6 Experimentação do processo proposto

Neste capítulo iremos descrever os experimentos práticos do método de geração de testes para interface gráfica realizados sobre o software real V3O2/Tecgraf, bem como os resultados obtidos e a discussão sobre eles.

Tal como mencionado no Capítulo 1, o software V3O2 é voltado para a área da geofísica e dá suporte a diversos tipos de operações e análises sobre os dados sísmicos, além da visualização e da manipulação destes dados. Naturalmente, o software possui diversas interfaces gráficas, cada uma voltada para uma operação específica a ser realizada. Além das diversas GUIs, há também a janela principal, composta em sua maior parte pela área de visualização 3D e, na porção lateral direita, por uma "árvore" dos objetos geofísicos carregados pelo usuário. (Figura 4a)

As GUIs das funcionalidades que dão suporte às operações sobre os dados são compostas por *widgets* de diversos tipos e estão sujeitas a uma série de eventos não determinísticos e algumas vezes concorrentes, que podem ter vindo do usuário ou do sistema. Essas características se encaixavam bem a uma forma de experimentar e avaliar o processo de teste de GUIs proposto neste trabalho.

Para medir a qualidade deste processo, focado no uso do modelo Rede de Petri como representação da GUI e gerador de casos de teste, faremos as seguintes avaliações:

1. Eficiência do método: Comparação do tempo consumido neste processo com o tempo médio de testes criados manualmente; (Seção 6.2.1)
2. Eficácia do método: Apresentação do número de defeitos revelados, apresentação do escore de mutação da Análise de Mutantes e comparação dos escores obtidos em simulações manuais com as randômicas e automáticas. (Seção 6.2.2)
3. Contrapor as características de outros métodos de geração de teste para GUI, com os deste trabalho; (Seção 6.2.3)

6.1 Funcionalidades testadas e estatísticas

Quatro funcionalidades foram escolhidas para a aplicação do processo de geração de teste estudado. O que ajudou na escolha destas quatro foi o fato de suas

porções de código relativas ao toolkit de interface gráfica GTK já estarem no padrão que vem sendo adotado em todo o software. Esse padrão diz respeito a uma camada implementada sobre o código de GTK, para facilitar o uso e reduzir a repetição de código. Para ilustrar o uso dessa camada, podemos dar o seguinte exemplo: Existe uma classe para representar cada *widget*, por exemplo, a classe ComboBox. O método para recuperar um conteúdo é getSelectedText(int index). O método que recupera a sua visibilidade é isVisible(). E a implementação desses métodos contém código "puro" do GTK. Portanto, é importante que as funcionalidades de GUI usem essa camada que age sobre os *widgets* do GTK, para que os scripts gerados contenham o código equivalente.

Também é importante para a geração do script executável que os nomes dos *widgets* estejam no padrão usado na RP.

A existência de padrões de programação torna viável a geração de scripts de teste, que contêm códigos que recuperam conteúdo e executam eventos sobre *widgets*.

A seguir, apresentamos cada funcionalidade com seu nome, uma breve descrição, uma figura da funcionalidade em uso, uma figura da RP modelada e um dos logs gerados na simulação da rede, a partir do qual será gerado um dos casos de teste. Também é mostrado o relatório da análise de mutantes de cada uma.

1 - OPERAÇÃO SOBRE HORIZONTES

Esta é a janela principal das quatro funcionalidades que serão apresentadas, onde cada uma permite realizar algum tipo de operação sobre o objeto sísmico Horizonte. Ainda assim, a janela principal pode se testada de forma separada, porque possui possibilidades de interação que já devem ser validadas, como seleção do horizonte a ser visualizado, seleção do atributo do horizonte que será mapeado em cores, remoção ou renomeação de algum atributo, fechamento da janela e a própria abertura das funcionalidades que editam o horizonte, disponíveis nos links em azul no lado direito da janela principal. (Figura 31)

