Victor Porev

PATTERN OF OWNEDRAW GUI FOR MULTI-MODE SOFTWARE APPLICATIONS

The article considers some aspects of building graphical interfaces with non-standard elements for multi-mode software applications. The onDraw-onTouch pattern is proposed and analyzed.

Key words: application, GUI, multi-mode, pattern.

Fig.: 5. Bibl.: 6.

Relevance of the research topic. Design patterns play an important role in software engineering by providing sample solutions for a particular class of software applications.

Formulation of the problem. A convenient and adequate graphical user interface (GUI) is a necessary component for a wide list of different software applications. This is especially true for multimode applications, such as those that provide extensive functionality for entering, editing, and displaying information. Multimode of such applications can be thought of as the ability to transition from one state to another, with context-sensitive graphical controls for each state. Application development environments typically provide programmers with some set of standard controls. Such standard elements are supported by corresponding API classes and functions. But if a developer wants to diversify the user interface by adding his own custom controls, he has to write a lot of code himself. This can be simplified by describing typical structures in the form of an architectural pattern. The pattern largely unifies the solution, which makes it easier to build assets and ensure their reuse.

Analysis of recent research and publications. The first significant advance in the classification of patterns is the Design Patterns book [1]. Since then, more than 2 decades have passed, but the relevance of patterns does not decrease, as they make it easier for programmers to implement effective architectural solutions that have already been developed.

An important step in the development of the science of patterns was the work of Martin Fowler, in particular, the description of the dependency injection pattern [2]. Dependency injection makes it easier to design software systems with extensibility, in particular, with a developed graphical user interface.

In general, an event-driven approach is mainly used for GUI implementation. This approach was described as an Observer pattern in the Design Patterns book.

Subsequently, the technology of interaction between system elements, such as message exchange or event processing, was designated as callback. For example, as in the Windows message system [3]. In some systems, the term ‘listener’ is used to denote such mechanisms [4].

Some examples of implementation of non-standard GUI elements in the form of software libraries are known [5, 6].

Uninvestigated parts of general matters defining. In the opinion of the author, there is a need for some generalization of the approach to the design of non-standard graphical user interfaces. Such a generalization should be made in the form of appropriate pattern design.

The research objective. The purpose of the research is to find some unified template - a pattern for describing the construction of event-driven architecture, focused on the active use of graphic and sensor capabilities of modern systems.

Presentation of the main material. Proposed pattern structure. The figure below shows the simplified class diagram of the onDraw-onTouch pattern.

