Arquillian testing platform support for Android web and native applications

Štefan Miklošovič

Terminology

- Enterprise Application Server
  - JBoss AS, EAP, GlassFish
- Deployment
- Maven
- Jenkins
- Continuos integration

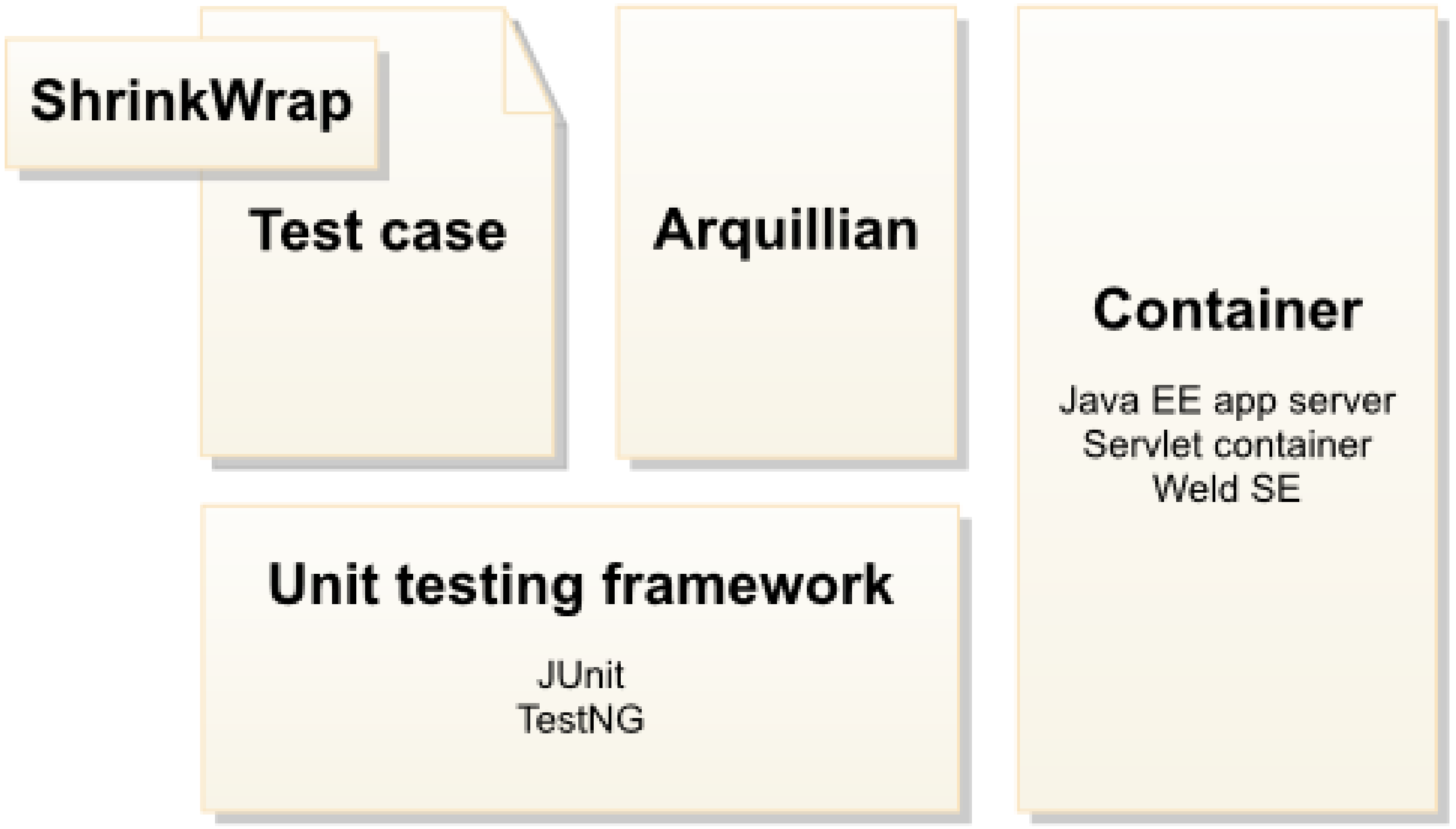
Testing of Java EE applications

- JUnit
- Mocking
  - Mockito
- Problems
  - how to mock everything in AS context?

