IBM Simon
1994
CHAPTER 27

Mobile Devices
Mobile Device Characteristics

- Small screens
  - Users interact with one app at a time
  - Users interact with one screen at a time
  - Onscreen help is minimal
Mobile Device Characteristics
Mobile Device Characteristics

Osmos HD

Pulse
Mobile Device Characteristics

• Small screens
  • Users interact with one app at a time
  • Users interact with one screen at a time
  • Onscreen help is minimal

• Limited memory $\Rightarrow$ Restrictive memory management

• Limited power $\Rightarrow$ Efficient code crucial for battery life

• New input and sensor technologies
Mobile Device Characteristics

• Small screens
  • Users interact with one **app** at a time
  • Users interact with one **screen** at a time
  • Onscreen help is minimal

• Limited memory $\Rightarrow$ Restrictive memory management

• Limited power $\Rightarrow$ Efficient code crucial for battery life

• New input and sensor technologies

• **Context is key** $\Rightarrow$ (task focus, peripheral use)
Context is Key
Context is Key
10 Golden Rules of Interface Design (see DIS 1)

1. Keep the interface simple!
2. Speak the user’s language!
3. Be consistent and predictable!
4. Provide feedback!
5. Minimize memory load!
6. Avoid errors, help to recover, offer undo!
7. Design clear exits and closed dialogs!
8. Include help and documentation!
9. Offer shortcuts for experts!
10. Hire a graphics designer!
Life as an App

• A mobile OS is an **app-centric environment**

• **One app per task**
  • Hence, do one thing but do it well

• **Sandboxing**
  • Data is typically stored per app and not visible to others

• Data exchange between apps difficult

• Define the task that users want to accomplish with your app
Designing the UI

• Make it obvious how to use your app
• Sort information from top to bottom
• Use alignment to ease scanning and communicate groupings
Designing the UI

• Make it obvious how to use your app
• Sort information from top to bottom
• Use alignment to ease scanning and communicate groupings
• Minimize text input
• Provide fingertip-size targets
• Touch is always in focus
Designing the UI

- Make it obvious how to use your app
- Sort information from top to bottom
- Use alignment to ease scanning and communicate groupings
- Minimize text input
- Provide fingertip-size targets
- Touch is always in focus
Interaction Design

• There are no on-screen signifiers how and where to perform multitouch gestures

• On some targets pressure input possible

• Interaction patterns vary between different platforms: Follow the respective guidelines for intuitive operation

• If you use complex gestures, help the user
Gestures

- Tap
- Double tap
- Touch and hold
- Drag
- Flick
- Swipe
- Pinch open
- Pinch close
- Rotate
- Shake
Platform-Independent Gestures

Pinch for zooming in and out

Double Tap zooms into content, or toggles between zoom levels
iOS Gestures

Swipe reveals actions in tables or return to previous screen

Touch and hold selects a word when performed inside editable text

Three Finger Pinch to copy and paste text
Android Gestures

- **Swipe** to switch between content tabs
- **Swipe** to complete actions upon passing a threshold
- **Long Press** reveals additional features, e.g. the edit mode of a table
Designing the User Interface

• Different resolutions and aspect ratios:

<table>
<thead>
<tr>
<th>Device</th>
<th>Classic</th>
<th>Retina</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>480 x 320</td>
<td>960 x 640</td>
</tr>
<tr>
<td>iPhone 5 (C,S), SE</td>
<td>1136 x 640</td>
<td></td>
</tr>
<tr>
<td>iPhone 6/S</td>
<td>1334 x 750</td>
<td></td>
</tr>
<tr>
<td>iPhone 6/S Plus</td>
<td>1920 x 1080</td>
<td></td>
</tr>
<tr>
<td>iPad, iPad mini</td>
<td>1024 x 768</td>
<td>2048 x 1536</td>
</tr>
<tr>
<td>iPad Pro 9.7”</td>
<td>2048 x 1536</td>
<td></td>
</tr>
<tr>
<td>iPad Pro 12.9”</td>
<td>2732 x 2048</td>
<td></td>
</tr>
</tbody>
</table>

• Device orientation:
  • Portrait or landscape
  • Designing for the iPad requires more than increasing the resolution
CHAPTER 28