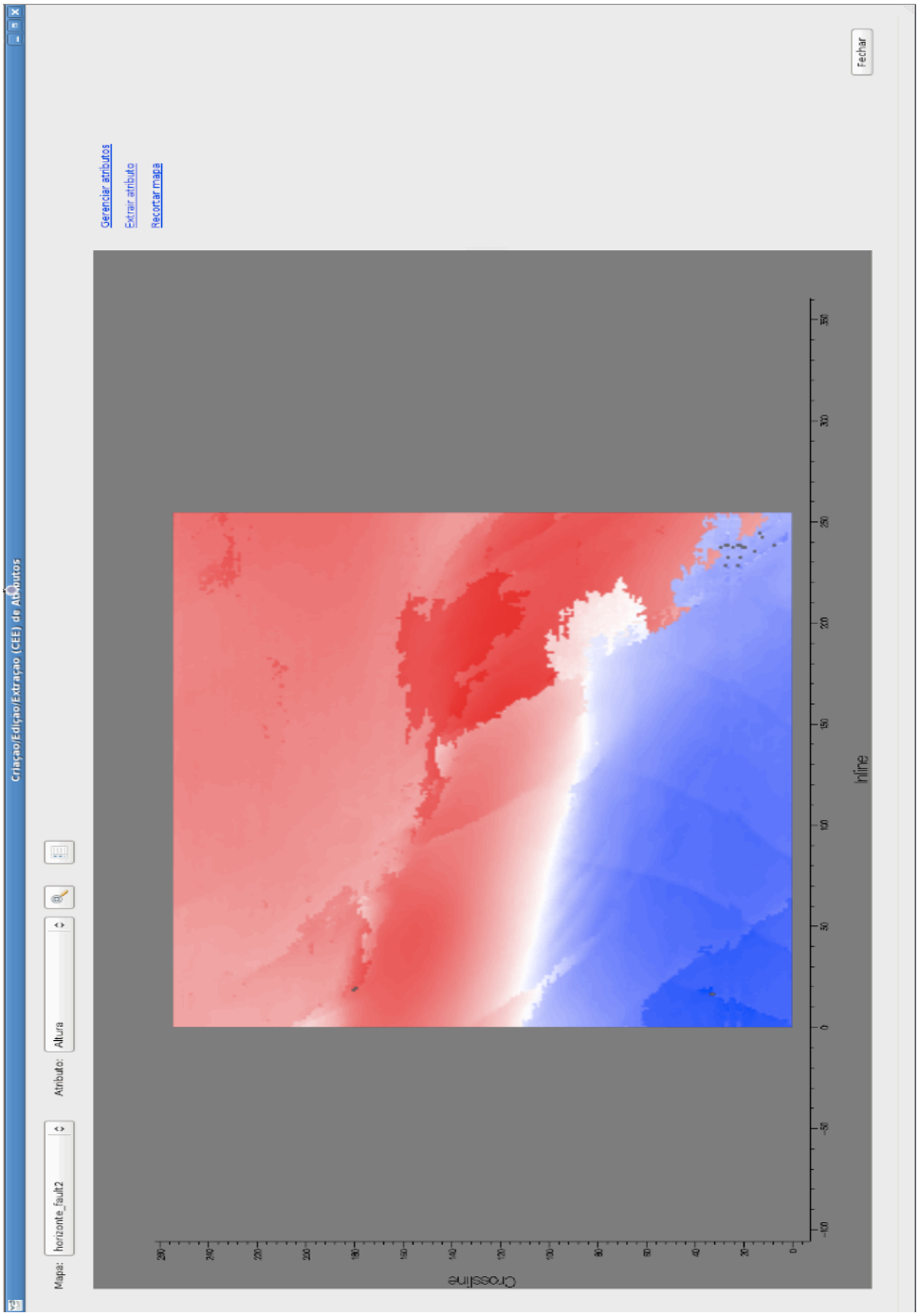

Figura 31 – Funcionalidade Operação sobre Horizontes

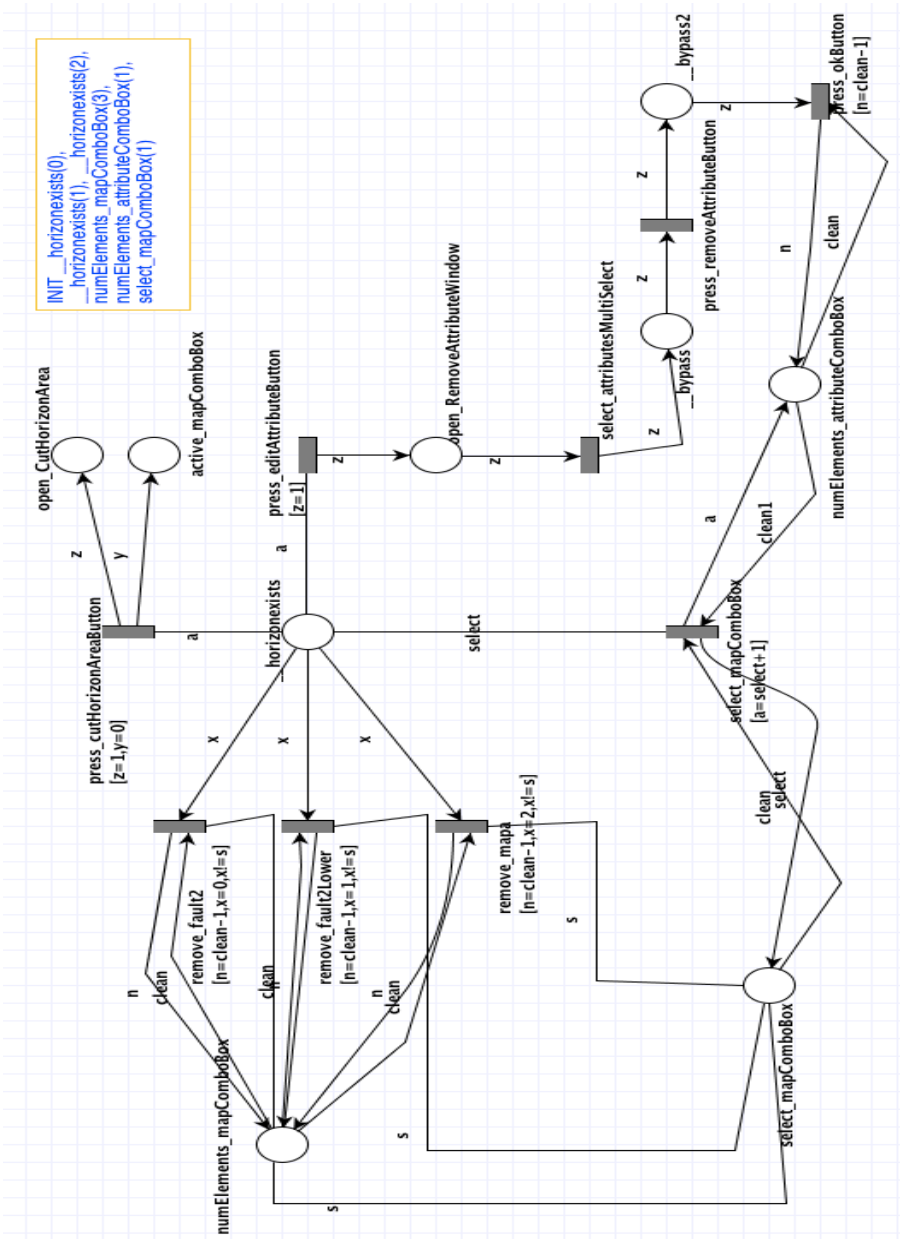

Figura 32 – Rede de Petri da funcionalidade Operação sobre Horizontes

```
SEQUENCE (INIT 0) select_mapComboBox [clean1/1, clean/1, se-
lect/0], select_mapComboBox [clean1/1, clean/0, select/1],
remove_mapa [s/1, clean/3, x/2],
select_mapComboBox [clean1/2, clean/1, select/1],
press_editAttributeButton [a/0], select_attributesMultiSelect
[z/1], press_removeAttributeButton [z/1], press_okButton [z/1,
clean/2], select_mapComboBox [clean1/1, clean/1, select/0]
```

Figura 33 – Um dos logs gerados na simulação.

```
Relatório da Análise de Mutantes

Classes sob mutação:
../v3o2_pesquisa_raquel/v3o2/src/horizon/presenter/HorizonOperationManagerPresenter.cpp
../v3o2_pesquisa_raquel/v3o2/src/horizon/window/HorizonOperationsWindow.cpp

Suíte de teste:
../v3o2_pesquisa_raquel/v3o2/src/horizon/HorizonOperationUITest.cpp


Número de operadores de mutação: 14

Estatísticas de operadores e mutantes:
OPERADOR         MUTANTES GERADOS        MUTANTES MORTOS
ROR                     38                      7
LOR                     6                       0
AOR                     12                      0
BVR                     33                      0
LRQD                    1                       0
LRLD                    2                       1
AOD                     3                       1
ROD                     3                       0
GTK_SRSD                0                       0
GTK_SSND                3                       0
GTK_SSLD                1                       0
GTK_SID                 3                       1
GTK_SSNC                0                       0
GTK_SSLC                1                       0


Escore de Mutação: (Mut. mortos / Mut. gerados - Equivalentes)
Total de mutantes gerados: 106
Total de mutantes mortos: 10
Mutantes equivalentes: 11


Escore de Mutação = 10.5263%
```

Figura 34 – Relatório da Análise de Mutantes para Operação sobre Horizontes

2 - GERÊNCIA DE ATRIBUTOS

Essa funcionalidade é acessada pelo primeiro link na janela principal mostrada acima. A Gerência de Atributos permite que o usuário edite ou crie um atributo de horizonte, definindo regiões e aplicando valores sobre elas. (Figura 35)

