Transformações de Especificações LOTOS Direcionadas para a Implementação

P. Cunha[1], E. Najm[2], J. Queiroz[1]

[1] Departamento de Informática da UFPE
Av. Prof. Luiz Freire, S/N - Edifício do CCEN
Campus UFPE - Cidade Universitária
CP 7851 / 50740-514 Recife - PE

[2] INRIA
Domaine de Voluceau - Rocquencourt
BP 105 / 78153 Le Chesnay Cedex / França

Resumo

A metodologia proposta no projeto Lotosphere (ESPRIT 2304) inclui um catálogo de transformação LOTOS que preservam a semântica de suas aplicações. Cada transformação no catálogo define as condições necessárias para a sua aplicação e o tipo de refinamento executado. O objetivo principal deste trabalho é a aplicação de técnicas de transformações do catálogo voltadas para as fases finais da trajetória de desenvolvimento do sistema. Duas transformações foram selecionadas em função de sua importância por estarem direcionadas à possíveis implementações: "multi-way" para "two-way rendez-vous" e a estrutura de transformações LOTOMATON. Demonstramos a utilidade das transformações escolhidas através da apresentação do Sistema de Segurança de um Museu.

1 Introdução

Uma metodologia de refinamentos sucessivos para a linguagem de especificação LOTOS [1,6] está sendo desenvolvida no projeto Lotosphere [3,4]. Começando de um conjunto inicial de requisitos do usuário, a metodologia descreve um processo formal e disciplinado para a derivação de implementações. Fundamental para a metodologia, encontram-se as noções de projeto, passo de projeto e trajetória de projeto. Um projeto é uma especificação LOTOS do sistema num certo grau de abstração, complementada com alguma informação textual informal representando aspectos do sistema ainda não formalizados. Um projeto pode ser refinado de várias maneiras em outros projetos do mesmo sistema. Um passo de projeto traduz uma escolha de refinamento a qual incorpora alguma decisão de projeto e leva em consideração um subconjunto relevante dos requisitos do usuário. Uma trajetória de projeto é uma sequência de passos válidos de projeto que começa dos requisitos do usuário e conduz à uma implementação onde estes requisitos foram totalmente satisfeitos.

A metodologia proposta no projeto Lotosphere distingue entre fases iniciais e finais na trajetória do projeto. As fases iniciais tentam produzir um projeto, o mais abstrato possível, descrevendo o sistema a ser desenvolvido e que compreendem as fases de

E-mail dos autores: prfc@di.ufpe.br | najm@inria.inria.fr | queiroz@di.ufpe.br

captura dos requisitos e de criação da arquitetura (projeto) inicial. As fases finais do projeto se preocupam com a derivação do sistema real a partir da arquitetura inicial.

O documento Lotosphere apresentado em [3] propõe um catálogo de transformações que preservam a semântica de especificações LOTOS. Cada transformação no catálogo define as condições necessárias para a sua aplicação, o tipo de refinamento na transformação e a propriedade semântica preservada.

O objetivo principal deste trabalho é a aplicação de técnicas de transformação constantes do catálogo Lotosphere nas fases finais da trajetória de desenvolvimento do sistema. Duas transformações foram selecionadas em função de sua importância por estarem direcionadas à possíveis implementações: "multi-way" para "two-way rendez-vous" e a estrutura de transformações LOTOMATON. Através do uso de um exemplo concreto demonstramos a utilidade das transformações escolhidas. Selecionamos o Sistema de Segurança de um Museu por ser pequeno e significativo, e por permitir uma apresentação clara e concisa das dificuldades encontradas. Em particular, mostramos como adaptações podem ser realizadas para se efetuar as transformações em questão.

Na seção 2 deste trabalho descrevemos, brevemente, as fases finais da trajetória de projeto e apresentamos as transformações correspondentes necessárias para estas fases. A seção 3 define as funcionalidades do exemplo, como também a especificação LOTOS inicial desta aplicação. A seção 4 discute o uso da transformação "multi-way" para "two-way rendez-vous", enquanto que a seção 5 mostra a aplicação da estrutura de transformações teste LOTOMATON. Finalmente, nas seções 6 e 7, discutimos algumas possibilidades de implementação do nosso exemplo e apresentamos as nossas considerações finais.

2 Transformações Direcionadas para a Implementação

As fases finais da trajetória de projeto começam a partir de uma especificação arquitetural inicial do sistema a ser desenvolvido, sem incluir detalhes específicos de um tipo de implementação. A fase de transformações direcionadas para a implementação procura produzir uma descrição LOTOS do sistema que pode ser facilmente mapeada num núcleo operacional que suporte a realização. Exemplos destas transformações incluem: i) "multi-way" para "two-way rendez-vous"; ii) tornar estados e transições explícitas na especificação; e iii) decompor o sistema em recursos diretamente disponíveis no núcleo operacional.

