Projeto de Sistemas Através de Refinamentos Sucessivos Utilizando a TDF LOTOS

Wanderley Lopes de Souza (+)

GRC/DSC/CCT
Universidade Federal da Paraiba
Av. Aprigio Veloso, 882
58100 Campina Grande (Pb)
Brasil

Luis Ferreira Pires (++)

Tele-Informatics Group
University of Twente
PO Box 217
7500 AE Enschede
the Netherlands

Sumário

A necessidade de uma metodologia para o projeto de sistemas distribuídos complexos é um consenso junto à comunidade científica na área. O uso de Técnicas de Descrição Formal (TDFs) em conjunto com uma série de conceitos caracterizam uma metodologia de projeto, que pode resultar em sistemas mais confiáveis e com menor custo de manutenção. Este artigo apresenta alguns conceitos e princípios de projeto, utilizados numa abordagem baseada em refinamentos sucessivos, que por sua vez é aplicada a um exemplo: um sistema de acesso por exclusão mútua. O objetivo principal deste trabalho é o de avaliar o uso de TDFs, em particular LOTOS, e o emprego de diferentes estilos de especificação durante o projeto sistemas distribuídos.

Abstract

The importance of a design methodology when using Formal Description Techniques is of general agreement into the scentific community in the area. In our paper we present some design principles in order to define a design methodology based on stepwise refinement. An example of a mutual exclusion access system is taken, on which the design methodology is applied, resulting in LOTOS specifications containing the design decisions. The objective of this work is to evaluate the use of FDTs, in special LOTOS, to support system design and consequently address the impact of specification styles when describing design decisions.

1. Introdução

Language of Temporal Ordering Specification (LOTOS) é uma das TDFs padronizada pela International Organization for Standardization (ISO) [ISO 88], para ser usada na especificação dos serviços e protocolos relativos ao modelo de referência Open Systems Interconnection (OSI). Entretanto, o uso de LOTOS não se restringe a este tipo de aplicação. LOTOS tem se mostrado útil para a descrição de sistemas que apresentam concorrência, distribuição e/ou sincronização.

(*) suportado parcialmente pelo CNPq (+) e Comissão da Comunidade Européia - CEC (++) no projeto ESPRIT II LOTOSPHERE (No 2304)

Uma vez que LOTOS tornou-se um padrão internacional, ganhando estabilidade, parece importante que o uso da linguagem seja explorado nos diversos campos passíveis de sua aplicação. No momento, o desenvolvimento de metodologias e de ferramentas, baseadas em LOTOS, deveria ser mais incentivado do que extensões e modificações da linguagem. A disponibilidade de metodologias e ferramentas é uma condição essencial para a difusão de LOTOS junto à comunidade acadêmica e para a sua utilização em escala industrial.

Todavia, o simples emprego de uma TDF não substitui uma das principais tarefa do projetista, que é a tomada das decisões de projeto. Uma TDF é apenas uma notação matemática, que permite ao projetista representar propriedades, do sistema que está sendo projetado, de acordo com um modelo formal.

Este artigo discute a utilização de LOTOS como ferramenta de suporte para o desenvolvimento de sistemas, através de um exemplo de porte médio não trivial. O objetivo é atingir uma especificação desse sistema exemplo, a partir da qual uma implementação possa ser derivada.

A seção 2. apresenta um sumário dos conceitos principais a serem considerados (e. g., refinamentos sucessivos e estilos de especificação) e relaciona este trabalho com outros trabalhos na área. A seção 3. apresenta o exemplo informalmente e sua especificação mais abstrata. O processo de refinamentos sucessivos é aplicado a essa especificação na seção 4. A seção 5. discute futuras decisões relacionadas a possíveis implementações e as conclusões são apresentadas na seção 6.

2. Princípios de Projeto

O objetivo final do projeto de um sistema é obter uma realização - uma "instância" concreta desse sistema - que atenda aos requisitos do seu usuário. No caso de sistemas distribuídos concorrentes complexos, o projeto é também uma tarefa complexa. Para enfrentar essa complexidade, o projeto deve ser realizado em passos, caracterizando uma trajetória de projeto.

Em cada passo da trajetória do projeto, uma versão mais refinada do projeto é elaborada. Alguns critérios qualitativos são definidos para avaliar projetos e a maneira na qual estes são formulados. Os estilos de especificação, além de guiar os projetistas na utilização eficiente dos elementos de uma linguagem, servem como uma medida fundamental da adequação das especificações em relação aos seus propósitos no contexto da trajetória de projeto.

2.1. Critérios Qualitativos de Projeto

Em [Viss 88] três critérios de projeto são apresentados:

- ortogonalidade: aspectos independentes devem permanecer independentes. Por exemplo, considerando a especificação de um serviço (e.g., serviços do modelo OSI), o projetista pode especificar restrições locais, para um ponto de acesso ao serviço, e remotas separadamente, usando processos LOTOS sincronizados;

- **generalidade** defina construções gerais que possam ser reutilizadas. Por exemplo, considere especificações de filas através de Tipos Abstratos de Dados (TAD). A especificação parametrizada, usando um elemento formal como elemento da fila, é preferível a múltiplas especificações de filas especializadas,

- **extensibilidade** não limite o que pode ser, no futuro, uma extensão da funcionalidade do sistema. Por exemplo, considerando a especificação de um protocolo na qual parte da funcionalidade do protocolo não é descrita, a especificação deve ser estruturada de tal forma que a inclusão da funcionalidade restante possa ocorrer sem grande necessidade de re-estruturação.

2.2. Trajetória de Projeto

Nos passos iniciais da trajetória de projeto, os requisitos do usuário devem ser analisados e formalizados (as vezes parcialmente) numa especificação inicial. Essa especificação inicial é a mais abstrata da trajetória de projeto e não deve conter detalhes irrelevantes (e.g., interfaces ou estruturas internas). Esses detalhes devem ser tratados em passos posteriores da trajetória de projeto, cobrindo os níveis mais baixos de abstração. Essa especificação formal inicial, conjuntamente com algumas propriedades descritas informalmente, é denominada **Arquitetura** (1).

