UMA VISÃO MODERNA PARA O TRATAMENTO DE CONCORRÊNCIA EM MODULA-2

Regina Claudia de Alencar Ximenes (*)
Paulo Roberto Freire Cunha (**)

RESUMO

Este trabalho apresenta uma proposta de extensão para a linguagem Modula-2, através de um núcleo multiprogramado que implementa o modelo de processos baseado em passagem de mensagens, tendo como destaque, o uso de portas como mecanismo indireto de designar os processos envolvidos numa transação de mensagens. O projeto do núcleo foi baseado na linguagem de programação Conic, e busca refletir as principais contribuições desta linguagem: reusabilidade de processos e independência de configuração.

ABSTRACT

This work presents an extension to the programming language Modula-2, through a multiprogrammed kernel which implements the model of process based on message passing. It uses ports as a indirect mecanism to call the processes related in a message transactions. The kernel was based on the programming language Conic, and reflects the principals contribuitions of this language: reusability of processes and configuration independency.

(*) Mestra em Informática (UFPE-90). Redes de Computadores , Sistemas Distribuídos, Engenharia de Software.

(**) Ph.D. em Computação (Waterloo-81), Prof. Adjunto (UFPE). Redes de Computadores, Sistemas Distribuídos, Metodologias de Progração.

1. INTRODUÇÃO

O surgimento dos Sistemas Distribuídos fomentou o mundo das linguagens de programação concorrente provocando um aprimoramento constante das mesmas, à medida em que se buscam formas alternativas de adequá-las mais eficientemente ao ambiente ao qual se propõem. Todavia, ainda são poucas as linguagens de programação concorrente disponíveis no mercado, fator que se agrava quando restringimos o nosso universo àquelas voltadas para ambientes uniprocessadores e que disponham de compiladores utilizáveis em mini ou microcomputadores. Além disto, as poucas linguagens existentes tratam concorrência basedas num modelo muito primitivo e de baixo-nível de abstração, o qual adota corotinas, como unidades básicas para estruturação de programas pseudo-paralelos, e transferências entre corotinas, como mecanismo para prover sincronização. Modula-2 é um exemplo típico de tais linguagens.

Modula-2 [Wirth 83,Gough 88,King 88] é uma linguagem moderna, voltada para ambientes uniprocessadores, e que engloba um número considerável de características desejáveis a uma linguagem de programação. Graças ao seu caráter flexível, não impõe um modelo fixo para o tratamento de concorrência de modo que modelos alternativos ao de corotinas, os quais adotam processos como unidades básicas para a construção de software concorrente, podem ser implementados utilizando como infraestrutura os recursos primitivos da linguagem. Assim, aproveitando esta facilidade de Modula-2, e motivados pelo desejo de dispor de uma linguagem orientada a processos e mensagens, dotada de recursos modernos que reflitam, pelo menos parcialmente, aqueles providos em linguagens concorrentes/distribuídas recentemente surgidas, apresentamos neste trabalho um núcleo multiprogramado, chamado TrocaMensagens, para implementação do modelo de processos, como proposta de extensão para a linguagem Modula-2 . 260

Além de ser baseado na filosofia de processos e mensagens, o núcleo adota o conceito de portas como forma de designação indireta dos processos envolvidos numa transação de mensagens. Em toda sua concepção, o núcleo foi baseado nos conceitos e principais características da linguagem de programação distribuída Conic [Dulay 84, Kramer 84], e busca, embora que de uma forma restrita, refletir as principais contribuições daquela linguagem: reusabilidade de processos e independência de configuração. Queremos frisar que o nosso esforço

esteve concentrado em tentar aproveitar ao máximo os recursos básicos de Modula-2, de modo que as extensões propostas fossem fundamentadas nos mesmos. Desta forma, os construtores adicionais não afetarão o perfil original da linguagem, mas sim, enriquecerão os recursos oferecidos.