Como sabemos, na fase de realização, o modelo de implementação descrito em LOTOS é mapeado em elementos operacionais finais. Esta fase envolve normalmente um processo de tradução no lugar de técnicas de transformação. Numa tradução, a especificação inteira é mapeada na linguagem objeto final, enquanto que, numa transformação, uma parte designada da especificação pode ser substituída por uma equivalente mais orientada para a implementação. Como exemplo de um tipo de mapeamento na fase de realização, temos a tradução de uma comunicação baseada em "rendez-vous" em primitivas de comunicação existentes no sistema operacional final.

Consideramos duas transformações do catálogo Lotosphere para aplicar no nosso exemplo concreto do Sistema de Segurança de um Museu: i) "multi-way" para "two-way rendez-vous" (fase de implementação); e a estrutura de transformações de LOTOMATON (fases de implementação e realização). Estas transformações são descritas nas subseções seguintes.

2.1 Transformação "multi-way" para "two-way rendez-vous"

Em muitas linguagens de programação concorrente não se permite executar sincronização "multi-way" entre vários processos em uma simples construção de comunicação, mas somente comunicações "two-way" envolvendo dois participantes. Isto implica que as especificações desenvolvidas nas fases finais da trajetória de projeto deverão conter unicamente comunicações "two-way". Esta restrição não se verifica nas especificações iniciais, desde que LOTOS proporciona uma total liberdade para incluir mais de dois processos num "rendez-vous" simples. Além do mais, o estilo orientado a restrições, frequentemente usado em especificações iniciais de LOTOS, explora esta possibilidade de comunicação "multi-way" quando coloca juntas as restrições representadas por vários processos em paralelo sincronizando em portas comuns.

Desta maneira, algum tipo de transformação de comunicação "multi-way" para "two-way" se faz necessário. Soluções para este problema foram incluídas no catálogo Lotosphere de transformações que preservam a corretude [3], onde temos a definição formal deste mecanismo. As soluções atacam subconjuntos diferentes da definição do problema e mostram várias técnicas de resolução no lugar de regras que são aplicadas automaticamente. Todas seguem, em princípio, o seguinte critério de corretude: "Escondendo a "multi-way" porta "a" da especificação original, mantem-se a equivalência observacional da especificação transformada onde as portas que substituem "a" são também escondidas".

Na literatura, podemos encontrar implementações para o problema do "multi-way rendez-vous" [12], porém estas propostas alteram substancialmente a estrutura de comunicação entre processos. A filosofia em [3], no entanto, opera só localmente nos processos substituindo somente as portas envolvidas no "multi-way rendez-vous". Achamos que esta proposta é mais próxima do conceito de transformar uma especificação numa outra, sem a necessidade de, por exemplo, modificar a arquitetura dos processos, as quais em especificações finais deverão estar já decididas.

A seguir, apresentamos os princípios gerais para ajudar na escolha da solução mais conveniente para a transformação do "multi-way rendez-vous". Nós assumimos que esta sincronização se verifica na porta indicada como "a" abaixo:

1. A comunicação "multi-way" pode implicar numa concordância entre os processos entre os valores passados. Em outros termos, a escolha não-determinística implícita na marca de "?" deve ser resolvida;

2. Quando a porta "a" está presente numa escolha não-determinística, uma concordância de como executar uma ação particular dentre as ações possíveis nos processos deve ser determinada;

3. Distingue-se dois casos quando a porta "a" participa numa escolha não-determinística: i) a concordância limita-se ao conjunto de processos sincronizando na porta "a" e, transitivamente, nas portas presentes na escolha não-determinística. Neste caso, um construtor "hiding" colocado adequadamente engloba a composição paralela dos processos envolvidos; e ii) processos externos participam da concordância na sincronização em "a" e um construtor "hiding" não é estabelecido como anteriormente;

4. A comunicação "multi-way" é usada com o propósito de sincronizar todos os processos envolvidos na comunicação. Desta maneira, após a ação de sincronização na porta "a" todos os participantes devem atingir um certo ponto em suas execuções. Isto sempre acontece quando assume-se uma concordância de ações como implicado no ítem 2 acima. O uso de sincronização "multi-way" pode ser realizado também no caso onde não há acordo nas ações envolvidas, acarretando, então, o tratamento da ação na porta "a" como atômica. Chamamos este requisito de sincronização completa de processos;

5. A concordância entre os processos participantes da comunicação pode ser realizada por uma entidade centralizada ou distribuída. Esta decisão deve se basear em considerações arquiteturais do sistema.

