SBA: Controle & Automação, Vol. 2, Nº 2, pp. 93-101, (TC)

LINGUAGENS PARA ESPECIFICAÇÃO DE SISTEMAS DISTRIBUÍDOS EM AUTOMAÇÃO INDUSTRIAL

Aloysio de Castro Pinto Pedroza
Jorge Lopes de Souza Leão

COPPE/UFRJ - Eng. Elétrica
C. Postal 68504, Rio de Janeiro,RJ

Sergio Vianna Fialho

UFRN - Eng. Elétrica
Lagoa Nova, Natal, RN

RESUMO

Os métodos de descrição formal são uma ferramenta básica em todo projeto de protocolos para sistemas distribuídos. À fase de especificação formal devem-se seguir as fases de validação, a implementação automatizada e o teste final. Este artigo apresenta de uma maneira sucinta duas das técnicas de descrição formal que vêm sendo largamente usadas em projetos de protocolos: as linguagens ESTELLE e LOTOS, ambas em vias de padronização pela ISO. O uso destes dois formalismos é ilustrado através de vários exemplos.

LANGUAGES FOR DISTRIBUTED SYSTEMS SPECIFICATION IN INDUSTRIAL AUTOMATION

ABSTRACT

Formal description techniques are a basic tool for protocol development in distributed systems. A complete protocol project includes the specification, validation, implementation and final tests phases. All these should be assisted by automated tools. This paper presents two formal description techniques which have been used sucessfuly in many distributed systems projects: The ESTELLE and LOTOS languages proposed and developed by ISO. A number of examples are presented to demonstrate the use of both techniques.

1. INTRODUÇÃO

Os trabalhos recentes na área de redes de computadores têm demonstrado que o uso de uma abordagem formal na concepção dos protocolos para tais sistemas contribui para o aumento da segurança e da qualidade.

Os métodos de descrição formal são uma ferramenta importante em todo o projeto de protocolos. As especificações devem ser feitas segundo um modelo formal ou uma linguagem de descrição formal que forneçam especificações completas. As fases seguintes do projeto incluem a validação da especificação formal, a implementação automatizada e o teste final.

Este artigo apresenta sucintamente duas das técnicas de descrição formal que vêm sendo usadas largamente em projetos de protocolos: a linguagem ESTELLE e a linguagem LOTOS, ambas em vias de padronização pela ISO.

O uso destas duas técnicas será ilustrado através de um exemplo simples apresentado no parágrafo 1.1, a seguir.

1.1 - Um Exemplo: Sincronismo entre duas Máquinas

Sejam duas máquinas cooperantes com a descrição informal dada por:

i) Máquina A:
- início
- executar operação A1
- liberar B
- esperar liberação (vinda de B)
- executar operação A2
- voltar ao início

ii) Máquina B:
- início
- esperar liberação (vinda de A)
- executar operação B1
- liberar A
- executar operação B2
- voltar ao início

A Figura 1.1 apresenta o modelo desta máquina usando redes de Petri etiquetadas. Este mesmo exemplo será retomado nos parágrafos subsequentes que se referem às linguagens ESTELLE e LOTOS.

2. A LINGUAGEM ESTELLE DA ISO

2.1 - Aspectos Básicos

A linguagem ESTELLE é uma técnica de descrição formal desenvolvida pela ISO adequada à descrição dos protocolos e dos serviços oferecidos pelas diversas camadas de um sistema de computadores distribuídos (Linn, 1986), (ISO, 1986).

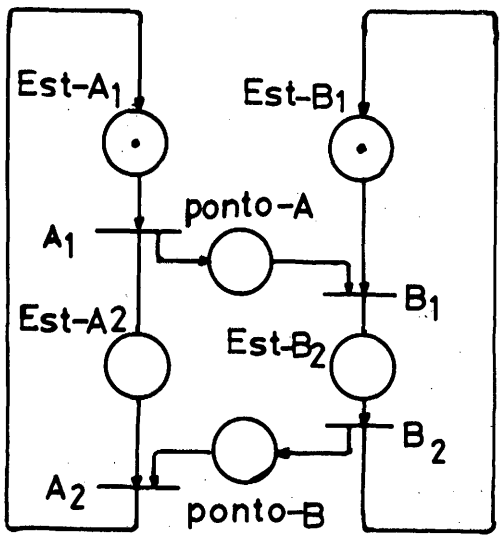

Fig. 1.1 - Modelo de duas máquinas que operam em sincronismo

As bases sobre as quais a linguagem ESTELLE está construída são as Máquinas de Estados Finitas Extendidas e a linguagem PASCAL. Além disso, os princípios de estruturação desta linguagem permitem a construção de especificações modulares. Uma especificação em ESTELLE de uma camada genérica (N) de protocolos consiste da descrição de um conjunto de entidades de protocolo (N) que cooperam utilizando o serviço oferecido pela camada inferior (N-1). Sob o ponto de vista de ESTELLE, tanto as entidas de protocolos (N) quanto o serviço da camada (N-1) são considerados como um conjunto de módulos cooperantes.