2. O NÚCLEO TrocaMensagens

O núcleo TrocaMensagens [Ximene 90] é uma ferramenta alternativa que incorpora a Modula-2 um novo estilo de programação concorrente, orientada a processos e mensagens [Cunha 81]. Nele, processos são vistos como uma abstração de corotinas, e operações (primitivas) de escalonamento como abstração de transferências entre corotinas. O conceito de portas introduzido reflete totalmente as idéias de Conic no que diz respeito a diferenciação entre portas de saída e de entrada, notificadas ("notify") ou pedido-resposta (request-reply"). Uma porta deve ser declarada internamente a um processo, de modo que apenas este pode referenciá-la. Portas de entrada notificada podem conter mensagens de uma mesma porta de saída, bem como mensagens de portas de saída diferentes. Portas de entrada pedido-resposta só podem conter mensagens de portas de saída diferentes, sendo que um máximo de uma mensagem por porta de saída conectada.

Apesar de não termos linguagens distintas para a programação dos módulos individuais e configuração do sistema, procuramos aproveitar estas idéias de Conic, mesmo que de uma maneira restrita. Para isto, propomos que a fase de conexão das portas dos processos seja feita num processo adicional separado, ao qual damos o nome de Processo de Configuração, e que deve ser criado em todo programa especialmente para tal finalidade. Desta forma, o fato de todas as referências num processo poderem ser feitas apenas a objetos locais, associado ao fato de nenhuma primitiva de configuração ser embutida no código do processo, permite a sua reusabilidade em outros programas, e como consequência, provê independência de configuração. Evidentemente, o único processo que não poderá ser reutilizado em outros programas é o processo de configuração, uma vez que seu código é específico de uma implementação particular. A seguir apresentamos o formato da estrutura básica de um programa Modula-2 (na sua forma mais simples) que tem como suporte para concepção e interação entre processos o núcleo

TrocaMensagens.

```
MODULE Exemplo;
(* Lista de Imports *)
(* Declaração de Constantes e Tipos *)
PROCEDURE Proc1;
   . . .
END Proc1;
      . . .
PROCEDURE ProcN;
   . . .
END ProcN;
PROCEDURE Configuracao;
    . . .
 END Configuracao;
 BEGIN
    (* Criação dos processos Proc1 a ProcN *)
    (* Criação do processo Configuração *)
    (* Inicialização da execução dos processos *)
 END Exemplo.
```

Note que na lista de Imports deverá estar explicitada a importação do núcleo TrocaMensagens. O formato básico da estrutura de um processo para o programa acima está mostrado a seguir:

```
PROCEDURE ProcN;
  (* Declaração de Constantes, Tipos e Variáveis *)
  (* Declaração de Portas - 1a. etapa *)
BEGIN
  (* Declaração de Portas - 2a. etapa *)
  (* Execução de um Receive Notificado *)
  (* Sequência de Comandos *)
END ProcN;
```

As duas etapas da declaração de portas serão discutidas posteriormente. Note que após a conclusão da fase de declaração de portas, e antes da execução da sequência de comandos que implementa as ações do processo, uma primitiva de recebimento notificada deve ser executada. Sua finalidade é receber um sinal do processo de Configuração, indicando que a fase de conexão das portas já encerrou, e que por isto o processo já pode interagir com outros processos ao mesmo conectados.

As três funções básicas do núcleo TrocaMensagens são as seguintes: criação e gerenciamento da Tabela de Processos, comunicação local, e escalonamento de processos. Os serviços (primitivas e funções auxiliares) do núcleo TrocaMensagens são implementados como

procedimentos/funções (PROCEDUREs) comuns, de tal maneira que a forma de invocação é a usualmente conhecida. A atomicidade das operações é garantida uma vez que as mesmas são implementadas num módulo biblioteca Modula-2, priorizado, de modo que nenhuma interrupção será reconhecida quando qualquer um dos serviços estiver sendo executado. O pseudo-paralelismo dos processos é controlado pelo núcleo TrocaMensagens através da corotina "Scheduler", servidora de interrupções do "clock", de modo que a cada processo será alocada uma pequena parcela do tempo do processador. A seguir apresentaremos cada um dos serviços oferecidos pelo núcleo TrocaMensagens.

