УДК 004.4

АРХИТЕКТУРА МОБИЛЬНОГО ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ОБРАБОТКИ СИГНАЛЬНЫХ ДАННЫХ

В.Э. БАЗАРЕВСКИЙ

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220000, Беларусь

Поступила в редакцию 5 сентября 2012

Предложена архитектура программной системы, предназначенной для обработки вибрационных данных, получаемых от различных технических объектов, состоящей из мобильной, настольной и веб-частей. Рассматриваются проблемы оптимальной организации вычислений на клиентской и серверной частях приложения. Предложен способ монетизации мобильных веб-приложений, базирующийся на использовании механизма плагинов для PhoneGAP.

Ключевые слова: архитектура веб-приложений, мобильные приложения, цифровая обработка сигналов, HTML5, PhoneGAP.

Введение

Расширение пропускной способности каналов передачи данных и развитие технологий браузерной визуализации позволяет создавать интерактивные приложения по обработке и отображению сложной графической информации. В качестве задачи, подлежащей решению в рамках подобных приложений, может рассматриваться задача визуализации вибросигналов, а также их спектров, каскадов спектров, вейвлетов, и других результатов преобразований в рамках браузерного HTML5 приложения [1]. Такое приложение может использоваться для просмотра, анализа и сбора данных пользователями, не являющимися экспертами, с целью выявления дефектов обследуемых механизмов на основе ряда заранее сконфигурированных преобразований, а также при экспресс-анализе состояния этих механизмов экспертами в условиях, когда отсутствует возможность использования специализированных, но при этом значительно более тяжеловесных приложения для анализа вибросигнальных данных.

На рисунке показана диаграмма способов использования подобного приложения.

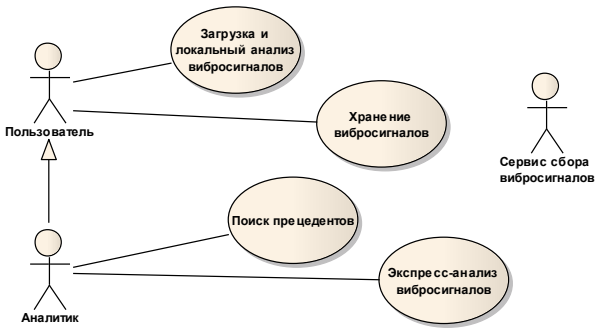

Рис. 1. Диаграмма способов использования веб-приложения обработки сигнальных данных

В приложении можно выделить три независимых роли: пользователь, аналитик и сервис сбора вибросигналов. Пользователь имеет возможность загружать вибросигналы, произво-

дить их локальный анализ и запрашивать произведение анализа экспертом. Роль эксперта предполагает расширение роли пользователя с добавлением возможности просмотра и поиска прецедентов, подобных исследуемой ситуации, в базе прецедентов, построенной на основе снятых ранее выборок вибросигнальных данных. Роль сервиса сбора вибросигналов автоматическая и предполагает постоянный сбор вибродданных со специализированных объектов для дальнейшего их анализа и построения базы прецедентов. Компонентная диаграмма приложения показана на рисунке.

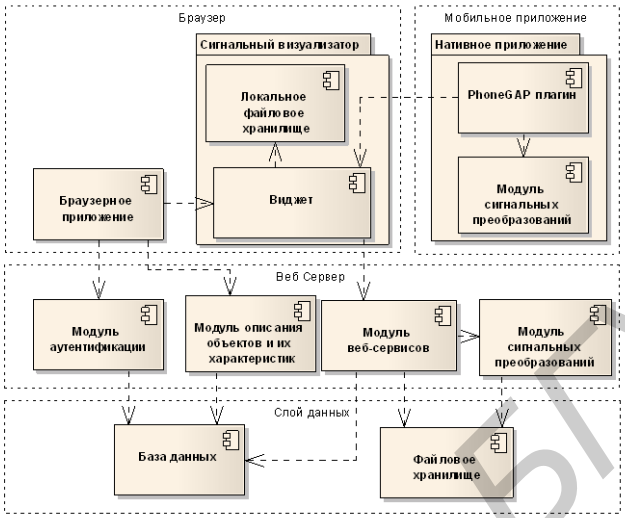

Рис. 2. Компонентная диаграмма веб-приложения для обработки сигнальных данных

Приложение можно разделить на три слоя: слой представления, логики и хранения данных. Слой представления, в свою очередь, можно разделить на веб-приложение и мобильное приложение.

