Ficha Prática #03

3.1 Objectivos

Esta Ficha Prática tem como objectivo praticar o desenho de **Diagrama de Máquinas de Estado**.

Uma das possíveis utilizações das máquinas de estado e a modelação do comportamento da interface de uma aplicação (veja a Secção 3.1.1). Não é no entanto a única e esta ficha sobre diversos tipos de utilização.

3.1.1 Prototipagem da interface

Na primeira fase de entrega do trabalho de DSS é-lhe pedido que pense a camada de apresentação que a aplicação que está a modelar vai apresentar.

Pretende-se que especifiquem num protótipo de baixa fidelidade a camada de interface com o utilizador, descrevendo o controlo de diálogo com recurso à utilização de Diagramas de Máquinas de Estado.

Recomenda-se que siga o processo de modelação que se apresenta:

1. Desenhe ***mockups*** dos diversos écrans da interface da sua aplicação seguindo a abordagem apresentada nas aulas teóricas. Quando estiver satisfeito com o resultado, utilize um programa de prototipagem (e.g. Pencil[1]), ou então o IDE da sua preferência (com a vantagem acrescida de estar a poupar tempo de desenvolvimento mais à frente), para criar a versão final.

2. Defina, numa Máquina de Estado, o **mapa de navegação** entre os diferentes écrans (se estiver a utilizar uma ferramenta de prototipagem que o suporte, poderá incluir essa navegação no protótipo).

[1] `http://pencil.evolus.vn`

3. Poderá depois detalhar a **interacção com cada um dos écrans**, novamente recorrendo a Máquinas de Estado.

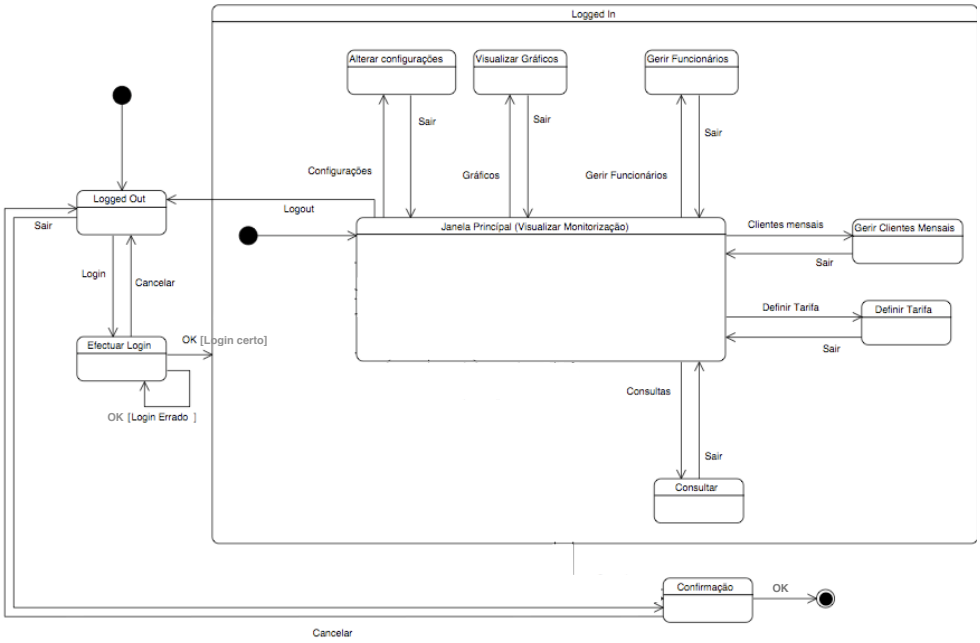

Figura 3.1: Mapa de navegação

Nas Figuras 3.1 e 3.2 mostram-se dois exemplos de diagramas de um trabalho entregue numa edição passada da disciplina. A imagem 3.1 mostra o diagrama para a janela inicial e a imagem 3.2 descreve a interacção numa *form* da aplicação.