Mobile Application Styles
Productivity Applications

Calendar

Notes

Microsoft Excel
Utility Applications

Compass

Stocks

The Elements
Immersive Applications

Monument Valley 2

Pokémon GO

Google Earth
CHAPTER 29

iOS
iOS History: Interaction Milestones

- **2007**: Multitouch
- **2008**: App Store
- **2010**: iPad
- **2011**: Siri
- **2014**: Apple Watch
- **2015**: 3D Touch
- **2015**: iPad Pro & Pencil Split-Screen Mode
- **2017**: ARKit
- **2019**: iPadOS, SwiftUI
iOS Architecture

Core OS

Core Services

Media

Cocoa

Cocoa Touch

Core Services

Core OS

Media
iOS Development: Similar to macOS Development

- **From AppKit to UIKit**
  - Redesigned Views for limited screen space, no menu bar
  - Different event handling (multiple touch input events instead of 1 mouse click; no hover menus)
  - More modern framework, e.g. target-action is no longer 1:1 but 1:n
  - Originally only RGB color space

- **Changes in Foundation**
  - No Cocoa bindings
  - No distributed objects
  - No garbage collection*

- **New & adapted Frameworks**
  to interact with phone hardware
iOS: Handling Touch Input

• Override `touch methods`

```swift
override func touchesBegan(_ touches: Set<UITouch>,
with event: UIEvent?) {
    if let touch = touches.first {
        print(touch.location(in: self))
    }
}
```

• Use a `gesture recognizer`

```swift
self.recognizer = UIPinchGestureRecognizer(target: self,
action: #selector(action(_:)))
self.view.addGestureRecognizer(recognizer)
```
**iOS App Lifecycle**

- Lifecycle delegate methods allow apps to react to state changes
- Explicit termination only by user in task switcher, only in case of problems
- The OS decides when to terminate an app, mostly depending on memory footprint, and the app may not be aware of the termination
iOS Apps: Components

App Specification

Code

User Interface

Assets
Adapting to Different Devices

• How to create an interface that works well on different aspect ratios?

• Storing pixel coordinates is not flexible, requires values for every different screen size

• Solution: **AutoLayout** (2011) describes relationships between widgets instead

• Key addition to AutoLayout in iOS: Declare variations in UI depending on size classes
iOS: Trait Variations
iOS: Trait Variations
### Some iOS SDKs

#### App Frameworks
- App Extensions
- Handoff
- Multitasking
- Notifications

#### Media & Web
- AirPlay
- MapKit
- MusicKit
- WebKit

#### App Services
- CloudKit
- HealthKit
- Machine Learning
- SiriKit

#### Graphics & Games
- GameCenter
- Metal
- SceneKit
- SpriteKit
iOS Design Themes

• **Clarity**
  - Subtle decorations
  - Sharpened focus on functionality
  - Use negative space, colors, fonts to highlight important content
  - Direct manipulation to support understanding

• **Deference**
  - UI does not compete with content
  - Content fills entire screen
  - Fluid motion
  - Typically no bezels
  - Translucency hints at more content

• **Depth**
  - Distinct visual layers
  - Realistic motion
  - Navigational transitions provide sense of depth
iOS Architecture Recap

Cocoa
- AppKit
- Foundation
- CoreData

AppKit
- View
- Controller
- Model
  Foundation, CoreData

UIKit
- View
- Controller
- Model
  Foundation, CoreData

CoreData

Cocoa Touch
- UIKit
- Foundation
- CoreData
IOS GOES DECLARATIVE

SwiftUI
**SwiftUI (2019)**

- **Designed for Swift** as successor to ObjC-based AppKit (Mac) and UIKit (iOS, etc.)

- **Unified** framework across Apple’s devices (on macOS, iOS, watchOS, tvOS)
  - Generates code using native widgets and technologies on each device

- **Declarative** instead of imperative programming paradigm
  - Declare UI and link to model code (with declarative **Combine** framework)
  - SwiftUI derives appropriate widgets and layout for each device class
  - Very compact code for standard UIs, supporting many features automatically (Dark Mode,…)
  - Fewer opportunities for fine-tuning your UI
  - Major rethinking from imperative OO UI design paradigm needed for complex apps

- Still in its early stages
SwiftUI: How it works

- **One language** for UI and code, using Swift’s language features (closures,..)
  - Like Motif’s UIDL, very compact, but here it’s the actual source code

- **Views** are structs (lightweight classes), discarded and generated frequently

- **Closures** (lambda functions) enable Views, e.g., to contain executable subviews

