Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Translator Widget for Android

Курсовая работа студента 345 группы
Василинца Сергея Павловича

Научный руководитель ……. В.Б. Филиппов
COO SPB Software

Санкт-Петербург
2010

Оглавление

Введение.

Платформа Android.

Android — операционная система для мобильных телефонов, основанная на ядре Linux. Это система довольна молода, первое устройство (HTC T-Mobile G1) с Android было презентовано в сентебре 2008 года. Для сравнения первое устройство с WinMobile – арпель 2000, Symbian Ltd была основана в 1998, а первые смартфоны с этой системой на борту появились также 2000 году. Вернемся к Android, изначально эта система разрабатывалась компанией Android Inc., которую затем купила Google. Впоследствии, Google инициировала создание Open Handset Alliance (OHA), которая сейчас и занимается поддержкой и дальнейшим развитием платформы. На данный момент в OHA входит 65 компаний, включающих в себя производителей мобильных телефонов, разработчиков программного обеспечения, некоторых мобильных поставщиков и изготовителей чипов, среди них: Google, HTC, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, Nvidia. Платформа Android уже очень популярна на рынке мобильных устройств Соединенных Штатов, так согласно данным исследовательской конторы NPD Group в первом квартале 2010 года Android OS обошла iPhone OS по популярности. По данным NPD, на сегодняшний день у Google 28% рынка, что ставит их на второе место после Research in Motion (36%), iPhone OS находится на третьем месте с 21%[1].

Android используется не только, как мобильная операционная система . Например, шведская компания People of Lava анонсировала телевизор на платформе Android, также на Mobile World Congress 2010 было анонсировано несколько планшетов и нетбуков с Android в качестве операционной системы, созданы роботы, управляющим устройством которых являются смартфоны с Android OS. Так же на Google IO 2010, было анонсировано Google TV, основанное тоже на Android OS.

Разработка под Андроид.

Приложения для Android являются программами в нестандартном байт-коде для виртуальной машины Dalvik . Dalvik Virtual Machine — основанная на регистрах виртуальная машина, она оптимизирована для низкого потребления памяти. Программы для Dalvik пишутся на языке Java. Несмотря на это, стандартный байт-код Java не используется, вместо

него Dalvik VM исполняет байткод собственного формата. После компиляции исходных текстов программы на Java (при помощи javac) утилита dx из «Android SDK» преобразует .class файлы в формат .dex (Подробное описание dex формата http://www.dalvikvm.com/), пригодный для интерпретации в Dalvik.

В версии Android 1.5 разработчики добавили Native Development Kit который позволяет писать собственные низкоуровневые модули для системы на языке C/C++, опираясь на стандартные linux-библиотеки.

Постановка задачи.

Изначально задача ставилась — реализовать программу переводчик под Android, со следующим функционалом:

- Использование веб-сервисов машинных переводов и словарей, имеющих API.
- Предложения.
- Распознавание текста.

Так как для предложений нужно создавать серверную часть, которая бы по началу слова выдавала возможные слова, было решено добавить еще одну цель:

- Серверная часть со своими словарями.

Актуальность задачи.

Перед всеми переводчиками, существующими на данный момент, у нас будет преимущество — распознавание текста с фотографии. Однако существует программа Google Goggles, которая также умеет распознавать текст. Давайте рассмотрим её более детально.

Google Goggles.

Изначально, Google Goggles имел своей целью поиск в интернете по картинке, а распознавали они логотипы , штрихкоды, книги, DVD , текст и многое другое, но текст они не переводили. Перевод текста появился в версии 1.1, но распознавание текста идет только с латиницы, то есть иероглифы и кириллица распознана не будет. И переводить соответственно можно на языки, которые для записи используют латиницу. При этом программа делает только машинный перевод, то есть переводит целые предложения и не может выдать различные значения слова.

Теперь становится ясно, что можно противопоставить Google Goggles:

- распознавание кириллицы, иероглифов.
- словари. (то есть умение выдавать несколько переводов одного слова)
- предложения

И последнее , что стоит отметить, что все существующие программы для перевода используют только один источник, из которого берутся эти самые переводы, у нас же стоит цель — поддерживать несколько, что даст возможность пользователю выбирать тот сервис для перевода , который ему больше нравится, или использовать одновременно несколько, получая больше информации.

Средства Разработки.

Клиентское приложение. Android.