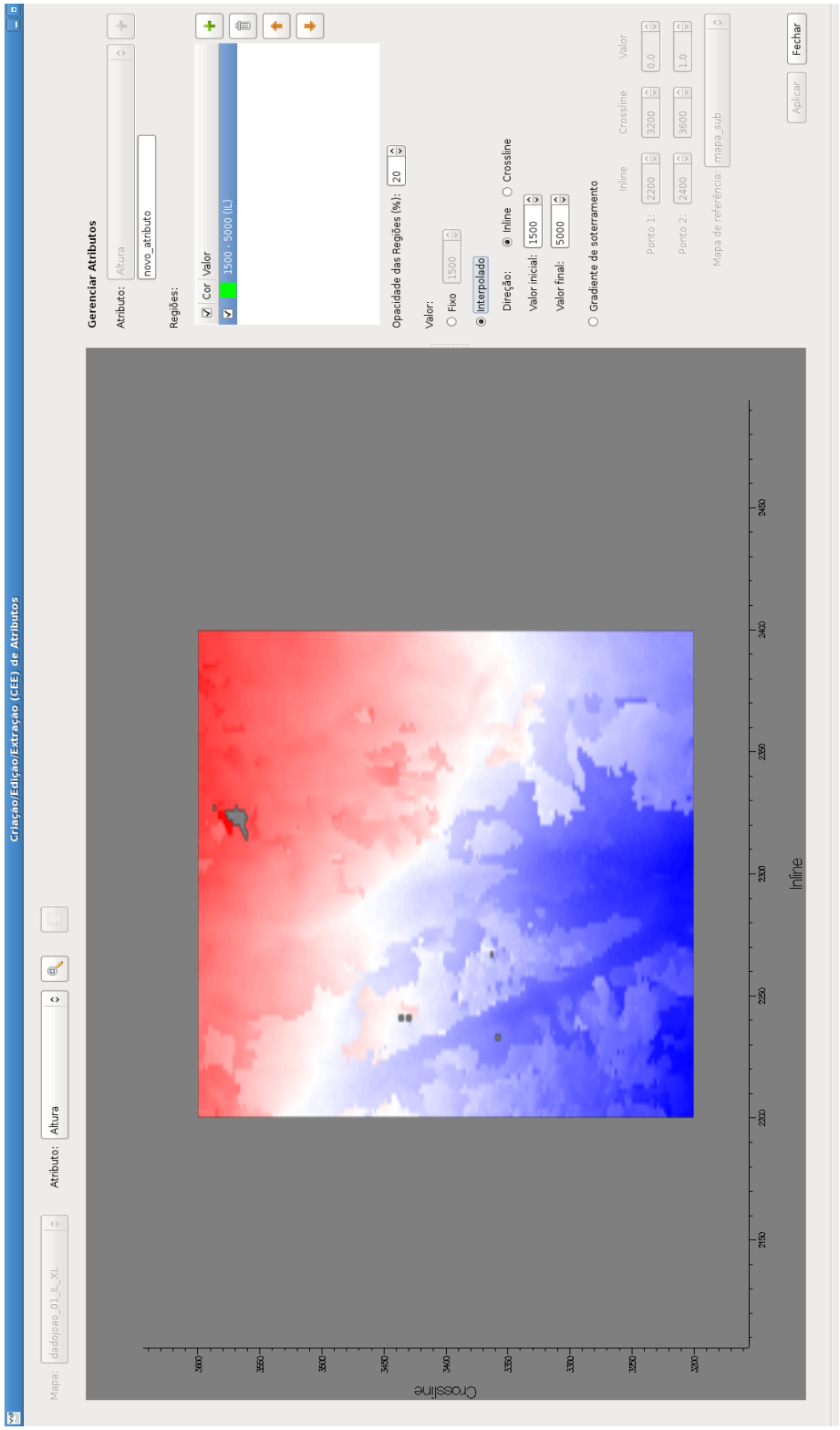

Figura 35 – Funcionalidade Gerência de Atributos

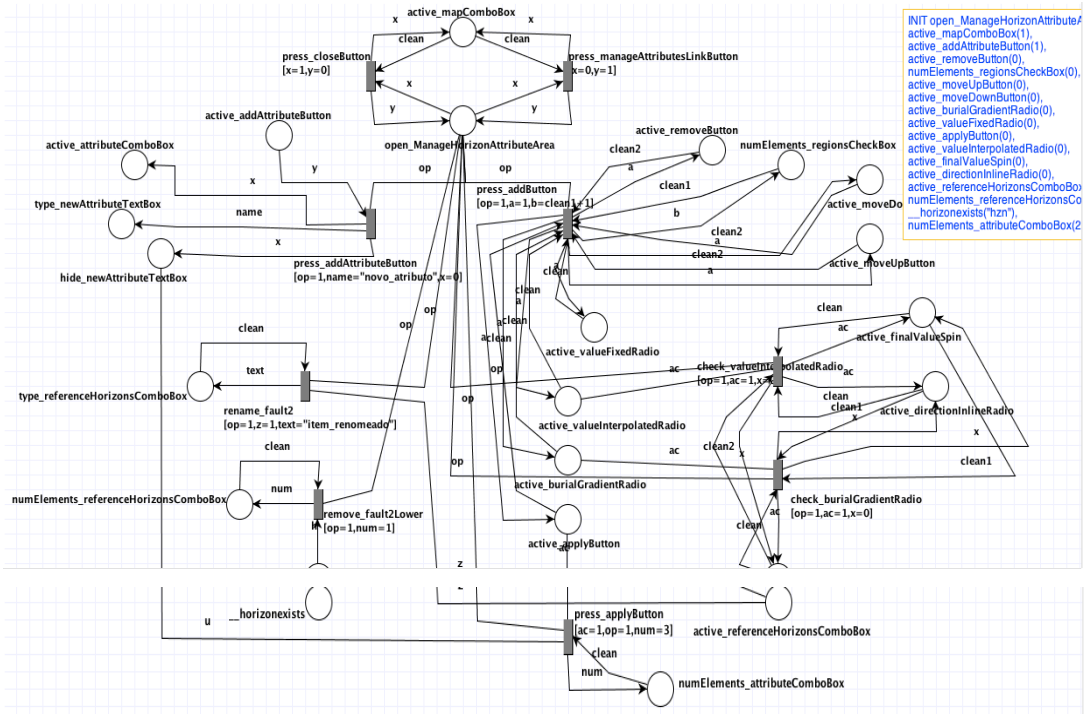

Figura 36 – Rede de Petri da funcionalidade Gerência de Atributos

```
SEQUENCE (INIT0) press_manageAttributesLinkButton 1[x/0, clean/1],
press_addAttributeButton 2[op/1, y/1], press_addButton 0[op/1,
clean2/0, clean1/0, clean/0], check_burialGradientRadio 5[ac/1,
clean/0, clean1/0, op/1], check_burialGradientRadio 5[ac/1, clean/1,
clean1/0, op/1], press_addButton 0[op/1, clean2/1, clean1/1,
clean/1], check_burialGradientRadio 5[ac/1, clean/1, clean1/0,
op/1], press_addButton 0[op/1, clean2/1, clean1/2, clean/1],
press_applyButton 8[ac/1, u/0, clean/2, op/1],
check_burialGradientRadio 5[ac/1, clean/1, clean1/0, op/1],
press_applyButton 8[ac/1, u/0, clean/3, op/1], press_addButton
0[op/1, clean2/1, clean1/3, clean/1], remove_fault2Lower 7[op/1,
clean/2, h/"hzn"], press_closeButton 3[x/1, clean/0],
press_manageAttributesLinkButton 1[x/0, clean/1], press_closeButton
3[x/1, clean/0], press_manageAttributesLinkButton 1[x/0, clean/1],
press_applyButton 8[ac/1, u/0, clean/3, op/1], press_addButton
0[op/1, clean2/1, clean1/4, clean/1], press_closeButton 3[x/1,
clean/0]
```

