Theoretical and Applied Informatics
ISSN 1896-5334
Vol. 26 (2014), no. 1, 2
pp. 87 - 104

Designing Human-Computer Interaction for Mobile Devices with the FMX Application Platform

ZDZISŁAW SROCZYŃSKI [a]

[a]Institute of Mathematics, Silesian University of Technology

Abstract: The article contains a survey of methods of the mobile development with the FMX Application platform, which allows to build multi-platform solutions for the efficient Human-Computer Interaction.

The test projects illustrate some important conclusions regarding the design and testing of mobile applications for iOS and Android operating systems. They demonstrate navigation controls designed in the skeumorphic and flat manner with some comparisons based on interviews with users in several age ranges. Test apps provide also touch gestures, sensor integration and connections to remote internet data sources. Some additional considerations about the localization and internationalization of the mobile applications built with the FMX platform were also presented.

The proposed implementation of the software engineering methods for the mobile application development provides new insights, valuable for software developers dealing with the new FMX platform on iOS and Android.

Keywords: HCI, mobile devices, multi-platform applications

1. Mobile HCI in modern software projects

The development of modern software is very often dedicated specifically for smartphones, tablets and the other mobile devices. The advantages and disadvantages of the mobile hardware and mobile operating systems should be taken into account for such projects. Mobile devices, in opposite to desktop computers, often have less computational possibilities, smaller screens, no keyboard, but they are equipped with touch recognizers, gyroscopes, accelerometers, GPS sensors and compass. Although their multimedia efficiency is lower than desktop ones, the obligatory equipment is an integrated hi-resolution digital camera. Moreover, the mobile devices often use 3G or 4G services, which have a lower response time and often a lower bandwidth then cable internet connections [11]. These factors have the strong negative influence on the user experience while accessing mobile web pages and using mobile applications (apps).

Clearly, that is possible to get a proper project of the Human-Computer Interaction (HCI) adapted for efficient mobile solutions, only taking into account factors mentioned above. First, the mobile application should make use from touch gestures not limited to simple movement in four directions, but also special ones, as pinch-in and pinch-out for zooming or shaking. Even simple gestures, as tapping, which corresponds with mouse clicking, need customized large controls, easy to point with fingertips. Placing relatively big-sized controls on a small screen (about 4 to 5 inches in diagonal for an average smartphone) results in a division of the communication area into separated units. These units, called ”views” in Apple’s iOS, represent somewhat equivalents of windows in desktop programming. The main difference is that they always occupy the whole screen and cannot partially overlap each other.

For example, the set of detailed formal regulations for the user interface used by applications for iOS operating system is enclosed in document titled ”iOS Human Interface Guidelines” [2]. The corporate standard establishes certain restrictions for developers this way. These include methods of controls arrangement, the visual appearance up to detailed design of icons and behaviour of standard and custom elements of the user interface. Human-computer interaction unified this way ensures similar graphical interface in different applications, making them easier to learn for the user.

Touch screens, leading HCI towards natural interfaces, are suitable for tasks of pointing, but often fail during accurate manipulation of objects, for example CAD works or advanced and fast text processing [12]. Moreover, they cause troubles for elder people with disabilities and are almost useless for visually impaired.

Building human-computer interaction for mobiles requires also the use of the other extra functionalities of the device, as calling, sandboxed filesystem and sharing sheets, contacts database, digital maps interface. Users are familiar with these standard interfaces and desire to see it unchanged in whole infrastructure of business applications.

The most of mobile operating systems has rather limited possibilities when talking about multitasking. The first reason for that is reduced processor’s efficiency, the second – the obvious problem to represent many views simultaneously on the same small screen and to navigate between them. Some novice users of iPhone are certainly even not aware of the possibility to browse through the list of executed applications. These apps are not ”running” in the strict meaning, because iOS forces suspension of the processes in background, freezing them so the user has only the impression of multitasking, while the efficiency of the mobile device is not degrading.

Continuing with this example, lets examine the application lifetime cycle for iOS application, which can be in one of the following states:

– not running – the application has not been launched yet,

– inactive – the application is running in the background but not receiving events (a temporary state),

– active – the application is running in the foreground and is receiving events,
– background – a temporary state before being suspended,
– suspended – the application is frozen in its current state and does not execute nor receive events.

