
arquillian

Arquillian testing platform support for Android web and native applications

Bc. Štefan Miklošovič

FI MUNI 6/2013

Testing of Java EE applications

- How to be sure isolated and complicated enterprise services play smoothly together once they are integrated?
  - Injection of services into application
  - Entity managers
  - Logging
  - Failure scenarios
  - Security threats

Testing of Java EE applications

- Traditional techniques
  - xUnit
  - Mocking
    - Mockito
- Problems
  - how to mock everything in AS context?
  - some services and functionality can be simulated only with difficulties, if at all
  - need for something *real*

Identified problems

- Application is not tested sufficiently for real target environment, only in simulated and faked one
- We can not cover all possible scenarios manually to be sure application behaves correctly under the stress
- Flaws and bugs of any kind have to be spotted and repaired before they are exposed in real runtime

Arquillian core principle

Arquillian brings tests into runtime,

not runtime into tests.

Arquillian test

- just “normal” **JUnit / TestNG** like test
- Own test runner
- **Injections** into test class are enriched on container side **after deployment**
- Specification of **what to deploy** into container is done **via ShrinkWrap**

Arquillian architecture

Container abstraction

- **central abstraction of Arquillian**
- specifies container lifecycle
  - what does it mean?
- controls test deployment into container
- types
  - managed container
  - remote container

Arquillan Android container

Android device* is also some kind of container.

* emulator or mobile phone (tablet)

