APPROACH ALLOWING DESIGN ADAPTIVE VIEW MODELS

Olena Chebanyuk, Vladyslav Pashystyi

Abstract: *The problem of setting interface settings for global software development processes is actual now. Successful solving of this problem can be source many benefits in view models, other words interface representation designing. Thus, in turn, allow to increase amount of web-site visitors and web applications users. Other advantage is to allow different developers to understand interface well and design application using n-tired architectural style.*

One of the important task is to archive dynamic localization for switching interface languages. Paper represents an approach of view models designing in context of dynamic localization for desktop application using WPF technology.

Paper contains the review of existing frameworks for flexible interface designing supporting localization process.

Approach touches some mathematical foundation of interface components representation, recommendation for framework designing, touching design patterns. Paper contains some practical examples of preparation interface using different languages.

Keywords: *n-tired architecture, view model, localization, graphic user interface, Screen, interface elements scaling.*

DOI: *https://doi.org/10.54521/ijita28-04-p03*

Introduction

Today, web applications represented in the market widespread. Despite their popularity, they cannot completely replace programs designed for a specific

operating system. The reason for this is that all features of such applications are used by the operating system itself providing more effective conditions for resources using (for example graphical memory). Frameworks for interface designing mainly the development of ready-made solutions, which are then used by other developers when writing their programs (Enhan, 2011). Examples are IFML that was adopted as a standard by the Object Management Group (OMG) in March 2013 (IFML, 2016). IFML supports the platform independent description of graphical user interfaces for applications accessed or deployed on such systems as desktop computers, laptop computers, PDAs, mobile phones, and tablets. The focus of the description is on the structure and behavior of the application as perceived by the end user (Marco Brambilla and Pietro Fraternali, 2015).

There are many researches reflecting achievements in model-based user interface approaches. They are Drive (Mitchell K. et. al, 1996), MOBI-D (Puerta A. and Eisenstein J., 1999), Wisdom (Nunes, N.J. at. el., 2005), Teresa (Mori G. et. al., 2004), Teallach (Griffiths, Tonya, Barclay and Peter J, Teallach, 2001), JUST-UI (Pedro J. Molina et. al., 2002), SUPPLE (Krzysztof Gajos and Daniel S. Weld, 2004), etc. Many approaches for UI design and model-based user interface development environments (MB-UIDEs) have been proposed. Such approach do not consider totally complexity of UI controls as grid, graph and tree, sharing presentation space that can overlapping many present units so can show different content in different context, and their operational relations (Lu, Xudong, and Jiancheng Wan., 2007).

As a rule, usage of existing frameworks provides an excellent solution for developers, which makes them more relevant for the development of large software. In addition to the inconvenience of designing small applications, they have many other problems, such as complex updates, closed source, confusion, obsolescence, and low performance.

To perform localization task WPF is chosen as the target platform. Also, this platform is the most relevant among desktop applications, looking at the latest news of updates to development tools and programming languages.

The idea of adaptive view models designing admires that components are consist from graphical primitive or combination of them.

Adaptive view models are those view models that allow making visible full interface including all its details. All user interface elements are visible fully in the screen not depending about different factors as language, scale of components and other factors.

Thus, in order to save information about primitive, processing unit must store their key points, namely coordinates of vertexes and to propose an algorithm allowing to restore outside configuration of menu component.

WPF platform allows to support this idea, because it has information about coordinates of web application in XAML file.

Contribution of this paper representation of approach allowing designing adaptive view models to perform dynamic localization task.

Review of existing technologies for view models designing

Taking into account analysis of analogues, it is possible to determine indicators that should be obtained as a result of development.

For a clear definition of indicators, real applied research was conducted. To do this, we chose DevExpress (DevExpress, 2019) as a framework-analogue. The research was performed within the framework while development a commercial project and took place from 04/19/18 to 04/19/20. During the study the developers performed a commercial order using only ready-made components from DevExpress. From the beginning of the project, there were three developers with different qualifications who were completely unfamiliar with the this framework.

