Um Simulador Funcional para Especificações LOTOS

Carlos André Guimarães Ferraz

Paulo Roberto Freire Cunha

Departamento de Informática

Universidade Federal de Pernambuco

CP 7851, 50739, Recife-PE, Brasil

Sumário

Este trabalho mostra o funcionamento de um sistema que executa protótipos de especificações LOTOS. Os protótipos são descritos através de programação funcional e permitem a simulação das especificações a partir da interação do usuário com o sistema. É o usuário quem faz o papel do ambiente externo do sistema especificado, gerando a sequência dos eventos de sincronização. O trabalho descreve o modo como se dá esta interação entre o usuário e o sistema.

1 Introdução

A linguagem de especificação LOTOS[ISO88,BoB87] é uma das Técnicas de Descrição Formal (TDF) desenvolvidas pela ISO (*International Organization for Standardazation*) por especialistas do ISO/TC97/SC21/WG1/FDT/Subgroup C, tendo sido especificamente projetada para descrever formalmente os serviços e protocolos do Modelo de Referência OSI (*Open Systems Interconnection*), embora se aplique a sistemas distribuídos

e concorrentes em geral.

LOTOS é um acrônimo para *Language Of Temporal Ordering Specification* (Linguagem de Especificação de Ordenamento Temporal), e apesar do nome talvez sugerir relação com lógica temporal, a técnica (TDF) se baseia em *álgebra de processos*. Assim sendo, LOTOS é uma técnica algébrica de especificação formal, que permite definições claras, não-ambíguas e independentes de implementação.

A linguagem LOTOS é composta de uma parte dinâmica, onde são descritos os comportamentos e interações dos processos que formam o sistema, e uma segunda parte, que trata da descrição das estruturas de dados e expressões de valores. O componente dinâmico de LOTOS se baseia, principalmente, no trabalho de Milner em CCS (*Calculus of Communicating Systems*) [Mil80], sendo também influenciado por CSP (*Communicating Sequential Processes*) [Hoa85]. O segundo componente é baseado na teoria formal de tipos abstratos de dados e, particularmente, na especificação equacional de tipos de dados. O maior volume de idéias para a concepção deste componente vem da técnica algébrica de tipos abstratos de dados ACT ONE[EhM85].

2 O Estilo Funcional de Especificação

Aqui nos preocupamos em apresentar a *programação funcional*, principalmente o que diz respeito ao objetivo de nosso trabalho, que é o de mostrar como são tratados os protótipos funcionais, possibilitando a simulação de especificações LOTOS.

Por causa da base matemática simples da programação funcional, o desenvolvimento de programas corretos é mais fácil de ocorrer do que usando o estilo imperativo tradicional de programação. Programas funcionais combinam a clareza requerida para a especificação formal de software com a habilidade para validar um projeto através de sua execução. Por isso, é que estes programas são chamados de *especificações executáveis*, e são considerados ideais para prototipagem.

A programação funcional tem seus primórdios na década de 60 com os trabalhos publicados por McCarthy[McC60] e Landin[Lan64]. O modelo computacional em que se baseia a programação imperativa, por exemplo, é o de que o efeito da execução de comandos em sequência é o de mudar o valor de uma variável ou variáveis, isto é, o *estado* da máquina. No entanto, há um modelo diferente, o do *cálculo*, que indica que uma expressão é calculada aplicando-se operadores a argumentos. Desta forma, a definição de funções ou operadores e sua aplicação a argumentos exclui *atribuição* e *controle* como elementos da programação. As linguagens de programação que têm como base funções e aplicações são denominadas *funcionais* ou *aplicativas*.

Função é usada no seu sentido matemático original:

> *Uma função retorna um resultado dependendo apenas de seus argumentos (ou entradas).*

As linguagens funcionais permitem que a programação adote um estilo *denotacional*, onde o que se deseja construir é mais importante do que como se constrói.

Se pensarmos em termos de um interpretador, uma especificação LOTOS é traduzida para um formato "interno" de aspecto funcional. A fidelidade da representação é garantida sob o ponto de vista de que a especificação pode ser recuperada a partir do formato funcional (protótipo) [Fer89].

O sistema executa sobre a representação interna, avaliando as expressões funcionais correspondentes às expressões de comportamento dos processos que interagem em uma especificação descrita em LOTOS. Uma de suas principais características é fato de tratar o *fluxo de informações* como uma *lista*. O resultado da execução é a sequência das ações que ocorreram – *história* – durante a simulação. O usuário exercita a especificação dando entrada em sequências de eventos que possam ser tratados em determinados pontos, simulando o ambiente que envolve o sistema descrito.