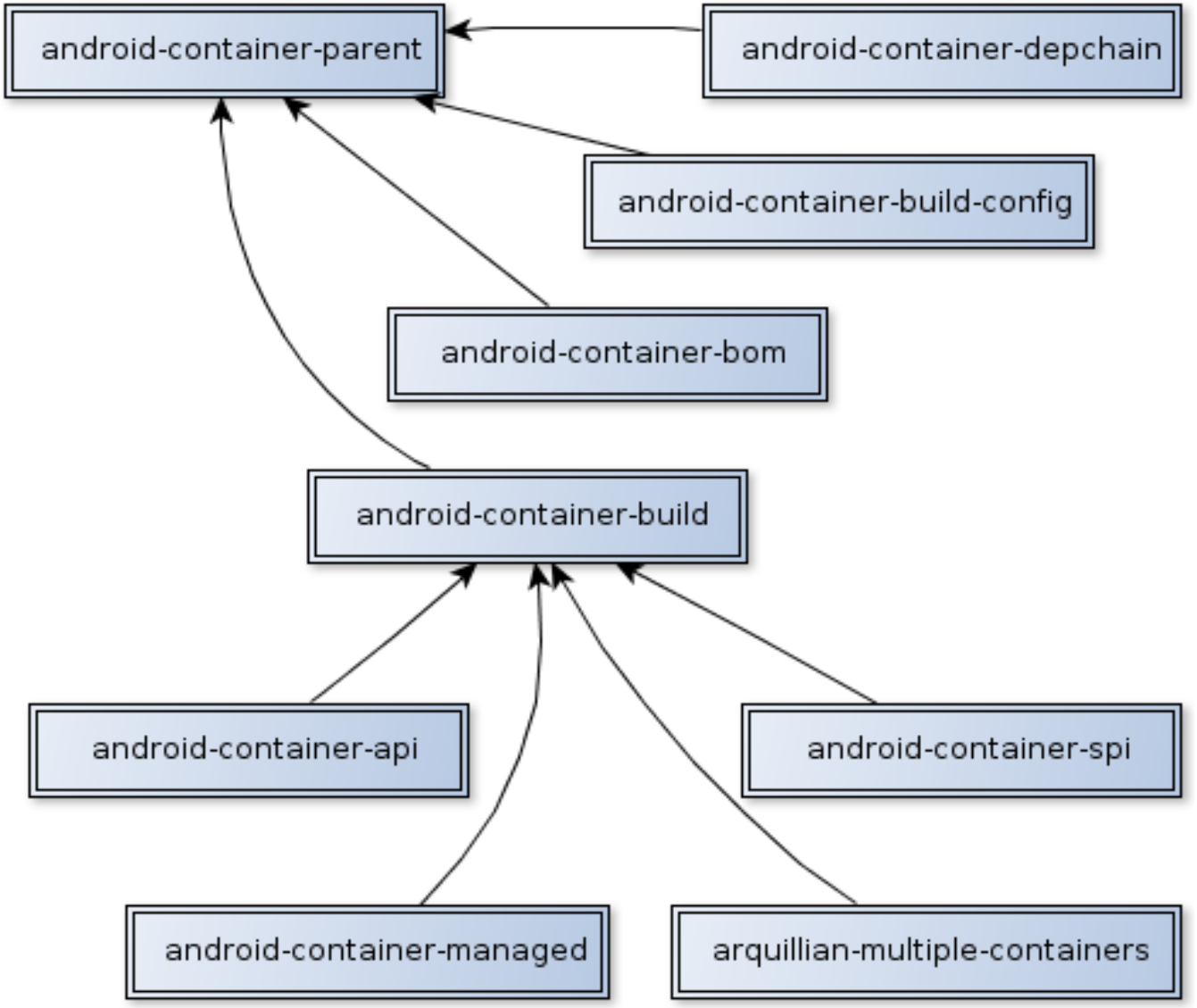
Arquillan Android container

- Android device is also some kind of container
- Managed or remote?
  - Both!
- Supported device types
  - emulator or real device (mobile phone)
- Multiple devices in one test
- Multiple containers in one test

