# Utilização de restrições e recursos predefinidos na especificação de protocolos com a TDF LOTOS

*Mirela Sechi Moretti Annoni Notare* *
*Bernardo Gonçalves Riso* **

Universidade Federal de Santa Catarina (UFSC) - Centro Tecnológico
Departamento de Informática e de Estatística ( INE / CTC / UFSC )
88040 900 Florianópolis - SC
Fone: (0482) 319498 Telex: (0482) 240 UFSC BR Fax: (0482) 319770
e-mail: mirella@inf.ufsc.br riso@inf.ufsc.br

## Sumário

Neste trabalho apresenta-se uma utilização de construções predefinidas, descritas em LOTOS, para a especificação de sistemas distribuídos. Tais construções são úteis em todas as fases do ciclo de vida de um sistema e podem ser descritas tanto no estilo monolítico como no estilo orientado para estados. Seu emprego pode se dar na representação de aspectos internos e externos de sistemas. No caso de aspectos internos, os sistemas são descritos no estilo orientado para recursos e, nesse caso, uma biblioteca fornece os recursos utilizados. No caso de aspectos externos, os sistemas são descritos no estilo orientado para restrições e, nesse outro caso, a biblioteca fornece as restrições. Cada construção predefinida pode ser associada a diferentes tipos de dados e mapeada para construções em linguagens de programação, facilitando as tarefas de implementação e manutenção de sistemas.

## 1. Introdução

O projeto de sistemas distribuídos envolve a superação de dificuldades que não estão presentes no caso de sistemas convencionais. Essas dificuldades são resultantes de fenômenos de paralelismo (onde vários processos são executados simultaneamente), concorrência (com a cooperação dos processos envolvidos), sincronização (de modo que os processos possuem alguma interdependência) e outros.

Muitos desses fenômenos, que dão grande complexidade aos sistemas distribuídos reais, podem ser compreendidos através da análise de exemplos simples. Eventualmente, os mecanismos propostos como solução para os casos

* Aluna do Curso de Pós-Graduação (Mestrado) em Ciência da Computação da UFSC.
** Doutor em Engenharia Elétrica (UFPB, 1991). Professor adjunto da UFSC.

simples são extensíveis para situações mais abrangentes.

A complexidade dos sistemas distribuídos reais sugere que sejam adotadas metodologias de projeto e desenvolvimento baseadas em técnicas de descrição formais (TDFs) e assim garantir descrições completas e não-ambíguas. Para protocolos de comunicação, utilizados em redes de computadores, a percepção da importância do uso de TDFs levou à padronização das linguagens SDL (pelo CCITT), Estelle e LOTOS (ambas pela ISO).

Os conceitos relacionados à arquitetura de sistemas, embutidos nessas TDF's aliados aos recursos de análise das descrições permite a realização de projetos e desenvolvimentos de modo rápido e seguro. Essas vantagens podem ser ainda maiores caso haja a disponibilidade de ferramentas que conduzam à automatização de tarefas.

LOTOS [ISO 8807] é uma TDF cujo modelo baseia-se em sistemas de transições rotuladas. Ela reúne duas álgebras: a primeira, que corresponde a uma extensão de CCS [Miln 80,89], trata os aspectos de comportamento dos sistemas, enquanto que a segunda, baseada em ACT ONE [EhMa 85], trata os aspectos de dados.

Embora tenha sido criada especificamente para a descrição e a análise de serviços e protocolos do Modelo OSI [ClJo 92], LOTOS também tem sido empregada com sucesso em outras áreas. Por exemplo, no caso de sistemas operacionais distribuídos [PiCu 90] [Pech 92], telefonia [ErHo 92] [DrCh 92] e teste [Sanz 92].

O emprego eficiente de TDFs como LOTOS exige o estabelecimento de metodologias que guiem os seus usuários. Freqüentemente, as metodologias propostas apóiam-se na idéia de ciclo de vida de um sistema, onde as fases principais são as de especificação, implementação e teste.

Para a elaboração de especificações formais têm sido propostas abordagens baseadas em refinamentos sucessivos, com prova de correção *a posteriori* [PiLo 90], e abordagens baseadas em transformações, que dispensam tais provas uma vez que as regras de transformação utilizadas levam a resultados sempre corretos [BoGo 86 ] [Lang 90]. Em qualquer abordagem diferentes estilos de especificação podem ser explorados para dar elegância e agilidade ao processo de desenvolvimento.

Em [ViSc 88], [ViSc 89] e [ISO 3067] quatro estilos fundamentais são analisados. Os dois primeiros dão ênfase à representação dos aspectos externos de um sistema. São eles: o estilo monolítico (onde o sistema é representado por um único processo) e o estilo orientado para restrições (onde o sistema é representado por um conjunto de características). Os dois últimos dão ênfase à representação dos aspectos internos. São eles: o estilo orientado para estados (onde o sistema é representado de modo a explicitar os estados que ele pode assumir) e o estilo orientado para recursos (onde o sistema é representado por

uma combinação de subsistemas). Outro estilo que começa a ser usado é o estilo orientado a objetos [MoCl 93] [KoBo 91] [Mayr 89].

O principal objetivo deste trabalho é propor a utilização de construções predefinidas (disponíveis em um Banco de Dados) no intuito de se obter maior rapidez e confiablilidade no projeto e desenvolvimento de sistemas. Tais construções predefinidas podem ser utilizadas em todos os níveis de abstração do projeto de um sistema.

Este trabalho está organizado da seguinte maneira: no item 2 mostra-se um breve resumo da linguagem LOTOS; no item 3 apresenta-se um processo de geração sistemática de construções predefinidas; no item 4 faz-se uma aplicação de restrições básicas predefinidas em um caso simples mas não trivial. Seguem-se as conclusões (item 5), agradecimentos (item 6) e bibliografia (item 7).

## 2. Noções da técnica de descrição formal LOTOS

Nesta seção apresenta-se um resumo de LOTOS para permitir que o leitor ainda não familiarizado com essa técnica leia facilmente as especificações das seções subseqüentes. Esse resumo baseia-se no padrão ISO 8807.