Uma realização pode ser obtida a partir da arquitetura tomando-se um conjunto de decisões de projeto. Um bom hábito é avaliar e tomar decisões de projeto isoladamente e sistematicamente, produzindo especificações intermediárias cada vez mais refinadas, até que uma especificação final, que possa ser diretamente mapeada numa realização, seja atingida. A especificação final é chamada **implementação** e a abordagem, onde decisões de projeto são avaliadas e tomadas isoladamente, é chamada **refinamentos sucessivos**. A trajetória de projeto é apresentada na Figura 1.

Entretanto, LOTOS apresenta limitações com relação à representação dos elementos mais concretos necessários à implementação. Consequentemente, espera-se que uma tradução de uma especificação LOTOS para uma linguagem de implementação (software ou hardware) ocorra em algum passo da trajetória de projeto. Como muitos aspectos da funcionalidade de um sistema podem ser descritos eficientemente em LOTOS, parece interessante que essa TDF seja utilizada em grande parte da trajetória de projeto, em diferentes níveis de abstração.

2.3. Estilos de Especificação

Na Literatura costuma-se relacionar estilo com a maneira através da qual um autor se expressa utilizando os elementos de uma língua. No contexto de especificação formal, estilo designa a maneira pela qual um projetista expressa a funcionalidade de um sistema utilizando os elementos de uma linguagem de especificação (sintaxe e semântica).

---

(1) o termo **arquitetura**, utilizado neste artigo, representa o conceito de "architecture" enfatizado em [Viss 88]

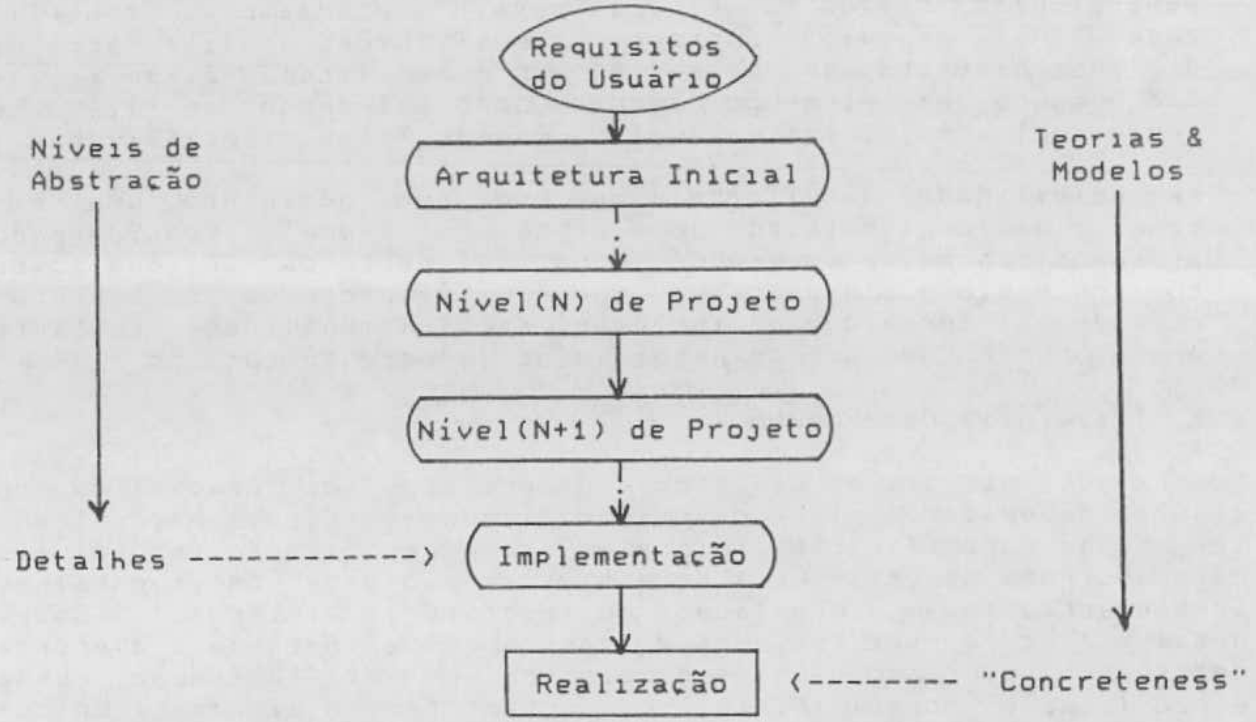

Figura 1 Trajetória de Projeto

Embora estilos sejam inerentes aos projetistas, estes devem disciplinar-se para que possam, de acordo com os objetivos da especificação, utilizar eficientemente os elementos da linguagem A especificação pode geralmente ser vista como uma:

- descrição de aspectos externos: especifica "o que" o sistema realiza, abstraindo a sua estrutura interna e detalhes;

- descrição de aspectos internos: especifica "como" o sistema realiza, mostrando a sua estrutura interna e/ou detalhes.

Nas descrições de aspectos externos os estilos monolítico e orientado a restrições podem ser utilizados, enquanto que nas descrições de aspectos internos os estilos orientado a recursos e orientado a estados é que podem ser utilizados. Informações adicionais sobre estilos são apresentadas em [Viss 88].

3. O Problema de Acesso por Exclusão Mútua

De acordo com a trajetória de projeto descrita na seção 2.2, o processo de projeto de um sistema começa com um conjunto de requisitos do usuário (descritos informalmente) a partir do qual uma arquitetura é derivada.

