



Evolution of an Android OTA management application

Diego Rondini - Kynetics

About Kynetics

Kynetics is company that provides:

- OSes for embedded systems
 - Android based OSes
 - Open Embedded (Yocto) based OSes
- Over the Air (Wire) technology (Update Factory) for remotely update embedded devices based on Eclipse hawkBit.
 - Cloud Based Management platform
 - OS provisioning for Development and Production
 - Android Apps

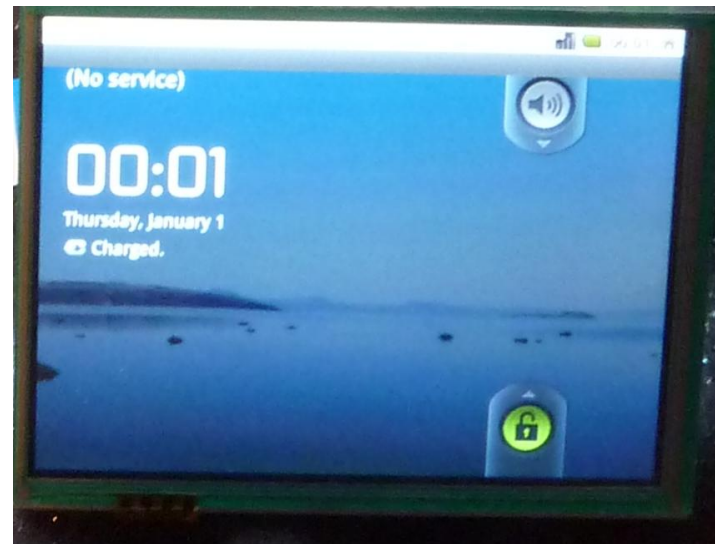
Offices in Padova and Santa Clara (CA).

Member of the Eclipse Foundation.

About me

Diego Rondini

- Embedded Engineer
- started doing embedded Android 1.5 / 1.6 in 2011
- experience with Linux Yocto / Openembedded



