1
Introdução ao Java Swing e AWT

Como se vê pela figura, saber construir interfaces gráficas com o utilizador não significa saber construir boas interacções. No entanto, para saber desenhar e implementar boas interfaces, o primeiro passo é um passo meramente técnico que consiste em aprender uma linguagem de construção de interfaces gráficas. Assim, este capítulo do Guia dos Laboratórios de Interacção Homem-Máquina pretende iniciar os alunos na construção de Interfaces com o utilizador usando as Java Foundation Classes (JFC) Swing.

No que diz respeito à linguagem Java, a programação de Interfaces Gráficas com o Utilizador, frequentemente designadas por GUI's (do inglês Graphical User Interfaces) deu os seus primeiros passos com o Abstract Window Toolkit (AWT). O AWT tem estado presente em todas as versões do Java. Os objectos AWT são construídos sobre objectos de código nativo, o que fornece um *look-and-feel* nativo. Costumam ser designados como "o menor denominador comum de todas as plataformas".

Os objectos Swing (que são os que iremos estudar) são escritos em Java puro e apresentam o mesmo *look-and-feel* em todas as plataformas. No entanto, podem ser adaptados pelo programador para estilos particulares. No JDK 1.2, os objectos Swing são parte integrante das JFC's. Por todos estes motivos, os objectos AWT caíram em desuso e em importância ao longo do tempo.

Componentes Básicos

O seguinte esquema mostra a maioria das classes que compõem o Java Swing e mostra também a relação entre as classes AWT (a amarelo) e as classes Swing (a azul):

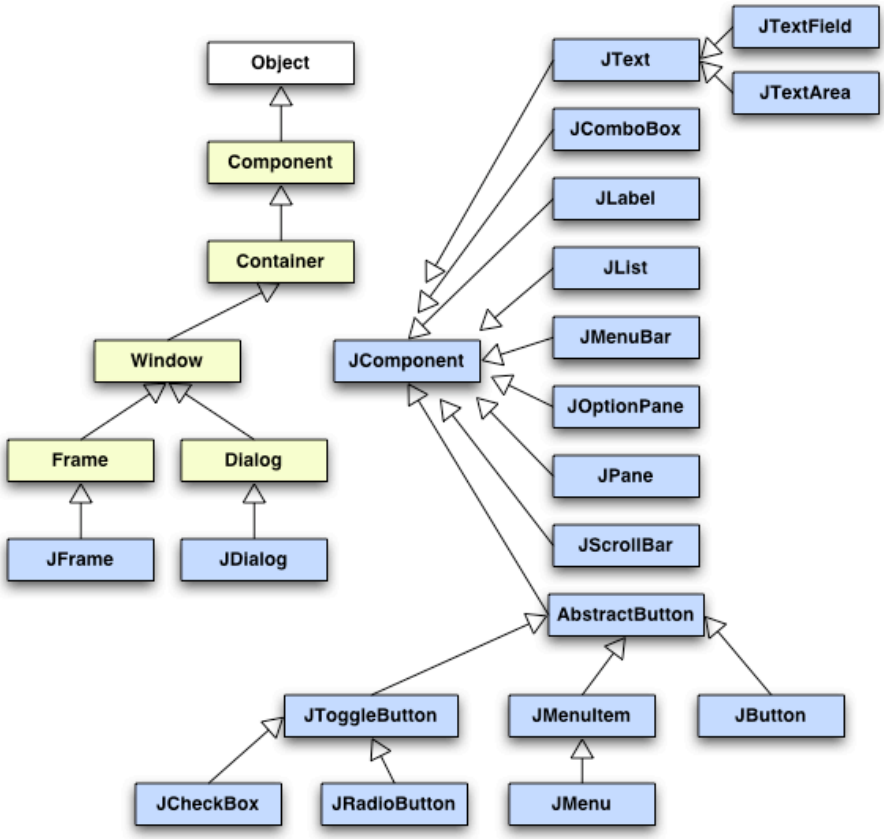


Figura 1: Diagrama resumido das classes AWT (amarelo) e Swing (azul).

Vamos começar pelas classes `JButton` e `JFrame`. Vamos criar dois botões com imagens e responder perante a selecção deles. Além de reagir ao rato vamos responder também a eventos vindos do teclado através dos chamados "aceleradores".

Obtenha os ficheiros para este exemplo: `FirstSwing.java`, `bee.gif` e `dog.gif`. Importe o pacote Swing. Defina duas instâncias da classe `Icon` na classe `FirstSwing` chamadas `bee` e `dog`. Carregue os ficheiros dessas duas imagens.

Crie dois botões com ícone chamados `bee` e `dog`. Adicione um acelerador a cada botão. Ao trabalharmos com `JFrame's` adicionamos componentes a uma área interior à `JFrame`: obtenha uma referencia para este `Container` através do método `getContentPane()`.

Para fazer com que os botões apareçam um por cima do outro, modifique o esqueleto do construtor para que o *layout* (disposição) do conteúdo seja um `GridLayout` com 2 linhas e 1 coluna. Adicione os botões a esse `GridLayout`.

Corra e teste a IU. Note que a tecla aceleradora é a "`Alt`" em Windows e Solaris e "`Option`" em Mac.

Fornecendo opções ao utilizador

Nesta secção vamos explorar as classes `JRadioButton`, `JCheckBox` e `JScrollPane`. São as classes mais usadas quando se pretende possibilitar ao utilizador a realização de uma ou mais escolhas. Serão utilizadas na construção de um programa para encomendas de pizzas que ilustra esta situação. Existem três possíveis tamanhos para as pizzas (Pequena, Média e Grande) assim como três possíveis ingredientes (Pimentos, Ananás e Pepperoni). Como designer da IU, é seu trabalho mostrar as opções ao cliente e deixá-lo seleccionar a encomenda. Uma vez seleccionada a encomenda, deve ser mostrada uma imagem da pizza especificada.

Comece por obter os ficheiros `Order.java` (código esqueleto inicial) e `pizza.jar`. Guarde o ficheiro da imagem na sua directoria de trabalho.

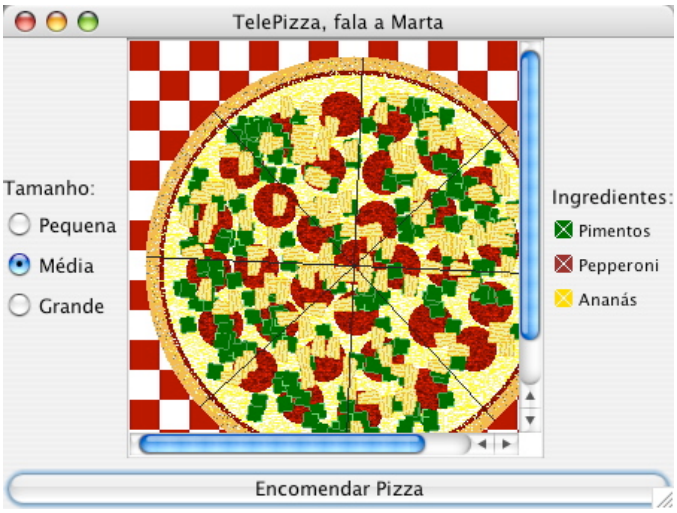
Primeiro vamos criar um `ButtonGroup` chamado `sizeGroup` com três objectos `JRadioButton`. Para o código de tratamento de eventos (já incluído) funcionar sem modificações, temos de declarar a variável `sizeGroup` como final. As etiquetas (*labels*) dos botões devem ser "Small", "Medium" e "Large". "Large" deve estar seleccionado inicialmente. Para cada botão, associe um `ActionCommand` à etiqueta de texto. Isto permite perguntar ao `ButtonGroup` qual o `JRadioButton` que se encontra seleccionado.

