Designing Interactive Systems 2

Lecture 5: macOS

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/dis2
CHAPTER 11

Classic Mac
Mac
Macintosh 128k & Macintosh System 1

• Introduced in 1984

• Based on Xerox PARC Smalltalk, Star, Tajo

• Few technical innovations (QuickDraw)

• But landmark in UI design and consistency policies

• First commercially successful machine

• Price of ~2500$
Macintosh 128k & Macintosh System 1

- Saving hardware for an “affordable” product
  - No hard disk
  - 128k RAM
  - 64k ROM containing the Macintosh Toolbox

- Single process, single address space
  - No OS, the app is in charge
  - No multitasking
Macintosh Toolbox

• **Event Manager**
  Event loop core of any app
  Application polls for new events with a `GetNextEvent()`

• **Control Manager**
  Create, manipulate, redraw buttons, checkboxes, scroll bars, …
  Respond to user actions

• **Dialog Manager**
  Create and manage dialogs and alerts
Macintosh Toolbox

- **Window Manager**
  Creates, moves, updates windows

- **Menu Manager**
  Offers menu bar, pull-down and cascading menus

- **Finder interface**
  Defining icons for applications and documents
  Interacting with the Finder

- Many more
  Scrap Manager, Help Manager, Sound Manager, Memory Manager, …
ResEdit

• Graphical Resource Editor (Apple)

• Overview of resources in resource fork of any file (app or doc), sorted by resource type

• Opening a type shows resources of that type sorted by their ID

• Editors for basic resource types built in (ICON, DLOG, ...)

![ResEdit Image]
Mac System 7
CHAPTER 12

Mac OS X — macOS
Mac OS X Roots: NeXT
Mac OS X
Darwin

• The open-source base operating system of macOS

• Mach kernel
  • Preemptive multitasking
  • Protected memory

• BSD
  • Process model
  • Threading
  • Networking
Cocoa

- The OO API for developing macOS Apps, evolved out of NeXTSTEP

- Three main frameworks
  - Foundation
  - AppKit
  - Core Data

- Programming languages
  - Objective-C
  - Swift
Carbon

- Encapsulates the functionality of the Mac Toolbox in one API
- Runs on top of the native OS X, i.e. not an emulator
- Large parts of Foundation had to be reimplemented in C
- Finally deprecated in 2012
macOS: Architecture

- Cocoa Application
  - Carbon
  - AppKit
  - Java

- Media
  - AVFoundation
  - Core Animation
  - Core Image
  - Quartz

- Core Services
  - Core Data
  - Foundation
  - Security
  - WebKit

- Core OS
  - Accelerate
  - Open CL
  - System Configuration

- Kernel and Device Drivers
  - BSD
  - File System
  - Mach
  - Networking
macOS: Four Layer Model

- Apps
- UITK: AppKit
- WM: Window Server
- BWS: AppKit
- GEL: Quartz, I/O Kit
- Hardware
Event Handling

• Similar to our Reference Model

• Window Server distributes events to queues

• Single queues per process
CHAPTER 13

Cocoa & Objective-C
Cocoa

• **Foundation**
  - Basic programming support
  - NSObject, values, strings, collections, OS services, notifications

• **AppKit**
  - Interface, fonts, graphics, color, documents, printing, OS support, international support, InterfaceBuilder support

• **CoreData**
  - Object-graph management and persistence framework
Objective-C

- Implementation language of the Cocoa framework
- Created in 1983 to combine OO principles with C
- Dynamic typing, binding, and loading
- **Categories** allow to extend classes without subclassing
- **Protocols** as alternative to multiple inheritance
**Objective-C: Syntax**

```objective-c
NSImage *image = [self importImage:@"sheep.png" withScaleFactor:3];
```

- Square brackets make it clear which object receives a message
  - Increases readability
  - Method signature contains names for all parameters
- Prefixes determine the type of a declared method
  - – for instance methods
  - + for class methods
Objective-C: Dynamic Typing

• Objective-C checks whether a method exists at runtime

• You can call known methods of a subclass without casting

• id is the type that matches any Objective-C object

• Example:

  if ([unknownThing isKindOfClass:[NSNumber class]]) {
    NSLog(@"%ld", [unknownThing integerValue]);
  }
Objective-C: Dynamic Binding

- A **Method** is a tuple of a selector (**SEL**) that defines the method signature, the type of the parameters, and an implementation pointer (**IMP**).

- Each object has a method list and executes a method when it receives a known selector message from another object.

