УДК 004.7

В. О. Демиш, **Б. Н. Пищик**

Новосибирский государственный университет
ул. Пирогова 2, Новосибирск, 630090, Россия

E-mail: v.demish@gmail.com; Boris.Pishchik@gmail.com

СИНХРОНИЗАЦИЯ ДАННЫХ НА МОБИЛЬНЫХ ПЛАТФОРМАХ *

Анализируются задачи синхронизации данных применительно к мобильным платформам. Рассмотрены существующие методы решения, принятые стандарты и инструменты синхронизации. Предложена формальная модель инструмента синхронизации данных на мобильных платформах.

Ключевые слова: синхронизация данных, мобильные платформы, базы данных, интеграция данных, мобильные устройства.

О синхронизации данных

В свете непрерывно растущего количества используемых мобильных платформ в настоящее время остро стоит задача синхронизации данных на мобильных устройствах и вычислительных серверах. Эта задача возникает достаточно часто, вот список наиболее распространенных примеров:

- синхронизация контактов, календаря на смартфоне, планшетном компьютере;
- синхронизация данных в CRM (Customer Relationship Management) системах и на мобильных устройствах;
- редактирование документов на различных устройствах, поддержка актуальности данных на всех устройствах;
- общее хранилище файлов, медиаконтента (фотографии, видеоролики);
- синхронизация данных приложений в режимах offline / online.

Здесь приведен далеко не весь перечень примеров задачи синхронизации данных в мобильных платформах, однако этого достаточно, чтобы оценить актуальность данной задачи. Настоящая статья представляет собой результат исследования существующих подходов к синхронизации данных, применимых на мобильных платформах.

Постановка задачи синхронизации данных

Определим ряд понятий, касающихся мобильных устройств.

Мобильное устройство – вычислительное устройство небольших габаритов (легко переносимое), снабженное дисплеем для отображения информации, а также мини-клавиатурой либо поддержкой технологии *touch-screen* для ввода информации. Современные мобильные устройства имеют доступ к сети Интернет благодаря различным технологиям (Wi-Fi, GPRS,

* Работа выполнена при поддержке РФФИ (проект № 11-07-00561-а).

Демиш В. О., Пищик Б. Н. Синхронизация данных на мобильных платформах // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2013. Т. 11, вып. 4. С. 46–58.

EDGE, 3G и т. д.). Мобильные устройства широко представлены смартфонами, коммуникаторами, а также планшетными компьютерами.

Мобильная операционная система (ОС) – операционная система, используемая на мобильных устройствах. Наиболее популярны в настоящее время: iOS, Android, Symbian, Windows Phone.

Мобильная платформа – синоним мобильной операционной системы устройства. Термин широко используется в случаях, когда речь идет не о самой ОС, а о разработке приложений на ее базе. К примеру: разработка приложений на платформе Android.

В дальнейшем изложении указанные термины используются в соответствии с приведенными выше определениями.

Будем считать, что мы рассматриваем совместную работу в сети различных устройств, включающих различные мобильные устройства (на разных платформах), настольные персональные компьютеры, ноутбуки, сервера.

Важным свойством данной сети является наличие *различных* мобильных платформ. В рамках одной мобильной платформы многие задачи синхронизации данных решены разработчиками этих платформ: устройствам на Android доступны сервисы Google, для владельцев устройств на iOS есть iCloud и т. д. В случае же одновременного функционирования различных мобильных платформ синхронизация данных между ними штатными средствами не представляется возможной.

Подчеркнем также наличие потоков информации в обоих направлениях: загрузка данных на мобильные устройства и выгрузка данных с мобильных устройств в облачную среду. Однонаправленный поток является частным случаем данной задачи и имеет второстепенное значение в контексте настоящего исследования.

Стандарты синхронизации данных

Разработкой открытых стандартов для мобильных платформ занимается *Open Mobile Alliance* (OMA, http://www.openmobilealliance.org, *Открытый мобильный альянс*) – международная организация по стандартизации. Основана в 2002 г., в настоящее время ее поддерживают такие гиганты IT и мобильной индустрии, как Microsoft, Intel, Oracle, Samsung, LG, Nokia, HTC и др. Под эгидой этой организации развивается множество действующих стандартов, в том числе и наиболее интересный в рамках настоящей статьи OMA: Data synchronization (OMA DS). Остановимся на нем подробнее.

OMA Data Synchronization (OMA DS) – это спецификация инструментария синхронизации данных и основанного на XML формата обмена данными между устройствами, объединенными в сеть. OMA DS разработан для использования на мобильных платформах, для которых подключение к Интернету периодически прерывается. Также OMA DS применим для организации *peer-to-peer* [1] синхронизации.

Предпосылки к созданию OMA возникли в 2000 г. с выходом в свет языка Synchronization Markup Language (SyncML) [1], удостоившегося внимания наиболее весомых производителей мобильных устройств и мобильного программного обеспечения, таких как Nokia, Ericsson, IBM,Lotus, Motorola, Palm, Psion, Starfish. Как и многие другие языки разметки, SyncML основан на XML. Его задача – синхронизация данных на устройствах сети. Он спроектирован для синхронизации данных между мобильными устройствами, подключенными к сети, и сервисами, которые доступны в этой сети.

Всего за время существования OMA принято четыре версии стандарта OMA: Data synchronization 1.1.2 (21.07.2004), 1.2.1 (10.08.2007), 1.2.2 (19.03.2009) и существенно переработанный 2.0 (19.07.2011).

Проведем краткий анализ актуальных возможностей стандарта, т. е. не отмененных последними версиями.