Разработка программ ведётся на языке Java, и для этого существует плагин под Eclipse — «Android Development Tools» (ADT), предназначенный для Eclipse версий 3.3-3.5. Для IntelliJ IDEA также существует плагин, облегчающий разработку Android-приложений, но он доступен только в ULTIMATE EDITION, то есть в платной форме.

Серверная Часть.

Серверную часть для словарей было решено писать при помощи Grails. Grails программный каркас для создания веб-приложений, пара слов об его особенностях:

- Для разработки используется динамический язык Groovy, который является расширением языка Java в сторону Smalltalk, Ruby и работает поверх JVM.
- Фреймворк не написан с нуля, а представляет собой скорее DSL поверх Spring & Hibernate.
- Первые два пункта дают простую интеграцию с множеством библиотек и проектов на Java.
- Поддержкой проекта занимается компания SpringSource — один из технологических лидеров в Java-индустрии.

Поддержка разработки под Grails существует в Eclipse IDE в виде плагина, в IntelliJ IDEA, но только в ULTIMATE EDITION, в NetBeans 6.8. IDEA была отброшена в виду платности, а из оставшихся двух вариантов было решено использовать NetBeans, так как: в плагине под

Eclipse почти отсутствует поддержка в редакторе, а также входе работы в самом плагине возникает большое количество ошибок. В то время как в NetBeans отсутствуют эти проблемы, к тому же существуе поддержка плагинов для grails.

Решение поставленных задач.

Создание серверной части со словарями.

Постановка подзадач.

Итак мы хотим создать серверную часть со словарями, хранение словарей реализовано наиболее естественным вариантом — в базе данных , а конкретнее была использована MySQL. Следующий шаг, создание структуры базы данных для словарей, чтобы её создать надо чётко поставить задачи, которые она должна решать. Итак:

- Выдавать структурированный перевод слова по паре языков и самому слову.
- Выдавать подсказки по паре языков и префиксу слова. В подсказку входят само слова и ровно один его перевод.
- Поиск подсказок(предложений) должен происходить максимально быстро.
- Релевантность, наиболее популярные слова должны в подсказках выдаваться первыми.(Популярные в данном случае, это часто переводимые слова).

В первой задаче было использовано словосочетание «структурированный перевод», разъясним, что под ним подразумевалось. Итак, очевидно, что у одного слова может быть более одного перевода, при этом это слово может выступать в роли глагола, существительного и т.п.(пример: drop - капля, капать), иметь несколько существенно разных значений с различными транскрипциями(пример: зАмок, замОк). Если пренебречь этим, и хранить и выдавать эти переводы вперемешку, в результате мы получим базу, которая не подвергается редактированию, а что еще хуже - пользователям придётся с трудом в выискивать нужный им перевод. Поэтому надо выдавать переводы в некотором структурированном виде(например: набор глаголов, набор существительных итп.).

Создание структуры базы данных.

Давайте рассмотрим созданную структуру:

```
class Word {
   // Само слово
   String word
   // язык, из которого это слово
```

```
    String langSource
    String searchKey
   // частота
    int frequency = 0
    static hasMany = [posBeans:PosBean]
}

class Translation {
    // Перевод
    String translation
     static belongsTo=[posBean:PosBean]
}
```

```
class PosBean {
  //От Part of speech., т. е. noun /adj etc
  String pos
  // Транскрипция
  String transcription
  // язык, на который переводится
  String langDest
  static hasMany = [translations:Translation]
  static belongsTo = [word:Word]
}
```

Небольшой комментарий по коду: hasMany и belongsTo определяют отношения между Domain классами. Domain класс — это буква M в MVC, то есть отвечает за данные, для каждого Domain класса создается соответствующая таблица в базе данных [3].

Однако, вернемся к структуре. Наверно, первый возникающий вопрос, зачем в отдельный класс вынесена часть речи (PosBean), ведь для каждого перевода её можно указывать заново, но с таким форматом не удобно было бы работать, из-за того, что результаты надо было бы группировать по части речи, но пришлось бы еще и учитывать транскрипцию для одинаковых частей речи, таким образом сбор результатов сильно усложнился. Рассматривая класс PosBean, мы замечаем, что там заложен язык, на который будет переводиться слово, но слово является той или иной частью речи вне зависимости от того, на какой язык оно переводится. Почему же так сделано? Надо вспомнить, что позже эти результаты будут показаны пользователю, и поэтому надо , чтобы поле pos было на языке, на который переводят(langDest), таким образом langDest поднялся на уровень выше в класс PosBean.

