электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.05

Проблемы использования веб-технологий при разработке мобильных приложений

***Гаврилова М.А.**, аспирант*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»

***Скворцов В.А.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»

Научный руководитель: Пролетарский А.В., д.т.н., профессор
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»
chernen@bmstu.ru

Введение

Революция в мобильных технологиях меняет нашу жизнь — меняет то, как мы учимся, играем, общаемся и работаем. Меняется все: коллеги, время и место работы, а также способы ее выполнения и то, зачем мы, собственно, работаем. Этот новый технологический уклад обусловлен стремительным развитием инноваций и такими ключевыми глобальными тенденциями, как смена поколений, урбанизация, многофакторная производительность и рост среднего класса в развивающемся мире. Изменения происходят стремительно и прямо сейчас.

В современных условиях бизнесу необходимо гибко и быстро приспосабливаться к требованиям рынка. Даже незначительные изменения в процессах могут повлечь переработку множества информационных систем, которые изначально были разработаны как монолитные хранилища. Для сохранения конкурентоспособности, затраты на поддержку должны постоянно снижаться, а системы постоянно эволюционировать[11-13].

Сравнение различных типов мобильных приложений

Есть много факторов, на которые стоит обратить внимание при разработке мобильных приложений, например, такие как навыки командной разработки, требуемая функциональность устройства, работа в автономном режиме, совместимость, важность обеспечения безопасности и др.

В настоящее время, выделяют три типа приложений:

1. Нативные приложения.
2. Web(HTML5)-приложения.
3. Гибридные приложения.

Для нативных приложений специфическим является то, что их необходимо разрабатывать под разные мобильные платформы(*IOS, Windows Phone или Android*) с использованием соответствующих сред разработки, языка и платформы(например, Xcode и Objective-C с прошивкой, Eclipse и Java с Android, Windows Phone SDK). Также эти приложения могут запрашивать доступ к вашим контактам, доступ к календарю и присылать уведомления.

Ещё одним явным отличием от веб и гибридных приложений является то, что нативные приложения обеспечивают лучшее удобство, лучшие черты и лучшие общий мобильный опыт[2-3].

Web(HTML5)-приложения используют стандартные веб-технологии HTML5 и CSS(Cascading Style Sheets). При разработке в таком подходе создаются кросс-платформенные мобильные приложения, которые работают на нескольких устройствах[10]. Обычно это реализуется следующим образом: весь контент и интерфейс пишутся при помощи веб-технологий. В результате получается определённый набор связанных между собой html-страниц, которые можно перенести без изменений на любую платформу. Веб-приложения выигрывают в сравнении с остальными приложениями в понятности и доступности информации. Содержимого намного больше обнаружить в Интернете, чем в нативном приложении: Когда у людей есть вопрос, они идут в поисковую систему, вводят их в строку запроса, далее выбирают страницу из результатов поиска. Но они не заходят в магазин приложений, чтобы сначала найти подходящее для них приложения, загрузить его, а потом найти ответ на вопрос[1-3].

Кроме всего сказанного, мобильные приложения не занимают дополнительного места в памяти и хранилище устройства, что даже для современных устройств является огромным плюсом. В сферу развития высокоскоростного интернета (например, 4G и 5G) скорость отклика на запрос пользователя сравниться с нативным приложением.

Гибридная разработка сочетает в себе лучшее (или худшее) двух предыдущих типов приложений(например, кроссплатформенность и возможность использования ПО телефона). Данный подход позволяет создать часть (а иногда и все) компонентов кроссплатформенными, и при этом получить доступ к различным возможностям устройства, сохранив хорошую скорость работы. Однако здесь всё зависит от того, каким инструментом будут пользоваться разработчики[2-4].

В настоящее время наиболее известными и популярными фреймворками для разработки подобных решений являются PhoneGap и Xamarin[10].

Есть два способа реализации гибридного приложения[2]:

Локальное - можно упаковать HTML(HyperText Markup Language) и код JavaScript внутри мобильного двоичного приложения, аналогично структуре нативного приложения. В этом случае используется REST API, для перемещения данных назад и вперёд между устройством и облаком.

Сервер - В качестве альтернативы вы можете реализовать полный веб-приложение с сервера (с дополнительным кэширования для лучшей производительности), просто используя контейнер в виде тонкой оболочки над UIWebView[2].