Em LOTOS um processo é representado por uma caixa preta dotada de portas para a comunicação com o ambiente e com outros processos [Brin 88]. A definição de um processo tem o seguinte formato:

```
process
  <identificador>[<lista-de-parâmetros>]:<funcionalidade>:=
  <expressão-de-comportamento>
endproc
```

onde o `identificador` é o nome do processo e a `lista-de-parâmetros` refere-se às portas de comunicação. A funcionalidade de um processo é `exit` se ele termina com sucesso, habilitando um processo subseqüente. A funcionalidade é `noexit` em caso contrário (por exemplo, no caso de processos infinitos).

Uma expressão de comportamento em LOTOS relata o que pode ser observado em termos de eventos. Nela podem ser usados operadores como:

| | |
|---|---|
| `;` | seqüenciamento de eventos; |
| `[]` | escolhas indeterminísticas entre comportamentos; |
| `\|\|\|` | composição de processos independentes; |
| `\|\|` | composição de processos dependentes; |
| `\|[ ]\|` | composição geral; |
| `hide...in...` | ocultação de portas; |
| `>>` | habilitação de processos; e |
| `[>` | interrupção de processos. |

Os processos `stop` e `exit` pertencem à linguagem LOTOS. Nenhum deles possui qualquer evento, quer seja observável (Act - { i } ) ou invisível (i). O processo `stop` representa a ausência completa de atividade. Um exemplo de sua utilização é dado a seguir:

```
process buffer[ent, sai]: noexit:=
     ent; sai; stop
endproc
```

Nesse caso, há a ocorrência do evento `ent` seguido do evento `sai`. Após o segundo evento tem-se um processo completamente inativo (`stop`).

O processo `exit`, por sua vez, indica uma terminação com sucesso, habilitando a realização de algum comportamento subseqüente. Um exemplo de sua utilização é dado a seguir:

```
process le_escreve[ in, out] : noexit :=
     leitura[in] >> escrita[out]
     where  -  process leitura[in]: exit :=
               in; exit
               endproc
               process escrita[out]: noexit :=
               out; stop
               endproc
endproc
```

Nesse caso, após a execução do processo `exit` o controle é transferido para o estado inicial do processo `escrita[out]`.

Outro meio de expressão é a chamada recursiva de processos para representar comportamentos infinitos. Exemplo:

```
process Ciclico_2 [a1, a2]: noexit :=
      a1; a2;  Ciclico_2 [a1, a2]
endproc
```

Neste exemplo o processo `Ativo` repete a seqüência de eventos `a1; a2` infinitamente.

Tutoriais sobre LOTOS podem ser encontrados em [ISO 8807] e [BoBr 87]. Um estudo de LOTOS a partir de exemplos é apresentado em [DrCh 92].

## 3. Construções predefinidas em LOTOS

Partindo dos casos mais simples, os primeiros processos que podem ser incluídos em uma biblioteca de processos básicos são `stop` e `exit`.

O passo seguinte para a geração sistemática de processos básicos, cada vez mais complexos, leva aos processos finitos, nos quais, após a execução de uma seqüência de eventos, a atividade cessa completamente. Exemplo:

```
process Sumidouro [absorve] : noexit:=
    absorve; stop
endproc
```

Este é um processo que absorve um conjunto de valores e não age mais.

Agora podem ser considerados os processos básicos, finitos e seqüenciais, que têm expressão de comportamento da forma

$$B = \left( \prod_{j=1}^{n} aj \right); \texttt{stop} = a1; a2; \ldots; an; \texttt{stop} \quad aj \in Act$$

Nesse caso tem-se a possibilidade de definir seqüências sem repetição ou com repetição de eventos. A tais processos podem ser atribuídas as funcionalidades `exit` ou `noexit`. Além disso, a utilização de diferentes identificadores para os processos e para as portas permite as variações semânticas que interessam às aplicações. Exemplo:

```
process Convite_Seletivo [ a1, a2, a3]: noexit :=
    a1; a2; a3; stop
endproc
```

O caso mais simples de processos indeterminísticos é o do comportamento que apresenta uma escolha com duas opções. Cada uma dessas opções pode ter o comportamento de um processo básico já definido, representado por:

$$\sum_{i=1}^{2} Bi = B1 \; [] \; B2$$

onde `Bi` é `stop` ou `exit` para representar a escolha entre dois comportamentos sem eventos, ou ainda

$$B = \left( \pi_{i=1}^{m} ai \right); \textbf{stop} \; [] \; \left( \pi_{i=1}^{n} bi \right); \textbf{stop} \qquad ai, bi \in Act$$

para representar uma escolha entre duas seqüências de eventos. Exemplo:

```
process Escolha_2 [a1, a2]: noexit :=
     a1; stop
 []  a2; stop
endproc
```

expressando uma exclusão mútua entre os eventos `a1` e `a2`. Qualquer que seja o evento ocorrido, o comportamento posterior é completamente inativo.

Nessa linha, outros processos básicos podem ser definidos, considerando uma quantidade finita de opções entre seqüências de eventos. Cada seqüência, que envolve a ocorrência de eventos observáveis intercalados com eventos invisíveis, pode ter uma finalização com `stop` ou `exit`.

A introdução de recursividade (tail recursivity) cria a possibilidade de definição de processos básicos infinitos. Nesse caso, os processos têm funcionalidade `noexit`. Exemplo:

```
process Sumidouro_rec [absorve] : noexit :=
     absorve; Sumidouro_rec[absorve]
endproc
```

que define um comportamento infinito, sempre pronto para absorver valores.

A introdução do operador `[]` permite expressar escolhas indeterminísticas no comportamento infinito. Exemplo:

```
process Multi_recursivo [a1, ..., an] : noexit :=
     n
     Σ ( aj Multi_recursivo [a1, ..., an] )
    j=1
endproc
```

Neste caso, há uma escolha indeterminística entre `n` opções de eventos. Esse indeterminismo é resolvido com a participação do ambiente (na ocorrência de eventos observáveis) ou internamente (na ocorrência do evento invisível *i*).

A consideração conjunta de aspectos de dados e aspectos de comportamento permite a definição de processos estruturados básicos. Esses processos são definidos em LOTOS Completo.