It is worth noting that in iOS version 3 and lower there were only two states: not running and active, as it had no multitasking features at all. The application had been terminated in these versions, instead of being suspended. Moreover, in case of low-memory conditions, the iOS operating system automatically sends a low memory warning, and after that purges suspended applications without notice to make more memory available for the foreground application. Although, there are opportunities to interact with operating system or remote data sources which may result in semi-simultaneous acting of more then one app. It can be notifications through notification center and network push notifications. But all these methods require appropriate extra iOS API calls, as runtime machine code of the suspended application is frozen.

The part of above considerations concerns only iOS, but in general many problems with mobile application design are common for the most popular operating systems, as Android, Windows Phone, iOS or BlackBerry OS. And one of them is the market segmentation itself, in with four or five major players and many others (for example Tizen, Firefox OS, Sailfish OS or Ubuntu Touch OS) are attempting to build their own, incompatible development infrastructures. This way every mobile operating system is completely different, requires different programming languages and skills, from Java, through C++ and C#, Javascript, up to Objective-C and Swift. What is more, there are also miscellaneous design recommendations for the user interface and concepts for the navigation in these systems.

2. Foundations of the FMX platform

Some kind of universal framework could become the solution for problems described above. This framework should be built from common base of source code, and moreover – common design of user interface expandable with skins appropriate for a particular operating system and mobile device.

There are many approaches trying to meet such requirements, mostly based on HTML5 and Javascript with some kind of proxy library connecting the application with the hardware. These solutions are often called hybrid mobile applications. But the experiments shown that native code can be about five times faster than interpreted Javascript code used in mobile web applications and embedded web applications (for example built with Phonegap). What's more, the ARC (automatic reference counting) architecture used by native code LLVM compiler for iOS increases performance of memory

management in applications. The web apps using garbage collector technology suffer also from non-deterministic pauses caused by necessary heap scans [5].

So, to gain maximum performance and fully integrate with environment of the mobile operating system, the best solution is a native compiler, using ARM op-codes and directly translating the communication between the framework and the API of the OS. Thus obtained application can be fast enough to behave as fully native UI ones, and easily portable to different platforms the same time.

The FMX Application Platform (called FireMonkey or FM in previous editions) developed by Embarcedero meets these requirements. It is designed to develop applications for desktop Windows and OS X workstations, Apple's mobile operating system: iOS (in iPhones, iPads, iPods Touch) as well as current (in Rad Studio XE5-XE7 releases) and future versions of Android [6]. The usage of FMX Platform simplifies the multi-platform development at least at the level of the other software tools, as Silverlight, HTML5 or Flash [3], while the range of target operating systems supported by FMX is much wider. Although there are some dedicated frameworks with similar level of portability, as for example Unity 3D (specialized for game development), FMX appears to be the most sophisticated general usage tool for multi-platform development [1].

The user experience in rich GUI client applications built with FMX platform uses efficient 2D and 3D vector graphics, allows rapid prototyping with visual designer, animation effects and transformations of bitmapped graphics. The apps take full advantage of capabilities of the mobile OS and hardware, supported by the native cross-compiler for the ARM architecture.

The FMX platform is a fully object-oriented library designed for multimedia and visual effects in rich client applications. Thanks to that, controls can be embedded in the other ones. The styles engine of the platform provides an easy way to fit the look of the application into the framework of the particular operating system. It includes mobile iOS, as well as Android "skin". Furthermore, FMX assures the universal API for cooperation with many services of the operating system, as sensors, network connection, camera interface, as well as a software bridges giving the way to direct connection to platform-specific services with all their details [13]. MKMapKit from iOS can be a good example of such service.

The programming IDE used for FMX development, called RAD Studio, runs on Windows machines. Despite of a need to deploy using a computer running Mac OS X (Apple's Xcode code-signing tools and iOS simulator require OS X), that is a major simplification of the whole development process for mobile targets, making it similar to the common desktop application design.

2.1. Customization of the HCI with graphics effects and animations

The graphics effects in the FMX Application Platform are objects and they can be applied to any of the other objects in the platform. This is possible in the visual way – making an effect object the child of the visual control, or programmatically. The effects can be combined by adding more then one to the same control. Some interactive (triggered) effects slow-down the iOS apps, so they are switched-off by default. This makes the significant difference between desktop machines and mobile ones [4].

In the IDE, animations can be applied to any object's property signed with a film strip in the object inspector. Triggers for animations include mouse, focus, visibility and drag events. They can be combined and reversed if necessary. AnimationType and InterpolationType can be defined for each animation, in addition to the duration. AnimationType determines how the specified property changes from the start to the end value. The path of the interpolation curve determines InterpolationType.

