2 Teste baseado em modelo

O MBT tem-se tornado foco de estudos e experimentações práticas. Apesar das vantagens observadas, ainda há obstáculos para a adoção em grande escala de suas técnicas: Um deles é a construção de modelos formais de sistemas complexos do mundo real, que podem tornar-se extremamente grandes e difíceis de manipular [9]. Outro obstáculo é o grande número de testes gerados, especialmente quando um trabalho manual é requerido para ajustar os scripts de forma a torná-los fielmente executáveis.

Uma das abordagens de testes baseados em modelo é voltada para a modelagem de GUIs. Uma GUI possui objetos gráficos e cada um contém um conjunto de propriedades. A qualquer momento durante a execução, estas propriedades recebem valores, os quais caracterizam o estado da GUI.

Tendo em vista essas características, alguns estudos são voltados para a modelagem de GUIs como um tipo de máquina de estado, visando principalmente a geração de casos de teste. Neste modelo, um caminho percorrido representa um caso de teste. Com isso, os modelos de GUI equivalem a grafos de fluxo de eventos.

Uma característica importante e complexa é o tipo de resposta que um evento associado a um elemento de interface pode produzir, que é dependente dos estados e da ordem de execução anteriores. O grau de liberdade a que um usuário final dispõe quando interage com uma interface gráfica revela o grande número de permutações permitidas de eventos, principalmente se considerados sistemas não triviais. Determinar o que testar, definindo um critério de cobertura, e o quanto testar é um dos maiores desafios do teste em GUI, devido à possível explosão do número de estados possíveis.

2.1 Redes de Petri

Rede de Petri é uma representação matemática para sistemas concorrentes e como linguagem de modelagem define graficamente a estrutura de um sistema como um grafo direcionado. A Rede de Petri foi inventada por Carl Adam Petri e documentada muito tempo depois como parte de sua tese de doutorado, em 1962 na Alemanha. [10]

As RPs são poderosas na análise de propriedades e problemas associados a sistemas concorrentes, tais como: Relações de precedência entre eventos, sincronização e existência/inexistência de bloqueios.

Tal como ilustra as figuras 6a e 6b, uma RP é composta por nós e arcos [10]. Os nós são os lugares e as transições, representados por círculos e retângulos respectivamente. Os lugares armazenam marcas (ou *tokens*, como são referidos nesse trabalho) e as transições representam um processamento ou um evento. Os lugares também podem ser ditos pré ou pós condições de uma transição (evento). Os arcos direcionais conectam lugares com transições e vice-versa, mas nunca conectam dois lugares ou duas transições, o que caracteriza uma RP como um grafo bipartido. A qualquer momento na execução de uma Rede de Petri, um lugar pode estar vazio ou armazenar um ou mais *tokens*. Uma transição só pode disparar quando o número especificado de *tokens* aparecer em cada lugar de entrada. Um disparo é atômico e, nesse momento, a transição consome os dados de cada lugar de entrada, realiza uma tarefa de processamento e produz dados nos lugares de saída (podendo também apenas consumir os *tokens* entrantes, sem produzir nenhuma saída. Isso é possível quando a transição não possui, de fato, lugares na saída).

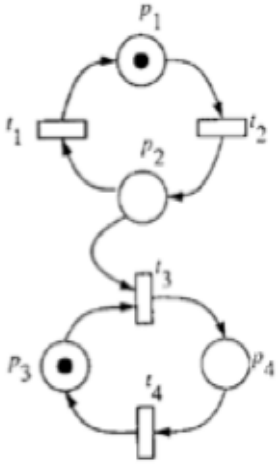


Figura 6.a – Exemplo de RP do tipo ordinária

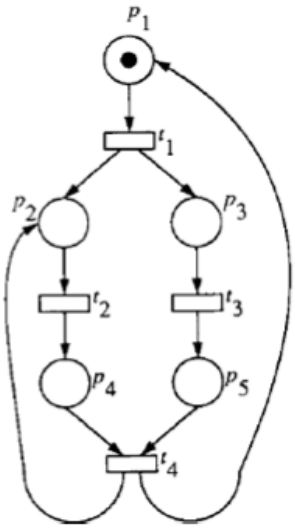


Figura 6.b – Exemplo de RP do tipo ordinária

A execução de uma RP é não-determinística. Isso significa que quando mais de uma transição se encontram habilitadas, não há uma ordem de disparo, qualquer uma pode executar a qualquer momento. Como disparos são não-

determinísticos, as Redes de Petri são utilizadas para modelar o comportamento concorrente em sistemas deste tipo.