## 4. Utilização de restrições predefinidas

Nesta seção ilustra-se, com um exemplo simples, uma abordagem baseada em refinamentos sucessivos, para o projeto de sistemas distribuídos e protocolos de comunicação, que utiliza um conjunto de construções predefinidas. O exemplo considera um sistema do tipo Pergunta-Resposta onde um provedor de comunicação confiável e half-duplex, `Serv_Com`, transmite mensagens entre os seus dois usuários. Veja a Figura 4.1.

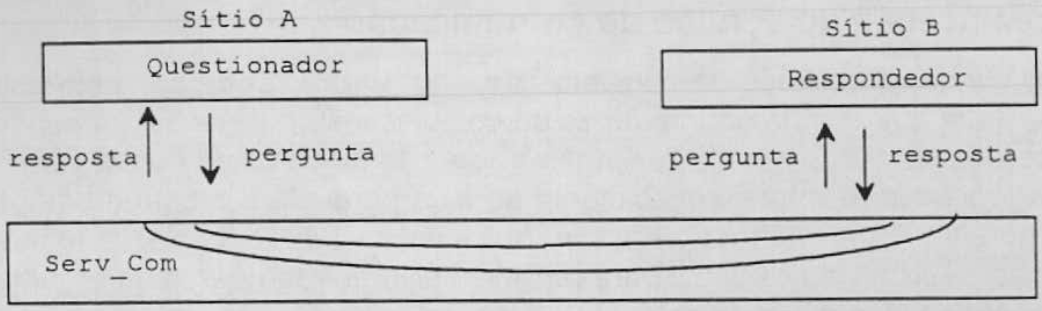


Figura 4.1 - Sistema `Pergunta-Resposta`

O sistema `Pergunta_Resposta` compreende dois sítios: o sítio A e o sítio B. No sítio A localiza-se o usuário `Questionador`, que formula as perguntas, entrega-as (uma por vez) ao provedor de comunicação `Serv_Com` e dele recebe as respostas correspondentes. No sítio B localiza-se o usuário `Respondedor`, que recebe as perguntas de `Serv_Com`, obtém as respostas correspondentes (através da consulta a uma base de dados local ao sítio B) e transmite-as para o usuário `Questionador`.

### 4.1. Especificação de um sistema Pergunta-Resposta

Em LOTOS o sistema `Pergunta-Resposta` pode ser especificado considerando que os usuários, `Questionador` e `Respondedor`, formam uma parte do sistema e o provedor de comunicação, `Serv_Com`, forma a outra parte. Como os usuários não se comunicam diretamente, eles podem ser combinados com o uso do operador ||| (composição independente). Como o conjunto de usuários comunica-se diretamente com o provedor de comunicação, a combinação das duas partes do sistema pode ser realizada com o uso do operador |[ ]| (composição geral), permitindo explicitar os eventos comuns às duas partes:

```
specification Pergunta-Resposta [...] : noexit :=
    behaviour
    (Questionador [...]
```

```
    |||
    Respondedor [...] )        (* primeira parte *)
    |[ ... ]|
    Serv_Com [ ...]            (* segunda parte *)
where process Questionador [...]: noexit :=  ...  endproc
      process Respondedor [...] : noexit :=  ...  endproc
      process Serv_Com [...]: noexit :=      ...  endproc
endspec
```

## 4.2. Especificação do serviço de comunicação

O provedor do serviço de comunicação, `Serv_Com`, pode ser visto como uma caixa preta que dispõe de interfaces de comunicação com os seus usuários. A interface de `Serv_Com` com o `Questionador` é formada pelas portas `perg_A` e `resp_A` (para representar o recebimento de uma pergunta e a entrega de uma resposta no sítio A). A interface de `Serv_Com` com o `Respondedor` é formada pelas portas `perg_B` e `resp_B` (para representar a entrega de uma pergunta e o recebimento de uma resposta no sítio B). Veja a Figura 4.2.

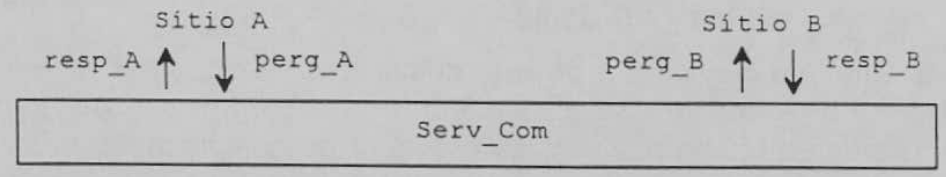


Figura 4.2 - Interfaces de `Serv_Com`

Utilizando o estilo orientado para restrições, o serviço de comunicação oferecido por `Serv_Com` pode ser especificado a partir de aspectos de `Localidade` (de cada sítio do sistema de comunicação) e de aspectos `Fim-a-fim` (ligando os sítios). Veja a Figura 4.3.

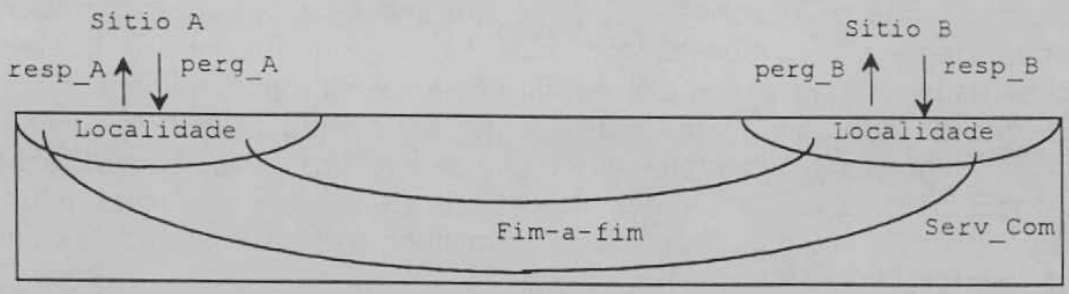


Figura 4.3 - Restrições de `Serv_Com`

Os aspectos de Localidade compartilham todos os eventos com os aspectos Fim-a-fim. Por causa disso, o operador ||(composição dependente) pode ser usado na especificação do serviço oferecido por Serv_Com:

```
process Serv_Com [perg_A, resp_A, perg_B, resp_B]: noexit :=
     Localidade  [perg_A, resp_A, perg_B, resp_B]
 ||  Fim_a_fim   [perg_A, resp_A, perg_B, resp_B]
     where      process Localidade[...]        ...  endproc
                process Fim_a_fim[...]         ...  endproc
endproc
```

Esses aspectos de Localidade, compreendem os aspectos locais do sítio A (Loc_A) e os aspectos locais do sítio B (Loc_B). Veja a Figura 4.4.

