Проблемы выбора языков программирования при разработке кроссплатформенных приложений

Захаров В.Б, Мальковский М.Г., Мостяев А.И.

***Аннотация* — Одним из первых шагов при создании любой программы является выбор языка программирования. А с кроссплатформенными приложениями эта проблема встает особенно остро. В данной работе рассматриваются факторы, влияющие на выбор языка разработки. Приведены примеры использования возможных языков и технологий, подробно описан практический опыт применения некоторых из них при создании реальных приложений.**

***Ключевые слова* — мобильные приложения, переносимость программ, пользовательский интерфейс, шахматные приложения.**

I. Введение

В настоящее время роль персонального компьютера, как единственного инструмента для выполнения программ и общения через интернет, стала заметно снижаться. Появилось множество новых устройств – смартфоны, планшеты, умные часы и телевизоры, другая бытовая техника. Большинство пользователей в своей повседневной жизни используют несколько устройств и хотят использовать знакомые приложения на каждом из них. Для создания программ для различных устройств и платформ существует множество инструментов, зачастую в корне различающихся используемыми технологиями.

Перед разработчиками приложений встает проблема разработки кроссплатформенных приложений, способных работать на разных устройствах и операционных системах. Например, на персональных компьютерах под управлением операционных систем Windows, Linux или macOS, на телефонах, планшетах, телевизорах и умных часах под управлением мобильных операционных систем Android и iOS или в веб-браузерах. Хорошим примером такой универсальной программы может служить программа облачного хранения файлов Dropbox, имеющая приложения для персональных компьютеров, мобильных устройств и веб-браузера.

Но далеко не каждая компания может себе позволить разработку отдельных приложений для каждой из платформ. В связи с этим встает вопрос об оптимизации процесса разработки для нескольких платформ. В настоящий момент программисты не могут исходить в выборе языка программирования по принципу любимого языка, на котором они уже написали много программ. Потребуется предварительная проработка всех вопросов, связанных с созданием программного продукта и принятие взвешенных решений. В данной работе проводится анализ возможных подходов к созданию кроссплатформенных приложений.

Статья получена 16 июня 2017.

Захаров Виктор Борисович, МГУ, ф-т ВМК, н.с. (e-mal: victorldis@gmail.com).

Мальковский Михаил Георгиевич, МГУ, ф-т ВМК, профессор (e-mal: malk@cs.msu.su).

Мостяев Артем Игоревич, МГУ, ф-т ВМК, аспирант (e-mal: reistlin12@gmail.com).

II. Актуальность проблемы

Проблема переносимости сейчас затрагивает большую часть имеющихся на рынке программ. В зависимости от назначения программ [1], проблемы переносимости проявляются по-разному. Рассмотрим возможные варианты.

A. Вычислительные программы

Наименее всего проблема переносимости затрагивает различные вычислительные программы, например, программы расчета математических или физических моделей, которые не требуют сложного графического интерфейса. Чаще всего требуется просто ввести определенный набор данных и получить результаты. Кроме того, часто возникает задача удаленного доступа к вычислительным мощностям (в частности, суперкомпьютерам) на разных платформах через удаленные программы-терминалы, работающие с командной строкой. Такие программы-терминалы используют текстовый ввод-вывод и сравнительно легко переносятся на все платформы.

B. Простые приложения

Другим примером являются программы, реализующие алгоритмы, не требующие больших вычислительных мощностей. Например, алгоритмы кодирования и декодирования аудио- и видео-информации. Мощности современных мобильных устройств хватает для выполнения таких алгоритмов, и многие пользователи хотят иметь возможность работать с данными мультимедиа не только со стационарных компьютеров. Интерфейс указанных программ состоит в последовательном выборе нескольких опций и файлов, то есть, он также достаточно примитивный. Проблема переносимости проявляется в выборе языка программирования, имеющего компиляторы для всех

необходимых платформ.

C. Простые приложения

В следующие группу входят приложения, использующиеся людьми в повседневной жизни. Например, приложения для доступа к электронной почте или обмена сообщениями. Они существуют для всех платформ и предоставляют одни и те же возможности в работе на различных платформах. Причем для каждой платформы создается отдельное приложение. Графический интерфейс данных программ обычно не сложен и чаще всего использует стандартные графические элементы. Проблема переносимости здесь проявляется уже не только в выборе языка программирования, но и в способе организации пользовательского интерфейса, ведь для каждой платформы рекомендованы свои правила построения интерфейса, зависящие от требований операционной системы и физических размеров устройства.

