Universidade Federal de Santa Catarina

Curso de Pós-Graduação em Ciências da Computação - Mestrado
Sistemas de Computação - Especificação de Sistemas

Engenharia de protocolos com transformação de especificações formais: a ferramenta TranSP

por

Neilor Avelino Tonin

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

Bernardo Gonçalves Riso
Orientador

Florianópolis, Fevereiro de 1996

Universidade Federal de Santa Catarina (UFSC)
Centro Tecnológico - Departamento de Informática e de Estatística (CTC-INE)
Curso de Pós-Graduação em Ciências da Computação (CPGCC)
Campus Universitário da Trindade - Caixa Postal 476 - Florianópolis SC Brasil - CEP: 88040-970
fone: (048) 231 9738 / 231 9739 telex: (048) 2240 UFSC BR fax: (048) 231 9770

Engenharia de protocolos com transformação de especificações formais: a ferramenta TranSP

Neilor Avelino Tonin

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciências da Computação, na área de concentração Sistemas de Computação e aprovada em sua forma final pelo Curso de Pós-Graduação em Ciências da Computação.

Bernardo Gonçalves Riso, Dr.
Orientador

Rogério Cid Bastos, Dr.
Coordenador do curso

Banca Examinadora:

Bernardo Gonçalves Riso, Dr.

Rosvelter João Coelho da Costa, Dr.

Murilo Silva de Camargo, Dr.

Vitório Bruno Mazzola, Dr.

Dedico este trabalho aos meus mestres e amigos,
e em especial, à minha namorada e aos meus pais.

Agradecimentos

Agradeço ao professor Vitório Bruno Mazzola e à pesquisadora Mirela Sechi Moretti Anonni Notari pelas discussões sobre parte deste trabalho.

Agradeço aos pesquisadores Gregor von Bochmann (Universidade de Montreal - Canadá) e à professora Ana Maria Diniz Moreira (Universidade Nova de Lisboa) pelo interesse em relação a uma parte deste trabalho, pelas discussões e envio de artigos.

Agradeço aos pesquisadores José A. Mañas (Universidad de Madrid - Espanha), Hubert Garavel - VERIMAG (França), Paulo Roberto Freire Cunha (Universidade Federal de Pernambuco), Luís Ferreira Pires (Universidade de Twente - Holanda), Rosa Maria Leão Rust Carmo (LAAS -CNRS / OLC Toulouse - França) e Paul Gibson (University of Stirling - Escócia) pelo envio de especificações, normas, artigos e ferramentas.

Agradeço aos amigos e colegas de curso, aos professores do Centro Tecnológico e funcionários do Departamento de Informática e Estatística da UFSC.

Agradeço à CAPES pela bolsa de Mestrado recebida.

Agradeço a ajuda do colega Roberto M. Scheffel e do prof. Rosvelter C. da Costa na formulação das provas formais dos algoritmos apresentados nesse trabalho.

Agradeço especialmente ao pessoal da Universidade Regional Integrada - Campus de Erechim, em particular à professora Helena Confortin pela obtenção de recursos que possibilitaram a viagem a Arica, no Chile, para apresentação de um artigo e, também, pela atenção dispensada sempre que foi necessário.

Agradeço em especial, a orientação do professor Bernardo Gonçalves Riso.

Sumário

Lista de figuras

Lista de tabelas

Lista de siglas

ACSE	Association Control Service Element
CCITT	Consultative Committee for International Telegraph and Telephone
CCS	Calculus of Communicating Systems
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMISE	Common Management Information Service Element
CPGCC	Curso de Pós-Graduação em Ciências da Computação
CTC	Centro Tecnológico
Estelle	Extended Finite State Transition Language
FDT	Formal Description Techniques
ISO	International Organization for Standardization
IMAG	Institut d'Informatique et de Mathématiques Appliquées de Grenoble
ITU	International Telecommunications Union
ISO	International Organization for Standardization
LOTOS	Language of Temporal Ordering Specification
LOLA	LOTOS Laboratory
LOWE	LOTOS Windows Environment
MIB	Management Information Base
OSI	Open Systems Interconnection
SDL	Specification and Description Language
SNMP	Simple Network Management Protocol
TranSP	Transformator from Service to Protocol
TDF	Técnica de Descrição Formal

Resumo

Este trabalho apresenta um estudo na área de engenharia de Protocolos de Comunicação para redes de computadores, redes de telecomunicações e sistemas distribuídos em geral. Nesse contexto o trabalho volta-se, principalmente, para as abordagens de projeto que visam a automatização do processo de síntese de protocolos. Tais abordagens de projeto baseiam-se em regras de transformação que, uma vez aplicadas à especificação de um serviço distribuído, geram automaticamente a especificação do protocolo correspondente a esse serviço. Algumas das propostas já apresentadas nesse sentido e que foram registradas na literatura são estudadas. A evolução das investigações nessa área são acompanhadas, até o estabelecimento do estado atual dessas investigações, e o delineamento de encaminhamentos futuros de pesquisa na área. Neste trabalho apresenta-se um conjunto de algoritmos de transformação, que são aplicáveis a especificações de serviço realizadas com a Técnica de Descrição Formal LOTOS (um padrão internacional da International Organization for Standardization - ISO) para a obtenção dos protocolos correspondentes. Tais algoritmos são implementados a partir de uma ferramenta de auxílio à realização de projetos de Engenharia de Protocolos de Comunicação. Essa ferramenta, que foi desenvolvida em Borland C++ para ambiente Windows, recebeu o nome de TranSP - Transformação de especificação de Serviço em especificação de Protocolo. Esta ferramenta recebe como entrada uma especificação de serviço monolítica e gera como saída as especificações de protocolo correspondentes às duas entidades de protocolo (E1 e E2) orientadas a recursos. A ferramenta TranSP é utilizada na transformação de uma parte do serviço CMISE (Common Management Information Service Element) correspondente aos serviços M-GET e M-CANCEL-GET para o protocolo CMIP (Common Management Information Protocol) correspondente. Finalmente são apresentadas indicações para o prosseguimento das pesquisas relatadas neste trabalho assim como sugestões para a realização de extensões à ferramenta TranSP.

Abstract

This work presents a study in the area of Engineering of Communication Protocols for computer networks, telecommunication networks and distributed systems. In this context this work mainly focuses those design approaches which aim the automation of the protocol synthesis process. Such design approaches are based on transformation rules that automatically generate the correspondent protocol specification when applied to the specification of a distributed service. The early propositions described in the literature are considered. The evolution of the investigations in this area are studied until the establishment of the current research and insights for future directions. This work presents a few of transformation algorithms which are to be applicable to service specifications made with the Formal Description Technique LOTOS (an international standard from International Organization for Standardization - ISO) to obtain the correspondent protocol. These algorithms are implemented and constitute a tool for supporting the Engineering of Communication Protocols area. Such tool that was developed in Borland C++ for Windows, received the name TranSP - Transformation from Service specification to Protocol specification. This Tool receive a monolithic service specification and generate the protocol specifications of two protocol entities (E1 and E2) constraint-oriented. The TranSP tool are used to translate a part of CMISE (Common Management Information Service Element) service that match with the M-GET and M-CANCEL-GET services to relative CMIP (Common Management Information Protocol) protocol. Finally some indications for the development of more research from the results described in this work and suggestions on the realization of extensions to the tool TranSP are presented.

Palavras-chave

Sistemas distribuídos,
engenharia de protocolos,
transformações de especificações,
gerência de redes de computadores,
serviços de comunicação,
protocolos de comunicação,
especificação formal e
técnica de descrição formal LOTOS.

1. Introdução

Um dos aspectos fundamentais da natureza, que se revela imediatamente ao conhecimento humano, é o aspecto da transformação dos elementos que constituem um ambiente físico qualquer. Muito cedo se toma consciência de que o dia se transforma em noite, a qual se transforma novamente em dia, as estações do ano mudam o clima, os seres vivos sofrem mutações e assim por diante.

Aparentemente, essas transformações estão sujeitas a regras que as determinam. Tais regras, em seu conjunto, são denominadas leis da natureza (de ordem física, química, biológica e de outras ordens). A busca da definição precisa dessas leis constitui um dos objetivos da ciência.

O homem descobriu que não apenas os elementos da natureza se transformam segundo leis próprias, mas o homem pode, ele mesmo, transformar os elementos naturais através de uma intervenção consciente. Essa intervenção é o trabalho.

A realização do trabalho humano visa a transformação de uma situação dada para uma situação desejada. Conforme o tipo de situação tem-se caracterizado um trabalho no contexto agrário, industrial, social ou outro contexto de trabalho. Para obter bons resultados, nesses contextos, o trabalho deve ser organizado segundo leis próprias conhecidas da ciência ou objeto de investigação.

No contexto industrial visa-se a transformação de matéria-prima em produto. Para realizar esse tipo de transformação cada indústria vale-se de conhecimentos científicos aplicados à sua área industrial específica. Tal conhecimento constitui a tecnologia.

A tecnologia envolve não apenas conhecimento científico aplicado, mas envolve, além disso uma experiência acumulada na construção e no uso de ferramentas de auxílio ao trabalho.

No contexto da produção industrial de sistemas computacionais (software e hardware) utiliza-se um conjunto de conhecimentos científicos, que incluem técnicas de desenvolvimento, além de experiências no uso da principal ferramenta de auxílio ao trabalho nessa área: o computador.

Particularmente, as técnicas adequadas à produção de programas computacionais são estudadas na disciplina de Engenharia de Software.

Um dos tipos de software demandados pela sociedade é o software de comunicação (serviços e protocolos de comunicação utilizados em redes de computadores, sistemas de telecomunicações e sistemas distribuídos em geral). A crescente sofisticação da demanda vem impondo ao software de comunicação complexidade e porte cada vez maiores.

A análise detalhada dos requisitos que estes sistemas devem atender é dificultada devido à complexidade dos aspectos envolvidos, tais como paralelismo e sincronização, entre outros aspectos.

Na descrição informal de um sistema, uma afirmação pode ter várias interpretações. A utilização de métodos convencionais para a especigicação desses sistemas pode resultar, assim, numa descrição sujeita a erros e ambigüidades. Como garantir que a implementação deste sistema será efetuada de maneira correta?

Problemas dessa natureza levaram vários pesquisadores a perceber a necessidade da utilização de Técnicas de Descrição Formais (TDFs), matematicamente bem fundamentadas, para que os sistemas sejam concebidos e descritos de uma maneira concisa, consistente, precisa e sem ambigüidades.

Isto é possível porque as descrições realizadas com o auxílio de uma TDF não necessitam fazer referência a qualquer conhecimento informal do sistema que é descrito. As TDFs permitem

descrever o comportamento de um sistema em uma linguagem com sintaxe e semântica formais, em vez da utilização de uma linguagem natural.

A tecnologia empregada para a produção de software de comunicação adequado ao atendimento das necessidades atuais é estudada na disciplina de Engenharia de Protocolos. Esse campo de estudo combina a Engenharia de Software com o emprego das TDFs. O emprego de TDFs no projeto e no desenvolvimento de sistemas de comunicação dá à engenharia de protocolos um caráter eminentemente rigoroso e, graças a isso, possibilidades de desenvolvimento de modo sistemático que evolui naturalmente para a produção semi-automática ou automática desse tipo de sistema. Tal modo de produção de protocolos é denominada Síntese de Protocolos.

A Síntese de Protocolos procura definir regras de transformação, que aplicadas a uma especificação de serviço disponível, produzem a especificação do protocolo cujas entidades se comunicam para oferecer o serviço desejado a partir do serviço disponível. Na síntese de protocolos são empregadas TDFs como SDL (Specification and Description Language), Estelle (Extended State Transition Language) e LOTOS (Language of Temporal Ordering Specification).

Neste trabalho poderia ser utilizada qualquer TDF acima mencionada, além de outras existentes, mas a opção por LOTOS foi feita devido a se acreditar que dentro das técnicas formais, é a mais completa e atualmente, uma das mais utilizadas.

Neste trabalho apresenta-se um estudo na área de Síntese de Protocolos. Nesse estudo faz-se uma revisão de trabalhos importantes já publicados nessa área e apresentam-se algoritmos para a transformação de especificações LOTOS. Esses algoritmos são implementados e constituem uma ferramenta de projeto (ferramenta TranSP). Uma aplicação dessa ferramenta é apresentada.

1.1 Estabelecimento da problemática a ser tratada

O estabelecimento da problemática a ser tratada pode ser feito através de um exemplo simples. Suponha que se tenha uma máquina de venda de sucos com a seguinte funcionalidade: após inserir uma ficha obtém-se suco ou chá, ou pode-se solicitar a devolução da ficha, se desejado. A arquitetura inicial desse sistema é vista como um processo monolítico (veja a Figura 1.1).

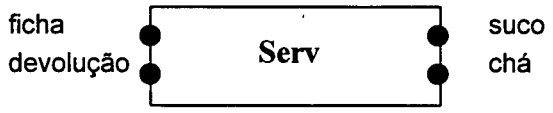


Figura 1.1 - Representação de um processo monolítico.

Obviamente, esta arquitetura inicial é apresentada em um nível muito abstrato e deve ser refinada para que possa ser implementada. Uma decisão de projeto consistiria em dividir o sistema inicial em duas partes (dividir o processo Serv em duas entidades de protocolo E1 e E2). A razão para isso seria para separar as ações pertinentes a cada sítio, pois elas são de natureza diferentes.

Uma parte do sistema (parte comercial), seria responsável pela coleta e devolução da ficha e recairia diretamente sobre o comportamento da entidade E1. A outra parte do sistema (parte de atendimento) seria responsável pela produção e fornecimento de suco ou chá e recairia diretamente sobre o comportamento da entidade E2. As duas partes são dependentes uma da outra, e deve haver sincronização entre elas através de portas de comunicação (*sync*, por exemplo). Veja a Figura 1.2.

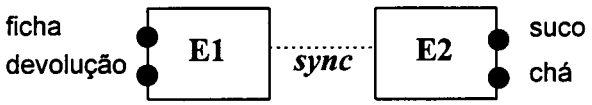


Figura 1.2 - Representação da estrutura de implementação.

O comportamento da estrutura de implementação não pode ser diferente do comportamento da arquitetura inicial, e como conseqüência então, a sincronização pela porta *sync* deve ser ocultada e o comportamento da implementação deve ter uma relação semântica com o comportamento da arquitetura inicial. Deve-se então provar que Serv ≈ **hide** *sync* **in** E1 |[*sync*]| E2.

A seguir, no *capítulo 2* é apresentada a TDF LOTOS, considerando inicialmente a parte dinâmica da linguagem (definição de processos) e posteriormente a parte estática da linguagem (tipos abstratos de dados).

No *capítulo 3* é apresentado um estudo das abordagens de diversos pesquisadores com respeito a engenharia de protocolos com transformações.

No *capítulo 4* são propostos algoritmos para transformações de especificações. Cada algoritmo aborda um operador LOTOS individualmente e explicita o tratamento dado a esse operador.

No *capítulo 5* é apresentada a implementação dos algoritmos propostos. Tal implementação constitui a Ferramenta TranSP - Transformação de especificação de Serviço em especificação de Protocolo.

No *capítulo 6* é apresentada uma aplicação prática da ferramenta TranSP na área de gerenciamento de redes (transformação de uma parte do serviço CMIS na parte correspondente do CMIP). Nesta aplicação dois serviços são abordados: o serviço M-GET e o serviço M-CANCEL-GET.

Seguem as conclusões e sugestões para futuros trabalhos (capítulo 7), anexos (capítulo 8) e bibliografia (capítulo 9).

2. A TDF LOTOS

A linguagem LOTOS (Language of Temporal Ordering Specification) [ISO 8807] é uma técnica de descrição formal (TDF) relativamente recente, que se desenvolveu entre 1981 e 1988, mas já vem ocupando espaço destacado em diversas áreas de projeto e desenvolvimento de sistemas. LOTOS vem tendo aplicações principalmente quando se trata da definição de sistemas distribuídos que envolvam fenômenos de paralelismo e cooperação, entre outros fenômenos típicos desses sistemas.

A TDF LOTOS tem dois componentes principais:

- Definição de processos: esse componente descreve os aspectos de comportamento dos processos e as interações entre eles. É a parte dinâmica da linguagem LOTOS. Baseia-se principalmente no CSP (Communication of Sequential Processes) [Hoar85] e no CCS (Calculus of Communicating Systems) [Miln80] [Miln89];
- Tipos abstratos de dados: este componente descreve os tipos de dados. É a parte estática da linguagem LOTOS. Baseia-se na linguagem ACT ONE [EhMa85]. Essas especificações de tipos de dados geralmente são longas e complexas, mas pode-se recorrer à biblioteca de tipos predefinidos oferecida por LOTOS, para agilizar a construção de tais especificações.

Importantes trabalhos vêm sendo realizados na tentativa de estender a linguagem LOTOS para torná-la apta para especificar sistemas com tempo real [CaFa94]. Um exemplo completo em LOTOS básico pode ser encontrado em [ToNo95c].

2.1. LOTOS básica

A LOTOS básica é a parte da linguagem LOTOS que trata exclusivamente da parte dinâmica (a definição dos processos) sem se preocupar com os tipos de dados.

2.1.1. Especificação de sistemas

Em LOTOS a especificação de um sistema tem o seguinte formato:

```
specification ident_spec>[<parâmetros>]:<funcionalidade>
      <definição de dados>
   behaviour
      <definição de comportamento>
   where
      ...
endspec
```

onde o `ident_spec` é o nome da especificação e `parâmetros` refere-se às portas de comunicação. A funcionalidade da especificação pode ser `exit` (no caso de comportamento finito) ou `noexit` (no caso de comportamento infinito). No caso da utilização de dados, os tipos abstratos de dados são definidos antes do comportamento (`behaviour`) da especificação. A `definição de comportamento` consiste na composição dos processos que farão parte da especificação.

Em LOTOS um processo é representado por uma "caixa preta" dotada de portas, que permitem a sua comunicação com o ambiente [Brin88] [BoBr87]. Essa comunicação determina o seu comportamento externo observável. Um processo tem o seguinte formato:

```
process <ident_process>[<parâmetros>]:<funcionalidade>:=
    <expressões_de_comportamento>
endproc
```

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

Outro meio de expressão de LOTOS é a chamada recursiva de processos para representar comportamentos que se repetem indefinidamente. Exemplo:

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

Neste exemplo o processo `Ciclico` repete a seqüência de eventos `a1, a2` infinitamente.

2.1.2. Semântica operacional de LOTOS

A partir de uma expressão B, por exemplo, deriva-se transições rotuladas que são triplas do tipo:

$B \xrightarrow{x} B'$

onde x é uma ação e B' é outra expressão de comportamento. B pode realizar a ação x e se transformar em B'. Na definição da semântica teremos:

G	indica o conjunto de portas definidas pelo usuário;
g, g*1*, ..., g*n*	indica as ações contidas em G;
i	indica as ações não - observáveis;
Act	indica o conjunto G ∪ {i} de ações definidas pelo usuário;
μ	indica as ações contidas em Act.

Uma ação especial δ que não é definida pelo usuário indica a terminação com sucesso de um processo, habilitando então um processo subsequente. Então teremos:

δ	indica a terminação com sucesso de uma ação;
G+	indica o conjunto de ações observáveis G ∪ {δ};
g+	ações contidas em G+;
Act+	indica o conjunto de ações Act ∪ {δ};
μ+	ações contidas em Act+.

O processo LOTOS predefinido **stop** é incapaz de realizar qualquer ação ou interagir com qualquer outro processo. A tabela 2.1 contém a semântica operacional formal de lotos.

Tabela 2.1 - Semântica operacional formal dos operadores LOTOS

CASO (operador)	SEMÂNTICA OPERACIONAL FORMAL
prefixo de ação	$\mu;B \xrightarrow{\mu} B$
escolha B1 [] B2	$B1 \xrightarrow{\mu+} B1'$ implica $B1 [] B2 \xrightarrow{\mu+} B1'$ $B2 \xrightarrow{\mu+} B2'$ implica $B1 [] B2 \xrightarrow{\mu+} B2'$
paralelismo B1|[g1,..., gn]|B2 onde S = [g1,..., gn]	$B1 \xrightarrow{\mu} B1'$ e $\mu \notin S$ implica $B1 \|S\| B2 \xrightarrow{\mu} B1' \|S\| B2$ $B2 \xrightarrow{\mu} B2'$ e $\mu \notin S$ implica $B1 \|S\| B2 \xrightarrow{\mu} B1 \|S\| B2'$ $B1 \xrightarrow{g+} B1'$ e $B2 \xrightarrow{g+} B2'$ e $g+ \in S \cup \{\delta\}$ implica $B1 \|S\| B2 \xrightarrow{g+} B1' \|S\| B2'$
ocultação	$B \xrightarrow{\mu+} B'$ e $\mu+ \notin \{g1,...,gn\}$ implica **hide** g1,...,gn **in** $B \xrightarrow{\mu+} B'$ $B \xrightarrow{g} B'$ e $g \in \{g1,...,gn\}$ implica **hide** g1,...,gn **in** $B \xrightarrow{i} B'$
instanciação de processo: renomeação e recursividade P[g1,..., gn]	$B \xrightarrow{g'} B'$ ϕ= [g1/g'1, ..., gn/g'n], e $g/g' \in \phi$ implica $B\,\phi \xrightarrow{g} B'\,\phi$ $B \xrightarrow{\mu+} B'$ e $\mu+ \notin \{g'1, ..., g'n\}$ implica $B\,\phi \xrightarrow{u+} B'\,\phi$ se '**process** P[g'1, ..., g'n] := **endproc**' é uma definição de processo: $Bp[g1/g'1, ..., gn/g'n] \xrightarrow{\mu+} B'$ implica $P[g1, ..., gn] \xrightarrow{\mu+} B'$
terminação com sucesso	**exit** $\xrightarrow{\delta}$ **stop**
habilitação B1 >> B2	$B1 \xrightarrow{\mu} B1'$ implica $B1 >> B2 \xrightarrow{\mu} B1' >> B2$ $B1 \xrightarrow{\delta} B1'$ implica $B1 >> B2 \xrightarrow{i} B2$
interrupção B1 [> B2	$B1 \xrightarrow{\mu} B1'$ implica $B1 [> B2 \xrightarrow{\mu} B1' [> B2$ $B1 \xrightarrow{\delta} B1'$ implica $B1 [> B2 \xrightarrow{\delta} B1'$ $B2 \xrightarrow{\mu+} B2'$ implica $B1 [> B2 \xrightarrow{\mu+} B2'$

2.1.3. Expressões de comportamento

As expressões de comportamento podem ser escritas utilizando processos básicos, processos definidos pelo projetista e operadores da linguagem LOTOS.

Os processos básicos não executam nenhum evento, quer seja observável (`Act - {i}`) ou invisível `{i}`. `Act(A)`, por exemplo, indica o conjunto de todas as ações observáveis (não incluindo, portanto, ações internas) que ocorrem no processo A. O processo completamente inativo é representado por **`stop`**. O processo **`exit`**, indica uma terminação com sucesso, habilitando a realização de algum comportamento subseqüente.

Os operadores básicos da TDF LOTOS são prefixo de ação "`;`" e escolha "`[]`". Para o caso do seqüenciamento de eventos utiliza-se o operador "`;`". Por exemplo, "`a;b`" indica a ocorrência de um evento na porta "`a`" seguido de um evento na porta "`b`".

Para representar a escolha indeterminística entre comportamentos pode-se utilizar o operador "`[]`". Por exemplo, "`B1 [] B2`" indica a escolha indeterminística entre o comportamento representado pelo identificador "`B1`" e o comportamento representado pelo identificador "`B2`".

O comportamento de um processo pode ser representado através de uma estrutura em árvore. Veja a Figura 2.1.

```
process P[ ConRes, DatReq, DisInd] :=
  ConRes; (DatReq; stop [] DisInd; stop)
endproc
```

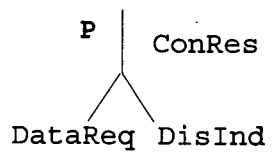


Figura 2.1 - Processo **P**.

2.1.4. Recursividade

A recursão é definida em LOTOS, pela instanciação de um processo, e é utilisada para expressar infinitos comportamentos, envolvendo seqüência de ações de tamanho finito. Considere um sistema (veja a Figura 2.2) que recebe e entrega dados infinitamente.

```
process buf[entDado,saiDado] :=
   entDado; saiDado;
   buf[entDado, saiDado]
endproc
```

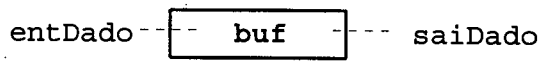


Figura 2.2 - Armazenamento temporário de dados.

2.1.5. Paralelismo

A construção de sistemas complexos a partir de sistemas mais simples pode ser realizada com a utilização de três operadores de composição:

- Paralelismo de processos independentes : `|||`
- Paralelismo de processos dependentes: `||`
- Paralelismo geral: `|[ ]|`

a) Paralelismo de processos independentes

O operador "`|||`" permite combinar processos de modo independente, isto é, de modo que esses processos podem evoluir independentemente uns dos outros. Por exemplo, na composição "`A ||| B ||| C`" os processos "`A`" , "`B`" e "`C`" podem evoluir sem compartilhar eventos. Considere o sistema triplex (com três buffers) abaixo (Veja a Figura 2.3), capaz de armazenar no máximo três dados, um em cada buffer. Um buffer é completamente independente do outro, ou seja, inicialmente pode ocorrer `in_a` ou `in_b` ou `in_c`.

```
process triplex[a_ent,a_sai,b_ent,b_sai,c_ent,c_sai]: noexit :=
    buf[a_ent,a_sai] ||| buf[b_ent,b_sai] ||| buf[c_ent,c_sai]
where
   process buf[ inp, outp]: noexit :=
      entp; saip; stop
   endproc
endproc
```

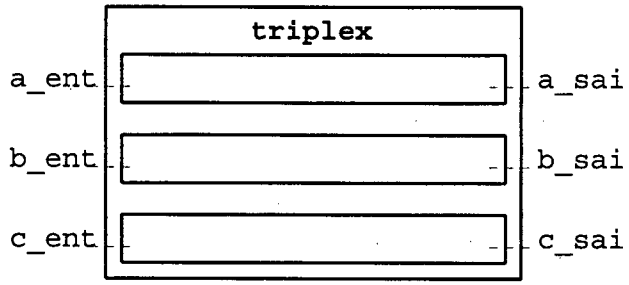


Figura 2.3 - Processo triplex.

b) Paralelismo de processos dependentes

O operador "||" permite combinar processos de modo dependente, isto é, de modo que esses processos evoluem conjuntamente. Por exemplo, na composição "C||D" os processos "C" e "D" compartilham todos os eventos.

Um típico exemplo do uso deste operador paralelo dá-se quando as restrições de um processo são determinadas por dois ou mais de seus subprocessos. Nesse caso cada subprocesso representa uma restrição.

```
process producao[a,b,c,d]:=
   item_disponivel[a,b,c,d] || item_aceitavel[a,b,c,d]
where
  process item_disponivel[a,b,c,d]:=
    a; (b; item_aceitavel[a,b,c,d] [] c; item_disponivel[a,b,c,d])
  endproc
```

```
  process item_aceitavel[a,b,c,d]:=
    a; (b; item_aceitavel[a,b,c,d] [] d; item_aceitavel[a,b,c,d])
  endproc
endproc
```

c) Operador geral de paralelismo

O operador "|[ ]|" permite combinar processos de modo geral, explicitando os nomes das portas onde os eventos são compartilhados pelos processos combinados. Por exemplo, na composição "E|[a,b]|F" os processos "E" e "F" compartilham eventos nas portas "a" e "b".

Considere como exemplo uma máquina de café (Veja a Figura 2.4) apresentada em duas partes: uma parte descrita com as ações observáveis cafe e cha. A outra, com as ações observáveis moeda e devolucao. Certamente as duas partes são dependentes uma da outra, já que a retirada de cafe ou cha implica na inserção prévia de uma moeda. Assim, deve existir sincronização das duas partes [Lang90].

```
process MaqCafe[moeda, devolucao, sincronizacao, cafe, cha] :=
  comercial[ moeda, devolucao, sincronizacao]  |[sincronizacao]|
  produtiva[ sincronizacao, cafe, cha]
  where
  process comercial[ moeda, devolucao, sincronizacao] := ...
  endproc
  process produtiva[ sincronizacao, cafe, cha] := ...
  endproc
endproc
```

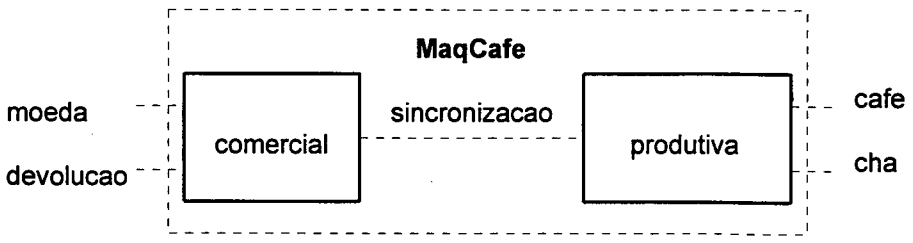


Figura 2.4 - Estrutura de uma máquina de café ou chá.

2.1.6. Ocultação de eventos

O refinamento de um sistema normalmente revela as comunicações que ocorrem entre os subsistemas que compõem esse sistema. A prova de equivalência de observação entre as duas especificações (a especificação mais abstrata e a especificação refinada) pode ser realizada se as comunicações entre os subsistemas forem ocultadas do observador. Tal ocultamento pode ser realizado com o operador "`hide ... in ...`". Por exemplo, "`hide a,b in A`" indica que os eventos que ocorrem nas portas "`a`" e "`b`" não são percebidos pelo observador do sistema.

Considere a máquina de café do exemplo anterior. A porta de sincronização das duas partes será escondida com a utilização do operador **`hide ... in`** [Lang90].

```
process MaqCafe[ moeda, devolucao, cafe, cha] :=
  hide sincronizacao in
  comercial[ moeda, devolucao, sincronizacao]
  |[sincronizacao]|
  produtiva[ sincronizacao, cafe, cha]
where
  process comercial[ moeda, devolucao, sincronizacao] := ...
  endproc
  process produtiva[ sincronizacao, cafe, cha] := ...
  endproc
endproc
```

2.1.7. Indeterminismo

Uma situação de indeterminismo é representada em LOTOS através da indicação de que há mais que uma opção de comportamento. Em LOTOS, o indeterminismo é representado com o uso do operador de escolha "`[]`".

Exemplo: usando um modelo de máquina de café imune a falhas (Veja a Figura 2.5), após a inserção de uma moeda ela deve oferecer `cafe` ou `cha`.

```
process MaqCafe[ moeda, cafe, cha] :=
  moeda;  (  cafe; MaqCafe[ moeda, cafe, cha]
          []  cha; MaqCafe[ moeda, cafe, cha] )
endproc
```

Figura 2.5 - Visualização mais abstrata da máquina de café ou chá.

Utilizando somente os operadores de seqüenciamento "`;`" e de escolha "`[]`" e com o evento interno "`i`", também se pode representar o exemplo: é inserida uma moeda, e pode-se pegar `cafe` ou `cha` e reiniciar o processo ou ocorre um evento interno que resulta na reinicialização do processo.

```
moeda;    (cafe;      MaqCafe[ moeda, cafe, cha]
          [] cha;     MaqCafe[ moeda, cafe, cha]
          [] i;       MaqCafe[ moeda, cafe, cha])
```

Pode-se admitir que a presença do evento interno `i` é importante nesse contexto. Esse evento pode ser utilizado para modelar situações especiais. Em um contexto OSI frequentemente existirá uma variante da escolha abaixo:

```
CursoNormalDeAcao  []  i; IndicacaoDeDesconexao;...
```

Expressando que um processo pode aceitar uma indicação de desconexão ainda que, em princípio, outras alternativas existam.

2.1.8. Composição seqüencial de processos

Quando um processo termina com sucesso (por exemplo, ao estabelecer uma conexão entre entidades de protocolo) esse processo pode habilitar o início de um outro (por exemplo, um processo referente à transferência de dados entre as entidades conectadas). Essa habilitação é representada através do uso do operador ">>". Por exemplo "A>>B", indica que o término do processo "A", com sucesso, habilita o início do processo "B".

No exemplo abaixo ilustra-se a utilização do operador ">>" para o caso simples de uma leitura seguida por uma escrita. São também utilizados os processos **stop** e **exit**.

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

Neste caso inicialmente é executado o evento `in` do processo `leitura[in]`. Após a execução do processo **exit** o controle é transferido para o estado inicial do processo `escrita[out]`.

2.1.9. Interrupção de processos (B1 [> B2)

A possibilidade de interrupção de um processo (digamos "A") por um outro processo (digamos "B") é representada com o uso do operador [>. Por exemplo "A [> B", indica que o processo "B" pode iniciar a qualquer momento, antes ou durante a execução do processo "A", mas não após o término da execução do processo "A".

Considere o gerenciamento de um sistema de transporte, que possui um seqüenciamento de três processos representando a fase de conexao, a fase de transferência de dados e a chamada do gerente para o reinício do processo. O processo dados pode ser interrompido, antes de chegar ao seu final, pelo processo termino. Os processos dados e gerente são habilitados pelos processos imediatamente anteriores a eles [Brin88].

```
process gerente[ConRq,ConInd,ConRes,ConCnf,DtRq,DtInd,DisRq,DisInd]:=
  conexao[ ConRq, ConInd, ConRes, ConCnf]
 >> ( dados[ DatRq, DatInd] [> termino[ DisRq, DisInd] )
 >> gerente[ConRq,ConInd,ConRes,ConCnf,DtRq,DtInd,DisRq,DisInd]
where
  process conexao[ ConRq, ConInd, ConRes, ConCnf] :=  ...
  endproc
  process dados[ DatRq, DatInd] :=  ...
  endproc
  process termino[ DisRq, DisInd]:=  ...
  endproc
endproc
```

2.1.10. Resumo dos operadores LOTOS

A seguir é apresentado um resumo dos principais operadores LOTOS, com o significado de cada operador. Veja a Tabela 2.2.

Tabela 2.2 - Resumo dos operadores LOTOS.

OPERADOR	SIGNIFICADO
;	seqüenciamento de eventos
[]	escolhas indeterminísticas entre comportamentos
|||	composição de processos independentes
||	composição de processos dependentes
|[ ]|	composição geral
hide... in...	ocultação de ações nas portas especificadas
>>	habilitação de processos
[>	interrupção de processos

3. Engenharia de protocolos com transformação

Nesta sessão, inicialmente são apresentados alguns conceitos utilizados na área de engenharia de protocolos e, a seguir, as abordagens de diversos pesquisadores sobre transformações de especificações LOTOS.

Um protocolo de comunicação consiste em um conjunto de regras que governam a troca ordenada de mensagems entre os componentes de um sistema a fim de fornecer um especificado conjunto de serviços para os usuários de serviço localizados em diferentes pontos de acesso [PrSa91].

A especificação de um serviço de comunicação descreve as funções distribuídas fornecidas por esse sistema de comunicação para seus usuários de serviço [PrSa91].

A especificação de um protocolo de comunicação descreve o comportamento das entidades deste protocolo, cada uma delas servindo um ponto de acesso ao serviço particular [PrSa91].

3.1. Abordagem de Gregor von Bochmann e Reinhard Gotzhein

Estes pesquisadores apresentaram um algoritmo para obter uma especificação de protocolo a partir de uma especificação formal do serviço. A metodologia aplicada exige, além da especificação de serviço, as informações sobre os locais onde as primitivas de serviço são executadas (SAPs - Pontos de acesso ao serviço) [BoGo86][BoGo92].

As expressões que descrevem os serviços incluem as interações entre as primitivas de serviço e os operadores de seqüenciamento ";", paralelismo dependente "||" e escolha indeterminística entre as sub-expressões de serviço "|".

O local onde uma primitiva de serviço deve ser executada é definido através de um identificador. Por exemplo a notação a_4 significa que um usuário do serviço executa uma primitiva de serviço a no SAP_4. Ao projetar-se em um local, a especificação de serviço gera o comportamento de uma entidade de protocolo com as peculiaridades do local.