Наиболее существенной частью слоя представления является сигнальный визуализатор, состоящий из локального хранилища вибросигнальных файлов и компонента визуализации вибросигналов. Такой визуализатор может быть довольно легко встроен в существующие приложения путем добавления нескольких html-тегов в разметке приложения.

Мобильное приложение содержит в себе функционал веб-приложения и расширяет его рядом сигнальных преобразований, защищенных от копирования на основе функционала соответствующих провайдеров мобильных платформ.

Слой логики содержит стандартный для веб-приложений модуль аутентификации, модуль веб-сервисов для интеграции с мобильными приложениями и сигнальным визуализатором, модуль описания объектов и их характеристик, а также модуль сигнальных преобразований, необходимый для построения базы прецедентов путем серверной обработки собранных вибросигнальных выборок.

Слой данных содержит базу данных, содержащих данные о пользователях системы, характеристиках обследуемых объектов, прецедентах, а также ссылки на файлы вибросигнальных выборок.

Процесс создания подобного приложения может быть разделен на несколько этапов, каждый из которых может рассматриваться как разработка отдельного модуля приложения и его интеграция с существующими модулями. В данной статье рассмотрены элементы системы, наиболее специфичные для сектора мобильных веб-приложений.

Модуль преобразований, серверная и клиентская часть на Javascript

Приложение для обработки и визуализации вибросигналов требует наличия библиотеки для цифровой обработки сигналов, содержащей ряд преобразований, способных выполняться как на клиентской, так и на серверной стороне. Указанное требование обосновано, в первую очередь, спецификой клиент-серверного решения, когда большие массивы данных хранятся на серверной стороне, но должны отображаться на клиентской стороне, таким образом требуя пересылки больших объемов данных между клиентом и серверов.

Зачастую целесообразно настроить преобразования для части выборки на клиентской стороне, тем самым минимизировав размер пересылаемых исходных данных, а потом произвести обработку всей выборки на серверной стороне. При этом осуществление большого числа запросов на сервер в процессе настройки преобразований даже для небольших размеров выборок так же является нецелесообразным, так как подобная настройка зачастую происходит интерактивно. Таким образом, даже минимальные задержки в процессе настройки, вызванные пересылкой данных с сервера на клиент, активацией запроса, созданием соединения, проверкой прав доступа, хранением сессии существенно снизят качество пользовательского взаимодействия с системой.

Основной сложностью, связанной с созданием библиотеки для цифровой обработки сигналов, способной работать как на клиентской, так и на серверной стороне, является гетерогенность двух указанных слоев: клиентская библиотека (браузерная) должна быть написана на Javascript, так как другие языки программирования практически не поддерживаются современными браузерами. При этом Javascript практически не использовался до недавнего времени в качестве языка описания серверной части веб-приложений, соответственно библиотека для цифровой обработки сигналов, написанная на Javascript, требует дополнительного функционала либо среды, выполняющей код данной библиотеки.

В настоящее время среди существующих вариантов решения указанной проблемы наиболее подходящими и распространенными являются библиотека Iron JS [4], способная исполнять Javascript-код в рамках процесса .NET приложений, а также серверный каркас Node.js [3], построенный на базе производительной виртуальной машины браузера Google Chrome.

Основным преимуществом Iron JS является полная интеграция в среду .NET, соответственно Javascript-код будет выполняться в контексте самого приложения, с теми же политиками безопасности; соответственно, время на пересылку данных для выполнения преобразования на серверной части будет минимизировано. Недостатком является ограниченная функциональность данного решения, так как Iron JS все же является портом Javascript на платформу .NET. Соответственно, не исключена вероятность несогласованной работы одного и того же Javascript кода на клиентской и серверной частях.