3.2 Exercícios

Para os exercícios abaixo propostos analise os enunciados e crie os Diagramas de Máquinas de Estado pedidos.

3.2.1 Stack

Considere uma classe `StackBilhete` que implementa os seguintes métodos:

```
public void push(Bilhete b);
public void pop();
public Bilhete top();
public boolean isEmpty();
```

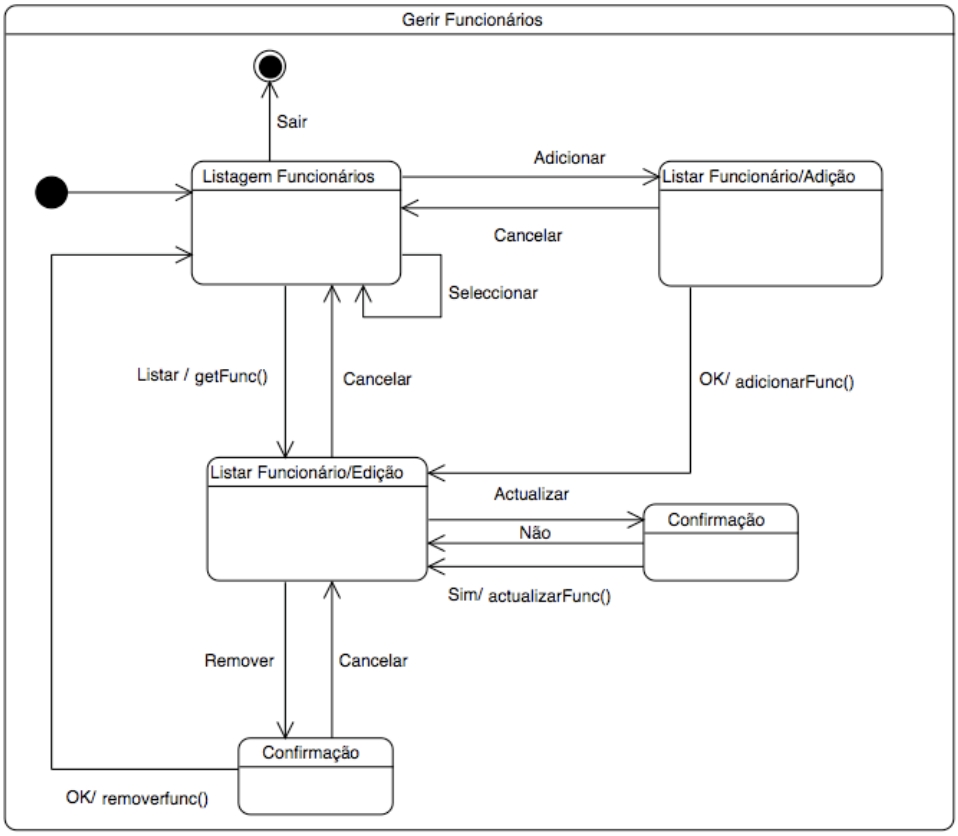

Figura 3.2: Diagrama de Estado de um écran

com o comportamento esperado (e bem conhecido) de uma Stack.

Escreva um **Diagrama de Máquina de Estado** que descreva o ciclo de vida da Stack de bilhetes.

3.2.2 Torniquete

Como parte da preparação para o Mundial 2022 foi-lhe pedido que desenvolvesse o software de controlo dos torniquetes de validação de acessos aos estádios. Os torniquetes possuem braços metálicos que podem impedir ou permitir a passagem de pessoas conforme estejam bloqueados ou não. Possuem também um leitor de bilhetes para validação dos títulos de acesso aos jogos.

