Extending Android's Platform Toolsuite

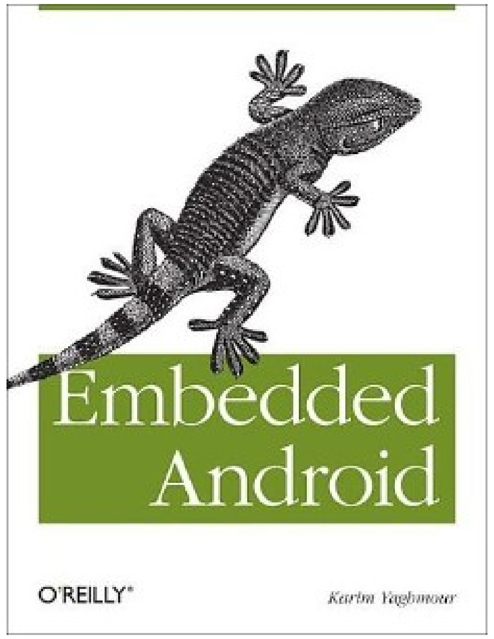
Embedded Linux Conference Europe 2015

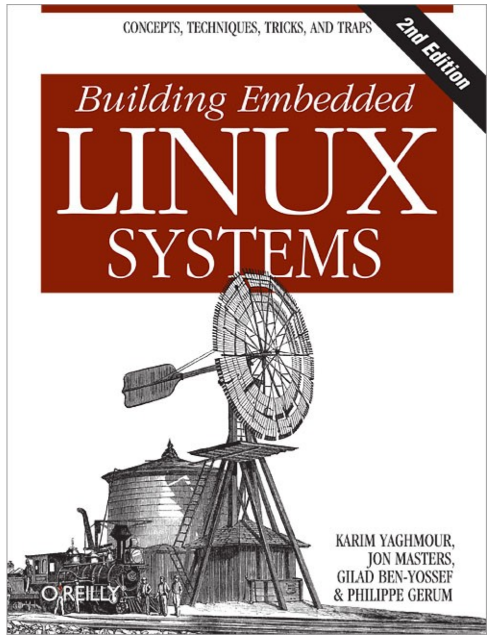
Karim Yaghmour
@karimyaghmour / +karimyaghmour
karim.yaghmour@opersys.com

Delivered and/or customized by

About

- Author of:




CONCEPTS, TECHNIQUES, TRICKS, AND TRAPS
2nd Edition
Building Embedded LINUX SYSTEMS
KARIM YAGHMOUR, JON MASTERS, GILAD BEN-YOSSEF & PHILIPPE GERUM
O'REILLY

- Introduced Linux Trace Toolkit in 1999
- Originated Adeos and relayfs (kernel/relay.c)
- Ara Android Arch Oversight
- Training, Custom Dev, Consulting, ...

Agenda

- What's out there?
- What's the problem?
- Our Objectives
- Discovering System Service Interfaces
- Architecture for Creating Monitoring Tools
- Process Monitoring
- Filesystem Monitoring/Browsing
- Understanding Binder Relationships
- Boot Animation
- The Road Ahead

1. What's Out There Now?

- Official App Dev Tools
- Official Platform Dev Tools
- 3rd Party Power User Tools
- 3rd Party App Dev Tools
- 3rd Party Platform Dev Tools

1.1. Official App Dev tools

- Android Studio (IntelliJ)
- Android Monitor (formerly DDMS)
- Several tools built into Studio for performance analysis:
  - Rendering
  - Memory
  - CPU usage
  - Battery
- Plenty of documentation

1.2. Official Platform Dev Tools

- Tools on the previous pages
- gdb / gdbserver
- ftrace, systrace, atrace
- perf

1.3. 3rd Party Power User Tools

- Google Play has a ton of apps for:
  - Process monitoring
  - File management
  - System information
  - etc.
- They all assume that the device's screen is for output
- Useless if you're trying to use the tool while doing something else on screen.

