К вопросу использования компонентов из Android Architecture Components в мобильном приложении на платформе Android

В.А. Жжонов, В.А. Евсина, С.Н. Широбокова

Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова, Новочеркасск

Аннотация: Мобильные приложения повсеместно используются множеством людей в повседневных задачах. С каждым годом возрастает потребность во все более функциональных, удобных и надежных программных средствах, которые смогут обеспечить быструю и безопасную работу в различных областях. Но для разработки такого приложения необходимо использовать соответствующую всем требованиям архитектуру. В данной работе пойдет речь об использовании компонентов из *Android Architecture Components*, разработанных компанией *Google*, которые позволяют реализовать некоторые паттерны проектирования с учетом особенностей операционной системы *Android*. В статье будет приведен перечень наиболее используемых компонентов, а также краткая справка по их функциональным возможностям. Рассмотрена работа одного из компонентов с базовыми элементами операционной системы *Android*. Также схематично показано взаимодействие компонентов на примере реализации одного из паттернов проектирования.

Ключевые слова: архитектура приложения, *Android Architecture Components*, приложение, *Android*.

Мобильные приложения являются наиболее удобным способом взаимодействия с информационным миром [1-2]. Каждый день мы пользуемся огромным количеством различных программ, которым приходится решать задачи разного уровня [2-4]. С каждым годом для приложений задаются всё новые требования в плане безопасности системы, скорости ее работы, возможности расширения и многие другие требования [4-5]. Это заставляет разработчиков искать различные пути решения для вновь и вновь поставляемых им задач.

Разработка любого приложения начинается, во многом, с выбора шаблона проектирования программы, который будет определять принципы работы программы и содержать в себе различные решения, позволяющие избежать множество неприятных и проблемных ситуаций в работе приложения. Выбор подходящего паттерна проектирования архитектуры позволяет достичь высокого уровня гибкости приложения, минимизировать

количество кода, обеспечить обширные возможности для дальнейшего расширения и многое другое.

При разработке приложений под операционную систему «*Android*», не было конкретных требований или указаний по выбору того или иного подхода, поэтому разработчикам приходилось самим разрабатывать инструменты или же использовать готовые решения для реализаций известных паттернов проектирования таких как, например, «*MVC*» и «*MVP*», в рамках данной операционной системы. Это заставило компанию *Google* задаться решением данного вопроса. В конечном итоге это привело к тому, что в 2017 году на ежегодной конференции «*Google I/O*» был представлен набор специальных библиотек – «*Android Architecture Components*» [6]. Данные решения направлены на помощь разработчикам в построении архитектуры, которая бы во всех смыслах подходила под особенности работы операционной системы «*Android*», что являлось знаменательным событием для всех разработчиков приложений под данную платформу.

В список библиотек входили следующие компоненты: «*Lifecycles*», «*LiveData*», «*ViewModel*» и «*Room Persistence*». Кратко разберем каждый компонент:

– «*Lifecycles*» позволяет определять жизненный цикл «Activity» приложения [6], который проиллюстрирован на рис. 1.

– «*LiveData*» работает на основе паттерна «издатель-подписчик», хранит в себе какое-либо значение и позволяет осуществить подписку на него так, что при изменении данных подписчики получат уведомление об этом [6].

– «*ViewModel*» является объектом, в котором реализуется логика работы по хранению и управлению данными, компонент существует до полного закрытия «Activity» или «Fragment», к которому он привязан [6].

– «*Room Persistence*» является удобным ORM решением для работы с *SQLite* [6].

