National College of Ireland

Mobile Architecture & Security

Patterns for Mobile App Development

Mikhail.Timofeev@ncirl.ie

Overview

① Critical Aspects of Mobile Software

② New Patterns and Practices

③ Patterns for Interaction

④ Patterns for Presentation

⑤ Behavioral Patterns

⑥ Summary

Critical Aspects of Mobile Software

Mobile software is software designed to run only on very special devices

This apparently simple fact imposes a number of constraints on developers and raises a few issues for which new patterns and practices are required

- **The Interaction Model**
  - A key difference between desktop and mobile applications is that users of modern mobile apps likely will use a finger to point to any content
  - Using a finger to select a menu item is quite natural and a pleasant experience for the user, but not if the clickable item is too small
  - But a finger will never be as precise as a mouse pointer or a stylus
  - This is addressed at the design level by keeping usability practices in mind

Critical Aspects of Mobile Software

- **The Presentation Model**
  - Touch-sensitive screens force you to increase the size of any clickable content, like buttons and links
  - You cannot simply make each button and link larger—the overall size of the screen of a mobile device is much smaller than the screen of a laptop.
  - The combined effect of these two factors leads to a complete re-thinking of the user interface of a mobile app
  - <u>Presentation of the data is also subject to new rules</u>
  - Scrolling is highly encouraged, and it is done vertically for the most part
  - However, forms of horizontal scrolling are coming up, especially in Windows Phone with the panorama
  - But a widely agreed guideline says that horizontal scrolling should be avoided altogether on webpages and minimized in desktop user interfaces
  - Or think of the horizontal swipe gesture: it is an accepted way to flip through distinct but related blocks of content

Critical Aspects of Mobile Software

- **The Behaviour of the Application**
  - Screen real estate is not the only resource that is limited on a mobile device:
    - CPU, GPU, local storage (SD memory, etc)
  - Memory consumption should be kept under strict control, and optimizing algorithms is more important than ever
  - A mobile device is interruptible and is utilized by users in a way that requires a strong multitasking logic
  - Multitasking conflicts with a lack of processing power and memory
  - The net effect is that all apps should be created from the ground up around the idea that they can be put in a rest state and even removed from memory when the system reclaims their resources
  - The actual behavior of mobile applications, therefore, depends on a variety of issues that most developers never faced before

New Patterns and Practices

- Mobile design patterns apply regardless of the actual operating system and development platform
- Essential for keeping mobile apps effective and aligned with the needs and expectations of people
- Three main areas of programming are affected by patterns:
  - Application Life Cycle
  - Storage
  - Connectivity

New Patterns and Practices: App Life Cycle

- *Background Applications*
- What BAs are allowed to do may vary across different mobile operating systems like iOS, Android, and Windows Phone
- Truly unlimited multitasking (where multiple applications are alive and kicking at the same time) is just not affordable
- Apps swing between 3 possible states:
  - Actively running
  - Paused with some pending work / Paused waiting to be resumed
  - Unloaded
- Various OS differ for the actual implementation of these states

New Patterns and Practices: Storage

- Mobile Apps are fundamentally conceived to be stand-alone applications
- Mobile Apps require special tricks and application programming interfaces (APIs) to share data
- Apps are allowed to persist data locally to the device in a number of different ways:
  - Custom files
  - Local database
  - System-managed data repositories such as:
    - *Settings* bundle facility of iOS
    - *ApplicationSettings* in Windows Phone
    - *SharedPreferences* in Android
    - *PersistentStore* object in BlackBerry.

New Patterns and Practices: Storage

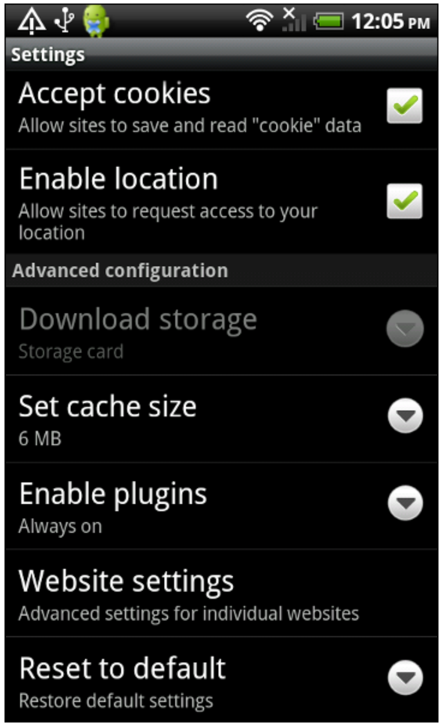
- **Custom files**
  - Offer a stream-based programming interface and can receive any serialisable in-memory object
  - Sometimes apps are assigned a specific section of the file system and persistent data is isolated
  - In Android and BlackBerry the developers can control the visibility of the files and can access SD cards
  - In Windows Phone and iOS sharing requires tricks: SkyDrive and iCloud
- **Local databases**
  - SQLite is a popular choice that requires little or no administraton
  - Well-suited as an app-specific local database and is cross-platform
  - SQL Anywhere (Sybase): commercial cross-platform solution
  - MS SQL Compact Edition and Sync Framework (Windows Phone)