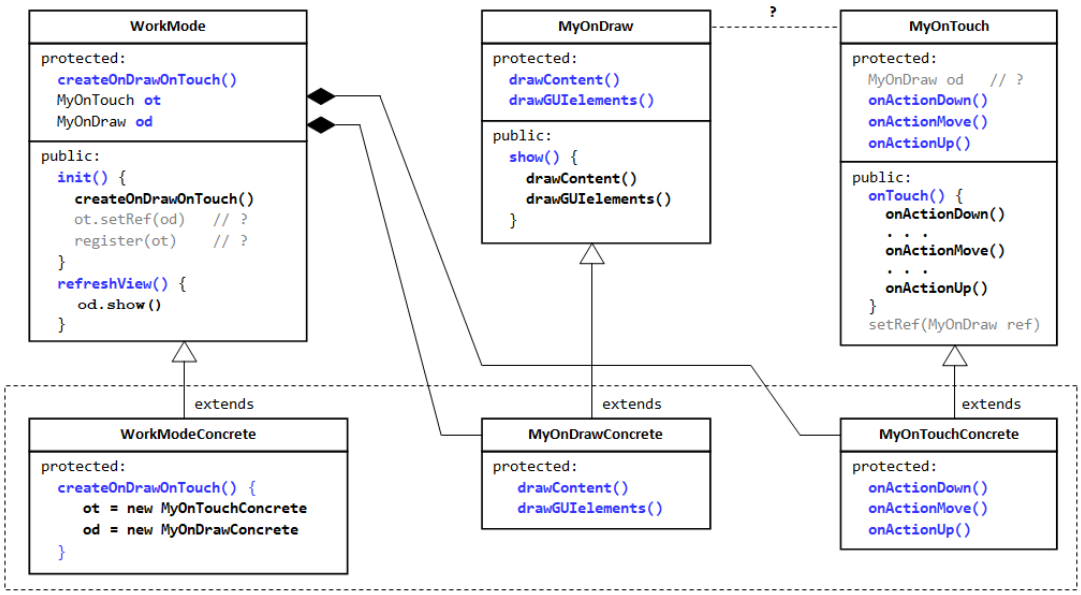

Fig.1. Pattern onDraw–onTouch

Classes of the WorkMode hierarchy describe the states (modes of operation) of the program. The base class may or may not be abstract - it may have default member definitions, such as for the start state of the program. The WorkModeConcrete classes describe the states in each specific work mode.

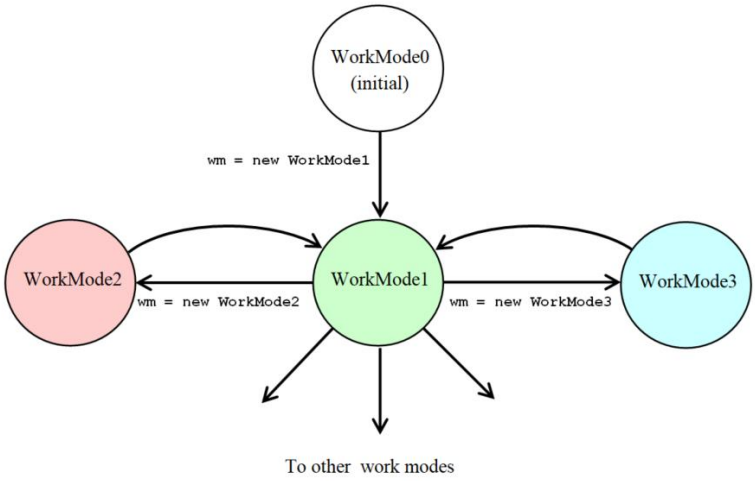

Fig. 2. State transitions in a multi-mode application such as an editor

Thus, it can be imagined that in order to switch to some mode, for example, WorkMode2, it is necessary to create an appropriate state object

```
WorkMode wm = new WorkMode0     //start from some initial state
. . .
wm = new WorkMode2
wm.init()
wm.refresh()
```

During the initialization of an object of the WorkModeConcrete class (in this example, it is WorkMode2), objects of the MyOnDrawConcrete and My On Touch Concrete classes are created. It may be appropriate in some cases to use a constructor instead of the init() method. This is at the discretion of the programmer.

Thus, each object of the WorkMode class creates and encapsulates objects of the MyOnDraw and MyOnTouch classes. In this pattern, it is implicitly assumed that access to objects of the MyOnDraw and MyOnTouch classes from the outside is closed, although this is not necessary. Depending on the implementation platform, you can provide for registering a touch event listener, for example, through the API of the corresponding sensor. This is partially shown in the class diagram in the body of the init() method.

One of the interface methods of the WorkMode class is the refreshView() method, which calls the show() method of the MyOnDraw class. The show() method

displays two things: some background content plus images of active custom GUI elements. As a general rule, active GUI elements should be in the foreground, so the call to drawGUI elements() in the body of the show() method is written last.

Based on platform considerations, it is possible for an implementation to provide a dependency of the MyOnTouch class on the MyOnDraw class. To do this, you can pass a reference (pointer) to an object of the MyOnDraw class to an object of the MyOnTouch class by calling the setRef(MyOnDraw) method. When might it be needed? Imagine that each time you move the cursor, you need to redraw the image in the window. Then, in the body of the onActionMove() method of the MyOnTouch class, you should provide a call to the drawing method of an object of the MyOnDraw class for example od.show().

