федеральное государственное автономное образовательное учреждение
ВЫСШЕГО ОБРАЗОВАНИЯ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет *Информационных Технологий и Программирования*
Направление (специальность) *Прикладная математика и информатика*
Квалификация (степень) *Бакалавр прикладной математики и информатики*
Кафедра *Компьютерных технологий* Группа *4539*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Разработка методики автоматического тестирования взаимодействий мобильных приложений с системой Android

Автор квалификационной работы *Баглай Богдан* (подпись)

Руководитель *Макаров Никита Сергеевич* (подпись)

Консультанты:

а) По экономике и организации производства _____ (подпись)

б) По безопасности жизнедеятельности и экологии _____ (подпись)

в) _____ (подпись)

К защите допустить

Зав. кафедрой *Васильев В.Н.* (подпись)

«___» _____ 2015 г.

Санкт-Петербург, 2015 г.

Дата защиты «____» _____ 2015 г.

Секретарь ГАК _____

Листов хранения _____

Чертежей хранения _____

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка подхода автоматического тестирования взаимодействий мобильных приложений с операционной системой Android. Под взаимодействием приложения с системой, имеются в виду любые действия пользователя в операционной системе, имеющие какое-либо влияние на работу приложения и наоборот. Любые действия пользователя в приложении, влияющие на работу операционной системы. Зачем такое тестирование необходимо?

В настоящее время широкое распространение получили приложения, взаимодействующие с системой. Так например, системные уведомления, без которых уже трудно представить современное мобильное приложение. «Список дел» напоминает нам выполнить запланированное важное дело, а приложение «Здоровье» рекомендует сделать зарядку. Кликнув по одному из таких уведомлений, мы перейдем в приложение уведомившее нас. В какую часть приложения мы попадем зависит от самого уведомления. Уведомления могут вести в разные части приложения, которых может быть не мало. А сами уведомления, создаются при определенных условиях: вышло обновление, вам написали сообщение, вам отправили файл. Логику работы таких уведомлений хочется полноценно тестировать перед релизом приложения. Но, большое количество всевозможных комбинаций уведомлений, и не малое время создания необходимых условий, очень сильно замедляет ручное тестирование данной функциональности приложения. На решение данной проблемы и нацелена эта работа.

Для достижения цели ставятся следующая задача: разработать подход автоматизированного тестирования приложений на ОС Android, объединяющий существующие подходы с целью гибкого использования преимуществ каждого из них.

На момент реализации, поставленной в данной работе, цели, в качестве традиционного автоматического тестирования мобильных приложений на Android использовалось два подхода. Эти подходы отличны друг от друга и они не взаимозаменяемы по функциональности. Существующие инструменты автоматического тестирования базируются на использовании одного из этих подходов. Но, во многих случаях, в мобильном автоматическом тестировании есть необходимость в применении обоих способов. В виду такой необходимости существует инструмент — «обертка» над проектами обоих подходов. По своей сути, это два независимых инструмента. А при запуске автоматического теста, в зависимости от необходимой функциональности, запускается один из инструментов. Большой минус такой
«обертки» — это отсутствие связи двух подходов на программном уровне. Кроме того, проекты, входящие в состав данного решения, разрабатываются разными командами. Поддержание работоспособности автоматических тестов для двух инструментов несет дополнительные временные расходы и ведет к понижению общей стабильности, и надежности системы автотестирования.

В данной работе, в рамках реализации новой методики был создан программный комплекс объединяющий оба подхода автоматического тестирования на программном уровне, и, как следствие, решающий проблемы традиционных подходов. Разработанный комплекс успешно прошел проверку в системе автоматизированного тестирования мобильных приложений под Android на предприятии. В настоящее время данная разработка обеспечивает функционирование автоматических тестов взаимодействующих с системой Android, а так же дала возможность покрыть автотестами более тридцати тесовых ситуаций основанных на взаимодействии с системой.

