Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Ефимов Глеб Дмитриевич

Автоматизация сборки персонифицированных клиентских приложений на основе платформы Ubiq Mobile

Дипломная работа

Допущена к защите.

Зав. кафедрой:

д.ф.-м.н., профессор Терехов А. Н

Научный руководитель:

д.ф.-м.н., профессор Терехов А. Н

Рецензент:

Оносовский В.В.

Санкт-Петербург

2015

SAINT PETERSBURG STATE UNIVERSITY

Chair of Software Engineering

Gleb Efimov

Build automation of personalised client applications based on platform Ubiq Mobile

Graduation Thesis

Admitted for defence.

Head of the Chair:

Professor Andrey Terekhov

Scientific supervisor:

Professor Andrey Terekhov

Reviewer:

Onossovski V.V.

Saint Petersburg

2015

Оглавление

Введение

В настоящее время сфера мобильных технологий активно развивается: постоянно появляются новые устройства с уникальными особенностями, новые платформы и соотвествующие средства разработки. Использование всего многообразия технологий создает избыточные сложности, разработчики вынуждены реализовывать одну и ту же логику для каждой из мобильных платформ, учитывая их особенности, что требует большого количества специалистов с высоким уровнем квалификации.

Для решения проблемы платформенной фрагментации существует два подхода. Первый заключается в использовании программных решений, которые могут генерировать платформозависимый код. Примерами такого подхода являются технологии разработки мобильных приложений Xamarin [14], Apache Cordova [8] или Appcelerator [9], а также технологии для разработки игр Unity [13], Corona [11]. Второй подход заключается в использовании клиент-серверной архитекуры, которая организует «трансляцию» пользовательского интерфейса приложений на конечные устройства. В таких платформах присутствуют «тонкие» клиенты для различных мобильных платформ и единый сервер, координирующий работу пользователей. Примерами реализации подобного подхода являются сервисы Amazon AppStream [10] и Azure RemoteApp [12], которые позволяют транслировать приложения на мобильные устройства, используя мощности облачных вычислительных ресурсов.

На математико-механическом факультете в течение последних лет разрабатывается платформа Ubiq Mobile [5] [15], использующая клиент-серверный подход. При передаче информации используется оригинальный протокол, инкапсулируемый в транспортном уровне в модели сетевого взаимодействия OSI. Протокол специально оптимизирован для работы в реальных условиях мобильных соединений, которые не всегда достаточно

быстры, могут прерываться и восстанавливаться. Использование оригинального двоичного протокола повышает устойчивость работы системы. Это обстоятельство в совокупности с компактностью передаваемых данных предоставляет обширные возможности оптимизации сетевого трафика, позволяет эффективно использовать ресурсы и обеспечивает приемлемые характеристики работы в медленных сетях. Система Ubiq Mobile позволяет реализовывать приложения, которые сочетают сложность внутренней логики с минимизацией нагрузки на устройства и сетевые каналы, при этом максимально используя возможности конкретных мобильных платформ. В настоящее время в данной среде реализованы универсальные приложения для основных операционных систем: iOS, Android, Windows Phone и Java ME.

Все платформы кроссплатформенной разработки сталкиваются с вопросом создания конечных приложений, которые исполняются на целевом устройстве. Среди существующих решений данной задачи можно выделить три варианта:

1. Сборка нативными средствами разработки. Разработчик самостоятельно производит сборку приложений для целевых операционных систем. В этом случае иногда проект разделяется несколько частей, одна из которых содержит кросплатформенный, а остальные платформозависмый код каждой из платформ.
2. Автоматизированная сборка. Среда кроссплатформенной разработки предоставляет средства, позволяющие выполнять сборку программ на рабочей станции пользователя. Зачастую это требует предварительной установки нативных комплектов средств разработки.
3. Облачный сервис. Программист средствами среды разработки создает запрос к облачному сервису на генерацию приложений. Данный подход очень удобен в разработке, не требует нативных комплектов средств разработки и предоставляет дополнительные

возможности по распространению приложения.

Универсальные клиенты используются для тестирования сервисов Ubiq Mobile на физических устройствах и генерации персонифицированных мобильных приложений.

Разработчику в процессе работы досутпны эмулятор и универсальные клиентские приложения. При помощи универсального клиента разработчик может получить доступ ко всем смоим приложениям, выгруженным в облако. Универсальные клиенты предназначены в первую очередь для отладки: они позволяют программисту проверить логику работы сервиса Ubiq Mobile, протестировать интерфейс на различных разрешениях и оценить общее удобство использования.

Вместо универсальных клиентов, для публикации и размещения в магазинах приложений используются персонифицированные клиенты. Процесс сборки приложения для каждой операционной системы требует множества ручных рутинных действий и, как следствие, влечет за собой достаточно серьезные накладные временные расходы по подготовке универсального клиента к персонификации.

Первоначально, рапространение мобильных приложений, реализованных с помощью технологии Ubiq Mobile, планировалось через доступные в магазинах приложений универсальные клиенты. На практике данное решение оказалось спорным. Во-первых, конечным пользователям необходимо совершить дополнительные действия, чтобы подключиться к нужному приложению, что значительно снижает удобство использования. Во-вторых, публикация универсального клиентского приложения для конечного пользователя может нарушать правила некоторых магазинов приложений, что делает невозможным размещение таких универсальных клиентов, например, в App Store.

Поэтому было принято решение распространять мобильные приложения

через персонифицированные клиенты. Таким образом, возникла задача разработки простого и удобного механизма генерации персонифицированных приложений. С учетом архитектуры платформы Ubiq Mobile самым подходящим вариантом оказалось использование облачного сервиса, в котором от пользователя не требуется установки дополнительного программного обеспечения в силу отсутствия необходимости самостоятельно собирать мобильные приложения.

1. Постановка задачи

Плафторма Ubiq Mobile нуждалась в простом и удобном способе генерации конечных пользовательских приложений, в связи с чем было принято решение реализовать в рамках системы облачный сборочный сервис, который по запросу пользователя производит сборку персонифицированных клиентов.

Целью данной дипломной работы является создание и внедрение облачного сервиса сборки в платформу Ubiq Mobile.

Для достижения поставленной цели были выделены несколько подзадач:

- спроектировать и реализовать облачный сервис сборки персонифицированных приложений для целевых операционных систем

- реализовать службы сборки персонифицированных клиентских приложений для платформ iOS и Windows Phone

- интегрировать облачный сервис сборки в платформу Ubiq Mobile.