3 A Interface do Sistema

Para facilitar a entrada de dados (*lista-ambiente*), assim como a interpretação do resultado obtido, visto que as duas sequências são listas de eventos, elaboramos uma *interface* simples, que permite ao usuário descrever os dados de entrada sem preccupação com a sua estruturação, bem como percorrer a sequência final apresentada em uma forma mais legível.

Os itens que se apresentam no sistema são:

1. nomeação de eventos: os nomes dos eventos (ou portas) são inteiros para atender ao modelo funcional [Fer89] que visa tornar mais direta a passagem das especificações LOTOS para os protótipos funcionais. Neste caso específico da nomeação de eventos, o modelo considera as ações observáveis como inteiros positivos, enquanto as ações não-observáveis são inteiros negativos;

2. entrada de dados: é o usuário quem decide a sequência de eventos ofertados pelo "ambiente" externo;

3. execução de protótipos: o sistema permite que protótipos de especificações LOTOS escritos em uma linguagem funcional sejam executados considerando as sequências de eventos de entrada;

4. visualização dos resultados: os resultados das execuções dos protótipos são apresentados de forma mais legível ao usuário da simulação para que ele confira com os resultados esperados.

Os itens 1 e 2 são apresentados na tela da Fig. 1, na qual é requisitado o nome do arquivo onde se encontra definido o protótipo funcional da especificação a ser simulada para que os nomes dos eventos e a sequência de entrada sejam incorporados ao arquivo.

O item 3 se desenvolve em algum ambiente funcional que inclua uma biblioteca de

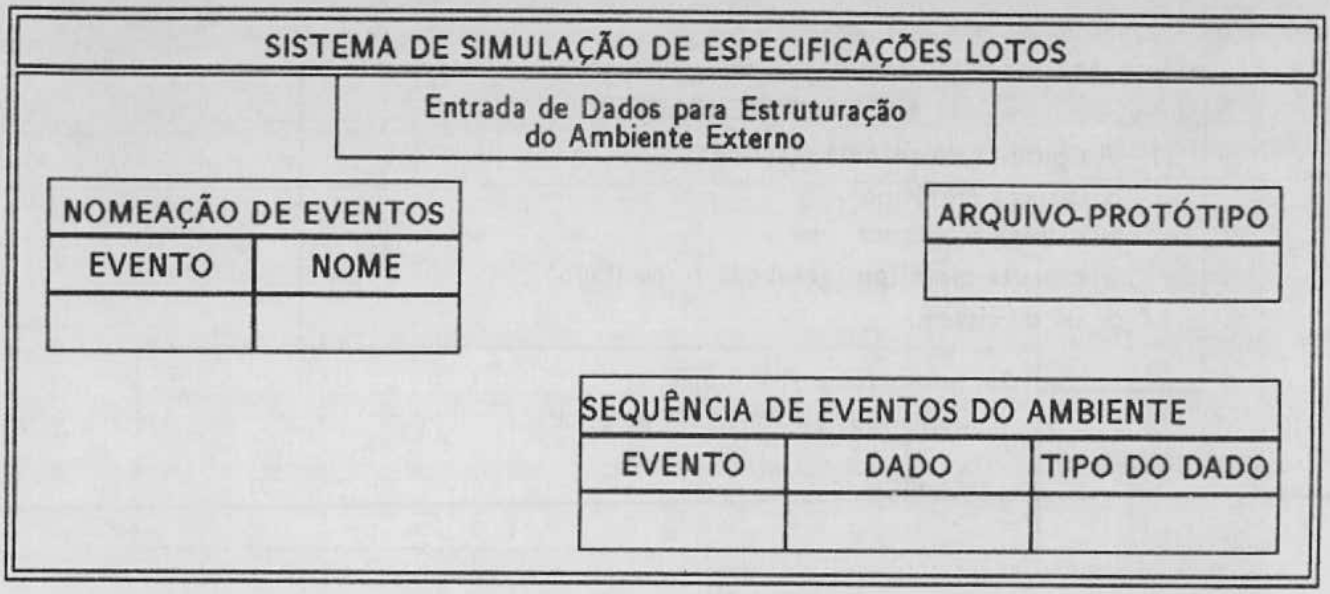

Figura 1: Tela de Inicialização do Sistema-simulador.

operadores correspondentes aos construtores LOTOS [Fer89,FCM89]. Os procedimentos devem ser[1]:

> carrega protótipo funcional (a lista de entrada está incorporada ao arquivo-protótipo)

> carrega biblioteca de operadores

> executa

> sai

Importa esclarecer que os procedimentos que devem ser seguidos são apresentados pelo sistema nos quadros da Fig. 2. O quadro 1 apresenta os comandos básicos para a execução do protótipo, enquanto o quadro 2 mostra comandos que podem ajudar o usuário, por exemplo, verificar o *script*[2] de uma função, saber quantas funções estão definidas nos arquivos carregados etc.