As soluções propostas variam do caso mais simples para o mais complexo, implementando diferentes tipos de concordância e técnicas entre processos. Escolhas de arquitetura e os critérios apresentados anteriormente devem ser adaptados ao problema específico, a fim de ajudar na seleção da solução mais conveniente. O leitor pode se referir ao documento do catálogo Lotosphere [3] para uma discussão mais detalhada dos critérios descritos acima.

2.2 Transformações de LOTOMATON

LOTOMATON [9,10,11] é uma álgebra de processos, a qual estende os operadores LOTOS com um novo grupo de operadores chamados de "transdutores", permitindo uma ampliação das aplicações de LOTOS. Transdutores em LOTOMATON se baseiam em paradigmas orientados à implementação tais como atribuição de variáveis, separação entre dados e controle, estados e transições explícitas, e operações atômicas.

Um transdutor é um caso especial de uma máquina de estados finitos. Cada transdutor tem uma aridade fixa. Um transdutor de aridade n, ou *n-transdutor*, define uma máquina de estados que controla o comportamento destes "n" argumentos (expressões de comportamento LOTOMATON). Contextos se apresentam como os objetos básicos de LOTOS que podem ser manipulados ou transformados em LOTOMATON. Contextos são expressões de comportamento abertas representando a falta de subexpressões de comportamento. Podemos considerar a noção de contexto como uma generalização da

noção de operador, ou seja, um contexto com "n" espaços representando um operador de aridade "n".

LOTOMATON permite uma aplicação extensiva de tipos de transformações que tornam os estados explícitos. Em particular, os operadores LOTOS têm, além de sua representação original LOTOS, uma representação de transdutor baseada, por definição, em estados finitos. O operador de habilitação, por exemplo, é definido como um transdutor de dois estados:

1. o estado inicial onde somente o argumento do lado esquerdo está ativo com as seguintes ações possíveis:

   - ações diferentes de "exit" são permitidas e apresentadas sem modificação ao meio ambiente,
   - ações "exit" são permitidas e escondidas do meio ambiente (rotuladas com "i") e o transdutor move-se para o estado de terminação;

2. o estado de terminação onde somente ações do lado direito do operador de habilitação são permitidas e oferecidas, sem modificação, ao meio ambiente.

2.2.1 Sintaxe de LOTOMATON

Na sintaxe de LOTOMATON, a parte relativa aos operadores originais de LOTOS não se modifica. A sintaxe dos operadores adicionais, ou seja, os transdutores é descrita a seguir:

```
Transdutor de aridade "n":  AUT t CURST s ON (B1, ..., Bn) ENDAUT
Transdutor vazio:           AUT t CURST s ENDAUT
```

onde "t" é uma lista de transições da forma (estado, transdução, estado), "s" o estado corrente do transdutor (números naturais como identificadores de estados), e "(B1, ..., Bn)" uma lista de "n" argumentos de "aut" formada por expressões de comportamento. Transduções, os rótulos constantes das transições de "aut", formam uma espécie de linguagem imperativa simples usando a seguinte gramática.

```
transdução :=
    ação-denotação                          (* ação simples *)
|   @j                                      (* ação do argumento "j" *)
|   ⊥                                       (* ação impossível *)
|   [guarda] :: transdução                  (* ação guardada por valor *)
|   x1=E1, ..., xn=En :: transdução         (* pré-atribuição de ação *)
|   is{g1, ..., gn} :: transdução           (* ação guardada por portas *)
|   not{g1, ..., gn} :: transdução          (* ação guardada por portas *)
|   relab{g1/h1, ..., gn/hn} :: transdução
                                            (* renomeação de portas *)
|   transdução || ... || transdução         (* sincronização de ações *)
```

onde o número "j" em "@j", não deve exceder a aridade do transdutor. Por exemplo, "@j" não é permitido em transduções ocorrendo em transdutores vazios. Chama-se de ações simbólicas as transduções onde não ocorrem ações do tipo "@j".

2.2.2 Exemplos de Transformações

LOTOMATON aumenta as possibilidades de transformações LOTOS para LOTOS através da inclusão de suas transformações específicas envolvendo o operador "transdutor". Três transformações básicas, formalmente definidas em [8,10], são descritas abaixo:

1. **Substituição de um operador LOTOS**

Cada operador LOTOS de aridade *n* tem um *n-transdutor* equivalente. Esta equivalência do operador é derivada naturalmente de sua semântica.