Первая глава работы посвящена обзору известных систем автоматического тестирования мобильных приложений на Android. Там же обсуждаются недостатки существующих систем автоматического тестирования мобильных приложений.

Во второй главе, на основании анализа различных систем автоматического тестирования мобильных приложений, а также опыта автоматического тестирования системных взаимодействий на предприятии, сформулированы основные принципы и требования к системе. Далее приведено общее описание и модель автоматической системы тестирования системных взаимодействий.

В третьей главе приведена архитектура созданного решения. Описано взаимодействия тестовой программы с сервером тестирующего инструмента и других частей системы автоматического тестирования.

В заключении дано описание текущего состояния разработки и перспективы её развития.

ГЛАВА 1. Обзор

1.1. Описание предметной области

В данном разделе подробно опишем цели работы, а также выведем термины, используемые в других частях представленной работы.

1.1.1. Термины и понятия

Введем термины и понятия касающиеся данной работы. В дальнейшем приведенные термины будут использоваться в указанных значениях, если не оговорено обратное.

Системное взаимодействие - любые действия пользователя в операционной системе, имеющие какое-либо влияние на работу приложения и наоборот, любые действия пользователя в приложении влияющие на работу операционной системы. Речь идет о обычном пользователе, который, в обоих случаях, взаимодействует с системой или приложением посредством пользовательского интерфейса (далее UI).

1.1.2. Цель работы

Как уже было упомянуто в ведении, целью данной работы является разработка подхода позволяющего осуществлять автоматизированное тестирование системных взаимодействий мобильного приложения на Android. Для простоты понимания, приведем несколько примеров таких взаимодействий.

Самым простым взаимодействием пользователя с ОС, влияющим на работу приложения, может быть изменение текущего времени в настройках телефона. Приложение, состояние которого зависит от времени («Будильник», «Фазы сна» и т.д.) , должно будет корректно отреагировать на данное системное взаимодействие. Подобным образом, изменения языка в системе, может повлечет некоторые изменения в приложении.

Особый интерес представляют собой Push уведомления. В Android нет никакой встроенной системы для прямого отображения пользователю push-уведомлений. Все данные передаются («пушатся») исключительно в само приложение и в произвольной форме. После получения данных, приложение может, например, выдать стандартное для Android уведомление. Такое уведомление отобразится в верхней части экрана и в “шторке”. А может появится баннер, как в iOS. Учитывая открытость и гибкость ОС Android, после получения push-уведомления, оно может выводиться, практически в любой форме.

В виду гибкости и экономичности к ресурсам, данная технология получила широкое распространение в современных мобильных приложениях. Функциональность приложений все чаще задействует технологию push-уведомлений. А тестирование такой функциональности, связанной с push-уведомлениями, занимает много времени ручного тестировщика. В основном, из-за необходимости создавать определенные условия, при которых трестируемое push-уведомление генерируется. Но, также, из-за необходимости производить определенные манипуляции в UI, вне трестируемого приложения,

что увеличивает вероятность ошибки тестировщика. В случае ошибки все действия, проделанные для создания определенных условий, необходимо повторить заново. Учитывая, что таких уведомлений может быть десятки в одном приложении, процесс ручного тестирования данной функциональности будет очень сильно тормозить цикл разработки приложения.

Далее в работе, из системных взаимодействий, по большей части, мы будем говорить о push-уведомления, ввиду их большего использования. Но рассмотренные методы и разработанное решение актуальны и применимы ко всем видам системных взаимодействий.

Уточним, что наша цель — не просто иметь возможность тестировать системные взаимодействия, но делать это в условиях процесса разработки реального мобильного приложения, нацеленного на широкий круг пользователей. Это означает, что приложение должно корректно работать на довольно большом диапазоне версий Android. Что накладывает дополнительные требования к функциональности инструмента используемого в целях автотестирования. Данное ограничение сильно повлияло на обзор существующих решений автоматического тестирования.

