

5411-9

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ESTUDO COMPARATIVO DAS LINGUAGENS  
ESTELLE E LOTOS  
NA ESPECIFICAÇÃO DE PROTOCOLOS

por

Carlos Augusto Prolo

Dissertação submetida como requisito parcial  
para a obtenção do grau de Mestre em  
Ciência da Computação

*- PAULO DALCARI*

Prof. Maurizio Tazza  
Orientador

Prof. Juergen Rochol  
Co-orientador

Porto Alegre, novembro de 1989.

UFRGS  
INSTITUTO DE INFORMÁTICA  
BIBLIOTECA



UFRGS

SABi



05221291

**Prolo, Carlos Augusto**

**Estudo comparativo das linguagens Estelle e Lotos na especificação de protocolos. Porto Alegre, CPGCC da UFRGS, 1990.**

**iv. (mestr. ci. comp) UFRGS-CPGCC**

**Diss. (mestr. ci. comp) UFRGS-CPGCC, Porto Alegre, BR-RS, 1990.**

**Dissertação: Redes: Comunicação: Dados**

**Dissertação: Redes: Comunicação: Dados:  
Protocolos: Especificação formal**

Aos leitores que gostarem desta  
dissertação de mestrado e  
encontrarem nela subsidio para  
seus estudos.



## AGRADECIMENTOS

Não vou enumerar nomes aqui pois a sequência e mesmo o conjunto dos nomes iriam ser provavelmente diferentes se definidos no mês passado ou no mês que vem. Os agradecimentos não refletiriam a importância real das pessoas citadas ou omitidas durante o período de mestrado, mas sim, o modo como elas socorrem o meu atual estado de espírito.

Agradeço, por exemplo, aqueles que aceitaram com naturalidade a possibilidade de que esta tese não fosse concluída, aqueles que até sugeriram alternativas. Agradeço aqueles que não disseram que eu estava perdendo quatro anos de minha vida (??). Agradeço a um Baiano (desculpem, citei) que numa destas longas madrugadas do PGCC, tomou a iniciativa de correr ao aeroporto de moto para que eu pudesse levar um artigo, a tempo de ser enviado para análise pela SBC. E a outros amigos, que agem assim, com total desprendimento, e a quem sempre se pode recorrer. Deixa eu ver ... Ah, sim! Aquelles que não me cobraram multa na biblioteca e também aquele que um dia cobrou uma fortuna (não pelas multas é obvio). Aquelle pessoal da secretaria, biblioteca, etc., que não se enquadra na visão comum do "servidor público", pois além de trabalharem, são extremamente camaradas, suportam os chatos (quem? eu?) e até quebram (e consertam) um monte de galhos para Eles. A grande vantagem desta página é que a gente não precisa pedir desculpas pra terminar, mesmo ela ficando incompleta. A outra vantagem é que ninguém tem direito de corrigir os erros de português ou o caráter mais ou menos formal do texto. Obrigado!



## SUMARIO

LISTA DE FIGURAS .....	11
RESUMO .....	13
ABSTRACT .....	15
1 INTRODUÇÃO .....	17
2 ESPECIFICAÇÃO FORMAL DE PROTOCOLOS .....	21
2.1 <u>Conceitos fundamentais</u> .....	21
2.1.1 Especificação .....	21
2.1.2 Interpretações de uma especificação .....	22
2.1.3 Ambiguidade .....	23
2.1.4 Especificação formal .....	23
2.1.5 Implementação .....	25
2.1.6 Níveis de especificação e níveis de abstração .....	25
2.1.7 Objetivos do uso de FDTs .....	27
2.1.8 Correção de especificações .....	27
2.1.9 Correção de implementações e implementação automática .....	29
2.1.10 Concisão .....	29
2.1.11 Precisão .....	30
2.1.12 Excesso de especificação .....	30
2.1.13 Características desejáveis em uma FDT ...	30
2.2 <u>Modelo ISO de interconexão de sistemas abertos</u> .	31
2.3 <u>Validação e verificação de protocolos</u> .....	34
2.3.1 Validação .....	34
2.3.2 Verificação .....	34
2.3.3 Verificação X validação .....	35
2.3.4 Verificação aplicada a interconexão de sistemas de processamento de informação .	35
2.3.5 Consistência .....	37
2.3.6 Completeza .....	38
2.3.7 Correção .....	38

2.4	<u>Da existência de uma linguagem ideal para a especificação de protocolos</u> .....	39
2.4.1	Objetivo do trabalho de especificação ...	39
2.4.2	Ambiente onde a especificação será considerada .....	41
2.4.3	Adequação ao usuário .....	42
2.4.4	Adequação às características peculiares a cada nível de protocolo .....	42
2.4.5	Conclusão .....	43
2.5	<u>Uso de linguagens de programação para a especificação formal</u> .....	44
2.6	<u>Panorama das linguagens de especificação formal de protocolos</u> .....	45
3	<b>DESCRIÇÃO DAS LINGUAGENS</b> .....	55
3.1	<u>Estelle</u> .....	56
3.1.1	Modelo .....	56
3.1.2	Definição da sintaxe e semântica .....	68
3.2	<u>Lotos</u> .....	83
3.2.1	Modelo .....	83
3.2.1.1	Modelo dos tipos de dados ....	83
3.2.1.2	Modelo dos processos .....	84
3.2.2	Definição da sintaxe e semântica .....	84
4	<b>ESTUDO COMPARATIVO DAS LINGUAGENS</b> .....	111
4.1	<u>Definição dos parâmetros</u> .....	111
4.1.1	Concorrência .....	111
4.1.2	Comunicação .....	112
4.1.3	Sincronização .....	112
4.1.4	Não determinismo .....	113
4.1.5	Imparcialidade ("fairness") .....	113
4.1.6	Tempo .....	113
4.1.7	Especificação de dados .....	114
4.1.8	Nível de especificação .....	114
4.1.9	Nível de abstração .....	115



4.1.10	Implementação automática .....	115
4.1.11	Concisão .....	115
4.1.12	Consistência .....	115
4.1.13	Completeza .....	116
4.1.14	Adequação ao modelo OSI .....	116
4.1.15	Formalismo .....	116
4.1.16	Verificação .....	117
4.1.17	Adequação a objetivos .....	117
4.1.18	Adequação ao usuário .....	117
4.1.19	Adequação aos níveis de protocolo .....	118
4.2	<u>Comparação</u> .....	118
4.2.1	Concorrência .....	119
4.2.2	Comunicação .....	131
4.2.3	Sincronização .....	140
4.2.4	Não determinismo .....	143
4.2.5	Imparcialidade ("fairness") .....	150
4.2.6	Tempo .....	150
4.2.7	Especificação de dados .....	152
4.2.8	Nível de especificação .....	157
4.2.9	Nível de abstração .....	158
4.2.10	Implementação automática .....	160
4.2.11	Concisão .....	160
4.2.12	Consistência .....	164
4.2.13	Completeza .....	164
4.2.14	Adequação ao modelo OSI .....	165
4.2.15	Formalismo .....	166
4.2.16	Verificação .....	167
4.2.17	Adequação a objetivos .....	170
4.2.18	Adequação a usuários .....	170
4.2.19	Adequação aos níveis de protocolo .....	172
5	CONCLUSÃO .....	173
6	BIBLIOGRAFIA .....	177



## LISTA DE FIGURAS

Figura 3.1	Tipos de ligações possíveis entre pontos de interação .....	58
Figura 3.2	Processo de criação de módulos .....	66
Figura 3.3	Sequência de eventos na criação de módulos	67
Figura 3.4	Exemplo de definição da máquina de estados	77
Figura 3.5	Diagrama de transições geradas .....	77
Figura 3.6	Exemplo de definição da assinatura de um tipo: sorts e operações .....	87
Figura 3.7	Definição das equações do tipo .....	90
Figura 3.8	Exemplos de utilização do operador "[]" ...	101



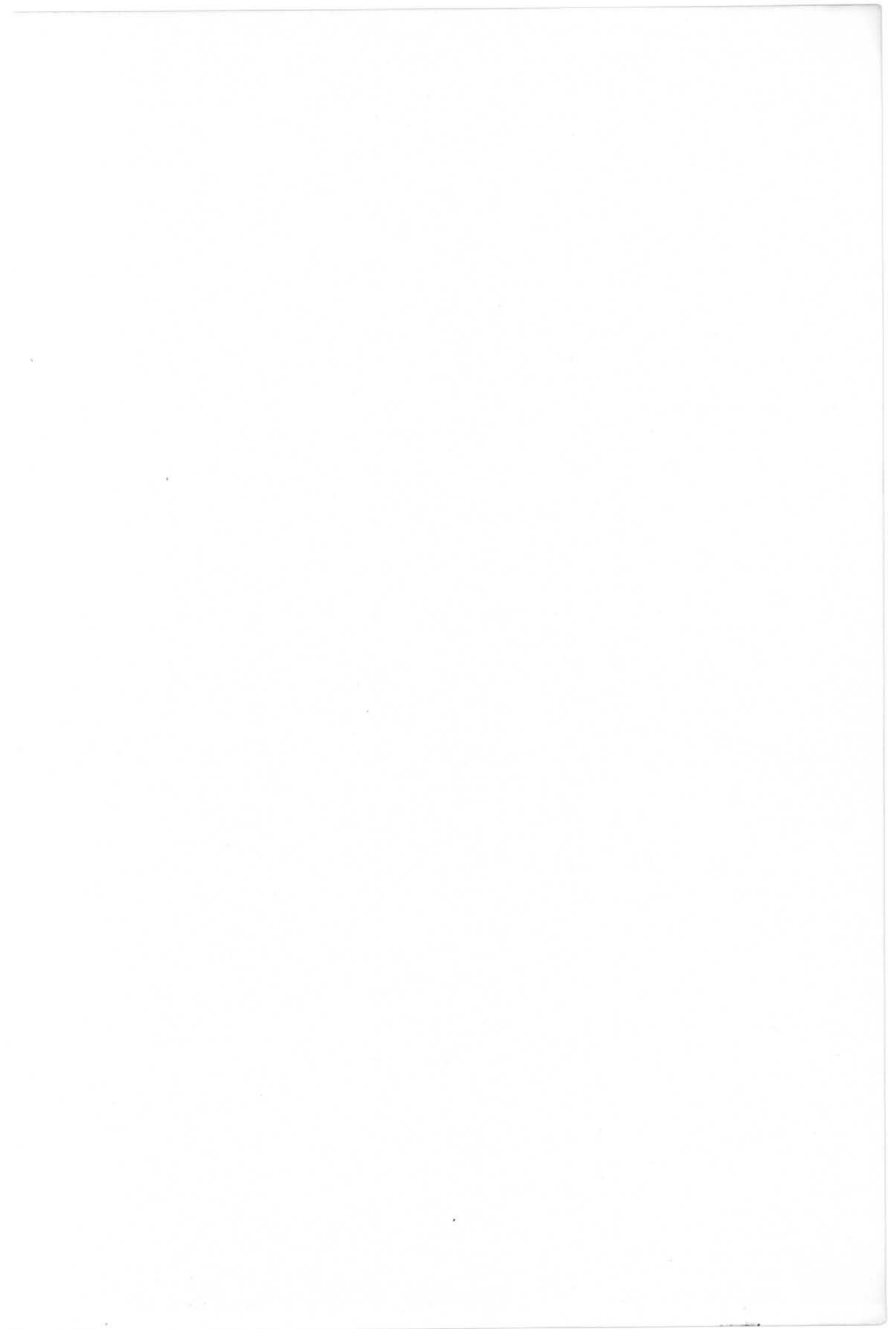
**RESUMO**

Desenvolve-se um trabalho na área de especificação formal de protocolos de comunicação de dados. É feita uma apresentação rigorosa dos conceitos e terminologia associados ao tema. É proposto um conjunto de critérios para comparação de linguagens formais para especificação de protocolos. Estes critérios são aplicados para a comparação entre duas destas linguagens, Estelle e Lotos, atuais e bastante difundidas, em fase final de padronização pela ISO.



**ABSTRACT**

This work stands on the area of formal protocol specification. It is given a rigorous presentation of the concepts and terminology related to the area. A set of criteria is proposed for the comparison of formal specification languages for protocols. The criteria are applied in the comparison of two of these languages, Estelle and Lotos, modern and well accepted in the community, whose standardization by ISO is in a final step.





## 1 INTRODUÇÃO

Dentro do contexto de interconexão de sistemas de processamento de informação, aborda-se o tópico de especificação formal, com ênfase na especificação de protocolos de comunicação de dados.

Os objetivos do trabalho são: definição de conceitos básicos concernentes ao tema de especificação formal, em especial de protocolos e a comparação de linguagens utilizadas para este fim. São definidos parâmetros para a comparação, orientados para a especificação de protocolos de comunicação. Os protocolos são aplicados a duas linguagens bem conhecidas na literatura, Estelle e Lotos.

Com o enorme crescimento das necessidades de interconexão entre sistemas computacionais, surgiu a necessidade de desenvolvimento de protocolos complexos que fossem padronizados para evitar a necessidade de um emaranhado de protocolos específicos para a comunicação entre cada grupo de sistemas similares. Os métodos informais utilizados para especificação, que em geral utilizam linguagens naturais, são geradores potenciais de especificações ambíguas, e tiveram esta característica acentuada devido à complexidade dos protocolos e à heterogeneidade dos grupos de pessoas que deveriam ter acesso aos padrões. Tornou-se necessária a definição de linguagens formais para a especificação.

A utilidade da especificação formal não está apenas na geração de padrões nacionais ou internacionais, mas em toda atividade em que for constatada ambiguidade na descrição de protocolos e esta for considerada intolerável. Isto foi demonstrado através de uma interação com empresa nacional, durante a qual foi especificado, na linguagem

Estelle, o comportamento de uma entidade que opera o protocolo BSC3 em uma estação escrava IBM 3274 de um subsistema de teleprocessamento.

A literatura a respeito do assunto é bastante vasta a partir do final da década de 70. Foram criados novos conceitos e outros adaptados de outras áreas, surgindo uma terminologia na área de especificação de protocolos que via de regra não tem sido utilizada com rigor. Quando a ISO (International Standards Organization) começou a se preocupar com o assunto, um conjunto de novos e antigos termos começou a tornar-se popular, porém com uso dependente da interpretação de cada um. O resultado é que termos como precisão, concisão, técnica formal, verificação, correção e tantos outros aparecem citados nos artigos sem dar ao leitor uma noção rigorosa do significado do texto. Isto motivou o primeiro objetivo, para o qual foi consultada a literatura concernente à especificação (formal) de protocolos, lógica matemática, semântica formal, especificação formal de programas, modelos, teoria da computação e matemática aplicada à computação.

No desenrolar do processo de desenvolvimento das linguagens formais de especificação, surgiram diversas técnicas, publicadas em artigos, com exemplos de utilização, sugestão e descrição de ferramentas de apoio. É difícil, contudo, encontrar auxílio para usuários que precisem escolher uma dentre as técnicas que melhor se adapte aos seus objetivos. Isto motivou o segundo objetivo, de definição de parâmetros para comparação e aplicação a um par de linguagens.

No capítulo 2 encontram-se definições básicas coletadas ou inferidas a partir da literatura existente e a resposta para diversas questões que surgem no contato com a área. É apresentado também um panorama mostrando a evolução

das principais correntes de linguagens, justificando-se ao final a escolha das linguagens apresentadas no capítulo seguinte.

No capítulo 3 descreve-se as linguagens Estelle e Lotos, representativas da tendência atual em especificação de protocolos. Para cada uma é feita uma análise individual enfocando as facilidades para a sua utilização.

No capítulo 4 descreve-se os parâmetros de comparação das linguagens, com vistas à especificação de protocolos de comunicação, e o resultado de sua aplicação às linguagens do capítulo 3.

No capítulo 5, conclusão, aborda-se possíveis rumos no desenvolvimento do trabalho.



## 2 ESPECIFICAÇÃO FORMAL DE PROTOCOLOS

### 2.1 Conceitos fundamentais

#### 2.1.1 Especificação:

**Especificação** de um sistema é a descrição, em um formalismo conhecido, das observações que dele podem ser feitas [ISO 85a] segundo uma **visão** restrita resultante de um **processo de abstração** sobre este sistema.

Parte-se de um **domínio de aplicação** [TUR 87] formado por um conjunto de objetos possivelmente hierarquizados junto com um conjunto de relações entre estes objetos. Um domínio de aplicação pode ser um banco, uma planta industrial, o sistema solar, etc. Nosso interesse reside em ambientes de interconexão de sistemas de processamento de informação.

A partir do domínio de aplicação é feito um **processo de abstração**, que consiste em determinar todos os elementos (objetos e relações) que são importantes para descrever o domínio parcialmente, de acordo com algum objetivo previamente estabelecido. Se o objetivo do estudo do domínio bancário é a segurança, vão entrar objetos do tipo guarda, cabines blindadas e procedimentos ante situações de perigo, mas se o objetivo é agilizar o atendimento, os elementos serão outros, a saber, a distribuição estimada de afluxo de clientes, número de funcionários, tempo médio de atendimento. O processo de abstração define uma **visão** do domínio de aplicação. A especificação é uma descrição do resultado do processo de abstração.

Em um ambiente de interconexão de sistemas de processamento de informação, onde o objetivo é exatamente a

facilidade de interconexão de modo geral (e não o funcionamento e as características de cada sistema como um todo) é de interesse que a abstração inclua entre outras coisas:

- . mecanismos de comunicação (sequências de mensagens possíveis, estabelecimento da comunicação);

- . mecanismos de segurança e confiabilidade (como garantir ou aumentar o grau de confiança de que uma mensagem chegará a seu destino, de que não chegará a outros sistemas, de que não haverá alteração da informação);

- . mecanismos que assegurem a correta interpretação de dados transmitidos (efetiva transferência de informação).

Não é de interesse a inclusão das características do sistema operacional dos sistemas envolvidos, o número de processos utilizados para operar a comunicação, a gerência das filas de recebimento de mensagens.

### 2.1.2 Interpretações de uma especificação

Uma especificação pode ser vista como uma restrição imposta ao universo de sistemas (domínios de objetos e suas relações) reais ou imaginários. A especificação, deste modo, define um subconjunto destes sistemas que a satisfazem: que atendem à restrição imposta. A cada uma destas estruturas de domínios de objetos e relações que satisfazem a especificação chamar-se-á uma interpretação da especificação. Uma especificação é portanto o conjunto de interpretações que satisfazem suas restrições. Entre as várias interpretações de "banco" estão as contidas em "banco de investimento", "banco comercial", "banco em que existe apenas uma fila para todos os caixas e no qual o gerente manda abrir uma nova caixa cada vez que o tamanho da fila ultrapassar 10 pessoas".

### 2.1.3 Ambiguidade

Uma especificação é ambigua, quando ela dá margem a conjunto(s) de interpretações diferente(s) do conjunto desejado pelo agente que gerou a especificação. Isto pode ocorrer em qualquer nível de abstração e em geral é debitado às características da linguagem utilizada na especificação.

### 2.1.4 Especificação formal

Uma característica das linguagens naturais é a de permitir a geração de descrições ambíguas. Uma especificação informal, com componentes de linguagens naturais, herda a característica de permissão de ambiguidade que é indesejável e mesmo inadmissível em muitos casos, como por exemplo na divulgação de padrões internacionais de protocolos de comunicação.

A definição de especificação formal tem vários níveis de tolerância. O primeiro é que uma especificação formal é aquela expressa através de uma linguagem que não permite descrições ambíguas, chamada linguagem formal de especificação. Considere-se a seguinte sentença da Língua Portuguesa: "As águas, junto com seus habitantes, maravilhas da natureza, foram criadas por Deus". Existe uma ambiguidade concernente à expressão "maravilhas da natureza" que pode ser um qualificador para os habitantes das águas, para as águas, ou para ambos: águas e seus habitantes. Esta ambiguidade não é uma questão de interpretações diferentes para diferentes pessoas. Ela é inerente à linguagem, que neste caso admite múltiplas interpretações da sentença para qualquer bom conhecedor da língua. A Língua Portuguesa não é, portanto, uma linguagem formal de especificação.

Na definição do parágrafo anterior, não é considerado o modo como a linguagem formal é assimilada. Mesmo que uma linguagem, para algum conhecedor, não dê margem a ambiguidades, a ambiguidade pode surgir pela diferença de interpretação da linguagem, que frequentemente é bastante complexa, segundo dois conhecedores diferentes. Isto conduz a uma outra definição, mais rigorosa, a qual exige que uma linguagem formal de especificação tenha sintaxe e semântica formalmente definidas. A rigor, poderíamos ter infinitos níveis de definições, uma vez que toda a sintaxe de toda linguagem é formalmente definida através de uma metalinguagem (que precisa por sua vez ser definida) e toda a semântica é formalmente definida pelo mapeamento em um modelo (matemático, lógico ou outro qualquer) que também precisa ser definido. É razoável, atualmente, aceitar como linguagem formal de especificação aquela que tem sintaxe e semântica formal ou informalmente bem definidas. Uma especificação formal é aquela descrita por meio de uma linguagem formal de especificação.

Quando a semântica da linguagem é definida informalmente, a ausência de ambiguidade em suas sentenças, isto é, a certeza de que a cada sentença da linguagem é atribuída apenas uma regra de interpretação, não pode ser provada formalmente. Se a semântica da linguagem é formalmente definida pode-se, em alguns casos, provar a ausência de ambiguidade. No caso geral, no entanto, a ausência de ambiguidade inerente à linguagem não pode ser provada, pois as linguagens de programação ou de especificação não pertencem, em geral, à classe das linguagens regulares, integrando a classe livre-de-contexto ou uma classe mais restrita, e é indecidível o problema de determinar se uma gramática livre de contexto qualquer é ambigua ou não, ou mesmo se existe uma gramática não-ambigua que a represente [HOP 79]. Isto significa que ao atribuirmos



formalmente significado às construções da linguagem, poderá haver alguma para a qual seja atribuído significado duas vezes e de forma diferente.

São considerados sinônimos da expressão "linguagem formal de especificação", neste texto, as expressões "técnica formal de descrição" abreviada "FDT" (em inglês "formal description technique"), "técnica formal de especificação", "linguagem formal de descrição", "método formal de especificação ou descrição". Nem todos os autores concordam com a igualdade do significado de especificação e descrição nas expressões acima [CCI 85, SAR 87].

#### 2.1.5 Implementação

Em um ambiente computacional, de posse de uma especificação, pode-se, mediante um processo de satisfação [TUR 87], criar um sistema particular em que se possa visualizar os objetos e relações definidas na especificação. Este sistema é uma implementação da especificação. Diz-se também que uma (pretensa) implementação está correta se ela satisfaz à especificação que lhe deu origem.

#### 2.1.6 Níveis de especificação e níveis de abstração

Há duas formas de encarar o grau de detalhamento de especificações. Em uma delas diz-se que uma especificação é mais detalhada do que a outra quando esta (a primeira) contém informações que restringem o número de sistemas reais cujo comportamento observável se adapta à especificação. Por exemplo: Uma especificação A1 pode definir que o intervalo de tempo entre a ocorrência dos eventos e1 e e2 deverá ser de no máximo t segundos. A especificação A2 pode definir que o intervalo vai ser sempre menor do que  $t/2$  segundos. Diz-se, então, que A2 detalha A1 em aspectos observáveis do

sistema. Será usado o termo **nível de especificação** [ISO 85a] para referir-se ao grau de detalhe sob este ângulo. Assim, uma especificação é dita **ideal** [ISO 85a] quando ela descreve todas as alternativas de interpretação consideradas aceitáveis para o sistema. Isto corresponde a um processo de abstração com o menor grau de restrição possível. O outro extremo é a **especificação orientada à implementação**, que descreve apenas uma dentre as interpretações corretas.

Os diversos níveis de especificação de um sistema podem ser vistos como uma árvore em que a raiz é a especificação ideal, as folhas são as especificações de implementações e os nodos internos representam níveis intermediários de abstração. A especificação dos eventos observáveis de um sistema como um grafo que mostre a dependência real entre as ocorrências de eventos seria um ancestral da especificação dada por uma sequência (uma das sequências possíveis) fixa de eventos.

Um outro tipo de detalhamento é aquele que provê informação adicional sobre como construir um sistema com determinadas características observáveis. Para referir-se a este tipo de detalhamento será usado o termo **nível de abstração**. A especificação de um programa que calcula a raiz quadrada de um número poderia ser, em alto nível de abstração,

$$\text{valor-de-entrada} = (\text{valor-de-saida})^2$$

enquanto um programa em Pascal que imprimisse a raiz de um valor lido seria uma possível especificação de implementação, com nível de abstração bem mais baixo, associada à máquina que executará o programa. Note-se que neste caso o nível de especificação é o mesmo. A especificação de um algoritmo como um grafo que mostre o fluxo de dados entre as operações, mostrando a dependência

real entre as várias operações, seria um ancestral (em termos de níveis de abstração) da especificação dada por uma sequência (uma das sequências possíveis) fixa de operações.

Quando quiser-se referir aos dois sentidos conjuntamente, será usado o termo nível de detalhamento. A literatura usa, frequentemente, o termo nível de abstração com os dois sentidos, em alguns casos ressaltando a duplicidade [ISO 87]. [TUR 87] chama a atenção para a imprecisão dos conceitos dos parágrafos anteriores. Quando é que uma especificação descreve todas as interpretações, ou, equivalentemente, contém apenas informação essencial (o que é essencial?) ? [TUR 87] descreve como deslocamento da especificação ("specification bias") casos em que a especificação é tão próxima à implementação (linguagens de programação, por exemplo) que ela pode ser tomada como uma implementação.

#### 2.1.7 Objetivos do uso de FDTs

O objetivo principal do uso de FDTs é exatamente a geração de especificações não-ambíguas [CCI 85, ISO 85b, ISO 87] (pela definição, toda especificação feita através de uma FDT é não-ambígua).

Outros objetivos importantes são [VIS 83]:

- a) aumentar a capacidade de determinar a correção de especificações e implementações (itens 2.1.8 e 2.1.9);
- b) abrir a possibilidade de automação da geração de implementações a partir da especificação (item 2.1.9).
- c) aumentar capacidade de predição de desempenho.

#### 2.1.8 Correção de especificações

Para poder-se afirmar que algo está correto é

preciso, em primeiro lugar, que se tenha um conjunto de critérios que sirva de base para a determinação da correção. A afirmação de correção equivale a dizer que "algo" satisfaz completamente o conjunto de critérios. Em segundo lugar é necessário que tanto o elemento sobre o qual se quer proclamar correção como o conjunto de critérios que servem como base de correção estejam formalmente definidos. Em terceiro lugar é preciso um método formal que permita, em espaço de tempo finito, mapear conceitos do formalismo usado na definição do elemento, no formalismo utilizado para definir os critérios de correção.

O parágrafo acima permite afirmar que não se pode provar a correção de uma especificação contra um domínio de aplicação que não é formalmente definido, como é o caso do ambiente de interconexão de sistemas de processamento de informação. Pode-se no máximo mostrar a ausência de correção, através de um contra-exemplo.

O uso de FDTs permite que uma especificação possa ser proclamada (formalmente) correta contra uma especificação com um nível de detalhamento menor (uma especificação ancestral) que lhe serve de critério de correção. A especificação estará correta se o seu conjunto de interpretações possíveis for um subconjunto daquele do ancestral. Diz-se também que a primeira satisfaz a segunda ou que as duas são equivalentes, diferindo no nível de detalhamento. Note-se que ainda resta o problema complexo de encontrar o método formal que permita o cálculo, em muitos casos não computável.

No caso específico da interconexão de sistemas de processamento de informação estabeleceu-se, por influência de órgãos internacionais de padronização, especialmente ISO (International Standard Organization) e CCITT (Comité

Consultativo Internacional para Telefonia e Telegrafia), dois níveis de especificação na descrição dos componentes do ambiente de interconexão chamados serviço e protocolo. Uma questão importante é verificar se um protocolo (especificação mais detalhada) satisfaz o serviço correspondente.

#### 2.1.9 Correção de implementações e implementação automática

Um segundo aspecto da correção é o da determinação de que uma implementação satisfaz uma especificação de protocolo. Isto é possível se existir uma definição formal da especificação, como é o caso de um programa em uma das linguagens usuais de programação de computadores. A dificuldade reside no mapeamento de formalismos.

O fato de uma implementação estar correta é frequentemente definido como conformação ("conformance", "compliance") da implementação com a especificação [ISO 83a, RAY 87]. As tentativas de assegurar (aumentar as probabilidades de) conformação podem não ser formais.

Se for possível determinar formalmente se uma implementação satisfaz uma especificação, então também é possível pensar em implementação automática de especificações de protocolos através de compiladores, por exemplo. Existem compiladores semi-automáticos a partir de algumas linguagens de especificação com características orientadas a linguagens de programação, uma delas, Estelle [VUO 88, SOU 87b], analisada nesta dissertação.

#### 2.1.10 Concisão

Concisão (conciseness), em uma especificação, expressa a ausência de detalhes excessivos em um nível de especificação onde o detalhamento não é importante e até

prejudica a clareza [ISO 85b]. Concisão, aplicada a uma linguagem de especificação [ISO 83a] expressa a ausência de excessivos detalhes nas construções da linguagem.

#### 2.1.11 Precisão

Precisão garante, complementarmente à concisão, que a especificação não tenha possíveis interpretações além daquelas que forem consideradas aceitáveis em um determinado nível de especificação, isto é, que a especificação não seja mais abstrata que o desejado, omitindo detalhes importantes.

#### 2.1.12 Excesso de especificação

Chama-se excesso de especificação (*overspecification*) o fato de uma especificação limitar o comportamento de um sistema (eliminando padrões de comportamento possíveis) em um nível de especificação em que o detalhamento não é necessário. É o oposto da concisão.

#### 2.1.13 Características desejáveis em uma FDT

- a) Facilidade de aprendizado [CCI 85].
- b) Facilidade de interpretação das especificações [CCI 85]. Clareza das especificações geradas [ISO 85b].
- c) Alto nível de abstração: independência de métodos de implementação (geração de especificações suficientemente abstratas para não induzir a escolha ou exclusão de algum método). Utilizando-se da noção intuitiva, é comum afirmar-se [JON 80] que uma especificação não deve ser orientada à implementação, ou que ela deve manter-se em um nível de abstração que sugira (novamente o conceito é intuitivo) "o que fazer" e não "como fazer".
- d) Concisão [ISO 87].

e) Capacidade de geração de especificações concisas e precisas [ISO 87]. Considerado um dado nível de especificação (cuja definição é claramente subjetiva) quer-se uma linguagem que permita especificar elementos exatamente naquele nível sem detalhes a mais ou a menos (o que é igualmente subjetivo).

## 2.2 Modelo ISO de interconexão de sistemas abertos

A ISO (International Standards Organization) sentindo a necessidade de aumentar a facilidade de interconexão de sistemas gerou um documento [ISO 84a] descrevendo um modelo de arquitetura de sistemas de processamento de informações quanto ao aspecto de interconexão destes sistemas. Este modelo, conhecido como **Modelo de Referência para Interconexão de Sistemas Abertos** (doravante abreviado por RM-OSI do original "Reference Model for Open Systems Interconnection") tem tido tamanha penetração na comunidade científica que passou a ser uma norma de fato. A seguir serão definidos alguns conceitos do RM OSI [ISO 84a], importantes para o desenvolvimento do trabalho.

