lógica (ou vice-versa).

Se o objetivo é utilizar o compilador Estelle para simular a execução de uma especificação, visando fornecer um certo grau de validação para a mesma, ao usuário cabe definir as propriedades a serem verificadas, ou os eventos a serem retidos para análise [JaMoGr 85]. Portanto, é importante que o sistema de simulação seja suficientemente flexível para que o usuário possa descrever as diferentes propriedades e possa ter acesso às diferentes informações durante a simulação [Groz 86].

O sistema todo, composto da descrição de vários módulos, é então compilado e o resultado será um conjunto de estruturas de dados e procedimentos. Dentre os procedimentos alguns indicarão o estado do sistema e possíveis alternativas para o progresso do mesmo, enquanto outros poderão executar ações atômicas, produzindo mudanças de estado.

As decisões a serem tomadas, no que diz respeito à execução do sistema, podem ser conduzidas segundo:

(a) uma estratégia de testes pré-definida pelo usuário. Tais estratégias podem ser programadas em Prolog e as ações associadas podem ser descritas através de um predicado Prolog (novamente a interface Pascal/Prolog será empregada);

(b) uma sequência de testes gerados automaticamente. Esta geração automática pode ser feita usando-se um sistema especialista capaz de encontrar algumas sequências de testes à partir de uma especificação de um protocolo em Estelle.[FaLi 86]

Quanto à verificação prorpiamente dita 02 alternativas são possíveis:

(a) "on line", através da execução em "paralelo" de observadores que captam informações do sistema simulado, sem interferir na sua execução ou

(b) "off line", através da análise, por um verificador externo, dos eventos retidos.

Em geral, observadores são processos executados conjuntamente com o sistema alvo e que utilizam sondas para observar os pontos de acesso dos módulos que compõem o sistema, captando assim, a informações relativas às interações e ao estado do sistema global. A fim de facilitar a programação de observadores, podemos usar os conceitos de observador local, que observa parte do sistema composto, e um observador global, que reúne o resultado de cada uma das observações locais para a avaliação do estado global do sistema. [DsBo 86]

Como ilustração do uso de observadores, utilizaremos o exemplo apresentado na Fig. 1. Nesse caso, para observar a evolução do sistema, ou seja, elementos internos do módulo e interações, definiremos 01 observador global que utiliza as informações captadas por 02 tipos de observadores locais distintos, utilizando sondas distintas (Fig. 4).

Inicialmente é definido o objeto de observação (3). Em seguida são definidos os módulos global e locais de observação com suas respectivas sondas (7-18). Nessa etapa são utilizadas certas construções não inerentes a Estelle e portanto não aceitas na versão atual do compilador. Posteriormente o comportamento desses módulos são descritos em Estelle, através de processos que efetuam a observação propriamente dita (20-30). Finalmente, são feitas as conexões necessárias para associar-se as sondas dos observadores locais aos elementos do sistema observado e o observador global aos observadores locais (32-43).

A observação pode ser passiva (como por exemplo um simples "tracing" que é mostrado na Fig. 5) de forma a não interferir na evolução da execução do sistema, ou pode ser ativa, quando o observador, em função das informações recolhidas, analisa-as e escolhe alternativas que guiarão a execução do sistema.

4. Análise de desempenho através de simulação

Em função da escolha dos parâmetros, da análise e confiabilidade desejadas, é que se decide, normalmente, o tipo de metodologia a ser empregada para verificar a performance de um sistema.

Modelos analíticos (possivelmente baseados em cadeias de Markov ou redes de filas) fornecem uma boa interpretação dos parâmetros de performance (correspondentes às diferentes partes do sistema modelado) e de suas relações. Entretanto, as limitações impostas pelo modelo, em muitos casos, não correspondem à realidade.

Os modelos que podem ser empregados em simulação geralmente são mais próximos da realidade. Infelizmente, análises baseadas em simulação requerem uma grande quantidade de recursos computacionais e os resultados obtidos são, frequentemente, de difícil compreensão.

Medidas em sistemas reais podem ser realizadas com os mesmos objetivos da simulação, ou seja, estudar os casos onde as suposições efetuadas pelos modelos analíticos não são válidas, ou para determinar se tais modelos fornecem resultados realísticos.