Есть еще одно немаловажное требование к системе автоматического тестирования. Система должна базироваться на использовании одного инструмента автоматического тестирования. Данное ограничение необходимо для обеспечения стабильность системы автоматического тестирования в целом. Использование нескольких инструментов влечет за собой следующие сложности:

- дополнительные трудозатраты на поддержание в работоспособном состоянии написанных автоматических тестов
- трудности, связанные с переходами на новые версии используемых инструментов
- может частично пропасть совместимость инструментов
- разработка разных инструментов ведется разными командами.

1.2. Анализ

В данном разделе описаны требования к инструменту автоматического тестирования приложений написанных под операционную систему Android, описаны существующие, на момент написания данной работы, инструменты, а так же, рассмотрены недостатки готовых решений.

1.2.1. Требования

Учитывая требования выдвинутые в части 1.1.2., составим список из наиболее важных требований к инструменту автоматического тестирования мобильных приложений. Стоит отметить, что все рассматриваемые инструменты удовлетворяют некоторому подмножеству требований, и потому, такие требования не будут включены в список требований. Перечислим наиболее показательные требования:

- Тестовое приложение не должно изменяться.
- Поддержка старых(< 4.2) версий Android.
- Поддержка тестирования системных взаимодействий.
- Система автотестирования должна базироваться на одном инструменте.

1.2.2. Существующие решения

В виду большого количества инструментов автоматического тестирования для платформы Android, в обзор попали инструменты, максимально удовлетворяющие основным требованиям, описанным в части 1.2.1.

Все существующие решения, после отсеивания по первому требованию, делятся на два вида. Первый вид, использующий подход Instrumentation. Второй, использующий подход UIAutomator.

UIAutomator разрабатывается корпорацией Google и поставляется вместе с Android SDK. Android SDK предоставляет следующие инструменты для поддержки автоматизированного функционального тестирования пользовательского интерфейса: UIAutomatorviewer - графический инструмент для распознавания компонентов пользовательского интерфейса в Android приложении, UIAutomator - библиотеки Java API, содержащие методы для создания тестов пользовательского интерфейса.

Для использования UIAutomator, необходима версия Android SDK Tools 21 или выше, API 16 или выше. Из чего следует, что UIAutomator может тестировать только приложения, версия которых выше 4.1.

Instrumentation разрабатывается корпорацией Google и поставляется вместе с Android SDK. Данный способ был изначально, API level 1. Недостатком данного подхода является отсутствие возможности тестирования объектов вне трестируемого приложения. Это означает, что с помощью инструментов, использующих данный подход, нет возможности тестировать системные взаимодействия.

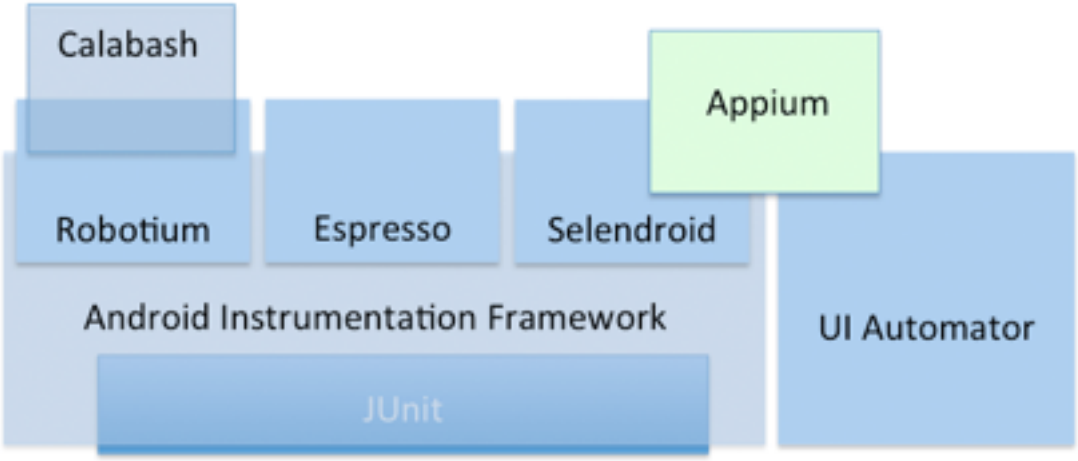

Рис.1. Схема готовых решений и методов которые они используют.