**Sistema aberto** é um sistema que obedece os padrões do modelo OSI na sua comunicação com outros sistemas. Dentro do ambiente de interconexão de sistemas abertos (ambiente OSI), considera-se **sistema** um todo autônomo capaz de executar processamento e transferência da informação. O processamento da informação para uma aplicação particular é executado por um **processo de aplicação**.

O conceito de interconexão de sistemas abertos (OSI) atêm-se apenas à interconexão dos sistemas. Isto inclui não só a transferência de informação entre sistemas (transmissão), mas também sua capacidade de cooperar para

executar uma tarefa distribuída.

Em uma aplicação distribuída, os sistemas se comunicam fazendo uso do serviço de interconexão definido pelo RM-OSI. A ISO convencionou dividir as tarefas concernentes a este serviço em 7 níveis ou camadas que são, começando pelo nível 7: Aplicação, Apresentação, Sessão, Transporte, Rede, Enlace e Físico. Nas definições que seguem, o termo "(N)" referencia genericamente um dos sete níveis. Por exemplo, entidade de transporte (ou de nível 4) é uma substituição válida para entidade (N).

Cada camada presta um serviço à camada de nível superior (a camada 7 presta um serviço aos processos da aplicação distribuída). O serviço(N) (serviço da camada(N)) é prestado por entidades (N) através da execução de um protocolo(N), utilizando-se do serviço(N-1). O protocolo define o conjunto de regras através das quais entidades de mesmo nível, em sistemas diferentes, também chamadas entidades pares (peer entities), interagem, na execução da parte que lhes toca no processo de cooperação. Um processo da aplicação distribuída solicita a uma entidade de nível 7, a execução de uma determinada facilidade integrante do serviço(7), por exemplo, a transferência de um arquivo. O protocolo de nível 7 conhecido como FTAM (File Transfer Access Method) define os formatos e sequências de mensagens trocadas pelas entidades dos sistemas envolvidos para que a transferência ocorra. Mas o FTAM não resolve todos os problemas: por exemplo, ele não se envolve com as diferenças na representação de informações entre os sistemas (caracteres, inteiros, reais, etc...). Ao invés, ele solicita a uma entidade (6) que transmita a mensagem (facilidade(6) integrante do serviço(6)) e esta se encarrega da correta "tradução" da informação. As entidades(6) por sua vez, na consecução de suas funções utilizam facilidades(5)



definidas no serviço(5), executadas por entidades(5). E assim por diante. É interessante ainda citar a facilidade do serviço de nível 4, de garantir entrega e recebimento correto de mensagens às entidades de nível 5 (preocupando-se com detecção de erros, confirmação de mensagens, retransmissão quando necessário), a de nível 3, de gerenciar o roteamento de mensagens pela rede, a de nível 2, de gerenciar o fluxo de transmissão de mensagens entre cada par de nodos da rede, e a de nível 1, de efetiva troca de bits pela linha.

Saliente-se que o serviço(N) é prestado a uma entidade(N+1) pelas entidades de nível (N) através do protocolo(N), utilizando o serviço(N-1).

Quanto ao trabalho de descrição da interconexão de sistemas abertos e seus elementos, [ISO 85b] vê três níveis de especificação possíveis:

- . Descrição da arquitetura OSI
- . Descrição de serviços
- . Descrição de protocolos

A descrição dos elementos da arquitetura OSI é introduzida em [ISO 85a] considerando os elementos do modelo (camadas, subsistemas, entidades, saps, etc.) como objetos, interligados por relações do tipo "tem como subsistemas", "tem como entidades", etc., relações estas que podem ser 1:1, 1:n, n:m. Não será dado prosseguimento ao estudo da especificação da arquitetura OSI.

[ISO 85b] também vê como desejável a descrição de interfaces entre camadas e descrição de implementações, que conforme [ISO 85a] consistiriam nos nodos folhas da hierarquia das especificações.

## 2.3 Validação e verificação de protocolos

### 2.3.1 Validação

Durante a execução das várias atividades que se iniciam com a visão do domínio de aplicação (em Engenharia de Software o termo "ciclo de vida" se aplica perfeitamente) há um conjunto de atividades que se preocupam não propriamente com o desenvolvimento de novos estágios, mas da garantia de funcionamento, correção, análise de características e predição de características de estágios futuros. Este conjunto de atividades, cuja função é assegurar (ou aumentar a confiabilidade de) que um sistema "satisfaz as especificações de projeto e opera (espera-se) a gosto do usuário final" é conhecido por **validação** [BOC 80a]. Inclui, entre outras, as tarefas de teste de implementação, estudos de simulação, predição e análise de desempenho, comprovação da existência de certas propriedades nas especificações, como por exemplo que não haja situações de impasse ou ciclos improdutivos e outros aspectos da correção das especificações. Os resultados das tarefas de validação tem em geral caráter probabilístico ou de estimativa.

### 2.3.2 Verificação

**Verificação** é o conjunto de tarefas de validação que tem caráter exato, baseando-se em métodos formais. Para que uma propriedade seja verificada ela deve ser formalmente definida, bem como os objetos sobre os quais ela se aplica e deve haver um procedimento calculável [TUR 87] (computável) que prove a existência da propriedade sobre os objetos.

### 2.3.3 Verificação X validação

O cuidado nas definições de validação e verificação acima teve duas origens: a primeira é que em muitas publicações os conceitos de validação e verificação são considerados equivalentes. [SAJ 85] cita vários trabalhos que usam a palavra verificação da mesma forma que outros usam validação. A definição acima é coerente com a de reconhecidos autores que tentam dar um sentido preciso às mesmas [TUR 87, BOC 80a, BER 82].

A segunda razão é que outro conjunto de autores usa corretamente a definição precisa, ou melhor, não a contradiz, mas o faz de maneira ambígua e redundante como em [MOU 86, página 50] que diz sobre verificação de protocolos: "... determinação de certas características lógicas da especificação do protocolo que indicam se ela tem defeitos ou não (ex.: possibilidade de impasse - 'deadlock')", e mais adiante, "... é usado em alguns trabalhos, por exemplo [BOCH 80, CUNH 83], para indicar também a atividade de constatação de que uma implementação de um protocolo concorda com a sua especificação. Neste sentido, preferimos falar de 'teste' da implementação ...". Ora, as duas citações são parte do mesmo conceito de verificação, considerando que a intenção de Bochmann [BOC 80a] era a de uso de método que previsse "todas as situações possíveis" na determinação da validade da implementação, contrapondo-se à tarefa de teste que [BOC 80a] inclui como validação mas não verificação, as duas podendo aplicar-se à implementação.

### 2.3.4 Verificação aplicada à interconexão de sistemas de processamento de informação

O trabalho na área tem-se concentrado na verificação de protocolos de comunicação de dados cujo objetivo é [MOU 86, BOC 85, BOC 80a] demonstrar que um

conjunto de entidades que executem as funções definidas em uma dada especificação de protocolo de nível (N), utilizando o serviço definido na especificação de serviço de nível (N-1) satisfazem (isto é, detalham) a especificação de serviço do nível (N), isto é

(especificação de protocolo (N))

+

(especificação de serviço (N-1))

satisfaz

(especificação de serviço (N))

Costuma-se dividir em dois grupos as propriedades a serem verificadas no trabalho de verificação de protocolos: propriedades de cunho geral e propriedades de cunho específico.

As propriedades de cunho geral são aquelas que uma especificação de protocolo deve possuir, independentemente do serviço que deve satisfazer. Um exemplo clássico é a ausência de impasse ("deadlock freedom"), isto é, não deve haver a possibilidade de o conjunto de entidades que executam o protocolo entrar em um estado de onde não consiga sair. [MER 79, BOC 80a, BOC 78] entre outros trabalhos definiram, sem um critério específico, um grande número de propriedades. [SAJ 85], mais recentemente, apresenta uma visão geral das propriedades segundo diversos autores, e métodos usados para classificá-las.

As propriedades de cunho específico são aquelas que tem a ver com a função do protocolo específico, por exemplo, "entrega correta de mensagens" para um protocolo de transporte [ISO 84b, ISO 84c], "repetição do envio de mensagens em caso de erro" para um protocolo de enlace [ISO 84a]. Aqui está certamente o grande problema da validação, pois é muito difícil enfrentá-lo com resultados

significativos por métodos formais. Note-se que o segundo caso, por exemplo, envolve uma caracterização formal do meio (adulteração de mensagens).

Há uma vasta quantidade de artigos sugerindo métodos de verificação, exemplificando através de especificações ISO e CCITT. [SAJ 85] também tem um apanhado dos principais métodos publicados (análise de alcançabilidade, prova por asserções, execução simbólica, análise de invariantes, projeção, lógica temporal, verificação algébrica, métodos híbridos, etc...), cruzando-os com os subconjuntos de propriedades que eles permitem verificar e com as diversas técnicas de especificação formal às quais se adaptam.

Há três propriedades desejáveis em especificações formais que são parte do processo de verificação e que merecem ser consideradas em particular: consistência, completeza e correção, inclusive porque alguns conceitos de verificação como o apanhado por [SAJ 85] da engenharia de software os envolvem: verificação (de protocolos) é a demonstração da consistência, completeza e correção da especificação (do protocolo).

#### 2.3.5 Consistência

Uma especificação é consistente [TUR 87] se ela não tiver interpretações que permitam tirar conclusões contraditórias (isto é, a afirmação de uma proposição A e a negação de A). O termo consistente é também usado associado a um conjunto de especificações de elementos relacionados, para expressar o fato de não haver contradição entre elas. Haveria falta de consistência, por exemplo, se em uma especificação de entidade de protocolo fosse permitido o envio de uma sequência de mensagens que não é prevista na

especificação da entidade de protocolo adjacente.

### 2.3.6 Completeza

Uma especificação é completa se para toda afirmação A, que tiver sentido em relação ao objeto especificado for possível decidir ou que A é verdade ou que não A é verdade. Uma definição um pouco mais pragmática, fugindo da lógica, é que uma especificação é completa se ela inclui reações para a ocorrência de todos os eventos externos possíveis em cada momento [BOC 80a]. Note-se que a consideração das sequências de eventos possíveis deve ser consistente com o comportamento presumido das entidades adjacentes.

### 2.3.7 Correção

A característica de correção já foi discutida anteriormente restando dizer que ela é frequentemente citada como correção total e dividida em duas propriedades: correção parcial e terminação ou progresso. A correção parcial é frequentemente determinada pela manipulação lógica de asserções de acordo com o significado das construções do programa, cuidando de garantir ou não que uma especificação está correta, caso ela termine ou caso conclua um determinado ciclo. A propriedade de terminação ou progresso, cuida de garantir que a especificação descreve um processo que efetivamente termina (terminação) ou como é o caso de protocolos de comunicação em que o funcionamento é por período indefinido, garantir que a partir de qualquer ponto da execução o processo fatalmente (eventually) progrida (progresso) na execução da sua função. As duas características juntas garantem que o processo efetivamente "trabalha" e "corretamente".

Para tirar a impressão vaga da definição acima,

imaginemos um programa correto, ao qual acrescentamos um ciclo infinito improdutivo. Alguns métodos tradicionais para correção parcial garantem que após o termino do ciclo a execução do protocolo está no caminho correto (uma vez que o ciclo não produziu alterações no funcionamento). É necessário o teste de progressão para verificar que a partir de um dado ponto a execução não progride.

#### 2.4 Da existência de uma linguagem ideal para a especificação de protocolos

[ISO 85a] afirma que diferentes linguagens tem diferentes graus de propriedade para diferentes tarefas de especificação, bem como diferentes propriedades em si mesmas. Na análise da adequação de uma linguagem para a especificação formal de protocolos parece adequado considerar os aspectos desenvolvidos a seguir.

##### 2.4.1 Objetivo do trabalho de especificação

Genericamente o desenvolvimento de uma especificação formal de protocolo serve para:

a) Documentação e divulgação do protocolo. É o objetivo que mais pesa na justificativa de uso de uma linguagem formal, pois uma característica essencial a uma especificação é a ausência de ambiguidade. É importante que a linguagem permita a geração de especificações com alto grau de abstração, para não limitar seu uso ou induzir uma ou outra forma de implementação (o item 1.5 discute a limitação das linguagens de programação neste aspecto). É também desejável, conforme referido anteriormente, que a linguagem seja concisa e clara. Uma linguagem muito complexa transfere o problema de correto entendimento do protocolo (que se tenta resolver) para o de correto entendimento da

linguagem.

b) Validação (verificação) de especificações em diferentes níveis de detalhe para certificar-se de que são equivalentes. Para a consecução deste objetivo é importante que a linguagem tenha um modelo para o qual haja uma teoria que permita calcular equivalência de especificações. Ainda um problema concernente a este objetivo é o de verificar a equivalência de especificações feitas em linguagens diferentes. A análise de uma linguagem deverá, portanto, levar em conta a compatibilidade com outras linguagens com as quais precise ser integrada.

c) Validação das idéias do protocolo, isto é, assegurar que o protocolo que está sendo definido resolve a contento os problemas que o originaram, tem as características de funcionamento e desempenho desejadas. Para esta finalidade é desejável que a linguagem tenha um modelo sobre o qual se consiga com alguma facilidade mapear manualmente, de acordo com o conhecimento do usuário, as propriedades do protocolo a serem analisadas, em propriedades verificáveis no modelo. Entre os modelos que tem sido usados para este fim estão os de transição de estados, especialmente redes de Petri e máquinas de estados finitas.

d) Apoio a implementações. Aqui há uma dicotomia. Uma especificação com baixo nível de abstração, induz a alternativas de implementação. Aparentemente isto é agradável (na consideração deste sub-item) porque torna mais fácil o mapeamento da especificação à implementação. Porém, na medida em que induz a algumas alternativas se afasta de outras. A linguagem Estelle analisada nesta dissertação é uma extensão do Pascal ISO [ISO 83b] para comportar a definição explícita de máquinas de estado comunicantes. Ora, e se for mais eficiente implementar o protocolo na linguagem



C ? E se for mais apropriado desenvolver um circuito integrado dedicado à execução das funções do protocolo ?

e) Apoio ao teste de implementações. A mesma dicotomia do parágrafo anterior vale aqui, com um agravante. Não se pode ceder à tentação de seguir as alternativas induzidas pela especificação porque a escolha já está definida pela implementação que se quer testar. Genericamente quer-se uma especificação suficientemente distante de um tipo particular de implementação para que ela possa auxiliar o teste de qualquer alternativa de implementação.

#### 2.4.2 Ambiente onde a especificação será considerada

A tarefa de especificação pode ser usada em um ambiente científico, onde se pesquisam novas técnicas de controle de fluxo, de roteamento eficiente de mensagens em uma rede, métodos de controle distribuído de acesso a informação, etc. Neste caso o uso de ferramentas de simulação e de análise quantitativa exigem uma linguagem de especificação com um modelo matemático subjacente facilmente analisável como o de máquinas de estados finitas ou redes de Petri.

Uma organização de normatização é um ambiente mais pragmático que, de posse da tecnologia existente para protocolos, tenta agrupar as idéias e definir um padrão de protocolo com as características desejadas. Durante a elaboração do protocolo, há também necessidade de validação de novos conceitos e verificação de correção, mas a principal utilização de especificações neste ambiente está na divulgação de versões parciais e do protocolo definitivo.

Um ambiente industrial estará interessado principalmente no uso interno de especificações para

desenvolver implementações de protocolos, equipamentos compatíveis com outros protocolos, equipamentos de teste de implementações.

### 2.4.3 Adequação ao usuário

Uma linguagem com um aspecto muito matemático tem aceitação difícil entre profissionais de computação, especialmente de indústria. Uma linguagem semiformal, gráfica, sem muito rigor, provavelmente será bem aceita, embora seja questionável sua utilidade, de acordo com os critérios anteriores. Parece razoável, no entanto, supor que: assim como o usuário teve que se adaptar ao formalismo sintático e semântico das linguagens de programação, assim como em geral ele se convence que o uso de um formalismo para a definição de sintaxe de linguagens de programação acaba tornando mais fácil o aprendizado de linguagens, também o profissional envolvido com especificações de protocolos deve adaptar-se às linguagens formais mais apropriadas ao objetivo da especificação. A especificação formal da semântica das linguagens deve ser considerada como última instância de solução de dúvidas, uma vez que, em princípio, não existem ferramentas automáticas que permitam "testar" o significado das construções.

A adequação ao usuário deve ser perseguida sem prejuízo dos objetivos aos quais a linguagem se destina.

### 2.4.4 Adequação às características peculiares a cada nível de protocolo

[ISO 83a] fala em "the likely possibility that" um certo conceito possa ser apropriado para uma determinada camada mas inadequado para outra. Outros como [MER 79] afirmam que diferentes classes de protocolo requerem

diferentes técnicas de especificação de protocolos. Em [MER 79], no entanto, percebe-se que a afirmação é baseada na análise de técnicas muito simples, como os Sistemas de Condições e Eventos e as máquinas de estados finitas, que serão abordados na seção 2.6. São linguagens concisas mas que geram, em contrapartida, especificações muito pouco concisas e/ou imprecisas, para protocolos de qualquer nível.

"The likely possibility that" coloca [ISO 85a] em uma posição mais segura que, embora não exclua no futuro uma linguagem apropriada para qualquer nível, tem em vista diferenças como por exemplo entre as necessidades dos níveis superiores, que especificam trocas de mensagens com parâmetros e transformações de dados, e dos níveis inferiores, em que pode ser preciso especificar uma constante de tempo ou intervalo preciso. O nível físico é o que apresenta as maiores diferenças. Precisa-se especificar níveis de tensão nas saídas, retardos, proteção das entradas, velocidade de transmissão. Por mais que estas últimas características pareçam fornecer muito detalhe, elas são absolutamente necessárias para garantir o funcionamento correto de um ambiente de interconexão de sistemas de processamento de informações, em que cada sistema tenha sido implementado independentemente (objetivo do OSI-RM).

#### 2.4.5 Conclusão

Parece, ao autor, razoável centrar a análise das linguagens nos objetivos das especificações e na aplicação a cada nível. A adequação ao usuário aparece em segundo plano, sem prejuízo dos critérios anteriores. As diferenças de ambiente traduzem-se em grupos de objetivos.

As diferenças de objetivos e de necessidades em cada nível, pela análise do exposto anteriormente e do quadro atual de pesquisa, sugerem que se está no mínimo

muito longe de uma linguagem de especificação ideal, que há necessidade de integração entre linguagens e que, antes mesmo disto, há necessidade de aperfeiçoar/desenvolver (novas ) técnicas para cumprir separadamente cada objetivo eficazmente.

### 2.5 Uso de linguagens de programação para especificação formal

Viu-se que uma especificação deve ser expressa em alto nível de abstração, sem depender dos recursos disponíveis para implementá-la. A primeira vista, as características das linguagens de programação chamadas "de alto nível" parecem não depender dos recursos computacionais, aliás, elas foram criadas justamente para que os programas fossem portáveis de uma máquina para outra. Observa-se, no entanto, que a independência não é tão grande por duas razões: porque estas linguagens necessitam tradução automática (compiladores, montadores, pré-processadores); e porque elas são desenvolvidas visando a eficiência da tradução e do código gerado (facilidade e rapidez de tradução, possibilidade de gerar estruturas eficientes de dados e código de máquina). Com isso não se quer dizer que não seja interessante a existência de ferramentas automáticas que convertam eficientemente a especificação do protocolo em uma implementação. Pelo contrário. No entanto, considera-se que este é um objetivo secundário.

Outra característica que em geral também norteia o desenvolvimento de linguagens de programação, e que nada tem a ver com a máquina, é a generalidade das construções para permitir a implementação de sistemas para qualquer finalidade. Defende-se aqui que a linguagem de especificação deve ser concisa. Quer-se uma linguagem com capacidades

funcionais para especificar a(s) classe(s) de protocolos de interesse. As construções básicas devem representar elementos básicos (estruturas e atividades) envolvidos na definição de protocolos, tanto quanto possível. Vejamos um exemplo. É uma característica forte em protocolos a comunicação entre processos de sistemas independentes. É desejável, portanto, que a linguagem tenha comandos para envio e recebimento de mensagens. Não seria natural que a comunicação entre processos fosse especificada por mecanismos de compartilhamento de variáveis e exclusão mútua.

Ainda contra o uso de linguagens de programação para especificação formal depõe a dificuldade de uso de técnicas de validação (verificação) em especificações (programas) utilizando estas linguagens que normalmente caem em teste de programa (não valida totalmente, em geral), ou métodos de prova de programas utilizando assertivas com lógica clássica ou temporal, que são dos mais difíceis de aplicar.

## 2.6 Panorama das linguagens de especificação de protocolos

Tendo situadas as necessidades de uma linguagem formal de especificação e analisada a classe particular das linguagens de programação acompanha-se a seguir a evolução das correntes mais fortes na área de especificação de protocolos. Esta visão não pretende ser exaustiva. Artigos como [MER 79, BOC 80a, SAJ 85], dão uma visão ampla dos trabalhos na área. [SAJ 85] levanta a dificuldade de organizar os métodos de especificação em classes e sugere algumas alternativas. Apresenta também um levantamento dos métodos de verificação disponíveis associando-os às técnicas de especificação para as quais se adequam.

Inicia-se com a máquina de estados finita [HOP 79] (MEF, Finite State Machine, FSM, autômato finito), um dos primeiros métodos a se tornar popular na especificação de protocolos [MER 79, BOC 80a, BOC 80b, BOC 78]. As MEFs básicas (sem ou com poucas extensões) prestam-se à especificação de conceitos de protocolos em nível de especificação muito alto, sendo difícil obter precisão para qualquer protocolo real como veremos a seguir.

A aplicação mais simples da MEF em especificação de protocolos consiste em modelar uma única máquina global, que contenha todos os estados possíveis das entidades como um todo [MER 79]. Neste caso, as transições de cada sistema componente da interconexão estão implícitas nas transições da máquina única. Cada estado é uma combinação, possível de ser atingida, dos estados reais de cada entidade componente modelada.

Uma abordagem mais estruturada e concisa consiste em modelar cada entidade cooperante na interconexão como uma máquina de estados, e incorporar ao modelo básico destas a noção de comunicação entre máquinas. Tipicamente isto é feito colocando inscrições com um mesmo nome de evento de habilitação em transições de diferentes máquinas, impondo a restrição de que este fato implica que as transições com inscrições idênticas só podem ocorrer simultaneamente (comunicação síncrona) [BOC 80b]. [BOC 78] atribui nomes diferentes a cada transição e especifica informalmente pares (ou grupos) de transições que só podem ocorrer sincronizadas, chamando este procedimento de acoplamento direto (direct coupling). Por vezes distingue-se inscrições de mesmo nome em máquinas distintas indicando qual a máquina (transição) que gera o evento (mensagem) e qual a que recebe [RUD 85a, CHO 85]. Saliente-se que um gerador e receptor potencial de eventos é o ambiente externo, isto é, o

conjunto de entidades que supostamente interagem com aquelas que executam o protocolo para obter o benefício da interconexão.

Outra abordagem, similar à do parágrafo anterior, consiste em colocar duas inscrições em cada transição: uma referenciando um evento (mensagem) de habilitação, gerado por uma outra máquina ou pelo meio externo; e a outra inscrição referenciando os eventos (mensagens) enviados a outras máquinas ou ao ambiente [MOU 86]. Esta abordagem permite mais facilmente a interpretação da comunicação como assíncrona (dissocia-se a ocorrência de uma transição de uma máquina das transições possivelmente habilitadas pelo evento de saída da primeira).

A especificação de máquinas separadas deu também versatilidade à análise de protocolos que envolvem um número variável de entidades. Tipicamente as redes de teleprocessamento associadas a computadores de grande porte, que originaram os primeiros protocolos de comunicação tinham esta característica. O protocolo, conhecido como "mestre-escravo" definia a comunicação entre uma entidade controladora ou mestre, associada ao computador, e as entidades controladas ou escravas, associadas aos terminais remotos. Define-se uma máquina para a estação mestre e tantas instâncias quantas forem oportunas de máquinas com comportamento de estação escrava, por exemplo para uma atividade de predição de desempenho da rede.

Considere uma especificação composta pela definição de  $n$  máquinas. Seja  $e(i)$  o número de estados da máquina de índice  $i$ , com  $1 \leq i \leq n$ . O número de estados total máximo da máquina global é dado por

$$\text{PRODUTÓRIO } ( e(i) ) \\ i = 1 \text{ até } n$$

O número de estados real da máquina global, no entanto, é menor: nem todos os estados globais obtidos pelo produto cartesiano dos conjuntos de estados das  $n$  máquinas são atingíveis a partir do estado inicial. A experiência mostra, conforme [HOL 87], que uma estrutura que mostre os estados atingíveis dentro do conjunto de estados dado pelo produto cartesiano forma um vetor esparso.

Junto com as máquinas de estados, surgiu um dos métodos mais difundidos para verificação de propriedades em modelos de transição de estados, conhecido como **análise de atingibilidade** ou **análise de alcançabilidade** (*reachability analysis*) [CHO 85, MOU 86, BOC 80a, SAJ 85]. O método consiste em derivar totalmente ou em parte o grafo dos estados globais atingíveis, de acordo com a especificação das máquinas de estado cooperantes. Isto permite a verificação de propriedades gerais como a existência ou não de impasses (deadlocks) e também propriedades específicas desde que se atribua previamente um significado correspondente dentro do protocolo à propriedade verificada no modelo. Por exemplo, podemos verificar se o estado global  $Y$  é atingível a partir do estado  $X$ . Este procedimento só faz sentido se descobrirmos qual a afirmação correspondente no protocolo que estamos verificando, digamos, se uma estação pode transmitir uma nova mensagem após receber uma confirmação da anterior.

Um dos grandes inconvenientes das máquinas de estado apresentadas até o momento é o grande número de estados necessário para descrever formalmente e precisamente um protocolo real. A separação em diversas máquinas diminui a complexidade, porém, para fins de análise, quando é necessária a determinação da máquina global a multiplicação dos números de estados de cada máquina (mesmo com a consideração de que o vetor é esparso), em caso reais produz



a chamada **explosão de estados** ("state explosion") [MOU 86, BOC 80a, HOL 87]. Modificações da análise de alcançabilidade tem sido propostas [CHO 85, BOC 80a, SAJ 85] conduzindo a resultados parciais na verificação das propriedades.

Ocorre no entanto que, em geral, mesmo as entidades isoladas contêm aspectos que conduzem a uma explosão de estados da máquina. O exemplo típico é a inserção de uma variável de contagem ou de objetos em geral que assumem um entre um domínio de  $n$  valores. Este contador ou objeto, nada mais é do que uma máquina de estados de  $n$  elementos que deve ser multiplicada (produto cartesiano) pela máquina inicial. Note que aqui o problema da explosão de estados não é na análise mas na própria máquina especificada. Sem contar que o objeto (contador) perde o sentido dentro da especificação.

Os problemas até aqui citados sugerem que o uso da máquina de estados na prática é inviável. Merlin [24] levanta uma restrição ainda mais forte concernente à limitação do método à modelagem de aspectos finitos. Uma máquina finita não pode modelar filas infinitas ou número ilimitado de entidades. Estas estruturas infinitas, ou outras que possam eventualmente surgir, embora dificilmente ocorram em uma implementação, precisam por vezes serem previstas na especificação.

Para aumentar a precisão da especificação de protocolos com os modelos acima, frequentemente foram colocadas descrições textuais de atividades executadas como parte das transições, tornando a técnica semi-formal (se é que se pode falar em precisão com técnicas não formais). A estrutura da máquina de estados era usada para modelar os aspectos de controle do protocolo e as descrições textuais definiam as transformações de dados. Esta tem sido a técnica utilizada pela ISO e CCITT até o momento para divulgação de

seus protocolos de comunicação. É uma alternativa claramente "ao gosto do usuário", conforme discutido anteriormente, que traz benefícios questionáveis dentro dos objetivos aos quais as especificações se destinam.

Surgiram então as técnicas híbridas, que no caso de máquinas de estados associam tipicamente o conceito de MEF para controle e uma linguagem de programação para a descrição de objetos e transformações destes durante a execução da transição. O modelo fica bastante mais complicado, trazendo prejuízos às atividades de validação, em especial verificação. Obtém-se, porém, precisão e concisão das especificações, às custas da redução de concisão da linguagem. O balanço final resulta bastante favorável ao objetivo principal, de divulgação e documentação.

O modelo da linguagem Estelle [ISO 85c, BUD 87], apresentada no capítulo 2 segue por este caminho de máquina de estados estendida (MEFs).

Outra corrente dentro da classe dos modelos de transição que foi bastante utilizada [DIA 86, COU 84, JUR 84, WHE 85a] foram as redes de Petri, em suas diversas formas. Um Sistema de Condições e Eventos (Redes de Petri de Condições e Eventos, Sistemas C/E ou ainda Redes C/E) [REI 82, PET 81] é a classe mais simples das Redes de Petri. Tem o mesmo poder de especificação de uma MEF. A informação de estado está distribuída pela rede (C/E) inteira, através da existência ou não de uma marca nas condições. Pode ser representado através de seu grafo de casos [REI 82] que define uma MEF equivalente. A especificação traz os mesmos problemas que as MEFs: especificações em alto nível de abstração (ou no nível desejado com informalidade ou falta de precisão). A vantagem é que a distribuição da informação de estados dá mais informação qualitativa sobre o sistema,

como a distinção entre o estado dos vários objetos determinantes do estado e visualização dos diversos processos cujos eventos ocorrem em paralelo ou entrelaçados. A representação é, via de regra, mais concisa. Em [PRO 87] é descrita uma experiência de modelagem do método de acesso CSMA/CD por redes C/E sem inscrições.

Os métodos de determinação de invariantes [REI 82] e a própria análise de atingibilidade pela determinação do grafo de casos [REI 82, PET 81] foram utilizados em protocolos [MER 79, MER 76]. Veja-se também [SAJ 85].

Da mesma forma que nas FSMs, para aumentar a capacidade de expressão do método foram incorporadas extensões:

1a) Cada condição, que podia armazenar um valor booleano (vale ou não vale), passa a se chamar lugar, e pode conter um número variável de elementos (possivelmente limitado) chamados tokens ou marcas (**redes de lugares e transições** [REI 82, PET 81]) e estes tokens podem ser diferenciados. Os arcos passam a ter inscrições com predicados de habilitação das transições envolvendo número e tipo das marcas em cada lugar de entrada da transição. Estas redes conhecidas como **redes de predicados e transições** (**redes Pr/T**) [REI 82, PET 81] são uma expansão qualitativa e quantitativa dos sistemas C/E. [BUR 85] é um exemplo de uso em protocolos.

2a) Inclusão de ações de transformação de tokens nas transições (formal ou informalmente). As especificações geradas utilizando estas redes modelam os aspectos de controle na rede e a manipulação de dados nas ações detransformação de tokens. **Numerical Petri Nets (NPNs)** [WHE 85a, WHE 85b] é uma técnica de especificação que vai por este caminho.

3a) Inclusão explícita no modelo do conceito de tempo. Estas redes foram genericamente chamadas de **redes de Petri temporizadas** (time Petri nets) utilizadas em protocolos por [MER 76, MAR 87, RAZ 85].