Arquillian overview

- Brings test into runtime, not otherwise
  - what does it mean?
- Simulates how application behaves when deployed in target environment
- No more mocking and faking of services
- Supported types of testing
  - Integration testing
  - Behavioral testing
  - **Functional testing**

Arquillian architecture



ShrinkWrap

- library for building archives on the fly
- specifies, isolates and packages exactly what we want to test
- Resources for test picked from project itself
- Archive support
  - jar, war, ear
- Used directly in test class

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
- configuration in arquillian.xml in src/test/resources

arquillian.xml

- Place where container and extensions are configured

```
<arquillian>
    <container qualifier=”jboss-as”>
        <configuration>
            // configuraiton of jboss as container adapter
        </configuration>
    </container>
</arquillian>
```

Container abstraction

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

```
@RunsWith(Arquillian.class)
public class SomeTestCase {

    @Deployment(name = “myDeployment”)
    @TargetsContainer(“jboss-as”)
     public static Archive<?> getDeployment() {
         return ShrinkWrap.createFromZipFile(
             “mywebapp.war”);
     }

     // tests
}
```

```
@RunsWith(Arquillian.class)
public class SomeTestCase {
    @Deployment ...
    // injection from application server
    @Inject
    SomeService service;

    @Test
    @OperatesOnDeployment(“myDeployment”)
    public void someTest() {
        // tests
        Assert.assertEquals(service.getUsers().size(), 0);
    }
}
```