[1] Suponha '>' o prompt do sistema-simulador.

[2] *Script* é o conjunto de equações que definem uma função.

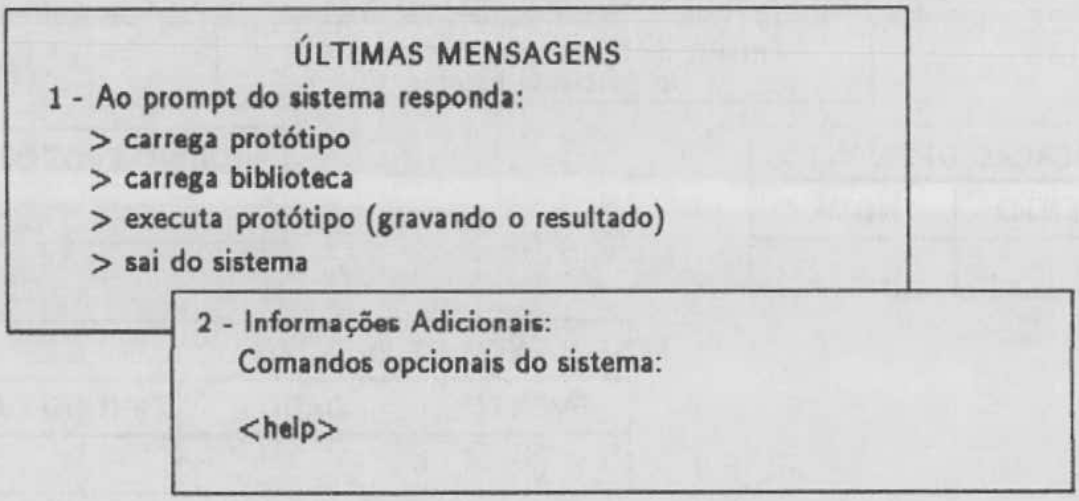

Figura 2: Procedimentos para execução de protótipos.

Finalmente, o item 4 se desenrola na tela da Fig. 3, onde é pedido o nome (do arquivo) da especificação simulada para que seja buscada a sequência (lista) que corresponde ao resultado da simulação.

A interface mostra que os dados de entrada não são passados durante a execução, mas apenas através de uma lista que corresponde à sequência de todos os eventos que podem ocorrer na interação do sistema em estudo com seu ambiente externo.

4 Estrutura e Comportamento do Sistema

Nesta seção, nosso objetivo é descrever como se comporta o sistema que visa ajudar o processo de desenvolvimento de projetos para sistemas distribuídos. Sabemos da importância, principalmente em projetos de grande porte, da fase de *especificação*. Da mesma forma, temos consciência do grande valor das simulações de diversas etapas de desenvolvimento para a validação de projetos. Portanto, o sistema que ora descrevemos