Uma terceira corrente da especificação de protocolos tem sua origem com Milner e seu "Calculus on Communicating Systems" (CCS) [MIL 80]. O protocolo é especificado pelas possíveis seqüências de eventos observáveis externamente ao ambiente. Na descrição da seqüência é possível criar caminhos paralelos facilitando a especificação de paralelismo e entrelaçamento de ocorrência de eventos. É uma técnica bastante concisa e com aspecto matemático, sobre a qual Milner construiu uma teoria para determinação de equivalência de especificações com algumas restrições. A abordagem de Milner foi usada para compor parcialmente a linguagem LOTOS de especificação descrita no capítulo 3.

A outra abordagem integrante da linguagem LOTOS é a de tipos abstratos de dados (ADTs) [EHR 82], para descrição das estruturas de dados referenciadas na descrição do comportamento.

Uma última abordagem que será citada é a da **Lógica Temporal** [RES 71]. É uma extensão da lógica de predicados para acomodar explicitamente a variável "tempo", através da definição do operador **inevitavelmente** (em inglês, "eventually", que é um falso cognato em relação ao termo "eventualmente") que aplicado a um predicado significa que em algum momento do futuro, em relação ao estado atual, aquele predicado estará satisfeito; e do operador **sempre**, que aplicado a um predicado significa que em todos os estados futuros o predicado estará satisfeito. Devido à grande dificuldade na geração e interpretação de

especificações utilizando lógica temporal, poucas tem sido suas aplicações à especificação, sendo mais comum seu uso em verificação de protocolos, aliada a outra técnica para a especificação [MAN 81].

No capítulo 3 serão descritas duas linguagens (Estelle e LOTOS), em alto grau de definição, isto é, com uma sintaxe e semântica bem definidas, propostas pela ISO para constituírem padrões de linguagens formais de especificação de protocolos. Estas linguagens são representativas da tendência atual abordada nesta seção: Estelle, como representante das linguagens híbridas baseadas em máquinas de estados finitas, e LOTOS, também uma técnica híbrida, unindo a especificação de comportamento por sequências de eventos, com uma linguagem de definição de tipos abstratos de dados.



### 3 DESCRIÇÃO DAS LINGUAGENS

Neste capítulo descrevemos duas linguagens de especificação em desenvolvimento pela ISO, já citadas no capítulo anterior. A descrição de Estelle é dada aqui para que o volume seja autocontido. Algumas descrições alternativas podem ser encontradas em [ISO 85c] (original da ISO) e [BUD 87] (com algumas diferenças por referir-se a uma versão um pouco diferente de [ISO 85c]). A descrição de Lotos tem mais pertinência, pois no documento da ISO [ISO 87] ela se encontra descrita apenas formalmente, o que torna difícil a compreensão. Em outras fontes (tutorial de [ISO 87], [BOL 87, BRI 85, CAR 86, SOU 87a]), encontrou-se os principais aspectos da linguagem, omitindo detalhes. A descrição aqui apresentada, assessorou-se fortemente na descrição formal para a interpretação da semântica.

Na descrição da linguagem é empregada a notação BNF (Backus Naur Form) para a descrição da sintaxe da linguagem. Nas sentenças descritas por BNF adotaremos as seguintes convenções:

::	é definido como;
< >	delimitadores de meta_símbolos.
" "	delimitadores de símbolos terminais.
.	fim da definição.
	alternativa
[x]	indicação de sequência opcional: 0 ou 1 instância de x.
{x}	número ilimitado (0 até infinito) de instâncias de x.
+{x}	no mínimo uma instância de x.
(x y ...)	utilizado para definição de pedaços alternativos no meio da sequência.

### 3.1 Estelle

A linguagem Estelle está sendo desenvolvida pela ISO, pelo subgrupo B do grupo de trabalho WG 1, subcomité SC 21 do comité técnico TC 21, encontrando-se no estágio de "draft proposal" (DP), no documento [ISO 85a] aqui utilizado.

#### 3.1.1 Modelo

Uma especificação em Estelle consiste na definição hierárquica de um conjunto de módulos. A especificação é considerada um módulo especial, no topo da hierarquia, não tendo pai nem irmãos. Um módulo pode ter associado uma máquina de estados finita extendida (Extended Finite State Machine, abreviada EFSM) que rege seu comportamento ativo. Módulos folha (módulos que não tem filhos) devem tê-la obrigatoriamente definida. O modelo da EFSM é descrito adiante.

Os módulos podem ser ligados através de canais para troca de interações. Uma interação pode conter parâmetros passados "by value". Os canais são possivelmente diferentes uns dos outros. Cada canal tem definidos papéis. Um papel é um conjunto de interações que podem ser enviadas através de um dos extremos do canal. Cada um dos pontos conceituais de um módulo onde são feitas as associações com os canais é chamado ponto de interação. Um ponto de interação tem definido qual dentre os papéis do canal ele estará interpretando, isto é, qual dentre os conjuntos de interações ele poderá enviar ao módulo que está do outro lado do canal.

Um ponto de interação tem associado uma fila de capacidade ilimitada de armazenamento que mantém ordenadas as interações que a ele chegam através do canal. O ponto de



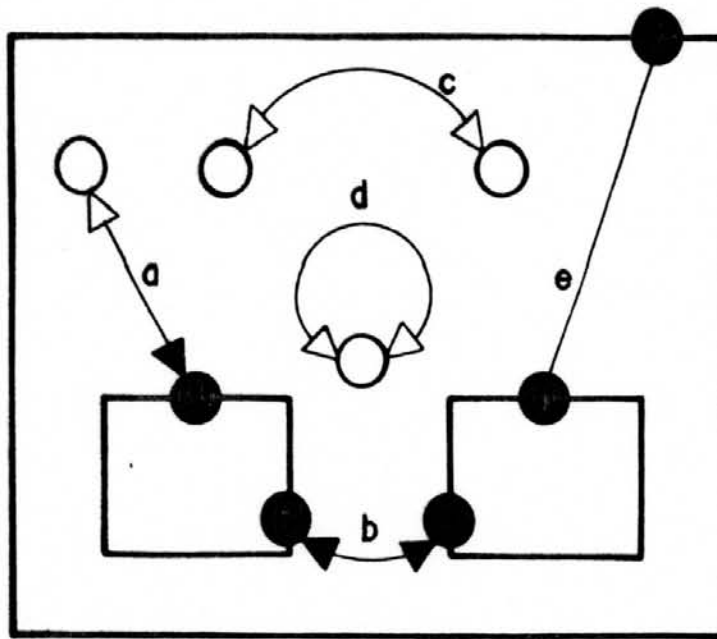
interação pode optar entre ter uma fila individual ou compartilhar o uso da fila comum do módulo. As operações que podem ser realizadas sobre esta fila são a inserção de interação na fila (em decorrência da chegada da interação pelo canal); a procura de uma determinada interação (fornecida como argumento) no topo da fila, com acesso aos valores dos parâmetros; e a retirada da interação presente no topo da fila com apropriação dos parâmetros. As operações (inserção, procura e retirada), fazem parte do modelo da EFSM.

Um módulo só pode estar ligado (através de canais) ao pai, a filhos ou a irmãos, não existindo visibilidade direta entre módulos com outro grau de parentesco. As ligações via canal podem ocorrer entre dois módulos irmãos, entre um módulo pai e um filho (estas duas primeiras, ligações `inter_módulo`), entre dois pontos de interação de um mesmo módulo, ou ainda em laço, de um ponto de interação para si mesmo (estas duas últimas, `intra_módulo`).

Os pontos de interação definidos anteriormente podem ser de dois tipos: pontos de interação externos ou pontos de interação internos. O primeiro é visível pelo pai e pelos irmãos e é usado para ligação com estes. O segundo, não é visível externamente ao módulo (pai e irmãos), sendo usado para comunicação com módulos filhos ou comunicação `intra_módulo`.

As ligações entre pontos de interação para comunicação entre módulos (ou `intra_módulo`) podem ser de dois tipos: conexão ou transferência (adaptação do termo "attach" utilizado na definição original da linguagem). A conexão é o tipo normal de ligação entre dois pontos de interconexão através de um canal, em que cada ponto de interação tem associado um papel diferente do outro e os

módulos aos quais pertencem os pontos são diretamente responsáveis pelo envio (recebimento) de interações para o



○ PONTO DE INTERAÇÃO INTERNO

● PONTO DE INTERAÇÃO EXTERNO

- a - CONEXÃO ENTRE UM PONTO DE INTERAÇÃO INTERNO DO MÓDULO COM UM EXTERNO DE UM FILHO.
- b - CONEXÃO ENTRE PONTOS DE INTERAÇÃO EXTERNOS DE FILHOS.
- c - CONEXÃO ENTRE PONTOS DE INTERAÇÃO INTERNOS DO PRÓPRIO MÓDULO.
- d - CONEXÃO EM LAÇO DO PONTO DE INTERAÇÃO.
- e - TRANSFERÊNCIA (ATTACH) DE UM PONTO DE CONEXÃO EXTERNO DO MÓDULO AO EXTERNO DO FILHO.

figura 3.1 Tipos de ligações possíveis entre pontos de interação

(do) outro lado. Há duas possibilidades de conexão entre módulos: entre pontos de interação externos de irmãos ou entre um ponto de interação interno de um módulo e um ponto de interação externo de um filho; e duas possibilidades de conexão intra\_módulo: entre dois pontos de interação internos de um mesmo módulo ou em laço, de um ponto de interconexão interno consigo mesmo (neste caso, existindo apenas um papel). As quatro possibilidades estão representadas na figura 3.1. Um ponto de interação não pode estar envolvido em mais de uma conexão.

A transferência (também na figura 3.1), é um tipo de ligação especial visando aumentar o grau de visibilidade na descendência, por concessão de módulos intermediários. Ocorre sempre de um ponto de interação externo de um módulo para um ponto de interação externo de um módulo filho. Tem o sentido de "delegação de poderes" (o ponto de interação do filho substitui as funções do módulo pai de envio e recebimento de interações). As interações que chegam ao ponto de interação do módulo pai são automaticamente transferidas para o filho e vice-versa. O ponto de interação do módulo filho poderá, por sua vez, ter uma transferência estabelecida para um "módulo neto", transferindo para um nível inferior o destino das interações. A recepção é feita na fila associada ao ponto inferior da linha de transferência. O ponto de interação mais acima desta linha de transferência estará conectado a um outro ponto qualquer do qual as interações são recebidas. A transferência existe nos dois sentidos, isto é, as interações enviadas pelo ponto de interação inferior da linha de transferência acabarão no canal utilizado pela conexão do ponto de interação superior da linha. Uma vez estabelecida a transferência de um ponto de interação de um módulo para um ponto do filho, todas as interações que estavam armazenadas na fila do primeiro, são

automaticamente transferidas para a fila do extremo inferior da linha de transferência. Quando é desfeita uma transferência, todas as interações armazenadas na fila (do ponto inferior da linha de transferência, obviamente), são transferidas para o ponto do módulo que a desfez.

A especificação não tem definidos pontos de interação externos, pois só se comunica com filhos ou consigo mesma (ligações intra\_módulo).

A máquina de estados finita pode ou não estar presente em um módulo intermediário da hierarquia, sendo obrigatória em módulos folha (módulos que não tem filhos). Ela provê a especificação do comportamento dinâmico do sistema. Pode-se dizer que um módulo com máquina de estados definida é um módulo ativo e que, um módulo sem máquina tem todas as funções delegadas aos filhos. Neste caso, nota-se a importância da transferência para comunicação com módulos externos.

A EFSM de Estelle é uma extensão do conceito tradicional de máquina de estados finita (FSM). Não é descrito aqui o modelo que rege uma FSM, mas sim, os elementos da EFSM de Estelle a partir do modelo considerado conhecido das FSMs [HOP 79]. Uma EFSM de Estelle é composta por um estado inicial e um conjunto de transições, cada transição definida por uma quádrupla

(estado de origem,  
conjunto de cláusulas de habilitação,  
estado seguinte,  
conjunto de ações),

analogamente às máquinas de Mealy [HOP 79].

Define-se os conceitos de estado atual e execução de uma transição como segue: No princípio do funcionamento da máquina, o estado inicial é o estado atual. Se, num dado

momento, S1 for o estado atual da máquina e for executada um transição com estado de origem igual a S1 e estado seguinte igual a S2, então, após a execução, S1 deixa de ser o estado atual e S2 passa a sê-lo.

Uma transição só pode ser executada se estiver habilitada. Uma transição estará habilitada se o estado atual for seu estado de origem e se todas as cláusulas de seu conjunto de cláusulas de habilitação estiverem satisfeitas. Quando mais de uma transição estiver habilitada em um dado instante, a escolha da transição a ser executada é não-determinística, configurando uma liberdade para a implementação.

Sempre que houver transições habilitadas, uma delas deverá necessariamente ser imediatamente executada, isto é, o intervalo de tempo entre o surgimento de transição habilitada e sua execução é nulo. Contudo, o tempo gasto pela seleção e execução da transição não é especificado no modelo. Pode ocorrer, ainda, a situação de postergação indefinida, devido ao não determinismo.

Sempre que, havendo uma fila não vazia, não existir uma transição habilitada configura-se uma situação de erro de especificação.

O conjunto de cláusulas de habilitação de uma transição pode conter uma cláusula de cada um dos tipos abaixo:

a) Um ponto de interação do módulo, junto com uma interação específica que possa ser recebida por este ponto de interação (pertencente ao papel do ponto associado a outra extremidade do canal). Esta cláusula estará satisfeita se a interação especificada estiver presente no topo da fila associada ao ponto de interação citado. Esta cláusula não é obrigatória. Uma transição que não contenha esta cláusula é

dita espontânea.

b) Um predicado de habilitação para a transição, cuja avaliação deverá ser "verdadeiro" para que esta cláusula seja satisfeita. A formação do predicado segue o modelo da linguagem Pascal-ISO [ISO 83b] para expressões booleanas e pode referenciar parâmetros da interação da cláusula descrita acima. É cláusula obrigatória.

c) Uma cláusula de prioridade. Transições diferentes tem possivelmente prioridades diferentes. Em caso de conflito, a diferença de prioridades pode decidir quais as transições que estão realmente habilitadas (as de maior prioridade, que simultaneamente satisfizerem os outros critérios). É uma cláusula obrigatória.

d) Uma cláusula temporal: uma tupla  $\langle dt1, dt2 \rangle$ ,  $dt1$  pertencente aos naturais e  $dt2$  pertencente aos naturais com inclusão de "infinito". Os elementos da tupla representam dois intervalos de tempo, cuja unidade não é especificada. A partir do momento (tempo)  $t$  em que todas as outras cláusulas (citadas acima) da transição estiverem satisfeitas, enquanto permanecerem continuamente satisfeitas, vale que, a transição tornar-se-á habilitada (mas não necessariamente será executada) dentro do intervalo de tempo  $[t + dt1, t + dt2]$ . É cláusula obrigatória. Note-se outra forma de não determinismo gerada pelo fato de a transição ter um intervalo possível para estar habilitada, que ficará a cargo de implementação, e pode ser entendido como uma especificação de tolerância, ou margem de erro aceitável pelo sistema. Note-se ainda que, se alguma das outras condições que habilitam a transição se tornar falsa, dentro do intervalo e, a transição ainda não tiver sido habilitada, a avaliação desta cláusula e da habilitação da transição como um todo volta ao ponto de partida. Uma

restrição existente em Estelle é que, se estiver presente a cláusula referenciando a interação, a cláusula temporal deverá ser  $\langle 0,0 \rangle$ .

Observe-se que a obrigatoriedade das cláusulas não significa que elas sejam obrigatórias na sintaxe da linguagem, pois podem existir valores pré-estabelecidos (defaults), tipicamente: "verdadeiro" para o predicado, "mínima" para a prioridade e  $\langle 0,0 \rangle$  (habilitação imediata) para a cláusula temporal.

A determinação da habilitação pode ser vista como uma avaliação simultânea e ininterrupta das cláusulas. Mesmo com cláusula temporal diferente de  $\langle 0,0 \rangle$ , a cada instante é feita uma avaliação (continuamente) de todas as cláusulas para garantir o "progresso" da satisfação da cláusula temporal.

A execução de uma transição, uma vez habilitada, é indivisível. Compreende a retirada da interação da fila, a apropriação dos parâmetros por variáveis e a execução do conjunto de ações associado à transição. Nenhum resultado intermediário durante a execução de uma transição é observado. Por exemplo: o progresso da habilitação de uma transição qualquer com cláusula temporal não é afetado se seu predicado de habilitação se tornar momentaneamente falso devido ao valor intermediário de uma variável durante a execução de uma outra transição.

Dentre o conjunto de ações possíveis está o envio de interações através de pontos de interação do módulo. No final da execução (indivisível) de uma transição, todas as interações enviadas estarão automaticamente nas filas associadas aos pontos de interação no outro extremo dos canais pelos quais foram enviadas. Outros tipos de ação serão mencionados adiante, quando for descrita a capacidade

de alteração da estrutura hierárquica de módulos e ligações. No documento da ISO definindo Estelle é proposta a inclusão dos elementos do modelo do Pascal-ISO [ISO 83b] com algumas adaptações, para dotar o conjunto de ações de Estelle do poder de uma linguagem de programação, caracterizando assim uma linguagem híbrida.

Uma transição de um módulo não pode ser executada em paralelo com transições de qualquer um de seus descendentes (ou ascendentes) em linha direta. Se uma transição de um módulo estiver habilitada, nenhuma transição de módulos descendentes poderá ser iniciada (contudo, não existe preempção, uma vez que a execução da transição é indivisível), isto é, existe uma relação de prioridade embutida no modelo entre as transições de módulos pais e filhos.

Quanto a módulos irmãos, a possibilidade ou não de ocorrência de transições em paralelo depende do tipo de módulo. Um módulo qualquer pode ser de dois tipos: **processo** ou **atividade**. Módulos irmãos serão sempre do mesmo tipo (um módulo pode ser estruturado ou só por processos filhos ou só por atividades filhas). Processos irmãos podem executar transições em paralelo, enquanto que entre atividades irmãs apenas uma delas pode ter uma transição sendo executada em um dado instante (execução entrelaçada das transições) sem distinção de prioridade. Módulos do tipo atividade não podem ser subestruturados; são sempre módulos\_folha.

Estelle permite comunicação via compartilhamento de variáveis de um módulo com o módulo pai. Permite também o compartilhamento de variáveis entre irmãos, desde que estes sejam atividades (o fato de serem atividades elimina o problema de exclusão mútua no acesso às variáveis).

Algumas operações (ou ações) que podem ser



efetuadas durante as transições foram postergadas; descrevem a capacidade de alteração da estrutura hierárquica (módulos e ligações) da especificação durante o funcionamento do sistema. São elas: operações de criação e destruição de módulos filhos (executada pelo pai); operações de estabelecimento/terminação de conexões e transferências (efetuadas pelo módulo pai entre dois filhos ou entre filho e o próprio pai; ou efetuada por um módulo qualquer entre dois pontos de interação internos do próprio módulo). Um módulo tem um conjunto definido de pontos de interação externos e internos, que podem estar ou não ligados em um dado momento.

Quando um módulo é criado ele executa de imediato (como parte da operação de criação) uma transição inicial para o estado inicial da máquina (se houver uma EFSM definida). Esta transição permite a execução de ações do mesmo tipo que nas transições da máquina de estados, inclusive criação de módulos, estabelecimento de conexões, envio de interações, etc. A criação de um módulo pode então causar uma complexa operação atômica em cadeia de criação de toda uma sub\_árvore da hierarquia de módulos e ligações. Isto pode acontecer se, dentro da transição inicial do módulo que está sendo criado, estiverem presentes novas ações de criação de submódulos, que portanto deverão ser executadas como parte da execução da referida transição inicial. Ora, cada uma destas operações de criação de submódulos consiste na execução da respectiva transição inicial. De modo que, ao final da operação de criação do primeiro módulo, que é parte da execução de uma transição e portanto uma operação atômica, o resultado é a criação de uma árvore de módulos, de cuja o primeiro módulo é a raiz. Na figura 3.2 é apresentado um exemplo de especificação, em uma sintaxe simplificada diferente da de Estelle (item 3.1.2), mostrando a criação de módulos filhos na transição

inicial de um módulo. A figura 3.3 mostra a sequência de eventos quando é executada a criação de um módulo do tipo "Exemplo", conforme a figura 3.2. A especificação é um módulo automaticamente criado. A criação e conexão de módulos durante a transição inicial de um módulo é dita subestruturação estática.

```

tipo_módulo    Exemplo
  transição_inicial:
    cria_instância    ma1    do_tipo    Mod_def_a
    cria_instância    mb1    do_tipo    Mod_def_b
    cria_instância    ma2    do_tipo    Mod_def_a

tipo_módulo    Mod_def_a
  transição_inicial:
    cria_instância    mc1    do_tipo    Mod_def_c

tipo_módulo    Mod_def_b
  transição_inicial:
    cria_instância    md1    do_tipo    Mod_def_d
    cria_instância    me1    do_tipo    Mod_def_e

tipo_módulo    Mod_def_c
  transição_inicial: /* não tem criação de módulo */

tipo_módulo    Mod_def_d
  transição_inicial: /* não tem criação de módulo */

tipo_módulo    Mod_def_e
  transição_inicial: /* não tem criação de módulo */

```

figura 3.2 Processo de criação de módulos

- 1o) inicio da criação da instância do módulo Exemplo
- 2o) inicio da criação da instância mai de Mod\_def\_a
- 3o) criação de mc1 (filho de mai)
- 4o) fim da criação de mai
- 5o) inicio da criação da instância mbi de Mod\_def\_b
- 6o) criação de mdi
- 7o) criação de mei
- 8o) fim da criação de mbi
- 9o) inicio da criação da instância ma2 de Mod\_def\_a
- 10o) criação de mc1 (filho de ma2)
- 11o) fim da criação de ma2
- 12o) fim da criação da instância do módulo Exemplo

figura 3.3 Sequência de eventos na criação de módulos

Um outro modelo que foi imaginado no decorrer deste trabalho para descrever a semântica de Estelle, diferente mas equivalente ao da transição inicial, consiste em considerar que a cada operação de criação de módulo durante uma transição de uma máquina de estados, ou no inicio da execução (para a especificação), é gerada uma (sub\_)estrutura inicial, com ligações, variáveis inicializadas e filas (dentro da subestrutura) possivelmente não vazias. Esta estrutura poderia ser alterada durante o funcionamento. O modelo da transição inicial apenas está mais próximo da linguagem, vista no item seguinte.

### 3.1.2 Definição da sintaxe e semântica

Descreve-se aqui os principais componentes da linguagem. São omitidos detalhes ou alternativas que complicam o entendimento da linguagem e não abordam pontos relevantes para fins de uso ou análise da linguagem. Meta\_símbolos escritos com letras maiúsculas são definições do Pascal ISO apropriadas por Estelle. Algumas destas definições são extendidas por Estelle. Meta\_símbolos em letras minúsculas são particulares de Estelle. Assume-se "bom senso" na interpretação de termos com terminação do tipo `_identifier` (um nome que identifica o objeto sendo definido ou referenciado) ou `_type` (um nome que identifica o tipo sendo definido ou referenciado).

Na definição da linguagem consideraremos inicialmente a definição de um módulo qualquer e depois a definição da especificação que é uma definição de módulo com algumas características diferentes. A definição de um módulo é uma definição de tipo, não implicando obrigatoriamente na existência do módulo (instância) durante o funcionamento do sistema. Quando da execução da criação de um módulo, como veremos mais tarde, é criada uma instância do tipo de módulo desejado.

Um módulo é definido através de duas construções: `<module_header_definition>` e `<module_body_definition>`. A primeira define as características visíveis na interface externa do módulo (visível pelo pai e irmãos).

```
<module_header_definition>::
    "module" <header_type> <class>
    [ "(" <module_parameter_list> ")" ] ";"
    [ "export" <exported_variable_list> ";" ].

<class>:: "process" | "activity".
```

```

<module_parameter_list>::
    <interaction_point_list> ";" <parameter_list>
  | <interaction_point_list>
  | <parameter_list>.

<interaction_point_list>::
    <interaction_point_declaration>
  { ";" <interaction_point_declaration> }.

<interaction_point_declaration>::
    <interaction_point_identifier> ":"
      <interaction_point_type>
  | <interaction_point_group_identifier> ":"
    "array" "[" <index_type_list> "]"
    "of" <interaction_point_type>.

<interaction_point_type>::
    <channel_type_identifier>
      "(" <role_identifier> ")"
  [ <queue_discipline> ].

<queue_discipline>:: "common" "queue"
  | "individual" "queue"

```

A definição do tipo de interface acima é composta por:

- a) um identificador (<header\_type>) do tipo;
- b) a classe: processo ou atividade;
- c) possivelmente uma lista de variáveis (cláusula "export") que poderão ser acessadas pelo módulo pai e irmãos (este último caso somente se for uma atividade). Observe-se que as variáveis exportadas deverão estar declaradas nas <module\_body\_definition> associadas a esta interface;

d) possivelmente um conjunto de parâmetros (<parameter\_list>) que terão valores atribuídos pelo comando de criação de instância de módulo, permitindo variações das instâncias de um mesmo tipo;

e) o conjunto de pontos de interação do módulo (<interaction\_point\_list>) individualmente ou como vetores, associando-os a tipos de canais (pré-definidos), adotando um dos papéis (<role>) destes canais. Especifica-se ainda se o ponto receberá interações por uma fila individual ou pela fila comum ao módulo. Os canais aqui referenciados devem ter sido declarados antes desta declaração de interface, e em um módulo cujo escopo engloba o que está sendo definido. O módulo mais externo (especificação) não tem canais pré-definidos nem pontos de interação.

Pode-se ter diversas definições de corpo de módulo (<module\_body>) para a mesma definição de interface, utilizando-se a construção abaixo. <body\_identifier> é o nome do tipo de módulo completamente definido (interface + corpo) que será usado na criação de instâncias de módulos .

```
<module_body_definition>::
    "body" <body_identifier>
        "for" <header_type> ";"
    ( <body_definition> "end" ";"
    | "external" ";" ).
```

A especificação é uma definição de módulo que não tem interface (<module\_header>). O sistema descrito é fechado, não tendo pontos de interação externos. E por definição um processo. Define apenas um corpo que é automaticamente instanciado (não fazendo sentido o uso de parâmetros).

```

<specification>:: "specification" <IDENTIFIER> ";"
                [ <default_options> ]
                <body_definition>
                "end" "." .

```

```

<body_definition>:: <declaration_part>
                  <initialization_part>
                  <transition_declaration_part>
                  <termination_part>

```

Na construção <body\_definition>, que define o corpo dos módulos, inclusive da especificação, o terceiro meta\_símbolo do lado direito define a máquina de estados, o segundo, a transição inicial do módulo, o quarto uma "transição de saída", executada quando da destruição do módulo pelo pai. Antes de analisá-los melhor vamos ao primeiro meta\_símbolo, que contém as definições estruturais embutidas no módulo.

```

<declaration_part>:: {<declarations>}

<declarations>:: <CONSTANT_DEFINITION_PART> |
                 <TYPE_DEFINITION_PART> |
                 <channel_type_definition> |
                 <module_type_definition> |
                 <VARIABLE_DECLARATION_PART> |
                 <state_set_definition_part> |
                 <use_clause> |
                 <PROCEDURE_AND_FUNCTION_DECLARATION_PART> .

```

As construções <channel\_type\_definition> e <module\_type\_definition> só terão sentido se existir a possibilidade de o módulo em questão criar filhos: ou na

inicialização ou durante a execução de transições da máquina de estados. Definições de módulos de tipo atividade não terão estas cláusulas.

A definição de um tipo de canal, cuja sintaxe é apresentada abaixo, compreende o nome do tipo (<channel\_type\_identifier>) e um conjunto de papéis (<role\_identifier>s). A cada papel ou grupo de papéis citados na construção <channel\_heading>, é associado um conjunto de interações em <channel\_block>. Na definição de um ponto de interação de um módulo, apresentada acima, viu-se que era especificado um canal e um papel definido naquele canal. Isto significa que, por aquele ponto de interação, após ser conectado a outro ponto de interação, o módulo poderá enviar apenas as interações associadas a este papel.

```

<channel_type_definition>::
    <channel_heading> <channel_block>.

<channel_heading>::
    "channel" <channel_type_identifier>
        "(" <role_list> ")" ";"

<channel_block>:: +{ <interaction_group> }.

<interaction_group>::
    "by" <role_list> ":"
        +{ <interaction_definition> }.

<role_list>:: <role_identifier>
    { "," <role_identifier> }.

<interaction_definition>::
    <interaction_identifier>
    [ "(" <VALUE_PARAMETER_SPECIFICATION>
        { ";" <VALUE_PARAMETER_SPECIFICATION> }
    "]" "]" ";".

```



A definição de uma interação é um identificador, possivelmente com parâmetros definidos de acordo com o Pascal.

Para a definição completa de módulos é preciso o uso repetido da construção <module\_type\_definition> abaixo (através de várias instâncias de <declarations>), de forma que as definições de corpo dos módulos (<module\_body\_definition>) sejam precedidas da declaração da interface (<module\_header\_definition>) usada. Neste momento já se pode notar o procedimento recursivo utilizado na definição aninhada de tipos de módulos (<module\_header\_definition> possui uma <body\_definition>, que deriva em <declarations>, seguindo-se <module\_type\_definition>, que permite a definição de um módulo aninhada).

```
<module_type_definition>::
    <module_header_definition>
    | <module_body_definition>.
```

<state\_set\_definition> permite associar um nome com um conjunto de estados para uso na definição da máquina.

<use\_clause> define os tipos de interface (<header\_type>) a cujas variáveis exportadas o módulo tem acesso.

Os outros meta\_símbolos na definição de <declarations>, escritos em maiúsculas, são os do Pascal, com algumas alterações impostas por Estelle.

Os seguintes denotadores de tipo são

adicionalmente aceitos por Estelle para declaração de variáveis:

a) `<interaction_point_type>`, para a definição de pontos de interação internos (uma variável definida com um tipo `<interaction_point_type>` é um ponto de interação interno com as características especificadas conforme sintaxe vista anteriormente.

b) `<looping_interaction_point_type>` define um tipo de ponto de interação interno, automaticamente conectado consigo mesmo.

```
<looping_interaction_point_type>::
    <channel_type_identifier>
    "(" <role_identifier> "," <role_identifier> ")"
    [ <queue_discipline> ].
```

c) `<header_type>`, conforme definido em `<module_header_definition>`. Variáveis cujo tipo seja o de uma interface (`<header_type>`) definida são usadas como apontadores para instâncias de módulos com aquela interface. O apontamento é feito quando a variável é referenciada em um comando de criação de módulo, podendo ser usada para acessar os elementos da interface do módulo.

d) um denotador qualquer de tipo pode ainda ser precedido do termo "optional", denotando um tipo semelhante ao anterior, podendo, porém, assumir o valor "undefined".

Um identificador ou constante, pode ser substituído por "...", significando um tipo ou constante ou expressão não especificado, deixado para a implementação.

Na definição abaixo de `<initialization_part>`, a construção `<init_procedure>` foi simplificada.

```

<initialization_part>::
    { "initialize" <init_procedure> }.

<init_procedure>::
    +{ [ "provided" ( BOOLEAN_EXPRESSION
                    | "otherwise" )
        ]
        "to" <state_identifier>
        <transition_block> ";" .
    }

```

<initialization\_part> é executada quando da criação de uma instância do tipo de módulo em questão. A inicialização consiste na execução de uma transição inicial (<transition\_block>) para o estado inicial (<state\_identifier>). O par <transição inicial, estado inicial> é escolhido de acordo com a avaliação do predicado associado pela construção "provided", podendo haver não determinismo. As expressões booleanas poderão fazer uso dos parâmetros recebidos.

