UMA PROPOSTA PARA A IMPLEMENTAÇÃO DE MODELOS BASEADOS EM REDE DE PETRI A OBJETOS

RICARDO PEREIRA E SILVA *

JEAN-MARIE FARINES **

RESUMO

O objetivo deste trabalho é propor uma metodologia, baseada na abordagem de tradução, para a implementação de sistemas especificados a partir do modelo Rede de Petri a Objetos (PNO). Esta metodologia de tradução parte de uma especificação formal e gera de forma semi-automática, um programa em uma linguagem de programação alvo - que para o presente trabalho é uma linguagem de programação distribuída, chamada Linguagem de Implementação de Sistemas (LIS), cujas principais características serão apresentadas neste artigo.

As várias etapas da metodologia de tradução são descritas, como também o programa tradutor, que a automatiza.

ABSTRACT

The aim of this work is to propose a methodology for the implementation of systems specified in Petri Nets with Objects (PNO). This methodology is based on semi-automatic translation of a formal specification in a program. The target language used in the present work is a distributed programing language which is called Systems Implementation Language (LIS), whose main features will be shown in this paper.

The several stages of the translation methodology are described, as well as the translator software, which has been built.

---

* Depto. de Ciências Estatísticas e da Computação - UFSC
c.p. 476 - 88049 - Florianópolis - SC

** LCMI - Depto. de Engenharia Elétrica - UFSC
c.p. 476 - 88049 - Florianópolis - SC

1 - INTRODUÇÃO

O grande desafio colocado na questão da produção de software é o desenvolvimento de programas corretos, confiáveis e eficientes, a baixo custo. Em particular, a complexidade inerente às aplicações distribuídas exige a adoção de formalismos que além de permitirem a definição do comportamento de um sistema de forma completa, consistente, precisa, concisa e sem ambigüidades, se constitui na base formal para a validação das descrições assim obtidas e para a construção de ferramentas que permitam automatizar as várias fases do Ciclo de Vida do software.

O modelo formal Rede de Petri (RDP), muitas vezes adotado para especificar aplicações distribuídas, apresenta diversas qualidades em termos de expressividade, abstração e formalização. Do ponto de vista da expressividade, a RDP permite a representação de seqüencialidade, sincronização, concorrência e conflito. A limitação quanto à inexistência de um formalismo associado ao modelo para a representação de estruturas de dados apresentada pela Rede de Petri Ordinária, é superada pelas RDP de Alto Nível (Rede de Petri a Objetos [Sibertin 85], Rede Predicado/ Transição [Genrich 87], Rede de Petri Colorida [Jensen 86] etc). O nível de abstração permitido pelas RDP é tal que a modelagem de um sistema é completamente independente da implementação. Outrossim, o formalismo matemático utilizado no modelo, torna possível a localização de erros na especificão, a partir da análise das propriedades da rede.

No que diz respeito à implementação de RDP, duas abordagens tem sido utilizadas: a tradução e a interpretação [Cantú 90]. A abordagem baseada na tradução consiste em a partir da especificação em RDP, obter um programa numa linguagem alvo (normalmente procedural). A abordagem baseada na interpretação se utiliza de um mecanismo de tomada de decisões, responsável por procurar ciclicamente as transições sensibilizadas, escolher uma delas a partir de uma estratégia pré-definida e dispará-la, alterando a marcaçaõ da rede.

A abordagem de tradução favorece uma maior eficiência da implementação e oferece a possibilidade de automação total ou parcial da operação. Por outro lado, a abordagem de interpretação dispensa a necessidade de uma compilação a cada alteração da especificação e evita erros na passagem de uma etapa do Ciclo de Vida a outra, por ser uma execução direta da própria especificação.

A seguir será apresentada uma metodologia de implementação de RDP através da abordagem de tradução. A partir de uma especificação usando o modelo Rede de Petri a Objetos (RPO) ela permite gerar de forma semi-automática, software compilável em Linguagem de Implementação de

Sistemas (LIS) [Silva 88]. Inicialmente serão apresentados o modelo RPO e a linguagem alvo; em seguida, a metodologia de tradução proposta - ilustrada com um exemplo - assim como o programa de tradução automática.

2 - O MODELO DE REPRESENTAÇÃO E A LINGUAGEM ALVO

2.1 - REDE DE PETRI A OBJETOS (RPO) [Sibertin 85]

O modelo RPO assemelha-se ao modelo Rede Predicado/Transição proposto em [GENRICH 87].