Еще одной альтернативой было бы хранить часть речи и дальше все переводы. (то есть pos = = “verb” translations = “1) drop /n 2) … /n ”). Но такой вариант плохо масштабируем, так в нашей структуре к переводам можем легко добавить поле tag, которое бы отмечало, что такой перевод используется в определенной предметной области (медицина, финансы, it итп), в то время как в альтернативную тэги ввести крайне сложно.

Быстродействие.

Одной из поставлен подзадач было быстродействие подсказок (предложений), но наша структура явно медленная, так как чтобы найти подсказку с переводом , надо сделать два JOIN, (сначала таблицу Word с PosBean, и результат с Translation), и эти действия производят довольно медленно, поэтому пришлось денормализовать нашу структуру, то есть была введен еще один Domain Class:

```
class Suggestion {
    String searchKey
    String word
    String langSource
    String langDest
    String translation
    int frequency = 0
}
```

Хоть в таблице, соответствующей этому классу, данные повторяют часть данных, хранящихся в таблицах, соответствующих Word, PosBean, Translation, зато поиск предложений происходит при помощи одного Select, что ускорило его в 4 раза.

Android. Архитектура приложения.

Постановка подзадач.

Еще одной задачей является создание BackEnd для клиентского приложения. Перед его созданием, определим цели, которые мы хотим достичь:

- Не накладывать ни какие ограничения на поддерживаемые переводческие сервисы.
- Простота добавления нового сервиса для перевода.

Если со вторым пунктом всё понятно, первый требует расшифровки. Разные сервисы для перевода предоставляют разные данные, так какие-то кроме самих переводов еще предоставляют синонимы и транскрипцию, другие сервисы — примеры использования и из какого тематического словаря слово. Итак нашей целью является не обрезать информацию, приходящую от сервисов, и предоставить её в полном объёме.

Подготовительный процесс.

Но перед тем, как строить архитекруту бэк-енда, кратко рассмотрим созданный UI, так как возможно из-за этого нам придётся добавить некоторые специфические требования к бэкенд.

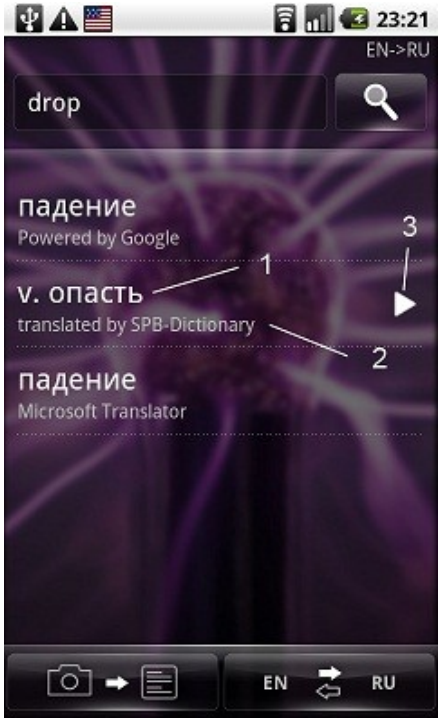
Итак нам надо продумать UI, который сумел бы отображать все переводческие сервисы со всем разнообразием передаваемой информации. Нам надо как-то отображать результаты перевода, как же наилучшим способом это сделать? Первым и самый простой вариант показывать все результаты подряд, но так результаты одних сервисов сложно будет отделить от результатов других, к томуже многие переводы будут повторятся, и в итоге будет сплошная путаница.

Разумным выходом из этой ситуации — писать по одному переводу от сервиса, а по клику переходить на экран, со всеми переводами и другой информацией от сервиса.

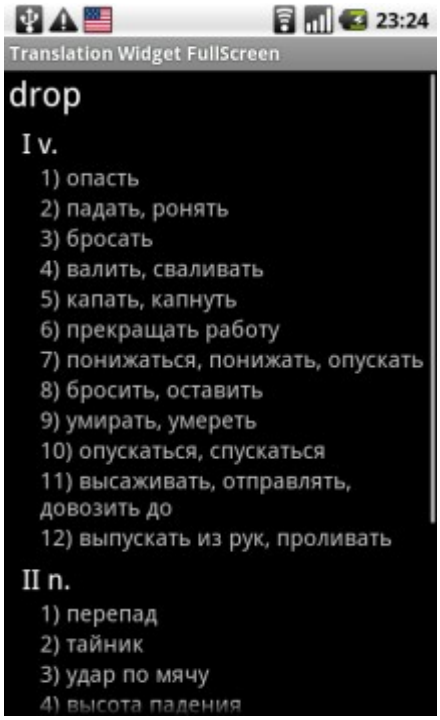
На изображении 1, получивший UI:

- 1 — ровно один перевод от сервиса
- 2 — подпись сервиса , общее требование для использования того или иного вебсервиса.
- 3 — стрелочка уведомляющая пользователя о том, что если нажать элемент списка будет показана дополнительная информация.