New Patterns and Practices: Connectivity

- Not all mobile applications need connectivity…
- But when you wed mobile to business, connectivity becomes a crucial point
- Connectivity triggers a number of potential headaches that can be rooted in one common cause: **mobile connectivity may not be reliable**
- User interface and network-dependent operations should be designed try repeatedly before succeeding or giving up
- Connectivity should be checked constantly and if any network state change is detected – user interface (UI) elements can be enabled or disabled in a timely fashion

Patterns for Interaction

- **The Back-and-Save Pattern**
  - The core idea behind this pattern is to take any reasonable measures to minimise the typing effort of the user
  - Save the content of input screens when the user leaves (or is forced to leave) the screen
  - Don't request an explicit command (i.e., a tap) from the user to save the data just entered
- Implementation:
  - Timer to save at intervals
  - Detect when screen is being unloaded and auto-save
  - Save-as-you-go


12:05 PM

Settings

Accept cookies ✓
Allow sites to save and read "cookie" data

Enable location ✓
Allow sites to request access to your location

Advanced configuration

Download storage
Storage card

Set cache size
6 MB

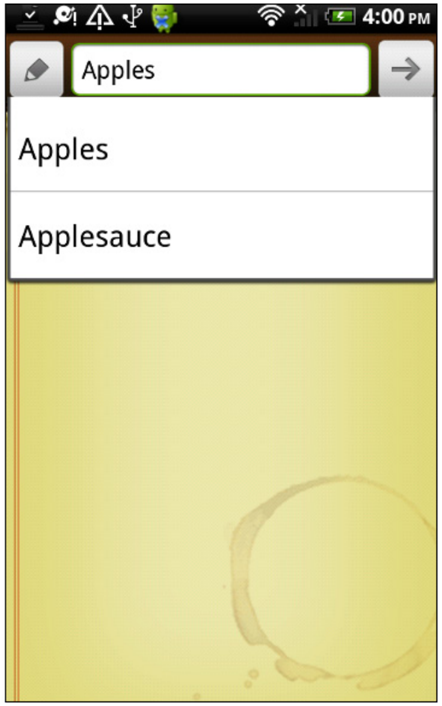
Enable plugins
Always on

Website settings
Advanced settings for individual websites

Reset to default
Restore default settings

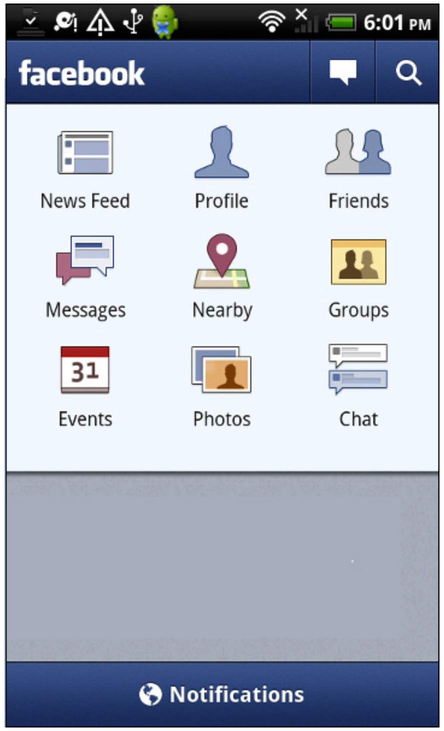
Patterns for Interaction

- **The Guess-Don't-Ask Pattern**
  - It expresses the idea that the application should try to limit data entry as much as possible
  - If there's something that you, as a developer, can do to save your users a click or some typing, then by all means do that
  - Use any available resources to make intelligent guesses and save users the largest possible bit of interaction
  - Offer suggestions to users and guess what they would most likely do
- Implementation
  - Prefill fields with geo-information, input data by enabling word completion, facilitate voice-based input, etc.