Coloque os botões rádio numa `Box` vertical na área esquerda do écran. Adicione uma etiqueta "Tamanho" ao topo dessa `Box`. Faça com que ela apareça a uma distância igual (verticalmente) com 5 pixels a separar os itens. O painel de conteúdo da `JFrame` pode ser referenciado pela variável local `content`.

Para os ingredientes, crie três itens `Checkbox` chamados `peppers`, `pepperoni` e `pineapple` com as etiquetas textuais apropriadas para os três ingredientes (veja a Figura com o resultado final). Devem aparecer todos não-seleccionados e todas as variáveis devem ser declaradas como final para que o código de tratamento de eventos funcione sem modificações.

Para cada uma das *checkboxes*, em vez de usar a caixa de escolha standard, crie ícones nas cores apropriadas para os estados seleccionado/não-seleccionado. Existe uma classe que suporta isto, `ColoredToggle`, cujo construtor requer uma cor e o estado de `toggle`. Existem três cores pré-definidas que podemos usar: `pepperColor`, `pepperoniColor`, `pineappleColor`.

Associe os ícones aos objectos `JCheckbox` apropriados. Use o método `setIcon()` para associar o ícone ao estado não-seleccionado e `setSelectedIcon` para o estado seleccionado.



Coloque as *checkboxes* numa `Box` vertical na área direita do écran. Adicione uma etiqueta "Ingredientes" a caixa do tamanho.

Por fim, crie um `JScrollPane` com um `JLabel` para a zona central do écran. O `label` será usado para mostrar a imagem da pizza. Necessita ser declarado como `final` e ter o nome `imageLabel`.

O botão de encomenda de pizza está já colocado no écran no quadrante Sul (no código esqueleto inicial). Corra o programa desenvolvido e teste a IU.

Organizando espacialmente a informação

Para organizar a informação em termos de grupos e espaço existe outro mecanismo além dos *grid layouts*: as fronteiras (*borders*). Obtenha o exemplo referente a esta secção e importe o pacote *border*. Vamos criar uma grelha com 2 linhas e 3 colunas de botões que demonstrem as *borders* pré-existentes assim como as que podemos criar.

Comece por obter uma referência à área de conteúdo interno da `JFrame`. Crie uma grelha de 2 linhas por 3 colunas para colocar botões. Deixe um espaço de 5 pixéis na horizontal e na vertical.

Para o primeiro botão, crie um `JButton` e coloque a sua fronteira a cinzento com uma linha `LineBorder`. Adicione-lhe uma etiqueta "Um" e adicione o botão à interface. Para o segundo botão, crie um JButton com a border `EtchedBorder` e a etiqueta "Dois".

Para o terceiro botão crie um JButton com a fronteira vazia e dois *pixéis* de espaço à volta. A etiqueta deste botão é "Três".

Para o quarto, use uma `TitledBorder` para mostrar o título "Clica-me" num tipo de letra 10 pontos, itálico centrado no topo de uma `LineBorder` preta e adicione a etiqueta "Quatro".

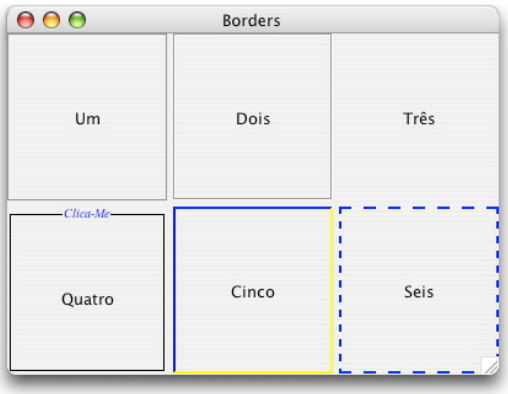
Antes de criar o quinto botão, crie uma subclasse de `JButton` que irá mostrar uma fronteira diferente quando o botão é seleccionado (clicado). Chame essa classe `DoubleBorderedButton` e faça o seu construtor aceitar dois objectos `Border` como parâmetros.

Adicione um `MouseListener` e faça-o alternar entre os dois tipos de *border* quando o rato for clicado.

Para o quinto botão, crie um `DoubleBorderedButton`. Faça uma das suas *borders* ficar com relevo (`BevelBorder`) com uma cor de destaque azul e sombra amarela. Para a outra faça o mesmo mas com o relevo ao contrário. A etiqueta deste botão deve ser "Azul".

O último botão usa uma border diferente, `DashedBorder`. Esta mostra uma linha a tracejado em vez da sólida. Antes de usá-la, porém, é preciso completar a definição. A interface Border tem três métodos, `paintBorder()`, `isBorderOpaque()` e `getBorderInsets()`. Os métodos `paintBorder()` e `isBorderOpaque()` já estão definidos. Defina o método `getBorderInsets()` para retornar um objecto `Insets` com `THICKNESS` à volta. `THICKNESS` é uma constante pré-definida na classe `DashedBorder`.

Compile, corra e teste. O resultado deverá ser algo semelhante a esta figura:



Uma aplicação completa: editor de texto

Nesta secção vamos construir um editor de texto muito simples, mas com uma barra de menus, barra de ferramentas e *tooltips*, como nas aplicações completas. O editor precisa de menus para criar um novo texto, abrir um texto já existente, gravar um texto, fechar a janela e mostrar uma caixa "Acerca de...". A barra de ferramentas precisa de botões para "Novo", "Abrir", "Guardar" e "Acerca de" e mostra também *tooltips*.

Obtenha o código esqueleto para este exemplo. Este código já contém as seguintes funcionalidades:

```
doNewCommand()
doOpenCommand()
doSaveCommand()
doAboutCommand()
doCloseCommand(statusCode)
```

À excepção do `doAboutCommand()`, será necessário fazer as ligações necessárias ao tratamento de eventos. Para o comando `doAbout()` será necessário criar o correspondente `JDialog`.

No código esqueleto comece por criar do topo para baixo, uma `JToolbar`, `JTextArea` e uma `JLabel` (para uma mensagem de estado do sistema). Guarde as referências para a `JTextArea` e `JLabel` no painel de variáveis de instância `pane` e `statusInfo`, respectivamente, já que os métodos de tratamento de eventos necessitam de lhes aceder. Coloque a `JTextArea` dentro de um `JScrollPane` a fim de suportar *scrolling*.

A `JToolbar` não deve ser flutuável para não ser arrastada fora da `frame`. Crie agora uma `JMenuBar` e adicione-a à `JFrame`.

Crie o primeiro menu nessa `JMenuBar`. Crie um `JMenu` chamado "Ficheiro" com os respectivos itens (`JMenuItems`): "Novo", "Abrir...", "Guardar" e "Fechar". Coloque um separador entre "Guardar" e "Fechar". Adicione uma mnemónica de teclado para cada item.

Para cada `JMenuItem`, crie um `ActionListener` anónimo para processar o evento e chamar o método apropriado ao tratamento desse evento (ex^{o}: Novo ... `doNewCommand()` etc.)

No método `doAboutCommand()`, crie um `JDialog` chamado `dialog` que irá mostrar a informação do "Acerca de...". Esta variável tem de ser `final` a fim de poder ser usada no tratamento de eventos interno.

Agora que os menus estão feitos, resta a barra de ferramentas. Vamos criar quatro `JButtons`, chamados "Novo", "Abrir...", "Guardar" e "Acerca de...", e colocá-los na barra. Adicione um separador entre o "Guardar" e "Acerca de...".

Para cada botão, adicione uma *tooltip* com uma mensagem apropriada. Crie ainda um `ActionListener` anónimo para processar o evento e chamar o método necessário ao tratamento do evento. Compile e corra o editor de texto.