Agenda

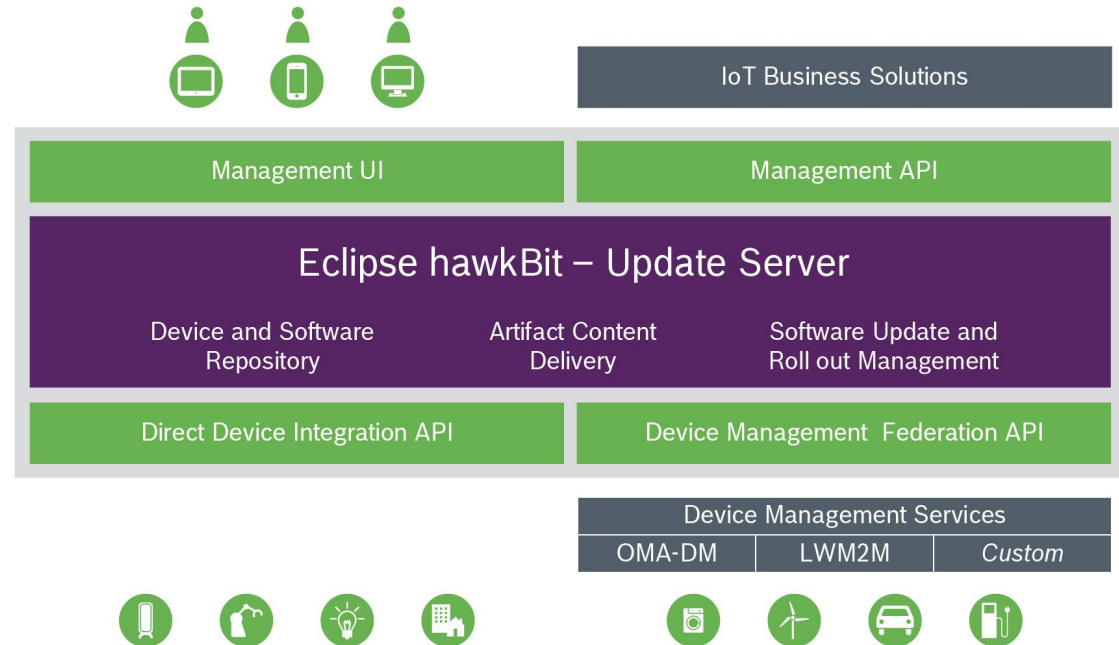
Android OTA management application

1. Brief history
2. Overview
3. Interfacing with Android AOSP system APIs

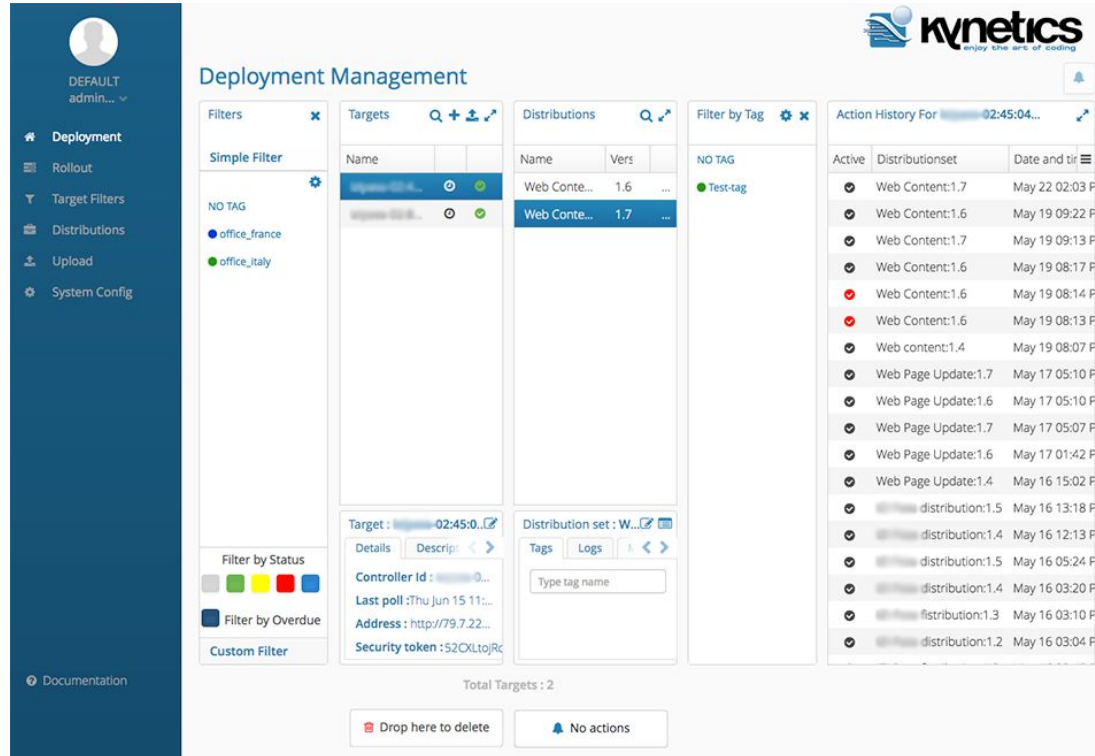
1. Brief history

Remote Update Management Platform

Embedded Linux
Conference 2017



Eclipse hawkBit: Management UI



Deployment Management

Filters

Simple Filter

NO TAG

- office_france
- office_italy

Filter by Status

Filter by Overdue

Custom Filter

Targets

Name	Status
office_france	Success
office_italy	Success

Target : [ID] 02:45:0...

Controller Id : [ID]...

Last poll : Thu Jun 15 11:...

Address : http://79.7.22...

Security token : 52OXLtoRk

Distributions

Name	Vers
Web Conte...	1.6
Web Conte...	1.7

Distribution set : W... [ID]...

Tags

Logs

Type tag name

Filter by Tag

NO TAG

Test-tag

Action History For [ID] 02:45:04...

Active	Distributionset	Date and time
✓	Web Content:1.7	May 22 02:03 P
✓	Web Content:1.6	May 19 09:22 P
✓	Web Content:1.7	May 19 09:13 P
✓	Web Content:1.6	May 19 08:17 P
✗	Web Content:1.6	May 19 08:14 P
✗	Web Content:1.6	May 19 08:13 P
✓	Web content:1.4	May 19 08:07 P
✓	Web Page Update:1.7	May 17 05:10 P
✓	Web Page Update:1.6	May 17 05:10 P
✓	Web Page Update:1.7	May 17 05:07 P
✓	Web Page Update:1.6	May 17 01:42 P
✓	Web Page Update:1.4	May 16 15:02 P
✓	[ID] distribution:1.5	May 16 13:18 P
✓	[ID] distribution:1.4	May 16 12:13 P
✓	[ID] distribution:1.5	May 16 05:24 P
✓	[ID] distribution:1.4	May 16 03:20 P
✓	[ID] distribution:1.3	May 16 03:10 P
✓	[ID] distribution:1.2	May 16 03:04 P