3.1. Definição do Problema

O exemplo é um sistema de acesso por exclusão mútua, que deve coordenar o acesso dos usuários a um recurso compartilhado (e.g., disco, impressora). O serviço de acesso por exclusão mútua apresenta duas primitivas de serviço (PS): ME_Begin para indicar o início do acesso ao recurso e ME_End para indicar a liberação

ao recurso. Qualquer ME_Begin num certo ponto de acesso ao serviço (PAS) deve ser seguido de um ME_End no mesmo PAS, o que caracteriza a exclusão mútua, pois evita que outro usuário tenha acesso ao recurso ao mesmo tempo

Os requisitos do usuário do sistema determinam que os usuários do serviço de acesso por exclusão mútua estão dispostos em localidades distintas (sistema distribuído). Esse requisito impõe que algumas decisões iniciais de projeto devem ser tomadas, a fim de que haja conectividade entre os usuários (serviço da camada inferior) e inteligência na coordenação da exclusão mútua (protocolo). Para tal, os usuários do serviço são conectados a controladores locais e todos os controladores são vinculados a um anel virtual, que representa o meio de comunicação do sistema ([Boch 87], [Souz 88]). A estrutura global do sistema é apresentada na Figura 2.

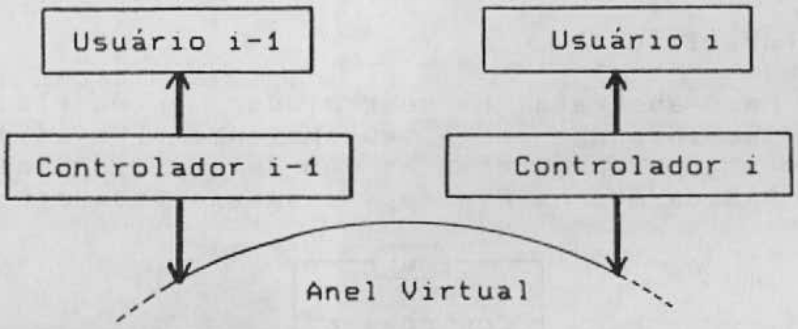

Figura 2 Estrutura Global do Sistema

O privilégio para o acesso ao recurso é negociado pelos controladores, através da troca de informações de estado. Dependendo do estado do vizinho da esquerda e de seu próprio estado, cada controlador decide se o seu usuário pode (ou não) ter acesso ao recurso. O algoritmo que define a obtenção do privilégio é uma versão modificada daquele apresentado em [Dijk 74]. Aqui considera-se que somente um controlador possui o privilégio por vez. O controlador que detém o privilégio permite ao seu usuário o acesso ao recurso. Decorrido um determinado intervalo de tempo, esse controlador modifica o seu estado, passando o privilégio para o seu vizinho da direita. Informações mais detalhadas sobre esse algoritmo são apresentadas em [Moss 77].

O serviço de anel virtual suporta a troca de informações de estado entre dois controladores vizinhos. Uma PS S_Req (solicitando o estado do vizinho da esquerda) num PAS i do anel virtual (AV) acarretará numa PS S_Ind no PAS i-1 do AV (2). O AV aguarda então uma PS S_Resp (com o valor do estado solicitado) no PAS i-1, o que acarretará numa PS S_Conf no PAS i.

A estrutura do sistema de acesso por exclusão mútua pode ser concebida como um conjunto de controladores (entidades de protocolo) vinculados ao serviço de anel virtual (serviço da camada subjacente). Os controladores utilizam o serviço de anel para fornecer, aos seus usuários, o serviço de acesso por exclusão mútua. Na Figura 3 é apresentada a estrutura do sistema em termos de fronteiras e de pontos de acesso aos serviços

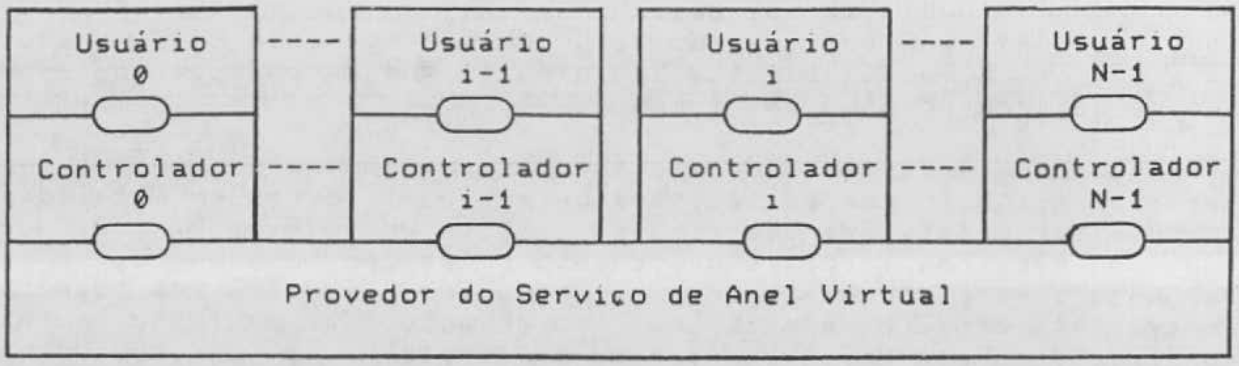

Figura 3 Estrutura em termos das Fronteiras dos Serviços

Este artigo concentra-se no projeto do controlador, visto que este é uma parte fundamental do sistema a ser produzido.

3.2. Arquitetura Inicial

Na descrição mais abstrata, o controlador é modelado por uma caixa preta, que interage com o seu ambiente através das portas u e r. A porta u representa o PAS de exclusão mútua (EM) e r representa o PAS de AV. Na Figura 4 é apresentado este modelo.

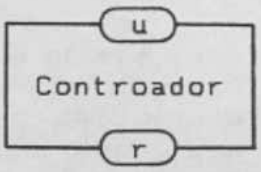

Figura 4 Modelo do Controlador

Visto que cada controlador possui somente um PAS EM e um PAS AV, ignora-se a identificação dos PASs na especificação formal. Entretanto, é importante ressaltar que na combinação dos controladores com o anel virtual, para a formação do sitema global, a identificação dos PASs torna-se obrigatória. Os procedimentos de falha não são considerados neste modelo, ao contrário do que ocorre em [Boch 87].