Если оценивать по критерию безопасности гибридные приложения, то они удачно совмещают высокую безопасность и простоту разработки. Но есть небольшой нюанс, который относится и к HTML5-приложениям. При таком многообразии версий браузеров в различных версиях iOS и Android безупречную работу HTML5 и гибридных можно гарантировать не на каждом из устройств, так как некоторые функции могут отсутствовать в ранних версиях браузеров[3].

В таблице описаны преимущества и недостатки для этих типов приложений [1-3].

	Преимущества	**Недостатки**
Native	• Максимальная функциональность и скорость работы • Не требуется интернет- соединение для использования • Имеет доступ к ПО смартфона (GPS, плеер, камера, multi tuch, отпечатки пальцев) • Шифрование данных • Простота использования • Большое количество документации • Распространение через магазины приложений • Получение оригинального UI для каждой платформы.	• Выше стоимость и длиннее сроки разработки • Требует от разработчика знаний определённой среды программирования • Работает только с одной платформой • При косметических изменениях необходимо выпускать обновление
Web (HTML5)	• Кроссплатформенность • Не требует загрузки из магазина мобильных приложений • Можно легко адаптировать обычный сайт • Легче найти веб-разработчика нежели разработчика под определённую платформу • Простота создания и поддержки	• Требует подключения к интернету • Не имеет доступа к ПО смартфона • Не может отправлять push-уведомления • Должен быть запущен интернет-браузер • При продаже требуется использование своей платежной системы

	Преимущества	Недостатки
	• Не требует обновления	• не все веб-приложения оптимизированы под мобильные телефоны • Проблема хранения данных • Проблема с обеспечением необходимой безопасности
Hybrid	• Функциональность нативного приложения на независимой платформе • Запускается не из браузера в отличии от веб приложения • Работает на всех устройствах • Возможность независимого обновления • Распространение через магазины приложений • Может работать как в онлайне, так и в офлайне Использует камеру телефона, геолокацию • Можно настроить push-уведомления.	• Загружается из магазина мобильных приложений (необходимо соответствовать требованиям) • Разработчик должен быть знаком с разными API • ограниченность в функциональных возможностях • ненативный интерфейс • необходимость создавать различные компоненты с нуля, когда можно было бы использовать стандартные

Как видно из таблицы, каждый тип приложений имеет свои достоинства и недостатки.

Гибридные и нативные приложения должны соответствовать определённым правилам, для того чтобы их можно было опубликовать в магазине приложений. Поддержка гибридных приложений обходится дешевле, так как программный код является один для всех платформ.

На данный момент не существует средства, которое можно было бы с чистой совестью назвать настоящей кроссплатформенной средой для разработки мобильных приложений, но в современном мире выбор инструментов для разработки может удовлетворить вкус любого разработчика. Однако нужно понимать, для каких целей и для кого вы создаёте приложение. Как же человеку определиться с выбором приложения?

Если для работы Вашего приложения необходимо использовать мощности устройства или для его работы очень важна скорость обработки информации (игры, социальные сети, геолокационные сервисы, сервисы обмена фотографиями и т.д.), то необходимо делать нативное приложение. Если скорость работы не так важна, то делайте лучше гибридное приложение. А если вам просто важно, чтобы у пользователя была возможность получить всю необходимую информацию при наличии интернета, и его это устраивает, то можно обойтись веб-приложением.

Виды ограничений и пути их решения

Среди основных препятствий, сдерживающих распространение мобильных онлайн-сервисов, выделяют[7] следующие:

- Невозможность прямого переноса «компьютерных» сервисов на мобильные устройства, связанная с разницей в форм-факторе и вычислительных возможностях мобильных устройств по сравнению с «обычными» компьютерами.
- Более высокая сложность разработки мобильных приложений по сравнению с традиционным программированием.
- Разнообразие несовместимых между собой платформ.

Кроме этого, существует и ряд других ограничений, которые будут рассмотрены ниже. Одним из основных ограничений является использование Cookies. Cookies представляет собой информацию, хранящуюся в виде пар ключ/значение в текстовом файле или в памяти и отправляемую сервером в браузер. Содержимое Cookies используется создавшим его Web-приложением. В мобильной сети, использование Cookies может отрицательно сказаться на производительности приложения. Для статических ресурсов не нужна поддержка Cookies, поэтому производительность может быть улучшена за счёт использования другого домена, ограничения доступа к Cookies, использование другого пути для статических ресурсов основного приложения [5].