O serviço é definido por uma expressão de comportamento cuja sintaxe é definida pelas 4 regras de produção da seguinte gramática livre de contexto[1] :

1. Para cada símbolo terminal `x` em {a,b,...} `e-->x`
2. `e --> e;e`
3. `e --> e-->e||e`
4. `e --> e|e`

O algoritmo de derivação

A ocorrência de eventos de comunicação entre as entidades de protocolo são representadas pelas ações de envio "`s`" e recebimento "`r`" de mensagens.

Por exemplo, no caso de uma expressão de serviço `es=a1;b2`, a entidade de protocolo 1 deve executar a primitiva de serviço `a1` e enviar a mensagem de sincronização `s2` para `b2`. Por sua vez, para que a entidade de protocolo 2 execute a primitiva de serviço `b2`, é preciso que, antes, essa entidade tenha recebido uma mensagem de sincronização `r1` da entidade de protocolo 1. Veja a Figura 3.1.

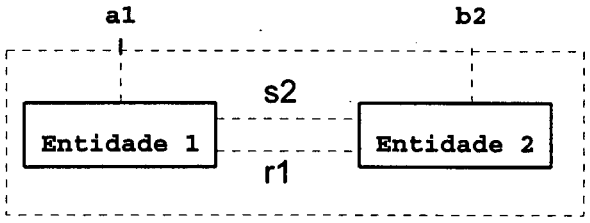


Figura 3.1. - Entidades de protocolo para o caso do serviço `es=a1;b2`.

[1] Gramática livre de contexto: denotada por G = (N,T,P,S), onde N e T são conjuntos distintos de variáveis e terminais, respectivamente, P é um conjunto finito de produções, cada uma da forma A::=α onde A é uma variável do conjunto N e α é uma cadeia de símbolos de (N∪T). S é uma variável especial denominada símbolo inicial [HoUl79].

Aplicando as regras anteriores 1-4 para um não-terminal (e), obtém-se uma árvore sintática para cada expressão de serviço 'e_S' (que contém somente símbolos terminais). Para cada nó são definidos os *atributos sintetizados* e os *atributos herdados*. Os *atributos sintetizados* passam as informações para cima da árvore (S,E) e são avaliados a partir das folhas, e os *atributos herdados* passam as informações para baixo da árvore (P,F) e são avaliados a partir da raiz. Veja as Figuras 3.2 e 3.3.

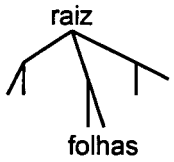


Figura 3.2. - Representação de uma árvore sintática.

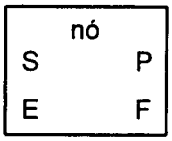


Figura 3.3. - Atributos da árvore sintática.

Os atributos S, E, P e F estão relacionados com a detecção da ocorrência dessas mensagens de sincronização. Assim, os atributos S e F detectam mensagens de envio "s" e os atributos E e P detectam mensagens de recepção "r".

A seguir são apresentadas as regras de avaliação dos atributos. Veja a Tabela 3.1.

Tabela 3.1 - Regras de avaliação dos atributos.

Regra	S	E	P	F
1			P(x) := P(e) p/ todo x em {ai,bj,...}	F(x) := F(e) p/ todo x em {ai,bj,...}
2	S(e) := S(e1)	E(e) := E(e1)	P(e1) := P(e) P(e2) := E(e1)	F(e1) := S(e2) F(e2) := F(e)
3	S(e) := S(e1)"||"S(e2)	E(e) := E(e1)"||"E(e2)	P(e1) := P(e2) := P(e)	F(e1) := F(e2) := F(e)
4	S(e) := S(e1)	E(e) := E(e1)	P(e1) := P(e2) := P(e)	F(e1) := F(e2) := F(e)

Em seguida apresentam-se as regras de transformação. Veja a Tabela 3.2.

Tabela 3.2 - Regras de transformação.

Regra	Tp
1	Tp(e) := se local(x) = "p" então P(x) "; x ;" F(x) senão "vazio" para todo x em {ai,bj,...}
2	Tp(e) := Tp(e1) ";" Tp(e2)
3	Tp(e) := Tp(e1) "||" Tp(e2)
4	Tp(e) := Tp(e1) "|" Tp(e2)

Exemplo 1 - Considere o caso da especificação de serviço e_s = a1 ; b2 , onde o serviço realizado por um provedor de serviço é visto como uma caixa preta, com comunicação com o meio através dos SAPs (Pontos de acesso ao serviço). Para cada entidade de protocolo são definidas especificações derivadas, chamadas Ti(es). Veja a Figura 3.4.

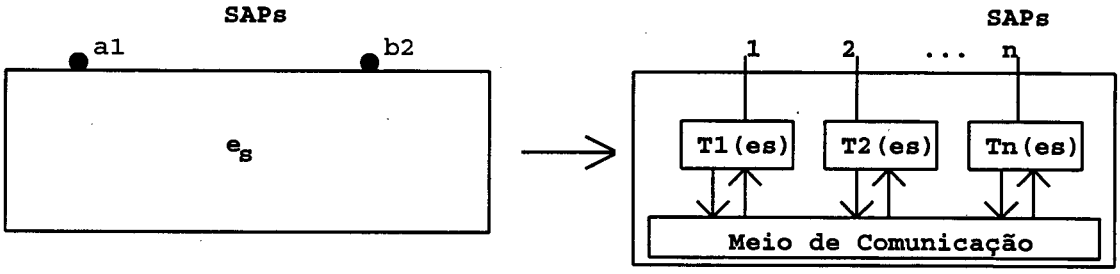


Figura 3.4. - Arquitetura de Serviço e representação do Protocolo derivadas.

Calcula-se os atributos dos nós da árvore de sintaxe. Veja a tabela 3.3.

Tabela 3.3 - Atributos dos nós da árvore de sintaxe

nodo \ atributo	S	E	P	F
1	"s_1"	"r_2"	-	-
2	"s_1"	"r_1"	-	"s_2"
3	"s_2"	"r_2"	"r_1"	-
4	não definido	não definido	-	"s_2"
5	não definido	não definido	"r_1"	-

Aplicando-se as regras de transformação, obtém-se a derivação das especificações de protocolo para os SAPs 1 e 2:

$T1(e_s)$ = T1 (a1; b2)	$T2(e_s)$ = T2 (a1; b2)
= T1 (a1) ";" T1(b2)	= T2 (a1) ";" T2(b2)
=P(a1)";a1;" F(a1) "; vazio"	="vazio;" P(b2) ";b2;" F(b2)"
= "vazio ; a1 ; s_2; vazio"	= "vazio ; r_1 ; b2; vazio"
= a1; s_2	= r_1; b2

A entidade de protocolo no SAP_1 executa a ação "a1" e envia a mensagem de sincronização "s_2" para o SAP_2, enquanto que a entidade de protocolo no SAP_2 recebe a mensagem de sincronização "r_1" e, após, executa a operação "b_2".

3.2. Abordagem de Gregor von Bochmann, Perhat Khendek e Christian Kant

Estes pesquisadores descreveram um algoritmo para a derivação de especificações de entidades de protocolo a partir da especificação de serviço. Uma particularidade dessa abordagem constitui a consideração de um meio de comunicação subjacente confiável e uma quantidade não definida de pontos de acesso ao serviço (SAPs) [KhBo89].

Uma especificação de serviço define um ordenamento (seqüência) para a execução de primitivas de serviço em diferentes pontos de acesso ao serviço (SAPs) utilizando operadores para execuções seqüenciais, paralelas e alternativas. Através da troca apropriada de mensagens de sincronização, as entidades de protocolos derivadas asseguram o ordenamento correto dessas primitivas de serviço.

Um provedor de serviço é visto como uma caixa preta, com comunicação com o meio através dos SAPs (Pontos de Acesso ao Serviço). Veja a Figura 3.5.

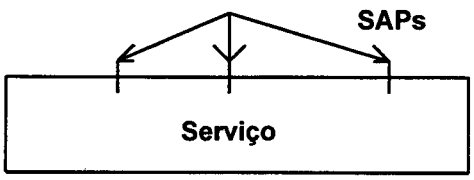


Figura 3.5 - Arquitetura de Serviço.

Para cada entidade de protocolo é definida uma especificação, sendo que a arquitetura do sistema derivado apresenta-se como na Figura 3.6.

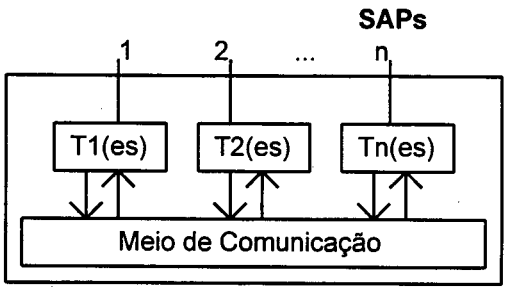


Figura 3.6 - Especificações derivadas.

Esta abordagem, introduzida em [BoGo86] e estendida posteriormente [KaHi90], [KaHi92] e [KhBo94] é tal que somente a especificação de serviço (Figura 3.5) é requerida. Além disso, parâmetros de entrada e saída são considerados, enquanto que falhas de subsistema e canais sem confiabilidade não são considerados.

Uma aproximação similar foi feita em [ChLi 88a]. Nessa aproximação para a derivação de especificações de protocolos, a partir de uma dada especificação de serviço, uma designação de interações para os diferentes pontos de acesso ao serviço deve ser dada. A derivação algorítmica fornece especificações de todas as entidades de protocolos servindo pontos de acesso diferenciados.

O algoritmo foi implementado em Prolog junto com traduções entre um subconjunto da linguagem de especificação LOTOS [BoBr87] e uma linguagem de especificação de serviço particular [Khen89]. Entretanto, o método de derivação tem algumas limitações. Uma limitação é que deve-se assumir o meio de comunicação como confiável. A outra é uma certa restrição com relação ao poder da linguagem usada para as especificações de serviços. Somente três operadores foram considerados: "`;`" para execução sequencial, "`[]`" para alternativas e "`|||`" para paralelismo independente.

O algoritmo de derivação

O algoritmo de derivação de protocolo transforma construções realizadas com uma certa linguagem e que podem ser utilizadas para escrever uma especificação de serviço (ponto inicial para a derivação do protocolo). Foi adotada a linguagem de especificação definida pelas seguintes regras de sintaxe:

```
(1) Service-Def   ---> Proc-Def
(2) Proc-Def      ---> PROC Proc-Id = e END          Proc-Def1
(3) Proc-Def      ---> PROC Proc-Id = e END
(4) e    ---> Proc-Def
(5) e    ---> Event-Id
(6) e    ---> e1 ; e2
(7) e    ---> e1 ||| e2
(8) e    ---> e1 [] e2
(9) e    ---> ( e1 )
```

A definição de diversos procedimentos (regras (1) até (3)) bem como a sentença chamando a execução de um procedimento (regra (4)) são as extensões que tornam possível definir comportamentos infinitos tais como no seguinte exemplo:

Especificação de serviço:

```
PROC A=( a1 ||| b2 ||| c3 )    ; B END
PROC B=( e3 ||| ( b3 [] d3 )) ; A END
```

O serviço é definido pelo comportamento do primeiro processo, processo "A" neste exemplo. O comportamento dos processos A e B são descritos abaixo:

Procedimento A:

As primitivas de interação a^1, b^2 e c^3 são executadas independentemente (paralelamente) nas portas 1, 2 e 3 respectivamente.

Procedimento B:

Primeiro, a primitiva de interação e^3 é executada na porta 3, em paralelo com a primitiva c^3 ou d^3 (a escolha entre as primitivas de interação c^3 e d^3 é feita na porta 3).

As especificações de protocolo resultantes são escritas na mesma linguagem da especificação de serviço, exceto que adições de primitivas para a troca de mensagens de protocolo são introduzidas.

As seguintes restrições são impostas na forma de especificações de serviço, a fim de simplificar a derivação de protocolo. Para cada subexpressão na forma "`e1 [] e2`", contida na especificação de serviço, as restrições abaixo devem ser satisfeitas:

R1: Todas interações iniciais de `e1` e todas as interações iniciais de `e2` devem ser associadas com a mesma porta.

R2: O conjunto de portas de interações finais de `e1` e de `e2` devem ser iguais, a menos que um deles esteja vazio.

Uma vez que os atributos estejam avaliados, eles são utilizados para derivar as especificações de entidades de protocolo. Para cada SAP, identificado pela porta `p`, a especificação da entidade de protocolo naquela porta é obtida (aplicando-se uma função denominada Tp) para o nodo raiz da especificação de serviço.

O resultado é uma string (seqüência) de caracteres na forma de uma especificação, contendo um conjunto de processos com os mesmos identificadores da especificação de serviço original. Entretanto, a definição de seu tronco (corpo principal) é trocada. Somente as interações de serviço, ocorrendo na porta para a qual a entidade de protocolo é derivada, será incluída na

especificação de protocolo (veja regra (5)) e sentenças adicionais para enviar e receber mensagens de sincronização serão incluídas.

Regras de Produção

```
(1) Service-Def  ---> Proc-Def
(2) Proc-Def     ---> PROC Proc-Id = e END          Proc-Def1
(3) Proc-Def     ---> PROC Proc-Id = e END
(4)  e     ---> Proc-Def
(5)  e     ---> Event-Id
(6)  e     ---> e1 ; e2
(7)  e     ---> e1 ||| e2
(8)  e     ---> e1 [] e2
(9)  e     ---> ( e1 )
```

Função Tp

```
(1)  Tp(Service-Def)  := Tp( Proc-Def)
(2)  Tp(Proc-Def)     := "PROC" Proc-Id "=" Tp(e) "END"  Tp(Proc-Def1)
(3)  Tp(Proc-Def)     := "PROC" Proc-Id "=" Tp(e) "END"
(4)  Tp(e)   := Proc-Id
(5)  Tp(e)   := if place(Event-Id) = p then Event-id else "empty"
(6)  Tp(e)   := Tp(e1) ";" if p ∈ Ep(e1) then trans(FP(e1),N(e1)) else
     "empty" ";" if p ∈ Sp(e2) then rec(P(e2)) else "empty" ";" Tp(e2)
(7)  Tp(e)   := Tp(e1) "|||" Tp(e2)
(8)  Tp(e)   := Tp(e1) "[]" Tp(e2)
(9)  Tp(e)   := "(" Tp( e1 ) ")"
```

Exemplo:

Considere novamente a especificação de serviço:

PROC A=(a^1 ||| b^2 ||| c^3) ; B END

PROC B=(e^3 ||| (b^3 [] d^3)) ; A END

A derivação algorítmica leva à seguinte especificação de protocolo:

Entidade de protocolo no sítio 1:

```
PROC A = a1 ; s3(2) ; B   END
PROC B = r3(13) ; A       END
```

Entidade protocolo no sítio 2:

```
PROC A = b2 ; s3(2) ; B   END
PROC B = r3(13) ; A       END
```

Entidade protocolo no sítio 3:

```
PROC A =   c3 ; s3(2) ; (r1(2) ||| r2(2) ||| r3(2) );   B END
PROC B = ( e3 ||| ( b3 [] d3 )) ;
     (s1(13) ||| s2(13) ||| s3(13) ) ;
     r3(13) ;  A          END
```

A abordagem destes autores considera a derivação automática da especificação de protocolo correspondente a uma especificação de serviço dada. A especificação de protocolo derivada automaticamente apresenta-se livre de impasses e recepções não-especificadas, etc. Essa abordagem fornece ainda as interações de serviço (nos diferentes pontos de acesso ao serviço) na ordem especificada pela especificação de serviço dada. A necessária sincronização de mensagens entre as diferentes entidades de protocolo são determinadas automaticamente.

É apresentado também um algoritmo simplificado de derivação, que é muito mais simples de entender que um outro apresentado anteriormente em [BoGo86]. Ele fornece especificações de protocolos que são mais otimizadas. Outra importante diferença é a extensão do algoritmo para lidar com especificações de serviço que contêm ciclos e chamadas de processos recursivas. Juntamente com os outros operadores (";" , "[]" e "|||") isto permite à linguagem poder similar a LOTOS ou CCS [Miln80].

O algoritmo de derivação de protocolos assume que a comunicação subjacente é confiável. Esta suposição é usualmente satisfeita para protocolos acima do nível do protocolo de Transporte

OSI. Acredita-se que este algoritmo pode ser utilizado para automatizar o projeto de protocolos de Aplicação, num contexto onde as especificações de serviço mudam freqüentemente.

3.3. Abordagem de Rom Langerak

Este pesquisador apresentou algoritmos para transformar uma expressão de comportamento LOTOS em duas expressões [Lang90]. Cada uma das expressões obtidas corresponde a um processo. Como cada processo está situado em um sítio diferente, fica distribuída a funcionalidade do sistema original.

A abordagem proposta por esse autor permite desenvolver o projeto de sistemas em duas fases. Na primeira fase realiza-se a coleta das requisições, e obtém-se como resultado a *arquitetura inicial* (que é a especificação da funcionalidade do sistema em alto nível de abstração). A segunda fase é a fase de *implementação*.

O sistema que este pesquisador adotou para ilustrar o uso de sua abordagem foi uma máquina de preparação e venda de café, chá ou sopa, com a seguinte funcionalidade: após inserir uma moeda obtém-se café ou chá; após inserir duas moedas obtém-se sopa. Pode-se solicitar a devolução das moedas, se desejado. A arquitetura inicial desse sistema é vista como um processo monolítico (veja a Figura 3.7).

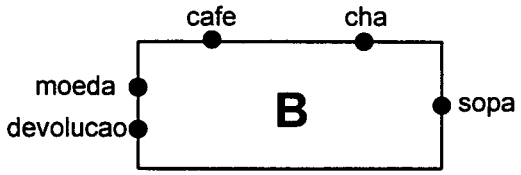


Figura 3.7 - Representação da máquina de café, chá ou sopa.

O autor descreve o processo B através de escolhas entre eventos que guardam comportamentos subseqüentes (prefixação de eventos).

$B=\Sigma\ \{g_i; B_i \mid i \in I\}$ para o conjunto de índices I. Cada B_i é um identificador de processo ou uma expressão na forma de prefixo de ação.

```
B = moeda; ( cha; B [] cafe; B  [] devolucao; B
           [] moeda; (sopa; B [] devolucao; B) )
```

O conjunto de ações observáveis que ocorrem em qualquer parte de B é denominado Act(B). O conjunto Act(B) é subconjunto de um universo de ações denominado A. Esse universo de ações A é dividido em A1 e A2, sendo que $A1 \cup A2 = A$ e $A1 \cap A2 = \varnothing$.

A primeira decisão de projeto consiste em dividir o sistema em duas partes (dividir o processo B em B1 e B2). Uma parte do sistema (parte comercial), é responsável pela coleta da moeda e pela devolução da moeda (B1). A outra parte do sistema (parte de atendimento), é responsável pela produção e fornecimento de café, chá ou sopa (B2). As duas partes são dependentes uma da outra, e sua sincronização é feita por uma porta chamada *sync*, compartilhada por ambos os processos. Veja a Figura 4.8.

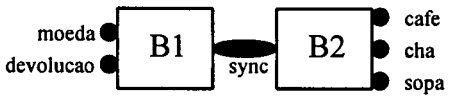


Figura 3.8 - Representação da estrutura de implementação.

Para decompor a funcionalidade do sistema, são criadas duas regras de transformação, T1 e T2. Dado um particionamento de A, as transformações T1 e T2 fornecem B1 e B2 para todo B, ou seja T1(B)=B1 e T2(B)=B2.

O autor desenvolve os algoritmos em três estágios de crescente complexidade.

- **Primeiro estágio:** nesse estágio assume-se que B não contém eventos internos (i) e não permite a escolha entre as ações de A1 e A2. Dessa forma, após um evento ser escolhido pelo ambiente, T1(B) e T2(B) sincronizam-se através da troca de mensagens na porta *sync*.

Algoritmo 1:

Se $B = \Sigma\{a_i;\ B_i \mid i \in I\}$ **então**
  $T1(B) = \Sigma\{a_i;\ sync!m_i;\ T1(B_i) \mid i \in I\}$
  $T2(B) = \Sigma\{sync!m_i;\ T2(B_i) \mid i \in I\}$

Se $B = \Sigma\{b_j;\ B_j \mid j \in J\}$ **então**
  $T1(B) = \Sigma\{sync!m_j;\ T1(B_j) \mid j \in J\}$
  $T2(B) = \Sigma\{b_j;\ sync!m_j;\ T2(B_j) \mid j \in J\}$

A seguir, é apresentado um novo processo B, mais complexo, permitindo a escolha entre as ações de A1 e A2.

```
Dado B= (a1; (b1;a1;B [] b2;a1;B)
       [] a2; a1; B)
```

onde A={a1; a2; b1; b2}. A é decomposto em A1={a1; a2} e A2={b1;b2}, ou seja, a1;a2 ocorrem no sítio 1 e b1;b2 ocorrem no sítio 2. Dessa maneira, deve haver uma sincronização entre B1 e B2 (Veja a Figura 3.9).

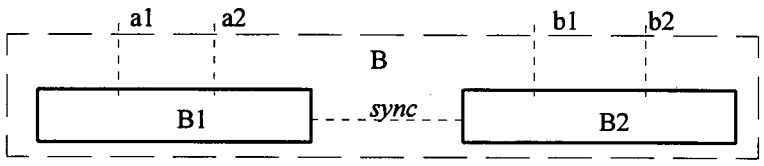


Figura 3.9 - Ocorrência de a1 e a2 no sítio 1 e b1 e b2 no sítio 2.

O algoritmo inicial é apresentado da seguinte maneira:

```
T1(B) = a1; sync!ma1; (sync!mb1; a1; sync!ma1; T1(B)
       [] sync!mb2; a1; sync!ma1; T1(B) )
      [] a2; sync!ma2; a1; sync!ma1; T1(B)
T2(B) = sync!ma1; (b1; sync!mb1; sync!ma1; T2(B)
       [] b2; sync!mb2; sync!ma1; T2(B) )
      [] sync!ma2; sync!ma1; T2(B)
```

e então é otimizado a fim de que T1(B) e T2(B) sejam simplificados.

```
T1(B) = a1; sync!ma1; (sync!mb; a1; T1(B)  [] a2; a1; T1(B) )
T2(B) = sync!ma1; (b1; sync!mb; T2(B)      [] b2; sync!mb; T2(B) )
```

O comportamento inicia em `T1(B)` através de `a1` ou `a2`. A sincronização de `T1(B)` com `T2(B)` é feita através da emissão da mensagem `m12` na porta *sync*, e a sincronização de `T2(B)` com `T1(B)` é feita através da emissão da mensagem m21 na porta *sync*.

- **Segundo estágio:** nesse caso assume-se que `B` não contém eventos internos (`i`), mas é permitida a escolha entre eventos de `A1` e `A2`. Considere, por exemplo, `B = a [] b`, sendo que `a ∈ A1` e `b ∈ A2`. Se `a` e `b` são oferecidos, uma vez que o ambiente realiza a escolha de um deles (`a` por exemplo), o evento `b` não pode mais ser oferecido. Tal restrição é obtida através de um mecanismo de polling (convite seletivo alternado). Veja o funcionamento desse mecanismo nas Figuras 3.10 e 3.11:

Figura 3.10 - Funcionamento do mecanismo de polling (convite seletivo alternado).

Figura 3.11 - Funcionamento do mecanismo de polling (convite seletivo alternado).

Algoritmo 2:

dado B= $\Sigma\{a_i;\ B_i \mid i \in I\}$ [] $\Sigma\{b_j;\ B_j \mid j \in J\}$.
então
T1(B) = $\Sigma\{a_i;\ sync!m_i;$ **T1(B$_i$)** $\mid$ **i** $\in$ **I**$\}$
[] *sync*!poll; ($\Sigma\{sync!m_j;$ **T1(B$_j$)** $\mid$ **j** $\in$ **J**$\}$
[] *sync*!poll; **T1(B)**)
T2(B) = $\Sigma\{sync!m_i;$ **T2(B$_i$)** $\mid$ **i** $\in$ **I**$\}$
[] *sync*!poll; ($\Sigma\{B_j;\ sync!m_j;$ **T2(B$_j$)** $\mid$ **j** $\in$ **J**$\}$
[] *sync*!poll; **T2(B)**)

As mesmas otimizações realizadas para o algoritmo anterior aplicam-se a este algoritmo. Com o algoritmo 2 e suas otimizações obtém-se o resultado apresentado nas Figuras 3.12 e 3.13.

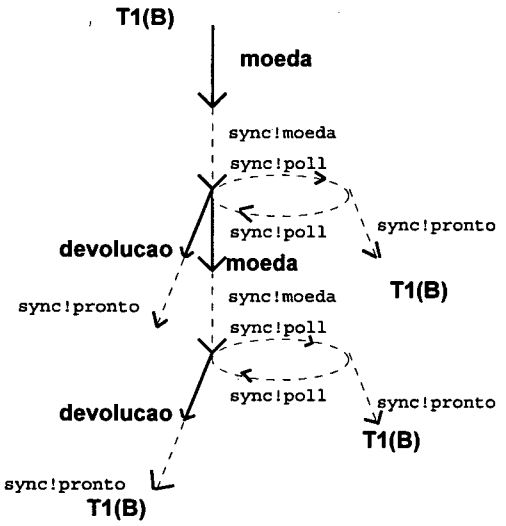


Figura 3.12 - Comportamento da entidade de protocolo T1(B).

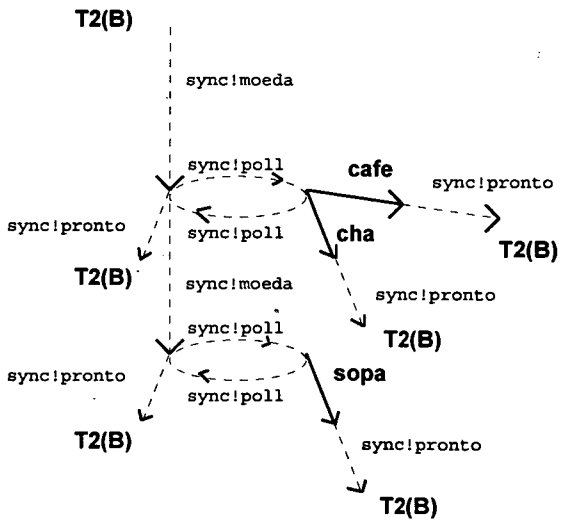


Figura 3.13 - Comportamento da entidade de protocolo T2(B).

- **Terceiro estágio:** nesse último estágio os algoritmos são semelhantes aos anteriores. Considera-se, entretanto, a escolha entre eventos de A1 e eventos de A2 e também a ocorrência do evento interno (i).

  O evento interno (i) pode ser utilizado para várias finalidades. No caso do recebimento de uma moeda falsa, o reconhecimento da moeda como falsa poderia ser indicada com a ocorrência de um evento interno (i) na parte comercial da máquina (**T1(B)**). No caso da máquina ficar sem água e assim impossibilitada de produzir café, chá ou sopa, seria natural situar um evento interno (i) na parte de atendimento (**T2(B)**).

  Portanto, a divisão dos eventos internos, que ocorrem em **T1(B)** e **T2(B)**, deve ser especificada. Se o comportamento de B após os eventos internos é indexado por **K**, então **K** deve ser particionado em $\mathbf{K_1}$ e $\mathbf{K_2}$. O índice de $\mathbf{K_i}$ (**i**=1 ou **i**=2) indica em que comportamento o evento interno é adicionado a (**Ti(B)**), ou seja, utiliza-se $\mathbf{K_1}$ se **i** $\in$ **T1(B)** e $\mathbf{K_2}$ se **i** $\in$ **T2(B)**.

A especificação inicial é alterada, considerando agora a ocorrência do evento interno (**i**):

```
B = moeda; (   cha; B   [] cafe; B   [] devolucao; B
       [] moeda; (sopa; B [] devolucao; B )  )
       [] i; devolucao; B
```

Algoritmo 3:

dado $B = \Sigma\{a_i; B_i \mid i \in I\}$
$\quad [] \Sigma\{b_j; B_j \mid j \in J\}$
$\quad [] \Sigma\{i; B_k \mid k \in K\}$.

juntamente com um particionamento de K em K1 e K2, tem-se

```
T1(B) = Σ{a_i; sync!m_i; T1(B_i) | i ∈ I}
        Σ{i; sync!m_k; T1(B_k) | k ∈ K1}
    []sync!poll; (Σ{sync!m_j; T1(B_j) | j ∈ J}
            []Σ{sync!m_k; T1(B_k) | k ∈ k2}
            []sync!poll; T1(B))

T2(B) = Σ{sync!m_i; T2(B_j) | i ∈ I}
    Σ{sync!m_k; T2(B_k) | k ∈ K1}
    []sync!poll;(Σ{B_j; sync!m_j; T2(B_j) | j ∈ J}
          [] Σ{i; sync!m_k; T2(B_k) | k ∈ k2}
          [] sync!poll; T2(B))
```

Além da comunicação síncrona, o autor propõe ainda algoritmos que consideram o caso de comunicação assíncrona. No caso da comunicação síncrona, **T1(B)** iniciava sempre no estado ativo, e **T2(B)** iniciava sempre no estado passivo. No caso da comunicação assíncrona, se, por exemplo, **T1(B)** realizar uma ação observável e enviar uma mensagem, ele deve entrar em estado passivo, pois, caso contrário, ele pode realizar alguma ação observável antes que **T2(B)** tenha recebido todas as mensagens de sincronização.

Esses algoritmos aceitam como entrada uma especificação do comportamento do processo **B** que inclui o operador de paralelismo independente (|||) além dos operadores já utilizados até aqui.

3.4. Avaliação das abordagens

Para fazer uma avaliação das abordagens dos autores apresentados neste capítulo, devem ser considerados alguns aspectos de fundamental importância na síntese de protocolos. Esses aspectos são apresentados a seguir.

Muitas pesquisas têm sido feitas no desenvolvimento de métodos formais para o projeto de protocolos de comunicação. Pode-se considerar que estes métodos seguem duas abordagens de projeto, denominadas abordagem analítica e abordagem sintética [Zafi80].

Na abordagem analítica, o projetista inicia com uma versão preliminar do protocolo, obtida de maneira particular, pela definição das mensagens e o efeito da troca de mensagens entre as entidades de protocolo. Essa abordagem possui várias deficiências. Uma delas, é que os projetos resultantes de sua utilização podem ficar errôneos e incompletos. Por essa razão, uma etapa posterior de análise e verificação é necessária, com intenção de detectar erros e omissões. A seqüência de redefinição do projeto, análise, detecção de erros e correções é aplicada até que o projeto esteja livre de erros. Outra deficiência dessa abordagem é que a reformulação ou modificação de um projeto existente é, na maioria das vezes, uma tarefa muito difícil e consome muito tempo. Essa abordagem pode ser encontrada em [Sajk84] e [Yuan88].

Na abordagem sintética, o projeto de um protocolo é construído de tal maneira que as interações entre suas entidades de protocolo prossigam sem manifestar qualquer erro, fornecendo os serviços especificados. O processo de síntese é realizado ou interativamente (passo-a-passo, com o projetista) ou automaticamente (sem qualquer interação com o projetista). Além disso, uma vantagem do método puramente sintético é que a correção do protocolo sintetizado pode ser um resultado direto do processo de síntese, provando ele mesmo que o método de síntese é bem fundado e corretamente utilizado. Portanto, nesse caso, não é necessária verificação futura do projeto do protocolo como no caso da abordagem analítica.

Independente da abordagem utilizada, dois tipos de propriedades devem ser garantidas em um projeto de protocolo:

- As propriedades de segurança (*safety properties*) que asseguram que o protocolo nunca entra em um estado indesejável. Estas propriedades incluem ausência de impasses, ausência de evoluções inúteis, e ausência de imperfeições (isto é, ausência de erros de recepção inesperada).
- As propriedades de vivacidade (*liveness properties*) que asseguram que o protocolo eventualmente entrará em um estado desejável, isto é, alguma coisa boa eventualmente

acontecerá, significando que o protocolo realiza as funções pretendidas com respeito à especificação de serviço.

As propriedades de segurança e vivacidade contribuem muito para que não ocorram erros sintáticos e semânticos. Esta é a abordagem utilizada para demonstrar equivalência observacional entre o serviço válido e protocolo correspondente. Os erros sintáticos e semânticos são descritos abaixo:

- Erros sintáticos e lógicos de projeto:

a) Impasse: um sistema entra em estado de impasse quando nenhuma mensagem de transmissão é possível e todos os canais de comunicação estão vazios, isto é, não é possível progresso futuro.

b) Imperfeição: quando uma entidade recebe uma mensagem que era inesperada.

c) Instabilidade: quando ocorre perda de sincronicação entre as entidades cooperantes.

d) Evolução inútil: quando um protocolo entra em um estado em que infinitos ciclos improdutivos são possíveis.

e) Superlotação: ocorre quando um canal recebe mais mensagens que sua capacidade predefinida.

- Erros semânticos de projeto: caracterizados pelo funcionamento anormal do protocolo e sua incapacidade de atingir o fim intencionado pelo serviço correspondente.

Classificação dos métodos de síntese

Os métodos de síntese de protocolo podem ser classificados de acordo com algumas características gerais. Essas características são [PrSa91]:

- *O ponto inicial do método*: o processo de síntese pode iniciar de:

  a) de anotações contendo somente especificações informais

  b) de uma especificação completa de uma entidade de protocolo

  c) de especificações parciais das entidades de protocolo envolvidas

d) da especificação de serviço completa

- *O formalismo utilizado para descrever o protocolo*: podem ser utilizados vários modelos formais, tais como máquinas de estados finitos, redes de Petri, LOTOS, etc.
- *As construções do modelo de comunicação*: uma das construções relatadas nos modelos de comunicação são as interações das entidades comunicantes e os usuários do serviço. Outra construção seria o número de entidades de protocolo que podem ser sintetizados pelo método. Outras construções consideram ainda um meio de comunicação considerado confiável.
- *O modo de interação*: indica se o protocolo é gerado automaticamente ou construído interativamente, permitindo ao projetista intervir e fazer escolhas de projeto sempre que forem necessárias.
- *As propriedades de protocolo garantidas pelo método*: seriam as propriedades de segurança e vivacidade descritas anteriormente.

Na abordagem do Langerak consideram-se algoritmos, em três estágios sucessivos, que resolvem problemas cada vez mais complexos. Nessa abordagem não se distingue o tratamento específico que é dado a cada operador LOTOS, uma vez que eles são considerados simultaneamente. Por exemplo, no mesmo algoritmo são considerados os casos de seqüenciamento, escolha distribuida e recursividade. Sua abordagem não trata os casos de habilitação, interrupção e paralelismo. Não é feita implementação de seus algoritmos.

Com relação à Bochmann, sua abordagem é baseada em gramática de compiladores. Todos os operadores LOTOS são considerados mas não é tratado o caso de escolha distribuída. São ainda consideradas *n* entidades de protocolo. Veja a comparação entre as duas abordagens apresentadas na tabela 3.4, considerando todos os aspectos acima mencionados.

Tabela 3.4 - Avaliação das abordagens apresentadas.

Aspectos	Gregor v. Bochmann	Rom Langerak
Abordagem	Sintética	Sintética
Ponto de partida	especificação de serviço completa	especificação de serviço completa
Formalismo utilizado	TDF LOTOS	TDF LOTOS
Meio de comunicação	Considerado confiável	Considerado confiável
Modo de interação	Não interativo	Não interativo
Implementação	Sim (Prolog)	Não
Entidades de protocolo	n	2
Permite escolha entre sítios distribuídos	Não	Sim
Operadores LOTOS tratados	Todos	Sequenciamento e escolha
Restrição de localidade	Sim	Sim
Abordagem baseada em	Gramática de compiladores	Algoritmos

Justifica-se uma investigação sobre a possibilidade de elaboração de algoritmos para a transformação de especificações LOTOS que possam constituir uma ferramenta de apoio ao projeto de protocolos de comunicação e sistemas distribuídos.

