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

УДК 004.057.5

Обзор и анализ методов обеспечения совместимости приложений для операционной системы Android

***Баришок Н.И.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю.Е., к.т.н., доцент,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gapyu@bmstu.ru

Android является динамично развивающей операционной системой. Это подтверждает тот факт, что с момента выхода первого устройства на Android 1.0 (T-Mobile G1/HTC Dream) в октябре 2008 года, каждый год выходила новая «главная» версия платформы. Так, в декабре 2011 года появилось первое устройство (Galaxy Nexus) на платформе Android 4.0. Помимо основных версий, 1.0 – 4.0, за этот промежуток времени вышло некоторое количество «промежуточных» версий, например, 2.1, 2.2, 2.3 и т.д. Каждая из этих версий привносила весомый вклад в развитие операционной системы. К моменту написания статьи, последней версией Android является Android 4.4.

На рисунке 1 представлена диаграмма распределения относительного количества версий платформы.

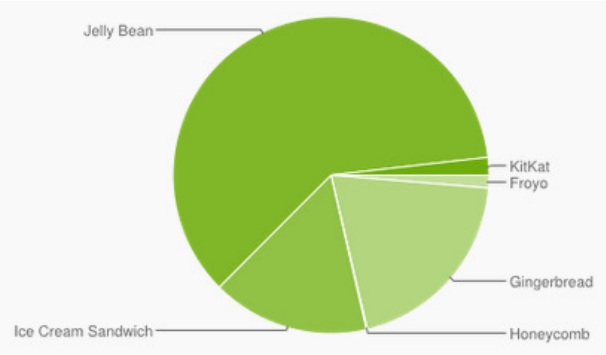

Рис. 1. Диаграмма распределений версий Android к 4 февраля 2014

Описание диаграммы сведено в таблицу 1.

Таблица 1

Процентное отношение количества версий платформы Android к 4 февраля 2014.

Версия	Название	API	Распределение
2.2	Froyo	8	1.3%
2.3.3-2.3.7	Gingerbread	10	20.0%
3.2	Honeycomb	13	0.1%
4.0.3-4.0.4	Ice Cream Sandwich	15	16.1%
4.1.x	Jelly Bean	16	35.5%
4.2.x		17	16.3 %
4.3		18	8.9 %
4.4	KitKat	19	1.8 %

Исходя из данных таблицы можно отметить, что, версии Android, начиная с 4.0.3 занимают большую часть диаграммы, 78.6%. В целом, этот показатель является достаточно значительным, однако, он также означает, что более 20% устройств всё еще используют 2.x.x версию Android. С учетом того, что общее число «активированных» устройств в 2013 году достигло 900.000.000 [2], 20% является оказывается внушительным числом. Хорошей новостью является тот факт, что это число уменьшается с каждым месяцем. Это подтверждается данными таблицы 2, где представлена статистика, актуальная на май 2013 года.

Таблица 2

Процентное отношение количества версий платформы Android к 1 мая 2013.

Версия	Название	API	Распределение
Android 4.1 – 4.2.x	Jelly Bean	16-17	28.4%
Android 4.0.3 - 4.0.4	Ice Cream Sandwich	15	27.5%
Android 3.1 – 3.2	Honeycomb	12-13	0.1%
Android 2.3 - 2.3.7	Gingerbread	9-10	38.5%
Android 2.2	Froyo	8	3.7%
Android 2.1	Eclair	7	1.7%
Android 1.6	Donut	4	0.1%

Анализируя данные представленных выше таблиц, можно сделать вывод, что число устройств с Gingerbird и ниже с мая 2013 упало примерно в 2 раза, в то время, как число устройств с Ice Cream Sandwich (ICS) выросло за данный отрезок времени примерно в полтора раза, что является хорошей динамикой. Но, конечно, предстоит провести еще немало работы и обновлений устройств, прежде чем Android 2.x.x наконец исчезнет.