1.2.2.1 Selendroid

Самым ярким представителем Instrumentation подхода является проект Selendroid. Данный проект активно разрабатывается компанией ebay inc. Поддерживает тестирование приложений начиная с версии API 10.

Selendroid работает с наивными приложениями и гибридными. Тесты пишутся с использованием клиентского API WebDriver. Работает с эмуляторами, реальными устройств и может быть интегрирован в качестве узла в Selenium Grid для масштабирования и параллельного тестирования.

На схеме (рис.2.) представлена архитектура проекта Selendroid.

Рис. 2. Схема архитектуры Selendroid

1.2.2.1 Appium

Проект Appium является наиболее интересным для нас инструментов. Само его существование подтверждает важность выдвинутых нами требований к инструменту автоматического тестирования.

Appium объединяет в себе оба подхода автоматического тестирования мобильных приложений под Android, что дает возможность тестировать как старые (<4.2) версии Android посредством Instrumentation подхода, так и системные взаимодействия с помощью UIAutomator. Такая функциональность нам и нужна.

Но не все так гладко с Appium. Как можно увидеть из схемы (Рис.3.), Appium использует оба подхода, Instrumentation и UIautomator. Но, «честно», они это делаю только в случае с UIAutomator походом. А вот по части Instrumentation подхода, Appium является оберткой над Selendroid, которая переводит команды Selendroid WebDriver в Selendroid команды. В итоге, получаем, что для версий Android 4.2+ используется Google UiAutomator, а для версий Android 2.3+ используется Selendroid, работающий посредством Google Instrumentation.

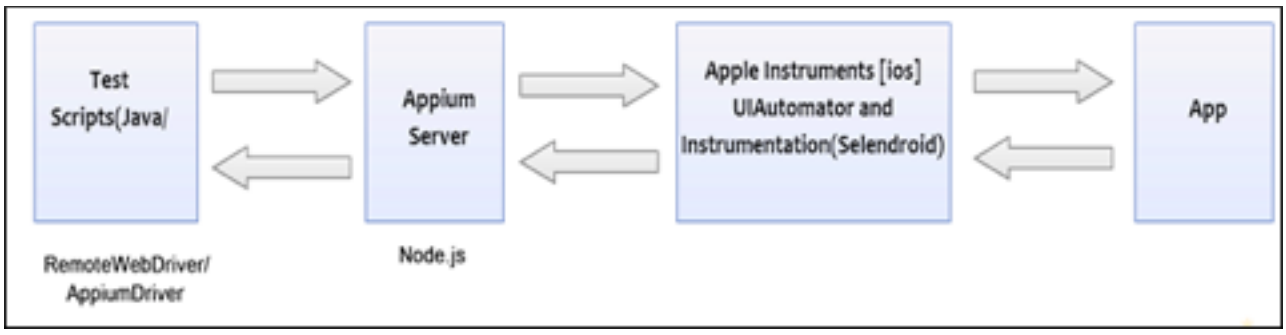

Рис.3. Схема составных частей Appium.

По сравнению с Selendroid, проект Appium удовлетворяет большему количеству, поставленных нами, требований.

1.2.3. Недостатки готовых решений

Недостатком всех инструментов, использующих Instrumentation подход, в том числе и Selenium-a, является полное отсутствие возможности тестирования системных взаимодействий. Автоматическому тестирования подлежит только трестируемое приложение, а доступ вне приложения отсутствует.

Подход UIAutomator имеет возможность тестирования системных взаимодействий, но обладает другим недостатком. Он связан с отсутствием возможности работы со старыми версиями Android. Это означает, что все инструменты, использующие только данный подход, тоже будут обладать данным недостатком.

