Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Предметно-ориентированное моделирование приложений для платформы Android

Курсовая работа студентки 345 группы
Никоновой Ольги Анатольевны

Научный руководитель ……. Т.А. Брыксин
ст. преподаватель

Санкт-Петербург
2010

Оглавление

1 Введение

Создание предметно-ориентированного решения - это способ качественного повышения эффективности разработки. За счет поднятия уровня абстракции предметно-ориентироавнный язык моделирования получается намного проще и лаконичнее, чем язык целевой платформы. Но успешность такого решения всягда напрямую зависит от правильного ограничения предметной области.

В данной работе была сделана попытка оценить применимость данного похода к достаточно широкой области задач - разработке приложений для мобильной платформы Android. Вопрос о применимости к данной области интересен в первую очередь в силу ее обширности и сложности.

На данный момент уже существуют примеры успешного создания предметно-ориентированных решений для разработки мобильных приложений. Стивен Келли и Юха-Пекка Толванен в своей книге “Предметно-ориентированное моделирование”[1] описывают предметно-ориентированный язык моделирования, созданный в компании Nokia для создания приложения для операционной системы Symbian, которая устанавливалась на мобильные телефоны, производимые корпорацией. Но мощь Android OS как мобильной платформы намного больше, и поэтому приложения, написанные под нее, могут быть куда более сложными и разнообразными. Таким образом, встает вопрос о применимости подхода к такой похожей, но при этом более глубокой предметной области.

Невозможно создать предметно-ориентированный язык, не определив точно границы предметной области. Часто бывает так, что говоря о множестве решаемых задач на самом деле подразумевают не его, а достаточно точное его приближение. Большая часть из них покрывается решением, но остаются и те, для которых оно не применимо. Чаще всего это задачи, решение которых требует индивидуального подхода. Предметно-ориентированное решение всегда ориентированно именно на общую массу задач, не претендуя на способность решить что-то, требующее индивидуального подхода. Таким образом и в данном случае скорее всего должны остаться приложения, которые будет невозможно создать с помощью данного решения.

Целью данной работы являлось выделение четко определенного класса приложений для платформы Android и создания для нее предметно-ориентированного решения. Основной задачей было выделение такого набора предметных понятий, который имел бы высокий уровень абстракции, но при этом позволял реализовать генерацию кода.

2 Обзор

2.1 Предметно-ориентированный подход

Предметно-ориентированный подход в программировании заключается в выделении специфической предметной области и создании для нее языка моделирования с более высоким уровнем абстракции.

При этом подходе рассматривается узкая предметная область, в которой выделяются основные понятия. Выделение обычно осуществляется с помощью экспертов предметной области или разработчиком с большим опытом написания приложений в данной области. Из этих понятий формируется язык моделирования, который используется при разработке и потом транслируется в язык целевой платформы.

Суть предметно-ориентированного подхода в том, что задачи из узкой предметной области во многом сильно схожи между собой. Поэтому для описания каждого конкретного решения достаточно описать только его уникальные свойства, отличающие его от остальных решений. И описывать эти отличия в терминах предметной области значительно проще. Таким образом значительная часть реализации, которая является общей для всех задач данной области, скрывается, и при разработке требуется только описание уникальных особенностей конкретной задачи через абстракции предметной области.

Выделение высокоуровневых абстракций возможно и для достаточно широких предметных областей. На этом основываются такие языки моделирования, как UML. Но в таком случае задача генерации целевого кода серьезно усложняется. Возникают проблемы при построении отображения из высокоуровневых абстракций в понятия целевой области. В силу широты предметной области обычно существует более одной возможной реализации для каждого понятия, при этом нет возможности выбрать, какая именно из них будет более эффективной. В некоторых случаях лучше будут работать одни реализации, в некоторых - другие. Таким образом, в задачу генерации кода из такого языка моделирования автоматически включается задача выбора реализации для каждой абстракции. Принять фиксированное решение для каждой абстракции - значит сгенерировать код, который заведомо будет менее эффективен, чем написанный вручную. Попытка же выбирать более подходящую реализацию “на лету” в процессе генерации кода сделает задачу написания генератора крайне сложной.

Предметно-ориентированный подход предлагает решение для этой проблемы. При сужении предметной области задача выбора реализации может решиться сама за счет предметной специфики. Таким образом, главной задачей становится выделение предметной области и абстракций в ней так, чтобы была возможна “однозначная” генерация, но при этом уровень абстракции не упал бы до уровня целевого языка. Также важно, чтобы сама по себе предметная область не была бы слишком узкой и представляла собой некий востребованный класс задач.

