7
Conclusão

Neste trabalho propomos uma arquitetura que objetiva facilitar o processo de verificação formal de protocolos e algoritmos distribuídos. A motivação inicial foi a dificuldade encontrada na verificação do protocolo RDP (ESO00) utilizando o verificador de modelos Spin (Hol97). Nesta verificação, a especificação do RDP ficou razoavelmente grande (em torno de 500 linhas código em Promela) e com trechos da especificação que não eram relevantes para a essência do protocolo (relacionados ao verificador), o que comprometia a legibilidade e a confiança no que tinha sido especificado.

A arquitetura em camadas proposta é similar às arquiteturas apresentadas em (BGM02, JHA+96, Cas98, BGL+00), por exemplo. Entretanto, uma diferença significante é o foco da nossa arquitetura, voltada para a linguagem de entrada LEP, enquanto que nas outras abordagens o foco é a linguagem intermediária. No nosso caso, abdicamos de transformações intermediárias menos complexas para prover uma linguagem com construções de mais alto-nível.

Em LEP, dentre outras construções, propomos um mecanismo para a definição das topologias baseada em gramáticas de grafo e de atributos (AGG). A junção deste mecanismo com um ambiente de verificação permite maior generalização do processo de análise formal dos protocolos, o que resulta em maior confiança no correto funcionamento destes. Entretanto, alguns pontos precisam ser levados em consideração como o comportamento dinâmico da rede. No caso da criação/destruição dinâmica de nós, na situação atual da arquitetura, poderemos ter uma sobreposição da computação das topologias. Ou seja, um mesmo número de nós numa rede poderia ser verificada mais de uma vez. Nesta situação, a solução é definirmos a rede de forma explícita, ou utilizarmos alguma das macros para definição de redes sem faixa de valores, de forma que apenas uma rede seja gerada.

A especificação de uma gramática de grafos com atributos requer uma certa familiaridade do usuário com os conceitos de gramática de grafos e gramática de atributos. Na arquitetura disponibilizamos algumas macros para que o usuário, na maioria dos casos, não precise entender este funcionamento. Acreditamos assim que a definição de uma topologia para ser utilizada na

verificação de um protocolo fica compatível com o nível de abstração de LEP.

Com respeito à modelagem da movimentação em LEP, esta pode ser feita de forma explícita ou implícita (parâmetros *static* e *dynamic* da topologia, respectivamente). Para a movimentação implícita (transparente para o projetista do protocolo), a arquitetura gera a movimentação seguindo o modelo de mobilidade aleatório (*Random Walk Mobility Model*) (CBD02). Entretanto, combinando a exploração exaustiva dos verificadores de modelos e o modelo de mobilidade aleatório, temos a verificação de todas as movimentações possíveis na rede. A implementação de outros modelos de mobilidade, geralmente mais específicos (baseados em cenários, movimentação de grupos, restrições de tempo, etc), exigiria que a especificação fosse menos abstrata, o que não é o objetivo de LEP. Como um trabalho futuro poderíamos buscar uma maneira menos particular de agregar diferentes modelos de mobilidade à especificação em LEP.

Outra característica que não é contemplada por LEP é a definição de topologias híbridas. Ou seja, topologias com formatos ou comportamentos distintos. Por exemplo, se observarmos a confiabilidade da conexão entre estações base e estações móveis numa rede móvel. No estudo de caso apresentado (seção 4.1.1), estas diferenças são definidas na própria especificação. Entretanto, uma definição mais alto nível, assim como fazemos na definição da topologia, seria mais adequada.

Na especificação dos módulos, usualmente especificamos a inicialização dos módulos em LEP de duas (2) maneiras, para os exemplos verificados: através da transição *init* e através de transições não-determinísticas (pré-condição *true*). A primeira opção é mais comportada, pois colocamos no sistema apenas as mensagens que temos interesse, as quais serão simuladas em todas as ordens possíveis, seguindo a idéia de exploração exaustiva dos verificadores de modelos. O segundo tipo de inicialização é mais aleatório. Este tipo pode ser considerado como um *teste de estresse* da verificação, onde não só a ordem, como também a quantidade e a frequência das mensagens colocadas na rede são todas as possíveis.

Sobre a utilização de pronomes, as especificações em LEP sempre se mostraram mais compactas e legíveis para os casos analisados. A utilização de pronomes como *everyone*, *neighbours* e *any(k)* permite uma rápida especificação de cenários complexos de sistemas concorrentes como *broadcast* e *multicast* de mensagens. Entretanto, nenhuma otimização foi feita na geração do código destes pronomes. Ou seja, o problema de explosão de estados não foi efetivamente abordado, apesar da falsa impressão dada pelas especificações compactas de LEP.

Os pronomes também serviram para simplificar as propriedades sobre

os protocolos em LEP. Entretanto, é preciso estar alerta para não especificar além do desejado, como na utilização de variáveis em propriedades. Uma combinação interessante, comentada na seção de especificação de propriedades (seção 3.2.4), é a utilização de PSL (Property Specification Language) (Acc04) combinada com pronomes em propriedades. Assim, por exemplo, uma propriedade descrita da forma abaixo:

$$[]\ (p_1!alive(p_2)\ \text{and}\ p_1!alive(p_3)\ \text{and}\ p_1!alive(p_4)\ \text{and}\ p_1!alive(p_5)) \rightarrow <>(p_1?ack(p_2)\ \text{or}\ p_1?ack(p_3)\ \text{or}\ p_1?ack(p_4)\ \text{or}\ p_1?ack(p_5))$$

, poderia ser escrita como:

Always (p!alive(everyone) implies eventually p?ack(any))

Neste caso, apenas o termo *implies*, que substitui a implicação (→), não faz parte nem de LEP e nem de PSL mas, como previsível, pode ser facilmente inserido. Este enfoque se aproxima de trabalhos como (DCN+00), que propõe o uso de linguística para a manipulação de ferramentas de verificação formal.