2. Обзор

2.1 Платформа Ubiq Mobile

2.1.1 Протокол

Платформа Ubiq Mobile имеет клиент-серверную архитектуру, в основе которой лежит оригинальный протокол сетевого взаимодействия. Передача данных по этому протоколу происходит над транспортным уровнем в сетевой модели OSI, используя TCP соединение. Данный выбор обусловлен требованием к надежности работы в нестабильных сетевых условиях, что делает необходимым прикладывать специальные усилия по обеспечению целостности передаваемых данных.

Оригинальный протокол Ubiq Mobile относится к типу «атрибуты и структуры». Терминальный сервер и клиент могут принимать и отправлять протокольные управляющие сообщения. Каждая команда имеет древовидную структуру, элементами которой являются секции с некоторым набором полей. Секции представляют собой непрерывные области данных переменной длины, последовательно закодированные в полезной нагрузке сообщения. Структура главной секции выглядит следующим образом:

Main section length	Main section code	Main section data
3 bytes	1 byte	variable

Рис. 1

В поле «Main section data» содержатся дочерние подсекции, имеющие следующую структуру:

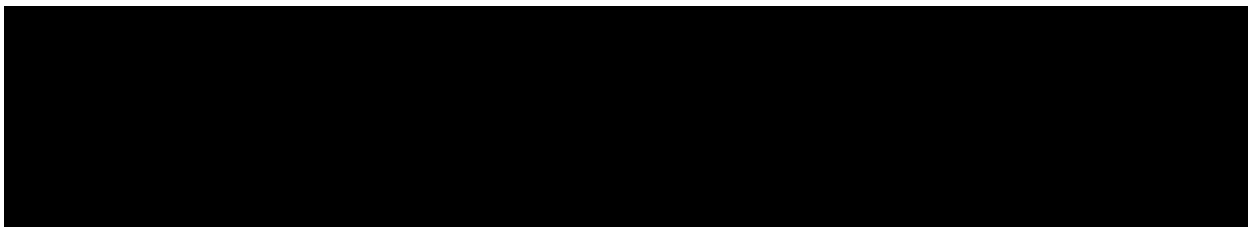


Рис. 2

Полезная нагрузка секции структурирована в виде последовательности полей фиксированной или переменной длины в разделе «Section data». Длина атрибута зависит от типа данных. Примером полей переменной длины служат массивы, изображения, а также строки. Все числа представляются в порядке от младшего байта к старшему.

2.1.2 Организация сервисов

Серверная часть приложения Ubiq Mobile представляет собой облачный сервис, доступ к которому осуществляется через тонкий клиент, являющийся рапространяемым мобильным приложением В процессе работы происходит передача текущего «виртуального» экрана сервиса на конечное устройство, на котором отрисовывается (rendering) пользовательский интерфейс с использованием нативных визуальных элементов для соответствующей операционной системы. Сервис однозначно определяется следующими параметрами:

- IP-адрес сервера,
- порт
- идентификатор сервиса

Платформа UbiqMobile поддерживает два способа публикации разработанных на ее основе облачных сервисов. Первый, «локальный», предназначен для разработки и тестирования приложений. Данные сервисы досутпны только их непосредственным разработчикам. Второй, «глобальный», используется для готовых к распространению мобильных приложений. Сервисы этого типа находятся в публичном доступе. Такое

разделение позволяет вести многопользовательскую разработку, используя один сервер и обеспечивая достаточный уровень безопасности. Локальные сервисы размещаются в персональных рабочих пространствах пользователей, называемых «песочницами», доступ к которым осуществляется с помощью механизма идентификации. Для доступа к «песочнице» пользователя-программиста используется сертификат, выдаваемый платформой.

2.1.3 Функциональность универсальных клиентов

Основная задача «тонких» клиентов заключается в выполнении управляющих команд терминального сервера и передаче совершенных пользователем действий. Они могут быть нескольких видов:

- отображение графики, заключается в отрисовке интерфейса сервиса для соответствующей операционной системы клиента нативными средствами мобильной платформы
- осуществление операционной деятельности – требуется для обслуживания клиента, например, поддержания кэша изображений в когерентном состоянии или проверка сетевого соединения
- выполнение различных аппаратных функций для осуществления универсальных пользовательских сценариев или использования интерфейсов программирования приложений, предоставляемых клиентской операционной системой, например, к данной категории относятся возможности геопозиционирования, чтения штриховых кодов, поддержка push-уведомлений

2.1.4 Сценарий работы

Создание кроссплатформенных приложений происходит в интегрированной среде разработки Visual Studio с использованием дополнительного расширения Ubiq Mobile. Готовый сервис публикуется на сервере в выделенном пространстве программиста. В процессе создания

разработчик использует для программирования и тестирования универсальные клиентские приложения. Данные «тонкие клиенты» предоставляют избыточную для конечного пользователя информацию о системе, поэтому используются только в процессе разработки. Для рапространения пользователям и доставки в магазин приложений гораздо удобнее пользоваться персонифицированными клиентами для каждого сервиса. По мере готовности, разработчик запрашивает с помощью расширения Visual Studio или специального web-интерфейса построение конечных приложений на основе универсальных клиентов для целевых операционных систем.