A figura 6.b acima ilustra uma estrutura tipicamente concorrente. Após o disparo de t1, t2 e t3 estão habilitadas. O disparo de t2 não afeta t3 e vice-versa. Elas são transições concorrentes/paralelas. A concorrência não é unicamente definida pela estrutura, ela também pode depender da marcação. Uma estrutura aparentemente não concorrente pode implicar em disparos concorrentes de transições. Considere a estrutura condicional na Figura 7, que dessa vez representa uma rede de alto nível. Dada a marcação inicial M0 = p1(1,2,3), onde há em p1 três tokens de valores 1, 2 e 3 respectivamente, as três transições podem ser disparadas simultaneamente por respectivas satisfações de variáveis x = 1, x = 2 e x = 3.

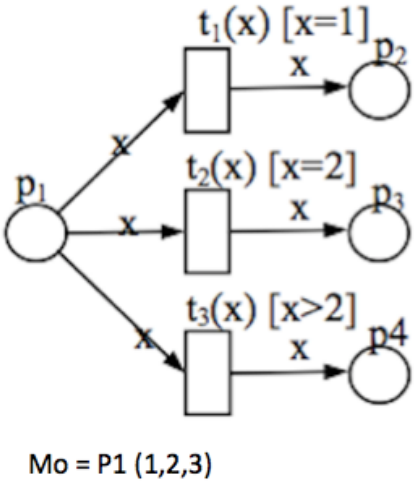


Figura 7 – Exemplo de RP do tipo alto nível, com condições e marcação incial que configuram transições concorrentes.

Uma RP pode ser definida formalmente pela quíntupla: PN = (P, T, F, w, Mo), onde:

P = {p0, p1, …, pm} é o conjunto finito de lugares

T = {t0, t1, ..., tn} é o conjunto finito de transições

F $\subseteq$ (PxT) U (TxP) é o conjunto de arcos (relação de fluxo)

W: F → {1,2,3,…} é uma função que dá peso aos arcos

Mo: P → {0,1,2,…} é a marcação inicial da rede (número de tokens em cada lugar)

Além disso, tem-se que: $P \cap T = \varnothing$ $P \cup T \mathrel{!}= \varnothing$

Uma característica importante das Redes de Petri é que elas podem ser simuladas – ou executadas – processo também conhecido por *token game*. A execução da rede numa ferramenta gráfica permite a validação visual do comportamento do sistema, no qual lugares com *tokens* representam o estado atual e as transições, com suas regras de disparo, modelam o comportamento dinâmico do sistema. Essa validação busca garantir que a rede esteja corretamente definida e corresponda com exatidão ao sistema desejado.

A modelagem de um sistema como uma RP permite prever e testar o comportamento do software antes da implementação, seja do código principal ou dos testes.

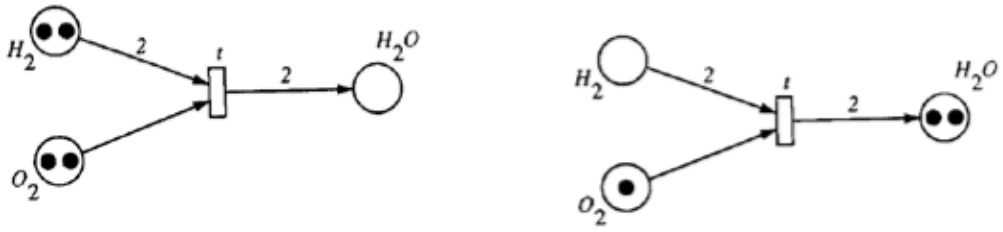


Figura 8 – Exemplos de RPs Generalizadas, que passam a conter peso nos arcos

O estado de uma Rede de Petri é caracterizado pelos *tokens* dispostos na rede num instante t. Dado um conjunto inicial de *tokens*, conhecido como a marcação inicial Mo, a(s) transição(ões) que se tornam ativas por esse conjunto podem disparar, consumindo os *tokens* dos lugares de entrada e produzindo outros nos lugares de saída. No caso de RPs de alto nível, a qual será introduzida mais adiante, os valores definidos para os *tokens* iniciais podem sofrer alterações quando passam pelas transições, através de expressões contidas nas mesmas. Ou seja, a cada disparo de transição o valor de um *token* pode mudar de forma que satisfaça as próximas condições.