A seguir é apresentada a especificação LOTOS de um controlador, escrita no estilo orientado a restrições. Essa especificação e as demais deste artigo estão acompanhadas de comentários minuciosos. O leitor familiarizado com LOTOS pode suprimí-los.

```
(* ------------------------------------------------------------ *)
specification ME_Controller[u,r] : noexit
(*
A fim de compactar as especificações, os Tipos Abstratos de Dados
não são apresentados. A definição completa desses TADs é apresen-
tada em [PiSo 90]. As seguintes definições estão disponiveis:
```

---

(2) i ∈ (0, 1, ..., N-1), um S_Req no PAS 0 acarreta num S_Ind no PAS N-1

```
- Primitivas para o Acesso à Exclusão Mútua
  ME_Begin, ME_End
- Primitivas para o Anel Virtual
  S_Req, S_Ind, S_Resp(state), S_Conf(state)

Considera-se o comportamento como uma composição de aspectos
locais à porta r (RingConstraints) e aspectos relacionados à
comunicação entre as portas u e r (ControllerProtocol).
*)
behaviour
  RingConstraints[r]
  |[r]|
  ControllerProtocol[u,r]
where
(*
As restrições locais à porta r são descritas como uma composição
sincronizada das restrições relacionas à solicitação do estado do
vizinho da esquerda (AskLState) com as restrições relacionadas à
resposta ao vizinho da direita (AnswerRState).
*)
process RingConstraints[r] : noexit:=
  AskLState[r]
  ||
  AnswerRState[r]
where
(*
As restrições relacionadas à solicitação de estado impõem que uma
primitiva de requisição de estado (S_Req) deve ser confirmada por
uma primitiva contendo o estado do vizinho da esquerda (S_Conf)
*)
process AskLState[r] : noexit:=
r !S_Req ; WaitForConf[r] >> AskLState[r]
where
  process WaitForConf[r] : exit:=
  r ?prim : RingPrim [IsNotReq(prim)] ;
  ( [IsConf(prim)] -> exit
  [] [IsNotConf(prim)] -> WaitForConf[r] )
  endproc (* WaitForConf *)
endproc (* AskLState *)
(*
As restrições, relacionadas à resposta, impõem que uma primitiva
de indicação (S_Ind) deve ser imediatamente seguida de uma
resposta (S_Resp).
*)
  process AnswerRState[r] : noexit:=
  r ?prim1 : RingPrim [IsNotResp(prim1)] ;
  ( [IsInd(prim1)] -> r ?prim2 : RingPrim [IsResp(prim2)] ;
                      AnswerRState[r]
  [] [IsNotInd(prim1)] -> AnswerRState[r] )
  endproc (* AnswerRState *)
endproc (* RingConstraints *)
(*
As restrições abrangendo as portas u e r representam o protocolo
de controle, isto é, como o acesso ao recurso é garantido ao
usuário. O processo ControllerProtocol é modelado por uma
"inicialização", onde um estado inicial é escolhido indetermins-
ticamente, seguida pela operação do protocolo.
```

```
A escolha indeterministica, nesse nivel de abstração, permite
considerar diferentes modos para a implementacão da inicializa-
cão Na implementação, o estado inicial deverá responder às
regras do algoritmo de exclusão mútua e, portanto, essa escolha
passará a ser determinística
*)
process ControllerProtocol[u,r] : noexit :=
choice InitialState : State
[] i ; ProtocolOperation[u,r](InitialState)
where
(*
A operacão do protocolo apresenta três fases: (a) tentativa de
obtencão do privilégio; (b) oferta de acesso ao recurso ao
usuário, (c) modificacão do estado do controlador

Periodicamente o controlador tenta obter o privilégio. Este fato
é modelado através de um evento interno. Eventos internos em
LOTOS representam ocorrências não observáveis do sistema, mas que
alteram o seu comportamento observável.
*)
process ProtocolOperation[u,r](s : State) : noexit:=
i ; AskPrivilege[u,r](s)
    >> AccessOffered[u]
    >> ( choice NewState : State
       [] [NewState ne s] ->
          i ; ProtocolOperation[u,r](NewState) )
where
(*
Enquanto tenta obter o privilégio, o controlador deve responder
com o seu estado a eventuais solicitações do vizinho da direita.
*)
process AskPrivilege[u,r](s : State) : exit:=
r ?prim : RingPrim [IsResp(prim) implies (s IsStateOf prim)] ;
( [IsConf(prim)] -> ( [s HasPrivilegeWith State(prim)] -> exit
                    [] [s NoPrivilegeWith State(prim))] ->
                       ProtocolOperation[u,r](s) )
[] [IsNotConf(prim))] -> AskPrivilege[u,r](s) )
endproc (* AskPrivilege *)
(*
Após a obtencão do privilégio pelo controlador, o recurso estará
disponivel ao usuário do serviço de acesso por exclusão mútua. O
controlador decidirá quando a oferta de acesso deve ser interrom-
pida, no caso do usuário não utilizá-lo. Este fato é modelado
através de um evento interno.
*)
      process AccessOffered[u] : exit:=
      ( SingleAccess[u]
      [] i ; exit )
      where
        process SingleAccess[u] : exit:=
        u !ME_Begin ; u !ME_End ; exit
        endproc (* SingleAccess *)
      endproc (* AccessOffered *)
    endproc (* ProtocolOperation *)
  endproc (* ControllerProtocol *)
endspec (* ME_Controller *)
(* ------------------------------------------------------------ *)
```

4. Passos de Projeto

Esta seção discute os passos de projeto que levam à produção de especificações intermediárias. Em cada passo considera-se a existência de uma especificação inicial, em algum nível de abstração (N), que é transformada numa especificação mais refinada, em algum nível de abstação (N+1). Uma vez que em cada passo de projeto mais detalhes são incluídos na especificação (N+1) resultante, caracterizando o refinamento, a especificação (N+1) estará num nível de abstração inferior ao da especificação (N).