1.4. 3rd Party App Dev Tools

- CrossWalk / Cordova
- Delphi
- Xamarin Android
- etc.

1.5. 3rd Party Platform Dev Tools

- Qualcomm tools
- Intel tools, Nvidia tools, etc
- ARM Tools – DS-5
- JTAG -- Lauterbach

2. What's The Problem?

- Google obviously catering to app developers
  - App dev tools have nice finish
  - Platform dev tools ... not so much
- Official tools heavily tied to app dev IDE
  - Requires IDE-specific knowledge to extend/customize
  - Assumes official IDE is being used and/or is present
  - Assume the developer is developing an app and is trying to fix his app's problems
- Platform is huge
- Documentation is often spartan
- Existing platform dev tools assume internals understanding
  - Do you truly know how to use “dumpsys procstats”, “dumpsys meminfo” or “vdc”?
- Most platform tools can only be used on the command line
- Most 3rd party tools assume on-screen rendering of information

3. Our Objectives

- Reduce barrier to entry for platform development
- Address unmet patform developer needs
- Easy to use platform dev tools
- Build on lightweight/mainstream technologies:
  - No IDE-specific tie-in
  - Extensible language
  - Large ecosystem of reusable packages/add-ons
- Avoid monopolizing device screen

4. Discovering System Service Interfaces

- Question: What is service X's AIDL interface?
- find -name “*File.aidl”
- godir
- Android documentation
  - Focused on app developers
  - Doesn't cover everything

4.1. Raidl - Features

- Returns the AIDL interface of a service
  - AIDL based service
  - Best effort for other services (Activity service)
  - No interface for C++ service
  - No parameters names

4.2. Example Output

```
root@generic:/data/local/tmp # ./raidl iface -n power
// Service: power, Interface: android.os.IPowerManager
package android.os;

interface IPowerManager {
    void acquireWakeLock(IBinder p1, int n2, String s3, String s4, WorkSource p5, String s6); // 1
    void acquireWakeLockWithUid(IBinder p1, int n2, String s3, String s4, int n5); // 2
    void releaseWakeLock(IBinder p1, int n2); // 3
    void updateWakeLockUids(IBinder p1, int[] p2); // 4
    void powerHint(int n1, int n2); // 5
    void updateWakeLockWorkSource(IBinder p1, WorkSource p2, String s3); // 6
    boolean isWakeLockLevelSupported(int n1); // 7
    void userActivity(long n1, int n2, int n3); // 8
    void wakeUp(long n1); // 9
    void goToSleep(long n1, int n2, int n3); // 10
    void nap(long n1); // 11
    boolean isInteractive(); // 12
    boolean isPowerSaveMode(); // 13
    boolean setPowerSaveMode(boolean p1); // 14
    void reboot(boolean p1, String s2, boolean p3); // 15
    void shutdown(boolean p1, boolean p2); // 16
    void crash(String s1); // 17
    void setStayOnSetting(int n1); // 18
    void setMaximumScreenOffTimeoutFromDeviceAdmin(int n1); // 19
    void setTemporaryScreenBrightnessSettingOverride(int n1); // 20
    void setTemporaryScreenAutoBrightnessAdjustmentSettingOverride(float p1); // 21
    void setAttentionLight(boolean p1, int n2); // 22
}
```

4.3. Raidl - Demo

4.4. Raidl – How Does It Work?

```
ServiceStubClass = Raidl.class.getClassLoader()
                              .loadClass(serviceClass.getCanonicalName()+"$Stub");

for (Field serviceField : serviceStubClass.getDeclaredFields()) {
    // Get the fields that look like transaction code.
}

for (Method serviceMethod : serviceClass.getMethods())
    serviceMethods.put(serviceMethod.getName(), serviceMethod);

for (Integer serviceCode : serviceCodesMethods.keySet()) {
    // ...

    if (serviceMethod != null && isRemoteMethod(serviceMethod))
        aidlService.addMethod(serviceCode, serviceMethod);
}
```