Criando interfaces Swing usando o padrão Modelo-Vista-Controlador

O padrão Modelo-Vista-Controlador serve para tornar a apresentação independente dos dados e das operações sobre esses dados. Tornou-se nos últimos anos um padrão muito utilizado e os próprios programadores do Swing o aplicaram ao desenhar as classes que vimos na secção anterior. A robustez deste padrão permite criar interfaces gráficas com o utilizador que são extremamente flexíveis, requerendo um esforço mínimo para alterá-las, mesmo que já estejam colocados todos os componentes no programa.

A arquitectura MVC é composta por:

- Modelo: o modelo representa os dados do domínio da aplicação. Serve como uma aproximação via software ao processos e tarefas da vida real. Ex.^{o}: num sistema de reservas de hotel, o modelo contém os dados sobre clientes, quartos, reservas.

- Vista: A vista apresenta o conteúdo de um modelo. É responsabilidade da vista manter a consistência quando o modelo se altera.
- Controlador: o controlador traduz as interacções realizadas na vista em acções a serem realizadas pelo modelo. Em Swing, por exemplo, as interacções podem ser cliques no botão do rato ou selecções de itens num menu. As acções realizadas pelo modelo incluem activar processos de negocio (por exemplo, iniciar uma reserva) ou alterar o estado do modelo. Baseado nas interacções do utilizador e no resultado das acções do modelo, o controlador responde seleccionando uma vista apropriada.

Vistas em árvore

O Swing fornece um componente que permite desenhar vistas em árvore que mostram uma vista hierarquizada de outros componentes: `JTree`. Cada nó da `JTree` implementa a interface `TreeNode`, onde a classe fornecida por defeito é a `DefaultMutableTreeNode`. Nesta secção, vamos criar uma vista hierárquica muito simples contendo algumas das classes da biblioteca Swing. Vamos mostrar a árvore numa caixa com *scroll* para o caso da árvore ficar maior do que a *frame* que a contém. Quando o utilizador seleccionar um nó, devemos mostrar tanto a localização específica desse nó como o caminho para lá chegar.

Obtenha o código esqueleto para este exemplo e comece por importar o pacote Swing `tree`. De seguida, crie um conjunto de objectos `DefaultMutableTreeNode` no construtor do `TreeExample`, um para cada nó na hierarquia de classes.

```
Component
        Container
                Box
                JComponent
                        AbstractButton
                                JButton
                                JMenuItem
                                JToggleButton
                        JLabel
                ...  [literalmente "..."]
```

Use construções do tipo `DefaultMutableTreeNode component = new DefaultMutableTreeNode("Component");`

Uma vez criados todos os nós, adicione-os a cada um para construir a hierarquia. `DefaultMutableTreeNode` inclui um método `add` que define os filhos de um nó.

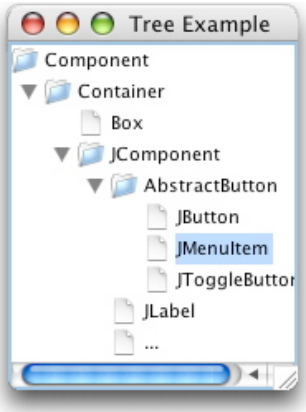
Uma vez criada a hierarquia, crie uma classe que implemente `TreeSelectionListener`. Faça-a mostrar a etiqueta da célula seleccionada e o caminho completo até ela.

Depois de criar a classe `TreeSelectionListener`, instancie-a. Crie uma classe `TreePanel` e passe-a à raiz do componente da hierarquia e ao `TreeSelectionListener`. Adicione o painel à frame. Vamos completar a descrição do `TreePanel` depois.

A classe `TreePanel` irá oferecer a representação visual da árvore. Modifique o construtor do `TreePanel` para que este crie uma `JTree` chamada `tree` ligada à `root` passada no construtor.

Modifique a propriedade `lineStyle` da árvore para `Angled`. Associe o `TreeSelectionListener` à `JTree` se este for não-nulo.

Crie então um objecto `JScrollPane` e coloque a árvore nesse objecto. Compile, corra o programa e teste a IU.



Exercícios

1.1. Escreva um programa que permite ao utilizador escolher uma (e apenas uma) de entre as seguintes três opções: Dolby B, Dolby C e Dolby S. O programa deve também mostrar uma legenda com a opção seleccionada.

1.2. Escreva um programa que apresente três botões: Desporto, Música e Arte. O programa deve permitir ao utilizador seleccionar os botões apropriados através do rato e também através do teclado via uma mnemónica onde uma letra sublinhada é utilizada para a selecção.

1.3. Crie uma simples interface de login/password em Swing. Não se preocupe em validar ou verificar os campos de texto, concentre-se apenas na interface.

1.4. Crie uma interface que permita ao utilizador seleccionar uma estação de rádio de entre as seguintes:

- Rádio Clube Inglês
- Rádio Especial
- Rádio Renascimento
- Rádio 4

(a) a partir da barra de menus da aplicação Swing;

(b) a partir de um menu pop-up;

(c) a partir de uma toolbar;

1.5. A Loja do Cidadão contratou-o para desenhar em Java Swing uma interface de introdução de dados do cidadão, para ser usado pelos funcionários. A interface deve permitir a introdução de:

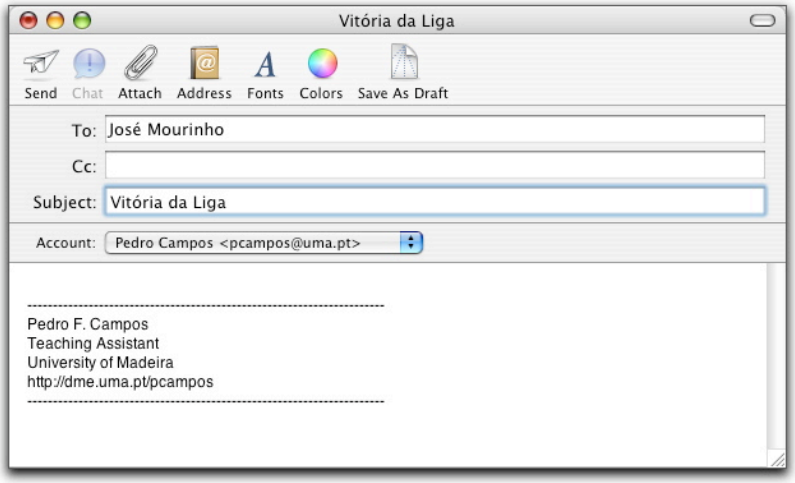
- Primeiro e Último Nome;
- Morada, que inclui Rua, Número de Porta, Código Postal e Cidade;
- Tipo de requisição (há três tipos: B.I., Passaporte e Carta de Marinheiro).

1.6. Redesenhe a interface anterior usando *containers* para dividir a informação no espaço de interacção.

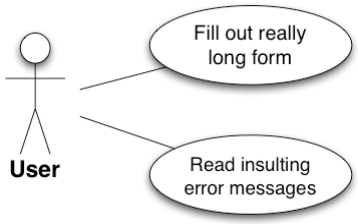
1.7. Crie uma interface simples que apresente:

- Três botões são dispostos horizontalmente e etiquetados como H1, H2 e H3;
- Três botões são dispostos verticalmente e etiquetados como V1, V2 e V3;
- Os botões horizontais devem apresentar uma *border* vermelha à volta;
- Os botões verticais devem apresentar uma *border* azul à volta;
- Os botões verticais devem ser centrados e alinhados com o botão H2 do painel horizontal.

1.8. Crie uma interface que permite ao utilizador compor uma mensagem de correio electrónico semelhante à da seguinte figura:



2
Casos de Utilização Essenciais



In "The Interaction Designer's Coffee Break", www.guuui.com

"Most of us do many things quite well without having much of a clue as to precisely how we do them." – L. Constantine, *Software For Use.*

Motivação

Como são desenhadas as Interfaces com o Utilizador (IU) hoje em dia? As **boas** abordagens são aquelas onde se ganha uma correcta percepção das tarefas do utilizador e do papel que este assume perante o sistema. São abordagens iterativas, em que se constroem e revêem modelos e protótipos da IU.