Total Targets : 2

Drop here to delete

No actions

Embedded update management: Android

Built-in features (integrated by default):

	Android AOSP	Yocto Linux
Update file format	✓	✗
Update installation system (Recovery / Update Engine)	✓	✗
Download of update	✗	✗

On the device side

Linux:

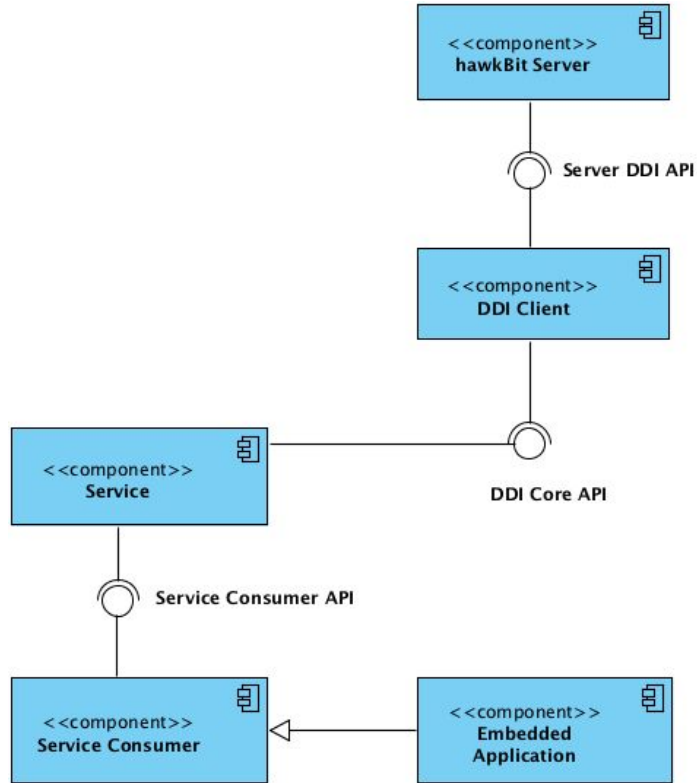
- SWUpdate
- RAUC

Android:

- No library to communicate using hawkBit DDI API
- No existing open source Android application

2. Overview

Architecture components



Library

1. Reusable
2. Independent from Android
3. Manages hawkBit DDI communication
4. [2017] Initially written in Java
5. [2020] Rewritten in Kotlin
6. [2021] Contributed as Eclipse Hara “hara-ddiclient”

Eclipse Hara

- Fill the gap that was intentionally left out by the hawkBit project
- Provide an open source reference implementation of an hawkBit DDI client
 - Define architecture components and their IPC
- Develop hawkBit clients for different frameworks, OSes and architectures
 - Android
 - Linux pure Java
- EPLv2 license

<https://www.tldrlegal.com/license/gnu-general-public-license-v2>



HARA

Android Update Service

Implements an Android OTA management application

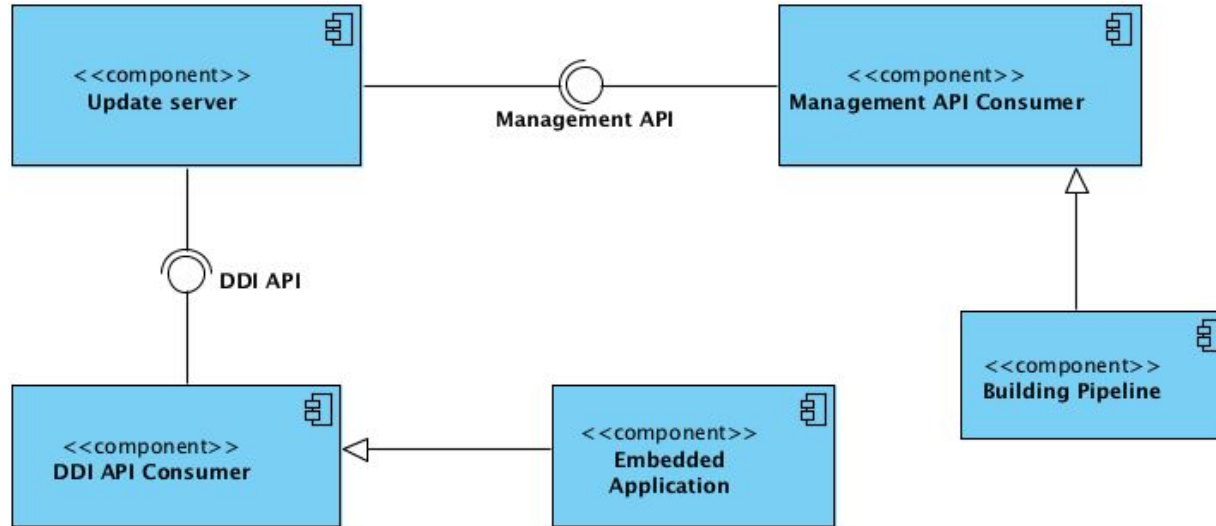
- background service
- communicates using hawkBit DDI API
- manages state machine
- handles installation of different type of updates (applications, OSes)
- offers integration with third-party applications via APIs
 - configuration
 - state
 - notifications
- EPLv2 license