[1] Sync ML Iniative, Building an Industry – Wide Mobile Data Synchronization Protocol (Sync ML White Paper), 2001. P. 1–14.

Классификация видов синхронизации данных

OMA DS предлагает [2] оперировать семью типами синхронизации данных, подразделяемых на двусторонние и односторонние:

Двусторонние типы синхронизации:

- двусторонняя синхронизация (Two-way sync fast) – обычный тип синхронизации, в котором клиент и сервер обмениваются информацией об изменениях в данных, клиент первым посылает информацию об изменениях;
- медленная синхронизация (Slow sync) – вид двусторонней синхронизации, в котором все элементы данных на устройствах сравниваются друг с другом поатрибутно. Практически это означает, что клиент посылает все свои данные (а не только изменения) на сервер, после чего последний анализирует (поатрибутно) их, сравнивая с собственной копией данных. После выполнения анализа сервер посылает обратно изменения, которые должны быть применены к данным на клиенте.

Односторонние типы синхронизации:

- односторонняя синхронизация только от клиента (One-way sync from client only) – вид синхронизации данных, в котором клиент посылает изменения данных на сервер, но сервер не отправляет изменения обратно клиенту;
- обновление только от клиента (Refresh sync from client only) – клиент отправляет все свои данные на сервер, который целиком заменяет ими собственную информацию (например, экспорт данных);
- односторонняя синхронизация только от сервера (One-way sync from server only) – сервер посылает изменения в данных на клиент, но клиент не отправляет изменения своих данных на сервер;
- обновление только от сервера (Refresh sync from server only) – сервер отправляет на клиент все данные целиком, клиент замещает ими собственные данные;
- синхронизация при изменении серверных данных (Server-alerted sync) – вид синхронизации, в котором сервер предупреждает клиента о необходимости проведения сеанса синхронизации специфического типа – какого именно, зависит от конкретной реализации в приложении (т. е. данный вид ориентирован на разработчиков, которым недостаточно определенных ранее видов синхронизации).

Базовая функциональность для синхронизации данных

Остановимся на базовых функциях, доступных для всех типов синхронизации данных, обозначенных OMA DS.

Используемые протоколы передачи данных. SyncML работает с протоколами HTTP (HyperText Transfer Protocol, OBEX (OBject EXchange, используется в Bluetooth), WSP (Wireless Session Protocol). Обмен данными между участниками синхронизации данных осуществляется посредством обмена сообщениями специального формата.

Журналирование изменений (*Change Log Information*). Все устройства, участвующие в синхронизации (клиент и сервер), должны иметь возможность вести учет изменений, происходящих от синхронизации к синхронизации. Иначе говоря, устройства несут ответственность за управление журналом всех операций, связанных с конкретными данными в базе данных. Под операциями в этом случае подразумеваются такие, как обновление, добавление и удаление. Стандартом не предусмотрен формат журнала, важно, чтобы перед началом синхронизации данных каждое устройство могло сформировать набор измененных данных.

Использование идентификаторов сеансов синхронизации (*Sync Anchors*). Для идентификации различных сеансов синхронизации применяется строковое значение. Как правило, это либо порядковое значение, либо некоторое представление даты и времени.

Соответствия идентификаторов (*Mapping of Indentifiers*). Исходя из того что идентификаторы данных в таблицах на различных мобильных устройствах и сервере могут различаться, необходимо построение соответствия идентификаторов. Таким образом, на сервере ис-

[2] Open Mobile Alliance Ltd., SyncML Data Sync Protocol, version 1.1.2, 2003.

пользуется GUID (глобальный идентификатор), а на мобильном устройстве – LUID (уникальный в пределах мобильного устройства). На сервере ведется соответствие GUID'ов с LUID'ами, используемыми на мобильных устройствах.

Разрешение конфликтов (*Conflict Resolution*). Конфликты при синхронизации возможны в случае изменения одних и тех же данных на сервере и клиенте. Для разрешения конфликтов предложено несколько путей: разрешение конфликта на сервере (автоматически) либо предупреждение клиента о конфликте и обеспечение его возможностями по ликвидации конфликта.

Безопасность (*Security*). Требуется проведение аутентификации, а также MD5 на уровне сервера. Оба (клиент и сервер) участника синхронизации данных должны быть в состоянии провести аутентификацию и отправить результаты проведения аутентификации обратно.

Адресация для устройств (*Device Addressing*). Сервера и устройства, участвующие в синхронизации данных, должны быть доступны по какому-либо адресу. К примеру, подключенные к сети сервера могут использовать в качества адреса URI (Uniform Resource Identifier), например: http://my-server-address.ru/. Участники синхронизации вправе использовать иную адресацию, к примеру на базе IMEI (International Mobile Equipment Identity – международный идентификатор мобильного оборудования).

Адресация для баз данных (*Database addressing*). Адресация баз данных в SyncML происходит с использованием специализированных URI. Пример такого адреса: http://my-server-address.ru/calendar/james_bond/.

Использование пакетов сообщений, управление большими объектами (*Multiple messages in packages, Large object handling*). Специальные сообщения на SyncML группирутся в пакеты для обмена информацией между участниками синхронизации данных. В случае если в обмен включен объект, не вмещаемый в одно сообщение, тогда он разбивается на несколько сообщений.

Сигналы занятости (*Busy Signalling*). Если сервер получил данные с клиента, но не способен в требуемое время (это время зависит, например, от используемого транспортного протокола), он должен сообщить об этом клиенту. В этом случае он отправляет клиенту пакет «Busy Status». После того как клиент получил сигнал о занятости сервера, он может запросить результаты синхронизации позже либо начать синхронизацию данных сначала.