4.5. Raidl - AOSP integration

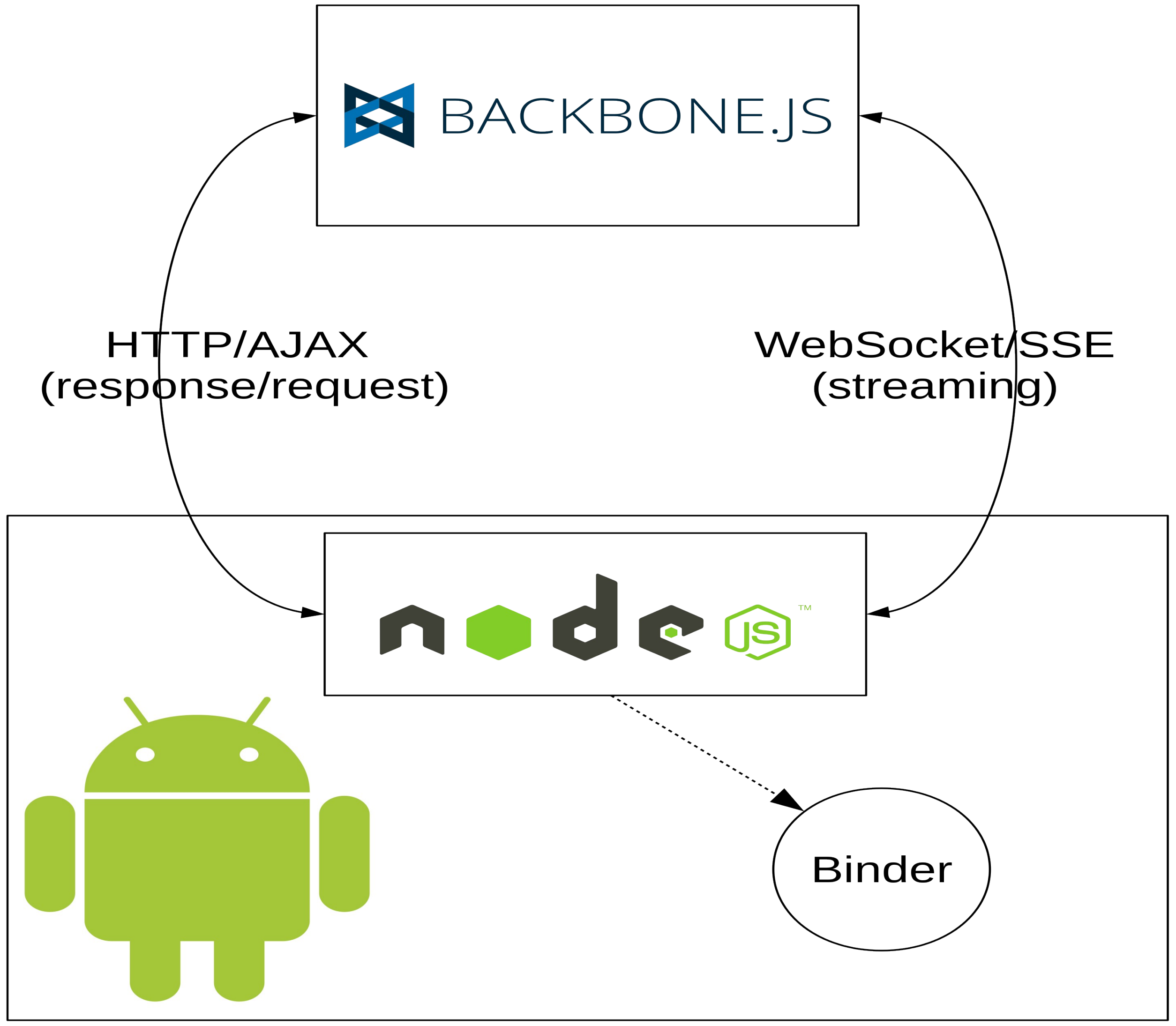
```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES := $(call all-java-files-under, src)
LOCAL_PACKAGE_NAME := raidl
LOCAL_MODULE_TAGS := optional
LOCAL_PROGUARD_ENABLED := disabled

include $(BUILD_PACKAGE)

include $(CLEAR_VARS)

LOCAL_SRC_FILES := raidl
LOCAL_MODULE_PATH := $(TARGET_OUT)/bin
LOCAL_MODULE_CLASS := EXECUTABLES
LOCAL_MODULE := raidl
LOCAL_MODULE_TAGS := optional
include $(BUILD_PREBUILT)
```