Num passo de projeto, algumas típicas decisões de projeto são: definição da estrutura interna, remoção (ou isolamento) de indeterminismos e refinamento de interfaces. Neste artigo somente os dois primeiros tipos de decisões de projeto são exemplificados.

Em relação à definição da estrutura interna, pode-se aplicar o princípio de "separação dos aspectos independentes", que é uma decorrência do critério de ortogonalidade (seção 2.1). Esse princípio consiste em identificar aspectos (parcialmente) independentes e, baseado nesses aspectos, definir componentes cooperantes. O grau de independência dos componentes pode determinar a complexidade de suas interfaces. Frequentemente, uma separação inconveniente torna as interfaces muito confusas ou complicadas.

A remoção de indeterminismos pode ser alcançada complementando a especificação com detalhes (e.g., descrição completa de algoritmos). O isolamento de indeterminismos pode ser aplicado aos aspectos que não podem ser formalmente representados no modelo da linguagem de especificação utilizada (e.g., tempo em LOTOS).

Na definição da estrutura, componentes internos do sistema e suas interações devem ser descritos. Todavia, alguns aspectos do comportamento da arquitetura devem ser preservados ao longo do processo de refinamentos sucessivos. Em LOTOS, isto pode ser alcançado com a utilização de elementos da linguagem, tais como os operadores de paralelismo e de escondimento de portas. O emprego de tais operadores caracteriza o estilo orientado a recursos.

Na metodologia empregada neste artigo, muitas vezes informações de projeto são representadas através dos elementos sintáticos de LOTOS para preservar aspectos comportamentais das especificações mais abstratas. Por exemplo, considerando que uma especificação (N+1) é equivalente por bissimulação a uma especificação (N), a especificação (N+1) não é apenas uma combinação diferente de elementos da linguagem LOTOS, mas contém também informações adicionais de projeto (e.g., processos LOTOS representam componentes cooperantes e eventos internos representam suas interfaces).

4.1. Refinamento do Controle de Acesso ao Recurso

Na especificação do controlador (seção 3.2) pode-se identificar duas correntes principais de funcionalidades: (a) controle de acesso ao recurso; (b) gerência do privilégio através do anel.

A Figura 5 ilustra como esses dois aspectos podem ser utilizados para definir componentes cooperantes.

Figura 5 Refinamento do Controle de Acesso ao Recurso

Os processos AccessControl e PrivManager comunicam-se através da porta a, considerada interna ao controlador. Embora essa nova especificação ostente o estilo orientado a recursos, cada um de seus componentes (PrivManager e AccessControl) serão novamente especificados através de descrições de aspectos externos (estilos monolítico ou orientado a restrições).

```
(*-------------------------------------------------------------*)
specification ME_Controller1[u,r] : noexit
(*
Os mesmos TADs da seção 3.2. são considerados disponíveis
*)
behaviour
  hide a in
    AccessControl[u,a]
    |[a]|
    PrivManager[a,r]
where
(*
As interações na porta a indicam se o acesso está habilitado ou
não. Portanto as seguintes definições estão disponíveis:

- EnableAccess, DisableAccess

O processo AccessControl espera até que o acesso seja habilitado
pelo gerente do privilégio. Uma vez habilitado, o usuário tem a
oportunidade de "acessar" o recurso. A sequência termina quando o
acesso é desabilitado.
*)
process AccessControl[u,a] : noexit:=
WaitEnable[a]
>> AccessOffered[u]
>> WaitDisable[a]
>> AccessControl[u,a]
where
  process WaitEnable[a] : exit:=
  a !EnableAccess ; exit
  endproc (* WaitEnable *)
  process WaitDisable[a] : exit:=
  a !DisableAccess ; exit
  endproc (* WaitDisable *)
  process AccessOffered[u] ... (* idêntico ao da seção 3.2. *)
endproc (* AccessControl *)
process PrivManager[a,r] : noexit:=
  RingConstraints[r]
  |[r]|
  PrivOperation[a,r]
where
(*
```

```
O processo PrivOperation é similar ao ProtocolOperation da seção
3.2 ; a diferença é que agora o temporizador de acesso ao recurso
foi deslocado para o processo AccessControl e não aparece em
PrivOperation como acontecia no ProtocolOperation da seção 3 2
*)
process PrivOperation[a,r]   noexit:=
choice InitialState : State
[]  i ; ProtocolOperation[a,r](InitialState)
where
  process ProtocolOperation[a,r](s   State) : noexit:=
  i ; AskPrivilege[a,r](s)
      >> AccessInstance[a]
      >> (  choice NewState : State
          [] [NewState ne s] ->
             i ; ProtocolOperation[a,r](NewState)  )
  where
       process AskPrivilege[a,r] ... (* vide seção 3.2. *)
       process AccessInstance[a] : exit:=
       a !EnableAccess ; a !DisableAccess ; exit
       endproc (* AccessInstance *)
     endproc (* ProtocolOperation *)
   endproc (* PrivOperation *)
  endproc (* PrivManager *)
  process RingConstraints[u] ... (* vide seção 3.2. *)
endspec (* ME_Controller1 *)
(* ------------------------------------------------------------ *)
```

É interessante notar a aplicação do critério de generalidade no desenvolvimento das especificações Alguns processos, que foram definidos na arquitetura, puderem ser reutilizados nesta descrição intermediária, economizando-se tempo e papel

4.2. Refinamento do Gerente do Privilégio

Este passo de projeto concentra-se em PrivManager, onde dois aspectos principais podem ser identificados: manipulação das primitivas do anel e gerência central. PrivManager é decomposto, sendo que a nova estrutura do controlador é apresentada na Figura 6.

