Uma ferramenta para apoio gráfico na geração e simulação de Teste Baseado em Modelos

[1] **Lucas Lagôa Nogueira**[1], **Sofia Larissa da Costa Paiva**[2]

[1]Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação - São Carlos - São Paulo

[2]Universidade Federal de São João del Rei - São João del Rei - Minas Gerais

lucaslagoa@usp.br, sofia@ufsj.edu.br

Abstract. *Software testing is an important activity to be performed to ensure defect detection during software creation and provides evidence of software reliability. There are several techniques that can be applied and one of them is Model-Based Testing, however many of contributions are only theoretical, requiring a tool that supports the generation and execution of the tests in real contexts, enabling a real application of such research. The objective of this work was to design and implement a graphical interface for the tool that supports the W_{iots} method, which uses the Input-Output Transition Systems (IOTS) model. This tool provides graphical support in test generation and execution, enabling the tester to track execution results and points in which software can have problems.*

Resumo. *O Teste de Software é uma atividade importante a ser realizada para garantir a detecção de defeitos durante a criação de um software e fornece evidências de confiabilidade do software. Existem várias técnicas de teste que podem ser aplicadas e uma delas é o Teste Baseado em Modelos, mas muitas das contribuições são apenas teóricas, tendo a necessidade de uma ferramenta que apoie a geração e execução dos testes em contextos reais, possibilitando uma aplicação real de tais pesquisas. O objetivo deste trabalho foi projetar e implementar uma interface gráfica para a ferramenta que apoia o método W_{iots}, que usa o modelo Input-Output Transition Systems (IOTS). Essa ferramenta fornece apoio gráfico na geração e execução dos testes, possibilitando acompanhar os resultados da execução e pontos na qual podem existir problemas no software.*

1. Introdução

Quando se deseja criar um software, é esperado que haja algum procedimento e técnica pré-estabelecidos para diminuir a taxa de falhas que podem ser geradas. Uma das principais atividades para evitar que defeitos sejam introduzidos e que o software atinja os níveis de qualidade especificados é o teste de software. Tal atividade está relacionada com a garantia da qualidade de software e tem como objetivo minimizar a ocorrência de erros e riscos fornecendo evidências em relação à confiabilidade do software [Myers 2006].

Para realizar a atividade de testes corretamente, deve-se selecionar técnicas para estabelecer vários fatores como: um plano de teste, casos de teste, critérios de teste e por fim, os procedimentos de teste. Logo, o objetivo do teste é executar casos de teste que

sejam bons o suficiente para revelar defeitos [Myers 2006]. Apesar disso, a atividade de teste nem sempre é executada com todos esses passos devido ao custo e tempo necessários para sua realização.

O Teste Baseado em Modelos (TBM) utiliza modelos formais com o objetivo de automatizar a geração e execução de testes e para torná-lo formal e sistemático. Essa técnica não considera a forma como foi desenvolvida a implementação testada. Portanto, ele é considerado um teste caixa-preta e pode ser aplicado em qualquer fase de teste (unidade, integração e sistema) [Utting et al. 2012]. O TBM tem grande potencial para automatização se for utilizado junto a ferramentas que tornam o processo de geração de casos de teste mais rápidos e também menos propenso a erros humanos [Utting and Legeard 2010].

Um dos modelos formais mais adequados para o teste de software são os Sistemas de Transição Rotulados com Entrada/Saída (do inglês, *Input/Output Transition Systems - IOTS*) por lidar com não-determinismo e por ser genérico o suficiente para lidar com uma ampla gama de domínio de sistemas [Tretmans 2008]. Esse modelo é representado por estados e transições entre os estados rotuladas por uma ação que o sistema pode realizar [Tretmans 2008]. O conjunto de ações é subdividido em entradas e saídas, e a cada transição apenas um tipo de ação é modelada.

O uso de ferramentas em TBM é essencial para auxiliar na automatização do processo de teste. Apesar de existirem técnicas recentes para o uso de IOTSs no TBM, poucos estudos na área de teste mencionam o uso ou implementação de ferramentas que apoiam o uso dos métodos propostos [Paiva and Simao 2015b]. São poucas as ferramentas existentes para o modelo IOTSs [Paiva and Simao 2015a] e elas são, em sua maioria, por interface de linha de comando. Porém, uma ferramenta que apoie a execução, visualização e simulação de testes trará benefícios em termos de facilidade de aplicação, uma vez que o uso de modelos formais pode limitar a aplicação desta técnica em contextos reais [Paiva and Simao 2015b].