As RPs possuem dois tipos de propriedades, estruturais e comportamentais, as quais adicionam poder de verificação e análise do modelo apoiadas em critérios matemáticos. As propriedades estruturais dependem da estrutura topológica da rede e independem da marcação inicial. As propriedades comportamentais são válidas para uma dada marcação inicial. Alguns exemplos destas propriedades são:

Comportamentais

Reachability: Uma marcação (estado) Mn é alcançável a partir de Mo se existe uma sequência de disparos de transições que transformam Mo em Mn.

Liveness: Em qualquer marcação (estado) alcançável, pelo menos uma transição pode se tornar disparável. Ou, há algum estado que não será mais alcançado indicando um possível *deadlock*? (Um estado final da rede pode não ser um *deadlock,* mas um estado esperado de término de execução.)

Reversibility: É possível retornar ao estado inicial?

Boundedness: Indica o número máximo de *tokens* que permanecerão em um lugar. Uma rede é dita *k-bounded* se o número máximo de *tokens* for no máximo k. Se k = 1, a PN é chamada de segura.

Estruturais

Controllability: Qualquer marcação é alcançável a partir de qualquer outra marcação.

Conservativeness: O número total de *tokens* na rede é constante.

Há alguns tipos de RP atualmente, originadas a partir de evoluções e extensões do modelo original projetado por Carl Petri. O tipo utilizado neste trabalho é a RP de Alto Nível, especificamente a Function Net, no entanto as outras serão introduzidas aqui para fins de contextualização.

- Rede de Petri Ordinária (original, tradicional): Os *tokens* são unicamente marcas, sem valor. São indistinguíveis. Além disso os arcos têm peso um, implicando que cada transição consome apenas um *token* por vez de cada arco. [11]
- Rede de Petri Generalizada: Pesos são associados aos arcos. Isto indica que cada arco pode consumir ou produzir mais de um *token* em um dado lugar. [11] (Figura 8)
- Rede de Petri de Alto Nível: Adiciona recursos para representar e manipular tipos abstratos de dados. *Tokens* são valorados e podem ser um multiconjunto de valores estruturados. Os arcos possuem expressões com variáveis ou constantes que definem como será feita a transmissão dos *tokens*.

- RP Colorida: *Tokens* recebem valores. O tipo do valor de um *token* é a sua "cor". Tal como numa linguagem de programação, os *tokens* são tipados, e os lugares ganham uma inscrição sobre qual tipo podem conter. [12]
- RP Predicado/Transição: Lugares são "predicados" com extensão variável. Representam propriedades variáveis de indivíduos, ou relações entre indivíduos. Ao invés de simples *tokens*, os predicados são marcados com suas extensões correntes. Por exemplo: Um lugar W<*u,m*> pode ter o seguinte significado: "O usuário *u* deseja usar o recurso em modo *m*". E um *token* entrante é uma tupla <u,m> com valores em u e m. [13]
- RP Function Net: É uma versão simplificada *(light-weight)* das Coloridas e Predicado/Transição. Não há *tokens* duplicados no mesmo lugar. Não permite a inscrição em cada lugar, dos tipos de valores aceitos. Arcos são rotulados com variáveis. [9]

- RP P-Temporizada: Adiciona um tempo pelo qual um *token* deverá permanecer em um lugar, desde o momento em que ele é adicionado lá. Somente após decorrido esse tempo, o lugar pode contribuir para que uma transição fique habilitada e consequentemente remova este *token* do lugar. [11]
- RP T-Temporizada: Adiciona tempo de latência a uma transição. Ou seja, o disparo de uma transição leva tanto tempo quanto o definido para ela. [11]

As RPs temporizadas podem ainda ser determinísticas ou estocásticas. Na primeira os atrasos são determinísticos, enquanto na segunda os atrasos podem ser modelados segundo uma variável aleatória.