That is also possible to fire animations in code, in the synchronous and asynchronous way, using AnimateFloatWait and AnimateFloat methods. This is illustrated by the following short example:

```
Panel.AnimateFloat('Position.X', -50, 0.4, TAnimationType.atIn,
                                  TInterpolationType.itExponential);
```

Due to test-proven limitations in iOS implementation, it is recommended to use asynchronous version on that platform and FM version 1, controlling the sequence of events in the application by hand, with the use of separate timer objects.

Thanks to the modular object-oriented design of FMX platform the application of many visual effects, common in modern UX design, is straightforward and does not require special effort from the developer. This way the author can focus at the most important functionalities needed by the app and then make it more attractive in a few independent directives. There is a blur effect shown in the example educational game in the Fig. 1. The blur is one of the popular trends in the development of modern mobile applications [14], putting the contents in the first plane while the overall app's look is still available to the user and reminds about the current context.

In the mentioned example all the movement in the view of the game is frozen, then the view is blurred and overlapped by a settings pane. The object arranging the blur effect (i.e. BlurEffectGame) is set-up as a child of the main layout:

```
object LayoutGame: TLayout
  Align = Client
  Size.Width = 320.00
  Size.Height = 460.00
  Size.PlatformDefault = False
  object BlurEffectGame: TBlurEffect
```

```
      Softness = 1.20
      Enabled = False
    end
  end
```

So, to turn it on when the user touches menu icon in the upper-left corner of the layout, there is only a need to change the value of the Enabled field. This applies the blur effect to all the children of the LayoutGame object.

Fig. 1. Blur effect used in the user interface of the educational game developed with FMX library .

Moreover, the example with blur effect proves, that high-level development libraries as FMX can simplify the design of the complex visual human-computer interaction, especially when compared to Apple's classes UIBlurEffect and UIVisualEffectView from the newest iOS 8 SDK [9]. The latter require much more work from the programmer and have less capabilities. For the former versions of the iOS (i.e. 6 or 7) the developer needs to implement the blur effect himself, which costs even more effort.

2.2. Definition and triggering of FMX styles

The styles engine for the FMX platform allows to design fully customizable user interface. The application can use many styles embedded in its executable or loaded from external files on demand. Styles are defined with the use of a special description language coded in plain-text files. So, there is also a possibility to alter them by hand.

The style definition language describes every aspect of a visual appearance of the application, from simple borders and fills of the controls, up to dynamic actions, changing the look of the controls under certain conditions. These triggers allow to respond to the user's activity, as mouse clicking, tapping or the other gestures.

Fig. 2: Sample user interface for image gallery in FMX app (English version in the iOS application on the left, Polish version in MS Windows application in the middle, Polish version in iOS 7 styled application on the right).

The FMX platform allows the usage of separate styles on different platforms (while maintaining the same source code base). On the other hand, it is possible to apply the same style on different platforms (see Fig. 2). This creates the opportunity to evaluate the mobile application compiled for testing with target on the Windows desktop platform, or adjust style to the destination platform, or even dynamically according to user settings. The skinnable user interface helps to develop modern and agile HCI for business applications built with FMX platform. For example, introduction of iOS version 7 required just to prepare new style, compatible with new Apple's guidance for "the look and feel" of an iOS 7 app.

TStyleBook control can contain a complete set of style definitions and allow to change the style in the application dynamically in the OnCreate event of the form as follows (assuming that form's name is FormTestStyle and StyleBook's name is StyleBookAndroidL):

```
procedure TFormTestStyle.FormCreate(Sender: TObject);
begin
 {$IFDEF ANDROID}
  if TOSVersion.Major = 5 then
   FormTestStyle.StyleBook := StyleBookAndroidL;
 {$ENDIF}
end;
```

In the above example the conditional define checks for Android platform and then TOSVersion static class field Major is checked against the newest version of Android (i.e. ”L” or ”5”). For that version the special look-and-feel is loaded from the StyleBook control. Given this style in .fsf file, the developer is able to adapt it in very efficient manner, assuring the best possible interaction of the mobile app and the user.

3. Testing and profiling HCI in mobile applications

Mobile applications need careful design and thorough testing. Software engineering of the mobile application should take into consideration not only emulator, but as many as possible real devices, running different versions of the mobile operating system. For example making iPhone 4 app compatible with iOS 6 and iPhone 5 requires an important detail: appropriate sized splash-screen bitmaps.