Thus, for each specific state (mode of operation), the view of the application window is described by the program code for implementing the drawContent(), drawGUIelements() methods, and the logic for handling touch events is described in the onActionDown(), onActionMove(), onActionUp() methods. This is the main essence of this pattern. The list of touch event methods can be extended, for example, to implement multitouch.

When implementing this pattern, it is necessary to provide for the consistency of the display and touch coordinates of all active elements. To do this, you can provide appropriate members in the MyOnDraw class, which would also be visible in the MyOnTouch class.

Pattern onDraw-onTouch can be used to build ownerdraw GUIs for a variety of applications for many operating systems and platforms.

Features of the implementation of the onDraw-onTouch pattern on the Microsoft Windows platform. It is possible to choose several approaches for implementation. The figure below shows an implementation of the onDraw-onTouch pattern based on the Windows API.

You need to program a window callback function that calls the Windows message handlers. In the WorkMode base class, you can define such methods as message handlers:

onPaint() - WM_PAINT handler

onLBdown() - WM_LBUTTONDOWN handler

onMouseMove() - WM_MOUSEMOVE handler

onLBup() - WM_LBUTTONUP handler

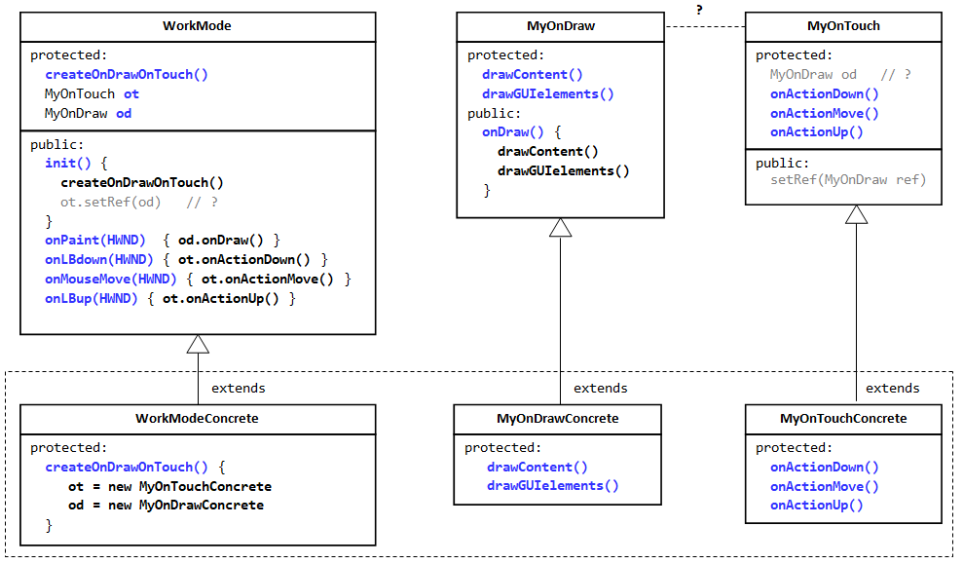

Fig.3. Pattern onDraw–onTouch implementation for Windows API

The refreshView() method from the WorkMode class can be omitted, since the InvalidateRect() Windows API function can be called directly instead.