D. Игровые программы

Также можно выделить еще одну большую группу программ — игровые программы. Они в основном используют специальные инструменты для отображения — игровые движки. Здесь для поддержки какой-либо платформы, в первую очередь, необходимо наличие версии игрового движка для нее.

E. Программы со сложным интерфейсом

Самую проблемную группу составляют программы со сложным графическим интерфейсом, требующие активного взаимодействия как различных экранных элементов, так и функциональных блоков программы. Это могут быть аналитические и деловые программы, стратегические и логические игры и многое другое. В создании таких программ важную роль играют дизайнеры пользовательского интерфейса. Указанная группа программ требует наибольших усилий по переносу, так как, с одной стороны, существует множество форм-факторов используемых устройств, а с другой стороны, концепции дизайна фирм-производителей (Apple, Google, Microsoft) сильно отличаются друг от друга.

Из краткого описания видно, что для любого современного приложения, независимо от его назначения, возникает потребность решения проблем кроссплатформенности на самых ранних этапах разработки. В следующих главах будут описаны возможные подходы к решению проблемы переносимости.

III. Выбор языка программирования

Компании-поставщики операционных систем или платформ предоставляют разработчикам свои средства для разработки приложений. Далее в статье мы будем называть эти средства ***родными***. Родные средства базируются на каком-либо языке программирования и включают в себя набор системных библиотек. Ключевым, применительно к теме данной работы, является язык программирования, выбранный компанией-поставщиком.

Так, компания Microsoft предоставляет набор языков программирования для разработки приложений для Universal Windows Platform (единая платформа, работающая на персональных компьютерах, телефонах и планшетах) — C++, C#, Visual Basic и JavaScript [2]. Компания Apple предлагает разработчикам использовать языки программирования Objective-C и Swift для создания программ для операционных систем macOS, iOS, watchOS and tvOS [3]. А компания Google предлагает использовать язык Java [4] для разработки программ для операционной системы Android.

Несколько иная ситуация с созданием веб-приложений. Традиционно они разделяются на две составляющие - клиентскую (front-end) и серверную (back-end), для каждой из которых может использоваться свой язык программирования. Для клиентской части в подавляющем большинстве случаев используется язык JavaScript. Для серверной части набор языков программирования просто огромен. Наиболее распространенные из них – это PHP, C#, Java, C++, Scala, Ruby, Python (Django), JavaScript (Node.js). Хотя есть и довольно экзотические варианты вроде Hack, Erlang, Haskell и Prolog. Такое обилие связано с тем, что общение между клиентской и сетевой частями стандартизовано, и реализация серверной части может быть выполнена на любом языке, лишь бы она предоставляла интерфейс доступа, соответствующий стандарту.

Мы видим, что основные языки программирования для разных платформ различаются. Тем не менее, практически каждый из указанных языков в той или иной степени доступен почти на всех платформах, вопрос только в качестве его реализации на неродных платформах.

Достаточно популярно стало использование наборов интегрированных библиотек (фреймворков), позволяющих вести разработку приложения на одном языке программирования для нескольких платформ. На данный момент существует несколько фреймворков. Описание возможностей и принципов работы каждого из них требует отдельной статьи, поэтому приведем краткое описание наиболее распространенных из них, различающихся языком разработки, чтобы показать неоднозначность в выборе языка программирования даже при выборе фреймворка.

Одним из самых распространенных фреймворков является Qt, позволяющий создавать приложения на языке C++ для множества платформ – Unix, Windows, macOS, Android, iOS [5]. Он предоставляет встроенные библиотеки для взаимодействия с платформой на языке C++, а также встроенный редактор интерфейса.

Распространенным вариантом кроссплатформенных фреймворков являются реализации, использующие веб-технологии, в частности, язык JavaScript. Так как сейчас никакая платформа, предназначенная для широкого круга пользователей, не может существовать без возможности отображения веб-содержимого, то программы, написанные на языке JavaScript и

использующие HTML и CSS для отображения, могут быть запущены практически на всех платформах. Существуют специальные библиотеки, которые упрощают процесс создания таких приложений и расширяют их возможности. Например, Apache Cordova для мобильных платформ и React для веб-сайтов и персональных компьютеров.

