7 Conclusão

Neste trabalho investigamos a geração automática de suíte de teste para interface gráfica a partir de Rede de Petri. Essa abordagem de teste é conhecida como testes baseados em modelo, ou *Model-Based Testing* (MBT) e requer que a estrutura e o comportamento do software (ou parte dele) tenham sido formalizados em modelos com regras bem definidas.

A Rede de Petri de Alto Nível demonstrou-se apropriada como linguagem de especificação de GUIs porque consegue representar sem restrições a sua estrutura e o seu comportamento. O diferencial em relação a outros modelos existentes é o fato de a RP ter um conjunto de atributos importantes para expressar as características de uma GUI. Enquanto que outros modelos não conseguem reunir todos esses atributos.

O atributos importantes de Redes de Petri são:

- É capaz de lidar com situações paralelas e não-determinísticas, que é o caso de uma interface gráfica, onde diversos eventos, inclusive externos, estão aptos a ocorrer num determinado instante e sem uma ordem conhecida.
- O item acima também evidencia que a partir de um determinado estado, a RP não vai para um determinado outro estado fixo. Em uma RP a partir de um estado, pode-se ir para uma série de outros, dependendo de qual evento será disparado e com que parâmetros.
- O estado da Rede de Petri, não é como em outros diagramas, onde um único elemento gráfico no modelo representa o estado corrente do objeto modelado. Na RP, o estado é representado por diversos lugares na rede com *tokens* valorados, representando estados de *widgets*. Isso é coerente com o estado de uma interface gráfica, que é dado pelo conjunto dos estados dos *widgets*.
- A Rede de Petri de Alto Nível é versátil na representação de um sistema, porque permite a definição de expressões condição de guarda das transições. Isso representa os valores resultantes de um processamento quando ocorre um evento.

- A Rede de Petri possui o mecanismo de execução do modelo, o qual permite a verificação da corretude do modelo e da especificação, antes mesmo de se desenvolver a funcionalidade, bem como a geração de casos de teste representados pelos caminhos percorridos na execução.

Neste trabalho foi preciso definir a semântica dos lugares na RP, a sintaxe e as regras para a modelagem, de forma que fosse possível para a ferramenta guiftG interpretar os arquivos de estrutura e de execução (log) da rede. O padrão de programação a ser seguido estabelece que os lugares representam um *widget* e uma propriedade sua. Um exemplo da sintaxe adotada são os nomes dos lugares e transições, que devem ter um nome válido de propriedade como prefixo, seguido de um *underscore* e sucedido pelo nome do wigdet. O final do nome de um *widget* deve conter o seu tipo para que seja identificado na ferramenta guiftG, por exemplo: applyButton, firstValueTextBox, volumesComboBox, etc.

A partir das pesquisas realizadas acerca do tema testes de GUI, das representações experimentais de GUIs como Redes de Petri, das aplicações práticas num software real e das observações sobre os resultados obtidos, chegamos às seguintes conclusões:

- Quanto à modelagem na prática, a Rede de Petri de alto nível demonstrou ser apropriada à representação de GUIs em geral, pelos atributos citados acima. Alguns problemas com os quais nos deparamos estão descritos na próxima seção 7.1, mas não invalidam essa constatação, pois são problemas contornáveis.
- A geração dos casos de teste através da simulação automática da rede mostrou ser eficiente porque gera os casos de teste em um tempo curto. Demonstrou também alguma eficácia quando os testes gerados foram capazes de detectar problemas em interfaces já testadas (manualmente). Além disso, obtiveram resultado no escore de mutação, mesmo que modesto.
- Um dos motivos levantados para o escore de mutação dos testes não ter sido alto, está relacionado com a completeza da rede. No entanto, isso envolve a evolução da sintaxe e das regras permitidas na modelagem, considerando mais eventos internos e externos à GUI. Essa evolução diz respeito

unicamente à ferramenta guiftG, responsável por interpretar essas regras e gerar o código do teste.

- Nas interfaces testadas, os testes gerados aleatória e automaticamente obtiveram melhores resultados do que os gerados com simulação manual.
- O tamanho de modelo de uma funcionalidade real pode ficar bastante grande e complexo. Apesar de ser um problema comum em modelos que de fato pretendem ser úteis, isso pode assustar na adoção do método por pessoas novas. No entanto, esse problema torna-se mais um sobre a decisão do quanto se deseja investir em prol da qualidade.
- A Análise de Mutantes demostrou ser uma técnica interessante no controle de qualidade de testes em geral. Ela fornece uma medida para cada análise executada e permite que se adote uma meta de eficácia. Nisso, está incluído a possibilidade de se aprimorar os casos de testes criados.
- A eficácia dos casos de testes gerados pela RP:
  - Depende da corretude do modelo;
  - Depende da estratégia definida para a cobertura, seja ela aleatória ou baseada em critérios relacionados a caminhamento em grafo. Esse é um assunto complexo e alvo de estudos empíricos, porque os resultados sobre a cobertura adotada variam entre as funcionalidades de software e mesmo entre GUIs. Além disso tendem a gerar grande quantidade de casos de teste, exigindo outras técnicas para filtrá-las.
  - É diretamente proporcional à sua completeza. A completeza é relacionada aos eventos possíveis, incluindo os eventos externos que impactam na GUI sob teste, e aos dados de teste disponíveis na forma de *tokens*.
  - Se medida pela técnica Análise de Mutantes, depende de que o número de mutantes equivalentes seja fidedigno.
  - Quando gerados randomicamente, depende do número de casos de teste gerados e de sua extensão, os quais devem ser suficientemente grandes. O quão grande, deve ser medido através de experimentos.