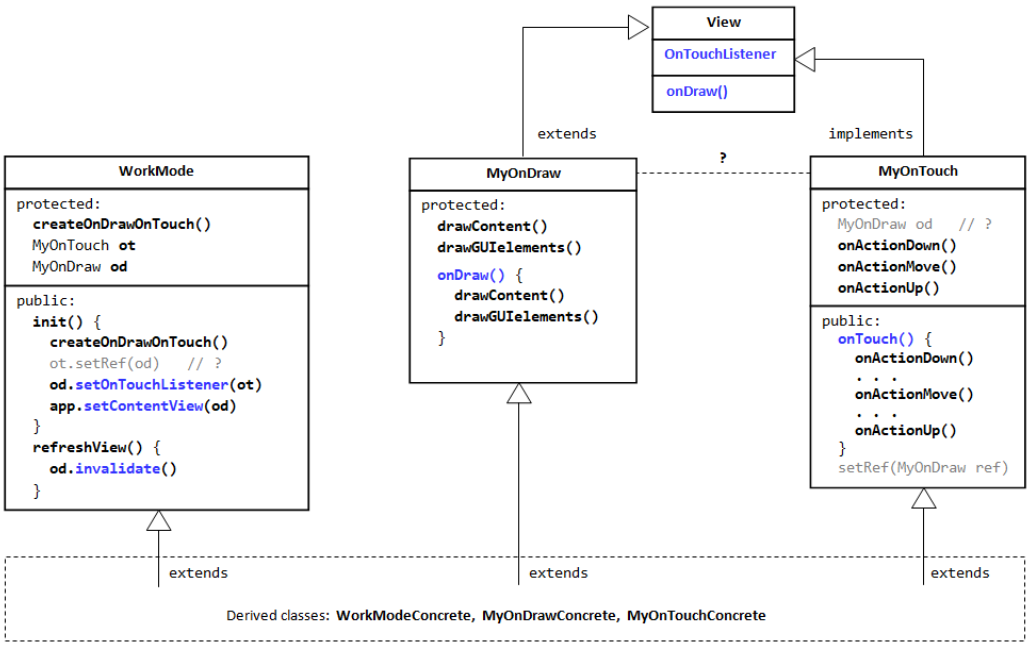

Fig.4. Pattern onDraw–onTouch implementation for Android API

Next, let's look at the features of the implementation of the onDraw-onTouch pattern on the Android platform. To implement this, it is convenient to use the View class from the Java and Kotlin Android API classes. In order to organize the display of graphics, it is enough to override the onDraw() method of the View class in a derived

class. In our case, in the MyOnDraw class. And to access messages from the touch sensor, the View class provides the onTouchListener interface with the onTouch() method, which should be implemented in the user class, for example, MyOnTouch.

To redraw the contents of the window, you can call the invalidate() method of the View class, which leads to a subsequent call to the onDraw() method of the MyOnDraw class. To register the callback methods of the drawing and touch classes, the setContentView() and setOnTouchListener() methods are used.

Below in Fig. 5 illustrates an example of the implementation of this pattern in the Android application MyGIS created and developed by the author of this article.