Основным преимуществом Node.js является полная согласованность результатов исполнения одного и того же кода как на серверной части, так и на клиентской, так как серверный код исполняется на одной из наиболее распространенных и документированных клиентских виртуальных машин для Javascript кода.

Оценка производительности браузерного и серверного кода для алгоритмов цифровой обработки сигналов

В связи с разделением программного приложения на клиентскую и серверную части с вынесением существенной части функционала приложения на клиентскую часть необходимо разделить необходимые для работы системы алгоритмы на две группы, преимущественно клиентские и преимущественно серверные. Такое разделение алгоритмов обосновано требованием минимизации временных задержек при преобразовании малых выборок сигналов и минимизацией количества пересылаемых исходных данных сигнала с клиента на сервер.

Для оценки времени выполнения преобразования в зависимости от алгоритма преобразования и размера обрабатываемой и результирующей выборки предлагается следующая формула:

$$t_{\text{кл}} = t_{\text{трансп}}(l_{\text{исх}}, V_{\text{трансп}}) + t_{\text{алг}}^{\text{кл}}(l_{\text{исх}}),$$

где $t_{\text{кл}}$ – время, требуемое для преобразования исходной выборки на стороне клиента; $t_{\text{алг}}^{\text{кл}}(l_{\text{исх}})$ – время выполнения преобразования на стороне клиента, зависящее от длины исходной выборки (вид зависимости определяется вычислительной сложностью алгоритма преобразования); $t_{\text{трансп}}(l_{\text{исх}}, V_{\text{трансп}})$ – время передачи данных с сервера на клиент, имеющее прямую зависимость от длины передаваемой выборки и обратную от скорости передачи данных (при этом, если данные уже были переданы и закэшированы, оно принимается равным 0).

Для оценки времени выполнения преобразования на стороне сервера предлагается следующая формула:

$$t_{\text{серв}} = t_{\text{акт}}^{\text{серв}} + t_{\text{алг}}^{\text{серв}}(l_{\text{исх}}) + t_{\text{сер}}^{\text{серв}}(l_{\text{рез}}) + t_{\text{трансп}}(l_{\text{рез}}, V_{\text{трансп}}) + t_{\text{десер}}^{\text{кл}}(l_{\text{рез}}),$$

где $t_{\text{серв}}$ – время, требуемое для преобразования исходной выборки на стороне сервера; $t_{\text{акт}}^{\text{серв}}$ – время активизации сервера при запросе; $t_{\text{алг}}^{\text{серв}}(l_{\text{исх}})$ – время выполнения преобразования на серверной стороне, зависящее от длины исходной выборки (вид зависимости определяется вычислительной сложностью алгоритма преобразования); $t_{\text{сер}}^{\text{серв}}(l_{\text{рез}})$ – время сериализации результирующей выборки на стороне сервера, имеющее прямую зависимость от длины результирующей выборки; $t_{\text{трансп}}(l_{\text{рез}}, V_{\text{трансп}})$ – время передачи результирующей выборки с сервера на клиент, имеющее прямую зависимость от длины результирующей выборки и обратную от скорости передачи данных; $t_{\text{десер}}^{\text{кл}}(l_{\text{рез}})$ – время десериализации результирующей выборки на стороне клиента, имеющее прямую зависимость от длины результирующей выборки.

Выбор места выполнения преобразований определяется тем, где (на клиентской, либо на серверной стороне) меньше время выполнения требуемых действий. При этом следует учесть, что существенной временной затратой является первичная загрузка данных с сервера на клиент, которая производится только единожды. Таким образом, при выполнении многократных преобразований выборок сравнительно небольшого размера предпочтительным является выполнение преобразований на стороне клиента, но при однократном просмотре сигнала, либо данных о сигнале, целесообразным является выполнение преобразования на стороне сервера. Основываясь на анализе зависимости времени выполнения преобразования как на клиентской, так и на серверной сторонах можно выделить два основных вида составляющих времени преобразования данных:

– время передачи данных с сервера на клиент, имеющее прямую зависимость от исходной выборки (для варианта с преобразованием данных на стороне клиента);

– время передачи данных с сервера на клиент, имеющее прямую зависимость от результирующей выборки (для варианта с преобразованием данных на стороне сервера);

– время выполнения алгоритма преобразования данных (определяется спецификой преобразования).