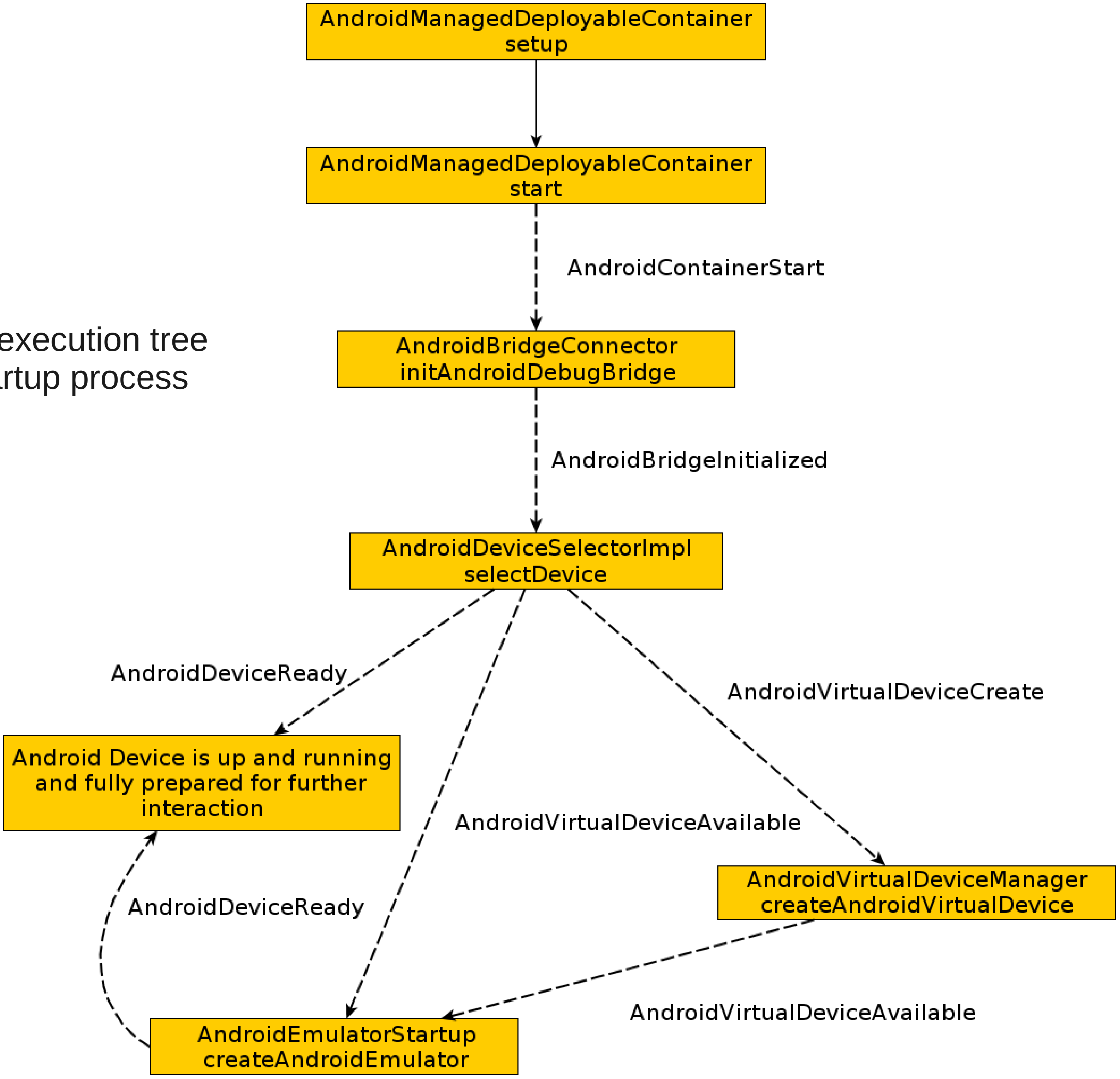
Arquillan Android container

- 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
  - problems and solution