Подделка Cookies – это изменение его содержимого после выполнения Web-приложения. Общепринятым подходом продвинутого механизма защиты является использование цифровых подписей, гарантирующих отсутствие изменений хранилища текстовых файлов. Еще одним решением – является защита Cookies путем шифрования при передаче для снижения риска изменения и перехвата [8].

Также следует с осторожностью использовать API (application programming interface), так как браузер находясь в состоянии онлайн не может гарантировать последующего удачного соединения с приложением [5].

При первом входе в приложение пользователю необходимо предоставить информацию, о том какие данные будут использоваться в приложении, и каким образом эти данные будут обмениваться с сервером. Если пользователь отказывается от предоставления данных, то необходимо предусмотреть возможность автоматического восстановления.

Также, используя перенаправления запросов в приложении могут возникать задержки. Для того, чтобы избежать данной проблемы, количество перенаправлений должно быть сведено к минимуму. К примеру, не использовать редирект, оптимизировать

сетевые запросы, свести к минимуму внешние ресурсы, использовать стандартные методы кэширования, такие как кэш AJAX(Asynchronous Javascript And Xml) данных, а также ограничить количество информации в DOM(Document Object Model)[9].

Особое внимание следует уделить ряду факторов, которые влияют на запуск приложения, таких как: ожидание, методы взаимодействия и согласованности данных. Для решения проблемы можно использовать автономные технологии (например, web-технологии AppCache, Consider Partitioning Large Scripts, использовать локальное хранилище данных, минимизировать количество запросов локального хранилища данных. Для снижения имеющихся задержек необходимо разрешить добавочную визуализацию, избегать перезагрузки страницы и предварительно загружать вероятные просмотры. При проектировании пользовательского интерфейса следует учитывать такие способы взаимодействия как: browser focus "jumps", нажатие на основание и указатель. Оптимальная конфигурация элементов пользовательского интерфейса изменяется в зависимости от способа взаимодействия, используемого в устройстве. Пользовательский интерфейс должен быть разработан на основе знаний о методах взаимодействия, поддерживаемых целевым устройством. Для борьбы с этим, необходимо классифицировать целевые устройства и создать единый вариант приложения для каждого класса. Так как сетевой трафик в мобильном устройстве истощает батарею и может потребовать расходов – важно проинформировать об этом пользователя.

Чаще всего таким атакам подвергаются веб-приложения. К этим атакам, направленным на самые распространенные уязвимости, относятся к межсайтовому скриптингу, SQL-инъекциям, фальсификации, отравлению cookie и утечки информации. Традиционные системы защиты периметра, такие как сетевые экраны и системы обнаружения вторжений, не защищают от атак подобного рода, поскольку эти атаки используют уязвимость программ. Мобильные веб-приложения во многом подвержены тем же уязвимостям, которые присущи настольным веб-приложениям. Рассмотрим пути решения проблем с безопасностью.

При межсайтовом вызове скриптов вредоносный код внедряется в аутентичный Web-сайт. Если в HTML-код отображаемой страницы можно вставить входные данные HTTP-запроса, сайт открыт для таких атак. Для предотвращения выполнения межсайтовых скриптов (XSS): не отображайте пользователям непроверенные входные данные, удаляйте вредоносный код из входных и выходных данных, кодируйте выходные данные для предотвращения выполнения браузером [7,9].

Внедрение SQL тоже связан с использованием уязвимостей в запросах и направлен на вставку SQL-выражений в поля веб-приложения, предназначенных для ввода данных.

Возможность вставлять запросы в поля ввода позволяет злоумышленнику обойти механизмы аутентификации веб-сайта и получить доступ к базе данных. Для предотвращения этой атаки: выполняйте очистку данных(например, посредством фильтрации или использования белого списка), не доверяйте данным, которые вводит пользователь.

При утечке информации настойчивый злоумышленник может исследовать приложение, пытаясь обнаружить уязвимости. Утечку информации необходимо рассматривать в контексте приложения, и лучшей защитой является компетентность разработчика. Существуют различные меры по смягчению последствий утечки информации, которые следует рассмотреть разработчикам[6,9,10].

- Удаляйте из HTML-кода все комментарии, сообщающие что-либо о приложении.
- Не отображайте в браузере конкретные исключительные ситуации.
- Не раскрывайте информацию о неудачной аутидентификации.
- Настройте Web-сервер и сервер приложений на предотвращение произвольной навигации по Web-приложению.