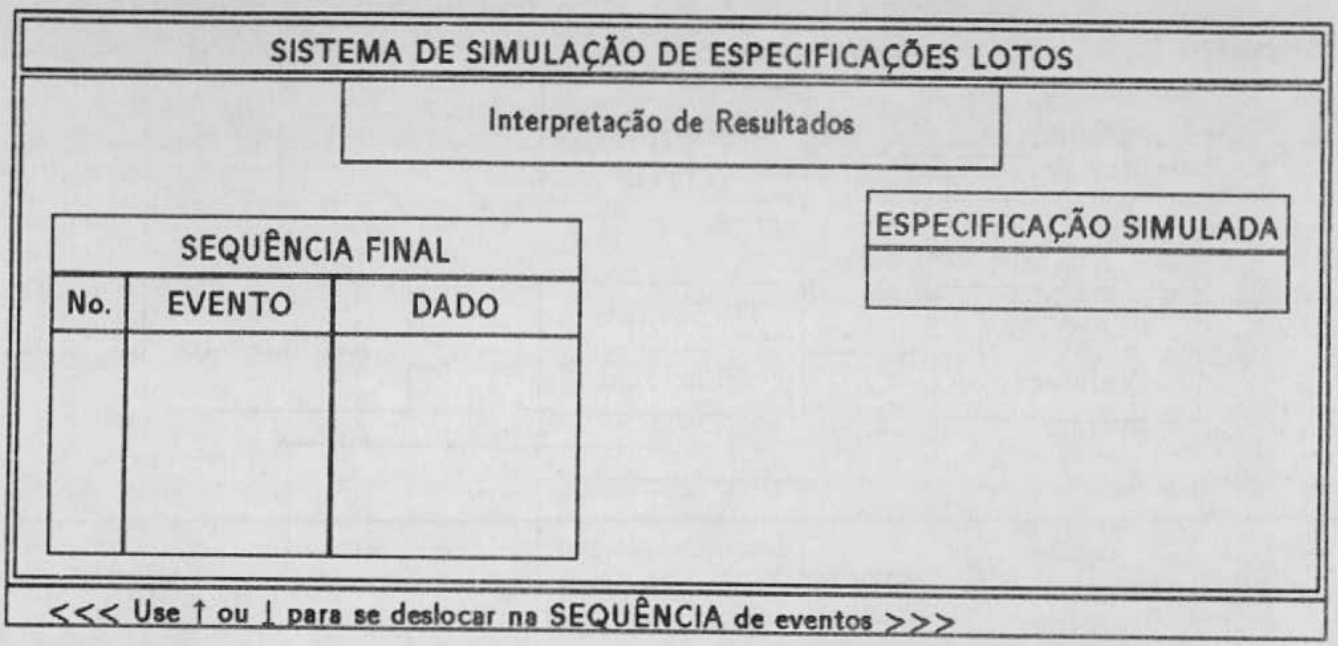

Figura 3: Tela de Interpretação de Resultados.

faz parte de um processo geral que tem por meta construir um ambiente de desenvolvimento de sistemas distribuídos [Cun81,Cun83,FCM89,Fer89,Mou89]. A Fig. 4 mostra que um projetista, de posse de uma especificação LOTOS, gera um protótipo funcional, que é simulado/validado e, enquanto o sistema especificado não possuir um nível de detalhamento que justifique sua implementação, ele será sucessivamente refinado até que restem apenas detalhes que dependam de implementação.

O sistema desenvolvido destaca três pontos fundamentais que determinam o seu comportamento:

- **A entrada de dados**: qual deve ser o formato dos dados que ativarão o sistema?
- **Os protótipos**: devem contar com os operadores implementados para que a construção dos protótipos seja capaz de torná-los equivalentes às suas descrições em LOTOS;

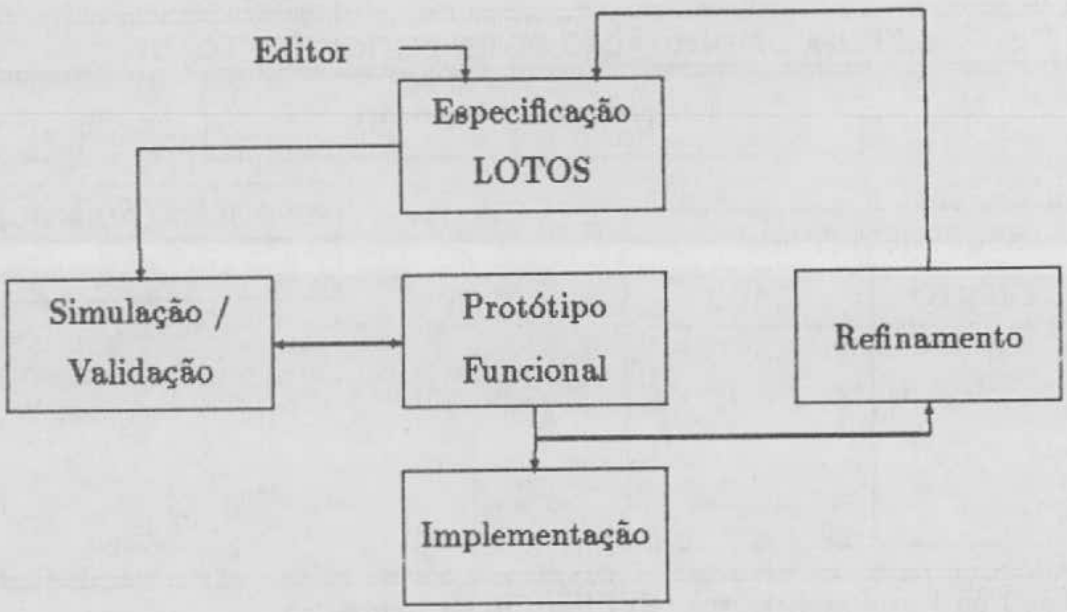

Figura 4: Ambiente de Desenvolvimento.

- **A saída**: como deve ser estruturada a apresentação do resultado da simulação?

Vamos agora, detalhar cada um dos pontos para que possamos compreender este nosso simulador.

4.1 Entrada de Dados: A Ativação do Sistema

Aqui tratamos da maneira como o usuário deve se relacionar inicialmente com o sistema, ou seja, como deve ser feita a ativação do processo de simulação. A ativação se dá através da entrada dos dados a serem manipulados pelo sistema. Os dados serão aqueles que influenciarão nos caminhos a serem seguidos na simulação do protótipo. Portanto, os dados serão os eventos ofertados pelo ambiente que engloba o sistema especificado.