7.1 Limitações deste trabalho

Dados os levantamentos e as conclusões acima, extraímos as limitações deste trabalho e as listamos nos tópicos seguintes.

- Nas modelagens deste trabalho, não foi possível usar todas as opções disponíveis na modelagem gráfica da ferramenta MISTA, porque a ferramenta guiftG implementada para interpretar a rede ainda não é capaz de interpretar todas as regras e expressões existentes na MISTA. Um exemplo de limitação é quando queremos definir uma expressão na condição de guarda da transição, para definir um valor numa variável de saída. A expressão permitida atualmente pode ter os seguintes formatos: var1=*constante* ou var1=var2 [+ , - , * , /] *constante*. Contudo, a MISTA permite o uso de expressões mais complexas.
- Outra restrição quanto à modelagem refere-se aos eventos permitidos até o momento. Ainda é necessário implementar outros tipos de eventos de interação da GUI, na ferramenta guiftG, para que estes possam entrar na modelagem da rede.
- A ferramenta de Análise de Mutantes é um protótipo e possui poucos operadores de mutação, porque alguns envolvem uma complexidade alta de implementação. Além disso, faltou uma análise mais fina a fim de identificar operadores, cujo código não deveria ser coberto dada a natureza dos testes gerados. Isso contribuiu para gerar mais mutantes que não são mortos, comprometendo a obtenção de um resultado melhor na eficácia.
- A falta de uma estimativa mais precisa ou de uma forma concreta de determinar o número de mutantes equivalentes.
- Ausência de mais funcionalidades nos experimentos. Não havia muitas GUIs com o código no padrão esperado para a geração dos scripts de teste.
- Ausência de comparação da eficácia, nos testes inteiramente manuais realizados atualmente no projeto e nos testes gerados pelo método proposto, nas mesmas funcionalidades. Não era possível porque não havia testes manuais para as GUIs modeladas e as GUIs que possuíam testes manuais, não podiam ser modeladas ainda devido ao problema do padrão do código.
- A falta de suítes maiores, com mais casos de teste e mais extensos. É esperado que isso nos traga uma melhora na eficácia, por se tratar de testes

aleatórios. Trabalhar com suítes grandes neste momento foi um empecilho, porque os scripts ainda exigem um trabalho considerável de ajuste manual.

7.2 Contribuições

As contribuições deste trabalho incluem:

- A formalização de regras que permitem modelar uma GUI numa Rede de Petri de Alto Nível.
- Uma ferramenta que interpreta uma RP modelada e executada na ferramenta MISTA e gera casos de teste num padrão genérico e definido em [29]. Esse script genérico permite que outros ambientes de desenvolvimento de software façam uso do método, devendo apenas implementar um módulo que gere scripts na linguagem de script utilizada no ambiente.
- A apresentação de uma abordagem para testes baseados em modelo para interface gráfica, que utiliza a Rede de Petri como especificação. As características teóricas levantadas somadas às experimentações do método em ambiente real, podem ser um atrativo para mais experimentações e uso.
- O método apresentado demonstrou-se mais vantajoso do que processos manuais de testes, uma vez que é mais eficiente na geração de testes mais abrangentes, ou seja, que envolvem sequências longas e aleatórias. São eficientes porque podem gerar quantos casos de teste se queira (bastando deixar a simulação automática rodar até quando o testador quiser), fazendo isso em poucos minutos.

7.3 Trabalhos futuros

Para evoluir o método de teste proposto neste trabalho e possibilitar a sua inserção no processo de teste de um ambiente real de desenvolvimento de software, listamos alguns trabalhos a serem realizados.

- Investigar a aplicabilidade de outra semântica, que vise diminuir o tamanho da RP, identificada ao longo deste estudo. Os lugares passam a ser apenas o *widget*, e não mais com uma propriedade associada. Os lugares então recebem um *token* estruturado. Nesse caso, o *token* passa a ter a semântica <propriedade, valor>. Um exemplo dessa nova forma de modelar a RP está ilustrado na Figura 47: O *widget* chamado volumesComboBox será repre-

sentado por um lugar de mesmo nome (agora não tem mais o prefixo com o nome da propriedade). O *token* que esse lugar irá receber, pode em um momento ter o valor <selected,3> e em outro, o valor <active,0>, por exemplo. Essa forma de representar, reduz o tamanho da rede, porque um *widget* não precisa ter tantos lugares na rede quantos forem as suas propriedades. Haverá apenas um lugar para cada *widget*. E os *tokens* conterão as propriedades que se deseja verificar.