A construção <transition\_declaration\_part>, abaixo, permite definir completamente a máquina de estados. A construção <transitions> facilita a definição das transições agrupando-as de acordo com a semelhança entre as cláusulas.

```

<transition_declaration_part>::
    { "trans" <transitions> }.

<transitions>::
    +{ "when" <interaction_reference>
        [ <interaction_argument_list> ]
        <transitions>
    }
    | +{ "from" <state_list> <transitions>
    }

```

```

| +{ "to" <to_list> <transitions> }
| +{ "provided" ( <BOOLEAN_EXPRESSION> |
                  "otherwise" )
      <transitions> }
| +{ "delay" "("
      ( <EXPRESSION> "," <EXPRESSION> |
        <EXPRESSION> "," "*" |
        <EXPRESSION> )
      ")" <transitions> }
| <transition_block> ";"

```

Permite-se a colocação em qualquer ordem dos elementos associados a uma transição, exceto o <transition\_block>, que contém o conjunto de ações, que sempre aparece no final da transição. Note-se que não tem sentido especificar, para uma mesma transição, mais de uma vez a mesma cláusula ("from", "delay", etc.).

As expressões usadas (<EXPRESSION>) não podem ter efeitos colaterais. A expressão da cláusula "provided" pode utilizar os valores dos argumentos da cláusula "when" (sem haver ainda a apropriação dos argumentos, que se dá apenas durante a execução da transição).

A construção acima também permite a estruturação da definição das transições em "níveis de aninhamento", permitindo a definição de diferentes transições, através da definição de uma mesma cláusula repetidas vezes, em um mesmo nível, conforme mostra a figura 3.4. A figura 3.5 mostra a tabela de transições geradas pela definição da figura 3.4.

```

FROM e1
  WHEN ip1. i1 (p1)
    PROVIDED p1 = 0
      TO e2, e3
      <tb1>;

    PROVIDED p1 = 1
      TO e4
      <tb2>;

    PROVIDED otherwise
      To e5
      <tb3>;

  WHEN ip1. i2
    TO e3
    <tb4>;

FROM e1, e2
  TO e1
  DELAY (1, 2)
  <tb5>;

```

figura 3.4 Exemplo de definição da máquina de estados

	t1	t2	t3	t4	t5	t6	t7
est. inicial	e1	e1	e1	e1	e1	e1	e2
est. final	e2	e3	e4	e5	e3	e1	e1
interação	ip1. i1	ip1. i1	ip1. i1	ip1. i1	ip1. i2	-	-
predicado	p1= 0	p1= 0	p1= 1	p1	-	-	-
cl. temporal	-	-	-	-	-	<1,2>	<1,2>
bloco trans.	<tb1>	<tb1>	<tb2>	<tb3>	<tb4>	<tb5>	<tb5>

figura 3.5 Diagrama de transições geradas

O meta\_simbolo <state\_list>, que permite definir uma lista de estados, na verdade está definindo um conjunto de transições, partindo cada uma de um estado diferente (constante da lista), mas com mesmas características (estado final, bloco de transição, etc.). O meta\_simbolo <to\_list>, que permite a definição de um conjunto de próximos estados possíveis, é uma fonte de não determinismo.

Na cláusula temporal ("delay") o intervalo é dado por duas expressões. "\*" significa "infinito". Se for fornecida apenas uma expressão "E", isto é equivalente ao intervalo <E, E>. Se não for especificada, é assumido <0, 0>.

Na construção "when", especifica-se um ponto de interação (individual ou um elemento de um vetor de pontos) e uma interação específica, conforme abaixo, com os argumentos se houver.

```
<interaction_reference>::
    <interaction_point_reference>    "."
    <interaction_identifier>.
```

Note-se na figura 3.5 que existem duas transições de e1 para e3 com dois conjuntos de cláusulas de habilitação e ação.

Quando aparecerem dois ou mais estados em uma cláusula "from", a diferença entre os estados terá que aparecer em outro ponto, em que não apareçam juntos (como e1 e e2 na figura 3.5).

Se houver cláusula "when" não pode haver cláusula "delay" (associadas a uma mesma transição). Se a construção "provided" não estiver presente, é assumido o valor

"verdadeiro". Se "priority" não for especificada assume-se a prioridade mais baixa. Se o estado inicial não estiver presente, significa que a transição procede de qualquer estado. Pode-se definir como estado final o termo "same", indicando uma transição reflexiva. Se não for fornecida a cláusula "to", é assumido "same".

A seguir é definido o bloco de transição.

```
<transition_block>::
    <LABEL_DECLARATION_PART>
    <CONSTANT_DEFINITION_PART>
    <TYPE_DEFINITION_PART>
    <VARIABLE_DECLARATION_PART>
    <PROCEDURE_AND_FUNCTION_DECLARATION_PART>
    [ <transition_name> ]
    <STATEMENT_PART>.
```

A cláusula <STATEMENT\_PART> do Pascal é estendida para comportar novas derivações de <simple\_statement>s e <structured\_statement>s definidas abaixo:

<init\_statement> e <release\_statement> são respectivamente os comandos de criação e destruição de módulos. <module\_variable\_access> deve ser uma variável de tipo <header\_type> igual ao <body\_identifier>. Pode-se, usando comando "forall" ou "forone" criar (destruir) módulos (destruir) sem ter uma variável para apontá-los (vide adiante).

```
<init_statement>::
    "init" <module_variable_access>
    "with" <body_identifier>
```

```
[ "(" <actual_module_parameter_list> ")" ].
```

```
<release_statement>::
```

```
  "release" <module_variable_access>.
```

Comandos <attach> e <detach> são usados para estabelecer/desfazer transferências. Comandos <connect> e <disconnect> são usados para estabelecer/desfazer conexões.

```
<attach_statement>::
```

```
  "attach" <interaction_point_variable>
```

```
  "to" <module_variable_access> "."
```

```
    <interaction_point_variable>.
```

```
<connect_statement>::
```

```
  "connect" <interaction_point_access>
```

```
  "to" <interaction_point_access>.
```

```
<detach_statement>:: "detach"
```

```
  ( <interaction_point_variable> |
```

```
    <module_variable_access>  ","
```

```
    <interaction_point_variable>  ).
```

```
<disconnect_statement>:: "disconnect"
```

```
  ( <interaction_point_access> |
```

```
    <module_variable_access>  ).
```

```
<module_variable_access>:: <VARIABLE_ACCESS>.
```

```
<interaction_point_variable>::
```

```
  <interaction_point_identifier>.
```

```
  (ou um elemento de um vetor de
```

```
<interaction_point_identifier>s).
```

```
<interaction_point_access>::
```

```
  <module_variable_access>  "."
```



```

    <interaction_point_variable>
    | <interaction_point_variable_access>.

```

```

<interaction_point_variable_access>::

```

```

    <VARIABLE_ACCESS>.

```

(utilizado para referenciar pontos de interação internos).

```

<output_statement>:: "output"

```

```

    ( <interaction_point_variable>      |
      <interaction_point_variable_access> ) "."

```

```

    <interaction_identifier>

```

```

    [ ACTUAL_PARAMETER_LIST ].

```

Os comandos estruturados <all\_statement> e <for\_one\_statement> permitem executar um comando (que pode ser composto) usando "bounded variables". Estas variáveis, cujo escopo é o próprio <STATEMENT> são definidas dentro de um tipo ou pertencentes a um conjunto. Para o <all\_statement>, <STATEMENT> é executado uma vez para cada combinação de valores dos domínios dos tipos da <all\_binding\_list>. No <for\_one\_statement>, o primeiro <STATEMENT> que aparece na construção é executado apenas uma vez para uma combinação de valores da <all\_binding\_list> que satisfaça a expressão booleana (pode ocorrer não determinismo). Se a expressão booleana não puder ser satisfeita por qualquer combinação, é executado o segundo <STATEMENT> ("otherwise") se estiver presente, caso contrário nada acontece.

```

<all_statement>::

```

```

    "all" <all_binding_list> "do" <STATEMENT>.

```

```

<for_one_statement>::

```

```

"forone" <all_binding_list>
"suchthat" <BOOLEAN_EXPRESSION>
"do" <STATEMENT>
[ "otherwise" <STATEMENT> ].

```

```

<all_binding_list>::
    <domain_list>
    | <identifier> "in" <EXPRESSION>.

<domain_list>::
    <IDENTIFIER_LIST> ":" <domain_type_all>
    { ";" <IDENTIFIER_LIST> ":"
      <domain_type_all> }.

<domain_type_all>::
    <ORDINAL_TYPE> | <header_type>.

```

A expressão <exist\_one> é acrescentada ao Pascal retornando "verdadeiro" ou "falso" dependendo da existência ou não de uma combinação sobre as variáveis da <all\_binding\_list> que satisfaça a expressão booleana.

```

<exist_one>:: "exist" <all_binding_list>
              "suchthat" <BOOLEAN_EXPRESSION>.

```

Um último comando, que convém incluir para completar o elenco, é o <nextstate\_statement>, usado para definir o próximo estado de uma transição, quando na <to\_list> da transição forem especificados mais de um estado.

```

<nexstate_statement>:: "nextstate"
                      ( "same" | <state_identifier> ).

```

### 3.2 LOTOS

A linguagem Lotos (Language Of Temporal Ordering Specification) está sendo desenvolvida pelo comitê técnico TC 97 da ISO, subcomitê SC 21, no subgrupo C do grupo de trabalho WG 1 para técnicas de descrição formal. A descrição aqui apresentada baseia-se no documento [ISO 87] em que a linguagem já se encontrava em estágio de "Draft International Standard".

Lotos foi desenvolvida centrada em dois modelos: o primeiro para descrição algébrica de tipos abstratos de dados [EHR 82], baseando-se na linguagem ACT ONE, e o outro, para descrição de comportamento através de sequência de eventos observáveis, baseado no Calculus on Communicating Systems (CCS) de Milner [Mil 80].

#### 3.2.1 Modelo

O modelo adotado para a linguagem LOTOS será apresentado em duas partes. A primeira é um modelo para os tipos abstratos de dados, e a segunda um modelo para a especificação do comportamento dinâmico dos processos.

##### 3.2.1.1 Modelo dos tipos de dados

O modelo matemático alvo de uma definição ou conjunto de definições de tipos de dados será uma álgebra polissortida (many-sorted algebra) ou simplesmente álgebra. Uma álgebra (polissortida) é frequentemente definida como uma dupla  $\langle D, O \rangle$ , onde  $D$  é um conjunto de portadores de dados (data carriers, DCs) e  $O$  é um conjunto de operações, mapeando o produto cartesiano dos portadores sobre um portador. Os portadores de dados são conjuntos de valores de dados (data values, dvs). Simbolicamente:

$$\begin{aligned}
 D &= \{DC1, DC2, \dots, DCn\} \\
 DC1 &= \{dv11, dv12, \dots\} \\
 DC2 &= \{dv21, dv22, \dots\} \\
 &\dots \quad \dots \quad \dots \quad \dots \\
 DCn &= \{dvn1, dvn2, \dots\} \\
 O &= \{OP1, OP2, \dots, OPm\} \\
 OP1: & DC1 \times DC2 \times \dots \times DCn \rightarrow DCj, \\
 & \quad \quad \quad \emptyset < i \leq m, \quad \emptyset < j \leq n.
 \end{aligned}$$

### 3.2.1.2 Modelo dos processos

O modelo proposto na definição de LOTOS [ISO 87] é o de um sistema de transições rotuladas, definido como uma quádrupla  $\langle S, A, T, So \rangle$ , onde:

$S$  é um conjunto de estados;

$A$  é um conjunto de ações;

$T$  é um conjunto de relações de transição, que contém exatamente uma relação, esquematicamente definida como "-a-", pertencente a  $S \times S$ , para cada  $a$ , pertencente a  $A$ ;

$So$  é o estado inicial.

A dinâmica do sistema de transições rotuladas acima é dada da seguinte forma: se o estado atual é  $S1$  pertencente a  $S$ , e existe uma transição  $\langle S1, S2 \rangle$  pertencente a  $T$ , associada à ação  $a$  ( $S1 -a- S2$ ), pode ocorrer uma transição de  $S1$  para  $S2$ , isto é, o estado atual pode deixar de ser  $S1$ , passando a ser  $S2$ , desde que haja participação do meio externo ou ambiente na ocorrência do evento.

### 3.2.2 Definição da sintaxe e semântica

Iniciemos com as definições dos dados em Lotos.  $\langle \text{data\_type\_definition} \rangle$  encerra a definição sintática de um tipo: uma porção da especificação que define uma álgebra de

dados. A álgebra de dados da especificação é formada pela união das álgebras associadas a cada definição de tipo. Um dos elementos básicos da definição do tipo são os sorts (`<sort_identifiers>`), que serão interpretados como nomes de portadores da álgebra. As operações da álgebra são definidas por um `<operation_descriptor>` (nome da operação), junto com sua funcionalidade (`<argument_list>` e `<result>`).

Para a interpretação da sintaxe que segue, definiremos inicialmente uma assinatura (signature) SIG como uma tupla (S, O) onde:

S é um conjunto finito de sorts (que serão interpretados como portadores);

O é um conjunto finito de símbolos de operações, com as respectivas funcionalidades.

```
<data_type_definition>::
    "type"          <type_identifier>
    "is"            <p_expression>
    "endtype"
  | "library"      <type_identifier>
                    { "," <type_identifier> }
    "endlib".
```

```
<p_expression>::
    <type_union> <p_specification>
  | <type_identifier>
    "actualizedby" <type_union>
    "using"        <replacement>
  | <type_identifier>
    "renamedby"   <replacement>.
```

```
<type_union>:: <type_identifier>
               [ "," <type_union> ].
```

```

<p_specification>::
    [ "formalsorts"    <sort_list>          ]
    [ "formalopns"    +{ <operation>        } ]
    [ "formaleqns"    <equation_lists>     ]
    [ "sorts"         <sort_list>          ]
    [ "opns"          +{ <operation>        } ]
    [ "eqns"          <equation_lists>     ].

```

Uma das alternativas para definição de um tipo é dada pela alternativa "<type\_union> <p\_specification>" na construção <p\_expression>. <type\_union> é uma lista de tipos previamente definidos. Esta cláusula faz com que todas as definições feitas em cada um destes tipos sejam "importadas" para esta definição, incluindo sorts, operações, etc. Na cláusula <p\_specification>, "sorts" define os novos sorts componentes do tipo, "opns" define as novas operações (vide derivação da cláusula <operation>, abaixo. O conjunto dos novos sorts com os importados pela cláusula <type\_union>, junto com as operações (novas e importadas), definirá a assinatura do tipo. Note que uma operação sem argumentos define um literal pertencente a um determinado tipo.

```

<operation>::
    <operation_descriptor>
        { "," <operation_descriptor> }
    ":" [ <argument_list> ]
    "->" <result>.

<operation_descriptor>::
    <operation_identifier>
    | "_" <operation_identifier> "_".

<argument_list>:: <sort_list>.

```

<result>:: <sort\_Identifier>.

Um termo básico (ground term) é definido recursivamente da seguinte forma sobre a assinatura: todo literal (operação sem argumentos) é um termo básico; todo termo obtido pela aplicação de uma operação sobre termos básicos já obtidos é termo básico. O sort de um termo básico é o sort da última operação aplicada para sua obtenção. A álgebra definida pela assinatura do tipo é dita gerada por termos (term generated), pois os componentes de cada portador são exatamente os termos básicos do sort associado, possíveis de serem gerados. Na figura 3.6 é apresentado um exemplo com uma definição parcial de um tipo com sorts e operações.

```

type EXEMPLO
is  sorts    NAT, BOL
    opns     0      :          -> NAT
           SUCC   : NAT      -> NAT
           +      : NAT, NAT -> NAT
           =      : NAT, NAT -> BOL
           TRUE   :          -> BOL
           FALSE  :          -> BOL

```

figura 3.6 Exemplo de definição da assinatura de um tipo: sorts e operações

São definidos dois sorts: NAT e BOL; e cinco operações, três das quais são literais: 0, TRUE e FALSE, uma unária SUCC (sucessor) e as operações binárias de soma: "+" e igualdade: "=". A assinatura correspondente seria < {NAT, BOL}, {0, SUCC, +, =, TRUE, FALSE} >, onde subentende-se a

funcionalidade de cada operador definida acima. Alguns dos possíveis termos gerados são:

Termos de sort NAT:  $0$ ,  $SUCC(0)$ ,  $SUCC(SUCC(0))$ ,  $+(0, 0)$ ,  $+(0, SUCC(0))$ ,  $SUCC(+ (SUCC(0), 0))$ , etc.

Termos de sort BOL: TRUE, FALSE,  $=(0, 0)$ ,  $=(SUCC(0), SUCC(0))$ ,  $=(0, +(SUCC(0), SUCC(0)))$ , etc.

A álgebra gerada pelos termos acima é aquela em que os portadores correspondentes aos sorts NAT e BOL contém, respectivamente, todos os infinitos termos de sort NAT e todos termos de sort BOL. Ora, do nosso conhecimento dos conjuntos de números naturais e de valores lógicos clássicos, este último sabidamente um conjunto de dois elementos, vemos que falta ainda uma informação que agrupe os valores segundo uma relação de equivalência entre eles (que divida, por exemplo, os valores lógicos em apenas duas classes). Esta informação é dada pela opção "eqns" `<equation_lists>`, em `<p_specification>`.

```

<equation_lists>::
  [ "forall" <identifier_declarations> ]
  +{ "ofsort" <sort_identifier>
    [ "forall" <identifier_declarations> ]
    <equation> { ";" <equation> }
    { ";" }
  }.

<equation>:: [ <premisses> "=>" ]
              <simple_equation>.

<premisses>:: <premiss> { "," <premiss> }.

<premiss>:: <simple_equation> |
            <boolean_expression>.

<simple_equation>::

```



`<value_expression> "=" <value_expression>.`

`<boolean_expression>:: <value_expression>.`

As duas construções "forall" `<identifier_declarations>` permitem a declaração de variáveis de diferentes sorts para uso na definição das equações. A diferença entre as duas é apenas o escopo de aplicação. "ofsort" `<sort_identifier>` identifica o sort a que pertencem os pares de expressões de cada equação que vai ser definida. A construção mais simples em `<equation>` é uma `<simple_equation>`, que consiste na definição de igualdade entre duas expressões. Diferente dos termos básicos vistos acima, estas expressões podem conter variáveis declaradas, citadas acima ("forall"), definindo na verdade "esquemas" de equações. Serão chamadas instâncias básicas de uma definição de equação E, todas as equações possíveis de serem obtidas pela substituição, em E, de todas as variáveis por termos básicos de sort correspondente. Assim, cada equação definida acima, define um conjunto possivelmente infinito de instâncias básicas de equação que integrarão o tipo sendo definido. A outra construção de `<equation>`, incluindo `<premisses>`, permite incluir novas instâncias básicas de equação, por inferência, do seguinte modo: se houver uma possibilidade de substituição de variáveis por termos básicos em uma `<equation>`, tal que, cada uma das premissas (`<premiss>`) se torne uma instância básica de equação já incluída no tipo, então, também é incluída a instância obtida pela substituição na `<simple_equation>` do lado direito de "`=>`" em `<equation>`. Completa-se, na figura 3.7, o exemplo iniciado anteriormente, na figura 3.6, com a inclusão da definição de algumas equações.

```

eqns forall  X, Y, Z: NAT,

ofsort  NAT
+ (0, 0)      = 0
+ (X, SUCC (Y)) = SUCC ( + (X,Y))
+ (X,Y)      = + (Y,X)
X = Y        => SUCC (X) = SUCC (Y)

ofsort  BOL
= (0, 0)      = TRUE
= (0, SUCC (X)) = FALSE
= (SUCC (X), SUCC (Y)) = = (X,Y)
X = Y        => = (X,Z) = = (Y,Z)
= (X, Y)     = = (Y, X)

```

figura 3.7 Definição das equações de tipo

As equações acima definidas, permitem obter como instâncias básicas de equação, entre outras:

Instâncias do tipo NAT:

```

+ (0, SUCC (0)) = SUCC (+ (0, 0))
+ (0, 0) = 0

```

SUCC (+ (0, 0) = SUCC (0)), obtida da quarta equação, substituindo de acordo com a segunda instância encontrada.

Instâncias do tipo BOL:

```

= (0, 0) = TRUE
= (SUCC (0), SUCC (0)) = = (0, 0)
= (SUCC (SUCC (0)), SUCC (SUCC (0))) =
= (SUCC (0), SUCC (0)).

```

Se considerarmos o fechamento reflexivo e transitivo da relação dada pelo conjunto de equações básicas define-se uma relação de equivalência entre os termos

básicos. Estão na mesma classe todos os termos que estiverem relacionados por uma instância de uma equação. Se tomarmos a álgebra gerada por termos vista acima e particionarmos os portadores segundo as relações de equivalência definidas para cada sort, teremos uma álgebra quociente, em que cada elemento de um portador é uma classe de equivalência da relação associada ao sort, aplicada ao portador deste sort da álgebra gerada por termos. Esta é a álgebra associada à definição do tipo. Alguns componentes de portadores da álgebra quociente do exemplo seriam: [ SUCC (SUCC (0)), + (SUCC (0), SUCC (0)), + (0, SUCC (+ (SUCC (0), 0))), ...], de sort NAT, e [ TRUE, = (0, 0), = (SUCC (0), SUCC (0)), = (+ (0,0), 0), ...], de sort BOL, onde [...] encerra uma classe de equivalência.

A tupla  $\langle S, O, E \rangle$ , onde  $\langle S, O \rangle$  é a assinatura de um tipo e  $E$  é seu conjunto de equações originais (com variáveis), é conhecida como apresentação (presentation) do tipo. É uma parte puramente sintática (assim como a assinatura) da especificação.

As cláusulas "formalsorts", "formalopns" e "formaleqns", em  $\langle p\_specification \rangle$ , acima, permitem definir tipos parametrizáveis. Um exemplo típico é o de definição de estruturas de dados compostas como filas, pilhas, árvores, cujos elementos podem ser de diversos sorts. Neste caso, poder-se-ia defini-las apenas uma vez, parametrizáveis e posteriormente definir instâncias para o sort de elemento desejado. Em "formalsorts" são definidos nomes dos "parâmetros-sort"; em "formalopns" são definidas operações, cujos identificadores são parâmetros, e em cujas funcionalidades comparecem apenas os sorts definidos como parâmetros. Em "formaleqns" da mesma forma só comparecem parâmetros (sorts ou equações). Voltando às definições anteriores de "sorts", "opns" e "eqns": as operações podem

incluir parâmetros-sort e as equações podem incluir tanto parâmetros-sort como parâmetros-operações. Chamando de FS, FO e FE, respectivamente, os conjuntos de definições formais de parâmetros sort, operações e equações; e chamando de S, O e E, respectivamente os conjuntos de definições reais de sorts, operações e equações (que podem conter parâmetros na definição), podemos imaginar que o resultado sintático da definição de um tipo é uma apresentação parametrizável  $\langle \text{ppres} \rangle = \langle \text{fpres}, \text{tpres} \rangle$ , onde  $\text{fpres} = \langle \text{FS}, \text{FO}, \text{FE} \rangle$  (apresentação formal) e  $\text{tpres} = \langle \text{FS} \cup \text{S}, \text{FO} \cup \text{O}, \text{FE} \cup \text{E} \rangle$  (apresentação alvo). Uma apresentação é dita completa quando  $\text{fpres} = \langle \emptyset, \emptyset, \emptyset \rangle$ , isto é, quando não houverem parâmetros "em aberto". Na especificação do comportamento, poderão ser referenciados sorts e operações apenas de tipos cuja apresentação seja completa.

Mostra-se agora como é feita a substituição dos parâmetros, utilizando a opção "actualizedby" de  $\langle \text{p\_expression} \rangle$ . Nesta opção,  $\langle \text{type\_identifier} \rangle$  é o nome do tipo que vai ser parametrizado. São "importadas" as apresentações parametrizáveis dos tipos em  $\langle \text{type\_union} \rangle$  gerando uma nova ppres, e são acrescentadas à nova apresentação alvo (tpres, componente de ppres) o resultado da substituição, na tpres do tipo parametrizável, dos parâmetros formais, pelos sorts e equações reais, de acordo com a cláusula  $\langle \text{replacement} \rangle$  apresentada abaixo.

```

<replacement>::
    [ "sortnames"
      +{ <sort_identifier>
        "for" <sort_identifier>      } ]
    [ "opnames"
      +{ <operation_identifier>
        "for" <operation_identifier> } ].

```

Há ainda uma opção ("renamedby", em <p\_expression>) de criar um tipo a partir de outro, modificando nomes de sorts e equações.

A seguir é apresentada a definição da parte comportamental de LOTOS. A especificação de um sistema em Lotos consiste na definição das possíveis sequências de eventos observáveis externamente. A observação da ocorrência de eventos se dá através de portas, definidas no sistema. Na especificação, em Lotos, da possibilidade de ocorrência de um evento, associada a uma porta, é especificada uma forma de participação na ocorrência do evento. Para que o evento efetivamente ocorra, é necessária uma participação também do meio externo na ocorrência, de forma convergente. Adiante serão vistas as duas formas de participação na ocorrência de eventos (geração e aceitação de valor), bem como a interpretação das possíveis combinações de participação em eventos por parte do sistema especificado e do meio externo (comunicação síncrona, sincronização e geração não determinística de valor).

A definição de um sistema começa com a cláusula <specification>, que deriva em um identificador para o sistema, uma lista de parâmetros formais, possivelmente um conjunto de definições globais de tipos de dados, vistas anteriormente, e a definição do comportamento propriamente dito (<definition\_block>).

```
<specification>::
    "specification" <specification_identifier>
                    <formal_parameter_list>
                    { <data_type_definition>      }
    "behaviour"    <definition_block>
    "endspec".
```

`<formal_parameter_list>` define o conjunto de portas para interação com o meio externo (`<gate_parameter_list>`), um conjunto de identificadores (`<value_parameter_list>`) que permitem tornar a especificação parametrizável, e a funcionalidade ("exit" ou "noexit") vista adiante.

```

<formal_parameter_list>::
    [ <gate_parameter_list> ]
    [ <value_parameter_list> ]
    ":" ( "exit" [ "(" <sort_list> ")" ] |
        "noexit" ).

<gate_parameter_list>:: <gate_tuple>.

<value_parameter_list>::
    "(" <identifier_declarations> ")".

```

`<definition_block>` tem como objetivo definir as seqüências de eventos possíveis através da `<behaviour_expression>`. Estas seqüências, de modo geral, contêm um número de eventos indeterminado, fazendo-se necessária a existência de uma facilidade que permita a definição recursiva de seqüências parciais de eventos. A construção `<process_definition>` tem dupla finalidade: permitir uma melhor estruturação da definição das seqüências de eventos e permitir a definição recursiva de seqüências. Uma `<process_definition>` encerra uma definição de seqüência de eventos como em `<specification>`. Quando um identificador de uma `<process_definition>` declarada na opção "where" for referenciado como parte de uma seqüência de eventos na `<behaviour_expression>`, a interpretação é a seguinte: a

continuação da definição da sequência de eventos prossegue conforme especificado na <behaviour\_expression> da <process\_definition> referenciada. Note-se que, em <gate\_parameter\_list>, não está sendo definida uma lista de novas portas, mas sim, uma lista de identificadores, que serão associados a portas, que serão passadas como parâmetros, quando a <process\_definition> for invocada.

```
<definition_block>:: <behaviour_expression>
    [ "where" +{ <data_type_definition>
                | <process_definition> } ].
```

```
<process_definition>::
    "process"      <process_identifier>
                  <formal_parameter_list>
    "=="          <definition_block>
    "endproc".
```

Em uma construção <definition\_block> poderão comparecer diversas definições de tipos de dados locais e <process\_definitions> que podem ser referenciadas na <behaviour\_expression>, podendo-se referenciar mutuamente.

Ainda cabe salientar que tem-se evitado o uso da palavra processo pois <process\_definition> não é uma definição de processo na conotação usual de sistemas operacionais, como um tipo de entidade concorrente que pode ser posta em execução.

E analisada, a seguir, a definição das sequências de eventos, iniciando pelos casos mais simples. As referências ao meta\_símbolo <behaviour\_expression> serão abreviadas por beh.

```

<behaviour_expression>:: <beh_lev_7>.

<beh_lev_7>::
    <beh_lev_6>
    | <hiding_expression>
    | <general_parallel_expression>
    | <sum_expression>
    | <local_definition_expression>.

<beh_lev_6>:: <beh_lev_5> | <enable_expression>.

<beh_lev_5>:: <beh_lev_4> | <disable_expression>.

<beh_lev_4>:: <beh_lev_3> | <parallel_expression>.

<beh_lev_3>:: <beh_lev_2> |
    <general_choice_expression>.

<beh_lev_2>:: <beh_lev_1> | <guarded_expression>.

<beh_lev_1>:: <beh_lev_0> |
    <action_prefix_expression>.

<beh_lev_0>:: <atomic_expression>.

```

Uma beh pode ser derivada em <beh\_lev\_1>, que por sua vez deriva em uma sequência de número indefinido de <action\_denotation>, separados por ";", terminada (a sequência) por uma <atomic\_expression> (<beh\_lev\_0>). Cada <action\_denotation> define uma participação em ocorrência de evento, vista adiante. ";" é o operador para especificação de sequência de eventos. <beh\_lev\_1> é interpretada, portanto, como a especificação de uma sequência de participações em eventos, terminada por uma <atomic\_expression>. <atomic\_expression>, pode derivar em uma referência a um <process\_identifier>, indicando que a sequência de eventos continua na <process\_definition>



referenciada, parametrizada pela lista de portas e de valores.

```

<action_prefix_expression>::
    <action_denotation> ";" <beh_lev_1>.

<atomic_expression>::
    "stop"
  | "exit" [ "(" +{ <exit_parameter> } "]" ]
  | <process_identifier>
      { <gate_tuple> }
      { <value_expression_list> }
  | "(" <beh_lev_7> ".

<exit_parameter>::    <value_expression>
                    | "any" <sort_identifier>.

```

A derivação de <atomic\_expression> em "stop" indica aborto da definição da sequência de eventos e tem funcionalidade "noexit". Uma beh definida como "exit", deve ter especificada uma funcionalidade consistente com os sorts dos valores derivados nos <exit\_parameter>. O significado da sequência "exit" é de término normal da sequência de eventos. Ver-se-á, mais adiante, o uso dos <exit\_parameter> e a diferença efetiva entre os termos por "stop" ou "exit", junto com a funcionalidade.

A opção "(" <beh\_lev\_7> ")" permite especificar <behaviour\_expression> de forma embutida.