2.2 - Conceitos Básicos

Os conceitos básicos em ESTELLE são os seguintes:

. módulo;
. ponto de interação;
. primitivas ESTELLE.

Uma especificação em ESTELLE é constituída por um conjunto de *módulos* que trocam interações através de *pontos de interação* (em alguns casos restritos é permitida a existência de variáveis comuns).

Os módulos são definidos como *tipos*; ao longo de uma especificação podem ser criadas várias *instâncias* de módulos de um determinado tipo.

A linguagem ESTELLE define um conjunto de *primitivas* ESTELLE que permitem a criação, a destruição e a manipulação de instâncias de módulos. Este conjunto de primitivas pode ser visto como sendo o "Sistema Operacional de ESTELLE".

Os próximos parágrafos apresentam estes três conceitos importantes de ESTELLE: módulo, pontos de interação e primitivas ESTELLE. Informações adicionais podem ser encontradas em (ISO, 1986), (Linn, 1986) e (Courtiat, 1986).

2.2.1 - Módulo

O módulo é um componente básico de uma especificação ESTELLE. Deve-se ter bem clara a distinção existente entre o *tipo* de um módulo e uma *instância* de módulo:

Tipo: corresponde à definição de um cabeçalho (*header*) e de um corpo (*body*); vários corpos podem ser definidos para um mesmo cabeçalho;

Instância: uma instância de módulo de um dado *tipo* é criada pela primitiva *init* e destruída pela primitiva *release*. A criação de uma instância implica na associação de um cabeçalho (*header*) a um corpo (*body*), ambos definidos para um dado módulo.

- Definição do cabeçalho ("*header*")

O cabeçalho define a visibilidade externa de cada instância do módulo. Os componentes da definição do cabeçalho são:

. pontos de interação externos
. variáveis exportadas
. parâmetros de inicialização de uma instância
. atributos referentes à estruturação hierárquica: atributo *processo* (*process*), atributo *atividade* (*activity*), atributo processo sistema (*systemprocess*) e atributo atividade de sistema (*systemactivity*). Estes atributos são detalhados no parágrafo "Princípios de Estruturação".

- Definição do corpo ("*body*")

O corpo define o comportamento genérico de cada instância de módulo; para um mesmo cabeçalho podem ser definidos vários corpos. Assim sendo, um módulo pode apresentar vários comportamentos, definidos cada um por um corpo, para uma mesma visibilidade externa, definida pelo cabeçalho.

Os componentes da definição de um corpo de módulo são:

. parte declarativa: declaração de constantes, tipos, cabeçalhos e corpos de módulos filhos, de pontos de interação internos, de variáveis locais, de procedimentos e de funções;

. parte inicialização: declaração de um conjunto de transições das quais uma será executada durante a inicialização de uma instância de módulo;

. parte declaração de transições: declaração das transições da máquina de estados finita extendida que descreve o comportamento do corpo do módulo.

- Princípios de Estruturação

Uma especificação em ESTELLE é composta por uma hierarquia de módulos: no topo desta hierarquia encontram-se os módulos sistema

(atributos *systemprocess* e *systemactivity*).

Um módulo com atributo *systemprocess* pode ser estruturado em submódulos processo (atributo *process*) ou atividades (atributo *activity*). As instâncias de processo se executam em paralelo e as instâncias de atividade se executam sequencialmente.

Um módulo com atributo *systemactivity* pode ser estruturado em submódulos atividade (atributo *activity*).

Um módulo processo pode ser estruturado em submódulos processo ou atividade. Um módulo atividade só pode ser estruturado em submódulos atividade.

A semântica de ESTELLE estabelece que não há execução em paralelo entre uma transição de uma instância de módulo e as transições de seus descendentes. Além disso, uma transição de uma dada instância de módulo tem prioridade em relação às transições das instâncias de seus módulos descendentes.

2.2.2 - Pontos de Interação (IP)

As instâncias de módulos trocam interações através dos Pontos de Interação (IP). É importante notar mais uma vez a distinção existente entre o *tipo* do IP e uma *instância* do IP. As instâncias de IP são criadas durante a criação da instância do módulo ao qual o IP está associado.

Os Pontos de Interação podem ser internos, quando são definidos no corpo de um módulo, ou externos, quando são definidos no cabeçalho do módulo.

A definição de um IP corresponde à definição dos seguintes atributos:

. tipo do canal associado: lista das interações aceitas;

. papel ("role") do ponto de interação: permite definir o sentido em que as interações trafegam entre pontos de interações ligados;

- Associação a filas de entrada que pode ser:

- individual: uma fila é associada a cada instância de ponto de interação;

- comum: uma única fila é associada a todos os pontos de interação de uma dada instância de módulo.