Предметно-ориентированное решение состоит из нескольких составляющих:

- Язык моделирования создается на основе понятий из предметной области и предназначен для описания специфики конкретных задач в рамках области. То, что является общим для всех задач предметной области, не описывается.
- Модели создаются на этом языке в специальном редакторе. Редактор может быть разработан специально для данного решения, но это требует дополнительные средства и время. Поэтому обычно для создания редактора используется уже существующая metaCASE-система.
- Генератор кода транслирует созданные модели в код целевой платформы.

Иногда в составляющие предметно-ориентированного решения также включают целевую платформу.

Основное преимущество предметно-ориентированного подхода - в повышении эффективности разработки. После того, как создан язык и написан генератор, разработка в предметной области становится в разы эффективнее ручного кодирования.

Кроме того, предметно-ориентированным языком могут пользоваться не только программисты, но и эксперты из предметной области, далекие от программирования.

Таким образом, основная трудность заключается в выборе предметной области и разработке языка. Если это сделано правильно, то написание генератора будет решаемой задачей.

2.2 Предметная область

Android OS - это операционная система для мобильных устройств, в основе которой лежит виртуальная машина Dalvik, запущенная на ядре Linux. Dalvik по своей сути является аналогом Java Virtual Machine, и разработка программ для нее также ведется на языке Java.

Хотя Java является языком общего назначения, приложения для платформы Android достаточно специфичны. При их разработке может быть использован только определенный набор библиотек. Кроме того, у Android OS свой собственный графический интерфейс со своей системой событий, своя поддержка многопоточности и много классов и библиотек, отвечающих за специфические для мобильного устройства вещи.

Более того, само по себе класс мобильных приложений имеет достаточно сильную специфику. Это происходит из-за особенностей мобильных устройств и их использования. Мобильные приложения - это чаще всего либо игры, либо программы, ориентированные на работу с веб-сервисами. Во многих из них используется стандартный графический интерфейс, а при разработке для многих задач применяются готовые решения, которые предоставляет платформа. Набор таких решений покрывает почти все возникающие задачи - от хранения настроек до поддержки многопоточности. К тому же максимальное использование возможностей платформы является для андроид-разработчика хорошим тоном.

Предметно-ориентированный подход эффективен только в том случае, когда предметная область представляет собой класс однотипных задач. При моделировании мы по сути описываем различия между разными задачами, поэтому для того, чтобы уровень абстракции языка моделирования был выше, чем у целевого, необходимо чтобы и для описания различий между задачами было достаточно более высоких абстракций.

Таким образом, не имеет смысла пытаться применить предметно-ориентированный подход для разработки игр, потому что они недостаточно сильно похожи между собой, и каждая в отдельности является уникальным продуктом. Но приложения, ориентированные на работу с веб-сервисами (клиенты для социальных сетей, электронные журналы и т.д.), являются достаточно однотипными для того, чтобы к ним мог бы быть применен предметно-ориентированный подход.

2.3 Существующие аналоги

Наиболее известным аналогом подобного предметно-ориентированного решения является Google App Invertor for Android. Этот эксперметальный проект был выпущен в 2010 году, и до сих пор находится в состоянии бета-версии.

Google App Invertor позиционируется как средство для разработки простых андроид-приложений ориентированный на простых пользователей. Он является полноценным предметно-ориентированным решением, позволяющий графически описывать интерфейс пользователя и поведения программы с помощью специального редактора, доступного через веб-браузер. После этого пользователь имеет возможность протестировать полученное приложение на встроенном эмуляторе, а потом загрузить его на свой телефон.

Данное решение - пример успешного применения предметно-ориентированного подхода. Его эффективность демонстрирует, в первую очередь, простота использования. Любой пользователь смартфона на Android, посмотрев коротокое обучающее видео, сможет создавать приложения с его помощью. С другой стороны, на этом примере очень наглядно видно жесткое ограничение предметной области. С помощью Goggle App Invertor невозможно создать приложение со сложной логикой работы, и он неприменим в промышленной разработке.

Решение, которое разрабатывается в данной работе, отличается от Google App Invertor тем, что имеет более широкую предметную область. При этом оно расчитано на использование программистами, знакомыми с платформой Android, а не на простых пользователей. Поэтому набор абстракций, которые используется для создания языка моделирования, отличается от аналогичного набора в Google App Invertor.