Проект Appium сделал первый шаг к устранению данной проблемы. В нем используются оба подхода. Большой минус в том, что это не один проект а два отдельных. Appium имеет в своем составе проект Selendroid, работающий при тестировании версии Android <4.2.

1.2.4 Итоги

В ходе исследования рынка, на предмет наличия готового решения, было выявлено, что существует ряд инструментов, которые частично удовлетворяют выдвинутым требованиям. Однако, не было найдено решения, обладающего всеми достоинствами одновременно. Потребность в таком решении сподвигла на разработку своего проекта.

1.3. Постановка задачи

Далее робота будет посвящена проектированию и разработке своего решения для автоматического тестирования мобильных приложений под Android. Решение должно удовлетворять выдвинутым требованиям.

Задача — разработать инструмент, объединяющий подходы автоматического тестирования, использующие Instrumentation и UIAutomator.

ГЛАВА 2. ТЕОРЕТИЧЕСКИЕ РЕЗУЛЬТАТЫ

В данной главе рассматриваются теоретические аспекты создания инструмента автоматического тестирования объединяющего подходы Instrumentation и UIAutomator. Так же приведена общая схема инструмента.

2.1. Предлагаемое решение

Для решения поставленной задачи, предлагается разработать инструмент на базе уже существующего проекта авто тестирования одного из подходов.

Очевидными кандидатами являются проекты Selendroid и Appium. Оба проекта разрабатываются как open-source. Selenium использует Instrumentation. Appium использует UIAutomator и Selenium. Как упоминалось в обзоре, Appium уже сдел свой шаг в сторону использования Instrumentation. А Selenium, еще не делал подвижек в сторону UIAutomator-а. Это склоняет наш выбор в сторону Selenium.

Рассмотрим подробнее схему измененного проекта Selendroid (Рис.4.)

