Boletim Técnico da Escola Politécnica da USP

Deptº de Engª de Computação e Sistemas Digitais

BT/PCS/9305

Uma Ferramenta para o Desenvolvimento de Protótipos de Programas Concorrentes

Jorge Kinoshita
João José Neto

São Paulo - 1993

O presente trabalho é baseado na dissertação de mestrado apresentada, em 1991, pelo Engº Jorge Kinoshita, sob orientação do Prof. Dr. João José Neto: "Uma Ferramenta para a Construção de Programas Concorrentes".

A íntegra da dissertação encontra-se à disposição com o autor e na biblioteca de Engenharia de Eletricidade da Escola Politécnica da USP.

Kinoshita, Jorge
Uma ferramenta para o desenvolvimento de protótipos de programas concorrentes / J. Kinoshita, J. José Neto. -- São Paulo : EPUSP, 1993.
p. -- (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia de Computação e Sistemas Digitais, BT/PCS/9305)

1. Programação concorrente (Computadores) 2. Redes de Petri (Computadores) I. José Neto, João II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais III. Título IV. Série CDD 005.43 003

UMA FERRAMENTA PARA O DESENVOLVIMENTO DE PROTÓTIPOS DE PROGRAMAS CONCORRENTES

Autores: Jorge Kinoshita e João José Neto
Escola Politécnica - Depto. de Eletricidade.
Cidade Universitária CEP 5508 São Paulo - SP.
Cx. Postal 11455 fone 815-9322 ramal 3392

Resumo

Este trabalho apresenta uma ferramenta de auxílio ao projeto de programas concorrentes, baseada em especificações através de Redes de Petri. A ferramenta foi desenvolvida em microcomputador compatível com IBM-PC, em ambiente Smalltalk-V. São mostrados alguns exemplos de utilização dos recursos da ferramenta, ilustrando seu emprego no desenvolvimento de protótipos.

Abstract

The present work shows a Petri net based tool intended to help designing concurrent software. It has been built under the Smalltalk-V environment, running on an IBM-PC compatible machine. Some significant examples are presented, illustrating the use of the tool's facilities in developping concurrent software prototypes.

Palavras-chave: Programação concorrente, Smalltalk e redes de Petri.

1. Introdução

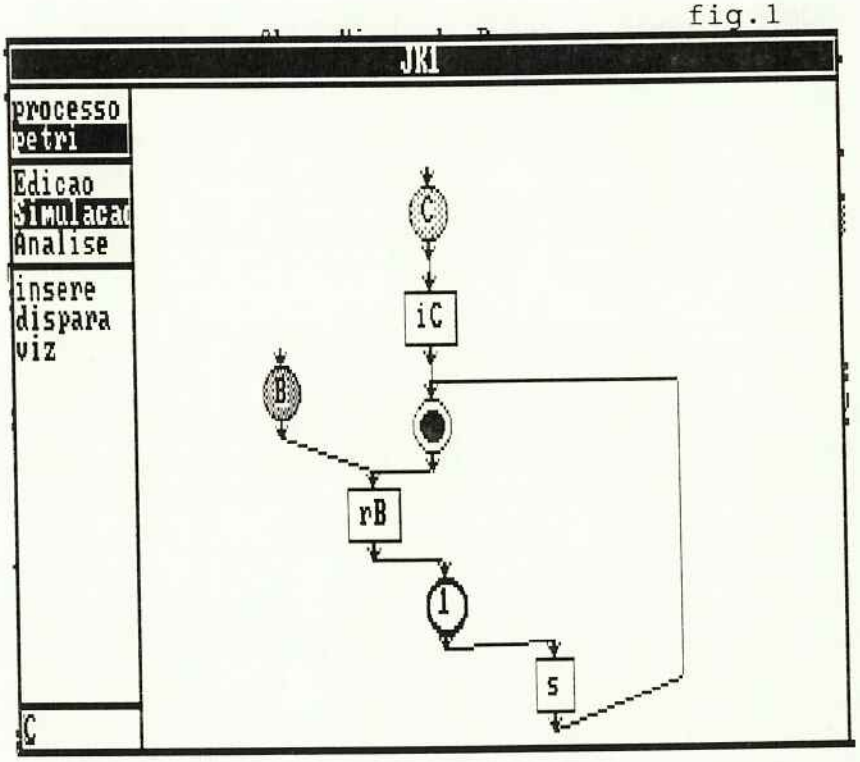
Neste artigo é apresentada uma ferramenta de apoio à fase de projeto de um programa concorrente, onde são descritos os processos que o compõem e como eles interagem entre si. A ferramenta permite a simulação e análise da descrição em questão com o objetivo de se prever erros no programa final, como "deadlocks" ou protocolos mal definidos. Este artigo tratará com ênfase a operação de simulação.