O objetivo principal deste trabalho é apresentar uma ferramenta que apoia um método de geração de testes a partir de IOTSs com visualização gráfica desenvolvido recentemente [Paiva and Simao 2016]. Tal método realiza a geração automática de casos de testes por meio de um algoritmo *offline* (gerado antes da execução) e com garantia de cobertura de um determinado domínio de defeitos. Nesse sentido, este trabalho apresenta a ferramenta gráfica e seus benefícios em um estudo de caso.

O artigo está organizado da seguinte forma: A Seção 2 apresenta a fundamentação teórica em TBM e trabalhos relacionados. A Seção 3 descreve como a ferramenta desenvolvida. A Seção 4 apresenta uma prova de conceito sobre a ferramenta. Por fim, a Seção 5 apresenta as considerações finais do trabalho.

2. Ferramentas do TBM a partir de IOTSs

Após a realização de uma revisão sistemática de literatura com o objetivo de conhecer ferramentas de TBM que utilizem o modelo IOTS, foram extraídas as ferramentas descritas a seguir:

1. TGV: A ferramenta TGV permite a geração de um caso de teste a partir de uma especificação e uma proposta de teste. A geração dos testes é feita de forma *on-*

line e foi baseada no algoritmo de Tarjan[1], que auxilia a encontrar componentes fortemente conectados de um grafo direcionado. Nessa ferramenta, o modelo utilizado foi o IOLTS (do inglês, *Input/Output Labeled Transition System*). Diferentemente dos IOTS, os IOLTS não consideram a característica de habilidade para entrada [Paiva 2016]. O TGV possui uma interface gráfica denominada Eucalyptus para interação com o usuário visando a geração dos testes e uma visualização do autômato. Foi observado durante o estudo dessa ferramenta, que é possível gerar a imagem do autômato e editar a visualização gerada com a funcionalidade de mover os estados gerados. O TGV usa sua ferramenta para converter uma proposta de teste no formato AUT para o formato BCG e transformar o caso de teste gerado ou um grafo de teste completo no formato AUT. [2]

2. UPPAAL: Ferramenta integrada para modelagem, simulação e verificação de sistemas de tempo real que são modelados por meio de autômatos de tempo. Geralmente, essa ferramenta é utilizada em projetos cujo aspectos relativos a tempo são cruciais e ela é composta por três partes principais: uma linguagem de descrição, um simulador e um verificador de modelos. Sobre o simulador, pode-se destacar que é uma ferramenta de validação que simula possíveis execuções do sistema durante a modelagem do projeto, garantindo assim uma verificação de falhas cobrindo o comportamento dinâmico exaustivo do sistema. Em sua documentação, os desenvolvedores afirmam que os principais critérios de desenvolvimento do UPPAAL foram a eficiência e sua facilidade de uso, e é possível perceber isto pois a ferramenta é bem intuitiva e bem visual. O formato principal de arquivos utilizado pela ferramenta é o XML. As linguagens permitidas pela ferramenta são C++ e Java, e também é uma ferramenta multi-plataforma: *Windows, Linux e MacOS*. Um das características que pode ser bem aproveitada dessa ferramenta seria sua simulação dos estados dos autômatos. Quando o sistema é executado, gera-se aleatoriamente as simulações, e é demonstrado a cada estado transitado uma visualização do estado passado. Porém, a ferramenta não possui tanto suporte para outras linguagens de visualização de autômatos como DOT e GraphViz e também não fornece apoio para simulação e execução do teste. [3]
3. JTorX: é descrita como uma ferramenta para o teste *ioco* (do inglês, *Input/Output Conformance*) relacionando um modelo de especificação e um modelo da implementação. A relação *ioco* verifica com precisão se uma implementação está em conformidade com uma determinada especificação [Tretmans 2008]. Em comparação com a sua versão antiga (TorX), a JTorX possui algumas funcionalidades adicionais como a derivação de casos de teste que estão prontos para aceitar uma saída em seu teste. Também destaca-se o teste com *Utraces* (do inglês, *Underspecified traces*), juntamente com o *iocoChecker*, que é uma ferramenta para verificar se ambos modelos são relacionados ao *ioco*, um verificador que tem o objetivo de certificar que os *Straces* (do inglês, *Suspension Traces*) coincidem com os *Utraces* e uma simulação. A interface gráfica provida pelo JTorX é bem intuitiva e fácil de ser manuseada, porém ela não gera nenhuma visualização dos modelos para o usuário e nem gera um arquivo externo sobre os casos de teste.