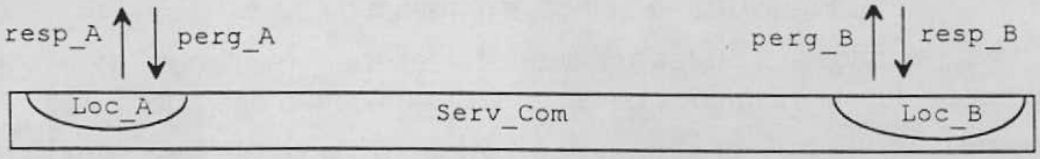


Figura 4.4. - Aspectos locais de Serv_Com

Como Loc_A e Loc_B constituem restrições locais independentes, elas devem ser combinadas através do uso do operador |||.

```
process Localidade [perg_A,resp_A,perg_B,resp_B] : noexit :=
     Loc_A  [perg_A, resp_A]
 ||| Loc_B  [perg_B, resp_B]
     where      process Loc_A [...]       ...  endproc
                process Loc_B [...]       ...  endproc
endproc
```

As restrições locais expressas por Loc_A envolvem o recebimento da pergunta (na porta perg_A) e a entrega da resposta (na porta resp_A):

```
process Loc_A [perg_A, resp_A] : noexit :=
     perg_A; resp_A; Loc_A[ perg_A, resp_A]
endproc
```

As restrições locais expressas por Loc_B envolvem a entrega da pergunta (na porta perg_B) e o recebimento da resposta (na porta resp_B):

```
process Loc_B [pergB, resp_B] : noexit :=
     perg_B; resp_B; Loc_B [perg_B, resp_B]
endproc
```

As restrições `Fim-a-fim` referem-se aos aspectos de comunicação que ligam os sítios A e B. Elas podem ser estruturadas segundo o fluxo de questões (`F_q`) e segundo o fluxo de respostas (`F_r`). Veja a Figura 4.5.

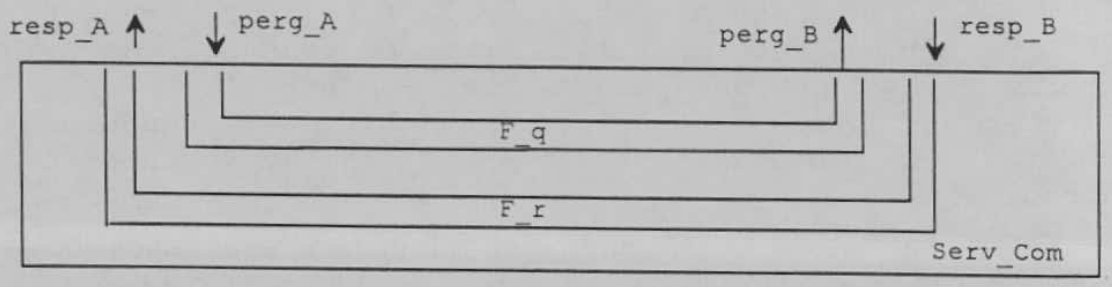


Figura 4.5 - Restrições fim-a-fim de `Serv_Com`

As restrições `F_q` são independentes das restrições `F_r`. Assim, a combinação dessas restrições deve ser realizada com o operador `|||`:

```
process Fim_a_fim [perg_a,resp_A,perg_B,resp_B] : noexit :=
     F_q [perg_A, perg_B]
 ||| F_r [ resp_A, resp_B]
     where
     process F_q [...]    ...  endproc
     process F_r [...]    ...  endproc
endproc
```

As restrições `F_q` correspondem ao recebimento de uma pergunta (na porta `perg_A`) e à sua entrega (na porta `perg_B`):

```
process F_q [perg_A, perg_B] : noexit :=
     perg_A; perg_B; F_q [perg_A, perg_B]
endproc
```

As restrições `F_r` correspondem ao recebimento de uma resposta (na porta `resp_B`) e à sua entrega (na porta `resp_A`):

```
process F_r [resp_A, resp_B] : noexit :=
     resp_B; resp_A; F_r[ [resp_A, resp_B]
endproc
```

Os processos `Loc_A`, `Loc_B`, `F_q` e `F_r` são restrições simples que podem ser obtidas a partir de uma biblioteca de construções predefinidas. Elas realizam um comportamento cíclico que repete dois eventos infinitamente. Tal comportamento é expresso pela construção predefinida `Ciclico_2`.

Neste exemplo, a especificação do serviço oferecido por `Serv_Com` é realizada no estilo orientado para restrições. As restrições de `Localidade` e

Fim-a-fim envolvem restrições simples que reúnem dois processos através de um operador de composição. Pela sua reusabilidade, tais restrições podem constar em uma biblioteca de construções predefinidas.

## 4.3. Especificação do protocolo de comunicação

Pode-se conceber um provedor de serviço de comunicação Serv_Com' como um refinamento do provedor Serv_Com. Tal refinamento é uma combinação de três recursos: uma entidade de protocolo Ent_A (no sítio A), uma entidade de protocolo Ent_B (no sítio B) e um provedor de serviço de comunicação subjacente, Sub_Com, que não é confiável (isto é, que pode perder ou danificar mensagens). Veja a Figura 4.6.

