Poznań University of Technology

Faculty of Control, Robotics and Electrical Engineering

Institute of Robotics and Machine Intelligence

Division of Control and Industrial Electronics

Mobile application Software patterns

Mobile and embedded applications for Internet of Things

Teaching materials for laboratory

Dominik Łuczak, Ph.D.; Adrian Wójcik, M.Sc.

Dominik.Luczak@put.poznan.pl
Adrian.Wojcik@put.poznan.pl

I. Goal

Knowledge

The aim of the course is to familiarize yourself with:

- software design and architectural patterns,
- the concept of data binding,
- available programming tools supporting development of mobile apps in accordance with software patterns,
- the concept and applications of dynamic user interfaces,

Skills

The aim of the course is to acquire skills in:

- mobile apps development based on selected software patterns,
- using data binding in mobile apps,
- creating dynamic user interfaces in mobile apps,
- software functionality separation.

Social competences

The aim of the course is to develop proper attitudes:

- strengthening the understanding of the role and application of network communication in IT systems and related security issues,
- strengthening understanding and awareness of the importance of non-technical aspects and effects of the engineer's activities, and the related responsibility for the decisions taken,
- proper technical communication in context of software development,
- choosing the right technology and programming tools for the given problem,
- use of tools and techniques facilitating workload distribution in software development team,
- use of tools and techniques facilitating automated software testing.

II. Laboratory report

Complete laboratory tasks as per the instructor's presentation. Work alone or in a team of two. **Keep safety rules while working!** Prepare laboratory report documenting and proving the proper execution of tasks. Editorial requirements and a report template are available on the *eKursy* platform. The report is graded in two categories: tasks execution and editorial requirements. Tasks are graded as completed (1 point) or uncompleted (0 points). Compliance with the editorial requirements is graded as a percentage. The report should be sent as a *homework* to the *eKursy* platform by Thursday, May 6, 2021 by 23:59.

Prepare **single laboratory report** from instructions *L06* and *L07*.

III. Prepare to course

a) Know the safety rules

All information on the laboratory's safety instructions are provided in the laboratory and on Division website [1]. All inaccuracies and questions should be clarified with the instructor. It is required to be familiar with and apply to the regulations.

Attend the class prepared. Knowledge from all previous topics is mandatory.

b) Software patterns

Software development, including IoT systems, requires solving the same design problems over and over again, often with the use of different programming tools. In professional software engineering, such repetitive, generic problems are usually solved using an adequate software *pattern*. A software pattern is a form of representing developers knowledge and experience in the form of a description of a frequently recurring problem in software development and its solution that can be used many times over in other circumstances. In software engineering, we distinguish, i.a. *design patterns* and *architectural patterns*. Design patterns are used primarily in projects that use object-oriented programming and describe how to organize classes and the relationships between them [2]. Examples of design patterns are:

- **Singleton** - limits the number of class instances to one [3].
- **Adapter** - allows to use an interface of an existing class as another interface. It is often used to make existing classes work with others without modifying their source code [4].
- **Command** - encapsulates the information needed to perform a certain action or trigger an event at a later time. This information includes, e.g. a method name, an object that owns the method, and method parameter values [5].

Architectural patterns define the general structure of a given IT system, the elements it consists of, the scope of functions performed by a given element as well as the rules of communication between individual elements [6][7]. One of the basic problems of software architecture is the separation of the user interface and application logic according to the principles of *separation of concerns* [8] and *single responsibility principle* [9]. Examples of architectural patterns addressing this issue are:

- **Model-view-controller**, MVC - in this pattern, the application is broken down into a model (representing data structures), a view (i.e. a user interface), and a controller processing the user input data. A distinctive feature of this pattern is that all three elements can communicate with each other and that one controller can control multiple views [10].
- **Model-view-presenter**, MVP - in this pattern, the application is divided into a model (representing data structures), a view (i.e. a user interface) and a presenter formatting model data for view and the view input data for model. A characteristic feature of this pattern is that the view is completely separate from the model and that each view can only have a single presenter [11].
- **Model-view-view model**, MVVM - in this pattern, the application is divided into a model (representing data structures), a view (i.e. a user interface) and a view model adjusting the model data for the view and the view data for the model. A characteristic feature of this pattern is that the view is fully separated from the model and that one view model can control multiple views [12] [13].