A RPO é uma evolução do modelo da RDP Ordinária, ao qual foi acrescido um formalismo para descrição da estrutura de dados do sistema, semelhante ao utilizado pelas Linguagens Orientadas a Objetos. As fichas representam indivíduos aqui chamados objetos, que fazem parte de classes de objetos, com propriedades associadas; existem também pré-condições e ações associadas às transições.

DEFINIÇÃO [Cantú 90]:

Na definição que se segue, para qualquer conjunto S, S* denota o conjunto das n-uplas de S e P(S) o conjunto das partes de S.

A estrutura da RPO é definida por:

- Um conjunto finito de classes de objetos (**CO**), onde cada classe é representada por um nome de classe e por um conjunto de atributos (**atr**), cada um com domínio de valores; e por um conjunto finito de transições (T), de lugares (P) e de variáveis (V);

- Funções entre os lugares e as n-uplas de classes de objetos (**clp: P -> P(CO*)**), o que define a Classe de Lugares, e entre as variáveis e as classes de objetos (**clV: V -> CO**), o que define a Classe de Variáveis;

- Funções de incidência de entrada (**pre: P x T -> P(V*)**) e de saída (**pos: P x T -> P(V*)**), que definem as variáveis sobre os arcos;

- Para cada transição (**t ∈ T**):

* condições podem ser definidas através de predicados (**{Aj}**) sobre as variáveis de entrada (**Ve**);

*ações podem ser definidas através de afetações (**{Bj}**) sobre as variáveis de saída (**Vs**), (estas ações tem a forma **vs.atr <- Bj(...,ve,...)**, onde atr é um atributo de vs).

O estado da RPO é representado pela:

- Marcação (M) da rede, definida por uma função de distribuição dos objetos nos lugares da rede (M:P -> P(O*), onde O é o conjunto de instâncias de classes de objetos, cada uma definida por seu nome e por um conjunto de atributos com valor.

A evolução dos estados da RPO é descrita a seguir:

- Uma transição (t) é dita sensibilizada por uma marcação (M), quando:

* existir uma substituição (S) que associa um objeto capturado num lugar de entrada com a correspondente variável de entrada (S: V -> O / ∀ p, S(pre(p,t) ⊆ M(p));

*as condições sobre o objeto capturado foram simultaneamente verificadas (∀ ve ∈ pre(.,t), Aj(...,ve,...) = Verdadeiro).

- O disparo de uma transição sensibilizada (t) leva a uma nova marcação (M') (a marcação M' é atingida removendo objetos dos lugares de entrada de t, executando as ações associadas a t e adicionado objetos (modificação ou criação) nos lugares de saída de t).

2.2 -LINGUAGEM DE IMPLEMENTAÇÃO DE SISTEMAS (LIS) [Silva 88]

A Linguagem de Implementação de Sistemas (LIS), parte integrante do Ambiente de Desenvolvimento e Execução de Software (ADES) [Fraga 89] - desenvolvido no LCMI/ UFSC - incorpora o paradigma de programação modular apresentado em [De Remer 76]. Ela é composta por duas linguagens: a Linguagem de Programação de Componentes Elementares (LINCE) [SILVA 88] e a Linguagem de Configuração de Sistemas (LINCS) [SOUZA 88].

Através da linguagem LINCE são definidos os módulos e suas interfaces. Através da linguagem LINCS é gerada a configuração do sistema.

A linguagem LINCE foi construída a partir de uma linguagem base (na versão utilizada, a linguagem Pascal) na qual foi incorporado um conjunto de extensões: definição de módulo, declarações de tarefas, de portos, de importação de objetos, de ligação de portos e instruções de comunicação, de "loop" e de reconfiguração.

Através da linguagem LINCS é possível estabelecer uma configuração para um sistema a partir das operações de carregar/remover tipo módulo, criar/destruir instâncias e ligar/desligar portos de módulos. Outrossim é possível declarar constantes, famílias, portos para comunicação e contexto.

3 - A METODOLOGIA DE TRADUÇÃO PROPOSTA

3.1 - CONSIDERAÇÕES INICIAIS

O procedimento de tradução, de uma forma geral, parte de dois conjuntos de informação: a RDP a ser traduzida e um conjunto de blocos de código (instruções executáveis e declarações, na linguagem alvo) pré-definidos. A tradução em si consistirá em, a partir da rede, selecionar os blocos de código adequados e compor a implementação. A figura 1 ilustra a abordagem de tradução.