4. Algoritmos propostos

Nesta seção propõem-se algoritmos baseados em regras de transformação que permitem obter uma especificação refinada de um sistema (protocolo) a partir de uma especificação abstrata do mesmo sistema (serviço).

A linguagem utilizada nas especificações é a Técnica de Descrição Formal LOTOS, e tem como restrição que cada ação deve conter a indicação do sítio em que ocorre, sítio 1 ou sítio 2. Por exemplo, no caso de uma especificação de entrada (serviço) `Serv:= a;b;stop`, onde a ação `a` ocorre no sítio 1 e a ação `b` ocorre no sítio 2, esta especificação deve ser escrita na forma `Serv:= a1;b2;stop`.

A base para este trabalho é apresentada em [Riso91], que utiliza a linguagem CCS (Calculus of Communicating Systems) e apresenta algoritmos para seqüenciamento e escolha entre ações em um mesmo sítio. A principal diferença é que, neste trabalho, utiliza-se a TDF LOTOS básica. Além disso foram desenvolvidos, além dos algoritmos para seqüenciamento e escolha entre ações em um mesmo sítio, algoritmos para tratar o caso de escolha distribuída, habilitação e interrupção, e estes algoritmos são implementados e constituem uma ferramenta de uso prático. Em [ToNo94] pode ser encontrado um artigo que considera a transformação de especificações de serviço em especificações de protocolo considerando apenas os casos de seqüenciamento de ações e escolha simples.

As regras de transformação aplicam-se a um modelo de sistema que, em uma visão mais abstrata, é representado por um processo (correspondente ao serviço). Na visão refinada, o modelo compreende dois processos: a entidade de protocolo **E1** (no sítio 1); e a entidade de protocolo **E2** (no sítio 2). Veja a Figura 4.1.

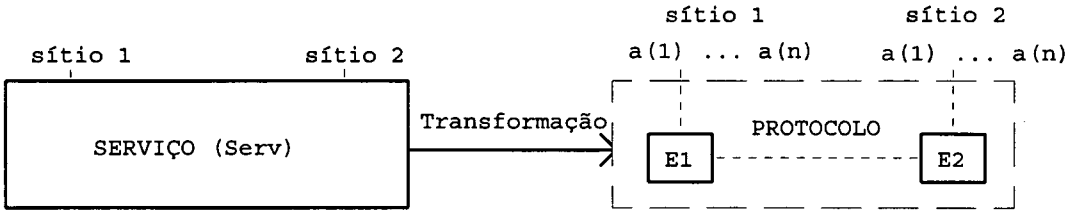


Figura 4.1 - Refinamento do serviço para obtenção do protocolo.

A especificação do comportamento de `E1` e `E2` deve ser obtida com a aplicação de regras de transformação. Considera-se que o primeiro evento realizado pelo sistema ocorre no sítio 1. Como esse evento envolve a entidade de protocolo `E1`, esta é designada entidade primária ou iniciadora. A entidade de protocolo `E2` é designada entidade secundária ou respondedora.

As entidades de protocolo comunicam-se diretamente (isto é, sem um meio de comunicação intermediário). Conjuntamente, elas apresentam o mesmo comportamento observável do provedor de serviço. A comunicação direta entre as entidades não considera a utilização de um serviço subjacente.

Para a definição das regras de transformação analisa-se um conjunto de aspectos isolados que podem estar presentes na especificação de serviço. Cada aspecto permite a criação de uma regra que mapeia a estrutura da especificação de serviço na estrutura do protocolo correspondente. As regras são validadas e apresentadas de forma algorítmica.

4.1. Algoritmo 1 - Definição do conjunto de ações de cada entidade

Neste algoritmo, a cada ação observável (`a;B`) ou não observável (`i`) da especificação de serviço corresponde uma ação igual definida na especificação de uma das entidades de protocolo. Se a ação definida na especificação de serviço ocorre no sítio 1, então essa mesma ação deve participar do comportamento da entidade de protocolo do sítio 1 (isto é, da entidade `E1`). Caso

contrário, essa ação deve participar do comportamento da entidade de protocolo do sítio 2 (isto é, da entidade E2).

Por exemplo (veja a Figura 4.2), no caso em que `Serv := a1; a2; stop`, onde a ação `a1` ocorre no sítio 1 e a ação `a2` ocorre no sítio 2, `L(E1)` é o conjunto de eventos do processo `E1` (*Labels* de `E1`) e `L(E2)` é o conjunto de eventos do processo `E2` (*Labels* de `E2`), os comportamentos das entidades de protocolo `E1` e `E2` são tais que `a1 ∈ L(E1) e a2 ∈ L(E2)`.

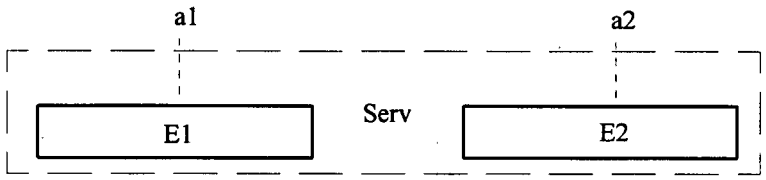


Figura 4.2 - Ocorrência de a1 no sítio 1 e ocorrência de a2 no sítio 2.

```
Dado Serv := a(1); ...; a(n); Serv'
L(E1) = {}; L(E2) = {};  j = 1
enquanto j ≤ n  faça
  se a(j) ocorre no sítio 1 então
    se a(j) é um evento observável então
      L(E1) = L(E1) ∪ {a(j)}
    senão inclua i no comportamento de E1
    fim se
  senão
    se a(j) é um evento observável então
         L(E2) = L(E2) ∪ {a(j)}
    senão inclua i no comportamento de E2
    fim se
  fim se
     j = j + 1
fim enquanto
```

4.2. Algoritmo 2 - Seqüenciamento finito de ações no mesmo sítio

Este algoritmo define o comportamento de cada entidade de protocolo no caso do serviço envolver apenas seqüenciamento de eventos em um único sítio do sistema.

Por exemplo (veja a Figura 4.3), no caso em que `Serv := a1; b1; c1; stop`, onde as ações `a1`, `b1` e `c1` ocorrem no sítio 1, os comportamentos das entidades de protocolo `E1` e `E2` podem ser definidos por

```
E1 := a1; b1; c1; stop
E2 := stop
```

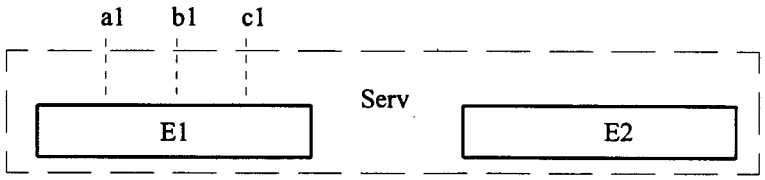


Figura 4.3 - Ocorrência de a1, b1 e c1 no sítio 1.

Os vetores A e B utilizados no seguinte algoritmo, armazenam as ações que ocorrem no sítio 1 e as ações que ocorrem no sítio 2 respectivamente.

```
Dado Serv := a(1); ...; a(n); Serv'
E1:= A(1); ...; A(n); E1'
E2:= B(1); ...; B(n); E2'

Se as ações a(1); ...; a(n) ocorrem no sítio 1 então
  j = 1
  enquanto j ≤ n  faça
    A(j) = a(j)
    B(j) = ε (seqüência vazia de eventos)
    j = j + 1
  fim enquanto
senão (Se as ações a(1); ...; a(n) ocorrem no sítio 2)
  j = 1
  enquanto j ≤ n faça
    A(j) = ε (seqüência vazia de eventos)
```

```
    B(j) = a(j)
    j = j + 1
  fim enquanto
fim se
```

Para provar a validade da regra de transformação expressa através do algoritmo 2 pode-se considerar a Figura 4.4. Nessa figura, a árvore `E1 ||| E2` tem os mesmos eventos que a árvore `Serv` e com a mesma ordenação.

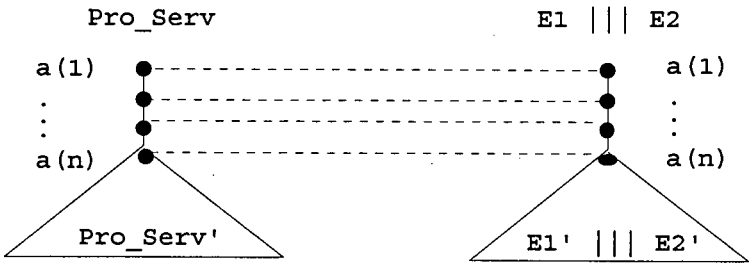


Figura 4.4 - Bissimulação entre E1 ||| E2 e Serv.

Considerando que `Serv' ≈ E1' ||| E2'`, pode-se traçar as linhas que representam a bissimulação entre `Serv e E1 ||| E2`. Tal bissimulação prova a equivalência requerida.

4.3. Algoritmo 3 - Ações em sítios diferentes

No caso em que `Serv := a1; b1; a2; b2; stop`, por exemplo, onde as ações `a1` e `b1` ocorrem no sítio 1 e as ações `a2` e `b2` ocorrem no sítio 2, os comportamentos das entidades de protocolo `E1` e `E2` podem ser definidos por

```
E1: = a1; b1; s1; stop
E2: = s1; a2; b2; stop
```

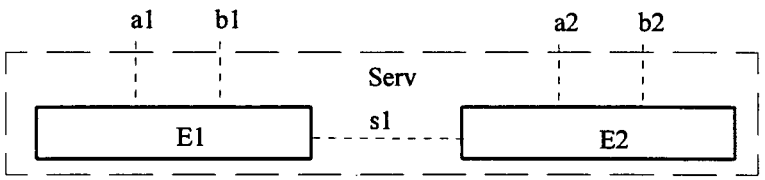


Figura 4.5 - Ocorrência de a1 e b1 no sítio 1 e a2 e b2 no sítio 2.

A sincronização entre as duas entidades de protocolo é realizada através de portas de comunicação `s1` a `sn` de acordo com a necessidade de sincronização. Nesse exemplo, a sincronização das entidades de protocolo `E1` e `E2` se dá através da porta `s1`.

Nos casos simples em que o provedor de serviço executa uma seqüência de ações em um dos sítios e, posteriormente, executa uma seqüência de ações no outro sítio, tonando-se depois inativo, as entidades de protocolo realizam apenas uma sincronização.

```
Dado Serv := a(1); ...; a(n); Serv'
as entidades de protocolo E1 e E2 têm comportamentos definidos por:
Se a(1) ocorre no sítio 1 então
  E1 := A(1);...; A(n); sn; E1'
  E2 := sn; B(1); ...; B(n); E2'
senão
  E1 := sn; A(1); ...; A(n); E1'
  E2 := B(1); ...; B(n); sn; E2'
fim se
j = 1
enquanto j ≤ n faça
  se a(j) ocorre no sítio 1 então
    A(j) = a(j)    B(j) = ε (seqüência vazia de eventos)
  senão
    A(j) = ε (seqüência vazia de eventos)    B(i) = a(j)
  fim se
  j = j + 1
fim enquanto
```

4.4. Algoritmo 4 - Comportamento alternado entre os dois sítios

A situação é mais complexa quando o provedor de serviço alterna seqüências de ações do sítio 1 com seqüências de ações do sítio 2. Por exemplo (veja a Figura 4.6), no caso em que `Serv:= a1; a2; a1; b1; a2; b2; stop` onde esta seqüência de ações é tal que `a1` ocorre no sítio 1, `a2` ocorre no sítio 2, `a1; b1` ocorrem no sítio 1 e `a2; b2` ocorrem no sítio 2.

As entidades de protocolo E1 e E2 podem ser definidas por

```
E1 := a1; s1; s1; a1; b1; s1; stop
E2 := s1; a2; s1; s1; a2; b2; stop
```

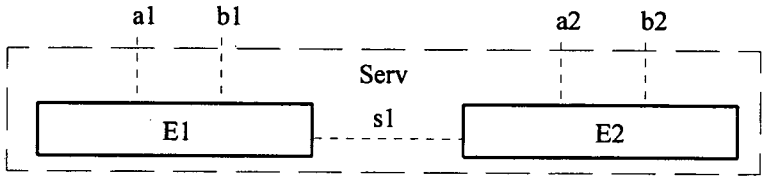


Figura 4.6 - Sincronizações de E1 com E2 para o caso de alternância de eventos nos sítios 1 e 2.

Nesse exemplo a entidade `E1` transfere o comportamento ativo para a entidade `E2` através de um evento de sincronização na porta `s1`. A entidade E2 transfere o comportamento ativo para a entidade E1 através de um novo evento de sincronização na porta `s1`.

```
Dado Serv := a(1); ...; a(n); Serv'
```

as entidades de protocolo E1 e E2 têm comportamentos definidos por:

```
E1 := A(1); ...; A(n); E1'
E2 := B(1); ...; B(n); E2'
j = 1
enquanto j < n  faça
  se a(j) e a(j+1) ocorrem no mesmo sítio então
    se a(j) e a(j+1) ocorrem no sítio 1 então
      A(j) = a(j)        B(j) = ε
    senão
      A(j) = ε           B(j) = a(j)
    fim se
  senão
    se a(j) ocorre no sítio 1 e a(j+1) ocorre no sítio 2 então
      A(j) = a(j); sn       B(j) = sn
    senão
      A(j) = sn          B(j) = a(j); sn
    fim se
  fim se
  j = j + 1
fim enquanto
```

```
se a(j) ocorre no sítio 1 então
  A(j) = a(j)          B(j) = ε
senão
  A(j) = ε           B(j) = a(j)
fim se
```

No caso em que a especificação de serviço é definida por uma seqüência finita de eventos, o algoritmo 4 pode ser utilizado para a derivação automática da especificação do protocolo correspondente. O algoritmo 4 copia os eventos da seqüência, colocando-os com o mesmo ordenamento no comportamento de `E1` ou no comportamento de `E2`, conforme o sítio em que ocorrem.

Além disso, o algoritmo 4 introduz eventos novos. Estes eventos correspondem às sincronizações das entidades de protocolo `E1` e `E2` e indicam a transferência do comportamento ativo, que vinha sendo exercido por uma entidade, para a outra entidade. Então esta outra entidade assume a continuação das ações.

4.5. Algoritmo 5 - Comportamento recursivo

No caso do comportamento recursivo, deve-se considerar que a especificação de entrada é uma expressão guardada[1]. Se o agente provedor de serviço (identificado por `Serv`) apresenta um comportamento infinito, e esse comportamento inclui eventos nos dois sítios do sistema, então os agentes derivados (as entidades de protocolo identificadas por `E1` e `E2`) também devem apresentar comportamentos infinitos.

[1] "P where X_1 := P_1, ... X_n :=P_n" é uma especificação LOTOS guardada se, pela substituição recursiva, em um número finito de vezes, das expressões P_i's para os identificadores de processos X_i's ocorrendo em P e nos próprios P_i's, é possível obter uma specificação expandida LOTOS "Q where X_1 := Q_1, ... X_n :=Q_n" onde Q e Q_i's são expressões guardadas, isto é, se todas instanciações de Q_i's em X_j's são precedidas ao menos por uma ação (observável ou não).

Assim, o identificador `E1` deve ser acrescido ao final da expressão de comportamento da entidade `E1` e o identificador `E2` deve ser acrescentado ao final da expressão de comportamento da entidade `E2`. Entretanto, essas medidas não são suficientes para que a derivação esteja correta.

Por exemplo, pode-se considerar `Serv := a1; a2; Serv`, onde a ação `a1` ocorre no sítio 1 e a ação `a2` ocorre no sítio 2. Aparentemente, os comportamentos das entidades de protocolo `E1` e `E2` seriam `E1 := a1; s1; E1` e `E2 := s1; a2; E2`

Se fosse assim, a entidade `E1` poderia realizar a ação a1 logo após a sua sincronização com a entidade `E2`. Desse modo, o refinamento estaria errado.

Para garantir que a seqüência de eventos observáveis apresentada por `Serv` seja também apresentada pela composição de `E1` e `E2`, introduz-se um evento de sincronização na porta `c` (veja a Figura 4.7).

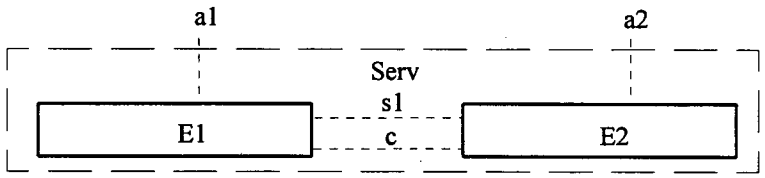


Figura 4.7 - Sincronizações de E1 com E2 para o caso de comportamento infinito.

Considerando que `Serv` ≈ **`hide`** *`s1,...,sn,c`* **`in`** `(E1 |[`*`s1,...,sn,c`*`]| E2)`, podem-se traçar as linhas que representam a bissimulação entre `Serv` e **`hide`** *`s1,...,sn,c`* **`in`** `(E1 |[`*`s1,...,sn,c`*`]| E2)`, provando, dessa maneira, a equivalência requerida.

Desse modo,

```
E1 := a1; m12; c; E1
E2 := m12; a2; c; E2
```

Dado `Serv := a(1); ...; a(n); Serv`

as entidades de protocolo `E1` e `E2` têm comportamentos definidos por:

```
E1 := A(1); ...; A(n); c; E1
E2 := B(1); ...; B(n); c; E2
```

o comportamento a seguir é idêntico ao do algoritmo anterior (algoritmo 4).

4.6. Algoritmo 6 - Escolha indeterminística entre ações do mesmo sítio

Neste algoritmo, supõe-se que as escolhas que ocorrem na especificação de serviço se dão entre ações de um mesmo sítio.

Por exemplo, pode-se considerar `Serv := a1; a2; stop [] b1; b2; stop` onde as ações `a1` e `b1` ocorrem no sítio 1 e as ações `a2` e `b2` ocorrem no sítio 2. Nesse exemplo, a escolha se dá entre eventos do sítio 1 (isto é, entre o evento `a1` e o evento `b1`). Dessa forma, o comportamento das entidades de protocolo `E1` e `E2` pode ser dado por

```
E1 := a1; s1; stop [] b1; s2; stop
E2 := s1; a2; stop [] s2; b2; stop
```

Para fins de derivação das entidades de protocolo, quando uma escolha se apresenta em um sítio do sistema, ela deve ser reproduzida pela entidade localizada nesse sítio e ignorada pela outra entidade. No exemplo, como a escolha apresenta-se entre eventos do sítio 1, ela reproduz-se no comportamento da entidade `E1`.

A questão da sincronização das entidades de protocolo, posteriormente à realização de uma escolha com a participação de uma delas, diz respeito à transferência de comportamento ativo de uma para outra entidade. Só a discriminação entre as sincronizações possíveis tem a ver com a escolha realizada.

No exemplo, se a escolha entre `a1` e `b1` recai sobre `a1` a sincronização de `E1` com `E2` se dá em `s1` (nesse evento, o índice `1` lembra a primeira escolha). Se a escolha recai sobre `b1` a sincronização de `E1` com `E2` se dá em `s2` (agora, o índice `2` lembra a segunda escolha). A utilização de um mesmo evento de sincronização para todas as escolhas resultaria numa derivação errada das entidades de protocolo. Por exemplo, se o comportamento das entidades de protocolo `E1` e `E2` fosse dado por

```
E1 := a1; s1; stop [] b1; s1; stop
E2 := s1; a2; stop [] s1; b2; stop
```

e fosse escolhido o evento `a1`, a sincronização `s1` permitiria uma nova escolha entre `s1; a2; stop` e `s1; b2; stop`, o que não estaria correto, uma vez que o comportamento posterior à escolha deveria ser somente `s1; a2; stop`.

Em geral, pode-se considerar que uma escolha presente no comportamento do provedor de serviço resulta em uma escolha no comportamento de cada entidade de protocolo.

Com relação à maneira de resolução e às otimizações, consideramos a especificação de serviço:

```
Serv := a1; (a2; a1; stop [] b2; a1; stop) [] b1; a1; stop
```

A resolução é realizada a partir da parte mais interna da especificação, feita em partes. Cada parte é representada por X*y*, sendo que 1<=*y*. Neste caso X1 seria `(a2; a1; stop [] b2; a1; stop)` e numa segunda etapa seria resolvido `Serv := a1; (X1) [] b1; a1; stop`.

Após a resolução, o comportamento das entidades de protocolo é apresentado da seguinte forma:

```
E1 := a1; s1; (s1; a1; stop [] s2; a1; stop ) [] b1; a1; stop
E2 := s1; (a2; s1; stop [] b2; s2; stop ) [] ε; stop
```

Com a primeira etapa de otimização fica eliminado (`[] ε; stop`) e a especificações de protocolo são apresentadas da seguinte forma:

```
E1 := a1; s1; (s1; a1; stop [] s2; a1; stop ) [] b1; a1; stop
E2 := s1; (a2; s1; stop [] b2; s1; stop )
```

Após a segunda etapa de otimização as especificações de protocolo são apresentadas:

```
E1 := a1; s1; s1; a1; stop [] b1; a1; stop
E2 := s1; (a2; s1; stop [] b2; s1; stop )
```

O algoritmo é então apresentado da seguinte maneira:

$$\text{Dado Serv} := \sum_{j=1}^{n} a(j);\ Serv(j)$$

As entidades de protocolo E1 e E2 têm comportamentos definidos por:

se a escolha se dá entre eventos do sítio 1 então

$$E1 := \sum_{j=1}^{n} a(j);\ E1(j) \qquad E2 := \sum_{j=1}^{n} \varepsilon;\ E2(j)$$

caso contrário

$$E1 := \sum_{j=1}^{n} \varepsilon;\ E1(j) \qquad E2 := \sum_{j=1}^{n} a(j);\ E2(j)$$

fim se

Os comportamentos dos agentes Serv, E1 e E2 posteriores à escolha não são considerados nesse algoritmo. Isso se dá porque esses comportamentos podem envolver vários outros aspectos como o de seqüenciamento de ações (finito ou infinito), o de mudanças de sítio e mesmo o de novas escolhas indeterminísticas entre ações. Tais aspectos exigem a aplicação de algoritmos anteriores e, no último caso, a reaplicação do algoritmo 6.

4.7. Algoritmo 7 - Escolha indeterminística entre ações de sítios diferentes

Neste algoritmo supõe-se que as escolhas que ocorrem na especificação de serviço se dão entre ações de sítios diferentes. Considere, por exemplo, uma simples escolha entre a1 e a2 (a1 [] a2). A dificuldade neste caso é que tanto a1 quanto a2 devem ser ambos oferecidos ao ambiente, e imediatamente após a escolha de a1, por exemplo, o ambiente não pode mais participar de um evento em a2. Não é possível uma sincronização para desabilitar a ocorrência de a2 porque a2 poderia ocorrer antes da sincronização.

Esse fato exige a adoção de um mecanismo de escolha seletiva para que o usuário possa fazer uma escolha entre sítios diferentes. Veja a Figura 4.8.

Figura 4.8 - Árvores de comportamento para poll.

O algoritmo 7 pode então ser apresentado da seguinte maneira:

$$\text{Dado } \texttt{Serv:=} \sum_{j=1}^{n} \texttt{a(j); Serv(j)} \quad \texttt{[]} \sum_{j=1}^{n} \texttt{b(j); Serv(j)}$$

Então:

$$\texttt{E1:=} \sum_{j=1}^{n} \texttt{a(j);E1(j)} \quad \texttt{[] poll;} \quad \left(\sum_{j=1}^{n} \varepsilon\texttt{;E2(j)} \quad \texttt{[] poll;E2)}\right.$$

$$\texttt{E2:=} \sum_{j=1}^{n} \varepsilon\texttt{;E2(j)} \quad \texttt{[] poll;} \quad \left(\sum_{j=1}^{n} \texttt{b(j);E2(j)} \quad \texttt{[] poll;E2)}\right.$$

No caso em que `Serv:= a1;Serv [] a2;a1;Serv [] b2;Serv` as entidades de protocolo E1 e E2 podem ser definidas por:

```
E1:= a1;c;E1 [] poll;( s1;a1;c;E1 [] c;E1 [] poll; E1)
E2:= c;E2 [] poll;( a2;s1;c;E2 [] b2;c;E2 [] poll; E2)
```

Como o comportamento sempre inicia no sítio 1, o usuário pode optar entre escolher `a1` ou executar `poll`, permitindo que a escolha seja feita entre eventos do segundo sítio (entre `a2`, `b2`) e executando `poll` novamente. Veja a Figura 4.9.

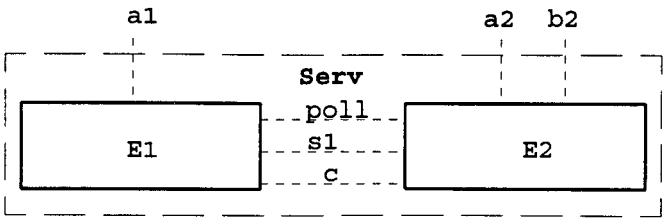


Figura 4.9 - Sincronizações de E1 com E2 para o caso de escolha entre sítios diferentes.

As otimizações aplicadas neste algoritmo são as mesmas aplicadas no algoritmo de escolha indeterminística (algoritmo 6).

A transformação realizada pelos algoritmos apresentados anteriormente (do algoritmo 2 até o algorítmo 7) implementa a transformação proposta por Langerak.

Langerak utiliza somente uma mensagem, com um campo de dados associado, o qual faz a diferenciação das mensagens. Como os algoritmos aqui apresentados utilizam LOTOS básica, a diferenciação das mensagens se dá através da criação de uma nova mensagem onde Langerak utiliza um dado diferenciado.

Desse modo, para provar a equivalência observacional entre a especificação de serviço e a especificação de protocolo gerada, considera-se o lema a seguir, proposto por Langerak.

Lema [Lang90]: Sejam `X` e `Y` dois processos, e seja `P` definido por `P := X [] i;(Y [] i;P)`. Então `P` ≈ `X[]Y`.

Como o algoritmo apresentado implementa o proposto por Langerak, podemos utilizar o teorema proposto por ele:

Teorema [Lang90]: Sejam `E1` e `E2` definidos como os processos resultantes da transformação do algoritmo. Seja `B` um processo na forma de prefixo de ação. Então:

`(hide s1,..,sn,poll,c in E1 [s1,...,sn, poll, c] E2)` ≈ `B`.

Prova:

Como B é uma especificação de entrada do algoritmo, temos que B é definido por:

B := Σ{a(i); B(i)}

[] Σ{b(j); B(j)}

onde os a(i)'s são as ações do sítio 1 e os b(j)'s são as ações do sítio 2, e B(i) e B(j) são processos da mesma forma que B, gerados recursivamente. Podemos então escrever B := X(i) [] Y(j), onde X(i) := Σ{a(i); B(i)}, e Y(j) := Σ{b(j); B(j)}.

Por expansão, temos que **hide** s1,..,sn,poll,c **in** E1[s1,..,sn,poll,c]E2 = P, com:

P := Σ{a(i);**i**;(**hide** s1,..,sn,poll,c **in** E1(i)[s1,..,sn,poll,c]E2(i)}

[]**i**;(Σ{b(j);**i**;(**hide** s1,..,sn,poll,c **in** E1(j)[s1,..,sn,poll,c]E2(j)}

[]**i**;P)

Ora, se tomarmos:

X'(i):=Σ{a(i);**i**;(**hide** s1,..,sn,poll,c **in** E1(i)[s1,..,sn,poll,c]E2(i)}e

Y'(j):=Σ{b(j);**i**;(**hide** s1,..,sn,poll,c **in** E1(j)[s1,..,sn,poll,c]E2(j)}, podemos escrever que P := X'(i) [] **i**;(Y'(j) [] **i**;P). Pelo lema acima P ≈ P', onde P' = X'(i) [] Y'(j).

Pode-se ver diretamente que X'(i):=Σ{a(i);**i**;B(i)} e Y'(j):=Σ{b(j);**i**;B(j)}, pela transformação do algoritmo aplicada sobre B(i) e B(j). Novamente por inspeção, temos que X(i) = X'(i) e Y(i) = Y'(i), onde o sinal de = indica congruência observacional. Daí temos que P ≈ P'≈ B. □

4.8. Algoritmo 8 - Habilitação

Se o agente provedor de serviço (identificado por `Serv`) apresentar uma expressão "X >> Y", por exemplo, o operador de habilitação indica que se o processo definido por X terminar com sucesso, a execução do processo definido por Y estará habilitada.

Por exemplo no caso da especificação de serviço ser definida por :

```
Serv := (a1;Serv >> a2;stop) [] b1;b2;exit
```

O comportamento das entidades de protocolo pode ser definido por:

```
E1:= (a1; c; E1 >> stop) [] b1; s2; exit
E2:= (c; E2 >> a2; stop) [] s2; b2; exit
```

O tratamento de seqüenciamento, escolha e recursividade se dá nos algoritmos anteriores (algoritmo 1 a algoritmo 7), deixando o algoritmo 8 tratar somente da habilitação. Neste exemplo, é feita uma escolha inicial entre `a1;serv` e `b1;b2;exit`. Se a escolha recair sobre `a1;serv`, então haverá uma chamada recursiva e o sistema volta ao estado inicial. Quando a escolha recair sobre a segunda opção (`b1;b2;exit`), este `exit` habilitará a execução de `a2;exit`.

O algoritmo é então apresentado da seguinte forma:

No caso em que

```
Serv := A >> B
onde
A:= a(1);...; a(n); exit
B:= b(1);...; b(n); stop
```

as entidades de protocolo `E1` e `E2` têm comportamentos definidos por:

```
Se os eventos de A e B ocorrem em 1 sítio apenas
   Se os eventos de A e B ocorrem no sítio 1
      E1:= A1 >> B1
      E2:= ε; E2'
      onde
```

```
      A1:= a(1); ...; a(n); exit
      B1:= b(1); ...; b(n); stop
   senão
      E1:= ε; E1'
      E2:= A2 >> B2
      onde
      A2:= a(1); ...; a(n); exit
      B2:= b(1); ...; b(n); stop
   fim se
senão
  E1:= A1 >> B1
  E2:= A2 >> B2
onde
  A1:= a(1); ...; a(n); exit
  A2:= a(1); ...; a(n); exit
  B1:= b(1); ...; b(n); stop
  B2:= b(1); ...; b(n); stop
fim se
```

4.9. Algoritmo 9 - Interrupção

Neste caso o agente provedor de serviço (identificado por `Serv`) pode apresentar uma expressão como por exemplo "b1 [> *an*; b2", onde b1 e b2 são expressões e *an* é um evento. Isto significa que a ocorrência do evento *an* deve interromper a execução de "B1".

Por exemplo no caso da especificação de serviço ser definida por :

```
Serv := ((a1;serv >> a2;stop) [] b1;b2;exit) [> c1;b1;c2;stop.
```

Essa especificação de serviço considera que a especificação de serviço do algoritmo 8 pode ser interrompida, a qualquer momento, pela execução de `c1;b1;c2;stop`. Neste caso, o comportamento das entidades de protocolo `E1` e `E2` pode ser definido por:

```
E1:= ((a1; c; E1 >> stop) [] b1; s2; exit) [> c1; b1; s1; stop
E2:= ((c; E2 >> a2; stop) [] s2; b2; exit) [> s1; c2; stop
```

Como acontece nos casos de escolha indeterminística, na interrupção deve-se ter muito cuidado com as sincronizações. A sincronização utilizada na interrupção deve ser distinta das utilizadas pelas escolhas indeterminísticas, independentes do nível. Por exemplo, se as especificações de protocolo fossem:

```
E1:= ((a1; c; E1 >> stop) [] b1; s1; exit) [> c1; b1; s1; stop
E2:= ((c; E2 >> a2; stop) [] s1; b2; exit) [> s1; c2; stop
```

poderia haver a sincronização errada de `c1; b1; s1; exit` com `s1; b2; exit`, o que tornaria errôneas as especificações de protocolo.

O algoritmo é então apresentado da seguinte forma:

No caso em que

```
Serv := A [> B
onde
A:= a(1);...; a(n); A'
B:= b(1);...; b(n); B'
```

as entidades de protocolo `E1` e `E2` têm comportamentos definidos por:

```
Se os eventos de A e B ocorrem em 1 sítio apenas
   Se os eventos de A e B ocorrem no sítio 1 (j=1)
     E1:= A1 [> B1
     E2:= ε; E2'
     onde
     A1:= a(1); ...; a(n); A1'
     B1:= b(1); ...; b(n); B1'
   senão
     E1:= ε; E1'
     E2:= A2 [> B2
     onde
     A2:= a(1); ...; a(n); A2'
     B2:= b(1); ...; b(n); B2'
   fim se
senão
  E1:= A1 [> B1
```

```
  E2:= A2 [> B2
  onde
  A1:= a(1); ...; a(n); A1'
  A2:= a(1); ...; a(n); A2'
  B1:= b(1); ...; b(n); B1'
  B2:= b(1); ...; b(n); B2'
fim se
```

Embora acredita-se que a transformação realizada pelos algoritmos de habilitação (algoritmo 8) e interrupção (algoritmo 9) esteja correta (preserva a eqüivalência de observação entre a especificação de serviço e a especificação de protocolo derivada), a prova não é apresentada neste trabalho.

5. A ferramenta TranSP

Com o objetivo de implementar os algoritmos desenvolvidos, foi construída uma ferramenta de uso prático. Publicações relativas a esta ferramenta podem ser encontradas em [ToNo95a] [ToNo95b], [ToNo95d] e [NoTo95a]. Esta ferramenta foi programada em Borland C++, para o ambiente Windows e suas principais características podem ser visualizadas na Figura 5.1. A ferramenta TranSP trabalha somente com LOTOS básico e estilo monolítico, mas permite que futuras extensões dêem mais flexibilidade à ferramenta.

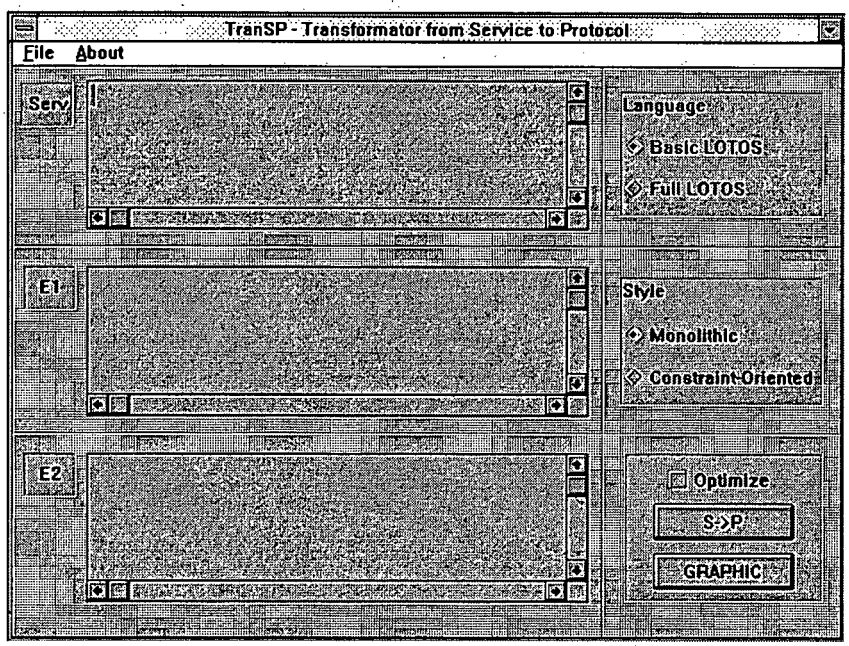


Figura 5.1 - Tela principal da ferramenta TranSP.

A ferramenta TranSP recebe uma especificação de serviço elaborada com o uso da TDF LOTOS básico (no estilo monolítico), na janela *Serv* (veja a Figura 5.1). Após a entrada de uma especificação de serviço, clica-se com o mouse na tecla "**S->P**" (Serviço -> Protocolo) para gerar o comportamento correspondente às entidades de protocolo E1 e E2 (nas janelas correspondentes), relativas ao sítio 1 e sítio 2, respectivamente. A especificação dessas entidades de protocolo é correta por construção.

