Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Разработка архитектуры сетевого многопоточного приложения системы мобильного маркетинга на платформе Java ME

Курсовая работа студента 345 группы
Удалова Александра Николаевича

Научный руководитель ……. В.А.Кириллин
кандидат ф.-м.н.

Санкт-Петербург
2010

Оглавление

Введение

SmartKupon – это система, позволяющая пользователям мобильных телефонов и смартфонов получать уникальные скидки в различных местах и заведениях Санкт-Петербурга. В отличие от других средств подобного рода (SMS/Bluetooth-рассылки), система ненавязчива и не создаёт о себе впечатления спама: пользователь может воспользоваться ей в любой момент и только по собственному желанию, равно как и в любой момент может прекратить ей пользоваться. Основным понятием является «купон» – виртуальный объект, имеющий название, описание, изображение и дающий доступ к определённой скидке. Система состоит из нескольких частей, среди них имеет смысл выделить две основные:

1. Сервер, хранящий информацию о доступных купонах, подключённых партнёрах, различную статистику и т.д., а также веб-интерфейс, позволяющий редактировать купоны, наблюдать за деятельностью клиентов и т.д.
2. Клиентские приложения, отображающие в удобном виде информацию о доступных для использования купонов и позволяющие воспользоваться любым из существующих предложений.

Данная курсовая работа посвящена разработке клиентского приложения системы мобильного маркетинга SmartKupon на платформе Java ME.

С точки зрения клиентского приложения, существует два списка купонов – «доступные» и «взятые». В доступных пользователю предлагается выбрать любой купон из существующих в данный момент и переместить во взятые. Взятые купоны – это предложения, заинтересовавшие пользователя, которыми он вскоре может воспользоваться. Придя в заведение, пользователь должен предъявить взятый купон, в результате чего на экране отобразится код, который сотрудник заведения введёт в специальное приложение на своём мобильном телефоне. Если валидация купона пройдёт успешно, пользователь тут же сможет получить заветную скидку в этом заведении.

В проекте SmartKupon я в команде из двух человек занимался проектированием и разработкой логики («бекенда») клиентского приложения на платформе Java ME. К этому, в частности, относятся взаимодействие с сервером по заранее обговорённому протоколу, локальное хранение данных и настроек, логгирование действий приложения и пользователя, и др. Стоит отметить, что в отличие от приложений под другие платформы, одновременно разрабатывающиеся в нашем проекте (такие, как iPhone, Android, Windows

Mobile, Symbian), немало хлопот доставили жёсткие ограничения платформы, происходящие из требования её адекватной работы на большинстве современных мобильных телефонов (даже самых слабых по некоторым показателям) и привели к различным трудностям в процессе разработки, многие из которых были успешно решены, а некоторые решаются до сих пор.

В процессе проектирования были поставлены следующие основные требования:

1. Приложение не должно «зависать», то есть пользователь должен в каждый момент понимать, что происходит в приложении, будь то загрузка информации из сети или рендеринг изображения.

2. Приложение должно быть максимально устойчиво ко всевозможным неполадкам (и адекватно на них реагировать): таким, как отсутствие доступа к сети, нерабочее состояние сервера, плохо сформированный ответ от сервера на запрос, повреждённое хранилище данных, всяческие действия пользователя.

3. Приложение должно быть максимально соптимизировано под работу на мобильных устройствах с небольшими вычислительными возможностями.

Итогом работы стало приложение, устанавливающееся на большинство современных мобильных телефонов с поддержкой Java (MIDP 2.0, CLDC 1.1) и выполняющее поставленную задачу, то есть позволяющее просматривать и предъявлять купоны на выбранные скидки. В данный момент система работает и доступна каждому пользователю мобильного телефона с платформой Java ME.

Обзор

Первые версии приложения, разрабатывавшиеся до поставленных выше требований, обладали существенными недостатками, которые делали их использование непригодным. Ниже изложены основные замеченные проблемы и выбранные решения.

1. Синхронная и асинхронная работа с сетью. Изначально приложение делало все запросы к серверу синхронно, в одном потоке. Это означало, что перед тем, как сделать любой запрос, скажем, запрос на синхронизацию состояния (т.е. спросить у сервера, какие купоны появились и удалились с момента последней синхронизации), приложение показывало экран «Загрузка», отправляло запрос и получало ответ, и только затем показывало содержательную информацию. Помимо запросов синхронизации, другим запросам, таким, как запрос информации о купоне или изображения купона, запрос карты проезда для купона, и др., также предшествовал экран «Загрузка», причём невозможно было отменить процесс загрузки и делать в приложении что-либо ещё, помимо ожидания конца загрузки. Скажем, выбрав в списке купонов приглянувшийся по названию купон, нажав на него, необходимо было дождаться конца загрузки, хотя в это же время пользователь мог бы просматривать другие предложения.