The MVVM pattern is currently widely used in mobile application development. There are sets of programming tools that facilitate the implementation of mobile app in accordance to the assumed pattern, i.e. Lifecycle library [14]. A practical issue closely related to the implementation of software patterns is *data binding*. Data binding is the collective name of the techniques that ensure the relationship between

a data source (e.g. a GUI slider) and a data recipient (e.g. an integer representing the slider's state), and the data synchronization mechanism between them [15][16]. To use the data binding mechanism in the Android Studio project, the appropriate configuration should be added to the `build.gradle` script (see: listing 1.). Listing 2. shows examples of binding XML GUI elements to Java class fields.

Listing 1. Gradle script configuration: enable data binding.

```
android {
  ...
  dataBinding {
    enabled=true
  }
  ...
}
```

Listing 2. Data binding examples in Android Studio

```
...
<data>
  <variable name="vm" type="iot.examples.myapp.viewmodel.MainViewModel" />
</data>

<LinearLayout>

  <androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rv_measurements"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    app:adapter="@{vm.adapter}"/>                    <!--One-way adapter binding-->

  <EditText
    android:id="@+id/url"
    android:text="@={vm.url}"/>                      <!--Two-way string binding-->

  <Button
    android:id="@+id/update"
    android:text="UPDATE"
    android:onClick="@{vm::updateMeasurements}"/> <!--One-way method binding-->

</LinearLayout>
...
```

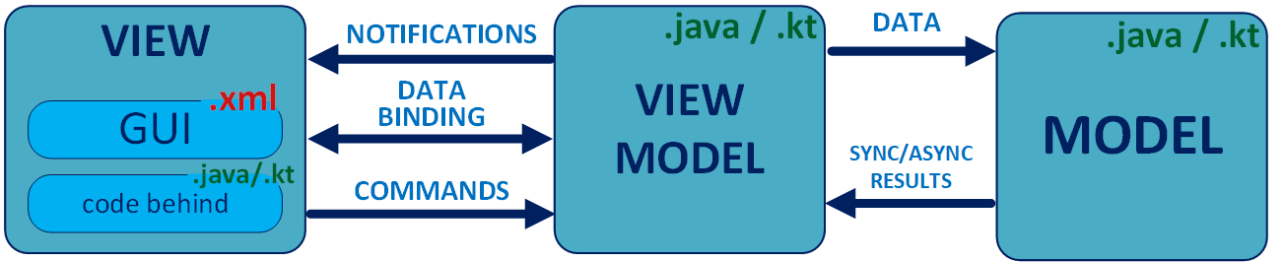

Fig. 1. Relations between view, model and view model in MVVM architectural pattern.

c) Data visualization

One of the basic tasks of IoT system client apps is a legible presentation of measurement data to the end user. Graphical user interface allows data visualization with diffrent types of *charts*. Examples of presented mobile apps use free chart library GraphView [17] . This library offers chart types for different purposes: presentation of measurement history data e.g. with *line* charts as well as current measurement

values e.g. with *bar* charts. Fig. 2. shows an example of chart with signal time series created with GraphView library. Other data visualization tools are i.a. AnyChart-Android [18] or MPAndroidChart [19].