The first factor, of course, can be the time that an ordinary developer needs to spend to master the basic capabilities of the tool. Namely, to master the basic ready-made components, their properties, to understand the interaction for the use of adaptability of appearance models and to arrange their own approach to their use.

The result was not excellent and set an approximate average - 124 working days (8 hours of working time). Based on the this, the target is set. Seventy five working days were selected to be experience developer with some framework.

The seconds' factor is the ability to upgrade user controls that are part of the framework. It is not possible to modify all components as part of DevExpress. The reason for this is the complexity of some components, such as the document editor.

We should not forget about the time for which you can understand how to modify a component. The condition will be a change in the design of the component "DateEdit", which is a field for selecting the date using the calendar. Quite a normal situation and its implementation takes an average of 7 working hours. This time should be reduced to 4 hours.

The next factor considers the time to implement a small application that includes a main menu and a few simple windows. An important condition for the purity of the study is the mandatory modernization, so that the menu was on the whole window, not rectangular, normally stylistically designed and scalable. The study established the current average of 26 working days. Most of the time was spent on creating the main menu. The expected result is 18 working days.

Design is a very important part of the application. In the course of applied research, the complete obsolescence of the existing designs was determined, which was stated without controversy by all four investors of the project. Thus, they were forced to hire a designer, and the developers who participated in the study, redesigned all components of the interface.

Also, for the indicator we take into account the ability to support the dynamic change of the application language. DevExpress and some other frameworks not even support possibility to change languages on some of the components. This feature is mandatory factor for modern frameworks to support dynamic language change of view model.

Today from the most used technologies and frameworks are DevExpress and Telerik (Telerik, 2019). They provide a relatively large number of ready-to-use components without problems and errors. DevExpress and Telerik have a large

number of regular users (developers) who actively use them for their own purposes, mainly commercial in nature, namely - writing products for the customer. Of course, this advantage is due to their long stay on the market. A number of characteristics have been identified that have certain analogues and that will have a new library (see Table 1).

Table 1. Comparison of different characteristics frameworks for flexible interface designing

Criterion	Designed framework	DevExpress	Telerik
Dynamic interface localization	+	-	-
Convenience for small project usage (till 10000 lines of code)	+	-	-
Convenience for middle project usage (10000-40000 lines of code)	+	-	+
Convenience for large project usage (more than 40000 lines of code)	+	+	+
Adaptive models supporting	+	+	+
Extensibility	+	-	-
New view models	+	-	-
Easy support	+	-	-
Well documented	-	+	+
Possibility to support not only WPF	-	+	+
Free licence	+	-	-
View model update	+	+	+
Additional view models	+	+	+

Requirements to the approach of adaptive view models designing

Requirements to the approach of adaptive view models designing based on localization experience are represented below:

1. Support screen scaling, other words allowing to scale all components that are located in the screen;
2. Be extensible, namely allow adding of new graphical primitives into the component calculating actual size of every interface component;
3. Perform a dynamic localization without interface refreshing;
4. Quick reaction on event language changing;
5. Extensible source code based on design patterns;
6. Representation for the user of every localization event result;
7. Visible part of interface must be friend looking for user's comprehension.

Review of localization problems in existing technologies of interface designing

The localization of the application always requires special attention when an applications is used in different countries with different languages. Development of software projects for large amount of visitors usually requires the supporting of several interface languages.

Most often, changing the interface language is performed by selecting the desired language in the settings of the application, site, or user profile. At this very moment, the main problem arises - changing all the labels that the user sees. Interface restarting mechanism will be an extremely inconvenient way to change the localization. The following describes the solution by combining the use of dynamic resources and a custom mechanism in code behind. By using Dynamic resources, you can create resources directly in the XAML markup that will replace each other and tell control that the current value has changed. Everything would be great if from time to time there was no need to form the displayed text programmatically. For example, when we have a collection of elements, each of which changes language, or when translations are taken from

another service, such as API. Then Binding will be used instead of Dynamic resource.

Key points of the proposed algorithm

Describe a rectangle outside the User control. Define left right, top, and down borders. In the figure 1 key points are A,B,C, and D.

