Accessing Hardware on Android

FTF2014

04/08/2014

Gary Bisson

Embedded Software Engineer

SESSION OVERVIEW

ABOUT THE PRESENTER

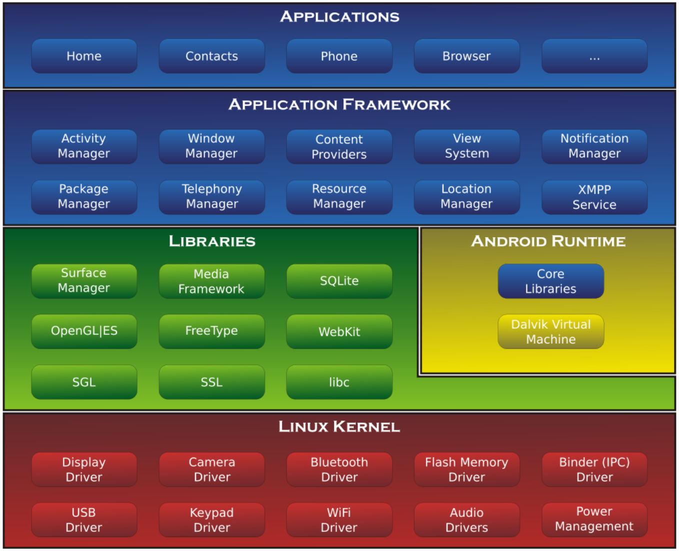
- Embedded Software Engineer at **Adeneo Embedded** (Bellevue, WA)
  - Linux / Android
    - BSP Adaptation
    - Driver Development
    - System Integration
  - Partners with **Freescale**

Introduction

ACCESSING THE HARDWARE

- How different from a GNU/Linux system?
  - No difference for native dev
  - What about Java applications?
- Android Architecture
  - Android API
  - SDK/NDK

ANDROID ARCHITECTURE



ACCESSING THE HARDWARE

Different ways of accessing devices from Android applications:

- Direct access from the application
  - Either in the Java or JNI layer
- Using the available Android hardware API
  - HAL adaptation
- Adding a custom System Service
  - API modification

Native Development

WHAT IS IT?

- Different from JNI/NDK
    - The word “native” in the NDK can be misleading as it still involves all the limitations of Java applications
    - NDK gives you access only to a very limited subset of the Android API
- Native application/daemon/library: can be run directly on the system without the full Java stack

NATIVE APPLICATION

- Can be built statically
  - Avoids libc issues
  - Not preferred solution though
- Can be built against Bionic
  - Every binary/library in Android
  - Some adaptation may be required

BIONIC VS. GLIBC

- **C++**
  - No exception handling!
  - No STL! (Standard Template Library)
- **Libpthread**
  - Mutexes, condvars, etc. use Linux futexes
  - No semaphores
  - No pthread_cancel
- **Misc**
  - No wchar_t and no LOCALE support
  - No crypt()

BUILD A NATIVE APPLICATION

- Such applications can be found in AOSP:
  - `system/core/`
  - `frameworks/base/cmds/`
  - `external/`
- Same as a Java application, an `Android.mk` must be created:

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := hello-world
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := hello-world.cpp
LOCAL_SHARED_LIBRARIES := liblog
include $(BUILD_EXECUTABLE)
```

ADD A NATIVE APPLICATION

- If `LOCAL_MODULE_TAGS` is set as `optional`, the package name must be registered in the device.mk
- Once built, the binary is copied to `<out_folder>/system/bin`
- Modify `init.rc` to start the application at startup:

```
service myapp /system/bin/myapp
    oneshot
```

Direct Access

ACCESSING THE HARDWARE

- Using the user-space interface (*devfs, sysfs*...)
  - Can be done either in Java or in Native C code
  - Simple Open / Read / Write / Close to a “file”
  - Every application that uses a specific hardware must have code to handle it
- The correct permissions must be set
  - The device node shall be opened by all users (not allowed by default) or by the UID/GID of the relevant application(s)
  - `init.rc` or `eventd.rc` must be modified

JAVA SAMPLE CODE