As abordagens **aceitáveis** (e infelizmente mais frequentes) consistem em implementar um protótipo da IU recorrendo aos populares Interface Builders, focando-se na disposição dos elementos, mas imitando IU's familiares ou bem sucedidas e seguindo as convenções (as famosas *Guidelines*).

As abordagens realmente **más** focam-se na implementação e nos aspectos técnicos do sistema, deixando a construção da IU para último plano. Normalmente os designers que seguem estas abordagens assumem que são eles próprios os utilizadores do sistema. Portanto, se eles se sentem familiares e confiantes, também os utilizadores finais do produto se sentirão.

Casos de Utilização Essenciais de Constantine

No contexto do desenvolvimento de software, e em particular no desenvolvimento da Interface com o Utilizador (IU), perguntar meramente aos utilizadores que tarefas eles realizam e como as realizam não é suficiente para nos permitir desenhar bons programas. Eles podem responder-nos, com toda a honestidade, o que eles acreditam que fazem no seu quotidiano laboral, mas se os espreitarmos a trabalhar poderemos concluir que os vemos a fazer coisas diferentes.

Aprender a desenhar bons programas requer por isso um trabalho conjunto entre analistas e utilizadores, a fim de evoluírem ambos para uma compreensão precisa do trabalho que deverá ser suportado pelo produto a realizar. É esse o objectivo dos

modelos de tarefas que representam a estrutura das necessidades dos utilizadores: a arquitectura do uso do sistema (Constantine, 2003).

Existem muitas formas de modelar o uso de um sistema. Desde diagramas de fluxo (que descrevem as tarefas em termos de eventos sequenciais) até cenários (que representam narrativas de uma ou mais actividades).

Nos casos de utilização estruturados, a narrativa do cenário de utilização do sistema é dividida em duas secções: o modelo das acções do utilizador, que mostra o que o utilizador faz, e o modelo de resposta do sistema, que mostra como o sistema reage ao utilizador.

A Tabela seguinte mostra um exemplo clássico de um caso de utilização estruturado para a tarefa de obter dinheiro a partir de uma caixa multibanco:

Obter dinheiro **Acções do Utilizador**	**Resposta do Sistema**
Inserir cartão	
	Ler fita magnética Pedir PIN
Inserir PIN	
	Verificar PIN Mostrar menu de opções de transacção
Carregar na tecla	
	Pedir quantia a levantar
Inserir quantia	
	Mostrar quantia
Carregar na tecla	
	Devolver cartão
Remover cartão	
	Fornecer dinheiro
Remover dinheiro	

Os casos de utilização estruturados tipicamente contêm muitas pré-definições (quer implícitas quer explícitas) sobre a forma da IU que ainda está para ser desenhada. Tendem por isso a ficar muito próximos dos detalhes de implementação do sistema, e se nos basearmos num caso de utilização muito concreto que contenha decisões de desenho implícitas, corremos o risco de perder muitas hipóteses de obter um desenho bom, focado nos utilizadores e não no computador, linguagem de programação, etc.

Isto levou à invenção dos casos de utilização essenciais, que são definidos da seguinte maneira:

"Um caso de utilização essencial é uma narrativa estruturada, expressa na linguagem do domínio da aplicação e dos utilizadores, composta por uma descrição simplificada, generalista, abstracta e independente da tecnologia da tarefa ou interacção que é completa, com significado e bem definida do ponto de vista dos utilizadores, cujos papéis personificam o propósito ou intenções subjacentes à interacção".

Um caso de utilização essencial é estruturado em três partes: uma frase que descreve a intenção expressa do utilizador, e uma narrativa dividida em duas partes: o modelo da intenção do utilizador e o modelo da responsabilidade do sistema.

A diferença entre um caso de utilização essencial e um caso de uso estruturado pode parecer subtil, mas as consequências não. Constantine descreve o seguinte exemplo, que consiste em reescrever o caso anterior na forma essencial, focando-se nas intenções e não nas acções, e também simplificando e generalizando a partir do caso concreto.

Começamos por nos interrogar sobre o porquê de um utilizador inserir um valioso cartão numa caixa multibanco. À primeira vista, pareceria uma acção ridícula, sobretudo tendo em conta que uma máquina pode falhar e ficar com o cartão, nunca o devolvendo. No entanto, milhões de utilizadores cumprem esta tarefa todos os dias. Portanto, a questão que pomos é: "Para quê?" Para que ele (o utilizador) se identifique perante o sistema multibanco. Eles fazem isto porque não querem que mais ninguém retire dinheiro das suas contas.

O caso essencial é reescrito desta forma:

Obter dinheiro **Intenções do Utilizador**	 **Responsabilidades do Sistema**
Identificar-se	
	Verificar identidade
	Oferecer escolhas
Escolher	
	Fornecer dinheiro
Levar dinheiro	

Este caso de utilização essencial é dramaticamente mais curto e simples que o concreto (para a mesma interacção) porque inclui *apenas os passos que são essenciais e de interesse genuíno ao utilizador*.

Visto ser orientado ao problema, e não à solução, deixa propositadamente em aberto muitas possibilidades em aberto em relação ao desenho e implementação do sistema. Por exemplo, além de cartões de fita magnética, existem muitas outras maneiras de identificar um utilizador perante um sistema. A caixa multibanco pode examinar a retina ou a impressão digital para descobrir quem está a utilizar o sistema. Outra possibilidade que é deixada em aberto pelo caso essencial é oferecer as escolhas através da voz ou confirmá-las pelo processamento da resposta vocal.

Introdução ao método Wisdom

O método Wisdom (*Whitewater Interactive System Development with Object Models*) foi desenvolvido para ir de encontro às necessidades de desenvolvimento das pequenas equipas de software que precisam de desenvolver e manter sistemas interactivos seguindo um processo de qualidade (Nunes, 2000b).

O Wisdom é uma nova proposta de um método de desenvolvimento de software simplificado que compreende três aspectos importantes:

(i) o **processo** Wisdom, que define uma alternativa ao Processo Unificado especialmente adaptada ao desenvolvimento de software por pequenos grupos de pessoas de uma forma rápida e assente num modelo controlado de prototipificação evolutiva;

(ii) o **modelo** de desenvolvimento subjacente ao processo Wisdom, que assenta numa nova arquitectura de modelos UML que suporta o desenvolvimento centrado nos utilizadores e o desenho de interfaces com o utilizador;

(iii) a **arquitectura** Wisdom introduz novos modelos UML que suportam a modelação de perfis de utilizadores, da interacção, do diálogo homem-máquina, e das questões de apresentação;

(iv) a **notação** Wisdom é uma contextualização e extensão do UML que define um novo conjunto de elementos de modelação que suportam o processo e a arquitectura Wisdom.

Como o Wisdom é um método focado nos sistemas interactivos, utiliza a interpretação essencial dos casos de utilização de Constantine, que vimos na secção anterior. O Wisdom é guiado pelos casos de utilização, os quais também servem para encontrar e especificar classes de análise, fluxos de tarefas, espaços de interacção e mesmo requisitos não-funcionais (usando notas/comentários).

Os fluxos de tarefas são fulcrais na abordagem Wisdom. Um fluxo de tarefas corresponde a uma descrição das intenções do utilizador e responsabilidades do sistema durante o decorrer de uma tarefa específica. Essa descrição é feita de uma forma independente da tecnologia.