Se o objetivo é utilizar o compilador Estelle para simular a execução de uma especificação, visando fornecer um certo grau de análise de desempenho para a mesma, é necessário antes desenvolver uma notação, que permita a inclusão à especificação, de declarações relacionadas à performance do sistema.

Em [VaBo 84] algumas sugestões são propostas em relação à linguagem Estelle, a fim de permitir a inclusão de parâmetros de performance:

(a) o conceito de TIME, para que a duração das ações ou a

duração de ocupação dos recursos sejam expressos. A inclusão desse conceito não acarreta nenhuma alteração sintática na linguagem mas acarreta a inclusão da variável TIME ao código da implementação. Isso implica que não é possível referenciá-la diretamente em uma expressão da especificação. A cada transição essa variável é atualizada pelo "scheduler", ou pelo executor da simulação. Essa variável representa o "clock" do sistema;

(b) a solicitação de recursos via cláusula **HOLD** (relacionada às transições) e que deve bloquear uma transição enquanto o recurso solicitado não estiver disponível. A cláusula HOLD tem a seguinte forma:

hold <núm. de unidades> **units** <recurso> **of** <intervalo de tempo>

Isto significa que o recurso requerido deve estar disponível para se efetuar a transição. Caso não esteja, a transição é bloqueada até que os recursos se tornem disponíveis. A implementação desta cláusula pode ser feita com uma função **ALOCA_RECURSOS(r,u,t)**, onde **r** é o ponteiro para o recurso desejado, **u** é a quantidade de recursos requerida e **t** o tempo necessário na utilização desses recursos;

(c) a especificação de **delays**, através de funções aleatórias, para modelar não determinismos e o uso de **delays** de transmissão nos canais (que modelam as filas) e

(d) o emprego da cláusula **DELAY**, pelas transições espontâneas, para indicar que a condição deve permanecer verdadeira durante um certo tempo, antes que a transição ocorra. Pode ser empregada também para modelar o ambiente, onde **delay** corresponderá ao intervalo de tempo entre requisições externas de serviço. A cláusula DELAY <expressão> pode ser implementada através de uma função que retorna **TRUE** quando termina o intervalo de tempo estipulado quando da chamada dessa função.

5. Conclusão

Neste artigo foram apresentadas as estruturas de dados e os procedimentos gerados pelo compilador Estelle à partir da especificação de um protocolo de re-sincronização. É analisada também a inclusão desse compilador em ambientes de simulação que permitam a validação de especificações e/ou a análise de desempenho.

A simulação, embora não seja um método que garanta totalmente a validade da especificação de um protocolo, pode fornecer um certo grau de confiabilidade para o mesmo. Do mesmo modo, embora simulação não forneça resultados definitivos da análise de desempenho de um sistema, os modelos por ela tratados são mais próximos da realidade que os modelos tratados analiticamente.

Além das sugestões apresentadas, que indicam uma linha de

desenvolvimento em direção à simulação, trabalhos que visem uma melhor performance do compilador Estelle podem ser realizados.

6. Referências

[Boch 86] G.v. Bochmann, "Recent developments in protocol specification, validation and testing", publicação interna n.557, Dèp. IRO - Université de Montréal, jan-1986. Anais do 4. Simpósio Brasileiro sobre Redes de Computadores, Recife, abr-1986, pp. 354-368.

[ClMe 81] W.F. Clocksin, C.S. Mellish, "Programming in Prolog", Springer-Verlag, 1981.

[DsBo 86] R. Dssouli, G.v. Bochmann, "Conformance testing with multiple observers", VI IFIP WG6.1 International Workshop on Protocol Specification, Verification and Testing, Montréal, 1986, pp. 6.35-6.54.

[FaLi 86] J.-P. Favreau, R.J. Linn Jr, "Automatic generation of test scenario skeletons from protocol specifications written in Estelle", VI IFIP WG6.1 International Workshop on Protocol Specification, Verification and Testing, Montréal, 1986, pp. 6.1-6.12.

[FeLo 86] E. Ferneda, W. Lopes de Souza, "Compilador para a linguagem de especificação Estelle", anais do XIII SEMISH, Recife, jul-1986, pp. 420-428.

[Gerb 83] G.W. Gerber, "Une méthode d'implantation automatisée de systèmes specifiés formelement", tese de mestrado, Dép. IRO - Université de Montréal, out-1983.