Conforme dito anteriormente, <action\_denotation> define a participação do sistema na ocorrência de um evento. "I" representa um evento interno. A especificação de "I" em <action\_denotation>, indica que o evento pode ocorrer, dentro da sequência, sem a participação do meio externo,

como indica seu próprio nome. É um elemento importante na definição de escolhas não determinísticas de sequências, vistas adiante. A outra opção é a de referenciar uma porta, através da qual o sistema interagirá com o meio na ocorrência do evento. Os dois elementos que definem a forma de participação na ocorrência do evento são "!" e "?", aqui chamados respectivamente de geração e aceitação de valor. Analisemos, inicialmente, a presença de apenas um `<experiment_offer>`. Se este for definido como uma geração ("!") de valor, poderá haver a efetiva ocorrência do evento se o meio externo participar com uma entre as seguintes formas:

a) com uma geração, de um mesmo valor que o avaliado na `<value_expression>`. Nesta caso, diz-se que houve uma ação ou evento de sincronismo;

b) com uma aceitação de valor, através de uma variável de mesmo sort do da `<value_expression>`. Neste caso, diz-se que houve um evento de comunicação, em que o valor gerado pelo sistema foi enviado ao meio externo.

Se o `<experiment_offer>` for definido como uma aceitação ("?") de valor, poderá haver a ocorrência do evento com a participação do meio em uma das formas abaixo (será considerado o caso em que `<identifier_declaration>` só contém um identificador de variável. A ocorrência de uma sequência de identificadores declarados pode ser desdobrada, com significado equivalente, em uma sequência de `<experiment_offer>` do tipo "?", uma para cada identificador):

a) com uma geração de valor, de mesmo sort do identificador declarado em `<identifier_declaration>`. Analogamente ao caso anterior, houve uma comunicação de valor, porém do meio para o sistema. No prosseguimento da

especificação da sequência de eventos, no escopo da <beh\_lev\_1> que segue a <action\_denotation>, o identificador aqui declarado pode ser usado, sendo interpretado como o valor recebido durante a ocorrência do evento.

b) com uma aceitação de valor, de mesmo sort do identificador declarado em <identifier\_declaration>. Neste caso, diz-se que houve uma geração não determinística de valor, que será apropriada pelo identificador do sistema e pelo meio externo. Valem as mesmas observações sobre utilização do valor no prosseguimento da sequência feitas no parágrafo anterior.

```

<action_denotation>::
    "I"
    | <gate_identifier>
      [ +{ <experiment_offer> } [ <guard> ] ].

<experiment_offer>:: "?" <identifier_declaration>
                      | "!" <value_expression>.

<guard>:: "[ " <premiss> "]" .

```

Diz-se que ocorre um casamento ("matching") entre duas participações de eventos, uma do sistema e outra do meio, se as participações forem tais que permitem a ocorrência do evento, conforme as regras acima apresentadas. Supondo-se a existência de vários <experiment\_offer>, considerando-se o desdobramento para "um identificador por <identifier\_declaration>", entende-se que o sistema está participando da transição através de uma tupla ordenada composta por gerações e aceitações de valor. A transição poderá ocorrer, se o meio externo puder participar desta com uma tupla de mesmo número de elementos, e houver um

casamento ("matching") entre cada par de elementos de mesma posição das duas tuplas. O resultado será uma mistura de trocas e gerações não determinísticas de valor ou simplesmente sincronismo.

A presença de uma guarda (<guard>), definida por uma equação (<premiss>, vista na especificação dos tipos de dados), envolvendo possivelmente os valores e identificadores trocados nas <experiment\_offer>, impõe uma restrição adicional à ocorrência do evento especificado. Só poderá ocorrer o evento, se ao final da ocorrência a guarda estiver satisfeita.

Analisando <beh\_lev\_2> vemos que uma <beh\_lev\_1>, cujas derivações foram vistas nos parágrafos anteriores, pode ser precedida por várias instâncias da construção '<guard> "->" ' em <guarded\_expression>. A ocorrência do primeiro evento de <beh\_lev\_1> só poderá se dar quando todas as guardas estiverem satisfeitas.

```
<guarded_expression>:: <guard> "->" <beh_lev_2>.
```

```
<choice_expression>::
```

```
<beh_lev_2> "[]" <beh_lev_3>.
```

Em <beh\_lev\_3> permite-se, pela aplicação recursiva de <choice\_expression>, obter um conjunto de expressões derivadas de <beh\_lev\_2>, separadas por "[]" (operador de alternativa ou "choice"). A interpretação desta construção é que, existe um conjunto de alternativas de possíveis sequências de ocorrência de eventos. A ocorrência de um evento implica na escolha da alternativa. Na figura 3.8 são apresentados alguns casos. Considere que a, b e c são especificações de eventos e B é qualquer continuação

válida de sequência de eventos. A possibilidade de parentização é aquela vista em <atomic\_expression>.

Exemplo 1: a; (b; B [] c; B).

Exemplo 2: (a; b; B [] a; c; B).

Exemplo 3: I; (b; B [] c; B)

Exemplo 4: (I; b; B [] I; c; B)

figura 3.8 Exemplos de utilização do operador "[]"

No exemplo 1, após a ocorrência do evento a, pode ocorrer ou a sequência iniciada por b ou a iniciada por c. O exemplo 2 apresenta uma sutil diferença: quando a execução do evento a for iniciada, é escolhida, não deterministicamente, uma das opções de sequência possível. A partir de então, dependendo da escolha, apenas uma alternativa de evento pode ocorrer: b ou c. Suponha que o meio externo nunca participe da ocorrência do evento b, e esteja sempre disposto a cooperar com o c (digamos, aceitação de valor numa porta). O exemplo 1 descreve um sistema que não teria bloqueio para chegar na sequência dada por B. O exemplo 2 descreve um sistema em que poderia ocorrer bloqueio, com a espera pelo evento b.

Os exemplos 3 e 4 são semelhantes aos anteriores, enfatizando o uso do evento interno (I) na descrição de escolhas não determinísticas. A interpretação deve ser: existe um evento ou conjunto de eventos que não estão descritos (não podem ser observados pelo meio externo) neste nível de especificação, que influirão na escolha da alternativa de sequência de ocorrência dos outros eventos.

A derivação de <beh\_lev\_4> permite ter

especificadas várias derivações de <beh\_lev\_3> separadas por operadores de concorrência (<parallel\_operator>). A interpretação é de que existem várias sequências de eventos que podem ocorrer concorrentemente. As restrições à concorrência são impostas na especificação de <parallel\_operator>. O caso mais simples é o de entrelaçamento (interleaving) de sequências de eventos (operador |||). Neste caso, considera-se que as sequências de eventos ocorrem independentemente, sincronizando apenas com o meio externo, com a restrição imposta pelo modelo de concorrência conhecido como entrelaçamento: os eventos das diferentes sequências ocorrem um de cada vez, não há simultaneidade na ocorrência de eventos de sequências diferentes. A ocorrência dos eventos ocorre entrelaçada.

Outro caso extremo ocorre com o uso do operador de sincronização (||). Neste caso assume-se que a ocorrência de qualquer evento (exceto evento interno) implica na participação de todas as sequências descritas em paralelo na ocorrência do evento, além do meio externo. Na prática, isto significa que, no mínimo, as sequências de eventos terão que conter a mesma sequência de portas e listas de <experiment\_offer> compatíveis (número de elementos, sort). A interpretação da ocorrência do evento assume diversas formas: comunicação entre sequências de eventos, difusão (broadcast), etc.

A opção ' "|[" [ <gate\_identifier\_list> ] "|]" ' expressa o caso geral, em que são especificadas as portas (dentre as existentes) para as quais deverá haver participação das duas sequências. Os dois extremos deste caso são o entrelaçamento (<gate\_identifier\_list> vazia) e a sincronização (<gate\_identifier\_list> contém todas as portas existentes).

Pode-se, na descrição das sequências em paralelo,

utilizar diferentes operadores de paralelismo, incluindo diferentes portas no caso geral. A interpretação é a seguinte: considera-se os operadores de paralelismo como operadores binários, sobre duas expressões, e assume-se que: se houver uma maneira de parentizar a expressão para que um dado evento possa ocorrer, então ele pode ocorrer. No exemplo abaixo, g1, g2, g3 são três portas e B1, B2, B3, três expressões derivadas de <beh\_lev\_3>. Considera-se sempre, implicitamente, a necessidade de participação do meio externo na ocorrência dos eventos.

Exemplo: B1 [| g1, g2 |] B2 [| g2, g3 |] B3.

Pode ocorrer um evento na porta g3, apenas com a participação de B1: " B1 [| g1, g2 |] (B2 ... B3) ". Para que possa ocorrer um evento na porta g1, com a participação de B1 é necessária também a participação de uma das outras duas expressões (B2 ou B3). No entanto, um evento pode ocorrer, na porta g1, apenas com a participação de B3: " (B1 ... B2) [| g2, g3 |] B3 ". Para a ocorrência de um evento em g3 com a participação de B3, é necessária a participação de B1 ou B2. Finalmente, para que ocorra um evento na porta g2, é necessária a participação das três expressões. Salieta-se, novamente, que foi omitida a participação do meio, que é necessária sempre (exceto para os eventos internos, que aliás, nunca são sincronizados, e para as portas definidas pela construção <hiding\_expression>, vista no final).

```
<parallel_expression>::
    <beh_lev_3> <parallel_operator> <beh_lev_4>.

<parallel_operator>::
    |   "||"
    |   "|||"
```

```
|   "[[" [ <gate_identifier_list> ] "]"|".
```

<beh\_lev\_5> permite (vide <disable\_expression>) derivar expressões do tipo " B1 [> B2 [> ... [> Bn ". onde B1, B2, ..., Bn são derivações de <beh\_lev\_4>, vistas nos parágrafos acima. A interpretação é a seguinte: Cada uma das expressões B<sub>i</sub> tem especificado implicitamente um evento inicial (ou vários, se forem usados operadores de paralelismo). Enquanto forem ocorrendo eventos da sequência definida por B<sub>i</sub>, ou antes de qualquer ocorrência de evento, as outras expressões B<sub>i</sub>, 1 < i ≤ n estão habilitadas também a participar da ocorrência de seus eventos iniciais. No momento em que ocorrer efetivamente um evento de uma sequência B<sub>i</sub>, considera-se que estão definitivamente desabilitadas as ocorrências de eventos das expressões B<sub>j</sub>, com j < i. Isto é, a ocorrência de um evento inicial de uma expressão interrompe ou aborta as sequências à sua esquerda. Se, ocorrer o evento definido por "exit", considera-se desabilitadas todas as expressões B<sub>i</sub>.

```
<disable_expression>::
```

```
    <beh_lev_4> "[>" <beh_lev_5>.
```

```
<enable_expression>::
```

```
    <beh_lev_5> ">>"
```

```
    [ "accept" <identifier_declarations> "in" ]
```

```
    <beh_lev_6>.
```

Até agora foram definidos operadores que iniciam descrição de sequências alternativas ou concorrentes de eventos (com ou sem desabilitação). Note-se que é impossível, até o momento, encerrar estas sequências e



prosseguir em um caminho único. Do mesmo modo, é impossível, após a inclusão de uma subsequência de eventos pela chamada de uma `<process_definition>`, continuar a definição da sequência global. Para isto define-se na construção `<beh_lev_6>` a possibilidade de expressar uma "composição sequencial" de processos. `<beh_lev_6>` deriva em uma sequência de expressões derivadas de `<beh_lev_5>`, separadas por ">>" utilizando, possivelmente, a opção "accept". A interpretação é a seguinte: após terminada a sequência de eventos da primeira (mais a esquerda) expressão delimitada por ">>", a descrição da sequência global de eventos continua na segunda expressão, e assim por diante. Na passagem de uma expressão para a outra, é possível a passagem de parâmetros, especificados na opção "accept". Foram postergadas as definições de término de sequências e de funcionalidade, que serão dadas a seguir para o correto entendimento da composição sequencial.

Analizemos o término e a funcionalidade de expressões derivadas de `<beh_lev_2>`. Uma sequência de eventos acabada por "stop" aborta a sequência de eventos global da qual participa. Atribui-se a esta sequência a funcionalidade "noexit". Uma sequência de eventos acabada por "exit", com ou sem parâmetros, termina e tem funcionalidade igual a tupla de sorts correspondentes a cada `<exit_parameter>`. A última possibilidade de acabar uma sequência de eventos é pela referência a um `<process_identifier>`. Neste caso, a condição de terminação e a funcionalidade são as da `<process_definition>` referenciada. Note-se que a terminação efetiva ocorre sempre através de um "exit". Este é considerado um evento que gera valores de acordo com sua funcionalidade para uso na composição sequencial. Na composição sequencial, a funcionalidade da expressão da esquerda, tem que ser compatível com a definição dos identificadores na opção

"accept".

A funcionalidade de uma expressão contendo alternativas de sequência (<beh\_lev\_3>) é definida se todas as funcionalidades das sequências forem iguais, exceto possivelmente algumas que tenham funcionalidade "noexit". Neste caso a funcionalidade é a funcionalidade idêntica das sequências, podendo, inclusive, ser "noexit", se todas as alternativas assim forem.

Na definição de sequências em paralelo, se uma das sequências tiver funcionalidade "noexit", então, a expressão inteira tem funcionalidade "noexit". Isto pode ser interpretado da seguinte forma: se um dos braços de sequências paralelas não termina ("stop") então não há possibilidade de recompor sequencialmente a sequência de eventos. Se todos os braços tiverem funcionalidade diferente de "noexit", então todos devem ter a mesma funcionalidade. No término de uma expressão com várias sequências paralelas, considera-se que todas elas participam juntas do evento "exit". Isto implica na passagem de uma sequência "combinada" de valores para a expressão seguinte, na composição sequencial.

A definição da funcionalidade envolvendo <disable\_expression> é análoga a da <choice\_expression>. Se as duas funcionalidades forem diferentes de "noexit", elas devem ser iguais identificando a funcionalidade da expressão. Se a funcionalidade de uma delas for "noexit", a funcionalidade resultante é a da outra. A funcionalidade de uma composição sequencial é a funcionalidade da última expressão da composição.

<beh\_lev\_7> permite derivações entrelaçadas de quatro tipos de construções, seguidas ao final de uma <beh\_lev\_7>. São elas: <hiding\_expression>.

<general\_parallel\_expression>, <general\_choice\_expression> e <local\_definition\_expression>.

A construção <local\_definition\_expression> permite definir identificadores como sinônimos para expressões contendo operadores e variáveis previamente definidos, para posterior uso. A funcionalidade é a da expressão seguinte a palavra "in".

```
<local_definition_expression>::
    "let"    <identifier_equations>
    "in"     <beh_lev_7>.
```

```
<identifier_equations>::
    <identifier_equation>
    { ",", <identifier_equation> }.
```

```
<identifier_equation>:
    <identifier_declaration>
    "=" <value_expression>.
```

<general\_parallel\_expression> permite iniciar um conjunto finito de sequências, obtidas a partir de uma sequência parametrizável <beh\_lev\_7> em <general\_parallel\_expression>, gerando todas as combinações possíveis de substituição em <beh\_lev\_7> de cada identificador declarado em uma <gate\_identifier\_list> por portas contidas na <gate\_tuple> associada. A funcionalidade é a da própria <beh\_lev\_7>.

```
<general_parallel_expression>::
    "par"    <gate_declarations>
            <parallel_operator> <beh_lev_7>.
```

```

<gate_declarations>::
    <gate_declaration>
    { "," <gate_declaration>      }.

<gate_declaration>::
    <gate_identifier_list> "in" <gate_tuple>.

<gate_tuple>:: "[" <gate_identifier_list> "]" .

```

Uma <general\_choice\_expression> tem duas opções: uma é semelhante à <general\_parallel\_expression>, porém, gerando sequências alternativas de eventos. Na outra opção, são declarados identificadores, que podem ser substituídos, dentro da <beh\_lev\_7>, por qualquer valor do sort especificado, gerando possivelmente infinitas alternativas de sequências de eventos. A funcionalidade é a de <beh\_lev\_7>.

```

<general_choice_expression>::
    "choice" ( <identifier_declarations>
              | <gate_declarations>      )
    "[]" <beh_lev_7>.

<hiding_expression>::
    "hide" <gate_identifier_list>
    "in" <beh_lev_7>.

```

A construção <hiding\_expression> permite definir uma nova lista de portas, que nada tem a ver com o meio externo, não sendo, portanto, observáveis, cuja única função é sincronizar subsequências em paralelo sob seu escopo. Note-se que toda expressão sob o escopo da <hiding\_expression> pode fazer menção a estas portas, e neste caso não há participação do meio externo na ocorrência

do evento, diferente da maneira generalizada como se tinha considerado até este parágrafo. A funcionalidade é a de <beh\_lev\_7>.

A funcionalidade constante da <formal\_parameter\_list> de uma <process\_definition> ou <specification> deve corresponder à funcionalidade de fato da <behaviour\_expression>.

A seguir estão as definições de algumas construções genéricas, que foram utilizadas neste item.

```

<value_expression>::
    [ <value_expression>
      <operation_identifier>      ]
    <simple_expression>.

<simple_expression>::
    <term_expression>
    [ "of" <sort_identifier>      ].

<term_expression>::
    <operation_identifier>
    [ <value_expression_list>    ]
    | <value_identifier>
    | "(" <value_expression> ")".

<gate_tuple>:: "[" <gate_identifier_list> "]"

<identifier_declarations>::
    <identifier_declaration>
    { "," <identifier_declaration> }.

<identifier_declaration>::
    <value_identifier_list>
    ":" <sort_identifier>.

<sort_list>:: +{ <sort_identifier> }.
  
```

<gate\_identifier\_list>:: +{ <gate\_identifier> }.  
<value\_identifier\_list>:: +{ <value\_identifier> }.

<specification\_identifier>:: <identifier>.

<process\_identifier>:: <identifier>.

<type\_identifier>:: <identifier>.

<sort\_identifier>:: <identifier>.

<gate\_identifier>:: <identifier>.

<value\_identifier>:: <identifier>.

<operation\_identifier>:: <identifier>.

## 4 ESTUDO COMPARATIVO DAS LINGUAGENS

Na seção 4.1 é descrito um conjunto de parâmetros para comparação de linguagens de especificação de protocolos, alguns deles constituindo capacidades funcionais cuja importância se destaca na especificação de protocolos; os outros são parâmetros gerais de avaliação de linguagens formais de especificação aplicadas a protocolos, vistos no capítulo 1. A seguir, na seção 4.2, é apresentada uma comparação das duas linguagens, Estelle e Lotos, apresentadas no capítulo 3.

### 4.1 Definição dos parâmetros

#### 4.1.1 Concorrência

Concorrência descreve a característica de duas ou mais sequências de ocorrência de eventos se sobreporem no tempo (as sequências como um todo). Cada sequência de eventos é comumente chamada um processo. A concorrência é uma característica importante em especificação de protocolos, porque as entidades constituem-se em processos independentes que interagem.

Existem dois modelos básicos de concorrência: paralelismo e entrelaçamento. Paralelismo ocorre quando os eventos das diferentes sequências podem sobrepor-se no tempo. As sequências são totalmente independentes. E decorrencia deste modelo que a ocorrência dos eventos não é instantânea. Entrelaçamento é o modelo de concorrência em que se assume que nunca há ocorrência simultânea de eventos. Não é especificada a ordem relativa de ocorrência entre cada dois eventos de sequências concorrentes: qualquer um pode ocorrer antes do outro, mas não simultaneamente. Este modelo pode estar associado ao de que a ocorrência de eventos é

instantânea, atômica (nada afeta o resultado da ocorrência de um evento, uma vez disparado), ou a um modelo de exclusão mútua no acesso a recursos.

#### 4.1.2 Comunicação

A comunicação é a característica mais forte na especificação de protocolos, pois estes contêm basicamente a definição das sequências de troca de mensagens entre entidades para interconexão de sistemas. Os modelos básicos de comunicação são a comunicação síncrona e a comunicação assíncrona. Na comunicação síncrona entre processos, assume-se que o envio de dados ocorre simultaneamente ao recebimento pelo processo destinatário. Na comunicação assíncrona, o envio de dados é independente do recebimento. A comunicação assíncrona pode dar-se pelo envio de mensagens ou por compartilhamento de áreas. A comunicação síncrona pode ser feita via atrelamento de eventos ou "rendez-vous", esta última, um envio de mensagem com espera de confirmação. A comunicação pode ser entre pares de processo ou difusão.

Na especificação de protocolos, pode-se distinguir três tipos de comunicação: a comunicação entre entidades pares, abstraindo o modelo de serviço subjacente, a comunicação entre entidades de níveis adjacentes, ou de uma entidade com o provedor de serviço subjacente, e finalmente a comunicação entre entidades de mesmo nível e mesmo sistema que cooperam para a execução do serviço da camada.

#### 4.1.3 Sincronização

Sincronização é uma imposição feita a dois ou mais processos concorrentes para que executem um mesmo evento, ou uma t-upla de eventos associados (um evento de cada processo) simultaneamente, isto é, um processo não pode executar seu evento enquanto o outro não estiver em



condições de o fazer. A comunicação síncrona, além de comunicação é uma atividade de sincronismo entre processos. A sincronização pode portanto ser vista como uma característica semelhante à comunicação onde não há passagem efetiva de informação além da própria entrada em sincronismo.

#### 4.1.4 Não determinismo

Um sistema pode ter como característica a existência de situações em que há mais de uma possibilidade de continuação da sequência de eventos. Esta característica é chamada não determinismo, uma vez que não é totalmente determinado o comportamento do sistema, seja em função das interações com o meio, seja em função do seu próprio estado. Esta característica é importante em uma especificação para atingir diferentes níveis de especificação, bem como para a correta especificação de funcionamento das próprias implementações (que podem conter escolhas não determinísticas). Um caso particular de não determinismo é aquele ditado pelas diferentes sequências de eventos em processos concorrentes.

#### 4.1.5 Imparcialidade ("fairness")

Uma característica que frequentemente é relacionada com o não determinismo é a imparcialidade, que indica que não deve haver favorecimento na escolha entre as alternativas, ou no mínimo, não deve haver postergação indefinida de nenhuma delas, no caso de repetir-se várias vezes a mesma situação.

#### 4.1.6 Tempo

O conceito de tempo pode ocorrer de duas formas em

um sistema ou especificação: relacionado à ordem temporal de ocorrência de eventos ou com um caráter preciso, como uma quantidade bem definida de unidades de tempo. Os requisitos de tempo, quando colocados de forma precisa, constituem um forte fator de rebaixamento do nível de especificação. Vejamos um exemplo: a transmissão de uma sequência de bits pode ser modelada, em um nível de especificação razoável, como uma sequência de pares de eventos sincronizados (transmissor e receptor). Porém, uma especificação pode, para ser precisa, ter que definir que o intervalo de tempo entre cada um dos eventos (bit) é de 1 micro-segundo. Este é um dos aspectos problemáticos da especificação de protocolos. Complica a definição formal das linguagens e dificulta a validação (especialmente os aspectos de verificação) das especificações. Note-se, adicionalmente, que o conceito de tempo é em geral expresso como um intervalo de tolerância, e que o próprio conceito de instante definido ou intervalo de tempo definido não existe na realidade. Este aspecto está também relacionado à discussão dos diferentes níveis de protocolo, no item 4.1.19.

#### 4.1.7 Especificação de dados

A definição dos tipos de dados é uma importante característica da especificação, sendo, em geral, facilmente caracterizável como um todo, diversamente da definição do comportamento. Aqui são incluídas a existência de variáveis, criação, escopo, atribuição, efeitos colaterais na manipulação de variáveis e visibilidade da variável pelos diversos processos.

#### 4.1.8 Nível de especificação

E desejável que a linguagem permita a geração de

especificações com níveis de detalhe diferentes, permitindo que através de passos sucessivos se possa chegar ao nível de precisão desejado, sem que a omissão de informações nas especificações de mais alto nível seja considerada incorreção sintática ou semântica.

#### 4.1.9 Nível de abstração

Uma linguagem de especificação induzirá a especificações mais ou menos abstratas, no sentido de atrelamento ao "como implementar".

#### 4.1.10 Implementação automática

Relacionada à facilidade de construir processos de tradução total ou parcial de especificações de protocolos gerando entidades (em geral de software) de um sistema, cujo comportamento, na interconexão com outros sistemas, satisfaz as restrições do protocolo especificado, desde que suportadas por entidades que prestem serviço do nível inferior ao do protocolo.

#### 4.1.11 Concisão

A concisão de uma linguagem pode ser analisada separadamente na solução de cada problema. É interessante, porém, comentá-la também de forma geral, considerando aspectos de concorrência, comunicação, etc., bem como aspectos de complexidade das construções, multiplicidade de construções com mesmo propósito, etc.

#### 4.1.12 Consistência

Serão analisadas as condições que a linguagem oferece para que sejam geradas especificações consistentes (no sentido de não ensejarem afirmações contraditórias), bem

como a possibilidade de determinação formal (verificação) da consistência em especificações.

#### 4.1.13 Completeza

A completeza assume características diversas em cada linguagem. Serão vistos os aspectos que determinam a completeza em cada linguagem e a dificuldade de gerar/garantir uma especificação completa. Cabe salientar que a completeza de uma especificação, caracterizada pela existência de reação a qualquer sequência de estímulos do meio está associada à consistência (no sentido alternativo ao do item 4.1.12, de acordo com o visto em 2.3.5) entre a especificação e os elementos do meio externo com os quais o elemento especificado interage. O meio externo, que pode ser por exemplo o prestador de um serviço, tem um comportamento presumido. Isto significa, que nem todas as sequências de evento podem ser possíveis na entrada, só precisando serem previstas as sequências que forem consistentes com o comportamento previsto das entidades do meio externo.

#### 4.1.14 Adequação ao modelo OSI

Considera-se aqui o modelo OSI como referência, não só porque ele é um padrão de fato, mas principalmente porque considera-se que este é apenas uma formalização da estruturação natural da interconexão. Não se está considerando aqui a razoabilidade da divisão em sete níveis, as funções de cada camada, as primitivas, etc., mas sim a filosofia de prestação de serviço e execução de protocolos por entidades.

#### 4.1.15 Formalismo

Aqui será tratado o estágio em que se encontram as linguagens, de acordo com os documentos a que se tem acesso,

com respeito a sua formalização. Ressalte-se que, dado o caráter de linguagem de especificação, é extremamente importante que exista uma instância de decisão para a interpretação dos elementos da linguagem, sobre a qual não parem dúvidas: um modelo formal e um método de mapeamento, cujas bases sejam bem conhecidas ou facilmente assimiláveis sem ocorrência de interpretações ambíguas; em geral bases matemáticas. Isto é importante devido a inexistência, em princípio, de ferramentas automáticas, nas quais se possa "testar" o efeito de alguma construção, bem como o reduzido escopo de aplicação das linguagens, que dificulta sua difusão.

#### **4.1.16 Verificação**

Linguagens diferentes dão condições diferentes de determinar formalmente o cumprimento dos requisitos necessários para uma especificação estar correta.

#### **4.1.17 Adequação a objetivos**

Neste tópico serão abordados os objetivos, conforme seção 2.4.1, para os quais cada uma das linguagens se adequa (documentação e divulgação, validação de especificações, apoio à implementação e teste) e sugestões de possíveis alterações.

#### **4.1.18 Adequação ao usuário**

Serão abordadas as possíveis restrições de usuários a cada uma das linguagens, a validade das objeções e sugestões de alterações das linguagens para que, sem prejuízo de seus objetivos, elas se tornem mais agradáveis ao usuário.

#### 4.1.19 Adequação aos níveis de protocolo

Os protocolos das camadas inferiores do RM-OSI contêm características de baixo nível que os diferenciam das outras camadas de protocolo. Um exemplo típico é a especificação de intervalos definidos de tempo visto no item 4.1.6. Outros são a especificação das características dos sinais que serão enviados para a linha: formato da onda a ser colocada na linha, níveis de tensão e tolerâncias, reconhecimento de portadora, etc. que são características difíceis de modelar formalmente. Os protocolos de aplicação, por outro lado, são bastante genéricos e facilmente modeláveis por uma linguagem de alto nível de propósitos gerais.

#### 4.2 Comparação

Na comparação entre as duas linguagens, segundo os parâmetros da seção 4.1, serão utilizados exemplos retirados da especificação do protocolo BSC3 de uma estação IBM 3274, elaborado como parte auxiliar deste trabalho. Isto evita o artificialismo que em geral ocorre quando as situações são criadas apenas para suprir a necessidade de exemplos. Contudo, considerando que tal especificação não esgota as situações aqui consideradas, alguns exemplos foram imaginados apenas para este capítulo. Procurou-se torná-los o mais próximo possível de situações reais.

Em cada item é feita inicialmente a análise de Estelle em relação ao parâmetro considerado, utilizando-se exemplos conforme necessário, para justificar os pontos positivos e negativos. A seguir é feita a análise de Lotos. Quando, na análise do parâmetro, for considerado apropriado um confronto direto entre as duas linguagens na solução de um problema, a análise de Lotos fará referência à situação

apresentada na consideração de Estelle, descrevendo sua solução.

Nos exemplos foram utilizadas reticências (.....) para indicar a omissão de pedaços de cláusulas, comandos ou mesmo trechos de especificação.

No processo de estudo e comparação das linguagens foram levadas em conta as discussões de [SPE 87a, SPE 87b, COU 87], sendo o primeiro artigo, uma publicação revisada (mais cautelosa nas afirmações) do segundo.

#### 4.2.1 Concorrência

O suporte da concorrência em Estelle é feito pela possibilidade de definição de tipos de módulos e instâncias hierarquizadas destes tipos. Existe paralelismo e entrelaçamento. Ocorre entrelaçamento entre eventos de módulos em uma mesma linha de hierarquia e entre eventos de atividades (um dos tipos de módulos) irmãs. Nos outros casos tem-se paralelismo. Aparentemente houve uma escolha inicial pelo modelo de paralelismo, adotando-se o entrelaçamento, nos casos acima, para suportar a possibilidade de compartilhamento de variáveis, vista adiante, no item 4.2.7. No exemplo abaixo tem-se um "esqueleto" da especificação em Estelle do BSC3, conforme visto por uma estação terminal, mostrando apenas a estruturação dos módulos. Poder-se-ia ter omitido, vários detalhes, facilitando a compreensão do exemplo. Optou-se, no entanto, por colocar todos os elementos envolvidos na especificação dos módulos, incluindo definição dos canais com papéis e interações, definição de interface e corpo dos módulos. Isto foi feito para que se pudesse avaliar o tamanho e número de construções necessárias em Estelle para definir tal estrutura de módulos, mostrada na figura 4.1, podendo-se comparar com a correspondente especificação em Lotos, dada mais adiante. As

linhas pontilhadas na especificação indicam, neste caso, a omissão de pedaços correspondentes à definição das estruturas de dados e comportamento dos módulos.

A especificação tem como raiz um módulo, 'BSC3\_spec', para o qual só existe uma instância, de mesmo nome. Por definição de Estelle, este módulo não tem interface, isto é, Estelle descreve sistemas fechados. Como não queremos descrever todo um sistema de terminais e computador, mas apenas um módulo, que executa as funções do protocolo BSC3 em um terminal (uma entidade de protocolo), definimos apenas um módulo de nível inferior, chamado 'BSC3' (com interface do tipo 'BSC3\_type'). Este é a especificação propriamente dita do comportamento do terminal na execução do BSC3. Note-se que a imposição de que a especificação em Estelle seja fechada, induz a dois tipos de especificação: ou um ambiente completo, fechado, com estação controladora e número determinado de terminais, ou a alternativa utilizada nesta especificação de apenas descrever o módulo sem criar qualquer instância, que é o objetivo proposto. Por isto, na figura 4.1, o módulo 'BSC3' está representado por um retângulo tracejado. Nada nesta especificação diz que é criada uma instância deste. Porém, supondo que uma tal instância seja criada, a figura 4.1 mostra toda a sub-árvore de instâncias de módulos hierarquicamente inferiores, criados para a execução do protocolo, de acordo com a especificação abaixo. Note-se a dificuldade, em Estelle, de separar a descrição do comportamento, da descrição da implementação de um sistema que tenha este comportamento, como se estivessem sendo especificados processos em uma linguagem concorrente. Esta característica é abordada novamente no item 4.2.9.