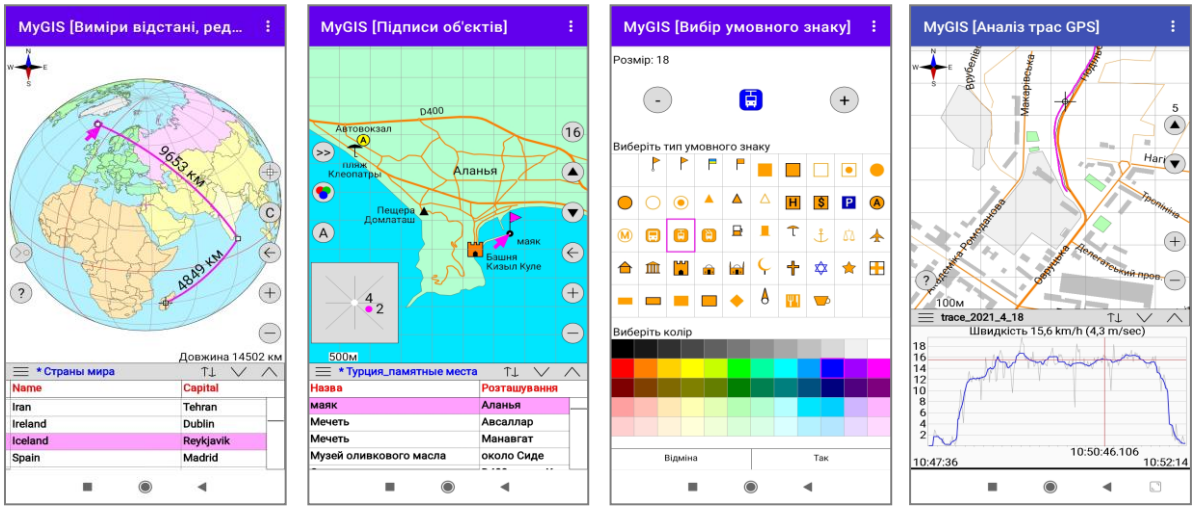

Fig. 5. Examples of custom ownerdraw GUI elements based on the onDraw-onTouch pattern in the MyGIS Android application

The main idea of the GUI is being implemented: What we see is what we can touch.

Conclusions. The possibilities of generalizing the description of building user interfaces for multimode applications based on the proposed onDraw-onTouch pattern are considered. Using this pattern can reduce development costs in object-oriented style applications with non-standard GUI elements.

References

1. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994, 395 p.

2. Martin Fowler. Inversion of Control Containers and the Dependency Injection pattern - Forms of Dependency Injection. 23 January 2004.

URL: https://martinfowler.com/articles/injection.html

3. Microsoft. Windows Dev Center. URL: https://developer.microsoft.com/en-us/windows/

4. Google. Documentation for app developers.

URL: https://developer.android.com/docs

5. Fully Skinned UI in wxWidgets (Trying to Emulate OwnerDraw) // wxWidgets Discussion Forum. URL: https://forums.wxwidgets.org/viewtopic.php?t=42924

6. SEGGER. LISTVIEW - Custom (Sample)

URL: https://wiki.segger.com/LISTVIEW_-_Custom_(Sample)

AUTHORS

Victor Porev – associate professor, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: v_porev@ukr.net

EXTENDED SUMMARY

Victor Porev

PATTERN ONDRAW–ONTOUCH AND ITS USABILITYFOR OWNERDRAW GUI

Relevance of research topic. Design patterns play an important role in software engineering by providing decision patterns for a particular class of software applications.

Formulation of the problem. A developed, convenient and adequate graphical user interface is a necessary component for a large number of different software applications. It is possible to simplify the creation of a non-standard GUI for multimode applications by describing the typical constructions in the form of an architectural pattern, which largely unifies the solution, which makes it easier to build assets and ensure their reuse.

Analysis of recent research and publications. The first significant advance in the classification of patterns is the Design Patterns book [1]. Since then, more than 2 decades have passed, but the relevance of patterns does not decrease, as they make it easier for programmers to implement effective architectural solutions that have already been developed.

Uninvestigated parts of general matters defining. There is a need for some generalization of the approach to the design of non-standard graphical user interfaces. Such a generalization should be made in the form of appropriate pattern design.

The research objective. The purpose of the research is to find some unified template - a pattern for describing the construction of event-driven architecture, focused on the active use of graphic and sensor capabilities of modern systems.

Presentation of the main material. Programs can work out different modes of operation (states). Each mode is described by a class that is derived from some base class. The onDraw-onTouch pattern is proposed, which describes the relationship between the main display classes of GUI elements and event handlers related to user interaction with these elements. Examples of implementation and use of such a pattern are considered.

Conclusions. Possibilities of construction of the graphic user interface on the basis of the offered pattern onDraw-onTouch are considered. This pattern describes a simplified generalized approach to GUI implementation. Instead of combining different API controls, each of which usually has significant features of implementation in the program code, the pattern allows you to unify the construction of program code for the GUI in an object-oriented style. This can reduce the cost of developing programs.

Key words: GUI, pattern.