Структура сообщений SyncML. Каждое сообщение содержит заголовок (Header) и тело (Body). Заголовок содержит метаинформацию, такую как база-источник <Source>, база-приемник <Target>, информация аутентификации <Cred>, идентификатор сессии <SessionID>, идентификатор сообщения <MsgID>, версия используемого стандарта SyncML. Тело содержит в себе запросы (requests), сигналы (alerts) и данные (data).

Существующие подходы к решению задачи

Синхронизация персональных данных. Если человек пользуется смартфоном и планшетным ПК, вполне разумно синхронизовать между мобильными устройствами такие данные, как контакты (адресные книги), календари, задачи, заметки и др. Работа с информацией такого рода – это *Personal information management* (PIM), иначе говоря*, управление персональной информацией.*

Все существующие решения (включая решения от разработчиков мобильных платформ, таких как Google или Apple) имеют облачный принцип работы. Это означает, что поставщик сервиса предоставляет доступ владельцу мобильных устройств к вычислительному облаку, в котором и хранятся в конечном счете персональные данные пользователей. В дальнейшем, к примеру, при покупке нового устройства можно подключить его к этому облаку и осуществить синхронизацию данных.

Рассмотрим существующие сервисы, решающие задачу синхронизации персональных данных на разных мобильных платформах. Как уже сказано, синхронизация персональных данных между мобильными устройствами на одной платформе хорошо решается производителями этой платформы. Обращение же к услугам сторонних производителей, скорее всего, повлечет необходимость установки дополнительного программного обеспечения (программы-клиенты) на мобильные устройства, чего чаще всего не требуется при использовании

«родного» для устройства алгоритма. Однако, не считая уже обозначенной проблемы использования устройств на различных мобильных платформах, существует еще один открытый вопрос: кто же владеет данными пользователя в конечном счете? И, к примеру, для пользователей Android Dave Rosenberg еще в 2009 г. [3] сформулировал этот вопрос: если Google уже владеет вашим email'ом, календарем, данными поисковых запросов, действительно ли вы считаете, что отдать ваши телефонные контакты Google'у – настолько хорошая идея?

В 2010 г. в электронном журнале PCWorld.in появилась статья с обзором подобных сервисов [4]. Автор обзора привел такие сервисы: Mobical.net, Funambol, Rseven.com, Memotoo, Usynworld. К настоящему времени из перечисленных успешно функционирует Funambol и Memotoo, который в свою очередь предлагает осуществить синхронизацию данных также через Funambol. Помимо Funambol отметим Google Sync от Google, включающий поддержку не только устройств на Android, но и на iOS, Windows Mobile и др.

Кроме того, функционируют такие сервисы, как mobiledit.com, Everdroid.com, http://www.synthesis.ch/ и пара отечественных разработок – http://www.memiana.com, http://www.synchronet.ru/.

Это далеко не весь список предлагаемых на сегодня сервисов, однако его достаточно, чтобы положительно ответить на вопрос о том, успешно ли решена задача синхронизации персональных данных на мобильных устройствах.

Существующие решения для синхронизации персональных данных используют язык SyncML.

Компания Funambol распространяет [2] бесплатную open-source-версию SyncML сервера (Funambol Community Edition), позволяющую самостоятельно организовать собственный сервис синхронизации персональных данных на мобильных устройствах.

Организация общего хранилища файлов, медиаконтента. Организация общего хранилища файлов включает:

- загрузку файлов в хранилище, сохранение файлов из хранилища;
- работу с хранилищем как с обычной локальной директорией;
- настройку прав доступа к различным частям хранилища;
- дополнительные возможности работы с медиаконтентом – организацию альбомов фотографий, галерей видеороликов и т. д.;
- синхронизацию содержимого хранилища со всеми устройствами, подключенными к хранилищу.

В настоящее время существует множество сервисов, успешно решающих перечисленные задачи, наиболее известны Google Drive, DropBox, SugarSync. Указанные решения поддерживают все современные мобильные платформы и настольные операционные системы.

Синхронизация данных приложений – наиболее общая задача из рассматриваемых задач в настоящей статье. Она встает перед разработчиками самых различных приложений:

- приложений, загружающих постоянно пополняющийся контент с серверов в сети (например, приложения-энциклопедии, словари и т. д.);
- мобильных приложений для интернет-магазинов, отображающих состояние заказов пользователя;
- интернет-каталогов, доступных на мобильных платформах с возможностью формирования заказов;
- социальных приложений (клиентов социальных сетей, мобильных версий корпоративных порталов);
- приложений с возможностью просматривать отзывы о чем-либо и оставлять собственные отзывы (то же самое с возможностью комментирования).

Перечисленные категории включают большое количество конкретных приложений, так что задача крайне актуальна. Проведем анализ существующих решений данной задачи.

---

[3] http://news.cnet.com/8301-13846_3-10398137-62.html, 2009

[4] *Pachuau L.* Top third-party Mobile Cloud Sync Services. 2010. URL: http://www.pcworld.in/features/top-third-party-mobile-cloud-sync-services

Решение задачи внутри приложения. Любая задача синхронизации может быть решена непосредственно разработчиками конкретного приложения внутри него. В данном случае в приложение включаются необходимые модули синхронизации с сервером (может быть серверами), выполняющие синхронизацию данных требуемого типа. Схематично работа приложения в этом случае выглядит так, как представлено на рис. 1.