2.2.3 - Primitivas de ESTELLE

As primitivas de ESTELLE permitem:

. a gestão de instâncias de módulos:

  . primitiva Init: cria uma instância de módulo associando um cabeçalho a um corpo de módulo; as instâncias de objetos associados ao módulo são criadas, incluíndo aí as instâncias de pontos de interação e variáveis;

  . primitiva Release: destrói uma instância de módulo.

. a gestão de instâncias de Pontos de Interação (IP):

  . primitivas Connect e Disconnect: conectam e desconectam instâncias de IP de mesmo nível hierárquico;

  . primitivas Attach e Dettach: associam e desassociam instâncias de pontos de interação com níveis hierárquivos adjacentes (IP de módulo pai a IP de módulo filho).

. Sincronismo entre instâncias de módulo:

  . primitiva Output: envio de interação a um IP;

  . primitiva When: recepção de interação depositada em fila de entrada.

2.3 - Um Exemplo

Neste parágrafo, o exemplo apresentado na Figura 1.1 é retomado. A especificação em ESTELLE será composta por dois módulos, cada qual correspondendo ao comportamento de uma das máquinas, conforme ilustrado na Figura 2.1.

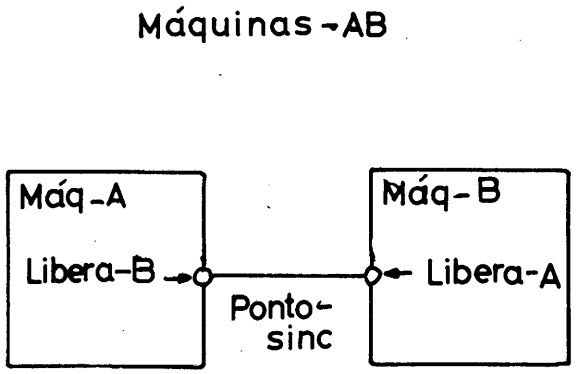

Fig. 2.1 - Esboço da configuração em ESTELLE do exemplo da Figura 1.1.

A especificação completa deste exemplo em ESTELLE é dada abaixo:

```
Specification Máquinas_AB;
  default individual queue;
Channel
  Ponto_sincronismo_type (ponto_A, ponto_B);
by ponto_A: (* role ponto_A*)
Libera_B; (* módulo com IP cujo role é
  "ponto_A" deposita mensagem libera_B*)
by Ponto_B:
Libera_A; (*módulo com IP cujo role é
  "ponto_B" deposita mensagem libera_A*)
(*definição dos módulos que compõem a
  especificação*)
```

```
Module máquina_A_type systemprocess;
ip Ponto Sinc:
  Ponto_sincronismo_type (Ponto_A);
end;
Body Máquina_A_body for Máquina_A_type;
Initialize
to EST_A1;
trans
from EST_A1;
to EST_A2
begin
  (*operação_A1*)
  output Ponto_Sinc.Libera_B
end;
from EST_A2
to EST_A1
when Ponto_Sinc.Libera_A
begin
  (*operação_A2*)
end;
end Máquina_A_body;
Module Máquina_B_type systemprocess;
  ip Pont_Sinc;
    Ponto_sincronismo_type (Ponto_B);
end;
Body Máquina_B_body for Máquina_B_type;
Initialize
to EST_B1
trans
from EST_B1
to EST_B2
when Ponto_Sinc.Libera_B
begin
  (*operação_B1*)
  output Ponto_Sinc.Libera_A
end;
from EST_B2
to EST_B1
begin
  (*operação_B2*)
end;
end Máquina_B_body;
(*declaração das variáveis da especificação*)
var Máquina_A: Máquina_A_type;
    Máquina_B: Máquina_B_type;
(*inicialização (ou configuração) da
  especificação*)
Initialize
begin
  init Máquina_A with Máquina_A_body;
  init Máquina_B with Máquina_B_body;
  connect Máquina_A. Ponto_Sinc
  to Máquina_B. Ponto_Sinc
end;
end Máquinas_AB.
```

3. A LINGUAGEM LOTOS DA ISO

3.1 - Aspectos Básicos

A linguagem de especificação LOTOS (Language for Temporal Ordering Specification), assim como a linguagem ESTELLE, foi concebida explicitamente para descrever sistemas distribuídos e, em particular, os padrões OSI da ISO. Isto não impede que ela seja usada como um modelo de programas e sistemas em geral.De fato, a linguagem LOTOS já foi usada com sucesso na especificação e validação (com a utilização do CCS) de sistemas de controle em tempo real (Puente, 1986).