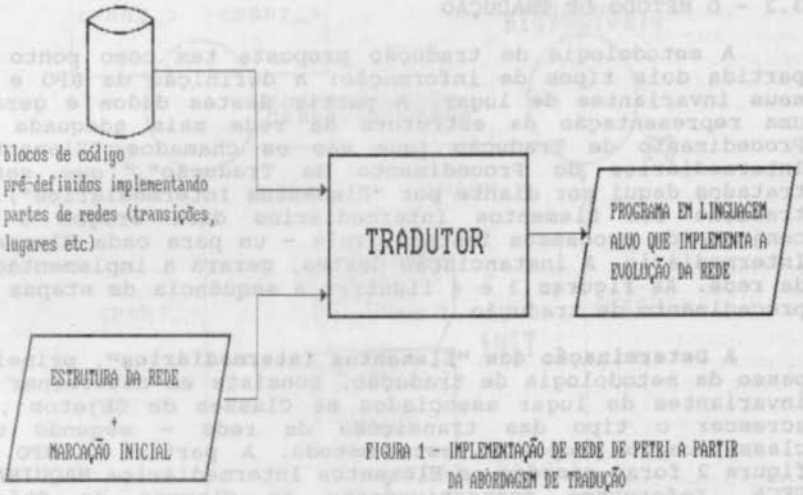

FIGURA 1 - IMPLEMENTAÇÃO DE REDE DE PETRI A PARTIR DA ABORDAGEM DE TRADUÇÃO

Para o presente trabalho foi adotado como requisito para a implementação a ser gerada, que suportasse processamento distribuído e em tempo real. Sob esta ótica, foram analisadas algumas metodologias de implementação de RDP já desenvolvidas [Bruno 86a], [Bruno 86b], [Cantú 90], [Cousin 88], [Colom 86], [Li 89], [Nelson 83]. Nenhuma delas se adequou totalmente aos requisitos estabelecidos, principalmente por razões como o uso de um modelo de representação pouco expressivo, a impossibilidade de gerar

implementação concorrente, dificuldade em automatizar o procedimento de tradução, perspectiva de chegar a uma implementação ineficiente, entre outras. Em conseqüência foi proposta uma nova metodologia [Silva 90], que será descrita a seguir.

Para ilustrar a metodologia de tradução, se utilizará um modelo RPO representando o sistema de controle de uma célula de usinagem, conforme apresentado na figura 2. A transição INPUT executa a passagem das peças do armazém (lugar P1) para o compartimento de carga/ descarga (lugar P2). As peças (pertencentes à CO PECA) são caracterizadas por uma seqüência de operações nas máquinas (pertencentes à CO MAQUINA). A transição INIT leva uma peça do compartimento de carga/ descarga para ser processada em uma das máquinas e a transição EEND a recoloca neste, após o processamento. Após a conclusão da seqüência de operações, a transição OUTPUT devolve a peça ao armazém.

3.2 - O MÉTODO DE TRADUÇÃO

A metodologia de tradução proposta tem como ponto de partida dois tipos de informação: a definição da RPO e os seus invariantes de lugar. A partir destes dados é gerada uma representação da estrutura da rede mais adequada ao Procedimento de Tradução (que são os chamados "Elementos Intermediários do Procedimento de Tradução", que serão tratados daqui por diante por "Elementos Intermediários"). A tradução dos Elementos Intermediários dará origem a um conjunto de processos instanciáveis - um para cada Elemento Intermediário. A instanciação destes, gerará a implementação da rede. As figuras 3 e 4 ilustram a seqüência de etapas do procedimento de tradução.