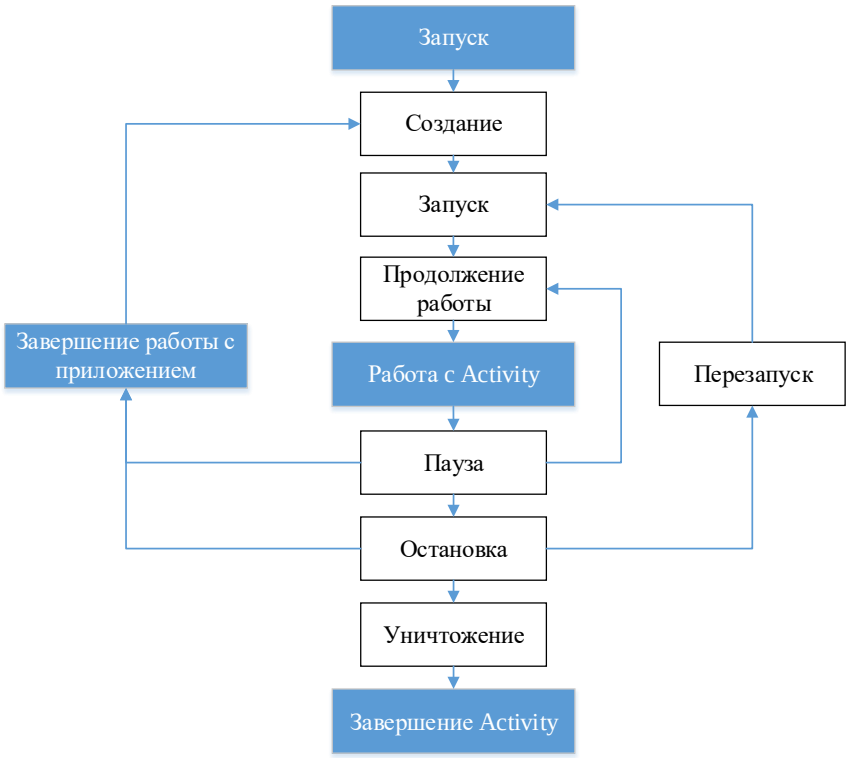

Рис. 1. – Схема цикла *«Activity»*

Общая схема использования этих модулей представлена на рис. 2.

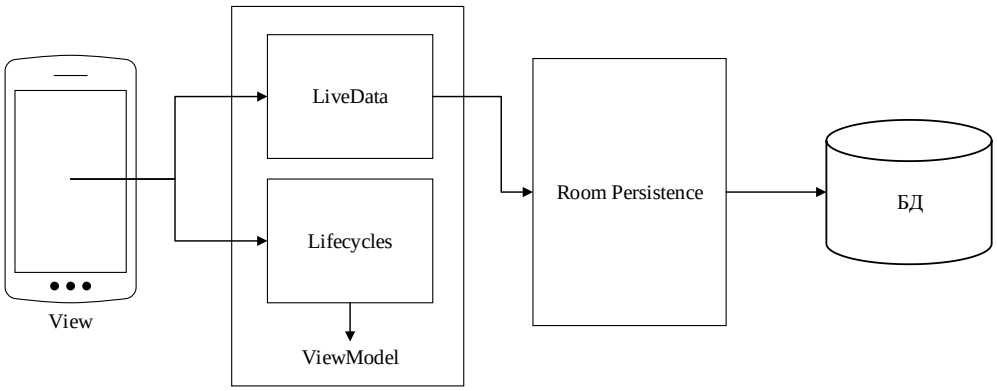

Рис. 2. – Схема общего взаимодействия модулей

Рассмотрим применение такого компонента, как «*ViewModel*», на основе воссоздания классического паттерна «*MVP*» или же «*Model View Presenter*». На рис. 3 показана схема работы данного шаблона.

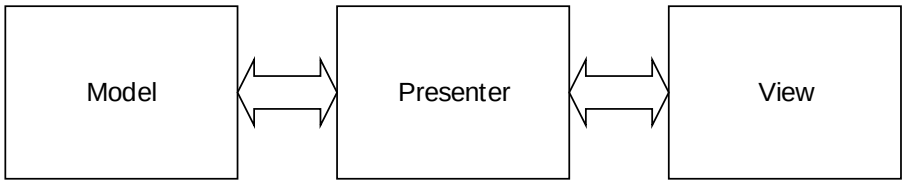

Рис. 3. – Схема шаблона проектирования «*Model View Presenter*»

В случае разработки по данному паттерну приложения на «*Android*», применяя объекты данной системы, а также используя компонент «*ViewModel*», схему можно представить, как показано на рис. 4.

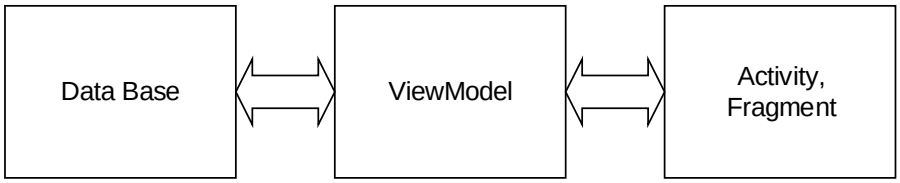

Рис. 4. – Схема шаблона проектирования «*Model View Presenter*» с использованием «*ViewModel*»

Рассмотрим преимущества использования данного компонента. Из-за особенностей системы такие объекты, как «*Activity*» [7] или «*Fragment*» [8], могут неоднократно пересоздаваться в процессе работы, например, при перевороте экрана. В этом случае все данные, находящиеся в них, будут уничтожены. Поэтому написание кода бизнес-логики внутри данных классов является ошибочным решением [9]. «*ViewModel*» существует вне «*Activity*» («*Fragment*») и уничтожается лишь после окончательного закрытия последнего [10]. Следить за состоянием формы помогает «*Lifecycles*», который уже встроен в «*ViewModel*». Внутри компонента реализуются методы по работе с данными, в том числе и с помощью других компонентов, которые были представлены ранее.