Отсюда, перед Android - разработчиком становится задача обеспечения как обратной, так и прямой совместимости разрабатываемого приложения. «Обратная совместимость» означает, что приложение, разработанное для конкретной версии платформы, будет запускаться и в более поздних версиях этой платформы. «Прямая совместимость» подразумевает, что приложение, разработанное для определенной версии платформы, корректно функционирует в более ранних версиях. Таким образом, данный термин можно обобщить, - разработанное приложение корректно функционирует в как можно более широком интервале актуальных версий платформы. В целом, задача обеспечения совместимости в мире разработки программного обеспечения является довольно распространенной, но в Android она поставлена острее, чем в других платформах. Так происходит во многом из-за того, что платформа развивается крайне динамично, - «релизы» новых версий происходят чаще, чем в других платформах, добавляя огромное количество новых функций, оптимизируя старые решения. Вдобавок к этому, немаловажным фактом является и то, что данная платформа установлена на различные мобильные устройства различных фирм-производителей, которые имеют отличия в своих аппаратных реализациях.

Целью данной статьи является обзор и анализ выработанных решений по обеспечению совместимости при разработке android-приложений. Также в статье описывается обобщённый список действий, которые необходимо выполнить для корректного обеспечения совместимости разрабатываемого приложения.

Итак, первый шаг в отношении обеспечения обратной и прямой совместимости – определиться с интервалом поддерживаемых версий платформы. Выбор в данном вопросе во многом зависит от средства распространения приложения. Суть заключается в том, что различные средства распространения приложений (будь то Google Play, Amazon AppStore Barnes и т.д.) имеют свою аудиторию, поэтому нужно учитывать какие устройства используют эти средства распространения.

Этот выбор далее должен быть отражен в коде разрабатываемого приложения. Для этого в приложении существует специальный файл конфигурации, *AndroidManifest.xml*, в котором имеется тег <*uses-sdk ... />* , который, несмотря на название, отвечает за уровень

API, используемый приложением. Уровень API это целочисленное значение, которое идентифицирует версию API, используемой в конкретной версии платформы Android[3].

-*android:minSdkVersion*. Значением данного атрибута является наиболее старая версия платформы, для которой планируется поддержка.

-*android:targetSdkVersion*. Значением данного атрибута является версия платформы, на которую в целом ориентировано приложение. Данный параметр определяется набором функций, которые планируются быть примененными в разрабатываемом приложении. Так, например, указывая *android:targetSdkVersion=11+*, разработчик получает доступ к такому элементу управления, как *ActionBar* и остальным дополнениям, появившимся вместе с версией платформы *HoneyComb*. По умолчанию значение данного атрибута совпадает со значением атрибута *android:minSdkVersion*. С другой стороны, значение данного параметра определяет версию API, для которой было проведено тестирование и, соответственно, подразумевается, что для данной версии не должно быть никаких проблем с совместимостью. Таким образом, непосредственно операционная система будет иметь возможность реализовать ожидаемое поведение приложения, у которого *android:targetSdkVersion* меньше, чем уровень API операционной системы.

-*android:maxSdkVersion.* Данный атрибут определяет последнюю версию платформы, которую разрабатываемое приложение имеет целью поддерживать. Использование данного атрибута не приветствуется, потому как он никаким образом не влияет на поддержание прямой и обратной совместимости, являясь только дополнительным аргументом для фильтрации теоретически подходящих приложений.

Данные значения анализируются как при установке приложения на устройство (кроме *android:maxSdkVersion*, начиная с Android 2.0.1), так и при публикации его в различные средства распространения. Например, Google Play использует систему фильтров, благодаря которым пользователю доступны только те приложения, которые совместимы с версией его устройства.

Хорошей практикой является увеличение значения *android:targetSdkVersion* при появлении нового уровня API в новых выпусках платформы, а также последующее тестирование приложения с данной версией API.

Помимо данных параметров, существует еще один, который встречает наибольшее непонимание разработчиков, - т.н. *build target*. Он не имеет мало общего с предыдущими параметрами, не является частью тега <*uses-sdk ... />* и подразумевает следующее:

-версия библиотеки классов Android, с использованием которой производится компиляция приложения. Таким образом, определяется набор типов и методов, которые доступны во время разработки приложения.

-правила, которые необходимо применить при интерпретации ресурсов и манифеста. Таким образом, определяется набор ресурсов приложения, которые не могут быть использованы во время компиляции.

Хорошей практикой для значения атрибута *build target* является, очевидно, минимальный уровень API, поддерживаемый приложением.

Следующим шагом является определение необходимости использования дополнительных библиотек, которые позволяют реализовать необходимые приложению функции на устройствах с более старым уровнем API.