A especificação de um sistema em LOTOS é feita por um observador externo, que descreve o sistema em termos de processos e eventos. Um sistema, no seu conjunto, pode ser considerado como um único processo que existe num certo ambiente e é capaz de se comunicar com ele. Este processo pode ser refinado em subprocessos que interagem entre si. Estes subprocessos também podem ser por sua vez refinados e assim por diante, de maneira que o sistema original pode ser pensado como uma hierarquia de processos comunicantes. O mecanismo abstrato de comunicação e sincronização entre processos ou entre processo e ambiente é chamado *interação* e sua forma atômica é o *evento*. Um evento é então a unidade de comunicação sincronizada entre dois processos que são de outra forma assíncronos.Este mecanismo de interação em sua forma mais simples não distingue entre entradas ou saídas e portanto diz-se que um processo *"oferece"* participação em um evento. O evento realiza-se quando dois processos oferecem participação em um mesmo evento. A ocorrência destes eventos é que serve basicamente para definir uma ordenação temporal para as ações dos processos.

Um sistema descrito em LOTOS pode então ser visto como uma máquina de estados assíncrona e o comportamento de cada processo pode ser visualizado de forma conveniente através de uma árvore de comportamento.

De maneira a aumentar o poder de expressão da linguagem, acrescentou-se a LOTOS uma linguagem de descrição de dados. Esta linguagem é ACT ONE, uma linguagem de definição de tipos de dados abstratos.

3.2 - A Estrutura de LOTOS

A especificação de um sistema em LOTOS consiste da definição de processos no contexto da definição de tipos de dados abstratos. A sintaxe deste primeiro nível de definição é dada por:

```
<especificação>::=
  Specification<nome-da-especificação>
                {<definição-de-tipo>} ≥ 0
                 <definição-de-processo>
  end spec
```

Como pode ser visto acima, a definição dos tipos de dados é opcional e pode ser múltipla. Existe na linguagem uma biblioteca padrão de tipos mais comuns como Boolean, Natural, Integer, etc.

3.2.1 - A Definição dos Tipos

A sublinguagem de definição de tipos de dados, ACT ONE, possui facilidades que permitem ao especificador definir tipos próprios, através da combinação de tipos já existentes, o enriquecimento de operações, renomeação, parametrização e atualização de tipos.

A sintaxe da <definição-de-tipo> é dada por:

<definição-de-tipo>::=

type <nome-do-tipo> *is*
{<nome-do-tipo> *with*} ≥ 0
<apresentação-do-tipo>
end type

A <apresentação-do-tipo> é definida por uma assinatura ε e um conjunto de equações E:

<apresentação-do-tipo>::=<assinatura>
{<equações>}

A assinatura é composta por um conjunto de nomes de domínios de dados ("sorts"), notados por "S", um conjunto de operadores e sua funcionalidade. A funcionalidade de um operador w é especificada por w: S_1, ..., S_n → S, onde S é o "sort" do dado abstrato resultante da aplicação do operador e S_1, ..., S_n é uma lista possivelmente vazia de "sorts".

<assinatura>::=

{*sorts* <"sort"> {,<"sort">} > 0}
{*opns* {<operador> {<operador>} > 0}:
{<funcionalidade>} ≥ 1}

Uma equação é um par de termos de mesmo "sort".

<equações>::=*equs* {<termo>=<termo>} ≥ 1

<termo>::= variável

<termo>::=<operador> {(<termo> {,<termo>}≥0)}

Um exemplo de definição de um tipo de dados é o seguinte:

```
type  código-primitiva is integer with sorts
      código-primitiva
opns  snd-r, rcv-ak, rcv-i:→ código-primitiva
      code: código-primitiva→integer
equs
      code snd-r=1,
      code rev-ak = 2,
      code rcv-i = 3
end type
```

3.2.2 - A Definição do Processo

O formato da definição de um processo em LOTOS é dado por:

<definição-de-processo>::=
process <abstração-do-processo>
{<especificação-local>}
end proc

<abstração-do-processo>::=
<nome do processo>
<lista-de-portas-formais>
<lista-de-parâmetros-formais>:=
<expressão-do-processo>

<especificação-local>::=
where
{<definição-de-tipo>} ≥ 0
{<definição-de-processo>} ≥ 0

A lista de portas ("gates") formais é uma lista de nomes formais de portas, ou pontos de comunicação e sincronismo entre o processo e o ambiente e outros processos. A lista de parâmetros formais é uma lista de nomes formais de parâmetros passados para o processo especificado. Esta lista pode ser seguida de uma barra vertical "|" e uma lista de nomes formais de parâmetros gerados numa terminação bem sucedida deste mesmo processo, representando sua funcionalidade.

Além da especificação do comportamento do processo dada pela <expressão do processo>, a declaração *where* permite a especificação local de tipos e processos constituindo assim uma facilidade de construção hierárquica descendente ("top-down") com regras de escôpo definidas.