Antes da descrição de um tipo de módulo, são declarados todos os tipos de canais que ele vai referenciar



em sua interface, seja na definição de pontos de interação externos, ou na declaração de variáveis (pontos de interação internos), incluindo os possíveis papéis e interações associadas.

```

specification BSC3_spec;

channel Line_interface_rx (User, Provider);
  by Provider: LINE (ch: char);
channel Line_interface_tx (User, Provider);
  by User: LINE (ch: char);
channel BSC3_service_access_point (User, Provider);
  by User: MSG (Data: string, Length: int);
           BLOCK (Data: string, Length: int);
           SUB_ENQ_MSG (Data: string, Length: int);
           EOT_MSG;
           WACK_MSG;
           RVI_MSG;
           ACK_MSG;
  by Provider: POL (Device_add: char);
              SEL (Device_add: char);
              MSG (Data: string, Length: int);
              BLOCK (Data: string, Length: int);
              ACK_MSG;
              TERM_DUE_TO_RVI;
              UNSUCCESSFUL_TRANSM;
              CONTROLLED_FW_ABORT;
              ABNORMAL_EOT;
              FIM_MSG;

module BSC3_type process (
  LOWER_LEVEL_RX: Line_interface_rx (User)
    individual queue;
  LOWER_LEVEL_TX: Line_interface_tx (User)
    individual queue;

```

```

UPPER_LEVEL: BSC3_service_access_point
              (Provider) individual queue      );

body BSC3 for BSC3_type;
.....
channel  Timer_interface (User, Provider);
  by  User:  INIT (Tempo: int);
              OFF;
  by  Provider:  TIMEOUT;
channel  Receiver_access_point (User, Provider);
  by  User:      RECEIVE;
              STOP;
              RECEIVE_TRANSPARENT;
              START_BCC_ACCUMULATION;
              RECEIVE_BCC_AND_TURN_NORMAL;
              CHAR_CONFIRMATION;
  by  Provider:  CHARAC (ch: char);
              BCC (bcc_ok: boolean);
              TOUT3S;
channel  Transmitter_access_point (User, Provider);
  by  User:      CTL_TX (ch: char);
              CTL_TX2 (ch1, ch2: char);
              STOP;
              START_TEXT_TX;
              TRANSMIT_TRANSPARENT;
              START_BCC_ACCUMULATION;
              BCC_FOLLOWS (ch: char);
              CHARAC (ch: char);
  by  Provider:  SEQUENCE_TRANSMITED;
channel  Msg_assembler_access_point
              (User, Provider);
  by  User:      STOP;
              RESET;
              RECEIVE;

```

```

                                LOST;
    by   Provider: FRAME_RECEIVED
                                (tipo: tipo_frame; data:string;
                                length: int                );
                                POL (device_add: char);
                                SEL (device_add: char);
channel  Msg_disassembler_access_point
                                (User, Provider);
    by   User:      TEXTO (data: string,
                                length: int                );
                                CONTROLE (Tipo:Tipo_controle);
    by   Provider: MSG_TRANSMITED;

module Timer_type process
                                (TIP: Timer_interface (Provider)
                                individual queue           );

body Timer for Timer_type;
.....
initialize to IDLE
begin end;
.....
/* descrição do comportamento */

module Receiver_type process
    (LOWER_LEVEL: Line_interface (User)
     individual queue;
     UPPER_LEVEL: Receiver_access_point
     (Provider) individual queue           );
module Transmitter_type process
    (LOWER_LEVEL: Line_interface (User)
     individual queue;
     UPPER_LEVEL: Transmitter_access_point
     (Provider) individual queue           );
module Msg_assembler_type process
    (LOWER_LEVEL: Receiver_access_point

```

```

        (User) individual queue;
    UPPER_LEVEL: Msg_assembler_access_point
        (Provider) individual queue    );
module Msg_disassembler_type process
    (LOWER_LEVEL: Transmitter_access_point
        (User) individual queue;
    UPPER_LEVEL: Msg_disassembler_access_point
        (Provider) individual queue    );
module LLC_type process
    (LOWER_LEVEL_RX: Msg_assembler_access_point
        (User) individual queue;
    LOWER_LEVEL_TX: Msg_disassembler_access_point
        (User) individual queue;
    UPPER_LEVEL: Bsc3_service_access_point
        (Provider) individual queue    );

body Receiver for Receiver_type;
var Tim: Timer_type;
    Timer_port: Timer_interface (User)
        individual queue;
.....
initialize to CAPTURE_SYNC_SEQ
begin
    init Tim with Timer;
    connect Tim.Tip to Timer_port
end;
.....
/* descrição do comportamento */

body Transmitter for Transmitter_type;
var Tim: Timer_type;
    Timer_port: Timer_interface (User)
        individual queue;
.....
initialize to IDLE

```

```

begin
  init Tim with Timer;
  connect Tim. Tip to Timer_port;
end;

.....
/* descrição do comportamento */

body Msg_assembler for Msg_assembler_type;
.....
  initialize to NOT_SELECTED
  begin end;
.....
/* descrição do comportamento */

body Msg_disassembler for Msg_Disassembler_type;
.....
  initialize to IDLE
  begin end;
.....
/* descrição do comportamento */

body LLC for LLC_type;
  var Tim: Timer_type;
      Timer_port: Timer_interface (User)
                individual queue;
.....
  initialize to RESET
  begin
    init Tim with Timer;
    connect Tim. Tip to Timer_port;
  end;
.....
/* descrição do comportamento */

var mod0: Receiver_type;

```

```

    mod1: Transmitter_type;
    mod2: Msg_assembler_type;
    mod3: Msg_disassembler_type;
    mod4: LLC_type;

initialize
begin
    init mod0 with Receiver;
    init mod1 with Transmitter;
    init mod2 with Msg_assembler;
    init mod3 with Msg_disassembler;
    init mod4 with LLC;
    connect mod0.UPPER_LEVEL to mod2.LOWER_LEVEL;
    connect mod1.UPPER_LEVEL to mod3.LOWER_LEVEL;
    connect mod2.UPPER_LEVEL to mod4.LOWER_LEVEL_RX;
    connect mod3.UPPER_LEVEL to mod4.LOWER_LEVEL_RX;
    attach LOWER_LEVEL_RX to mod0.LOWER_LEVEL;
    attach LOWER_LEVEL_TX to mod1.LOWER_LEVEL;
    attach UPPER_LEVEL to mod4.UPPER_LEVEL;
end;
end; /* of BSC3 body */
end; /* of the incomplete system specification */

```

Quando é gerada uma instância de 'BSC3', são também geradas automaticamente cinco instâncias de módulos filhos, correspondentes às descrições de módulo 'Receiver', 'Transmitter', 'Msg\_assembler', 'Msg\_disassembler' e 'LLC'. A geração das duas primeiras e da última inclui, para cada uma, a criação de uma instância de 'Timer'. As linhas cheias da figura 4.1 mostram a hierarquia dos módulos. As linhas tracejadas mostram as conexões (definidas na especificação pelos comandos 'connect') e transferências (comandos 'attach'), feitas posteriormente à criação, por onde se dará a comunicação entre os módulos. Não é feita a representação dos pontos de interação. Note-se que as transferências,

representadas na figura pelas três linhas tracejadas saindo do retângulo 'BSC3', na verdade indicam comunicação com o meio externo a 'BSC3', uma vez que os três pontos de interação referenciados pertencem à interface deste, representando a comunicação com a linha (transmissão e recepção) e com a aplicação embutida no terminal (suporte ao usuário).

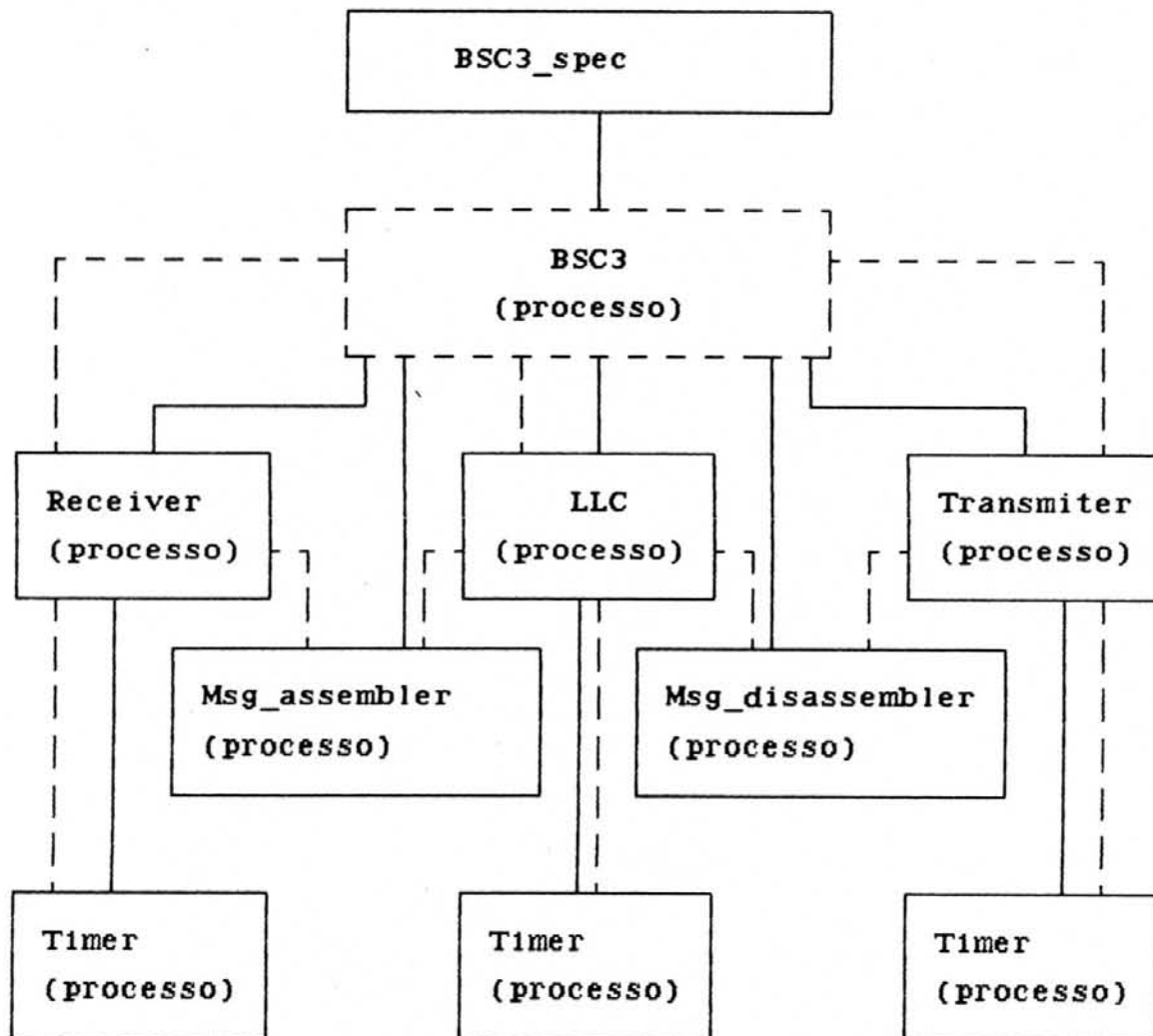


fig 4.1 Estrutura de módulos do BSC3

Na especificação acima foram utilizados apenas

módulos tipo processo. Isto implica que as transições de módulos irmãos ('Receiver', 'Transmitter', 'LLC', ...) podem ocorrer em paralelo, isto é, não há dependência entre a ocorrência de uma transição de 'Transmitter' e uma de 'Receiver' por exemplo, exceto aquela ditada pela troca de mensagens (habilitação das transições). Isto caracteriza o modelo aqui descrito como paralelismo. Entre módulos de uma mesma linha de descendência, no entanto, os eventos só podem ocorrer de forma entrelaçada. Por exemplo, uma transição de 'Receiver' não pode iniciar se houver uma transição sendo executada no módulo 'Timer', filho de 'Receiver'.

Se tivessem sido utilizados módulos tipo atividade (lembrando que atividades não podem gerar módulos filhos e que não podem existir irmãos de tipos - processo e atividade - diferentes), haveria entrelaçamento também na ocorrência de transições das atividades irmãs.

Em Lotos, o suporte de concorrência é feito exclusivamente pelas três variantes (|||, || e |[...]|) do operador de concorrência (<parallel\_op>), nas construções <parallel\_expression> e <general\_parallel\_expression>. O modelo único é o de entrelaçamento, observando-se certo conflito com a terminologia aqui adotada ("parallel" X paralelismo). A seguir é apresentado um "esqueleto" de especificação em Lotos equivalente ao visto acima em Estelle. Não há necessidade de especificação de canais, nem declaração prévia das interações. A especificação é vista como um sistema aberto, de modo que não há sentido em ter um identificador correspondente a 'BSC3\_spec'. Por definição, todos os eventos ocorrem de forma entrelaçada. O operador '|[...]|' indica que o comportamento de BSC3 é dado pelo comportamento de 'Receiver', 'Transmitter', etc., considerados de forma concorrente (com eventos diferentes entrelaçados). A comunicação e a sincronização são feitas



pelas portas expressas no interior do operador que especifica a concorrência. As portas colocadas como argumentos das referências aos processos são passadas em substituição aos parâmetros usados na definição destes. As portas referenciadas como 'Porta\_interna\_'1'' ( $0 \leq 1 \leq 4$ ) não são portas de comunicação com o meio externo como 'LOWER\_LEVEL\_RX', 'LOWER\_LEVEL\_TX' e 'UPPER\_LEVEL', servindo apenas para comunicação e sincronização internas no auxílio à especificação, precisando por isto serem introduzidas por 'hide'. Note-se ainda que as mesmas construções abaixo, de concorrência, são usadas na descrição das sequências possíveis de eventos, não havendo como em Estelle, uma diferenciação rígida entre estruturação de módulos e comportamento de cada um deles.

specification BSC3

```
[LOWER_LEVEL_RX, LOWER_LEVEL_TX, UPPER_LEVEL]
: noexit
```

.....

behaviour

```
hide Porta_interna_0, Porta_interna_1,
      Porta_interna_2, Porta_interna_3,
      Porta_interna_4
```

in

```
Receiver [LOWER_LEVEL_RX, Porta_interna_0]
  |[ Porta_interna_0 ]|
Msg_assembler [Porta_interna_0,
                Porta_interna_2]
  |[ Porta_interna_2 ]|
LLC           [Porta_interna_2, Porta_interna_3,
                UPPER_LEVEL]
  |[ Porta_interna_3 ]|
Msg_disassembler [Porta_interna_1,
                  Porta_interna_3]
```

```

[[ Porta_interna_1 ]]
Transmitter [LOWER_LEVEL_TX, Porta_interna_1]

```

where

```

process Receiver
    [LOWER_LEVEL, UPPER_LEVEL]
    :noexit
:=   hide Timer_port
    in   Timer [Timer_port]
        [[ Timer_port]]| .....
        /* descrição do comportamento */
endproc
process Transmitter
    [LOWER_LEVEL, UPPER_LEVEL]
    : noexit
:=   hide Timer_port
    in   Timer [Timer_port]
        [[ Timer_port]]| .....
        /* descrição do comportamento */
endproc
process Msg_assembler
    [LOWER_LEVEL, UPPER_LEVEL]
    :noexit
:= .....
    /* descrição do comportamento */
endproc
process Msg_disassembler
    [LOWER_LEVEL, UPPER_LEVEL]
    :noexit
:= .....
    /* descrição do comportamento */
endproc
process LLC
    [LOWER_LEVEL, UPPER_LEVEL]

```

```

        :noexit
:=   hide Timer_port
      in   Timer [Timer_port]
          |[ Timer_port]| .....
      /* descrição do comportamento */
endproc
process Timer [TIP]: noexit
:= .....
      /* descrição do comportamento */
endproc
endspec

```

#### 4.2.2 Comunicação

Há dois modelos de comunicação presentes em Estelle, ambos para comunicação assíncrona. Um deles é o de comunicação por troca de mensagens (interações parametrizadas). Associada a esta modelo, conforme visto no item anterior e também no capítulo 3, existe toda uma estrutura de definição de tipos de canais, de pontos de interação pelos quais os módulos se comunicam e de tipos de ligação, bastante rígida. Dois pontos de interação compatíveis de módulos diferentes podem ser ligados através da execução de um comando 'connect', dentro do bloco de comandos de uma transição ou da inicialização de um módulo (embora possa ser feita ligação de pontos de um mesmo módulo, neste contexto eles podem ser tratados como dois módulos distintos), permitindo-se a partir então que sejam trocadas interações entre eles, segundo as características do canal definido. O envio de uma interação é feito por um comando 'output', contido em um bloco de comandos de uma transição, que referencia um ponto de interação de seu módulo (previamente conectado a um ponto de interação do módulo destinatário) e a interação a ser enviada. A

interação é colocada na fila associada ao ponto de interação do módulo destinatário. A presença da fila associada a cada ponto de interação foi incluída no modelo justamente para caracterizar a comunicação assíncrona. O módulo destinatário não precisa estar atento à interação no momento em que ela é enviada pelo remetente, podendo inclusive haver acúmulo de interações, ordenadas de acordo com a ordem de chegada. Na especificação de transições da máquina de estados, a construção 'when' permite que seja pesquisada a presença de uma dada interação, no topo da fila associada a um dado ponto de interação do módulo. A resposta afirmativa a tal pesquisa contribui para a habilitação da transição. A pesquisa ainda permite a observação dos parâmetros da interação através da cláusula 'provided'. Caso seja executada a transição, a interação é retirada da fila e seus parâmetros são apropriados. O modelo de fila restringe bastante o tipo de modelagem a ser feita, pois induz a um modelo em que não há perda de mensagens por acúmulo na recepção ou por falta de atendimento imediato. Além disso, qualquer característica de envio diferente, como a possibilidade de entrega das mensagens em ordem diferente do envio torna totalmente inadequado o modelo. No item 4.2.18 são consideradas as deficiências do modelo da fila a nível de adequação ao usuário.

Abaixo são apresentados alguns trechos das definições das máquinas de estados dos módulos apresentados no item anterior, escolhidos para exemplificar a comunicação assíncrona. O primeiro trecho, do módulo 'Receiver', mostra que, estando o módulo no estado 'RECEIVING\_NORMAL', se houver uma interação 'LINE' no topo da fila associada ao ponto 'LOWER\_LEVEL' (recepção de dados da linha de comunicação), dependendo do valor do parâmetro 'ch' da interação será executada a primeira ou a segunda transição. A primeira transição, correspondente a 'ch = SYN'

(recebimento do caracter de sincronismo), provoca o envio de uma mensagem ao módulo 'Timer' (vide item 4.2.1) conectado pela porta 'Timer\_port'. O tratamento desta mensagem 'INIT (3)', cuja função é reinicializar um contador de tempo máximo de recepção de 3 segundos, pode ser acompanhado no item 4.2.6, adiante, onde é apresentada a especificação do módulo 'Timer'. Aqui será feito acompanhamento da segunda transição, que provoca o envio de uma interação 'CHARAC' com o mesmo parâmetro recebido e leva a máquina ao estado 'WAIT\_NORMAL\_CHAR\_CONF'. Lembrando que o ponto de interação 'UPPER\_LEVEL' de 'Receiver' está conectado ao 'LOWER\_LEVEL' de 'Msg\_assembler', pode-se agora acompanhar o funcionamento da máquina associada ao módulo 'Msg\_assembler', parcialmente descrita no segundo trecho abaixo, supondo que esta esteja no estado 'WAITING\_LOWER\_LEVEL'. O recebimento da interação 'CHARAC' na fila associada a 'LOWER\_LEVEL' habilita a transição mostrada caso o parâmetro 'ch' seja igual a 'EOT', provocando mudança de estado e envio da interação 'CHAR\_CONFIRMATION', de volta ao módulo 'Receiver', que volta a estar habilitado.

/\* trecho da especificação da máquina de estados  
do módulo Receiver \*/

```

from RECEIVING_NORMAL
  when LOWER_LEVEL. LINE (ch)
    provided ch = SYN
      to SAME
      begin
        output (Timer_Port. INIT (3));
      end;
    provided OTHERWISE
      to WAIT_NORMAL_CHAR_CONF
      begin

```

```

        CALCULA_BCC (ch);
        output (UPPER_LEVEL. CHARAC (ch));
        END;
from WAIT_NORMAL_CHAR_CONF
    when UPPER_LEVEL. CHAR_CONFIRMATION
        to RECEIVING_NORMAL
            begin end;
    when UPPER_LEVEL. RECEIVE_TRANSPARENT
        to WAIT_TRANSP_CHAR_CONF
            begin end;
    .....

/* trecho da especificação da máquina de estados
do módulo Msg_assembler */

from WAITING_LOWER_LEVEL
    when LOWER_LEVEL. CHARAC (ch)
        provided ch = EOT
            to WAIT_PAD_EOT
                begin
                    .....
                    output (LOWER_LEVEL.
                                CHAR_CONFIRMATION);
                end;
    .....

```

O outro modelo em Estelle é a comunicação assíncrona por compartilhamento de área. Estelle permite que um módulo exporte algumas de suas variáveis para que sejam compartilhadas com o módulo pai e possivelmente com os irmãos. Aqui surge fortemente a necessidade de garantir exclusão mútua no acesso às variáveis, o que induz a uma restrição no modelo de paralelismo. Conforme visto na concorrência, a ocorrência de eventos nos módulos de uma

mesma linha de descendência é entrelaçada, o mesmo ocorrendo entre atividades irmãs. Este modelo garante a exclusão mútua, e parece ser a única razão para sua existência junto com o modelo de paralelismo. Na especificação do BSC3, não foi usado o compartilhamento de variáveis (e, por conseguinte, a definição de módulos tipo atividade) por considerar-se a alternativa muito próxima da implementação (vide item 4.2.9). Esta forma de comunicação pode, apesar disto, ser o meio mais apropriado, em Estelle, para estabelecimento de sincronismo entre módulos, razão por que é abordada no item 4.2.3.

Em Lotos, a comunicação é síncrona, feita durante a ocorrência de eventos, de acordo com a participação de cada expressão envolvida e do meio externo. É um modelo extremamente flexível, permitindo em um mesmo evento a comunicação bidirecional, sincronização e geração não determinística de valor. Pode-se ainda expressar comunicação interna entre expressões, utilizando-se `<hiding_expression>` (conforme introduzido no item 4.2.1). É um modelo conciso, flexível e bastante poderoso. A seguir são apresentados três trechos em Lotos: os dois primeiros consistindo em duas alternativas diferentes, equivalentes ao trecho em Estelle, acima, para o módulo 'Receiver'; o terceiro trecho, equivalente ao de Estelle para o módulo `Msg_assembler`. Na primeira alternativa para o módulo 'Receiver' foi feito um mapeamento imediato de estado de Estelle para processo de Lotos. Na segunda alternativa, mostra-se que a existência de estados em Estelle não é equivalente a de processos em Lotos. O processo de Lotos pode ser usado para modularização ou para recursão. Recordando-se, da Teoria da Computação, que qualquer função computável pode ser definida como função recursiva parcial com utilização uma única vez do operador de minimização [BRA 74] e considerando-se, mediante

observação da linguagem Lotos, que a chamada de processo para continuação da sequência de eventos tem função análoga à da minimização, conclui-se que a chamada de processo só é teoricamente necessária uma vez na especificação. Conclui-se que o mapeamento da especificação do processo de Estelle para Lotos pode manter a mesma forma, não sendo o caminho inverso necessariamente verdadeiro. Ainda quanto ao estilo, no terceiro trecho, ao invés de associar-se guardas à especificação dos eventos, como nos casos anteriores, estas foram associadas às possíveis alternativas de continuação de sequência. Saliencia-se ainda que a chamada ao processo 'CALCULA\_BCC' não foi adaptada adequadamente por simplicidade. Uma versão correta se encontra no item 4.2.7, quando é discutido o estilo funcional de Lotos.

/\* trecho equivalente ao do módulo Receiver -  
primeira alternativa \*/

```

process  RECEIVING_NORMAL
:=      LOWER_LEVEL ! LINE ? ch: ch_sort [ch = SYN];
        Timer_port ! INIT ! 3; RECEIVING_NORMAL
[] LOWER_LEVEL ! LINE ? ch: ch_sort [ch NEQ SYN];
        ( CALCULA_BCC (ch) >> UPPER_LEVEL ! CHARAC !ch;
          WAIT_NORMAL_CHAR_CONF )
endproc

process  WAIT_NORMAL_CHAR_CONF
:=      UPPER_LEVEL ! CHAR_CONFIRMATION; RECEIVING_NORMAL
[] UPPER_LEVEL ! RECEIVE_TRANSPARENT;
        WAIT_TRANSP_CHAR_CONF
[] .....
endproc

```



```

/* trecho equivalente ao do módulo Receiver -
   SEGUNDA alternativa */

process RECEIVING_NORMAL
:=   LOWER_LEVEL ! LINE ? ch: ch_sort [ch = SYN];
      Timer_port ! INIT ! 3; RECEIVING_NORMAL
[] LOWER_LEVEL ! LINE ? ch: ch_sort [ch NEQ SYN];
      ( CALCULA_BCC (ch) >> UPPER_LEVEL ! CHARAC !ch;
        ( UPPER_LEVEL ! CHAR_CONFIRMATION;
          RECEIVING_NORMAL
        [] UPPER_LEVEL ! RECEIVE_TRANSPARENT;
          WAIT_TRANSP_CHAR_CONF
        [] .....))
endproc

/* trecho equivalente ao
   do módulo Msg_assembler */
process WAITING_LOWER_LEVEL
:=   LOWER_LEVEL ! CHARAC ? ch: ch_sort
      ( [ch = EOT] -> (.....
        >> LOWER_LEVEL ! CHAR_CONFIRMATION;
          WAIT_PAD_EOT      )
      [.....] .....
endproc

```

Considerando-se, conforme especificado no item 4.2.1, que os parâmetros formais 'UPPER\_LEVEL' de 'Receiver' e 'LOWER\_LEVEL' de 'Msg\_assembler' foram, ambos, substituídos na chamada por 'Porta\_interna\_0' e sincronizados por esta no operador de paralelismo, a interpretação dos trechos acima pode ser dada assim: supondo que a continuação da sequência de eventos de 'Receiver' seja aquela dada por 'RECEIVING\_NORMAL' e a continuação em 'Msg\_assembler' seja dada por 'WAITING\_LOWER\_LEVEL', então, a sequência avança pela ocorrência do evento de

sincronização através do "valor" 'LINE' com o meio externo (pela porta formal 'LOWER\_LEVEL' de 'Receiver' que, de acordo com o item 4.2.1, foi substituída por 'LOWER\_LEVEL\_RX', da interface), simultâneo à recepção de um valor em 'ch'. Note-se que a composição acima é equivalente à parametrização das interações de Estelle. Uma das continuações possíveis, supondo 'ch' diferente de 'SYN', é o envio do valor recebido em 'Receiver' ('RECEIVING\_NORMAL') para 'Msg\_assembler' ('WAITING\_LOWER\_LEVEL'), pela porta 'Porta\_interna\_0' que, conforme visto, substitui os parâmetros 'UPPER\_LEVEL' de 'Receiver' e 'LOWER\_LEVEL' de 'Msg\_assembler'. Isto está especificado pelas expressões "UPPER\_LEVEL ! CHARAC ! ch" na quarta linha de 'RECEIVING\_NORMAL' e "LOWER\_LEVEL ! CHARAC ? ch: sort" no início de 'WAITING\_LOWER\_LEVEL'. Note que o envio (!) e o recebimento (?) ocorrem simultaneamente. Após, caso 'ch' seja igual a 'EOT', ocorre simultaneamente nos dois processos o evento dado por 'CHAR\_CONFIRMATION'. Note-se que em Estelle este evento era expresso pelo envio de uma interação específica 'CHAR\_CONFIRMATION', sem parâmetros. Em Lotos fica caracterizado que esta operação é apenas um ato de sincronização, uma vez que não há variáveis passadas.

É importante ressaltar que a especificação equivalente em Lotos aqui apresentada, utilizou-se do fato de que o problema não exigia a presença de filas na comunicação. Os dois módulos de Estelle cooperavam de forma entrelaçada, característica comum em se tratando de protocolos. Quando realmente for necessária uma fila ela deverá ser explicitamente introduzida em Lotos, isto é, a fila como tantos outros elementos só aparece na especificação se for um elemento componente do problema. A especificação de filas em Lotos é abordada como exemplo no item 4.2.7.

A geração não determinística de valor é um mecanismo bastante interessante, que permite definir faixas de tolerância de consenso entre as expressões envolvidas na comunicação, através das guardas, além das restrições impostas pelo meio. No exemplo abaixo é apresentado, em Lotos, o processo de estabelecimento de conexão entre duas entidades de protocolo de transporte, em alto nível de especificação, mostrando a versatilidade dos recursos para especificação de comunicação e geração de valor. Foram omitidos, por simplicidade, vários parâmetros do processo.

```

process Comunica_entidades [Canal1, Canal2]
:= Transporte [Canal1, Canal2]
    |[ Canal1, Canal2 ]|
    Transporte [Canal2, Canal1]
where
process Transporte [INICIADOR, ACEITADOR]
:= INICIADOR ! CONEXAO
    ! SRC_REF ? DEST_REF: address
    ? CLASS: class_sort ? SIZE: size_sort
    [ ((CLASS = 4) or (CLASS = 2)) and
      (SIZE leq 2048) ] ;
    .....
[] ACEITADOR ! CONEXAO
    ? SRC_REF: address ! DST_REF
    ? CLASS: class_sort ? SIZE: size_sort
    [ (CLASS = 2) and (SIZE leq 4096) ] ;
    .....
endproc
endproc

```

Nas duas instâncias invocadas do processo 'Transporte', correspondentes a duas entidades pares, os canais aparecem em posições invertidas: uma vez como

'INICIADOR', outra como 'ACEITADOR'. Assim, qualquer entidade pode iniciar ou aceitar a conexão. O 'INICIADOR' envia 'SRC\_REF' para o 'ACEITADOR' e recebe deste 'DST\_REF'. São estabelecidos valores para 'CLASS' e 'SIZE' de acordo com as restrições impostas por cada parte. Pode-se concluir, no caso, que 'CLASS' receberá o valor '2' e 'SIZE' receberá qualquer valor válido (size\_sort) desde que menor ou igual a '2048'. Se não tivesse havido possibilidade de consenso a ocorrência do evento não seria possível.

#### 4.2.3 Sincronização

Estelle não tem mecanismos específicos para sincronização. A sincronização em Estelle tem que ser feita pelos mecanismos de comunicação. É bastante complicado fazer sincronização pela transferência de mensagens, principalmente pela existência das filas nos pontos de interação. O compartilhamento de variáveis passa a ser uma alternativa segura frente à inexistência de outras.