Exemplos:

```
stop                 :   AUT CURST 0 ENDAUT

g!x;B                :   AUT (0, g!x,    1)
                             (1, @1,     1)
                         CURST 0 ON B ENDAUT

[guarda] -> B        :   AUT (0, [guarda]    ::  @1, 1)
                             (1,             @1,     1)
                         CURST 0 ON B ENDAUT

B1[]B2               :   AUT (0, @1, 1)
                             (1, @1, 1)
                             (0, @2, 2)
                             (2, @2, 2)
                         CURST 0 ON (B1, B2) ENDAUT

B1>>B2               :   AUT (0, not{exit}   :: @1,  0)
                             (0, relabel{i/exit} :: (is{exit} :: @1,1)
                             (1, @2, 1)
                         CURST 0 ON (B1, B2) ENDAUT
```

2. **Fusão do transdutor com um dos argumentos**

Em contraste com o tipo de transformação apresentada previamente, a fusão de dois transdutores permite reduzir o número de nós na hierarquia de máquinas de estados finitos. O procedimento de fusão aplica-se a um *n-transdutor* tendo como um de seus argumentos um outro *k-transdutor*. A fusão destes trandutors, ou seja, o operador e o seu argumento, resulta num transdutor de aridade *(n+k-1)* possuindo os argumentos dos transdutores fusionados. O comportamento conjunto dos transdutores antes da fusão é simulado pelo transdutor resultante.

3. **Remoção da recursão**

A remoção da recursão apresenta-se como uma das transformações básicas de LOTOMATON. A idéia proposta é de uma transformação mínima, no sentido de que

modifica apenas as partes da definição do processo onde aparece a recursão sem alteração das demais. Considere a seguinte definição de um processo recursivo:

```
PROCESS P[...](...) := C(B1, B2, RE-INST) ENDPROC
```

onde "C" define um contexto com tres expressões de comportamento abertas, **RE-INST** uma instanciação de **P** ou de um processo que chama **P**, e **B1** e **B2** duas expressões de comportamento que não chamam **P**. Remover a recursão da definição de **P** significa encontrar um transdutor, o qual equivale a C(**B1**, **B2**, **RE-INST**) mas não tem nenhum dos seus argumentos chamando ou instanciando **P**. Se este transdutor existe, ele não se apresenta, em geral, como único. Em LOTOMATON, sob algumas condições detetáveis sintaticamente, existe uma transformação mínima a qual remove a recursão de **P** sem afetar **B1** e **B2**. A definição não recursiva resultante teria a seguinte forma final:

```
PROCESS P[...](...) := AUT ... ON (B1, B2) ENDAUT ENDPROC
```

3 Descrição do Sistema de Segurança de Museu

3.1 Funcionalidades do Sistema

A aplicação proposta, como exemplo, neste artigo é a de um sistema para segurança de museu. Quando um visitante entra no museu, um sensor sinaliza este evento, o mesmo ocorre quando um visitante deixa o museu. Desta maneira o sistema é capaz de informar o número de visitantes presentes no museu a cada instante. Este sistema opera em um de seus dois modos: o modo dia (**Day**) ou o modo noite (**Night**). Ele pode ser chaveado do modo dia, no qual visitantes podem entrar/sair do museu e um contador mostra o número de visitates presentes, para o modo noite, no qual se supõe que nenhum visitante permanece no museu e nem poderá entrar ou sair dele. Um alarme será disparado se o modo noite for ativado e ainda restarem visitantes no museu. O alarme será também disparado quando um visitante entrar ou deixar o museu durante o modo noite.

3.2 Um Descrição Refinada em LOTOS do Sistema

A descrição apresentada foi obtida depois de alguns passos de refinamento. Uma decomposição prévia da especificação abstrata permitiu exibir o processo **Counter** que descreve o comportamento de uma peça disponível de hardware. O processo **Counter** tem quatro portas: porta **incr** que é mapeada para o sensor **visitor_in**, porta **decr** que é mapeada para o sensor **visitor_out**, portas **get** e **val** que estão conectadas ao computador que suporta o resto do sistema de segurança. O valor do contador é oferecido no porta **val** quando a porta **get** é estimulada. A descrição LOTOS do processo **Counter** é apresentada a seguir:

```
PROCESS Counter [incr,decr,get,val] (n:nat) : NOEXIT :=
        incr ; val!succ(n) ; Counter [incr,decr,get,val] (succ(n))
    []  decr ; val!pred(n) ; Counter [incr,decr,get,val] (pred(n))
    []  get  ; val!n ;       Counter [incr,decr,get,val] (n)
ENDPROC
```