2.1 PRIMITIVAS DO NÚCLEO

a) CRIAÇÃO DE PROCESSOS

- NOME DA PRIMITIVA: **CreateProcess**
- FINALIDADE: Criar um processo.
- FORMA DE INVOCAÇÃO:

```
CreateProcess(Nome-do-Processo,Código-do-Processo,Tamanho-Pilha);
```

- COMENTÁRIOS: Esta primitiva cria um processo e insere-o na Tabela de Processos e na Fila de Processos Prontos. O parâmetro Nome-do-Processo deve ser uma cadeia de caracteres de tamanho máximo 20; o parâmetro Código-do-Processo corresponde ao nome da PROCEDURE sem parâmetros que constituirá o corpo da corotina embutida, utilizada como código do processo. O parâmetro Tamanho-Pilha deve ser um CARDINAL que denota o tamanho do espaço de memória (em "bytes") a ser usado como área de trabalho ("Workspace") da corotina associada. Vale ressaltar que esta primitiva apenas cria um processo, mas não o ativa. Réplicas de um processo podem ser criadas invocando a primitiva CreateProcess com os mesmos parâmetros, excetuando-se o primeiro que deve ser diferente para cada processo criado.

b) ATIVAÇÃO DE PROCESSOS

- NOME DA PRIMITIVA: **StartProcess**
- FINALIDADE: Iniciar a execução dos processos.
- FORMA DE INVOCAÇÃO: StartProcess();
- COMENTÁRIOS: Esta primitiva simplesmente ativa o primeiro processo da Fila de Processos Prontos. A ativação dos demais processos fica a cargo do núcleo e obedece a ordem em que os mesmos

aparecem na fila de processos prontos. A primitiva StartProcess só deve aparecer uma única vez no corpo do programa, logo em seguida à criação de todos os processos.

c) DESTRUIÇÃO DE PROCESSOS

- NOME DA PRIMITIVA: **EndProcess**
- FINALIDADE: Encerrar a execução de um processo.
- FORMA DE INVOCAÇÃO: EndProcess();
- COMENTÁRIOS: A primitiva EndProcess encerra a execução do processo que a invocou, retirando-o da Tabela de Processos e da Fila de Prontos. Também efetua a desconexão de suas portas com as portas de outros processos ao mesmo conectados (se existirem). Testa ainda o fim de execução do programa, encerrando-o caso não existam mais processos na Tabela de Processos, ou detecta um "deadlock" (impasse) e também encerra o programa, caso não existam mais processos na Fila de Prontos para serem ativados e a Tabela de Processos não esteja vazia, porque existem processos bloqueados.

d) DECLARAÇÃO DE PORTAS

A declaração de portas é composta de duas etapas. A primeira corresponde à declaração de uma variável local que representará uma porta no processo. Dois tipos de dados, definidos e exportados pelo núcleo TrocaMensagens, podem ser usados nesta etapa da declaração de portas, EXITPORT e ENTRYPORT, os quais especificam se a porta é de saída ou de entrada, respectivamente. O nome de uma variável do tipo EXITPORT ou ENTRYPORT é conhecido como o Nome Interno da porta. Esta primeira etapa é importante para que o compilador reconheça uma variável como uma porta, e aceite o seu uso como parâmetro nas primitivas de comunicação. A segunda etapa da declaração de portas é a etapa conclusiva; é através desta que o núcleo TrocaMensagens terá conhecimento de uma porta e das informações gerais sobre a mesma, tais como: seu Nome Externo, útil para a fase de conexão de portas, e tipo, Notificado ou Pedido-Resposta. O Nome Externo de uma porta deve ser uma cadeia de caracteres de tamanho máximo 10 e sua definição deve obedecer a certas regras, conforme definido em [Ximene 90]. Um aspecto importante de Conic, não aproveitado no nosso núcleo, é a especificação do tipo do valor que poderá ser enviado ou recebido através de uma porta, uma vez que não temos como checar este tipo com o tipo da

variável que será enviada como parâmetro de informação nas primitivas de envio ou recebimento de mensagens.