Patterns for Interaction

- **The A-la-Carte-Menu pattern**
  - Mobile Apps are built around a few specific use-cases
  - It should be self-evident to the user from the beginning which options are available
  - At any time, it should be clear for the user which action to take and how many options he/she has
  - Any functions should be no more than two or three clicks away
- Implementation
  - An effective implementation of this pattern deeply affects the usability and design of the application
  - The difference between great and not-so-great applications is mostly in the selection of use-cases
  - In addition, this pattern also relates to the idea of hiding rather than disabling controls that are not going to be used by the user in a given context



Patterns for Interaction

- **The Sink-or-Async Pattern**
  - Opt for asynchronous implementations of potentially long operations
  - Implement asynchronously any operations expected to perform for longer than a bunch of milliseconds
- Implementation
  - Asynchronous operations can be coded in one of two ways:
    - By having a worker thread make a synchronous request
    - By using the asynchronous API that the SDK of choice makes natively available
- Chaining Async Network Operations
  - If you only have an async API available, how can you chain two or more calls?
  - You probably can design the code to maintain a high level of readability with only two or maybe three sequential calls
  - With more calls, it becomes a very intricate mess

Patterns for Interaction

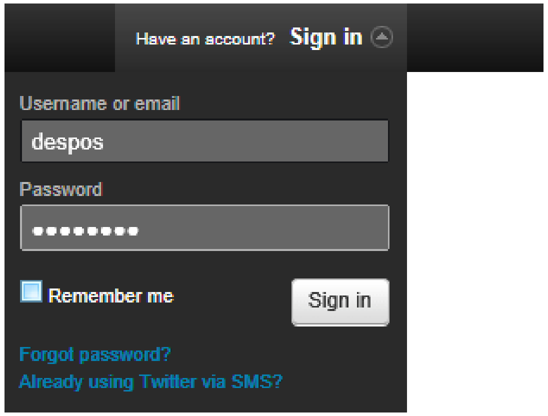
- An elegant way out would be to use **coroutines**:

```
while (!Task1_IsCompleted || !Task2_IsCompleted || !Task3_IsCompleted)
{
        if (!Task1_IsPending && !Task1_IsCompleted)
        ExecuteTask1(...);
        if (!Task2_IsPending && !Task2_IsCompleted)
        ExecuteTask2(...);
        if (!Task3_IsPending && !Task3_IsCompleted)
        ExecuteTask3(...);
}
```

- The various properties used for controlling the loop are wrapped as members of a class and set by *ExecuteTask*N procedures as appropriate
- You write operations as distinct async tasks, and the surrounding routine ensures that the overall workflow runs steps in the right sequence

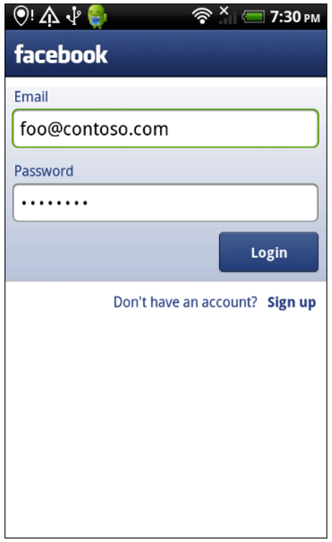
Patterns for Interaction

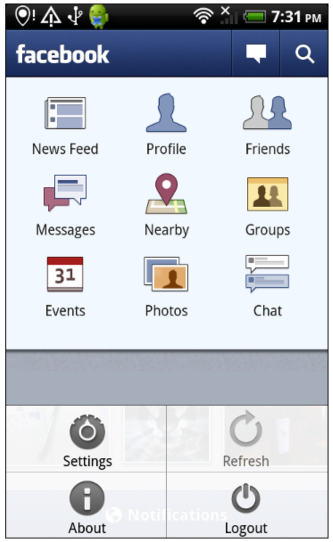
- **The Logon-and-Forget Pattern**
  - When you land on a webpage that requires authentication, you typically have a chance to ask the system to keep you logged on for a number of days
  - Having verified your credentials, the system emits an authentication cookie that the browser will use for future sessions until it expires
  - Mobile apps should do the same, and possibly store the credentials for an even longer time
  - *You should ask for credentials once, store them safely, and transparently authenticate the user in every session*