- Views are a function of their state (program state), not a result of a sequence of events

- **Editable live preview** in Xcode, synchronizing UIDS and code (⇒ Morphic)
  - Using Swift’s dynamic code replacement for live updates
  - Simultaneous previews for multiple device classes
SwiftUI: Demo
SwiftUI: Layout

VStack

HStack

ZStack
View Modifiers

A view modifier can change attributes of its input

Text("Label")
  .foregroundColor(Color.red)

VStack {
  Text("Label 1")
  Text("Label 2")
}.foregroundColor(Color.red)

Text("Label")
  .padding()
  .background(Color.red)

Text("Label")
  .background(Color.red)
  .padding()

It can also work recursively on all child views

Padding inside

Padding outside
MVVM Paradigm

- View
- ViewModel
- Model

**Bindings**
- Update
- Create
SwiftUI: Trends

• From imperative to declarative programming

• From event-based programming to Publish/Subscribe mechanism linking to model

• Still in its early steps
  • API still evolving (similar to Swift in 2014)
  • Until provided natively, integrating legacy ObjC-style UIKit frameworks is unelegant (similar to Carbon apps in 2001)

CHAPTER 30

Android
Android History

- **2007**: Roots of Android
- **2008**: First Android device
- **2009**: Android Market
- **2011**: Support for tablets
- **2013**: “OK Google”
- **2014**: Support for TVs and watches
- **2015**: Fingerprint gestures
Android Stack

- **Linux Kernel**
  - Drivers, Shared Memory, Power Management, USB, WiFi

- **HAL**
  - Audio, Bluetooth, Camera, Storage, Graphics, Sensors

- **Native Libraries**
  - Audio Manager, Freetype, OpenGL, SQLite, SSL, WebKit

- **Android Framework**
  - Content Providers, Activities, Resources, Telephony

- **Android Runtime**
  - ART, Core Libraries

- **Applications**
  - Alarm, Browser, Calendar, Photo Album, SMS, ...

- **Content Providers, Activities, Resources, Telephony**

- **Core Libraries**
  - Audio, OpenGL, SQLite, SSL, WebKit

- **Android Framework**

- **Linux Kernel**

- **HAL**

- **Native Libraries**

- **Applications**

- **Android Stack**
Application Fundamentals

• Idea: **Share** elements of applications
  - No single entry point

• All of the four different **types of components** are entry points for the system or user
  - Activities
  - Services
  - Broadcast receivers
  - Content providers
Activity

• Single screen of your application’s UI
  • Contains a tree of views
  • Defines menus
• Starts & stops services
• Calls other activities via intents
Intent

• Messaging object to request an action from another app (component)

• Explicit intent
  • Open another activity in the same app

• Implicit intent
  • Requesting an abstract “service”
  • Caller does not know callee

• Intent filters expose functionalities to other components
Broadcast Receiver

• Broadcasts are implicit intents
  • e.g. for system events like timezone change, device shutdown, ...

• Broadcast receivers are used to register from system or application events

• Use a dynamic broadcast receiver
to make your app react to changes during runtime

• Use a static broadcast receivers
to start your app on a specific broadcast
Service

- Long-running operation in the background and does not provide a UI
  - e.g., network transactions, play music, perform file I/O

- (Unbound) service
  - Is kept alive by the system even if the starting Activity has finished executing

- Bound service
  - Components can bind to a service and interact with it through an interface exposed by the Service
  - Client Server architecture
  - When the last client unbinds from the service, the system destroys it
Content Provider

- Content Provider
  - SQLite Database
  - Cursor
  - Widgets
  - Search

- Other Apps

App
Android Manifest

• XML file that defines a black box view of an application

• Interface between the OS and the app
  • Icon
  • Requirements (e.g., minimum API level)
  • Permissions (e.g., making calls)

• Exposes app’s functionality
  • Available activities
  • Intent filters (e.g., entry point)
Tasks and Multitasking

- **Tasks** are a sequence of activities (possibly from different apps)

- Every time a new activity is started, the previous one is moved to the task’s **back stack**
  - The same activity can be instantiated multiple times in one back stack
  - The back button switches to the previous activity in the stack
  - The home button signals that the user switches to a new task
Back Navigation
Deep Links

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.
Conceptual Models

Designer

Design Model

System Image

User

Mental Model

← = →
The Up Button

• The on-screen up button navigates to a parent activity that is statically defined by the developer.