Os *eventos* são unidades de sincronização, e são atômicos no sentido de ocorrerem instantaneamente. Os *processos*, que realizam ações (eventos), são ativados através de chamadas ou de instanciações, que ocorrem em alguma *expressão de comportamento*. Há

processos que realizam ações simplesmente por serem chamados. No entanto, há os que também necessitam saber quais os eventos ofertados pelo ambiente no qual se encontram para prosseguirem corretamente em sua execução. Por exemplo, o processo

```
process P[a,b,c] :=
        a ; c ; stop
        []
        b ; c ; stop
endproc
```

computa a expressão "a ; c ; **stop**" se o evento ofertado pelo ambiente for "a". Por outro lado, se o evento ofertado for "b", então a expressão "b ; c ; **stop**" será realizada.

De acordo com o que acabamos de observar, a simulação será realizada com o usuário fazendo o papel do "ambiente" que oferta os eventos que estabelecem a sequência de ações que representam o comportamento do sistema especificado. Assim, é o usuário que elabora a série dos eventos ativadores da especificação a ser simulada (cenários de testes). Tais eventos serão dispostos em uma lista finita, visto que não interessa o comportamento infinito em uma simulação, embora isto seja possível em linguagens funcionais não-estritas[3], como KRC[Tur82] e Miranda[Tur85], entre outras. Aproveitando esta decisão, determinamos que os processos recursivos – de comportamento infinito – contenham uma equação inicial prevendo sua terminação quando a lista que representa os eventos ofertados pelo ambiente se esgotar. Sendo assim, o processo

```
process buffer[in,out] :=
        in ?x:data;
        out !x;
        buffer[in,out]
```

[3]Linguagens que permitem o tratamento de objetos parcialmente definidos, como *listas infinitas*, por exemplo.

endproc

seria adaptado às condições impostas para a simulação do seguinte modo:

process buffer-T[in,out] :=
in ?x:data;
out !x;
buffer-T[in,out]
[]
in ?y:eot;
exit
endproc

onde "eot" indica uma mensagem de fim de transmissão (*end-of-transmission*), que funcionalmente corresponde à lista vazia ([]). O protótipo funcional equivalente ao processo acima seria

```
buffer_T [ind,outd] [] = []                                    (1)
buffer_T [ind,outd] ([e1,(!,x,data,ftrue)]:e2:lev)             (2)
                          = [ind,(?,x,data,ftrue)] :
                            [outd,(!,x,data,ftrue)] :
                            buffer_T [ind,outd] lev
```

O exemplo anterior considera que o ambiente oferta eventos (e1 e e2) de um atributo, por exemplo (!,x,data,ftrue), agrupados em uma lista (lev), que é consumida até se esvaziar (eq.1); [e1,(!,x,data,ftrue)] é a cabeça da lista e lev é a cauda, e o consumo da lista é feito através de recursão na cauda (eq.2)[4]. Como os eventos ofertados pelo ambiente devem ser os eventos dos processos que precisam ser ativados,

[4] "Cabeça" é o primeiro elemento corrente de uma lista, enquanto a "cauda" contém os elementos restantes da lista.

neste caso específico "e1" deve ser ind e "e2", outd. Note que e2 serve, neste caso, apenas para que seja checado o número de eventos produzidos pelo "ambiente" com os eventos gerados por buffer_T. Assim, não é necessário considerar sua forma. Uma discussão mais aprofundada sobre o modelo funcional desenvolvido pode ser encontrada em [Fer89], onde apresentamos a forma funcional de eventos.

Assim sendo, a ativação do sistema ocorre através da sequência (lista) de eventos "ofertados" pelo "ambiente" elaborado pelo usuário, de acordo com o caminho que ele deseja observar na simulação da especificação.

4.2 Considerações Finais sobre Entrada e Saída dos Dados

A sequência de entrada, que ativa o sistema, deve ser elaborada considerando como ela deve ser tratada em cada ponto do sistema ou de cada processo. A lista é consumida em cada um dos pontos, e portanto, deve ser gerada para que em um ponto específico o elemento que esteja na cabeça da lista, ou seja, o evento ofertado "naquele instante" pelo ambiente externo, que interage com o sistema, seja o evento, ou um dos eventos, que realmente podem realizar esta interação. Por exemplo, se um processo necessita de um evento 'a' para iniciar sua execução e, no momento em que recebe a lista que representa o fluxo de informações do ambiente com o qual ele interage, esteja na cabeça da lista um evento 'b', será criado um impasse na computação do resultado. Portanto, o usuário deve ter consciência sobre quais caminhos deseja percorrer dentro do sistema para projetar a lista de entrada e comparar o resultado obtido com a sua expectativa. Conclui-se, por isso, que o *usuário é o ambiente externo* que interage com o sistema. O *resultado da interação* é, por conseguinte, a sequência produzida.