Duas primitivas distintas são providas para a segunda etapa da declaração de uma porta: **DeclareEP**, utilizada se a porta é do tipo ENTRYPORT, e **DeclareXP**, utilizada se a porta é do tipo EXITPORT. A seguir apresentaremos cada uma delas:

PRIMITIVA PARA DECLARAÇÃO DE UM PORTA DE SAÍDA:

- NOME DA PRIMITIVA: **DeclareXP**
- FINALIDADE:

  Realizar a segunda etapa da declaração de uma porta de saída.
- FORMA DE INVOCAÇÃO:

  DeclareXP(Nome-Interno,Nome-da-Porta,Tipo-da-Porta);
- COMENTÁRIOS: O parâmetro Nome-Interno corresponde ao nome interno da porta de saída (EXITPORT); o parâmetro Nome-da-Porta corresponde ao seu nome externo. O parâmetro Tipo-da-Porta especifica se a porta é do tipo Notificada (Not) ou Pedido-Resposta (ReqRep).

PRIMITIVA PARA DECLARAÇÃO DE UM PORTA DE ENTRADA:

- NOME DA PRIMITIVA: **DeclareEP**
- FINALIDADE:

  Realizar a segunda etapa da declaração de uma porta de entrada.
- FORMA DE INVOCAÇÃO:

  DeclareEP(Nome-Interno,Nome-da-Porta,Tipo-da-Porta);
- COMENTÁRIOS: Os comentários feitos para a declaração de uma porta de saída são válidos aqui. A única diferença é que a porta referenciada como primeiro parâmetro deve ser uma porta de entrada.

e) **CONEXÃO DE PORTAS**

- NOME DA PRIMITIVA: **Link**
- FINALIDADE: Conectar uma porta de saída notificada (pedido-resposta) a uma porta de entrada notificada (pedido-resposta).
- FORMA DE INVOCAÇÃO:

  Link(Nome-Proc-XP,Nome-Porta-XP,Nome-Proc-EP,Nome-Porta-EP);
- COMENTÁRIOS: Os dois primeiros parâmetros correspondem a informações relativas à porta de saída, no que diz respeito ao nome do processo que a declarou e ao nome externo da mesma. Os dois últimos são relativos à porta de entrada, e similarmente denotam o nome do

processo, onde ocorreu a sua declaração, e o nome externo da mesma. Os padrões de conexões permitidos são: 1-1, 1-n e m-1, excetuando-se o padrão 1-n que só é válido para portas notificadas.

f) ENVIO NOTIFICADO

- NOME DA PRIMITIVA: **Send**
- FINALIDADE: Passar uma informação de um processo transmissor para um ou mais processos receptores.
- FORMA DE INVOCAÇÃO: Send(Nome-Interno-XP,Informação);
- COMENTÁRIOS: O parâmetro Nome-Interno-XP corresponde ao nome interno da porta de saída notificada, através da qual fluirá a informação. O parâmetro Informação é o valor que será passado como mensagem. O transmissor continua sua execução tão logo a informação seja colocada na fila de mensagens do processo destino.

g) RECEBIMENTO NOTIFICADO

- NOME DA PRIMITIVA: **Receive**
- FINALIDADE: Receber uma mensagem.
- FORMA DE INVOCAÇÃO: Receive(Nome-Interno-EP,Var-Informação);
- COMENTÁRIOS: O parâmetro Nome-Interno-EP corresponde ao nome interno da porta de entrada notificada através da qual a mensagem será recebida. A informação é copiada no parâmetro Var-Informação e a mensagem é removida da fila de mensagens da porta de entrada. Não existindo mensagem a ser recebida, o processo receptor é bloqueado até que uma mensagem chegue. O recebimento notificado assincrono pode ser simulado combinando esta primitiva com a função *IsThereMsg*, que testa se existem mensagens para serem recebidas numa porta de entrada, da seguinte forma:

```
        . . .
IF IsThereMsg(Nome-Interno-EP) THEN
   Receive(Nome-Interno-EP,V)
END;
        . . .
```

h) ENVIO PEDIDO-RESPOSTA

O envio Pedido-Resposta pode ser feito de duas formas. Em particular, a segunda forma deve ser usada quando se deseja que a fonte temporize o término da transação, a fim de não ficar bloqueada

indefinidamente a espera de uma resposta que pode nunca chegar.