На изображении 2 — пример экрана одного сервиса. К сожалению, к нему не был сделан дизайн.



Изображение 1



Изображение 2

Архитектура Back-end.

Теперь перейдем, непосредственно к архитектуре бэкенда, Стандартно в андроиде для бэкенда используют Content Provider. Content Provider сохраняют и извлекают информацию обычно из баз данных, но можно и из других источников. Контент провайдеры имеют синтаксис похожий на синтаксис работы с базами данных, так у них есть методы query, update, delete, которые должны выполнять действия аналогичные действиям с базами данных. Нам это подходит, запросы будут строиться, как будто это запрос идет к базе данных, где лежат переводы, хотя на самом идет запрос в веб-сервис. При вызове query Content Provider возвращают , как несложно догадаться, аналог таблице — Cursor [2].

В итоге, у нас получается следующая последовательность действий: пишем запрос в наш Content Provider, и в результате получаем табличку с результатами. Но тут возникает некоторые проблемы — мы не хотим обрезать данные пришедшие от переводческих веб-сервисов, поэтому, логично ответы от них сохранять в том формате, в котором они были получены, иначе нам придётся создавать структуру для их хранения, которая бы удовлетворяла потребности каждого из сервисов. И даже если мы такую создадим, то появление какого-нибудь нового сервиса может заставить переделывать нас эту структуру, а

это противоречит нашему второму пункту (о простоте добавления).

Итак, передавать в Cursor мы будем непосредственно то, что получили от сервисов. Но тогда у нас в Cursor хранятся данные в формате, который нельзя показывать пользователю (xml, json итп). Отсюда следует, что нам надо как-то эти данные конвертировать, и так как класс Cursor является абстрактным, мы можем с каждой записью в него добавлять некий объект Converter, который бы переводил эту информацию в нужный нам вид (например, изображении 1 для циф. 1, Converter бы выбирал один перевод из информации). А так же этот класс должен был бы создавать экран, как на Изображении 2, то есть выдавать нам View(это класс для визуальных компонент, то есть того, что отрисовывается на экране). Это уже выходит за пределы стандартного Cursor, поэтому нам надо было бы добавить некий новый метод:

View getTranslView(...);

Назовем наш новый Cursor(с добавленным методом) — TranslCursor.

Но реализовать такую архитектуру не возможно, так как обращения к Content Provider идут не напрямую, а через специальный класс ContentResolver, то есть query указывается:

ContentResolver cResolver;

…

Cursor cursor = cResolver.query(...);

И переменная cursor, полученная в результате, будет враппер классом над курсором, который реально был получен от Content Provider, то есть этот класс будет содержать нужный нам TranslCursor, но доступ будет дан только к методам, которые есть у стандартного Cursor [4].

Поэтому было решено передавать в Cursor необработанную информацию и имя сервиса, а дальше после получения информации из Cursor, мы будем обращаться к классу соответствующего вебсервиса, то есть класс работы с веб сервисом должен будет содержать методы, для выделения нужных данных из необработанных данных.

Для того, чтобы вызывать эти методы, мы создадим класс TranslationFactory(названный так в связи , с некоторой схожесть на паттерн Factory, но полного совпадения с этим паттерном нет). Этот класс содержит статический Map, где ключом является ключом имя сервиса, а значение — это объект сервис. TranslationFactory содержит те же методы что и TranslationService, только лишь с той разницей, что в каждый метод передается имя сервиса. Эти методы в свою очередь вызывают аналогичный метод, у сервиса с переданным именем. Таким образом все обращания идут не напрямую к сервисам,а к нашему классу

TranslationFactory.