Na opção **File**, pode-se recuperar uma especificação armazenada anteriormente ou armazenar uma nova especificação de serviço (**Open**), gravar uma especificação digitada (**Save**), ou copiar a especificação apresentada na janela *Serv* para outro nome qualquer (**Save As**). A ferramenta TranSP possui ainda a opção (**Save Protocol**), que permite que a especificação completa de protocolo que foi derivada seja armazenada para que outras ferramentas, como a LOLA e CÆSAR por exemplo, sejam utilizadas para simular a especificação de protocolo obtida . Veja na Figura 5.2 como são as janelas para abrir e gravar arquivos (abrir, neste caso).

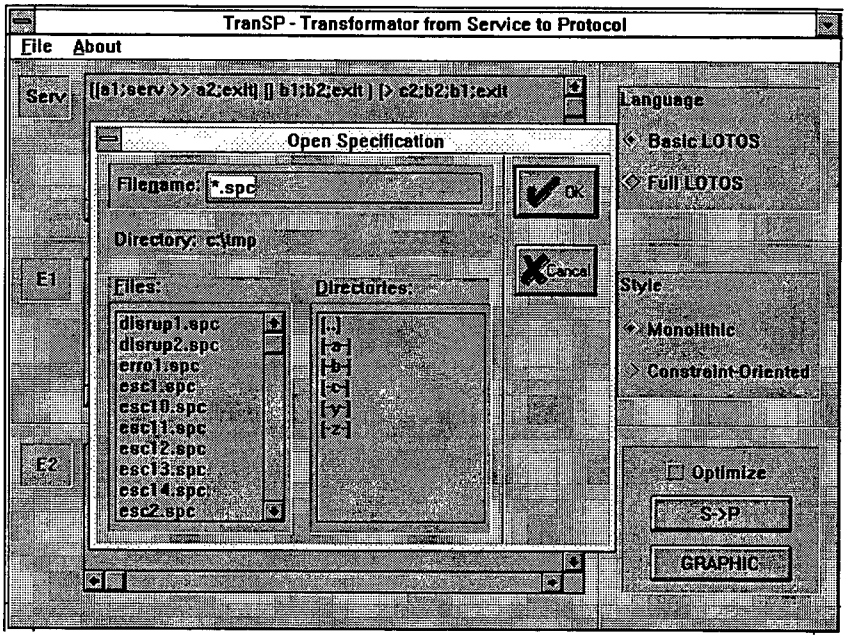


Figura 5.2 - Abertura de arquivos.

O exemplo a seguir considera o caso das ações ocorrerem em apenas um sítio. É dada a entrada de uma especificação de serviço com várias ações no sítio 1 (`a1; b1; c1; d1;`) seguidos de `stop`. Como esta ação ocorre no sítio 1, ela deve participar somente do comportamento da entidade de protocolo do sítio 1 (`E1`). Através da opção **GRAPHIC**, pode-se gerar o gráfico para visualizar o fluxo de mensagens e as ações que ocorrem em cada sítio. Veja a Figura 5.3.

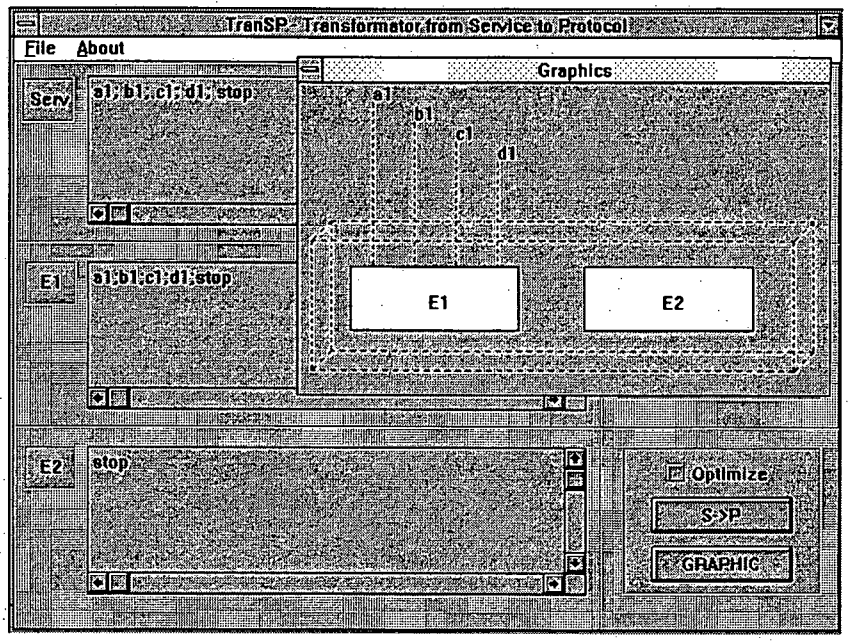


Figura 5.3 - Ocorrência de a1; b1; c1; d1 no sítio 1.

O exemplo seguinte considera a passagem de comportamento de um sítio para outro. A sincronização entre o sítio 1 e o sítio 2 é feita através de eventos de sincronização na porta `s1`. Veja a Figura 5.4.

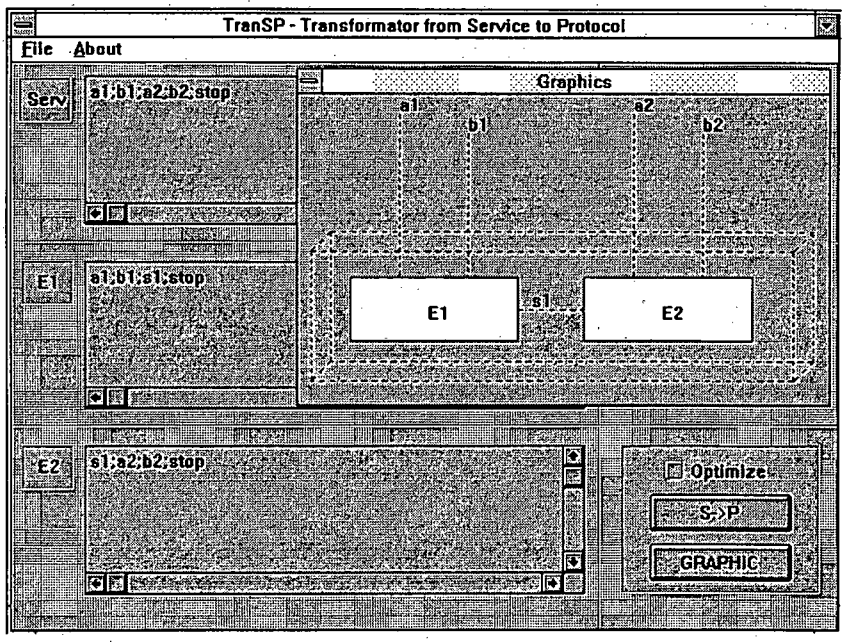


Figura 5.4 - Ocorrência de a1 e b1 no sítio 1 e a2 e b2 no sítio 2.

Veja na Figura 5.5 um exemplo em que `Serv:= a1; b1; a2; b2; a1; a2; b1; b2; stop`, alternando as ações entre o sítio 1 (`E1`) e o sítio 2 (`E2`).

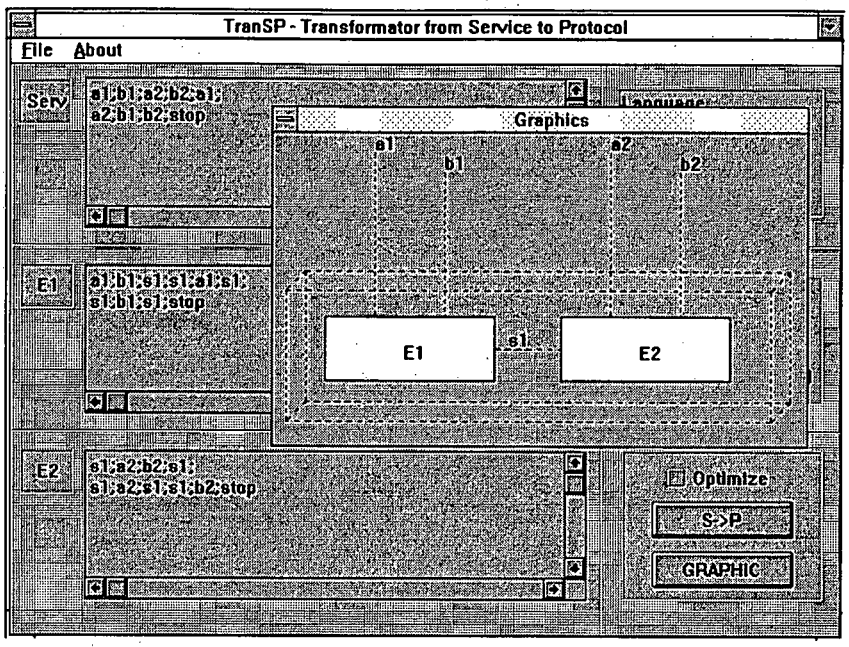


Figura 5.5 - Alternância de ações entre o sítio 1 e o sítio 2.

No caso de comportamento infinito do agente provedor de serviço, as entidades de protocolo também apresentam comportamento infinito. Este caso considera ações em um sítio seguidas de ações no outro sítio. Para garantir a retomada correta das ações, é introduzido um evento de sincronização na porta c. Veja a Figura 5.6.

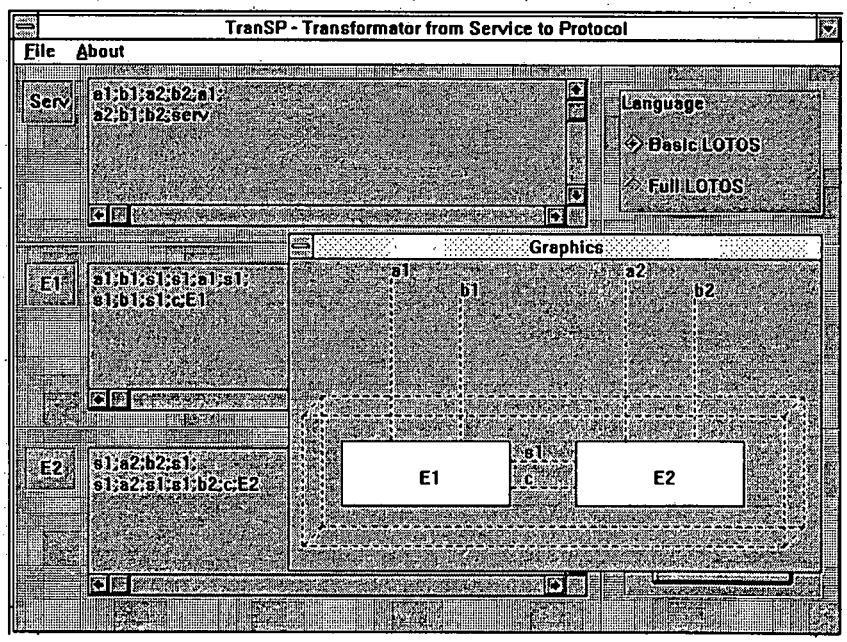


Figura 5.6 - Comportamento recursivo infinito.

Considera-se agora o caso de escolhas indeterminísticas entre ações em um mesmo sítio. Veja na Figura 5.7 o exemplo em que `Serv:= a2;(b1;a2;serv [] c1;a2;serv) [] d1;serv [] a1;(b2;serv [] c2;serv))`. A derivação das entidades de protocolo envolve, além da escolha indeterminística, seqüenciamento, recursividade e sub-expressões.

No momento em que é gerado o comportamento das entidades de protocolo `E1` e `E2`, pode-se optar por otimizar, ou não, a especificação que será gerada, através da opção **Optimize**. Compare, por exemplo, as especificações da Figura 5.7 com as especificações da Figura 5.8. Em ambas aparece a mesma especificação de serviço, mas na Figura 5.8, o comportamento da entidade de protocolo `E1` e o comportamento da entidade de protocolo `E2` estão otimizados.

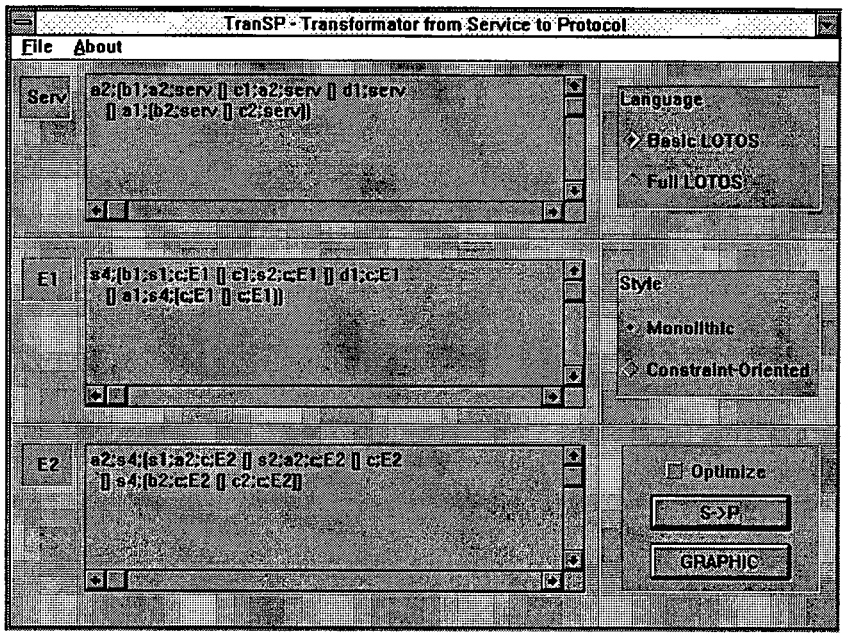


Figura 5.7 - Escolha indeterminística entre ações do mesmo sítio (algoritmo não otimizado).

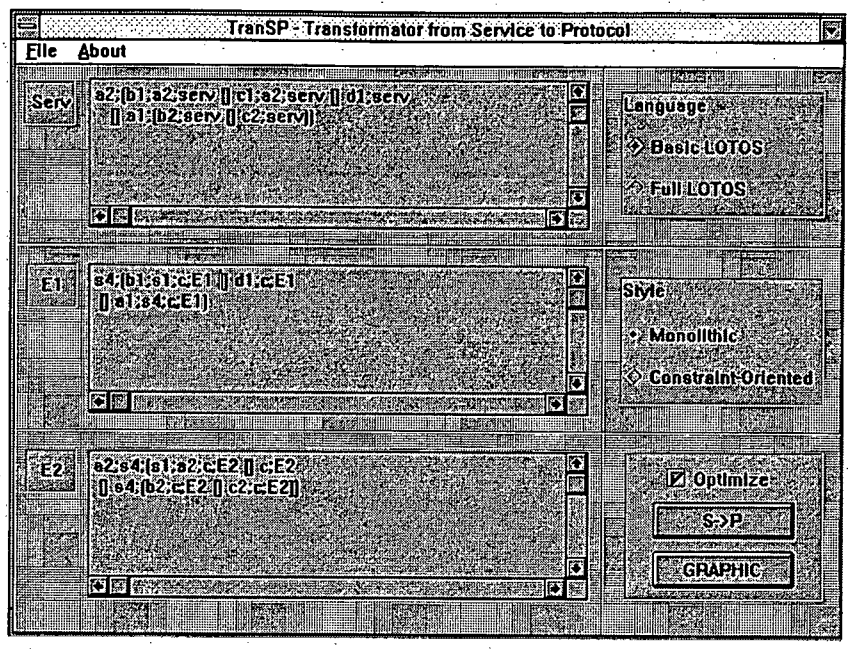


Figura 5.8 - Escolha indeterminística entre ações do mesmo sítio (algoritmo otimizado).

Em um sistema distribuído simples como, por exemplo, a máquina que serve café, chá ou sopa em [Lang90], são consideradas escolhas indeterminísticas entre ações de sítios diferentes. O sistema inicial (monolítico) é dividido em duas partes: **comercial** e **produtiva**. A parte comercial da máquina recebe, verifica e devolve moedas, enquanto que a parte produtiva oferece café ou chá no caso de inserção de uma moeda e sopa no caso de inserção de duas moedas. Veja a Figura 5.9.

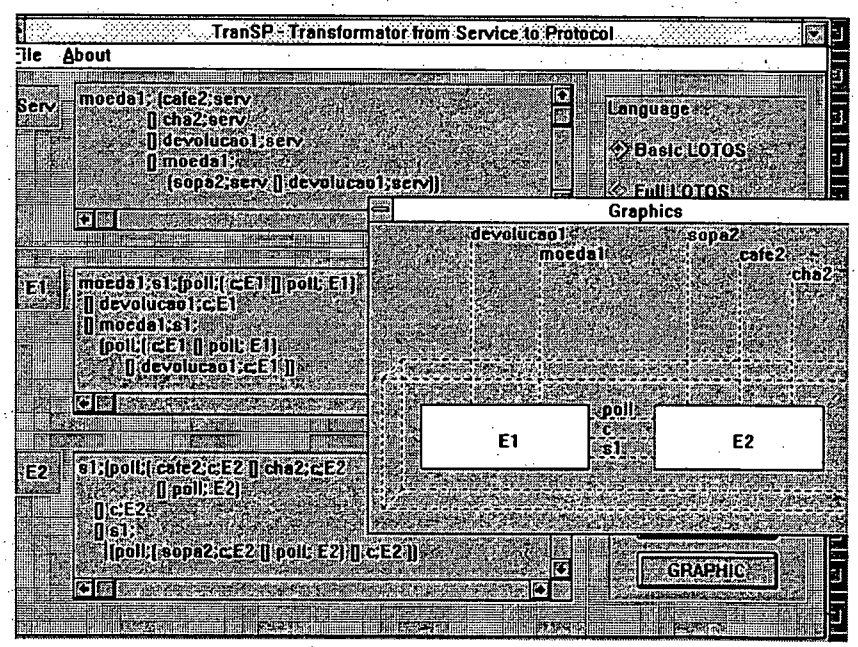


Figura 5.9 - Escolha indeterminística entre ações de sítios diferentes.

Na Figura 5.10 é apresentado um caso de habilitação de processos. Inicialmente, há uma escolha entre `a1;serv` e `b1;b2;exit`. Se a escolha recair sobre `a1;serv`, então haverá uma chamada recursiva e o sistema volta ao estado inicial. Quando a escolha recair sobre a segunda opção (`b1;b2;exit`), este `exit` habilitará a execução de `a2;exit`.

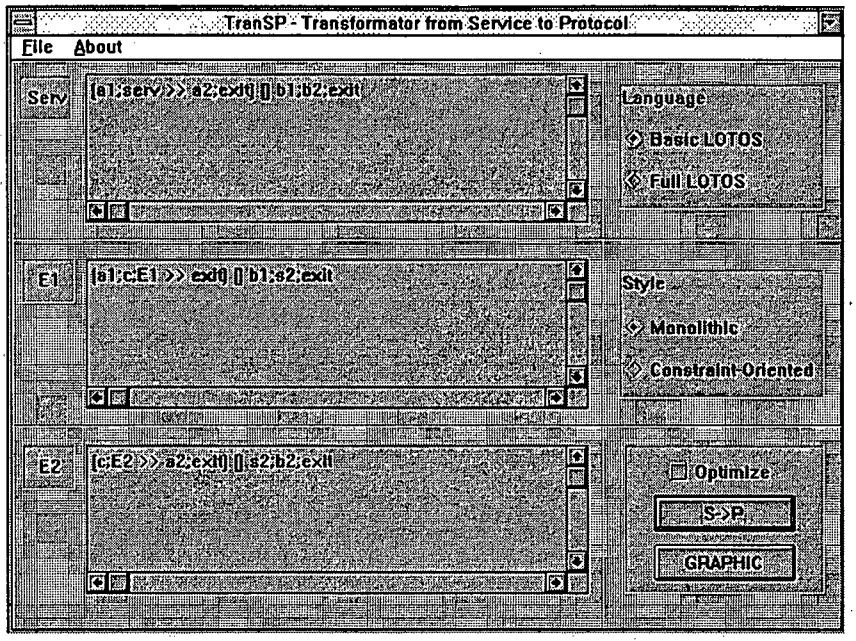


Figura 5.10 - Habilitação de processos.

Agora considere que a especificação anterior (Figura 5.10) pode ser interrompida, a qualquer momento, pela execução de `c2;b2;b1;exit`.

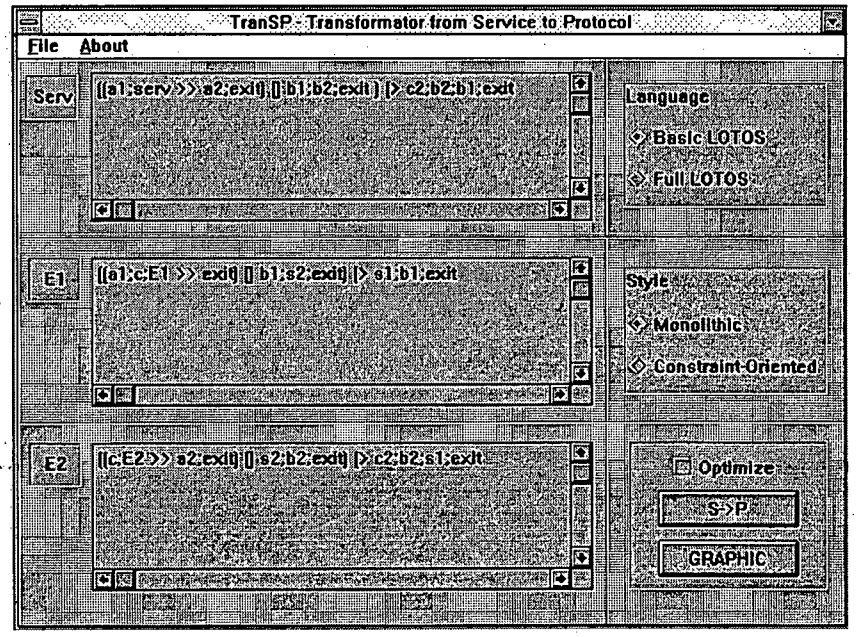


Figura 5.11 - Interrupção de processos.

6. Aplicação da ferramenta TranSP

Neste capítulo é apresentada uma aplicação prática da ferramenta TranSP, na área de gerenciamento de redes. Neste caso a ferramenta TranSP é utilizada para transformar uma parte do serviço **CMIS** [IS9595] na parte correspondente do protocolo **CMIP** [IS9596].

6.1. Introdução

A grande diversidade de sistemas atualmente disponíveis na área de redes de computadores tem tornado a atividade de gerenciamento dessas redes uma tarefa cada vez mais complexa. Normalmente, cada fornecedor oferece ferramentas próprias para o controle de sua rede, de modo que as diferentes tecnologias empregadas dificilmente permitem interoperação [Carv92].

Uma solução para esse problema de interoperação consiste em utilizar um computador que interage com diversos componentes da rede para extrair deles as informações necessárias para o gerenciamento da rede. Nesse caso pode-se montar um banco de dados neste computador gerente da rede, contendo todas as informações necessárias para o gerenciamento completo da mesma.

O gerenciamento de uma rede deve ser feito de maneira organizada e deve abranger a configuração de toda a rede (gerenciamento de configuração), identificar e resolver falhas físicas (gerenciamento de falhas), monitorar o desempenho (gerenciamento de desempenho), controlar e segurança (gerenciamento de segurança) e contabilizar os recursos desta rede (gerenciamento de contabilização) [Carv92].

Dentre os modelos de gerenciamento existentes, dois se destacam: o modelo de gerenciamento OSI, que utiliza o protocolo CMIP (*Common Management Information Protocol*) e o modelo de gerenciamento INTERNET que utiliza o protocolo SNMP (*Simple Network Management Protocol*). Ambos os protocolos (CMIP [IS9596] e SNMP [Carv92]) possuem o mesmo objetivo,

que é transmitir informações entre sistemas de gerenciamento de redes, dando condições ao gerente de atuar sobre os recursos gerenciados, recuperar informações e identificar problemas. As diferenças entre esses protocolos se dá, principalmente, na filosofia de acesso aos dados, funcionalidade, complexidade, desempenho, suporte de comunicação requerido e disponibilidade de produtos. O CMIP é orientado a conexão, enquanto que o SNMP opera no modo de acesso sem conexão. Além disso o CMIP é um padrão de direito, enquanto que o SNMP é um padrão de fato [Carv92].

Além das arquiteturas OSI e INTERNET, existem outras tais como, por exemplo, a arquitetura SNA. Às diversas arquiteturas de redes correspondem diversos sistemas de gerenciamento, tais como o Net-View da IBM, o UNMA da AT&T e o Open View da HP [Carv92].

A comunicação entre o gerente e o agente se dá com o auxílio de um serviço de comunicação. No modelo OSI tal serviço de comunicação é o CMIS.

6.2. Especificação do CMIS

O CMIS (*Common Management Information Service*) [IS9595] é o serviço de comunicação de informações de gerência que é oferecido pelo provedor CMISE às aplicações de gerência que seguem o Modelo OSI/ISO (veja a Figura 6.1). Este serviço pode ser utilizado por uma aplicação em um ambiente de gerenciamento, centralizado ou descentralizado, para intercambiar informações e comandos com o propósito de gerenciar sistemas. O serviço CMIS define ainda um conjunto de primitivas de serviço, assim como os parâmetros que são passados através de cada primitiva de serviço e as informações necessárias para a descrição semântica de cada primitiva de serviço.

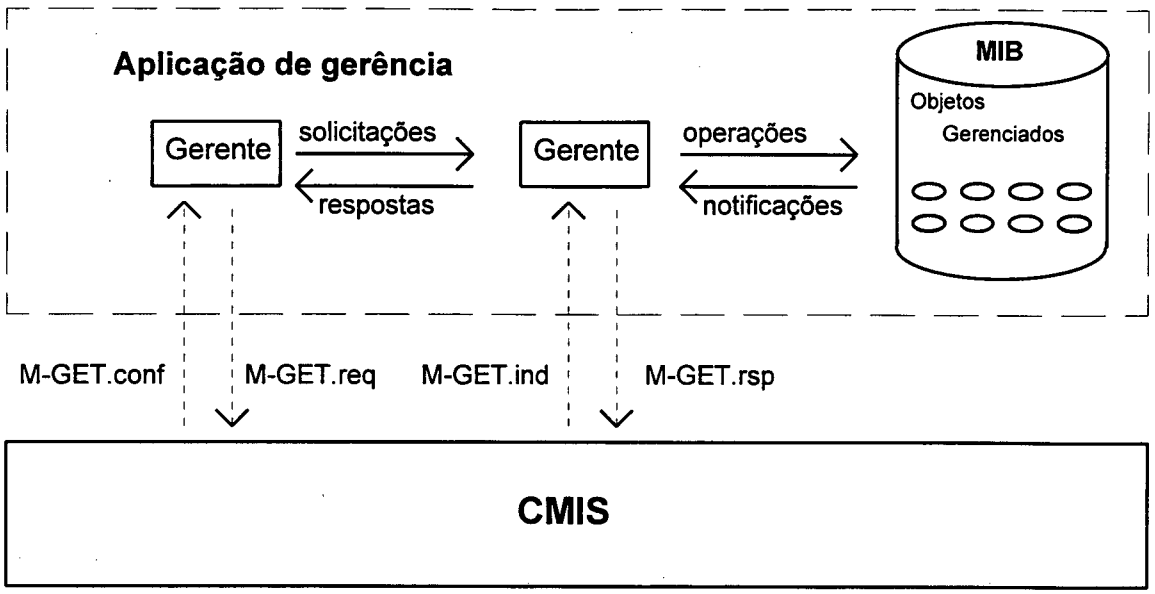


Figura 6.1 - Aplicação de gerência e serviço de comunicação CMIS.

6.3. Derivação do CMIP a partir do CMIS

O protocolo CMIP (*Common Management Information Protocol*) [IS9596] é o protocolo que implementa o CMIS (veja a Figura 6.2). Um importante trabalho com relação à especificação do CMIP pode ser encontrado em [LePe94]. A linguagem utilizada em tal trabalho foi a linguagem formal Estelle, e a especificação obtida foi verificada e simulada. Nesta seção busca-se mostrar como, na área de *gerência de redes*, uma especificação de protocolo pode ser obtida a partir de uma especificação de serviço correspondente, através do uso da ferramenta TranSP. Para ilustrar o uso da ferramenta TranSP nesse contexto, consideram-se as especificações formais, em LOTOS Básico, do CMIS e do CMIP da ISO.

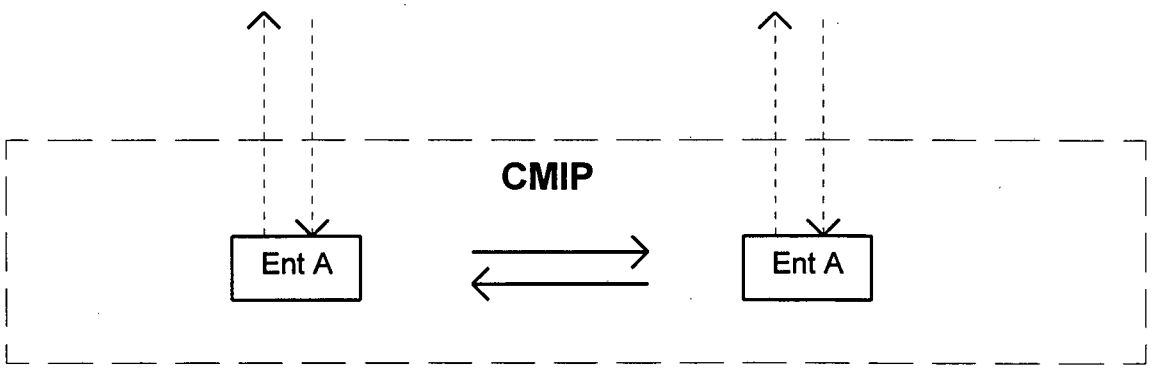


Figura 6.2 - Protocolo CMIP (protocolo que implementa o CMIS).

Para obter simplificação nesta seção, considera-se apenas uma parte do serviço CMISE. A parte considerada corresponde ao serviço M-GET e ao serviço M-CANCEL-GET. O serviço M-GET é empregado por um usuário (local) do serviço CMISE para obter valores de atributos. Tais valores são fornecidos por um outro usuário (remoto) do serviço CMISE. O serviço M-CANCEL-GET é empregado por um usuário (local) do serviço CMISE para cancelar um M-GET solicitado anteriormente.

Considerando apenas o serviço M-GET, pode-se especificar uma parte do comportamento CMISE como um processo de gerência cujo comportamento é definido como segue:

```
Serv := MGetReq1; MGetInd2; MGetRsp2; MGetCnf2; stop
```

Nesse caso, a ferramenta TranSP produz como resultado a saída apresentada na Figura 6.3.

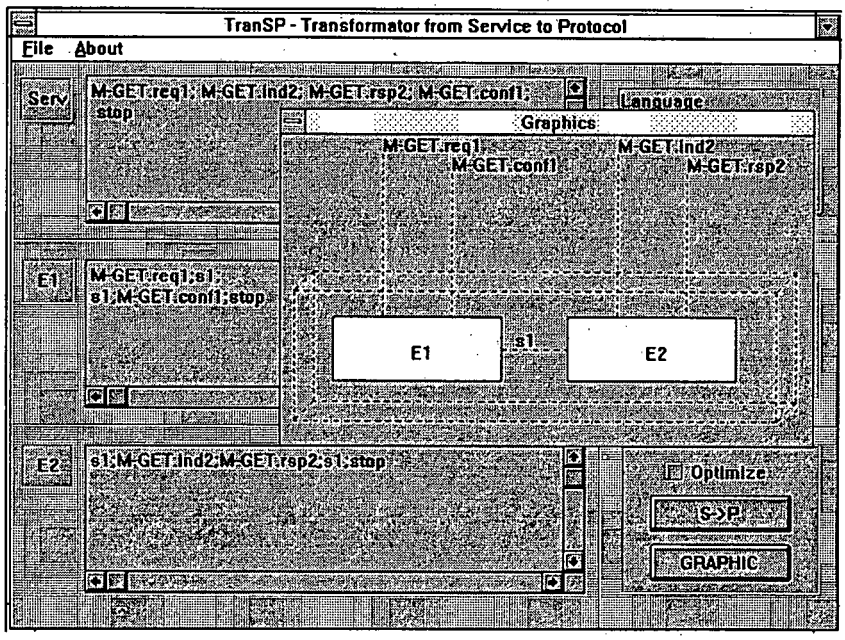


Figura 6.3 - Tela da ferramenta TranSP no caso do serviço M-GET.

O usuário *remoto* do serviço CMISE pode aceitar ou rejeitar a operação de request (`M-GET.req`). No caso de aceitação, ele deve executar uma primitiva `M-GET.rsp` para comunicar tal aceitação ao usuário do serviço CMISE. No caso da operação não poder ser

realizada, o usuário remoto do serviço CMISE pode rejeitar o request através da execução da primitiva `M-GET.rsp` com um código de erro apropriado. Neste caso, para fins de simplificação, esse código de erro é representado pela letra N - (Negativo) acompanhando a primitiva `M-GET.rsp`.

O provedor de serviço CMISE executa uma primitiva `M-GET.conf` (positiva ou negativa) completando a operação M-GET para qualquer dos casos (aceitação ou rejeição). Veja o exemplo a seguir:

```
Serv := MGetReq1; MGetInd2;
(MGetRsp2; MGetCnf1; stop  [] MGetRspN2; MGetCnfN1; stop)
```

Nesse caso, a ferramenta TranSP produz como resultado a saída apresentada na Figura 6.4.

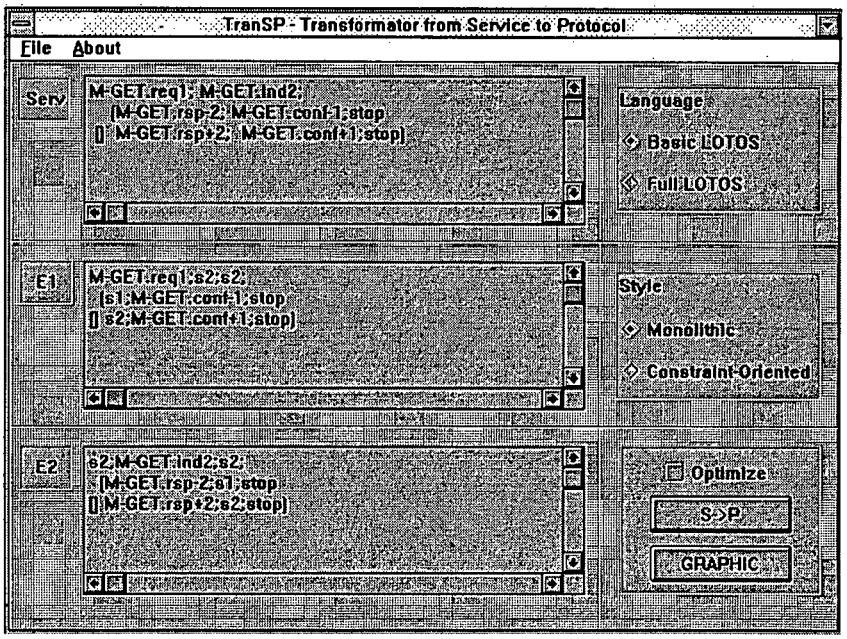


Figura 6.4 - Tela da ferramenta TranSP no caso do serviço M-GET com escolha indeterminística.

Para obtenção do protocolo CMIP, correspondente ao serviço M-GET e M-CANCEL-GET, deve-se considerar as normas que definem estes serviços, que são apresentadas a seguir.

Comportamentos correspondentes ao M-GET:

Comportamento 8.3.1.2.1. O usuário do serviço CMISE executa uma primitiva request do provedor de serviço CMISE para obter o valor de um atributo de um objeto. Veja a Figura 6.5.