2. A Ferramenta

A ferramenta, desenvolvida como trabalho de mestrado [Kino-91], é gráfica e permite a edição, simulação e análise de protótipos de programas concorrentes. Baseia-se essencialmente nos conceitos de programação concorrente, redes de Petri [Murata-89], e Smalltalk [Gold-83].

Foi construída no ambiente Smalltalk-V e é vista como uma das janelas deste ambiente (fig. 1).

fig.1



A edição de protótipos é efetuada em dois passos. O primeiro passo consiste em descrever os processos que compõem o programa concorrente e a forma como eles estão relacionados, através de uma rede de processos. O segundo passo consiste em associar uma rede de Petri a cada processo que compõe a rede de processos.

Através da rede de processos e das redes de Petri, é possível obter uma rede de Petri global que descreva todo o programa concorrente. A ferramenta permite gerar um arquivo chamado "petri" contendo uma descrição desta rede de Petri global.

A ferramenta apresenta dois programas de análise de redes de Petri, que têm o arquivo "petri" como entrada.

O primeiro programa fornece os invariantes da rede de Petri e faz a análise de "deadlock". O algoritmo é apresentado em [Bressan-85]. Utilizando os invariantes de uma rede de Petri é possível prever o uso inadequado de semáforos na sincronização de processos.

O segundo programa testa se uma seqüência de disparos de transições é possível. Entre duas transições consecutivas que compõem esta seqüência, podem ocorrer outras transições que

devem ser disparadas. Erros no protocolo entre dois ou mais processos podem ser detectados através deste teste.

O usuário da ferramenta tem as opções de simular o protótipo criado disparando transições nas redes de Petri. Para que a simulação se aproxime mais da realidade, a ferramenta oferece a possibilidade de designar código Smalltalk, a ser executado na ocasião do disparo das transições.

A análise de redes de Petri não será discutida neste artigo. É apresentada a maneira como código Smalltalk pode ser associado às transições das redes de Petri.

2.1 - Edição do Protótipo

A edição de um protótipo, com a ajuda da ferramenta, consiste em descrever:

- redes de processos: especificam os processos que compõem o programa concorrente e suas mútuas dependências.
- redes de Petri: descrevem as estruturas de controle de cada processo.

A ferramenta emprega uma área da tela do microcomputador, onde são representadas graficamente as redes de processos e as redes de Petri.

Neste ítem mostra-se a forma como as redes de processos e as redes de Petri são representadas na tela do microcomputador. Esta especificação leva em conta a facilidade de representação de figuras no ambiente Smalltalk.

2.1.1 - Edição de redes de processos

Uma rede de processos é um grafo biconexo direcionado, onde os nós podem ser de um dos dois tipos: processos ou canais. Processos podem ser conectados somente a canais e vice-versa. A palavra canal geralmente aparece relacionada apenas à troca de mensagens. Neste artigo, contudo, empregaremos a palavra canal para designar "mail-boxes" ou semáforos, devido à mesma representação adotada para ambos em redes de Petri.

