**ОПИСАНИЕ АРХИТЕКТУРЫ ИНСТРУМЕНТАЛЬНОГО ОКРУЖЕНИЯ СБОРКИ ГОТОВЫХ ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ ПЛАТФОРМ**

*Тюрнин Алексей Валерьевич*

Пармский государственный национальный исследовательский университет, 614990, Россия, г. Пермь, ул. Букирева, 15, [turlex251@gmail.com](mailto:turlex251@gmail.com)

Ключевые слова: мобильная разработка, архитектура, инструментальное окружение

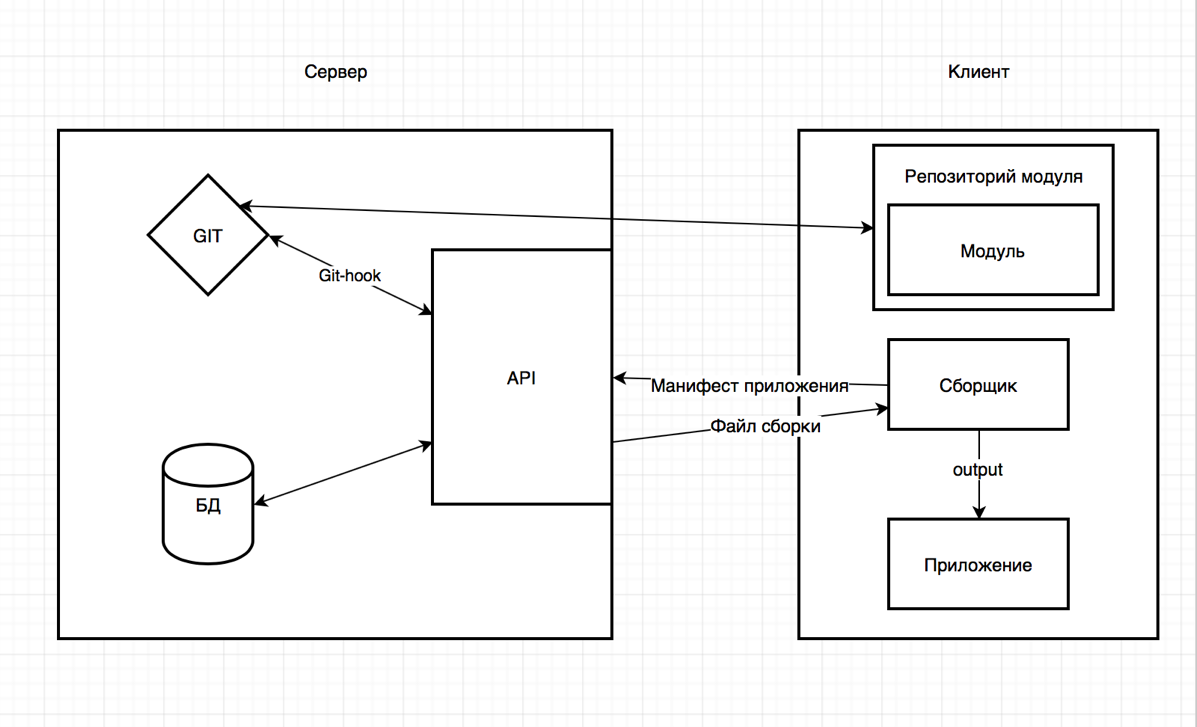
Смартфоны давно уже не являются роскошью или чем то экстраординарным. Их массовое распространение в начале 2010-х привело к тому, что рынок мобильной разработки вырос многократно и конкуренция на нем в том числе. И все важнее стало правильных пропорциях совмещать скорость и качество разработки. Кроме того, довольно часто приложения для разных заказчиков отличаются лишь некоторым уникальным функционалом, что приводит к вопросу о использовании конструкторов и case-средств. И в своем время такой же высокий рост развития интернета и необходимость разработки сайтов со схожим функционалом привел к широкому распространению разных конструкторов сайтов и case-средств.

В мобильной разработке таких средств нет и вся автоматизация разработки пока что сводится к менеджерам пакетов. К тому же многие идеи case-средств невозможно реализовать из-за различных ограничений мобильных платформ на работу уже установленных приложений. Кроме того, хотелось бы использовать все возможности исключений и обработки отсутствующего функционала, что позволило бы писать код один раз, а потом просто использовать его в нужных местах.