[1] `http://cadp.inria.fr/man/tgv.html`
[2] `http://cadp.inria.fr/man/tgv.html`
[3] `http://www.uppaal.org/`

Sua única dependência para executar é Java. [4]

A Tabela 1 apresenta algumas características desejáveis de uma ferramenta gráfica para geração e execução de testes. Tais características foram obtidas analisando-se as três ferramentas apresentadas acima. Algumas informações não foram encontradas e estão marcadas com um hífen (-). Pode-se observar que todas as ferramentas possuem interface gráfica, mas nem todas geram uma imagem ou visualização gráfica dos modelos para o usuário e também não permitem visualização gráfica para acompanhamento da simulação e execução dos testes. Tal tipo de visualização permitiria visualizar os caminhos percorridos. Além disso, algumas ferramentas só suportam modelos em algum formato específico. Outro problema observado é que algumas ferramentas não possuem funcionalidade de salvar os casos de testes e outras geram somente um resultado para o usuário que não é tão interessante para a visualização, como pode-se observar na ferramenta JTorX. Assim, a ferramenta aqui proposta reúne as melhores características gráficas e funcionalidades das ferramentas pesquisadas.

Tabela 1. Características das ferramentas

	GUI	Arquivos Suportados	Formato	Imagem Modelo	Simulação dos Testes	Aponta a falha	Salva os testes
TGV	✓	BCG, AUT	IOLTS	✓	-	-	-
UPPAAL	✓	XML	-	✓	✓	✓	-
JTorX	✓	AUT, DOT, GV, GRAPHML	LTS	X	✓	✓	✓

A ferramenta TGV [Jard and Jéron 2005] foi desenvolvida para atender testes baseado no modelo IOLTS. O IOLTS é um modelo similar ao IOTS, porém ele não considera a característica de habilidade para a entrada [Paiva 2016]. A arquitetura do TGV possui diversos níveis de comunicação e eles se comunicam por meio das *(Application Programming Interface)*(APIs). Cada uma dessas APIs é uma simulação de um grafo de funções transversais de IOLTS e nessa simulação é realizada a computação dos estados iniciais, dos estados de transmissão, dos sucessores e também a comparação entre os estados.

Devido a sua arquitetura, o TGV simula uma API para diferentes linguagens com o mesmo código fonte, exceto pela API pai. Com isso, ocorre uma portabilidade para vários sistemas (SunOS 5, Linux e Windows XP). Outras linguagens suportadas são: Lotos, SDL e UML (*Unified Modeling Language*).

Outra ferramenta é a TorX [Tretmans and Brinksma 2003] que implementa o teste *ioco* e é uma das abordagens fundamentais que utiliza TBM com IOTS [Tretmans 2008]. As especificações podem ser expressadas nas linguagens formais Lotos, Promela e FSP. Em cada passo, o TorX computa apenas os testes primitivos de acordo com as especificações formais que são necessárias em cada passo: o estímulo que pode ser dado e as observações esperadas. Chega-se no passo de teste que é decidido entre simulação e observação. Logo após, é selecionado aleatoriamente o estímulo e é enviado para a implementação [Tretmans and Brinksma 2003]. A execução e derivação dos testes pode

[4] `https://fmttools.ewi.utwente.nl/redmine//projects/jtorx/wiki/`

```
s0 ? a s3
s0 ? b s1
s1 ? a s3
s1 ? b s2
s2 ! 0 s0
s3 ! 1 s1
```