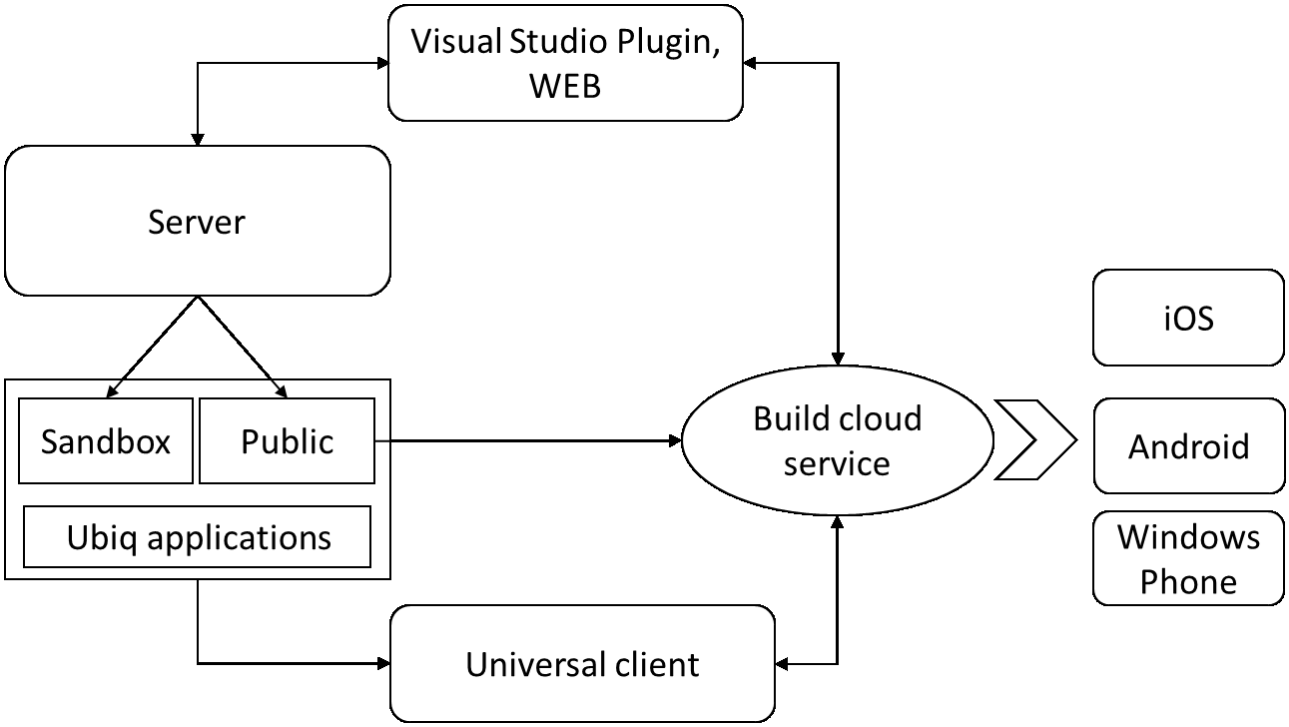


Рис. 3

2.2 Существующие решения

Проблема сборки персонифицированных кросплатформенных приложений появляется в явном виде в решениях с использованием клиент-серверной архитектуры, где непосредственное участие принимают «тонкие» клиенты. В платформах, которые автоматически генерируют

платформозависимый код, конечные приложения проектируются, создаются и тестируются непосредственно самим пользователем-разработчиком. Процесс сборки приложений отличается в зависимости от принципов, на которых построена система кроссплатформенной разработки.

2.2.1 Amazon AppStream

В среде Amazon AppStream трансляция приложений осуществляется с использованием потокового мультимедиа. Конечный клиент решает задачу по декодированию аудио- и видеопотоков и отправляет данные пользовательского ввода и различных датчиков устройства. Персонифицированные клиенты в данной платформе не предоставляются, поэтому от пользователя требуется реализовать конечное клиентское приложение. Для этого платформа предлагает разработчику комплект средств разработки Amazon AppStream SDK. В его состав входят библиотеки для различных платформ, которые решают поставленные перед конечным клиентом задачи. Разработчик в данном случае отвечает за полноценную разработку оставшейся части клиента, включающую в себя проектирование архитектуры приложения, дизайна интерфейсов, тестирования, доставки до пользователя и остальные сопуствующие действия. Таким образом, задачи сборки и персонификации входит в сферу ответственности программиста.

2.2.2 Azure RemoteApp

Среда Azure RemoteApp от Microsoft технологически напоминает платформу Amazon AppStream. В основе лежит известный протокол Remote Desktop Protocol, первая публичная версия которого появилась в 1996 году. Характерное отличие RemoteApp от AppStream заключается в наличии конечных «тонких» клиентов под большинство современных операционных систем. Данные клиентские программы невозможно персонифицировать и загружать отдельными приложениями в магазин. Такой подход сводит на нет

задачу создания индивидуальных клиентов, позволяя сосредоточиться непосредственно на самом транслируемом приложении.

2.2.3 Apache Cordova

Платформа с открытым исходным кодом Apache Cordova состоит из множества платформозависимых библиотек, которые имеют единый интерфейс программирования приложений. Они дают доступ разработчику к специфическим функциям устройств, таким как работа с камерой, геолокация или акселерометр. При использовании сторонних библиотек и платформ для создания пользовательских интерфейсов, такие как jQuery Mobile, Dojo Mobile, Sencha Touch и многие другие, возможна разработка полноценных мобильных приложений с использованием технологий HTML, CSS и JavaScript.

На основе Apache Cordova существуют два известных коммерческих решения: Phonegap от компании Adobe и Icenium от Telerik. Они предлагают программистам дополнительные возможности для создания пользовательских интерфейсов, разработки, отладки, тестирования приложений и другие различные сервисы. В данных средах предлагается два вида сборки приложений под конечные платформы.

Первый тип сборки, локальный, подразумевает создание исполняемого файла на рабочей машине программиста. Сначала происходит процесс компиляции с использованием Apache Cordova в исходный код целевой операционной системы, затем происходит сборка конечного мобильного приложения. Этот вид требует дополнительной установки соответствующего мобильной платформе SDK, например Android NDK, который дополнительное требует наличие JRE и JDK. Для создание приложения в iOS, в силу ограничений компании Apple, потребуется компьютер с установленной системой OS X. Данный тип сборки влечет за собой выполение множества подобных условий.

Второй вид, удаленный, представляет собой сборку в облачном сервисе, на серверах Adobe или Telerik. Для его использования достаточно установленного SDK Phonegap или Icenium. В данном случае исходный код программы упаковывается определенным образом в архив и отправляется в облачный сервис, где происходит процесс локальной сборки. Собранный исполняемый файл предоставляется пользователю для дальнейшего тестирования или загрузки в магазин.

В решениях на основе Apache Cordova персонификация мобильного приложения полностью находится в сфере ответственности разработчика в силу подхода к разработке. В свою очередь, предоставляется возможность унификации процесса сборки и снижаются требования к средствам разработки.

2.2.4 Appcelerator Titanium

Программная платформа с открытым исходным кодом Appcelerator Titanium позволяет создавать кроссплатформенные мобильные приложения с использованием сценарного языка JavaScript. В процессе компиляции происходит «связывание» кода приложения с кодом платформы и последующая сборка исполняемого файла. Программа динамически выполняет исходный Javascript код, используя программу для его обработки (V8 для Android, JavaScriptCore для iOS) и набор заранее скомпилированных стандартных компонент для исполняемой платформы.

Appcelerator Titanium требует персонификации от разработчика, аналогично платформе Apache Cordova. Облачный сборочный сервис отсутствует, поэтому сборка происходит на локальной машине пользователя, что делает актуальными требования к установленному SDK и рабочим станциям.