Другим примером кроссплатформенного фреймворка является сочетание пакетов Delphi RTL и FireMonkey. Существуют реализации для платформ Windows, Unix, macOS, iOS и Android [6]. Фреймворк состоит из двух частей – среды выполнения (Delphi RTL) и кроссплатформенной библиотеки интерфейса (FireMonkey).

Опять же отдельно стоит выделить средства для создания игровых программ. Они, как правило, уже базируются на фреймворке или движке, который является кроссплатформенным. Примером может служить игровой движок Unity, предлагающий разработку на языке C#. Движок доступен для множества платформ, включая Windows, Unix, macOS, iOS и Android [7].

Каждый из фреймворков обладает определенными достоинствами и недостатками. Так, Qt и Delphi обеспечивают хорошую производительность программ, но для поддержки графического интерфейса требуются бОльшие ресурсы, чем это требуется для родных приложений. JavaScript обеспечивает самую лучшую переносимость графических элементов под разные форм-факторы устройств, но более-менее сложные программы, написанные на JavaScript заметно тормозят, особенно на устаревших устройствах. Программы на C# зачастую плохо поддерживают весь набор библиотек, встроенных в операционную систему.

Получается, что выбор языка программирования для создания многоплатформенного приложения очень неоднозначный.

- Для каждой платформы можно использовать её родной язык программирования.
- Можно использовать хорошо освоенный разработчиками приложения их любимый язык, но постоянно решать проблемы, связанные с его некачественной реализацией на какой-либо платформе. Проблема не только в том, что группа программистов привыкла к определенному языку, а в том, что на этом языке уже написаны значительные объемы кода. И встает вопрос, использовать его как есть, или переписывать на другой язык.
- Можно использовать универсальный фреймворк, но каждый из них имеет как определенные достоинства, так и значимые недостатки, которые могут не позволить создать программу приемлемого качества. Каким бы заманчивым не было решение использовать единый код для всех платформ, стоит четко понимать, что фреймворки не могут быть также хороши как родные средства. Во-первых, в них с опозданием вносится поддержка новых функций операционных систем. Во-вторых, они добавляют слой программного кода. Наличие этого слоя как ведет к некоторой (а иногда и значительной) потере производительности, так и вызывает неизбежные ошибки, борьба с которыми вызывает постоянную головную боль у разработчиков.

Но обязательно ли ограничивать выбор только одним языком разработки? Ведь для каждой из платформ выбран свой язык по какой-то причине. Может использования нескольких языков не стоит так категорично избегать? В следующей главе будут рассмотрены варианты совмещения нескольких языков программирования при создании кроссплатформенных приложений.

IV. Совмещение нескольких языков программирования

Любая программа, как только она выходит за рамки одного программного модуля или класса, становится представима в виде сочетания нескольких составных частей.

В программе можно выделить основную часть, не зависящую от конкретной платформы. Если говорить очень обобщенно, то это множество алгоритмов, реализующих основные функции программы, так называемая бизнес-логика приложения. Данная часть является основной составляющей программы, делающей эту программу значимой, весомой для пользователя. Поэтому к ней предъявляются повышенные требования по корректности, надежности и отказоустойчивости. Соответственно, на создание этой части требуются значительные усилия команды разработчиков, и возникает естественное желание использовать единственную, проверенную и надежную реализацию этой бизнес-логики на всех поддерживаемых платформах.

Но приложение состоит не только из бизнес-логики. В нем также есть и интерфейс взаимодействия с пользователем (пользовательский интерфейс). Как правило, пользовательский интерфейс зависит от платформы. Компания-поставщик платформы предоставляет специальные инструменты для построения пользовательского интерфейса, а в процессе работы программы он управляется программным кодом, написанным на одном из языков, предоставляемым поставщиком платформы. То есть, интерфейс является частью приложения, зависящей от конкретной платформы.

A. Реализация бизнес-логики

Принимая во внимание выводы, сделанные выше, можно считать оправданным разделение приложения на бизнес-логику и интерфейс, где бизнес-логика является кроссплатформенной частью. К счастью, для подобного разделения есть несколько доступных решений. Например, язык C# широко используется в программировании для Windows и Web, а также имеет специальную интегрированную среду разработки (IDE) Xamarin для создания приложения для мобильных

платформ Android и iOS. Кроме того, с недавнего времени, язык поддерживается на macOS и Unix. Иными словами, бизнес-логика, написанная на языке C#, может быть использована на всех популярных платформах, и, с помощью специальных средств разработки, подключена к родному интерфейсу системы.