Para caracterizar o problema de sincronização em Estelle será considerada a máquina de estados que descreve o processo de recepção de caracteres do BSC3 ('Receiver'). A partir do estado inativo ('IDLE') ela deve esperar uma interação 'RECEIVE' do módulo 'Msg\_assembler', conforme o trecho abaixo (por questão de completeza deve ser previsto o descarte de caracteres vindos da linha):

```

from IDLE
  when UPPER_LEVEL. RECEIVE
    to CAPTURE_SYNC_SEQ
      begin
        output (TIMER_PORT. OFF);
      end
  when LOWER_LEVEL. LINE (CH)
    to SAME

```

```
begin end;
```

Continuando o exemplo, considera-se a necessidade de prever a partir de qualquer estado a vinda da primitiva 'TIMEOUT' do módulo 'TIMER', com a qual a máquina é devolvida ao estado 'IDLE', após informar-se o fato ao módulo 'Msg\_assembler' com a primitiva 'TOUT3S', conforme trecho abaixo.

```
priority 1
  when Timer_port. TIMEOUT
    to IDLE
      begin
        output (Timer_port. OFF);
        output (UPPER_LEVEL. TOUT3S);
      end
```

O problema da sincronização aqui consiste em garantir que logo após executada a transição acima de tratamento de 'TIMEOUT' o módulo 'Msg\_assembler' reconheça a situação (através do recebimento da interação 'TOUT3S') e mude de estado, sem que sejam geradas interações relativas ao contexto anterior (apropriação da sequência de caracteres recebidos da linha), como por exemplo 'RECEIVE\_TRANSPARENT' e 'START\_BCC\_ACCUMULATION', enviadas do módulo 'Msg\_assembler' ao 'Receiver' quando constada a necessidade de, respectivamente, iniciar recebimento transparente de texto e iniciar cálculo de carácter de controle de erro de bloco. Como o envio de mensagens não é síncrono duas soluções alternativas podem ser visualizadas. Uma delas é fazer com que os módulos irmãos 'Receiver' e 'Msg\_assembler' sejam do tipo atividade (forçando entrelaçamento de eventos), atribuindo alta prioridade ao recebimento da interação 'TOUT3S' em 'Msg\_assembler' e atribuindo baixa prioridade ao reconhecimento de 'TIMEOUT' em 'Receiver' para

garantir que não haja mais nenhuma transição habilitada (o que não é muito razoável, visto que o "timeout" deve ter uma característica de interrupção do contexto). Além disto teria que existir um canal específico para esta interação, garantindo que ela fosse sempre colocada na cabeça da fila. A mudança para atividades forçaria tal mudança também para os outros três módulos irmãos. A segunda alternativa é incluir na máquina de 'Receiver' o trecho a seguir que prevê a existência de "lixo" até que as duas máquinas efetivamente sincronizem.

```

from IDLE
  to SAME
    when UPPER_LEVEL. START_BCC_ACCUMULATION
      begin end;
    when UPPER_LEVEL. RECEIVE_TRANSPARENT
      begin end;
    .....

```

As duas alternativas são trabalhosas e altamente dependentes do problema. Conclui-se que para especificar sincronização de modo simples e claro tem-se que recorrer à "implementação" desta por compartilhamento de variáveis.

Em Lotos, a sincronização é feita pelo mesmo mecanismo genérico de casamento ("matching") de participação em eventos que descreve a comunicação, conforme visto no item 4.2.2. A geração não determinística de valor, a própria comunicação, síncrona, e a existência das guardas envolvem aspectos de sincronização.

Em Lotos, o problema acima de sincronização é trivialmente resolvido pela utilização do mecanismo de sincronização/comunicação síncrona: uma das sequências previstas em `Msg_assembler` é a que sincroniza pelo evento 'TOUT3s' e prossegue de acordo com o novo contexto.

#### 4.2.4 Não determinismo

Além da concorrência, ocorre não determinismo em Estelle nas seguintes situações:

a) quando há mais de uma transição de uma máquina de estados habilitada num mesmo momento. No exemplo abaixo, a partir do estado 'WAIT\_CONTROLLER\_ACK', se os dois pontos de interação referenciados ('LOWER\_LEVEL\_RX' e 'TIMER\_PORT') tiverem no topo da fila associada, respectivamente, as interações 'FRAME\_RECEIVED' e 'TIMEOUT'; e ainda, o parâmetro 'data\_in' da primeira interação for igual a 'RVI\_MSG', então, qualquer das duas transições estará habilitada para ser executada.

```

from WAIT_CONTROLLER_ACK
  when LOWER_LEVEL_RX. FRAME_RECEIVED
    (tipo_in, data_in, length_in)
    provided tipo_in = RVI_MSG
      to RESET
      begin
        .....
      end
    .....
  when TIMER_PORT. TIMEOUT
    to WAIT_ONLY_MSG_TRANSMITED
    begin
      .....
    end;

```

b) quando existem transições habilitadas simultaneamente em módulos irmãos.

c) na execução do comando "forone", quando existir mais de uma opção que satisfaça a condição "suchthat"; ou na

execução do comando "all", quanto à sequência da escolha das combinações de valores para os quais o comando associado será executado. Nos exemplos abaixo, é feita uma conexão do módulo que executa o comando "forone" com um terminal disponível qualquer. A seguir, todos os terminais desconectados são conectados (através da porta 'interface') a uma das portas representadas pelo vetor 'driver' do módulo executante, porém não é determinado quem está conectado com que porta (índice do vetor) do módulo.

```

forone    p: tipo_terminal
          suchthat p. disponível
          do    connect MY_PORT to p.TERM_PORT;

all    p: tipo_terminal
       suchthat p. desconectado
       do    begin
             connect driver [próximo] to
                   p. interface;
             próximo:= próximo + 1;
             end;

       end;

```

d) quando é incluída uma cláusula "delay" para habilitação da transição. No exemplo abaixo, a cláusula "delay" indica que, a partir do momento em que a máquina entra no estado 'TAREFA\_SOLICITADA', a primeira transição não estará habilitada antes de 'Tempo\_minimo' intervalos de tempo, e certamente estará habilitada após 'Tempo\_máximo' intervalos de tempo. O momento exato, no entanto, não está determinado. Além disto, nota-se que este momento pode influir nas chances de execução da segunda transição especificada, uma vez que, se a 'SOLICITAÇÃO\_URGENTE' surgir antes da habilitação da primeira transição, ocorrerá a



segunda transição ao invés da primeira.

```

from TAREFA_SOLICITADA
  delay (Tempo_minimo, Tempo_máximo)
  to INICIA_TAREFA
  begin end;

from TAREFA_SOLICITADA
  when MAIL. SOLICITAÇÃO_URGENTE
  to VERIFICA_SOLICITAÇÃO
  begin end;

```

Alguns fatores que restringem ou permitem restringir o não determinismo são:

a) a imposição de que, ao ser completada uma transição (por definição, atômica) todas as interações enviadas aparecem imediatamente no final das respectivas filas de destino.

b) a existência de cláusula de atribuição de prioridades entre transições de um módulo. No primeiro exemplo deste item, concernente a possibilidade de duas transições estarem habilitadas ao mesmo tempo, se fosse acrescentada a uma delas uma cláusula de prioridade 0 (mais alta), conforme abaixo, a escolha passaria a ser determinística em benefício da transição de prioridade 0.

```

from WAIT_CONTROLER_ACK
.....
  when TIMER_PORT. TIMEOUT
  priority 0
  to WAIT_ONLY_MSG_TRANSMITED
.....

```

c) a existência implícita de prioridade de transições de um módulo sobre as de seus descendentes.

d) a obrigatoriedade de execução de uma transição quando não houver outra habilitada.

A presença de não-determinismo nem sempre é visível em Estelle, e frequentemente precisa-se tomar cuidado, sendo necessária a inclusão de mecanismos de prioridade, o conceito de atividade e um certo controle na troca de mensagens pelas filas, para evitar alternativas não determinísticas indesejadas. Isto ocorre especialmente devido à possibilidade de aninhamento das cláusulas de uma transição ("from", "to", "when", ...) com qualquer ordem de embutimento. Nos exemplos anteriores, o aninhamento foi quase sempre feito na ordem "from", "when", "provided", "to", ... Considere, no entanto, o exemplo a seguir, onde existe a necessidade de especificar transições com diferentes tipos de embutimento: algumas a partir de qualquer estado (omissão da cláusula "from"), algumas aninhando a cláusula "from" após a "when" e outras fazendo o aninhamento inverso. Neste caso, pode passar despercebida a possibilidade de duas transições estarem habilitadas ao mesmo tempo. A cláusula "priority" foi usada para resolver o conflito que, neste caso, era indesejável. Através dos trechos mostrados abaixo (grande parte foi omitida) nota-se que podem existir dois eventos habilitados ao mesmo tempo, se a máquina estiver no estado 'WAITING\_UPPER\_LEVEL', e no mesmo momento uma interação 'RESET' estiver no topo da fila associada a 'UPPER\_LEVEL' e alguma interação ('TOUT3S', 'CHARAC', 'BCC') estiver em 'LOWER\_LEVEL', tendo sido dadas a estas últimas prioridade sobre a primeira. Ainda, no primeiro conjunto de transições poder-se-ia ter usado um "state-set" ao invés de explicitar os nove estados na cláusula "from". Neste caso, ficariam ainda menos aparentes os conflitos.

```

trans
when LOWER_LEVEL. TOUT3S
    from NOT_SELECTED, WAIT_PAD_EOT, GOING_TO_TMM,
        CONTROL_MODE, WAIT_END_POL_SEL_SEQ,
        DISCARD, TRANSPARENT_MONITOR_MODE,
        CU_SELECTED, GOING_OUT_TMM
    priority 0
    to NOT_SELECTED
    .....

from WAITING_UPPER_LEVEL
    priority 0
    to SAME
    .....

.....

trans
from WAITING_UPPER_LEVEL
    priority 0
    when LOWER_LEVEL. CHARAC (CH)
    .....
    when LOWER_LEVEL. BCC (OK)
    .....

.....

trans
when UPPER_LEVEL. RESET
    to CONTROL_MODE
    .....

```

Em Lotos, além da concorrência, o não determinismo pode ser expresso por:

a) geração não determinística de valor na ocorrência de eventos, conforme exemplificado no item 4.2.2.

b) especificação do evento interno I. O evento interno é uma fonte de não determinismo na medida em que

determina que a sequência de eventos que o sucede pode ou não ocorrer. É usado especialmente de forma combinada com os operadores de alternativa e desabilitação, citados nos incisos "c" e "d" abaixo.

c) operador de alternativa (<choice\_expression> e <general\_choice\_expression>), que permite escolha não determinística, conforme exemplificado a seguir. A especificação do protocolo BSC3 a nível de mensagem, sob o ponto de vista do terminal, inicia com a reação às mensagens de "polling" e "selection". Este tipo de mensagens é sempre reconhecido por uma estação, embora nem sempre se destine a ela. Isto pode ser modelado explicitamente pela inclusão do mecanismo de verificação de endereço ou conforme o trecho abaixo em que tal detalhe é ignorado, aparecendo duas alternativas não determinísticas para cada mensagem recebida, uma reconhecendo a mensagem e outra ignorando-a.

```

LOWER_LEVEL_RX ! POL; TRATA_POL
[] LOWER_LEVEL_RX ! POL; /* ignora */
[] LOWER_LEVEL_RX ! SEL; TRATA_SEL
[] LOWER_LEVEL_RX ! SEL; /* ignora */

```

É importante notar a diferença do trecho acima, não-determinístico, com o apresentado a seguir, determinístico, em que o terminal estaria sempre disposto a aceitar o convite a transmitir/receber ("polling" ou "selection"), tendo o comportamento dependente apenas da sincronização com o controlador.

```

LOWER_LEVEL_RX ! POL ( TRATA_POL;
                        [] /* ignora */ )
[] LOWER_LEVEL_RX ! SEL ( TRATA_SEL;
                        [] /* ignora */ )

```

O trecho seguinte utiliza o evento interno para prever a existência de vários modelos diferentes de

comportamento para diferentes equipamentos com pequenas variações no protocolo BSC3. Alternativas equivalentes podem ser usadas para especificação de funções de protocolo opcionais.

- I; Comportamento\_controladora\_IBM3274
- [ ] I; Comportamento\_terminal\_SCOPUS
- [ ] I; Comportamento\_software\_emulador\_terminais
- [ ] .....

d) <disable\_expression> que pode permitir, em um dado momento, a ocorrência de eventos tanto na sequência normal, como na de desabilitação, de modo análogo ao operador de alternativa.

e) opção "any" dentro da funcionalidade de "exit". Esta opção é fonte de não determinismo na passagem de valores na composição sequencial, uma vez que tal opção significa que qualquer entre um conjunto de valores (sort) é válido.

Note-se que o não determinismo em Lotos só está presente quando é desejado, ou, em outras palavras, a existência de não determinismo é perfeitamente visível na especificação. Existe, no entanto, uma construção em Lotos que pode apresentar certo problema de interpretação: a derivada de <parallel\_expression>, quando são misturados os diferentes operadores de paralelismo, conforme visto em exemplo no item 3.2.2, onde eventos em uma porta g1 podiam ou não ter a participação de uma expressão B1. Esta se constitui em uma possibilidade de não determinismo camuflada bastante perigosa. No item 4.2.1, ao exemplificar-se a concorrência em Lotos, utilizou-se uma construção semelhante a seguinte:

```

Receiver
    |[Porta_interna_1]|
Msg_assembler
    |[Porta_interna_2]|
LLC
.....

```

Nela, não é imediato perceber que, considerando as diversas possibilidades de associatividade (conforme visto no item 3.2.2), 'Receiver' pode sincronizar com 'LLC' via 'Porta\_interna\_1' ou 'Porta\_interna\_2', sem a interferência de 'Msg\_assembler'. Isto não acontece apenas porque 'Porta\_interna\_1' não é utilizada em 'LLC' e 'Porta\_interna\_2' não o é em 'Receiver'.

#### 4.2.5 Imparcialidade (Fairness)

Nenhuma das duas linguagens suporta diretamente a especificação de imparcialidade na escolha não determinística.

#### 4.2.6 Tempo

Em Estelle, o conceito de tempo aparece de forma relativa pela especificação de sequências de comandos, pela estrutura da máquina de estados e pela definição de entrelaçamento (não simultaneidade). Estelle tem uma construção para a especificação precisa do tempo que é a cláusula temporal ("delay"). Esta cláusula permite estabelecer um intervalo, após serem satisfeitas todas as demais restrições, dentro do qual a transição passará efetivamente de desabilitada para habilitada. Note-se, porém, que o tempo de seleção de uma transição em Estelle, bem como de sua execução não é passível de ser especificado, o que reduz a capacidade de especificação precisa. O tempo é dado como um número natural ou "infinito", sem menção de uma

unidade.

O seguinte exemplo mostra a descrição de um relógio em Estelle modelado para controlar "timeout" em protocolos, utilizando a cláusula "delay". O estado inicial da máquina é 'IDLE'. Existe um ponto de interação externo ('TIP'), pelo qual o módulo pode receber uma solicitação de início de contagem ('INIT'), com um parâmetro definindo o número de unidades de tempo, bem como uma solicitação de término (interrupção) de contagem ('OFF'). O tratamento destas duas solicitações tem prioridade máxima em qualquer dos estados, conforme definido nas duas primeiras transições, permitindo que haja interrupção da contagem a qualquer momento. Ao receber 'INIT', a máquina passa ao estado 'CONTA\_SEGUNDOS', iniciando a contagem de tempo para habilitação da cláusula "delay" da terceira transição. Passados 'Tempo' intervalos, a transição fica habilitada e é imediatamente executada, enviando uma indicação de "timeout" ao módulo com o qual está conectado.

```

channel Timer_interface (User, Provider);
  by User: INIT (Tempo: int);
           OFF;
  by Provider: TIMEOUT;

module Timer_type process
  (TIP: Timer_interface (Provider)
   individual queue           );

body Timer for Timer_type
  var Tempo, Duração: int;
  initialize to IDLE
  begin end;

trans
  priority 0

```

```

when TIP. OFF
  to IDLE
  begin
    Tempo := 0;
  end;
when TIP. INIT (Duração)
  to CONTA_SEGUNDOS
  begin
    Tempo := Duração;
  end;

trans
  from CONTA_SEGUNDOS
  delay (Tempo)
  to IDLE
  begin
    output (TIP. TIMEOUT);
  end;

end; /* of Timer body */

```

O conceito de tempo aparece, em Lotos, apenas relativamente, na especificação de sequências de eventos, o que aliás é um dos aspectos característicos da linguagem. Não há suporte para especificação precisa de tempo. O exemplo acima não tem, portanto, equivalente em Lotos. A possibilidade de ocorrência de "timeout" seria especificada como um evento qualquer.

#### 4.2.7 Especificação de dados

Os dados, em Estelle, são definidos do mesmo modo que no Pascal ISO: com a filosofia de associação de um dado a uma estrutura de armazenamento, com a especificação pré-definida dos tipos básicos e com operações implícitas. Os



valores computados pelas operações são armazenados em variáveis mediante a operação de atribuição. Estelle acrescenta o conceito de valor "undefined" (discutido também no item 4.2.8).

Algumas faltas que se fazem sentir, considerando a impossibilidade de definir tipos abstratos de dados, são as estruturas ilimitadas: filas e listas infinitas, especialmente "strings" de tamanho indeterminado. Na especificação da transmissão de mensagens no BSC3, o módulo 'Msg\_disassembler' recebe do módulo 'LLC' uma mensagem com um cabeçalho para o qual existem algumas variantes e um número qualquer de caracteres de texto. Esta mensagem é transmitida, caracter por caracter ao módulo 'Transmitter'. As limitações do Pascal de Estelle obrigam ao uso de uma variável do tipo "array", com tamanho pré-definido, para comunicação da mensagem entre módulos, bem como uma variável para indicar o tamanho efetivo da mensagem. Além disto o formato do cabeçalho nada mais é do que uma sequência de caracteres dentro do "array". Os caracteres da mensagem, para envio ao 'Transmitter' são acessados via indexação sobre o "array", através de um variável inteira que é incrementada a cada acesso.

Dentro do modelo é definido um tipo de dado especial que comporta a definição do estado da máquina como uma variável. São criados tipos para suporte da concorrência, quais sejam os tipos de módulo e de ponto de interação (que tem uma fila infinita associada).

O escopo de uma variável segue regras semelhantes ao Pascal, mesmo dentro da definição da máquina de estados. Quanto a visibilidade, um tipo de módulo pode usar definições de tipos (incluindo canais e módulos) declaradas por módulo ancestral, mas não pode referenciar variáveis de

ancestrais. Por outro lado, um módulo pode acessar uma variável de um filho se esta tiver sido "exportada" na definição do tipo de módulo do filho. Também pode acessar variáveis exportadas de irmãos se estes forem atividades. Existem algumas restrições quanto aos efeitos colaterais, em especial nas expressões da cláusula "provided" e quanto a impossibilidade de mudança de estado dentro de um procedimento ou função.

O suporte de dados em Lotos é feito através da descrição de tipos abstratos de dados parametrizáveis e combináveis (reusáveis), conferindo flexibilidade a especificação destes. É possível, no entanto, descrever um tipo de dados através do modelo comportamental derivado de Milner. Uma fila, por exemplo, pode ser modelada pela definição de uma <process\_definition> recursiva, onde a inserção e retirada de elementos é feita através de comunicação síncrona entre as diversas instâncias de sequências, com uso de <hiding\_expression> [apêndice C de ISO 87]. [LED 87] enfoca especificamente este aspecto de "intertwining" entre as possibilidades de especificação de dados em Lotos. A especificação de uma fila simplificada (apenas três operações) de capacidade infinita em Lotos é apresentada abaixo em duas versões: a primeira como um processo, extraído de [LED 87] e a segunda, como um tipo abstrato de dado, aqui proposta para comparar com a primeira. Nota-se que o funcionamento da fila modelada como processo é bastante difícil, sugerindo que, para cada situação de modelagem, uma alternativa normalmente se mostra bem mais adequada do que a outra. Compare-se ainda, a especificação de retirada de fila vazia, que no primeiro modelo é feita através da ausência de sincronização na porta "out", indicando que pode haver uma "espera", até que algo seja inserido, enquanto no segundo modelo, é devolvido o valor "erro" como integrante do sort "element" .

```

process Queue [in, out] :noexit (* [LED 87] *)
:= in ? x:element;
   hide mid in          ( Queue [in, mid]
                        |[ mid] |
                        cell [mid, out] (x) )

where
   process cell [in, out] (x:element)
                                       :noexit
   := out ! x;
      in ? x:element;
      cell [in, out] (x)
   endproc
endproc

type queue_type is type_element
  sorts queue
  opns vazia :-> queue
      in: element, queue -> queue
      head: queue -> element
      out: queue -> queue
  eqns forall e:element, q:queue
      ofsort element
          head (in (e, q)) = e;
      ofsort queue
          out (in (e, q)) = q;

```

Ainda, em Lotos, o problema apresentado na discussão de Estelle, de transmissão de mensagens, pode ser resolvido de maneira elegante pela especificação de um tipo de dado cujos elementos básicos são os tipos de cabeçalho, com dois parâmetros (dados do cabeçalho e texto), acessados por operações diferenciadas. O texto, por exemplo, pode ser acessado definindo-se uma operação "próximo", que devolve o

primeiro caracter ainda não lido ou "fim\_de\_texto".

É importante salientar ainda que em Lotos não existe o conceito de atribuição de valor a uma variável, caracterizando um estilo de linguagem funcional. No item 4.2.2 foi apresentado um trecho em Lotos equivalente ao módulo Receiver visto em Estelle. O procedimento 'CALCULA\_BCC', ao qual não foi dada importância naquele contexto, estava definido com um parâmetro (o caracter) e esperava-se dele o efeito colateral de atribuir a uma variável global o resultado do cálculo do "block control character" (caracter de controle do bloco, calculado a partir dos caracteres da mensagem). Desta forma foi transcrito para Lotos por simplicidade. A seguir mostra-se uma versão correta de uma das versões apresentadas de 'Receiver', considerando a característica citada.

/\* trecho equivalente ao do módulo Receiver -  
primeira alternativa \*/

```

process  RECEIVING_NORMAL (bcc)
:=      LOWER_LEVEL ! LINE ? ch: ch_sort [ch = SYN];
        Timer_port ! INIT ! 3; RECEIVING_NORMAL (bcc)
[]     LOWER_LEVEL ! LINE ? ch: ch_sort [ch NEQ SYN];
        UPPER_LEVEL ! CHARAC !ch;
        WAIT_NORMAL_CHAR_CONF (CALCULA_BCC (ch, bcc))
endproc

process  WAIT_NORMAL_CHAR_CONF (bcc)
:=      UPPER_LEVEL ! CHAR_CONFIRMATION;
        RECEIVING_NORMAL (bcc)
[]     UPPER_LEVEL ! RECEIVE_TRANSPARENT;
        WAIT_TRANSP_CHAR_CONF (bcc)
[]     .....
endproc

```

Os dados em um processo de Lotos são trazidos como parâmetros ou pela comunicação síncrona (?), conforme visto acima, uma vez que não existem variáveis. Pelo mesmo motivo não existem efeitos colaterais. Os identificadores são meras referências para especificação do direcionamento dos dados, não devendo serem confundidos com variáveis. Os valores utilizados por outros processos são, do mesmo modo, enviados como parâmetros ou pela comunicação (!). O escopo dos identificadores de valor é um trecho da especificação do processo, não se estendendo aos processos auxiliares (opção "where" em <definition\_block>). A visibilidade entre processos só se aplica quanto aos tipos e processos: A definição de tipos pode referenciar tipos previamente definidos e a definição de processos pode referenciar tipos e processos definidos dentro da mesma opção "where" ou herdados da definição de processo na qual está embutido.

#### 4.2.8 Nível de especificação

Estelle provê as seguintes facilidades para obtenção de diferentes níveis de especificação:

a) possibilidade de gerar variáveis de tipo "optional", que adicionam ao domínio original o elemento "undefined". Este elemento pode ser usado em expressões. A interpretação das expressões também é estendida, para definir o resultado de operações em que participe um valor "undefined". Não se constitui em não determinismo porque a interpretação das expressões bem como do próprio elemento "undefined" não implica no surgimento de novas alternativas não determinísticas de comportamento.

b) possibilidade de substituir um identificador qualquer por "...", ficando a cargo de uma especificação em um nível mais detalhado a substituição de cada ocorrência de "..." por um identificador, valor ou expressão válidos.

Lotos possui uma facilidade bastante poderosa, concisa e elegante para especificação em diferentes níveis de detalhamento: o evento interno I. O evento interno permite a omissão da descrição de partes do sistema, indicando as situações onde poderá ou não haver determinadas formas de comportamento, controladas pela posterior definição da parte omitida. A presença de "I" em uma sequência pode ser interpretada como: existe uma parte do sistema que não está especificada, a qual, dependendo das interações com o meio externo (ou com outras expressões concorrentes a esta) poderá possibilitar a ocorrência dos eventos especificados na expressão conseqüente a "I". Outros recursos de especificação de não determinismo podem ser usados para obtenção de diferentes níveis de especificação como no exemplo da omissão de verificação de endereço de "pooling"/"selection" no item 4.2.4.

#### 4.2.9 Nível de abstração

As seguintes características colocam Estelle como uma linguagem com nível de abstração bastante baixo, próximo à implementação:

- a) inclusão da linguagem Pascal completa.
- b) restrições na especificação de tipos de dados, conforme item 4.2.7.
- c) uso de variáveis compartilhadas para comunicação e sincronização, que nada mais é do que um método de implementação da comunicação em linguagens concorrentes.
- d) especificação da concorrência através de processos (aqui no sentido mais amplo), que a tornam comparável a uma linguagem de programação concorrente.

e) rigidez no estabelecimento dos canais de comunicação, conforme visto no item 4.2.1.

f) associação de uma fila aos pontos de interação, em detrimento de qualquer outro esquema possível, de acordo com a situação.

Estelle pode ser considerada uma linguagem de programação concorrente, exceto por algumas poucas características: variáveis "optional", opção "...", a consideração de que as filas associadas a pontos de interação tem tamanho infinito. A interpretação da hierarquia de módulos na maior parte das vezes pode ser mapeada em processos de um computador, até porque Estelle induz a modelagem de um protocolo como o conjunto de interações entre entidades de níveis adjacentes de um mesmo sistema. Exceções ocorrem com os processos que fazem a conexão entre dois sistemas (modelagem do serviço subjacente ou, no caso de entidades do nível físico, modelagem do meio de comunicação ).

A inclusão rígida do conceito de tempo, é um indicador de baixo nível de especificação, mas não é considerado aqui como baixo nível de abstração em relação a um mesmo nível de especificação.

As seguintes características conferem a Lotos um alto nível de abstração:

a) a concorrência dada pela especificação de seqüências concorrentes de eventos com sincronização, independente de como ela vai ser controlada. Há uma certa similaridade entre o surgimento das seqüências e o disparo de processos, ficando menos explicitos o controle das participações (quem participa, quando participa) na ocorrência de eventos e a passagem de valor na composição sequencial.

b) a versátil especificação de comunicação (inclusive bidirecional), sincronização e geração consensual de valor não determinística em uma só construção, vistas no item 4.2.2.

c) os tipos abstratos de dados, conforme item 4.2.7.

d) a orientação à descrição de sequências possíveis de eventos de forma geradora e não reconhecedora (procedural).

#### 4.2.10 Implementação automática

As características de Estelle, próximas a uma linguagem de programação tornam fácil a especificação de um tradutor para uma linguagem concorrente, desde que contornadas as considerações do item anterior ("optional", "...", e tamanho das filas) [VUO 88, SOU 87b].

Lotos apresentaria maiores dificuldades de implementação automática, não tendo sido esta característica suficientemente analisada neste trabalho para avaliar o grau de dificuldade.

#### 4.2.11 Concisão

Uma das piores características de Estelle é a falta de concisão: há muitas alternativas para a solução de poucos problemas, alternativas frequentemente complicadas, mistura de modelos, etc. Se não, vejam-se alguns exemplos:

a) inclusão de toda uma linguagem de propósitos gerais como o Pascal, totalmente orientada aos detalhes característicos de uma linguagem de programação.



b) inclusão de dois modelos de comunicação, nenhum dos dois suportando adequadamente sincronização.

c) o modelo de comunicação por compartilhamento de variáveis, que induz à existência de um modelo de entrelaçamento. Por que, então, não foi adotado este como único modelo de concorrência?

d) a dupla alternativa de definição do próximo estado: na estrutura da transição (razoável) ou como um comando estendido dentro do conjunto de ações. Ora, a máquina de estados existe justamente para salientar a estrutura de controle do protocolo. A determinação do próximo estado dentro do trecho em Pascal estendido descaracteriza a intenção inicial, transferindo parte da estrutura de controle para dentro do bloco de transformação de dados. Pode-se alegar um aumento de versatilidade, considerando que a transferência de controle da máquina para o bloco em Pascal estendido caracterizaria uma mudança de nível de especificação. Adicione-se a isto a possibilidade de determinar vários próximos estados na estrutura da transição (cláusula "to"). Os exemplos a seguir mostram algumas das possibilidades de assinalamento de próximo estado.

Exemplo 1: declaração única de próximo estado na estrutura de controle.

```

from CAPTURE_SYNC_SEQ
  when LOWER_LEVEL. LINE (ch)
    provided ch = SYN
      to RECEIVING_NORMAL
        begin
          output (TIMER_PORT. INIT (3));
        end;

```

Exemplo 2: declaração do próximo estado dentro do bloco em Pascal (comando 'nextstate', extensão do Pascal).

```

from CAPTURE_SYNC_SEQ
  when LOWER_LEVEL. LINE (ch)
    provided ch = SYN
      begin
        output (TIMER_PORT. INIT (3));
        nextstate = RECEIVING_NORMAL;
      end;

```

Exemplo 3: Transferência da estrutura de controle para o bloco em Pascal estendido: Ao invés de aparecer na estrutura de controle, através da especificação de diversas transições (no caso abaixo, duas), a linguagem permite que decisões de controle de transição de estados sejam feitas dentro do bloco de ação, esvaziando a função da estrutura de controle representada pela máquina de estados.

```

from CAPTURE_SYNC_SEQ
  when LOWER_LEVEL. LINE (ch)
    begin
      if ch = SYN
        then begin
          OUTPUT (TIMER_PORT. INIT (3));
          nextstate = RECEIVING_NORMAL;
        end;
      else nextstate = SAME;
        /* same é usado para
           permanecer no mesmo estado */
    end;

```

Exemplo 4: Uso de conjunto de estados com escolha de estados no bloco em Pascal.

```

stateset FROM_NOT_SELECTED =
    [ WAIT_PAD_EOT, GOING_TO_TMM, DISCARD,
      NOT_SELECTED ];
.....
from NOT_SELECTED
  when LOWER_LEVEL.CHARAC (ch)
  to FROM_NOT_SELECTED
  begin
    case ch of
      EOT: nextstate = WAIT_PAD_EOT;
      DLE: nextstate = GOING_TO_TMM;
      ETX, ETB, ENQ, ITB:
        begin
          OUTPUT (LOWER_LEVEL.
                  RECEIVE);
          nextstate = SAME;
        end;
      else: nextstate = DISCARD;
    end
  end;

```

e) existência dos dois tipos de ligações (conexão e transferência), que exige uma série de restrições, que embora razoáveis, são muitas, do tipo: quais as possíveis combinações de conexões e transferências possíveis de serem associadas a um ponto de interação? Ou: o que acontece com as interações enfileiradas em um ponto de interação quando da execução de um comando "detach"? E outras.

f) existência de duas construções para a definição de módulos (<module\_header> e <module\_body>); existência dos dois tipos de módulo (processo e atividade).

g) inclusão da cláusula de tempo na estrutura da máquina de estados, que parece causar certo desacerto com a cláusula de espera de interação ("when"), a ponto de serem mutuamente excludentes em uma especificação de transição.

