**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 стали частью нашей повседневного быта. 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 по итогу 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.

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 выглядит следующим образом:

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

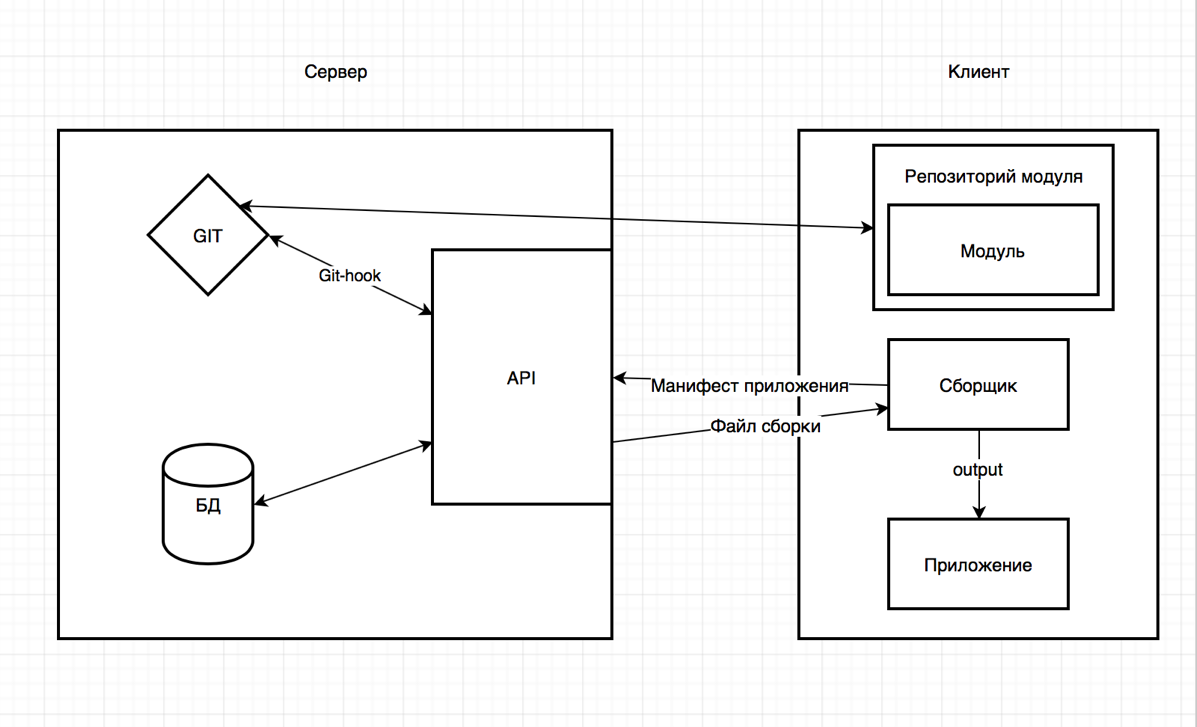
Но работа над модулем на этом не заканчивается. На стороне сервера модуль валидируется на основании его манифеста и, если модуль валиден, то данные о нем добавляются в БД модулей. Тем самым на этапе разработки существует возможность поддерживать версионность модулей на каждом этапе разработки. И, в теории,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

Технологический процесс сборки приложения будет выглядит следующим образом:

1. Создание манифеста приложения. Указание все необходимой информации: имени приложения, пути к файлам конфигурации и asset’ам, список всех необходимых модулей и корневого модуля.
2. Передача манифест приложения программе-сборщику.
3. Программа сборщик отправляет данные из манифеста на сервер.
4. На стороне сервера строится граф зависимостей и генерируется файл сборки, который отправляется обратно на клиент.
5. Программа сборщик на основании файла сборки создает проект приложения и загружает в него все необходимые зависимости.
6. Запускается генератор сборки. Это программа, которая, используя мета-данные каждого модуля, генерирует исходный код для инициализации каждого модуля и «протягивания» всех необходимых и доступных ему зависимостей.
7. Запускается процесс компиляции. Если во время этого процесса происходит ошибка, то по каждой ошибке в определенном модуле генерируется отчет и отправляется на сервер. На сервере автор модуля оповещается об ошибке письмом, с прикрепленным к нему файлом с кодом и описанием ошибки, манифестом приложения и файлом сборки. Если ошибок нет, то приложение является завершенным и отправляется отчет о сборке на сервер.

Технологический процесс, описанный выше, подразумевает использование определенный паттернов проектирования как самом приложения, так и модуля в частности. Кроме того, желательно наличие централизованного сервера, на котором будут храниться модули, мета-информация о них и, в том числе, должен быть реализован механизм разрешения зависимостей, который, при наличии большого количества модулей и связей между ними, может требовать больших вычислительных мощностей. И не мало важен доступ к этому функционалу на только на определенной машине, но и удалённо. При этом сама сборка не требует высокой вычислительной мощности и может выполняться на локальной машине. Кроме того, в следствии некоторых ограничений мобильных платформ, сборка iOS-приложения, написанного на Obj-C или Swift, не может производиться не на базе операционной системы macOS, но в то же время на хранение и обработку такого исходного кода ограничеий нет.



**Рис. 1 Схема архитектуры**

Все это приводит к тому, что оптимальная архитектура для такой системы является архитектура клиент-сервер, представленную на Рис. 1.

Такая архитектура системы позволяет ускорить процесс разработки за счет выделения функционала приложения в отдельные модули с унифицированным описанием и доступом, что позволяет при разработке абстрагироваться от применения кода в определенном приложении, и сконцентрироваться на самой стабильности и функциональности. В том числе процесс сборки становится практически полностью автоматизированным, что достигается сильным переиспользованием готового кода. Система имеет хороший потенциал и будет реализована для разработки приложений на платформе iOS и в последствии будет расширена и для разработки под другие платформы.