Como especificação em LOTOS completa do sistema temos:

```
SPECIFICATION SecuritySystem [display,bell,switch,visitor_in,visitor_out]
    : NOEXIT

TYPE            (* Definições dos Tipos Abstratos de Dados *)
...
ENDTYPE
...

BEHAVIOUR
    HIDE channel IN
            Console [switch,display,bell,channel] (0)
    |[channel,switch]|
            Counter [visitor_in,visitor_out,switch,channel] (0)
WHERE
    PROCESS Console [switch,display,bell,channel] (x:nat) : NOEXIT :=
                            Day [switch,display,     channel] (x)
    >> ACCEPT y:Nat IN    Night [switch,        bell,channel] (y)
    >> ACCEPT z:Nat IN Console [switch,display,bell,channel] (z)
    WHERE
        PROCESS Day [switch,display,channel] (x:Nat) : EXIT (Nat) :=
        display!x   (   channel?y:Nat ; Day [switch,display,channel] (y)
                    []  switch ; channel?y:Nat ; exit (y)
                    )
        ENDPROC
        PROCESS Night [switch,bell,channel] (x:Nat) : EXIT (Nat) :=
            [x eq 0] ->          Night_loop [switch,bell,channel]
        []  [x ne 0] -> bell ;   Night_loop [switch,bell,channel]
        WHERE
            PROCESS Night_loop [switch,bell,channel] : EXIT (Nat) :=
                channel?y:Nat ; bell ; Night_loop [switch,bell,channel]
            []  switch ; channel?y:Nat ; exit (y)
            ENDPROC
    ENDPROC
    PROCESS Counter
        ...                 (* como descrito anteriormente *)
    ENDPROC
ENDSPEC
```

4 Transformação de "three-way" para "two-way rendez-vous"

Na especificação **SecuritySystem** a porta **switch** está envolvida em uma sincronização "three-way". De fato, tanto o processo **Counter** quanto o processo **Console** atuam sobre a porta **switch**, além de que esta porta está aberta para intervenção externa. Isto significa que uma comunicação externa sobre a porta **switch** modela a presença de uma chave física que se comunica com os dois processos. A aplicação dos cinco critérios, descritos anteriormente, para escolha de uma solução para este caso deve ser tentada:

1. não existe concordância sobre os valores;

2. a porta **switch** está presente em ambos os processos numa escolha não-determinística;

3. a concordância sobre os valores está aberta para os processos do ambiente;

4. os processos **Counter** e **Console** assumem que depois da comunicação "multi-way" sobre **switch**, o parceiro está pronto para comunicar através da porta **channel** o conteúdo do contador. Por outro lado, nehuma suposição é feita sobre o ambiente externo (o terceiro parceiro). Uma sincronização completa sobre os três processos é necessária;

5. neste caso, não se tem nenhum interesse particular na arquitetura (centralizada/distribuída) da solução.

Infelizmente, seguindo a avaliação dos cinco critérios, só se encontra no catálogo Lotosphere soluções de natureza bastante complexa. Além do mais, esta solução poderia violar os requerimentos de que, sendo uma peça do hardware, o processo **Counter** não poderia ser modificado.

Para iniciar foi considerada a única solução do catálogo que deixa um dos processos comunicantes sem modificação, independente da renomeação das portas. Esta solução é a mais simples, proposta para comunicação de difusão na ausência de requisitos completos de sincronização. Nesta solução processos que recebem não são modificados, e a comunicação "multi-way" do processo que envia é substituída por uma sequência de comunicações para cada processo recebedor.

Esta solução pode ser aplicada considerando (o primeiro critério) que não existe comunicação de valores na porta **switch** e que, por conseguinte, a escolha de qual será o processo que envia pode ser feita arbitrariamente. O processo **Console** foi considerado como aquele que envia, então nem o processo **Counter** nem o ambiente externo precisam ser modificados. Consequentemente, cada ocorrência da ação **switch** em **Console** é substituída pela sequência de ações "**switch** ; **req**" , onde **switch** é ainda a porta que está aberta para o ambiente externo e **req** é uma nova porta de **Console** que está conectada à porta com mesmo nome de **Counter** que está escondida do ambiente externo.

Entretanto, neste caso, o requisito completo de sincronização não foi satisfeito. Se o ambiente externo executa na sequência as ações "**visitor_in** ; **switch**", o processo

Counter fica amarrado à oferta do conteúdo através de **channel** enquanto o processo **Console** está fazendo uma requisição em **req**. Desta maneira, uma condição de impasse é, então, criada.