Рассмотрим работу «*ViewModel*» на примере. При открытии формы «Fragment» будет создан соответствующий ему «*ViewModel*», в который поступит сообщение о необходимости вывода списка данных, после чего, метод, реализуемый в компоненте, запросит информацию у базы данных. После того, как данные будут отобраны, они поступят во «*ViewModel*», а

оттуда уже на отображение во «Fragment». При повороте экрана «*Fragment*» будет пересоздан и подключен к советующему ему «*ViewModel*», который уже содержит необходимые данные и просто отправит их во «Fragment». Так, не нужно снова делать запрос в базу данных, который может быть затратным процессом, в случае простого поворота экрана. Ранее разработчикам приходилось самим создавать решения для реализации такой возможности.

Ниже приведен фрагмент кода из мобильного приложения, который показывает присоединение к «*Fragment*» созданного для него «*ViewModel*», а также вызова метода для получения данных. В данном случае «*TourViewModel*» наследует базовый класс «*ViewModel*», после чего в нем был реализован метод для запроса информации.

```
public class TourFragment extends Fragment {
    private TourViewModel tourViewModel;
    public View onCreateView(@NonNull LayoutInflater inflater) {
        tourViewModel = new ViewModelProvider(this).get(TourViewModel.class);
tourViewModel.getTourInformation(getArguments().getString(ID_TOUR)); }
```

Итак, разработанный компанией «*Google*» компонент значительно упрощает процесс разработки приложений на операционной системе «*Android*» и позволяет реализовать некоторые паттерны проектирования наиболее эффективно с учетом особенностей работы данной системы. А это значительно улучшает производительность, безопасность и другие показатели приложения.

Литература

1. Mentsiev A.U., Alams M.T. Mobile forensic tools and techniques: Android data security // Инженерный вестник Дона, 2019, №2. URL: ivdon.ru/ru/magazine/archive/n2y2019/5766

2. Александровский В.Г. Мобильные технологии в строительстве. Программное обеспечение на платформе Android. Часть 1 // Инженерный вестник Дона, 2019, №4. URL: ivdon.ru/ru/magazine/archive/n4y2019/5874

3. Robert Destefano. Four steps to successfully deploy android in your supply chain // Supply & Demand Chain Executive, 2018. URL: sdcexec.com/software-technology/blog/20998353/ivanti-supply-chain-4-steps-to-successfully-deploy-android-in-your-supply-chain

4. Paula Natoli. Upwardly mobile—mobile technologies span the supply chain // Supply & Demand Chain Executive, 2015. URL: sdcexec.com/warehousing/article/12091968/upwardly-mobilemobile-technologies-span-the-supply-chain

5. Rachel Harrison, Derek Flood & David Duce. Usability of mobile applications: literature review and rationale for a new usability model // Supply & Journal of Interaction Science, 2013. URL: journalofinteractionscience.springeropen.com/articles/10.1186/2194-0827-1-1

6. Guide to app architecture. URL: developer.android.google.cn/jetpack/guide?hl=en

7. Activity. URL: developer.android.com/jetpack/androidx/releases/activity?hl=ru.

8. Fragment. URL: developer.android.com/reference/android/app/Fragment?hl=ru=ru.

9. Mark L. Murphy. Android’s Architecture Components. M.: CommonsWare, 2019. 413 с.

10. ViewModel Overview. URL: developer.android.com/topic/libraries/architecture/viewmodel.html?hl=ru

References

1. Mentsiev A.U., Alams M.T. Inzhenernyj vestnik Dona, 2019, № 2. URL: ivdon.ru/ru/magazine/archive/n2y2019/5766

2. Alexandrovsky V.G. Inzhenernyj vestnik Dona 2019, № 4. URL: ivdon.ru/ru/magazine/archive/n4y2019/5874

3. Robert Destefano. Supply & Demand Chain Executive, 2018. URL: sdcexec.com/software-technology/blog/20998353/ivanti-supply-chain-4-steps-to-successfully-deploy-android-in-your-supply-chain

4. Paula Natoli. Supply & Demand Chain Executive, 2015. URL: sdcexec.com/warehousing/article/12091968/upwardly-mobilemobile-technologies-span-the-supply-chain

5. Rachel Harrison, Derek Flood & David Duce. Supply & Journal of Interaction Science, 2013. URL: journalofinteractionscience.springeropen.com/articles/10.1186/2194-0827-1-1

6. Guide to app architecture. URL: developer.android.google.cn/jetpack/guide?hl=en

7. Activity. URL: developer.android.com/jetpack/androidx/releases/activity?hl=ru

8. Fragment. URL: developer.android.com/reference/android/app/Fragment?hl=ru=ru

9. Mark L. Murphy. Android's Architecture Components. Moscow: CommonsWare, 2019. 413 p.

10. ViewModel Overview. URL: developer.android.com/topic/libraries/architecture/viewmodel.html?hl=ru