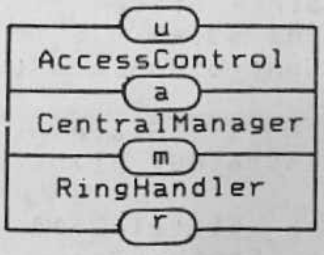

Figure 6. Refinamento do Gerente do Privilégio

Uma vez que somente PrivManager é decomposto e AccessControl permanece inalterado, omite-se a especificação de AccessControl nesta seção.

```
(* ------------------------------------------------------------ *)
specification PrivManager1[a,r] : noexit
(*
Os mesmos TADs das seções 3.2. e 4.1. são disponíveis. As
```

```
interações  na porta m são para responder e requisitar o privilé-
gio  As seguintes definições são disponíveis

- PrivReq(state), PrivResp(state)
*)
behaviour
hide m in
  CentralManager[a,m]
  |[m]|
  RingHandler[m,r]
where
(*
O  processo  CentralManager  é  descrito  através  da  composição
sincronizada do  processo  ManagConstraints,  que  representa  as
restrições na porta m, com o processo ManagControl, que relaciona
as interações na porta m às interações na porta a
*)
process CentralManager[a,m]   noexit:=
  ManagConstraints[m]
  |[m]|
  ManagControl[a,m]
where
  process ManagConstraints[m] : noexit:=
  m !PrivReq(s) ;
  m ?p : PrivPrim [IsPrivResp(p)] ;
  ManagConstraints[m]
  endproc (* ManagConstraints *)
(*
O processo ManagControl descreve o escalonamento  da  solicitação
do privilégio e da concessão do acesso
*)
process ManagControl[a,m] : noexit:=
choice InitialState : State
[]  i ; ManagControlRun[a,m](InitialState)
where
  process ManagControlRun[a,m](s : State) : noexit:=
  i ; AskCompare[a,m](s)
      >> AccessInstance[a]
      >> (  choice NewState : State
            [] [NewState ne s] ->
               i ; ManagControlRun[a,m](NewState)   )
  where
    process AskCompare[a,m](s : State) : exit:=
    m ?p1   PrivPrim (s IsStateOf p1) ;
    m ?p2   PrivPrim ;
    (  [s HasPrivilegeWith (State(p2))] -> exit
    [] [s NoPrivilegeWith (State(p2))] ->
       ManagControlRun[a,m](s)   )
    endproc (* AskCompare *)
      process AccessInstance[a] ... (* vide seção 4.1 *)
    endproc (* ManagControlRun *)
  endproc (* ManagControl *)
endproc (* CentralManager *)
(*
O processo RingHandler, quando requisitado, tenta obter o  privi-
légio iniciando uma sequência de primitivas de anel
*)
```

```
process RingHandler[m,r]   noexit =
  RingConstraints[r]
  |[r]|
  RingSeqHandler[m,r]
where
  process RingSeqHandler [m,r]  noexit =
  WaitPrivState[m]
  >> accept  s : State in WaitConf[r](s)
  >> accept s1 : State in RespPrivState[m](s1)
  >> RingSeqHandler[m,r]
  where
      process WaitPrivState[m] : exit(State) :=
      m ?p : PrivPrim ; exit(State(p))
      endproc (* WaitPrivState *)
      process WaitConf[r](s : State) : exit(State) :=
      r ?p : RingPrim [IsResp(p) implies (s IsStateOf p)] ;
      (  [IsConf(p)] -> exit(State(p))
      [] [IsNotConf(p)] -> WaitConf[r](s)  )
      endproc (* WaitConf *)
      process RespPrivState[m](s1 : State) : exit :=
      m ?p : PrivPrim [s1 IsStateOf p ] ; exit
      endproc (* RespPrivState *)
    endproc (* RingSeqHandler *)
    process RingConstraints[r] ... (* vide secao 3.2. *)
  endproc (* RingHandler *)
endspec (* PrivManager1 *)
```

4.3. Refinamento do Gerente Central

Neste passo refina-se **CentralManager**, identificando-se funções cooperantes a serem desempenhadas por este componente. Tais funções são: (a) inicialização de estado; (b) armazenamento de estado; (c) iniciativa de requisição de privilégio; (d) concessão de acesso (algoritmo de privilégio); (e) atualização de estado

A estrutura escolhida para refletir essas funções consiste nos processos **Initialization** e **Memory**, que aparecem como periféricos, que são gerenciados pelo processo **CentralControl**. Essa estrutura é apresentada na Figura 7.