Patterns for Interaction

- Implementation
  - An application that implements the Logon-and-Forget pattern starts by offering a classic logon screen where the user can enter the user name and password
  - We assume that "Remember Me" check is always on (don't display it)
  - Either store credentials or access token
  - Keep credentials until the user explicitly decides to log out
- Security
  - Make it an option or avoid using this pattern
  - Always use encryption





Patterns for Presentation

- **The Babel-tower Pattern**
- *Avoid hard-coded and fixed layout text and design your application to support the dynamic injection of properly translated text*
- Internationalization vs. Localization:
  - **I18n** refers to changing the internal architecture to support multiple languages and regional settings
  - **L10n** is the specific action of adding support for a specific language
- Implementation
  - Replacing text over all the screens
  - Replacing graphics
  - Replacing screen layouts
  - Replacing specific portions of the application logic
  - Think about ways of storing localisation resources?

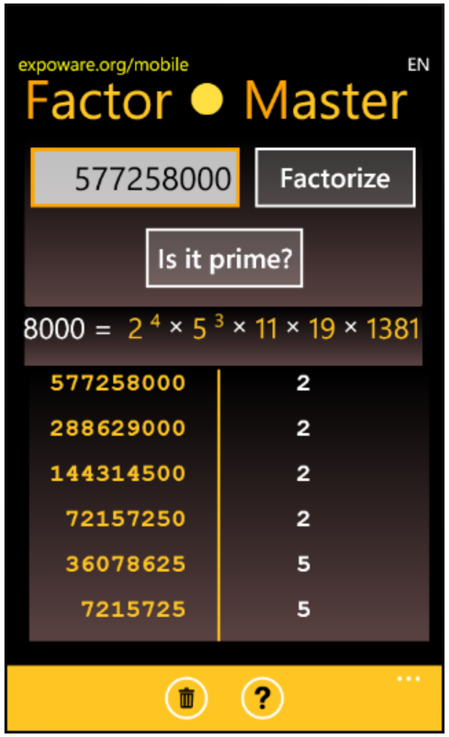
Patterns for Presentation

- **The Do-as-Romans-Do Pattern**
  - *It is compelling for users, and possibly also advantageous for developers, to abide by the look-and-feel and capabilities of the host operating system*
  - The Do-as-Romans-Do pattern is another example of a pattern that has more of a guiding principle than a concrete and detailed strategy to solve a common problem
  - Steps to implement the pattern boil down to keeping the overall user interface as close as possible to the standards of the platform
  - Refer to design guidance for each of the platforms

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com
WHEN IN ROME DO AS THE ROMANS DO
search ID: jfa2388

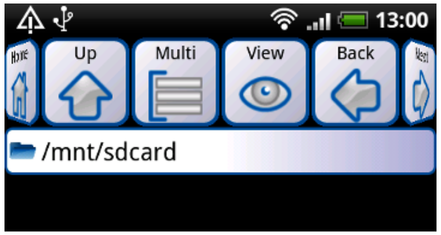
Patterns for Presentation

- **The List-and-Scroll Pattern**
  - Displaying too many items on a single webpage would make the page slower to download
  - Displaying too many items within a desktop application would force the user to deal with the scroll bar, which may not be fun
  - *Don't be afraid of using (vertical) lists in your mobile application, even long lists that contain more than 100 items to scroll*
  - Scrolling both vertically and horizontally is a natural gesture on touch devices, especially handheld devices like a phone
- Implementation
  - While building lists is a common task in mobile applications, the level of automation offered by the various platforms is not the same
  - Use *pagination* if more than 100 items
  - Can also be applied to toolbars (horizontal scroll)



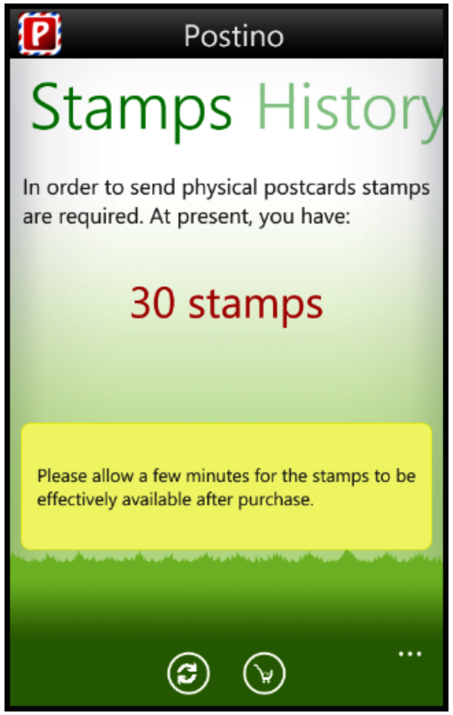
8000 = $2^4 \times 5^3 \times 11 \times 19 \times 1381$