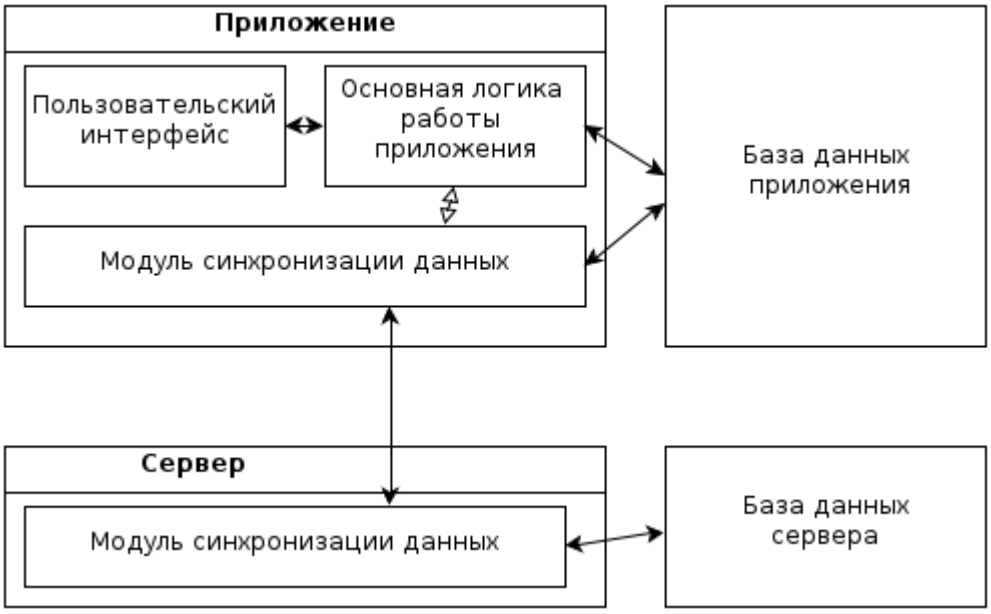

Рис. 1. Реализация модуля синхронизации данных внутри приложения

Стрелка между блоком основной логики работы приложения и модулем синхронизации данных выделена, так как данная связь не всегда имеет место. Например, синхронизация данных может быть инициирована в одном из модулей приложения при возникновении какого-либо события (связь есть) либо же вызываться независимо ни от чего через определенные промежутки времени (связи нет).

Такое решение выглядит достаточно лаконичным, однако, очевидно, обладает существенным недостатком – модуль синхронизации в данном случае слишком тесно интегрирован с конкретным приложением и не может существовать отдельно от него, т. е. если разработчик захочет создать новое приложение, в котором снова потребуется синхронизовать данные, – придется разрабатывать другой модуль синхронизации данных.

Подавляющее большинство существующих приложений используют такую схему работы. Чаще всего, правда, требуется довольно простой вид синхронизации связи: обновление только от сервера (Refresh sync from server only), т. е. периодическое полное обновление информации на клиентских устройствах информацией с сервера, так что реализация программных модулей обмена информацией в данном случае не является особенно сложной. Ярким примером таких приложений являются различные информационные приложения: ДубльГис, информаторы о различных событиях вроде Олимпийских игр, энциклопедии, туристические справочники и т. д.

Обновление только от сервера довольно быстро перерастает в существенно более сложный вид синхронизации данных: либо в медленную двустороннюю, либо в двустороннюю. Для этого достаточно добавить в приложение простейшую форму обратной связи, например, возможность оставить отзыв о чем-либо в режиме оффлайн (т. е. без доступа к Сети). Необходимость отражения этого отзыва на сервере потребует выполнения синхронизации данных, как только появится доступ к серверу. Кроме того, двусторонняя синхронизация требуется

во многих других приложениях: в клиентах социальных сетей, в игровых приложениях, в клиентах корпоративных порталов и др., т. е. доля приложений, в которых реализация модуля синхронизации данных представляет собой непростую задачу, велика.

Использование механизма веб-сервисов, интеграционных платформ. Отметим, что многие технологии интеграции данных применимы для синхронизации данных. Принципиальным условием задачи интеграции данных является работа с источниками данных различной структуры. В случае же с синхронизацией мы имеем дело с несколькими копиями наборов данных одинаковой структуры.

Таким образом, можно утверждать, что синхронизация данных является частным случаем интеграции данных, и инструменты для интеграции могут быть успешно применены и для синхронизации.

Рассмотрим подход Integration as a Service (IaaS) [3], успешно применяемый для интеграции облачных сервисов как друг с другом, так и с необлачными информационными системами. Такой подход практически не применим к мобильным платформам. Аргументируем данное утверждение.

Прежде всего, напомним схему работы интеграционных платформ в среде облачных вычислений на примере использования локальной информационной системы и пары облачных сервисов (корпоративной почты и CRM-системы) (рис. 2).

