GERAÇÃO AUTOMÁTICA DE ESPECIFICAÇÕES DE IMPLEMENTAÇÃO A PARTIR DE ESPECIFICAÇÕES ABSTRATAS (*)

Eílson Barbosa Medeiros
PROMON Eletrônica Ltda
Rod. SP 340 - Km 118,5, Campinas, SP
13081 - Campinas, SP.

Mauricio Ferreira Magalhães
Depto. de Engenharia da Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica (FEE) - UNICAMP
Cx. Postal 6101 - 13081, Campinas, SP.

Wanderley Lopes de Souza (**)
Departamento de Sistemas e Computação (DSC)
Centro de Ciências e Tecnologia (CCT)
Universidade Federal da Paraíba (UFPb)
Caixa Postal 10106, 58100 Campina Grande (Pb)

SUMÁRIO

Neste artigo é descrita uma metodologia, para o projeto de sistemas, baseada em duas linguagens de especificação: *LOTOS* e *STER*. Essas linguagens possuem diferentes níveis de abstração e, de acordo com as suas características, são empregadas em diferentes fases de uma trajetória de projeto. Uma tradução automática de especificações *LOTOS* em especificações *STER* é proposta e a execução de especificações *STER* é descrita. O objetivo principal deste trabalho é demonstrar como diferentes linguagens podem ser combinadas, a fim de dar suporte ao projeto de sistemas, sem exigir novas características para essas linguagens.

ABSTRACT

The paper describes a methodology for system design based on the use of two specification languages: *LOTOS* and *STER*. These languages have different abstraction levels and, according to their characteristics, they are used in different phases of a design trajectory. An automatic translation from *LOTOS* specifications to *STER* specifications is proposed and the execution of *STER* specifications is described. The main goal of this work is to demonstrate how different languages can be combined to support system design without demanding new features for these languages.

Palavras chave : Projeto, Especificação, Técnicas de Descrição Formal, *LOTOS*, *STER*, Tempo Real, Prototipagem

(*) realizado com auxílio do CNPq e da CAPES
(**) atualmente Professor Visitante junto ao DCA/FEE/UNICAMP.

1. Introdução

Algumas linguagens de descrição formal, atualmente disponíveis, possuem poder de abstração, expressão e análise, que permitem ao projetista produzir especificações claras e concisas de sistemas.

Language of Temporal Ordering Specification (*LOTOS*) [ISO 88] é uma linguagem de descrição formal padrão, desenvolvida pela International Organization for Standardization (ISO) para a especificação formal de sistemas distribuídos e protocolos de comunicação, sobretudo os relativos ao modelo Open Systems Interconnection [ISO 83]. Entretanto, *LOTOS* tem demonstrado ser útil também para a descrição de outros tipos de sistemas de engenharia [BIBI 86, FaLo 89].

Em *LOTOS*, um sistema é especificado definindo-se a ordem temporal das interações trocadas entre o sistema e o seu ambiente. *LOTOS* possui duas partes distintas: uma parte de controle, baseada em álgebras de processos [Miln 80, Hoar 85], e uma parte de dados, baseada em tipos abstratos de dados [EhMa 85]. Portanto, *LOTOS* pode ser utilizada para descrever sistemas em um estilo orientado para processos, em um estilo orientado para dados ou em algum tipo de estilo misto. Essa característica é muito útil aos projetistas, mas torna a linguagem complexa, uma vez que *LOTOS* incorpora dois tipos diferentes de álgebra.

Se for necessário descrever aspectos de um sistema que não podem ser especificados em *LOTOS* padrão e tais aspectos estão mais relacionados à fase de implementação, o que é melhor: estender *LOTOS*, aumentando a sua complexidade, ou utilizar *LOTOS* nas primeiras fases do projeto do sistema e, posteriormente, utilizar outra linguagem nas fases posteriores ?

Este artigo aborda a segunda solução, através de um exemplo no contexto de sistemas em tempo real, e está organizado da seguinte forma: a seção 2 descreve as fases principais de uma trajetória de projeto; a seção 3 apresenta o exemplo e sua especificação abstrata em *LOTOS*; a seção 4 descreve, sumariamente, a linguagem de especificação de implementação, denominada *STER*, utilizada neste tipo de aplicaçao; a seção 5 discute a abordagem adotada na tradução automática de especificações *LOTOS* para a linguagem de especificação de implementação *STER*; a seção 6 descreve a estrutura da especificação final desse exemplo; a seção 7 apresenta algumas conclusões.