PRIMEIRA FORMA:

- NOME DA PRIMITIVA: **SendWait**
- FINALIDADE: Enviar uma mensagem de pedido e esperar a resposta.
- FORMA DE INVOCAÇÃO: SendWait(Nome-Interno-XP,Informação,Var-Resp);
- COMENTÁRIOS: O primeiro parâmetro corresponde ao nome interno da porta de saída Pedido-Resposta. O parâmetro Informação corresponde à mensagem de pedido, e o parâmetro Var-Resp é uma variável na qual deverá retornar a resposta. O processo transmissor é bloqueado até que a resposta seja copiada na variável Var-Resp.

SEGUNDA FORMA:

- NOME DA PRIMITIVA: **SendWaitFail**
- FINALIDADE: Enviar uma mensagem de pedido e esperar a chegada da resposta correspondente, dentro de um período de tempo determinado.
- FORMA DE INVOCAÇÃO:
  SendWaitFail(Nome-Interno-XP,Informação,Var-Resp,Tempo,Var-Status);
- COMENTÁRIOS: Os comentários para os três primeiros parâmetros são equivalentes aos feitos para a primitiva SendWait. O parâmetro Tempo corresponde ao tempo máximo no qual o transmissor estará bloqueado esperando uma resposta. O parâmetro Var-Status é uma variável na qual retornará o resultado ("Status") da primitiva. Os possíveis resultados e seus significados estão expressos na tabela 2.1.

STATUS	SIGNIFICADO
OK	Transação terminada com sucesso. O parâmetro Var-Resp contém a resposta do pedido.
UnLinked	A porta de saída não está conectada a uma porta de entrada; nenhuma mensagem retornará. O parâmetro Var-Resp é indefinido e não deve ser acessado. Quando este resultado é detectado a operação termina.
Incompatibility	O tipo da porta de saída utilizada como primeiro parâmetro é incompatível com a primitiva de envio Pedido-Resposta, ou a porta de saída especificada como parâmetro não foi completamente declarada. Quando este resultado é detectado a operação termina.
Timeout	O período de tempo especificado expirou antes que a mensagem de resposta tenha chegado.

TABELA 2.1 - POSSIVEIS STATUS RETORNAVEIS DA PRIMITIVA SendWaitFail

i) **RECEBIMENTO PEDIDO-RESPOSTA**

O recebimento de um pedido e o envio da resposta correspondente podem ser realizados através de duas formas distintas. A primeira forma diz respeito às primitivas ReceiveRequest e Reply; a segunda, constituída por uma única primitiva chamada ReceiveReply, é uma forma compacta na qual o envio da resposta já está implícito na primtiva de recebimento. Mais de uma mensagem de pedido pode ser aceita numa mesma porta de entrada pedido-resposta, sendo que cada **Reply** executado naquela porta de entrada enviará a resposta para o último pedido pendente, sempre obedecendo a ordem LILO ("Last-In-Last-Out").

RECEBIMENTO DO PEDIDO

- NOME DA PRIMITIVA: **ReceiveRequest**
- FINALIDADE: Receber uma mensagem de solicitação de serviço.
- FORMA DE INVOCAÇÃO: ReceiveRequest(Nome-Interno-EP,Var-Informação);
- COMENTÁRIOS: O parâmetro Nome-Interno-EP corresponde ao nome interno da porta de entrada Pedido-Resposta. A informação de pedido será copiada no parâmetro Var-Informação, e a mensagem removida da fila de mensagens da porta de entrada. Caso não existam mensagens disponíveis, o processo receptor é bloqueado até que uma mensagem chegue. Equivalentemente ao caso notificado, o recebimento Pedido-Resposta assíncrono pode ser simulado combinando a função *IsThereMsg* com a primitiva de recebimento Pedido-Resposta.