4.2.1 O Resultado da Simulação

O significado da lista de eventos de saída é que ela corresponde à ordem na qual os eventos ocorreram, relatando a *história* da simulação para cada ambiente externo

imaginado pelo usuário. A saída dá condições para que sejam comparados os resultados com o que de fato deveria ocorrer no sistema. Portanto, a simulação é uma forma de validar um sistema especificado, colocando-se para ele as interações que podem ocorrer para verificar se os resultados estão de acordo com o que foi pensado durante a concepção do sistema.

A apresentação do resultado se dá, também, através de uma lista dos eventos produzidos pelo sistema durante a sua interação com o ambiente. Assim sendo,

- a *lista de entrada* representa o ambiente externo;
- a *lista de saída* corresponde à sequência dos eventos que representam a interação do sistema com o ambiente, significando o comportamento observável da parte dinâmica do sistema.

O comportamento "observável" pode incluir também ações não-observáveis, se for usado o modelo em que estas ações são inteiros negativos. Portanto, pode-se assistir todo o comportamento *especificado* do sistema. Observe, então, que o modelo é flexível no aspecto de permitir que o usuário decida "observar" as ações não-observáveis como inteiros negativos ou fazer com que não apareçam na lista que representa o resultado final.

4.3 Um Exemplo

Todo o processo necessário para a simulação de especificações LOTOS se encontra em [Fer89], onde propomos uma metodologia para a geração dos protótipos, considerando os operadores funcionais desenvolvidos, correspondentes aos construtores LOTOS, a estruturação dos dados de entrada e o modo como devem ser tratados pelos protótipos, além da forma como os resultados devem ser interpretados. Assim sendo, vamos mostrar como um protótipo simples é executado para compararmos os resultados com nossas expectativas, lembrando que é na tela da Fig. 3 que os resultados são realmente exibidos.

Vamos considerar um processo que realiza uma ligação telefônica, envolvendo uma fase de chamada, que a termina quando a ligação é atendida (processo 'Chama'), seguida ('>>') da fase de conversação (processo 'Conversa'), que pode ser interrompida ('[>') a qualquer instante (processo 'Desliga'):

```
process Ligacao-Telefonica[disca,atende,nao-atende,fala,cancela] :=
        Chama[disca,atende,nao-atende]
        >>
        (Conversa[fala] [> Desliga[cancela])
where
        process Chama[disca,atende,nao-atende] :=
                disca; (atende; exit
                        [] nao-atende; Chama[disca,atende,nao-atende]
        endproc
        process Conversa[fala] :=
                fala; Conversa[fala]
        endproc
        process Desliga[cancela] :=
                cancela; stop
        endproc
endproc
```

Supondo que após um processo de tradução [Fer89], tenhamos chegado ao seguinte protótipo funcional:

```
Ligacao_Telefonica [disca,atende,nao_atende,fala,cancela] lev =
   SEQ (Chama [disca,atende,nao_atende] lev)
       (INT (Conversa [fala]) (Desliga [cancela]))  (* 'lev' eh
                                                       a lista
```

```
                                                    de eventos
                                                    de entrada
                                                  *)
where
   Chama [disca,atende,nao_atende] [] = []                    (1)
   Chama [disca,atende,nao_atende] (e1:e2:lev)
            = [disca]:[atende]:[exit]:lev
                                    , hd e2 = atende           (2)
            = [disca]:[nao_atende]:
              Chama [disca,atende,nao_atende]
                                    , hd e2 = nao_atende       (3)

   Conversa [fala]      [] = []                               (1)
   Conversa [fala] (e:lev) = [fala]:Conversa[fala]
                                    , hd e = fala              (2)
                           = []                                (3)

   Desliga [cancela]      [] = []                             (1)
   Desliga [cancela] (e:lev) = [cancela]:[]
                                    , hd e = cancela           (2)
                             = Desliga [cancela] lev           (3)
```

Os comportamentos dos operadores SEQ e INT equivalem aos de '>>' e '[>' e podem ser verificados em [Fer89], assim como o modo de suas aplicações. Observe que na equação (2) de Chama, em seguida a [exit] é incorporada a cauda da lista de entrada para que seja passada por SEQ ao seu segundo operando, que neste caso é INT (Conversa [fala]) (Desliga [cancela]). (Veja na execução a seguir.) Vamos considerar que os eventos ofertados pelo ambiente estejam dispostos na lista

```
[[disca],[nao_atende],[disca],[atende],[fala],[fala],[fala],[cancela]]
```

para observarmos a execução do protótipo. Antes disso, vamos descrever nossa expectativa em relação ao resultado que pode ser gerado a partir desta lista:

- Inicialmente, espera-se que sejam feitas duas tentativas de estabelecimento da ligação para que se inicie uma conversação;
- Após atendida a ligação, deve-se esperar que ocorram três momentos de conversação, representados pelas três ocorrências do evento 'fala', que é interrompida pelo aparecimento do evento 'cancela'.

