Hi, I am Cédric, I work for Genymobile as a System Engineer. Genymobile is a company specialized in Android. We are based in France (Paris and Lyon) and SF. We develop and customize Android ROM for our customers. We also have our own products like Genymotion (android emulator, you may have heard of it).

Today I’d like to talk about how to customize a Android system.
Let's see what problem we want to solve
Android is a “full stack” OS

How to use my own hardware?

Android is a full operating system. It come with a SDK to build apps. Every hardware modules can be accessed with a coherent Java API (eg: camera, gps, sensors) Everything is protected by a Permission mecanism

Very convenient for application developer.

As a linux developper, I’d like to port my own hardware to Android. Eg: board with a serial port, gpio, ...
Let’s see how to customize android.
We start from the kernel and go all the way up to the app!

In this presentation we will follow what is done in AOSP.
This mean changing google code.
This is the easiest way, however this can bring problems when we want to port to another android version.

Some other approach need less change in AOSP. To understand how the layers are put together we will keep this method.
Summary

01 Kernel
02 Hal
03 Jni
04 System Service
05 Framework
06 App
01 KERNEL
Kernel

Not part of AOSP

Driver: Built-in or Module

GPLv2
Every file that is used to build and customize your device goes to "device". This module will create a "character device". We do not want to give access to the device to every application; we restrict access to the system user. We also do not want our app to run as system.
No demo with real hardware
Our “device” is a simple module that convert upper case to lower case

We want to protect access to the device, only “system” is allowed to r/w. Of course our application will not have system permission.

We need the glue to let system_server use it
Kernel

APPLICATION FRAMEWORK
BINDER IPC PROXIES
ANDROID SYSTEM SERVICES

MEDIA SERVER
- AudioFinger
- Camera Service
- MediaPlayer Service
- Other Media Services

SYSTEM SERVER
- Search Service
- Activity Manager
- Window Manager
- Other System Services & Managers

HAL
- Camera HAL
- Audio HAL
- Graphics HAL
- Other HALs

LINUX KERNEL
- Camera Driver
- Audio Driver (ALSA, OSS, etc.)
- Display Drivers
- Other Drivers
HAL: Hardware Abstraction Layer
Hal
Hardware Abstraction Layer

C library

Expose hardware feature

Part of AOSP, often closed source
With this hardware, I want to get some data (abs_getdata), put new data (abs_putdata) and clear the buffer (abs_clear). This is here that you will put your device specific code. This code is not android specific.
System server run java code.
We need a bridge between java (system_server) and C (hal) => JNI
JNI: Java Native Interface
Jni
Java Native Interface

Simple glue between C and Java

Do not do smart thing here
Expose your hal function through jni
Manage error code with exceptions
Only glue code
package android.abs;

public class Abs {
    static {
        System.loadLibrary("abs_jni");
    }

    public native static void clear();
    public native static String getData() throws AbsException;
    public native static void putData(String in) throws AbsException;
}
package android.abs;

/** @hide */
public class Main {
    public static void main(String[] args) {
        try {
            Abs.putData("Hello ABS");
            String out = Abs.getData();
            System.out.println(out);
            Abs.clear();
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}

Trick: Add a Main.java
Trick: Add a Main.java

$ dalvikvm -cp /system/framework/framework.jar android.abs.Main

Allow you check that everything works from java to the device
Jni
Java Native Interface
System server is called by the application (framework) through the binder protocol
System server run as the “system” user
The binder RPC allow us to check the permission of the caller live into service.jar
System Server

Hack SystemServer.java

```java
private void startOtherServices() {
    ... try {
        Slog.i(TAG, "Abs Service");
        absService = new AbsService(context);
        ServiceManager.addService(Context.ABS_SERVICE, absService);
    } catch (Throwable e) {
        reportWtf("starting abs Service", e);
    }
}
```

start our service
Note that we decide that our device will be manageable by system_server. However we could have created a special daemon to deal with the device and let system_server talk to the daemon. The daemon could run as root. This is another choice of architecture but do not change much.
Framework
package android.abs;

public class AbsManager {
    IAbsManager mService;
    /** @hide */
    public AbsManager(IAbsManager service) {
        mService = service;
    }

    public String getData() {
        try {
            return mService.getData();
        } catch (RemoteException e) {
            ...
            return null;
        }
    }
}
static {
    ...
    registerService(ABS_SERVICE, new ServiceFetcher() {
        public Object createService(ContextImpl ctx) {
            IBinder b = ServiceManager.getService(ABS_SERVICE);
            IAbsManager service = IAbsManager.Stub.asInterface(b);
            return new AbsManager(service);
        }
    });
}
Framework

make update-api && make sdk

Configure IDE
06 App
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {
    private ActionBar ab;
    
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ab = getActionBar();
        if (ab != null)
            ab.setDisplayHomeAsUpEnabled(true);

        Button btn = (Button) findViewById(R.id.button1);
        btn.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent intent = new Intent(v.getContext(), SecondActivity.class);
                startActivity(intent);
            }
        });
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == R.id.action_settings)
            return true;
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onBackPressed() {
        if (ab != null)
            ab.popBackStack(null, 0);
        else
            super.onBackPressed();
    }
}

Conclusion
Conclusion
Conclusion

Easy to add new driver and expose an “Android API”

Most of the kernel and HAL is reusable

Lot of Glue Code
Conclusion

https://thenewcircle.com/s/post/1044/remixing_android


https://source.android.com/devices/
Thank you!

Cédric Cabessa
ccabessa@genymobile.com
www.genymobile.com

https://github.com/CedricCabessa/abs2015
http://goo.gl/nDVszZ