Figura 37 – Um dos logs gerados na simulação automática e randômica

```
Relatório da Análise de Mutantes

Classes sob mutação:
../v3o2_pesquisa_raquel/v3o2/src/horizon/attributeManagement/ManageHorizonAttributePresenter.cpp
../v3o2_pesquisa_raquel/v3o2/src/horizon/attributeManagement/ManageHorizonAttributeArea.cpp

Suíte de teste:
../v3o2_pesquisa_raquel/v3o2/src/horizon/ManageHorizonAttributeUITest.cpp


Número de operadores de mutação: 14

Estatísticas de operadores e mutantes:
OPERADOR         MUTANTES GERADOS        MUTANTES MORTOS
ROR                     28                      6
LOR                     7                       0
AOR                     6                       0
BVR                     40                      6
LRQD                    1                       0
LRLD                    0                       0
AOD                     3                       1
ROD                     3                       0
GTK_SRSD                0                       0
GTK_SSND                20                      2
GTK_SSLD                7                       0
GTK_SID                 8                       2
GTK_SSNC                6                       0
GTK_SSLC                2                       0


Escore de Mutação: (Mut. mortos / Mut. gerados - Equivalentes)
Total de mutantes gerados: 131
Total de mutantes mortos: 17
Mutantes equivalentes: 13


Escore de Mutação = 14.4068%
```

Figura 38 – Relatório da Análise de Mutantes para Gerência de Atributos

3 – EXTRAÇÃO DE ATRIBUTOS

A Extração de Atributos permite que o usuário extraia de um volume sísmico selecionado, um novo atributo para o horizonte. A maneira como será feita essa atribuição de valores do volume para o atributo, é definida através da seleção de um dos RadioButtons disponibilizados na interface. (Figura 39)