3.2.3 - Expressão do Processo: Os Operadores

A expressão de comportamento do processo ocupa uma posição central em LOTOS, porque é através dela que se pode modelar a ordem de ocorrência dos eventos e a sua dependência de valores. Estas expressões, que definem o comportamento observável do processo, são escritas com o auxílio de um conjunto de operadores temporais. Vamos portanto introduzir os operadores básicos através de pequenos exemplos e mostrar também a árvore de comportamento que lhes corresponde.

O primeiro deles é a *inação*, representada pela palavra chave *stop*. Este operador não executa nenhuma ação. Sua árvore de comportamento é um nó simples.

A seguir temos o *prefixo de ação*, representado pelo símbolo ";"

```
process  P1[g1, g2] :=
         g1; P2
end proc
```

A expressão g1; P2 denota que se o evento g1 ocorrer, o comportamento resultante do processo é descrito por P2. A árvore de comportamento associada é um ramo rotulado por g1, pré-fixado à árvore de P2.

O terceiro operador que completa o conjunto mais elementar é a *escolha*, representado pelo símbolo "[ ]"

```
process  P1[g1, g2]:=
         g1; P2[ ]g2; P3
end proc
```

A árvore de comportamento deste último exemplo é apresentada na Figura 3.1.

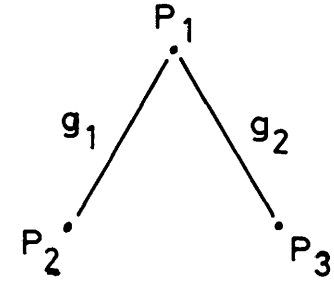

Fig. 3.1 - Árvore de comportamento do operador escolha

A expressão mostrada no exemplo significa que se o evento g1 ocorrer, o comportamento do processo será dado pela expressão P2; caso o evento g2 ocorra, a expressão de comportamento será dada por P3.

O resultado da escolha é determinado pela interação do processo com o ambiente. Caso o ambiente ofereça participação nos eventos g1 e g2 o resultado da escolha é não-determinístico.

Existem três operadores para representar *a composição paralela* de processos e eles se diferenciam pela maneira de exprimir a possibilidade de comunicação interna ou externa através de eventos.

O primeiro deles, representado pelo símbolo "|||", especifica um *paralelismo sem comunicação interna*, onde a comunicação é feita somente através de eventos externos que tenham nomes comuns aos dois comportamentos e com um intercalamento arbitrário, por exemplo:

(g1; stop [ ] g2; stop)|||(g3; g4; stop)

A árvore de comportamento correspondente é dada na Figura 3.2.

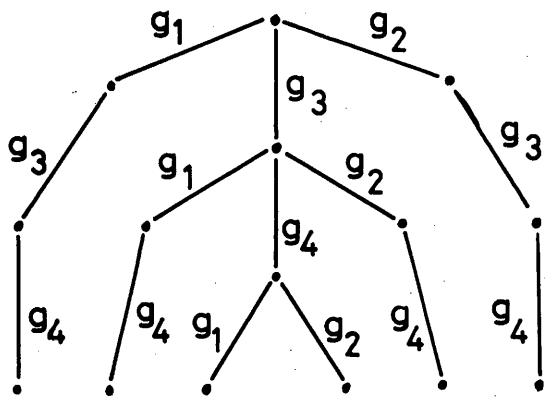

Fig. 3.2 - Árvore de comportamento da composição paralela

O intercalamento arbitrário aproxima a noção de concorrência.

O segundo operador, representado pelo símbolo "||", é usado para especificar o *paralelismo com comunicação interna*, isto é, comunicação através dos eventos comuns aos dois comportamentos, sejam internos ou externos. Surge então, às vezes, a necessidade de especificar alguns eventos que só servirão à comunicação interna, isolando a interação interna do ambiente, e isto é feita através do *operador restrição*, representado pelo símbolo "\".

```
process PIPE [in, out]: =
        (P1[in, out-i ]
        ||P2[out-i, out]
        )\[out-i]
where
   process P1[in, out]: =
           in; out; P1 [in, out]
   end proc
   process P2 [in, out]:=
           in; out; P2 [in, out]
   end proc
end proc
```

O terceiro operador, representado pelo símbolo "|", permite especificar o *paralelismo com possibilidade de comunicação interna somente para um dado sub-conjunto de eventos comuns*.

No exemplo a seguir, somente g1 e g2 são usados para, e somente para, a comunicação interna das duas instâncias de P2:

```
process P1[g1, g2, g3, g4]:=
        (P2[g1, g2, g3, g4]
        |[g1, g2]|
        P2[g2, g1, g3, g4]
        )\[g1,g2]
where process P2[g1, g2, g3, g4]:=
  end proc
end proc
```

Embora a especificação de um processo em LOTOS evidencie somente o comportamento externo, podem existir *eventos internos* que numa situação de escolha sejam relevantes para observações externas. Este tipo de evento interno modela basicamente uma forma de não-determinismo e é representado pelo operador i.