3 Описание решения

При описании данного предметно-ориентированного решения сначала будет рассмотрен выбор предметной области и возможности ее расширения. Далее будут последовательно описаны компоненты предметно-ориентированного решения - язык моделирования, редактор моделей и генератор кода.

3.1 Ограничение предметной области

Как уже говорилось ранее, в качестве предметной области рассматриваются приложения для платформы Android, использующие стандартные средства, предоставляемые платформой и ориентированные на работу с веб-сервисами.

При андроид-разработке сама концепция приложения играет отнюдь не первоочередную роль. При полном внешнем различии с точки зрения использования и идеи, два приложения могут быть крайне близки по структуре и реализации. Куда большее влияние оказывает то, какие именно возможности платформы используются.

Отсюда следует, что ограничивать предметную область надо не с “внешней” стороны. Надо разделять приложения не по принципу задач, которые они решают, а с точки зрения того, какие возможности платформы в них используются.

Возможности плтаформы не равнозначны между собой, и кроме неотъемлимых элементов, без которых не может обойтись ни одно приложение, есть много “дополнительных” средств для работы со специфическими аппаратными устройствами, специализированными протоколами или нестандартными графическими возможностями.

Поэтому можно формировать различные предметные области, выделив некий базовый набор средств и добавляя к нему дополнительные возможности.

При разработке языка моделирования применялся именно такой подход. Были выбран некий базовый набор функциональности, на основе которого и был создан язык. Так же были рассмотрены возможные пути расширения языка за счет добавления “дополнительной” функциональности.

Преимущество подобного способа определения предметной области заключается в его гибкости. При грамотном выборе базовой области и архитектуры генератора, добавление новых возможностей не должно усложнять генерацию, таким образом баланс между шириной предметной области и сложностью генерации не будет нарушен. При этом, можно легко расширять предметную область при возникновении потребности в какой-то новой возможности.

3.2 Язык моделирования

Язык моделирования описывает работу приложения в терминах предметной области, т.е., в данном случае, в терминах платформы Android. Основные требования, которые выдвигаются к нему - высокий уровень абстракции и удобство использования. В языке не должно возникать низкоуровневых терминов, и он должен быть интуитивно понятен для андроид-разработчика.

Так же язык должен удовлетворять требованиям расширяемости. То, что предметная область задается в терминах доступных для разработчика возможностей платформы, должно отразиться на языке. Абстракции этих компонент платформы должны перейти в язык, и он должен иметь возможность расширятся за счет их добавления.

Работа мобильного приложения - это в первую очередь его взаимодействие с пользователем. Процессы, которые выполняются программой в фоновом режиме, являются второстепенными, и этот принцип очень важен для платформы Android. Главное - то, что пользователь видит на экране, и, значит, описывать работу приложения надо тоже в терминах отображаемых графических элеметов.

Таким образом, в основу языка должен быть положен поток выполнения программы, описанный с помощью элементов пользовательского интерфейса. В разрабатываемом языке поток выполнения описывается с помощью переходов между различными состояниями, которые, в большинстве случаев, описывают то, что в данный момент отображается на экране.

Но это лишь общая концепция. Деятельность сколь либо сложной программы нельзя описать только через элементы графического интерфейса. В качестве абстракций в язык необходимо ввести и понятия предметной области, не имеющие графического представления. Это, в первую очередь, работа с “внешней средой” - интернет-запросы, звонки, работа с таким аппаратными возможностями платформы, как использование камеры или GPS-навигация.

Другая важная составляющая программы, не имеющая графического представления - это работа с информацией. Язык должен иметь средства для описания процесса получения информации, ее отображения и передачи между различными компонентами приложения.

Для этого в разрабатываемом языке используется собственное пространство переменных, для возможности ссылаться на полученную информацию и отображать ее.

При описании приложения возникает потребность в более детальном рассмотрении некоторых его составляющих. Язык должен иметь средства для подробного описания сложных элементов пользовательского интерфейса, обработки результатов интернет-запросов и т.д.. Но слишком большое количество дополнительной информации перегрузило бы основную диаграмму описания работы программы. Поэтому было принято решение создавать дополнительные языки для подробного описания таких вещей. В данный момент рализованы только два таких языка, но при расширении предметной области могут быть созданы новые.

Эти дополнительные виды диаграмм, по сути, не описывают работу программы с другой точки зрения, а лишь являются уточнениями к основной диаграмме, в соотвествующих элементах которой на них имеются ссылки.

Далее будут подробно описаны три языка диаграмм и связи между ними.