4.6. Raidl - Running an .apk

- .apk == .jar (with some DEX code)
- `DexClassLoader:` "A class loader that loads classes from .jar and .apk files [...]"
- Ergo:

```
export CLASSPATH=/system/app/raidl.apk:/system/app/raidl/raidl.apk
exec app_process /system/app com.opersys.raidl.Raidl "$@"
```

5. Architecture for Creating Monitoring Tools



5.1. Tool architecture

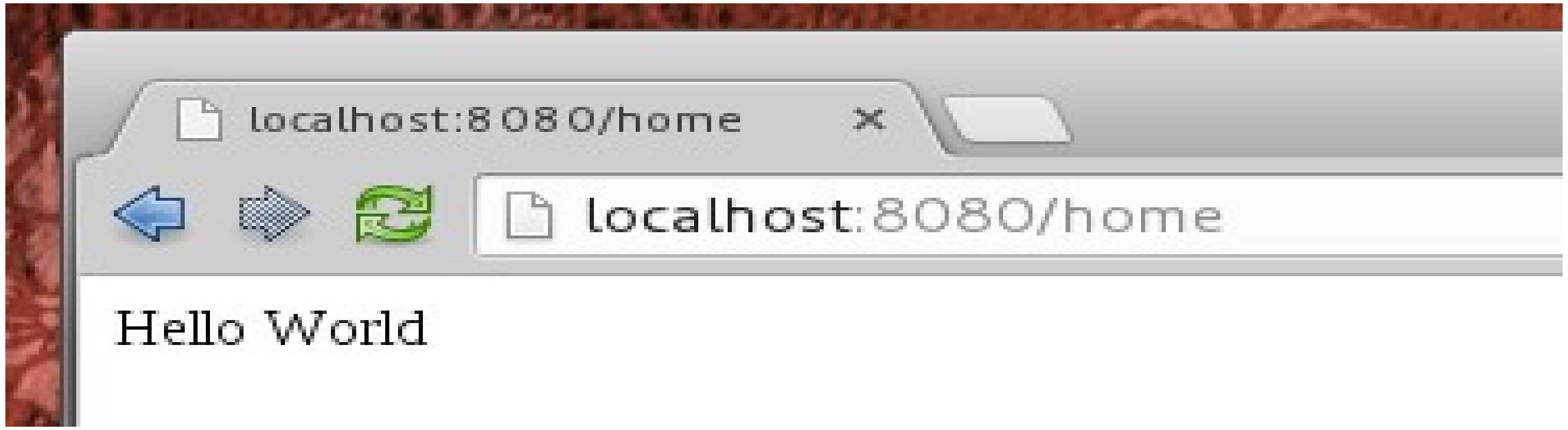
- Backend: Node.js + Express
- Frontend: Backbone.js + w2ui
- AJAX communication
- Websocket or Server-Sent events

5.2. Node.js in Android – Why?

- One language to rule them all: Javascript
- 132 510 Node.js packages
- Ease of use
- Web 2.0 support (SSE, WebSockets)
- Speed
  - V8
  - Binary modules
- **Runtime independence**
- Few actual alternatives: Go, C/C++, Python, etc.

