Synthesis for Developing Apps on Mobile Platforms

Jeff Foster
University of Maryland, College Park

Armando Solar-Lezama
Massachusetts Institute of Technology
Schedule for session

- Jeff Foster and Armando Solar-Lezama, Overview of current plans and work

- Thibaud Hottelier, Programming By Demonstration: But what if the user is fallible?

- Joel Galenson, CodeHint demo
Android Popularity

- Android — most popular smartphone platform
  - 70.1% of the market in 4Q12
  - 850K Android apps on Google Play (Apr. 2013)
  - 40 billion apps downloaded (Apr. 2013)

- Apps can be distributed by anyone
  - Easy to install app from any site

- Android makes a good research platform
The Android Platform

- A software stack for mobile devices
  - Operating system, middleware & key applications
- Use Android SDK to create applications
  - Libraries & development tools
- Lots of documentation
  - http://developer.android.com

(Thanks to Adam Porter for the next batch of slides)
Android Architecture

[Diagram of Android Architecture]

Dalvik Virtual Machine

- Applications typically written in Java
  - Do not run in a standard Java virtual machine

- `dx` program transforms Java classes into .dex-formatted bytecodes

- Bytecode executed in Dalvik Virtual Machine

- Applications typically run in their own processes, inside their own instance of the Dalvik VM
Dalvik Virtual Machine (cont.)

- Dalvik VM designed to run on a handset
  - Slow CPU
  - Little RAM
    - e.g., 64Mb total, ~10Mb available at runtime
  - No swap space
  - Limited battery life
Dalvik Design Choices

- Register- rather than stack-based
  - More compact .dex files
  - More efficient execution

- One .dex file for multiple classes
  - More compact .dex files

- Modified garbage collection to improve memory sharing

- Optimizations applied at installation time
Application Framework

- Window Manager
  - Manages top-level window’s look & behavior

- View system
  - lists, grids, text boxes, buttons, etc.

- Content Providers
  - Inter-application data sharing

- Activity Manager
  - Application lifecycle and common navigation stack
Application Framework (cont.)

- Package manager
  - Manages application packages

- Telephony manager
  - State of telephony services

- Resource Manager
  - Manages non-code resources: strings, graphics, and layout files

- Location manager
  - Access to system location services

- Notification Manager
  - Notify users when events occur
Applications

- Standard apps include:
  - Home – main screen
  - Contacts – contacts database
  - Phone – dial phone numbers
  - Browser – view web pages
  - Email reader – Gmail & others

- Your App!
Building an Application

See: developer.android.com/guide/developing/building/index.html
Running an Application

- By default, each application:
  - assigned a unique Linux user ID
  - executes in its own Linux process

- By default, each process runs its own Dalvik virtual machine

- Android manages process creation & shutdown
  - Starts process when any of the application's code needs to be executed
  - Shuts down when process is no longer needed and system resources are required by other applications
An App can have multiple entry points
- i.e., not just main() method

App comprises components that the system can instantiate and run as needed

Several key component classes, discussed next
Activity

- Primary class for interacting with user
  - Usually implements a focused task
  - Usually involves one screenful of data

- Typically supports one thing a user can do
  - View an email message
  - Show a login screen

- Applications can include several activities
Activity Stack

developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html
Service

- Runs in the background to perform long-running or remote operations
- Does not have a visual user interface
BroadcastReceiver

- Component that listens for broadcast announcements (events)
  - Events implemented as Intent instances
- Does not have a visual user interface
Content Providers

- Store & retrieve data across applications
- Uses database-style interface
- Example
  - Contacts
Other Interesting Things on Android

- Intents — interprocess communication mechanism
  - Also used to start up new Activities

- Fragments — layout for tables vs. phones

- Permissions — protects sensitive capabilities
Tools for Android

- We’ve been doing some work on Android security
- As a result, we’ve built up several tools for working with Android apps
- To explain those tools, brief aside into this other research
Permissions on Android

- Permissions associated with resources and OS features
  - Internet, GPS, telephony, ...