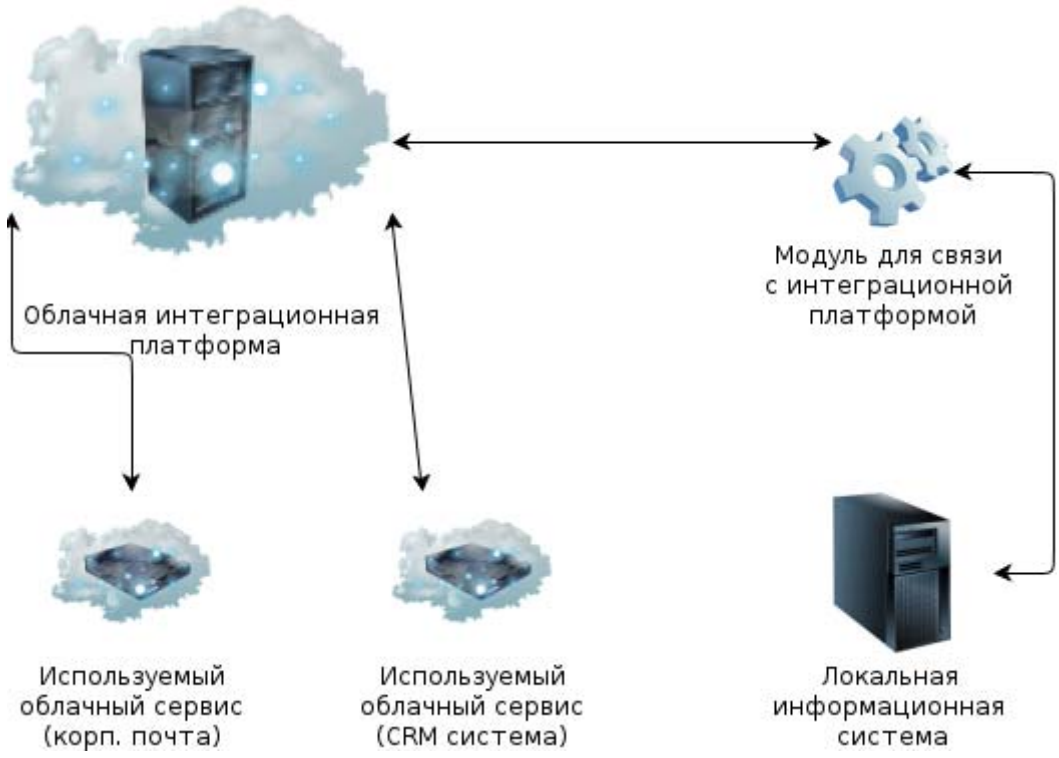

Рис. 2. Пример использования облачной интеграционной платформы

Каждый субъект интеграции данных (речь об информационных системах вне облака и облачных сервисах) должен быть доступен для интеграционной платформы. Если облачные сервисы обладают этой доступностью «по определению», то для доступа к локальным информационным системам необходимо использовать дополнительный модуль (поставляемый интеграционной платформой). Непосредственно интеграция данных осуществляется посредством настроенных правил обмена в интеграционной платформе.

Проблема использования такого подхода для мобильных платформ в том, что практически невозможно предоставить облачной интеграционной платформе доступ к мобильному устройству. Этому препятствуют:

- нахождение устройства вне доступа к сети Интернет (offline-режим);
- чрезмерная нагрузка на мобильное устройство в случае использования аналога дополнительного модуля для связи с интеграционной платформой.

Основное удобство IaaS для немобильных платформ в виде инициации всех процессов обмена со стороны интеграционной платформы уже не приносит здесь никаких преимуществ, а скорее наоборот, наделяет модель обмена данными излишней неповоротливостью.

Использование сервисов синхронизации данных. Потенциальная трудоемкость реализации модуля синхронизации внутри приложения, а также практическая невозможность использования подхода Integration as a Service привели к появлению специального сервиса, нацеленного непосредственно на работу с мобильными устройствами.

Сервис Mobeelizer.com предоставляет наиболее удобный на сегодняшний день метод синхронизации информации. Предлагаемая схема работы представлена на рис. 3.

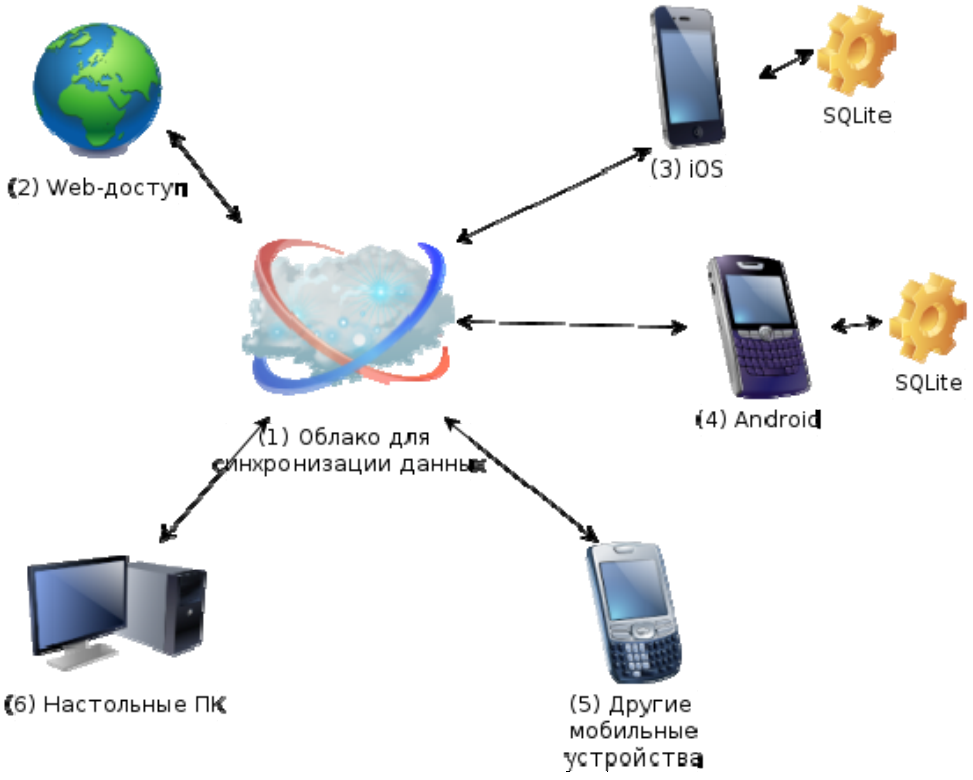

Рис. 3. Принцип работы Mobeelizer.com

Распишем по шагам принцип работы сервиса.