A **Determinação dos "Elementos Intermediários"**, primeiro passo da metodologia de tradução, consiste em determinar os invariantes de lugar associados às Classes de Objetos , e acrescer o tipo das transições da rede - segundo uma classificação proposta neste método. A partir da RPO da figura 2 foram gerados os Elementos Intermediários MAQUINA e PECA, referentes respectivamente às Classes de Objeto Máquina e Peça. A seguir é apresentado o Elemento intermediário MAQUINA:

```
MAQUINA>
P3;
INIT [ R , PECA ];
P4;
EEND [ Q , PECA ];
```

R e Q correspondem à classificação das transições INIT e EEND, respectivamente. Q, por exemplo, significa que há mais

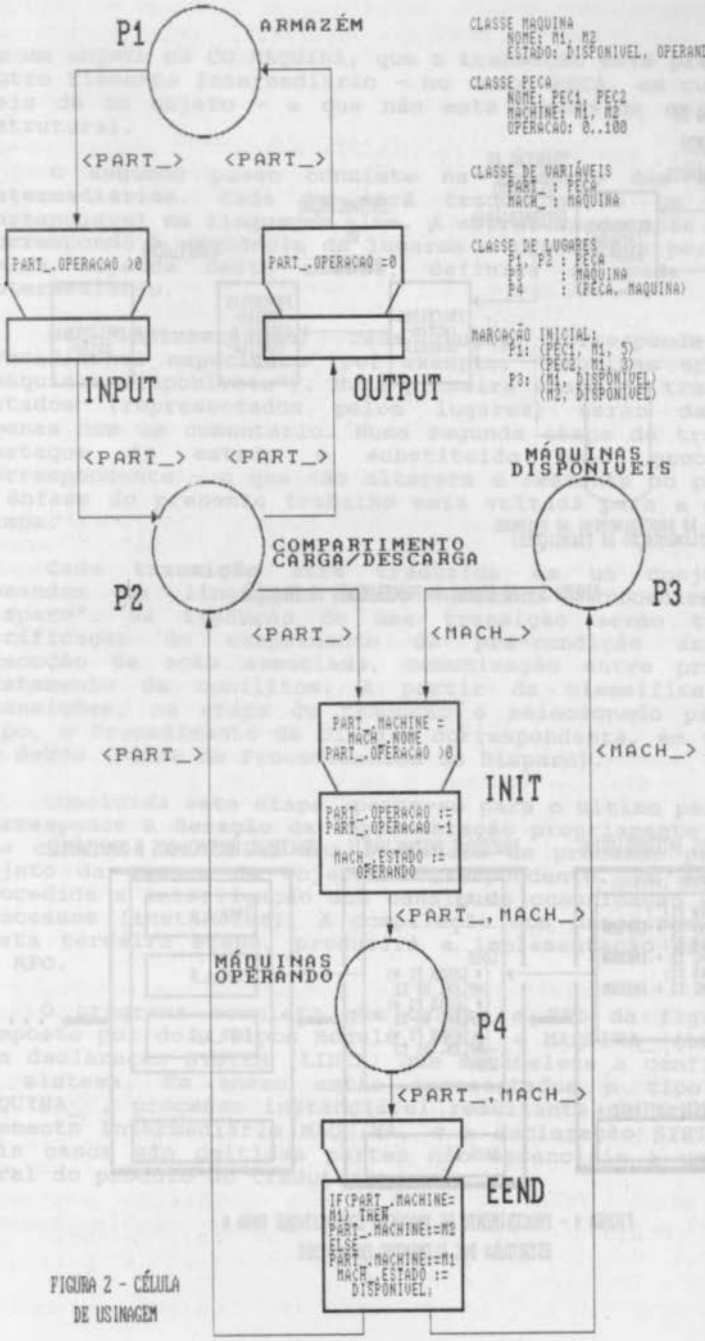

FIGURA 2 - CÉLULA DE USINAGEM

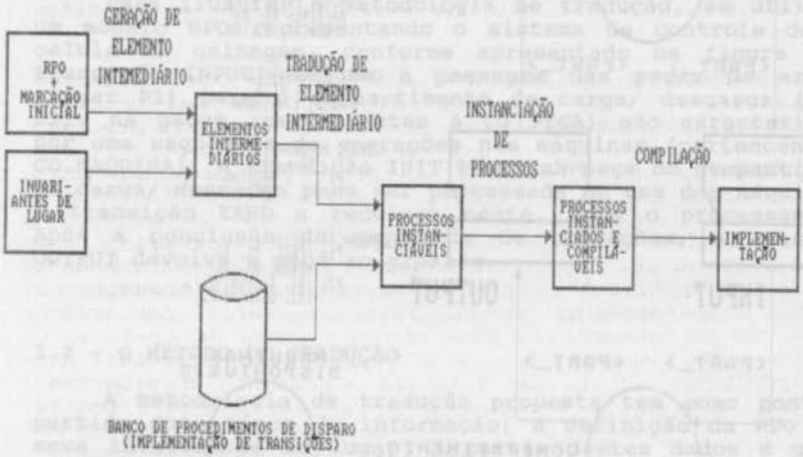

FIGURA 3 - AS ETAPAS DO PROCEDIMENTO DE TRADUÇÃO

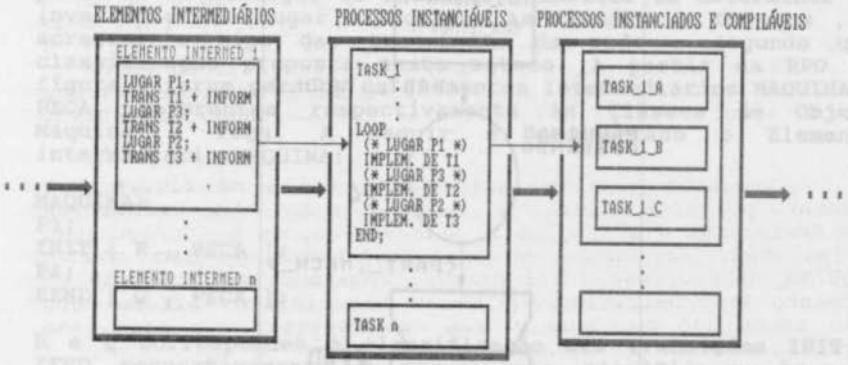

FIGURA 4 - PROCEDIMENTO DE TRADUÇÃO COM DESTAQUE PARA A ESTRUTURA DOS ELEMENTOS ENVOLVIDOS

de um objeto na CO MAQUINA, que a transição está presente em outro Elemento Intermediário - no caso, PECA, em cuja CO há mais de um objeto - e que não está envolvida em conflito estrutural.

O segundo passo consiste na **Tradução dos Elementos Intermediários**. Cada um será traduzido em um processo instanciável na linguagem alvo. A estrutura de cada processo corresponde à seqüência de lugares e transições percorridos pelos objetos desta classe, definida em cada Elemento Intermediário.

Na implementação, cada **lugar** corresponde a um procedimento específico (por exemplo, "máquinas operando", "máquinas disponíveis"). Numa primeira etapa de tradução os estados (representados pelos lugares) serão destacados apenas com um comentário. Numa segunda etapa de tradução o destaque do estado é substituído pelo procedimento correspondente - o que não alterará o restante do programa. A ênfase do presente trabalho está voltada para a primeira etapa.

