IMPLEMENTAÇÃO

Interfaces Pessoa Máquina

2010/11

©2009-11 Salvador Abreu
BASEADO EM MATERIAL ©Alan Dix
Application

- Hardware
- OS
- Window System
- UI Toolkit (Widgets)
- Higher level Tool
Windowing systems

• Allocation and management
  ➡ display areas for each application program
  ➡ Input devices → application

• Is invoked by the application programs to:
  ➡ Create/close windows
  ➡ Manipulate windows (move, resize, hide, ...)
  ➡ Receive the user’s input via interaction devices
Windowing systems

• Sharing resources
  ➡ Simultaneous execution of different tasks
  ➡ Support independent processes (control coordination of concurrent processes)
  ➡ Isolation of individual applications

• Device independence
  ➡ Programmer wants to direct commands to an abstract terminal which understands a more generic language and can be translated to the language of many other specific devices (device driver) -> + portability.
Client-server architecture
Programming paradigms

• “Read-evaluation loop”
  ➡ Internal to the application

```plaintext
repeat
  read-event(myevent)
  case myevent.type
    type_1:
      do type_1 processing
    type_2:
      do type_2 processing
    ...
    type_n:
      do type_n processing
  end case
end repeat
```


### Programming Paradigms

- "Notification" or "Callback" based

```java
void main(String[] args) {
    Menu menu = new Menu();
    menu.setOption("Save");
    menu.setOption("Quit");
    menu.setAction("Save", mySave);
    menu.setAction("Quit", myQuit)
    ...
}

int mySave(Event e) {
    // save the current file
}

int myQuit(Event e) {
    // close down
}
```
Toolkits

- Interaction objects
  - From the user’s perspective, input and output behaviours are intrinsically linked
  - From the programmer’s perspective, input and output are quite separate
  - Ex: Mouse
Toolkits

- Provides a set of ready-made interaction objects or **widgets** (**Window Gadget**) with a predefined behaviour.
- Help to synchronize input and output.
- Promote consistency (similar “look” and “feel”).
- **Ex: Java AWT (Abstract Window Toolkit)**
  - Java classes for buttons, menus, ...
  - Notification based
- **Ex: Java Swing**
  - built on top of AWT – higher level features
  - uses MVC architecture (see later)
• **Buttons (several types)**

• **Scroll bars and sliders**

• **Pulldown menus**
**Hierarchy**

- **Widgets**
  - Hierarchical organization with inheritance

```
Hierarchical organization with inheritance
```

```
Primitive

- Text
- Label
- Button
- Scroll Bar

  - Push Button
  - Drawn Button
  - Toggle Button
```

Thursday, June 2, 2011
Widgets

• Visual appearance

• Attributes

PushButton {
  Color BackGround;
  int MarginLeft;
  int MarginRight;
  int BorderWidth;
  Pixmap ArmPixmap;
  Boolean FillOnArm;
  CallbackList ActivateCallback;
}

• Interactive behaviour
Toolkits provide only a limited range of interaction objects, limiting the kinds of interactive behaviour allowed between user and system.

User Interface Management Systems

- Another level above the toolkit level
  - Toolkits are difficult to use by non-programmers
  - Separation between application semantics and presentation
- Provides assistance and automation in the interface building process.
**Semantic/presentation separation**

- **Application**
  - Operations
  - Data

- **Interface**
  - Components
  - Graphics
  - I/O

- Users interact with the interface.
- User actions should be communicated to the application.
- The application should respond accordingly.
Semantic/presentation Separation

- Separation between application **semantics** and **presentation**
- Improves
  - **Portability** – to allow the same application to be used on different systems.
  - **Reusability** – components reusability > cost reduction.
  - **Multiple interfaces** – several different interfaces can be developed to access the same functionality > + flexibility
  - **Customization** – interface can be customized by both the designer and the user to increase its effectiveness without having to alter the underlying application.
The logical components of a UIMS were identified as:

- **Presentation** – the component responsible for the appearance of the interface, including what input and output is available to the user.

- **Dialog control** – the component which regulates the communication between the presentation and the application.

- **Application interface** – the view of the application semantics that is provided as the interface.
• Does not address how to build large and complex interactive systems from smaller components.
• Interactive applications architecture developed for Smalltalk

  ➡ Used by Java Swing UI widget library

Model – represents the application semantics; application state and behaviour.

View – Manages the graphical and/or textual output of the application (output).

Controller – controls and manages the input (user interaction).
Model - View - Controller

- Separates frontend concerns from backend concerns.
- Separates input from output
- Permits multiple views on the same application data
- Permits views/controllers to be reused for other models

- A single model can be associated with several MVC triads, so that the same piece of application semantics can be represented by different input-output techniques.
- Each view-controller pair is associated to only one model.
Model - View - Controller

- **Example: button**
  - **Model** – a boolean – on or off
  - **View** – a ≠ image for each possible state
  - **Controller** – tells the model to change the state and tells the View to update the output.

- **Application/Presentation Separation**
  - Modifications and maintenance are easier
  - Different look:
    - change View, nothing else
Model - View - Controller

- **Model**
  - Responsible for data
    - Maintains application state
    - Implements state changing behaviour
    - Notifies dependent views/controllers when changes occur

- **View**

- **Controller**
Model - View - Controller

- **Model**
- **View**
  - Responsible for output
    - Occupies screen (position, size)
    - Draws on the screen
    - Listens for changes to the model
- **Controller**
Model - View - Controller

- Model
- View
- Controller

➡ Responsible for input

- Listens for keyboard and mouse events
- Instructs the model or the view to change accordingly
Model - View - Controller

- View and controller are tightly coupled
  - intense communication between them
  - pairs view/controller

- Some architectures combine the two in just one class
  - Model-view
  - PAC (presentation-abstraction-control)