1. После регистрации на сайте разработчику предоставляется личное облако для синхронизации данных.

2. Доступ к облаку возможен непосредственно через браузер. В специальном разделе сайта можно определить структуры данных, которые подлежат синхронизации.

3. Для iOS скачиваются специальные модули доступа к облаку. Для работы с облаком требуется подключить эти модули, а также автоматически сгенерированный код для работы с моделями данных, определенными на шаге (2). После подключения этих модулей можно работать с базой SQLite, которая содержит определенные на шаге (2) структуры данных.

4. Для Android все так же, разница только в подключаемых библиотеках (не для Objective C, а для Java).

5. Другие мобильные устройства могут работать с облаком через HTTP API [5]. Кроме того, существуют библиотеки для работы в приложениях, разрабатываемых на Titanium (http://www.appcelerator.com/) и работающих на Windows Phone 7.

6. Настольные персональные компьютеры могут также работать через HTTP API.

Работа с облаком при этом крайне проста, приведем пример из официальной документации (на Java для платформы Android):

```
MobeelizerOperationError error = Mobeelizer.login("user", "password");
MobeelizerDatabase database = Mobeelizer.getDatabase();

Blog blog = new Blog();
blog.setTitle("First entry");
blog.setContent("I'm proud to announce that my first application is working properly.");
MobeelizerErrors errors = database.save(blog);
```

В этом примере происходит подключение к облаку, после чего создается экземпляр сущности Blog. Далее для нового экземпляра задаются значения полей Title и Content (эти поля должны быть определены разработчиком на шаге 2 из приведенной выше последовательности работы с облаком). Затем новый экземпляр сохраняется в базе данных мобильного устройства.

Чтобы осуществить синхронизацию данных с облаком, достаточно вызвать лишь один метод sync():

```
MobeelizerOperationError error = Mobeelizer.sync();
```

Никаких других действий выполнять не требуется – все сделает библиотека от Mobeelizer. Несколько слов о том, как происходит определение моделей данных. Структура данных представляет собой реляционную модель данных с поддержкой таких типов, как булево, дата, строка, число, файл и ссылка (аналог внешнего ключа). После определения модели данных происходит автогенерация кода, который необходимо подключить к приложению. Этот код позволяет работать с базой SQLite, в контексте определенной схемы базы данных, т. е. самостоятельно им даже нет необходимости осуществлять подключение к базе данных – достаточно просто пользоваться сгенерированными в Mobeelizer библиотеками, которые заодно предоставляют разработчику возможности удобной реализации ORM (Object Relational Mapping) [4].

Работа с локальной базой SQLite возможна и в режиме offline. В этом случае после выхода в режим online приложение должно вызвать метод синхронизации данных. Кроме того, сервис предлагает функционал по разрешению конфликтов, возникающих в процессе синхронизации данных (эта тема опускается в настоящей статье).

Из недостатков можно отметить только непрозрачность синхронизации (нет возможности настроить требуемый вид синхронизации, основываясь на знании функций приложения), а также необходимость обновления подключенных к приложению библиотек в случае изменения структуры используемых данных. Также заметим, что разработчики сервиса создавали его, не ориентируясь на OMA DS, и не использовали язык SyncML.

Такой метод работы в мобильных приложениях существенно отличается от других, рассмотренных в данной статье, в лучшую сторону в плане удобства пользования и является очень удобным инструментом для разработчиков мобильных приложений.

Использование синхронизационной платформы. Для разработчиков на платформе .NET существует библиотека Microsoft Sync Framework. Начиная с версии 4.0, она поддерживает работу с различными мобильными платформами (ранее была поддержка исключительно

[5] NORD Software Consulting, Classification of HTTP-based APIs, http://nordsc.com/ext/classification_of_http_based_apis.html, 2010

.NET). Платформа не соотносится со стандартами SyncML, поскольку имеет более широкую область применения, с уклоном на сервера баз данных от Microsoft (SQL Server).

Неудобство применения платформы на мобильных платформах заключается в необходимости работы на мобильном устройстве среды выполнения Microsoft Sync Framework Runtime, а не только хранения там данных и метаданных.

Платформа поддерживает четыре способа разрешения конфликтов.

1. Приоритет клиента – в данном случае первостепенны изменения на клиенте.
2. Приоритет сервера – первостепенны изменения на сервере.
3. Объединение – частный случай разрешения конфликтов, допустимый для численных типов данных, суть которого – в сложении данных клиента и сервера.
4. Отложить решение – в данном случае платформа не производит действий по разрешению конфликта и оставляет эту задачу для приложения.

Конфликтом в этом случае называется изменение двумя участниками синхронизации данных одного и того же элемента данных.

Формальная модель для синхронизации данных в приложениях

Исходя из проведенного анализа, можно сделать вывод о том, что синхронизация персональных данных и файлов на мобильных устройствах в настоящее время успешно решается многими компаниями. А задача синхронизации данных в приложениях находится совсем на другой стадии и до сих пор у разработчиков приложений не так много инструментов для решения этой задачи.