Existem outras extensões para Redes de Petri. É importante frisar que quanto mais se adiciona complexidade às redes, mais difícil se torna utilizar ferramentas tradicionais para calcular certas propriedades.

O tipo de RP utilizado neste trabalho é a Function Net. A razão disso é que o trabalho requeria uma RP do tipo Alto Nível para que se pudesse representar elementos de interface com suas propriedades e valores. Na busca por uma ferramenta gráfica adequada e atualizada que satisfizesse alguns requisitos, como, su-

porte a RP de alto-nível, simulação e arquivo texto com a descrição da rede modelada, identificou-se a ferramenta MISTA [14], que trabalha com Function Nets. Após uma avaliação do programa concluiu-se que suas características e tipo de Rede de Petri abordada satisfaziam os requisitos desse trabalho.

2.2 GUI como Rede de Petri

A característica hierárquica da GUI, em termos de seus componentes e sub componentes, permite que ela seja decomposta em partes, onde cada uma possa ser vista como uma unidade de teste. Essa divisão em pequenos componentes visa tornar o teste mais simples e efetivo, dado que uma GUI apresenta por natureza uma grande gama de combinações de eventos e explosão no número de estados.

Na representação de uma GUI como uma Rede de Petri, um lugar pode denotar um *widget* com uma propriedade associada*; Tokens*, os valores das propriedades; E transições, os eventos do usuário.

Na figura 9b a interface gráfica de login de um site de cartão de crédito está modelada como uma RP. A figura 9a é o *screen shot* desta funcionalidade. Esta GUI funciona da seguinte forma: Uma Combobox dispõe as opções para os tipos de cartão de crédito existentes: Pessoal, Corporativo, e outros. O primeiro item é "ESCOLHA UMA OPÇÃO". Se este item for selecionado, os campos "Usuário" e "Senha" e o botão "Acessar" devem ser desabilitados. Qualquer outro item selecionado na Combobox deve reabilitá-los.

Figura 9a – *Screenshot* de tela de *login* de um site de cartão de crédito

Na rede da figura 9b, cada lugar (círculo) representa um *widget* com a respectiva propriedade a ser verificada. A sintaxe adotada para lugares é *propriedade_nomeWidget*. Nesta representação, só poderá haver um *token* por vez em um lugar, já que este *token* conterá um valor único, que é o da propriedade do *widget*.

Os seguintes lugares têm os seguintes significados:

select_optionsComboBox: Será verificado qual item está selecionado na Combobox de opções (tipos de cartões de crédito: Pessoal, corporativos, estabelecimentos, etc.) [índices de 1 a n]

active_userTextBox: Será verificado se o campo de texto de nome de usuário está habilitado (>0) ou desabilitado (0).

active_passwdTextBox: Será verificado se o campo de texto de senha está habilitado (>0) ou desabilitado (0).

type_userTextBox: Será verificado o que está digitado no campo de texto de nome de usuário (algum texto).

type_passwdTextBox: Será verificado o que está digitado no campo de texto de senha (algum texto).

active_accessButton: Será verificado se o botão de acesso está habilitado (>0) ou desabilitado (0).

__selections: Esse lugar é apenas um armazenador para os *tokens* iniciais (dados do teste) e não é considerado como oráculo no caso de teste. (O prefixo "__" é uma sintaxe adotada neste trabalho para identificar que um não é um elemento da GUI) Os *tokens* aqui representam os possíveis valores para seleção na combobox de opções.

__tobetyped: Mesmo caso do item acima, mas nesse caso os *tokens* armazenados são dados de entrada para os campos, nome de usuário e senha.

As transições (retângulos) têm a mesma sintaxe, *propriedade_nomeWidget,* denotando que uma ação será aplicada numa propriedade do *widget*. Foi convencionado que o valor aplicado à propriedade será definido pelo arco de entrada que contém o mesmo nome da propriedade.