ENVIO DA RESPOSTA

- NOME DA PRIMITIVA: **Reply**
- FINALIDADE: Enviar a resposta para um pedido recebido.
- FORMA DE INVOCAÇÃO: Reply(Nome-Interno-EP,resp);
- COMENTÁRIOS:

O parâmetro Nome-Interno-EP corresponde ao nome interno da porta de entrada Pedido-Resposta com pedido pendente. A execução de um Reply numa porta de entrada, não precedido pela execução de um ReceiveRequest, não tem efeito. A primitiva Reply não bloqueia a tarefa que a invocou.

RECEBIMENTO PEDIDO-RESPOSTA COM REPLY IMPLÍCITO

- NOME DA PRIMITIVA: **ReceiveReply**

• FINALIDADE: Receber uma mensagem de solicitação de serviço e enviar a resposta imediatamente, sem que seja necessário o uso da primitiva Reply.

• FORMA DE INVOCAÇÃO:

ReceiveReply(Nome-Interno-EP,Var-Informação,resp);

• COMENTÁRIOS: Todos os comentários feitos para a primitiva ReceiveRequest, sem exceção, são válidos aqui. No término desta primitiva a mensagem de resposta já terá sido enviada.

j) RECEBIMENTO SELETIVO

• NOME DA PRIMITIVA: Select

• FINALIDADE: Seleciona, arbitrariamente, uma entre as várias portas de entrada (Pedido-Resposta ou Notificada) com mensagens disponíveis de um processo. Caso não existam mensagens disponíveis, temporiza a espera do receptor, desbloqueando o processo após ter expirado o período de tempo especificado.

• FORMA DE INVOCAÇÃO: Select(Var-Ref-Nome-EP,Tempo,Var-Status);

• COMENTÁRIOS: O parâmetro Var-Ref-Nome-EP é uma variável que retorna um valor referente ao endereço da porta de entrada escolhida, onde uma primitiva de recebimento deverá ser invocada. posteriormente. O parâmetro Tempo deve conter a especificação do período de tempo máximo que o processo destino permanecerá bloqueado a espera de uma mensagem. O parâmetro Var-Status é uma variável na qual retornará o resultado ("Status") da primitiva. Os possíveis resultados retornáveis e os seus significados estão expressos na tabela abaixo:

STATUS	SIGNIFICADO
TimeOut	Nenhuma porta foi selecionada pois não existem mensagens disponíveis. Nenhuma primitiva de recebimento deverá ser invocada.
EPN	Uma porta de entrada notificada foi selecionada. A primitiva de recebimento notificado pode ser invocada.
EPR	Uma porta de entrada pedido-resposta foi selecionada. A primitiva de recebimento pedido-resposta pode ser invocada.

TABELA 2.2 - POSSÍVEIS STATUS RETORNÁVEIS DA PRIMITIVA Select

l) TRATAMENTO DE INTERRUPÇÕES

• NOME DA PRIMITIVA: WaitIO

- FINALIDADE: Retardar a execução do processo até que a interrupção especificada ocorra.
- FORMA DE INVOCAÇÃO: WaitIO(Vet-Int);
- COMENTÁRIOS: O parâmetro Vet-Int corresponde ao endereço do Vetor de Interrupção do dispositivo do qual se espera interrupções. A chamada de WaitIO suspende o processo, e o controle do processador é passado para o próximo processo da Fila de Prontos. Quando a interrupção especificada ocorre, o processo é então novamente inserido na Fila de Prontos, na frente do processo interrompido, e sua execução é imediatamente reativada.

2.2 FUNÇÕES AUXILIARES

a) IsThereMsg(Nome-interno-porta-de-entrada)

Esta função booleana testa se existe alguma mensagem numa porta de entrada (Notificada ou Pedido-Resposta).

b) Linked(Nome-interno-porta-de-saída)

Esta função booleana testa se uma porta de saída (notificada ou pedido-resposta) está conectada a alguma porta de entrada.

c) QueueSize(Nome-interno-porta-de-entrada)

Retorna o tamanho da fila de mensagens de uma porta de entrada, ou seja, a quantidade de mensagens a serem recebidas.

d) LinkPermited()

Verifica se a fase de conexão das portas dos processos do sistema já pode ser iniciada, retornando o valor TRUE caso possa, e FALSE caso contrário.