Другим языком кроссплатформенного программирования является C++. Так как первая его версия была создана давно (относительно других языков, предлагаемых для разработки на современных платформах), то при выходе более новых языков программирования (Java, Objective-C, C#) создателям приходилось оставлять возможность подключения модулей на C++. Отчасти, это было вызвано тем, что при появлении этих языков большое количество алгоритмов уже были реализованы на C++. К тому же, язык C++ отличается лучшей производительностью, нежели языки с автоматической сборкой мусора, поэтому части программ, требующие высокой производительности, по-прежнему было необходимо разрабатывать на C++.

В результате, сейчас программный код на языке C++ можно использовать на любой популярной платформе, в сочетании с другим языком программирования. Для Windows и Unix это осуществляется через динамически подключаемые библиотеки (.dll и .so), для Android с помощью технологии Java Native Interface (JNI), а для macOS и iOS компилятор языка Objective-C полностью поддерживает компиляцию языка C++. А новый язык Swift для разработки для платформ macOS и iOS поддерживает связь с языком Objective-C. Для подключения C++ к языку JavaScript есть несколько возможных вариантов — инструменты Emscripten и WebAssembly, позволяющие кросскомпилировать язык C++ в JavaScript и родные расширения для Node.js (Native Addons) [8].

В качестве примера кроссплатформенного использования языка C++ можно привести алгоритмы кодирования и декодирования мультимедиа (JPEG, mp3, ffmpeg), реализованные на языке C++, которые используются почти ежедневно любым человеком, работающим с компьютером.

B. Реализация сетевых функций

Также стоит отметить тот факт, что большинство современных приложений используют сетевые функции, которые разделяются на клиентскую и серверную части.

Взаимодействие с сервером ведется по заранее установленному протоколу, поэтому выбор языка программирования для основной программы не влияет на выбор серверного языка программирования. Для реализации серверной части доступен большой набор языков программирования, и здесь разработчики легко могут выбрать наиболее подходящий.

Другой вопрос состоит в реализации общения с сервером – использовать ли для него кроссплатформенный язык или нет. Однозначного ответа на этот вопрос нет. Родные языки программирования предоставляют библиотеки для работы с сетью, которые могут использовать различные настройки и особенности системы. Кроссплатформенные средства не всегда могут обеспечить весь набор возможностей. Но эти возможности не всегда необходимы для минимальных сетевых функций, поэтому окончательное решение зависит от требований к сетевой функциональности программы. Здесь тоже возможно смешанное решение. Например, формировать данные для сетевых запросов в кроссплатформенной части, а затем передавать их с помощью родных библиотек для каждой платформы.

V. Пользовательский интерфейс

Помимо бизнес-логики и сетевой части, в приложениях есть еще одна большая составляющая - пользовательский интерфейс. Его качеством никогда нельзя пренебрегать при создании приложений, ориентированных на конечных пользователей, а удобство и простота интерфейса облегчают жизнь как пользователям, так и разработчикам.

Можно выделить 3 основных технологии реализации пользовательского интерфейса:

A. Использование родных для платформы графических элементов.
B. Использование языка JavaScript и технологий HTML и CSS. Для краткости будем их далее называть JavaScript-технологиями.
C. Использование промежуточного графического движка, заменяющего родные графические средства (OpenGL, FireMonkey, Qt).

A. Родные технологии позволяют создавать интерфейс, наиболее отвечающий рекомендациям разработчиков платформ. Программы выглядят более привычными для пользователей, они знают, где искать необходимые функции. Кроме того, доступен большой дополнительный набор компонент, созданных как производителями платформ, так и сторонними разработчиками. Родной интерфейс при работе требует минимальных ресурсов и отличается хорошей реакцией. Единственный недостаток – для каждой новой платформы интерфейс программы требуется создавать заново в соответствии с требованиями этой платформы, её редактора интерфейса и родного языка программирования.