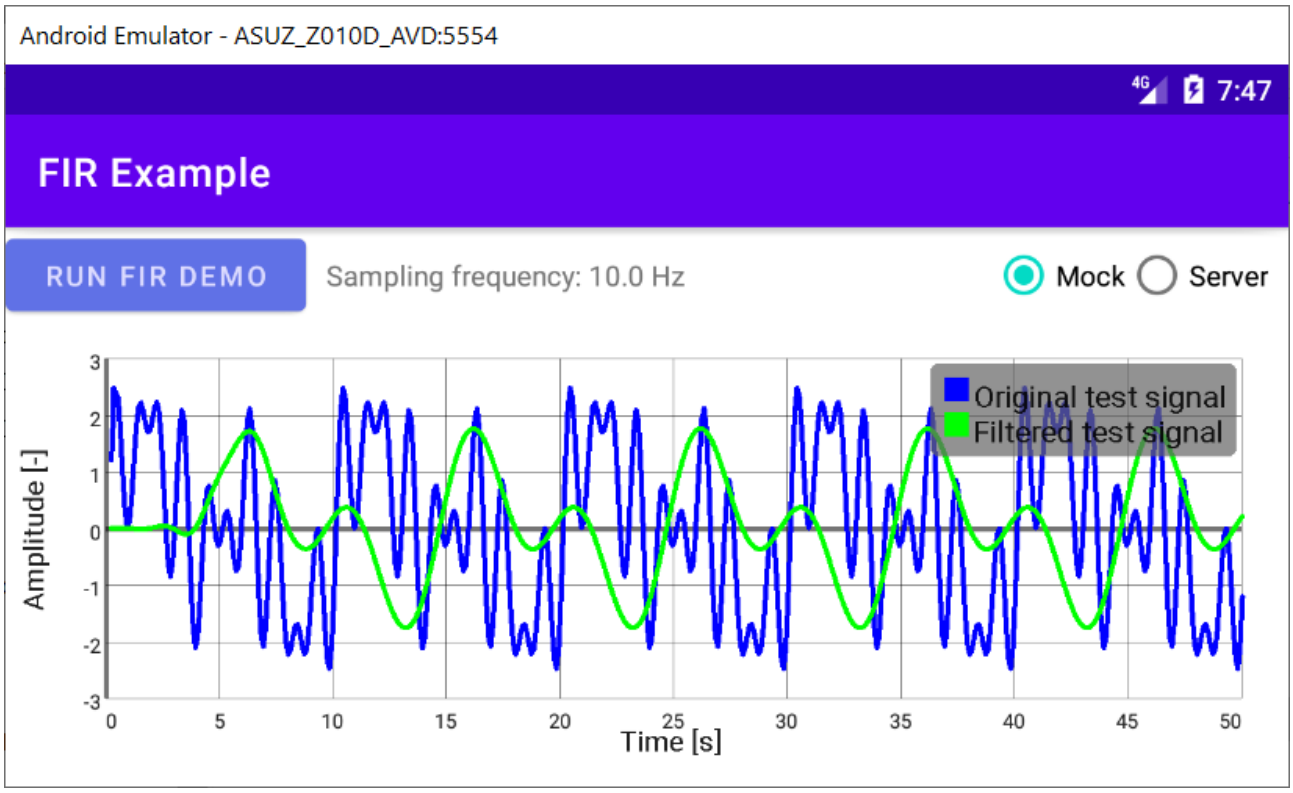

*Fig. 2. GraphView chart example: measurement signal time series (**FIR Example** app).*

D) Examples

Sample Android Studio projects are available in repository github.com/adrianwojcikpp/AMiWdIP-L. Before performing laboratory tasks, see examples all available examples. All applications are examples of simple REST *clients* for an embedded system web *server*. The *MVVM RecyclerView Example* application is an example of a client *asynchronously* retrieving measurement data from the server using the GET method. The application is based on the Model-view-viewmodel pattern and uses data binding. It is also an example of *dynamic user interface* - the list of measurements depends on the server's response, not the content of the client application. The *FIR Example* application is an example of a client *synchronously* requesting measurement data from the server and performing their digital filtering. This is an example of using the GraphView library to handle charts. This is also an example of using the Volley `RequestFuture` class to obtain a synchronous server response.

IV. Scenario for the class

A) Teaching resources

Hardware
- computer,
- mobile device (e.g. Android phone),

Software
- mobile apps IDE (e.g. Android Studio IDE).
- mobile device emulator (e.g. Android Virtual Device),
- web server with CGI scripts support (e.g. XAMPP),
- code editor (e.g. Notepad++, VS Code),

B) Tasks

Familiarize yourself with examples of mobile applications available in GitHub repository. Run and test applications with the use of sample PHP / Python scripts.