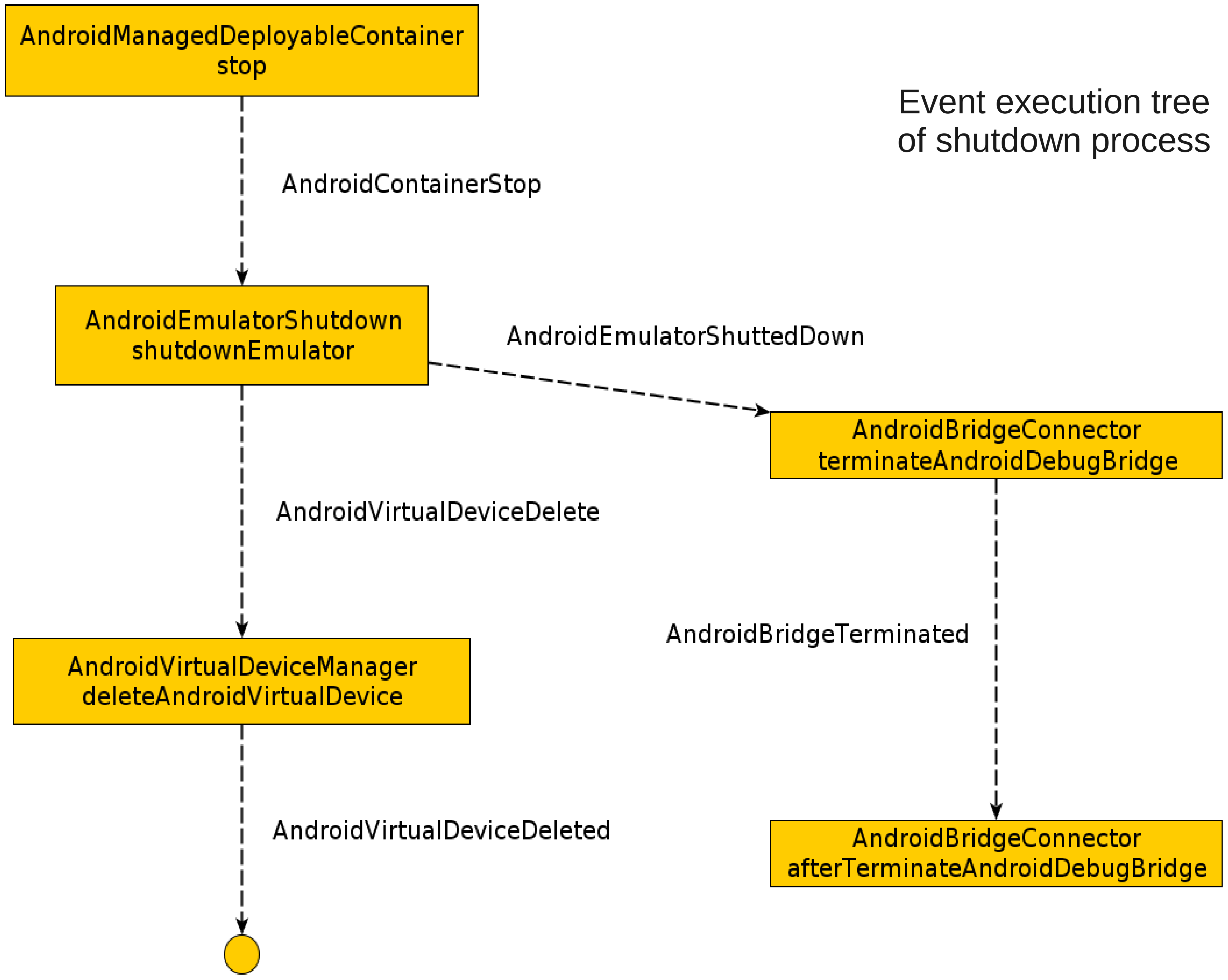
Arquillan Android container

- Container is testing 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

Event execution tree of startup process


AndroidManagedDeployableContainer setup
AndroidManagedDeployableContainer start
AndroidContainerStart
AndroidBridgeConnector initAndroidDebugBridge
AndroidBridgeInitialized
AndroidDeviceSelectorImpl selectDevice
AndroidDeviceReady
AndroidVirtualDeviceCreate
Android Device is up and running and fully prepared for further interaction
AndroidVirtualDeviceAvailable
AndroidVirtualDeviceManager createAndroidVirtualDevice
AndroidDeviceReady
AndroidVirtualDeviceAvailable
AndroidEmulatorStartup createAndroidEmulator

Event execution tree of shutdown process


AndroidManagedDeployableContainer
stop
AndroidContainerStop
AndroidEmulatorShutdown
shutdownEmulator
AndroidEmulatorShuttedDown
AndroidBridgeConnector
terminateAndroidDebugBridge
AndroidVirtualDeviceDelete
AndroidBridgeTerminated
AndroidVirtualDeviceManager
deleteAndroidVirtualDevice
AndroidVirtualDeviceDeleted
AndroidBridgeConnector
afterTerminateAndroidDebugBridge

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 fuctional testing

- Why functional testing?
- Existing solutions and standards
  - Selenium
  - WebDriver
  - Arquillian Graphene / Drone
- What to test?
- Web applications
  - Arquillian Andriod for web
- Native Android applications
  - Arquillian Android native