По этим причинам было принято решение разработать асинхронную систему, в которой запросы к серверу не должны были бы выполняться в текущем потоке работы приложения. В то же время, нужно было обезопаситься от ситуации, в которой в приложении в один момент времени работает много потоков, работающих с сетью – это важно, поскольку в силу ограниченной производительности мобильной платформы приложение могло бы начать сильно «тормозить» и вести себя неуклюже. В новой архитектуре количество потоков приложения всегда ограничено константой, потому что на запрос не создаётся нового потока. Скажем, при взятии купона приложение должно отправлять запрос на сервер, но если запросы идут недостаточно быстро, то, даже взяв подряд большое количество купонов, приложение не станет работать медленнее.

2. RMS и JSR 75. При разработке первых версий приложения было решено сделать его таким же, как и на других, более мощных, платформах (таких как iPhone, Android), что сделало бы его более узнаваемым для различных пользователей. В упомянутых приложениях купон представляет из себя два изображения на весь экран

(«передняя» и «задняя» сторона), а также различная текстовая информация. Поскольку изображения достаточно объёмны, приложение должно кешировать его на диске устройства, чтобы не скачивать лишний раз и не тратить деньги с мобильного счёта пользователя.

У Java ME в плане хранения данных дела обстоят несколько сложнее. Спецификация MIDP 2.0 гарантирует наличие постоянного хранилища RMS (Record Management System), собственного для каждого приложения. Однако минимальный размер RMS для одного приложения, определённый спецификацией, по нынешним меркам смехотворен, а это значит, что каждый производитель сам решает, сколько места в постоянной памяти телефона выделять Java-приложениям. На практике встречаются современные телефоны, выделяющие не более 512кб под RMS, что могло быть мало для нашего приложения с обилием изображений.

Первое решение, которое было принято – хранить изображения в файловой системе телефона, используя JSR (Java Specification Request) 75. Это API доступа к файловой системе, которое, однако, поддерживается не на всех современных телефонах. У него есть и другие недостатки: прежде всего, это довольно неудобная спецификация, «благодаря» чему различные производители по-своему реализуют её на своих устройствах (у разных телефонов отличаются такие параметры, как, скажем, название корневого диска или права доступа по умолчанию на каждом диске). Кроме того, поскольку запись в файловую систему может повредить данные пользователя, все современные телефоны перед первым доступом на запись или даже на чтение выводят запрос о безопасности, а многие – при каждом, что может сильно раздражать любого пользователя приложения, в котором общение с файловой системой может происходить постоянно. Тем не менее, приложение разрабатывалось с поддержкой JSR 75, во многом благодаря иллюзии о том, что если приобрести сертификат и подписать им приложение (см. ниже «Сертификация»), то запросы исчезнут, но после подробного изучения сертифицирования выяснилось, что это не так.

Поэтому было принято компромиссное решение – купоны специально для платформы Java ME выпускать в виде небольшой картинки и ещё нескольких параметров (таких, как цвет фона и текста вокруг изображения), а сами изображения (уже на порядок меньшего размера) хранить в RMS. Это и выглядит довольно прилично, и реализация для всех устройств едина, и места в RMS хватит на значительное число купонов (когда место всё же заканчивается, редко просматриваемые изображения удаляются).

3. Сертификация. С самого начала разработки приложения бытовало мнение, что запросы безопасности, которые каждый телефон во время работы приложения спрашивает на доступ в сеть, к файловой системе, к открытию ссылки во внешнем браузере, и т.д. можно убрать, сертифицировав приложение, т.е. сделав его в некотором смысле «доверенным». Это оказалось не так. Первая проблема в том, чтобы получить сертификат: есть две возможности, первая из которых – приобретение у доверенного поставщика, но, к сожалению, на данный момент нет единого поставщика, которому доверяют все популярные производители мобильных устройств, вторая возможность – программа Java Verified, но она не подходит по финансовым причинам. Вторая, и пожалуй, главная, проблема в том, что даже наличие сертификата не гарантирует, что приложение, подписанное сертификатом, не будет спрашивать вопросов о безопасности. Это происходит потому, что производители телефонов сами вольны решать, как относиться к доверенным приложениям, и на практике они доверяют им не намного больше, чем недоверенным, т.е. раздражающие запросы на доступ к файловой системе и сети обычно остаются. По результатам исследования на тему сертификации было принято решение отказаться от использования файловой системы (и пересмотреть всю спецификацию приложения) и смириться с запросом на доступ к сети.