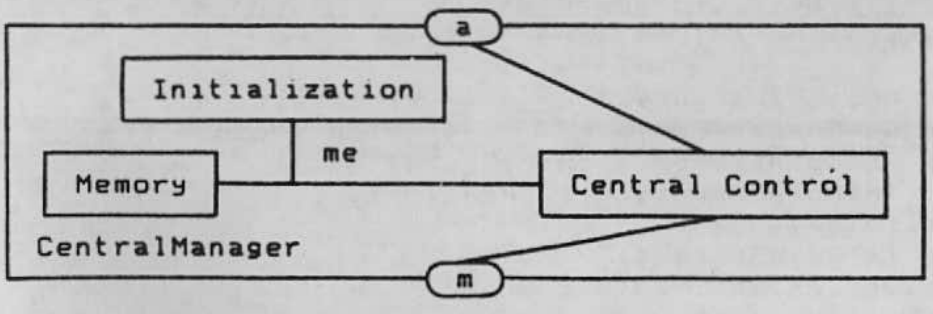

Figura 7. Refinamento do Gerente Central

```
(* ------------------------------------------------------------ *)
specification CentralManager1[a,m] : noexit
(*
Os mesmos TADs das secões 3.2, 4.1 e 4.2 estão disponiveis.
*)
```

```
behaviour
hide me in
  (  Initialization[me] ||| CentralControl[me,a,m]  )
  |[me]|
  Memory[me]
where
(*
O processo Memory interage com o seu ambiente através de
operações de leitura e escrita. Neste exemplo, inicialmente o
processo Initialization escreve na memória e só então as opera-
ções de leitura e escrita são permitidas em qualquer ordem. As
seguintes definições estão disponíveis: Read(state), Write(state)
*)
process Memory[me] : noexit :=
me ?op : MemoryOperation [IsWrite(op)] ;
MemoryRun[me](State(op))
where
  process MemoryRun[me](s : State) : noexit:=
  me ?op : MemoryOperation [IsRead(op) implies (s IsStateOf op)] ;
  MemoryRun[me](State(op))
  endproc (* MemoryRun *)
endproc (* Memory *)
(*
Após inicializar a memória, o processo Initialization para.
*)
process Initialization[me] : noexit:=
choice InitialState : State
[] i ; me !Write(InitialState) ; stop
endproc (* Initialization *)
(*
O processo CentralControl requisita o privilégio, executa o algo-
rítmo, eventualmente habilitando o acesso, e gera um novo estado.
*)
process CentralControl[me,a,m] : noexit :=
i ; ReadState[me]
    >> accept s : State in AskCompare[a,m](s)
    >> AccessInstance[a]
    >> choice Newstate : State
       [] [NewState ne s] ->
          i ; WriteState[me](s)
    >> CentralControl[me,a,m]
where
  process AccessInstance[a] ... (* vide seção 4.1 *)
  process AskCompare[me,a,m] ... (* vide seção 4.2 *)
  process ReadState[me] : exit(State):=
  me ?op : MemoryOperation [IsRead(op)] ;
  exit(State(op))
  endproc (* ReadState *)
  process WriteState[me](s : State) : exit:=
  me !Write(s) ; exit
  endproc (* WriteState *)
  endproc (* CentralControl *)
endspec (* CentralManager1 *)
(* ------------------------------------------------------------ *)
```

4.4. Refinamento do Controle Central

Neste passo refina-se CentralControl. Analisando-se este processo pode-se identificar aspectos relacionados à decisão do privilégio (encapsulamento do algoritmo) e uma função de escalonamento. A nova estrutura de CentralControl é apresentada na Figura 8.

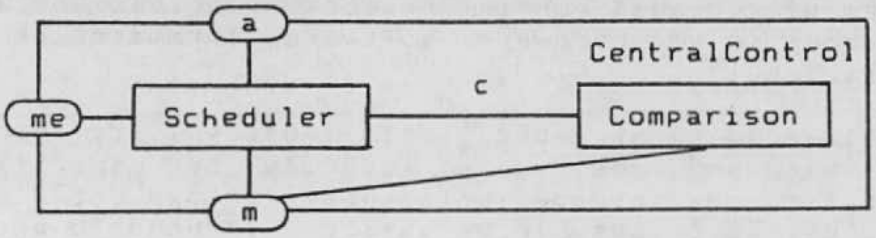

Figura 8. Refinamento do Controle Central

Comparison sincroniza-se com Scheduler na porta m, para obter o estado do controlador e o estado do vizinho da esquerda. O resultado da comparação é fornecido em termos de um novo estado através de uma interação na porta c. Se o novo estado for diferente do estado vigente (na memória), é porque o controlador realizou uma mudança e, consequentemente, ele detinha o privilégio.

```
(* ------------------------------------------------------------ *)
specification CentralControl1[me,a,m] : noexit
(*
Os mesmos TADs das seções 3.2, 4.1, 4.2 e 4.3 são disponíveis.
*)
behaviour
hide c in
  Schedulller[me,a,m,c] |[m,c]| Compararison[m,c]
where
  process Comparison[m,c] : noexit :=
  m ?p1 : PrivPrim ;
  m ?p2 : PrivPrim ;
  (  [State(p1) HasPrivilegeWith State(p2)] ->
     (choice NewState : State
      [] [NewState ne State(p1)] ->
         i ; c !NewState !true ; exit)
  [] [State(p1) NoPrivilegeWith State(p2)] ->
     (c !State(p1) !false ; exit)  )
  >> Compararison[m,c]
  endproc (* Comparison *)
  process Scheduler[me,a,m,c] : noexit :=
  i ; ReadState[me]
      >> accept s : State in
         (  m !PrivReq(s) ; m ?p : PrivPrim ;
            c ?NewState : State ?priv : Bool ;
            (  [priv] -> ( AccessInstance[a]
                           >> WriteState[me](NewState) )
            [] [not(priv)] -> exit  )  )
      >> Schedulller[me,a,m,c]
  where
    process ReadState[me] ... (* vide seção 4.3 *)
    process WriteState[me] ...  (* vide seção 4.3 *)
    process AccessInstance[me] ... (* vide seção 4.1 *)
  endproc (* Scheduler *)
endspec (* CentralControl1 *)
```

5. Decisões Posteriores de Implementação

As decisões de projeto consideradas até agora são relativas à definição de estrutura e á remoção de indeterminismos. Entretanto, num certo passo da trajetória de projeto, as decisões de projeto devem ser guiadas por informações relativas aos ambientes das realizações (e.g., hardware, software, firmware, recursos do sistema operacional).

Para uma realização em hardware, seria possivel definir blocos básicos de hardware, os quais poderiam ser especificados em LOTOS. Especificações intermediárias poderiam ser então descritas como uma composição desses blocos básicos, tornando o processo de implementação mais simples, eficiente e correto.

A fim de exemplificar essa idéia, considera-se AccessControl da seção 4.1 como sendo o processo a ser refinado. Além disso, considera-se um temporizador como um bloco a ser especificado em LOTOS, a partir do qual algumas implementações válidas poderão estar disponíveis. O processo AccessControl pode ser então estruturado como uma combinação de um sequenciador e de um temporizador, conforme ilustrado na Figura 9.