Na solução proposta no Wisdom, promove-se uma representação diagramática que facilita simultaneamente a integração com materiais utilizados em sessões participadas (à esquerda) e diagramas de actividade UML que são fáceis de manter, relacionar com outros elementos de modelação e expandir à medida que o conhecimento do problema aumenta. Desta forma, o Wisdom combina as narrativas essenciais propostas por Constantine com técnicas participadas.

Esta representação, consistente com a norma da UML, promove a recolha de requisitos através de técnicas participadas uma vez que permite uma melhor manipulação do que as narrativas em linguagem natural propostas por Constantine. Os fluxos de tarefas, construídos inicialmente em sessões participadas através de materiais "low-tech" (cartões adesivos, stickers, etc.), são depois traduzidos em diagramas UML fáceis de manter, estender, relacionar e documentar pela equipa de desenvolvimento.

A ideia é *garantir que a equipa de desenvolvimento produza uma interface de utilizador que reflicta a estrutura do uso do produto e não a estrutura interna da aplicação.*

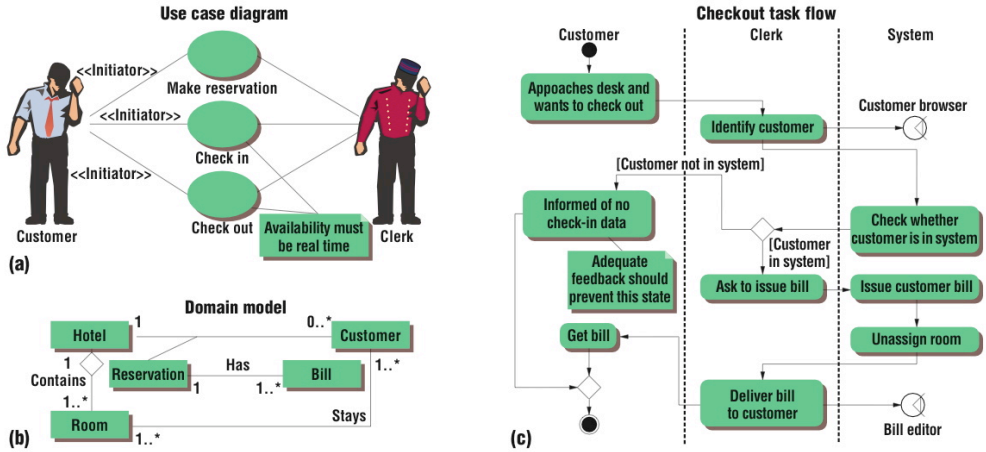


Figura 2: Exemplo de modelação Wisdom para um sistema de reservas de hotel: (a) modelo de casos de utilização essenciais para dois utilizadores distintos; (b) modelo do domínio e (c) fluxo de tarefas para o caso de uso "check-out". [Retirado de (Nunes, 2000a)].

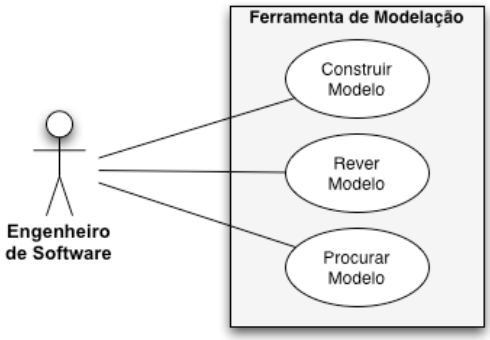
A Figura 2 ilustra a modelação de casos de utilização essenciais e fluxo de tarefas para um exemplo clássico: um sistema de reservas de hotel:

> *The guest makes a reservation with the Hotel. The Hotel will take as many reservations as it has rooms available. When a guest arrives, he or she is processed by the registration clerk.*
>
> *The clerk will check the details provided by the guest with those that are already recorded. Sometimes guests do not make a reservation before they arrive.*
>
> *Some guests want to stay in non-smoking rooms. When a guest leaves the Hotel, he or she is again processed by the registration clerk. The clerk checks the details of the staying and prints a bill. The guest pays the bill, leaves the Hotel and the room becomes unoccupied.*

O diagrama de casos de utilização essenciais (a) exemplifica um requisito não funcional ("*Availability must be real time*") . O modelo de domínio (b) representa os objectos mais importantes no contexto do sistema. O fluxo de tarefas (c) é modelado em UML por um diagrama de actividade que no exemplo da Figura refere-se ao caso de utilização da tarefa de "*Checkout*".

Exercícios

2.1. Suponha que está a construir uma ferramenta de modelação (UML por exemplo) e que após algumas iterações você e a sua equipa chegaram ao esquema de casos de utilização descrito na figura seguinte:



A fim de refinar o caso de utilização "Procurar Modelo", especifique o caso de utilização essencial seguindo a norma de Constantine. De seguida especifique (para o mesmo caso) o diagrama de fluxo de tarefas de acordo com o método Wisdom.

2.2. Considere um caso de utilização "consultar saldo bancário". Especifique o caso de utilização essencial e o diagrama de fluxo de tarefa. Repita o mesmo exercício para o caso de utilização "transferir dinheiro" (entre contas bancárias) e "consultar transacções do cartão"

3
Modelação da Arquitectura Wisdom

É na fase de análise que são refinados e estruturados os requisitos descritos na arquitectura do uso do sistema. As principais actividades de análise são identificar e estruturar classes de análise e classes específicas ao método Wisdom (espaços de interacção e tarefas).

As classes de análise representam abstracções de conceitos capturados na fase de requisitos. São focadas nos requisitos funcionais, deixando os requisitos não-funcionais para a fase de desenho. Seguindo a norma UML, as classes de análise são divididas em três tipos: *entity* (entidade, informação passiva), *control* (controlo, lógica de negócio complexa) e *boundary* (fronteira, comunicação com o exterior).

O método Wisdom estende a modelo conceptual de análise do UML introduzindo duas novas classes específicas à interface com o utilizador. Espaços de interacção (*interaction spaces*) são uma classe que modela a interacção entre o sistema e os actores. São responsáveis por receber e apresentar informação de/para os utilizadores. As classes *interaction spaces* têm acções (*actions*) em vez de operações e atributos estereotipados com input e output elements que modelam, respectivamente, informação recebida pelo utilizador e informação apresentada ao utilizador.

A outra classe é a das Tarefas (*tasks*). As classes tarefa são responsáveis pelo sequenciamento das tarefas e pela consistência das entidades de apresentação (espaços de interacção). As classes tarefa encapsulam as dependências temporais complexas, e outras restrições, entre actividades, isolando as alterações na estrutura de diálogo;

A Figura 3 mostra as notações para o modelo de análise e de interacção.

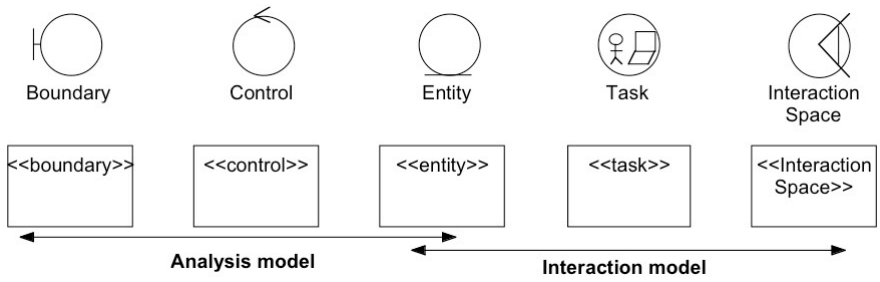


Figura 3: Notações alternativas para os estereótipos de classes dos modelos de análise e interacção.

A Figura 4 ilustra um exemplo de uma arquitectura Wisdom para o mesmo sistema de reservas de hotel do capítulo anterior, com as classes de análise à esquerda e as classes de interacção à direita.