5.3. Node.js – It's easy!

```
var express = require('express');
var app = express();

app.get('/home', function(req, res) {
 res.send('Hello World');
});

app.listen(process.env.PORT || 8080);
```



Hello World

5.4. Node.js in Android – Why not?

- Still pretty slow
- Runtime independence
  - Node is within its Linux bottle
- Difficult to package in Android
- It's Javascript
  - WAT! https://www.destroyallsoftware.com/talks/wat

5.5. How to use Node.js on Android

- Older versions (0.8), binaries available
  - Too old for comfort
- Development version (0.11, now 0.12) was patched with Android support
- Backported to 0.10
- V0.10 Binaries are avaible!
- Io.js and Node v0.12: TBD.
- https://github.com/opersys/node

5.6. Distribution

- Extracted in local files
- Multiple binary packages
  - ARM, ARM + PIE, ia32, ia32 + PIE
- Started by a simple Android application
- Able to start as root

6. Process Monitoring

- ps / top
- htop (Cyanogenmod)
- Studio/Eclipse/DDMS/Monitor integrated
- On the Play Store...
  - ... hundreds of candidates
  - Few are aimed at developers

6.1. Process Explorer

- Browser based process manager
- Displays logcat (live!)
- Process statistics
  - /proc based
- Needs root access for better function
- Works on Chrome and Firefox

6.2. Process Explorer - Demo

7. Filesystem Monitoring/Browsing

- ls, find
- adb push/pull
- On the Play Store...
  - ... hundreds of candidates
  - Few are aimed at developers