Esta condição de impasse pode ser eliminada simplesmente quando se permite a **Console** a possibilidade de recuperar imediatamente o valor em **channel** quando o sinal sobre **switch** foi trocado com o ambiente externo. Isto é obtido pela substituição de cada ocorrência da sequência "**switch** ; **channel**?y:Nat ; ..." em **Console** pela alternativa "**switch** ; (**req** ; **channel**?y:Nat ; ... [] **channel**?y:Nat ; ...)".

A especificação resultante terá o seguinte formato:

```
SPECIFICATION   TransformedSecuritySystem
          [display,bell,switch,visitor_in,visitor_out] : NOEXIT

TYPE            (* Definições dos Tipos Abstratos de Dados *)

...
ENDTYPE
...

BEHAVIOUR

    HIDE channel,req IN
        Console [switch,req,display,bell,channel] (0)
    |[cahnnel,req]|
        Counter [visitor_in,visitor_out,req,channel] (0)

WHERE

    PROCESS Console [switch,req,display,bell,channel] (x:Nat) : NOEXIT :=
                          Day [switch,req,display,     channel] (x)
    >> ACCEPT y:Nat IN   Night [switch,req,        bell,channel] (y)
    >> ACCEPT z:Nat IN Console [switch,req,display,bell,channel] (z)

    WHERE

        PROCESS Day [switch,req,display,channel] (x:Nat) : EXIT (Nat) :=
        display!x ; (   channel?y:Nat ; Day [switch,req,display,channel] (y)
                    [] switch ;    (   req ; channel?y:Nat ; exit (y)
                                   [] channel?y:Nat ; exit (y) ) )

        ENDPROC
```

```
        PROCESS Night [switch,req,bell,channel] (x:Nat) : EXIT (Nat) :=

            [x eq 0] ->          Night_loop [switch,req,bell,channel]
        []  [x ne 0] ->bell ;    Night_loop [switch,req,bell,channel]

        WHERE

            PROCESS Night_loop [switch,req,bell,channel] : EXIT (Nat) :=
                channel?y:Nat ; bell ; Night_loop [switch,req,bell,channel]
            []  switch ;     (req ; channel?y:Nat ; exit (y)
                             [] channel?y:Nat ; exit (y) )

            ENDPROC
        ENDPROC
    ENDPROC

    PROCESS Counter          (* Processo Counter NÃO MODIFICADO *)

        ...
    ENDPROC
ENDSPEC
```

5 Transformação de LOTOS para LOTOMATON

Considere agora a especificação orientada à implementação **TransformedSecuritySystem**. Nesta etapa do ciclo de projeto se apresenta a seguinte situação:

- o processo **Counter** tem sua realização baseada em hardware;
- fica faltando a realização do processo **Console** que inclui sua comunicação com o processo **Counter**.

O processo **Console** é uma composição sequencial cíclica dos processos **Day** e **Night**. Cada um dos dois processos continua ativo por um longo período de tempo até que um chaveamento ocorre e o outro processo é ativado. Assim, uma realização natural de **Console** poderia ser feita considerando três "processos operacionais": um processo para operar o modo dia, um outro para o modo noite, e um terceiro processo que gerencia os dois primeiros e mantém o contexto de execução (canais estabelecidos com **Counter**, recursos de exibição, etc.).

Esta implementação distribuída em três processos pode ser obtida usando o esquema transformacional LOTOMATON. Primeiro, a recursão exixtente é removida de cada um dos processos **Day** e **Night**. O resultado apresenta um par de máquinas de estados finitos expandidas (ou transdutores nulários), mostradas a seguir.

```
PROCESS Day [switch,req,display,channel] (x:Nat) : EXIT (Nat) :=

    AUT (0, n:Nat=x :: display!n,          1)
        (1,         channel?m:Nat,         2)
        (2, n=m     :: display!n,          1)
        (1,         switch,                3)
        (3,         req,                   4)
        (4,         channel?m:Nat,         5)
        (3,         channel?m:Nat,         5)
        (5,         exit(m),               0)
    CURST 0
    ENDAUT


PROCESS Night [switch,req,bell,channel] (y:Nat) : EXIT (Nat) :=

    AUT (0, k:Nat=y :: [k ne 0] :: bell,           1)
        (0, k:Nat=y :: [k eq 0] :: channel?k:Nat,  2)
        (0, k:Nat=y :: [k eq 0] :: switch,         3)
        (1,                        channel?j:Nat,  2)
        (2,                        bell,           1)
        (1,                        switch,         3)
        (3,                        req,            4)
        (4,                        channel?j:Nat,  5)
        (3,                        channel?j:Nat,  5)
        (5,                        exit(j),        0)
    CURST 0
    ENDAUT
```