С учётом сделанного UI и выше сказанного, определим interface TranslationService , который будут определять классы для работы с веб-сервисами для переводов :

```
interface TranslationService {
    TranslationRecord getTranslation(String text, String from,
                                     String to) throws IOException;
    String getSignature();
    String getName();
    boolean operateWith(String first, String second);
    View dataToView(String source, String data, Context context);
    String listString(String data);
    String shorcutString(String data);
    boolean hasMore(String data);
}
```

Рассмотрим чуть более детально, методы.

Метод getTranslation(...) , делает запрос в интернет и отдаёт информацию в Cursor.

Методы listString(...), getSignature и hasMore(...) соответственно обеспечивают работу пунктов 1,2,3 на Изображении 1.

Метод shortcutString(...) - выдает один перевод для сохранения истории перевода.

Метод operateWith(...) - узнает у сервиса поддерживает ли он данную пару языков, чтобы не обращаться к сервисам, которые заведомо не дадут никакого перевода. Метод dataToView(...) - создаёт экран, который отображается по нажатию на элемент списка, пример на Изображении 2.

В результате, мы создали интерфейс, который поможет нам решать одну из поставленных перед нами задачи: при помощи dataToView всегда будет возможность отображать специфические данные, при помощи других методов мы создаём однообразный экран с превью от разных сервисов.

Вернемся к классу TranslationFactory и вспомним о нашей второй задаче — надо упростить добавление сервиса. Класс TranslationFactory здесь выполняет важную роль - минимизирует доступ к классам сервисов. Представим, что в нескольких местах в коде идут напрямую обращения к этим классам, в каждом таком месте при добавлении нового сервиса, мы должны будем изменять код, и каждое такое место — место для потенциальной ошибки, так как однажды мы забудем в одном из мест изменить код. А у нас получается, чтобы

добавить еще один сервис нужно добавить реализацию TranslationService и добавить этот сервис в мэп в TranslationFactory. Альтернативой такой реализации было использование рефлексии (java.reflection) в связке с конфигурационным файлом. В таком случае надо было добавить реализацию интерфейса TranslationService и строчку в конфиг файле, о том что появился новый сервис, у такого варианта плюс в том, что не надо в искать в коде строчку, где дописать сервис. Но при этом усложняется код, и при случайном изменении конфигурационного файла или опечатке может возникнуть Runtime Error, в реализованном же варианте опечатка обнаружится еще во время компиляции.

Заключение.

Результат.

На данный момент реализованы:

- Серверная часть для переводов и подсказок.
- Серверная часть для распознавания, в виду того, что запущена на виртуальной машине, которой выделено крайне мало вычислительных мощностей, распознавание работает крайне медленно.
- Создан клиент, поддерживающий предложение, распознавания и несколько переводческих сервисов.

Развитие.

Фотографии.

На данный момент используется стандартное (встроеннное) приложение для камеры, но на некоторых устройствах, это приложение работает не так, как ожидалось, выдавая вместо полноразмерной картинки, сжатую. Поэтому надо реализовать свое фотографирование. Также это позволит уменьшить количество действий, для того чтобы распознать картинку, потому что можно будет сразу выделить область фотографии, с которой надо распознать текст.

Переводчик.

Сейчас UI приложения сделан так, что переводить удобно пару слов, но не целое предложение, то есть оно использовать его удобно, как словарь, но не как переводчик. Такой функционал был бы полезен так был найден соответствующий Use Case. Два основных use case для нашей программы:

- Человек знает язык, с которым работает, и тогда он будет использовать нашу программу, как словарь, то есть ему надо перевести слово или словосочетание.
- Если же человек совсем не знает язык, то он, естественно, захочет перевести предложение целиком.

Таким образом нужно разработать UI, в котором было бы удобно работать с более длинными(более 4 слов) кусками текста.

Список литературы:

[1]Android Shakes Up U.S. Smartphone Market//Сайт NDP Group URL:

http://www.npd.com/press/releases/press_100510.html

[2]Android Providers//Сайт разработчиков Android URL:http://developer.android.com/guide/topics/providers/content-providers.html (дата обращения: 26.05.2010)

[3] Mastering Grails: Many-to-many relationships with a dollop of Ajax

//Сайт IBM URL: https://www.ibm.com/developerworks/java/library/j-grails04158/ дата обращения: 26.05.2010)

[4] Документация по классу ContentResolver// Сайт разработчиков Android URL: http://developer.android.com/reference/android/content/ContentResolver.html (дата обращения: 26.05.2010)

[5] the grails 1.1 ide smackdown // Блог о grails : http://www.grailsblog.com/archive/show?id=10 (дата обращения: 26.05.2010)