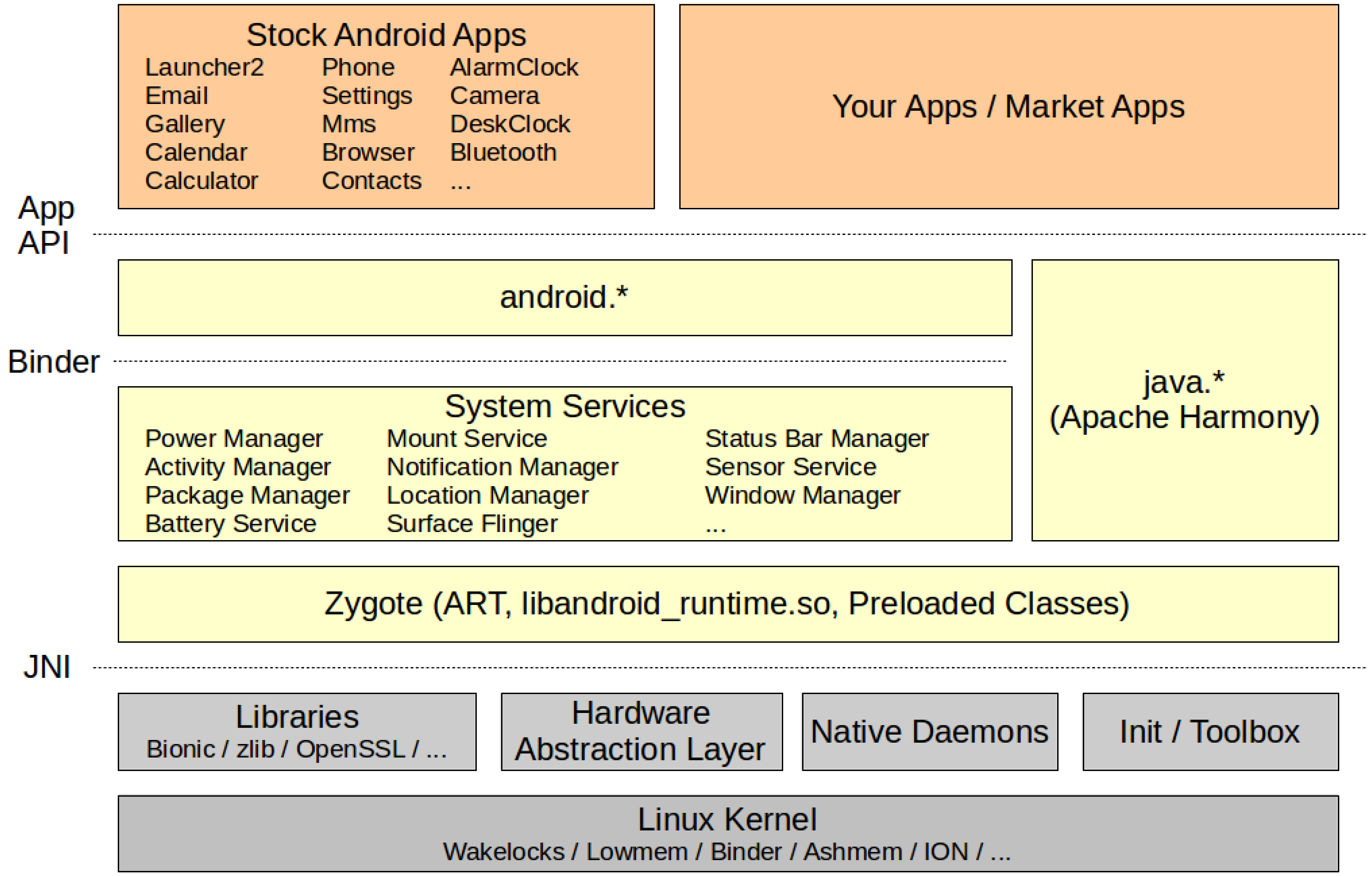
7.1. File Explorer

- Browser based file manager for Android systems
- File upload/download
- File updates (live!)
- Needs root access for better function

7.2. File Explorer - Demo

8. Understanding Binder Relationships

- Binder is essentially Android's core
- Binder enables us to have an Object Oriented OS on top of a General Purpose OS
- Problem: there is no way to understand which components talk to each other.


Stock Android Apps
Launcher2
Email
Gallery
Calendar
Calculator
Phone
Settings
Mms
Browser
Contacts
AlarmClock
Camera
DeskClock
Bluetooth
...
Your Apps / Market Apps
App API
android.*
java.* (Apache Harmony)
Binder
System Services
Power Manager
Activity Manager
Package Manager
Battery Service
Mount Service
Notification Manager
Location Manager
Surface Flinger
Status Bar Manager
Sensor Service
Window Manager
...
Zygote (ART, libandroid_runtime.so, Preloaded Classes)
JNI
Libraries
Bionic / zlib / OpenSSL / ...
Hardware Abstraction Layer
Native Daemons
Init / Toolbox
Linux Kernel
Wakelocks / Lowmem / Binder / Ashmem / ION / ...

8.1. Binder Explorer

- In development...
- Analyzes the links between Binder Services and Android applications
- Uses data from `/sys/kernel/debug/binder`
- Pushing further: JSLibBinder