2. Fases Principais de uma Trajetória de Projeto

De acordo com a trajetória de projeto apresentada em [Pilo 90], os requisitos do usuário são formalizados numa especificação inicial. Uma especificação final pode ser obtida, a partir da inicial, através de um processo baseado em refinamentos sucessivos. A cada passo, um conjunto de decisões de projeto é tomado e uma especificação mais detalhada é produzida. Dependendo da aplicação e do ambiente onde será realizada a implementação, mais de uma linguagem de especificação pode ser necessária para cobrir inteiramente a trajetória de projeto. Na Figura 1 são consideradas as fases principais de uma trajetória de projeto. As especificações intermediárias entre as especificações inicial e final são omitidas, já que duas especificações são suficientes para os propósitos deste artigo. Essa trajetória básica de projeto pode ser utilizada em aplicações simples.

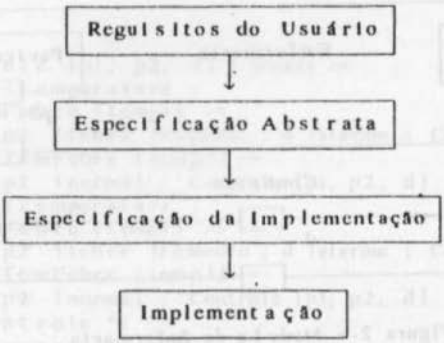

Figura 1 - Fases principais de uma trajetória de projeto

Na primeira fase principal, os requisitos do usuário são formalizados numa especificação inicial, utilizando-se uma linguagem de especificação (e.g., *LOTOS*). Uma vez que essa especificação não deve conter detalhes irrelevantes, ou seja, detalhes relacionados a fases posteriores do processo de especificação, ela é chamada de especificação abstrata.

Na segunda fase principal, a especificação abstrata é detalhada e orientada para uma implementação particular. Se há aspectos importantes, que não podem ser representados na linguagem de especificação adotada (e.g., tempo em *LOTOS*), a especificação da implementação pode ser formalizada em outra linguagem. Nessa fase, uma abordagem para a tradução pode ser muito útil.

Na última fase principal, uma implementação é obtida a partir da especificação da implementação, o que pode ser realizado manualmente ou utilizando-se ferramentas (e.g., um compilador ou um ambiente completo para a linguagem de especificação de implementação).

3. Exemplo e sua Especificação Abstrata em LOTOS

O exemplo é subtraído de [Mede 91], onde um monitoramento automático de leitos de pacientes é descrito informalmente. As funções vitais (e.g., temperatura) dos pacientes são captadas por sensores. Os valores dessas funções são periodicamente enviados a um controle, que os compara às condições normais. Se algo anormal é detectado (e.g., um paciente tem febre), o atuador correspondente é ativado e o médico é chamado. Somente dois pacientes são considerados nesse exemplo. Cada paciente tem um sensor de temperatura e um atuador, que é responsável pela administração de uma dose de medicamento ao paciente.

A enfermaria é modelada por três recursos, que são representados por três caixas pretas: *Paciente1*, *Paciente2* e *Controle*. O *Controle* interage com *Paciente1* e *Paciente2* através das portas *p1* e *p2*, respectivamente, e interage com o ambiente da *Enfermaria* através da porta *d* (para chamar o doutor). Esse modelo é apresentado na Figura 2.

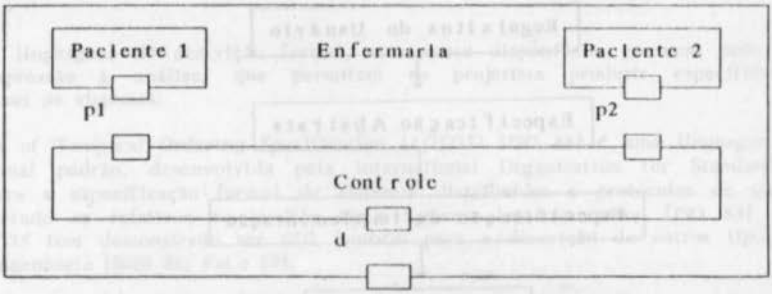

Figura 2 - Modelo da Enfermaria

O comportamento da enfermaria pode ser visto como uma composição desses três recursos. O *Paciente1* pode ser composto, sem sincronização, com o *Paciente2* (em *LOTOS*, isso é representado pelo operador de entrelaçamento puro |||). O resultado dessa composição pode ser combinado, com sincronização na porta *p1* para o *Paciente1* e na porta *p2* para o *Paciente2* com o recurso *Controle* (em *LOTOS*, isso é representado pelo operador de paralelismo genérico |[p1, p2]|). A especificação abstrata *LOTOS* da enfermaria é apresentada na Figura 3. Para o leitor que não está familiarizado com *LOTOS* sugere-se [BoBr 89].