Um exemplo interessante é mostrado abaixo:

```
Process Telefone[Ficha, ligação]:=
        ficha;(ligação; Telefone[ficha, liga
                ção])
        [ ]i; Telefone[ficha, ligação]
end proc
```

Neste caso este telefone pode às vezes "engolir" a ficha e não permitir que se faça a ligação.

Para poder descrever diretamente a *composição sequencial* de processos é primeiro necessário existir uma maneira de indicar a terminação bem sucedida de um processo, já que uma descrição simplesmente pelo comportamento externo não poderia detetar uma malha infinita ("livelock") dentro deste.

O operador que indica a *terminação normal* é o *exit*. Com ele é possível então descrever a composição sequencial de processos e o operador próprio para isto é B1 >> B2. Esta composição indica que se B1 terminar com sucesso, então B2 é habilitado para execução.

No exemplo a seguir, o segundo processo será executado caso ocorra o evento a.

```
Process Prim-proc[a, b]:=
        a; exit >>    Seg-proc[b]
end proc
```

LOTOS também possui um operador para descrever uma forma de *preempção*, representado pelo símbolo "[>". Suponhamos que B1 e B2 são as expressoes de comportamento de dois processos, então B1 [> B2 indica que em qualquer ponto na execução de B1, a ocorrência de um evento inicial de B2 aborta a execução de B1 e o controle passa para B2. Caso o evento inicial de B2 ocorra antes do evento inicial

de B1, este nem será executado.

A *instanciação de processos* em LOTOS possibilita a renomeação de "gates" e a instanciação de comportamento.

Para cada instanciação de processo p[g1, ..., g_m](e1,..., e_n) deve existir somente uma <abstração-de-processo> de nome "p", "m" "gates" formais e "n" parâmetros formais, de maneira que

P[f_1,...,f_m](x_1: S_1,...,x_n: S_n):=P_p, onde P_p é a expressão de comportamento do processo.

3.2.4 - Ofertas Estruturadas de Eventos

Os exemplos mostrados até aqui descrevem processos com ofertas de eventos simples. Podemos entretanto enriquecer as ofertas de eventos basicamente com um de dois atributos possíveis: uma declaração de valor ou uma declaração de variável.

Uma oferta de evento com uma declaração de valor tem por exemplo a seguir forma:

```
Process Transmissor [sap1]:=
        sap1! 'prim-1'; P1
        [ ] sap1! 'prim-2'; P2
end proc
```

A oferta de evento sap1! 'prim-1' indica que o valor 'prim-1' é declarado no "gate" sap1.

Uma oferta de evento com uma declaração de variável tem a forma mostrada no exemplo abaixo:

```
Process Receptor[sap2]:=
        sap2 ? x: primitiva[x='prim-1'];P3
        [ ] sap2? x:primitiva[x='prim-2'];P4
end proc
```

Neste exemplo x é o nome de uma variável e *primitiva* é o identificador do tipo (sort). O *predicado de seleção* que segue a declaração de variável impõe uma restrição aos valores que esta pode assumir, indicando uma forma de construção condicional.

Dependendo dos atributos utilizados com os "gates" dos eventos, teremos as várias formas de comunicação tradicionais: nenhuma interação, sincronização simples, "casamento" de valor, passagem de valor (entrada ou saída) e a geração de valor (não-determinística), como mostrada na Tabela 3.1.

LOTOS ainda possui um operador para descrever "guardas", o *operador condicionador*, que apresenta a seguinte sintaxe [e] → B, onde "e" é do tipo Booleano. A expressão significa que se o valor de "e" for verdadeiro então a expressão de comportamento é dada por B, caso "e" seja falso a expressão de comportamento é dada por "stop". Este operador associado ao operador de escolha permite criar uma estrutura do tipo "case", como mostrado a seguir.

```
Process Amplificador [in, out, chave]:=

   chave ! ligada, in ? x: real;

([x <  0] → out ! 0; amplificador [in, out,
   chave]
[ ] [0 < x < Vsat] → out ! G.x;
                     amplificador [in,out,chave]
[ ] [x > = Vsat] → out ! G.Vsat;
                     amplificador [in,out,chave]
end proc
```

3.2.5 - Funcionalidade de Processos

Conforme já foi visto anteriormente podemos associar à definição do processo uma lista de nomes formais de parâmetros, de maneira que nas ocorrências deste processo, as variáveis declaradas sejam substituídas por expressões de valor de mesmo "sort". É também possível declarar os domínios dos valores que são passados numa terminação bem sucedida do processo. Este produto cartesiano de domínios é chamado de funcionalidade e permite que a especificação seja feita com uma maior confiabilidade.