8.2. Binder Explorer - Demo

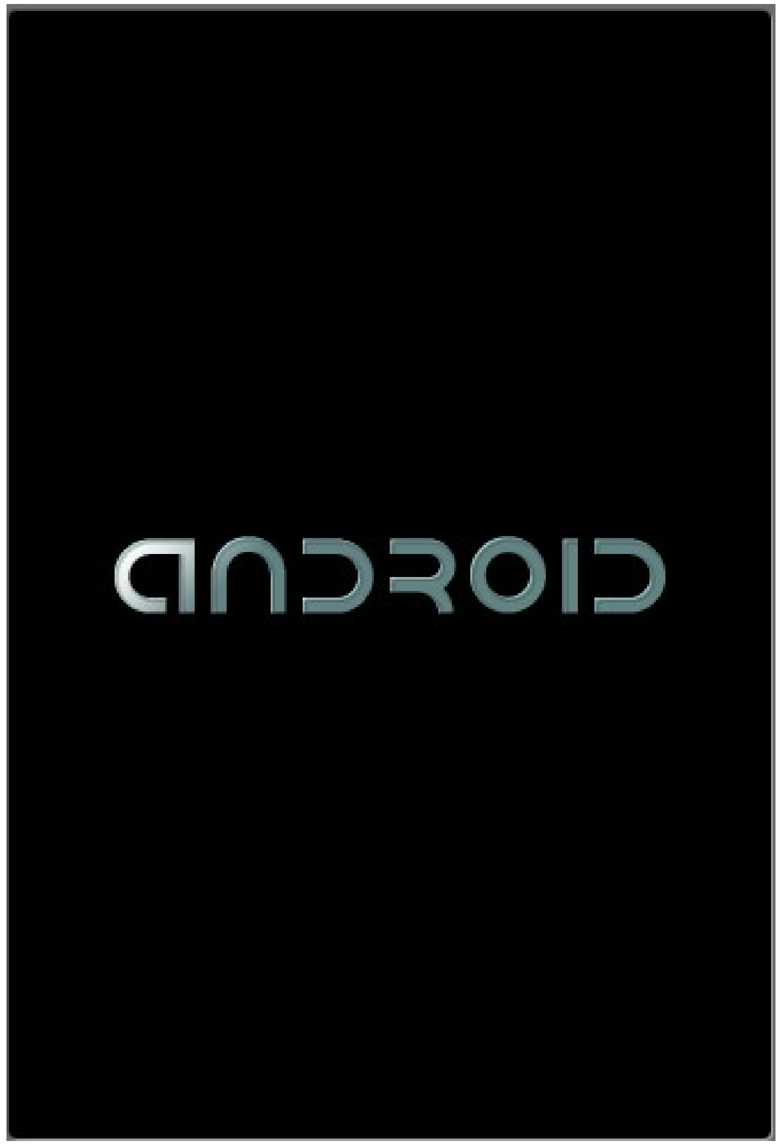
8.3. Reaching Inside Android

- JSLibBinder – libbinder for Android

```
var Binder = require("jslibbinder"),
var sm = new Binder.ServiceManager();
var services = sm.list();
var i = 0;

console.log("Found " + services.length + " services:");
services.forEach(function (s) {
    console.log((i++) + "\t" + s
                    + ": [" + sm.getService(s).getInterface() + "]");
});
```

9. Boot Animation

If Android fails to boot, you see this:



... forever ...

9.1. What do people do today?

- Wait for it to boot ... until it's abnormally long ...
- Manual poking:
  - Shell into device
  - Check processes (ps)
  - Check service (service list)
  - Check logs (logcat)
  - Check kernel logs (dmesg)
- Essentially ... It sucks

9.2. What do we want?

- Foremost: know that boot is failing
- Would be nice to also know:
  - What is failing
  - When it's failing (i.e. as soon as it fails)
  - Why it's failing

9.3. What do we have?

- SurfaceFlinger is one of the very first processes to start
- If SF failing = No boot animation
  - i.e. you know what your problem is already
- If SF comes up, it starts the default boot animation:
  - frameworks/base/cmds/bootanimation/
- Default boot animation has 2 modes of operation:
  - Built-in GL rendering of fixed images
  - Rendering of sequence of static images in ZIP file
- In short: the default boot animation is of no use to us
- Need custom boot animation that reports issues on screen
- Main issues:
  - Surfaces provided by SF are too low level
  - All rendering toolkits assume you've got a fully booted Android

9.4. What did we do?

- Looked for a toolkit that was:
  - Light weight
  - Properly licensed
  - Easily fixable to run on a standalone SF
- Custom version of gameplay3d:
  - Make if work on vanilla SF without the fully-booted framework
- Built different UIs on top of this framework:
  - On-screen split dmesg/logcat
  - On-screen boot progress
- Other UIs could be implemented
- Tangent: gameplay3d can now be used without Android

9.5. Boot Animation - Demo

10. The Road Ahead

- New Features
- New Tools
- Integration:
  - Across our tools
  - With existing tools

- We've got our ideas ;D
- We'd like to hear from you:
  - What are you looking for?

Thank you ...

karim.yaghmour@opersys.com