Consider an example Point A has coordinates:

- A(XL,YD), namely XLeft, YDown;
- Point C(XR,YU), namely XR – XRight, and YU-YUp.

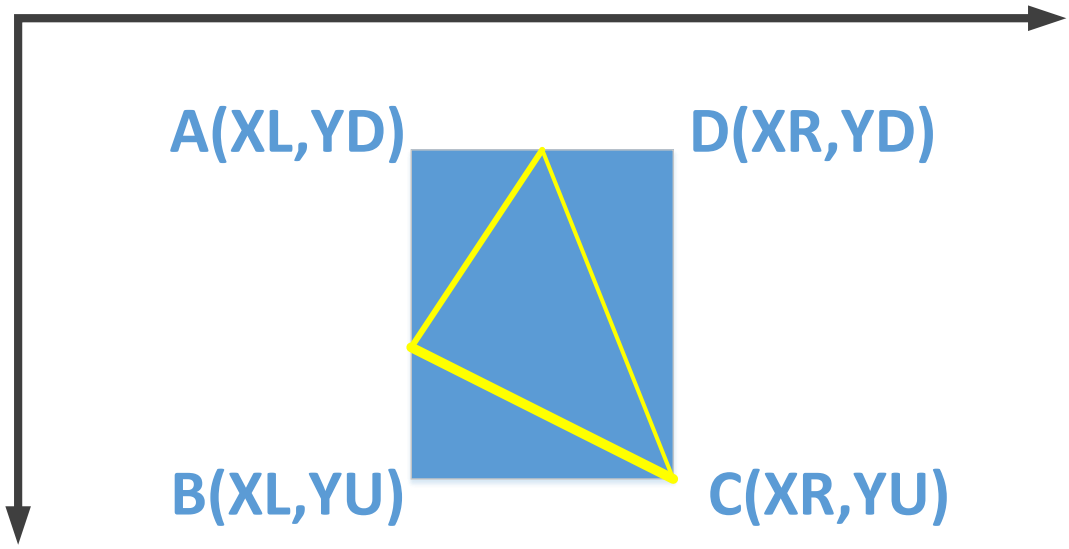

Figure 1. Rectungle with key points

Define a point of intersection of its diagonals (point O(XC,YC) (C - center) in the figure 2. This point is considered as central rectangle point, and considered as imaginary center of coordinate. Point O(XC,YC) provides further division of imaginary coordinate system into four coordinate quarters.