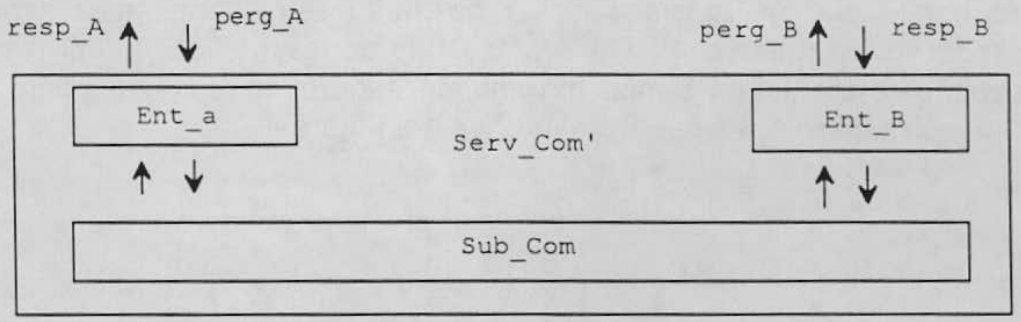


Figura 4.6. - O sistema de comunicação Serv_Com'

O serviço de comunicação oferecido por Serv_Com' pode ser especificado a partir de Ent_A, Ent_B e Sub_Com. Uma vez que Sub_Com não é confiável, as entidades Ent_A e Ent_B cumprem um protocolo para recuperar erros e perdas através de um mecanismo baseado no reenvio de mensagens. Tal protocolo implementa o serviço Serv_Com' de modo que

Serv_Com' Serv_Com

isto é, Serv_Com' e Serv_Com devem ser equivalentes quanto à observação. Assim, Serv_Com' é um refinamento válido de Serv_Com.

O protocolo executado pelas entidades Ent_A e Ent_B inclui a identificação de mensagens, através de sua numeração, para evitar que a base de dados localizada no sítio B seja consultada mais de uma vez com a mesma pergunta. A entidade Ent_B deve ter recursos de memória para armazenar uma resposta até certificar-se de que ela foi corretamente transferida.

Devido às características do serviço Pergunta-Resposta, pode-se adotar um protocolo de janela deslizante com largura máxima de janela igual a 1. Assim, a qualquer momento haverá, no máximo, uma pergunta sem estar

respondida, o que permite numerar as perguntas (e as respostas correspondentes) com 0 e 1, alternadamente.

### 4.3.1. A entidade de protocolo Ent_A

Na interface superior a entidade `Ent_A` possui duas portas, `perg_A` e `resp_A`, para realizar a comunicação com o usuário `Questionador`. Na interface inferior, a entidade `Ent_A` possui oito portas,

`pi, ri, ei, ti` `i = 0, 1`

onde `pi` são as portas usadas para o envio de perguntas, `ri` para o recebimento de respostas corretas, `ei` para o recebimento de respostas com erro (`e0` no caso de estar esperando uma resposta com número zero e receber uma resposta danificada na transmissão; `e1` de modo semelhante, mas esperando uma resposta com número 1). Os eventos nas portas `t0` e `t1` representam a temporização referente ao tempo máximo de espera por uma resposta com o número 0 e o número 1, respectivamente. Veja a Figura 4.7.

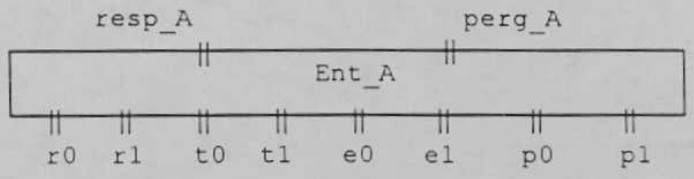


Figura 4.7 - A entidade `Ent_A`

O comportamento da entidade `Ent_A` pode ser segundo as restrições de interface, `R_itf`, e as restrições de ligação das interfaces, `R_lig`, similares às restrições de `Localidade` e `Fim-a-fim` de uma especificação de serviço. Veja a Figura 4.8.

Utilizando o estilo orientado para restrições, a entidade `Ent_A` pode ser especificada como uma composição (`||`) das restrições `R_itf` e `R_lig`:

```
processEnt_A[perg_A,resp_A,r0,r1 t0,t1,e0,e1,p0,p1]:noexit :=
     R_itf [perg_A, resp_A, r0,r1,t0,t1,e0,e1,p0,p1 ]
 ||  R_lig [perg_A, resp_A, r0,r1,t0,t1,e0,e1,p0,p1 ]
     where process R_itf [...]          ...  endproc
           process R_lig [...]          ...  endproc
endproc
```

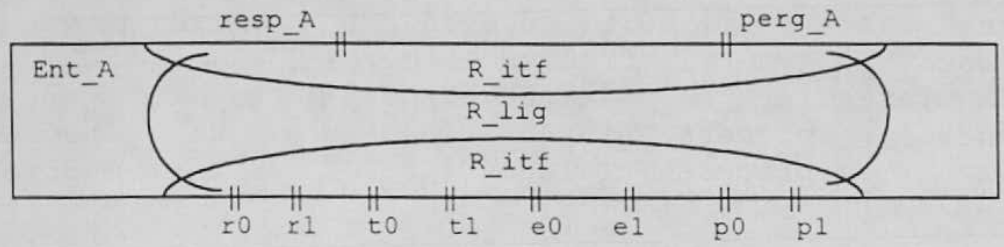


Figura 4.8 - Restrições da entidade Ent_A

Os aspectos de interface, por sua vez, compreendem os aspectos de interface superior (R_sup) e os aspectos de interface inferior (R_inf) que são compostos independentemente (|||):

```
processR_itf[perg_A,resp_A,r0,r1,t0,t1,e0,e1,p0,p1]:noexit :=
      R_sup [perg_A, resp_A ]
 |||  R_inf [r0,r1,t0,t1,e0,e1,p0,p1]
      where      process R_sup [...]          ...  endproc
                 process R_inf [...]          ...  endproc
endproc
```

### 4.3.1.1. Restrições da interface superior

As restrições R_sup reproduzem, exatamente, o comportamento da restrição Loc_A da especificação de serviço Serv_Com. Por isso, o processo R_sup (restrições da interface superior) pode ser obtido a partir de Loc_A por um mapeamento direto de comportamento (transformação identidade):

```
process R_sup [ perg_A, resp_A ] : noexit :=
      perg_A; resp_A ; R_sup [ perg_A, resp_A ]
endproc
```

### 4.3.1.2. Restrições da interface inferior

As restrições R_inf podem ser especificadas de modo estruturado, considerando as restrições das fases de numeração das mensagens (Fases_0_1) e as restrições entre essas fases (Int_fases). Veja a Figura 4.9.