A linguagem Lotos é extremamente concisa, na maior parte dos pontos analisados: expressão da concorrência, mecanismos de comunicação e sincronização, evento interno permitindo especificação em níveis de detalhamento. A necessidade de definir tipos abstratos básicos é compensada pela inclusão de uma biblioteca com sorts básicos e suas relações pré-definidos.

A falta de concisão de Estelle reflete-se no tamanho da descrição do modelo, comparado com Lotos, sem contar o modelo subjacente associado ao Pascal, que não está aqui descrito.

#### 4.2.12 Consistência

Devido ao caráter construtivo das duas linguagens a consistência é automaticamente assegurada. O problema de falta de consistência pode ocorrer em linguagens implícitas, como a Lógica temporal.

#### 4.2.13 Completeza

O elemento de determinação de falta de completeza, em Estelle, é a presença de uma interação em uma fila de um ponto de interconexão, que não esteja prevista pela máquina de estados do módulo, isto é, para a qual não exista a possibilidade de se atingir um estado onde uma transição seja habilitada pela retirada daquela interação (possivelmente pela combinação dos valores dos parâmetros). Para verificar a completeza é preciso considerar apenas as sequências consistentes com o comportamento presumido do

meio externo. A dificuldade de verificação da completeza é considerada dentro do item 4.2.16.

Em Lotos, considerando-se todas as possibilidades de sequências de participações do ambiente consistentes com seu comportamento presumido, pode-se caracterizar a falta de completeza pela possibilidade de ocorrer uma sequência de eventos após a qual se chegue a um estado em que: o meio externo "propõe" (está habilitado a participar de) a ocorrência de um evento e verifica-se que é impossível a ocorrência daquele evento a partir de então (pela impossibilidade presente e futura de participação de alguma expressão envolvida ou ausência de consenso).

#### 4.2.14 Adequação ao modelo OSI

Estelle tem uma capacidade bastante adequada de mapeamento com a estruturação do modelo OSI. Não existe o conceito de prestação de serviço, ou primitivas de serviço. Pode-se, entretanto, definir um módulo como sendo o sistema e vários tipos de módulos filhos representando os tipos de entidades de execução dos diferentes protocolos de cada nível. Os pontos de interação, com características fixas, refletiriam os "pontos de acesso ao serviço" do modelo OSI. Os canais teriam definidas como interações as primitivas correspondentes à comunicação entre entidades de diferentes níveis. Cada entidade (módulo), pode ser ainda subestruturada em um conjunto de entidades, se desejável, que no conjunto cumpririam as funções da entidade. Por outro lado, Estelle permite a criação dinâmica de módulos, que pode ser interpretada como a possibilidade de existir sistemas com diferentes conjuntos de entidades de protocolo ativas. Uma restrição, relacionada à limitação de definição de estruturas de tamanho indeterminado é a impossibilidade de especificar a existência de um número qualquer de

ligações entre entidades adjacentes (o número de pontos de interação de um módulo é fixo).

O modelo OSI da ISO, da maneira como está atualmente descrito em [ISO 84a], apesar de enfatizar que aquela descrição não deve ser tomada como alternativa de implementação, aproxima-se de uma estruturação de tipos de processos em uma linguagem concorrente. Em Lotos, devido ao maior nível de abstração em relação a Estelle, a adaptação não é tão imediata. Isto é, as alternativas de descrição de protocolos em Lotos não guardam semelhança estrutural com entidades, canais, pontos de interconexão, etc. Porém, considera-se que este é o real objetivo da ISO, a ser alcançado a partir do desenvolvimento de técnicas para formalizar a descrição do modelo de referência [ISO 85a].

#### 4.2.15 Formalismo

Tanto Estelle [ISO 85c] como Lotos [ISO 87] possuem sintaxe formalmente definida. Lotos tem a semântica formalmente e concisamente definida. Estelle ainda não a tem. Imagina-se, contudo, que uma vez definida formalmente a semântica de Estelle, esta será pouco concisa devido à falta de ortogonalidade e concisão da própria linguagem, especialmente considerada a inclusão do Pascal ISO. A ortogonalidade de Estelle é ainda reduzida em relação ao Pascal, uma vez que a várias extensões de comandos que são inseridos na sintaxe do Pascal são impostas restrições, como por exemplo a proibição de uso dos comandos "nextstate" e "output" dentro de procedimentos e funções; a proibição de efeitos colaterais nas expressões da cláusula "provided", etc. O excessivo detalhamento das construções de Estelle provém em especial do Pascal, através dos efeitos colaterais, comando "with", comando "goto" (um dos mais difíceis de formalizar) e multiplicidade de alternativas

para comandos condicional ("if" e "case") e repetitivo ("for", "repeat" e "while")

A definição formal de Lotos em [ISO 87] é dividida em duas partes: primeiro é aplicada à especificação uma função chamada "flattening", definida recursivamente sobre a definição sintática da linguagem, gerando uma especificação canônica (CLS, Canonical Lotos specification) dividida em duas partes: uma especificação algébrica canônica (CAS, Canonical algebraic specification) e uma especificação canônica de comportamento (CBS, Canonical behaviour specification). A CAS é interpretada como um mapeamento quociente sobre uma álgebra. A CBS é interpretada segundo um sistema de derivação de transições como um sistema de transições rotuladas.

O uso da instância formal permite o preciso entendimento da linguagem, mas não exclui a possibilidade de haver erros em relação à intenção original. No decorrer do trabalho foi encontrada uma discordância entre a especificação formal e o tutorial contido em um apêndice do mesmo documento [ISO 87], na interpretação de "formaleqns", cujo efeito é desprezado na formalização. Considera-se, no entanto, que a definição formal da linguagem é a instância a ser seguida, em caso de discordâncias.

#### 4.2.16 Verificação

A grande dificuldade de verificar protocolos em Estelle é a inclusão da linguagem Pascal, que implica, no mínimo, na inclusão de métodos de prova de programa. Uma opção de verificação que parece razoável consiste em verificar algumas propriedades gerais de correção, sobre a estrutura de controle das máquinas de estado, através de técnicas de análise de atingibilidade da máquina global. Aparecem aqui alguns empecilhos: o envio de interações em

uma transição, não é explicitado como uma cláusula da transição, encontrando-se embutido no bloco de comandos em Pascal extendido. Retirá-los de lá, sem considerar profundamente as construções em Pascal, implica em algum grau de imprecisão. Diferentes graus de precisão poderiam levar em conta ou não as especificações de cláusulas temporais e de prioridade. Embora isto possa parecer insuficiente a primeira vista, a tarefa de encontrar erros primários que conduzem à detecção de impasses, falta de completeza, ou inexistência de certas propriedades desejáveis, mapeáveis na estrutura de controle, é uma das que consomem mais tempo na validação manual. Considere o seguinte trecho em Estelle:

```

from WAIT_CONTROLLER_MSG
  when LOWER_LEVEL_TX. TTD_MSG
    to SAME
      begin
        Ttd_pending := true;
        OUTPUT (LOWER_LEVEL_TX. NAK_MSG);
      end;
  when LOWER_LEVEL_TX. EOT_MSG
    provided Ttd_pending
      to RESET
        begin
          .....
          OUTPUT (LOWER_LEVEL_RX.RESET);
          .....
        end;
  .....

```

Se for desconsiderada a cláusula provided e extraída a cláusula OUTPUT de dentro do trecho em Pascal, ignorando-se os outros comandos, temos uma descrição de máquina de estados pura, que podemos usar para detectar, por



exemplo que uma determinada sequência de eventos é impossível ou que um estado é inatingível. Em geral, precisaríamos ainda desprezar valor de parâmetros e outras cláusulas. Considere-se, no entanto, que na última transição, o comando "OUTPUT" pode estar embutido em um comando "while" do Pascal. Neste caso teríamos que considerar a geração de número indeterminado de interações. Este tipo de intromissão de aspectos de controle dentro da especificação de transformação de dados (também visto no item 4.2.11) deve ser evitado, se quisermos aumentar a capacidade de extração de propriedades da especificação.

A existência de uma definição formal concisa da semântica de Lotos, provê o suporte para verificação de especificações. Foi desenvolvido por Milner [Mil 80] um método de determinação de equivalência observacional de especificações em CCS que foi adaptado para a parte comportamental de Lotos [ISO 87]. A equivalência observacional consiste em determinar, genericamente, que duas especificações tem as mesmas sequências de eventos observáveis (nas portas de interação com o meio) definidas. A técnica, no entanto, não envolve a determinação de equivalência de termos da parte algébrica.

Um aspecto importante na verificação da correção refere-se as definições de tipos abstratos de dados. É simples definir uma assinatura, definindo a álgebra inicial gerada por termos. Pode, no entanto, tornar-se muito difícil em certos casos, determinar ou mesmo convencer-se de que a álgebra quociente definida a partir da assinatura e do conjunto de equações define realmente as classes de equivalência corretas para cada sort.

A possibilidade de especificar dados via especificação algébrica de tipos de dados ou por expressões

de comportamento, vista no item 4.2.7 torna difícil em geral a determinação de equivalência, incluídos todos os aspectos de Lotos (tipos de dados e comportamento) [LED 87]. Não foi feito neste trabalho um estudo aprofundado da computabilidade da equivalência comportamental.

#### 4.2.17 Adequação a objetivos

Pela grande diferença de nível de abstração, considera-se aqui Lotos muito mais apropriada do que Estelle para a documentação e divulgação de protocolos de maneira geral, principalmente de padrões nacionais e internacionais. Estelle é adequada para especificações dirigidas à implementação. A experiência de uso de Estelle para a especificação formal do protocolo BSC3 de uma estação IBM 3274, realizada em uma empresa fabricante de equipamentos de teste, mostrou-se adequada, conforme citado na introdução deste trabalho.

Para validação de idéias, Estelle poderá ser apropriada, na medida em que se puder concentrar os aspectos relevantes dentro da máquina de estados, podendo-se usar técnicas de análise de alcançabilidade. Para validação e verificação de protocolos reais, no entanto, é necessária uma técnica com forte fundamentação formal, bem como instrumental adequado para análise das propriedades (o problema não está na construção das ferramentas, mas na elaboração das teorias subjacentes). Sob este aspecto Lotos é bem mais adequada do que Estelle, mas também deixa muito a desejar até o momento em termos de ferramental.

#### 4.2.18 Adequação a usuários

Considerando as características da linguagem Estelle, sem prejuízo dos objetivos alcançados, foram detectadas especialmente as seguintes faltas na linguagem:

a) ausência de operações mais flexíveis na manipulação da fila associada a um ponto de interação. Em especial: ausência de um comando que descarte as interações presentes na fila e possibilidade de testar, em uma transição, a presença de mais de um tipo de interação em uma mesma especificação de transição. Esta falta de flexibilidade torna difíceis procedimentos de reinicialização do comportamento dos módulos, por exemplo após o recebimento de interações do tipo 'TIMEOUT', 'RESET', etc. (vide situação apresentada no item 4.2.3).

b) dificuldade de sincronizar módulos através dos pontos de interação, devido à presença das filas, conforme visto em 4.2.3).

c) a dificuldade de trabalhar com a cláusula temporal para obtenção de especificação precisa de tempo.

d) dificuldade de abstrair, na recepção de mensagens, o tamanho das estruturas de dados, pela falta de elementos para especificar "strings" de tamanho indeterminado, conforme visto no item 4.2.7.

e) dificuldade de solucionar dúvidas sobre a semântica de componentes da linguagem, devido à falta de uma instância formal.

Lotos é uma linguagem, com um estilo diferente do das linguagens convencionais de programação. E de se esperar dificuldade bem maior numa primeira aproximação do que com relação a Estelle. Uma vez dominado o estilo, no entanto, ela possui a vantagem de ser concisa, com soluções curtas e claras, embora no início, certamente mais demoradas. Com poucas exceções, ela não conduz, como acontece com Estelle, à existência de interpretações não previstas (vide item 4.2.4). Entre as dificuldades iniciais pode-se citar:

a) o estilo funcional da linguagem, com passagem de valores como parâmetros, cujo número pode ser bastante grande, pois são o único meio de transportar valores "globais" na composição sequencial de processos.

b) uso intensivo de recursão, característica associada ao estilo funcional.

c) definição dos tipos abstratos de dados, principalmente no tocante às equações, que devem definir corretamente a relação de equivalência entre os termos do tipo.

#### 4.2.19 Adequação aos diversos níveis de protocolo

Nenhuma das duas linguagens suporta especificação de características de elementos do nível físico, como tensões, frequências, ruído, etc. Quanto aos aspectos temporais, nenhuma suporta a especificação da ocorrência de eventos dentro de um intervalo preciso. Estelle permite a especificação precisa de intervalos de tempo mínimos entre ocorrência de eventos, o que a capacita a enfrentar problemas de especificação de nível de enlace e superiores. Lotos não suporta descrição precisa de tempo, enfrentando problemas na especificação de tempos mínimos de retransmissão, "timeouts", etc.

## 5 CONCLUSÃO

A grande diferença de nível de abstração entre as duas linguagens estudadas fez com que se refletisse bastante sobre a adequação ao usuário, levando às seguintes conclusões: sempre há uma rejeição inicial a mudanças no estilo de representação de sistemas, sempre há uma rejeição à adoção de um novo formalismo. No entanto, pode-se citar vários exemplos que mostram que, uma vez aceita a necessidade de uso de um formalismo, passado o processo inicial de adaptação, ele pode tornar-se agradável, dependendo apenas de sua real utilidade. Vejamos alguns exemplos: no primeiro contato com linguagens de programação os métodos de descrição sintática derivados das produções gramaticais como a BNF (Backus-Naur Form) parecem algo muito complicado, até o momento que se aprende a usá-los. A partir de então estes passam a ser a alternativa padrão para a assimilação da sintaxe de linguagens (não implicando, entretanto, em conhecimento da semântica). Um segundo exemplo é a mudança de estilo introduzida pelas linguagens não procedurais como Prolog e Lisp que tem-se difundido por representarem boas alternativas de implementação para certos tipos de sistemas. Conclui-se com isto que a análise da linguagem Lotos, quanto à adequação ao usuário não pode ser feita pela aceitação ou rejeição inicial, mas pela importância que lhe seria atribuída se já fosse difundida. E fato, no entanto, que a alternativa Lotos tem tido forte restrição de aceitação, se comparada a Estelle, devido à mudança de estilo em relação às tradicionais linguagens de programação a que os profissionais estão acostumados.

Durante o estudo da linguagem Lotos, verificou-se que, não obstante a importância dada à instância formal, também é importante uma definição informal completa da linguagem, mesmo que tenha-se que salientar que em caso de

dúvidas predomina a primeira. Não é necessário que todos os envolvidos com especificação formal tenham que se envolver com a definição formal da semântica, do mesmo modo como ocorre com linguagens tradicionais.

Um caminho que foi evitado no transcorrer deste trabalho foi o da análise de computabilidade da verificação de propriedades de especificações. Sabe-se do estudo de Teoria da Computação que o problema da determinação de equivalência entre descrições utilizando linguagens com pleno potencial de descrição de funções computáveis (linguagens de programação em geral) é indecidível, isto é, não existe procedimento mecânico ou computacional capaz de receber como entradas duas descrições e devolver sempre uma resposta (afirmativa ou negativa) para a questão de equivalência de comportamento dos dois sistemas. Quando alguém propõe como Milner [Mil 80] a determinação de equivalência observacional, fica-se com a pergunta: qual a restrição real de especificação que a linguagem impõe para que se possa chegar a tal ferramenta? No caso de Milner e de Lotos por extensão, não foi considerada a especificação de dados. Em Lotos isto implicaria na determinação de equivalência entre termos da álgebra inicial no mapeamento quociente para a álgebra do modelo. [LED 87] ao analisar este fato, diz que "não fica tão claro o uso combinado das leis de transformação da parte comportamental com as ADTs de Lotos". Esta afirmação conserva a forte possibilidade de não computabilidade. O estudo da computabilidade das linguagens de especificação formal é uma alternativa interessante, apesar de bastante pesada, de direcionamento na área.

Outra alternativa que não foi aprofundada por sua extensão é a dos diversos métodos de derivação de propriedades de especificações: análise de atingibilidade, execução simbólica, lógica temporal, transformação de

predicados, etc. Cada método tem conteúdo inesgotável para ser abordado separadamente.

Uma falta sentida durante o trabalho foi a de ferramentas de apoio à especificação em Estelle e Lotos. Este seria um conjunto de trabalhos interessante na área que vem sendo já desenvolvido, especialmente para Estelle. Ferramentas típicas incluiriam analisadores sintáticos e de semântica estática, compiladores ou semi-compiladores, ferramentas de acompanhamento de execução simbólica e simulação, editores orientados à linguagem, etc.

A análise das necessidades dos protocolos de baixo nível merece atenção, visto que a especificação de muitos aspectos não é contemplada, conforme visto no capítulo 4. Na tentativa de solucionar o problema de especificação precisa do tempo de forma elegante, surgiu a idéia de especificá-lo como um tipo abstrato de dado em Lotos, que poderia ser levada adiante em futuro trabalho. Aliás, a especificação do tempo tem produzido vários trabalhos [MAN 81, MAR 87, QUE 87, RAZ 85], associada a várias linguagens.

Outras linguagens mereceriam estudo semelhante às aqui apresentadas: SDL (Specification and Definition Language) [CCI 85], em fase de padronização pelo CCITT, Redes de Petri numéricas (NPN, Numerical Petri Nets) [WHE 85a, WHE 85b] entre outras.





## BIBLIOGRAFIA

- [BER 82] BERG, H. K.; BOEBERT, W. E.; FRANTA, W. R.; MOHER, T. G. Formal methods of program verification and specification. Englewood Cliffs, Prentice-Hall, 1982.
- [BOC 78] BOCHMANN, G. V. Finite state description of communication protocols. Computer Networks, Amsterdam, 2(4,5):361-72, Set/Out. 1978.
- [BOC 80a] BOCHMANN, G. V. ; SUNSHINE, C. A. Formal methods in communication protocol design. IEEE Transactions on Communications, New York, COM-28(4):624-31, Apr. 1980.
- [BOC 80b] BOCHMANN, G. V. A general transition model for protocols and communication services. IEEE Transactions on Communications, New York, COM-28(4):643-50, Apr. 1980.
- [BOC 85] BOCHMANN, G. V. ; CERNY, E. ; GERBER, G. ; DSSOULI, R. ; MAKSUD, M. ; PHAN, B. H. ; SARIKAYA, B. ; SERRE, J. M. Use of formal specifications for protocol design, implementation and testing. In: IFIP WG 6.1 INTERNATIONAL WORKSHOP ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 5, Toulouse-Moussac, 1985. Proceedings. Amsterdam, North-Holland, 1985. p. 137-44
- [BOL 87] BOLOGNESI, T. Introduction to the ISO specification language LOTOS. Computer Networks and ISDN Systems, Amsterdam, 14(1):25-59, 1987.

- [BRA 74] BRAINERD, W. S. ; LANDWEBER, L. H. Theory of computation. New York, John Wiley, 1974.
- [BRI 85] BRINKSMA, E. ; KARJOTH, G. A specification of the transport service in LOTOS. In: IFIP WG 6.1 INTERNATIONAL WORKSHOP ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 5, Toulouse-Moussac, 1985. Proceedings. Amsterdam, North-Holland, 1985. p. 227 - 51.
- [BUD 87] BUDKOWSKI, S. ; DEMBINSKI, P. An introduction to Estelle: A specification language for distributed systems. Computer Network and ISDN Systems, Amsterdam, 14(1):3-23, 1987.
- [BUR 85] BURKHARDT, H. J. ; ECKERT, H. Modelling of OSI communication services and protocols using Predicate/Transition nets. In: IFIP WG 6.1 INTERNATIONAL WORKSHOP ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 5, Toulouse-Moussac, 1985. Proceedings. Amsterdam, North-Holland, 1985. p. 165-92.
- [CAR 86] CARCHIOLO, V. ; FARO, A. ; MIRABELLA, O. ; PAPPALARDO, G. ; SCOLLO, G. A LOTOS specification of the PROWAY highway service. IEEE Transactions on Computers, New York, C-35(11):949-68, Nov. 1986.
- [CCI 85] CCITT. Recommendations Z.100-Z104. Functional specification and description language (SDL). In: THE INTERNATIONAL TELEGRAPH AND TELEPHONE CONSULTATIVE COMITEE (CCITT) PLENARY ASSEMBLY, 8., Malaga, Oct., 8-19, 1984. Geneva, International Telecommunication Union, 1985. Red Book, 6(6.10).

- [CHO 85] CHOI, T. Y. Formal techniques for the specification, verification and construction of communication protocols. IEEE Communications Magazine, New York, 23(10):46-52, Oct. 1985.
- [COU 84] COURTIAT, J. P.; AYACHE, J. M.; ALGAYRES, B. Petri nets are good for protocols. In: SIGCOMM '84 TUTORIALS AND SYMPOSIUM COMMUNICATIONS ARCHITECTURES & PROTOCOLS, Montréal, June, 6-8, 1984. Proceedings. Computer Communication Review, New York, 14(2), July, 1984. p. 66-74.
- [COU 87] COURTIAT, J. P. How could Estelle become a better FDT? In: INTERNATIONAL SYMPOSIUM ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, Zurich, May, 5-8, 1987. Proceedings. Rüschlikon, IBM Zurich Research Lab, 1987.
- [DIA 86] DIAZ, Michel. Petri net based models in the specification and verification of protocols. In: ADVANCES IN PETRI NETS, Bad Honnef, Sept., 8-19, 1986. Proceedings. Berlin, Springer-Verlag, 1986. 2v. v.2: Applications and relationships to other models of concurrency. p.135-170. Lecture Notes in Computer Science, 255.
- [EHR 82] EHRIG, H.; KREOWSKI, H.; MAHR, B.; PADAWITZ, P. Algebraic implementation of abstract data types. Theoretical computer science, Amsterdam, 20(3), July, 1982.

- [HOL 87] HOLZMANN, G. J. On limits and possibilities of automated protocol analysis. In: INTERNATIONAL SYMPOSIUM ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, Zurich, May, 5-8, 1987. Proceedings. Rüschlikon, IBM Zurich Research Lab, 1987.
- [HOP 79] HOPCROFT, J. E. ; ULLMAN, J. D. Introduction to automata theory, languages, and computation. Reading, Addison-Wesley, 1979.
- [ISO 83a] INTERNATIONAL STANDARDS ORGANIZATION/ TC 97/SC 21 N 1345. WG1 - formal description techniques - progress report. Dec. 1982. INWG Note 328, May. 1983.
- [ISO 83b] ISO/IS 7185. Programming language Pascal. 1983.
- [ISO 84a] ISO/IS 7498. Information processing systems - open systems interconnection - basic reference model. 1984.
- [ISO 84b] ISO/DIS 8072. Information processing systems - open systems interconnection - transport service definition. Jan. 1984.
- [ISO 84c] ISO/DIS 8073. Information processing systems - open systems interconnection - connection oriented transport protocol specification. Jan. 1984.
- [ISO 85a] ISO/TC 97/SC 21 N 421. Draft answer to question 40 - OSI - framework for formal description. Feb. 1985. INWG Note 385, Sep. 1985.

- [ISO 85b] ISO/TC 97/SC 21 N 420. Description of project 97.21.20 - formal description techniques. Feb. 1985. INWG Note 384, Sept. 1985.
- [ISO 85c] ISO/DIS 9074. Information processing systems - open systems interconnection - Estelle - a formal description technique based on an extended state transition model. Sept. 1985.
- [ISO 87] ISO/DIS 8807. Information processing systems - open systems interconnection - Lotos - a formal description technique based on the temporal ordering of observational behaviour. July, 1987.
- [JON 80] JONES, C. B. Software development: a rigorous approach. Englewood Cliffs, Prentice-Hall, 1980.
- [JUR 84] JURGENSEN, W. ; VUONG, S. T. Formal specification and validation of ISO transport protocol components, using Petri nets. In: SIGCOMM '84 TUTORIALS AND SYMPOSIUM COMMUNICATIONS ARCHITECTURES & PROTOCOLS, Montréal, June, 6-8, 1984. Proceedings. Publicado em: Computer Communication Review, New York, 14(2):75-82, 1984.
- [LED 87] LEDUC, G. J. The intertwining of data types and processes in LOTOS. In: INTERNATIONAL SYMPOSIUM ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, Zurich, May, 5-8, 1987. Proceedings. Rüschlikon, IBM Zurich Research Lab, 1987.

- [MAN 81] MANNA, Z. ; PNUELI, A. Verification of concurrent programs with temporal logic. In: Boyer, R. S. & MOORE, S. (Editors). The correctness problem in computer science. London, Academic Press, 1981.
- [MAR 87] MARSAN, M. A.; CHIOLA, G.; FUMAGALLI, A. Timed Petri net model for the accurate performance analysis of CSMA/CD bus LANs. Computer Communications, Surrey, 10(6):304-12, Dec. 1987.
- [MER 76] MERLIN, P. M. ; FARBER, D. J. Recoverability of communication protocols - implications of a theoretical study. IEEE Transactions on Communications, New York, COM-24(9):1036-43, Sep. 1976.
- [MER 79] MERLIN, P. M. Specification and validation of protocols. IEEE Transactions on Communications, New York, COM-27(11):1671-80, Nov. 1979.
- [MIL 80] MILNER, R. A calculus on communicating systems. Berlin, Springer-Verlag, 1980.
- [MOU 86] MOURA, J. A. B.; SAUVE, J. P.; GIOZZA, W. F.; ARAUJO, J. F. M. Redes locais de computadores - protocolos de alto nivel e avaliação de desempenho. São Paulo, Mc Graw Hill/EMBRATEL, 1986.
- [PET 81] PETERSON, J. L. Petri net theory and the modelling of systems. Englewood Cliffs, Prentice-Hall, 1981.

- [PRO 87] PROLO, C. A. ; TODT, E. Modelagem do método de acesso CSMA/CD a redes de computadores por meio de redes de Petri. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 7., Salvador, 11-19, jul., 1987. Anais. Salvador, SBC, 1987. p. 569-81.
- [QUE 87] QUEMADA, J. ; FERNANDEZ, A. Introduction of quantitative relative time in LOTOS. In: INTERNATIONAL SYMPOSIUM ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, Zurich, May, 5-8, 1987. Proceedings. Rüschlikon, IBM Zurich Research Lab, 1987.
- [RAY 87] RAYNER, D. OSI conformance testing. Computer networks and ISDN Systems, Amsterdam, 14(1):79-98, 1987.
- [RAZ 85] RAZOUK, R. R. & Phelps, C. V. Performance analysis using timed Petri nets. In: IFIP WG 6.1 INTERNATIONAL WORKSHOP ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 5, Toulouse-Moussac, 1985. Proceedings. Amsterdam, North-Holland, 1985. p. 561-76.
- [REI 82] REISIG, W. Petri nets an introduction. Berlin, Springer-Verlag, 1982.
- [RES 71] RESCHER, N.; URQUHART, A. Temporal logic. Wien, Springer-Verlag, 1971.
- [RUD 85a] RUDIN, H. An informal overview of formal protocol specification. IEEE Communications Magazine, 23(3):46-52, Mar. 1985.

- [SAJ 85] SAJKOWSKI, Michal. Protocol verification techniques: status quo and perspectives. In: IFIP WG 6.1 INTERNATIONAL WORKSHOP ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 5, Toulouse-Moussac, 1985. Proceedings. Amsterdam, North-Holland, 1985. p. 697-720
- [SAR 87] SARACCO, R.; TILANUS, P. A. J. CCITT SDL: overview of the language and its applications. Computer Networks and ISDN Systems, Amsterdam, 13(2):65-74, 1987.
- [SOU 87a] SOUZA, W. L. LOTOS: uma técnica para a descrição formal de serviços e protocolos de comunicação. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 5., São Paulo, 13-15, Abril, 1987. Anais. São Paulo, SBC, 1987. p. 121-44.
- [SOU 87b] SOUZA, W. L. ; FERNEDA, E. Aplicações do compilador Estelle. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 5., São Paulo, 13-15, Abril, 1987. Anais. São Paulo, SBC, 1987. p. 107-20.
- [SPE 87a] The SPECS Consortium ; BRUIJNING, J. Evaluation and integration of specification languages. Computer Networks and ISDN Systems, Amsterdam, 13(2):75-89, 1987.
- [SPE 87b] The SPECS Consortium. Evaluation and comparison of three specification languages: SDL, LOTOS and Estelle. In: SDL FORUM, STATE-OF-THE-ART AND FUTURE TRENDS, 3., Hague, 3-10, Apr. 1987. Proceedings. Amsterdam, North-Holland, 1987. p. 211-31.

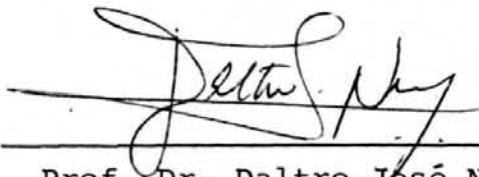


- [TUR 87]      TORSKI, W. M.; MAIBAUM, T. S. E.      The specification of computer programs. Reading, Addison Wesley, 1987.
- [VIS 83]      VISSERS, C. A.; TENNEY, R. L.; BOCHMANN, G. V.      Formal description techniques. Proceedings of the IEEE, New York, 71(12):1356-64, Dec. 1983.
- [VUO 88]      VUONG, S. T.; LAU, A. C.; CHAN, R. I.      Semiautomatic implementation of protocols using an Estelle-C compiler. IEEE Transactions on software engineering, New York, 14(3):384-93, Mar. 1988.
- [WES 86]      WEST, C. W.      A validation of the OSI session layer protocol. Computer Networks and ISDN Systems, Amsterdam, 11(3):173-182, 1986.
- [WHE 85a]      WHEELER, G. R.; WILBUR-HAM, M. C.; BILLINGTON, J.; GILMOUR, J. A.      Protocol analysis using numerical Petri nets. In: ADVANCES IN PETRI NETS, 1985. Proceedings. Berlin, Springer-Verlag, 1986. p.435-52. Lecture Notes in Computer Science, 222. Apresentado em: EUROPEAN WORKSHOP ON APPLICATIONS AND THEORY OF PETRI NETS, 6., Espoo, June, 1985.
- [WHE 85b]      WHEELER, G. R.      Numerical Petri nets - a definition. Clayton, Telecom Australia, 1985. Report 7780.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Estudo comparativo das linguagens  
Estelle e Lotos na especificação de protocolos

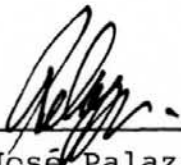
Dissertação apresentada aos Srs.



Prof. Dr. Daltro José Nunes



Prof. Dr. Maurizio Tazza



Prof. Dr. José Palazzo M. de Oliveira

Visto e permitida a impressão  
Porto Alegre, ..08../..08../..90.



Prof. Dr. Maurizio Tazza  
Orientador



Prof. Dr. Ricardo A. da L. Reis  
Coordenador do Curso de Pós-Graduação em Ciência da Computação