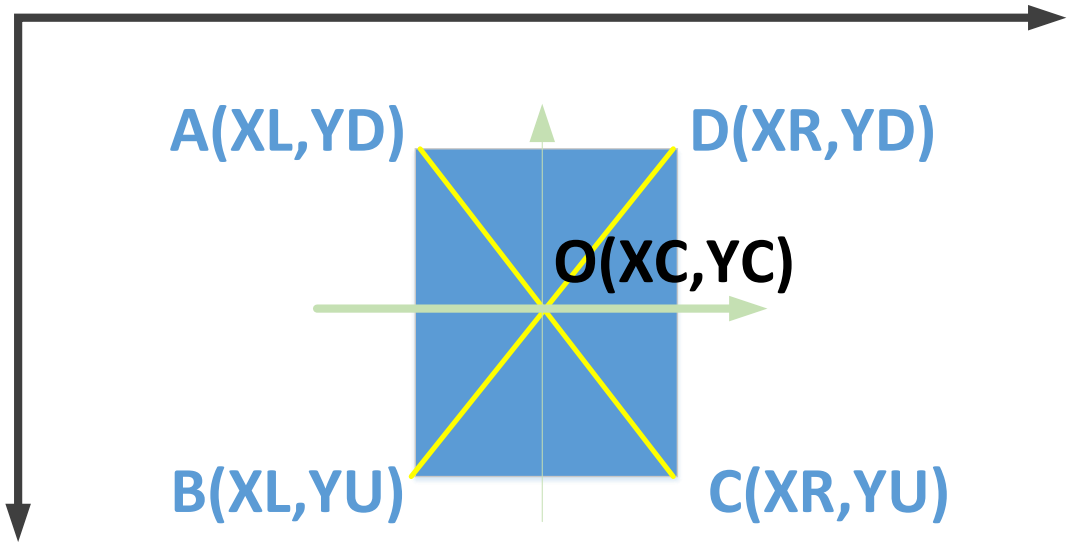

Figure 2. Central rectungle point

Denote a coefficient in which user wants to scale menu components as c, and its half as h, namely $h = c/2$ Increase the borders of the rectangle performing the next calculations:

$$\begin{cases} YU^{|} = YU - h \\ YD^{|} = YD + h \\ XL^{|} = XL - h \\ XR^{|} = XR + h \end{cases} \quad (1)$$

Example is shown in the figure 3. (scale is performed by cells when coefficient is +2)

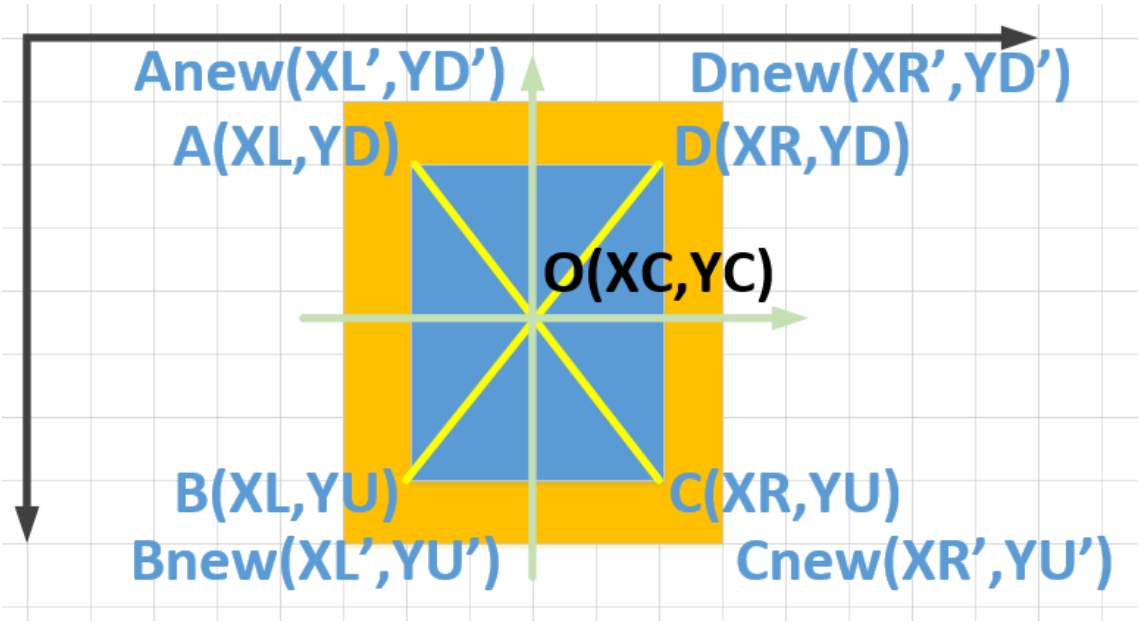

Figure 3. Border for scaling

Consider vertices of menu component.

If vertex V(X,Y) belongs to the first quarter it coordinates are calculated by

$$\begin{cases} X^{|} = X + h \\ Y^{|} = Y - h \end{cases} \quad (2)$$

If vertex V(X,Y) belongs to the second quarter it coordinates are calculated by

$$\begin{cases} X^{|} = X - h \\ Y^{|} = Y - h \end{cases} \quad (3)$$

If vertex V(X,Y) belongs to the third quarter it coordinates are calculated by

$$\begin{cases} X^{|} = X - h \\ Y^{|} = Y + h \end{cases} \quad (4)$$

If vertex V(X,Y) belongs to the fourth quarter it coordinates are calculated by

$$\begin{cases} X^{|} = X + h \\ Y^{|} = Y + h \end{cases} \quad (5)$$

In order to illustrate proposed hypnotize about coordinates consider new coordinate of the triangle from the figure 1.