Os arcos da figura que não apresentam a seta, são arcos bidirecionais, ou seja, representam dois arcos de mesmo valor, um na direção lugar->transição e outro na direção inversa. Estes arcos servem para conservar um *token* no lugar, quando a transição disparou. É usado por exemplo ligando o lugar __selections à

transição select_optionsComboBox. Dessa forma, ao disparar, a transição consome e põe de volta o *token* no lugar. Isso é útil porque este lugar armazena as possíveis opções de seleção da combo de tipos de cartão de crédito, as quais devem estar disponíveis sempre.

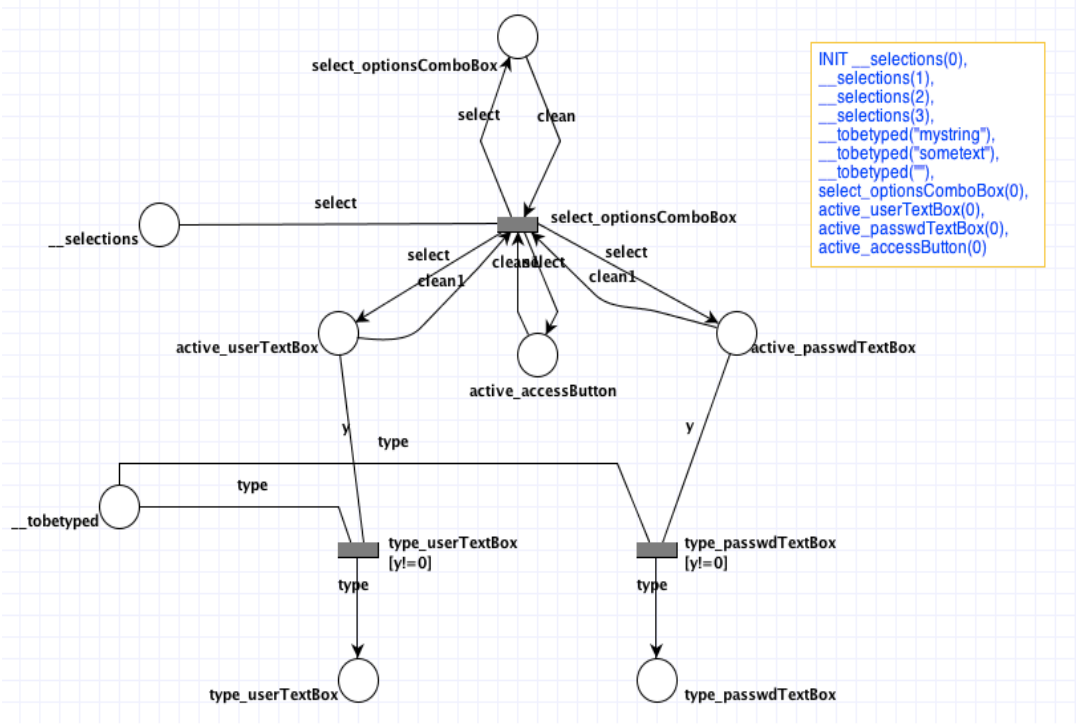


Figura 9b – Interface gráfica da tela de login modelada como Rede de Petri

Na simulação da Rede de Petri, uma marcação inicial é selecionada para dar início a uma sequência de eventos. Quando a execução é manual, o usuário pode escolher qual transição será disparada no momento em que mais de uma se encontra habilitada. Quando a execução é automática, a ferramenta escolhe a transição que irá disparar. Após o término de uma execução, a mesma marcação inicial pode ser usada para percorrer outro caminho, ou pode-se selecionar outra marcação inicial. Com uma ou outra opção, isso denota outro fluxo de eventos e assim por diante.

O conjunto das simulações sobre uma rede especificada produz uma suíte de teste. Para que seja possível gerar casos de teste a partir das simulações, a ferramenta gráfica de modelagem da RP deve produzir os logs destas execuções.