[Groz 86] R. Groz, "Unrestricted verification of protocol properties on a simulation using an observer approach", VI IFIP WG6.1 International Workshop on Protocol Specification, Verification and Testing, Montréal, 1986, pp. 7.25-7.36.

[ISO 83] ISO TC97/SC16/WG1 subgroup B, "A FDT based on an extended state model", 1983.

[ISO 85] ISO TC97/SC21 DP..., "Estelle, an FDT based on an extended state transition model", 1985.

[JaMoGr 85] C. Jard, J.F. Monin, R. Groz, "Experience in implementing Estelle-X250 (a CCITT subset of Estelle) in Veda", Protocol Specification, Test and Verification V, North-Holland, 1986, pp. 315-331.

[JeWi 74] K. Jensen, N. Wirth, "Pascal: User manual and report", Springer-Verlag, 1974.

[Lopes 85] W. Lopes de Souza, "Utilização dos conceitos de módulo, porta e canal em especificações formais de serviços,

protocolos e interfaces de comunicação", anais do 3. Simpósio Brasileiro sobre Redes de Computadores, Rio de Janeiro, abr-1985, pp. 25.1-25.23.

[VaBo 84] J. Vaucher, G.v. Bochmann, "A simulation tool for formal specifications", relatório técnico de Cerbo Informatique Inc., Montréal, jul-1984.

```
const DELTA = ...;
      D1 = ...; D2 = ...;
      INC1 = ...; INC2 = ...;

type TEMPO_LOGICO = integer;

channel RE_SINCRONISMO(user);

  by user :
    MEU_TEMPO(X : TEMPO_LOGICO);

end RE_SINCRONISMO;


module ESTACAO;

  R : RE_SINCRONISMO(user);

end USUARIO;

process EX_SIMPLES for ESTACAO;

  var U,
      H : TEMPO_LOGICO;

  function ATR(MIN.MAX : integer) : boolean;
    (* retorna TRUE se já se passou um tempo entre MIN *)
    (* e MAX após a ativação do "cronômetro"           *)

  initialize
    begin
      H:=0; U:=DELTA
    end;

  trans
    provided (ATR(INC1,INC2)) and (H < U)
      begin H:=H+1 end;
    provided ATR(D1,D2)
      begin out R.MEU_TEMPO(H) end;

  trans
    when R.MEU_TEMPO
      begin if U < X+DELTA then U:=X+DELTA end;

end EX_SIMPLES;


  ST1 : ESTACAO with EX_SIMPLES[false];
  ST2 : ESTACAO with EX_SIMPLES[true];


connect
  ST1.R to ST2.R[true];
  ST2.R to ST1.R[false];
```

Fig. 1. Especificação de um protocolo de re-sincronização.

```
01 type
02     ...
03   TEMPO_LOGICO = integer;
04   CHANNEL = (C1RE_SINCRONISMO);
05   S1RE_SINCRONISMO = (S1MEU_TEMPO);
06   S1TYPE = ->S0TYPE;
07   S0TYPE = record
08               NEXT : S1TYPE;
09               case CHANNEL of
10                 C1RE_SINCRONISMO :
11                   (case T1RE_SINCRONISMO : S1RE_SINCRONISMO of
12                      S1MEU_TEMPO :
13                        (D1MEU_TEMPO :
14                           record
15                             X : TEMPO_LOGICO
16                           end
17                        );
18                   );
19                 C0ESTACAO :
20                   (case T0ESTACAO : S0ESTACAO of
21                      R1ANY :
22                        ();
23                   );
24             end;
25   PROCESS = (P0EX_SIMPLES);
26   P2TYPE = packed array [1..10] of char;
27   P1TYPE = ->P0TYPE;
28   P0TYPE = record
29               IDENT : P2TYPE;
30               CHANLIST : C1TYPE;
31               NEXT,
32               REFINEMENT : P1TYPE;
33               case PROCESS of
34                 P0EX_SIMPLES :
35                   (D0EX_SIMPLES :
36                      record
37                        U,
38                        H : TEMPO_LOGICO;
39                      end;
40                   );
41             end;
42   ...
43
44 var
45   P0VAR : P1TYPE;
46   C0VAR : C1TYPE;
47   S0VAR : S1TYPE;
48
49   ...
```