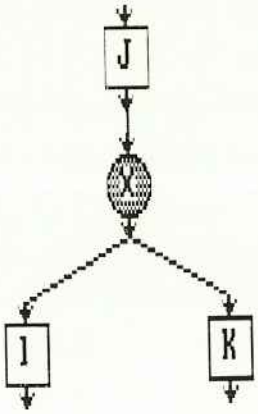
2.1.1.1 - Elementos da rede

Os elementos (nós) da rede representam processos e canais. Os processos são representados na tela através de retângulos, e os canais, através de elipses.

2.1.1.2 - Relação entre os elementos

Processos comunicam-se compartilhando canais. A representação de dois processos J e K, que se comunicam pelo canal X, é mostrada na seguinte figura:

Um canal pode ser compartilhado por mais de dois processos. Exemplo:



2.1.2 - Redes de Petri

Uma rede de Petri associada a um processo, apresenta na ferramenta, uma representação que segue as seguintes restrições:

1 - Não deve permitir o paralelismo, ou seja, deve ser estritamente seqüencial.

2 - Deve ser fácil uní-la a redes de Petri de outros processos de forma a obtermos a rede de Petri global do sistema.

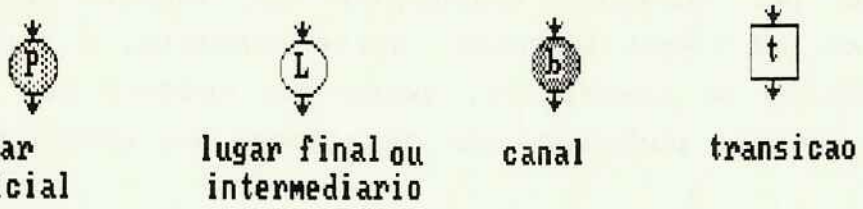
3 - Deve ser possível representá-la na tela do computador. Para isto, a rede de Petri será representada em níveis diferentes.

Uma rede de Petri possui o mesmo nome do processo associado.

2.1.2.1 - Elementos da rede de Petri

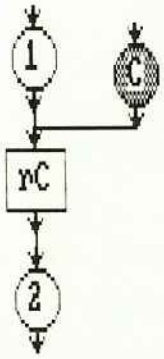
A representação de um processo na ferramenta, em redes de Petri, é basicamente uma rede de Petri restrita, onde cada transição possui apenas um lugar como entrada e apenas um lugar como saída, ou seja, é da forma de uma máquina de estados, evitando-se, deste modo, o paralelismo. A fim de se representar a comunicação entre processos, as máquinas de estados serão estendidas para ligarem-se umas com as outras através de lugares em comum, aos quais denominaremos canais. Uma rede de Petri associada a um processo será então, do tipo Máquina de Estados Estendida (M.E.E.)

Em uma MEE existe um lugar inicial, lugares finais e lugares intermediários, dependendo de como ela é executada, como ocorre em máquinas de estados. Lugares intermediários e finais são representados através de elipses brancas. A representação do lugar inicial é diferenciada através da cor cinza claro. Um lugar é identificado como lugar inicial quando possuir o mesmo nome da rede de Petri em que se encontra. Uma MEE pode ser ligada a outra MEE através de canais quando as duas MEE's se comunicam entre si. Desta forma, os elementos que compõem uma rede de Petri associada a um processo são:

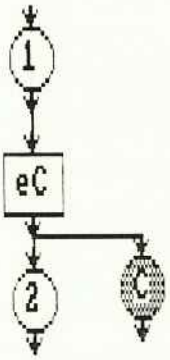


2.1.2.2 - Interligação entre MEE's

Um processo P que coloca uma mensagem em um canal C (ou executa uma operação V sobre o semáforo C) deve possuir o seguinte tipo de transição:



Um processo P que retira uma mensagem de um canal C (ou executa uma operação P sobre o semáforo C) deve possuir uma transição do seguinte tipo:



2.1.2.3 - Representação por níveis.