Como vimos, originalmente tanto Spin quanto SMV, diferentemente de algumas de suas variantes (TC96, MS04), não contemplam a especificação de restrições de tempo-real. Geralmente, a modelagem de restrições de tempo (por exemplo, *timeout*) em linguagens de especificação que não provêm tais construções é feita através de não-determinismo. Ou seja, ao invés de uma condição se tornar válida ou inválida num determinado período de tempo, criamos um não-determinismo para que a condição se torne válida ou inválida em algum momento (indeterminado). Por exemplo, na figura 7.1 temos a especificação de um leilão.

Num leilão usual, uma situação que caracteriza o término do leilão é o leiloeiro receber algum lance e, após um intervalo de tempo sem lances, encerrar o leilão. Na especificação em LEP, para modelarmos este comportamento, permitimos que o leiloeiro encerre o leilão a qualquer momento, desde que algum comprador já tenha dado algum lance (linha 13 da figura 7.1). Salvo a questão do tempo, a implementação em LEP é equivalente ao funcionamento comum dos leilões em função do comportamento exploratório dos verificadores de modelos.

Um aspecto a ser observado na especificação é a ausência de variáveis globais, já que LEP foi projetada para a especificação de protocolos em ambientes distribuídos. Entretanto, das traduções intermediárias até a linguagem de entrada do verificador adotado permitimos a utilização de variáveis globais, desde que estas só auxiliem na verificação de propriedades. Ou seja, estas variáveis globais não devem ter influência no comportamento do protocolo, já

```
1  topology is Star(Leiloeiro:1,Comprador:1..*);
2
3  module Leiloeiro
4          int valor=0, max;
5          Comprador ganhador;
6          init -> max=10; everyone!lance_inicial(max);
7          this?lance(value) ->
8                  if (valor > max) then
9                          max = valor;
10                         ganhador = sender;
11                         everyone!lance_inicial(max);
12                 endif
13         true -> if ganhador <> none then
14                         ganhador!win;
15                 endif
16 endmodule
17
18 module Comprador
19         int valor;
20         this?lance_inicial(valor) ->
21                 sender!lance(valor+10);
22         this?lance_inicial(valor) -> stop;
23         this?lance_inicial(valor) ->
24         this?win ->
25 endmodule
26
27 <> (leiloeiro.max<10) and leiloeiro!win
28 <> (comprador?win(leiloeiro) and (leiloeiro.valor<comprador.valor))
```

Figura 7.1: Especificação de um leilão em LEP

que supomos um ambiente distribuído. Esta distinção, apesar de clara, não é tão evidente quando especificamos um protocolo diretamente na ferramenta de verificação, como feito no início deste trabalho, já que o uso das variáveis locais e globais na especificação ocorre de forma indiscriminada.

Quanto às ferramentas de verificação de modelos Spin e NuSMV, ambas foram satisfatórias na verificação dos protocolos e algoritmos investigados. A linguagem de especificação Promela, assim como LEP, é baseada em cálculo de processos, enquanto que a linguagem de especificação do NuSMV é baseada em sistemas de transição. Esta diferença implicou em mudanças sutis na maneira de implementarmos as traduções dos protocolos na arquitetura, já que, por exemplo, NuSMV não tem o conceito de canal para troca de mensagens. Ou seja, a comunicação entre processos teve que ser simulada. Do ponto de vista de projeto, Promela está bem mais próxima de uma linguagem de programação do que NuSMV. Uma prova disso é a existência de construções em Promela que permitem a inserção de código explícito em linguagem C. Para as especificações apresentadas, não percebemos a necessidade de utilização de tais construções. Sobre as propriedades, em Spin, LTL é usada como a lógica temporal para a especificação de propriedades, enquanto que em NuSMV, tanto LTL quanto CTL podem ser utilizadas. Entretanto, apesar destas lógicas

não serem equivalentes (existem fórmulas de CTL que não são expressas em LTL e vice-versa), as propriedades comumente especificadas para sistemas concorrentes podem ser especificadas em ambas (DAC98).

A respeito dos contra-exemplos no nível de LEP, acreditamos que o fato destes se constituírem de comandos existentes na especificação seja fundamental para um melhor entendimento destas sequências de simulações da especificação por parte do projetista. Entretanto, a utilização de variáveis nestes contra-exemplos, as quais podem substituir elementos do protocolo que não são diretamente referenciados pela propriedade, pode tornar o contra-exemplo excessivamente abstrato. Na tese nos restringimos apenas à apresentação de uma nova abordagem para a interpretação destes contra-exemplos, ficando um estudo mais criterioso como trabalho futuro.

Ainda como trabalhos futuros podemos ter: uma versão de LEP para sistemas de tempo-real e probabilísticos; a utilização de outras ferramentas de verificação formal, como provadores de teoremas, que funcionem como *backend* da arquitetura, assim como os verificadores de modelos; otimização do código gerado pela arquitetura com o objetivo de amenizar o problema de explosão de estados dos verificadores para especificações mais complexas, e; tradução de LEP para a linguagem intermediária de outros ambientes de validação, como (BGM02, KG02, BGL$^+$00).