2.2.5 iBuildApp

Конструктор мобильных приложений iBuildApp использует графический web-редактор для создания кроссплатформенных приложений. В данной платформе пользователь освобождается от написания кода и сборки исполняемых файлов. От разработчика требуется предоставить графические ресурсы, сертификаты и другую административную информацию. По мере готовности программы разработчик отправляет запрос сборочному сервису конструктора на готовые исполняемые файлы. Далее iBuildApp генерирует архивные исполняемые файлы, которые отправляются пользователю.

2.3 Системы непрерывной интеграции

В области разработки программного обеспечания существует практика непрерывной интеграции проекта. Она заключается в периодическом выполнении сборки всех решений для проведения автоматизированного интеграционного тестирования, измерения производительности, профилирования, извлечения и форматирования документации из исходного кода и облегчения ручного тестирования.

Для достижения поставленных задач непрерывная интеграция основывается на следующих принципах:

- использование системы контроля версий для исходых кодов проекта – в ней всегда должна находится рабочая версия программного обеспечения, каждое пополнение основной ветки кодовой базы требует проверки, при этом все разработчики должны ежедневно синхронизироваться для уменьшения конфликтов
- автоматизирования сборка – выполнение построения решения с учетом конфигурации, компиляции исходного кода, сборки бинарного кода, генерация исполняемых файлов и прочие необходимые действия должны выполнятся с ипользованием одной команды
- самотестирование сборки – при каждом построеннии происходит проверка, все тесты дожны подтвержать ожидания разработчика в поведении системы
- тестовая среда – тестирование системы должно производится в копии рабочего окружения, что позволяет обнаружить проблемы при развертывании новой версии
- автоматизированное развертывание – оттестированную систему необходимо развертывать на тестовых средах

При организации системы непрерывной интеграции появляется необходимость в выделенной рабочей станции для выполнения основных задач. Поддержка правильной работы данной практики требует заметных усилий от всей команды разработчиков.

В непрерывной интеграции активно используется принцип автоматизации сборки. Он позволяет избавить построение от лишних ручных действий и зависимости в разработчике проекта. Данный процесс происходит либо с участием сценариев сборки, либо с использованием специальных утилит, например make [16], Apache Ant [17], Gradle [18].

Существуют три вида данного процесса:

- автоматизация по запросу – при возникновении потребности в свежей сборке происходит запуск сценария сборки
- запланированная автоматизация – происходит при запланированной интеграции в определенное время
- условная автоматизация – вызывается при появлении некоторого события, например, изменении кода в основной ветки разработки

При выборе организации облачного сервиса сборки был рассмотрен вариант с использованием системы непрерывной интеграции или автоматизированной сборки, который был признан неудовлетворительным. Во-первых, существующие средства автоматизации предназначены в первую очередь для оптимизации процесса разработки программного обеспечения, нацелены на повышение качества программного обеспечения и представляют собой довольно громоздкие программные продукты. Облачный сервис сборки решает специфичную задачу генерации конечных приложений по запросу пользователя. Во-вторых, при подключении сторонней системы непрерывной интеграции, происходит добавление зависимости в платформу Ubiq Mobile, что накладывает дополнительные условия на её распространение и использованияе.

3. Проектирование и реализация облачного сервиса сборки

Проведя исследование существующих подходов к генерации мобильных приложений, созданных кросплатформенными средствами разработки, было принято решение внедрить в платформу Ubiq Mobile облачный сервис сборки персонифицированных клиентов. Такая технология кардинально упрощает разработку — система в силу своей клиент-серверной архитектуры и наличия универсальных клиентов полностью освобождает пользователя от нативного мобильного программирования. Использование внутреннего сервиса генерации позволяет сохранить целостность системы и инкапсулирует её внутреннее устройство – исходный код клиентских приложений остается в системе и недоступен пользователю.

3.1 Сценарий использования облачного сервиса

В процессе генерации персонифицированных приложений участвуют три подсистемы:

- Visual Studio plugin или web-портал, которые взаимодействуют с разработчиком и создают запросы в к системе сборки
- облачный сервис сборки, который транслирует запросы пользователя службам сборок и возвращает результат
- службы сборки, которые непосредственно конфигурируют и собирают клиентские приложения