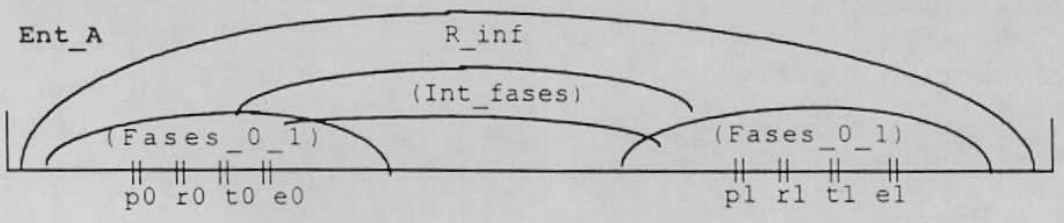


Figura 4.9 - Estruturação das restrições R_inf

```
process R_inf [ r0,r1,t0,t1,e0,e1,p0,p1 ]: noexit :=
     Fases_0_1 [ r0,r1,t0,t1,e0,e1,p0,p1 ]
  || Int_fases [ r0,r1,t0,t1,e0,e1,p0,p1 ]
     where      process Fases_0_1 [...]        ...  endproc
                process Int_fases [...]        ...  endproc
endproc
```

As restrições das fases de numeração das mensagens (representadas pelo processo `Fases_0_1`) podem ser estruturadas segundo duas fases independentes: a primeira, `Fase_0`, é relativa à tentativa de envio de perguntas e recebimento de respostas com o número 0. A segunda, `Fase_1`, é relativa às tentativas de envio de perguntas e recebimento de respostas com o número 1. Veja a Figura 4.10.

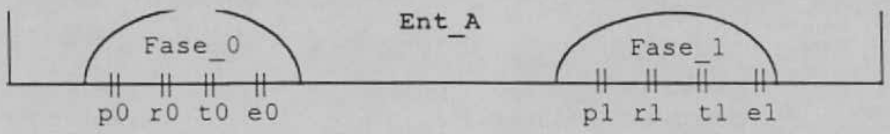


Figura 4.10 - Restrições `Fase_0` e `Fase_1` de `Ent_A`

```
process Fases_0_1 [ r0,r1,t0,t1,e0,e1,p0,p1 ]: noexit :=
     Fase_0 [ r0, t0, e0, p0 ]
  ||| Fase_1 [ r1, t1, e1, p1 ]
     where      process Fase_0 [...]           ...  endproc
                process Fase_1 [...]           ...  endproc
endproc
```

As restrições representadas pelo processo `Fase_0` envolvem o envio de uma pergunta (o evento `p0` ocorre) e a espera pela resposta correspondente, que pode chegar incólume (`r0`), com erro (`e0`) ou não chegar (`t0`).

A especificação de `Fase_0` pode ser feita considerando que, se a resposta chega incólume, o ciclo recomeça, caso contrário, a tentativa é repetida e continua a espera pela resposta correta:

```
process Fase_0 [ r0, t0, e0, p0 ]: noexit :=
     p0 ; Fase_0' [r0, t0, e0, p0 ]
where process Fase_0' [ r0, t0, e0, p0 ] noexit :=
          r0; Fase_0  [ r0, t0, e0, p0 ]
     []   e0; p0; Fase_0' [ r0, t0, e0, p0 ]
     []   t0; p0; Fase_0' [ r0, t0, e0, p0 ]
     endproc
endproc
```

De modo semelhante podem ser especificadas as restrições `Fase_1`:

```
process Fase_1 [ r1, t1, e1, p1 ]: noexit :=
     p1 ; Fase_1' [r1, t1, e1, p1 ]
where process Fase_1' [ r1, t1, e1, p1 ] noexit :=
          r1; Fase_1 [r1, t1, e1, p1 ]
     []   e1; p1; Fase_1' [r1, t1, e1, p1 ]
     []   t1; p1; Fase_1' [r1, t1, e1, p1 ]
     endproc
endproc
```

Já as restrições `Int_fases`, interligando `Fase_0` com `Fase_1`, amarram o envio de uma pergunta (0 ou 1) com o recebimento da resposta correspondente:

```
process Int_fases [r0,r1,t0,t1,e0,e1,p0,p1 ]: noexit :=
     Int_fases_0 [ r0, t0,  e0, p0 ]
 >>  Int_fases_1 [ r1, t1, e1, p1 ]
where process Int_fases_0 [ r0, t0, e0, p0 ] : noexit :=
          p0; Int_fases_0 [ r0, t0, e0, p0 ]
     []   e0; Int_fases_0 [ r0, t0, e0, p0 ]
     []   t0; Int_fases_0 [ r0, t0, e0, p0 ]
     []   r0; exit
     endproc
     process Int_fases_1  [ r1, t1, e1, p1 ] : noexit :=
          p1; Int_fases_1 [ r1, t1, e1, p1 ]
     []   e1; Int_fases_1 [ r1, t1, e1, p1 ]
     []   t1; Int_fases_1 [ r1, t1, e1, p1 ]
     []   r1; Int_fases [ r0, r1, t0, t1, e0, e1, p0, p1 ]
     endproc
endproc
```

### 4.3.1.3. Restrições de ligação das interfaces

As restrições de ligação das interfaces, `R_lig`, podem ser estruturadas segundo o fluxo de questões (`Lig_quest`) e o fluxo de respostas (`Lig_resp`) na entidade `Ent_A`. Veja a Figura 4.11.

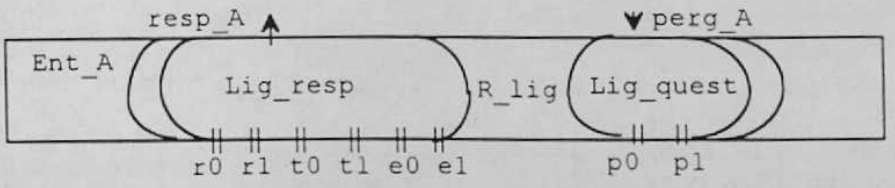


Figura 4.11 - Restrições `Lig_quest` e `Lig_resp` da `Ent_A`

Como as restrições `Lig_quest` são independentes das restrições `Lig_resp`, elas podem ser combinadas com o uso do operador `|||`:

```
process R_lig[perg_A,resp_A,r0,r1,t0,t1,e0,e1,p0,p1]:noexit :=
     Lig_quest [perg_A, p0, p1 ]
 ||| Lig_resp  [resp_A, r0, r1, t0, t1, e0, e1]
     where      process Lig_quest [...]          ...  endproc
                process Lig_resp [...]           ...  endproc
endproc
```