Одним из важнейших вопросов, требующих детального рассмотрения, является вопрос поддержки такого элемента управления приложением как *Action Bar*. *Action Bar* появился вместе с версией Android 3.0 (уровень API 11) и стал неотъемлемой частью любого Android-приложения, которое придерживается паттернов разработки пользовательского интерфейса для платформы Android.

Таким образом, если разрабатываемое приложение ориентировано на поддержку версий Android старше, чем Android 3.0, необходимым является реализация данного элемента управления нестандартными средствами. Среди доступных вариантов реализации можно выделить следующие:

-собственная реализация;

-android support library;

-ActionBarSherlock.

Среди критериев выбора можно выделить следующие:

-сложность интеграции в приложение;

-поддержка библиотеки с выходом новых версий Android;

-качество поддержки *Action Bar*;

-степень гибкости;

-наличие примеров/документации.

Первый вариант в большинстве случае неуместен, по тем причинам, что он обладает следующими недостатками:

-требует большого времени на создание;

-все нововведения встроенного в платформу *ActionBar'a* должны быть сымитированы вручную;

-вероятность того, что приложение будет отличаться от приложений, использующих встроенный *ActionBar*, который привычен пользователю.

Но плюсом, несомненно, является наибольшая вероятная гибкость итогового продукта.

Вкратце рассмотрим оставшиеся два варианта. С практической точки зрения, несмотря на то, что *ActionBarSherlock* больше не поддерживается, данный вариант является более приемлемым с той точки зрения, что обладает более полной документацией с всеобъемлющим набором примеров, а также, что еще более важно, изначально позиционируется как расширение для *android support library*, которое избавляет данную библиотеку от присущих ей проблем. Несмотря на тот факт, что с новыми ревизиями android support library избавилось от многих недостатков, переход от *ActionBarSherlock* к android support library должен сопровождаться как минимум тщательным тестированием нового приложения.

Таким образом, логично предположить, что, на момент написания статьи, выбор между *ActionBarSherlock* и *android support library* это скорее вопрос вкуса. Отсутствие поддержки для *ActionBarSherlock* компенсируется как минимум таким же уровнем стабильности работы с текущими версиями Android и большей гибкостью. Также стоит учесть тот факт, что в скором времени большинство разработчиков откажется от поддержки «устаревших» версий Android и, таким образом, не будет необходимости в использовании данных библиотек.

Еще одним методом помощи в обеспечении совместимости является возможность узнать в коде уровень API устройства, которое выполняет приложение. Это достигается путем использования элементов перечисления *android.os.Build.VERSION_CODES* и добавления логики в зависимости от результата проверки версии платформы. Стоит отметить, что данный метод действителен начиная с уровня API 4, что является приемлемым, потому как более ранние версии платформы уже не поддерживаются.

Типичной архитектурой для построения компонентов, имеющих различную реализацию в зависимости от версии платформы, на которой они выполняются, является архитектура, изображенная на рисунке 2.

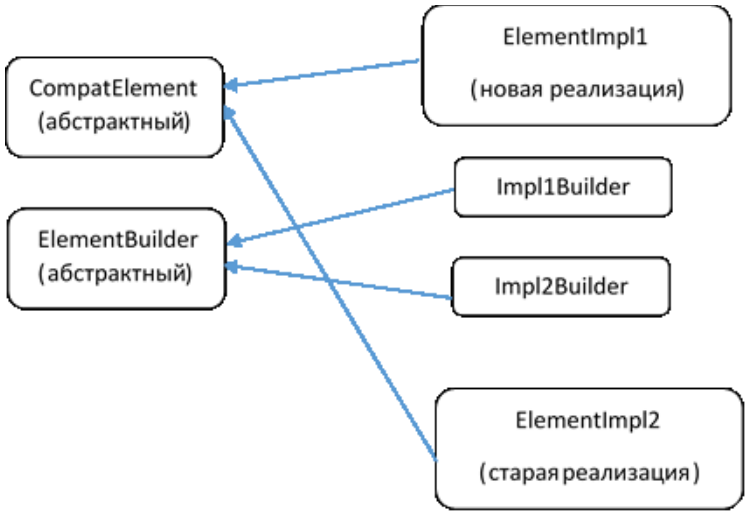

Рис. 2. Вариант реализации архитектуры, поддерживающей компоненты с поведением, зависимым от версии платформы