Figura 1. Arquivo IOTS.

```
digraph a{
s0 -> s3 [label=<?a>]
s3 -> s1 [label=<!1>]
s1 -> s3 [label=<?a>]
s1 -> s2 [label=<?b>]
s2 -> s0 [label=<!0>]
s0 -> s1 [label=<?b>]
}
```

Figura 2. Arquivo Dot.

ser feita de forma automatizada ou semi-automática (com intervenção) se for a preferência do usuário. Existe uma versão mais recente da ferramenta implementada em Java, a JTorX. [Belinfante 2010]

Outras ferramentas que valem ser citadas: TVEDA [Groz and Risser 1997], The AGEDIS Tool Set [Hartman and Nagin 2004] e TESTGEN [Vuong et al. 1994], mas não usam o mesmo modelo e técnicas aqui abordadas.

3. Uma Ferramenta de Apoio a Execução do Teste Baseado em Modelos a Partir de IOTSs

Essa seção apresenta como foi criada a ferramenta. Algumas características que a diferencia das demais: possui uma interface gráfica e gera uma imagem do modelo da especificação e do modelo da implementação, além de permitir realizar uma simulação do teste a partir de modelos em formato simples, além de verificar se há conformidade em relação à teoria *ioco* [Tretmans 2008].

As principais funcionalidades da ferramenta são:

- ***Gerar IOTS***: essa funcionalidade consiste em gerar um IOTS randomicamente. Pode-se observar a estrutura do arquivo IOTS na Figura 1. Cada linha representa uma transição, e em cada linha há quatro *tokens*: o primeiro indica o estado em que a transição está saindo, o segundo representa o tipo de ação (? para ações de entrada e ! para ações de saída), o terceiro indica o rótulo da ação e o quarto *token* indica o estado na qual a transição está chegando. O primeiro estado indicado na primeira linha é considerado o estado inicial [Paiva 2016].
- ***Gerar conjunto de IOTS***: quando o usuário seleciona essa funcionalidade é gerado um conjunto de IOTSs randomicamente e seus respectivos conjuntos de testes.
- ***Gerar caso de teste***: essa funcionalidade gera arquivos de casos de testes para uma certa implementação. O conjunto de teste é salvo em um diretório indicado pelo usuário. Cada caso de teste é salvo em um arquivo no formato de um IOTS com um único caminho.
- ***Executar caso de teste***: essa funcionalidade recebe uma implementação e um caso de teste, que são lidos e transformados na devida estrutura de IOTS da ferramenta. Uma vez que se inicia a execução do teste, para cada caso de teste, a ferramenta irá caminhar em paralelo na implementação e no caso de teste, de modo que, se a execução atingir o final do caso de teste, a implementação passa no caso de teste. Caso contrário, se houver alguma não-conformidade antes de atingir o final, a implementação falha [Paiva 2016].

Para gerar as aplicações presentes na interface gráfica, foi necessário algumas funcionalidades específicas. Para gerar a visualização gráfica dos IOTS, foi utilizada uma

linguagem de descrição de grafos, chamada DOT. Essa linguagem auxilia a geração das figuras dos IOTS, passando os estados e as transições como nas Figura 2 e 3. Juntamente com a adição da funcionalidade de geração de gráficos, foram adicionadas as seguintes:

- ***Transformar IOTS para DOT***: essa funcionalidade lê o arquivo no formato IOTS e como saída gera o arquivo no formato .dot que será utilizado para gerar a visualização do IOTS. O arquivo IOTS possui os 4 tokens explicados anteriormente que seguiam um formato específico, e o mesmo ocorre com o arquivo DOT: o primeiro token indica o estado em que a transição está saindo; o segundo é uma seta; o terceiro token é o estado em que a transição chega; o quarto indica o rótulo e a ação dos estados e por padrão do arquivo DOT deve-se encontrar dentro do campo de *label*. A conversão consiste em separar os tokens do arquivo IOTS e gerar um novo arquivo que contenha os tokens organizados no formato DOT para realizar a atividade de teste. Também é disponibilizado para o usuário realizar o *download* do arquivo gerado.
- ***Transformar DOT para IOTS***: essa funcionalidade lê um arquivo no formato .dot e como saída gera um arquivo no formato IOTS. A conversão ocorre em retirar os tokens que estão no formato do arquivo DOT explicados no item anterior e organizá-los no formato IOTS. A interface disponibiliza para o usuário o *download* do arquivo gerado.
- ***Gerar figura via DOT***: essa funcionalidade consiste na leitura do arquivo no formato .dot que é passado pelo usuário e como saída o sistema gera a figura no formato .png de acordo com o arquivo informado. A geração da figura é realizada com a utilização da biblioteca *Graphviz*[5] que recebe como entrada o arquivo DOT e gera a figura representativa. Também é disponibilizado para o usuário realizar o *download* da figura gerada pela ferramenta.
- ***Executar caso de teste***: essa funcionalidade simula a execução dos casos de testes diretamente na interface. Ela solicita dois arquivos, o IOTS da implementação e o IOTS do caso de teste, e após o botão de executar ser pressionado a ferramenta gera uma simulação da execução do caso de teste exibindo as transições sendo coloridas de acordo com o andamento da atividade de teste. O primeiro passo que o programa realiza é a conversão dos arquivos para o formato DOT, então a cada transição que é percorrida são gerados arquivos com as transições coloridas, indicando que a transição foi aceita. Após a geração de todas as figuras em que o caso de teste foi aceito de acordo com a implementação, as figuras são exibidas na interface para o usuário uma após a outra gerando a simulação da atividade de teste.

4. Prova de Conceito

Essa seção apresenta uma prova de conceito sobre a ferramenta que foi desenvolvida. Um exemplo das funcionalidades providas pela interface gráfica da ferramenta proposta é apresentado nesta seção. O arquivo IOTS escolhido como exemplo real [Peleska et al. 2011] é o da Figura 4.

A primeira funcionalidade a ser testada foi a de transformar o arquivo IOTS da Figura 4 para um arquivo DOT, como mostrado na Figura 5. Nela, pode-se observar que o

[5]https://www.graphviz.org/

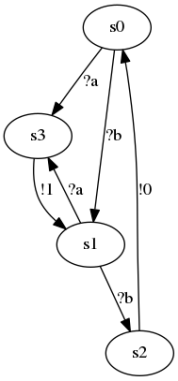

Figura 3. Figura Gerada pelo Programa via Dot.

```
OFF ? c1 1
OFF ? c2 OFF
OFF ? t.elapsed() OFF
1 ! 100 2
2 ! t.reset() ON
ON ? c1 3
3 ! 100 4
4 ! t.reset() ON
ON ? c2 5
ON ? t.elapsed() 5
5 ! 0 OFF
```

Figura 4. Arquivo IOTS para exemplificação.

arquivo foi convertido corretamente e exibido na tela para o usuário. Também é possível realizar o *download* do arquivo convertido. A segunda funcionalidade é a de transformar um arquivo DOT para um arquivo IOTS, realizando a operação reversa. Tanto nesse caso como no anterior pode ser observado que a conversão foi feita corretamente e exibida para o usuário, além de estar disponível para *download*. Outra funcionalidade é a de geração da figura via arquivo DOT. O arquivo DOT utilizado foi o da Figura 4 e a geração do IOTS pode ser observado na Figura 6. A ferramenta proposta permite a visualização do resultado na tela, bem como o *download* da imagem.

Arquivo IOTS!

```
OFF ? c1 1
OFF ? c2 OFF
OFF ? t.elapsed() OFF
1 ! 100 2
2 ! t.reset() ON
ON ? c1 3
3 ! 100 4
4 ! t.reset() ON
ON ? c2 5
ON ? t.elapsed() 5
5 ! 0 OFF
```

Arquivo DOT!

```
digraph a{
OFF -> 1 [label=<?c1>]
1 -> 2 [label=<!100>]
2 -> ON [label=<!t.reset()>]
ON -> 3 [label=<?c1>]
3 -> 4 [label=<!100>]
4 -> ON [label=<!t.reset()>]
ON -> 5 [label=<?c2>]
5 -> OFF [label=<!0>]
ON -> 5 [label=<?t.elapsed()>]
OFF -> OFF [label=<?c2>]
OFF -> OFF [label=<?t.elapsed()>]
}
```

Figura 5. Transformação do IOTS para DOT.