Обобщим результаты исследования задачи синхронизации в построении формальной модели платформы синхронизации данных. Такая модель действительно необходима и не является лишь объектом академического интереса, поскольку:

- позволит формально ставить задачи синхронизации;
- предоставит нам критерии сравнения различных решений задачи синхронизации данных;
- станет отправной точкой в создании инструментария для решения задач интеграции данных в мобильных приложениях;
- программная реализация модели позволит оценить рациональность того или иного подхода к синхронизации данных в каждом конкретном случае.

Отметим, что такая задача хорошо соотносится со стандартами OMA DS. Если OMA DS описывает протокол обмена сообщениями между участниками обмена данными, то предлагаемая модель описывает платформу для синхронизации данных, в рамках которых определяются все настройки осуществляемых сеансов обмена данными.

При этом можно, конечно, совсем не обращаться к SyncML, а использовать собственную структуру пакетов обмена информации между клиентом и сервером. К слову, так устроена работа Mobeelizer. Исходя из направленности SyncML, он будет наиболее оптимальным вариантом для персональных данных (PIM), а для другого рода информации оптимальность формата данных будет зависеть уже от деталей конкретной задачи.

Построим формализацию платформы синхронизации данных, основываясь на реляционной модели данных (исследование нереляционных моделей – тема отдельной статьи). Отметим, что все современные мобильные платформы поддерживают работу с легковесной СУБД SQLite (http://sqlite.org), которая может быть использована для практической реализации предлагаемой модели. Построенная модель в дальнейшем может быть обобщена для синхронизации файлов, а не только информации в базах данных.

Определим объекты платформы синхронизации (в скобках указано сокращение). Прежде всего, для описания базы данных будем использовать традиционную для реляционных баз данных терминологию отношений, атрибутов, схем отношений. Так, за S_1, S_2, …, S_n обозначим схемы всех имеющихся в базе данных отношений. Для каждого отношения определен соответствующий набор атрибутов. Множество всех S_i обозначим за S.

Среди участников обмена выделим сервер – Server, а также клиентов – Client. Клиенты и сервер являются участниками синхронизации данных, для каждого участника определен

свой экземпляр базы данных. База данных на сервере обязательно содержит все отношения из *S*, а база данных клиента, напротив, может содержать лишь некоторое подмножество *S*.

На каждом клиенте, для каждого отношения должны быть определены правила синхронизации данных. Каждое правило представляет собой упорядоченную тройку <*SyncType*, *Filter*, *ConflictResolution*>, содержащую информацию о деталях синхронизации:

- *SyncType* – один из семи рассмотренных ранее типов синхронизации;
- *Filter* – булевозначное условие на кортежи отношения; синхронизации подлежат только кортежи, для которых данное условие выполнено;
- *ConflictResolution* – метод разрешения конфликтов; один из четырех, как и в Microsoft Sync Framework.

В случае если какое-то отношение из *S* не используется в базе данных на устройстве, такая тройка не определяется для данного отношения. Все используемые правила объединим в множество Rules, в котором каждое отдельное правило выглядит так: <Client, S, <SyncType, Filter, ConflictResolution>>, где Client – устройство, участвующее в синхронизации данных, а *S* – схема отношения. Таким образом, построенные конструкции однозначно определяют конфигурацию сети, в которой осуществляется синхронизация данных. Что касается практической реализации, представление определенных множеств, к примеру в виде XML-файла, будет представлять собой настройку синхронизации данных между различными устройствами.

Для демонстрации описанной терминологии приведем простой пример с тремя отношениями:

1) Departments (department_id, department_name);
2) Employees (employee_id, first_name, last_name, birthday, department_id);
3) Nomenclature (nomenclature_id, nomenclature_name).

Отношения служат для работы с подразделениями (Departments) их работниками (Employees) и ведением некого перечня номенклатуры (Nomenclature). Первые два отношения связаны, а номенклатура представляет собой перечень наименований.

Предположим, в нашей сети работают следующие устройства: сервер, компьютер кладовщика, мобильное устройство исполнительного директора и руководителя некоторого отдела (скажем, с department_id = 1). Выделим для них используемые схемы данных (предположим, что работа с сервером осуществляется только с других перечисленных устройств, напрямую на сервере никто не работает):

1) сервер – центральный сервер с установленной на нем СУБД, доступны все отношения;
2) компьютер кладовщика (Storage) – устройство у работника склада, ведущего перечень номенклатуры, доступно только отношение Nomenclature;
3) мобильное устройство исполнительного директора (Director), контролирующего работу всех отделов и их работников, доступны отношения Departments, Employees;
4) мобильное устройство руководителя некоторого отдела (Department_Director), скажем, с department_id = 1, доступны отношения Departments, Employees, но не кортежи, у которых department_id <> 1.

Центральное место в примере занимает сервер – на нем установлена СУБД, с которой должны осуществлять сеансы синхронизации данных другие устройства. Для этого определим на каждом из них правила синхронизации данных (Rules). Для удобства считаем, что отношения пронумерованы в том порядке, в котором они приведены выше. Тогда множество Rules будет состоять из следующих элементов:

- <Storage, Nomenclature, <One-way sync from client only, True, Client priority>>;
- <Director, Departments, <Two-way sync fast, True, Client priority>>;
- <Director, Employees, <Two-way sync fast, True, Client priority>>;
- <Department_Director, Departments, <Two-way sync fast, department_id = 1, Server priority>>;
- <Department_Director, Employees, <Two-way sync fast, department_id = 1, Server priority>>.

Client priority и *Server priority* в данном контексте обозначают соответственно приоритеты клиента и сервера при разрешении конфликтов. Значения различны для исполнительного ди-

ректора и директора конкретного отдела, чтобы подчеркнуть приоритетность изменений, внесенных исполнительным директором (схему см. на рис. 4).