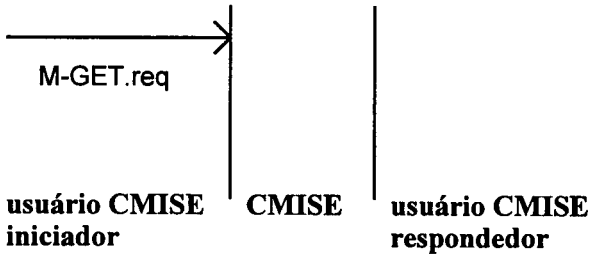


Figura 6.5 - Execução da primitiva M-GET.req.

Comportamento 8.3.1.2.2. Se o provedor do serviço CMISE aceita o request, então ele envia uma primitiva M-GET indication para o usuário do serviço CMISE. Caso contrário, o provedor do serviço CMISE rejeita o request e os seguintes procedimentos não são aplicados. Veja a Figura 6.6.

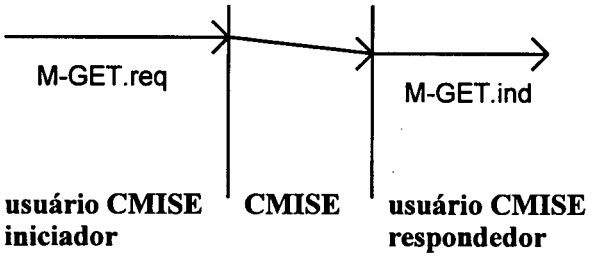


Figura 6.6 - Execução da primitiva M-GET.ind.

Comportamento 8.3.1.2.3. Se a operação não pode ser realizada, então o usuário do serviço CMISE rejeita o M-GET request executando uma primitiva M-GET response com o código de erro apropriado. Neste caso, os seguintes procedimentos não são aplicados. Veja a Figura 6.7.

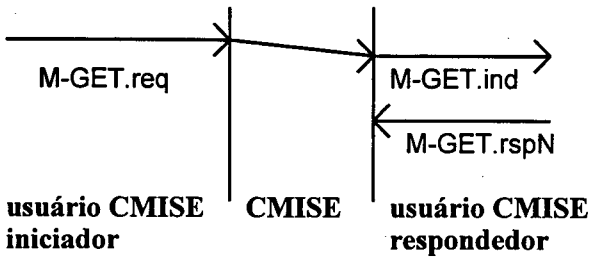


Figura 6.7 - Execução da primitiva M-GET.rsp com representação de código de erro.

Comportamento 8.3.1.2.4. Se somente um response será gerado, então os procedimentos 5, 6 e 7 devem ser ignorados.

Comportamento 8.3.1.2.5. O usuário do serviço CMISE recupera o valor do atributo requisitado e gera uma resposta que inclui resultado e/ou notificações de erros. O identificador de ligação deveria estar presente na primitiva de serviço, que é o valor a ser estabelecido igual ao identificador chamador da primitiva indication, e a classe de objeto gerenciado e instância deveriam estar presentes.

Comportamento 8.3.1.2.6. O provedor do serviço CMISE executa uma primitiva M-GET confirm para que o usuário do serviço CMISE chamador inclua o identificador de ligação e classe do objeto gerenciado e instância.

Comportamento 8.3.1.2.7. Os procedimentos 5 e 6 deverão ser repetidos até que as respostas sejam completadas.

Comportamento 8.3.1.2.8. Para completar a resposta, o usuário do serviço CMISE deve executar uma primitiva M-GET response que poderá não conter o identificador de ligação, e no caso de responses gerados pelos procedimentos 5 e 6, que não contêm a lista de atributos.

Comportamento 8.3.1.2.9. O provedor do serviço CMISE executa uma primitiva M-GET confirm para que o usuário do serviço CMISE chamador possa completar a operação M-GET. Veja a Figura 6.8.

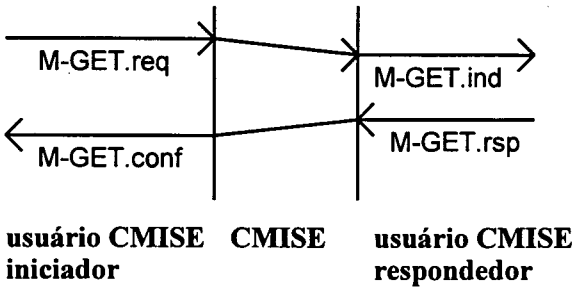


Figura 6.8 - Execução da primitiva M-GET.conf.

Comportamentos correspondentes ao M-CANCEL-GET:

Comportamento 8.3.1.3.2.1. O usuário do serviço CMISE chamador solicita a realização de um serviço CMISE para cancelar uma operação M-GET anteriormente requisitada. Tal se faz através da execução de uma primitiva M-CANCEL-GET request para que o provedor do serviço CMISE seja informado. Veja a Figura 6.9.

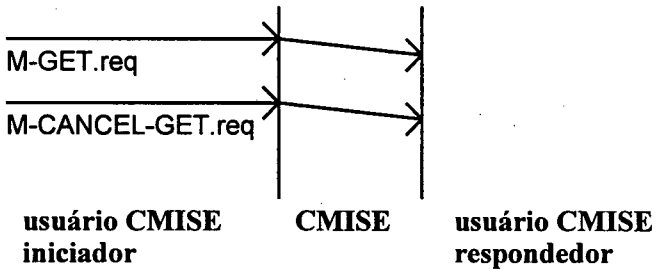


Figura 6.9 - Execução da primitiva M-CANCEL-GET.req.

Comportamento 8.3.1.3.2.2. Se o Provedor do serviço CMISE aceita o request, então ele executa uma primitiva M-CANCEL-GET indication para informar o usuário do serviço CMISE. Caso contrário, o provedor do serviço CMISE rejeita o request e os procedimentos seguintes não são aplicados.Veja a Figura 6.10.

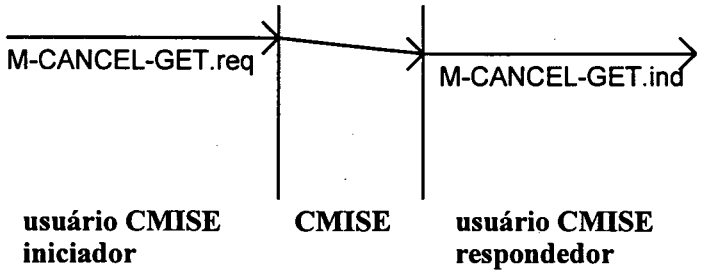


Figura 6.10 - Execução da primitiva M-CANCEL-GET.ind.

Comportamento 8.3.1.3.2.3. Se a operação não puder ser realizada, então o usuário do serviço CMISE rejeita o M-CANCEL-GET request executando uma primitiva M-CANCEL-GET response com o código de erro apropriado. Neste caso os procedimentos seguintes não são aplicados.Veja a Figura 6.11.

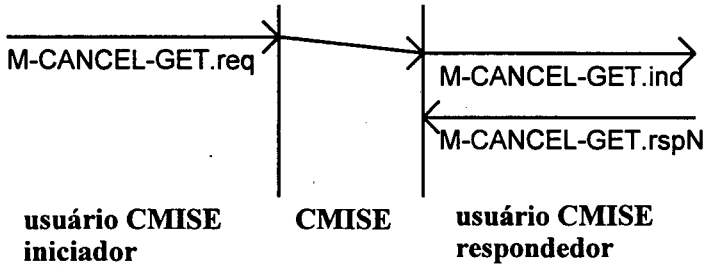


Figura 6.11 - Execução da primitiva M-CANCEL-GET.rsp com representação de código de erro.

Comportamento 8.3.1.3.2.4. O usuário do serviço CMISE cancela a operação M-GET pendente e executa uma primitiva M-GET response com código de erro apropriado, e um M-CANCEL-GET response para informar o provedor do serviço CMISE.Veja a Figura 6.12.

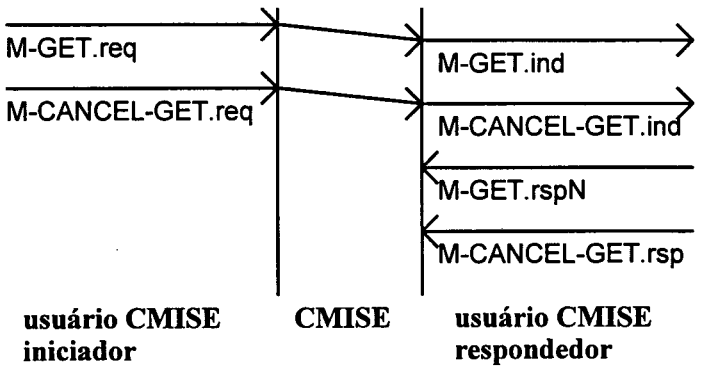


Figura 6.12 - Cancelamento da primitiva M-GET.req.

Comportamento 8.3.1.3.2.5. Se houver qualquer primitiva M-GET response executada pelo usuário do serviço CMISE depois que o usuário do serviço CMISE executar a primitiva M-CANCEL-GET request, mas antes que o usuário do serviço CMISE receba a informação de ocorrência da primitiva M-CANCEL-GET indication, então o usuário do serviço CMISE chamador deve informar-se através das primitivas M-GET confirm para aquelas primitivas M-GET response pendentes.Veja a Figura 6.13.

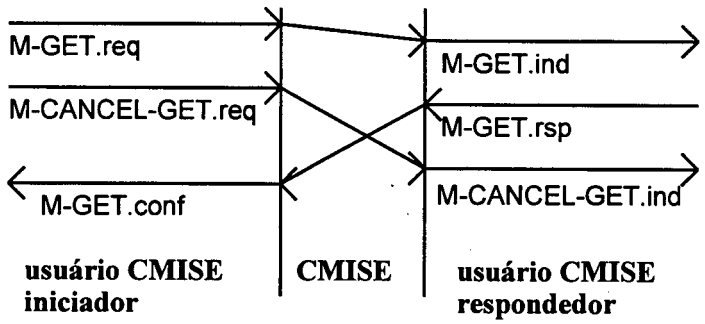


Figura 6.13 - Execução da primitiva M-GET.rsp antes da execução de M-CANCEL-GET.ind.

Comportamento 8.3.1.3.2.6. O provedor de serviço CMISE executa uma primitiva M-GET confirm para que o usuário do serviço CMISE chamador inclua a indicação de erro de operação cancelada, completando a operação M-GET, e uma primitiva M-CANCEL-GET confirm para que o usuário do serviço CMISE, complete a operação M-CANCEL-GET.Veja a Figura 6.14.

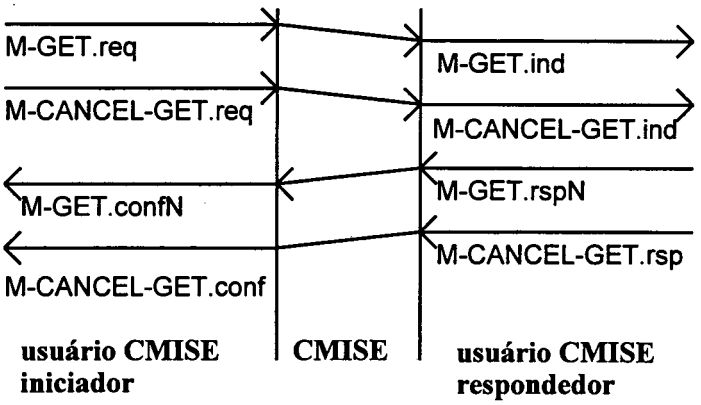


Figura 6.14 - Execução da primitiva M-CANCEL-GET.conf.

A partir dessas definições, pode-se então especificar o comportamento do serviço CMIS correspondente ao M-GET e ao M-CANCEL-GET.

Inicialmente, o usuário do serviço CMISE executa uma primitiva de request para que o provedor do serviço CMISE obtenha o valor de um atributo de um objeto (Comportamento 8.3.1.2.1). A partir desse request, dois comportamentos ser tomados, identificados por A ou B.

```
GetRq1;(A...
      []
         B...)
```

A) Se o provedor do serviço CMISE aceita o request, então ele executa uma primitiva M-GET indication para informar o usuário do serviço CMISE (Comportamento 8.3.1.2.2).

B) O usuário do serviço CMISE chamador solicita a realização de um serviço CMISE para cancelar uma operação M-GET anteriormente requisitada através da execução de uma primitiva M-CANCEL-GET request para informar o provedor do serviço CMISE (Comportamento 8.3.1.3.2.1).

Se a opção A for escolhida, o comportamento prossegue com três opções: C, D ou E.

```
GetRq1;(i2; GetInd2; ( C...
                []
                     D...
                []
                     E... ))
```

C) Para completar a resposta, o usuário do serviço CMISE deve executar uma primitiva M-GET response (Comportamento 8.3.1.2.8).

D) Se a operação não pode ser realizada, então o usuário do serviço CMISE rejeita o M-GET request executando uma primitiva M-GET response com o código de erro apropriado (Comportamento 8.3.1.2.3).

E) Neste caso ocorre o mesmo da letra B (Comportamento 8.3.1.2.9).

Se a opção C ou D forem escolhidas, o comportamento prossegue com o provedor do serviço CMISE executando uma primitiva M-GET confirm, para que o usuário do serviço CMISE chamador, complete a operação M-GET (Comportamento 8.3.1.2.9).

Se a opção E for escolhida, o comportamento prossegue com as opções F ou G.

```
GetRq1;(i2;GetInd2; (GetRsp2;i1;GetCnf1;Serv
                []
                    GetRspN2;i1;GetCnfN1;Serv
                []
                    CanGetRq1;i2;CanGetInd2; (F...
                                             [] G...)))
```

F) O usuário do serviço CMISE cancela a operação M-GET pendente e executa uma primitiva M-GET response que conterá o erro de operação cancelada (Comportamento 8.3.1.3.2.4). O provedor de serviço CMISE executa uma primitiva M-GET confirm para que o usuário do serviço CMISE chamador inclua a indicação de erro de operação cancelada, completando a operação M-GET, e uma primitiva M-CANCEL-GET confirm para que o usuário do serviço CMISE, complete a operação M-CANCEL-GET (Comportamento 8.3.1.3.2.6).

G) Se a operação não puder ser realizada, então o usuário do serviço CMISE rejeita o M-CANCEL-GET request executando uma primitiva M-CANCEL-GET response com o código de erro apropriado (Comportamento 8.3.1.3.2.3).

No caso da opção B ter sido escolhida, o comportamento prossegue com três opções: H, I ou J.

```
GetRq1;(i2; GetInd2; (GetRsp2;i1;GetCnf1;Serv
                []
                    GetRspN2;i1;GetCnfN1;Serv
                []
            CanGetRq1;i2;CanGetInd2;(GetRspN2;CanGetRsp2;i1;GetCnfN1;CanGetCnf1;Serv
                                        []
                                    CanGetRspN2;GetRsp2;i1;GetCnf1;CanGetCnfN1;Serv))

      []
         CanGetRq1;i2;GetInd2;(H...
                              []
                                 I...
                              []
                                 J...)
```

H) No caso de haver qualquer primitiva M-GET response executada pelo usuário do serviço CMISE depois que o usuário do serviço CMISE executar a primitiva M-CANCEL-GET request, mas antes que o usuário do serviço CMISE receba informação da ocorrência da primitiva M-CANCEL-GET indication, então o usuário do serviço CMISE chamador deve receber informação de ocorrência das primitivas M-GET confirm para aquelas primitivas M-GET response pendentes (Comportamento descrita em Comportamento 8.3.1.3.2.5).

I) Nesse caso ocorre o mesmo da letra D (Comportamento 8.3.1.2.3).

J) O usuário do serviço CMISE cancela a operação M-GET pendente e executa uma primitiva M-GET response que conterá o erro de operação cancelada e um M-CANCEL-GET response para informar o provedor do serviço CMISE (Comportamento 8.3.1.3.2.4).

No caso da ocorrência da opção J, o comportamento prossegue com as opções K ou L.

```
GetRq1;(i2; GetInd2; (GetRsp2;i1;GetCnf1;Serv
                []
                    GetRspN2;i1;GetCnfN1;Serv
                []
          CanGetRq1;i2;CanGetInd2;(GetRspN2;CanGetRsp2;i1;GetCnfN1;i1;CanGetCnf1;Serv
                                        []
                                   CanGetRspN2;GetRsp2;i1;GetCnf1;i1;CanGetCnfN1;Serv))

      []

CanGetRq1;i2;GetInd2;(GetRsp2;i1;GetCnf1;i2;CanGetInd2;CanGetRspN2;i1;CanGetCnfN1;
                                                                              Serv
                        []
                  GetRspN2;i1;GetCnfN1;i2;CanGetInd2;CanGetRspN2;i1;CanGetCnfN1;Serv
                        []
                          i2;CanGetInd2;(K...
                                    []
                                      L...
```

K) O provedor de serviço CMISE executa uma primitiva M-GET confirm para que o usuário do serviço CMISE chamador inclua a indicação de erro de operação cancelada, completando a operação M-GET, e uma primitiva M-CANCEL-GET confirm para que o usuário do serviço CMISE, complete a operação M-CANCEL-GET (Comportamento 8.3.1.3.2.6).

L) Se a operação não puder ser realizada, então o usuário do serviço CMISE rejeita o M-CANCEL-GET request executando uma primitiva M-CANCEL-GET response com o código de erro apropriado. (Comportamento 8.3.1.3.2.3).

```
GetRq1;(i2; GetInd2; (GetRsp2;i1;GetCnf1;Serv
                []
                    GetRspN2;i1;GetCnfN1;Serv
                []
```

```
CanGetRq1;i2;CanGetInd2;(GetRspN2;CanGetRsp2;i1;GetCnfN1;i1;CanGetCnf1;Serv
                                    []
                                CanGetRspN2;GetRsp2;i1;GetCnf1;i1;CanGetCnfN1;Serv))

  []

CanGetRq1;i2;GetInd2;(GetRsp2;i1;GetCnf1;i2;CanGetInd2;CanGetRspN2;i1;CanGetCnfN1;
                                                                        Serv
                        []
                GetRspN2;i1;GetCnfN1;i2;CanGetInd2;CanGetRspN2;i1;CanGetCnfN1;Serv
                        []
                i2;CanGetInd2;(GetRspN2;CanGetRsp2;i1;GetCnfN1;i1;CanGetCnf1;Serv
                                    []
                                CanGetRspN2;GetRsp2;i1;GetCnf1;i1;CanGetCnfN1;Serv)))
```

Nesse caso, a ferramenta TranSP produz como resultado a saída apresentada na Figura 6.15.

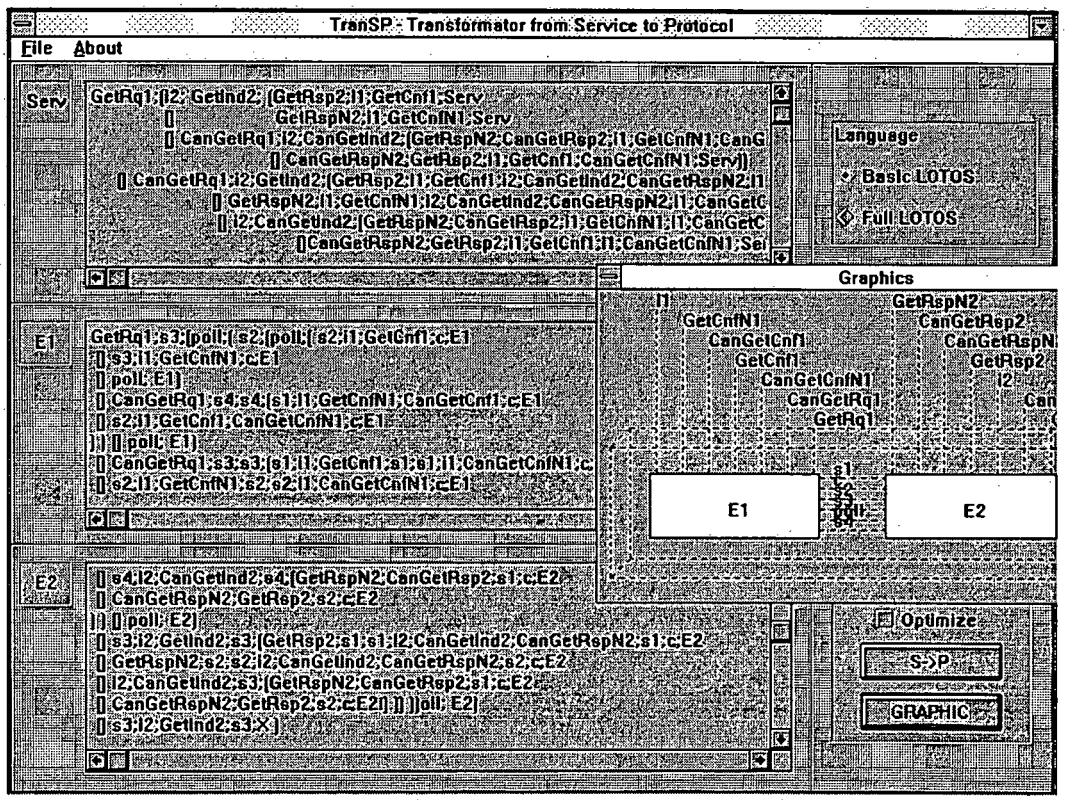


Figura 6.15 - Tela da ferramenta TranSP no caso do serviço M-GET e o serviço M-CANCEL-GET.

Através da opção **Save Protocol**, gera-se a especificação completa do protocolo correspondente ao serviço apresentado acima, que é a composição independente dos processos derivados E1 e E2 juntamente com a declaração do nome das portas. Esta especificação é apresentada a seguir.

```
specification E1E2[GetCnfN1,CanGetCnf1,GetCnf1,CanGetCnfN1,CanGetRq1,GetRq1,
    i1,i2,GetRspN2,CanGetRsp2,CanGetRspN2,GetRsp2,CanGetInd2,GetInd2] :noexit
 behaviour
    hide s1,c,s2,s3,poll,s4 in (
    E1[GetCnfN1,CanGetCnf1,GetCnf1,CanGetCnfN1,
       CanGetRq1,GetRq1,s1,c,s2,s3,poll,s4,i1]
    |[s1,c,s2,s3,poll,s4]|
    E2[GetRspN2,CanGetRsp2,CanGetRspN2,GetRsp2,
       CanGetInd2,GetInd2,s1,c,s2,s3,poll,s4,i2])
 where
 process E1[GetCnfN1,CanGetCnf1,GetCnf1,CanGetCnfN1,
          CanGetRq1,GetRq1,s1,c,s2,s3,poll,s4] :noexit:=
 GetRq1;s3;(poll;( s2;(poll;( s2;i1;GetCnf1;c;E1[GetCnfN1,CanGetCnf1,GetCnf1,
                                                 CanGetCnfN1,CanGetRq1,GetRq1,
                                                  s1,c,s2,s3,poll,s4,i1]
                              [] s3;i1;GetCnfN1;c;E1[GetCnfN1,CanGetCnf1,
                                                     GetCnf1,CanGetCnfN1,CanGetRq1,
                                                     GetRq1,s1,c,s2,s3,poll,s4,i1]
                              [] poll; E1[GetCnfN1,CanGetCnf1,GetCnf1,
                                          CanGetCnfN1,CanGetRq1,GetRq1,
                                          s1,c,s2,s3,poll,s4,i1])
                 [] CanGetRq1;s4;s4;(s1;i1;GetCnfN1;i1;CanGetCnf1;c;E1[GetCnfN1,
                                     CanGetCnf1,GetCnf1,CanGetCnfN1,
                                    CanGetRq1,GetRq1,s1,c,s2,s3,poll,s4,i1]
                                  [] s2;i1;GetCnf1;i1;CanGetCnfN1;c;E1[GetCnfN1,
                                     CanGetCnf1,GetCnf1,CanGetCnfN1,
                                    CanGetRq1,GetRq1,s1,c,s2,s3,poll,s4,i1]))
          [] poll; E1[GetCnfN1,CanGetCnf1,GetCnf1,CanGetCnfN1,
                      CanGetRq1,GetRq1,s1,c,s2,s3,poll,s4,i1])
[]CanGetRq1;s3;s3;(s1;i1;GetCnf1;s1;s1;i1;CanGetCnfN1;c;E1[GetCnfN1,
                                    CanGetCnf1,GetCnf1,CanGetCnfN1,CanGetRq1,
                                                GetRq1,s1,c,s2,s3,poll,s4,i1]
                 [] s2;i1;GetCnfN1;s2;s2;i1;CanGetCnfN1;c;E1[GetCnfN1,CanGetCnf1,
                                                GetCnf1,CanGetCnfN1,CanGetRq1,
                                                GetRq1,s1,c,s2,s3,poll,s4,i1]
                 [] s3;(s1;i1;GetCnfN1;i1;CanGetCnf1;c;E1[GetCnfN1,CanGetCnf1,
                                                GetCnf1,CanGetCnfN1,CanGetRq1,
                                                GetRq1,s1,c,s2,s3,poll,s4,i1]
                       [] s2;i1;GetCnf1;i1;CanGetCnfN1;c;E1[GetCnfN1,CanGetCnf1,
                                                GetCnf1,CanGetCnfN1,CanGetRq1,
                                                GetRq1,s1,c,s2,s3,poll,s4],i1)))
endproc
process E2[GetRspN2,CanGetRsp2,CanGetRspN2,GetRsp2,
             CanGetInd2,GetInd2,s1,c,s2,s3,poll,s4,i2] :noexit:=
    s3;(poll;(i2;GetInd2;s2;(poll;( GetRsp2;s2;c;E2[GetRspN2,CanGetRsp2,
                                              CanGetRspN2,GetRsp2,CanGetInd2,
                                              GetInd2,s1,c,s2,s3,poll,s4,i2]
                                 [] GetRspN2;s3;c;E2[GetRspN2,CanGetRsp2,
                                                CanGetRspN2,GetRsp2,CanGetInd2,
```

```
                                        GetInd2,s1,c,s2,s3,poll,s4,i2]
                                  [] poll; E2[GetRspN2,CanGetRsp2,CanGetRspN2,
                                        GetRsp2,CanGetInd2,GetInd2,
                                        s1,c,s2,s3,poll,s4,i2] )
                          []s4;i2;CanGetInd2;s4;
                              (GetRspN2;CanGetRsp2;s1;c;E2[GetRspN2,CanGetRsp2,
                                        CanGetRspN2,GetRsp2,CanGetInd2,
                                           GetInd2,s1,c,s2,s3,poll,s4,i2]
                            [] CanGetRspN2;GetRsp2;s2;c;E2[GetRspN2,CanGetRsp2,
                                        CanGetRspN2,GetRsp2,CanGetInd2,
                                        GetInd2,s1,c,s2,s3,poll,s4,i2]))
                [] poll; E2[GetRspN2,CanGetRsp2,CanGetRspN2,GetRsp2,
                           CanGetInd2,GetInd2,s1,c,s2,s3,poll,s4])
      [] s3;i2;GetInd2;s3;(GetRsp2;s1;s1;i2;CanGetInd2;CanGetRspN2;s1;c;
                           E2[GetRspN2,CanGetRsp2,CanGetRspN2,GetRsp2,
                              CanGetInd2,GetInd2,s1,c,s2,s3,poll,s4,i2]
                        [] GetRspN2;s2;s2;i2;CanGetInd2;CanGetRspN2;s2;c;
                           E2[GetRspN2,CanGetRsp2,CanGetRspN2,GetRsp2,
                           CanGetInd2,GetInd2,s1,c,s2,s3,poll,s4,i2]
                        [] i2;CanGetInd2;s3;
                                (GetRspN2;CanGetRsp2;s1;c;E2[GetRspN2,
                                 CanGetRsp2,CanGetRspN2,GetRsp2,
                                 CanGetInd2,GetInd2,s1,c,s2,s3,poll,s4,i2]
                              [] CanGetRspN2;GetRsp2;s2;c;E2[GetRspN2,
                                 CanGetRsp2,CanGetRspN2,GetRsp2,CanGetInd2,
                                 GetInd2,s1,c,s2,s3,poll,s4,i2] )))
 endproc
endspec
```

6.4. Simulação

Com a utilização da ferramenta Caesar pode-se, através da opção **executor**, simular a execução da especificação obtida.

```
caesar.open e1e2 executor
maximal number of visible transitions (between 0 and 4294967295) ? 28
available execution strategies:
   (1) deterministic
   (2) non-deterministic with random seed
   (3) non-deterministic with chosen seed
chosen strategy (between 1 and 3) ? 2
```

```
random seed = 811008412
GETRQ1          GETRQ1          GETCNF1         GETRQ1          GETRQ1
I2              CANGETRQ1       I2              I2              I2
GETIND2         I2              CANGETIND2      GETIND2         GETIND2
GETRSP2         GETIND2         CANGETRSPN2     GETRSPN2        CANGETRQ1
I1              GETRSP2         I1              I1
GETCNF1         I1              CANGETCNFN1     GETCNFN1
<39 states visited, 28 visible transitions executed>
```

7. Conclusões e futuros trabalhos

Neste trabalho foi feito um estudo das abordagens de diversos autores sobre transformações de especificações. A partir desse estudo, foram desenvolvidos algoritmos que permitem derivar especificações de protocolo automaticamente a partir de especificações de serviço. A transformação de especificações realizada através desses algoritmos é feita de tal modo que dispensa a prova de correção *a posteriori*. Esses algoritmos foram apresentados na forma de uma pseudo-linguagem e foram implementados em C++ para Windows, constituindo desse modo a ferramenta TranSP (Transformação de especificação de Serviço em especificação de Protocolo)

A ferramenta TranSP tem por principal objetivo auxiliar no projeto e desenvolvimento de protocolos e sistemas distribuídos, facilitando o trabalho de especificação e validação das especificações. A ferramenta TranSP recebe, como entrada, uma especificação de serviço e gera, como saída, a especificação das entidades de protocolo correspondentes (entidade E1 e entidade E2). Tem-se a opção de gerar a especificação final de protocolo, que é a composição independente das duas entidades E1 e E2.

A ferramenta TranSP foi aplicada no caso de um projeto de sistema de gerência de redes de computadores onde, a partir de uma parte do serviço CMIS, foi gerado o protocolo CMIP correspondente. Esse protocolo foi simulado com a utilização das ferramentas CÆSAR e LOLA. A ferramenta TranSP admite somente o LOTOS básico e os estilos monolítico e orientado a recursos, mas permite que futuras extensões dêem mais flexibilidade à ferramenta.

Também buscou-se apresentar a Técnica de Descrição Formal LOTOS em um ambiente gráfico agradável, onde a derivação automática de especificações de protocolo, a partir do serviço seja algo de fácil compreensão. Isso incentiva o interesse em LOTOS e diminui bastante a deficuldade que muitos têm por acreditarem que LOTOS é muito complexa. Tal ambiente também contribui para a difusão de LOTOS em ambientes de trabalho fora das universidades.

Com relação aos trabalhos existentes na área de transformações de especificações que considera a derivação do protocolo a partir do serviço, pode-se considerar duas linhas de pesquisa, uma realizada por Gregor von Bochmann e outra realizada por Rom Langerak.

Com relação ao trabalho realizado por Bochmann, pode-se concluir que o trabalho aqui apresentado possui restrição de ter-se que trabalhar com apenas duas entidades de protocolo (Bochmann considera **n** entidades) e não trata o operador de paralelismo, embora os algoritmos são muito mais simples que a gramática de compiladores utilizada por Bochmann e também tratam o caso de escolha distribuída (não considerado por Bochmann).

Exceto para os casos onde tratamos a habilitação e a interrupção, os algoritmos apresentados implementam as transformações propostas por Langerak.

Na continuidade deste trabalho, pode-se integrar esta ferramenta com outros trabalhos desenvolvidos na área, como por exemplo, com o ambiente integrado LOWE (LOTOS Windows Environment)[Nota95], bem como estender a funcionalidade da mesma para a parte de dados de LOTOS e permitindo também a utilização do estilo orientado a restrições ou do estilo orientado a recursos. Além disso, pretende-se apresentar as provas para os algoritmos de habilitação e interrupção, bem como desenvolver novos algoritmos para tratar os casos de paralelismo (geral `|[ ]|`, dependente `||` e independente `|||`).

8. Anexos

Nessa seção são apresentados os anexos desse trabalho. Eles constituem algumas ferramentas LOTOS disponíveis, aspectos de projetos e desenvolvimento de sistemas distribuídos, métodos para validações de sistemas e o código de implementação da ferramenta TranSP.

Anexo A. Ferramentas de apoio ao projeto

Nessa seção são apresentados algumas ferramentas LOTOS disponíveis atualmente para apoio de projetos e aspectos de projeto e desenvolvimento de sistemas distribuídos.

Muitas ferramentas têm sido desenvolvidas para auxiliar o projeto de sistemas com o uso de LOTOS. Algumas dessas ferramentas oferecem editor próprio e possibilitam a geração de diversos tipos de saídas como Redes de Petri, grafos de estados, etc. Ambientes de desenvolvimento como TOPO possibilitam uma série de simulações e geração de código de implementação, em C ou ADA. Descrições dessas ferramentas são encontradas, por exemplo, em [Vloe93].

A.1. Aldébaran

É um verificador que permite comparar sistemas de transições rotuladas quanto à equivalência de bissimulação forte e equivalência de observação. Suporta apenas LOTOS básica. Essa ferramenta numera cada estado com um número natural.

A ferramenta Caesar possui interface com ALDÉBARAN (gera arquivo .aut). Caesar/ Aldébaran podem ser utilizados para verificar problemas como deadlock.

A ferramenta aldébaran numera cada estado com um número natural. A primeira linha do arquivo .aut é chamado descritor, e tem a seguinte estrutura:

des(primeiro_estado, número_de_transições, número_de_estados)

Cada uma das demais linhas representa um deslocamento e tem a seguinte estrutura:

(do_estado,"nome_porta!valor1...!valorn",para_estados)

sintaxe:

aldebaran [option] *name1 name2 ...*

-help: Ajuda

-bin: Indica arquivos binários

-inv: Representação inversa para sistemas de transição rotulados (default: representação direta

-stat: Algumas estatísticas são mostradas

-fly: A igualdade dos sistemas de transições rotulados, contida em *nome1* e *nome2* com respeito à equivalência de bissimulação fraca (tau* a) é testada. O resultado TRUE ou FALSE é mostrado.

-bmin, -omin, -amin, -smin: Gera sistemas de transições rotuladas para uma especificação, com respeito à bissimulação forte, equivalência de observação, equivalência aceitável e equivalência segura.

-bequ, -oequ, -aequ, -sequ: Testa a equivalência entre duas especificações mostrando na tela TRUE ou FALSE, com respeito à bissimulação forte, equivalência de observação, equivalência aceitável e equivalência segura.

-bcla, -ocla, -acla, -scla: Para cada sistema de transição rotulado, contido em *name1*, são mostradas classes de equivalência com respeito à bissimulação forte, equivalência de observação, modelo de equivalência aceitável ou equivalência segura.

-b, -o, -a e -s significam respectivamente:

b (bissimulação)

o (observacão)

a (modelo aceitável)

s (segura)

A.2. ASDE

Suporta LOTOS básica em duas abordagens para desenvolver especificações: abordagem tradicional (baseada na edição estruturada seguida por uma análise) e abordagem baseada em transformações (desenvolvimento formal de aplicação de regras de transformação com correção preservada).

O ambiente é composto pelas seguintes ferramentas:

- dois editores
- um kernel baseado em transformações (para definir, selecionar e aplicar regras)
- listas de regras de transformação
- um expansor de expressões de comportamento
- um tradutor para Redes de Petri
- um controlador de desenvolvimento

A.3. Auto

Suporta LOTOS básica. É um verificador de sistemas que envolvem comunicação e paralelismo. Gera autômatos finitos a partir de especificações de comportamento.

A.4. Ferramenta CÆSAR

CÆSAR pertence à família de software CESAR que contém diversas ferramentas (QUASAR, XESAR, CÆSAR) para verificação e validação de sistemas distribuídos. Estas ferramentas seguem os mesmos princípios:

O sistema a ser verificado deve ser escrito em alguma linguagem de especificação formal, tal como QUASAR, ESTELLE e LOTOS.