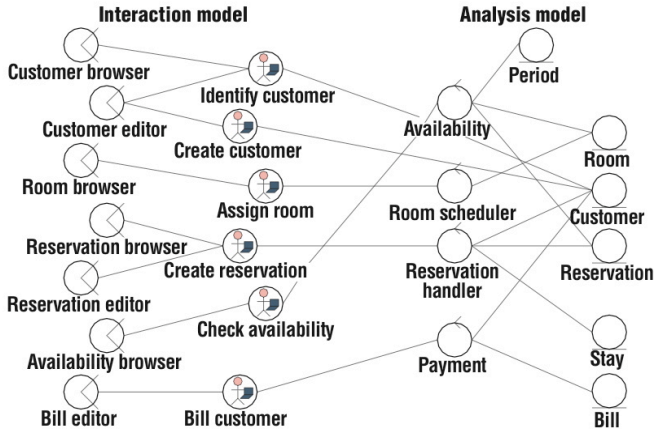
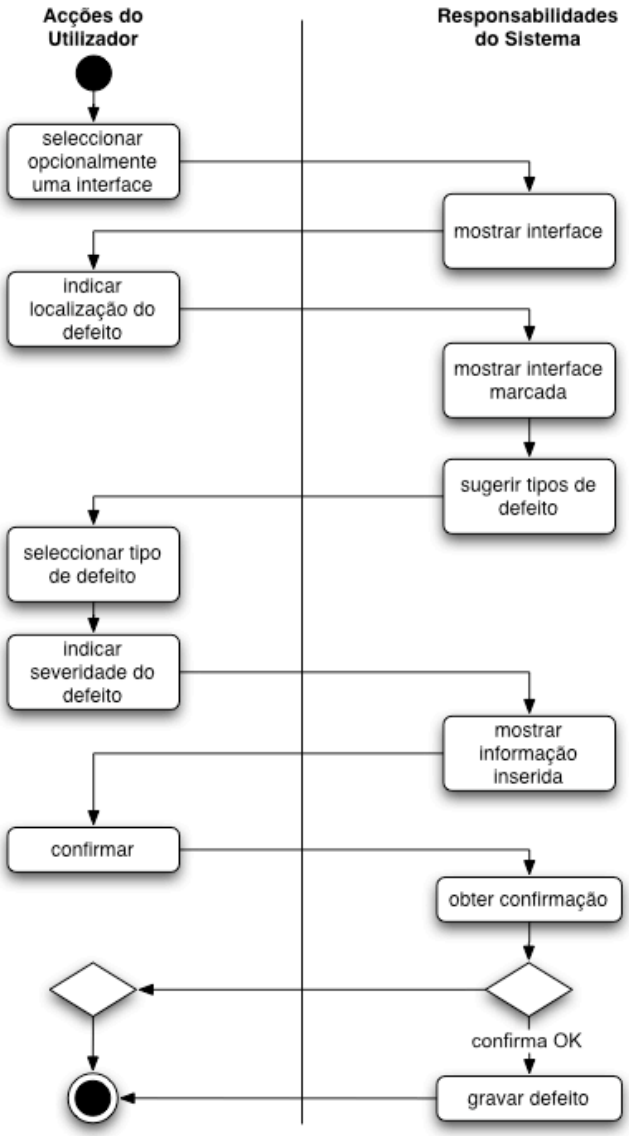


Figura 4: Exemplo da arquitectura Wisdom para o modelo de Interacção e de Análise, para um sistema de reservas de hotel.

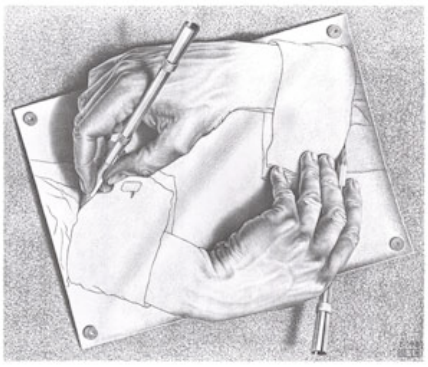
Exercícios

3.1. Considere que está a modelar uma sistema para detecção e gestão de informação sobre defeitos de usabilidade de programas. Para o seguinte diagrama de tarefas relativo ao caso de utilização "Gravar Defeito de Usabilidade", apresente uma arquitectura para o modelo de análise e de interacção seguindo o método Wisdom:



3.2. Para os diagramas de fluxo de tarefas relativos ao exercício 2.2 (casos de utilização "consultar saldo bancário", "transferir dinheiro" (entre contas bancárias) e "consultar transacções do cartão", especifique uma arquitectura conceptual Wisdom.

4
Modelação do desenho de Apresentação Wisdom



A notação Wisdom (Nunes, 2000a) consiste num conjunto de notações compatíveis com o UML, que suportam a modelação eficaz e eficiente de sistemas interactivos.

Para suportar a modelação dos aspectos de apresentação da IU numa fase de desenho, o método Wisdom propõe as seguintes extensões ao UML (Nunes, 2000a):

- «Interaction Space», um estereótipo de **classe** que representa o espaço contido na IU onde o utilizador interage com todas as ferramentas e materiais no decorrer de uma tarefa ou conjunto de tarefas inter-relacionadas;
- «navigates», um estereótipo de **associação** entre dois espaços de interacção que denota uma mudança de espaço de interacção por parte do utilizador;
- «contains», um estereótipo de **associação** entre duas classes espaços de interacção que denota que a classe fonte (o contentor) contém a classe alvo (o conteúdo); Este estereótipo de associação só pode ser utilizado entre dois espaços de interacção e é unidireccional;
- «input element», um estereótipo de **atributo** que denota informação recebida pelo utilizador, isto é, informação sobre a qual o utilizador pode operar (note-se a semelhança deste conceito com os híbridos da notação canónica);
- «output element», um estereótipo de **atributo** que denota informação apresentada ao utilizador, ou seja, informação que o utilizador recebe mas não manipula;
- «action», um estereótipo de **operação** que denota algo que o utilizador pode realizar na IU e que causa uma alteração significativa no estado interno do sistema.

A Figura 5 mostra a criação de interfaces concretas via modelo de apresentação Wisdom, para o mesmo problema que temos vindo a considerar nos capítulos anteriores.