Uma rede de Petri associada a um processo pode ser representada por níveis. Transições ou lugares podem ser expandidos em sub-redes de Petri. Neste trabalho, é indiferente expandir lugares ou transições, sendo que optamos por permitir que apenas lugares pudessem ser detalhados em novas sub-redes de Petri.

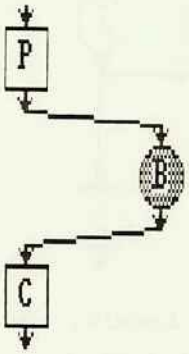
Uma sub-rede de Petri representando o detalhamento de um lugar L deve possuir todas as transições que entram e que saem de L. Todas as transições que chegam ao lugar L transformam-se em transições-fonte (transições sem lugares de entrada) e todas as transições que emanam do lugar L, transformam-se em transições-poço (transições sem lugares de saída) na sub-rede de Petri associada ao lugar L.

O nome que a sub-rede de Petri recebe é o mesmo nome do lugar que está sendo detalhado. Todos os lugares que possuem sub-redes de Petri associadas são representados através de elipses com bordas duplas.

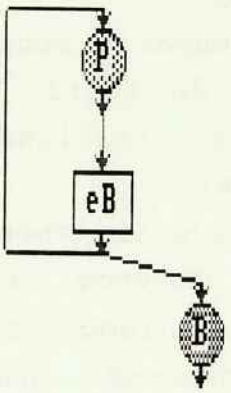
Exemplo A - O Produtor-Consumidor

Dois processos P e C compartilham o canal B, sendo que P coloca mensagens no canal e C retira mensagens do canal.

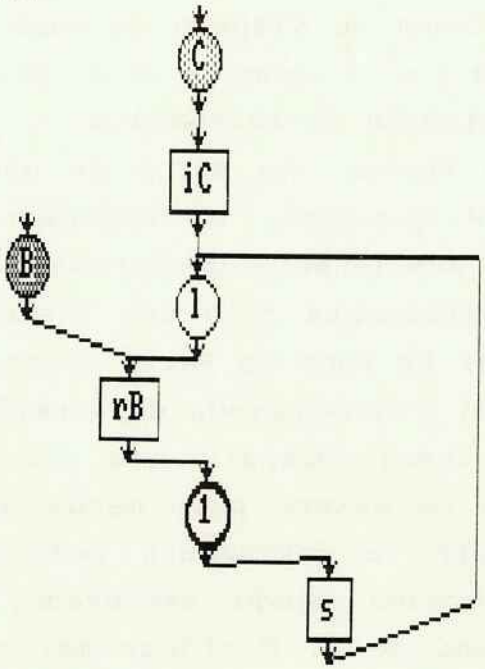
Rede de processos:



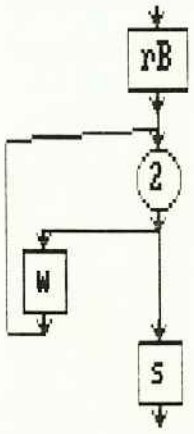
rede de Petri P:



rede de Petri C:



A sub-rede de Petri 1 detalhando o lugar 1 é:



A partir destas quatro redes, é fácil obter por simples composição, a rede de Petri global do programa concorrente.

2.2 - Simulação do Protótipo

A ferramenta permite simular o comportamento dos processos modelados através de redes de Petri. Duas operações básicas estão disponíveis para esta finalidade: inserir marcas nos lugares e disparar transições.

Na ferramenta, uma marca é implementada como um objeto que navega na rede de Petri. Objetos, e portanto marcas, são entidades que podem ser diferenciadas umas das outras.

Dada uma MEE de um processo qualquer, o usuário pode associá-la a uma classe do ambiente Smalltalk, contendo os métodos a serem executados no disparo de suas transições. Esta classe deve ser criada com o mesmo nome do processo e como uma subclasse de Token, definida na ferramenta.