Vamos, então, partir para a execução, propriamente dita, do protótipo, que deverá receber como parâmetro a lista de entrada acima, esclarecendo que o símbolo '⇒' representa o cálculo de uma expressão:

```
Ligacao_Telefonica [disca,atende,nao_atende,fala,cancela]
[[disca],[nao_atende],[disca],[atende],[fala],[fala],[fala],[cancela]]
⇒ SEQ (Chama [disca,atende,nao_atende]
[[disca],[nao_atende],[disca],[atende],[fala],[fala],[fala],[cancela]]
(INT (Conversa [fala]) (Desliga [cancela]))
(* a lista de entrada é passada para a função Chama que passará a executar sobre ela *)
⇒ SEQ ([disca]:[nao_atende]:Chama [disca,atende,nao_atende]
[[disca],[atende],[fala],[fala],[fala],[cancela]] (INT (Conversa [fala])
(Desliga [cancela]))
(* pela equação (3) de Chama *)
⇒ SEQ ([disca]:[nao_atende]:[disca]:[atende]:[exit]:
[[fala],[fala],[fala],[cancela]] (INT (Conversa [fala]) (Desliga
[cancela]))
```

(* pela equação (2) de Chama *)

⇒ [disca]:SEQ ([nao_atende]:[disca]:[atende]:[exit]:
[[fala],[fala],[fala],[cancela]]) (INT (Conversa [fala]) (Desliga
[cancela]))

(* comportamento de SEQ *)

⇒ [disca]:[nao_atende]:SEQ ([disca]:[atende]:[exit]:
[[fala],[fala],[fala],[cancela]]) (INT (Conversa [fala]) (Desliga
[cancela]))

⇒ [disca]:[nao_atende]:[disca]:SEQ ([atende]:[exit]:
[[fala],[fala],[fala],[cancela]]) (INT (Conversa [fala]) (Desliga
[cancela]))

⇒ [disca]:[nao_atende]:[disca]:[atende]:SEQ ([exit]:
[[fala],[fala],[fala],[cancela]]) (INT (Conversa [fala]) (Desliga
[cancela]))

⇒ [disca]:[nao_atende]:[disca]:[atende]: (INT (Conversa [fala]) (Desliga
[cancela]) [[fala],[fala],[fala],[cancela]])

(* com o aparecimento de [exit] na cabeça da lista, SEQ termina sua execução,
passando a cauda da lista para seu segundo operando *)

⇒ [disca]:[nao_atende]:[disca]:[atende]: (INT (Conversa [fala]
[[fala],[fala],[fala],[cancela]]) (Desliga [cancela]
[[fala],[fala],[fala],[cancela]]) [[fala],[fala],[fala],[cancela]])

(* note que INT passa a lista de entrada para seus operandos para que sejam
produzidos os resultados da aplicação da cada um deles à lista, permanecendo também
com esta lista para posterior tratamento *)

⇒ [disca]:[nao_atende]:[disca]:[atende]: (INT ([fala]:Conversa [fala]
[[fala],[fala],[cancela]]) (Desliga [cancela] [[fala],[fala],[cancela]])
[[fala],[fala],[fala],[cancela]])

(* pelas eqs. (2) de **Conversa** e (3) de **Desliga**. Observe que enquanto não aparecer o evento **cancela**, **Desliga** simplesmente estará consumindo a lista sem produzir resultado *)

⇒ [disca]:[nao_atende]:[disca]:[atende]: (INT ([fala]:[fala]:Conversa [fala] [[fala],[cancela]]) (Desliga [cancela] [[fala],[cancela]]) [[fala],[fala],[fala],[cancela]])

⇒ [disca]:[nao_atende]:[disca]:[atende]: (INT ([fala]:[fala]:[fala]:Conversa [fala] [[cancela]]) (Desliga [cancela] [[cancela]]) [[fala],[fala],[fala],[cancela]])

⇒ [disca]:[nao_atende]:[disca]:[atende]: (INT ([fala]:[fala]:[fala]:[]) ([cancela]:[]) [[fala],[fala],[fala],[cancela]])

(* pelas eqs. (3) e (2) de **Conversa** e **Desliga**, respectivamente *)

⇒ [disca]:[nao_atende]:[disca]:[atende]: INT [[fala],[fala],[fala]] [[cancela]] [[fala],[fala],[fala],[cancela]]

(* Regra: "[a]:[b]:[] = [a,b]" *)

⇒ [disca]:[nao_atende]:[disca]:[atende]:[fala]: INT [[fala],[fala]] [[cancela]] [[fala],[fala],[cancela]]