- Hence, the invoked method is resolved at runtime.

- You can even change methods and method lists at runtime.
Objective-C: Dynamic Loading

• An `NSBundle` is a representation of code and resources on disk

• These bundles can arbitrarily be loaded and removed from memory during program execution

• After loading a bundle, its contents can be accessed as if they were present right from the start
Cocoa Class Hierarchy

- NSObject
  - NSEvent
  - NSResponder
    - NSWindow
    - NSView
      - NSControl
        - NSButton etc.
    - NSApplication
  - NSCell (lightweight controls)
  - NSMenu
  - NSMenuItem
  - etc.
MVC Paradigm

User input ➔ Controller ➔ Model ➔ Notify ➔ View ➔ Update ➔ Controller ➔ Update

View ➔ Update ➔ Model ➔ Notify ➔ Controller ➔ Notify ➔ View

Diagram showing the MVC Paradigm with arrows connecting View to Controller to Model to Controller to View, indicating the flow of user input to update and notify actions.
Delegation

• A delegate is a class whose methods are called from another class that wants to plan for extending its functionality
Example: NSTableView
Example: NSTableView

- **NSTableViewDataSource**
  - numberOfRowsInTableView:
  - tableView:objectValueForTableColumn:row:

- **NSTableViewDelegate**
  - tableView:viewForTableColumn:row:
  - tableView:heightOfRow:
  - tableView:shouldEditTableColumn:row:
  - tableViewColumnDidResize:
  - selectionShouldChangeInTableView:
How could we extend the functionality of **NSString**?

- Could create a subclass, e.g. **MyNSString**
  but then we have to change all code to use that new class

- Could change **NSString** itself
  but this requires access to the source code for that class

- Instead: Create a **category**

  ```
  @interface NSString (NSStringExtensions)
  - (NSString *)reversedSentence;
  @end
  ```
Responder Chain

• Most UI objects are subclasses of `NSResponder` and can respond to events

• Sending an event up the chain:
  ```
  [NSApp sendAction:NSSelectorFromString(@"hello") to:nil from:self];
  ```

• The focused widget is called the **first responder**

• The framework takes care of **responder chain** and passes along an event until it can be handled by some object
Demo
Designing Interactive Systems 2

Lecture 6: macOS

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/dis2
CHAPTER 14

Swift
Swift

- Syntax very similar to scripting languages
- Compatibility to Objective-C code
- Designed for type-safety
- Introduces powerful tuples
- You can also implement functions in enums
- Open source
Hello Swift

• Declaration of a constant (with an inferred type)
  
  ```swift
  let a = 5
  ```

• Declaration of a variable (with a specified type)
  
  ```swift
  var b: Double = 7
  ```

• Type safety forces us to have two matching types on both sides of a math operation
  
  ```swift
  b = Double(a) * b
  ```

• You can even use emoji as names for your variables or classes
  
  ```swift
  let 😎 = "That's mind-blowing."
  ```
Optionals

• By default, variables and constants cannot be `nil`
  • But Cocoa and Objective-C love putting `nil` into properties
  • Hence, a more expressive way for nullable items is needed

• **Optionals** allow to express variable that can also be nil
  • Enum that can either have a value `Some(T)` or no value `None`
  • Note: Different definitions of nil between Objective-C and Swift
  • Optionals are identified by the ? in their type
    ```
    var someValue: Int?
    ```
Optional Binding

```swift
var error: NSError?
methodCallThatMightRaiseAnError(&error)

if let err = error {
    print(err.localizedDescription)
} else {
    print("No error occurred!")
}
```
Optional Chaining

- Often used in combination with delegates
  ```swift
  var delegate: MyDelegate?
  ```

- Optional chaining for elegant way to check for nil
  ```swift
  self.delegate?.numberOfItems(in: self)
  ```

- Explicitly unwrapping this variable if it is nil would result in an exception
  ```swift
  self.delegate!.numberOfItems(in: self)
  ```
CHAPTER 15

Cocoa App Basics
Views & Controllers

AppDelegate

NSWindowController

NSToolbar

NSViewController

NSView

NSWindow
Views & Controllers

• Window: **NSWindow** class

• **NSWindowController** manages a window
  • E.g., load, show, close a window
  • Useful if app has multiple windows, one NSWindowController for each NSWindow

• **NSWindow** has a contentView property of type NSView

• **NSViewController** manages an NSView (property: **view**)
  • Methods, e.g., `viewDidLoad`, `viewWillAppear`, `viewWillDisappear`, ...
  • Connect to Actions and Outlets
Coordinates