O processo `Lig_quest` liga a entrada de uma pergunta, na interface do usuário `Questionador` (porta `perg_A`), com o envio dessa pergunta na interface do serviço de comunicação subjacente `Sub_Com` (na porta `p0` ou `p1`):

```
process Lig_quest [perg_A,p0,p1]: noexit :=
     perg_A; Lig_quest'  [ p0, p1 ]
where process Lig_quest' [ p0, p1 ]: noexit :=
                p0; Lig_quest'_0 [perg_A, p0 ]
          []    p1; Lig_quest'_1 [perg_A, p1 ]
where process Lig_quest' _0 [ perg_A , p0 ]: noexit :=
                perg_A; Lig_quest' [ p0 , p1]
          []    p0; Lig_quest'_0 [perg_A, p0 ]
          endproc
          process Lig_quest' _1 [ perg_A , p1 ]: noexit :=
                perg_A; Lig_quest' [ p0 , p1]
          []    p1; Lig_quest'_1 [perg_A, p1 ]
          endproc
     endproc
endproc
```

O processo `Lig_resp` liga a entrada de uma resposta correta, na interface do serviço de comunicação subjacente `Sub_Com` (na porta `r0` ou `r1`) com a entrega dessa resposta na interface do usuário `Questionador` (porta `resp_A`):

```
process Lig_resp [resp_A,r0,r1,t0,t1,e0,e1,p0,p1] noexit  :=
     Lig_resp_0 [resp_A, r0, t0, e0]
 >>  Lig_resp_1 [resp_A, r1, t1, e1]
where process Lig_resp_0 [resp_A, r0, t0, e0 ] : noexit :=
          r0; resp_A; exit
     []   e0; Lig_resp_0 [resp_A, r0, t0, e0]
     []   t0; Lig_resp_0 [resp_A, r0, t0, e0]
     endproc
     process Lig_resp_1 [resp_A, r1, t1, e1 ] : noexit :=
          r1;resp_A;Lig_resp [resp_A, r0, r1, t0, t1, e0, e1]
     []   e1; Lig_resp_1 [resp_A, r1, t1, e1]
     []   t1; Lig_resp_1 [resp_A, r1, t1, e1]
     endproc
endproc
```

Pode-se representar graficamente a interação das restrições que estão presentes na entidade de protocolo `Ent_A`. Veja a Figura 4.12.

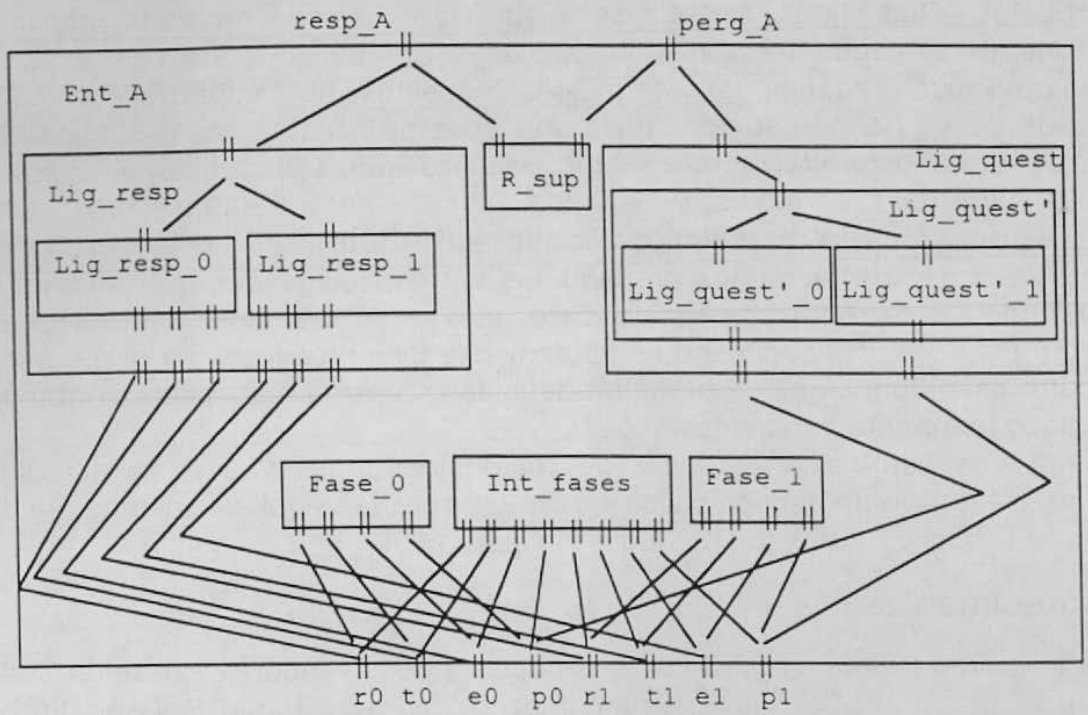


Figura 4.12 - Entidade de protocolo Ent_A

### 4.3.2. A entidade de protocolo Ent_B

A entidade de protocolo Ent_B, localizada no sítio do usuário Respondedor, tem um comportamento mais simples do que sua entidade par, Ent_A, localizada no sítio do usuário Questionador. Tal simplicidade se deve a dois fatores principais: (a) ausência de um temporizador e (b) emissão de resposta somente se recebe uma questão válida (correta e com o bit esperado).

Considerando que o objetivo do exemplo é apenas o de ilustrar uma abordagem de projeto onde são usados construções predefinidas, e que tal ilustração é apresentada com as especificações realizadas, o comportamento da entidade de protocolo Ent_B deixa de ser especificado. Contudo, fica claro que a estruturação em termos de restrições básicas, adotada para o caso da entidade Ent_A, também pode ser adotada para o caso da entidade Ent_B.

## 5. Conclusões

Neste trabalho busca-se dar um tratamento sistemático à utilização de construções predefinidas em LOTOS. Cinco tipos de construções predefinidas

são propostas. Dois tipos referem-se a LOTOS Básico: recursos básicos predefinidos e restrições básicas predefinidas. Um deles refere-se a tipos abstratos de dados predefinidos. A reunião de aspectos de comportamento com aspectos de dados permite sugerir mais dois tipos de construções predefinidas: recursos estruturados predefinidos e restrições estruturadas predefinidas.