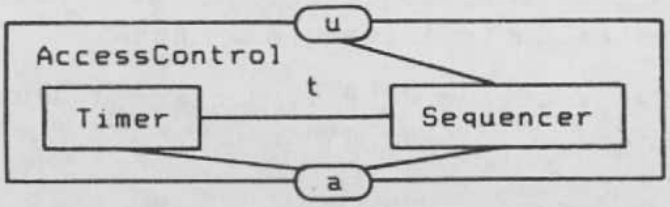

Figura 9. Refinamento do Controle de Acesso

Na especificação de AccessControl utilizou-se uma versão "renomeada" de um temporizador genérico. As interecões básicas de um temporizador são SetTimer, ResetTimer e TimeOut. Nessa especificação elas foram renomeadas, respectivamente, EnableAccess, DisableAccess (na porta a) e interação na porta t.

```
(* ------------------------------------------------------------ *)
specification AccessControl1[u,a] : noexit
(*
Os TADs das seções 3.2, 4.1, 4.2, 4.3 e 4.4 são disponiveis.
*)
behaviour
hide t in
  Sequencer[a,u,t] |[a,t]| Timer[a,t]
where
  process Sequencer[a,u,t] : noexit :=
  WaitEnable[a] ;
  (  t ; exit
  [] SingleAccess[u]  )
  >> WaitDisable[a]
  >> Sequencer[a,u,t]
  endproc (* Sequencer *)
  process Timer[a,t] : noexit :=
  WaitEnable[a]
  >> (  (i ; t ; stop)
```

```
    [] WaitDisable[a]  )
  )) Timer[a,t]
  endproc (* Timer *)
endspec (* AccessControl1 *)
(* ------------------------------------------------------------ *)
```

6. Conclusões

Este artigo considera, em cada passo da trajetória de projeto, algumas das múltiplas possibilidades de decisões de projeto, para em seguida formalizá-las em termos de versões mais refinadas de projeto. Uma melhor avaliação da qualidade do projeto deve ser obtida em trabalhos futuros, quando da implementação e realização dos componentes especificados e quando da avaliação do custo e performance em relação aos requisitos informais do usuário.

Em cada passo de projeto, a especificação resultante deve corresponder de alguma forma à especificação da etapa anterior. Esse conceito de conformidade pode ser traduzido em termos de propriedades semânticas, que variam de acordo com os objetivos de cada passo de projeto. Neste artigo a relação formal de equivalência entre duas especificações consecutivas não foi verificada. Entretanto, a sintaxe e a semântica das especificações foram testadas, através de simulação, utilizando-se a ferramenta SEDOS HIPPO [Eijk 88]. Trabalhos futuros, que relacionem as propriedades de conformidade e as estratégias de verificação com os objetivos do passo de projeto, são esperados. O desenvolvimento de ferramentas para a verificação automática da correção de projeto também é esperado.

A novidade da abordagem de projeto adotada neste artigo, em relação a que foi adotada em [Souz 88] onde o mesmo exemplo foi especificado em Calculus of Communicating Systems (CCS) [Miln 80], é que no presente artigo, em cada passo de projeto, busca-se uma decomposição a partir de funções que podem ser identificadas na especificação, enquanto que em [Souz 88] busca-se imediatamente uma composição de agentes CCS pré-definidos que seja equivalente a essa especificação. Acredita-se que esta nova abordagem é muito mais geral que a anterior e poderia ser eficientemente aplicada em sistemas bem mais complexos que o exemplo utilizado. Uma evidência dessa afirmação pode ser encontrada em [Boga 89].

Um outro aspecto importante da abordagem empregada neste artigo é o controle do projeto utilizando TDFs, isto é, decisões de projeto são tomadas, justificadas e documentadas através de especificações formais. Essas especificações podem ser verificadas utilizando-se ferramentas, permitindo que erros de projetosejam detectados e corrigidos nas etapas preliminares da trajetória de projeto. Em abordagens convencionais, tais erros são normalmente detectados nas implementações.

Portanto, uma metodologia de projeto baseada nos princípios discutidos neste artigo e usando TDFs pode resultar em uma economia em tempo e dinheiro.

7. Referências

[Boch 87] G.v. Bochmann , J.P. Verjus , "Some Comments on Transition-Oriented versus Structured Specification of Distributed Algorithms and Protocols", IEEE Transactions on Software Engineering, 13, 1987, pp 501-505.

[Dijk 74] E.W. Dijkstra, "Self-stabilizing Systems in Spite of Distributed Control", Communications of the ACM, Vol 17, No 11, 1974, pp. 643-644.

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

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

[Miln 80] R. Milner, "A Calculus of Communicating Systems", LNCS 92 (G. Goos,J. Hartmanis eds.), Springer-Verlag, 1980.

[Moss 77] J. Mossiere, J. Tchente, J.P. Verjus, "Sur l'exclusion mutuelle dans les reseaux informatiques", IRISA, Rennes, França, Publ. 75, 1977.

[PiSo 90] L. F. Pires, W. L. Souza, "Step-wise Refinement Design Example Using LOTOS", anais da Third International Conference on Formal Description Techniques (FORTE'90), 5-8/11/90, Madrid (Espanha), pp. 289-306.

[Pire 89] L. F. Pires, "Projeto de Protocolos de Inter-Rede com o Uso da Técnica de Especificação Formal LOTOS - Especificações de Sistemas Intermediários para Comunicação com e sem Conexão", tese de mestrado, EPUSP (São Paulo), 1989, 260 pags.

[Scol 87] G. Scollo, M.v. Sinderen, "On the Architectural Design of the Formal Specification of the Session Standards in LOTOS", Protocol Specification, Testing, and Verification, VI (B. Sarikaya, G.v. Bochmann eds.), North-Holland, 1987, pp. 3-14.

[Souz 88] W.L. Souza , B.G. Riso , "Using CCS for Protocol Specifications by Step-wise Refinements", anais do Second International Symposium on Interoperable Information Systems, INTAP (Tokyo-Japan), Nov. 1988, pp. 135-142.

[Viss 88] C.A. Vissers, G. Scollo, M.v. Sinderen, "Architecture and Specification Style in Formal Descriptions of Distributed Systems", Protocol Specification, Testing, and Verification, VIII (S. Aggarwal and K. Sabnani eds.), North-Holland, 1988, pp. 189-204.