A última funcionalidade a ser descrita é a de execução de caso de teste. Pode-se observar na Figura 7 uma simulação da execução do caso de teste. Com o arquivo de implementação e o de caso de testes, a ferramenta colore as transições que são caminhadas e exibe para o usuário uma figura após a outra, gerando uma simulação da atividade de teste passo a passo. Pode-se observar outro exemplo da mesma funcionalidade, porém

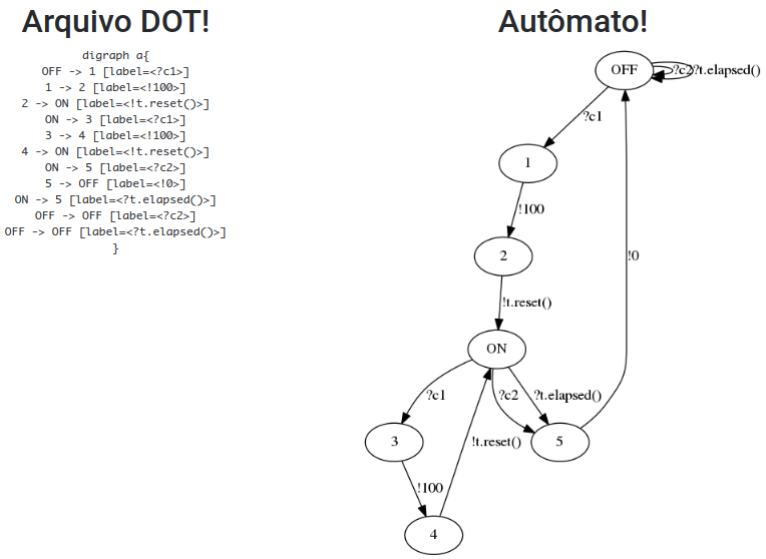

Figura 6. Geração da figura via arquivo DOT.

com um caso de teste diferente na Figura 8. Como dito previamente, utilizando o arquivo de implementação e o arquivo de caso de teste, a ferramenta demonstra para os usuários as transições que foram cobertas pelo caso de teste.

Com o protótipo aqui desenvolvido foi possível verificar que é possível ter uma ferramenta que apoie a simulação e execução de casos de teste de forma gráfica e dinâmica. Assim, os usuários que não tem muita experiência com a técnica podem facilmente compreender o motivo do teste ter passado ou ainda em qual(is) ponto(s) houve algum problema de conformidade.