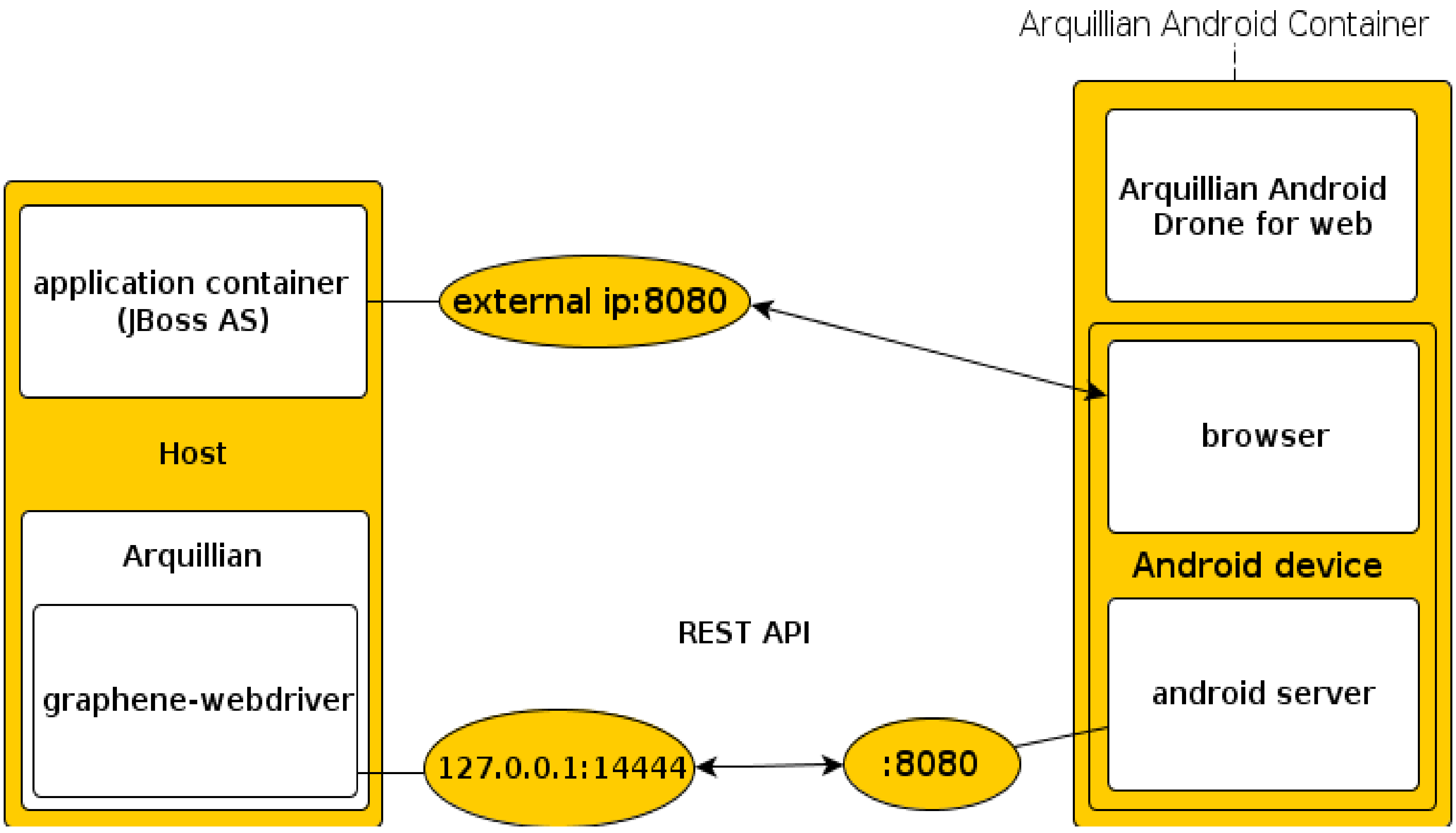
Arquillian Android web plugin

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

Arquillian Android web plugin

- extension into Android container
- standalone project
- tests web applications
- installs android-server.apk from Selenium project
- Uses Arquillian Graphene for injections of WebDriver interface