Именно поэтому появилась идея разработки инструментального окружения, позволяющего упростить и ускорить разработку приложений для мобильных платформ. Идея состоит в том, что бы свести разработку приложения к написанию модулей, необходимых для работы приложения, указать для каждого из них, какой функционал им необходим для работы. Кроме того дать возможность некоторым модулям отсутствовать в приложении, что не привело бы к ошибкам компиляции или ошибкам runtime и при это могло влиять на логику работы модулей, которые от них зависят. Зависимость от таких модулей будем называть «слабой» зависимостью. И, в конце концов, свести сборку нового приложения к простому указанию его конфигурации и конфигурации его модулей.

Одна из главных задач при разработке такого инструментального окружения – описать архитектуру системы, ее компоненты и взаимодействие между ними. Кроме того, так как для сборки будут использоваться уже готовые модули, необходимо определить их спецификации и спецификацию приложения в целом.

Архитектура системы будет клиент-серверная, так как она позволяет в очень удобной форме разделить функционал разработки и сборки и функционал хранения исходного кода и мета-данных.



**Рис. 1 Схема архитектуры**

В первую очередь введем понятие модуля, как функционально законченного фрагмента программы. При помощи описания модуля через специальные файлы-манифесты абстрагироваться от реализации конкретного модуля и обработать его отсутствие в конечной сборке приложения. Манифест модуля содержит в себе информацию о уникальном имени модуля, его «сильных» и «слабых» зависимостях, а так же дополнительная мета информация, которая позволяет на сборки генерации готового приложения генерировать код сборки для конкретного модуля.

Технологический процесс написания модуля будет выглядеть следующим образом:

* Создание git-репозитория модуля
* Написание модуля. Описание интерфейса доступа к модулю
* Написание манифеста модуля со всей необходимой для сборки информацией информации об интерфейсе.
* Отправка исходного кода модуля на сервер при помощи команды git «push».

Но работа над модулем на этом не заканчивается. На стороне сервера модуль валидируется на основании его манифеста и, если модуль валиден, то данные о нем добавляются в БД модулей. Тем самым на этапе разработки существует возможность поддерживать версионность модулей на каждом этапе разработки. И, в теории, не будет происходить ситуации, когда не законченный модуль попадет в финальную сборку приложения. Кроме того, использование интерфейсов для доступа к функциональности модуля позволят использовать MOK-классы для разработки.

Далее введем понятие приложения, как совокупность связанных между собой модулей. У приложение так же есть свой файл-манифест, который содержит в себе информацию, необходимую для сборки. Манифест приложения содержит в себе название приложения, список его модулей, модуль–«точка входа», а так же пути к файлам конфигурации и набору asset’ов.

К1. Первая часть клиентского инструментария представляет собой набор ПО для создания отдельных компонентов системы - модулей. Он состоит из следующих частей:

Инструмент для генерации и конфигурации шаболнных проектов модулей

Инструмент для оперирования мета-данными модуля и его зависимостями

К2. Вторая часть представляет из себя ПО по генерации готового приложения. Создает шаблон проекта приложения, подгружает все необходимые внешние и внутренние зависимости и генерирует код сборки всего приложения.

С1. Сервер состоит из 3 частей.

git-репозиторий, в котором хранятся модули.

api позволяющее генерировать итоговый файл сборки

бд, для хранения мета данных о модулях

Одна из основных задач - это проектирование всех необходимых компонент такого окружения. И клиент-серверная архитектура для этого очень хорошо подходит: и разработка модулей происходит на локальных машинах с использованием дополнительной места информации о приложении и возможностях взаимодействия между модулями и сборка готового приложения из готовых модулей по определённой конфигурации, а на сервере в свою очередь будут храниться сами модули индивидуальных репозиториях, валидация метаданных о модулях, разрешение зависимостей в конфигурации приложения и формирование под файла для клиентской части.

Технологический процесс написания модуля будет выглядеть следующим образом:

\* Генерация файла манифеста и описание всех необходимых зависимостей.

\* Написание модуля. Описание интрефейса доступа к модулю

\* Добавление в файл манифест информации об интерфейсе.

\* Далее модуль обычным гит Пушем отправляется в свой репозиторий

\* В гит репозитории с помощью специально написанного скрипта-хука интерфейс модуля сравнивается с его спецификацией. И если модуль Валиде, то информациям о нем попадает в базу доступных для использования модулей.

\* Так происходит для каждого следующего написанного модуля.

Манифест-модуля хранит метаинформацию о модуле и в том числе все его сильные и слабые зависимости. В момент формирования шаблона приложения модуля с сервера подтягиваются все необходимые интерфейсы модулей-зависимостей а также, если они реализованы, MOK-классы. Все это в общем позволяет разрабатывать модуль независимо от реализации интерфейсов его зависимостей, что является хорошим примером реализации принципов SOLID.