Кроме того, время выполнения как клиентского, так и серверного преобразований зависит от вычислительной мощности и загруженности соответственно клиентского и серверного вычислителей.

Файловое хранилище сигнальных выборок

В качестве постоянного файлового хранилища собранных сигнальных данных, результатов их преобразования, а так же другой бинарной информации предлагается использовать отдельный файловый сервер. Подобный подход позволяет настроить репликацию собираемых данных, минимизирует размер backup-версий баз данных, тем самым облегчая процедуры поддержки слоя данных в актуальном и работоспособном состояниях.

Что касается клиентского файлового хранилища, то с помощью входящей в HTML5 технологии File API стало возможным визуализировать данные, хранимые локально на клиентской машине. Таким образом, файл, содержащий вибродданные, может быть обработан и визуализирован без загрузки на сервер. Это позволит значительно минимизировать трафик между клиентской и серверной частью, а также избежать загрузки ненужных файлов (не содержащих информации, необходимой для анализа на сервере) за счет предварительного экспресс-анализа до начала процесса загрузки.

Кроме того, обработка локальных клиентских файлов позволит разделить функционал приложения, доступный для зарегистрированных или незарегистрированных пользователей (либо в зависимости от текущей роли пользователя). Таким образом, непривелигированные пользователи получат возможность просматривать вибродданные, хранимые в локальных файлах в виде ряда графиков (в зависимости от выбранного преобразования исходных данных), а

привелигированные пользователи смогут загружать файлы при необходимости на сервер. При этом механизм обработки и визуализации клиентских файлов может практически в неизменном виде применяться при визуализации файлов, хранимых на сервере. Для этого файлы, подлежащие визуализации, скачиваются в локальное файловое хранилище, а ссылка на скачанный локальный файл передается в виджет визуализации виброданных. Кроме того, подобный подход обеспечивает кэширование обрабатываемых файлов (так как при повторной работе с указанным файлом процедура скачивания не потребуется).

Кросс-платформенное мобильное приложение

Развитие мобильных технологий и платформ, расширение пропускной способности и удешевление стоимости передачи данных позволило создавать мобильные приложения на ряде платформ, способные собирать, обрабатывать и визуализировать сигнальные данные больших объемов. Подобные приложения могут быть использованы в рамках измерительно-вычислительных комплексов, решающих задачи вибродиагностики различных механизмов и агрегатов.

Большинство мобильных технологий способны предоставить достаточный программный интерфейс для разработки подобного приложения, включающий поддержку Wi-Fi, доступ к гироскопу и акселерометру мобильного устройства. Однако, как правило, из-за различия архитектуры данных платформ, портирование приложений с одной платформы на другую заключается в полном переписывании кода приложения, требуя значительных временных и финансовых затрат, а так же снижая надежность таких приложений, так как необходимо тестировать не одно, а несколько приложений. Дополнительным фактором является сложность добавления нового и изменения старого функционала, так как такие изменения необходимо вносить не в одно, а в несколько приложений.

Существует несколько каркасов, позволяющих разрабатывать единое приложение с использованием технологий PhoneGap [2], Titanium [5], но основным минусом подобных приложений является ограниченность доступных аппаратных функций, что очень существенно для приложений, осуществляющих сбор и обработку сигнальных данных, требующую тесной интеграции с аппаратными функциями мобильных устройств. В качестве решения указанной проблемы может быть предложена архитектура, в которой код приложения будет разделен на логику, представление и функционал, обращающийся к системным функциям, причем логика приложения не должна иметь зависимостей от той или иной платформы. В таком виде логика приложения может быть перенесена существующими средствами на другие языки, либо может быть реализована на C++, который доступен в большинстве мобильных платформ. При этом, представление может быть реализовано на основе браузерной компоненты на основе HTML5, доступной во всех мобильных платформах. С помощью технологии CSS одна и та же браузерная разметка может выглядеть по разному, в зависимости от выбранной мобильной платформы, в соответствии с требованиями к пользовательскому интерфейсу конкретной платформы.