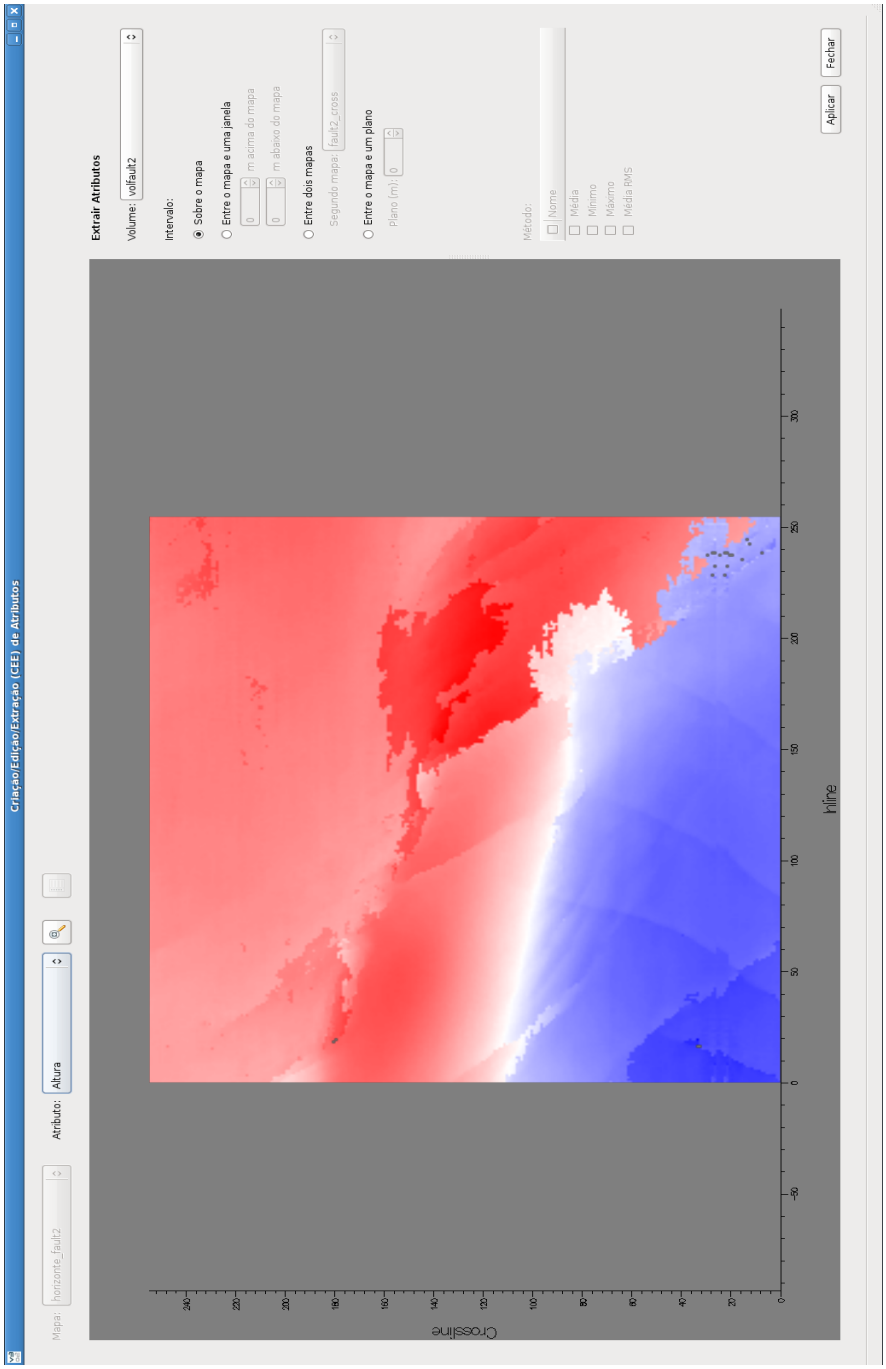

Figura 39 – Funcionalidade Extração de Atributos

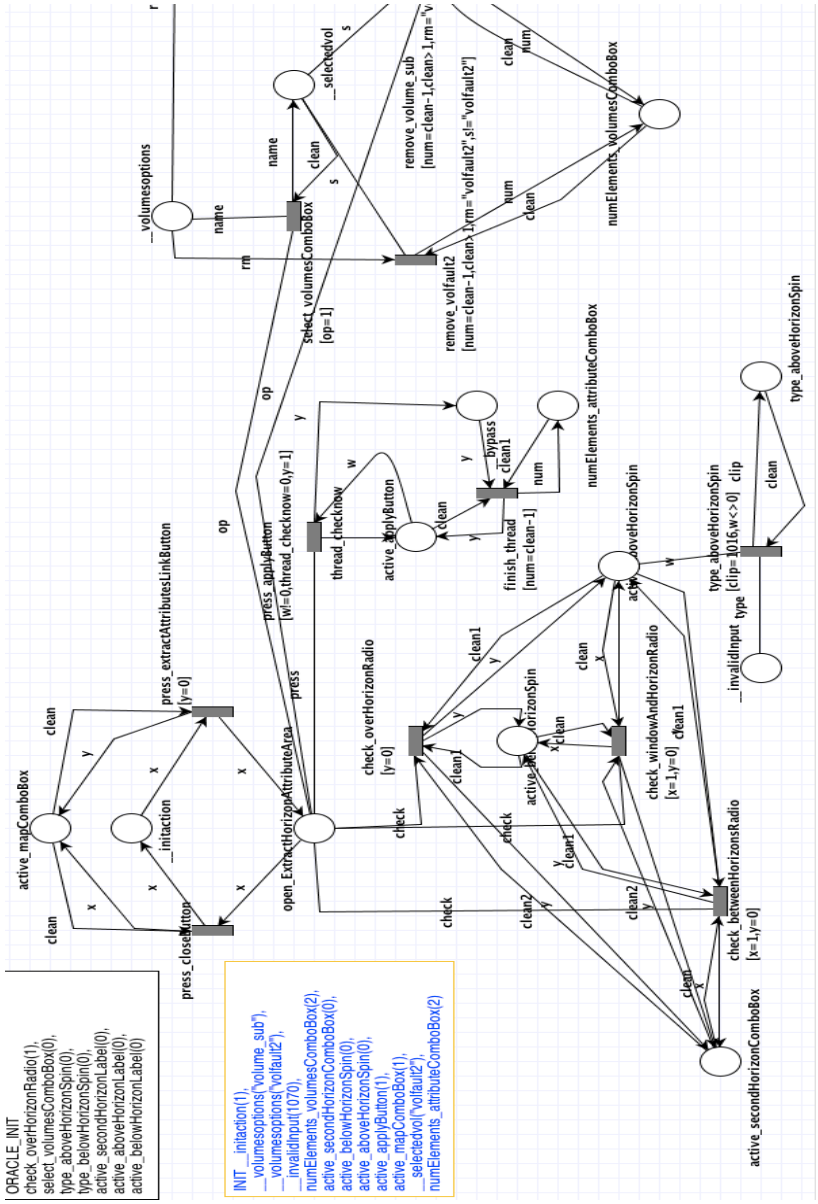

Figura 40 – Rede de Petri da funcionalidade Extração de Atributos

```
SEQUENCE (INIT0) press_manageAttributesLinkButton 1[x/0, clean/1],
press_addAttributeButton 2[op/1, y/1], press_addButton 0[op/1,
clean2/0, clean1/0, clean/0], check_burialGradientRadio 5[ac/1,
clean/0, clean1/0, op/1], check_burialGradientRadio 5[ac/1, clean/1,
clean1/0, op/1], press_addButton 0[op/1, clean2/1, clean1/1,
clean/1], check_burialGradientRadio 5[ac/1, clean/1, clean1/0,
op/1], press_addButton 0[op/1, clean2/1, clean1/2, clean/1],
press_applyButton 8[ac/1, u/0, clean/2, op/1],
check_burialGradientRadio 5[ac/1, clean/1, clean1/0, op/1],
press_applyButton 8[ac/1, u/0, clean/3, op/1], press_addButton
0[op/1, clean2/1, clean1/3, clean/1], remove_fault2Lower 7[op/1,
clean/2, h/"hzn"], press_closeButton 3[x/1, clean/0],
press_manageAttributesLinkButton 1[x/0, clean/1], press_closeButton
3[x/1, clean/0], press_manageAttributesLinkButton 1[x/0, clean/1],
press_applyButton 8[ac/1, u/0, clean/3, op/1], press_addButton
0[op/1, clean2/1, clean1/4, clean/1], press_closeButton 3[x/1,
clean/0]
```

Figura 41 – Um dos logs gerados na simulação automática e randômica

```
Relatório da Análise de Mutantes

Classes sob mutação:
../v3o2_pesquisa_raquel/v3o2/src/horizon/attributeExtraction/ExtractHorizonAttributePresenter.cpp
../v3o2_pesquisa_raquel/v3o2/src/horizon/attributeExtraction/ExtractHorizonAttributeArea.cpp

Suíte de teste:
../v3o2_pesquisa_raquel/v3o2/src/horizon/ExtractHorizonAttributeUITest.cpp


Número de operadores de mutação: 14

Estatísticas de operadores e mutantes:
OPERADOR         MUTANTES GERADOS        MUTANTES MORTOS
ROR                     26                      5
LOR                     11                      2
AOR                     0                       0
BVR                     40                      9
LRQD                    2                       1
LRLD                    4                       0
AOD                     9                       2
ROD                     3                       0
GTK_SRSD                2                       0
GTK_SSND                14                      7
GTK_SSLD                5                       1
GTK_SID                 4                       1
GTK_SSNC                0                       0
GTK_SSLC                4                       0


Escore de Mutação: (Mut. mortos / Mut. gerados - Equivalentes)
Total de mutantes gerados: 124
Total de mutantes mortos: 28
Mutantes equivalentes: 12


Escore de Mutação = 25%
```

Figura 42 – Relatório da Análise de Mutantes para Extração de Atributos

4 – RECORTE DE HORIZONTE

O Recorte de Horizonte permite que o usuário recorte regiões de um horizonte, ou para diminuir a área de interesse do dado ou par remover alguma região que considere anômala. Esse recorte é feito através da criação de regiões sobre a desenho do horizonte. (Figura 43)