Casos de teste são essencialmente sequências de disparos de transições a partir de uma marcação inicial (Mo). A marcação inicial é o conjunto dos *tokens* iniciais da rede, seus valores e os lugares onde eles se encontram. Várias marca-

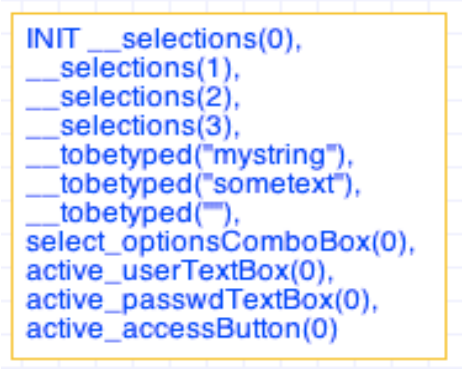
ções iniciais podem ser especificadas para a mesma rede. Dessa forma, pode-se repetir um caminho com marcações iniciais distintas, bem como executar caminhos diferentes com uma mesma marcação inicial. Com isso, uma sequência de disparos de transições produz um caso de teste concreto, já que os dados (*tokens*) possuem valor ao longo de toda a execução. Um exemplo de marcação inicial pode ser o seguinte: Um lugar P1 da rede vai armazenar inicialmente 6 *tokens*. Esse lugar vai estar ligado a uma transição TypeNumber, que digita um número num campo qualquer da interface. Cada um destes *tokens* vai ter um valor que deve ser usado no teste para preencher este campo, provavelmente valores que testem se ultrapassou-se o máximo ou o mínimo permitido naquele campo, que será parâmetro de um algoritmo qualquer. Portanto, os valores dos dados que comporão os casos de teste são definidos nas marcações iniciais ou em anotações nas transições, onde se podem definir valores de saída para arcos de saída.

Em um caso de teste, cada transição é um estímulo/ação e os *tokens* dispostos na rede a cada instante configuram os oráculos, por isso após cada transição, o número de verificações será igual ao número de *tokens* na rede.

O resultado da simulação da Rede de Petri acima, modelada para o exemplo do login de um cartão de crédito se encontra logo abaixo:

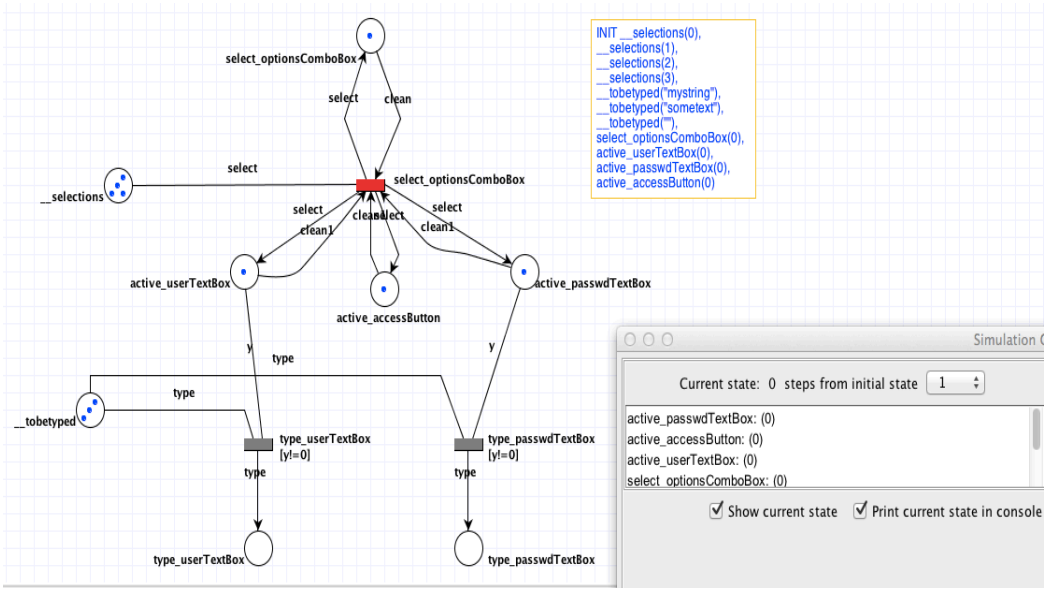
Marcação inicial

A marcação inicial pode ser usada pela ferramenta que gera os scripts de teste, para testar os valores *defaults* da interface gráfica, antes do início da execução da rede. Na ferramenta de modelagem, a ela deve ser definida no campo INIT.