- Permissions granted at install time
  - once granted, apps can use such permissions any way they want
Finer-grained Permissions

- **Goal:** Introduce finer-grained permissions
  - E.g., Gasbuddy has full access to Internet
  - But only needs access to gasbuddy.com

- Benefits of finer-grained permissions
  - Help *app developers*
    - improve apps’ robustness against security exploits
    - attest to apps’ proper usage of permissions
  - Help *users*
    - understand *how* apps use permissions

- Also, aim to do this without changing platform
Example Finer-grained Permissions

- Internet ➔ InternetURL($d$)
  - InternetURL(gasbuddy_com)

- GPS ➔ TruncatedLoc($d$)
  - Resolution up to distance $d$

- Phone state ➔ PhoneState($p$)
  - PhoneState(UniqueID)
Our Tool Chain

- **RefineDroid** infers how permissions are used in the app
- **Redexer/Dr. Android** retrofits the app with fine-grained permissions
- **Mr. Hide** enforces fine-grained permissions at runtime
RefineDroid

- infers how permissions are used
  - string analysis to search URL-like strings
  - constant propagation to determine key parameters to privileged APIs
  - e.g. for system settings:

```java
Uri uri = Uri.parse("my_ringtone.mp3");
RingtoneManager.setActualDefaultRingtoneUri(this, RingtoneManager.TYPE_ALARM, uri);
String path = uri.toString();
Settings.System.putString(
  getContentResolver(),
  Settings.System.RINGTONE, path);
```
**Dalvik Rewriter for Android**
- injects hidelib.dex
- modifies the app’s bytecode to use Mr. Hide
- removes Android perms. and adds Mr. Hide perms.
Mr. Hide

- the Hide interface to the droid environment
  - services
    - interact with a client app and resources
  - client-side library
    - a drop-in replacement for sensitive APIs
... def test_station
    click "Continue"
    click "I have a Pandora account"
    edit (0, "account@...")
    edit(1, "password...")
    click "Sign In"
    menu
    ...
    assert_text "Antonio Salieri"
    ...
    acts = getActivities
    finish
    puts acts
end
Synthesis to the Rescue

- Synthesis for tool developers
  - Developing these tools introduces challenges that synthesis can address

- Synthesis for expert programmers
  - Next generation of tools to help transition from checking to synthesis

- Synthesis for the end user
  - Can every phone user be a developer?
Better tools through synthesis

- **Case Study: SymDroid**
  - Symbolic execution for Dalvik bytecode
  - Uses $\mu$-Dalvik, an easy-to-analyze representation

- **Challenges**
  - App code is tightly tied to the Android framework
  - Framework is big and complex
    - Symbolic execution of the framework is infeasible
    - Hand creation of models is brittle and expensive

- Can you synthesize a model of Android?
Synthesizing an android model

- **Idea: Multimodal synthesis**
  - Record interactions between framework and application
  - “Sketch” a high-level view of the model
  - State high-level properties of the synthesized model
Android model as a reactive program

- **Inputs**
  - External events
    - Timers, user gestures,
      - Include method being called + parameters
  - Application calls
  - Application return

- **Outputs**
  - Calls from the framework to the application

- **Model is not just an FSM**
  - Basic building blocks include FSMs, Stacks, internal datastructures
  - User-provided sketch helps guide the choice of building blocks
    - Leverage known design patterns
Synthesis Challenges

- Conveying Structure
  - Sketch is too low level
  - We need a DSL based on design patterns

- Scalability
  - Early experiments are encouraging
  - Modularity will be crucial

- Client-Synthesis feedback loop
  - Model should only be as complex as required by the analysis
Synthesis tools for programmers

- Build on experience with synthesis tools for frameworks
  - Prospector, MatchMaker

- Key problems
  - How to describe what you want
  - How to cope with complexity

- Technical directions
  - Combining Data driven techniques with logic-based engines
Synthesis for End Users

- Can we make a programmer of every Android user?
End