<https://www.tldrlegal.com/license/gnu-general-public-license-v2>

The Device drives the Update Logic

- How should the devices react to connection timeouts/pools in terms of user experience?
- What if the board has an outage during an update?
- Can I handle single image update?
- What about A/B updates?
- How should the device orchestrate effectively, the server hawkBit states of an entire update cycle? (Device action feedbacks)
- How should the device handle the update of the **client software** itself?
- What if an update artifact is malformed?
- What if an update artifact is not signed correctly..
- Soft updates vs Forced updates
- ...

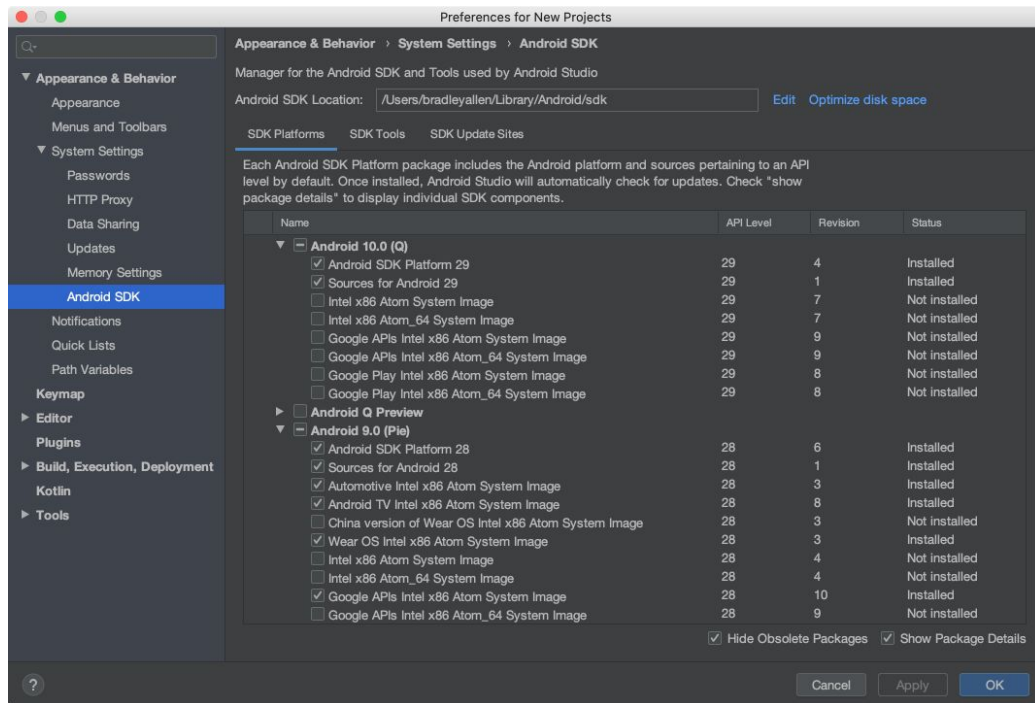
Remote OTA in a nutshell



3. Interfacing with Android system APIs

Android SDK: user perspective

- Android Studio IDE
- Emulator
- APIs
- debugging tools



Android SDK: OS perspective

- Android SDK APIs use the background services provided by the `system_server`
- SDK can be generated by the AOSP buildsystem
- customize SDK to expose new or modified APIs

Android apps can access **public but restricted system APIs** by being installed in the OS (vendor partition) or by being signed with the platform key:

https://www.kynetics.com/docs/2018/Accessing_Android_system_APIs/

Android apps can access **non-public system APIs** by using a modified SDK (`android.jar`).