B. Напротив, JavaScript-технологии теоретически позволяют добиться полной переносимости. Браузерные движки хорошо стандартизированы (по сравнению с родными средствами), они установлены фактически на всех платформах. Правда, отметим, что для совместимости со всеми устройствами разработчикам приходится ориентироваться не на самые современные версии этих движков, а на более ранние, что снижает возможную функциональность.

Из недостатков, в первую очередь, отметим, что язык JavaScript изначально ориентирован на интерпретацию исходного кода, что определяет медленное выполнение программ, написанных на этом языке. Но, так как браузерные движки являются важнейшей составной частью всех платформ, то разработчики движков

прикладывают значительные усилия для того, чтобы повысить эффективность выполнения JavaScript-программ. В частности, широко используется компиляция на лету. Это и многое другое позволяет добиться приемлемой скорости работы JavaScript-приложений, хотя отставание от других технологий по быстродействию часто остается заметным.

C. Использование промежуточного графического движка, в первую очередь, востребовано в приложениях, в которых присутствуют сложные графические объекты, с элементами движения и наличием слоев. В частности, хорошо подходит для подобных целей библиотека OpenGL. Использование же подобных движков для написания общего пользовательского интерфейса хотя и решает задачу переносимости, делает интерфейс приложений непохожим на родной интерфейс устройства. Разработчикам промежуточных движков приходится постоянно успевать за разработчиками операционных систем, реализующих всё новые и новые возможности. Чаще всего поддержка новых функций в промежуточный слой включается с заметным опозданием. Всё это может значительно усложнить разработку и сделать приложение, полностью написанное с использованием промежуточного движка, менее конкурентноспособным.

Как мы видим, каждая из технологий имеет как достоинства, так и недостатки. Соответственно возникает вопрос о совмещении в одном проекте разных технологий. Технически это реализуется с помощью добавления специальных функций-обработчиков событий для графических элементов. При этом функции-обработчики событий и функции отображения графических элементов могут быть написаны на разных языках программирования. В программе могут одновременно существовать формы с родными графическими элементами, HTML-формы и формы, использующие специфический графический движок.

Можно дать следующие рекомендации по средствам для реализации пользовательского интерфейса.

Если интерфейс сравнительно простой, то его можно сделать по любой из технологий, но лучше использовать родные средства, так как объем переносимого кода небольшой, а приложение будет смотреться гораздо лучше и использовать меньше ресурсов.

При наличии в приложении таблиц, картинок, размеченных текстов идеально подходит технология JavaScript. При этом JavaScript-элементы разумно включать в общую структуру графического интерфейса, совмещая родные и JavaScript-формы в общих контейнерах. Если обработка событий от JavaScript-элементов простая, то она обрабатывается непосредственно JavaScript-функциями. Сложную или критичную по времени выполнения обработку лучше реализовать в функциональной части программы.

В общем случае в целях переносимости надо пытаться реализовать на JavaScript как можно большую часть пользовательского интерфейса. Но следует иметь в виду, что JavaScript может не поддерживать специфические функции и не работать со всеми графическими элементами, доступными на устройстве, поэтому эта технология не подходит, если подобная функциональность существенно нужна. Дизайн, доступный в HTML-разметке, может не соответствовать принятому на устройстве дизайну. Также в JavaScript будет трудно поддерживать сложное взаимодействие функциональной части и графических элементов. В лучшем случае потребуется кодировать и отлаживать сложную логику взаимодействия частей программы, в худшем случае это приведет к значительному торможению программы и связанных с этим неудобствам.

Если от графики требуется высокая скорость, в частности, поддержка анимационных функций, то следует использовать соответствующий игровой движок. Его как можно включить в контейнеры родного интерфейса, так и полностью написать приложение на графическом движке.

VI. Практический опыт

Несмотря на все возможные плюсы, рекламируемые в описаниях технологий, на практике ситуация может кардинально измениться из-за неожиданно возникающих технических проблем. Поэтому стоит обратиться к примерам использования кроссплатформенных технологий при создании реальных приложений.