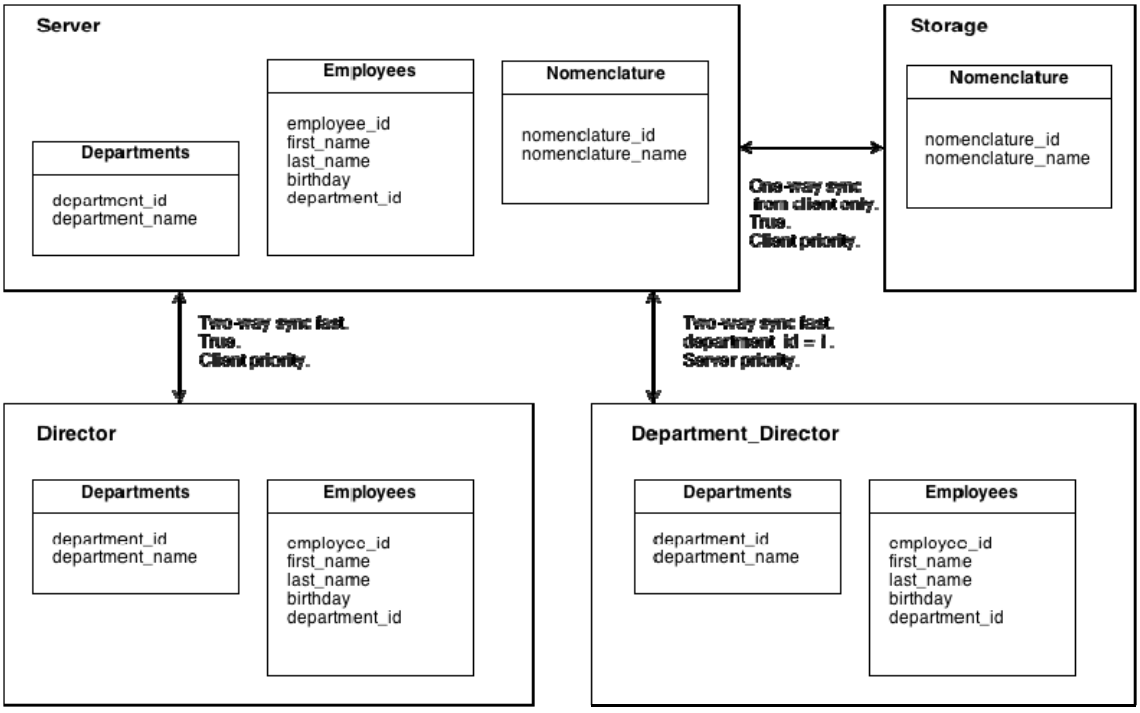

Рис. 4. Пример конфигурации сети и синхронизации данных

Добавим в формальную модель детали ее практической реализации. Поскольку первоочередная задача предлагаемой платформы синхронизации данных – работа на мобильных устройствах, то клиентская часть должна быть по возможности как можно менее требовательна к ресурсам. Достижение этой цели предполагается осуществить путем поддержки множества различных клиентских библиотек, в соответствии с разнообразием возможных конфигураций синхронизации данных. Например, если на устройстве предполагается использование только одного типа синхронизации Refresh sync from client only, то нет никакой необходимости нагружать его реализацией других типов синхронизации. То же самое касается и различных методов разрешения конфликтов.

Заключение

Проведенный анализ задачи синхронизации данных на мобильных платформах показал, что в настоящее время наиболее широко распространена самостоятельная реализация алгоритмов синхронизации данных непосредственно разработчиками приложений. Сервисов и платформ, позволяющих решать эту задачу, еще не так много, и их использование не является повсеместной практикой. Преимущества подобных сервисов понятны, и можно ожидать возрастающего интереса разработчиков к данному направлению развития информационных технологий.

Предложенная в данной работе модель платформы синхронизации данных позволяет построить прозрачную схему взаимодействия участников обмена информацией, наглядно представив правила синхронизации для каждого устройства. Подключение новых устройств в сеть происходит путем построения для них правил синхронизации данных и не требует каких-либо изменений в уже настроенных частях сети.

Программная реализация предлагаемой модели существенно упростит жизнь разработчиков мобильных приложений, которым можно будет сосредоточиться целиком на функциональности собственных проектов. Единственное, что необходимо для синхронизации данных – определить один раз правила синхронизации, после чего она будет осуществляться без участия разработчиков.

Список литературы

1. *Androutsellis-Theotokis S., Spinellis D.* A Survey of Peer-to-Peer Content Distribution Technologies // ACM Computing Surveys. 2004. Vol. 36 (4). P. 335–371.

2. *Кутовенко А*. Как настроить собственный SyncML-сервер // Мир ПК. 2012. № 1.

3. *Демиш В. О.* Интеграция SaaS-сервисов: актуальные проблемы, интеграционные платформы // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2012. Т. 2, вып. 2. С. 5–13.

4. *Ambler S., Lines M.* Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press, 2012.

Материал поступил в редколлегию 17.12.2013

V. O. Demish, B. N. Pishchik

DATA SYNCHRONIZATION ON MOBILE PLATFORMS

This article deals with data synchronization problem encountered on mobile platforms. It contains the analysis of the existing solving methods, standards and tools of data synchronization. A formal model of mobile data synchronization tool is proposed on the basis of the analysis.

Keywords: data synchronization, mobile platforms, databases, integration data, mobile devices.