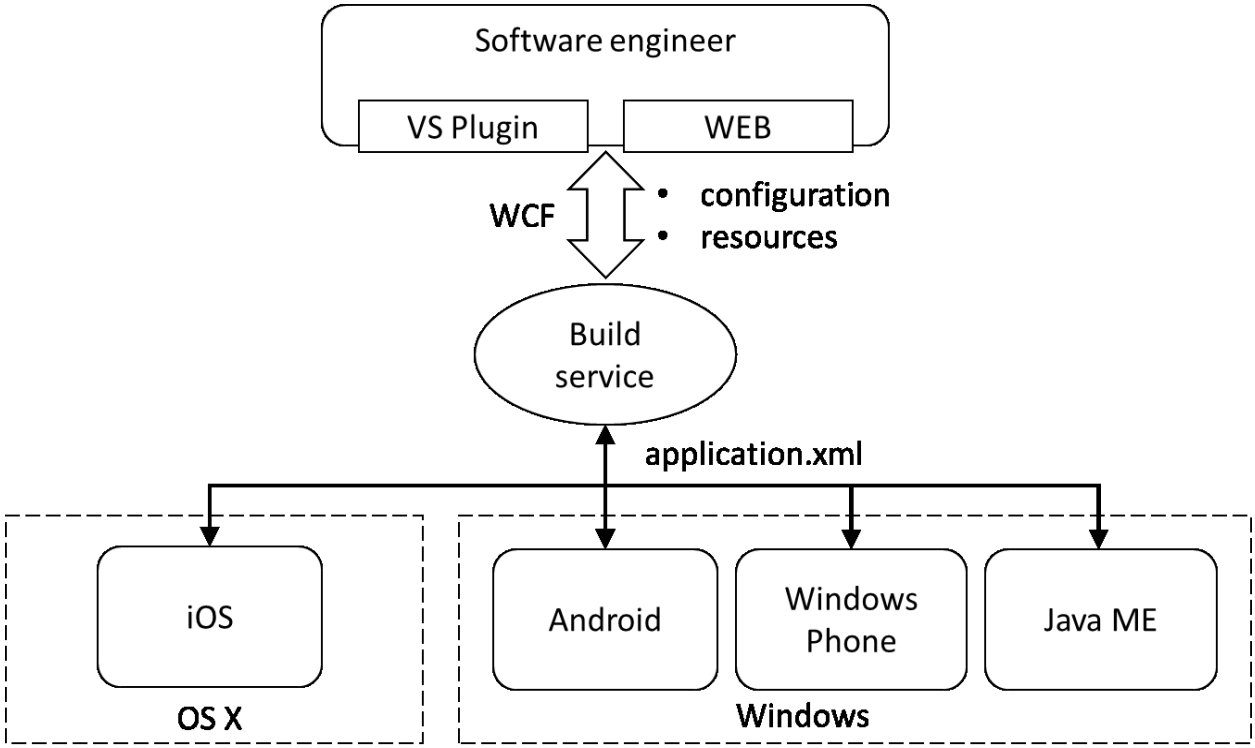


Рис. 4

Согласно сценарию работы с Ubiq Mobile, пользователю в некоторый момент времени требуется создать персонифицированное приложение. На данном этапе он предоставляет всю необходимую информацию о целевых клиентских приложениях, используя либо web-портал, либо среду разработки на базе Visual Studio.

Далее полученные ресурсы и пользовательские параметры передаются в облачный сервис сборки, где они дополняются внутренними системными настройками. Затем происходит создание конфигурационных файлов для выбранных операционных систем, которые передаются службам сборки.

Службы сборки конфигурируют последнюю рабочую версию проекта универсального клиента и производят построение модифицированного решения. По завершению сборки исполняемые файлы передаются обратно в облачный сервис сборки.

Получив ответ от всех вызванных служб сборки, сервис через Visual Studio или web-портал возвращет пользователю, инициировашему запрос, результат сборки персонифицированных клиентов.

3.2 Архитектура облачного сервиса

Основной задачей облачного сервиса является выполнение поступающих запросов на сборку конечных приложений. Эта задача включает в себя следующие подзадачи:

1. Трансляция пользовательских и системных параметров в конфигурационные файлы служб сборок
2. Отправка задач службам сборки и получение ответа
3. Возврат исполняемых файлов в ответ на запросы к облачному сервису

В настоящее время платформа UbiqMobile поддерживает четыре целевые мобильные платформы iOS, Android, Windows Phone и Java ME; впоследствии их число может вырасти. Поддержка каждой системы влечет за собой создание новых классов для каждой из задач. Поэтому в целях удобства интеграции в каждой подзадаче используется фабричный метод, основным достоинством которого является единый интерфейс создания объектов, независимо от их типов и сложности процесса порождения.

Отправка задач службам сборок и получение ответа от них является основной функцией облачного сервиса. При реализации его функциональности используются следующие классы:

- в основе иерархии лежит базовый абстрактный класс BuildTask, внешним интерфейсом которого является метод StartTaskAsync, возвращающий объект Task<BuildTaskResult>
- для каждой операционной системы создается наследник BuildTask, который переопределяет метод StartTaskAsync,

чтобы определить порядок работы со службой сборкой соответствующей операционной системы

- результатом StartTaskAsync служит объект асинхронной операции, которая после выполнения возвращает экземпляр реализации абстрактного класса BuildTaskResult. Он имеет наследников, аналогичных BuildTask, в которых присутствуют фабричные методы, позволяющие узнать результат операции и получить исполняемые файлы
- аргументом метода StartTaskAsync является объект BuildConfig, который инкапсулирует данные параметров конфигурации, общих для всех платформ. Его наследники содержат индивидуальные настройки сборки соответствующей платформы и имеют общие фабричные методы, которые создают файл конфигурации сборки и передают ресурсы, требуемые при сборке клиентского приложения.