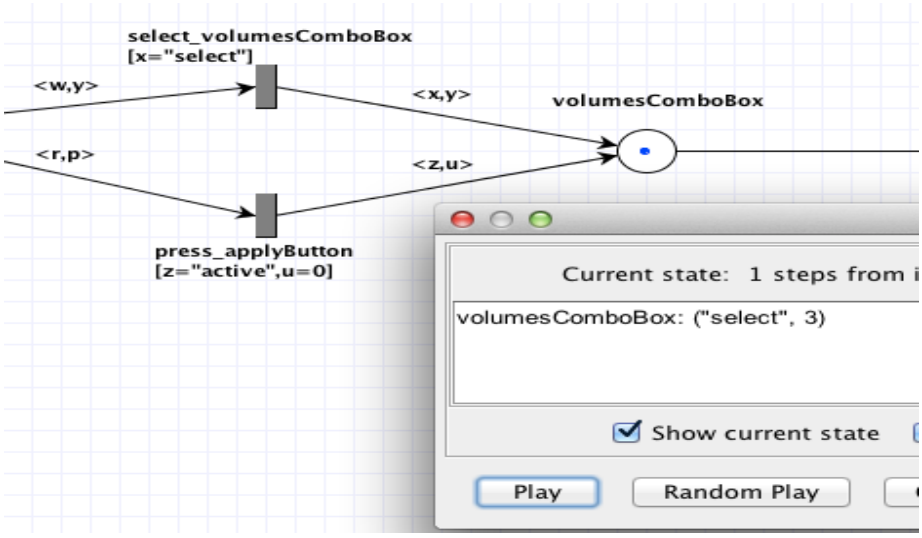

Figura 47 – Outra maneira de modelar uma RP, visando diminuir o seu tamanho

- Evoluir as regras permitidas na modelagem da RP e a ferramenta guiftG de forma a possibilitar a geração de uma suíte mais completa e mais eficaz.
- Implementar a geração de casos de teste permitindo a seleção de critérios de cobertura e comparar a eficácia das suítes geradas com os diferentes critérios. Essa comparação deve ser realizada em um mesmo software.
- Investigar a influência que o tamanho dos casos de teste exerce sobre a eficácia. Avaliar casos de teste mais extensos. (A extensão é o número de passos, ou, eventos disparados, em um ciclo da simulação da rede). Para isso deve-se fixar um número de casos de teste, por exemplo seis, e variar o seu comprimento. O ideal é fazer essa comparação em várias funcionalidades para se obter uma medição mais precisa.
- Investigar a influência que o número de casos de teste exerce sobre a eficácia. Para isso deve-se manter o número de passos fixo, por exemplo em vinte, e gerar suítes variando o número de casos de teste. Pode-se deixar a simulação rodando por mais tempo para gerar mais casos de teste. Como a

geração é aleatória, este item e o item acima podem trazer uma grande diferença para a eficácia. O ideal é fazer essa comparação também em várias funcionalidades para se obter uma medição mais precisa.

- Evoluir o protótipo MATool implementado para a Análise de Mutantes, incluindo nele mais operadores. A ferramenta também pode ser modificada com um custo baixo, para que os comandos de compilação do SUT e execução dos testes sejam configurados para outro ambiente de desenvolvimento. Assim como os operadores específicos do código de um software podem ser configuráveis.
- Na Análise de Mutantes, aplicar um método que forneça uma estimativa mais precisa do número de mutantes equivalentes. Ou mesmo, adotar na prática a verificação sobre esses mutantes.
- A fim de detectar mutantes que igualmente não contribuem para o escore da funcionalidade em teste, pode-se verificar os mutantes que estão fora do "domínio de exercício" da funcionalidade (estabelecida pela Rede de Petri) sendo testada. "Domínio de exercício" é a parcela do código do sistema que efetivamente pode vir a participar no processamento de determinada funcionalidade. Essa parcela do código é difícil de ser determinada. Pode envolver partes de uma classe e por extensão componentes (ao invés da classe inteira). No caso de métodos longos, pode envolver somente parte de métodos. Todos os mutantes que estão fora do domínio de exercício, equivalentes ou não, não poderão participar de qualquer execução da funcionalidade sob teste, portanto são equivalentes do ponto de vista desta funcionalidade. Determinar o domínio de exercício pode ser uma tarefa complexa.
- A modelagem da Rede de Petri neste trabalho requereu um tempo de aprendizado, devido às regras intrínsecas de RP, assim como requereu a idealização e experimentação dos formalismos necessários para este uso em específico. A modelagem de uma RP foi aprimorada ao longo do tempo e o tempo necessário para a sua construção foi decaindo. Seria interessante desenvolver um método de ensino para treinar pessoas a utilizar esse padrão de Rede de Petri.