```java
private void turnOnLed () throws IOException {
    FileInputStream fileInputStream;
    FileOutputStream fileOutputStream;
    File file = new File("/sys/class/leds/led_usr1/brightness");
    if (file.canRead()) {
        fileInputStream = new FileInputStream(file);
        if (fileInputStream.read() != '0') {
            System.out.println("LED usr1 already on\n");
            return;
        }
    }
    if (file.canWrite()) {
        fileOutputStream = new FileOutputStream(file);
        fileOutputStream.write('1');
    }
}
```

Android HAL Layer

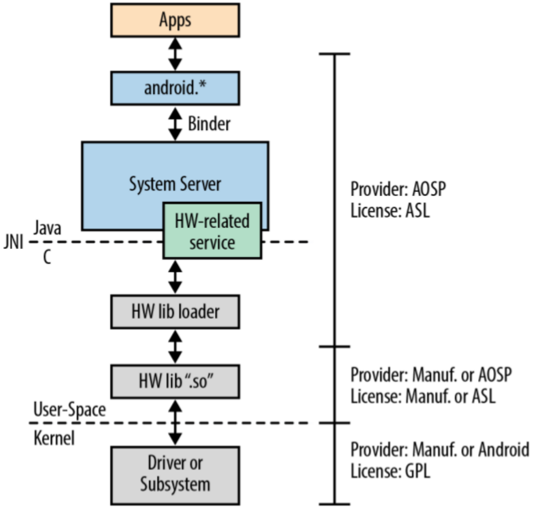
HARDWARE API

- Android Hardware API is accessed through the `android.hardware` class.
- This class only provides support for a limited number of devices such as:
  - Camera: used to set image capture settings, start/stop preview, snap pictures, and retrieve frames for encoding for video
  - Sensors: accelerometer, gyroscope, magnetometers, proximity, temperature...
- OEMs may provide their own HAL implementation to connect to the android hardware API (see *hardware/imx*)

HARDWARE API

- USB Host and Accessory: `android.hardware.usb`:
  - Provides support to communicate with USB hardware peripherals that are connected to Android-powered devices
- Input: `android.hardware.input`
  - Provides information about input devices and available key layouts
  - New in API Level 16 (Jelly Bean)
- Other APIs:
  - For instance the `android.app.Notification` can be used to toggle a LED (if properly registered) with the `FLAG_SHOW_LIGHTS` parameter

HARDWARE ABSTRACTION LAYER (HAL)



LIGHTS LIBRARY

- Interface defined in `hardware/libhardware/include/hardware/lights.h`
- Library must be named `lights.<product_name>.so`
- Will get loaded from `/system/lib/hw` at runtime
- See example in `hardware/imx/lights/`
- Mandatory to have backlight managed by the OS.

CAMERA LIBRARY

- Interface defined in `hardware/libhardware/include/hardware/camera.h`
- Library must be named `camera.<product_name>.so`
- Will get loaded from `/system/lib/hw` at runtime
- See example in `hardware/imx/mx6/libcamera/`

GPS LIBRARY

- Interface defined in `hardware/libhardware/include/hardware/gps.h`
- Library must be named `gps.<product_name>.so`
- Will get loaded from `/system/lib/hw` at runtime
- See example in `hardware/imx/libgps/`

SENSORS LIBRARY

- Interface defined in `hardware/libhardware/include/hardware/sensors.h`
- Library must be named `sensors.<product_name>.so`
- Will get loaded from `/system/lib/hw` at runtime
- See example in `hardware/imx/libsensors/`

EXAMPLE: ADDING A SENSOR

1. Kernel driver must be working and loaded
2. Change directory to `hardware/imx/libsensors`
3. Add Sensor definition into `sSensorList` structure in `sensors.cpp`
   - Applications will now be aware of a new sensor
   - This structure define the following parameters
     - Name
     - Vendor
     - Version
     - Type (Proximity, Temperature etc…)
     - ...

EXAMPLE: ADDING A SENSOR

4. Create object of new sensor
   - Set file descriptor and event type
5. Update `sensors_poll_context_t` structure
6. Add new sensor case to `handleToDriver` function
7. Implement your class:

```
class AccelSensor : public SensorBase {
    int mEnabled;
    int setInitialState();
    public:
    AccelSensor();
    virtual ~AccelSensor();
    virtual int readEvents(sensors_event_t* data, int cnt);
    virtual bool hasPendingEvents() const;
    virtual int enable(int32_t handle, int enabled);
};
```

EXAMPLE: TESTING A SENSOR

- Use existing tool: `hardware/libhardware/tests/nusensors`
  - This binary tool will list every sensor and try to pull data from it
- Use existing java application: `AndroSensor`
  - www.appsapk.com/androsensor
- Create your own application
  - Using `SensorManager`
  - www.vogella.com/tutorials/AndroidSensor/article.html

EXAMPLE: SENSOR MANAGER

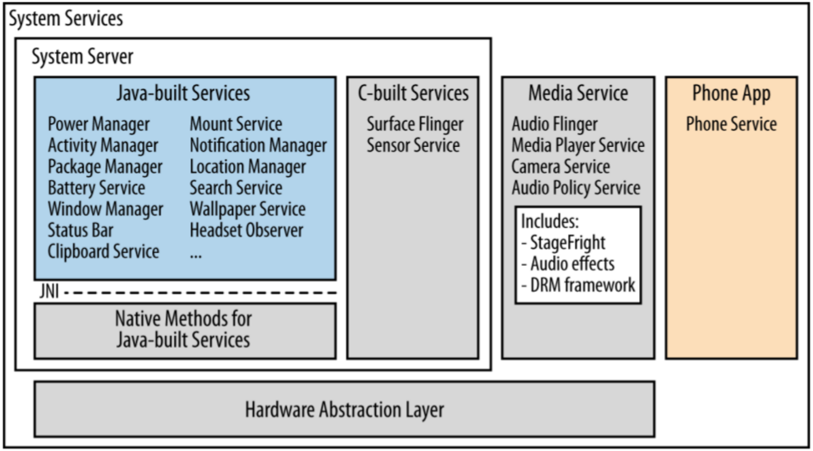
```
public class SensorActivity extends Activity, implements
     SensorEventListener {
    private final SensorManager mSensorManager;
    private final Sensor mAccelerometer;
    public SensorActivity() {
        mSensorManager = (SensorManager)getSystemService(
             SENSOR_SERVICE);
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.
             TYPE_ACCELEROMETER);
    }
    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer,
             SensorManager.SENSOR_DELAY_NORMAL);
    }
    [...]
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
    public void onSensorChanged(SensorEvent event) {}
}
```

Custom System Service

ANDROID SYSTEM SERVICES

- Service: component that performs **long-running operations** in the background and does not provide a user interface
- System Services vs. Local `Service`
  - System Services accessible for all
  - Access through `getSystemService()` method
  - Permissions required

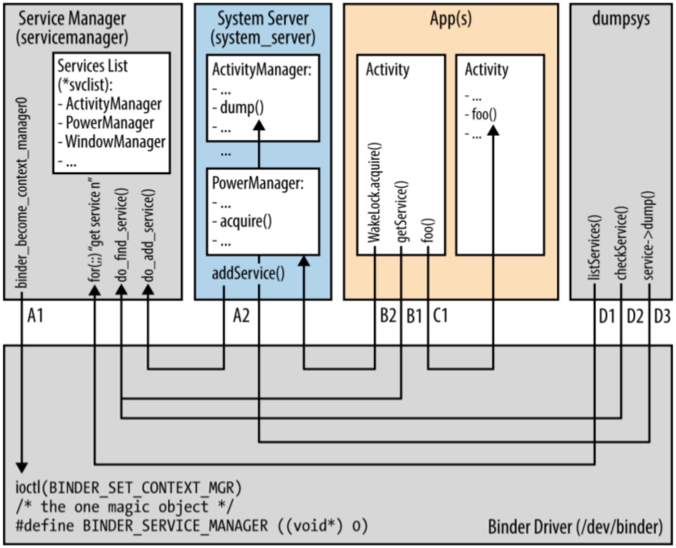
ANDROID SYSTEM SERVICES



MAIN SYSTEM SERVICES

- **System Server**
  - All components contained in one process: `system_server`
  - Mostly made up of Java-coded services with few written in C/C++
- **Media Server**
  - All components contained in one process: `media-server`
  - These services are all coded in C/C++
- Appear to operate independently to anyone connecting to them through **Binder**

ANDROID SYSTEM SERVICES



SERVICE MANAGER

- Service Manager = **YellowPages** book of all services
- Need to register every System Service to be usable
- Can list all services available: `service list`
- Application asks the Service Manager for a handle to the Service and then invokes that service's methods

ADDING A SYSTEM SERVICE

1. Creation of the API layer for the System Service (**aidl**)
    - Defines only exposed methods
    - API added to SDK/add-on
2. Creation of a wrapper class for the Service interface
3. Creation of an implementation of that class
4. Creation of a JNI layer if needed

ADDING A SYSTEM SERVICE

1st approach:

- System Service inside the System Server
- Advantages:
  - Part of the inner system
  - First to be started
  - System permissions
- Drawbacks:
  - SDK creation required

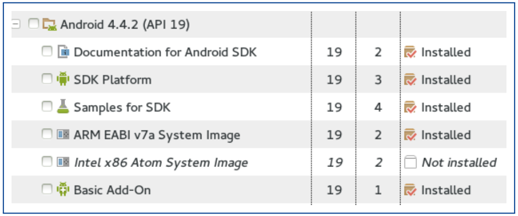
ADDING A SYSTEM SERVICE

2nd approach:

- System Service outside of the System Server
- Advantages:
  - No `framework/` modification
  - Located in one folder
  - Easier to port from one version to another
  - System permissions
  - SDK add-on
- Drawbacks:
  - Considered as a usual App
    - System might remove it in case it runs out of RAM

ADDING A SYSTEM SERVICE

- Example:
  - https://github.com/gibsson/BasicService
  - https://github.com/gibsson/BasicClient
- Although SDK generation is possible, SDK add-on is preferred:
  - https://github.com/gibsson/basic_sdk_addon



Android 4.4.2 (API 19)			
Documentation for Android SDK	19	2	Installed
SDK Platform	19	3	Installed
Samples for SDK	19	4	Installed
ARM EABI v7a System Image	19	2	Installed
Intel x86 Atom System Image	*19*	*2*	*Not installed*
Basic Add-On	19	1	Installed

BASIC SERVICE EXAMPLE