Considerando, agora, a definição do processo **Console**:

```
PROCESS Console [switch,req,display,bell,channel] (x:Nat) : NOEXIT :=

                        Day [switch,req,display,     channel] (x)
ACCEPT y:Nat IN    Night [switch,req,        bell,channel] (y)

ACCEPT z:Nat IN Console [switch,req,display,bell,channel] (z,
```

Notamos que ela está na forma apresentada na seção 2.2.2:

```
PROCESS P [ ... ] ( ... ) ... := C ( B1, B2, RE-INST ) ENDPROC

onde      B1      =     Day [switch,req,display,     channel] (x)
          B2      =   Night [switch,req,        bell,channel] (y)
     RE-INST      = Console [switch,req,display,bell,channel] (z)
```

A seção 2.2.2 forneceu algumas indicações de como proceder para remover a recursão destas definições de processos. Uma definição não-recursiva, fortemente equivalente, de **Console** é mostrada abaixo:

```
PROCESSO Console [switch,req,display,bell,channel] (x:Nat) : NOEXIT :=

    AUT (0,                    @1,                                  1)
        (1, not{exit}          :: @1,                               1)
        (1, relab{i/exit}      :: (is{exit} :: @1 || exit?y),       2)
        (2, not{exit}          :: @2,                               2)
        (2, relab{i/exit}      :: (is{exit} :: @2 || exit?x),       1)
    CURST 0 ON (      Day [switch,req,display,      channel] (x),
                    Night [switch,req,          bell,channel] (y) )
    ENDAUT
```

Desta maneira obteve-se os três processos **Console**, **Day** e **Night** em seus formatos transdutores orientados à implementação. O usuário pode escolher como mapear estes três processos LOTOMATON em três "processos operacionais". O usuário pode achar também que é mais apropriado fusionar estes três processos em um único processo e implementar este processo resultante. No caso de **Console** a fusão completa tem a sua justificativa: o tamanho do código resultante é menor e não vale a pena considerar os três processos em separado por causa do "overhead" dos seus gerenciamentos. Segue-se o código LOTOMATON resultante para os processos fusionados.

```
Console [switch,req,display,bell,channel] (0) =

    AUT (0, x:Nat=0 :: display!x,                  1)
        (1,            channel?x:Nat,              2)
        (2,            display!x,                  1)
        (1,            switch,                     3)
        (3,            req,                        4)
        (4,            channel?x:Nat,              5)
        (3,            channel?x:Nat,              5)
        (5, [x ne 0]    :: bell,                   6)
        (5, [x eq 0]    :: channel?x:Nat,          7)
        (5, [x eq 0]    :: switch,                 8)
        (6,            channel?x:Nat,              7)
        (7,            bell,                       6)
        (6,            switch,                     8)
        (8,            req,                        9)
        (9,            channel?x:Nat,              2)
        (8,            chanell?x:Nat,              2)
    CURST 0
    ENDAUT
```

6 Considerações sobre a Implementação

O objetivo principal desta fase final consiste na tradução de uma especificação LOTOMATON em um programa executável no ambiente alvo. Nesta seção, vamos tentar discutir com algum detalhe os passos de realização e as decisões envolvidas nesta fase da trajetória de projeto. O ponto inicial desta implementação está representado pela expressão LOTOS/LOTOMATON:

```
HIDE channel,req IN
        Console [switch,req,display,bell,channel] (0)
    |[channel,req]|
        Counter [visitor_in,visitor_out,req,channel] (0)
```

Nas transformações realizadas anteriormente obtivemos os seguintes resultados:

- **Console** e **Counter** tem comunicação binária privada pelas portas **req** (de **Console** para **Counter**) e **channel** (de **Counter** para **Console**); e
- **Console** está num formato de máquina de estados finitos expandida. Isto significa que a recursão foi removida e pode-se atribuir novos valores às variáveis.

O ambiente operacional escolhido para a implantação de especificações é o ambiente Unix [4]. A linguagem C [7] foi escolhida como linguagem de implementação. A realização completa, em C, do sistema apresentado neste artigo pode ser encontrada em [5].

O sistema **SecuritySystem** executa em uma máquina Unix. O processo **Console** está implantado em um processo operacional Unix. **Counter** é suportado por uma peça de hardware que está conectado a máquina Unix através de um ligação assíncrona RS232, bi-direcional. Desta maneira, o "rendez-vous" entre **Console** e **Counter** se processa por meio de trocas de mensagens assíncronas.