Cada **transição** será traduzida em um conjunto de comandos da linguagem alvo, chamado "Procedimento de Disparo". Na tradução de uma transição serão tratados: verificação do cumprimento da pré-condição associada, execução da ação associada, comunicação entre processos, tratamento de conflitos. A partir da classificação das transições, na etapa de tradução é selecionado para cada tipo, o Procedimento de Disparo correspondente, em um banco de dados (Banco de Procedimentos de Disparo).

Concluída esta etapa, parte-se para o último passo, que corresponde à **Geração da Implementação** propriamente dita, o que consiste em criar uma instância de processo para cada objeto da Classe de Objetos correspondente. Em seguida é procedida a interligação dos canais de comunicação entre os processos (instâncias). A compilação dos processos obtidos nesta terceira etapa, produzirá a implementação executável da RPO.

O programa completo que traduz a RPO da figura 2 é composto por dois Tipos Módulo, PECA_ e MAQUINA_ (LINCE), e uma declaração SYSTEM (LINCS) que estabelece a configuração do sistema. Em anexo estão apresentados o tipo módulo MAQUINA_ , processo instanciável resultante da tradução do Elemento Intermediário MAQUINA, e a declaração SYSTEM. Nos dois casos são omitidas partes não essenciais a uma visão geral do produto do tradutor.

3.3 - O PROGRAMA TRADUTOR

Foi elaborado um programa tradutor, implementado em linguagem Pascal, que a partir da descrição da RPO tratada (contida em arquivo), gera um conjunto de arquivos compiláveis, na linguagem alvo.

A sua execução gera inicialmente o conjunto de Elementos Intermediários. A seguir ocorre a busca no Banco de Procedimentos de Disparo (um arquivo) dos Procedimentos de Disparo que implementarão as transições da rede, assim como as declarações (TYPE, VAR e PORT) e procedimentos (Procedures PASCAL) associados. A partir destes elementos é composta a implementação.