1. Create REST client applications for a mobile device (smartphone, tablet) for embedded system measurement data visualization.
   (a) The interface should contain timeseries plots of *at least two* measurement quantities, e.g. angular orientation (RPY) or temperature, pressure and humidity from the Sense Hat add-on. Put on the chart all the necessary information for the correct and unambiguous interpretation of the data. Use previously prepared graphic interface prototype designs.
   (b) The application should *cyclically* request data from server and display responses in the form of timeseries plot. The sample time should be defined by the user.
   (c) Run and test app on a physical or virtual mobile device. Make sure the network communication is working properly. You can use a physical or virtual embedded device for testing, or a server *mock* in the form of a local CGI script generating random synthetic measurement data.
2. Expand client application with activity to control user output device (e.g. LED display).
   (a) The interface should allow setting the state of all available user outputs (e.g. the LED matrix from the Sense Hat add-on). The interface should clearly communicate whether the current user settings correspond to the output states (i.e. whether the user has applied the last changes). Use previously prepared graphic interface prototype designs.
   (b) The application should send a request to the server containing control commands. Use an adequate HTTP method.
   (c) Run and test app on a physical or virtual mobile device. Make sure the network communication is working properly. You can use a physical or virtual embedded device for testing, or a server *mock* in the form of a local CGI script saving control commands to text file.
3. Expand client application with activity presenting measurement data and user inputs in the form of a dynamic list or table.
   (a) The interface should contain a *dynamic* list or table with the available measurements (e.g. angular orientation (RPY), temperature, pressure and humidity from the Sense Hat add-on) and the status of the user inputs (e.g. the three counters (XYC) of the joystick from the Sense Hat add-on). Each element of the list / table should contain information about the name of the measurement or input and its current state with the unit. The list / table may contain additional information about the measurement or input (ID, range, timestamp, etc.). Remember that, for example, the temperature on the Sense Hat board can be measured by two different sensors. Use previously prepared graphic interface prototype designs.
   (b) The application should *cyclically* request data from server and display responses in the form of dynamic list / table. Application doesn't have to use data binding.
   (c) Run and test app on a physical or virtual mobile device. Make sure the network communication is working properly. You can use a physical or virtual embedded device for testing, or a server *mock* in the form of a local CGI script generating random synthetic measurement data.

References

1. *Regulations, health and safety instructions* [online]. [N.d.] [visited on 2019-09-30]. Available from: http://zsep.cie.put.poznan.pl/materialy-dydaktyczne/MD/Regulations-health-and-safety-instructions/.

2. Software design pattern. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Software_design_pattern&oldid=1007470468. Page Version ID: 1007470468.
3. Singleton pattern. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Singleton_pattern&oldid=1015974289. Page Version ID: 1015974289.
4. Adapter pattern. In: *Wikipedia* [online]. 2020 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Adapter_pattern&oldid=994552282. Page Version ID: 994552282.
5. Command pattern. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Command_pattern&oldid=1015488026. Page Version ID: 1015488026.
6. Architectural pattern. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Architectural_pattern&oldid=1017956139. Page Version ID: 1017956139.
7. *Guide to app architecture* [Android Developers] [online] [visited on 2021-04-18]. Available from: https://developer.android.com/jetpack/guide.
8. Separation of concerns. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=1001182339. Page Version ID: 1001182339.
9. Single-responsibility principle. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Single-responsibility_principle&oldid=1016959444. Page Version ID: 1016959444.
10. Model–view–controller. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1016730751. Page Version ID: 1016730751.
11. Model–view–presenter. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93presenter&oldid=1004303342. Page Version ID: 1004303342.
12. Model–view–viewmodel. In: *Wikipedia* [online]. 2021 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93viewmodel&oldid=1012221294. Page Version ID: 1012221294.
13. *ViewModel Overview* [Android Developers] [online] [visited on 2021-04-18]. Available from: https://developer.android.com/topic/libraries/architecture/viewmodel.
14. *Lifecycle | Android Developers* [online] [visited on 2021-04-18]. Available from: https://developer.android.com/jetpack/androidx/releases/lifecycle.
15. Data binding. In: *Wikipedia* [online]. 2020 [visited on 2021-04-18]. Available from: https://en.wikipedia.org/w/index.php?title=Data_binding&oldid=994734036. Page Version ID: 994734036.
16. *Data Binding Library | Android Developers* [online] [visited on 2021-04-18]. Available from: https://developer.android.com/topic/libraries/data-binding.
17. GEHRING, Jonas. *jjoe64/GraphView* [online]. 2021 [visited on 2021-04-18]. Available from: https://github.com/jjoe64/GraphView. original-date: 2011-07-04T13:25:41Z.
18. *AnyChart/AnyChart-Android* [online]. AnyChart, 2021 [visited on 2021-04-18]. Available from: https://github.com/AnyChart/AnyChart-Android. original-date: 2017-08-09T06:48:26Z.

19. JAHODA, Philipp. *PhilJay/MPAndroidChart* [online]. 2021 [visited on 2021-04-18]. Available from: https://github.com/PhilJay/MPAndroidChart. original-date: 2014-04-25T14:29:47Z.