UF Android Update Service

UF Update Service is an **Android application** that manages:

- App Apk updates
 - **PackageManager** APIs:
<https://developer.android.com/reference/android/content/pm/PackageManager>
- OS single copy updates
 - **RecoverySystem** APIs:
<https://developer.android.com/reference/kotlin/android/os/RecoverySystem>
- OS double copy A/B updates
 - **UpdateEngine**:
<https://android.googlesource.com/platform/frameworks/base/+refs/heads/android11-release/core/java/android/os/UpdateEngine.java>

Installing apks with PackageManager

- Support for installing and updating apks
- Use of Android PackageManager

<https://github.com/Kynetics/uf-android-client/tree/v1.4.1/uf-client-service/src/main/kotlin/com/kynetics/uf/android/update/application>

We use `android.content.pm.PackageManager` for:

1. getting metadata from an apk (version, package name)
2. install or update an apk (`android.content.pm.PackageInstaller`)
3. handle installation errors

OTA updates

Devices need OS updates to:

- provide new features
- fix bugs
- fix security issues

While nowadays “OTA” is used for any kind of updates, the terms originally referred to “Over The Air” updates, so updates distributed over internet.

Single Copy

- A single copy of the system is present
- An independent bootable system is required to manage the update
- Cooperation with the bootloader is necessary to boot in recovery mode
- Update client downloads the update artifact in a separate partition
- System is rebooted into recovery mode and the update is installed

Double Copy

- Each slot (set of partitions) is a full copy of the whole OS
- A slot should be able **to boot, run, and update the inactive copy.**
- A bootable slot that was not marked as successful (after several attempts were made to boot from it) should be marked as *unbootable* by the bootloader, including changing the active slot to another bootable slot (normally to the slot running immediately before the attempt to boot into the new, active one).

<https://source.android.com/devices/tech/ota/ab>

Device Update Approaches

- Double copy:
 - The device features two copies of the Application/OS/RootFS
 - Cooperation with bootloader to decide which copy should be booted
- + fallback
- + easy to implement
- - takes double space
- Single copy:
 - A separate upgrade OS is required
 - Cooperation with the bootloader to boot in update mode
- + takes little space
- - no fallback

Installing single-copy OTA with RecoverySystem

- Support for installing single-copy Android OTA
- Installation initialized from main Android, but happens in Android Recovery

<https://github.com/Kynetics/uf-android-client/blob/v1.4.1/uf-client-service/src/main/kotlin/com/kynetics/uf/android/update/system/SingleCopyOtaInstaller.kt>

We use `android.os.RecoverySystem` to:

1. verify signature of OTA package
2. start installation of OTA package

Installation success is managed by checking Recovery log files

Installing double-copy OTA with UpdateEngine

- Support for installing double-copy Android OTA (Android \geq 8.x Oreo)
- Installation happens “live” from main Android
- A \rightarrow B switch happens at next reboot

<https://github.com/Kynetics/uf-android-client/blob/v1.4.1/uf-client-service/src/main/kotlin/com/kynetics/uf/android/update/system/ABOtaInstaller.kt>

We use `android.os.UpdateEngine` for:

1. start installation of OTA package
2. monitor installation progress
3. handle installation errors

System applications / Hidden APIs

PackageManager & RecoverySystem require system permissions:

“Requires the android.Manifest.permission#REBOOT permission. **Not for use by third-party applications.**”

public but restricted APIs

non-public APIs

/**

- * UpdateEngine handles calls to the update engine which takes care of A/B OTA updates. It wraps up the update engine Binder APIs and exposes them as SystemApis, which will be **called by the system app responsible for OTAs.**
- * On a Google device, this will be GmsCore.

*/

Android hidden-API vs os-mock

Android Hidden APIs project

Build of Android SDK' `android.jar` that exposes internal and hidden APIs:

<https://github.com/anggrayudi/android-hidden-api>

CON: needs installing **custom file in Android SDK** folder

OS Mock library

<https://github.com/Kynetics/uf-android-client/tree/v1.4.1/os-mock>

1. mocked the UpdateEngine hidden Android APIs
2. used **os-mock library only at build time**
3. UF Android Update Service uses Android framework at runtime

Thank you.

Some links:

- Eclipse Hara
 - <https://github.com/eclipse/hara-ddiclient/>
 - <https://projects.eclipse.org/projects/iot.hawkbit.hara>
- Update Factory:
 - <https://github.com/Kynetics/uf-android-client>
 - <https://docs.updatefactory.io/>
- Kynetics website: www.kynetics.com
- Kynetics Open source projects: <https://github.com/kynetics>
- Eclipse hawkBit: <https://www.eclipse.org/hawkbit/>