Os seguintes Tipos Abstratos de Dados são considerados disponíveis na especificação da Figura 3:

- *temperatura*: constantes *normal* e *febre*; operações *TemFebre* e *NaoTemFebre*;
- *medicina*: constante *remedio*
- *ruido*: constante *alarme*

```
specification Enfermaria [d] : noexit
(* TADs *)
behaviour
  hide p1, p2 in
     (Paciente1 [p1] |||Paciente2 [p2])
     |[p1, p2]| Controle [p1, p2, d]
where
  process Paciente1 [p1] : noexit :=
  i ; p1 !temp ; (    p1 !febre ?remedio:medicina ; Paciente1 [p1]
                 [] p1 !normal ; Paciente1 [p1]   )
  endproc (* Paciente1 *)

  process Paciente2 [p2] : noexit :=
  i ; p2 !temp ; (    p2 !febre ?remedio:medicina ; Paciente2 [p2]
                 [] p2 !normal ; Paciente2 [p2]   )
  endproc (* Paciente2 *)
```

```
process Controle [p1, p2, d] : noexit :=
    p1 ?temp:temperatura ;
    (   [TemFebre (temp)] ->
            p1 !febre !remedio ; d !alarme ; Controle [p1, p2, d]
    [] [NaoTemFebre (temp)] ->
            p1 !normal ; Controle [p1, p2, d]    )
[] p2 ?temp:temperatura ;
    (   [TemFebre (temp)] ->
            p2 !febre !remedio ; d !alarme ; Controle [p1, p2, d]
    [] [NaoTemFebre (temp)] ->
            p2 !normal ; Controle [p1, p2, d]    )
endproc (* Controle *)
endspec (* Enfermaria *)
```

Figura 3 - Especificação abstrata *LOTOS* da enfermaria

As temperaturas dos pacientes são periodicamente enviadas ao controle. Uma vez que não é possível especificar explicitamente tempo em *LOTOS*, esse período é abstraído e representado através de um evento interno *i*. A representação concreta desse tempo é deixada para a especificação da implementação.

4. Linguagem de Especificação de Implementação

Sistema de Programação em Tempo Real (*STER*) [DTIA 88] é um ambiente de programação baseado no modelo *CONIC* [SIKr 86]. *STER* suporta duas linguagens: a Linguagem de Programação de Módulos (*LPM*), que permite a programação individual de componentes de software, e a Linguagem de Configuração de Módulos (*LCM*), que permite a construção de sistemas com esses componentes.

LPM é baseada em Pascal [IBM 84] e foi estendida para suportar modularidade e a comunicação por mensagens. A linguagem permite a definição de um tipo módulo. Módulos trocam mensagens e executam funções particulares. Todas as referências são a objetos locais e não há referências diretas a outros módulos. Consequentemente, nenhum tipo de informação sobre configuração é embutida em *LPM*.

As interfaces dos módulos são definidas através de portas tipificadas. Uma troca de mensagem pode ser iniciada por um módulo, através de uma de suas portas de saída (exitport), e essa mensagem pode ser recebida por outro módulo, através de uma de suas portas de entrada (entryport), desde que essa porta seja compatível com a porta do módulo emissor. A figura 4 mostra a estrutura das especificações de dois módulos em LPM.

LCM fornece os mecanismos para a configuração dos tipos módulo. "Instâncias" de módulos podem ser criadas e elos de comunicação, ligações entre portas de diferentes instâncias, podem ser estabelecidos. LCM fornece também os mecanismos para a configuração de sistemas distribuídos. A Figura 5 mostra uma possível configuração para as especificações dos módulos apresentadas na Figura 4 e também mostra uma representação para essa configuração.

```
module A ;
use FileDef.DEF ;
exitport
    p1: type1 ;
    p2: type2 reply type3 ;
entryport
    p3: type4 ;
message
  m1: type1 ;
  m2: type2 ;
  m3: type3 ;
  m4: type4 ;
(* procedimentos/funções *)
begin_module
   loop
     send m1 to p1 ;
     ......
     send m2 to p2 wait m3 ;
     ......
     receive m4 from p3 ;
     ......
   end_loop
end_module.
```

```
module B ;
use FileDef.DEF ;
exitport
    p1: type4 ;
entryport
    p2: type1 queue 2 ;
    p3, p4: type2 reply type3
message
    m1: type1 ;
    m2, m4: type2 ;
    m3, m5: type3 ;
    m6: type4 ;
 (* procedimentos/funções *)
begin_module
  loop
     receive m1 from p2 ;
     receive m1 from p2 ;
     ......
     receive m2 from p3 reply m3 ;
     receive m4 from p4 reply m5 ;
     ......
     send m6 to p1 ;
     ......
  end_loop
end_module.
```