3.2.1 Main Diagram

Основной язык описывает поток выполнения программы. Он организован как машина состояний. Состояния, т.е. вершины, можно разделить на две группы: на отображающие изменения пользовательского интерфейса и на описывающие внутренние процессы приложения. Последние позволяют подробно задавать реакцию приложения на действия пользователя.

Ключевую роль в андроид-приложениях играют объекты Activity и их жизненный цикл. Эти объекты являются компонентами приложения, которые предоставляют экран для взаимодействия с пользователем. У каждого объекта есть окно, в котором он отрисовывает свой графический интерфейс. Андроид-приложения обычно состоят из нескольких связанных между собой Activity.

Узлы данной диаграммы можно разделить на три логические группы. Первая группа отвечает за управление жизненным циклом объектов Activity. Вторая - за дополнительные графические элементы, выполняющие функций оповещения пользователя. И, наконец, третья группа - это так называемые неграфические элементы, описывающие обработку информации, незаметную для глаз пользователя.

В данный момент в языке реализована лишь частичная поддержка управления жизненным циклом Activity. Есть возможность описать логику поведения при его создании и при закрытии.

- **Create Activity**
    - Данный элемент соотвествует созданию нового объекта Activity. В него включено описание содержимого окна в момент создания в виде ссылки на диаграмму описания раскладки.
- **Set Content View**

    - Этот элемент отвечает за глобальное изменение содержимого окна Activity. Здесь, как и в предыдущем элементе, присутствует ссылка на Layout Diagram.
- **Create Tab Layout**
    - Данный элемент позволяет создать окно, содержащее в себе несколько объектов Activity, каждый из которых располагается в отдельной вкладке.
- **Close Activity**
    - Элемент указывающий на закрытие последнего созданного Activity.

Элементы, отвечающие за дополнительные графические объекты, управляют поведением диалогов и уведомлений.

- **Show Text Dialog**
    - Создание диалога с текстовым сообщением и несколькими кнопками.
- **Show Custom Dialog**
    - Создание диалога, содержащего более сложную структуру из графических элементов. В свойствах этого элемента присуствует ссылка на диаграмму с описанием расположения графических элементов.
- **Show Progress Wheel**
    - Создание всплывающего окна с крутящимся колесиком загрузки.
- **Close Dialog**
    - Элемент, указывающий на закрытие последнего созданного диалога.
- **Toast Notification**
    - Всплывающее уведомление с заданным текстом и длительностью.
- **Custom Toast Notification**
    - Всплывающее уведомление с более сложным содержимым. В свойствах есть ссылка на диаграмму с описанием расположения графических элементов.

Среди “неграфических” элементов важную роль играет управление объектами Service. Эти объекты используются для выполнения ресурсоемких заданий в фоновом режиме. Особенностью платформы Android является то, что она очень внимательно относится к основному потоку, в котором выполняется отрисовка графических элементов. При попытке заблокировать этот поток приложение будет остановлено с ошибкой, таким образом все процедуры, выполнение которых занимает ощутимое время, должны выполняться в других потоках.

Объекты Service бывают нескольких видов, и в данном языке реализована поддержка только “привязынных сервисов” - объектов, которые предоставляют вызвавшему их объекту Activity интерфейс для взаимдойствия, позволяющий получить уведомление о завершении выполнения работы сервисом.

- **Create Bound Service**
    - Элемент, отвечающий за создание нового сервиса. Он является “развилкой” в потоке выполнения программы.
- **Service Success**
    - Элемент, “обратный” предыдущему. Отмечает завершение выполнения работы сервисом и “точку слияния” потоков выполнения.

Так же к данной группе должны быть отнесены все элементы, отвечающее за обмен информацией с “внешним миром”. В данный момент реализована поддержка http-запросов.

Подробно элементы Main Diagram описаны в Приложении А.

3.2.2 Layout Diagram

В Android сложные элементы графического интерфейса описываются с помощью соответствующих xml-файлов. Каждый такой файл описывает раскладку графических элементов в каком-то родительском элементе. Такие xml-файлы имеют древовидную структуру, что позволяет представить это описание в виде модели, узлами которой являются графические элементы.

В качестве связей логично использовать отношения вложенности. Но в графическом описании важен порядок следования вложенных элементов, поэтому было принято решение добавить в качестве связи отношение следования. Таким образом у каждого элемента может быть не более одной связи типа “вложение” и одной - типа “следование”.

Узлы данной диаграммы тоже можно разделить на логические группы. Есть элементы, отвечающие за тип раскладки, за организацию вложенных элементов, за представление некоторого контента и за элементы управления.