Para a implementação de **SecuritySystem** são realizados os seguintes mapeamentos entre portas LOTOS e o ambiente real:

- as portas **req** e **channel** para o dispositivo de entrada/saída "/dev/ttyb" adaptando o esquema de comunicação existente entre **Console** e **Counter**,
- a porta **switch** para a entrada standard (o teclado),
- a porta **display** para a saída standard (a tela),
- a porta **bell** para o dispositivo "/dev/audio"

A realização, na linguagem C, da máquina de estados finitos expandida descrevendo **Console** é quase direta ou mesmo automatica. Adaptações tiveram que ser realizadas no autômata de maneira a substituir a comunicação baseada em "rendez-vous" pela passagem de mensagens assíncrona. É importante notar que a realização ideal do "rendez-vous" dependerá da velocidade com a qual mensagens são lidas por **Console**, ou seja, quanto mais rápida for a leitura melhor será a implementação do "rendez-vous".

7 Conclusões

Neste trabalho, acreditamos que ficou comprovada a utilidade das transformações de especificações LOTOS com preservação da semântica para as últimas fases de desenvolvimento de sistemas. A aplicação de transformações tem mostrado que o processo de transformações passo-a-passo não pode ser completamente automatizado, desde que, para certos casos, transformações não são aplicáveis nem imediatamente nem mecanicamente. Ao invés disso, elas requerem uma avaliação cuidadosa das condições

para a sua aplicação e uma escolha, também, cuidadosa das técnicas de transformação disponíveis.

Através de um exemplo simples e significativo, mostramos como implementar um sistema especificado em LOTOS sobre uma máquina Unix. O mapeamento das portas LOTOS sobre disposivivos de Entrada/Saída e das máquinas de estados finitos expandidas, resultantes das transformações aplicadas sobre a especificação LOTOS original, em processos Unix ou em dispositivos de hardware conectados à máquina hospedeira, possibilitou a derivação completa do sistema real. Para isto, necessitamos da modificação da comunicação baseada em "rendez-vous" para uma baseada em troca de mensagens assíncronas

Bibliografia

[1] BOLOGNESI, T., BRINKSMA, E. Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems, 14: 25-29, 1987.

[2] BOLOGNESI, T., BOCHM, P., FANTECHI, A., NAJM, E. (Editores). **Correctness Preserving Transformations** - First deliverable of Task T1.2 of project ESPRIT 2034 - Lotosphere. Reference: Lo/WP1/T1.2/N0020, April 1990.

[3] BOLOGNESI, T. (Editor). **Correctness Preserving Transformations** - Third deliverable of Task T1.2 of project ESPRIT 2034 - Lotosphere. Reference: Lo/WP1/T1.2/N0045, April 1992.

[4] BOURNE, S. R. **The Unix System**. Addison-Wesley, Reading, 1982.

[5] CUNHA, P., FANTECHI, A., MEKHANET, B., NAJM, E., QUEIROZ, J. Correctness Preserving Transformations for the Late Phases of Development. In: Lotosphere Workshop,3. Pisa, September 1992.

[6] ISO. Information Processing Systems - Open Systems Interconection: **LOTOS - A Formal Description Technique Based on the Temporal Ordering of Obsevational Behaviour.** International Standard - IS 8807, 1988.

[7] KERNIGHAN, B. W., RITCHIE, D. M. **The C Programming Language**. Prentice-Hall, Englewood Cliffs, 1988.

[8] NAJM, Elie, QUEIROZ, José, SERHROUCHNI, Ahmed. The Pre-Implementation and Verification of LOTOS. In: IFIP TC6 International Conference on Computer Networking. Budapeste, Hungria, Maio de 1990.

[9] NAJM, E. Transforming contexts in LOTOMATON. ESPRIT/Lotosphere project. Reference: Lo/WP1/T1.2/INRIA/N008, October 1990.

[10] NAJM, E., LAKAS, A., MADELAINE, E., SERHROUCHNI, A., de SIMONE, R. ALTO: An Interactive Transformation Tool for LOTOS and LOTOMATON. In: Lotosphere Workshop,3. Pisa, September 1992.

[11] QUEIROZ, J., SERHROUCHNI, A., CUNHA, P. NAJM, E. PIL: A Tool for Pre-implementation of LOTOS. In: International Conference on Formal Description Techniques - FORTE'90, 3 . Madri, Espanha, Novembro de 1990.

[12] SISTO, R., CIMINIERA L., VALENZANO A. *A Protocol for Multirendezvous of LOTOS Processes.* IEEE Transactions on Computers, 40 (1):437-447, April 1991.