Первый опыт применения кроссплатформенных технологий авторами статьи был получен при создании мобильного приложения "Шахматная Планета" для платформы Android в 2011 году. Он заключался в применении веб-технологий для создания пользовательского интерфейса приложения. Бизнес-логика была реализована родными средствами на языке Java. В качестве достоинства выбранного подхода следует отметить, что процесс разработки шел относительно быстро, потому что у команды был опыт работы с веб-технологиями, а приложение для Android создавалось впервые, и родные технологии создания интерфейса не были хорошо освоены. Также веб-технологии хорошо себя зарекомендовали для реализации элементов интерфейса, с низким уровнем взаимодействия с пользователем. То есть, те элементы, которые нужны для отображения информации, а не для взаимодействия. Например, в приложении есть элемент "Нотация", необходимый для отображения ходов в шахматной партии, его реализация через веб-технологии не вызывает неудобства у пользователя. Совершенно противоположная ситуация возникла с элементом "Шахматная доска" - пользователю необходимо быстро получать реакцию на свои действия. Например, при игре на время или при перемещении фигуры. Реализация этого элемента с использованием веб-технологий плохо показала себя в плане производительности, после чего данный элемент был сделан с использованием родных средств.

В процессе разработки были выявлены и следующие неудобства [9]:

1) Основной проблемой стала скорость работы и

отзывчивость интерфейса на действия пользователя. Элементы интерфейса, реализованные через веб-технологии, давали значительно большую задержку на нажатия, чем родные элементы интерфейса. В результате, заметное время после нажатия на элемент, на экране ничего не происходило.

2) Другой проблемой стало динамическое изменение информации в процессе работы. При необходимости обновить какой-либо элемент интерфейса, нужно вызывать функции JavaScript программно через веб-терминал (консоль). При изменении нескольких элементов или области большого размера, пользователю были видны различные артефакты. Например, при отображении информации в виде списка, если необходимо добавить элемент в начало, то все последующие необходимо сдвинуть, что происходит заметными скачками.
3) Возникали проблемы с разными версиями браузеров на разных версиях операционных систем. Часто возникали ситуации, что одна функция работала на одной версии, но не работала на другой. Или в одной версии программа работала корректно, а в другой возникала ошибка либо в форматировании, либо в работе

Следующим шагом стало выделение части бизнес-логики в кроссплатформенный модуль в 2013 году. В качестве языка реализации был выбран язык C++, как широко распространенный и знакомый команде разработчиков. С языка Java на язык C++ был перенесен модуль, отвечающий за хранение и представление шахматной партии. В его функции входит хранение текущей позиции и ходов партии, добавление и удаление ходов, проверка шахматных правил, а также загрузка и сохранение партии в текстовый формат (PGN). В модуль было перенесено значительное количество функций, а суммарный размер исходных файлов модуля получился около 200 кб. После перенесения части бизнес-логики с языка Java на язык C++ скорость работы приложения визуально мало изменилась. С одной стороны код на C++ выполняется быстрее, с другой стороны, при вызове функций C++ из Java есть накладные расходы. Основным минусом такого подхода стала возросшая сложность структуры проекта, но плюсы от перехода все равно гораздо значительнее.

Большой опыт кроссплатформенного программирования был получен при создании в 2014 году приложения "Таблицы Ломоносова" [10], используемого для доступа через сервер к таблицам семифигурных шахматных окончаний. У приложения есть реализации для платформ Android, iOS, Web и Windows. В таблицах шахматных окончаний хранятся точные оценки (ничья или выигрыш/проигрыш в указанное число ходов) всех позиций с семью и меньшим количеством шахматных фигур. Особая сложность возникает из-за того, что финальный размер файлов таблиц составляет более 100 терабайт [11], из-за чего их невозможно предоставить в полном объеме для локального доступа обычным пользователям. Поэтому было принято решение, что часть таблиц может использоваться локально, а часть удаленно через веб-сервер.

Шахматные таблицы хранятся в особом формате, алгоритм чтения которого был реализован на языке C++. Алгоритм был подключен к родным приложениям для платформ разработки Android, iOS и Windows, а также к веб-серверу для удаленного доступа. При этом, приложение для Android было реализовано на языке Java, для iOS на Objective-C, для Windows на Delphi, а веб-сервер на C#. Подключение кроссплатформенного модуля к родному приложению для Android было реализовано с помощью технологии JNI, для iOS - Objective-C++, для Windows код на языке C++ был скомпилирован в виде dll библиотеки и подключен к программе на языке Delphi с помощью метода статического связывания, и к веб-серверу на языке C# с помощью метода динамического связывания. Приложения могли работать как с локальными файлами таблиц, так и с веб-сервером, протокол общения с которым был реализован с использованием формата XML. Таким образом, приложение "Таблицы Ломоносова" получило наибольшее количество реализаций для нескольких платформ и подтвердило оправданность использования языка C++ в качестве базового для кроссплатформенной части проекта.