There is no possibility to complete tests of many mobile sensors without the use of real devices, as gestures, accelerometer, gps are not available on simulator or are simulated in reduced forms. It includes the changes of device orientation and running on devices with different screen sizes (smartphones vs tablets). The performance of the simulator is also different then real machine’s one. The iOS simulator is sometimes faster then real iPhones, especially when it comes to graphics. On the other hand, Android emulator on desktop computer is often much less efficient than real devices running that operating system.

Hardware limits of mobile devices (as low memory warnings in iOS, network time-outs, timers freezing while applications are going into the background) are the hardest to test and simulate. Sometimes it takes hours or even days to detect particular issue, as it appears only after long operation of the app, which is intended to work continuously. Testers of common desktop business application are not familiar with such problems and these requirements of the mobile environment.

Significant problems were observed during long-term tests of the animations and visual effects in sample application developed with FM version 1. Short tests ended successfully, but leaving the phone turn-on for a longer time led to the situation in which the animations were broken or the whole app crashed. This especially happened when application was running at midnight, suggesting problems with the change of date. The look into the source code of GetTick method showed that a wrong precision was used:

```
function TPlatformCocoa.GetTick: single;
var
  H, M, S, MS: word;
begin
  DecodeTime(time, H, M, S, MS);
  Result := ((((H * 60 * 60) + (M * 60) + S) * 1000) + MS) / 1000;
end;
```

The solution was to replace single precision with double one and introduce factors for all date elements in the calculation, as follows:

```
function GetTick: double;
var
  Y, MO, D, H, M, S, MS: word;
  sY, sMO, sD, sH, sM, sS, sMS: double;
begin
  DecodeDateTime(now, Y, MO, D, H, M, S, MS);
  sY:=Y; sMo:=MO; sD:=D; sH:=H; sM:=M; sS:=S; sMS:=MS;
  Result :=
    (((((((sY*12+sMo)*31+sD)*24+sH)*60*60)+(sM*60)+sS)*1000)+sMS)
      /1000;
end;
```

This time every animation proceed smoothly, and the app could run without problems for many days.

The FMX platform allows calling native libraries of the underlying mobile operating system, for example MKMapKit in iOS, showing maps with pins pointing into locations. There is also a possibility to use native controls instead of styled, vector graphics-based ones. This way the developer can overcome some difficulties with time-consuming animations to get better user experience, for example replacing TAniIndicator with native ActivityIndicatorView from iOS.

The task of transferring the data over the internet connection also needs the real device for testing. The timeouts and behaviour of the mobile device while the cellular signal is low or connection breaks, create some important user experience issues. With the common ActivityIndicatorView the application designer keeps this experience on the standard, well-known level. The application should refresh the data in selected intervals, short enough to present current information and long enough to provide convenient reading, or on a direct demand of the user. The FMX platform provides the interface to manipulate XML files and store the data in sqlite databases. In the test application additional MD5 hash function checks if remote data has changed, so unnecessary updates of the screen contents are not performed. This saves data transfer (which can be crucial for average user) and makes overall user experience much better, as the app shows the information contents much faster.

3.1. Internationalization and localization of the software

The internationalization should assure a proper behaviour of the application despite of changing user locale settings. On the other hand, localization is a process of translation of all the messages in the application. This often raises further issues, especially with string lengths in different languages. The best way to localize the application is to use an automatic feature listing all the messages to translate and helping to manage them.

In the FMX platform such feature is provided by the TLang object. Some examples of the user interface of the multilingual application are shown in Fig. 3.

The proper operation of multilanguage support in Firemonkey requires a slight fix in source code of the library, as original GetCurrentLangID method sets always only the English locale. This should be replaced with call to native method from iOS Foundation framework: NSLocale.preferredLanguages. FM platform version 1 needs the following correction:

```
function TPlatformCocoa.GetCurrentLangID: WideString;
begin
 result:=UTF8Decode(
      NSLocale.preferredLanguages.objectAtIndex(0).UTF8String);
end;
```

Newer versions of FMX Application Platform suffer from similar bug and the solution is as follows (with the use of language extensions designed for ARM compiler):

```
function TPlatformCocoaTouch.GetCurrentLangID: string;
var
  lngs : NSArray;
  LanguageISO: NSString;
begin
  lngs := TNSLocale.OCClass.preferredLanguages;
  LanguageISO:= TNSString.Wrap(lngs.objectAtIndex(0));
  Result := UTF8ToString(LanguageISO.UTF8String);
  if Length(Result) > 2 then Delete(Result, 3, MaxInt);
end;
```

The next point worth mentioning is the unicode support in modern applications. The unicode allows to write down all the string messages, xml files and databases in different languages not bothering with code pages and coding/decoding procedures.