Quando o usuário insere uma marca em algum lugar de uma MEE de um processo P qualquer, a ferramenta verifica se a classe Token possui a subclasse P declarada pelo usuário. Se a classe P existir, a ferramenta cria uma instância de P, que é então inserida no lugar da rede de Petri apontado pelo usuário. Caso contrário, a marca é considerada uma instância de Token.

Quando o usuário deseja disparar uma transição t qualquer, a ferramenta verifica se existe pelo menos uma marca em cada entrada de t. A seguir, a ferramenta pede ao objeto marca, correspondente ao processo sendo executado, que execute um método com o mesmo nome de t. O objeto marca verifica se ele

mesmo responde a mensagens do tipo t. Se responder, o método t passa a ser executado.

A declaração de um método t pelo usuário é composta de duas partes.

A primeira parte, que é opcional, consiste em verificar se o estado interno do objeto marca (processo) é tal que permita o disparo da transição t. Caso a transição t não seja disparável, o método t deve retornar o objeto "false" como resposta. Normalmente, esta primeira parte é declarada se a transição t compartilha lugares de entrada com outras transições, ou seja, se t estiver em conflito estrutural com outras transições.

Uma vez que a transição seja realmente disparável, a segunda parte consiste em descrever as ações que serão executadas no disparo de t. O método t deve então retornar qualquer valor diferente de "false".

Exemplo B

Na sub-rede de Petri 1 associada ao lugar 1 da rede de Petri C, no exemplo A, deseja-se sempre disparar a transição w, três vezes antes da transição s. Observar que as transições w e s estão em conflito estrutural.

Para tanto, o usuário deve associar a seguinte classe C ao ambiente Smalltalk:

Classe C

1. Cabeçalho da Classe:

```
Token subclass: #C
  instanceVariableNames:
    'contador '       " conta o número de disparos de w"
  classVariableNames: ''
  poolDictionaries: '' !
```

2. Métodos da Instância

iC

```
"inicializa contador"
 contador := 1.! "inicializando contador "
```

s

```
"dispara s após 3 disparos de w"
"primeiro, verifica se pode disparar"
 contador > 3
     ifFalse:[^false].
"segundo, dispara"
 contador := 1.!
```

```
_w
    "Dispara w 3 vezes em seguida"
    "Primeiro, verifica se pode disparar"
     contador <= 3
         ifFalse:[ ^false ].
    "Segundo, executa o disparo"
     contador := contador + 1.! !
    "Atualiza o número de disparos de w em seguida "
```

2.2.1 - Troca de Mensagens

Uma mensagem (marca colocada em um canal) corresponde, no ambiente Smalltalk, a uma instância da classe Msg. Toda instância da classe Msg possui três variáveis:

1. origem, que especifica o processo origem
2. destino, que especifica o processo destino
3. informações, que é o objeto contendo a mensagem propriamente dita.

Toda instância de mensagem pode conter os seguintes tipos de mensagens:

origem: umaIdentidade
    especifica o processo-origem da mensagem.
origem
    retorna a identificação do processo-origem da mensagem.
destino: umaIdentidade
    especifica o processo-destino da mensagem.
destino
    retorna a identificação do processo-destino da mensagem.
info: umObjeto
    especifica as informações da mensagem
info
    retorna as informações da mensagem.

Quando um canal é compartilhado por vários processos que dele podem retirar mensagens, há, em alguns casos, a necessidade de se especificar para qual dos processos a mensagem está endereçada. Neste caso, é necessário um tratamento adicional no disparo de transições que representam a recepção de mensagens.

Desta forma, antes que seja disparada uma transição que possua como entradas um lugar e um canal determinados é necessário verificar se a marca que está no canal (mensagem) possui como destino a marca que está no lugar (processo). Se

uma instância de Msg possuir a variável "destino" igual ao objeto "nil" então qualquer processo poderá retirá-la do canal.

Devido ao fato de que lugares e canais podem possuir mais de uma marca, quando uma transição que representa a recepção de mensagens é disparada, é necessário verificar se existe algum par (processo, mensagem) tal que o destino da mensagem seja aquele processo particular. Esta verificação é feita automaticamente pela ferramenta.