В данный момент реализовано три вида раскладок.

- **Frame Layout**
    - Простейший вариант раскладки, который располагает своего единственного потомка на всем доступном ему месте.
- **Linear Layout**
    - Раскладка, располагающая своих потомков последовательно по горизонтали или вертикали.
- **Table Layout**
    - Раскладка, располагающая своих потомков в виде таблицы. Для задания рядов из элементов используются объекты **Table Row**.

За организацию элементов отвечают следующие вершины:

- **Gallery**
    - Представляет своих потомков в виде прокручивающегося по горизонтали списка, при этом в каждый момент времени полностью виден ровно один потомок.

- **Grid View**
    - Располагает свои потомком в виде сетки, прокручивающейся по вертикали.
- **Scroll View**
    - Позволяет своему единственному потомку прокручиваться по вертикали.
- **List View**
    - Располагает своих потоков в виде списка.

Элементы, отвечающие за представление некоторого содержимого:

- **Image**
    - Представление картинки.

- **Text View**
    - Представление фиксированного текста.
- **List of Text View**
    - Фиксированное множество объектов Text View, организованных в виде списка.

Элементы управления имеют в среди своих свойств связанные с ними переменные, что позволяет ссылаться на них в основной диаграмме.

- **Button**
    - Кнопка
- **Edit Text**
    - Редактируемое текстовое поле
- **Check Box**
    - Флажок
- **Spinner**
    - Выпадающий список
- **Item**
    - Элемент List View, Gallery или Grid View. Является оберткой, хранящей переменную в качестве свойства.

Элементы Layout Diagram подробно описаны в Приложении В.

3.2.3 Handler Diagram

Данная диаграмма используется для описания обработки xml-файлов, полученных в качестве результата http-запроса. Она позволяет выделять структуры с нужной информацией и сохранять их в переменные или списки. Для случаев, если выделенную структуру надо сохранять полностью, предусмотрена возможность заворачивать информацию в специализированные объекты. Такая возможность может понадобится, к примеру, при обработке RSS-потока.

Узлы диаграммы, отвечающие за выделение элементов xml и создания из них структур:

- **Tag**

- **Parameter**
- **Text**

Вершины, являющиеся представлением переменных:

- **Variable**
- **List**

Также существует вершина **Object**, которая выражает то, что выделенная структура из элементов xml будет храниться целиком в виде объекта.

Элементы Handler Diagram подробно описаны в Приложении С.

3.3 Редактор моделей

Редактор моделей для данного предметно-ориентированного решения был создан с помощью metaCASE средства QReal. Данное средство позволяет описать язык моделирования с помощью специального графического метаредактора, а потом на основании созданной модели генерирует редактор для данного языка.

Полученный редактор позволяет создавать модели из описанных элементов и связей. Он осуществляет проверку корректности “на лету”, но только в той мере, в которой ее можно описать в метаредакторе. Поэтому окончательная валидация моделей будет происходить в генераторе непосредственно перед обработкой.

Метаредактор QReal позволяет описывать язык моделирования в терминах узлов и ребер. Таким образом описываются все виды вершин и переходов для каждой диаграммы, которая в свою очередь соотвествует языку моделирования. Для каждого узла есть возможность описать его типизированные атрибуты.

Но данное описание не позволяет проверить корректность пространств имен для переменных и кпопок, а так же корретность расположения ребер между вершинами. Поэтому окончательная валидация моделей происходит уже непосредственно в генераторе кода.

Модели, созданные с помощью редактора, сохраняются в репозитрий QReal. Данный репозиторий предоставляет API для генератора, с помощью которого он получает информацию о сохраненных моделях и обрабатывает ее.

3.4 Генерация кода

Основная диаграмма задает набор основных компонент приложения. С ее помощью определяются все объекты Activity и Service. На основе описываемого потока выполнения формируется содержимое методов вызова этих объектов, отвечающих за различные составляющие их жизненного цикла. По сути эти методы являются основной частью генерируемого кода. Код, соотвествующий остальным элементам диаграммы помещается именно в эти методы.

Кроме объектов Activity и Service, на основе главной диаграммы генерируется Android Manifest - файл, описывающий компоненты приложения и точку входа в него.

Диаграммы описания расположения графических элементов соотвествуют xml-файлам, которые являются ресурсами андроид-приложения. При генерации узлы диаграммы отображаются в соотвествующие им теги с параметрами, соответсвующими их свойствам. Они располагаются в порядке, определенном с помощью ребер, задающих отношения “следования” и “вложения”.