Другим примером является шахматная обучающая программа "Пешка", имеющая реализации для платформ Android, iOS, Web и Windows. Программа предоставляет пользователю возможность решать шахматные задачи, а также читать теоретические уроки в удобном интерактивном виде. База задач и уроков хранится в программе в особом формате cke. Алгоритм чтения этого формата реализован на языке C++, а визуальное представление – с помощью родных средств для каждой платформы. Технология подключения кроссплатформенного модуля к родным приложениям такая же, что и при локальном доступе к файлам в "Таблицах Ломоносова". Но показательным примером использования кроссплатформенного кода в этой программе является компонент "XMLPlayer". Данный компонент предоставляет возможность разыгрывания обучающих сценариев по шахматной партии. В его функции входит: автоматическое проигрывание ходов в партии, запрос правильного хода в позиции, отображение подсказок при ошибках, подсчет набранных очков, разыгрывание нескольких вариантов ходов в партии. Все эти функции требуют показа информации пользователю, а некоторые (вопросы) еще и непосредственного взаимодействия. Получается, что необходимо реализовать механизм асинхронного общения кроссплатформенного модуля с программным кодом на родных средствах. Данная задача была решена путем введения протокола общения между модулями и стандартизации интерфейсов доступа к ним. При необходимости отображения инфрмации, XMLPlayer вызывает метод интерфейса в реализации визуальной части. А при получении ответа от пользователя, визуальная часть вызывает необходимый метод XMLPlayer. Таким образом, визуальная часть была

реализована отдельно для платформ Android, iOS, Web и Windows, но все реализации работают с кроссплатформенной реализацией XMLPlayer.

Кроссплатформенные средства также были использованы в веб-приложении "Игровая зона Chess King", с помощью них была реализована игра в шахматы против компьютера. Алгоритм игры в шахматы написан на языке C++ и используется в приложениях для платформ Android и iOS с использованием технологий, описанных выше. Чтобы подключить его к веб-приложению был использован инструмент Emscripten, также известный как WebAssembly, являющийся кросс-компилятором языка C++ в подмножество языка JavaScript. Для вызова функций, реализованных на C++, из кода на языке JavaScript, необходимо добавить в C++ проект отдельный модуль с перечислением внешних функций, никаких особых преобразований в коде C++ при этом делать не нужно. Единственной проблемой такого способа является высокая сложность отладки полученного веб-модуля, потому что он является результатом автоматического преобразования программы на языке C++.

Также к кроссплатформенным технологиям можно отнести вынесение части бизнес-логики программы на сторону сервера. Причем, благодаря единому стандарту доступа к веб-приложениям, реализацию программы можно делать практически на любом языке программирования. Эту особенность можно использовать для подключения готовой кодовой базы команды программистов, написанной на другом языке программирования. Например, в веб-приложении "Игровая зона Chess King" используются модули, написанные на языке Delphi, использовавшиеся в более ранних продуктах. Одним модулем является "Игровая компонента", реализующая алгоритм онлайн-игры в шахматы двух соперников. Общение с "Игровой компонентой" реализуется через веб-сокеты, а информация передается в формате JSON. Другим модулем, реализованным на языке Delphi, является алгоритм жеребьевки игроков турнира по швейцарской системе. Алгоритм был реализован на языке Delphi много лет назад, и активно используется в программе "Шахматная Планета". Так как в нем довольно много тонких моментов, а программа использует много особенностей языка и информацию о низкоуровневом представлении структур данных, то было принято решение использовать готовый модуль, добавив в него специальные функции для веб-доступа по технологии ISAPI.

VII. Заключение

После изучения проблем выбора языков и технологий кроссплатформенного программирования можно констатировать тот факт, что имеется достаточно большое количество различных подходов, но ни один из них не является идеальным. Окончательное решение по выбору технологий должно приниматься, исходя из требований к программе и возможностей компании. В качестве примеров были использованы программы, в разработке которых авторы статьи принимали участие.