A descrição formal é compilada para um grafo de estado (autômato de estados finitos, chamado também de sistema de transições rotuladas).

O tradutor segue diversos passos. Primeiramente a descrição é traduzida para um modelo intermediário, constituindo uma representação de fluxo de controle paralelo e fluxo de dados. XESAR usa autômatos de comunicação (máquinas abstratas) enquanto que CÆSAR é baseado em redes dePetri.

Esta forma intermediária é traduzida para um gráfico, através da aplicação da *análise de alcançabilidade*, isto é, *simulação exaustiva* para todas as possíveis evoluções do sistema. A tradução é eficiente por causa da estrutura de controle estática fornecida pelo modelo intermediário.

As propriedades a serem verificadas são avaliadas no gráfico. Estas podem ser expressas como fórmulas de alguma lógica temporal (como CTL para QUASAR [FRV85] ou LTAC para XESAR [Rod88]). Elas podem também consistir na comparação de um gráfico com outro, para relação de equivalência (tal como equivalência de observação, equivalência de teste, etc.).

Funcionamento da ferramenta CÆSAR:

Ela recebe um programa LOTOS como entrada e produz um gráfico de rede correspondente e gráfico de equivalência forte.

O processo de tradução segue os seguintes passos:

análise léxica e sintática,

análise semântica,

restrição,

expansão,

geração,

otimização e

simulação

Sintaxe: caesar [-option$_1$] ... [-option$_n$] arquivo.lotos

Entradas e saídas:

A execução de CÆSAR considera 4 tipos de entradas :

opções na linha de comando

o programa LOTOS (arquivo "*arq.lotos*")

as bibliotecas LOTOS (arquivos "*type.lib*")

a implementação C dos tipos abstratos de dados (arquivo "*arq.h*")

e gera 4 tipos de saídas:

o gráfico (arquivos "*arq.aut*" se usada a opção *-aldebaran*, "*arq.gph*", ...)

a rede (arquivo "*arq.net*")

os mapeamentos abstrato/concreto (saída padrão e arquivo "*arq.err*")

os diagnósticos

Obs.: Ao se utilizar simplesmente CÆSAR <arq.lotos> , são geradas as seguintes saídas:

arq.gph

arq.net

arq.err (arquivo gerado no caso de ser detectado algum erro no processo de compilação).

Opções de interface:

CÆSAR fornece interface com sete sistemas de verificação existentes: ALDEBARAN, AUTO, MEC, PIPN, SCAN, SQUIGGLES e XESAR. Ele pode gerar gráficos corretos no formato esperado por cada ferramenta. Para utilizar uma destas ferramentas, uma opção apropriada deve ser selecionada. Diversas opções podem ser estabelecidas simultaneamente. Qualquer que seja a opção que o usuário escolher, a semântica do gráfico permanece a mesma, mas o número, o sufixo e o formato do arquivo gerado mudam. Veja a Tabela A.1

Tabela A.1 - Ferramentas que possuem interface com a ferramenta CÆSAR.

FERRAMENTA	INSTITUIÇÃO	OPÇÃO	ARQUIVOS GERADOS
ALDEBARAN	LGI-IMAG (Grenoble)	aldebaran	arq.aut
AUTO	INRIA (Sophia)	auto	arq.m0
MEC	University of Bordeauux I	mec	arq.mec
PIPN	LAAS (Tolouse)	pipn	arq.auto.pro
SCAN	BULL	scan	arq.scan
SQUIGGLES	CNUCE (Pisa)	squiggles	arq.graph
XESAR	LGI-IMAG (Grenoble)	xesar	arq.gra, arq.dp3, arq.ge3, arq.tai

A.5. Centaur

Suporta LOTOS básico. É formado por cinco componentes:

- editores
- impressor de árvore sintática
- verificador de semântica
- ferramenta para transformação de especificações
- simulador

A.6. Data Type Compiler

Suporta parte da linguagem LOTOS e gera código Assembler. Escrito em C e Prolog.

A.7. Lcris

Esta ferramenta foi implementada em Yacc para ser utilizada com o sistema operacional DOS, para equipamento IBM PC e compatíveis. Suporta LOTOS dinâmica e estática e é constituída de um analisador para sintaxe e semântica estática.

Sintaxe: LC [-cfile] [-lfile] [-s] arquivo.lot

exemplos:

lc arq.lot	= compila o arquivo
lc -s arq.lot	= compila e lista tabela de símbolos
lc -c arq arq.lot	= compila e gera arquivo arq.cod
lc -l arq arq.lot	= compila e gera arquivo arq.lis

A.8. LIS (LOTOS Interpretation Server)

Suporta LOTOS completa e realiza as seguintes funções:

- análise de especificações
- transformação em LOTOS canônico.
- simulação
- definições ADT

A.9. Lisp based LOTOS environment

Simulador que suporta LOTOS básica.

A.10. LITE (Lotosphere Integrated Tool Environment)

É um ambiente com ferramentas integradas em três tipos de objetos:

- editores
- semântica
- apresentação textual

Suporta as seguintes funções:

- editor
- verificador semântico
- gerador de relatórios
- simulador
- compilador (processos e tipos abstratos de dados LOTOS em C)
- transformações com correção preservada
- verificação de equivalência
- testador

A.11. LOLA (LOTOS Laboratory)

Desenvolvida pelo departamento de Engenharia Telemática da Universidade de Madrid, Espanha, é uma ferramenta experimental que suporta diferentes aspectos de projetos LOTOS. Seu desempenho capacita-a a ser utilizada em grandes especificações LOTOS de maneira eficiente.

sintaxe: LOLA <spec.lot> <lib>, onde

<spec.lot> é o nome da especificação LOTOS e <lib> é a biblioteca utilizada (caso necessário)

A versão atual é a 3R6, e tem as seguintes características:

- Depuração e simulação passo-a-passo de especificações LOTOS
  - O usuário manipula a execução passo-a-passo
  - O usuário pode atribuir valores para expressões opcionalmente
  - Composição automática de processos de teste
  - Descrição completa de estados (instanciações de processos, análise de sincronizações)

- Verificação de especificações LOTOS com diversas otimizações:
  - Computação completa de análise de resposta de teste (deve, pode ou recusado) de acordo com a equivalência de teste
  - Redução da ação interna de acordo com a equivalência de teste)

- Teste e execução ramdômica de especificações LOTOS
  - Análise de respostas dos testes em execuções ramdômicas

- Transformações de especificações LOTOS em
  - FSM (Expansão para uma máquina de estados finitos)
  - EFSM (Expansão com tratamento simbólico de valores)
  - Árvore de comportamento clara (Sem detecção de estados duplicados)

- Avaliação de expressões com valores de dados

- Duas interfaces disponíveis para o usuário:
  - Interface textual
  - Interface gráfica baseada em X-Windows

É executada nas seguintes plataformas:

- Sun-3, Sun-4 (sparc) com SunOs 4.0.3 ou Solaris 2.3.
- IBM-PC ou compatível (386 ou acima) com DOS.
- HP-9000 com HP-UX

A.12. LOTEST (LOTOS Test case generation tool)

Aceita LOTOS completa em estações Sun. Gera máquina de estados finitos. Formado pelas seguintes ferramentas interativas:

- cgtool
- dfgtool
- edittc

A.13. LOTOS Simulator in OBJ

Suporta LOTOS básica. O simulador é uma implementação das semânticas dinâmicas da linguagem LOTOS. Semânticas estáticas são obtidas definindo estruturas modulares hierárquicas em OBJ.

A.14. LOTTE (LOTOS Tools Environment)

Suporta LOTOS completa em ambientes UNIX. Contém verificador sintático e semântico, gerador de relatório e simulador (Hippo do ambiente integrado Sedos).

A.15. PIL (Pre-Implementation of LOTOS)

Para estações Sun, implementado em C, suporta somente LOTOS básica. Tem os seguintes componentes:

- analisador sintático e semânticas estáticas

- tradutor URL (unique reference LOTOS)
- tradutor PRIMOL
- transformador PRIMOL

A.16. SEDOS

É um ambiente integrado constituído de editor, compilador, interpretador e simulador LOTOS (parte dinâmica e estática). Foi implementado em C para ambientes UNIX, envolvendo onze organizações em seis países, composto das seguintes ferramentas:

a) MELO (mentor LOTOS Editor)

Editor para sintaxe LOTOS

b) SEAL (Structure Editor Adapted to LOTOS)

Editor para sintaxe LOTOS, com mais recursos e mais fácil de utilizar que o editor MELO

c) SCLOTOS (Syntax Checker for LOTOS)

Analisador sintático para LOTOS, baseado no LEX e YACC.

d) LASTB (LOTOS Abstract Syntax Tree Builder)

Produz a árvore sintática de uma especificação a partir da saída de SCLOTOS

e) LISA (LOTOS Integrated Static Analiser)

Analisa a semântica a partir da saída LASTB

f) PPLOTOS (Pretty Printer for LOTOS)

Mostra as árvores sintáticas; a saída gerada é um texto LOTOS.

g) LXREF (LOTOS Cross Reference Generator)

Produz uma referência cruzada de identificadores usando a saída de LASTB.

h) HIPPO

É uma ferramenta interativa que simula, por execução simbólica, as especificações LOTOS. A partir da saída de LISA, simula o comportamento da especificação, incluindo o cálculo das expressões valoradas. Suporta tanto os tipos abstratos de dados como a simulação de partes de processos de especificações LOTOS. BELASI (Behavioral Language Simulator) e STAR (Adapted CSP Simulator) foram substituídos pelo Hippo.

i) LIW (LOTOS Implementation Workbench)

É um ambiente que suporta funções tais como transformações de fonte para fonte com tradução de uma especificação de alto nível em uma especificação mais aproximada da máquina, bem como compilação de ordem temporal, que manipula processos em co-rotinas como códigos C que interagem com um kernel, ou sistema de suporte run-time, para implementar a sincronização semântica do LOTOS.

j) LOTOSTOOL

Sistema de gerência por janelas para supervisionar a execução dos componentes que compõem a ferramenta para reduzir a complexidade e facilitar a interpretação.

A.17. SMILE

Suporta LOTOS completa. É um simulador em que qualquer evento pode ser analisado sem instanciar as variáveis associadas ao evento.

A.18. SPIDER (Service and Protocol Interative Development Environment)

É um simulador gráfico para LOTOS textual ou gráfico.

A.19. Squiggles

Verifica equivalências forte fraca e de teste entre especificações básicas LOTOS ou sistemas de transições rotuladas finitas (FLTS) e gera os componentes conectados fortemente de um FLTS. Escrito em C, para estações Sun. Suporta LOTOS Básica.

A.20. TETRA (Test and Trace Analysis tool)

Pode comparar automaticamente as especificações de um teste de conformidade com a especificação do protocolo.

A.21. TIMED LOTOS Interpreter

Suporta LOTOS básica em estações Sun. Tem como finalidade simular o comportamento de sistemas de tempo real.

A.22. TOPO

TOPO consiste de um conjunto de ferramentas LOTOS agrupadas em diversos pacotes. TOPO fornece suporte para a verificação de sintaxe e semântica, e tradução de especificações LOTOS em linguagens de programação.

Possui dois componentes concretos: *toposet* e *topo.* Há alguns arquivos extras que são necessários em tempo de compilação e outros necessários para gerar interface C. Ao instalar TOPO, as seguintes variáveis de ambiente são configuradas:

Variável	default	Usada por
TOPO	usuário	raiz para ferramenta topo
TOPOBIN	$TOPO /bin	binário a ser executado
TOPOINC	$TOPO /lib	arquivos a serem incluídos
TOPOLIB	$TOPO /lib	bibliotecas a serem lidas
TOPOSTDLIB	$TOPO /stdlib	bibliotecas LOTOS a serem lidas

Utilização das ferramentas TOPO

É necessário que o usuário siga alguns passos sempre que abrir uma seção para trabalhar com as ferramentas **TOPO**:

setenv topo: Inicializa o ambiente de trabalho definindo a variável de ambiente *topo*.

toposet *spec[.lot] [options]*: Estabelece o ambiente de compilação para *spec[.lot]*

toposet deve ser utilizado antes de *topo*. Esta opção gera um arquivo de contexto denominado *spec.ctx* que é utilizado posteriormente pelas ferramentas TOPO.

Se toposet for utilizado da maneira mais simples (**toposet** *spec[.lot]*), o arquivo de contexto *spec.ctx* conterá os valores padrões:

Library

Language C

BehaviourName lbc

Dataname ldc

BehaviourPieces l

DataPieces l

GladLibrary

Estes valores podem ser modificados:

Obs: Os nomes a seguir, que estão entre parênteses correspondem aos valores *default* mostrados acima e contidos em *spec.ctx*.

toposet *spec[.lot] [options]*

-list:	Lista contexto corrente
-tname NOME:	Nome principal para arquivos C (BehaviourName lbc)
-tpieces #:	Número de partes a quebrar o arquivo C (BehaviourPieces 1)
-dname NOME:	Nome principal para arquivos C (Dataname ldc)
-dpieces #:	Número de partes a quebrar o arquivo C (DataPieces1).
-language ADA	Define que a linguagem a ser gerada é a linguagem ADA (Language C).

Após esta etapa de inicialização, pode-se executar Comportamentolmente as ferramentas TOPO através da única interface, que é denominada **topo**. As principais opções para compilação são descritas a seguir:

topo *spec.[.lot] [actions]*

-help	Mostra ajuda on-line.
-syntax	Verifica a sintaxe
-s	Verifica a semântica e constrói as bibliotecas de tipos de dados
-clean	Elimina arquivos intermediários
-tidy	Todos os arquivos gerados são removidos
-verbose	Mostra na tela a atividade de compilação (arquivos gerados, etc.)

Para a geração de código C de TOPO, utiliza-se a mesma sintaxe apresentada até agora, com algumas opções adicionais.

-behaviour: Executa o compilador de comportamento.
Gera o arquivo *spec.lbm* e arquivos fontes em C:
spec.c e spec.hh (de acordo com a opção -tname *NOME* definida em **toposet**)
-g adiciona informação simbólica para depuração

-data: Executa o compilador ADT (Tipos Abstratos de Dados).
Gera os arquivos *spec.agf, spec.asf, spec.idl, spec.lfe, spec.lsf* e fontes em C: spec.c e spec.hh (de acordo com a opção -dname *NOME* definida em **toposet**)

-p: Gera um protótipo da especificação. Ele chama o compilador de dados e de comportamento e fornece uma função principal.

Caso não seja definida nenhuma opção, o compilador assume a opção -s, que é *default*.
A arquivos gerados por esta compilação são: *spec.as*, *spec.fle* e *spec.ls*

A.23. UO-GLOTOS (The graphical LOTOS project)

Suporta LOTOS completa. Escrito em C para estações Sun. Possui cinco componentes:
- editor
- tradutor textual e tradutor gráfico
- gerador de teste
- executor em quatro modos:
    - passo simples
    - múltiplos passos
    - interrupção pelo ambiente
    - interrupção indeterminada

A.24. UO-LOTOS-Toolkit

Suporta LOTOS completa. Escrito em C para estações Sun. Verifica semânticas estáticas e executa a especificação para produzir as próximas ações possíveis. Tem dois modos de operação:
- Interpretador passo-a-passo
- Gerador da árvore simbólica.

Anexo B. Aspectos de projeto e desenvolvimento de sistemas distribuídos

É necessária uma metodologia para alcançar objetivos tais como: facilidade de decomposição, desenvolvimento simultâneo de projeto por diversas equipes, facilidade de mapeamento nos construtores de implementação predefinidos, etc.

São os princípios de projeto e os estilos de especificação que garantem a homogeneidade e a consistência das especificações quando realizadas por uma equipe de projetistas. Isto garante aos projetistas uma melhor trajetória do projeto e projetos de qualidade em um tempo menor. Possibilita também que o trabalho seja dividido em módulos e unificado posteriormente.

B.1. Princípios de projeto

Existem alguns princípios básicos que devem ser considerados no desenvolvimento de um sistema complexo. Estes princípios guiam o especificador, permitindo-lhe obter maior benefício das técnicas formais, e produzindo melhores especificações. Tais princípios são os de ortogonalidade, generalidade e flexibilidade [ViSc89].

B.1.1. Ortogonalidade

Este princípio atua como critério para avaliar estilos de acordo com a capacidade de reconhecer e enfatizar aspectos funcionais independentes que compõem um sistema.

A ortogonalidade leva em conta a separação de processos independentes, ou seja, requisições independentes devem ser especificadas por definições independentes, garantindo as particularidades de cada construção. Este princípio possui forte relação com os conceitos de AAO (Análise Orientada a Objetos), onde se procura agrupar objetos com características semelhantes. Isso facilita a legibilidade das especificações desenvolvidas.

B.1.2. Flexibilidade

Este princípio de projeto propõe o desenvolvimento e uso de construções que sejam fáceis de estender e modificar a sua funcionalidade, ou seja, que as construções possam ser facilmente alteradas, com um mínimo de esforço.

As arquiteturas deverão sofrer manutenção mais cedo ou mais tarde. Não somente para reparos de erros, mas em muitos casos também para estender a funcionalidade, ou para substituir algumas funções existentes por outras. No caso de arquiteturas complexas, o custo de manutenção supera muitas vezes o custo de novos projetos. Além disso, para desenvolver uma nova arquitetura, que tem uma razoável quantidade de aspectos comuns com outras existentes, é vantajoso iniciar o projeto por modificações do que já existe, partindo de soluções de projeto conhecidas, ao invés de iniciar do nada.

B.1.3. Generalidade

Quanto mais genéricas forem as especificações desenvolvidas, mais fácil serão as reutilizações dessas especificações. É interessante que se organize uma biblioteca com as construções reutilizáveis, onde ficam disponíveis de maneira organizada.

De acordo com este princípio, o especificador é educado de modo a reconhecer "a essência do problema" sob as diversas formas na qual o mesmo aparece. Como conseqüência direta, o especificador deve reutilizar soluções existentes, para com isso economizar tempo e esforço. As especificações devem ser construídas de tal modo que possam ser facilmente modificadas, de acordo com a necessidade.

B.2. Estilos de Especificação

São os estilos de especificação que permitem preservar a homogeneidade de especificações desenvolvidas por diversos especialistas. Cada projetista em particular possui um estilo próprio, um toque especial. Isso pode resultar em várias expressões diferentes para conceitos semelhantes.

Em cada estágio no desenvolvimento de um sistema complexo, pode-se refinar a estrutura desenvolvida no estágio anterior, permitindo que a mesma seja detalhada, passo-a-passo, até que possa ser utilizada como base para implementação. Diferentes estilos de especificação podem ser adotados em cada etapa do projeto.

Os estilos podem ser divididos em dois grupos: estilos que são voltados para a representação de aspectos externos do sistema e estilos voltados para a representação de aspectos internos do sistema [ViSc89].

B.2.1. Estilos voltados para a representação de aspectos externos

É a abordagem mais adequada para a representação de um sistema no nível mais alto de abstração. Constitui o "quê" do sistema. A descrição externa admite os estilos monolítico e orientado para restrições

B.2.1.1. Estilo monolítico

No estilo monolítico é apresentado somente o comportamento observável do sistema. A relação de ordem no tempo é definida como uma coleção de seqüências de interações.

É mais utilizado quando um sistema pode ser compreendido como uma caixa preta, portanto, é mais aplicável a especificação de serviço do que a especificação de protocolo. Ele tem pouca utilização em sistemas distribuídos mais complexos porque, devido à sua carência de estrutura, a compreensão das especificações é dificultada.

Os princípios de ortogonalidade e flexibilidade são violados por esse estilo, e a generalidade pode ser suportada numa extensão limitada.

A Figura B.1 representa um serviço de pergunta/resposta que é também utilizado para ilustrar os estilos orientado para restrições e orientado para estados.

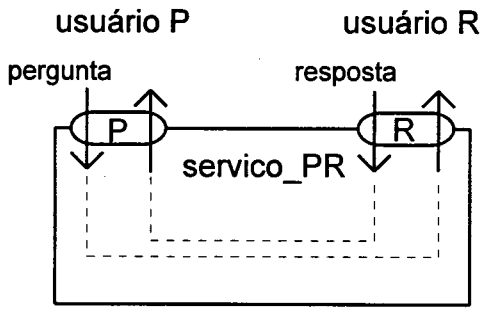


Figura B.1 - Serviço de pergunta/resposta.

De acordo com o estilo monolítico, a especificação é apresentada da seguinte forma.

```
process servico_PR_monolitico[P, R] :noexit :=
  P?p:pergunta; R!p; R?r:resposta; P!r; stop
endproc
```

B.2.1.2. Estilo orientado para restrições

No estilo orientado para restrições, somente as interações observáveis estão presentes, mas seu relacionamento de ordem no tempo é obtido como uma combinação de restrições separadas. Aspectos locais (restrições locais) são separados de aspectos fim-a-fim (restrições fim-a-fim).

A especificação a seguir representa o serviço pergunta/resposta (apresentado anteriormente na Figura 3.1) de acordo com o estilo orientado para restrições. Esta especificação formaliza uma decomposição do comportamento externo em comportamento local (isto é local na porta P ou local na porta R) e restrições fim-a-fim (isto é ligando a porta P à porta R). Isto significa que a restrição local no outro lado pode ser especificada por uma instanciação própria da mesma abstração do processo, denominada PR_local, suportando generalidade e consistência. Tal especificação é apresentada a seguir.

```
process servico_PR_restricoes[P,R]: noexit :=
  (PR_local[P] ||| PR_local[R]) || PR_remoto[P,R]
where
  process PR_local[X]: noexit :=
    X?x:pergunta; X?y:resposta; stop
  endproc
  process PR_remoto[X,Y]: noexit :=
    X?x:pergunta; Y!x; stop ||| Y?y:resposta; X!y; stop
  endproc
endproc
```

B.2.2. Estilo voltado para a representação de aspectos internos

É a abordagem mais aplicada para a representação dos aspectos internos de um sistema, caracterizando o resultado dos refinamentos. Constitui o "como" do sistema.

B.2.2.1. Estilo orientado para estados

O estilo orientado para estados apresenta somente o comportamento observável de um sistema. É particularmente utilizado na fase final da trajetória do projeto, quando o recurso pode ser definido como um objeto elementar que pode ser implementado em um único passo de implementação.

A especificação do serviço pergunta/resposta apresentada abaixo no estilo orientado para estados utiliza cinco valores de dados para representar os cinco estados globais. Tais estados são `esperaP`, `pendenteP`, `esperaR`, `pendenteR` e `executado`.

```
process servico_PR_estados[P,R]: noexit :=
  choice p:pergunta; r:resposta [] servico_PR_estados1[P,R] (esperaP,
p, r)
where
  process servico_PR_estados1[X,Y]:(s:estado, x:pergunta, y:resposta)
:noexit
    [s = esperaP] -> X?x1: pergunta;
      servico_PR_estados1[X,Y] (pendenteP, x1, y)
  [] [s = pendenteP] -> Y!x;
      servico_PR_estados1[X,Y] (esperaR, x, y)
  [] [s = esperaR] -> X?y1: resposta;
      servico_PR_estados1[X,Y] (pendenteR, x, y1)
  [] [s = pendenteR] -> X!y;
      servico_PR_estados1[X,Y] (executado, x, y)
  [] [s = executado] -> stop
  endproc
endproc
```

B.2.2.2. Estilo orientado para recursos

No estilo orientado para recursos tanto os eventos internalizados pelo operador "`hide... in...`" quanto os eventos externos (eventos observáveis) estão presentes. Cada recurso é definido por uma ordem de ações no tempo, internas e externas, ou somente externas. Assim como o estilo orientado para restrições, ele induz a uma grande utilização dos operadores de composição paralela. Utiliza o operador de ocultação "`hide... in...`" para tornar as interações dos componentes invisíveis ao observador externo.

A especificação de acordo com o estilo orientado para recursos assume que o serviço não pode ser implementado diretamente. Portanto recursos locais na forma de entidades de protocolo precisam ser identificados. Eles podem ser implementados separadamente e utilizam um serviço de mais baixo nível para fornecer o servico_PR_recursos. Veja a Figura B.2.

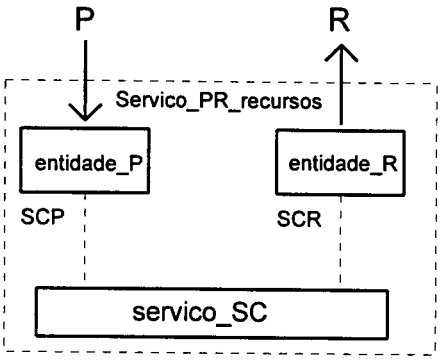


Figura B.2 - Serviço pergunta/resposta de acordo com o estilo orientado para recursos.

```
process servico_PR_recursos[P,R]: noexit
  hide SCP, SCR in
  (entidade_P[P,SCP] ||| entidade_R[R,SCR])
  |[SCP, SCR]|
  servico_SC[SCP, SCR]
where
  (* definições dos processos entidade_P, entidade_R e servico_SC *)
endproc
```

B.2.2.3. Estilo orientado para objetos

O conceito de orientação para objetos está sendo muito explorado na maioria das linguagens disponíveis atualmente e, cada vez mais, vem ganhando importância nos projetos de sistemas distribuídos.

Atualmente a maioria dos novos desenvolvimentos no campo dos sistemas operacionais distribuídos são orientados para objetos [Mayr89]. Isso faz com que haja necessidade de

especificar sistemas formalmente seguindo uma análise orientada para objetos e um estilo orientado para objetos [MoCl93].

Na parte de dados, LOTOS pode integrar subtipos para estender e reutilizar especificações. Essa interpretação de subtipos é importante no projeto de sistemas orientados para objetos.

Método de análise orientada para objetos

O principal objetivo de um método de análise orientado para objetos é compreender e identificar os objetos e classes que constituem um sistema, compreender o comportamento dinâmico de cada objeto e sua estrutura e, ao mesmo tempo, mostrar como todos os objetos de um sistema interagem estática e dinamicamente.

Na análise orientada para objetos deve-se seguir os seguintes passos:

- Compreender os requisitos do usuário;
- identificar e classificar os objetos;
- definir os objetos;
- identificar as associações ou relações entre os objetos;
- construir a documentação.

Um objeto é definido em termos de suas visões estática e dinâmica. A visão estática é dada por uma lista dos seus atributos e operações. A visão dinâmica é, algumas vezes, descrita por diagramas de transição de estados.

As visões estáticas são representadas pelo nome e cardinalidade. As visões dinâmicas constituem as mensagens de ligação e são representadas por setas que ligam o objeto que chama ao objeto que é chamado.

Um dos métodos de análise orientada para objetos é o método ROOA (Análise Orientada para Objetos Rigorosa [MoCl93]) que envolve três tarefas principais:

- *Construção do modelo de objetos*: esta tarefa pode ser considerada completamente separada das tarefas seguintes, podendo ser realizada por uma equipe diferente.

- *Refinamentos do modelo de objetos*: é onde se inicia o processo de normalização, garantindo a hierarquização e ação dos objetos, de modo que o sistema fique mais simples de compreender e de desenvolver.

- *Construção do modelo formal LOTOS*: esta etapa segue os seguintes passos:

1. Criação do diagrama de comunicação dos objetos

Este diagrama é um grafo onde, na primeira iteração, um nó representa um objeto e cada arco que liga dois objetos representa uma porta (*gate*) de comunicação entre eles.

2. Especificação das classes e objetos

Em geral, o comportamento de um objeto é especificado como um processo e seu estado como um ou mais tipos abstratos de dados (Abstract Data Types -ADTs). Para cada objeto deve-se:
-Especificar o processo identificando os eventos dos quais ele toma parte e a ordem em que ocorrem.
-Especificar ADTs simbólicos para descrever os seus atributos.

3. Composição dos objetos em expressões de comportamento LOTOS

Compõem-se os processos em expressões LOTOS por meio dos operadores de paralelismo, seguindo a estrutura do diagrama de comunicação dos objetos.

4. Prototipação da especificação

É utilizada para verificar os serviços e as mensagens de ligação.

5. Refinamento da especificação

Ao longo de sucessivos refinamentos novos grupos de objetos são identificados. São adicionadas informações novas nos processos existentes e também processos novos são criados, bem como novas operações com estes processos.

Exemplo de uma especificação orientada para objetos:

Considere um objeto impressora que modela um dispositivo físico impressora. A impressora I oferece as operações `iniciar`, `imprimir` e `finalizar` na porta `imp`, ilustrada abaixo. Veja a figura B.3.

Figura B.3 - Objeto impressora.

Vários objetos podem ter acesso simultâneo à interface `imp`. Assim, necessita-se diferenciá-los pela `id_origem`, necessária para que a impressora não misture as linhas oriundas de diferentes objetos. Deve também ser considerada a possibilidade de faltar papel na impressora. O objeto do tipo "impressora" oferece somente as operações que são permitidas em um certo estado.

```
[ I <- iniciar     (Id_origem)]
[ I <- imprimir    (Id_origem,linha)]
[ I <- finalizar (Id_origem)]
```

O comportamento da impressora pode ser especificado da seguinte maneira:

```
type Imp_op is
  sorts imp_op
  opns iniciar, imprimir, finalizar: ->imp_op
endtype
:
process impressora[imp] (e:estado) :noexit:=
  imp !iniciar ?ident:id_origem;
  impressora1[imp](ident,e)
  where
  process impressora1[imp](ident:id_origem,e:estado):noexit:=
    [e==pronta]->imp!imprimir!ident?l:linha;impressora[imp](ident,s)
[]  imp!finalizar!ident;impressora[imp] (e)
[]  [e==sem_papel]->i;impressora1[imp](ident,pronta)
[]  [e==pronta]->i;impressora1[imp](ident,sem_papel)
  endproc
endproc
```

A impressora possui 2 estados (e), que indicam se ela está pronta (`pronta`) ou sem papel (`sem_papel`). Ela continua imprimindo até receber a indicação de finalizar (`finalizar`) ou estiver sem papel (`sem_papel`).

B.3. Abordagens de projeto

No projeto e desenvolvimento de sistemas, podem ser adotadas diferentes abordagens de projeto. As abordagens consideradas mais importantes são as seguintes: a baseada em refinamentos sucessivos, a baseada em prototipação e a baseada em construções predefinidas.

B.3.1. Refinamentos Sucessivos

É a abordagem clássica. É mais utilizada para especificações complexas e de grande porte. Inicia-se com uma visão abstrata do sistema e através dos sucessivos refinamentos acrescentam-

se os detalhes, usando sempre as provas de equivalência entre duas versões sucessivas, até chegar à especificação que será utilizada para implementação.

Esta abordagem pode ser beneficiada com a introdução de estilos de especificação adequados às diferentes fases de projeto, no intuito de tornar as especificações mais legíveis e fáceis de manipular [ViSc88].

B.3.2. Prototipação

Já estão sendo utilizadas ferramentas para a geração de protótipos de implementações de protocolos de comunicação e especificações de sistemas concorrentes usando a TDF LOTOS. As especificações LOTOS podem ser analisadas e traduzidas para funções C, as quais são executadas no ambiente UNIX. Em alguns casos os processos LOTOS são primeiro traduzidos para máquinas de estados finitos estendidadas (EFSMs) [VaCi93].

B.3.3. Construções predefinidas

Esta abordagem consiste na exploração de construções predefinidas armazenadas em uma biblioteca. Considere, por exemplo, sistemas onde um processo habilita outro, como no exemplo abaixo, onde a `escrita` é habilitada pela `leitura`.

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

Os processos `leitura[in]` e `escrita[out]` são muito simples e podem ser utilizados como construções predefinidas.

Considere agora sistemas que realizem seqüenciamento de eventos infinitamente. Eles têm características de construções predefinidas. Veja as Figuras B.4 e B.5.

```
process ciclico_2[ a,b] :noexit:=
  a; b; ciclico_2[ a,b]
endproc
```

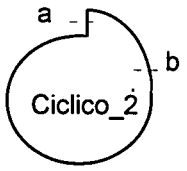


Figura B.4 - Processo ciclico_2.

```
process ciclico_3[a,b,c] :noexit:=
  a; b; c; ciclico_3[ a, b, c]
endproc
```

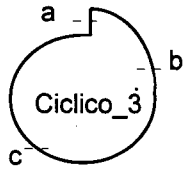


Figura B.5 - Processo ciclico_3.

Considere os sistemas com comportamento infinito em que há uma escolha indeterminística entre n opções de eventos.

No exemplo a seguir, após a ocorrência do evento `a1` há uma escolha indeterminística com duas alternativas. Cada alternativa oferece um retorno recursivo. Veja a Figura B.6.

```
process disjuntor[ a1, a2, a3] :noexit :=
  a1; (a2; disjuntor[ a1, a2, a3]
       []
       a3; disjuntor[ a1, a2, a3])
endproc
```

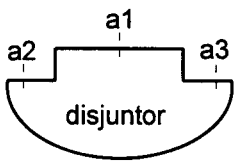


Figura B.6 - Processo disjuntor.

Existem ainda os sistemas em que a recursividade não pode ser usada em todas as opções. Neste exemplo, o primeiro somatório corresponde às opções que terminam em inatividade, enquanto que o segundo somatório corresponde às opções com recursividade. Veja a Figura B.7.

```
process opcoes_mistas[a1,...,an]:noexit :=
  m
  Σ ( ai; stop)
  i=1
  []
  n
  Σ ( ai; opcoes_mistas[a1,...,an])
  i=m+1
endproc
```

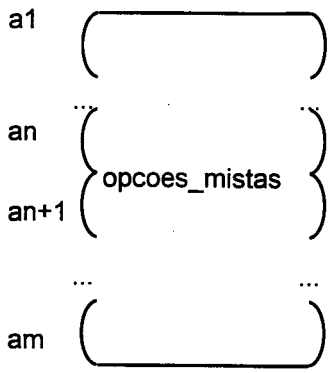


Figura B.7 - Processo opcoes_mistas.

As situações mais complexas envolvem escolhas com seqüências de ações (repetidas ou não) onde pode haver a intercalação de eventos observáveis com eventos não-observáveis (i). Cada escolha pode desdobrar-se em várias novas opções.

Pode-se fazer a transformação das implementações de referência numa composição de construções predefinidas (PDICs - Predefined Implementation Constructs). As PDICs possibilitam construções parametrizadas, tendo a reusabilidade dos componentes favorecida. A correção de uma implementação final depende exclusivamente da correção da composição de PDICs.

Métodos automatizados para transformar especificações genéricas numa composição de PDICs ainda não são disponíveis, sendo uma área interessante para pesquisa [ScPi92] [PiSi92].

Anexo C. Métodos para validações de sistemas

O uso de TDFs possibilita demonstrar a correção de uma especificação. O termo validação pode ser utilizado para descrever as atividades de demonstração de correção de uma especificação ou implementação [ErFr91]. As principais técnicas utilizadas (testes, simulações, verificações) são apressentadas a seguir.

C.1. Teste

Os testes podem ser realizados em todas as fases de desenvolvimento de um sistema, desde em especificações de alto nível até em versões de implementação. A desvantagem desse método de validação é que os testes não são exaustivos, ou seja, são usados para encontrar erros e não para demonstrar correção.

Algumas ferramentas podem ser utilizadas para teste de especificações, como é o caso da ferramenta *CÆSAR.open* (com a opção *exhibitor*), onde a especificação é testada através da procura por ações visíveis. Ainda com a ferramenta *CÆSAR.open* (com a opção *executor*), pode ser realizado a execução ramdômica do sistema de transições da especificação testada. Nessa execução escolhe-se o número de transições a serem executadas (e mostradas) bem como a estratégia para resolver o indeterminismo.

C.2. Simulação

Na simulação, um modelo executável é desenvolvido e observado. Se o modelo não for muito complexo, a simulação descobre a maioria dos erros. Quando, porém, o modelo torna-se complexo - a grande maioria - torna-se impossível simular todos os casos importantes.

A ferramenta *Cæsar*, por exemplo, realiza a simulação exaustiva da especificação na sétima fase de compilação (*simulation*). Essa fase explora exaustivamente todos os componentes da rede de petri gerada e produz um grafo de estados.