Разработка веб-приложения, содержащего большую часть логики на стороне клиента в значительной мере повышает его производительность и снижает нагрузку на серверную часть, однако значительно снижает и его защищенность от копирования исходного кода (подобный подход предполагает, что практически вся логика и алгоритмы пересылаются на сторону клиента). При этом при правильной, компонентной разработке функционала предполагается, что часть исходного кода программы может повторно использоваться злоумышленниками практически без его изменения и специальной адаптации. Кроме того, подобная архитектура подвержена атакам, заключающимся во вживлении дополнительного Javascript кода, подключающего функционал привелигированных пользователей (для случая, когда непривелигированным пользователям доступна лишь часть поддерживаемых алгоритмов преобразований и визуализации виброданных).

Для избежания возможности подобных атак, но при этом с сохранением преимуществ данного приложения (производительности, минимизации нагрузок на сервер, доступности базовой функциональности приложения незарегистрированным пользователям) предлагается использование механизма плагинов, поддерживаемого PhoneGAP. Использование подобного механизма позволит вынести наиболее ценные алгоритмы на сторону нативного приложения, таким образом защитив их от копирования за счет средств мобильной платформы либо средств операционной

системы для настольных приложений. При этом на стороне Javascript-кода останется только обертка над функционалом (что практически не изменяет архитектуру самого веб-приложения). Помимо защиты кода от копирования, подобный подход решает проблему монетизации мобильных приложений, так как для работы с расширенным набором алгоритмов преобразований вибродонных потребуется установка приложения на устройство пользователя (соответственно данная процедура может быть платной).

Заключение

Наиболее эффективным способом организации системы, предназначенной для обработки вибрационных данных, получаемых от различных технических объектов, и поддержки принятия решений по оценке состояния контролируемых объектов на основе их вибрационных характеристик, является реализация веб-приложения с поддержкой современных мобильных браузеров, способного корректно работать как на настольных компьютерах, так и на мобильных или планшетных устройствах. При наличии подобного веб-приложения можно создать кросс-платформенный мобильный клиент на базе библиотек и Javascript-каркасов, таких как Sencha Touch, с последующим его использованием в кросс-платформенных приложениях, построенных на основе технологии PhoneGap. При этом наличие веб-сайта, как основы приложения, позволяет использовать его для работы с обычных персональных компьютеров.

Хранилище исследуемых сигнальных выборок предлагается локализовать от базы данных на отдельном файловом сервере, что позволит распределить нагрузку между серверами системы, при необходимости расширить решение за счет переноса файлов в распределенные файловые системы, такие как Apache Hadoop, поддерживающие распараллеливание задач анализа и преобразования хранимых выборок. Также для организации кэша отображаемых сигналов на стороне клиента наиболее эффективным подходом является использование локального перманентного файлового хранилища, поддерживаемого технологией File API, входящей в HTML5. Такой подход обеспечит возможность без существенного изменения кода добавить функционал просмотра локальных сигнальных файлов пользователя без процедуры их загрузки на сервер.

MOBILE WEB-APPLICATION ARCHITECTURE CONCEPT FOR VIBRATIONAL SIGNALS PROCESSING

V.E. BAZAREVSKY

Abstract

A decision of support system architecture for handling the vibration data from various household objects, which consists of mobile, desktop and web parts is studied. Problems of optimal algorithms distribution between client and server sides of application is researched. The way to monethize mobile web applications on the basis of PhoneGAP plug-in feature are proposed.

Список литературы

1. *Zachary Kessin*. Programming HTML5 Applications, Building Powerful Cross-Platform Environments in JavaScript. Sebastopol CA, 2011.
2. *Andrew Lunny*. PhoneGap Beginner's Guide. Birmingham, 2011.
3. The Node Ahead: JavaScript leaps from browser into future. The Register. [Электронный ресурс]. – Режим доступа: http://www.theregister.co.uk/2011/03/01/the_rise_and_rise_of_node_dot_js/. – Дата доступа: 05.09.2012.
4. *John Paul Mueller*. Professional IronPython, Wrox Professional Guides. Chichester, 2010
5. *Boydlee Pollentine*. Appcelerator Titanium Smartphone App Development Cookbook. Birmingham, 2011.