Figura 4 - Especificações dos módulos A e B em *LPM*

```
Configuration exemplo ;
instance
  A1, A2: A;
  B1: B;
create
  A1 ; A2 ; B1 ;
link
A1.p1 to B1.p2 ; A2.p1 to B1.p2 ;
A1.p2 to B1.p3 ; A2.p2 to B1.p4 ;
B1.p1 to A1.p3 ; B1.p1 to A2.p3 ;
end_config
```

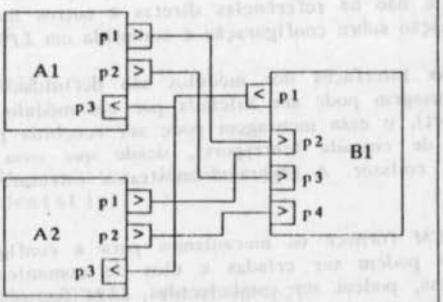

Figura 5 - Exemplo de configuração e sua representação

A execução e a comunicação entre as instâncias dos módulos (tarefas) são gerenciadas por um núcleo preemptivo, cuja estratégia de escalonamento é baseada nas prioridades dos módulos. O núcleo também fornece funções de tempo real, tais como, mudança de prioridade, atraso, temporização, etc.

Para facilitar a compreensão do texto, nas próximas seções a linguagem de especificação de implementação será referenciada simplesmente por *STER*.

5. Abordagem para a tradução LOTOS -> STER

Essa abordagem deve preservar, na especificação da implementação, as mesmas propriedades apresentadas na especificação abstrata. Uma vez que as semânticas de *LOTOS* e *STER* são muito diferentes, um mecanismo para uma tradução consistente deve assegurar a compatibilidade entre essas especificações. A abordagem é baseada em dois princípios:

(a) cada processo *LOTOS* é mapeado em um módulo de aplicação *STER*;

(b) um módulo adicional *STER*, denominado Interpretador Semântico (IS), é incluído para assegurar a compatibilidade semântica.

Para o modelo da enfermaria, o mapeamento é mostrado na Figura 6.

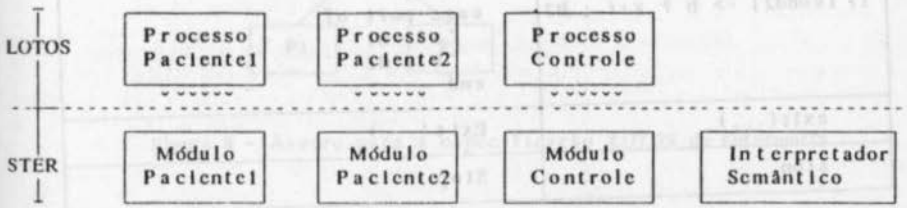

Figura 6 Mapeamento para o modelo da enfermaria

Esse mapeamento é realizado automaticamente, sendo que as construções *LOTOS* são traduzidas em chamadas a procedimentos, de acordo com as regras apresentadas na Figura 7.

A compatibilidade de comportamento entre uma especificação *LOTOS* e a correspondente especificação *STER* é assegurada pela:

(a) execução do modelo associado ao módulo IS, que permite uma interpretação para a seleção dos eventos oferecidos pelos processos *LOTOS*;

(b) execução do próprio módulo IS, que permite uma interpretação para o mecanismo de comunicação "multi-way rendez-vous" da linguagem *LOTOS*.