A ferramenta *CÆSAR.open* (com a opção *simulator*), permite a simulação interativa de uma especificação LOTOS, oferecendo ao usuário as opções de estados possíveis a cada transição.

C.3. Verificação

A verificação é uma prova formal de que uma especificação ou implementação satisfaz propriedades desejáveis, usando métodos matemáticos rigorosos. A análise de especificações permite várias verificações. Por exemplo, comparar uma especificação original com outra mais refinada.

Dois tipos de verificação podem ser destacados:

- **Verificação por redução**: os grafos gerados pelos compiladores são, em sua grande maioria, muito complexos para serem verificados. Então, aplicam-se critérios de abstração, reduzindo os grafos de acordo com relações de equivalência, tais como, equivalência forte e equivalência de observação.

- **Verificação por comparação**: uma especificação pode ser verificada através da comparação com outra especificação sob critérios, tais como, equivalência forte e equivalência de observação. Desse modo pode-se verificar, por exemplo, se um determinado protocolo corresponde a um determinado serviço.

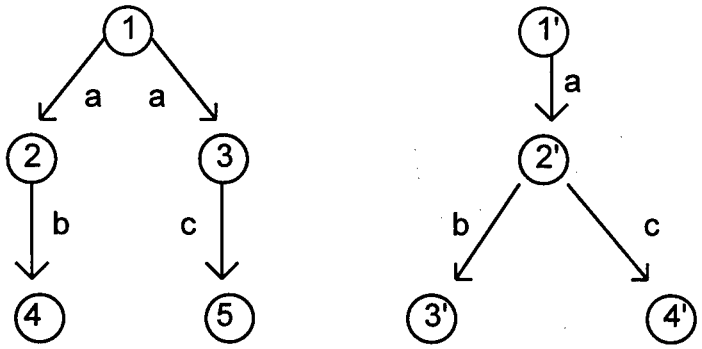
As relações equivalência forte, equivalência de observação e equivalência segura [BoFe90] [Fern90] [Fega93] [ErFr91] são descritas a seguir.

C.3.1. Equivalência forte (*strong bisimulation*)

A equivalência forte (strong bisimulation) entre dois sistemas de transições pode ser provada se for mostrada a existência de uma relação de bissimulação entre os dois conjuntos de estados

desses dois sistemas. Tal relação é obtida se observadas todas as transições, incluindo as transições internas com igual importância.

A equivalência forte requer que cada evento em um sistema de transições corresponda a exatamente um evento igual no outro sistema de transições. Um exemplo é demonstrado na Figura C.1, onde dois sistemas com as mesmas seqüências de ações não são equivalentes.



Comportamento do sistema A Comportamento do sistema A'

Figura C.1 - Não há equivalência forte entre os sistemas A e A'.

Dois estados p e q são *bisimilars* se para cada estado p' alcançável de p pela execução de uma ação a existe um estado q', alcançável de q pela execução de uma ação a tal que p' e q' são *bisimilars*.

Seja G1 e G2 dois sistemas de transições rotuladas com conjuntos de estados *S1* e *S2*, respectivamente, e seja $S = S1 \cup S2$. Uma relação de equivalência forte sobre *G1* e *G2* é uma relação $R \subseteq S \times S$ tal que $\langle p,q \rangle \in R$ implica

1. Se $p \rightarrow^a p'$ então $\exists\, q' : q \rightarrow^a q'$ e $\langle p',q' \rangle \in R$, e
2. Se $q \rightarrow^a q'$ então $\exists\, p' : p \rightarrow^a p'$ e $\langle p',q' \rangle \in R$.

Dois sistemas de transições rotuladas *G1* e *G2* são equivalentes se existe uma relação de equivalência forte relacionando os estados iniciais dos dois sistemas.

A equivalência forte é, geralmente, muito forte para os propósitos de verificação. Abaixo, são descritas duas relações de equivalências usadas na verificação.

C.3.2. Equivalência fraca (*weak bisimulation*)

Na equivalência fraca (*weak bisimulation*), também chamada de equivalência de observação, um evento em um sistema de transições não necessita estar relacionado a exatamente um evento no outro sistema de transições, pois um evento interno (τ-*event*) de um grafo pode corresponder a zero ou mais eventos internos no outro grafo e vice-versa.

Um sistema pode trocar de estado sem qualquer ação observável [Miln 80]. Na figura C.2, mostra-se o comportamento de um sistema *A"* que do estado *1"* passa ao estado *1a"* após ter sido executada a ação *a*. Em seguida, espontaneamente, o sistema passa para o estado *2"* ou para o estado *3"*. Prosseguindo, o sistema pode executar a ação *b* e a ação *c*. O comportamento do sistema *A"* é equivalente quanto a equivalência de observação ao comportamento do sistema *A*.

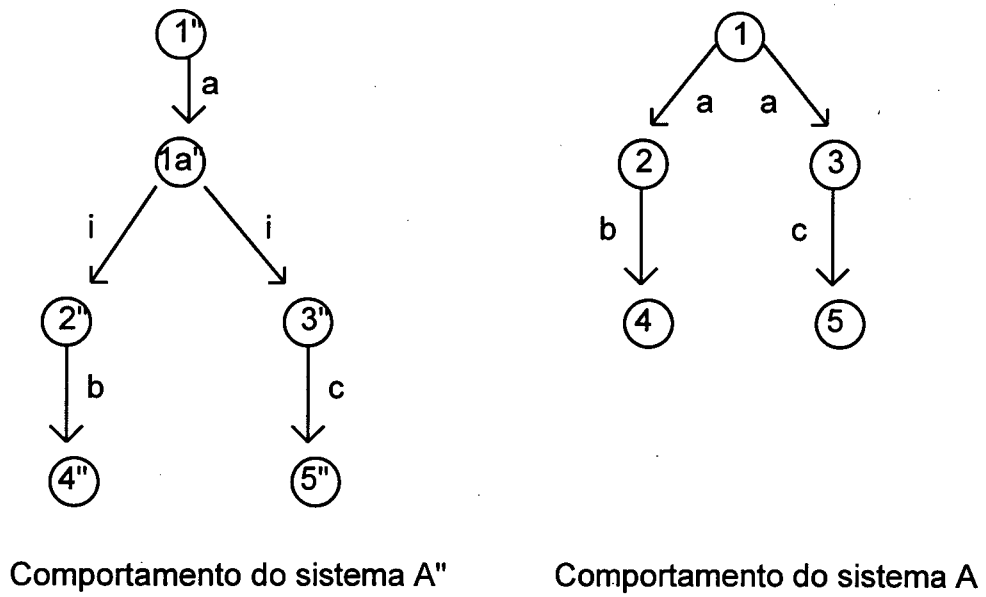


Figura C.2 - Equivalência de observação entre os sistemas A e A'.

Dados os estados p e p', escreve-se $p \Rightarrow^{\hat{a}} p'$ se p para executar um evento a precedido ou sucedido por qualquer número finito (inclusive zero) eventos internos (τ-*events*) e chegar a um estado p'. Escreve-se $p \Rightarrow^{\tau} p'$ se p para realizar zero ou mais eventos internos e chegar ao estado p'.

Seja G1 e G2 dois sistemas de transições rotuladas com conjuntos de estados *S1* e *S2*, respectivamente, e seja $S = S1 \cup S2$. Uma relação de equivalência fraca sobre *G1* e *G2* é uma relação $R \subseteq S \times S$ tal que $<p,q> \in R$ implica

1. Se $p \rightarrow^{a} p'$ então $\exists\, q' : q \rightarrow^{\hat{a}} q'$ e $<p',q'> \in R$, e
2. Se $q \rightarrow^{a} q'$ então $\exists\, p' : p \rightarrow^{\hat{a}} p'$ e $<p',q'> \in R$.

Dois grafos *G1* e *G2* são equivalentes quanto à equivalência de observação se existir uma relação de equivalência fraca relacionando os estados iniciais dos dois grafos.

Se para chegar a um determinado estado em um sistema de transição, foi necessária uma ação *a*, então para haver equivalência forte com outro sistema de transição, a mesma ação *a* é requerida. Já na equivalência fraca, podem ocorrer eventos internos antes e após a ação *a*, e os sistemas de transições são considerados equivalentes, pois os eventos internos, nesse caso, são invisíveis.

C.3.3. Equivalência segura (*safety equivalence*)

A equivalência segura (*safety equivalence*), altera todas as τ-*transitions* (transições internas e escondidas) por ε-*transitions* (transições espontâneas e não visíveis). Geralmente, tal equivalência produz grafos muito menores do que os produzidos pela equivalência forte. Por isso, é usada principalmente em grandes programas. Essa equivalência é útil para assegurar que alguma situação indesejável nunca acontece, como por exemplo, situações de impasse (*deadlocks*) [BoFe90].

A equivalência segura é mais fraca que as outras duas (equivalência forte e equivalência de observação). Contudo, ela é mais forte que a equivalência de traço (*trace equivalence*), onde dois grafos são equivalentes se podem executar as mesmas seqüências de eventos. Essa suposição não é provada formalmente, mas aparece em um grande número de exemplos [Gara 94a].

Anexo D. Código da ferramenta TranSP

A ferramenta Transp pode ser instalada em qualquer microcomputador que tenha o Windows. Basta criar um diretório e copiar os arquivos para dentro dele. A estrutura principal do software é apresentada na Figura D.1.

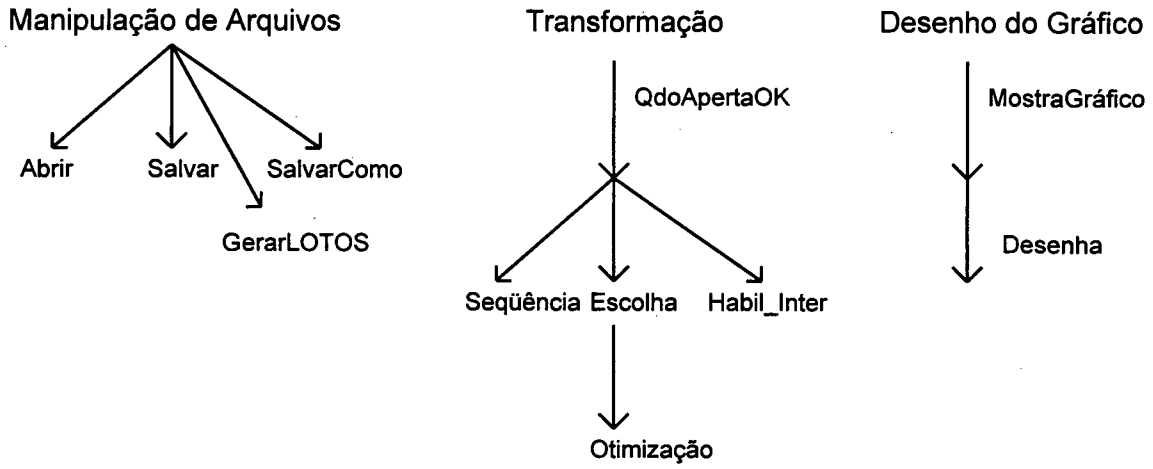


Figura D.1 - Estrutura principal da ferramenta TranSP.

Com relação à transformação, quando é pressionado o botão [S->P] na ferramenta TranSP, é executada a função **QdoApertaOK** que, a partir da expressão contida na janela **Serv**, executa as rotinas **Seqüência**, **Escolha** e **Habil_Inter** (Trata os casos de habilitação e de Interrupção), de acordo com a necessidade. No caso da expressão de serviço conter escolhas, pode ser realizada posteriormente, a **Otimização** da especificação (no caso do botão [Optimize] estar pressionado). Ao pressionar o botão [Graphic] na ferramenta, são executadas as rotinas **MostraGráfico** (prepara a janela para o desenho) e **Desenha**.

A parte de manipulação de arquivos da ferramenta TranSP compreende funções comuns, como **Salvar**, **Abrir**, **SalvarComo** e **GerarLOTOS**. Estas funções foram apresentadas no capítulo 6.

A seguir, é apresentado o código do arquivo principal da ferramenta TranSP.

```
#include <owl.h>
#include <button.h>
```

```
#include <radiobut.h>
#include <groupbox.h>
#include <editwnd.h>
#include <fstream.h>
#include <filewnd.h>
#include <string.h>
#include "transp.h"
#include <bwcc.h>

const WORD TAM_MAX_TEXTO = 1000;
char *allmtrim( char *);
char *alltrim( char *);
char *ltrim( char *);
char *rtrim( char *);
char *strtran(char *string, char *parte1, char *parte2);
int Hab1Int2;

BOOL Lotos = TRUE, Estilo = TRUE, Otimizar = TRUE;
char FileName[512]; //Nome do arquivo
int VemEscolha=0; // controla no sequenciamento se a expressao veio
                  //de uma escolha, devendo cuidar a sincronização
int ContSinc=0; //controla o número da sincronização (s1, s2, s3, ...s9)
int UltEscolha;  //UltEscolha controla qdo expressao for .... [] ....
                 //não colocando parenteses externos
int IsInterrupt=0; //Controla quando é interrupção por causa da sincroniz. s1,s2,...sn
char Mensagem[50];//Mensagem de erro
char Parte[12][400];
int existe=0, acoes1, acoes2, meios=0, VAltura, VLargura,Otimizacao;
int nivelX;//Nivel de cada pedaço da expressão
char parte[400]="\0",E1[400], E2[400];
char XE1[8][400], XE2[8][400],ParteE1[8][400],ParteE2[8][400];
char *ptr="";
char acao1[12][30],acao2[12][30];
char meio[7][5],Ex[4]="E1\0",Ey[4]="E2\0";
char ExprFinalE1[1000],ExprFinalE2[1000];
char Buffer1[20],ParteProServ[400];
char RestoEsq[400],RestoDir[400];
char AcaoInt[10]; //armazena a primeira acao de cada escolhas dentre os parênteses
char PriAcaoInt[10]; // ''   ''   de acordo coo o nivel
char evsitio,TodaExpressao[TAM_MAX_TEXTO], xTodaExpressao[TAM_MAX_TEXTO];
int VariaXN,PosX,linhas,TamLinEdit,PriFechPar,UltAbrePar,i;

class TTestApp : public TApplication{
public:
    TTestApp(LPSTR AName, HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int
nCmdShow)
    : TApplication(AName, hInstance, hPrevInstance, lpCmdLine, nCmdShow) {};
  virtual void InitMainWindow();
  void InitInstance();
};

class TFerrTransp : public TDialog{
private:
  BOOL Finaliza;
public:
  TRadioButton *RTButton1, *RTButton2, *RTButton3, *RTButton4;
  TCheckBox *CheckBox;
  TGroupBox  *Lot, *Est;
  TWindow *GraphicWindow;
  TEdit *Edit,*EditE1, *EditE2 ;
  TFerrTransp(PTWindowsObject AParent);
  virtual void QuandoApertaOK(RTMessage Msg)  = [ID_FIRST + ID_BOTAORWTR];
  virtual void MostraGrafico(RTMessage Msg)  = [ID_FIRST + ID_BOTAORWGR];
  virtual void GetWindowClass( WNDCLASS& WndClass );
  virtual void Sobre(TMessage& Message) = [ CM_FIRST + IDM_INFO ];
  virtual void Abrir( ) = [ CM_FIRST + IDM_NOVO ];
  virtual void SalvarComo( ) = [ CM_FIRST + IDM_SALVAR_COMO ];
  virtual void Salvar( ) = [ CM_FIRST + IDM_SALVAR];
  virtual void GerarLOTOS( ) = [ CM_FIRST + IDM_GERAR_LOTOS];
  void desenha(HDC DC);
  void FimTransp(char * );
```

```
  void SetupWindow();
  void sequencia(char *);
  void escolha(char *);
  void habil_inter_paral(char *);
//  void paralelismD();
};

void TFerrTransp::SetupWindow()
{TDialog::SetupWindow();
RTButton1->SetCheck(BF_CHECKED);
RTButton3->SetCheck(BF_CHECKED);
ModifyMenu( GetMenu( HWindow ), IDM_GERAR_LOTOS, MF_BYCOMMAND | MF_GRAYED, IDM_GERAR_LOTOS, "Save
&Protocol" );
DrawMenuBar( HWindow );

}

TFerrTransp::TFerrTransp(PTWindowsObject AParent):
  TDialog(AParent, DIALOG_1)
{
  HDC DC;
  DC = CreateDC("DISPLAY", NULL, NULL, NULL);
  VAltura=GetDeviceCaps(DC,VERTRES);
  VLargura=GetDeviceCaps(DC,HORZRES);
  DeleteDC(DC);

  Edit   = new TEdit(this, ID_EDIT_SERV,TAM_MAX_TEXTO);
  EditE1 = new TEdit(this, ID_EDIT_E1,TAM_MAX_TEXTO);
  EditE2 = new TEdit(this, ID_EDIT_E2,TAM_MAX_TEXTO);
  RTButton1 = new TRadioButton(this, ID_RBUTTON1,NULL); //Basic LOTOS
  RTButton2 = new TRadioButton(this, ID_RBUTTON2,NULL); //Full LOTOS
  RTButton3 = new TRadioButton(this, ID_RBUTTON3,NULL); //Monolithic
  RTButton4 = new TRadioButton(this, ID_RBUTTON4,NULL); //Constraint-Oriented
  CheckBox = new TCheckBox(this, ID_CHECKBOX, NULL);
  GraphicWindow = NULL;
}

void TFerrTransp::Abrir() {
   char TodaLinha[150],TodaLinhaAux[150];
   TFileDialog* M;
   M = new TFileDialog(this, SD_FILEOPEN, strcpy(FileName, "*.SPC"));
   if (M->Execute() == IDOK){
      for (i=20;i>=0;i--){
         EditE1->DeleteLine(i);
         EditE2->DeleteLine(i);
         Edit->DeleteLine(i);
      }
      ifstream inpfile(FileName);
      Edit->DeleteLine(0);
      int flag=0;
      while (inpfile){ //verdadeiro ate final do arquivo
         strcpy(TodaLinhaAux,"\r\n");
         inpfile.getline(TodaLinha,150);
         if (inpfile){
            if (flag==0){
               flag=1;
               Edit ->Insert(TodaLinha);
            }
            else{
               strcat(TodaLinhaAux,TodaLinha);
               Edit ->Insert(TodaLinhaAux);
            }
         }
      }
   }
}

void TFerrTransp::SalvarComo()
{
   TFileDialog* M;
   M = new TFileDialog(this, SD_FILESAVE, strcpy(FileName, "*.SPC"));
```

```
   if (M->Execute() == IDOK) {
      ofstream outfile(FileName);
      char TodaLinha[160];
      linhas=Edit->GetNumLines();
      for (i=0;i<linhas;i++){ //Extrai as linhas do texto
         TamLinEdit=Edit->GetLineLength(i);
         if (TamLinEdit !=0){
            Edit->GetLine(TodaLinha,TamLinEdit,i);
            if (strlen(TodaLinha)!=0)
               if (i+1==linhas) outfile << TodaLinha; //Última linha
               else outfile << TodaLinha  << "\n";
         }
         else  outfile << "\n";
      }
   }
}

void TFerrTransp::Salvar()
{
   if (strlen(FileName)==0 || strcmp(FileName,"*.SPC")==0) SalvarComo();
   ofstream outfile(FileName);
   char TodaLinha[160];
   linhas=Edit->GetNumLines();
   for (i=0;i<linhas;i++){ //Extrai as linhas do texto
      TamLinEdit=Edit->GetLineLength(i);
      if (TamLinEdit !=0){
         Edit->GetLine(TodaLinha,TamLinEdit,i);
         if (strlen(TodaLinha)!=0)
            if (i+1==linhas) outfile << TodaLinha; //Última linha
            else outfile << TodaLinha  << "\n";
      }
      else  outfile << "\n";
   }
}

void TFerrTransp::GerarLOTOS()
{
   TFileDialog* M;
   char parametros[400], paramE1[150], paramE2[150], paramMeio[50];
   parametros[0]='\0';
   paramE1[0]='\0';
   paramE2[0]='\0';
   paramMeio[0]='\0';
   M = new TFileDialog(this, SD_FILESAVE, strcpy(FileName, "*.LOT"));
   if (M->Execute() == IDOK) {
      ofstream outfile(FileName);

      for (i=0; i<=11; i++) {
           if (acoes1 > i)  strcat(paramE1,acao1[i]);
           if (acoes1-1 >i) strcat(paramE1,",");
           if (acoes2 >i)   strcat(paramE2,acao2[i]);
           if (acoes2-1 >i) strcat(paramE2,",");
           if (meios > i)   strcat(paramMeio,meio[i]);
           if (meios-1 > i) strcat(paramMeio,",");
      }

      char *temp=paramE1;
      allmtrim(temp);   // tira os espacos de paramE1
      temp=paramE2;
      allmtrim(temp);   // tira os espacos de paramE2
      temp=paramMeio;
      allmtrim(temp);   // tira os espacos de paramMeio

      strcat(parametros,paramE1);
      strcat(parametros,",");
      strcat(parametros,paramE2);
      strcat(paramE1,",");
      strcat(paramE1,paramMeio);
      strcat(paramE2,",");
      strcat(paramE2,paramMeio);
```

```
      outfile << "specification E1E2 ["<< parametros <<"] :noexit\n";
      outfile << "   behaviour\n";
      outfile << "      hide " << paramMeio << " in (\n";
      outfile << "      E1["<<paramE1<<"]\n";
      outfile << "      |[" <<paramMeio<< "]|\n";
      outfile << "      E2["<<paramE2<<"])\n";
      outfile << "   where\n\n";
      outfile << "   process E1["<<paramE1<<"] :noexit:=\n";


      char TodaLinha[160];
      linhas=EditE1->GetNumLines();

      for (i=0;i<linhas;i++){ //Extrai as linhas do texto
         TamLinEdit=EditE1->GetLineLength(i);
         if (TamLinEdit !=0){
            EditE1->GetLine(TodaLinha,TamLinEdit,i);
            if (strlen(TodaLinha)!=0)
               if (strstr(TodaLinha,"E1")!=NULL){
                  if (strstr(TodaLinha,"E1)")!=NULL){
                      TodaLinha[TamLinEdit-1]='\0';
                      strcat(TodaLinha,"[");
                      strcat(TodaLinha,paramE1);
                      strcat(TodaLinha,"])");
                  }
                  else {strcat(TodaLinha,"[");
                        strcat(TodaLinha,paramE1);
                        strcat(TodaLinha,"]");
                  }
               }
               outfile << "      " << TodaLinha  << "\n";
         }
         else  outfile << "\n";
      }


      outfile << "   endproc\n\n";
      outfile << "   process E2["<<paramE2<<"] :noexit:=\n";
      linhas=EditE2->GetNumLines();

      for (i=0;i<linhas;i++){ //Extrai as linhas do texto
         TamLinEdit=EditE2->GetLineLength(i);
         if (TamLinEdit !=0){
            EditE2->GetLine(TodaLinha,TamLinEdit,i);
            if (strlen(TodaLinha)!=0)
               if (strstr(TodaLinha,"E2")!=NULL){
                  if (strstr(TodaLinha,"E2)")!=NULL){
                      TodaLinha[TamLinEdit-1]='\0';
                      strcat(TodaLinha,"[");
                      strcat(TodaLinha,paramE2);
                      strcat(TodaLinha,"])");
                  }
                  else {strcat(TodaLinha,"[");
                        strcat(TodaLinha,paramE2);
                        strcat(TodaLinha,"]");
                  }
               }
               outfile << "      " << TodaLinha  << "\n";
         }
         else  outfile << "\n";
      }
      outfile << "   endproc\n\n";
      outfile << "endspec";
   }
}

void TFerrTransp::QuandoApertaOK(RTMessage)
{
  if ( (RTButton1->GetCheck() == BF_CHECKED ) && (RTButton3->GetCheck() == BF_CHECKED)){
     Finaliza=FALSE; //Habilita finalização para a execução na próxima vez
     nivelX=-1;
```

```
int j=0;
acoes1=acoes2=meios=0;
for (i=0;i<=7;i++){
   acao1[i][0]='\0';
   acao2[i][0]='\0';
   meio[i][0]='\0';
   XE1[i][0]='\0';
   XE2[i][0]='\0';
}

strcpy(TodaExpressao,"\0");
strcpy(xTodaExpressao,"\0");

linhas=Edit->GetNumLines();
for (i=0;i<linhas;i++){ //Extrai as linhas do texto
   strcpy(TodaExpressao," ");
   TamLinEdit=Edit->GetLineLength(i);
   Edit->GetLine(TodaExpressao,TamLinEdit,i);
   if (strlen(alltrim(TodaExpressao)) != 0) //{
      strcat(xTodaExpressao,alltrim(TodaExpressao));
}

char *temp=xTodaExpressao;
allmtrim(temp);   // tira os espacos de xTodaExpressao
ExprFinalE1[0]='\0';
ExprFinalE1[0]='\0';

while (1){
   if (Finaliza) break;
   UltAbrePar=-1;// Procura último Parentese (
   PriFechPar=-1; // Procura primeiro Parentese )
      for (i=0; i<=strlen(xTodaExpressao);i++)
         if (xTodaExpressao[i]=='(')  UltAbrePar=i;
      for (i=UltAbrePar; i<=strlen(xTodaExpressao);i++)
         if (xTodaExpressao[i]==')')  {PriFechPar=i; i=10000;}
     //Verifica erro de parênteses: )fsd ou ou sdf)g )asdf(asf
   if ((PriFechPar==0) || (PriFechPar<UltAbrePar) || (UltAbrePar==-1 && PriFechPar !=-1)) {
      FimTransp("Error: Unbalanced Parenthesis");
      break;
   }
         //Encontrou sub-expressao: ( xxxxxxxx )
   if ((UltAbrePar != -1) && (PriFechPar != -1)) {
      //MessageBox(HWindow, xTodaExpressao, "TODA EXPRESSAO:", MB_OK);
      //Copia para ParteProServ o que esta dentro do parenteses mais interno
      j=0;
      for (i=UltAbrePar;i<PriFechPar;i++){
         ParteProServ[j]=xTodaExpressao[i+1]; //Corta o Parenteses da esquerda
         j++;
      }
      ParteProServ[j-1]='\0';    // Corta o Parenteses da direita
      //pega o que sobrou à esquerda e à direita da expressão:
      j=0;
      strncpy(RestoEsq,xTodaExpressao,UltAbrePar+1);
      RestoEsq[UltAbrePar]='\0';
      for (i=PriFechPar+1; i<=strlen(xTodaExpressao);i++){
         RestoDir[j]=xTodaExpressao[i]; j++;
      }
      RestoDir[j-1]='\0';
      strcat(RestoEsq,"X");
      strcat(RestoEsq,RestoDir);
      strcpy(xTodaExpressao,RestoEsq);

      if (strstr(ParteProServ,"[]")!=NULL){
         UltEscolha=0;
         escolha(ParteProServ); //ParteProServ = xxxx [] xxxx [] xxxx
      }

         if  ((strstr(ParteProServ,">>")!=NULL)  ||  (strstr(ParteProServ,"[>")!=NULL)  ||
(strstr(ParteProServ,"|||")!=NULL)){
         if (strstr(ParteProServ,">>")!=NULL) Hab1Int2=1;
         else { if (strstr(ParteProServ,"[>")!=NULL) Hab1Int2=2;
```

```
                    else Hab1Int2=3;} //paralelismo
                UltEscolha=0;
                habil_inter_paral(ParteProServ); //ParteProServ = xxxx >> xxxx >> xxxx
            }
        }
        else {
            if (nivelX == -1) { // Não teve nenhum ()
                strcpy(RestoEsq,xTodaExpressao);
                if (strstr(RestoEsq,"[]")!=NULL){
                    UltEscolha=1;
                    escolha(RestoEsq);     //ParteProServ = xxxx [] xxxx [] xxxx
                    strcpy(ExprFinalE1,E1);
                    strcpy(ExprFinalE2,E2);
                    break;
                }
                        if  ((strstr(RestoEsq,">>")!=NULL)  ||  (strstr(RestoEsq,"[>")!=NULL)  ||
(strstr(RestoEsq,"|||")!=NULL)){
                    if (strstr(RestoEsq,">>")!=NULL) Hab1Int2=1;
                    else { if (strstr(RestoEsq,"[>")!=NULL) Hab1Int2=2;
                         else Hab1Int2=3;} //paralelismo
                    UltEscolha=1;
                    habil_inter_paral(RestoEsq); //ParteProServ = xxxx >> xxxx >> xxxx
                    strcpy(ExprFinalE1,E1);
                    strcpy(ExprFinalE2,E2);
                    break;
                }
                else {
                    sequencia(RestoEsq);
                    strcpy(ExprFinalE1,E1);
                    strcpy(ExprFinalE2,E2);
                    break;
                }
            }
            else {
                if (strstr(RestoEsq,"[]")!=NULL){
                    UltEscolha=1;
                    escolha(RestoEsq);
                    break;
                }
                        if  ((strstr(RestoEsq,">>")!=NULL)  ||  (strstr(RestoEsq,"[>")!=NULL)  ||
(strstr(RestoEsq,"|||")!=NULL)){
                    if (strstr(RestoEsq,">>")!=NULL) Hab1Int2=1;
                    else { if (strstr(RestoEsq,"[>")!=NULL) Hab1Int2=2;
                         else Hab1Int2=3;} //paralelismo
                    UltEscolha=1;
                    habil_inter_paral(RestoEsq);
                    break;
                }
                else{
                    sequencia(RestoEsq);
                    nivelX++;
                    strcpy(XE1[nivelX],E1);
                    strcpy(XE2[nivelX],E2);
                    break;
                }
            }
        }
    }  //fim do while

for (i=20;i>=0;i--){
   EditE1->DeleteLine(i);
   EditE2->DeleteLine(i);
}

if (!Finaliza){
if (nivelX!=-1){
  strcpy(ExprFinalE1,XE1[nivelX]);  //Inicio de ExprFinalE1 passa ser primeiro nivel de XE1
  strcpy(ExprFinalE2,XE2[nivelX]);
  for (VariaXN=nivelX;VariaXN>0;VariaXN--){
      for (i=0; i<=strlen(ExprFinalE1);i++)
   if (ExprFinalE1[i]=='X') {PosX=i; i=10000;}
```

```
      strncpy(RestoEsq,ExprFinalE1,PosX);
      RestoEsq[PosX]='\0';       //RestoEsq é a parte esquerda do X no (nivel)

      strcat(RestoEsq,XE1[VariaXN-1]); //Concatena ^ acima com todo o (nivel - 1)
      j=0;
      for (i=PosX+1; i<=strlen(ExprFinalE1);i++){
   RestoDir[j]=ExprFinalE1[i]; j++;
      }
      RestoDir[j]='\0';
      strcat(RestoEsq,RestoDir); //Concatena para o que ha em RestoEsqu o restoDir(Nivel)
      strcpy(ExprFinalE1,RestoEsq);

      for (i=0; i<=strlen(ExprFinalE2);i++)
   if (ExprFinalE2[i]=='X') {PosX=i; i=10000;}
      strncpy(RestoEsq,ExprFinalE2,PosX);
      RestoEsq[PosX]='\0';       //RestoEsq é a parte esquerda do X no (nivel)
      strcat(RestoEsq,XE2[VariaXN-1]); //Concatena ^ acima com todo o (nivel - 1)
      j=0;
      for (i=PosX+1; i<=strlen(ExprFinalE2);i++){
   RestoDir[j]=ExprFinalE2[i]; j++;
      }
      RestoDir[j]='\0';
      strcat(RestoEsq,RestoDir); //Concatena para o que ha em RestoEsqu o restoDir(Nivel)
      strcpy(ExprFinalE2,RestoEsq);
  }
}

  EditE1 ->Insert(ExprFinalE1);
  EditE2 ->Insert(ExprFinalE2);
    ModifyMenu( GetMenu( HWindow ), IDM_GERAR_LOTOS, MF_BYCOMMAND , IDM_GERAR_LOTOS, "Save
&Protocol" );
  DrawMenuBar( HWindow );
  }
}
  else
     MessageBox(HWindow, "Option not available!", "You has selected:", MB_OK);
}

//-----------------------------------------------------------------------------
//ESCOLHA
//-----------------------------------------------------------------------------
void TFerrTransp::escolha(char *ParteProServ){
   int PriColch,partes=0,vezes,i,MesmoSitio=1;
   ContSinc=0;
   while (1) {
      PriColch=0;
      for (i=0; i<=strlen(ParteProServ);i++)
         if (ParteProServ[i]=='[') { PriColch=i; i=10000;}
      if (PriColch==0){
          strcpy(Parte[partes],ParteProServ);
          break;
      }

      strncpy(Parte[partes],ParteProServ,PriColch); //pega parte esquerda do 1ro []
      Parte[partes][PriColch]='\0';
      ptr = strstr(ParteProServ,"[]"); //pega parte a direita da primeira escolha
      strcpy(ParteProServ,ptr); // ParteProServ passa a ser a parte direita
        for (i=0; i<strlen(ParteProServ)-1;i++) ParteProServ[i]=ParteProServ[i+2]; //"[]xxx" >>
"xxx"
      partes++; //incrementa o numero de expressoes: xxxx1 [] xxxx2 [] xxxxx3
   }
   for (vezes=0;vezes<=partes;vezes++){
      for (i=0; i<=strlen(Parte[vezes]);i++)
          if (Parte[vezes][0]=='X') {AcaoInt[vezes]=PriAcaoInt[vezes]; i=10000;}
          else{
             if (Parte[vezes][i]==';') {  // Armazena em sítio[vezes] 1 ou 2 (ações no sítio 1 ou
2)
                AcaoInt[vezes]=Parte[vezes][i-1]; i=10000;
          }
      }
   }
```

```
   for (vezes=1;vezes<=partes;vezes++) // Verifica se ações são no mesmo sítio
      if (AcaoInt[vezes] != AcaoInt[vezes-1]) MesmoSitio=0;

   for (i=0;i<=7;i++){
      ParteE1[i][0]='\0';
      ParteE2[i][0]='\0';
   }

   if (MesmoSitio) {  //Se as ações ocorrem no mesmo sítio
      for (vezes=0;vezes<=partes;vezes++){
         ContSinc++;
         VemEscolha=1;
         sequencia(Parte[vezes]);
         VemEscolha=0;

        //Otimização
        if (CheckBox->GetCheck() == BF_CHECKED ) {
           int encpart=0;
           for (i=0;i<=vezes;i++)
              if (strcmp(ParteE1[i],E1)==0 && strstr(ParteE1[i],"X")==NULL)
                 encpart=1;
            if (encpart==0) strcpy(ParteE1[vezes],E1);  //ParteE1[...] são as partes da expressao
entre parenteses
           encpart=0;
           for (i=0;i<=vezes;i++)
              if (strcmp(ParteE2[i],E2)==0 && strstr(ParteE1[i],"X")==NULL)
                 encpart=1;
           if (encpart==0) strcpy(ParteE2[vezes],E2);  //  parteE1 [] parteE2 [] ...
        }
        else{
           strcpy(ParteE1[vezes],E1);  //ParteE1[...] são as partes da expressao entre parenteses
           strcpy(ParteE2[vezes],E2);  //  parteE1 [] parteE2 [] ...
        }
     }
     nivelX++;
     PriAcaoInt[nivelX]=AcaoInt[0];
        for  (vezes=0;vezes<=partes;vezes++){   //até  o  strcat(XE2[0],")")  ...   (m12;stop  []
a1;m12...)
         if (vezes==0) {
            if (UltEscolha == 1){
               strcpy(XE1[nivelX],ParteE1[vezes]);
          strcpy(XE2[nivelX],ParteE2[vezes]);
            }
       else{
      if (ParteE1[1][0] != '\0') strcpy(XE1[nivelX],"(");
      strcat(XE1[nivelX],ParteE1[vezes]);
      if (ParteE2[1][0] != '\0') strcpy(XE2[nivelX],"(");
               strcat(XE2[nivelX],ParteE2[vezes]);
           }
        }
   else{
      if (ParteE1[vezes][0] != '\0'){
         strcat(XE1[nivelX]," [] ");
         strcat(XE1[nivelX],ParteE1[vezes]);
      }
      if (ParteE2[vezes][0] != '\0'){
         strcat(XE2[nivelX]," [] ");
         strcat(XE2[nivelX],ParteE2[vezes]);
      }
   }
     }
     if (UltEscolha == 0){
    if (ParteE1[1][0] != '\0') strcat(XE1[nivelX],")");
    if (ParteE2[1][0] != '\0') strcat(XE2[nivelX],")");
     }
  }
  else { //Ações entre sítios diferentes
     ContSinc++;   // para iniciar sincronização em s1
     existe=0;
     for (i=0;i<=meios;i++)
```

```
        if (strcmp(meio[i],"poll")==0) existe=1;
     if (existe==0) {
        strcpy(meio[meios],"poll"); //adiciona poll à sincroniz do meio
        meios++;  }

      //Otimização

     for (vezes=0;vezes<=partes;vezes++){
        ContSinc++;
        VemEscolha=1;
        sequencia(Parte[vezes]);
        VemEscolha=0;
        strcpy(ParteE1[vezes],E1);  //ParteE1[...] são as partes da expressao entre parenteses
        strcpy(ParteE2[vezes],E2);  //  parteE1 [] parteE2 [] ...
      }
      nivelX++;
      PriAcaoInt[nivelX]=AcaoInt[0];

      if (UltEscolha == 0) { // Verifica se há um nivel acima dessa escolha para colocar (
        strcpy(XE1[nivelX],"(poll;( ");
    strcpy(XE2[nivelX],"(poll;( ");
        }
      else {
        strcpy(XE1[nivelX],"poll;( ");
    strcpy(XE2[nivelX],"poll;( ");
      }

      int primeiroN2=0; //Identifica se ocorreu alguma ação no sítio 2
         for  (vezes=0;vezes<=partes;vezes++){   //até  o  strcat(XE2[0],")")  ...   (m12;stop  []
a1;m12...)
   if (AcaoInt[vezes]=='2') {
            if (primeiroN2==1) {
           strcat(XE1[nivelX]," [] ");
       strcat(XE2[nivelX]," [] ");
         }
         else primeiroN2=1;
         strcat(XE1[nivelX],ParteE1[vezes]);
         strcat(XE2[nivelX],ParteE2[vezes]);
   }
      }

      strcat(XE1[nivelX]," [] poll; E1)\r\n");
      strcat(XE2[nivelX]," [] poll; E2)\r\n");

         for  (vezes=0;vezes<=partes;vezes++){   //até  o  strcat(XE2[0],")")  ...   (m12;stop  []
a1;m12...)
   if (AcaoInt[vezes]=='1') {
      strcat(XE1[nivelX]," [] ");
           strcat(XE2[nivelX]," [] ");
           strcat(XE1[nivelX],ParteE1[vezes]);
      strcat(XE2[nivelX],ParteE2[vezes]);
        }
      }

      strcat(XE1[nivelX]," )");
      strcat(XE2[nivelX]," )");
  }
}




//---------------------------------------------------------------------
//HABILITAÇÃO && INTERRUPÇÃO &&PARALELISMO INDEPENDENTE
//---------------------------------------------------------------------
void TFerrTransp::habil_inter_paral(char *ParteProServ){
   int PriColch,partes=0,vezes,i;
   ContSinc=0;
   while (1) {
      PriColch=0;
```

```
      for (i=0; i<=strlen(ParteProServ);i++)
        if ((ParteProServ[i]=='>')||(ParteProServ[i]=='[')||(ParteProServ[i]=='|')) { PriColch=i;
i=10000;}
      if (PriColch==0){
         strcpy(Parte[partes],ParteProServ);
      break;
      }

      strncpy(Parte[partes],ParteProServ,PriColch); //pega parte esquerda do 1ro [> ou >> ou |||
      Parte[partes][PriColch]='\0';
         if (Hab1Int2==1) ptr = strstr(ParteProServ,">>"); //pega parte a direita da primeira
habilitação
      else {if (Hab1Int2==2) ptr = strstr(ParteProServ,"[>");
              else ptr = strstr(ParteProServ,"|||");}                    //pega parte a direita da
primeira interrupção
      strcpy(ParteProServ,ptr); // ParteProServ passa a ser a parte direita
      if (strstr(RestoEsq,"|||")!=NULL)
          for (i=0; i<strlen(ParteProServ)-1;i++) ParteProServ[i]=ParteProServ[i+3]; //"[]xxx" >>
"xxx"
      else
          for (i=0; i<strlen(ParteProServ)-1;i++) ParteProServ[i]=ParteProServ[i+2]; //"[]xxx" >>
"xxx"
      partes++; //incrementa o numero de expressoes: xxxx1 >> xxxx2 >> xxxxx3
   }
   for (vezes=0;vezes<=partes;vezes++){
      for (i=0; i<=strlen(Parte[vezes]);i++)
          if (Parte[vezes][i]==';') {  // Armazena em sítio[vezes] 1 ou 2 (ações no sítio 1 ou 2)
       AcaoInt[vezes]=Parte[vezes][i-1]; i=10000;
      }
   }

//   for (vezes=1;vezes<=partes;vezes++) // Verifica se ações são no mesmo sítio
//      if (AcaoInt[vezes] != AcaoInt[vezes-1]) MesmoSitio=0;

   for (i=0;i<=7;i++){
   ParteE1[i][0]='\0';
   ParteE2[i][0]='\0';
   }

      for (vezes=0;vezes<=partes;vezes++){

    ContSinc++;
    VemEscolha=1;
    IsInterrupt=0;
    if (Hab1Int2==2) IsInterrupt=1;
    sequencia(Parte[vezes]);
    VemEscolha=0;

    //Otimização
    if (CheckBox->GetCheck() == BF_CHECKED ) {
           int encpart=0;
          for (i=0;i<=vezes;i++)
        if (strcmp(ParteE1[i],E1)==0 && strstr(ParteE1[i],"X")==NULL)
           encpart=1;
      if (encpart==0) strcpy(ParteE1[vezes],E1);  //ParteE1[...] são as partes da expressao entre
parenteses
          encpart=0;
      for (i=0;i<=vezes;i++)
         if (strcmp(ParteE2[i],E2)==0 && strstr(ParteE1[i],"X")==NULL)
           encpart=1;
      if (encpart==0) strcpy(ParteE2[vezes],E2);  //  parteE1 [] parteE2 [] ...
      }
      else{
        strcpy(ParteE1[vezes],E1);  //ParteE1[...] são as partes da expressao entre parenteses
        strcpy(ParteE2[vezes],E2);  //  parteE1 [] parteE2 [] ...
    }
      }
      nivelX++;
      PriAcaoInt[nivelX]=AcaoInt[0];
         for (vezes=0;vezes<=partes;vezes++){   //até o strcat(XE2[0],")") ...   (m12;stop []
a1;m12...)
```

```
        if (vezes==0) {
           if (UltEscolha == 1){
              strcpy(XE1[nivelX],ParteE1[vezes]);
         strcpy(XE2[nivelX],ParteE2[vezes]);
           }
       else{
      if (ParteE1[1][0] != '\0') strcpy(XE1[nivelX],"(");
      strcat(XE1[nivelX],ParteE1[vezes]);
      if (ParteE2[1][0] != '\0') strcpy(XE2[nivelX],"(");
                strcat(XE2[nivelX],ParteE2[vezes]);
           }
        }
   else{
      if (ParteE1[vezes][0] != '\0'){
         if (Hab1Int2==1) strcat(XE1[nivelX]," >> ");
         else { if (Hab1Int2==2) strcat(XE1[nivelX]," [> ");
                else strcat(XE1[nivelX]," ||| ");}
         strcat(XE1[nivelX],ParteE1[vezes]);
      }
      if (ParteE2[vezes][0] != '\0'){
         if (Hab1Int2==1) strcat(XE2[nivelX]," >> ");
         else { if (Hab1Int2==2) strcat(XE2[nivelX]," [> ");
                else strcat(XE2[nivelX]," ||| ");}
         strcat(XE2[nivelX],ParteE2[vezes]);
           }
        }
     }
     if (UltEscolha == 0){
       if (ParteE1[1][0] != '\0') strcat(XE1[nivelX],")");
       if (ParteE2[1][0] != '\0') strcat(XE2[nivelX],")");
     }
  }


//---------------------------------------------------------------------------
//SEQÜÊNCIA
//---------------------------------------------------------------------------
void TFerrTransp::sequencia(char *xTodaExpressao){
   rtrim(xTodaExpressao);
   int PrimPtvg,i,j;
//,UsouSinc=0;

   if (ContSinc==0) ContSinc++; //caso seja apenas uma seqüência
   char dir='0',string[4],sincro[4];
   E1[0]='\0'; //zera E1
   E2[0]='\0'; //zera E2
   sincro[0]='\0';
   strcpy(sincro,"s");
   strcat(sincro,itoa(ContSinc,string,8));

   if (IsInterrupt==1){ //interrupcao deve ter sempre sincronização
      for (j=1;j<=10;j++){ //diferente das já existentes
         sincro[0]='\0';
         strcpy(sincro,"s");
         strcat(sincro,itoa(j,string,8));
         int XEncSinc=0;
         for (i=0;i<=meios;i++) if (strstr(meio[i],sincro) != NULL) XEncSinc=1;
         if (XEncSinc==0) j=11;
      }
   }
// MessageBox(HWindow, sincro, "xSINCROx", MB_OK);
   while(1) {
       PrimPtvg=0;
       for (i=0; i<=strlen(xTodaExpressao);i++)
       if (xTodaExpressao[i]==';') { PrimPtvg=i; i=10000;}
       if (PrimPtvg != 0){
      strncpy(parte,xTodaExpressao,PrimPtvg); //pega parte esquerda do 1ro ptvg
      parte[PrimPtvg]='\0';

      if (parte[strlen(parte)-1] =='1') evsitio='1'; //identifica se evento é no 1ro ou
      else{
```

```
        if (parte[strlen(parte)-1] =='2') evsitio='2'; // no segundo sítio
        else {
           FimTransp("Error: Event must occur in place 1 or 2 only!");
           break;
        }
      }
      if ((parte[0] =='X' ) && (dir =='0')) {
         FimTransp("Error: Sequence after recursive call or STOP");
         break;
               }
      ptr = strstr(xTodaExpressao,";"); //xPro_serv passa a ser a parte direita do 1ro ptvg
      strcpy(xTodaExpressao,ptr);
      for (i=0; i<strlen(xTodaExpressao);i++) xTodaExpressao[i]=xTodaExpressao[i+1]; //"; xxx" >>
" xxx"
      evsitio=parte[strlen(parte)-1];

      if (dir=='0') {
          dir=evsitio;
          if (evsitio=='1') {
            existe=0;
         for (i=0;i<=acoes1;i++)
              if (strcmp(acao1[i],parte)==0) existe=1;
         if (existe==0) {
              strcpy(acao1[acoes1],parte);
              acoes1++;}
         strcat(E1,strcat(parte,";"));}
          if (evsitio=='2')  {
            existe=0;
              for (i=0;i<=acoes2;i++)
            if (strcmp(acao2[i],parte)==0) existe=1;
         if (existe==0) {
            strcpy(acao2[acoes2],parte);
            acoes2++;}
         strcat(E2,strcat(parte,";"));}
      }
      else{
          if (parte[0] =='X') {
         FimTransp("Error: Sequence after recursive call or STOP");
         break;
                   }
          if (parte[strlen(parte)-1]=='1'){
             if (dir=='2'){
            dir='1';
            existe=0;
            for (i=0;i<=acoes1;i++)
                 if (strcmp(acao1[i],parte)==0) existe=1;
            if (existe==0) {
                 strcpy(acao1[acoes1],parte);
                 acoes1++;}
            strcat(E2,sincro);
            strcat(E2,";");
          //  UsouSinc=1;
            existe=0;
            for (i=0;i<=meios;i++)
                if (strcmp(meio[i],sincro)==0) existe=1;
            if (existe==0) {
               strcpy(meio[meios],sincro);
               meios++; }
            strcat(E1,sincro);
            strcat(E1,";");
            strcat(E1,strcat(parte,";"));
             }
            else {
                existe=0;
                for (i=0;i<=acoes1;i++)
               if (strcmp(acao1[i],parte)==0) existe=1;
               if (existe==0) {
                   strcpy(acao1[acoes1],parte);
                   acoes1++;}
               strcat(E1,strcat(parte,";"));
                    }
```

```
        }
    else{
    if (dir=='1'){
        dir='2';
        existe=0;
        for (i=0;i<=acoes2;i++)
            if (strcmp(acao2[i],parte)==0) existe=1;
            if (existe==0) {
               strcpy(acao2[acoes2],parte);
               acoes2++;}
               strcat(E1,sincro);
            //   UsouSinc=1;
                strcat(E1,";");
            existe=0;
            for (i=0;i<=meios;i++)
               if (strcmp(meio[i],sincro)==0) existe=1;
            if (existe==0) {
               strcpy(meio[meios],sincro);
               meios++;
            }

            strcat(E2,sincro);
             strcat(E2,";");
            strcat(E2,strcat(parte,";"));
         }
         else {
            existe=0;
            for (i=0;i<=acoes2;i++)
               if (strcmp(acao2[i],parte)==0) existe=1;
            if (existe==0) {
               strcpy(acao2[acoes2],parte);
               acoes2++;}
            strcat(E2,strcat(parte,";"));
         }
      }
   }
}
else{
   if (xTodaExpressao[0] =='X') {
      if ((E1[0]!='\0') || (E2[0]!='\0')){
         strcat(E1,sincro);  //  strcat(E1,"s..;X");
         strcat(E1,";X");
         strcat(E2,sincro);
         strcat(E2,";X");
      //   UsouSinc=1;
         existe=0;
         for (i=0;i<=meios;i++)
            if (strcmp(meio[i],sincro)==0) existe=1;
            if (existe==0) {
               strcpy(meio[meios],sincro);
               meios++;}
         break;
      }else{
         strcat(E1,"X");
         strcat(E2,"X");
         break;
      }
   }
   int encontrou = stricmp(xTodaExpressao, "STOP");
   if (encontrou ==0){
      strcat(E1,xTodaExpressao);
      strcat(E2,xTodaExpressao);
      break;}
   encontrou = stricmp(xTodaExpressao, "SERV");
   if (encontrou==0){
      for (i=0 ; i<=meios ; i++){
         if (strstr(meio[i],"c")) encontrou = 1;
         }
      if (encontrou==0) {
         strcpy(meio[meios],"c");
         meios++;
```

```
                                        }
                strcat(E1,"c;");
                strcat(E2,"c;");
                strcat(E1,"E1\r\n");
                strcat(E2,"E2\r\n");
                break;}
             encontrou = stricmp(xTodaExpressao, "EXIT");
             if (encontrou==0){
                strcat(E1,"exit");
                strcat(E2,"exit");
                break;}
             else{
                FimTransp("Syntaxe error: STOP,EXIT or SERV expected");
                break;
             }
       }
    }
//    if (UsouSinc==0) ContSinc--;
}

void TTestApp::InitMainWindow(){
  MainWindow = new TFerrTransp(NULL);
}

void TTestApp::InitInstance(){
      TApplication::InitInstance();
      BWCCGetVersion();  // init of BWCC required to assure load of BWCC.DLL
}

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
  LPSTR lpCmdLine, int nCmdShow)
{
    TTestApp TestApp("TranSP tool - Transformation from Service to Protocol", hInstance,
hPrevInstance,
    lpCmdLine, nCmdShow);
  TestApp.Run();
  return TestApp.Status;
}

//---------------------------------------------------------------------------
//FAZ GRÁFICO
//---------------------------------------------------------------------------
void TFerrTransp::desenha(HDC DC)
{
   SetBkColor( DC, 2);
   HPEN hCaneta;
   hCaneta=CreatePen (PS_SOLID,1,6);
   int linic=120,colic=10,linf=140,i;
   Rectangle( DC, colic+30, linic+20, colic+160, linic+70 );
   Rectangle( DC, colic+210, linic+20, colic+340, linic+70 );
   SetBkMode( DC, TRANSPARENT );
   TextOut( DC, colic+90, linic+40, Ex, strlen( Ex ) );
   TextOut( DC, colic+270, linic+40, Ey, strlen( Ey ) );
   hCaneta=CreatePen (PS_DOT,1,6);
   SelectObject( DC, hCaneta);
   MoveTo(DC,colic,linic);
   LineTo(DC,colic+370,linic);
   LineTo(DC,colic+370,linic+100);
   LineTo(DC,colic,linic+100);
   LineTo(DC,colic,linic);
   LineTo(DC,colic+15,linic-15);
   LineTo(DC,colic+385,linic-15);
   LineTo(DC,colic+385,linic+85);
   LineTo(DC,colic+15,linic+85);
   LineTo(DC,colic+15,linic-15);
   MoveTo(DC,colic+370,linic);
   LineTo(DC,colic+385,linic-15);
   MoveTo(DC,colic+370,linic+100);
   LineTo(DC,colic+385,linic+85);
   MoveTo(DC,colic,linic+100);
   LineTo(DC,colic+15,linic+85);
```

```
    const alt=0,cmess=25,cmess2=205; //linha inicial das acoes
    int fator;
    for (i = 1; i<=acoes1; i++) {
          fator=160/(acoes1+1);
          MoveTo(DC,cmess+fator*i,alt+i*15);
          LineTo(DC,cmess+fator*i,linf); //
          TextOut( DC, cmess+fator*i,alt+(15*i)-15,acao1[i-1],strlen(acao1[i-1]));
          }
    for (i = 1; i<=acoes2; i++) {
          fator=160/(acoes2+1);
          MoveTo(DC,cmess2+fator*i,alt+i*15);
          LineTo(DC,cmess2+fator*i,linf); //
          TextOut( DC, cmess2+fator*i,alt+(15*i)-15,acao2[i-1],strlen(acao2[i-1]));
          }
    for (i = 1; i<=meios; i++) {
          fator=60/(meios+1);
          MoveTo(DC,colic+160,linf-5+fator*i);//
          LineTo(DC,colic+210,linf-5+fator*i);
          TextOut(DC,colic+170,linf-20+fator*i,meio[i-1],strlen(meio[i-1]));}
}

char *ltrim(char *string)
{
    char *stringTrim;
    stringTrim=string;
    int i,j;
    for (i=0;i<strlen(stringTrim);i++) {
       if (stringTrim[i]==' ') {
          for (j=i;j<strlen(stringTrim);j++) stringTrim[j]=stringTrim[j+1];
          i--;
          }
       else i=strlen(stringTrim);
    }
    return stringTrim;
}

char *rtrim(char *string)
{
    char *stringTrim;
    stringTrim=string;

// int i;
    for (i=strlen(stringTrim)-1;i>=0;i--) {
       if (stringTrim[i]==' ') stringTrim[i]=stringTrim[i+1];
       else i=0;
    }
    return stringTrim;
}

char *alltrim(char *string)
{
    char *stringTrim;
    stringTrim=string;
    int i,j;
    for (i=0;i<strlen(stringTrim);i++) {
       if (stringTrim[i]==' ') {
          for (j=i;j<strlen(stringTrim);j++) stringTrim[j]=stringTrim[j+1];
          i--;
          }
       else i=strlen(stringTrim);
    }

    for (i=strlen(stringTrim)-1;i>=0;i--) {
       if (stringTrim[i]==' ') stringTrim[i]=stringTrim[i+1];
       else i=0;
    }
    return stringTrim;
}

char *allmtrim(char *string)
```

```
{
   char *stringTrim;
   stringTrim=string;
   int i,j;
   for (i=0;i<strlen(stringTrim);i++) {
      if (stringTrim[i]==' ') {
         for (j=i;j<strlen(stringTrim);j++) stringTrim[j]=stringTrim[j+1];
         i--;
      }
   }
   return stringTrim;
}


void TFerrTransp::GetWindowClass( WNDCLASS& WndClass )
{
   TDialog::GetWindowClass( WndClass );
   WndClass.style = 0;    //Estilo da Classe
// WndClass.hbrBackground = (HBRUSH)GetStockObject( GRAY_BRUSH );
   WndClass.lpszMenuName = "TRANSPMENU";
   WndClass.hIcon = LoadIcon( GetApplication()->hInstance, "transp" );
}


void TFerrTransp::Sobre( TMessage& ){
     GetApplication()->ExecDialog(new TDialog(this, "Sobre"));
}

void TFerrTransp::FimTransp(char *Mensagem) {
     MessageBox(HWindow, Mensagem,"TranSP", MB_ICONSTOP);
      ModifyMenu( GetMenu( HWindow ), IDM_GERAR_LOTOS, MF_BYCOMMAND | MF_GRAYED, IDM_GERAR_LOTOS,
"Save &Protocol" );
     DrawMenuBar( HWindow );
     Finaliza=TRUE;
}

class Tmp : public TWindow{
public:
   Tmp( TFerrTransp& tft ) : TWindow( &tft, "Graphics" )
   {Attr.Style |= WS_POPUPWINDOW | WS_CAPTION | WS_VISIBLE;
      Attr.X = 10; Attr.Y = 10; Attr.W = 410; Attr.H=260; }
   ~Tmp()
   { ((TFerrTransp*)Parent)->GraphicWindow = NULL; }
protected:
   void Paint( HDC hDC, PAINTSTRUCT& PS );
};

void Tmp::Paint( HDC hDC, PAINTSTRUCT& ){
   ((TFerrTransp*)Parent)->desenha( hDC );
}

void TFerrTransp::MostraGrafico(RTMessage){
   if (GraphicWindow)
      InvalidateRect(GraphicWindow->HWindow,NULL,TRUE);
        else
      GraphicWindow = (TWindow*)GetModule()->MakeWindow( new Tmp(*this) );
}


char *strtran(char *string, char *parte1, char *parte2)
{
//    012345678901234567890    3456
//    borland Internacional    land
//    tam2 = strlen (parte1)=4
  int eigual,i,j;
  char esquerda[50],resultado[150];
  resultado[0]='\0';

  for (j=0; j<strlen(string);j++){
    eigual=1;
    for (i=0; i<strlen(parte1)-1; i++){
```

```
      if (string[j+i] != parte1[i])
         eigual=0;
    }
    if (eigual==1) {
       strncpy(esquerda,string,j);
       strcat(resultado,esquerda);
       strcat(resultado,parte2);
    }
  }

  return resultado;
}
```