There is a need to carefully test the translation engine and all the messages. Although it can be done with the use of the iOS simulator, it is better to check on the real device. In fact, some UTF-8 encoded strings display well on the device, while the simulator shows question marks instead of non-ASCII special characters.

3.2. User experience in applications developed with FMX

The overall usability of the computer system has some attributes, identified slightly different in the literature. The best general reference here is [10], which distinguishes:

1. *efficiency* – how easy is to achieve the goal,
2. *satisfaction* – freedom from discomfort and positive attitude to the product,
3. *learnability* – the easiness to learn and start using the product,

Fig. 3: iOS MkMapView with custom bitmap logo in FireMonkey application A. Polish version of FM1 app, B. English version of FM1 app, C. Polish version, iOS6 (skeumorphic) style, D. Polish version, iOS7 (flat) style.

4. *memorability* – the easiness to return and operate the system after some break,

5. *faultlessness* – low error rate and the ability to recover.

The authors of [8] widen these attributes introducing *cognitive load* and *effectiveness*, trying to describe specific problems of the mobile development. On the other hand, there are many much more detailed, but uncategorised, factors described in [11], referring different aspects of the mobile design.

Assuming that satisfaction and learnability are key factors dependant on the visual aspect of the user experience, the survey of user opinions on FMX platform-developed apps was conducted. The test application was developed in three different versions, according to the development of FMX platform:

1. vector – iOS 6 like app (FM1),
2. pixel-perfect iOS 6 skeumorphic design app (FM for XE4),
3. pixel-perfect iOS 7 and 8 flat design app (FMX for XE5-XE7).

The first question was if the very exact UI design is important to common user, as FM1 application looked just similar to iOS 6 native ones, but not exactly the same. Many games provide some specific user interface without any criticism, while for business applications their users may require much more consistent design. The answer to this question determines the limits of learnability for the FMX app user.

Second problem, even more important and interesting, are user preferences for old-fashioned skeumorphic design or modern flat-looking one. The skeumorphic design (see Fig. 2 C), mimicking objects from real world, as buttons, sliders or tabs with shadows, gradients and borders forming 3D look, was the principal recommendation for the UI of Apple software for many years. On the opposite side there is a flat design, reinvented by Microsoft for Metro design in Windows 8 and Windows Phone operating systems. Flat design leaves only the pure information, basing on very simple graphic icons, a few colors, completely no shadows nor gradients, so overall user interface looks just ”flat”. Sometimes the resulting UI is simply black-and-white. Apple used this idea in iOS 7, being for the first time just a follower when talking about user experience (see Fig. 2 D). From this point of view the newest iOS 8 did not bring much new factors, as main novelties in this version of Apple’s system concern details and some new APIs. Anyway, the GUI of iOS stays almost the same in versions 7 and 8.

So the second question arises, if the new flat design is really better from the user point of view? Has the age or the knowledge of the user any connection with the rating? Thanks to FMX platform and the Rad Studio IDE which allow to change the theme for the application on-the-fly, that was possible to survey a number of respondents using exactly the same functionality of the applications having different appearance.

The results for the first question were rather surprising, as nobody noticed the differences between FMX app and native ones. The surveyed group was quite large, about 50 persons, but most of them were not every-day iOS users. Certainly, the iPhone or iPad users were able to distinguish native and FMX apps after a longer, close and thorough

examination. So, the conclusion is that for the user experience the general outline of the UI is more important than the exact fulfilment of the graphical project guidelines.

There were significant differences in perception of the application UI by the people of different age in the results for the second question. The youngest respondents (students aged around 20 years) were more open to the new paradigms, and their user experience with iOS 7 was full of satisfaction. Despite this, nearly half informed about mixed emotions for the details of the flat design: some pointed out several inconsistencies, as low visibility of clickable buttons, for example. Respondents about 30 were more sceptical about the new design in iOS 7, while people about 40 and elder were choosing definitely the skeumorphic iOS 6 version. The number of respondents divided into age groups was not very high for this question, but the trend seems to be evident (see Fig. 4). Moreover, the rate of acceptance for the flat design in iOS 7 was much higher in the group of active users of Apple's products. It follows that the loyal users are more likely to accept changes and news and they are less critical when it comes to minor issues.