В данном случае *ElementImpl1* и *ElementImpl2* являются реализациями компонента для новой и старой версий платформы, а *Impl1Builder* и *Impl2Builder* являются реализациями вспомогательных классов, задачей которых является создание экземпляров старых или новых компонентов, соответственно. Целью *ElementBuilder'a* является предоставление функций фабрики объектов, возвращающей экземпляр класса, актуальный версии платформы. Что касается ресурсов, используемых данными компонентами, то, так как они описываются с помощью xml, для них предусмотрены специальные техники управления версиями, рассмотренные детально ниже в статье. В качестве примера можно рассмотреть реализацию компонента "Tab", который работает как со старыми, так и новыми версиями платформы. Код примера представлен ниже.

```
public abstract class CompatTab {
    ...
    public abstract CompatTab setText(int resId);
    public abstract CompatTab setIcon(int resId);
    public abstract CompatTab setTabListener(
            CompatTabListener callback);
    public abstract CompatTab setFragment(Fragment fragment);

    public abstract CharSequence getText();
    public abstract Drawable getIcon();
    public abstract CompatTabListener getCallback();
    public abstract Fragment getFragment();
    ...
}

public abstract class TabHelper {
public static TabHelper createInstance(FragmentActivity activity) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            return new TabHelperHoneycomb(activity);
        } else {
            return new TabHelperEclair(activity);
```

```
        }
    }

    // Usage is mTabHelper.newTab("tag")
    public CompatTab newTab(String tag) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            return new CompatTabHoneycomb(mActivity, tag);
        } else {
            return new CompatTabEclair(mActivity, tag);
        }
    }

}

public class CompatTabHoneycomb extends CompatTab {
    //Так как в данной версии поддерживается Tab по умолчанию, данный класс          //выступает в качестве
прокси для реального объекта Tab
    ActionBar.Tab mTab;
   ...

//реализация перегруженных методов опущена
}

public class TabHelperHoneycomb extends TabHelper {
    ActionBar mActionBar;
    ...

    protected void setUp() {
       ...

    }

    public void addTab(CompatTab tab) {
        ...
    }

                ...
}

//реализация Tab для старых версий опущена, так как в данном случае важна //только архитектура, суть в
том, чтобы объект функционировал в результате //как ActionBar.Tab

public class CompatTabEclair extends CompatTab {
       ...
//реализация перегруженных методов опущена
       }

    ...
}

public class TabHelperEclair extends TabHelper {
    private TabHost mTabHost;
    ...

    protected void setUp() {
       ...    }

    public void addTab(CompatTab tab) {
        ...
          }
}
```

Что касается внешнего вида компонентов, то, как известно, в Android внешний вид описывает с помощью xml-структур. Таким образом, xml-описания для компонентов новых версий должны быть расположены в папке *res/layout-v11/... ,* в то время, как соответствующие xml-структуры для старых версий должны располагаться в папке *res/layout/….*

В целом, стоит отметить, что в Android используется суффикс –vNN для именования папок, где NN-номер уровня API, начиная с которого будет использоваться xml-ресурс внутри. Таким образом, имея папки *res/layout-v11/* и *res/layout/*, для уровня API 11+ Android будет выбирать специфичную наиболее подходящую папку (*res/layout-v11/*), в то время как для остальных уровней наиболее подходящей папкой будет являться папка по умолчанию *res/layout/.* Данный вид именования работает для версий Android 2.0+.

Таким образом, в статье были рассмотрены причины необходимости разработки приложений для платформы Android с учетом совместимости различных версий. Также был проведен обзор и анализ методов обеспечения совместимости разрабатываемых приложений, приведены соответствующие примеры.

Список литературы

1. Mark L. Murphy. The busy coder’s guide to Android development. CommonsWare, 2013.
2. Hugo Barra. Google I/O 2013 – Android Design for UI Developers. Режим доступа: http://www.youtube.com/watch?feature=player_detailpage&v=Jl3-lzlzOJI#t=224 (дата обращения 15.02.2014).
3. Официальный ресурс Google для android-разработчика. Режим доступа: http://developer.android.com/index.html (дата обращения 15.02.2014).
4. Juhani Lehtimaki. Smashing Android UI. John Wiley & Sons, Inc., 2013.