Construções LOTOS	Chamadas a procedimentos STER
a ! v	L(a,!,v,type(v)) ; sinc
b ? x:t	L(b,?,x,t) ; sinc
b ? x:t[c]	L(b,?,x,t,c) ; sinc
a !v ?x:t	L(a,!,v,type(v),?,x,t) ; sinc
a ! v ; b ? x:t	L(a,!,v,type(v)) ; sinc ; L(b,?,x,t) ; sinc
a ! v ; B1 [] b ? x:t ; B2	L(a,!,v,type(v)) ; L(b,?,x,t) ; sinc ; case port of 'a': B1 'b': B2 end
[cond1] -> a ! v ; B1 [] [cond2] -> b ? x:t ; B2	if cond1 then L(a,!,v,type(v)) if cond2 then L(b,?,x,t) sinc ; case port of 'a': B1 'b': B2 end
exit(...)	Exit(...)
stop	Stop

Figura 7 Regras para a tradução *LOTOS -> STER*

5.1 Modelo de Execução

A função principal do módulo IS é seleccionar uma interação para execução, a partir dos eventos oferecidos pelos outros módulos *STER*. O modelo de execução de IS foi inspirado em [WuBo 89], sendo o mesmo baseado numa árvore de atividades com atributos.

Durante a execução de uma especificação, a árvore de atividades reflete as relações existentes entre as expressões de comportamento *LOTOS*. Na medida em que o comportamento da especificação muda dinamicamente, a árvore cresce e é atualizada. As funções relacionadas aos atributos controlam a execução das interações e o crescimento e a atualização da árvore de atividades [Mede 91].

Crescimento significa que o sistema expande nós não terminais a fim de encontrar nós terminais com possíveis interações. Atualização significa que, após uma interação por rendez-vous, o sistema corta aquelas sub-árvores que representam comportamentos alternativos que já não podem mais ocorrer e permite que alguns possíveis comportamentos tornem-se ativos.

A árvore de atividades para a especificação abstrata *LOTOS* da enfermaria (seção 3), que é composta pelos processos *Paciente1* (P1), *Paciente2* (P2) e *Controle* (Ct), é apresentada na Figura 8.

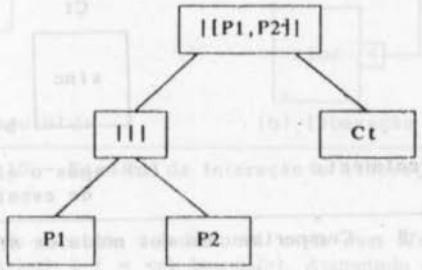

Figura 8 - Árvore para a especificação *LOTOS* da enfermaria

5.2 Execução do módulo IS

Numa especificação *STER* composta por módulos de aplicação e por um módulo IS, cada módulo de aplicação, capaz de participar de uma sincronização, oferece um evento a IS e, em seguida, é bloqueado.

O núcleo conduz a execução dos módulos *STER* com base em suas prioridades. Essas prioridades podem mudar dinamicamente. Na inicialização, os módulos de aplicação tem a mesma prioridade e IS tem a menor prioridade. Consequentemente, o núcleo conduz a execução dos módulos de aplicação executáveis até que todos estejam bloqueados.

Na execução da especificação *STER* da enfermaria, cada módulo *Paciente* oferece um evento a IS (a = <p1 ltemp> e b = <p2 ltemp>) e o módulo *Controle* oferece dois eventos a IS (c = <p1 ?temp: temperatura> e d = <p2 ?temp:temperatura>). Neste caso, os módulos da enfermaria tornam-se bloqueados e IS é executado. Esse comportamento é ilustrado na Figura 9.