It is of interest to know whether the results will be similar for the general case. The aim of this section was to sketch the problem of the user reaction for novelties in the design of mobile apps and define some significant factors. There is surely the need of further research and querying a greater number of respondents, which is of course a complex task from the logistical point of view.

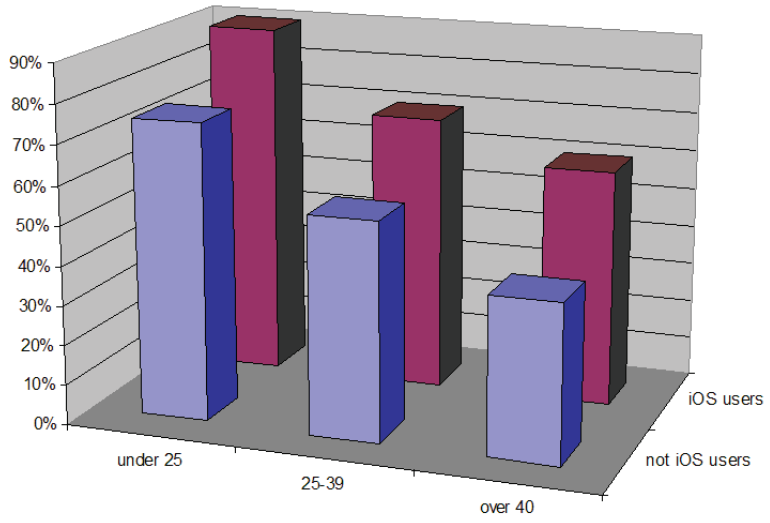

Fig. 4: Overall acceptance rate for "flat design" in iOS according to different age ranges of permanent and incidental/no iOS users.

Thanks to the development of the FMX platform and the possibility to compile for Android operating system from single code-base (in Rad Studio versions XE5-XE7), the survey can be extended by Android version. This gives the unique opportunity to compare UX on different platforms with the native look-and-feel without rewriting the whole application with separate design tools, programming languages and IDEs. This way the complete user-computer interaction built for the mobile application can be easily validated and evaluated.

3.3. Advantages of the newest versions of FMX

The FMX application platform introduced with the newest XE4-XE7 versions of RAD Studio allows much easier development of modern-looking applications. The new style engine and native components for pick data, communicate with sensors, the layout manager auto-adjusting form size and orientation, as well as non-visual improvements as xml and database support (Interbase ToGo and SQLite) give the new quality for the mobile development [7]. This helps in the fast design of an attractive user interface and creation of the efficient HCI in the application.

Firemonkey (FM) Application Platform, called just ”FMX” in the newest version, supports also Android based devices. Developers using FMX get the unique opportunity to develop applications for very different operating systems, covering the most of the market-share, including MS Windows, Apple desktop OS X, Apple mobile iOS and Google Android (see Fig. 5). The single code base in the FMX platform projects allows to focus on key problems in the application, leaving all details of needed interfaces to the platform and its style engine. This way there is possible the efficient development of rich human-computer interaction with much less effort from the programmer.

It is worth noting that the FMX developer can work efficiently in the agile, prototyping manner, providing a working version of the application to the user in a very short time. This can be achieved thanks to visual form designer, style engine and fast compiler, as well as the clear and easy to implement platform design. It is also important that the most of the functionality of the application can be tested just in the MS Windows development environment with the special ”Mobile Preview” functionality. ”Mobile Preview” invokes the windows compiler for the application developed for mobile targets (i.e. iOS or Android), when the developer chooses Windows platform as its target, but meanwhile it applies the semi-mobile visual style, which helps to debug the app. This way the app using the FMX platform can operate in Windows environment, connect to various internet sources, process the remote and local data, query databases etc. That is another time-saving factor, since the windows compiler is much faster then mobile/ARM one, and the deployment for Windows is almost instant, without the need of file transfer to the mobile device nor even obtaining provisioning profile which allows to use the iOS device in development.