9. Bibliografia

[BoGo86] Bochmann, G. v.; Gotzhein, R.: "*Deriving protocol specifications from service specifications*" in: Communications, Architectures & Protocols, Proceedings of the ACM SIGCOMM'86 Symposium, Vermont, USA, 1986.

[BoBr87] Bolognesi, T.; Brinksma, E.: "*Introduction to the OSI specification language LOTOS*". Computer Networks and ISDN Systems, V.14, pp. 25-29, 1987.

[BoFe90] Bouajjani, A.; Fernandez, J. C.; Graf, S.; Rodriguez, C.; Sifakis, J.: "*Safety for Branching time semantics*". LGI-IMAG / VERILOG, França, 1990, pp. 17.

[Brin88] Brinksma, E.: "*A tutorial on LOTOS*", ISO8807 Information Processing Systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour". 1988.

[CaFa94] Camargo, M. C.; Farines, J. M.: "*Tornando LOTOS apta para especificar sistemas Tempo Real*". Em: 12º Simpósio Brasileiro de Redes de Computadores, Vol I, pp 245-264, Curitiba, 16 a 20 de Maio de 1994.

[Carv92] Carvalho, T. C. M. B.: "*Gerenciamento de Redes. Uma abordagem de sistemas abertos*". Ed. Makron Books, pp. 364, outubro de 1992.

[ChLi 88a] Chu, P.M.; Liu, M. T.: "*Synthesizing protocol specifications from service specifications in FSM Model*" in Proc. Computer Networking Symp. 88, pp 82-173, Abril de 1988.

[ClJo92] Clark, R. G.; Jones, V. M.: "*Use of LOTOS in the formal development of an OSI protocol*", Computer Communications, V.15, N.2, pp. 86-92, março de 1992.

[ClJo92] Clark, R. G.; Jones, V. M.: "*Use of LOTOS in the formal development of an OSI protocol*", Computer Communications, V.15, N.2, pp. 86-92, março de 1992.

[DrCh92] Drayton, L: Chetwynd, A.; Blair, G.: "*Introduction to LOTOS through worked example*". School of Engineering Computing and Mathematical Sciences, Lancaster University - UK, 1992.

[EhMa85] Ehrig, H.; Mahr, B.: "*Fundamentals of algebraic specification 1*", Springer-Verlag, Berlin, 1985.

[ErFr91] Ernberg, P; FreudLund, L.; Hansson, H.; Jonsson, B.; Orava, F.; Pehrson, B.: "*Guidelines for specification and verification of communication protocols*". Swedish Institute of Computer Science, SICS Perspective, report n. 1, 1991, p.34.

[Fern90] Fernandez, J. C.: "*Aldébaran: a tool for verification of communicaing*". IMAG, França, 1990.

[FeGa93] Fernandez, J. C.; Garavel, H; Mounier, L.; Rasse A.; Rodrigues, C.; Sifakis, J.: "A toolbox for the v*erification of LOTOS programs*". LGI-IMAG / VERILOG, França, 1993, pp.14.

[FRV85] Fernandez, J. C.; Richier, J. L.; Voiron, J.: "*Verification of Protocol Specifications using the CESAR System*". In Michel Diaz, editor, *Proceedings of the 5th IFIP International Workshop on Protocol Specification, Testing and Verification (Moissac, France)* , pp 71-90, Amsterdam, June 1985. IFIP, North-Holland.

[Hoar85] Hoare C.A.R.: "*Communicating Sequential Processes*". Prentice Hall, 1985.

[Holz91] Holzmann, G. J.: "*Design and validation of computer protocols*". Prentice-Hall International Editions. 1991, pp. 500.

[Hult94] Hultström, M.: "*Structural Decomposition*"; In: Conferenc Proceedings of the *"Protocol Specification, Testing & Verification"* , Vancouver, B.C. Canadá, pp. 193-209. 7-10 Junho de 1994.

[HoUl79] Hopcroft, J. E.; Ullman, J. O.: "*Introduction to Authomata theory, languages and computation*". Addison - Wesley : Reading, 1979.

[IS3067] ISO / IEC JTC1/SC21 N3067: "*Specification styles for structuring of OSI formal descriptions*", Information Retrieval, Transfer and Management for OSI, 19 de setembro, 1988.

[IS8807] ISO - International Standard. Information processing systems - Open systems interconnection - "*LOTOS - A formal description technique based on the temporal ordering of observational behaviour*", 1989.

[ISO89] ISO, Information Processing Systems, Open Systems Interconnection, "*OSI conformance testing methodology and framework*", DIS 9646.

[IS9595] ISO - International Standard. Information Technology - Open systems interconnection - "*Common management information service définition (CMIS)"* , 1991.

[IS9596] ISO - International Standard. Information Technology - Open systems interconnection - "*Common management information protocol définition (CMIP)"* , 1991.

[KaHi90] Kant, C.; Higashino, T.; Bochmann, G. van.: "*Deriving protocol specifications from service specifications written in Basic LOTOS*". Département d' IRO. Montreal, Canadá. pp. 1-16, 1990.

[Khen89] Khendek, F.: "*Dérivation de protocoles à partir de services de communication écrits dans un sous-ensemble de LOTOS*" M.Sc. Thesis. Montreal, Canadá, 1989.

[KhBo89] Khendek, F.; Bochmann, G. von; Kant, C.: "*New results on deriving protocol specifications from service specifications*". Montreal, Canadá. pp. 1-16. 1989.

[KhBo94] Khoumsi, A.; Bochmann, G. von; Dssouli, R.: "*On specifying services and synthesizing protocols for real-time applications*"; In: Conference Proceedings of the *"Protocol Specification, Testing and Verification"* , Vancouver, B.C. Canadá, pp. 177-192. 7-10 Junho de 1994.

[Lang90] Langerak, R.: "*Decomposition of functionality: a correctness preserving LOTOS transformation*"; In: Participant's Proceedings of the 10th International Symposium on Protocol Specification, Testing, and Verification, Ottawa - Canadá. IFIP, pp. 203-218, 1990.

[LePe94] Lehmann, E, O Jr.; Pedroza, A. C. P.: "*Especificação e verificação do protocolo CMIP para gerenciamento de rede*". Em: 12º Simpósio Brasileiro de Redes de Computadores, Vol II, pp 639-659. Curitiba, 16 a 20 de Maio de 1994.

[LoCh92] Loureiro, A., Chanson, S., Vuong, S.: "*FDT tools for protocol development*". In: Forte'92. Dep. of Computer Science, University of British Columbia. Vancouver, B.C. Canadá. pp.36-78, 1992.

[Manh94] Manhas Junior, E B.: "*Uma metodologia para o desenvolvimento de aplicações distribuídas baseada na Técnica de Descrição Formal Estelle*". Dissertação de Mestrado em Engenharia Elétrica. UFSC. Florianópolis, Julho de 1994. pp. 114.

[Mayr89] Mayr, T.: "*Specification of object-oriented systems in LOTOS*". Technische Universität Wien - Institut für Angewand Informatik, Áustria. pp. 107-119, 1989.

[Mazz91] Mazzola, V. B.: "*Contribuition à la conception de systèmes Flexibles de prodution: Application de la Technique de description formelle Estelle*". Tese de Doutorado. Toulouse, novembro de 1991.

[MeBo83] Merlin, P., Bochmann, G. v.: "*On the construction of submodule specifications and communication protocols*". ACM Trans. on Programming Languages and Systems. No. 1, pp. 1-25, Jan. 1983.

[Miln80] Milner, R.: "*A calculus of communicating systems*". Lecture Notes in Computer Science, 92, Springer-Verlag, Berlin, 1980.

[Miln89] Milner, R.: "*Communication and concurrency*". Prentice-Hall, New York, 1989.

[MoCl93] Moreira, A. M. D.; Clark, R. G.: "*Os métodos formais na análise de orientação por objectos*". In: Anais do VIII Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, 26-29 outubro 1993, pp. 238-252. 1993.

[NoRi94] Notare, M. S. M. A., Riso, B. G.: "*Transformação de especificações com a técnica de descrição formal LOTOS*". In: Anais da 2. Jornada USP - SUCESU-SP de Informática e Telecomunicações, São Paulo. pp. 263-272, 30-01 junho de 1994.

[Nota95] Notare, M. S. M. A.: *"Uma metodologia para a especificação formal de serviços e protocolos de comunicação"*. Dissertação de Mestrado, Florianópolis-SC, pp. 203, abril de 1995.

[NoTo95a] Notare, M. S. M. A.; Tonin, N. A.; Riso, B. G. *"Especificação formal de um sistema de gerenciamento para redes heterogêneas"*. IX FENASOFT, São Paulo- SP, 17-21/07/95.

[NoTo95b] Notare, M. S. M. A.; Tonin, N. A.; Riso, B. G. *"Experiência na utilização de ferramentas LOTOS em aplicação para o gerenciamento de redes".* TELEMÁTICA' 95, Porto Alegre- RS, pp. 109-126, 22 a 24/08/1995.

[PiCu90] Pinheiro, T. S. de M.; Cunha, P. R. F.: "*Modelo de referência para especificação formal de sistemas operacionais distribuídos*". In: IV Simpósio Brasileiro de Engenharia de Software, pp. 14-29, Águas de São Paulo, SP, 1990.

[PiSc92] Pires, L. F.; Schot, J.: "*Design and implementation strategies*" , Memoranda Informatica 92-25, University of Twente, 1992.

[PiLo90] Pires, L. F.; Lopes de Souza, W.: "*Step-wise refinements design using LOTOS*". In: Proceedings of the 3rd International Conference on Formal Description Techniques (FORTE'90), Madrid, pp. 289-306, 1990.

[PiSi92] Pires, L. F.; Sinderen, M. van; Vissers, C.: "*On the use of pre-defined implementation constructs in distributed systems design*". Memoranda Informatica 92-08, University of Twente, 1992.

[PrSa91] Probert, R. L.: "*Synthesis of Communication Protocols: Survey and Assessment*". In: IEEE Transactions on Computers, Vol.40. Número 4. Abril de 1991, pp. 468-476.

[QuCu94] Queiroz, J. A. M. de; Cunha, P. R. F.: "*Sistemas distribuídos: de especificações LOTOS a implementações*". IX Escola de Computação. UFPE-DI, Recife, 24 a 31 de julho de 1994. pp. 208.

[Riso91] Riso, B.G. "Uma abordagem para o design de sistemas distribuídos e protocolos de comunicação". Tese de Doutorado, UFPB, Campina Grande PB, 11/91.

[RiNo95] Riso, B.G.; Notare, M. S. M. A; Tonin, N. A.: "Engenharia de protocolos com métodos formais". Livro em preparação.

[Rodr88] Rodríguez, C.: "*Spécification et validation de systèmes en XESAR*". Thèse de Doctorat, Institut National Polytechnique de Grenoble, Maio 1988.

[Sajk84] Sajkowski, M.: "*Protocol verification techniques: status quo and perspectives*". In: Proc. Fourth IFIP int Symp. Protocol Specification, Testing Verification, june, 1984.

[ScPi92] Schot, J.; Pires, L. F.: "*Design and implementation strategies*" Memoranda Informatica 92 - 57, University of Twente, 1992.

[SiPi92] Sinderen, M. van; Pires, L. F.; Vissers, C. A.: "*Protocol design and implementation using formal methods*". The Computer Journal, vol 35, no 5, pp. 478 - 491, 1992.

[ToNo94] Tonin, N. A.; Notare, M.S.M.A.; Riso B. G.: "*Algoritmos para a transformação de especificações LOTOS*". XXVII Congresso Nacional de Informática e Telecomunicações, SUCESU'94, Salvador BA, pp. 20, 21-24/11/94.

[ToNo95a] Tonin, N. A.; Notare, M. S. M. A.; Riso, B. G. *"The TranSP tool".* In: Proceedings of the Tool Demonstrations and Poster Displays of FORTE'95, Montreal, Canadá, 17-20/10/1995.

[ToNo95b] Tonin, N. A.; Notare, M. S. M. A.; Riso, B. G. *"TranSP: uma ferramenta para transformar especificações de serviço em especificações de protocolo".* INFOSUL'95, Porto Alegre, 23-27/10/1995.

[ToNo95c] Tonin, N. A.; Notare, M. S. M. A.; Riso, B. G. *"LOTOS specification of a flexible assembly cell".* In XV International Conference of the Chilean Computer Science Society. Arica, Chile, pp 393-401, 1-3/11/1995.

[ToNo95d] Tonin, N. A.; Notare, M. S. M. A.; Riso, B. G. *"Utilização da ferramenta TranSP na área de gerência de redes".* III Semana da Pesquisa, UFSC-SC, 6-10/11/1995.

[VaCi93] Valenzano, A.; Ciminiera, L; Sisto, R.: "*Rapid prototyping of protocols from LOTOS specifications*". Software-Practice and Experience, vol 23(1), pp. 31-54, 1993.

[ViSc88] Vissers, C. A.; Scollo, G.; Sinderen, M. van.: "*Architecture and specification style in formal descriptions of distributed systems*". In: Proceedings of the VIII International Conference on Protocol Specification, Testing, and Verification. IFIP, pp. 189-204, 1988.

[ViSc89] Vissers, C. A.; Scollo, G.; Sinderen, M. van; Brinksma, E.: "*On the use of Specification Styles in the Design of Distributed Systems*". In: Proceedings of TAPSOFT'89. Barcelona, Espanha, pp. 26, 1989.

[Vloe93] Vloedt, T. van der.: "*The LOTOS Toolbox*" In: Proceedings of the AMAST' 93 - Third International Conference on Algebraic Methodology and Software Technology. University of Twente, The Netherlands, june 21-25, 1993, pp. 337-338.

[Yuan88] Yuang, M. C.: "*Survey of protocol verification techniques based on finite state machine models*" In: Proc. Comput. Networking Symp., 1988, pp. 164-172.

[Zafi80] Zafiropulo, P.: "*Towards analyzing and synthesizing protocols*". In: IEEE Transactions on Computers, Vol.40. Número 4. Abril de 1980, pp. 651-661.