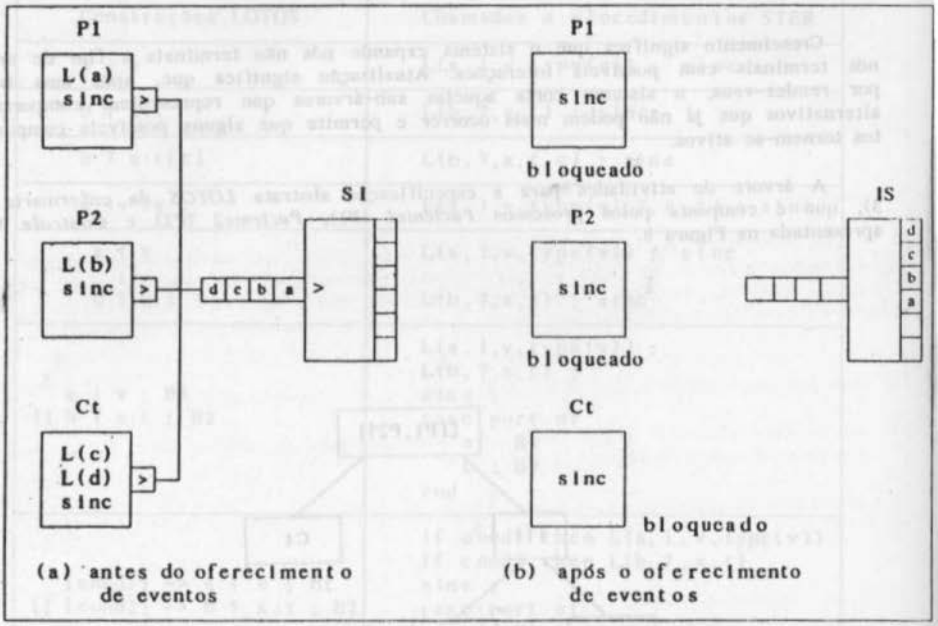

Figura 9 - Comportamento dos módulos *STER* da enfermaria

Após o bloqueio de todos os módulos de aplicação, o núcleo conduz a execução de IS. Esse módulo aumenta a sua prioridade a fim de torná-la superior às prioridades dos módulos de aplicação permitindo-o, desta forma, tratar a interação de modo atômico.

Baseado num conjunto de regras de avaliação, que são aplicadas à árvore de atividades da correspondente especificação *LOTOS*, IS resume as ofertas de eventos e escolhe, indeterministicamente, uma interação. Após essa escolha, IS envia as mensagens de liberação aos módulos de aplicação envolvidos na interação, habilitando-os a oferecerem novos eventos. Em seguida, IS muda a sua prioridade, a fim de torná-la inferior às prioridades dos módulos de aplicação permitindo, desta forma, a execução destes.

A Figura 10 ilustra um possível comportamento para a especificação *STER* da enfermaria. Nesse caso, a interação <a-c> foi escolhida e, depois de sua execução, os módulos P1 e Ct são liberados. O módulo P2 permanece bloqueado e sua oferta de evento é preservada por IS para futuras interações.

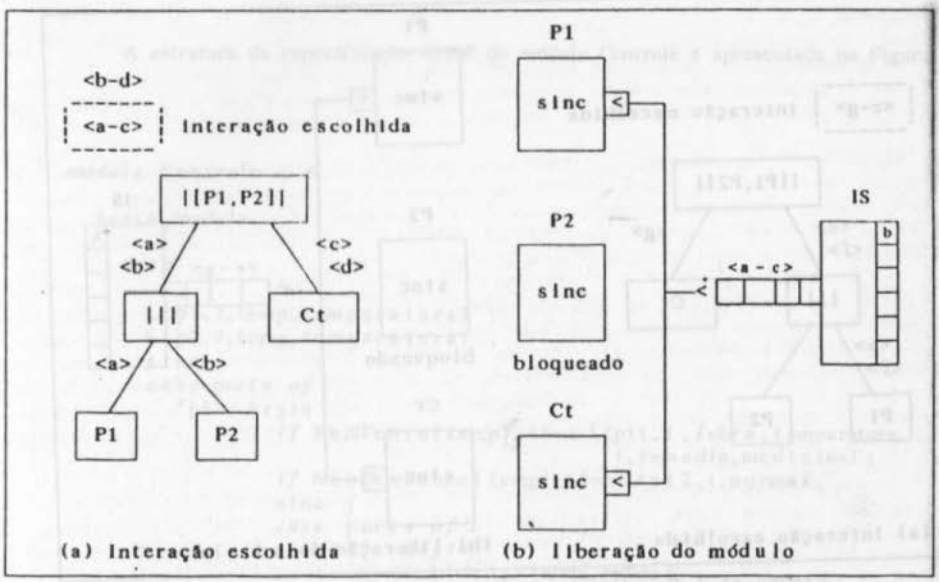

Figura 10 - Escolha da interação e liberação do módulo

Em um novo ciclo de execução o módulo P1 envia duas ofertas de eventos a IS (e = <p1 !febre ?remedio:medicina> e f = <p1 !normal>). Assumindo que o *Paciente1* tem febre (TemFebre(temp) = true), o módulo Ct envia a oferta de evento g = <p1 !febre !remedio> a IS. Novamente, os módulos da enfermaria tornam-se bloqueados e o módulo IS é executado. De acordo com a árvore de atividades da especificação *LOTOS* da enfermaria, a única interação possível é <e-g>, conforme indicado na figura 11. Após a execução do módulo IS, os módulos P1 e Ct são liberados. Novamente, para poder executar essas ações, IS aumenta e depois reduz a sua prioridade

A execução da especificação *STER* da enfermaria pode continuar até que esta volte ao seu estado inicial e, através de recursão, pode então iniciar um novo ciclo.