Диграммы описания обработки результатов http-запроса используются для генерации объектов Handler, который используется xml-парсером. Если в диаграмме используется узел Object, создается специальный класс, соотвествующей описанной структуре данных, экземпляры которого используются потом для передачи данных.

4 Возможности дальнейшего развития

4.1 Расширение языка

Основным направлением развития является расширение предметной области за счет добавления поддержки новых возможностей платформы. Это возможно делать как с помощью добавления новых элементов в уже существующие диграммы, так и создавая новые виды диаграмм.

В Main Diagram можно добавить другой вариант поддержки многопоточности с использованием обычно механизма потоков Java. В этом случае требуется создать специальный элемент - зону для нового потока, в котором выполнять соотвествующие действия.

Также можно добавить полную поддержку сервисов. В данный момент поддерживаются только так называемые Bound Service - сервисы, связанные с Activity и общающиеся с ним через интерфейс, подобный удаленному вызову процедур. Кроме таких сервисов существует и такие, которые выполняют некоторую работу в фоновом режиме, но не поддерживают связи с вызвавшим их Activity.

Еще одна важная возможность расширения - более полная поддержка жизненного цикла Activity. В идеале должна быть реализована возможность задавать реакцию в случае каждого перехода в цикле. Каждому такому переходу соответствует метод callback-вызова (onCreate, onClose, onPause, onResume и т.д.), и для полного контроля над программой у разработчика должна быть возможность реализовать каждый из этих методов.

В числе дополнительных возможностей надо упомянуть поддержку большего вида диалогов. В данный момент поддерживаются только AlertDialog и диалог с произвольным содержимым. В то же время платформа предоставляет еще несколько видов диалогов, потребность в использовании которых может возникнуть достаточно часто - DataPickerDialog и TimePickerDialog, для выбора даты и времени соответственно.

Из “неграфических” элементов в данный момент тоже реализована только малая часть. Нужна поддержка телефонных звонков и SMS-сообщений, а также работа с аппаратной частью - камерой, акселерометром и т.д.

Еще одной возможностью развития для основной диграммы является добавление поддержки работы с виджетами. Для этого потребуется ввести достаточно большое количество новых абстракций, соответствующих механизму обслуживания виджета виджет-провайдером.

У Layout Diagram также есть много возможностей расширения.

Есть некоторое количество видов раскладок, которые не поддерживаются или поддерживаются частично. Tab Layout в данный момент может применяться только при создании двух различных Activity в качестве вкладок. Возможно, есть смысл добавить и возможность создавать Tab Layout внутри одного Activity. Также в данный момент полностью отсутствует поддержка Relative Layout.

Также в данный момент нет элементов, соотвествующим многим базовым виджетам - View Flipper, Google Map View, AutoComplete, Toggle Button, Rating Bar, Radio Button. Так же есть возможность добавить поддержку произвольных кнопок со своей раскладкой. Но все эти элементы аналогичны многим из уже существующих и их добавление не должно потребовать много усилий.

Еще одна возможность расширения языка - добавление более гибкого управления раскладкой элементов. Это можно сделать, расширив поддержку стандартных свойств графических объектов, управляющих их расположением. Добавление более сложных свойств, управляющих раскладкой объектов в терминах “притяжения”, увеличила бы свободу при задании графического интерфейса.

Есть еще несколько возможностей расширения языка, добавление которых, возможно, потребует создание новых видов диаграмм или, по крайней мере, изменений сразу в нескольких существующих.

Первая из таких возможностей - поддержка меню. Скорее всего она потребует изменений, в первую очередь, в основной диаграмме, поскольку неразрывно связано с созданием активити. Это может быть решено как с помощью добавления новых атрибутов в элемент Set Content View, так и с помощью создания нового специализированного элемента, отвечающего за создание меню. Но и в том, и в другом случае потребуется специальная диаграмма, описывающая содержимое меню и связанные с ним переменные. Также возможно добавление контекстных меню к элементам Layout Diagram, что в свою очередь повлечет в ней изменения.

Другая воможность, которая также может потребовать создание нового вида диаграмм - поддержка локализации. Для этого может использованы таблицы локализации, сопоставляющие строки на различных языках и представляющие их переменные. Это решение вполне соотвествует механизму локализации, реализованному в платформе Android.

4.2 Возможности развития генератора

На уровне генерации также возможны некоторые улучшения. В данный момент конечным результатом является код на Java и специфические для Android xml-файлы. Все это требует ручной сборки для получения готового приложения.