Диаграмма классов BuildTask представлена на Рис. 5

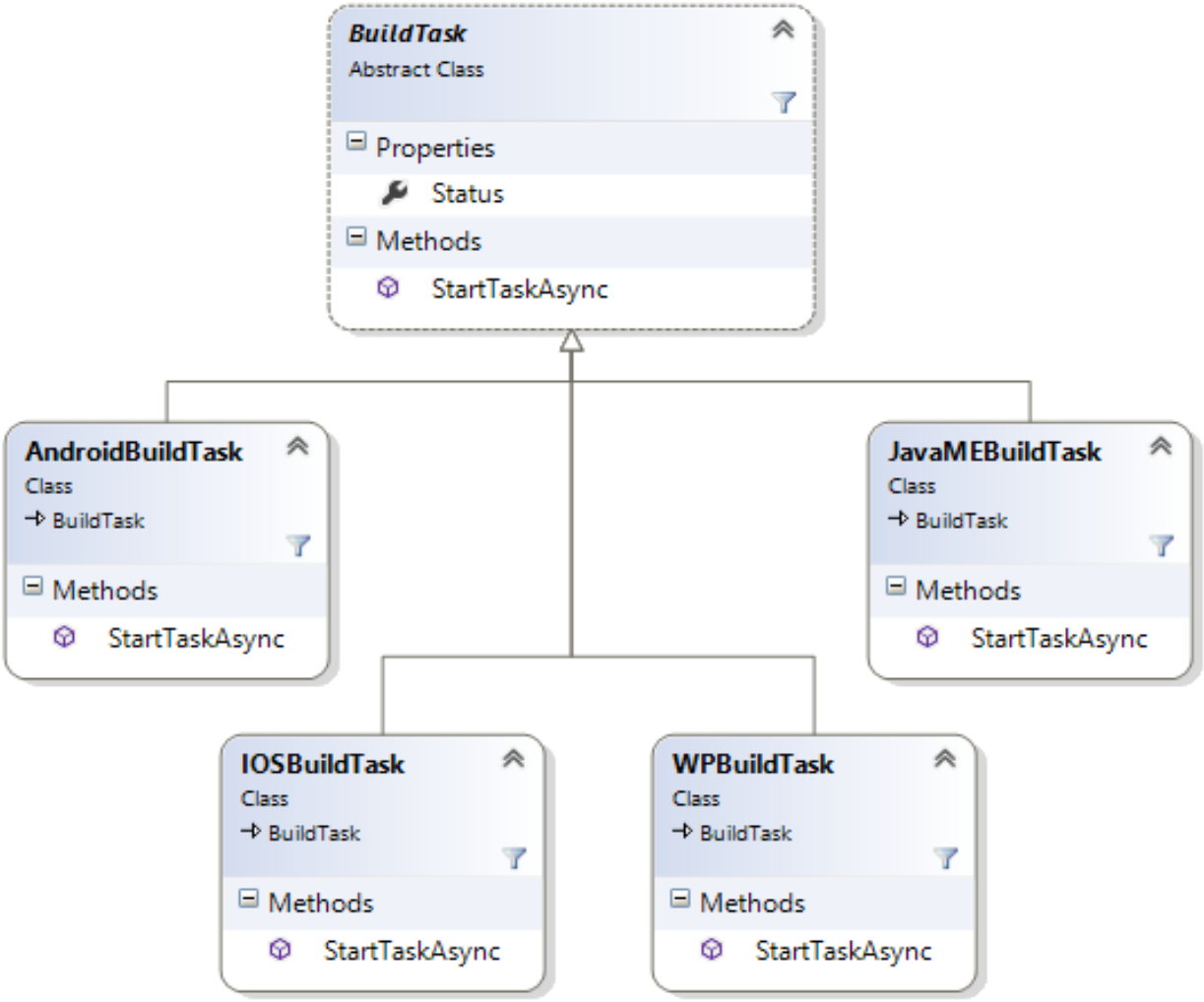

BuildTask
Abstract Class
Properties
Status
Methods
StartTaskAsync
AndroidBuildTask
Class
BuildTask
Methods
StartTaskAsync
JavaMEBuildTask
Class
BuildTask
Methods
StartTaskAsync
IOSBuildTask
Class
BuildTask
Methods
StartTaskAsync
WPBuildTask
Class
BuildTask
Methods
StartTaskAsync

Рис. 5

4. Реализация служб сборок

Персонифицированное клиентское приложение представляет слобой результат сборки проекта универсального клиента, сконфигурированного определенным образом. На вход каждой компоненте службы сборки подаются требуемые настройки, которые содержат в себе такие параметры доступа к требуемому облачному сервису Ubiq Mobile, как адрес сервера, порт, идентификационный номер сервиса и, при необходимости, сертификат. Также настройки включают информацию об используемых возможностях устройства – таких как геолокация, работа с камерой, push-уведомления и остальные подобные функции. В дополнение к файлу конфигурации, на вход подаются различные ресурсы приложения, например, графические файлы иконок, загрузочных экранов.

Перед последующей сборкой проекта происходит присваивание параметрам необходимых значений с помощью сценария (script) для каждой платформы. Затем выполняется построение модифицированного проекта, в результате которого генерируется архивный исполняемый файл, готовый к установке на устройство или отправке в магазин приложений.

4.1 Платформа Windows Phone

В проекте универсального клиента Windows Phone настройки платформы содержатся в файле ресурсов «.resx». Он представляет собой XML Scheme Definition документ, удовлетворяющий требованиям Microsoft ResX Schema. Файлы этого типа обратываются непосредственно в начале построения проекта специальным генератором кода PublicResXFileCodeGenerator. Итогом работы данного инструмента будет исходный код класса со статическими методами, которые возвращают типизированные ресурсы из файла «.resx», таким образом параметры конфигурации компилируются и подключаются к универсальному клиенту статически.

Для доступа к специфическим возможностям устройства, таким, как, например, например геолокация, работа с камерой, push-уведомления, также требуется внесение изменений в xml-подобные файлы проекта «UbiqClient.WP80.csproj» и манифеста приложения «WMAppManifest.xml».

Конфигурирование проекта осуществляется с использованием сценария на языке PowerShell [2]. Оперируя инструментарием работы с xml, он модифицирует файлы настроек, проекта и манифеста приложения, а также распространяет необходимые ресурсы.

Перед вызовом сборки проекта Visual Studio генерирует событие «Pre-build event», обработчиком которого в решении Visual Studio является конифгурационный сценарий. После завершения всех подготовительных процессов происходит сборка сконфигурированного проекта, по результатам которой создается программный «.xap» архив. Данный исполняемый файл с персонифицированным клиентом Ubiq Mobile является результатом работы службы сборки для платформы Windows Phone.