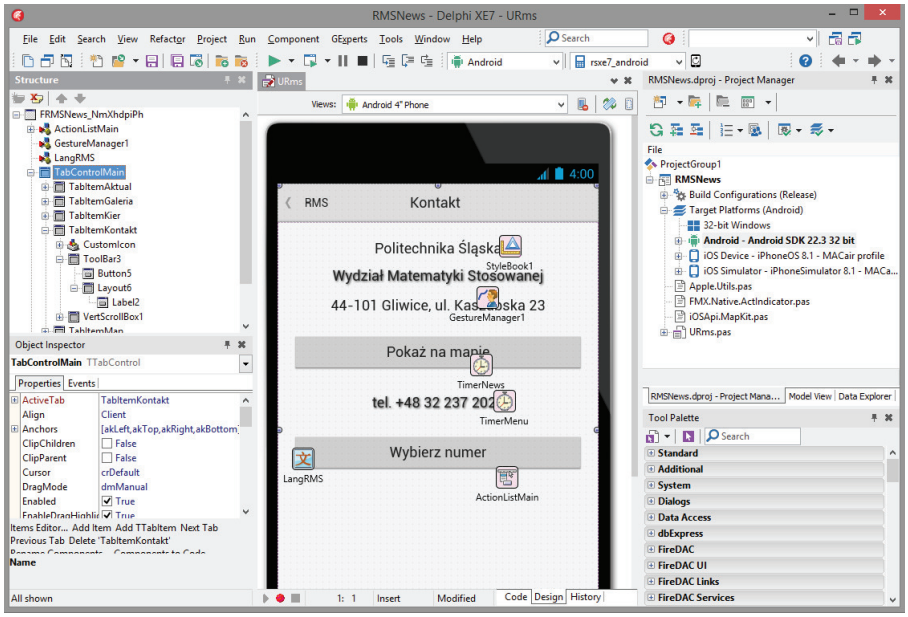

Fig. 5: Test app developed entirely for iOS in RAD Studio XE7 IDE targeting Android platform – the look-and-feel of Android UI was introduced automatically in the FireUI designer.

Regardless of all the details of the UI design, the maintenance of the single source code helps to ensure key factors of the proper multidevice project [11]:

– *visual continuity* – interactive elements should look similar at screens with different factors and sizes,

– *feature continuity* – the reduced functionality for smaller devices should remain consistent with overall design assumptions,

– *data continuity* – the same user's data should be used unchanged at every platform,

– *content continuity* – the content of the information should be composed in the same manner.

These goals can be easily achieved with the use of the FMX application platform with less error rate and better results for novel functionalities of the software, thanks to a separation of the model and the controller from UI design details (i.e. the view) at different development platforms.

The FireUI designer in the RAD Studio IDE simplifies the design process, allowing addictions and modifications of the form according to the current software platform. The one shared master form is subclassed for multiple platforms and device form factors giving the customized sub-view when necessary. This way the developer can maintain

one codebase without degrading native platform support. This happens for example while positioning navigation buttons (on the top in iOS, hardware one in Android) or tabs (usually on the bottom of the form in iOS, but on the top in Android). So, FireUI assures the best possible experience for the user, referring the rules of the particular, well-known mobile platform. Meanwhile, there are all the benefits and advantages of multidevice project preserved. This kind of GUI designer is unique for the RAD Studio IDE and for this reason it is remarkable.

4. Conclusions

There are examples of multi-platform mobile applications presented in the paper. The test applications demonstrate main controls and methods of human-computer interaction used for iOS mobile operating system, as navigation controls, touch gestures, sensor integration and connection with remote internet data source, as well as some graphics effects. These applications were successfully ported to Android without the need for extensive changes in source code at all.

Some important conclusions from testing the application in the mobile environment are also described. It was shown that the mobile project should be tested on a real device, including long term testing, taking into account life-cycle of the application in the mobile OS. This part of the article contains also some improvements to the source code of the first version [4] of the Firemonkey (FMX) platform. The test application was initially compiled with this version. The other improvements apply also to next versions of the FMX platform.

The following part of the article contains some additional considerations about localization and internationalization of the mobile application built with the FMX platform. The final conclusion states FMX as efficient tool to create well designed human-computer interaction for the mobile devices, using different operating systems.

The multidevice, multi-platform FMX development environment helped to conduct some experiments regarding the overall impact of UI design on users' satisfaction and effectiveness during the usage of mobile apps. The results indicated the differences in perception of the application UI dependant on age and former experiences of the user. Apparently the trendy flat design is not quite the best solution for everybody.

The presented implementation of software engineering methods for mobile application development provides new insights, valuable for software developers dealing with the new FMX platform on iOS and Android. It is also worth further investigations in more complex software projects and broadened surveys.

References

[1] Arsjentiev D.A., Pashkov P.S.: The environment for multi-device application development Delphi XE5, Bulletin of Moscow State University of Printing Arts (Russian edition), No 9'2013.