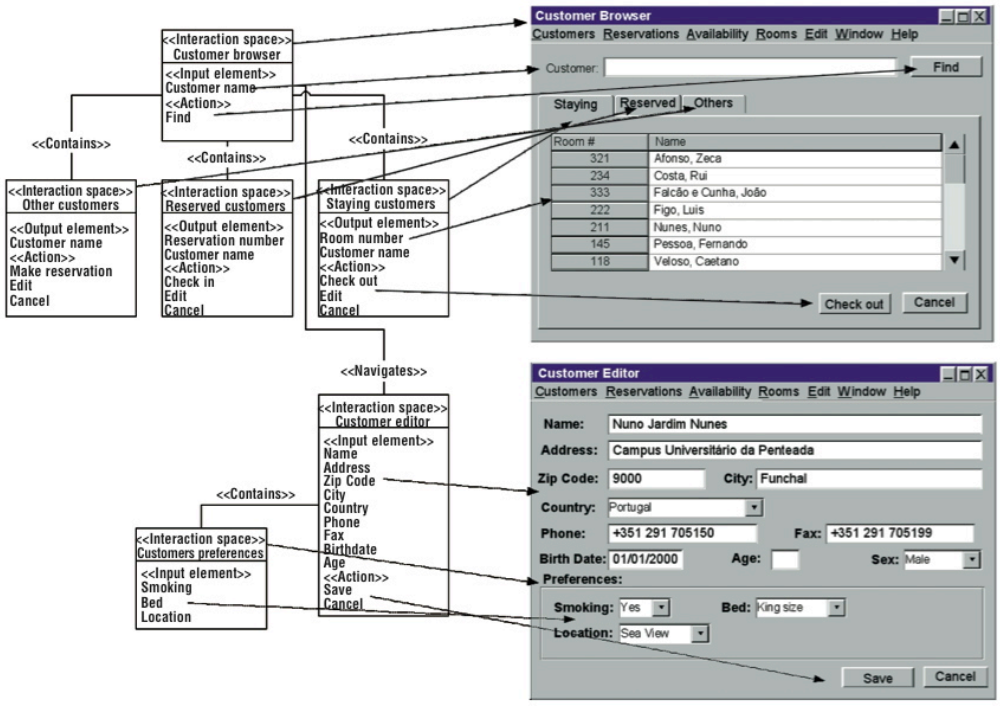
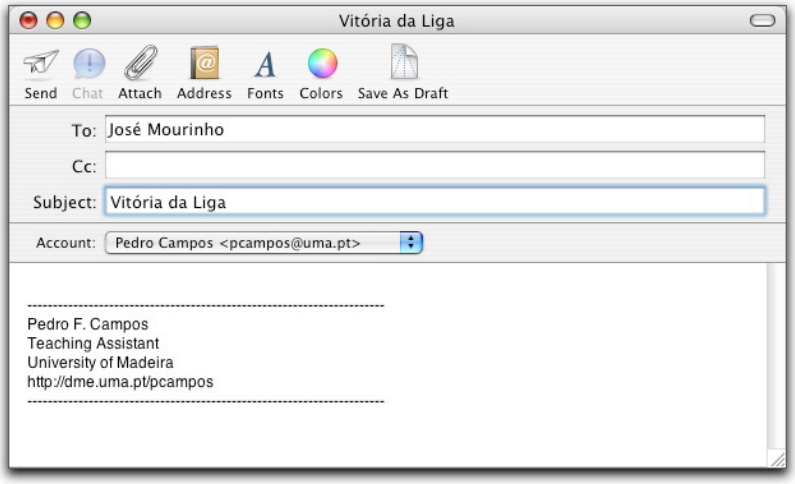


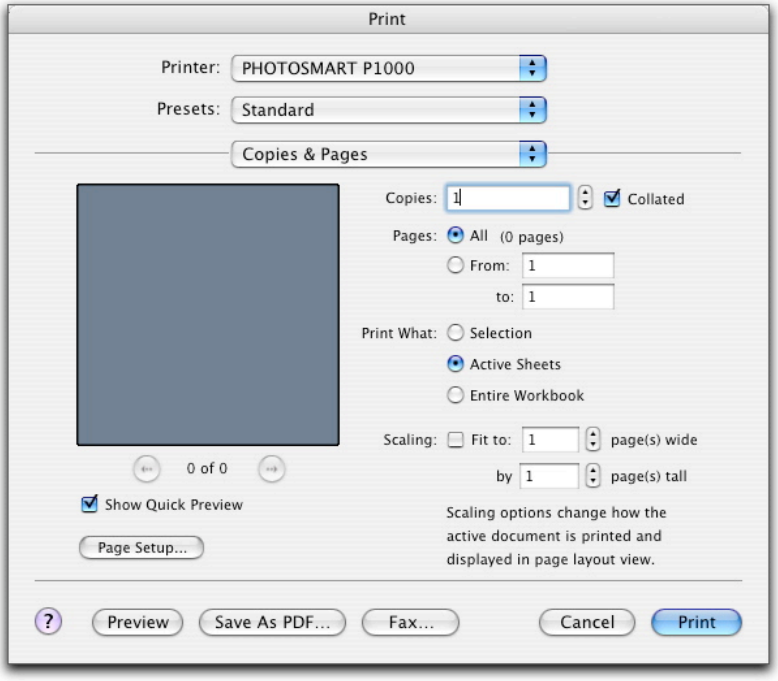
Figura 5: Transformação do modelo de apresentação Wisdom do sistema de reservas de hotel numa interface concreta.

Exercícios

4.1. (Exame Época Especial 2003/2004). A Figura seguinte representa uma interface que permite ao utilizador compor uma mensagem de correio electrónico. Apresente um modelo de apresentação para esta interface utilizando as técnicas do método Wisdom. Comece por reduzir o modelo aos espaços de interacção mais genéricos e depois detalhe cada um dos espaços de interacção em termos dos seus elementos de entrada, saída e acções.



4.1. A Figura seguinte representa outra interface bem conhecida que permite ao utilizador imprimir uma folha de cálculo. Apresente um modelo de apresentação para esta interface. Comece por reduzir o modelo aos espaços de interacção mais genéricos e depois detalhe cada um dos espaços de interacção em termos dos seus elementos de entrada, saída e acções.



4.3. Apresente um modelo de apresentação Wisdom para o diagrama de fluxo de tarefas relativo ao caso de utilização essencial "Procurar Modelo" do exercício 2.1. Esboce uma possível concretização desse modelo para um sistema Windows.

5
Modelação de Protótipos Abstractos Canónicos

Formas diferentes de retratar janelas: à esquerda o abstracto "Fenêtres" de R. Delaunay. À direita o concreto "Venetian Windows" de M. Flemingham.

A prototipagem rápida de sistemas interactivos é utilizada com o objectivo de validar e avaliar ideias de desenho numa fase inicial do processo de desenvolvimento. Tenta-se, portanto, promover a colaboração entre todas as pessoas envolvidas no projecto (desde os gestores, designers e programadores até aos utilizadores finais), assim como facilitar os ciclos iterativos de revisão e teste.

Neste capítulo, descreve-se alguns métodos para o desenvolvimento rápido de protótipos de IU e a notação base que iremos estudar, a notação Abstracta Canónica: uma nova linguagem de especificação de protótipos abstractos que assenta numa notação prática para definir e comparar ideias de interacção visuais (Constantine, 2003).

Técnicas para prototipagem rápida de Interfaces

Como vimos no capítulo anterior, a linguagem de modelação UML tornou-se uma alternativa muito popular para a estruturação dos elementos de apresentação de sistemas interactivos (Nunes, 2001). Em particular, a notação Wisdom segue a norma do UML, o que promove e facilita a comunicação com os programadores e designers de software. Esta é uma boa estratégia para se conseguir alcançar uma maior taxa de aceitação das ferramentas de modelação.

Outra estratégia, utilizada pela abordagem DiaMODL, combina este aspecto com uma ligação mais forte ao elementos concretos da IU (Trætteberg, 2003), esperando-se com isso favorecer o processo de teste, uma vez que o utilizador lida com um écran concreto.

A prototipagem de interfaces através de ferramentas de esboço electrónicas também obteve algum sucesso em sistemas como o SILK (Landay 2001) ou o DENIM (Newman, 2003). O esboço é importante durante o processo de prototipagem inicial porque incentiva a criatividade do designer: a ambiguidade inerente à dimensão ou tipo dos esboços encoraja a exploração de novos desenhos sem que o designer se perca em detalhes. Isto obriga a que o designer se concentre nos assuntos mais importantes desta etapa, como a estrutura geral e o fluxo da interacção (Landay, 2001).

Contudo, a tarefa de reconhecimento dos componentes concretos a partir do esboço não é fácil, uma vez que qualquer algoritmo de reconhecimento de padrões é susceptível de errar. Testes de usabilidade a estas ferramentas revelaram que alguns utilizadores sentiram dificuldades em manipular e introduzir elementos textuais, assim como em compreender a forma de seleccionar, agrupar e mover objectos.