6. Tempo-Real e Estrutura da Especificação de Implementação

Os aspectos relacionados ao tempo são modelados indiretamente em *LOTOS* através do evento interno *i*. Em *STER*, esse evento pode ser traduzido nas cláusulas *delay* e *time-out*, que podem ser incluídas nos mecanismos *STER* de passagem de mensagens.

No exemplo da enfermaria, o evento interno *i* representa o intervalo de tempo entre as tomadas de temperatura de um paciente. Na realidade o intervalo seria melhor traduzido por *i* ; *i* , onde o primeiro *i* representa o início do intervalo e o segundo *i*, o término do intervalo. Uma representação mais concreta desse tempo é obtida nas especificações *STER* correspondentes aos módulos Paciente. A estrutura da especificação *STER* do módulo *Paciente1* é apresentada na Figura 12 (a estrutura da especificação do módulo *Paciente2* é similar).

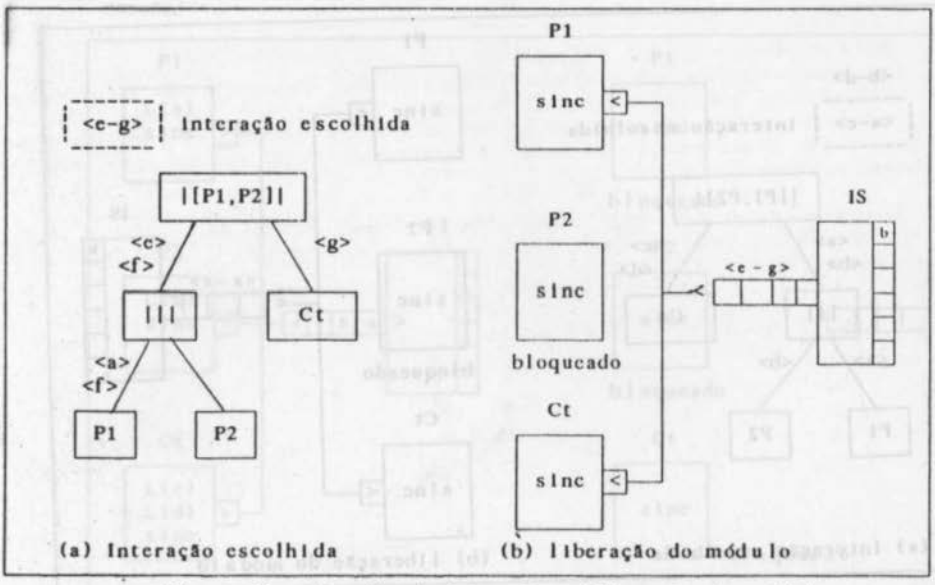

Figura 11 - Escolha da interação e liberação do módulo

```
module PacienteI ;
  ......
  begin_module
    ......
    loop
      ......
      delay (50)
      L(p1,1,temp,temperatura) ;
      sinc ;
      L(p11,1,febre,temperatura,?,remedio,medicina) ;
      L(p12,1,normal,temperatura) ;
      sinc;
      case porta of
        'p11': AutoExec ;
        'p12': AutoExec ;
      end
    end_loop
  end_module.
```

Figura 12 - Estrutura da especificação *STER* do módulo PacienteI

A estrutura da especificação *STER* do módulo Controle é apresentada na Figura 13.

```
module Controle ;
  ......
  begin_module
    ......
    loop
      ......
      L(p1,?,temp,temperatura) ;
      L(p2,?,temp,temperatura) ;
      sinc ;
      case porta of
        'p1': begin
                if TemFebre(temp) then L(p11,1,febre,temperatura,
                                         1,remedio,medicina) ;
                if NaoTemFebre(temp) then L(p12,1,normal,
                sinc ;
                case porta of
                  'p11': begin
                           L(d,1,alarme,ruido) ;
                           sinc ;
                           AutoExec
                         end ;
                  'p12': AutoExec
              end ;
        'p2': begin
                if TemFebre(temp) then L(p21,1,febre,temperatura,
                                         1,remedio,medicina) ;
                if NaoTemFebre(temp) then L(p22,1,normal,
                                            temperatura) ;
                sinc ;
                case porta of
                  'p21': begin
                           L(d,1,alarme,ruido) ;
                           sinc ;
                           AutoExec
                         end ;
                  'p22': AutoExec
              end;
    end_loop ;
  end_module.
```

Figura 13 - Estrutura da especificação *STER* do módulo Controle

7. Conclusão

A metodologia de projeto de sistemas, adotada neste artigo, apresenta uma característica fundamental: ela utiliza duas linguagens de especificação para cobrir toda uma trajetória de projeto, sem requerer extensões a essas linguagens.