Antes da compilação é necessária a intervenção do usuário, que deverá introduzir as pré-condições e ações das transições da rede, no programa gerado.

4 - AVALIAÇÃO DA METODOLOGIA E DA FERRAMENTA DE TRADUÇÃO

A metodologia desenvolvida cumpriu os requisitos estabelecidos, porém, na atual fase de desenvolvimento apresenta limitações: o conjunto de Procedimentos de Disparo desenvolvidos não abrange todas as possibilidades de situação em que podem estar envolvidas as transições de uma RPO; não é completamente automatizável, pois é necessária a intervenção do usuário antes da compilação do software gerado; em situações de comunicação que demandam grande número de mensagens entre processos ocorrerá "overhead".

Foi desenvolvido um protótipo para o tradutor, que se encontra em fase de testes. A partir deste protótipo já se procedeu a tradução de algumas RPO. Os programas gerados pelo tradutor foram compilados e executados, operando conforme especificado.

5 - CONCLUSÃO

No presente trabalho foram apresentados metodologia e ferramenta para produção automatizada de software, na fase de implementação.

A metodologia proposta de tradução de modelos baseados em RPO, é a mais independente possível da linguagem alvo, de modo que possa ser utilizada com outras linguagens além da Linguagem LIS.

Na atual fase já está sendo produzido software compilável a partir do tradutor desenvolvido (que implementa

a metodologia proposta), de forma semi-automática. Dos testes que foram efetuados, prevê-se extensões ao método, no sentido de resolver os problemas citados anteriormente.

6 - BIBLIOGRAFIA

[Bruno 86a]: Bruno,G.; Balsamo,A., (1986). **"Petri Net-Based Object Oriented Modelling of Distributed Systems"**. OOPLSA'86 Conference, 284-293.

[Bruno 86b]: Bruno,G.; Marchetto,G., (1986). **"Process Translatable Petri Nets for The Rapid Phototyping of Process Control Systems"**. IEEE Transactions on Software Engineering, vol.SE-12, no.2, 346-357.

[Cantú 90]: Cantú,E.; Farines,J-M.; Garnousset,H.E., (1990). **"Implementação de Especificações de Sistemas Descritos por Rede de Petri a Objetos"**. Anais do 8o. Congresso Brasileiro de Automática, Belém, PA.

[Cousin 88]: Cousin,B.; Estraillier,P., (1988). **"Generation of Ada Code from Petri Nets Models"**. Rapport Laboratoire MASI/CNRS, Paris, France.

[Colom 86]: Colom,J.M.; Silva,M.; Villarroel,J.L., (1986). **"On Software Implementation of Petri Nets and Colored Petri Nets using High-Level Concurrent Languagers"**. 7th Workshop on Application and Theory of Petri Nets, Zaragoza, Spain.

[DeRemer 76]: DeRemer,F.; kron,H.H., (1976). **"Programming-in-the-Large Versus Progamming-in-the-Small"**. IEEE transactions on Software Engineering, vol.SE-2, N.2, 80-86.

[Fraga 89]: Fraga,J.S.; Farines,J-M.; Abreu,W.M.; Silva,E.S.; Nacamura,L.; Coelho Filho,O., (1989). **"ADES: Ambiente de Desenvolvimento e Execução de Software Distribuído"**. Anais do Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos: 151-158, Florianópolis, SC.

[Genrich 87]: Genrich,H.J., (1987). **"Predicate/Transition Nets"**. Report of Institut fur Methodiche Greendlagen, R.F.Germany.

[Jensen 86]: Jensen,K., (1986). **"Coloured Petri Nets"**. Lecture Notes in Computer Science, Springer-Verlag.

[Li 89]: Li,P.; Von Thum,M.; Dillon,T.S., (1989). **"Semiautomatic Implementation of Communication Protocolos from a Petri Net Based Specification Language Description"**. Proceedings of the 2th International Conference on Formal Description Techniques.

[Nelson 83]: Nelson,R.A.; Haibt,L.; Sheridan,P.B., (1983). **"Casting Petri Nets into Programs"**. IEEE Transactions on Software Engineering, vo.19, no.5, 590-602.

[Sibertin 85]: Sibertin-Blanc,C., (1985). **"High-level Petri Nets with Data Structure"**. 6th European Workshop on Applications and Theory of Petri nets, Helsinski, Finland.