Таким образом возможностью расширения является создание инструмента для сборки готового андроид-пакета. Для этого необходимо реализовать дополнительный этап в работе генератора, на котором будет происходить сертифицирование, версионирование и запаковка приложения. Поскольку на этом этапе возникает необходимость в дополнительной информации от пользователя, возможно потребуются еше некоторые изменения в языке моделирования.

5 Заключение

Целью данной работы было продемонстрировать возможность применения предметно-ориентированного подхода к такой широкой предметной области, как разработка приложений для платформы Android. Результатом работы стала реализация данного подхода для узкой предметной области приложений и концепция расширения данной области. Таким образом, был продуман подход к разработке предметно-ориентированных решений для широких предметных областей.

При выполнении данной работы был разработан прототип предметно-ориентированного решения для базовой предметной области приложений для платформы Android. В рамках данного решения создан язык моделирования простых приложений для работы с веб-сервисами с поддержкой работы с Activity и Service, построения графических интерфейсов с помощью основных элементов, осуществления и обработки http-запросов и базового оповещения пользователя. Также разработана концепция дальшейшего развития данного языка. С помощью системы QReal был создан редактор моделей. Также был разработан генератор, получающий из редактора модели и транслирующий их в целевой код.

Созданное решение демонстрирует применимость предметно-ориентированного подхода для данной области, а описанная концепция развития - возможность расширения данного решения на содержательный класс задач.

6 Приложения

6.1 Приложение А

Подробное описание элементов Main Diagram.

6.1.1 Вершины

Название	*Комментарий*	*Свойства*
Begin	Точка входа приложения	
End	Завершение работы приложения	
Create Activity	Создание нового Activity	label: String layout: LayoutDiagram
Set Content View	Задание или изменение содержимого Activity	layout: LayoutDiagram
Http Request	Http-запрос по данном адресу и обрабатывает его в соотвествии с данной Handler Diargam.	url: String handler: HandlerDiagram

Show Text Dialog	Создание AlertDialog с заданным тесктовым сообщением и кнопками	text: String positiveButton: String negativeButton: String neutralButton: String isCancelable: boolean
Show Custom Dialog	Создание Dialog с произвольным содержимым	title: String layout: LayoutDiagram isCancelable: boolean
Show Progress Wheel	Создание Dialog с колесиком загрузки	
Close Dialog	Закрытие диалога	
Toast Notification	Простое Toast Notification с заданным сообщением и временем	text: String duration: ToastDurationValue[1]
Custom Toast Notification	Toast Notification с данным содержимым и временем	layout: LayoutDiagram duration: ToastDurationValue[1]
Create Tab Layout	Создание Tab Layout с отдельным Activity в каждой вкладке.	tabsText: String[] tabsIds: String[] selectedIcons: path[] deselectedIcons: path[]
Create Bound Service	Создание сервиса.	
Service Success	Успешное завершение работы сервиса.	

[1] ToastDurationValue = {Long, Short}

6.1.2 Ребра

Название	*Комментарий*	*Направленность*	*Свойства*
onCreate	Действия при создании Activity	От Create Activity	
onClose	Действия при закрытии Activity	От Create Activity	
onButtonClick	Действия при клике на данную кнопку	От элементов, где определяется данная кнопка	buttonName: String

onItemClick	Действия при клике на элемент обернутый в элемент Item		itemName: String
success	Действия при успешном запросе	От Http Request	
fail	Действия при ошибке запроса	От Http Request	
onDismiss	Действия пи закрытии диалога	От Show Custom Dialog или Show Text Dialog	
tab	Вкладка с новым активити	От Create Tab Layout к Create Activity	tabId: String
mainStream	Продолжение основного потока	От Create Bound Service	
serviceStream	Поток сервиса	От Create Bound Service	

6.2 Приложение B

Подробное описание элементов Layout Diagram.