[2] Apple Inc.: iOS Human Interface Guidelines, https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html, 2015-01-18.

[3] Buitendag A., Roux L., Botha A., Herselman M.,Van Der Walt J.: m-Living Labs, A Framework for Collaborative Community Advancement, in: Cunningham P., Cunningham M.(Eds): ST-Africa 2012 Conference Proceedings, ISBN: 978-1-905824-34-2, IIMC International Information Management Corporation, 2012

[4] Chandler G.: FireMonkey development for iOS and OS X with Delphi XE2, From the WYN (What You Need) Series, Coogara Consulting 2012.

[5] Crawford D.: Why mobile web apps are slow, http://sealedabstract.com/rants/why-mobile-web-apps-are-slow/, 2015-01-18.

[6] Embarcadero, Inc.: RAD Studio Mobile Roadmap, http://edn.embarcadero.com/article/43677, March 2014.

[7] Embarcadero, Inc.: RAD Studio Mobile Tutorials Version XE7, November 2014.

[8] Harrison R., Flood D., Duce D.: Usability of mobile applications: literature review and rationale for a new usability model, Journal of Interaction Science May 2013, 1:1.

[9] Nahavandipoor V.: iOS 8 Swift Programming Cookbook, O'Reilly Media 2014.

[10] Nielsen, J.: Usability Engineering. Academic Press, Boston, ISBN 0-12-518405-0, 1993.

[11] Nielsen, J., Budiu, R.: Mobile Usability, New Riders Press, ISBN 0-321-88448-5, 2012, Polish edition: Funkcjonalność aplikacji mobilnych. Nowoczesne standardy UX i UI, Helion 2013.

[12] Sikorski M.: Interakcja człowiek-komputer, Wydawnictwo PJWSTK, Warszawa 2010.

[13] Teti D.: Delphi Cookbook, ISBN 139781783559589, Packt Publishing 2014.

[14] Treder M., Pachucki A., Zielonko A., Łukasiewicz K.: Mobile book of trends 2014, UX Pin & Movade internal report, http://www.uxpin.com/mobile-design-book-of-trends.html, 2015-01-18.

Projektowanie interakcji człowiek-komputer dla urządzeń mobilnych z wykorzystaniem platformy FMX

Streszczenie

Współczesne oprogramowanie coraz częściej tworzone jest specjalnie dla urządzeń mobilnych – smartfonów i tabletów. Projekty tego rodzaju powinny spełniać szereg wymogów związanych ze specyfiką obsługiwanego sprzętu i mobilnych systemów operacyjnych. Właściwe zaprojektowanie pośrednictwa użytkowego pozwala wówczas zbudować efektywne narzędzia interakcji człowiek-komputer (*Human-Computer Interaction* – HCI).

Istotnym problemem związanym z budowaniem aplikacji dla urządzeń mobilnych jest duże rozwarstwienie systemów operacyjnych oraz wspierających je platform programistycznych. Z tego względu celowe jest dążenie do opracowania rozwiązań uniwersalnych, pozwalających w jak największym stopniu zunifikować bazę kodu źródłowego. Założenia takie spełnia platforma FMX (Firemonkey) opracowana w firmie Embarcadero, pozwalająca tworzyć oprogramowanie zarówno dla systemów klasy desktop (Windows i OS X), jak i systemów mobilnych (iOS oraz Android).

Platforma FMX ułatwia projektowanie interakcji z użytkownikiem za pomocą pośrednictwa budowanego za pomocą wydajnej grafiki wektorowej 2D oraz 3D, uzupełnionej o efekty animacji i przekształceń grafiki bitmapowej. Za pomocą wbudowanych stylów można uzyskać wrażenie wykorzystania natywnych elementów interfejsu różnych systemów operacyjnych, w tym systemów mobilnych. Ponadto Firemonkey zapewnia wsparcie dla usług oferowanych przez mobilny system operacyjny, takich jak dostęp do aparatu cyfrowego, akcelerometru, kompasu czy GPS.

W artykule przedstawiono przykłady aplikacji mobilnych zawierających podstawowe elementy interakcji człowiek-komputer na urządzeniu mobilnym pracującym pod kontrolą systemu operacyjnego iOS. Ponadto omówione zostały ważne z punktu widzenia inżynierii oprogramowania aspekty związane ze specyfiką projektowania oraz testowania aplikacji mobilnych.