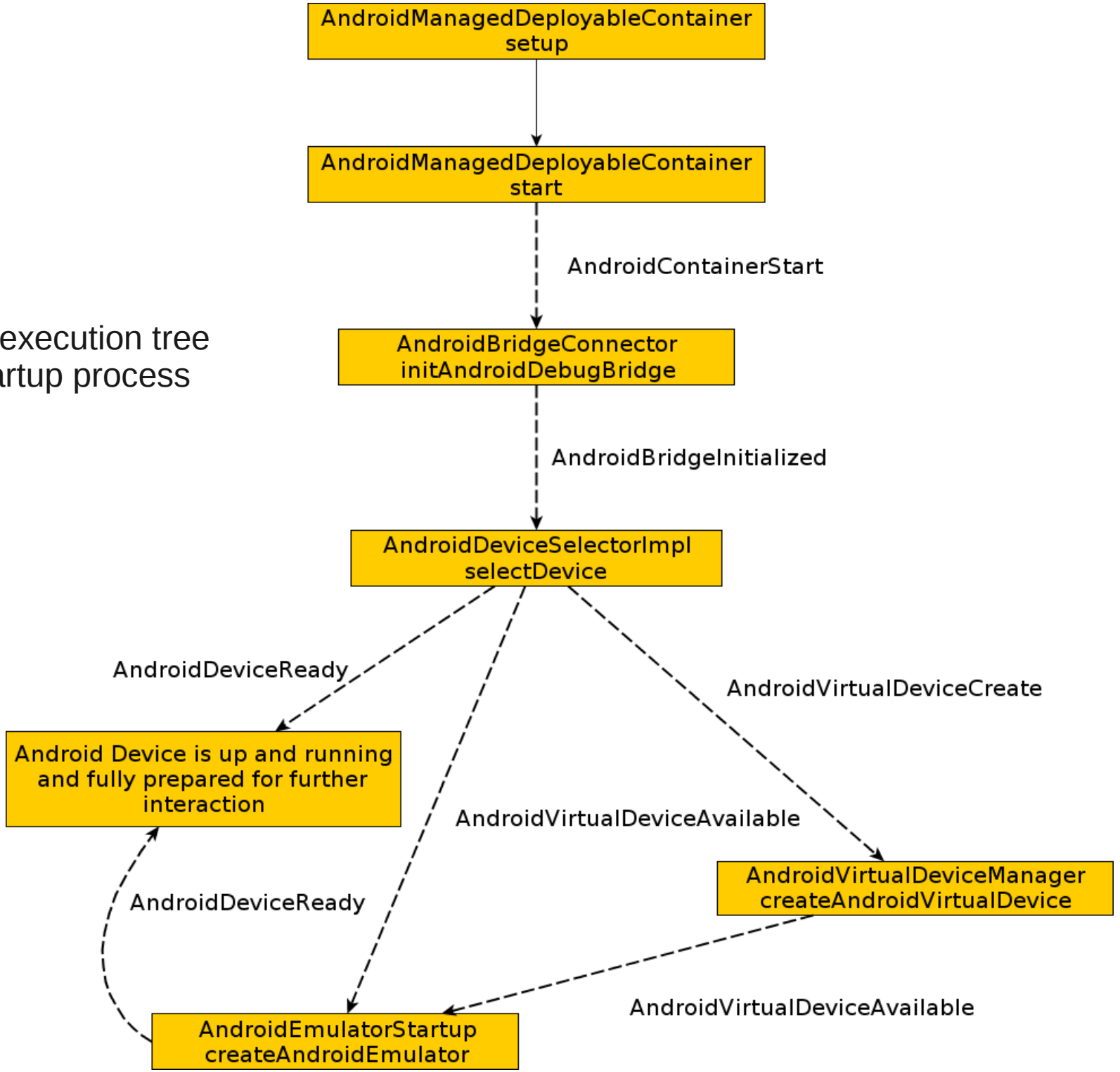
Arquillan Android container

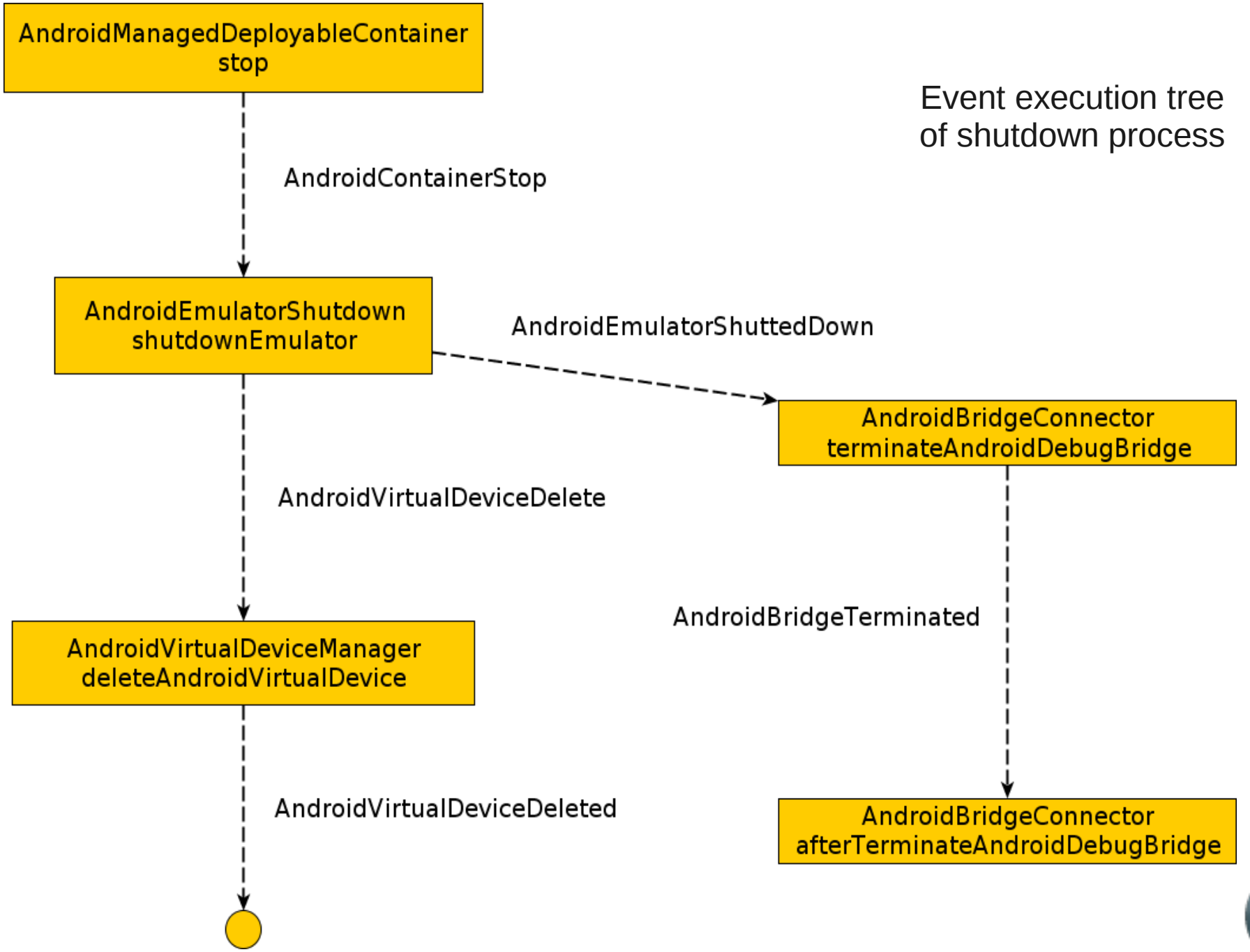
- Android container is testing-approach agnostic
  - how to support both native and web testing?
- How to support both types of testing
  - by extensions plugged into container once on classpath
- Implements container SPI and API just as any other “web” container
- container execution flow is event driven
  - observers and firing of events
  - advantages?
    - extensionability