Aspectos de um sistema, que são mais inerentes à implementação (e.g., tempo), podem ser abstraídos nas fases iniciais do projeto e uma especificação mais concreta de tais aspectos pode ser deixada para as fases posteriores. Dependendo da complexidade do sistema, várias especificações *LOTOS* e várias especificações *STER* podem ser produzidas até que a implementação final seja atingida.

Outro aspecto que pode ser deixado para as fases posteriores, é a importância das aplicações em sistemas de tempo real. Esse aspecto pode ser especificado em *STER* através de prioridades atribuídas aos módulos pelo próprio projetista. Por exemplo, na especificação da implementação da enfermaria, se os pacientes possuem diferentes doenças, um deles pode necessitar de um tratamento mais urgente. Nesse caso, prioridades poderiam ser atribuídas aos módulos Paciente e o módulo IS poderia escolher o módulo aplicação a ser executado de acordo com essas prioridades.

O mapeamento da especificação *LOTOS* de um sistema numa especificação *STER* é realizado automaticamente. Como resultado, um protótipo do sistema é produzido. O comportamento observável da especificação abstrata, obtido através da utilização de uma ferramenta de simulação tipo HIPPO [Eijk 88], pode ser comparado ao comportamento apresentado pela correspondente especificação de implementação, quando esta é executada no ambiente *STER*. Uma espécie de validação de projeto pode ser realizada desta forma.

8. Referências

[BiBl 86] F. Biemans, P. Blonk, "On the Formal Specification andVerification of CIM Architectures Using *LOTOS*", Computer in Industry 7, North-Holland, 1986, pp. 491-504.

[BoBr 89] T. Bolognesi, E. Brinksma, "Introduction to the ISO Specification Language *LOTOS*", The Formal Description Technique *LOTOS*, North-Holland, 1989, pp. 23-73.

[DITA 88] DTIA-001/88, "*STER* - Um Ambiente para Desenvolvimento de Software Tempo-Real", IA/DEI/CTI, Campinas (SP), 1988.

[EhMa 85] H.Ehrig, B. Mahr, "Fundamentals of Algebraic Specifi cation - 1", Springer-Verlag, 1985.

[Eijk 88] P. V. Eijk , "Software Tools for the Specification Language *LOTOS*", tese de doutorado, Twente University of Technology, Enschede (Holanda), 1988.

[FaLo 89] M. Faci, L. Logrippo, B. Stepien, "Formal Specifica tions of Telephone Systems in *LOTOS*", anais do Ninth IFIP International Symposium on Protocol Specification, Testing, and Verification, Enschede (Holanda), 06-09 junho, 1989.

[Hoar 85] C. A. R. Hoare,"Communicating Sequencial Process", Prentice- Hall Intl., 1985.

[IBM 84] IBM Corporation, "Pascal Compiler Language Reference - Version 2.00", 1984.

[ISO 83] ISO - Information Processing Systems- "Basic Reference Model for Open Systems Interconnection",IS7498,1983.

[ISO 88] ISO - Information Processing Systems - Open Systems Interconnection - "*LOTOS* - A Formal Description Technique Based on the Temporal Ordering of Observa tional Behaviour", IS 8807, 1988.

[Mede 91] E. B. Medeiros, "Prototipagem e Implementação de Especificações *LOTOS* Utilizando um Ambiente para De senvolvimento de Sistemas em Tempo Real", tese de mestrado, DCA/FEE/UNICAMP, Campinas (SP), 1991.

[Miln 80] R. Milner, "A Calculus of Communicating Systems", Lectures Notes in Computer Science, Vol. 92, Springer-Verlag, 1980.

[PiLo 90] L. F. Pires, W. Lopes de Souza, "Step-wise Refinement Design Example Using *LOTOS*", anais do Third IFIP International Conference on Formal Description Techniques", Madrid (Espanha), 05-08 novembero, 1990, pp. 289-306.

[SlKr 86] M. Sloman, J. Kramer, J. Magee, "The Conic Tool Kit for Building Distributed Systems", anais do 4o Simpósio Brasileiro de Redes de Computadores, Recife (Pe), 24-26 março, 1986, pp. 148-157.

[WuBo 89] C. Wu, G. v. Bochmann, "An Execution Model for *LOTOS* Specifications", relatório interno No 701, IRO/UdeM, Monreal, Canadá,1989.