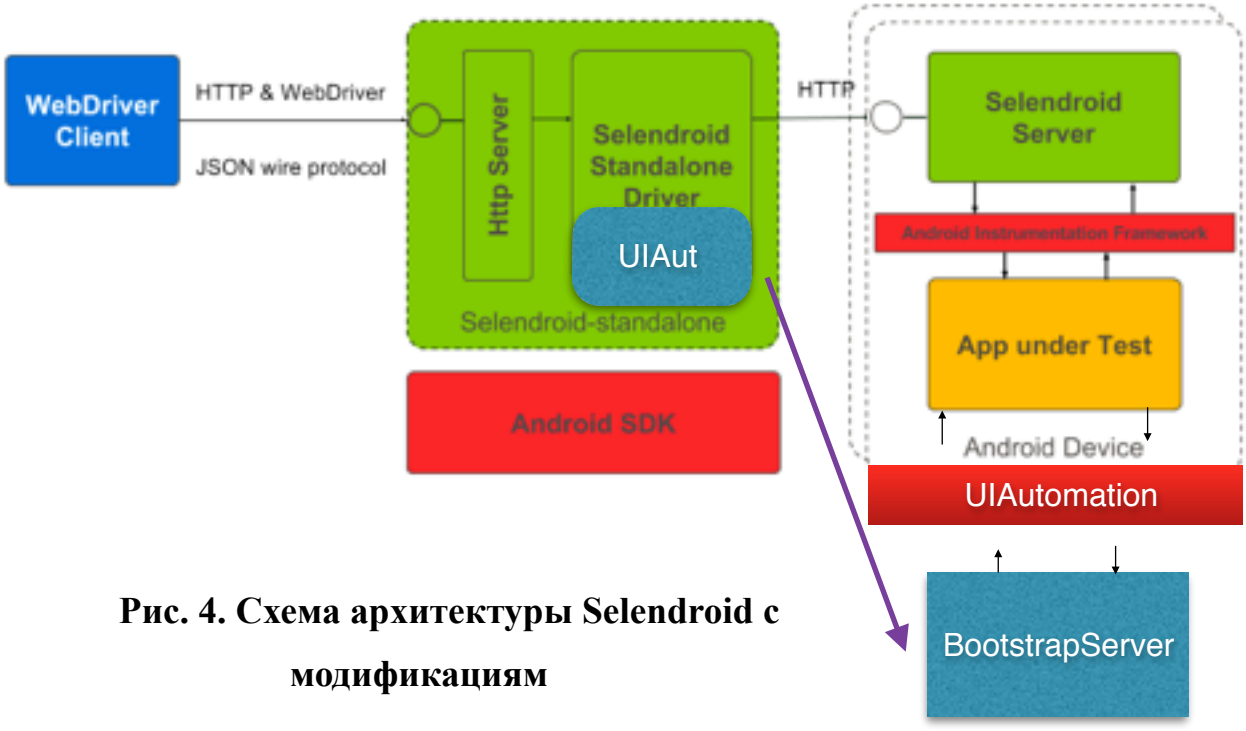

Рис. 4. Схема архитектуры Selendroid с модификациям

В исходной архитектуре Selendroid проекта, модификациям подлежит только Selendroid-standalone. WebDriver Client никак не изменится. Само тестовое приложение также останется без изменения. В системе появится новый элемент — BootstrapServer, выполняющий функции связующего звена между тестовым приложением и модифицированным Selendroid-standalone сервером. BootstrapServer можно использовать из проекта Appium без каких-либо изменений.

2.2. Требования

Предложенное решение удовлетворяет всем требованиям, изложенным ранее. Объединив подходы Instrumentation и UIAutomator, мы получим гибкий инструмент обладающий полезной функциональностью обоих подходов. В результате создания предложенного решения, мы получим единый инструмент имеющий возможность тестировать системные взаимодействия и работающий со старыми версиями Android.

2.3. Теоретическое сравнение

Как уже было подмечено ранее, Appium — проект который уже сделал некоторые попытки объединения возможностей обоих подходов в одном месте. Сделали они это путем написания «обертки» над проектом Selenium. Эо дало им возможность тестировать старые версии Android.

Наше решение отличается тем, что у нас будет один инструмент на базе Selendroid со встроенной возможностью переключаться в режим работы с UIAutomator. Selendroid позволяет тестировать более старые версии, а переход в режим UIAutomator позволит тестировать системные взаимодействия. Все это в одном инструменте, а режим работы можно будет переключать в коде самого теста.

ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

В предыдущих главах были описаны концепции положенные в основу системы автоматического тестирования мобильных приложений. В этой главе описаны технические подробности реализации данной системы.

3.1. Интерфейс теста

С точки зрения теста, в нашем инструменте не должно ничего измениться. Тест должен использовать стандартный WebDriver. Возникает вопрос: как сообщить Selendroid серверу что мы хотим начать работать в режиме UIAutomator.

Решение оказалось на поверхности. В протоколе WebDriver есть метод : session/:sessionId/window. Он используется чтобы перейти в режим WebView и обратно, в Native режим. Запускается с параметром «WEB» и «NATIVE».

Для перехода в режим UIAutomator, используется этот же метод с прараметрами «UIAutomator» и «Instrumentation».

3.2. Архитектура

В данном раздела будет представлена архитектура реализованного решения.

Основным изменениям подвергся Selendroid-standalone сервер. (Рис.5.)

Тест является клиентом, взаимодействующим с Selendroid-standalone сервером с помощью JSONE wire протокола. Сам клиент, WebDriver, не изменился. Все запросы, отправляемые клиентом, попадают в RequestRedirectHandler. Далее, поведение Selendroid-standalone сервера зависит от режима в котором он находится, UIAutomator или Istrumentation. Если сервер

в режиме Instrumentation, то запрос передастся в instumentationHendler, который, в свою очередь, передает запрос Selendroid серверу по HTTP. Selendroid server взаимодействует с трестируемым приложением с помощью Instrumentation API.