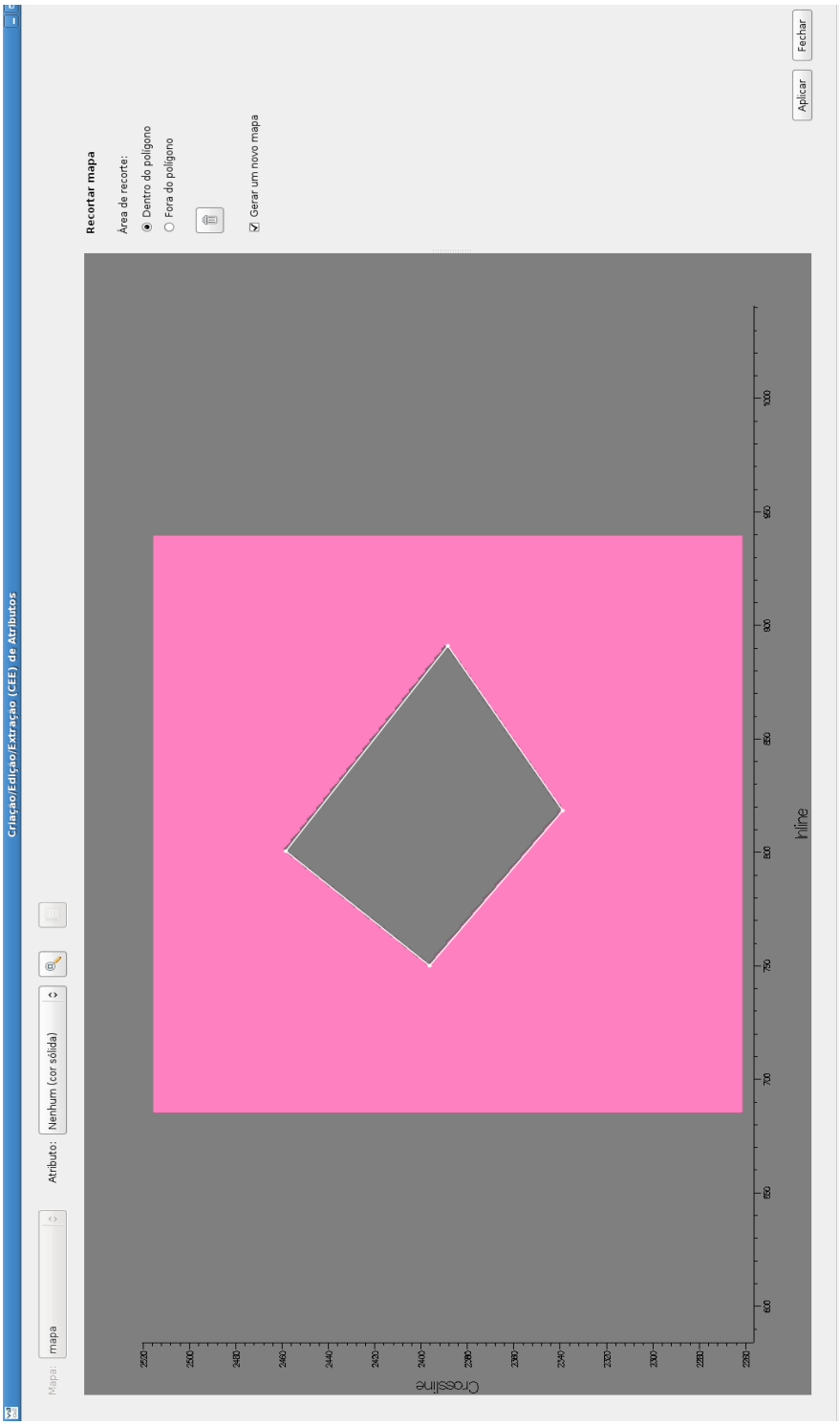

Figura 43 – Funcionalidade Recorte de Horizonte

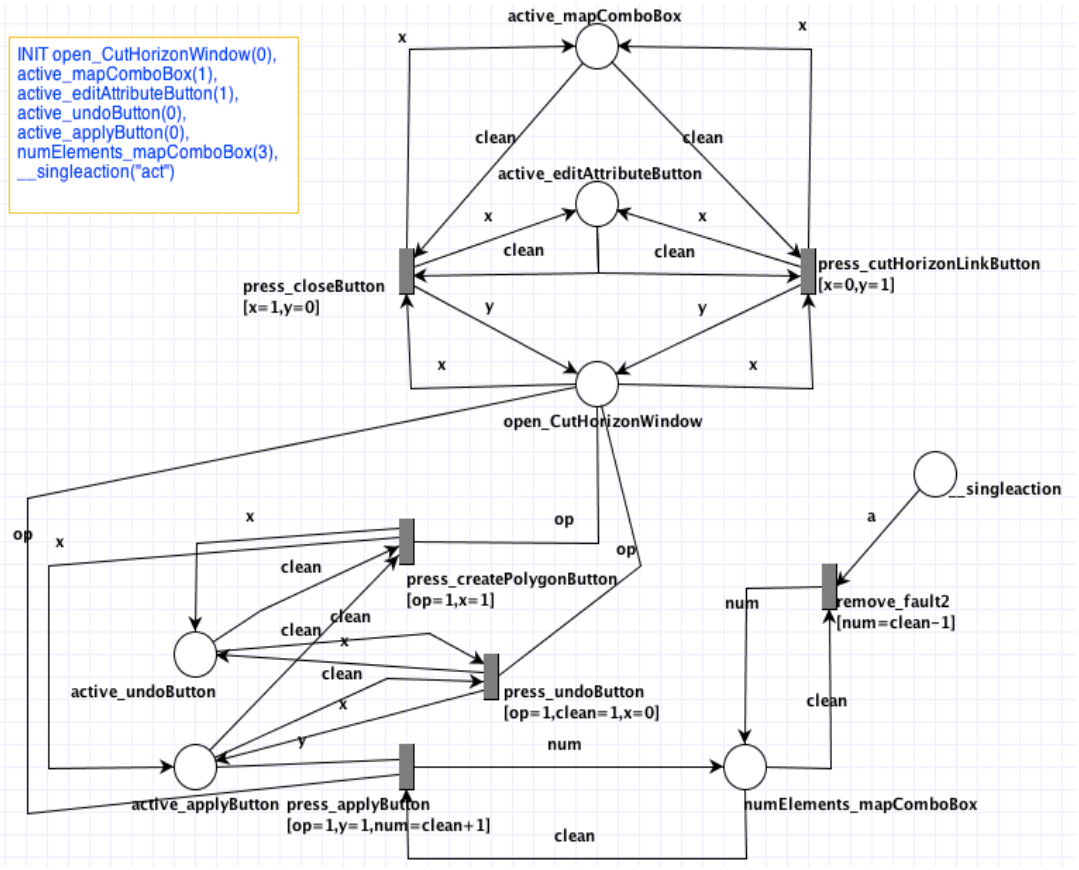

Figura 44 – Rede de Petri para a funcionalidade de Recorte de Horizonte

```
SEQUENCE (INIT0) press_cutHorizonLinkButton 0[clean/1, x/0],
remove_fault2 5[clean/3, a/"act"], press_createPolygonButton
3[op/1, clean/0], press_applyButton 2[op/1, clean/2, y/1],
press_applyButton 2[op/1, clean/3, y/1], press_undoButton
4[clean/1, op/1], press_closeButton 1[clean/0, x/1],
press_cutHorizonLinkButton 0[clean/1, x/0],
press_createPolygonButton 3[op/1, clean/0], press_closeButton
1[clean/0, x/1], press_cutHorizonLinkButton 0[clean/1, x/0],
press_applyButton 2[op/1, clean/4, y/1], press_undoButton
4[clean/1, op/1], press_closeButton 1[clean/0, x/1],
press_cutHorizonLinkButton 0[clean/1, x/0], press_closeButton
1[clean/0, x/1], press_cutHorizonLinkButton 0[clean/1, x/0],
press_createPolygonButton 3[op/1, clean/0],
press_createPolygonButton 3[op/1, clean/1],
press_createPolygonButton 3[op/1, clean/1]
```

Figura 45 – Um dos logs gerados na simulação automática e randômica

```
Relatório da Análise de Mutantes

Classes sob mutação:
../v3o2_pesquisa_raquel/v3o2/src/horizon/cutting/CutHorizonPresenter.cpp
../v3o2_pesquisa_raquel/v3o2/src/horizon/cutting/CutHorizonArea.cpp

Suíte de teste:
../v3o2_pesquisa_raquel/v3o2/src/horizon/CutHorizonUITest.cpp


Número de operadores de mutação: 14

Estatísticas de operadores e mutantes:
OPERADOR         MUTANTES GERADOS        MUTANTES MORTOS
ROR                      4                       1
LOR                      2                       0
AOR                      5                       0
BVR                      16                      6
LRQD                     0                       0
LRLD                     0                       0
AOD                      0                       0
ROD                      0                       0
GTK_SRSD                 0                       0
GTK_SSND                 3                       2
GTK_SSLD                 2                       1
GTK_SID                  0                       0
GTK_SSNC                 0                       0
GTK_SSLC                 1                       1


Escore de Mutação: (Mut. mortos / Mut. gerados - Equivalentes)
Total de mutantes gerados: 33
Total de mutantes mortos: 11
Mutantes equivalentes: 3


Escore de Mutação = 36.6667%
```

Figura 46 – Relatório da Análise de Mutantes para Recorte de Horizonte

ESTATÍSTICAS

	OPERAÇÃO SOBRE HORIZONTES	GERÊNCIA DE ATRIBUTOS	EXTRAÇÃO DE ATRIBUTOS	RECORTE DE HORIZONTE
Tempo de criação do modelo *	6h	4h	4h	2h
Tempo de simulação da RP **	10 minutos (manual)	1 minuto	1 minuto	1 minuto
Tempo da geração dos scripts executáveis na guiftG	< 1 minuto	< 1 minuto	< 1 minuto	< 1 minuto
Tempo de ajuste manual dos scripts gerados	1h	2h	2h	2h
Número de defeitos encontrados no SUT pela suíte gerada	0	2	2	1
Escore de mutação	10,5 %	14,4 %	25 %	36,7 %
Tempo de execução da Análise de Mutantes	1h	1h	1h	40 minutos

Tabela 1 – Estatísticas da aplicação do método sobre as funcionalidades testadas

* Tempo de criação do modelo inclui o tempo de uma simulação manual, para a verificação do comportamento e realização de possíveis correções.