A notação Abstracta Canónica

Protótipos Abstractos Canónicos (PAC) constituem uma nova linguagem de especificação de IU, proposta por Constantine. Esta linguagem é composta por uma colecção estável de componentes de interface abstractos, desenvolvida a partir de muitas iterações em projectos da vida real. Estes componentes podem ser seleccionados a partir de uma "palette" a fim de construir protótipos abstractos canónicos. A existência de um conjunto normalizado de componentes abstractos facilita a comparação de alternativas de desenho, promovendo simultaneamente a comunicação entre os membros da equipa de desenvolvimento (Constantine, 2003).

A notação simbólica inerente aos PAC é definida a partir de dois símbolos universais, genéricos e extensíveis: um material ou contentor genérico, representado por um quadrado e uma ferramenta ou acção genérica, representada por uma seta. Os materiais representam conteúdo, informação, dados ou quaisquer outros objectos da IU manipulados ou apresentados ao utilizador no decorrer de uma tarefa. As ferramentas representam operadores, mecanismos ou controladores que podem ser usados para manipular ou transformar os materiais.

Através da combinação destas duas classes de componentes é possível gerar uma terceira classe de componentes genéricos, designados por híbridos ou materiais activos. Estes representam qualquer componente com características de ambos os componentes material e ferramenta, como por exemplo um campo para entrada de texto (um material manipulável que consistem num elemento de IU que apresenta informação que pode ser editada ou introduzida).

Figura 6: Os três símbolos básicos sobre os quais assenta a notação abstracta canónica (da esquerda para a direita): ferramenta abstracta genérica, material abstracto genérico e material activo (ou híbrido) genérico.

A Figura 1 mostra os três símbolos básicos da notação Abstracta Canónica. A Figura contém a notação em mais detalhe. A notação completa e detalhada encontra-se descrita e exemplificada nas figuras seguintes.

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	action/operation*	Print symbol table, Color selected shape
	start/go/to	Begin consistency check, Confirm purchase
	stop/end/complete	Finish inspection session, Interrupt test
	select	Group member picker, Object selector
	create	New customer, Blank slide
	delete, erase	Break connection line, Clear form
	modify	Change shipping address, Edit client details
	move	Put into address list, Move up/down
	duplicate	Copy address, Duplicate slide
	perform (& return)	Object formatting, Set print layout
	toggle	Bold on/off, Encrypted mode
	view	Show file details, Switch to summary

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	container*	Configuration holder, Employee history
	element	Customer ID, Product thumbnail image
	collection	Personal addresses, Electrical Components
	notification	Email delivery failure, Controller status

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	active material*	Expandable thumbnail, Resizable chart
	input/accepter	Accept search terms, User name entry
	editable element	Patient name, Next appointment date
	editable collection	Patient details, Text object properties
	selectable collection	Performance choices, Font selection
	selectable action set	Go to page, Zoom scale selection
	selectable view set	Choose patient document, Set display mode

Apesar de os PAC's não possuírem um formalismo preciso nem a semântica necessária a fornecer ferramentas de suporte ou geração automática de IU, verificamos que a notação é suficientemente expressiva para gerar Interfaces concretas a partir da especificação abstracta canónica.

Desenho de Protótipos Abstractos Canónicos

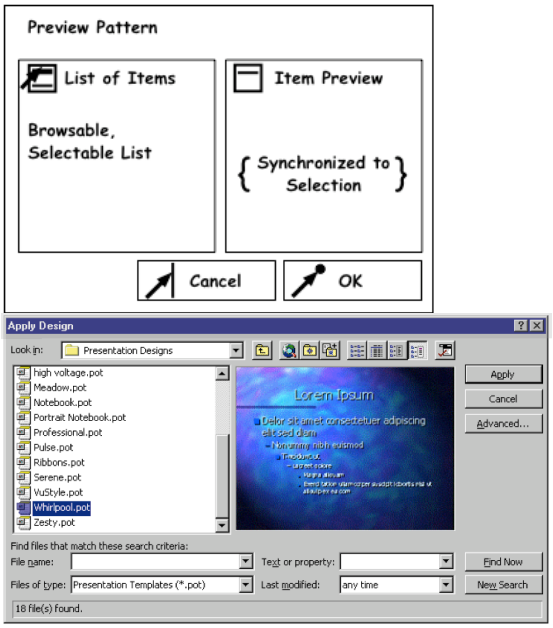


Figura 7: À esquerda, uma representação PAC de uma interface de pré-visualização. À direita, um exemplo concreto (diálogo do Microsoft PowerPoint).

Na Figura 7, apresentamos as representações Wisdom e Canónica para o padrão de IU identificado e classificado como "Pré-visualização" (Welie, 2000). Também apresentamos um exemplo concreto deste padrão (neste caso, um diálogo do Microsoft PowerPoint). O problema ocorre quando o utilizador procura por um item numa pequeno conjunto de itens e tenta encontrar o item desejado navegando nesse conjunto. Este padrão é particularmente útil quando a natureza do conteúdo do item não é a mesma que a natureza do seu índice de procura (por exemplo, um conjunto de imagens ou clips áudio indexadas pelo nome). A solução consiste em fornecer ao utilizador uma pré-visualização do item que estiver seleccionado num determinado instante (Welie, 2000).

Exercícios

5.1. Modele em PAC o padrão de "Navegação em Contentores". Quando o utilizador necessita de encontrar um item numa colecção de contentores, a solução deste padrão consiste em dividir o espaço de interacção em três partes: uma para visualizar a colecção de todos os contentores, outra para visualizar um determinado contentor e outra para visualizar um determinado item desse contentor. A mostra um exemplo concreto deste padrão: o visualizador de Mail/News do Netscape Navigator, assim como a representação de desenho Wisdom. Discuta as vantagens de uma notação em relação à outra.

Nota: na sua solução, tenha em conta que o utilizador selecciona, em primeiro lugar, um determinado contentor, depois um item contido nesse contentor e só então visualiza o item seleccionado. (**Sugestão**: lembre-se da forma ocidental de leitura/escrita).

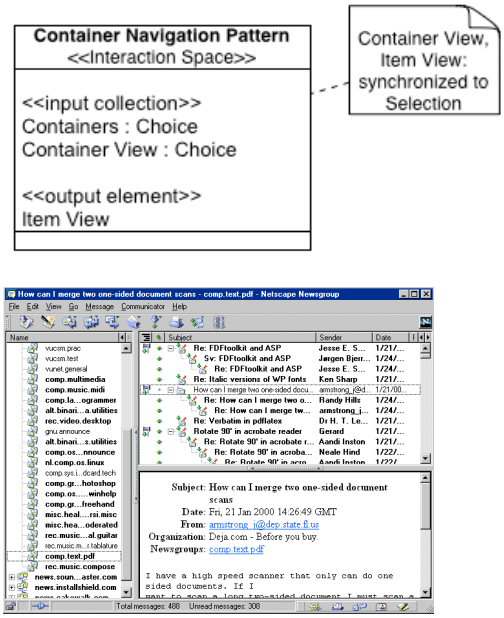


Figura 8: O padrão de "Navegação em Contentores".

5.2. Escolha um padrão de interface de entre os disponíveis na colecção Amsterdam (`www.welie.com/patterns`). Modele esse padrão em Wisdom e PAC. Discuta as diferenças de cada solução.

5.3. Considere o protótipo abstracto representado na figura seguinte. Apresente uma representação concreta (em esboço ou em Java Swing) para este protótipo. Lembre-se que a solução deve ser criativa e suportar a tarefa: mostrar uma lista de itens dando ao mesmo tempo a possibilidade de reordená-los. A primeira interface que desenhar provavelmente irá ter problemas. Identifique-os, corrija-os e reitere até chegar a uma solução não-trivial.

5.4. Proponha um protótipo abstracto canónico para o caso de utilização essencial relativo ao "Procurar Modelo" do exercício 2.1.

5.5. Proponha um protótipo abstracto canónico para o caso de utilização "Transferir dinheiro" do exercício 2.2.