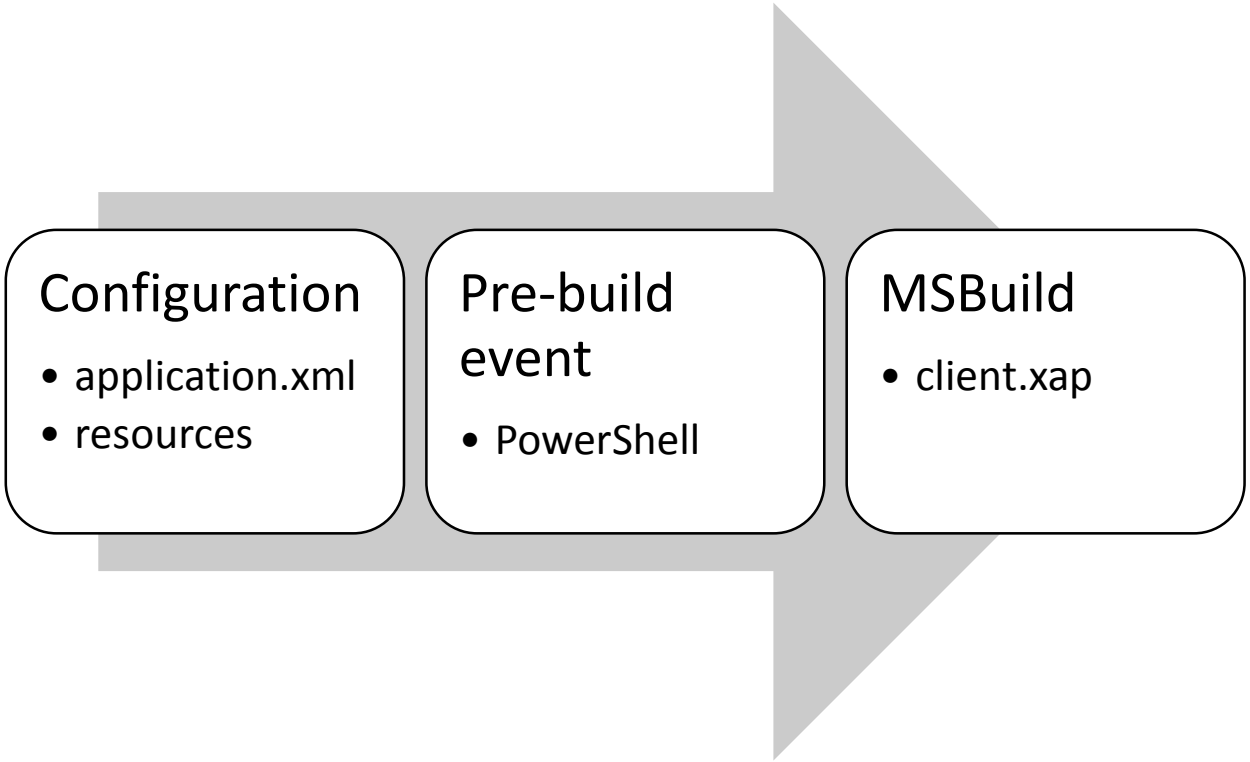


Рис. 6

4.2 Платформа iOS

В проекте универсального клиента iOS параметры среды Ubiq Mobile и приложения хранятся в файле свойств «.plist». Он представляет собой xml-подобный файл с DTD[1], определенным Apple. Файлы этого типа динамически подгружаются в процессе выполнения и преобразуются в объекты класса NSDictionary, который представляет собой ассоциативный массив, храня пары вида (ключ, значение).

Параметры самого проекта хранятся в файле с xml-подобным форматом «.pbxproj», но имеют гораздо более сложную структуру, так как содержат в себе информацию обо всем проекте: списки файлов с исходным кодом, ресурсами, свойствами сборки , виртуальную иерархию папок.

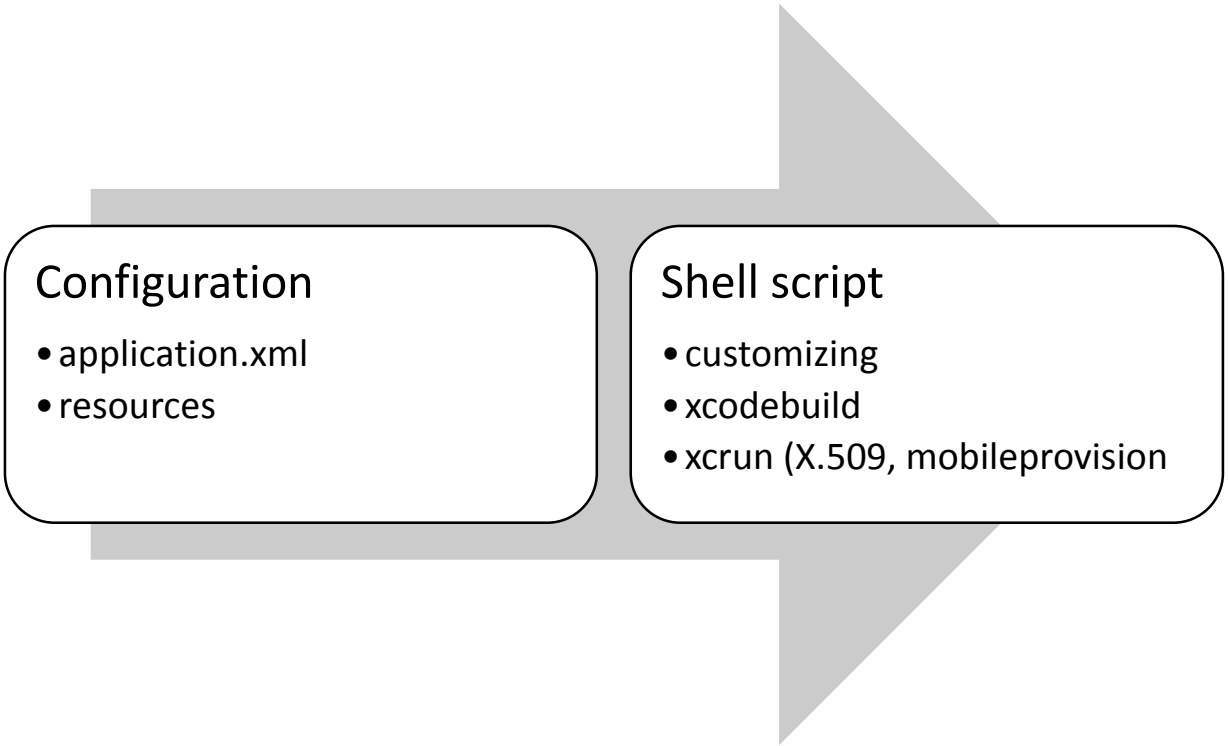
Файл с информацией об исполняемом файле (имя приложения, версия, используемое API операционной системы) хранится аналогично файлу с параметрами среды. Является обязательным объектом в каждом исполняемом архиве.

Конфигурирование проекта универсального клиента для операционной системы iOS происходит с использованием стандартного интерпретатора команд OS X [6]. В данном случае командный сценарий выполняет настройку проекта, сборку исполняемого файла и упаковку в архивный файл.

После того, как конфигурация завершена, происходят процессы компиляции приложения и дальнейшая упаковка в архивный исполняемый файл. На этом этапе необходимо предоставить два сертификата – сертификат разработчика и сертификат распространения (mobileprovisioning certificate). Первый требуется для идентификации разработчика программы. Второй сертификат идентифицирует, удостоверяет приложение и определяет сферу его распространения – разработка, тестирование или публичное использование. Оба файла являются входными данными для компоненты

[1] http://en.wikipedia.org/wiki/Document_type_definition

автоматизаци сборки. В результате работы компоненты автоматизации появляется «.ipa» файл персонифицированного iOS клиента платформы Ubiq



Mobile, готовый к отправке конечному пользователю.

Рис. 7

4.2.1 Идентификация разработчика

Тонким моментом в процессе сборки iOS приложений является подписывание программного пакета сертификатом разработчика [4]. Чтобы его получить, необходимо сгенерировать и занести в систему личный ключ, затем с его помощью создать запрос на подпись, который передается в Apple. В ответ предоставляется «.cer» файл с X.509 сертификатом, который устанавливается в системную связку ключей. Подписывание приложения от лица разработчика-пользователя требует от него передачу сгенерированного платформой Ubiq Mobile запроса на подпись в Apple и обратную загрузку сертификата в платформу. Автоматическая генерация частного ключа и запроса на подпись сертификата происходит один раз для каждого клиента. В процессе используется криптографический пакет openssl с открытым

исходным кодом и работа со связкой ключей через интерфейс командной строки.