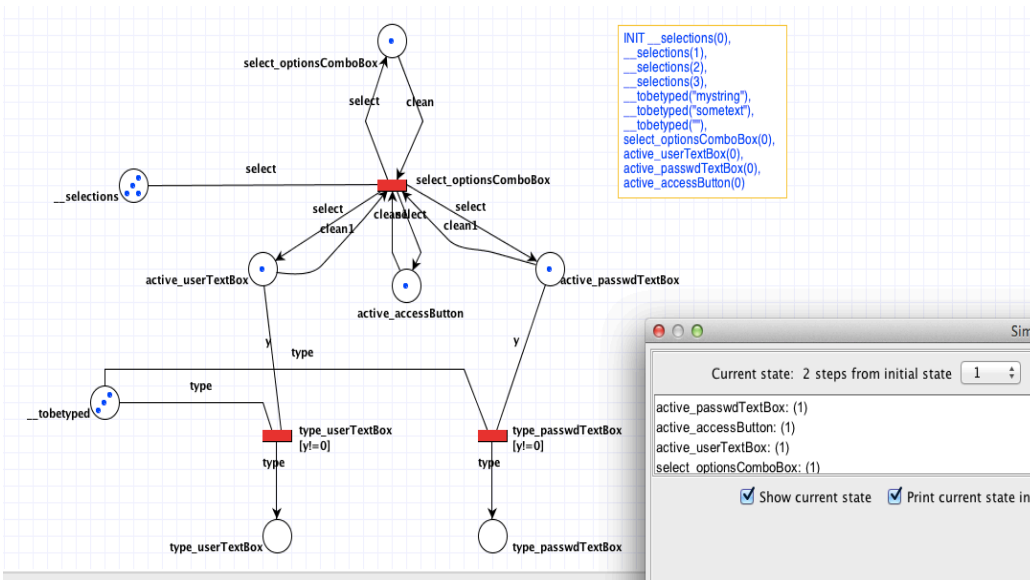
→ Testa se a combo de opções inicia selecionada no primeiro item.
→ Testa se os campos de nome de usuário e senha iniciam desabilitados. (Porque o item inicialmente selecionado na combo de opções, não é uma opção válida.)
→ Testa se o botão "Acessar" inicia desabilitado.

1º passo - 1ª Transição pronta para disparar



2º passo

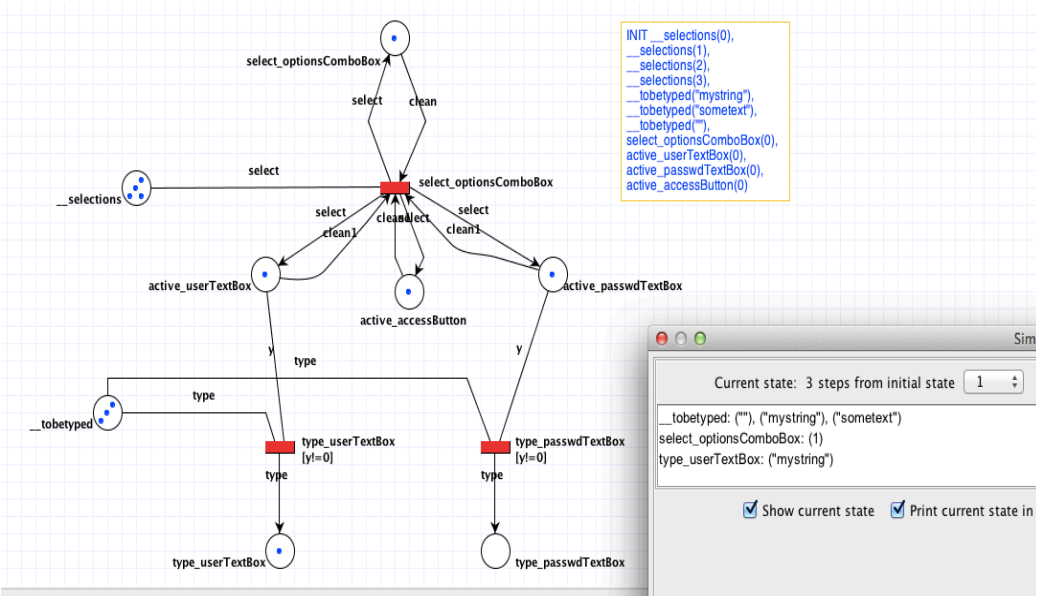
→ A transição select_optionsComboBox representa que o usuário vai selecionar a opção 1 (valor do *token*) da combo e esse evento vai produzir um *token* de valor 1 para as propriedades active (1 = habilitado) dos campos de texto de usuário e senha. E também produzirá um *token* de valor 1 no lugar select_optionsComboBox, indicando que agora de fato a opção selecionada é a primeira da combo.