Архитектура

1. События и обработчики. Одним из базовых принципов, на которых основана асинхронная работа приложения, является метод событий и обработчиков. Основной поток работы приложения вместо того, чтобы запустить метод из логики и ждать его окончания, подписывается на событие, т.е. становится обработчиком этого события, и вызывает этот метод в другом потоке, после чего может продолжать работать параллельно с исполняющимся действием из логики. Когда метод завершит работу, он выполнит код всех обработчиков, подписанных на событие о его окончании, и так основной поток узнает о том, что метод, вызванный им, завершился. Например, графический интерфейс должен отобразить изображение купона. Для этого он в другом потоке запускает метод, который начинает загрузку изображения с сервера, но перед этим подписывается на соответствующее событие, т.е. добавляет к событию, взятого у этого объекта-купона, новый обработчик-класс с одним методом, который рисует пришедшее по сети изображение на экране устройства. После того, как изображение скачается, будет запущен обработчик и оно отобразится на экране. С таким подходом у пользователя появилась возможность листать купоны в списке без ожидания, пока каждый из них загрузится. Кроме того, приложение теперь не «зависает», т.е. пользователь может в любой момент перейти в любое меню, не дожидаясь окончания какой-либо загрузки.

2. Задания и рабочие. Как уже было упомянуто ранее, нецелесообразно создавать новый поток на каждое действие, требующее асинхронного выполнения, потому что можно сэмулировать ситуацию, при которой в приложении будет работать неограниченно много потоков, а каждый новый поток требует ресурсов, на которые мобильная платформа не щедра (в частности, новый поток занимает место в оперативной памяти устройства, а также переключение контекста между одновременно выполняющимися потоками занимает время). Поэтому было решено запускать заранее какое-то конкретное количество потоков («рабочих»), которые будут по мере своей занятости принимать «задания» и выполнять их. Задания складываются в одну из блокирующих очередей, а рабочий ожидает появления задания в соответствующей очереди, при появлении вынимает задание из очереди и выполняет его. О реализации стоит отметить, что поскольку Java ME не новая технология, программирование ведётся на языке Java 1.3 с урезанной стандартной библиотекой, где единственным примитивом синхронизации

является монитор со схемой wait/notify (различные другие примитивы появились только в Java 5). Но эта проблема была успешно преодолена на этапе реализации, т.к., как известно, задача о заданиях и работниках решается с помощью любого из общеизвестных примитивов синхронизации, в частности монитора.

Кроме того, решено было создать несколько блокирующих очередей под различные типы заданий:

1) локальные задания – задания, выполняющиеся на текущем устройстве и не требующие доступа в сеть, например, извлечение данных из кеша или запись в кеш;

2) сетевые задания – задания, состоящие в отправке запроса на сервер и принятия и обработки ответа, например, загрузка текстовой информации о купоне;

3) срочные сетевые задания – сетевые задания, которые не имеет смысл откладывать, например, взятие/предъявление/отказ от купона. Выделение этих заданий в отдельную очередь обосновано тем, что если этого не сделать, то пользователю, взявшему купон, пока остальные купоны загружаются, пришлось бы ждать конца загрузки текстовой информации, прежде чем отправить запрос о действии с купоном, что нелогично;

4) задания отправки логов – сетевые задания, которые состоят в отправке логов (текстовой информации о всех действиях пользователя и приложения, см. ниже «Логгирование»);

С точки зрения реализации, задание представляет из себя класс с единственным методом, в котором содержится его суть. Блокирующая очередь оперирует ссылками на эти классы. Более того, чтобы избежать бесполезного повторного выполнения некоторых заданий (к примеру, не нужно дважды синхронизироваться или загружать информацию об одном и том же купоне), решено было модифицировать очередь так, чтобы в неё нельзя было положить элемент, который в ней уже есть, что в свою очередь потребовало корректное определение метода equals() у всех типов заданий. Рабочий представляет из себя класс-наследник Thread с полем-ссылкой на очередь, из которой он берёт задания и методом run(), в котором он в бесконечном цикле берёт задание из своей очереди и выполняет его. Стоит отметить, что рабочих также пришлось поделить на два типа: рабочие локальных заданий и «сетевые» рабочие. Это сделано для того, чтобы сетевой рабочий мог переиспользовать сетевое соединение, однажды установленное с сервером. Остальная часть приложения общается с интерфейсом заданий и рабочих через класс TaskDispatcher, который хранит ссылки на очереди и на рабочих, и умеет, определив тип данного ему задания, добавить его в соответствующую очередь.

3. Логгирование. Важной частью в обеспечении стабильности приложения является сбор и отправка на сервер информации об использовании приложения или логов. Для этого используется система, аналогичная системе log4j: каждый лог имеет один из нескольких уровней, от отладочной информации до фатальной ошибки, класс и метод, в котором происходит запись лога, и само сообщение.

Заключение.

Итогом работы стало клиентское приложение системы SmartKupon, пригодное для использования и показывающее хорошую производительность на большинстве современных мобильных телефонов с поддержкой технологии Java ME. Были приняты во внимание все вышеописанные проблемы и ограничения платформы и реализованы все вышеописанные особенности архитектуры.