577258000	2
288629000	2
144314500	2
72157250	2
36078625	5
7215725	5



Behavioural Patterns

- **The Predictive Fetch Pattern**
  - A design pattern historically bound to AJAX
  - With AJAX developers gained the ability to place out-of-band calls to the web server and get raw data instead of full-blown HTML pages
  - Try and fetch data ahead
  - *If you depend on network connectivity, download data that is likely to be used later and make sure you have enough data stored at any time to survive a lack of connectivity*
- Implementation
  - Use Analytics to see what is being used at what point
  - Armed with this information, you know what users are likely to do at a given stage and can code appropriately to get ahold of the data you may need next



In order to send physical postcards stamps are required. At present, you have:

30 stamps

Please allow a few minutes for the stamps to be effectively available after purchase.

Behavioural Patterns

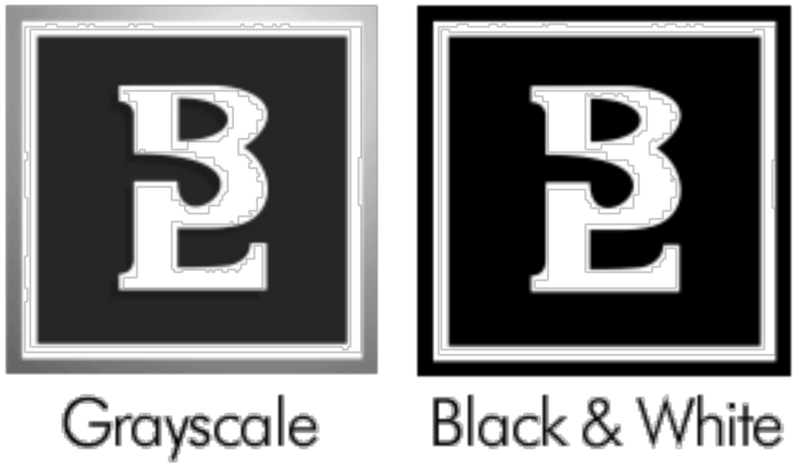
- **The Memento-Mori Pattern**
  - The expression “Memento mori” is a Latin phrase that translates as “Remember you will die”
  - Memento Mori is a reminder of the mortal condition of any app
  - *Applications always should save their relevant state when the operating system forces them into the background*
  - Every application is then responsible for determining which data needs to be saved and how
- Implementation
  - The Memento pattern is about object serialization and refers to the object’s ability to save its public state to a stream so that a brand new instance with the same state can be created later
  - All programming frameworks provide good native support for object serialization

Behavioural Patterns

- **The As-Soon-as-Possible Pattern**
  - For some apps it is critical to devise operations to work both with and without a functioning network
  - *Remote operations that are critical for the application should be implemented in a protected manner and reiterated a few times before failing. In case of failure, however, the operation should be recorded and played back as connectivity returns.*
  - The expression “as soon as possible” just indicates that the message that the user sends to the application when commanding an operation is “Perform this task as soon as possible and any way you can, at your earliest convenience.”
- Implementation
  - The As-Soon-As-Possible pattern is strictly related to network operations such as the POST of data to a remote server
  - You can implement the pattern in two ways that I like to call Black-or-White and Grayscale

Behavioural Patterns

- The Black-or-White algorithm indicates that you attempt to perform the operation, and if it fails, you notify the user and add the operation to an internal queue
- The Grayscale algorithm indicates that you break down the data to post in small pieces (a few kilobytes in size each) and attempt to send them separately



Summary

- Writing a mobile application, regardless of the target platform and the details of the SDK to use, is not the same as writing a desktop or web application
- Overall, it is an easier task because a mobile application has a far lower number of use-cases to plan for than a web or desktop application
- However, a mobile application poses a number of challenges that the developer must be ready to face
- We have discussed a dozen design patterns that attempt to provide guidance for the most common of these challenges

National College of Ireland

Questions?

Mikhail Timofeev

Office 3.18

MTimofeev@ncirl.ie