Quando o usuário especifica um método associado a uma transição que recebe mensagens, tal método deverá possuir um parâmetro que corresponde à mensagem que está sendo recebida.

Quando o usuário especifica um método associado a uma transição que envia mensagens, tal método deverá responder com um objeto, que é a mensagem que está sendo enviada. A ferramenta encarrega-se de inserir tal mensagem no canal.

Exemplo C

O objetivo deste exemplo é apresentar uma maneira como o usuário pode associar uma identidade a um processo e como podem ser declarados métodos que são ativados no envio e recepção de mensagens.

Deseja-se modelar um processo produtor enviando mensagens a diferentes processos consumidores. A rede de Petri é a mesma do exemplo A. Este exemplo engloba o exemplo B.

a. Objetivo:

Ao simular, o usuário insere uma marca no lugar inicial da rede de Petri P. Depois ele pode disparar as transições "eB" preenchendo o campo-destino das marcas (mensagens) a serem inseridas no canal "B".

O usuário pode inserir várias marcas no lugar inicial do processo C. Ao disparar "iC", a identidade do processo é estabelecida pelo usuário. O usuário pode então disparar as transições "rB". A ferramenta verifica se é possível o disparo de "rB", analisando os campos-destino das marcas (mensagens) no canal B e as identidades das marcas no lugar "l". O disparo das outras transições já foi comentado no exemplo B.

b. Descrição das classes

classe P

1. Cabeçalho da classe

```
Token subclass: #P
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !
```

2. Métodos da instância

eB

```
"Retorna uma mensagem com destino especificado pelo
    usuário"
^(Msg new) destino:(Prompter prompt:'tipo'
                          default:'1')! !
 "Ver observação abaixo sobre entrada de dados"
```

Classe C

1. Cabeçalho da classe C

```
Token subclass: #C
  instanceVariableNames:
    'contador '
  classVariableNames: ''
  poolDictionaries: '' !
```

2. Métodos da instância C

iC

```
"inicializa a identidade do processo e contador"
identidade := Prompter prompt:'processo' default:'1'
" identidade é a variável herdada da classe Token "
" Ver observação abaixo sobre entrada de dados"
contador := 1.!
```

rB:mensagem

```
"Imprime a mensagem recebida "
Menu message: mensagem destino.!
```

s

```
"dispara s se já disparou 3 vezes w "
"primeiro, verifica se pode disparar"
contador > 3
    ifFalse:[^false].
"segundo, dispara"
contador := 1.!
```

w

```
"Dispara w três vezes em seguida antes de s"
"Primeiro, verifica se pode disparar"
contador <= 3  "pode disparar ?"
    ifFalse:[ ^false ]. "Não. retorna false"
"Segundo, executa o disparo"
contador := contador + 1.! !
```

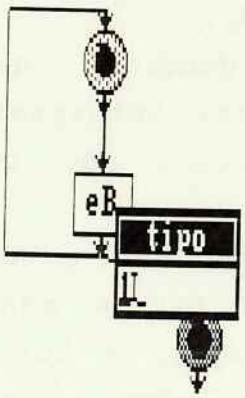
Observação:

Existem diversas formas de interação entre o usuário e o ambiente Smalltalk. Uma maneira de pedir dados ao usuário é através da classe Prompter. O comando:

```
Prompter prompt:'tipo' default:'1'
```

no método eB da classe P, ao ser executado, abre uma janela com o título "tipo", apresentando como entrada "default" uma cadeia ("string") simbolizando o número 1.

Desta forma, quando o usuário dispara a transição eB o "prompt" pedindo dados assume a seguinte forma:



2.2.2 - Semáforos

A associação de código Smalltalk para a simulação da sincronização através de semáforos é mais simples do que para a troca de mensagens, porque dentro de um canal que simula um semáforo a única informação útil é o número de marcas.

Toda marca dentro de um semáforo corresponde a uma instância de Msg com destino igual ao objeto "nil".