Para expressar esta passagem de valores entre processos, estendemos a noção do operador de terminação *exit*, associando a este operador a lista dos valores que devem ser passados. Assim temos

exit (e_1, ..., e_n) ou

exit (*any* s)

onde e_1,...,e_n são os parâmetros que serão passados aos processos subsequentes e "any" indica que qualquer valor no domínio "S" pode ser passado numa terminação bem sucedida do processo.

Numa composição sequencial de processos podemos expressar os valores passados da seguinte forma:

B_1 >> *def* x_1: S1,...,x_n: S_n *in* B2

onde a funcionalidade de B_1:func(B1) = domínio (S_1) x...x domínio (S_n) e x_1,...,x_n são nomes de variáveis usadas em B_2 para as expressões de valor passadas numa terminação bem sucedida de B_1.

Neste tipo de composição se faz necessário que a funcionalidade de B_1 seja bem definida, de maneira que toda terminação bem sucedida possível do processo apresente a mesma funcionalidade. Desta forma, algumas restrições devem ser impostas às expressões de comportamento, que são combinadas através de certos operadores, tais como a escolha e a desabilitação.

Um exemplo típico desta passagem de valores é encontrado na especificação do serviço de transporte, quando são negociados, durante a fase de estabelecimento de conexão, o envio de dados urgentes e a qualidade do serviço a ser prestado, como mostrado abaixo:

```
Fase-conexão[tsap1,tsap2](...|quality-serv-
   sort, boolean)
>> def qual-serviço: quality-serv-sort,dado-
   urgente: boolean
   in Fase-transmissão-dados [tsap1, tsap2]
   (qual-serviço, dado-urgente)
```

FORMAS DE INTERAÇÃO ENTRE PROCESSOS				
TIPO DE COMUNICAÇÃO	OCORRÊNCIA NO PROCESSO B1	OCORRÊNCIA NO PROCESSO B2	CONDIÇÃO PARA SINCRONIZAÇÃO	EFEITO VERIFICADO DEPOIS DA SINCRONIZAÇÃO
Sincronização	g; P_1	g; P_2	Nomes formais correspondem a um mesmo "gate"	Cada processo passa a se comportar conforme as expressões descritas (P_1 e P_2, respectivamente)
Sincronização por "casamento" de valor	g! E_1; P_1	g! E_2; P_2	Valores de E_1 e E_2 são iguais	idem
Sincronização com passagem de valor	g! E_1; P_1	g? y: S; P_2	Valor de E_1 pertence ao domínio S	No processo B_2, y=E_1 na expressão de comportamento P_2
Sincronização com geração não-determinística de valor	g? x: S_1	g? y: S_2	Domínios S_1 e S_2 devem ser iguais	x=y=E onde E é um valor definido no domínio S_1 = = S_2

Tabela 3.1 - Formas de interação entre processos

Existem ainda outras facilidades em LOTOS que não detalharemos. Acreditamos contudo que os aspectos abordados até o momento já permitem que se tenha uma visão introdutória da linguagem.

3.3 - Exemplo

O exemplo apresentado na Figura 1.1 é retomado neste parágrafo. A especificação em LOTOS será composta por dois processos, cada qual correspondendo ao comportamento de uma das máquinas cooperantes.

Neste exemplo utilizou-se a sincronização dos processos por casamento de valor (mensagens Libera_A e Libera_B). Os eventos internos i, especificados nos processos Máquina_A e Máquina_B representam as ações A1, A2, B1 e B2, que em LOTOS não são observáveis. A especificação completa é dada abaixo:

```
Specification Máquinas_AB
  Process Máquinas_AB :=
    (Máquina_A[Porta_A]
    ||Máquina_B[Porta_B]
    ||Canal[Porta_A, Porta_B]
    )\[Porta_A, Porta_B]
where
  Process Máquina_A[Porta] :=
    Porta ! Libera_B;
    i; Porta ! Libera_A;
    i; Máquina_A[Porta]
  end proc
  Process Máquina_B[Porta] :=
    Porta ! Libera_A;
    i; Porta ! Libera_B;
    i; Máquina_B[Porta]
  end proc
  Process Canal[Porta_1, Porta_2]:=
    (Porta_1 ! Libera_B;
    Porta_2 ! Libera_B;
    Canal[Porta_1, Porta_2]
    [ ]
    Porta_2 ! Libera_A;
    Porta_1 ! Libera_A;
    Canal[Porta_1, Porta_2]
    )
  end proc
end proc (*Máquinas_AB*)
end spec
```