Fig. 2. Declaracões geradas pelo compilador Estelle.

```
01 procedure EX_SIMPLES;
02
03 label 1;
04
05 var
06   C1VAR : C1TYPE;
07   S1VAR : S1TYPE;
08
09 function ATR(MIN,MAX : integer) : boolean;
10   ...
11
12 begin
13   C1VAR:=C0VAR;
14   S1VAR:=S0VAR;
15   with P0VAR->.D0EX_SIMPLES do
16     begin
17       if (ATR(INC1,INC2)) and (H < U) then begin
18         if C0VAR->.IDENT = 0 then
19           if S0VAR->.T0EX_SIMPLES = R1ANY then
20             begin H:=H+1 end;
21         goto 1 end;
22       if ATR(D1,D2) then
23         begin
24           if C0VAR->.IDENT = 0 then
25             if S0VAR->.T0EX_SIMPLES = R1ANY then
26               begin
27                 begin
28                   P0PROCEDURE(1);
29                   new(S0VAR,C1RE_SINCRONISMO,S1MEU_TEMPO);
30                   S0VAR->.T1RE_SINCRONISMO:=S1MEU_TEMPO;
31                   S0VAR->.D1MEU_TEMPO.X:=H;
32                   OUT
33                 end;
34                 goto 1 end;
35               end;
36           if C0VAR->.IDENT = 1 then
37             if S0VAR->.T1RE_SINCRONISMO = S1MEU_TEMPO then
38               with S0VAR->.D1MEU_TEMPO do
39                 begin
40                   if U < X+DELTA then U:=X+DELTA;
41                   goto 1;
42                 end;
43         end;
44 1 :
45   case C1VAR->.IDENT of
46     0 : case S1VAR->.T0ESTACAO of
47           R1ANY :
48             dispose(S1VAR,C0ESTACAO,R1ANY);
49         end;
50     1 : case S1VAR->.T1RE_SINCRONISMO of
51           S1MEU_TEMPO :
52             dispose(S1VAR,C1RE_SINCRONISMO,S1MEU_TEMPO);
53         end;
54   end;
55 end;
```

Fig. 3. Procedimento gerado pelo compilador Estelle.

```
01 simulation SIM;
02
03 import M0MODULE;  (* sistema compilado separadamente *)
04
05 observation OBSVTN;
06
07   module OBS_M(OPC : boolean);
08     M : module EX_SIMPLES;
09   end OBS_M;
10
11   module OBS_C;
12     C : channel RE_SINCRONISMO;
13   end OBS_C;
14
15   module OBS;
16     O1 : array [boolean] of module OBS_M;
17     O2 : module OBS_C;
18   end OBS;
19
20   local observer OBS_LOCAL_M for OBS_M;
21     ...
22   end OBS_LOCAL_M;
23
24   local observer OBS_LOCAL_C for OBS_C;
25     ...
26   end OBS_LOCAL_C;
27
28   global observer OBS_GLOBAL for OBS;
29     ...
30   end OBS_GLOBAL;
31
32   O_M_1 : OBS_M with OBS_LOCAL_M[false];
33   O_M_2 : OBS_M with OBS_LOCAL_M[true];
34   O_C   : OBS_C with OBS_LOCAL_C;
35   O_G   : OBS   with OBS_GLOBAL;
36
37   probe
38     O_M_1.M to ST1;
39     O_M_2.M to ST2;
40     O_C     to ST1.R;
41     O_G.O1[false] to O_M_1;
42     O_G.O1[true]  to O_M_2;
43     O_G.O2  to O_C;
44
45   end OBSVTN;
46
47 end SIM;
```

Fig. 4. Especificação de um módulo de observação usando um método formal do mesmo nível do Estelle.

```
local observer OBS_LOCAL_M for OBS_M;

  var
    A : TEMPO_LOGICO;

  trans
    begin
      A:=M.H
    end;

end OBS_LOCAL_M;


global observer OBS_GLOBAL for OBS;

  trans
    begin
      writeln('h1 : ',O1[false].A,' h2 : ',O1[true].A)
    end;

end OBS_GLOBAL;
```

Fig. 5. Definição dos observadores para a função de "tracing".