Quando um canal simula um semáforo, geralmente ele é inicializado pelo usuário. Para isto, o usuário apenas deve inserir neste canal um número de marcas correspondente à inicialização do semáforo.

3 - Observações finais

Foram mostrados neste artigo exemplos de simulação associando ou não código Smalltalk às transições. Quando uma rede de Petri exibe código executável associado às transições ou lugares, ela é denominada "rede de Petri interpretada".

Uma rede de Petri descreve apenas a estrutura de controle de um programa, mas, associando código Smalltalk às transições, a rede de Petri interpretada passa a especificar os dados, as operações sobre os mesmos e a ordem em que elas ocorrem.

Observa-se que o conjunto das marcações acessíveis na rede de Petri interpretada pode ser menor pois em certas ocasiões, transições disparáveis na rede de Petri original podem deixar de ser disparáveis na rede de Petri interpretada. Deste fato pode-se tirar algumas conclusões a respeito da análise da rede de Petri interpretada, a partir da análise da rede de Petri não-interpretada [Valette-90].

Quanto à análise de "deadlock" nada se pode afirmar. É possível que a rede de Petri interpretada possua "deadlocks" enquanto que a rede de Petri não interpretada não possua "deadlocks" e vice-versa.

Nesta versão da ferramenta, a inserção de marcas e o disparo de transições são feitos manualmente pelo usuário. Entretanto, é possível associar à ferramenta um programa que faça o papel do usuário. Este programa é conhecido na literatura como "jogador de marcas". Marcas são colocadas e transições disparadas de acordo com uma certa distribuição estatística pré-estabelecida. Desta forma é possível estimar, por exemplo, onde estará o gargalo do programa concorrente a ser implementado.

A ferramenta apresentada em [Dähler-87] possui características semelhantes. Nesta ferramenta, a estrutura de controle de um programa concorrente é definida através de redes de Petri e a estrutura de dados, bem como a operação sobre os mesmos, é definida em Smalltalk. Entretanto, ela lida com redes de Petri predicado-transição e a maneira de associar código Smalltalk às transições é feita de forma diferente.

A ferramenta apresentada neste artigo é ainda um protótipo, que precisa ser estimulado em situações reais para se constatar a sua viabilidade.

Bibliografia

[Bressan-85] Bressan,G. , "Ambiente de Programação distribuída: Definição, Análise, Síntese", tese de doutorado, Escola Politécnica, Universidade de São Paulo, 1985.

[Dähler-87] Dähler, J. et al., "A Graphical Tool for the Design and Prototyping of Distributed Systems". ACM SIGSOFT Software Engineering Notes, vol. 12, no. 3, pgs 25-36, July 1987.

[Gold-83] Goldberg, A. and Robson, D., "Smalltalk 80: The Language and its Implementation", Addison-Wesley, Reading, Mass., 1983.

[Kino-91] Kinoshita, J., "Uma ferramenta para a construção de programas concorrentes". Dissertação de mestrado a ser apresentada à Escola Politécnica da Universidade de São Paulo em 1991.

[Murata-89] Murata, T., "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, vol. 77, no. 4, pgs 541-580, April 1989.

[Valette-90] Vallete, R., notas do curso "Redes de Petri" apresentado na Escola Politécnica da Universisdade de São Paulo em Julho de 1990.

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PCS/9301 - Interligação de Processadores através de Chaves Ômicron - GERALDO LINO DE CAMPOS, DEMI GETSCHKO

BT/PCS/9302 - Implementação de Transparência em Sistema Distribuído - LUÍSA YUMIKO AKAO, JOÃO JOSÉ NETO

BT/PCS/9303 - Desenvolvimento de Sistemas Especificados em SDL - SIDNEI H. TANO, SELMA S. S. MELNIKOFF

BT/PCS/9304 - Um Modelo Formal para Sistemas Digitais à Nível de Transferência de Registradores - JOSÉ EDUARDO MOREIRA, WILSON VICENTE RUGGIERO