3. O PROCESSO DE CONFIGURAÇÃO

O processo de configuração, além de ser o ambiente onde ocorrem as devidas conexões entre as portas dos processos, é utilizado como uma ferramenta de sincronização da execução dos mesmos, uma vez que se encarrega de sinalizá-los que todas as conexões já foram efetuadas, e que por isto os mesmos já podem continuar suas execuções e trocar informações. O código para um processo de configurção tem o seguinte formato:

```
PROCEDURE Configuracao;
VAR Config: EXITPORT; (* Porta de Saída notificada conectada a uma
                         porta de entrada notificada declarada nos
BEGIN                    demais processos para receber a sinalização *)
   DeclareXPC(Config,"Config",Not);
   LOOP
      IF LinkPermited() THEN
         (* Efetua conexões entre portas dos processos *)
         Send(Config,Sinal);        (* Sinaliza os processos que as
         EXIT                          conexões já foram efetuadas  *)
      END
   END
   EndProcess()       (* Encerra sua execução *)
END Configuracao;
```

4. Exemplo Prático - Buffer Limitado

A fim de demonstrar a clareza, eficiência e facilidade de uso do núcleo TrocaMensagens para a construção modular, orientada a mensagens, de programas pseudo-paralelos em Modula-2, apresentaremos a seguir o problema do Buffer Limitado, o qual é composto pelos processos Produtor, Consumidor e Buffer, e ainda pelo processo de Configuração, responsável pela configuração do sistema.

```
MODULE BufferLimitado;
FROM TrocaMensagens IMPORT ...
FROM IO IMPORT ...
CONST MaxSize = 132;

(* Corpo do processo Produtor *)
PROCEDURE Produtor;
VAR     p : EXITPORT;
     conf : ENTRYPORT;
     info : CHAR;
       OK : SIGNAL;
BEGIN
   DeclareXPC(p,"put",ReqRep);
   DeclareEPC(conf,"conf",Not);
   Receive(conf,OK);
   LOOP
      info := RdChar();
      SendWait(p,info,OK);
   END
END Produtor;

(* Corpo do processo Consumidor *)
PROCEDURE Consumidor;
VAR     g : EXITPORT;
     conf : ENTRYPORT;
     info : CHAR;
       OK : SIGNAL;
BEGIN
   DeclareXPC(g,"get",ReqRep);
```

```
   DeclareEP(conf,"conf",Not);
   Receive(conf,OK);
   LOOP
      SendWait(g,Sinal,info);
      WrChar(info);
   END
END Consumidor;

(* Corpo do processo Buffer *)
PROCEDURE Buffer;
VAR     p,g,conf,aux : ENTRYPORT;
    full,inp,out : CARDINAL;
      request,OK : SIGNAL;
          Status : StatusSelect;
          buffer : ARRAY[1..MaxSize] OF CHAR;
BEGIN
   DeclareEP(p,"PutChar",ReqRep);
   DeclareEP(g,"GetChar",ReqRep);
   DeclareEP(conf,"conf",Not);
   Receive(conf,OK);
   full := 0; inp := 1; out := 1;
   LOOP
      Select(aux,0,Status);
      DO CASE Status OF
         TimeOut: !    (* Nenhuma mensagem a ser recebida *)
         EPR    : IF aux = p THEN
                     IF full < MaxSize THEN
                        ReceiveReply(p,buffer[inp],Sinal);
                              ...
                     END
                  ELSE
                     IF aux = g THEN
                        IF full > 0 THEN
                           ReceiveReply(g,request,buffer[out]);
                              ...
                        END
                     END
                  END !
      END
   END
END Buffer;

(* Corpo do processo de configuracao *)
PROCEDURE Config;
VAR    conf : ENTRYPORT;
BEGIN
   DeclareXP(conf,"conf",Not);
   LOOP
      IF LinkPermited() THEN
         Link("Prod","put","Buffer","putchar");
         Link("Cons","get","Buffer","getchar");
         Link("Config","conf","Buffer","conf");
         Link("Config","conf","Prod","conf");
         Link("Config","conf","Cons","conf");
         Send(conf,Sinal);
         EXIT;
      END
   END
```

```
   EndProcess()
END Config;

(* Corpo do Programa BufferLimitado *)
BEGIN
   CreateProcess("Prod",Produtor,2000);
      ...
   CreateProcess("Config",Config,2000);
   StartProcess()
END BufferLimitado.
```