(* comportamento de **INT** *)

⇒ [disca]:[nao_atende]:[disca]:[atende]:[fala]:[fala]: INT [[fala]] [[cancela]] [[fala],[cancela]]

⇒ [disca]:[nao_atende]:[disca]:[atende]:[fala]:[fala]:[fala]: INT [] [[cancela]] [[cancela]]

⇒ [disca]:[nao_atende]:[disca]:[atende]:[fala]:[fala]:[fala]: [[cancela]]

⇒ [disca]:[nao_atende]:[disca]:[atende]:[fala]:[fala]:[fala]: [cancela]:[]

(* Regra: "[[a]] = [a]:[]" *)

```
⇒ [[disca],[nao_atende],[disca],[atende],[fala],[fala],[fala]
,[cancela]]
(* Regra: "a:b:[] = [a,b]" *)
```

Observe que o resultado confirma as expectativas, e a lista seria então lida por um programa para ser exibida de forma mais legível na tela da Fig. 3, onde o usuário pode caminhar sobre a sequência para realizar estudos mais aprofundados. A entrada dos dados para a formação da lista que representa os eventos ofertados pelo ambiente é realizada na tela da Fig. 1 e a execução do protótipo deve seguir os passos descritos nos quadros da Fig. 2. O que mostramos neste exemplo foi como se dá a execução interna do protótipo funcional, e não sua simples aplicação à lista de entrada para a obtenção do resultado.

5 Conclusão

O sistema apresentado tem implementações realizadas em KRC[Tur82], tendo sido satisfatórios os resultados obtidos. Foram programados os operadores, possibilitando a geração da *biblioteca*, para que possam ser verificados os comportamentos de processos e especificações maiores implementados como funções. Os módulos de entrada e de exibição de resultados foram programados em Turbo Pascal[TuB87], devido à impossibilidade de montagem das telas em KRC, tornando o sistema (interface) híbrido apenas no tocante a esta tentativa de torná-lo fácil de se manipular.

Embora KRC tenha sido usada pelo fato de ser uma linguagem de programação puramente funcional e por sua simplicidade, acreditamos que qualquer outra linguagem funcional que possui conceitos como semântica não-estrita, alta ordem, entre outros, pode ser empregada para descrever protótipos de especificações em LOTOS.

Referências

[BoB87] Bolognesi, T. and Ed Brinksma: "Introduction to the ISO Specification Language LOTOS". *Computer Networks and ISDN Systems*, Vol. 14, 1987, pp. 25-59.

[Cun81] Cunha, P. R. F.: "Design and Analysis of Message Oriented Programming". PhD. Thesis, University of Waterloo, Canada, Sep. 1981.

[Cun83] Cunha, P. R. F.: "Técnicas para Especificação e Verificação Formal de Protocolos". *V Congresso Regional de Informática*, Olinda-PE, Maio 1983.

[EhM85] Ehrig, H. and B. Mahr: "Fundamentals of Algebraic Specification 1". Springer-Verlag, Berlin, 1985.

[FCM89] Ferraz, C., P. Cunha e S. Meira: "Simulação de Especificações LOTOS Usando Linguagens Funcionais". *III Simpósio Brasileiro de Engenharia de Software*, Recife, Out. 1989.

[Fer89] Ferraz, C. A. G.: "Um Estudo para o Desenvolvimento de Protótipos de Especificações LOTOS através de Programação Funcional". Dissertação de Mestrado, DI/UFPE, Recife, Dez. 1989.

[Hoa85] Hoare, C. A. R.: "Communicating Sequential Processes". Prentice-Hall Intl., 1985.

[ISO88] ISO: "Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour". IS8807, 1988.

[Lan64] Landin, P. J.: "The Machanical Evaluation of Expressions". *Computer Journal*, (6):308-320, 1964.

[McC60] McCarthy, J.: "Recursive Functions of Symbolic Expressions and their Computation by Machine". *Communications of ACM*, Vol. 3, No. 4, 1960.

[Mil80] Milner, R.: "A Calculus of Communicating Systems". Springer-Verlag, LNCS 92, Berlin, 1980.

[Mou89] Moura, M. T.: "Desenvolvimento Estruturado de Especificações em LOTOS". Dissertação de Mestrado, DI/UFPE, Recife, Dez. 1989.

[TuB87] Turbo Pascal - *Owner's Handbook*. Borland International, Inc., 1987.

[Tur82] Turner, D.: "Recursion Equations as a Programming Language". In *Functional Programming and its Aplications*. Cambridge University Press, Cambridge, 1982.

[Tur85] Turner, D.: "Miranda: A non-Strict Functional Language with Polymorphic Types". LNCS 201, Springer-Verlag, Sep. 1985.