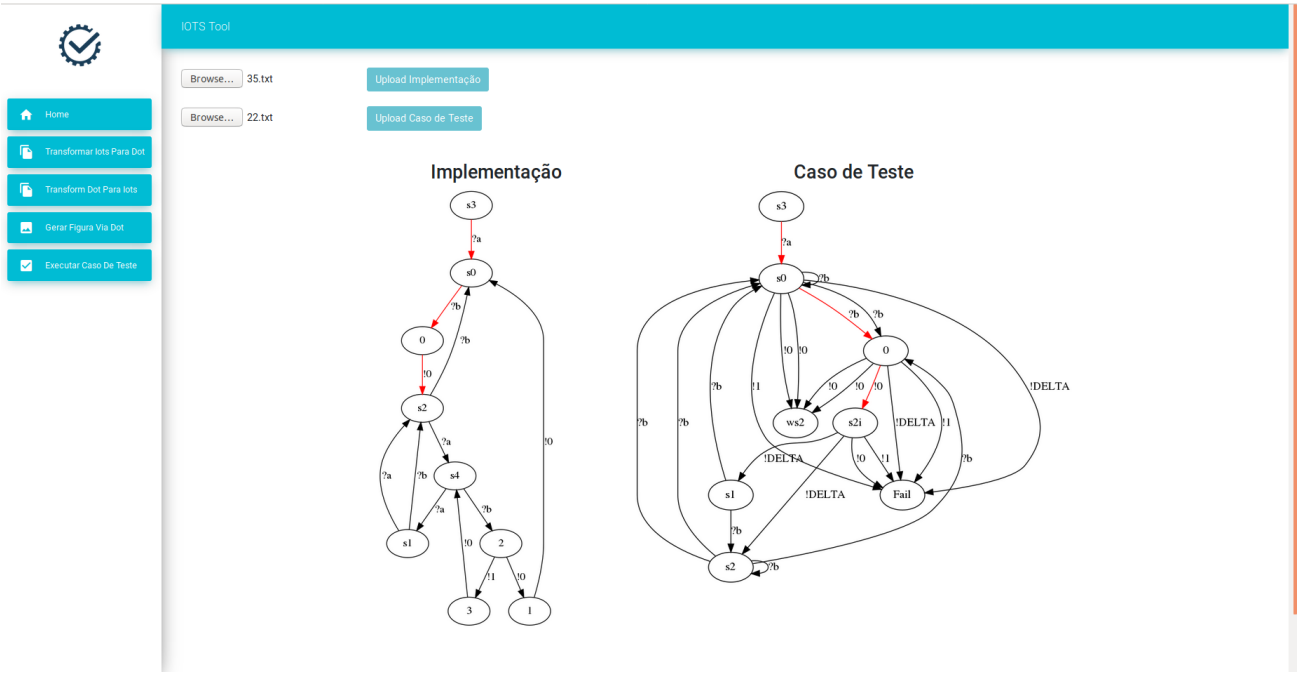

Figura 7. Execução de caso de teste.

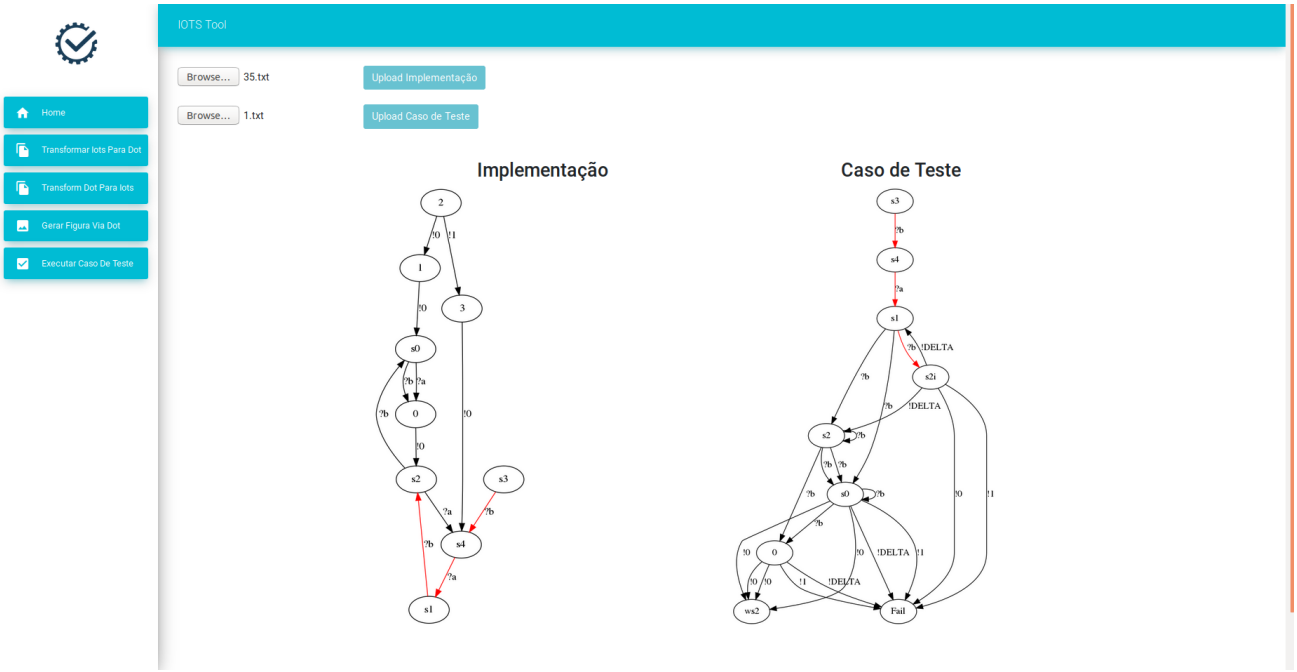

Figura 8. Execução de caso de teste.