• It never exits the app and hence does not exist on root activity.

• For tasks that remain in one app, up and back behave identically.

• Pressing the up button creates a new task if the current activity was presented from an activity of a different app.

• The up button cannot be used to navigate between sibling contents, e.g. paged contents inside of an activity.
Switching Tasks

TASK 1

My sheep

Sheep details

Betty

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

TASK 2

Compose

Email

TO: Betty

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.
CHAPTER 31
Designing for TVs
TV UIs: Design Considerations

- Across the Room
  - Users sit a few meters away from the screen
  - Resolution and viewing distance make it difficult to process too much information
    - Show ~ as much as on phone
  - “Connect” user with content
TV UIs: Design Considerations

- Immersion is key
- Primary use: entertainment
- Engulf people in a cinematic experience
- Exploit canvas with edge-to-edge scenery
- Use fluid animations, captivating audio and vibrant colors
TV UIs: Design Considerations

• **Clarity**
  - Use consistent layouts
  - Make the focus clear and unmistakable, even from at distance
  - Movement across space is consistent and predictable
Input?
Focus Model

- Fundamental interaction principle for TV UIs
  - Always one element highlighted
- Always move focus in the direction expected
  - Focus moves in the direction of the gesture
    - Content might move in the opposite direction of the focus
    - But: (fullscreen) objects move in the direction of the gesture
- Make the focused item obvious
Focus Model
Focus Model
Overriding the Default Navigation

• Needed if
  • Some UI elements are not accessible by the focus model
  • The semantic order of contents does not fit their physical arrangement
    (e.g., in two-column designs)

• Possible solutions
  • Statically define item successors (e.g., Xbox)
  • Add dynamic focus guides (e.g., tvOS)
Overriding Default Navigation on Xbox

SELECT A CHAPTER

One

Three

Two

Four
Overriding Default Navigation on Xbox

```xml
<StackPanel Orientation="Horizontal" Margin="300,300">
  <UserControl XYFocusRight="{x:Bind ButtonThree}">
    <StackPanel>
      <Button Content="One"/>
      <Button Content="Two"/>
    </StackPanel>
  </UserControl>
  <StackPanel>
    <Button x:Name="ButtonThree" Content="Three"/>
    <Button Content="Four"/>
  </StackPanel>
</StackPanel>
```
Overriding Default Navigation on tvOS: Focus Guides
Overriding Default Navigation on tvOS: Focus Guides

```swift
var focusGuide: UIFocusGuide = {
    let fg = UIFocusGuide()
    self.view.addLayoutGuide(fg)
    fg.rightAnchor.constraint(equalTo: shopButton.rightAnchor).isActive = true
    fg.bottomAnchor.constraint(equalTo: moreButton.bottomAnchor).isActive = true
    fg.leftAnchor.constraint(equalTo: shopButton.leftAnchor).isActive = true
    fg.topAnchor.constraint(equalTo: moreButton.topAnchor).isActive = true
    return fg
}()

override func didUpdateFocus(in context: UIFocusUpdateContext, with coordinator: UIFocusAnimationCoordinator) {
    super.didUpdateFocus(in: context, with: coordinator)
    switch context.nextFocusedView {
    case self.moreButton:  focusGuide.preferredFocusEnvironments = [shopButton]
    default:                focusGuide.preferredFocusEnvironments = [moreButton]
    }
```
Supporting Focus: Parallax Effect (tvOS)

- An example of the subtleties of UI design
- Focused item elevated
- Sways responding to small remote touches
- Illumination for reflection effect
- After inactivity: grow while other items dim
- Layered images create parallax effect
- Reinforces focus, but in subtle ways
Beyond Smartphones: In-Car Interaction
Beyond Smartphones: Wearables
Limitations repeat themselves in history.
APPENDIX

Code Usability
Explicit Parameter Names

Java

```java
Person joe = new Person("Joe", "Snuffy", 21);
joe.hi();
```

Objective-C

```objective-c
Person *joe = [Person personWithFirstName:@"Joe" lastName:@"Snuffy" andAge:21];
[joe sayHi];
```

Swift, Python

```swift
joe = Person("Joe","Snuffy", age=21)
joe.hi()
```
What Else Do Developers Need?

- A powerful API that lets you do the right thing
- Availability of frameworks
- Easy project management
- A good IDE
- Proficiency in their language of choice
Python: ♥ for File Handling and Data