Модификация параметров нацелена на управление параметрами, передаваемыми в приложение. Защита от такой атаки включает в себя проверку параметров и тщательный анализ логики приложения.

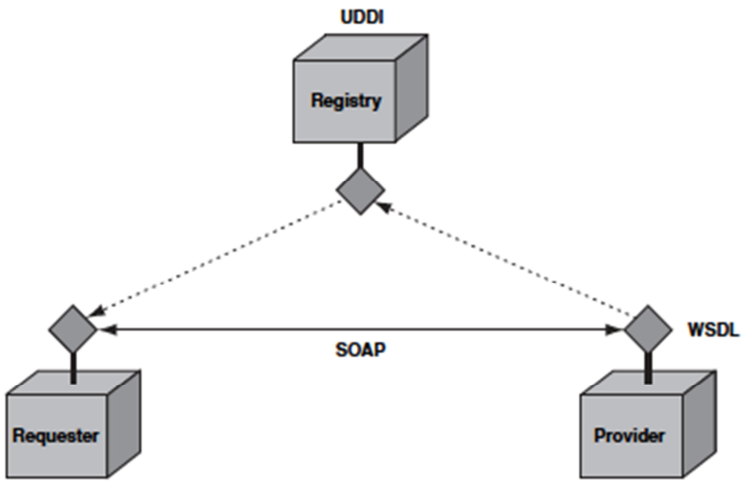
Следующей проблемой угрозы безопасности является использование ненадёжных данных JSON. Хотя это и мощный метод, который на устройствах с ограниченными Eval () может выполняться быстрее, чем альтернативы, но непосредственное исполнение в Datafeed представляет собой значительный риск безопасности данных. При случайном исполнение вредоносного JavaScript под угрозу потери данных может попасть личная информация пользователя, такая как: контактные данные, местоположение. Для решения этой проблемы вместо разбора данных JSON, выполненных с помощью функции Eval (), используйте JSON парсер [7,8].

Сервис-ориентированная архитектура и Веб-сервисы

SOA(Service-oriented Architecture) делает возможным переход от монолитных систем к сервис-ориентированным[3]. Сервис-ориентированная архитектура(SOA) — это метод построения корпоративной программной инфраструктуры, позволяющий разным приложениям обмениваться данными и процессами. Новизна SOA заключается в том, как моделируется инфраструктура архитектурного решения, основанная на сервисах, вместо

фокусирования на всём приложении. Сервисы это небольшие обособленные элементами ПО, которые решают одну задачу и могут повторно использоваться во многих приложениях. Таким образом, SOA — это не продукт и даже не технология, а концепция создания и интеграции отдельных корпоративных приложений[11].

Архитектура SOA полезна тем, что для обмена информации используется набор трех базовых стандартов. В их число, кроме WSDL(Web Services Description Language) и UDDI(Universal Description Discovery & Integration), входит протокол SOAP(Simple Object Access Protocol) (см. рис.) — простой механизм для создания структурированных пакетов данных, предназначенных для обмена информацией между сервисами (сетевыми приложениями). Эту тройку стандартов объединяет то, что все они построены на базе языка XML и являются открытыми. Это говорит о том, что развитием этих стандартов занимаются независимые комитеты по стандартизации. Интерфейсы сервисов обычно описываются с помощью контракта — документа на языке WSDL [11-12]. Основные стандарты веб-сервисов хороши для некоторых SOA-приложений, но не подходит для многих других [17].



При этом предоставление данных сервисов пользователю осуществляется через специальный Web-интерфейс, реализованный на портале и позволяющий пользователю «из одного окна» вызывать большое количество различных сервисов, в том числе из различных приложений, территориально удаленных центров обработки данных и прочих сервисов. Данный пользовательский интерфейс может обеспечить безопасную и удобную работу пользователя с большого количества «оконечных» устройств — персональных компьютеров, карманных ПК, мобильных телефонов[12,16].

Web-сервис представляет собой инкапсулированный программный компонент, включающий два элемента: интерфейс сервиса и реализацию сервиса. Реализация служит

для выполнения одной или нескольких функций, для которых предназначен сервис. Интерфейс является более интересным элементом. Ведь основная идея SOA состоит в том, что сервисы вызывают друг друга, и именно интерфейс задает способ программного взаимодействия между ними. Контракт задает характеристики сервиса и определяет, как потребитель может к нему обращаться. Поэтому интерфейс должен содержать достаточно информации, чтобы найти сервис, понять, что он делает, и использовать, не вникая во внутреннее содержание[13].