Если же, Selendroid-standalone находится в режиме «UIAutomator», то RequestRedirectHandler передаст запрос в uiAutomatorHendler который передаст запрос в UiAutomatorConnector. UiAutomatorConnector нужен для переработки запроса вида WebDriver в запрос для Bootstrap. Далее, переработанный запрос передается в uiAutomatorClient, который и пересылает его Bootstrap серверу. uiAutomatorClient подключенн к Bootstrap серверу, только если Selenium-standalone в режиме UIAutomation. В остальное время он бездействует.

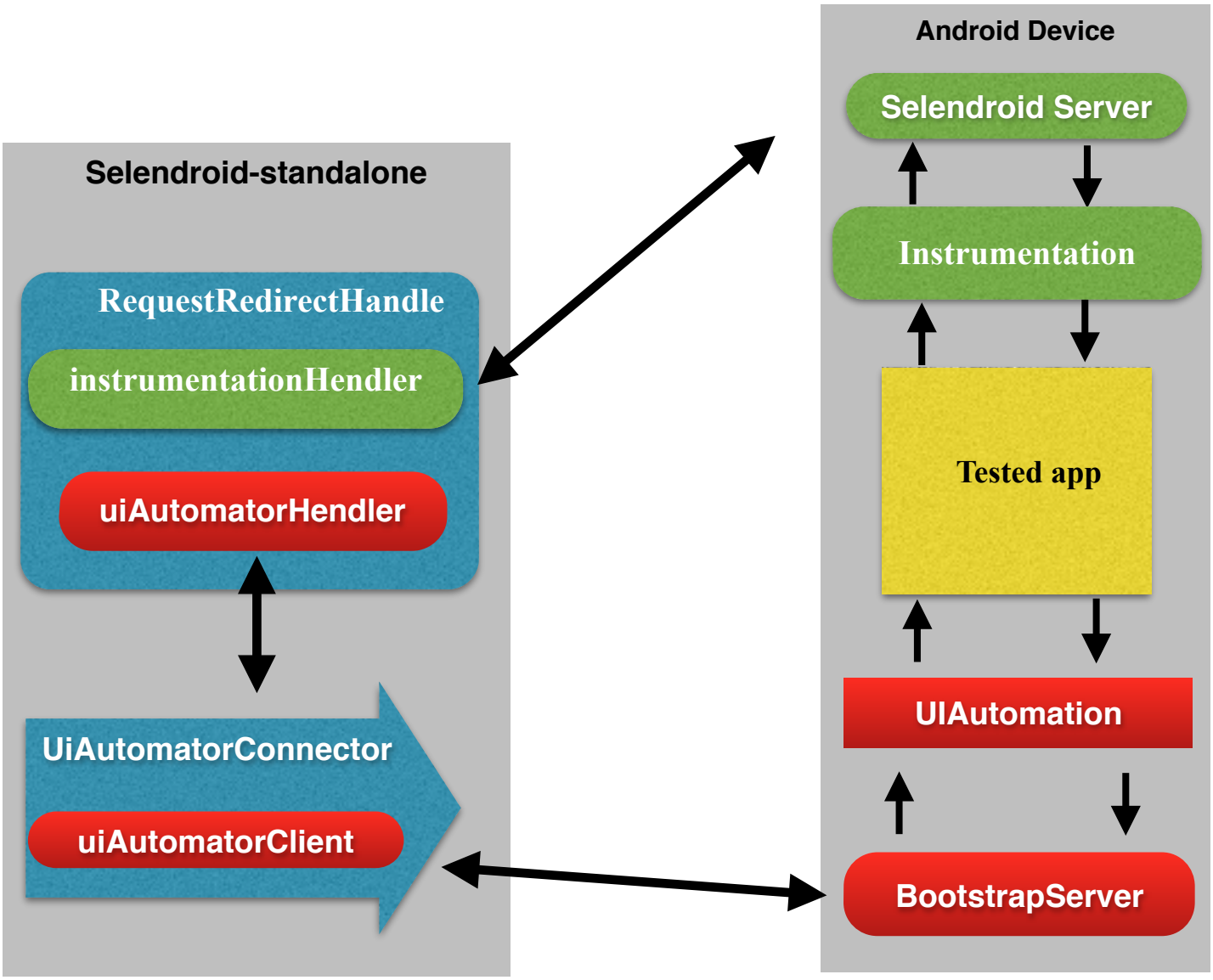

Рис.6. Архитектура реализованного инструмента

Данная архитектура разрабатывался с возможностью параллельного запуска тестов. Это возможно благодаря uiAutomatorClient который привязан к тестовой сессии. Таких сессий можно создавать много. На каждую будет выделен девайс или эмулятор. Состояния сессий независимы. Одна может находиться в Instrumentation режиме, а другая в UIAutomator

Приведем пример использования инструмента.

```
...
Controls.scrollDownFullScreen(driverId);
Webview.switchToUIAutomatorMode(driverId);
WebElement element = driver.findElement(By.xpath(locator));
String messageText = element.getText();
element.click();
Webview.returnFromUIAutomatorMode(driverId);
assert(...
```

3.2. Практическое применение

Разработанное решение было внедрено в реальный проект автоматического тестирования мобильных приложений на предприятии. Данное решение сразу позволило избавиться от, ранее используемых, UIAutomator тестов. Такие тесты, в виде jar файлов, копировались на тестовое устройство и запускались через shell из обычного Selendroid теста. Статус выполнения таких тестов недоступен. Для изменения данного теста, его нужно переписать, скомпилировать, и снова записать на устройство. Разобраться, в чем причина падения такого теста, не представляется возможным.

Заключение

Инструмент автоматизированного тестирования приложений под ОС Android на основе подходов Instrumentation и UIAutomation был успешно разработан и опробован в условиях предприятия.