Um exemplo simples, baseado em um sistema do tipo `Pergunta-Resposta` é apresentado como ilustração. Com esse exemplo, apresenta-se o uso de restrições predefinidas em especificações que adotam o estilo orientado para restrições (neste caso, têm-se as restrições predefinidas) assim como em especificações que adotam o estilo orientado para recursos (neste outro caso, têm-se os recursos predefinidos no estilo monolítico, orientado para estados ou orientado para restrições).

Uma vez suprida a carência de metodologias efetivas e ferramentas adequadas de auxílio ao especificador [LoCh 92], será possível a transferência da tecnologia LOTOS para o ambiente industrial [Leon 90].

## 6. Agradecimentos

Os autores desejam agradecer aos solícitos pesquisadores Rosvelter João Coelho da Costa (UFSC), Vitório Bruno Mazzola (UFSC), Paulo Roberto Freire Cunha (UFPE), Luís Ferreira Pires (U. Twente - Holanda), Ana Maria Diniz Moreira (U. Stirling - Escócia), Paul Gibson (U. Stirling - Escócia) e Rosa Maria Leão Rust Carmo (LAAS - França).

## 7. Bibliografia

**[BoGo 86]** Bochmann, G. v.; Gotzhein, R. *"Deriving protocol specifications from service specifications"* in: Communications, Architectures & Protocols, Proceedings of the ACM SIGCOMM'86 Symposium, Vermont, USA, 1986.

**[BoBr 87]** Bolognesi, T.; Brinksma, E.;*"Introduction to the OSI specification language LOTOS"*. Computer Networks and ISDN Systems, V.14, pp. 25-29, 1987.

**[Brin 88]** Brinksma, E.: *"A tutorial on LOTOS"*, ISO8807 Information Processing Systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour". 1988.

**[ClJo 92]** Clark, R. G.; Jones, V. M.; *"Use of LOTOS in the formal development of an OSI protocol"*, Computer Communications, V.15, N.2, pp. 86-92, março 1992.

**[DrCh 92]** Drayton, L.; Chetwynd, A.; Blair, G.: *"Introduction to LOTOS through a worked example"*, Computer Communications, V. 15, N. 2, pp. 70-85, março 1992.

**[EhMa 85]** Ehrig, H.; Mahr, B.: "Fundamentals of algebraic specification 1, Springer-Verlag, Berlin, 1985.

**[ErHo 92]** Ernberg, P; Hovander, T.; Monfort, F.: *"Specification and implementation of an ISDN telephone system using LOTOS"*. In: Participant's Proceedings of the Fifth International Conference on Formal Description Techniques, FORTE'92, pp. 179-194, Lannion, França, outubro 1992.

**[ISO 3067]** ISO / IEC JTC1/SC21 N3067; *"Specification styles for structuring of OSI formal descriptions"*, Information Retrieval, Transfer and Management fot OSI, 19 de setembro, 1988.

**[ISO 8807]** ISO - International Standard. *Information processing systems - Open systems interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour*, 1989.

**[KoBo 91]** Kouyzer, A. J; Boogaart, A. K. van den: *"The LOTOS framework for OSI Systems Management"*. Integrated Network Management II, Elsevier Science Publishers B. V. , North-Holland, 1991.

**[Lang 90]** Langerak, R.: *"Decomposition of functionality: a correctness preserving LOTOS transformation"* ; In: Participant's Proceedings of the 10th International Symposium on Protocol Specification, Testing, and Verification, Ottawa, pp. 203-218, IFIP, 1990.

**[Leon 90]** Leon, G.: *"On the technology transfer of formal methods: an experience on LOTOS"*. Third International Conference on: Formal Description Techniques, FORTE'90, pp. 567-582, Madrid, 5-8 november, 1990.

**[LoCh 92]** Loureiro, A. A. F; Chanson, S. T.; Vuong, S. T.: *"FDT tools for protocol development"*. Tutorials. In: Participant's Procedings of the Fifth International Conference on Formal Description Techniques, FORTE'92, Lannion, França, pp. 38-78,1992.

**[LoMe 90]** Logrippo, L; Melanchuk, T.; Wors, R.: *"The algebraic specification language LOTOS: an industrial experience"*, In: FORTE'90, pp. 59-66, 1990.

**[MoCl 93]** Moreira, A. M. D.; Clark, R. G.: *"Os métodos formais na análise de orientação por objectos"*. In: Anais do VIII Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, 26-29 outubro 1993, pp. 238-252. 1993.

**[Pech 92]** Pecheur, C. *"Using LOTOS for specifying the CHORUS distributed operating system kernel"*, In: Computer Communications, vol 15, n.2, pp. 93-102, março de 1992.

**[PiCu 90]** Pinheiro, T. S. de M.; Cunha, P. R. F.: *"Modelo de referência para especificação formal de sistemas operacionais distribuídos"*. In: IV Simpósio Brasileiro de Engenharia de Software, pp. 14-29, Águas de São Paulo, SP, 1990.

**[PiLo 90]** Pires, L. F.; Lopes de Souza, W.: *"Step-wise refinements design using LOTOS"*. In: Proceedings of the 3rd International Conference on Formal Description Techniques (FORTE'90), Madrid, 1990.

**[PiSi 92]** Pires, L. F.; Sinderen, M. van; Vissers, C.: *"On the use of pre-defined implementation constructs in distributed systems design"*. Memoranda Informatica 92-08, University of Twente, 1992.

**[Sanz 92]** Sanz, J. I.: *"Using FDTs in the development of a PICS editor"*. In: Participant's Proceedings of the Fifth International Conference on Formal Description Techniques, FORTE'92, pp. 405-419, Lannion, França, outubro 1992.

**[ViSc 88]** Vissers, C. A.; Scollo, G.; Sinderen, M. van: *"Architecture and specification style in formal descriptions of distributed systems"*. In: Proceedings of the VIII International Conference on Protocol Specification, Testing and Verification. IFIP, p. 189-204, 1988.

**[ViSc 89]** Vissers, C. A.; Scollo, G.; Sinderen, M. van.; Brinksma, E.: *" On the use of specification styles in the design of distributed systems"*. In: Proceedings of TAPSOFT'89. Barcelona, 1989.