```
BasicService/
├── AndroidManifest.xml
├── Android.mk
└── src/com/gibsson/basic/service/
    ├── app/
    │   ├── BasicServiceApp.java
    │   └── IBasicServiceImpl.java
    └── lib/
        ├── BasicManager.java
        ├── com.gibsson.basic.service.lib.xml
        └── IBasicService.aidl
```

AIDL EXAMPLE

```
/**
 * System-private API for talking to the BasicService.
 *
 * {@hide}
 */
interface IBasicService {
  int getValue();
  int setValue(int val);
}
```

WRAPPER CLASS EXAMPLE

```
public class BasicManager {
  private static final String REMOTE_SERVICE_NAME = IBasicService
        .class.getName();
  private final IBasicService service;

  public static BasicManager getInstance() {
    return new BasicManager();
  }

  private BasicManager() {
    this.service = IBasicService.Stub.asInterface(ServiceManager.
        getService(REMOTE_SERVICE_NAME));
    if (this.service == null) {
      throw new IllegalStateException("Failed to find
          IBasicService by name [" + REMOTE_SERVICE_NAME + "]");
    }
  }
  [...]
}
```

IMPLEMENTATION EXAMPLE

```java
class IBasicServiceImpl extends IBasicService.Stub {
  private final Context context;
  private int value;

  IBasicServiceImpl(Context context) {
    this.context = context;
  }
  protected void finalize() throws Throwable {
    super.finalize();
  }
  public int getValue() {
    return value;
  }
  public int setValue(int val) {
    value = val + 4;
    return 0;
  }
}
```

IMPLEMENTATION APP EXAMPLE

```
public class BasicServiceApp extends Application {
  private static final String REMOTE_SERVICE_NAME = IBasicService
      .class.getName();
  private IBasicServiceImpl serviceImpl;

  public void onCreate() {
    super.onCreate();
    this.serviceImpl = new IBasicServiceImpl(this);
    ServiceManager.addService(REMOTE_SERVICE_NAME, this.
        serviceImpl);
  }

  public void onTerminate() {
    super.onTerminate();
  }
}
```

Demonstrations

HARDWARE SELECTION

- i.MX6Q SabreLite
- Android 4.3 Jelly Bean
- 10" LVDS display
- Could be any other device

DEMONSTRATIONS

- Demonstration #1
  - Native app access
- Demonstration #2
  - Direct JNI access
- Demonstration #3
  - Using Sensor API
- Demonstration #4
  - Custom System Service

Conclusion

CONCLUSION

- Direct access from application
  - Permission issue
- HAL modification
  - Only few hardware targeted
- Adding a System Service
  - Most complex but elegant way
- Solution depends on constraints

QUESTIONS?

REFERENCES

- Karim Yaghmour: **Embedded Android**
  http://shop.oreilly.com/product/0636920021094.do
- Karim Yaghmour: **Extending Android HAL**
  http://www.opersys.com/blog/extending-android-hal
- Marko Gargenta: **Remixing Android**
  https://thenewcircle.com/s/post/1044/remixing_android
- Lars Vogel: **Android Sensors**
  http://www.vogella.com/tutorials/AndroidSensor/article.html