5. Conclusão

De acordo com os resultados da revisão sistemática, os estudos na área de testes baseado no modelo IOTS implicam que existem poucas ferramentas que apoiam esse modelo [Paiva and Simao 2015b] e como o uso de ferramentas é essencial para auxiliar aplicações em contextos reais bem como para compreender a técnica de TBM, foi vista a necessidade de fornecer uma ferramenta que apoie esse modelo em funcionalidades de visualizar, simulação e execução de testes.

O desenvolvimento da ferramenta proposta serve como auxílio para usuários que desejem utilizar o método W_{iots} para realizar a atividade de teste, auxiliando a aplicação de TBM em contextos mais amplos, já que o uso de métodos que envolvem algum modelo formal podem intimidar os praticantes de teste.

As ferramentas identificadas na literatura não possuem determinadas facilidades, como o acesso via *browser* nem a visualização gráfica dos modelos durante a simulação e execução de testes. Já a ferramenta proposta possui essas facilidades que se encaixam na necessidade de usuários que estão realizando a atividade de teste com esse método.

Algumas limitações podem ser destacadas: a ferramenta só atende ao método W_{iots} com o modelo IOTS. Além disso, ela utiliza a biblioteca *Graphviz* para a geração das figuras em formato DOT, que é pouco usual.

Como trabalho futuro, propõe-se evoluir a ferramenta e adicionar novas funcionalidades, como a geração de casos de testes de forma visual. A realização de estudos com usuários reais também deve ser realizada de modo a identificar melhorias e pontos fortes da ferramenta, além de melhorar elementos de usabilidade da ferramenta.

Referências

Belinfante, A. (2010). Jtorx: A tool for on-line model-driven test derivation and execution. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 266–270. Springer.

Groz, R. and Risser, N. (1997). Eight years of experience in test generation from fdts using tveda. In *Formal Description Techniques and Protocol Specification, Testing and Verification*, pages 465–480. Springer.

Hartman, A. and Nagin, K. (2004). The agedis tools for model based testing. *ACM SIGSOFT Software Engineering Notes*, 29(4):129–132.

Jard, C. and Jéron, T. (2005). Tgv: theory, principles and algorithms. *International Journal on Software Tools for Technology Transfer*, 7(4):297–315.

Myers, G. J. (2006). *The art of software testing*. John Wiley & Sons.

Paiva, S. C. and Simao, A. (2016). Generation of complete test suites from mealy input/output transition systems. *Formal Aspects of Computing*, 28(1):65–78.

Paiva, S. L. C. and Simao, A. (2015a). A systematic mapping study on test generation from Input/Output Transition Systems. In *Software Engineering and Advanced Applications (SEAA), 2015 41th EUROMICRO Conference on*. IEEE Computer Society.

Paiva, S. L. d. C. (2016). *Aplicação de modelos de defeitos na geração de conjuntos de teste completos a partir de Sistemas de Transição com Entrada/Saída*. PhD thesis, Universidade de São Paulo.

Paiva, S. L. d. C. and Simao, A. d. S. (2015b). A systematic mapping study on test generation from input/output transition systems. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conf. on*, pages 333–340. IEEE.

Peleska, J., Honisch, A., Lapschies, F., Löding, H., Schmid, H., Smuda, P., Vorobev, E., and Zahlten, C. (2011). A real-world benchmark model for testing concurrent real-time systems in the automotive domain. In *Proceedings of the 23rd IFIP WG 6.1 International Conference on Testing Software and Systems*, ICTSS'11, pages 146–161, Berlin, Heidelberg. Springer-Verlag.

Tretmans, G. and Brinksma, H. (2003). *TorX: Automated Model-Based Testing*, pages 31–43.

Tretmans, J. (2008). Model based testing with labelled transition systems. In *Formal methods and testing*, pages 1–38. Springer.

Utting, M. and Legeard, B. (2010). *Practical model-based testing: a tools approach*. Morgan Kaufmann.

Utting, M., Pretschner, A., and Legeard, B. (2012). A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312.

Vuong, S. T., Janssen, H., Lu, Y., Mathieson, C., and Do, B. (1994). Testgen: An environment for protocol test suite generation and selection. *computer communications*, 17(4):257–270.