[Silva 88]: Silva,E.S., (1988). **"Uma Linguagem de Programação de Componentes Elementares para Aplicações Distribuídas em Tempo Real: Projeto e Implementação"**. Dissertação de Mestrado do DEEL/UFSC, Florianópolis, SC.

[Silva 90]: Silva,R.P., (1990). **"Uma Proposta para a Implementação de Modelos Baseados em Rede de Petri a Objetos"**. Dissertação de Mestrado do DEEL/UFSC, Florianópolis, SC.

[Souza 88]: Souza,L.E., (1988). **"Um Suporte para a Configuração Estática de Sistemas Distribuídos Utilizando Abordagem por Linguagem: Projeto e Implementação"**. Dissertação de Mestrado do DEEL/UFSC, Florianópolis, SC.

```
****************** TIPO MÓDULO MAQUINA_ ******************
MODULE MAQUINA_;

TYPE
NOME_OBJETO = ( M1, M2,PEC1, PEC2);
MAQUINA = RECORD
              NOME : M1..M2 ;
              ESTADO : (DISPONIVEL, OPERANDO) ;
          END; ( MAQUINA )
PECA = RECORD
           NOME : PEC1..PEC2 ;
           MACHINE : M1..M2 ;
           OPERACAO : 0..100 ;
       END; ( PECA )
CODIGO_DE_TRANSICAO = RECORD
                          ORDEM : 0..100 ;
                          IDENT : NOME_OBJETO ;
                      END ; ( CODIGO_DE_TRANSICAO )
                          .
                          .
                          .
PORT
MINIT_PT1 : OUT ( OB_COD  ) ;
MINIT_PT2 : IN  ( ID_COD  ) ;
MINIT_PT3 : OUT ( ID_COD  ) ;
                          .
                          .
                          .
TASK T_MAQUINA ;

TYPE
NOME_TRANSICAO = ( INIT  , EEND  ) ;
                          .
                          .
                          .
VAR
PART_                     : PECA ;
MACH_                     : MAQUINA ;
CODIGO                    : CODIGO_DE_TRANSICAO ;
REM                       : NOME_OBJETO ;
                          .
                          .
                          .
PORT
INIT_PT1 : OUT ( OB_COD  ) ;
INIT_PT2 : IN  ( ID_COD  ) ;
INIT_PT3 : OUT ( ID_COD  ) ;
                          .
                          .
                          .
```

```
BEGIN
                                    .
                                    .
                                    .
LOOP
(  LUGAR          : P3 )
(  TRANSICAO      : INIT  )
          GERA ( INIT_CODIGO ) ;
          MENS_OBCOD.OBJETO := MACH_
          MENS_OBCOD.CODIGO := INIT_CODIGO
          SEND MENS_OBCOD TO INIT_PT1 ;
          REPEAT
               RECEIVE MENS_IDCOD FROM INIT_PT2 ;
          UNTIL ( MENS_IDCOD.CODIGO = INIT_CODIGO ) ;
          SEND MENS_IDCOD TO INIT_PT3 ;
          REPEAT
               RECEIVE MENS_OBCOD FROM INIT_PT4 ;
          UNTIL ( MENS_OBCOD.CODIGO = INIT_CODIGO ) ;
          MACH_ := MENS_OBCOD.OBJETO
(  LUGAR          : P4)
(  TRANSICAO      : EEND  )
                                    .
                                    .
                                    .
END; ( LOOP )
END; ( BEGIN )

LINK
MINIT_PT1 TO INIT_PT1;
MINIT_PT2 TO INIT_PT2;
MINIT_PT3 TO INIT_PT3;
                                    .
                                    .
                                    .
ENDMODULE.

********************* "SYSTEM" CELULA **********************
SYSTEM CELULA22;
       USE
          MAQUINA;
          PECA;
       CREATE AT STATION [ 0 ]
          M1, M2      : MAQUINA;
          PEC1, PEC2 : PECA;
       LINK
          PEC1 . MINIT_PT2  TO  M1   . MINIT_PT2 ;
          PEC1 . MINIT_PT2  TO  M2   . MINIT_PT2 ;
          PEC1 . MINIT_PT4  TO  M1   . MINIT_PT4 ;
          PEC1 . MINIT_PT4  TO  M2   . MINIT_PT4 ;
                                    .
                                    .
                                    .
       END.
```