Android container - Maven architecture



Event execution tree of startup process





Event execution tree
of shutdown process

Device management

- Possibility to start stopped AVD (emulator)
  - managed container mode
- Possibility to connect to already started device
  - remote container mode
    - emulator
    - physical device
- Possibility to generate whole AVD from scratch automatically upon every test execution
  - deleted after tests

Focus on functional testing

- Why functional testing?
- Existing solutions and tools
  - Selenium WebDriver
  - Arquillian Graphene
- What to test?
  - **Web applications**
    - Arquillian Android for web plugin
  - **Native Android applications**
    - Arquillian Android native plugin

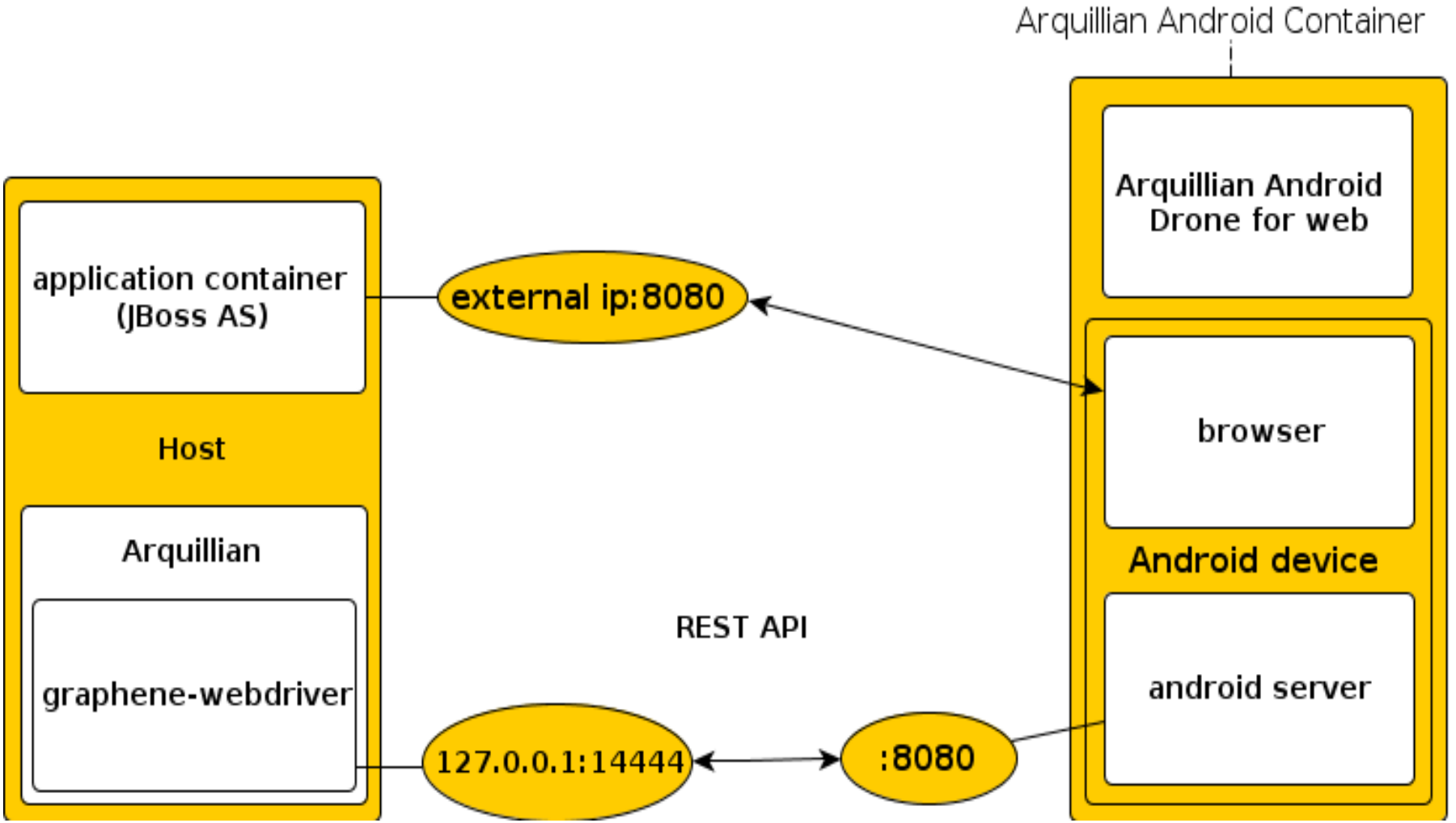
Arquillian Android web plugin

- Web application is deployed to AS
  - war, ear
- Arquillian Android container starts
  - Application container
  - Android container
- Developer writes JUnit-like functional tests via WebDriver API

Arquillian Android web plugin

- extension into Android container
- standalone project
- tests web applications
- Uses Arquillian Graphene for injections of WebDriver interface

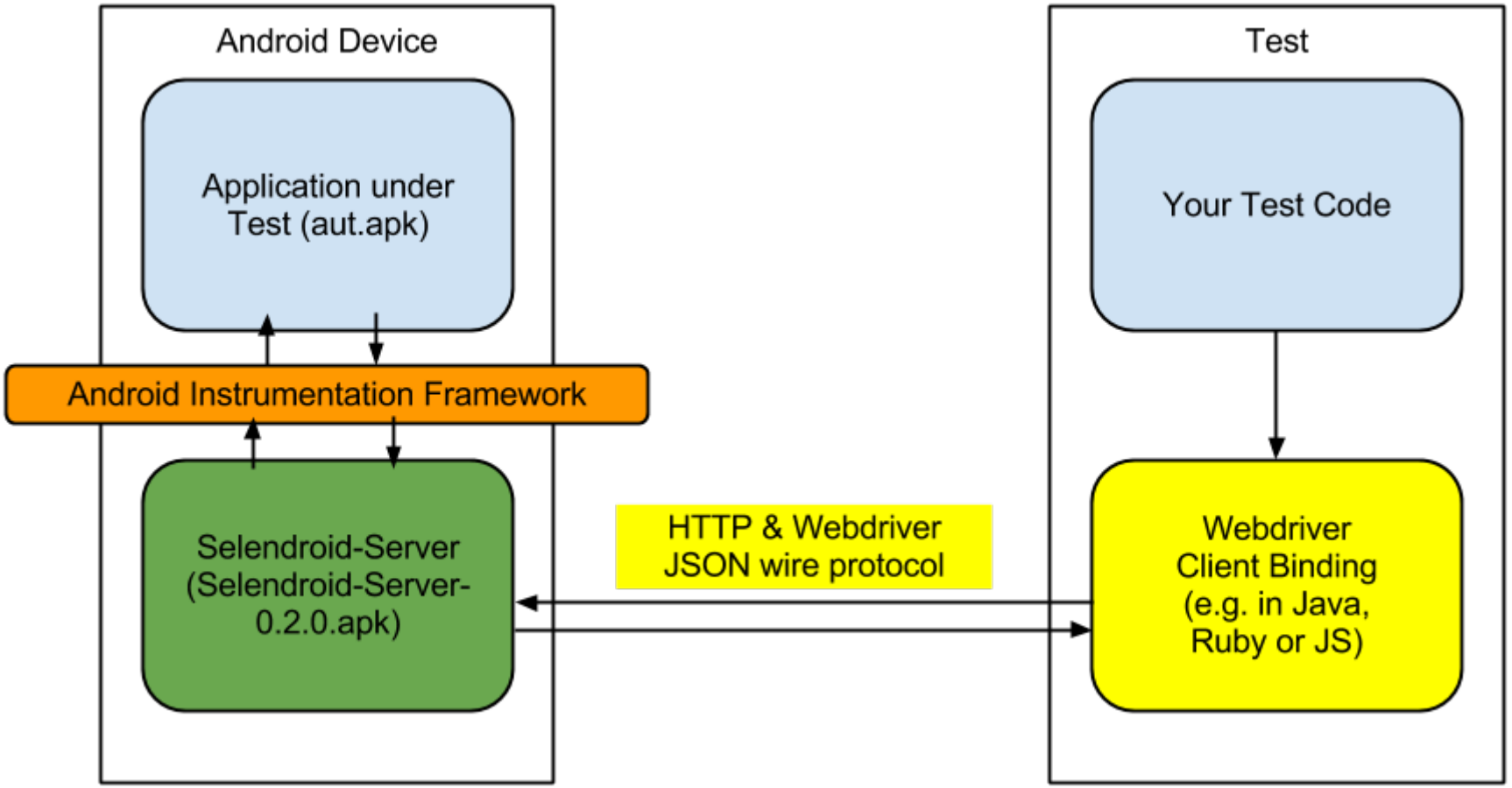
Architecture of Android container testing of web applications