** Com exceção da Operação sobre Horizontes, onde a simulação foi manual, o tempo exibido é relativo a três ciclos da simulação automática, o que significa três casos de teste gerados. Cada ciclo executou vinte passos, que é o máximo permitido pela ferramenta MISTA. Dessa forma, cada caso de teste possui vinte ações. (Se no meio da simulação ocorresse um *deadlock* na rede, o número de passos não chegaria a vinte, porém isso não ocorreu.)

6.2 Análise dos resultados

6.2.1 Avaliação quanto à eficiência

Nesta seção vamos avaliar a eficiência do método, observando as medições realizadas e relatadas na Tabela 1 das estatísticas e comentando sobre os aspectos relacionados aos resultados. O método é confrontado com a escrita de testes manuais realizada atualmente no processo de teste do SUT V3O2.

O tempo de criação das Redes de Petri levou entre duas e seis horas. Essa variação ocorreu porque, no início, a falta de experiência nessa modelagem tomava mais tempo e também em razão da diferença de complexidade existente naturalmente nas GUIs. Podemos considerar que para as funcionalidades Gerência de Atributos, Extração de Atributos e Recorte de Horizonte, a modelagem foi realizada já após bastante "treino". A funcionalidade Recorte de Horizonte levou menos tempo, duas horas, porque é a funcionalidade menos complexa. Portanto, podemos considerar que um tempo médio de modelagem de uma GUI como Rede de Petri para uma pessoa já com conhecimento e experiência nesta modelagem, seria quatro horas. Neste tempo está incluída uma rápida simulação para verificação e validação do comportamento do modelo, garantindo que ele condiz com a especificação da GUI. Estão incluídas também as correções, se necessárias.

Podemos considerar que temos ainda as etapas de simulação automática da RP para gerar os casos de teste e de geração dos scripts executáveis na ferramenta guiftG. No entanto, se observarmos na tabela, o tempo somado para essas duas não deve passar de dois minutos, o que o torna irrisório e nos permite manter por enquanto o tempo médio de geração da suíte de teste em quatro horas.

Há ainda o tempo de ajuste manual dos scripts gerados pela ferramenta. Esse ajuste é necessário, porque nem sempre um tipo de evento já estava implementado na ferramenta guiftG, já que algumas necessidades surgiram conforme as modelagens, como o evento de desenhar um polígono no horizonte. Também, porque pode haver diferentes formas de instanciar as classes envolvidas no teste. Quanto mais a sintaxe e a semântica do modelo forem evoluídas junto com a ferramenta que o interpreta, menor será a necessidade de ajuste dos scripts gerados. No entanto, é esperado que algum tipo de ajuste sempre será necessário. Podemos então somar ao tempo de criação dos scripts de teste do nosso método, o tempo de ajuste explicitado na Tabela 1. O tempo médio é de 1,75 horas, no entanto, com um es-

força pequeno podemos incrementar a ferramenta guiftG para que ela exija menos necessidade de ajuste. Logo, podemos considerar justo que este tempo seja de 1 hora, que somado ao tempo atual, temos um total de 5 horas.

Num processo onde os testes são manuais, a etapa de planejamento dos casos de teste que serão implementados pode ser comparada à criação do modelo. E a etapa da implementação dos testes, considerando também possíveis correções na execução do script criado, pode ser comparada à geração dos scripts executáveis. O tempo de planejamento dos testes de interface no V3O2 é de no máximo 2 horas. A implementação manual destes testes leva em média 3 horas. Isso nos leva a concluir que é esperado que no máximo cinco horas sejam gastas com um teste criado manualmente e que este tempo se assemelha ao tempo gasto na criação de um modelo. Contudo, podemos esperar que o rigor da suíte gerada por um modelo formal e completo em relação às possibilidades de interação na GUI, levará a uma suíte de maior qualidade.

O tempo de execução da Análise de Mutantes (AM) também está ilustrado na Tabela 1. A AM é uma avaliação que pode ser aplicada a ambos os testes, portanto não entra nesta análise o fator comparação entre os tipos de geração de teste. Além disso, ela não é executada repetidamente nos testes de um projeto de software. Deve ser executada apenas após a criação de uma suíte, para avaliar a sua eficácia e ajudar a incrementar os testes. O tempo médio de execução da AM foi de 1h, no entanto podemos afirmar que com uma melhoria da ferramenta de AM, resultando na adição de alguns operadores, esse tempo pode ficar entre 1h e 3h.

6.2.2 Avaliação quanto à eficácia

As suítes de teste geradas a partir do modelo e da simulação automática da Rede de Petri, foram capazes de revelar problemas nas funcionalidades testadas. Em outras palavras, os defeitos relatados a seguir foram detectados por um oráculo gerado a partir da Rede de Petri.

É válido recordar que a simulação automática tem o caráter randômico, significando que a escolha sobre qual transição irá disparar e com que parâmetros é feita de forma aleatória pela ferramenta que executa a simulação.

Na funcionalidade de Gerência de Atributos, dois novos defeitos foram detectados. Estes referiam-se a um estado interno do sistema, que representava um estado inconsistente no momento da verificação. Nesse momento não eram falhas

que seriam observadas pelo usuário, mas poderiam levar a uma. Um era referente à instância de uma variável que deveria se tornar nula quando a aba era fechada. A outra era o conteúdo da caixa de texto com o nome de um novo atributo, que mesmo sem aparecer, mantinha conteúdo "novo_atributo", o que poderia levar inconsistências no código. Estes dois defeitos foram corrigidos.

Na funcionalidade de Extração de Atributos, também foram detectados dois novos problemas. O primeiro era o mesmo da funcionalidade descrita acima, sobre uma instância que não era anulada após o fechamento da janela. Este defeito foi corrigido. O segundo foi uma falha de segmentação, cuja causa não chegou a ser identificada.

Na funcionalidade Recorte de Horizontes, a falha descoberta era relativa à criação do polígono que define a área de recorte. Somente depois que o desenho do polígono é finalizado (ele é fechado) é que o botão Aplicar deve tornar-se habilitado. O problema revelado era que esse botão tornava-se habilitado assim que se iniciava a construção do polígono.

Alguns dos defeitos encontrados foram relatados na ferramenta do tipo *bug tracker* do projeto V3o2, para futura correção.

Em relação ao número de defeitos encontrados com os casos de teste gerados, podemos fazer as seguintes observações:

- As quatro funcionalidades já foram finalizadas e já estão em produção há algum tempo, por isso já tiveram boa parte dos seus problemas identificados e corrigidos.
- As redes modeladas ainda não refletem 100% das interações e estados possíveis. Alguns *widgets* não foram considerados neste experimento para não tornar a rede ainda maior e complexa. Além disso, alguns eventos ainda apresentaram problemas na execução do teste, como o de renomear um horizonte, tendo que ser descartado.

A eficácia de uma suíte de teste é dada pelo número de defeitos que os testes são capazes de encontrar, em relação ao número total. Não temos como saber o número de defeitos de um software, mas através da análise de mutantes pode-se ter uma boa estimativa.

Na Tabela 1 podemos ver, por fim, os escores de mutação obtidos com a Análise de Mutantes, os quais refletem a eficácia. Os valores obtidos não foram

tão altos como seria esperado. Quanto a isso, pode-se fazer algumas considerações que dizem respeito à evolução das etapas do método proposto.