6.2.1 Вершины

Название	*Комментарий*	*Свойства*
View	Общий предок для все элементов. Инкапсулирует общие для всех свойства.	layout_width: layoutValue[1] layout_height: layoutValue[1] padding: int[4]
Linear Layout	Соотвествует одноименному виду раскладки. Может быть корнем.	orientation: orientationValue[2] background: {path to image}
Frame Layout	Соотвествует одноименному виду раскладки. Может быть корнем.	background: {path to image}
Button	Соответствует кнопке. Атрибут label - имя переменной, ассоциируемой с данной кнопкой.	name: String label: String
Image	Соотвествует одноименному виджету, отбражающему картинку.	source: {path to image} scaleType: scaleValue[3]

Text View	Соотвествует одноименному виджету, представляющему собой многострочное текстовое поле.	text: String
Edit Text	Соотвествует одноименному виджету, представляющему собой редактируемое текстовое поле. Атрибут var соответсвует имени переменной, ассоциируемой с данным элементом.	isPassword: boolean var: String
Table Layout	Соотвествует одноименному виду раскладки. Может содержать только вершины Table Row.	
Table Row	Соотвествует строке таблицы.	
Gallery	Соответствует виджету, отображающему элементы в виде горизонтального прокручивающегося списка.	
GridView	Соотвествует виджету, отображающему элементы в виде прокручивающейся сетки.	columnWidth: int verticalSpacing: int horizontalSpacing: int numColumns: int stretchMode: stretchValue[5]
ScrollView	Соответствует виджету, содержащего в себе ровно один элемент и создающего для него возможность скроллинга.	
Spinner	Соответсвует виджету выпадающего списка.	itemsContent: String[] itemsIds: String[]
CheckBox	Соответствует виджету чекбокса.	text: String default: checkValue[4] var: String
List View	Соответсвует виджету, представляющего свои элементы в виде списка.	

List of TextViews	Представляет собой частный случай списка отображающий TextView в качестве элементов.	text_view_layout_width: layoutValue[1] text_view_layout_height: layoutValue[1] itemsContent: String[] itemsIds: String[]
Item	Элемент List View, Gallery или Grid View. Является оберткой, хранящей переменную в качестве свойства.	variable: String

[1] layoutValue = {fill_parent, wrap_content}
[2] orientationValue = {horizontal, vertical}
[3] scaleValue = {matrix, fitXY, fitStart, fitCenter, fitEnd, center, centerCrop}
[4] checkValue = {checked, notChecked}
[5] stretchModeValue = {no_stretch, strectch_spacing, stretch_spacing_uniform, stretch_column_width}

6.2.2 Ребра

Название	*Комментарий*
contains	Связь типа “содержит”. Каждая вершина должна содержать не более одной и содержаться не более чем в одной.
followingBy	Связь типа “следовать за”. За каждой вершиной должно следовать не более одной и она сама должна следовать не более чем за одной.

6.3 Приложение C

Подробное описание элементов Handler Diagram.

6.3.1 Вершины

Название	*Комментарий*	*Свойства*
Tag	Соотвествует xml-тэгу.	name: String
Parameter	Соответсвует xml-атрибуту.	name: String
Text	Соотвествует тексту, вложенному в xml-тэг.	

Object	Соотвествует некоторому специализированному объекту.	name: String
List	Соотвествует списку.	name: String
Variable	Соотвествует переменной.	name: String

6.3.2 Ребра

Название	*Комментарий*	*Направленность*
saveAs	Указание сохранить данную конструкцию в виде специального объекта.	От Tag к Object
saveIn	Указание сохранить в данной переменной.	От Parameter, Text, Object к Variable
collectIn	Указание сохранить в данном списке.	От Parameter, Text, Object к List
enclose	Отношения вложения между xml-элементами.	От Tag к to: Parameter, Text, Tag

7 Список литературы

1. Kelly S., Tolvanen J.-P., Domain-Specific Modeling: Enabling Full Code Generation, John Wiley & Sons, Inc., Hoboken, 2008
2. Luoma J., Kelly S., Tolvanen J.-P., Defining Domain-Specific Modeling Languages: Collected Experiences, OOPSLA Workshop on DSM, 2004
3. Narraway, D., Designing and generating mobile phone applications, Presentation at MetaEdit Method Seminar, 6th Nov, Helsinki, Finland, 1998
4. Sprinkle J., Mernik M., Tolvanen J.-P., Spinellis D., What Kinds of Nails Need a Domain-Specific Hammer, IEEE Software, July /August 2009, c.15-18
5. Кузенкова А.С., Литвинов Ю.В., Брыксин Т.А., Метамоделирование: современный подход к созданию средств визуального проектирования // Материалы второй научно-технической конференции молодых специалистов «Старт в будущее», посвященной 50-летию полета Ю.А. Гагарина в космос. СПб. 2011. С. 228-231
6. Android Dev Guide, http://developer.android.com/guide/index.html
7. Android Resources, http://developer.android.com/resources/index.html
8. Android Reference, http://developer.android.com/reference/packages.html
9. Google App Invertor, http://appinventor.googlelabs.com