A solução que vai propor tem o seguinte modo de funcionamento:

> Ao ser ligado o torniquete fica bloqueado. Quando é inserido um bilhete válido o bilhete é retido e o torniquete é desbloqueado, ficando nesse estado até que alguém passe, altura em que bloqueia novamente. Quando é inserido um bilhete inválido, faz soar um alarme e o bilhete é

devolvido. O sistema mantém o alarme até o bilhete ser retirado do leitor, altura em que volta a estar bloqueado. Enquanto está desbloqueado o torniquete não aceita bilhetes.

O sistema possuirá também um modo de teste para permitir aos técnicos analisar o funcionamento do torniquete. Quando em modo de teste os técnicos podem realizar dois tipos de testes: testar os braços e testar o leitor de bilhetes. Para testar os braços o técnico pode bloquear e desbloquear o torniquete. Para testar o leitor de bilhetes o técnico insere bilhetes (que poderão ser válidos ou inválidos) que serão sempre devolvidos. No caso do bilhete ser inválido o sistema soa o alarme até o técnico retirar o bilhete. Os dois tipos de teste podem ser executados em paralelo.

O modo de teste pode ser activado em qualquer altura. Quando sai de modo de teste o sistema fica na situação em que estava no momento em que o teste foi iniciado. O sistema pode ser desligado em qualquer altura.

a) Escreva um **Diagrama de Máquina de Estado** que modele o comportamento do torniquete.

b) Agora que especificou o comportamento descrito, considera necessário alterar algum aspecto da proposta de modo a guarantir o correcto funcionamrnto do sistema?.

3.2.3 Relógio

Modele o modo de funcionamento de um relógio com alarme. O relógio deverá possuir três botões (modo, hora, minutos) e um comutador (activo/inactivo) para o alarme.

- O botão modo permite de percorrer de forma sequencial os diferentes modos do relógio:
  - Apresentação da hora actual (quer o alarme esteja activo ou não).
  - Regulação da hora actual
  - Regulação da hora do alarme
- O comutador alarme permite activar e desactivar o alarme. O relógio toca quando o alarme está activo, o relógio está em modo de apresentação da hora, e a hora actual é igual à hora do alarme.
- Os outros dois botões (hora e minutos) permitem incrementar as horas e os minutos. Quando o modo é regulação da hora actual, eles alteram a hora actual

do relógio. Quando o modo é regulação da hora do alrme, eles alteram a hora a que o alarme vai tocar. No modo de apresentação da hora actual os botões não têm qualquer efeito.

Escreva o **Diagrama de Máquina de Estado** que descreve o modo de funcionamento do relógio.

3.2.4 Encomenda

Considere a seguinte interface Java:

```
public interface Encomenda {
    public boolean satisfazLEnc(ArrayList<LinhaEnc> linhas);
    public boolean satisfazLEnc();
    public void envia();
    public void cancela();
    public void adiciona(ArrayList<LinhaEnc> linhas);
}
```

Uma classe que implemente a interface Encomenda deverá ter o seguinte comportamento:

> Após ser criada uma encomenda fica pendente a aguardar que todas as suas linhas sejam satisfeitas. Quando todas a linhas estiverem satisfeitas, a encomenda passa de pendente a completa.
>
> Existem duas formas de registar que uma (ou mais) linha(s) da encomenda já está/estão satisfeita(s). O método `satisfazLEnc(ArrayList<LinhaEnc> linhas)` assinala a satisfação das linhas passadas como parâmetro. O método `satisfazLEnc()` assinala a satisfação de todas as linhas ainda por satisfazer. Após estar completa a encomenda passa a poder ser enviada.
>
> O envio da encomenda é registado utilizando o método `envia()`. Após ser enviada a encomenda fica registada como expedida.
>
> Em qualquer momento antes do envio é possível cancelar a encomenda (método `cancela()`). Nesse caso ela fica registada como cancelada.
>
> O método `adiciona(ArrayLis<LinhaEnc> linhas)` pode ser utilizado em qualquer encomenda que não cancelada ou expedida para lhe adicionar novas linhas.

Escreva um **Diagrama de Máquina de Estado** que represente o comportamento descrito.