- Acredita-se que se o modelo da Rede de Petri for evoluído, no sentido de termos próximo de 100% da GUI e das interações externas possíveis, especificadas, teremos casos de teste mais completos e portanto, maiores chances de identificar os defeitos contidos naquela interface. Em consequência, aumenta também a probabilidade de que os mutantes gerados sejam mortos pelos testes, melhorando a eficácia.
- A eficácia obtida com o escore de mutação é sensível ao número de casos de teste gerados e ao seu comprimento, principalmente por se tratar de teste aleatório. Dessa forma, é possível que os caminhos gerados nas simulações dos experimentos não explorem sequências que provocam falhas. Se um defeito é sensível a um determinado estado e este estado depende de um fluxo específico, somente alguns caminhos vão acusar a falha.
- O resultado pode ser influenciado pelo erro atribuído ao número de mutantes equivalentes considerado neste trabalho. Como a decisão sobre mutantes equivalentes é complexa, nesta primeira abordagem consideramos um percentual fixo de 10% para o número de equivalentes, baseado em um número médio obtidos nos estudos em [34] e [22]. Entretanto, como já mencionado, em outros estudos relacionados segundo Jia e Harman em [32], o percentual médio fica entre 10% e 40%, o que pode influenciar bastante a eficácia obtida. O percentual fixo escolhido de 10% é otimista e foi uma proteção a não obtermos resultados ilusórios. Por outro lado, temos que os escores obtidos podem ser melhorados se for aplicado um método de estimativa mais precisa no número de mutantes equivalentes.
- Em [34], uma ferramenta de mutação para Java foi utilizada em um sistema. Mesmo com uma ferramenta já mais evoluída, muitos operadores tiveram escores baixos, que iam de 0% a 25%. No entanto, uma análise foi realizada e identificou-se que muitos destes mutantes pertenciam a partes do código irrelevantes ao modelo e por isso foram descartados. Com isso, o escore geral obtido com a aplicação da AM foi de 79%. É importante lembrar que a ferramenta MATool começou como um protótipo e pode ser

ainda evoluída no sentido de criarmos mais operadores voltados para a lógica da interface gráfica.

6.2.3 Evidências quanto às ferramentas Capture/Replay

Apesar de não terem sido realizados experimentos acerca de ferramentas Capture/Replay (descritas na seção 1.2), onde a criação de testes para interface gráfica é manual, podemos equiparar alguns fatores da eficiência e da eficácia desta ferramenta a scripts de teste criados manualmente. E então confrontarmos com a abordagem de teste baseado em modelo apresentada neste trabalho.

- Planejamento dos casos de teste: Teoricamente seria gasto o mesmo tempo do planejamento em um teste manual, que é próximo do tempo de modelagem da rede.
- Geração dos casos de teste e scripts: São gerados um por vez, interagindo-se com o software. Isso assemelha-se à implementação dos scripts, no caso do teste manual. Comparando com a geração dos scripts no método proposto, a simulação automática da Rede de Petri pode gerar em um minuto, três casos de teste longos.
- Atualização dos testes quando há atualização/evolução dos requisitos: Nas ferramentas Capture/Replay, realiza-se nova interação na GUI, recapturando todos os casos de teste ou altera-se manualmente os scripts (sujeito erros). Essa atualização é semelhante à atualização no caso de testes manuais. No caso de testes gerados pelo modelo da RP, a atualização é feito no modelo e a suíte completa é regerada automaticamente.
- Qualidade da suíte obtida: Em testes manuais e no método Capture/Replay, a qualidade da suíte gerada está diretamente ligada à capacidade do testador de contemplar todos os casos de teste necessários, o que pode ser uma tarefa falha. No método proposto a qualidade da suíte está ligada à qualidade da rede modelada. Estando a rede completamente especificada, a cobertura dos testes é dada pela cobertura da rede e está relacionada com a abordagem aleatória de geração de testes. Além disso, caso a RP esteja disponível antes do desenvolvimento, é permitido esperar que o número de defeitos injetados ao desenvolver seja menor do que quando a RP não esteja disponível.

Portanto, a eficiência e a eficácia de uma ferramenta Capture/Replay podem ser próximas as dos scripts de teste criados manualmente, com a diferença de que a primeira fornece uma abordagem de criação de testes mais intuitiva e prática, já que é realizada interagindo-se diretamente com o software. Contudo, se estas forem comparadas à abordagem de geração de testes pela RP, as primeiras devem ser significativamente mais caras, quando se procura o mesmo rigor nos testes.

6.3 Considerações

A Rede de Petri pode ser modelada antes de a funcionalidade ser desenvolvida, servindo para a verificação da especificação. Como já mencionado, isso contribui para um código inicial contendo menos defeitos.

Em termos de eficiência, os testes para GUI criados manualmente, seja à mão ou por ferramentas Capture/Replay, onde o custo inicial está no planejamento e na implementação dos testes, poderiam ser equivalentes aos testes gerados a partir do modelo Rede de Petri, porque a construção de um modelo completamente especificado e formal, demanda um custo.

No entanto, visamos testes com um alto grau de rigor e nesse caso o teste vindo do modelo RP apresenta maior eficiência, porque caso se invista um longo tempo planejando os testes manuais na tentativa de se obter um rigor alto, este planejamento ultrapassaria o tempo de modelagem da RP. Sendo assim, esta última passa a ser mais eficiente, mesmo porque, a partir daí irá gerar casos de teste longos, abrangendo muitos eventos, em cerca de minutos. Se pouco ou nenhum planejamento for realizado nos testes manuais, temos que a qualidade da suíte é no mínimo duvidosa.

A abordagem aleatória na geração de casos de teste tem várias frentes de estudo e em trabalhos como [27] e [28]**,** é avaliada como superior às gerações baseadas em critérios específicos. No caso de testes em GUI, por exemplo, se pensarmos em critérios de cobertura do tipo *all-states,* poderíamos chegar ao famoso problema da explosão de casos de testes. Como mencionado por Hamlet em [27], mesmo realizando-se uma filtragem dos casos de teste gerados, o testador não está apto a determinar quais são os melhores casos. O tipo de geração adotada na ferramenta MISTA procura cobrir todas as transições e escolhe uma sequência de

eventos aleatória, muitas vezes incoerentes com o que seria executado por um usuário comum. Por isso, testes aleatórios, com seus casos de teste longos e mesmo repetitivos, sugerem boas chances em detectar problemas.

Um indício sobre isso foi observado neste trabalho, na primeira vez em que as redes das funcionalidades foram simuladas. Inicialmente a simulação foi manual, executando-se os caminhos que se julgavam ser os necessários para os casos de teste.

Com exceção do Recorte de Horizonte, que só teve suíte gerada automaticamente, a evolução dos escores de mutação foram como mostra a Tabela 2 abaixo:

	Operação sobre horizontes	Gerência de Atributos	Extração de Atributos	Recorte de Horizonte
Simulação manual da RP	10,5 %	12,2 %	13,4 %	X
Simulação automática e randômica da RP	X	14,4 %	25 %	36,7 %

Tabela 2 – Escores de mutação obtidos com simulação automática e randômica comparados com simulação manual.

Apesar de a ferramenta de AM desenvolvida não estar completa quanto a sua implementação, a análise de mutantes foi percebida como uma boa técnica de avaliação da eficácia, principalmente porque auxilia no incremento dos testes desenvolvidos. Como fornece uma medida cada vez que é executada, permite a adoção de uma meta de eficácia e pode-se investir para chegar àquela meta.