- NSPoint, NSSize, NSRect

- A view has two ways to access its position:
  - `bounds` in widget’s coordinate system
  - `frame` in parent’s coordinate system
**NSEvent**

- Event objects are emitted for both mouse and keyboard events
- Contain the mouse’s position in the window’s coordinate system

```swift
override func mouseDown(with event: NSEvent) {
    self.mouseLocation = event.locationInWindow
    let windowPoint = event.locationInWindow
    let localPoint = self.convert(windowPoint, from: nil)
    ...
}
```
Drawing

- NSViews perform their drawing code in `draw(_ dirtyRect: NSRect)`
- Override this method and put all view-specific drawing instructions here
- If the view does not directly inherit from NSView, call `super.drawRect` first!
- Calling `setNeedsDisplay(_ invalidRect: NSRect)` or `needsDisplay = true` will force a redraw
CHAPTER 16

Interface Builder Basics
Storyboard
Widget Library
Guides

- Interface Builder helps developer implement a consistent Mac look by recommending positions for widgets.
- Considers margins, centerlines and baselines.
Segue

- A transition from one screen to another in the storyboard
- Typically opened in new window, but other styles possible
Actions & Outlets

• Interface Builder allows to connect Controller code with the designed UI by dragging

• Properties with the **@IBOutlet** keyword are widgets that are defined in the Interface Builder

• Methods with the **@IBAction** keyword are called from widgets defined in the Interface Builder
Designables & Inspectables

- Widget implementations with the `@IBDesignable` keyword will render a preview in Interface Builder.

- Properties with the `@IBInspectable` keyword can be set from the Attribute Inspector UI in Interface Builder.
Demo
CHAPTER 17

Auto Layout
Auto Layout

- Constraint based UI layout engine
- Tries to fulfill a set of equations when UI appears
- Support for internationalization
Frame-based Layout

Σ 2 pieces of information needed per direction
Layout is Dynamic...

External Changes
- Window is resized
- Support different screen sizes

Internal Changes
- Displayed content changes
- Language is changed
More Measures

The diagram illustrates the measurement points for a rectangle:

- **Top**
- **Bottom**
- **Leading**
- **Trailing**
- **Width**
- **Height**
- **Center X**
- **Center Y**
- **Baseline**
Constraints

\[
\text{red\.leading} = 1.0 \times \text{blue\.trailing} + 8.0
\]

```
NSLayoutConstraint *constraint =
    [blue\.trailingAnchor constraintEqualToAnchor:red\.leadingAnchor
    multiplier:1 constant:-8];

[constraint setActive:YES];
```
Goal

• Provide a series of equations that have one and **only one possible solution**

• Ambiguous constraints have more than one solution

• Unsatisfiable constrains do not have valid solutions.

• In general, the constrains must define both the size and the position of each view
Three Similar Designs?
Intrinsic Size

• Some views have a natural size given their current content

• E.g., a button’s intrinsic content size is the size of its title plus a small margin

• Views that have an intrinsic content size can be defined by two constraints alone
Priorities

• When creating a UI that suits multiple screens, we sometimes have more than one requirement on the position of a view

• Example
  A view that ideally takes 25% of the screen’s width but is always at least 40pt wide

\[
\begin{align*}
\text{yellow.width} &= 0.25 \times \text{parent.width} + 0 \quad \text{(Priority 250)} \\
\text{yellow.width} &\geq 1 \times \text{NotAnAttribute} + 40 \quad \text{(Priority 1000)}
\end{align*}
\]
Combining Widgets

Label

Textfield
Hugging Priority & Compression Resistance

I don’t want to grow!

Content Hugging

I don’t want to shrink!

Compression Resistance
Hugging Priority & Compression Resistance

I did not shrink, and I did not care about stretching.
Demo
CHAPTER 18

Cocoa Bindings & Core Data
Cocoa Bindings

- Cocoa Bindings are an approach to keep Model and View (as in MVC) synchronized without having to write a lot of glue code

- Example: Keeping a displayed table of sheep (View) synchronized with the corresponding array (Model) of data in memory, and provide a label informing the user about the count of selected sheep (another View)
Core Data

• Object-graph management and persistence framework

• To define application data model graphically

• Provides common functionality
  • Undo, Redo
  • Persistence (save to disk, read from disk in XML or SQLite format)
Cocoa Bindings: MVC

Controller

Cocoa Bindings
ArrayController

View

Interface Builder
File

Model

CoreData
Managed Store
Demo