Для получения качественных приложений разработчикам приходится вникать в тонкости программирования на используемых платформах и изучать их функциональность. В статье мы описали, чем нужно руководствоваться для принятия решений о реализации различных программ и их частей, окончательное же решение нужно принимать разработчикам, исходя как из имеющейся базы программного кода, знания разработчиками тех или иных технологий и требований, предъявляемых к программным продуктам.

Кроме того, отметим, что обильно появляющиеся в последние годы языки программирования и их версии не только не решают проблему переносимости, но и усугубляют ее. И если для создания требовательных к скорости выполнения частей программного кода легко сделать выбор в пользу языка C++, то для написания программного кода общего назначения однозначный выбор сделать очень трудно. Серьезно назрел вопрос о договоренности компаний – разработчиков платформ о родной поддержке, по крайней мере, одного общего для всех платформ языка программирования. Главными чертами этого языка должны быть как простота и поддержка современных конструкций языков программирования, так и, самое главное, беспроблемная стыковка с такими языками, как JavaScript, Java, Swift, C++. Последнее условие является необходимым, так как только в этом случае пользователи смогут постепенно переходить на новый единый язык, не отказываясь от имеющегося кода.

Библиография

[1] Головин И. Г., Захаров В. Б., Мостяев А. И. Влияние тенденций современного общества на процесс создания, распространения и поддержки программ для мобильных устройств // Научный взгляд в будущее. - 2016. - Т. 4, № 2. - С. 41-51.

[2] Выбор языка программирования [Электронный ресурс] // Центр разработки для Windows [Официальный веб-сайт]. URL: https://docs.microsoft.com/ru-ru/windows/uwp/porting/getting-started-choosing-a-programming-language.

[3] Баклин Д. Профессиональное программирование приложений для iPhone и iPad / Джин Баклин ; [пер. с англ. ООО "Айдиономикс"]. - М. : Эксмо, 2013. - 672 с.2.

[4] Майер Р. Android 4. Программирование приложений для планшетных компьютеров и смартфонов / Рето Майер ; [пер. с англ. ООО "Айдиономикс"]. - М. : Эксмо, 2013. - 816 с.

[5] Supported Platforms [Электронный ресурс] // Qt Documentation [Официальный веб-сайт]. URL: http://doc.qt.io/qt-5/supported-platforms.html.

[6] Supported Target Platforms [Электронный ресурс] // RAD Studio [Официальный веб-сайт]. URL: http://docwiki.embarcadero.com/RADStudio/Berlin/en/Supported_Target_Platforms.

[7] Многоплатформенность [Электронный ресурс] // Unity [Официальный веб-сайт]. URL: https://unity3d.com/ru/unity/multiplatform.

[8] Захаров В., Мальковский М., Мостяев А. Java или альтернативы? Опыт переноса приложений на платформу Android // Сборник научных трудов SWorld. - 2015. - Т. 5, №1(38). - С. 15-27.

[9] Захаров В., Мостяев А. Особенности переноса приложений на мобильные платформы // Программные системы и инструменты / Под ред. А. Н. Терехин. - Т. 15. - Издательский отдел факультета ВМК МГУ Москва, МГУ, 2014. - С. 16-24.

[10] Захаров В. Б., Мальковский М. Г., Мостяев А. И. Успехи шахматной информатики и возможность полного решения

задачи игры в шахматы // Евразийский Союз Ученых (ЕСУ). - 2016. - Т. 3, № 1 (22). - С. 124-126.

[11] Zakharov V. B., Mal'kovskii M. G., Shchukin Y. V. Compression of underdetermined data in a 7-piece chess table // Moscow University Computational Mathematics and Cybernetics. - 2016. - Vol. 40, no. 1. - P. 47-52. Block compression algorithms used for solving the problem of 7-piece chess endings are presented. The algorithms are based on data reordering before compression, the RE-PAIR compression algorithm, and use of the so-called underdetermined value method.

Programming Language Choice Problem in Cross-platform Application Development

Zakharov V.B., Mal'kovkij M.G., Mostyaev A.I

Abstract **- One of the first steps in the process of software development is programming language choice. In case of cross-platform application development the choice problem becomes especially actual one. This work considers a lot of factors influencing on language choice. The examples of using different languages and technologies are described. The practical experience of creating real applications is considered in details.**

Keywords**: chess applications, mobile applications, program compatibility, user interface**.