Note que a variável *aux* declarada no processo Buffer não é uma porta, desde que não foi concluída a segunda etapa da sua declaração. Tal variável serve apenas para indicar qual porta de entrada foi selecionada pela primitiva Select, a fim de que uma primitiva de recebimento seja invocada na mesma. Note também que pelo fato do Select não permitir condições de guardas, estas foram simuladas utilizando o comando IF antes das primitivas de recebimento.

5. Conclusões

O núcleo TrocaMnesagens é uma ferramenta poderosa para o desenvolvimento de programas concorrentes Modula-2, orientados a processos e mensagens, o qual torna transparente para os usuários da linguagem, o baixo nível de abstração inerente ao modelo primitivo de corotinas provido pela mesma. O principal destaque do núcleo é a utilização do conceito de portas para designar indiretamente os processos envolvidos numa transação de mensagens e suas principais contribuições é o fato de refletir, apesar de um forma restrita, as idéias de reusabilidade de processos e independência de configuração.

Outras propostas para implementação do modelo de processos em Modula-2 podem ser encontradas na literatura, tais como em [Hoppe 80] e em [Segre 84]. Porém, devido a simplicidade das mesmas, estamos convencidos de que a nossa proposta é muito mais rica, tanto no que diz respeito ao número de conceitos englobados, como na quantidade de recursos oferecidos.

Uma sugestão para futuros trabalhos seria a implementação do núcleo proposto para um ambiente multiprocessador com memória compartilhada, visto que isto não implicaria em nenhuma mudança com relação ao código que implementa as primitivas de comunicação, já que seria preservado o espaço de endereçamento global. O que fatamente necessitaria de mudanças seriam aspectos relacionados a políticas de

alocação e escalonamento de processos.

Referências

[Cunha 81] Cunha, P. R. F. - "Design and Analysis of Message Oriented Programs"; **Tese de Doutorado**, University of Waterloo, Ontario, 1981.

[Dulay 84] Dulay, N., Kramer, J., Magee, J., Sloman, M. e Twidle K. - "The CONIC Configuration Language: Version 1.3"; **Research Report Doc 84/20**, Dept. of Computing, Imperial College, November/1984.

[Gough 88] Gough, K. J., Mohay G. M. - "Modula-2: A Second Course in Programming"; **Prentice-Hall Ltd.**, Sydney, 1988.

[Hoppe 80] Hoppe, J. - "A Simple Nucleus Written in Modula-2"; **Soft. Pract. and Exp.**, Vol. 10, No. 9, (pp. 69-77), 1980.

[King 88] King, K. N. - "TopSpeed Modula-2 - Language Tutorial", **Jensen & Partners International**, 1988.

[Kramer 84] Kramer, J., Magee, J., Sloman, M., Twidle, K. e Dulay, N. - "The CONIC Programming Language : Version 2.4"; **Research Report Doc 84/19**, Dept. of Computing, Imperial College, October/1984.

[Segre 84] Segre L., Stanton M. - "Modula-2: Suporte para o Desenvolvimento de Software Concorrente"; **XVII Congresso Nacional de Informática**, Sucesu, Rio de Janeiro, Nov/84.

[Ximene 90] Ximenes, R.C.A., Cunha P.R.F. - "Proposta de um Núcleo para Implementação do Modele de Processos e Mensagens Utilizando o Conceito de Portas em Modula-2"; **Dissertação de Mesatrado**, Depto. de Informática, UFPE, 1990.

[Wirth 83] Wirth, N. - "Programming in Modula-2; 2a. ed., **Springer-Verlag**, 1983.