Architecture of Android container testing of web applications



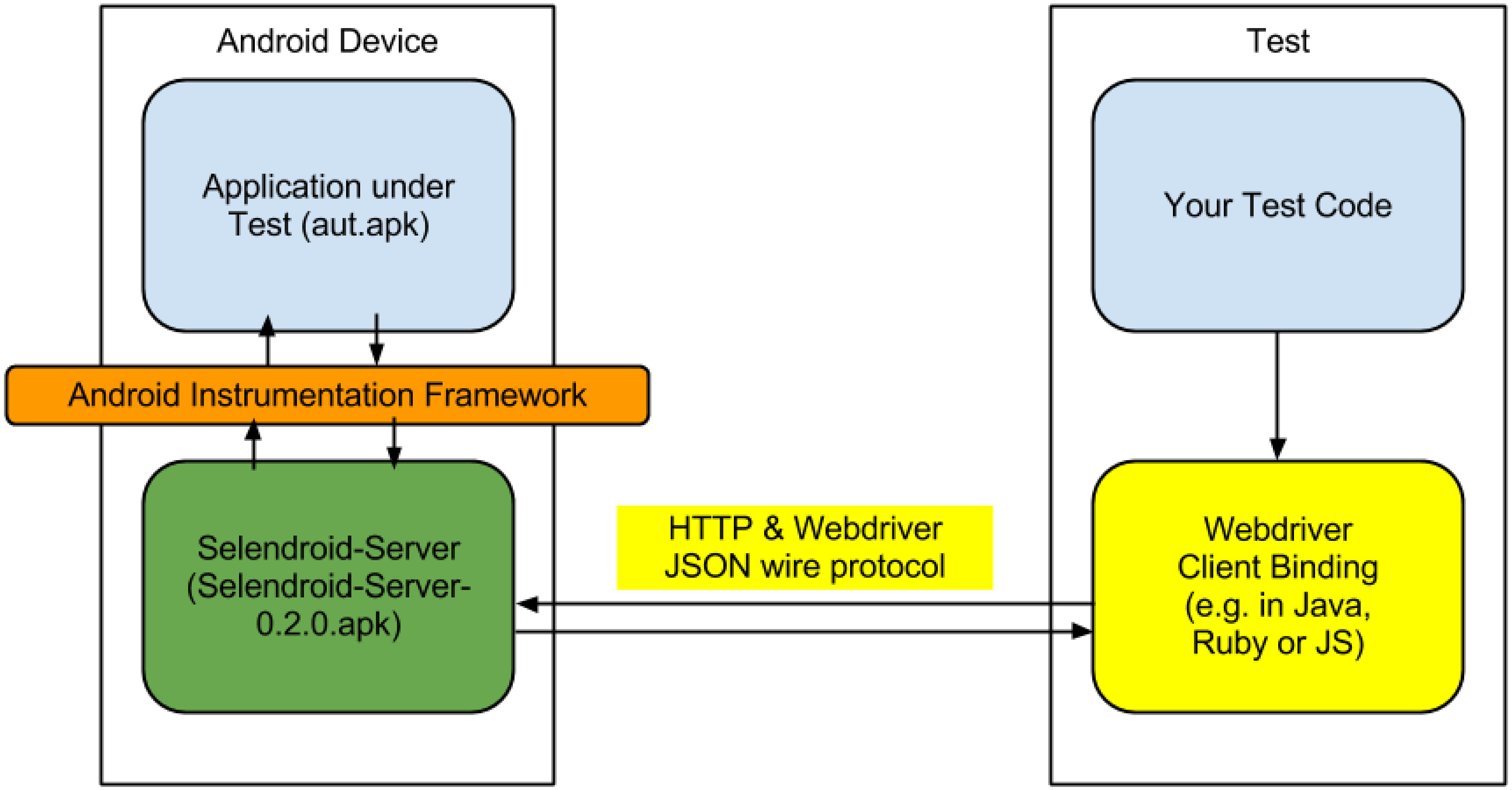
Proof-of-concept web test

```
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 ...
    }
}
```

Arquillian Android native plugin

- Motivation
- Uses only Android container
- Standalone project
- 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


Android Device
Application under Test (aut.apk)
Android Instrumentation Framework
Selendroid-Server (Selendroid-Server-0.2.0.apk)
HTTP & Webdriver JSON wire protocol
Test
Your Test Code
Webdriver Client Binding (e.g. in Java, Ruby or JS)

- 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

Proof-of-concept native test

```
public class MobileTestCase {
    // get application's APK as ShrinkWrap archive
    public static Archive<?> getDeployment() {
        // return APK archive as AUT
    }

    // inject resources from Arquillian
    @ArquillianResource AndroidDevice device;
    @Drone Webdriver driver;

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

Overall results and impact

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

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
- JBoss & Red Hat branding
- Implementing integration with Apple products
  - iPhone, iPad
- Support for APK fluent API for ShrinkWrap
- 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