Контракты составляет так называемый провайдер сервиса. Чтобы потенциальные потребители могли найти нужный сервис, провайдер регистрирует его с помощью спецификаций UDDI в специальном каталоге — реестре сервисов. В архитектуре SOA такие UDDI-реестры играют роль посредника-брокера[12-14].

Эти три компонента — провайдер сервиса, потребитель сервиса и реестр сервисов — и составляют классическую модель SOA. Помимо этих неотъемлемых частей реальная система может (и должна) включать другие элементы — программы промежуточного слоя, средства управления, защиты, мониторинга и т. д. Главное в модели и основная изюминка SOA в том, что между потребителем сервиса и его провайдером нет жесткой связи[13].

Важно понимать, что решение основанное на SOA не обязательно реализовывать с помощью Web-сервисов, и, наоборот, внедрение Web-сервисов не приводит автоматически к созданию SOA — необходимо наличие указанных выше трех главных компонентов этой архитектуры. Также выбор конкретной технологии реализации зависит от нужд и возможностей конкретного предприятия или организации[13-14]. К примеру, если у предприятия запланирован плановый переход на SOA реализацию, то такой переход может быт облегчён при использовании ПО сторонних компаний. Более подробно реализация данного ПО описана в статье[14].

SOA имеет достаточно положительных сторон, но при этом не всегда сможет интегрироваться в ваш бизнес. Среди плюсов подхода можно обозначить следующие пункты[13-15]:

- возможность построения сложных систем путем интеграции сервисов от различных производителей независимо от платформ и технологий;
- принцип слабых связей помогает организовать взаимодействие с унаследованными системами(старыми системами);
- возможность снижения затрат на сопровождение и высокая интероперабельность систем за счет опоры на стандарты;

- повышение гибкости и улучшение масштабируемости благодаря простоте создания множества сервисов путем интеграции существующих приложений;
- существенно снижает затраты на поддержку имеющихся решений;
- при необходимости можно разработать стандартный протокол общения между системами;
- доступ к данным не зависит от географии пользователя, для доступа можно использовать мобильные средства связи;
- возможность мобильного доступа к данным и инкрементальных обновлений для более быстрого решения задач заказчиков путем внедрения новых сервисов.

К минусам данного подхода можно отнести[13-15]:

- сложности с реализацией асинхронной связи между приложениями и сервисами;
- большое время отклика, трудности организации обмена большими объемами; данных, обусловленные тем, что XML дает надежность, но не скорость (существуют альтернативы XML — в частности, JSON);
- наличие уязвимостей в безопасности из-за совместного участия в процессах множества приложений и систем;
- потребность в сложных механизмах управления транзакциями при взаимодействиях между логически раздельными системам.

Перейти на SOA не просто, и на больший предприятиях это знают, поэтому особо и не торопятся переходить на использование данных сервисов. Для максимальной эффективности и гибкости рекомендуется инкрементально переводить традиционные среды на SOA.

Внедрение SOA не требует полной перестройки корпоративной инфраструктуры, поэтому нет необходимости отказываться от хорошо себя зарекомендовавших приложений. Достаточно снабдить их соответствующими интерфейсами. Поэтому на SOA нужно переходить медленно, например начиная с небольших пилотных проектов.

Благодаря Web-сервисам сотрудники компании могут гораздо активнее участвовать в создании корпоративных приложений. Сейчас некоторые компании(к примеру, UnitSpace) реализовали программное обеспечение промежуточного слоя BSR, которое позволяет адаптировать приложения к SOA, создавая Web-сервисы на основе заданных бизнес-аналитиками метаданных, не используя программирования[13-15].

На текущем уровне развития SOA не является решением всех проблем, связанных с разработкой ПО, потому что существует такие проблемы как: производительность, безопасность, понимание бизнес-процессов, поддержка собственные услуг и приложения и т.д. которые необходимо решать. В основном решение этих проблем зависит нового

поколения интеграции продуктов BEA от IBM, IONA, Microsoft, SAP, SeeBeyond, Systinet, Tibco, WebMethods. Следующим шагом на пути простой интеграции SOA решения будет использование облачных сервисов. «Облака» предоставляют возможность получения необходимых ресурсов по запросу, а также имеют богатый функционал по хранению и обработке данных. В этом контексте iPaaS (integration platform as a service) набирают популярность как подходящее решение для широкого спектра интеграционных задач. iPaaS – это набор облачных сервисов, которые позволяют пользователям создавать, управлять и организовывать интеграционные потоки для многих приложений или данных без установки специализированного оборудования или библиотек[14-17].