Использование инструмента дало возможность не только сильно улучшить качество существующих автоматических тестов системных взаимодействий, но написания новых тестов.

На данный момент инструмент успешно используется в составе системы автоматического тестирования.

СПИСОК ЛИТЕРАТУРЫ

1. Software Test Automation Paperback use pre formatted date that complies with legal requirement from media matrix, by Mark Fewster, Dorothy Graham – September 4, 1999.
2. Selenium 2 Testing Tools: Beginner's Guide Paperback use pre formatted date that complies with legal requirement from media matrix, by Burns David – October 19, 2012.

Ресурсы глобальной сети Интернет

4. http://automated-testing.info/ — информационный сайт посвященный автоматизации тестирования
5. http://selendroid.io — сайт инструмента автоматизации тестирования для Android — selendroid
6. http://selenium2.ru — сайт инструмента автоматизации тестирования — selenium

Приложения

3.3. Приложения 1. Пример теста Push-уведомления.

```
@Test
 public void notificationTest() {
    user1 = BotManager.getRandomUser();
    user2 = BotManager.getRandomUser();
    apiUser1 = new Messages(user1);
    apiUser2 = new Messages(user2);

    login(user1);

       String textInNotification =  MessagesUtils.generateRandomText();
       apiUser2.sendMessageTo(user1.getId(), textInNotification);
       Controls.scrollDownFullScreen(driverId);
       Webview.switchToUIAutomatorMode(driverId);
       WebElement element = driver.findElement(By.xpath(locator));
       String notificText = element.getText();
       element.click();
       Webview.returnFromUIAutomatorMode(driverId);

       assertTrue("Message in notifications is incorrect. Expected: " + user2
               + ": " + textInNotification + ", actual: " + notificText,
               notificText.equals(user2 + ": " + textInNotification), logger);

       assertTrue("Last message in chat is incorrect. Expected: " +
        textInNotification + ", actual: " + chatPage.getLastMessageText(),
    chatPage.getTheLastOfTheLastMessageText().equals(textInNotification),
               logger);
       logger.testWorkedFine();
 }
```