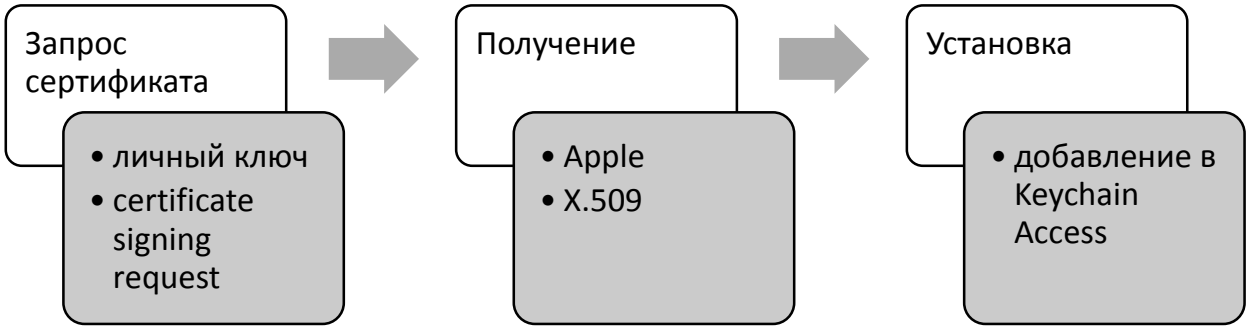


Рис. 8

4.2.2 Удаленный OS X агент

Инструментарий для нативной iOS-разработки существует только для операционной системы OS X. Так как облачный сервис разработан для платформы Windows Server, то он должен уметь производить удаленную сборку данного клиента. Для этих целей был создан OS X агент, реализованный на платформе Mono с использованием WCF [1] сервиса, который при получении запроса на персонифицированный клиент, производит обращение к службе, собирающей приложение и по мере готовности возвращает исполняемый файл.

Запрос к агенту состоит из конфигурационного файла, медиа ресурсов, сертификатов разработчика и распространения. Затем материалы передаются службе сборки, которая и производит персонифицированный клиент.

Другой задачей агента является генерация файлов «.certSigningRequest». При получении запроса от облачного сервиса происходи проверка наличия в системе частного ключа сертификата подписи, ассоциированного с этим пользователем. При наличии личного ключа в системе агент возвращает файл

запроса, который отправляется в Apple. При его отсутствии происходит создание личного ключа, а затем генерация файла запроса.

5. Интеграция в платформу

Для полноценной работы пользователя с облачным сервисом сборки необходима интеграция в систему Ubiq Mobile. Взаимодействие построено на основе модели WCF [1] [3] [7], которая используется для построения сервисо-ориентированной архитектуры. Облачный сервис публикует интерфейс, при помощи которого другие компоненты UbiqMobile могут с ним взаимодействовать. Желающие использовать автоматическую сборку должны добавить ссылку в свой проект на опубликованный интерфейс. Далее WCF автоматически генерирует классы, с помощью которых возможно непосредственное обращение к данному сервису.

Взаимодействие с WEB-порталом

Для проведения тестовой экплуатации было решено создать web-интерейс для облачного сервиса сборки. В нем можно выбрать построение для двух каждого типа сервисов. В глобальном случае приложение выбирается из списка опубликованных на публичном сервере Ubiq Mobile приложений. В варианте с локльным сервисом предоставляется максимально возможная настройка персонифицированных клиентов для приложений, опубликованных в «песочнице» разработчика. В обоих случаях создаются одинаковые конфигурации которые через класс, реализующий сервисный интерфейс поступают в облачный сервис. Когда запрашиваемые сборки произведены, срабатывает событие, уведомляющее о готовности. Web-интерфейс обновляет информацию о построении, в случае готовности уведомляет в личном кабинете и предлагает пользователю загрузить исполняемые файлы.

Заключение

В ходе работы были разработаны компоненты автоматизации сборки персонифицированных клиентских приложений для платформ iOS и Windows Phone. Предложенные реализации позволяют гибко конфигурировать универсальные клиентские компоненты под требования пользователя. Произведена интеграция данных модулей автоматизации в платформу Ubiq Mobile, что позволило собирать персонифицированные мобильные приложения для всех поддерживаемых мобильных платформ, минимизируя рутинные действия. Более детально, в рамках данной дипломной работы были достигнуты следующие результаты:

- спроектирована и реализована архитектура облачного сервиса сборки персонифицированных клиентских приложений
- реализованы службы сборки для платформ iOS и Windows Phone
- облачный сервис интегрирован в платформу Ubiq Mobile

Дальнейшее развитие

Реализованный облачный сервис сборки персонифицированных клиентских приложений можно развивать в следующих направлениях:

- необходимо ввести синхронизацию изменений в универсальном клиенте с персонализированными приложениями
- по мере роста функциональности универсального клиента, потребуется добавлять новые возможности для конфигурации конечных клиентов
- реализовать полный цикл автоматической генерации персонифицированных клиентов, включающий в себя доставку сборок и обновлений до конечного пользователя или магазина приложений

Литература

[1] Juval Lowy, Michael Montgomery. Programming WCF Services, 4th edition // Издательство: «O'Reilly Media», 2015, 900C

[2] Lee Holmes, Windows PowerShell Cookbook: The Complete Guide to Scripting Microsoft's Command Shell, 3rd edition // Издательство «O'Reilly Media», 2013, 1036C

[3] Jeffrey Richter, CLR via C#, 4th Edition // Издательство «Microsoft Press», 2012, 896C

[4] Apple's iOS Development Process Summary, URL: http://www.swfhead.com/AppleiOSDevelopmentProcessSummary.pdf

[5] Терехов А.Н., Оносовский В.В. "ПЛАТФОРМА ДЛЯ РАЗРАБОТКИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ UBIQ MOBILE". Вестник НГУ (серия: информационные технологии)

[6] Shell Scripting Primer, URL: https://developer.apple.com/library/mac/documentation/OpenSource/Conceptual/ShellScripting/Introduction/Introduction.html

[7] MSDN, URL: https://msdn.microsoft.com

[8] Apache Cordova, URL: https://cordova.apache.org/

[9] Appcelerator, URL: http://www.appcelerator.com/

[10] Amazon AppStream, URL: http://aws.amazon.com/appstream/

[11] Corona, URL: https://coronalabs.com/

[12] Microsoft RemoteApp, URL: https://www.remoteapp.windowsazure.com/

[13] Unity, URL: https://unity3d.com/

[14] Xamarin, URL: http://xamarin.com/

[15] Ubiq Mobile, URL: http://ubiqmobile.com/

[16] make, URL: http://www.gnu.org/software/make/

[17] ant, URL: http://ant.apache.org/

[18] gradle, URL: https://gradle.org/