Заглядывая в будущее, исследователи из компании Garther предсказывают что в 2016 году около 35 процентов компаний будут использовать iPaaS (Integration platform as a service) решения в том или ином виде. Однако эксперты не говорят, что iPaaS заменит SOA. Традиционные решения с SOA будут востребованы для сценариев, где важна скорость обработки сообщений и идет интенсивный обмен данных между базами и бизнесами в корпорации.

Список литературы

[1]. Гибридные, Нативные и Мобильные web приложения: взгляд изнутри. Режим доступа: http://magora-systems.ru/hybrid-vs-mobile-web-vs-native-applications, (дата обращения 20.11.2015)

[2]. Korf M., Oksman Eu. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options. Available at: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options/, accessed 11.11.2015.

[3]. Новожилова А. Мобильные приложения: нативные vs html5 vs гибридные Режим доступа: http://www.cmsmagazine.ru/library/items/moblile/native-vs-html5-vs-hybrid/ (дата обращения 20.11.2015).

[4]. Budio R. Mobile: Native Apps, Web Apps, and Hybrid Apps http://www.nngroup.com/articles/mobile-native-apps/, accessed 13.09.2015.

[5]. Connors A., Sullivan B. Mobile Web Application Best Practices. Available at: http://www.w3.org/TR/2010/REC-mwabp-20101214/, accessed 25.11.2015.

[6]. Gerhardt W., Kumar N., Lomardo A., Loucks J., Buckalew L. Next-Generation Knowledge Workers. Available at:

http://www.cisco.com/web/about/ac79/docs/sp/Business-Mobility.pdf, accessed 26.07.2015.

[7]. Терехов А.Н., Оносовский В.В. Технология разработки мобильных онлайн сервисов. Режим доступа: http://2011.secr.ru/2011/md/terekhov-presentation.pdf (дата обращения 20.11.2015).

[8]. Santiago C.E., Hondo M. Web 2.0 desktop and mobile application security design http://www.ibm.com/developerworks/rational/library/desktop-mobile-application-security-design/index.html?S_TACT=105AGX99&S_CMP=CP/, accessed 17.11.2015.

[9]. Устимов А.И. Архитектура клиент-серверных мобильных приложений // Молодёжный научно-технический вестник. Электрон. журн. 2015. № 4. Режим доступа: http://sntbul.bmstu.ru/doc/773931.html (дата обращения 04.11.2015).

[10]. Ермаков О.Ю. Способы кроссплатформенной разработки мобильных приложений // Молодёжный научно-технический вестник. Электрон. журн. 2015. № 5. Режим доступа: http://sntbul.bmstu.ru/doc/779205.html (дата обращения 04.11.2015).

[11]. Колесов А. SOA — проверка временем пройдена. Режим доступа: http://www.pcweek.ru/its/article/detail.php?ID=122848/ (дата обращения 20.11.2015).

[12]. Крушкин С. Стратегия перехода к SOA. Режим доступа: http://www.osp.ru/cio/2007/12/4651815/ (дата обращения 04.11.2015)

[13]. Гореткина Е. Непростой путь от Web-сервисов к SOA. Режим доступа: http://www.crn.ru/numbers/spec-numbers/detail.php?ID=12017/ (дата обращения 04.11.2015).

[14]. Serrano N., Hernantes J., Gallardo G. Service-Oriented Architecture and Legacy Systems. IEEE Software magazine, February, 2015, IEEE Computer Society. Available at: http://www.infoq.com/articles/service-oriented-architecture-and-legacy-systems/, accessed 20.08.2015.

[15]. Serrano N., Hernantes J., Gallardo G. Service-Oriented, Architecture and Legacy Systems. IEEE Software, September/October 2014, IEEE Computer Society. Available at: http://www.osp.ru/os/2014/08/13043486/, accessed 10.08.2015.

[16]. Service-oriented architecture (SOA) vs. Component based architecture. Available at: http://petritsch.co.at/download/SOA_vs_component_based.pdf/, accessed 20.08.2015.

[17]. SOA with Web Services. Available at: http://www.aw-bc.com/samplechapter/0321180860.pdf, accessed 20.08.2015.