→ Após o disparo haverá a verificação do estado da rede (oráculo). Cada *token* em um lugar representa o que será verificado naquele *widget*. Nesse momento, os próximos eventos já se encontram "disparáveis".

3º passo

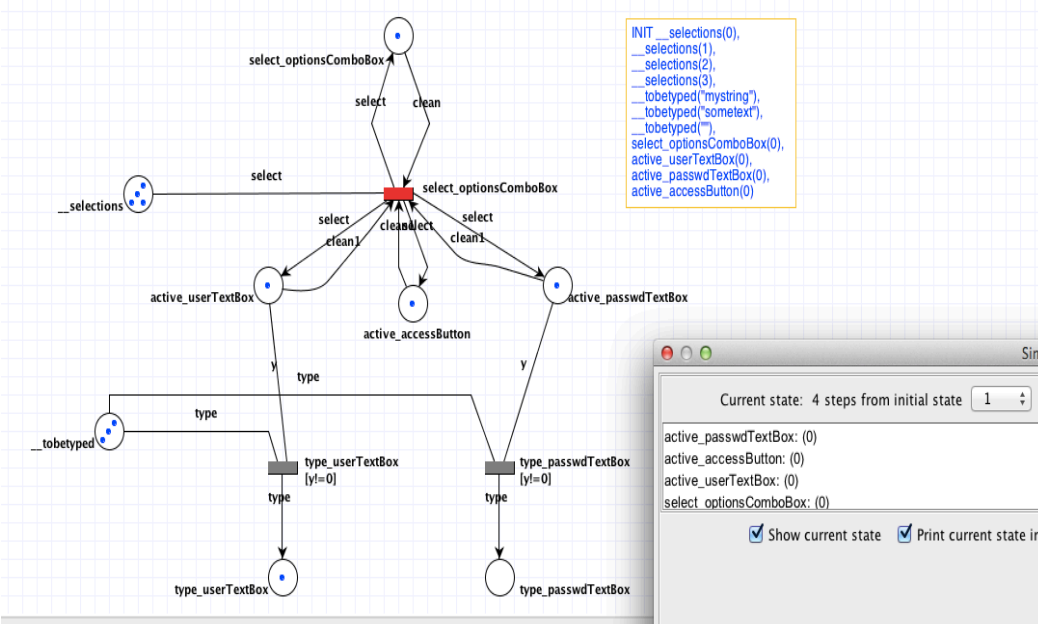
→ No próximo disparo, o usuário irá escrever um texto no campo "Usuário". Pode-se notar que as transições que escrevem nos campos "Usuário" e "Senha" só estão habilitadas porque há uma condição *y != 0*, onde *y* está definido num arco de pré-condição dessa transição. Essa pré-condição determina que só é possível escrever nestes campos se eles estão ativos.



→ Neste momento haverá novamente a verificação do estado da rede e um dos oráculos irá verificar se o campo de nome de usuário está com o conteúdo que foi digitado.

4º passo

→ Há uma nova seleção na Combobox, desta vez selecionando o item zero ("ESCOLHA UMA OPÇÃO"). Em seguida, haverá a verificação se os campos de texto e o botão "Acessar" tornaram-se desativados novamente (valor zero).



A modelagem acima, com lugares e transições associados a um par {propriedade,widget} foi a abordagem criada e utilizada neste trabalho.

As GUIs especificadas como RP fornecem um modelo mental para como os usuários entendem os comportamentos desejáveis e indesejáveis de um sistema.