4. CONCLUSÃO

Este artigo apresentou uma breve introdução às linguagens de especificação ESTELLE e LOTOS, que estão sendo normalizadas pela ISO. Estas duas linguagens apresentam características distintas. A linguagem ESTELLE toma como modelo de base as Máquinas de Estados Finitas Extendidas, enquanto que o modelo de base da linguagem LOTOS é o CCS*, uma extensão do CCS proposto por R. Milner (Milner, 1980). A linguagem ESTELLE adota um modelo já largamente difundido entre os especialistas da área de sistemas distribuídos e atualmente já existem ferramentas disponíveis,a nível de protótipo,que permitem a validação por simulação de mecanismos especificados nesta linguagem (Vissers, 1987). A linguagem LOTOS se baseia num modelo poderoso para a descrição do paralelismo e permite especificações concisas; as ferramentas para esta linguagem se encontram em fase final de desenvolvimento (Vissers, 1987).

A linguagem LOTOS, entretanto, apresenta certas limitações no que diz respeito à sua semântica, como por exemplo, a noção de composição paralela de processos, que aproxima a concorrência pelo conceito de intercalamento de processos, e também a falta de uma noção quantitativa de medida do tempo, que restringe LOTOS à modelagem de sistemas assíncronos somente. Um modelo de base mais poderoso, como o SCCS, proposto também por R. Milner (Milner, 1983) resolveria estes problemas citados.

A noção de equivalência de observação, conforme definida no CCS, distingue processos que nenhuma observação feita pode distinguir e falha na deteção de divergência ("livelock"). Estes problemas foram tratados por R. de Nicola e M.C.B. Henessy em (Nicola, 1983), chegando a resultados satisfatórios.

Uma outra limitação a ser apontada é a ausência de uma possível qualificação na escolha não-determinística entre vários comportamentos, como por exemplo prioridades ou probabilidades.

Um dos caminhos de pesquisas nesta área corresponde ao desenvolvimento de ambientes de programação que permitam escrever especificações de sistemas distribuídos em uma destas linguagens, especificações que serão em seguida validades antes de se passar a geração automática de implementações e ao teste final.

5. BIBLIOGRAFIA

Ben-Ari, M., (1982). *Principles of Concurrent Programming*, Tel-Aviv, Prentice-Hall Int., Capítulo 6: 102-104.

Berthomieu, B. & Menasche, M., (1983). An Enumerative Approach for Analyzing Time Petri Nets. *Proc. IFIP Congress*, Paris: 41-46.

Bochman, G.V., (1978). "Finite State Description of Communication Protocols". *Computer Networks*, nº 2: 361-372.

Brinksma, E., (1986). A Tutorial on LOTOS, *Int. Workshsop on Prot. Cpecif., Verif. and Testing*, 5, Toulouse.

Carchiolo, V.; Faro, A.; Mirabella, O.; Papalardo, G. & Scollo, G., (Nov. 1986). A LOTOS Specification of the PROWAY HIGHWAY Service. *IEEE Trans. on Computers*, Vol. C-35, nº 11: 949-967.

Courtiat, J.P.; Pedroza, A. & Ayache, J.M. , (1986). A Simulation Environment for Protocol Specification Described in ESTELLE. *Int. Workshop on Prot. Specif., Verif. and Testing*, 5, Toulouse.

Diaz, M., (1982). Modelling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models". *Computer Networks*, Vol. 6, nº 6: 419-441.

Diaz, M. & Guidacci da Silveira, C., (1983). On the Specification and Validation of Protocols by Temporal Logic and Nets". *Proc. of the IFIP Congress*, Paris.

Linn, R. J., (1986). The Features and Facilities of ESTELLE". *Int. Workshop on Prot. Specif. Verig. and Testing*, 5, Toulouse.

Milner, R., (1980). A Calculus of Communicating Systems". *Lect. Notes in Comp. Science*, nº 92, Springer-Verlag.

Milner, R., (1983). Calculi for Synchrony and Asynchrony". *Theor. Comp. Sci.*, Vol. 25: 267-310.

Nicola, R. de & Henessy, M.C.B., (Junho,1983). Testing Equivalences for Processes". Dept. Comput. Sci., Univ. Edinburg, Internal Rep. CSR-123.

Puente, J.A. de la; Crespo, A. & Pérez, T. (1986). Especification Algebraica de un Sistema de Control Industrial. *Automática e Instrumentacion*, Barcelona, Nov. 86: 149-152.

Vissers, Chris A., (1987). Trends and Proliferation of Formal Description Techniques for OSI Standards". *Simpósio Brasileiro de Redes de Computadores*, 5, São Paulo.

_____ . (Dez, 1986). ESTELLE - A Formal Description Technique Based on an Extended State Transition Model. ISO TC 97, SC 21, WG-1 FDT, SG-B.

_____. (Dez. 1986). LOTOS - A Temporal Ordering Specification. ISO TC 97, SC 21, WG-1, FDT, SG-C.