Arquillian Android native plugin

- Motivation
- Uses only Android container
- APK deployed via ShrinkWrap
  - why is this possible without APK support?
- Uses Selendroid server APK
- Selendroid Server instruments application under test
- Implements WebDriver API as Selenium / Arquillian Graphene does

Selendroid



- selendroid

Arquillian Android native plugin

- Controls
  - deployment & undeployment of AUT
  - resigning of APKs
  - port forwarding
  - instrumentation execution after deployment
  - Android device with Selendroid server and installed AUT prepared to handle WebDriver REST calls from Arquillian Graphene / WebDriver provider

Why is it so important?

- **Automatization of functional testing for mobiles**
- **Excellent for continuos integration model**
- **Testing *is* developing**
- **Brings Test Driven Development and eXtreme Programming into mobile development**
- Supports fast turn-around of test execution
- Supports multiple devices in one test
- Supports multiple container adapter implementations

Google Summer of Code 2013

- Project was selected for **GSoC 2013**
- Recognition of **Masaryk University** on global scale
- Team cooperation with other Russian student
- Goals
  - tight integration with Android platform as such
    - enhance user experience
    - zero configuration effort
    - testing out of box
    - arquillian-droidium-platform-support

(near) future of the project

- Renaming of the container + extensions
  - Arquillian Droidium
    - An**droid** + Selen**ium**
- Bringing project to main Maven repositories
- Red Hat & JBoss branding
- Implementing integration with Apple products
  - iPhone, iPad
- Possible integration into existing IDEs
  - JBoss Developer Studio

Questions

my public repository with project:

arquillian-container-android

official repositories created few days ago:

arquillian-droidium

How do you deploy with just Maven?

Answer:

- e.g. with jboss-as-maven-plugin
  - deploy, re-deploy, undeploy the application

What are you testing against with pure Selenium?

Answer:

- Selenium IDE
  - Plugin to Firefox, we are writing tests in domain specific language (Selenese) directly in that extension

How do you use just ShrinkWrap?

Answer:

- 1) It is very handy to do any modifications of archives, JAR, WAR, EAR
- 2) It is able to send archive (and its specific content) e.g. via network in a serializable form
- 3) Explode an archive to a file or exploded directory structure

Proof-of-concept web test

```
public class MobileTestCase {
    // build your application you want to test dynamically
    @Deployment(name = “myDeployment”)
    @TargetsContainer(“jboss-as”)
    public static Archive<?> getDeployment() {
        // return e.g. WAR archive with application
    }

    @Drone Webdriver driver;

    @Test @OperatesOnDeployment(“myDeployment”)
    public void test01() {
        driver.findElement(By.id("buttonTest"));
        // click button and see what happens ...
    }
}
```