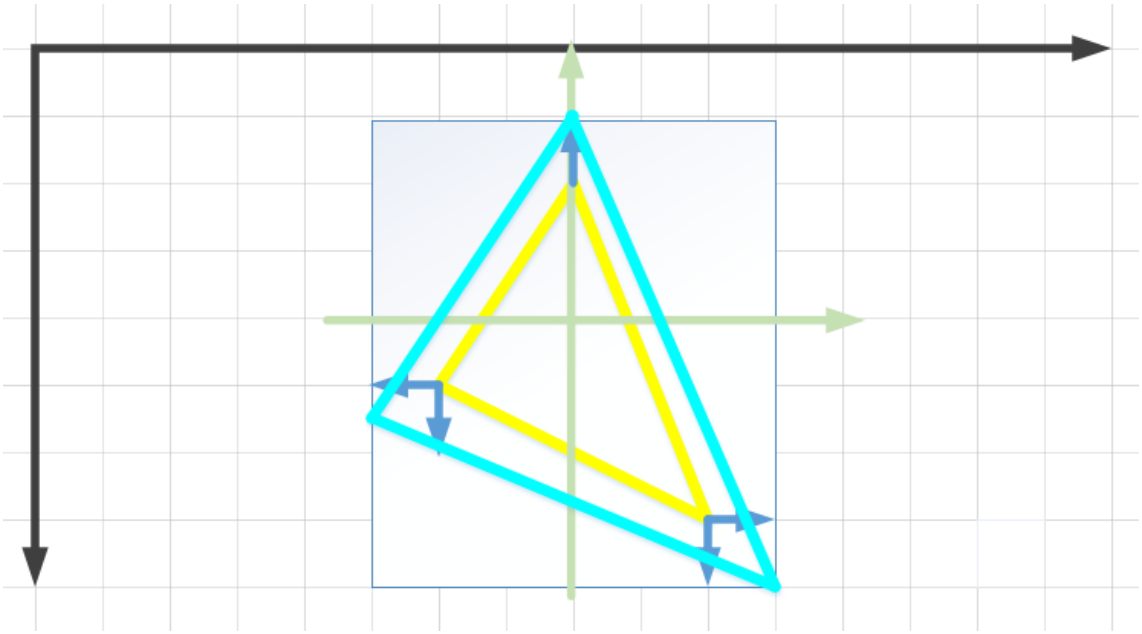

Figure 4. New triangle coordinates

The essence of the proposed approach:

In order to change the size and the scale of User Control when text inside of it is less or more the next activities are performed:

1. Define coordinates of the menu components as vertexes of the figure in vector representation;
2. Consider user input (for example mouse wheel event);
3. Define *h;*
4. Consider all coordinates of menu component. For each of one calculate new value of coordinates depending upon coordinate quarter to which coordinate belongs.

Key mechanisms of the framework for interface dynamic localization

Realization of set of components when their geometry depends upon length of labels supports by ***design pattern Abstract Factory.***

Drawing of the components when their geometry is designed as combination of different graphical primitives using vector-scaling algorithm is supported by ***design pattern Strategy***. Let us note that design pattern strategy allows adding new graphical primitives that support peculiarities of drawing algorithms.

Then design user controls that have complex structure are provided by ***Builder design pattern.***

Mechanism of defining new labels by means of dynamic resources loading is provided by flyweight design pattern and it is built into the WPF framework.

Design pattern Observer is used to notify all user controls that event of language changing is performed. This mechanism also is built into WPF framework.

Designing of framework for interface dynamic localization

Each interface component being developed must have a base Control class in the inheritance chain. This class will provide all the necessary properties for control maintains.

The Control class will allow you to adjust the background, frame, font, size, indents and more. You will also need to create a view of each interface component. To do this, create your own ControlTemplate and fill it with the necessary basic elements. Otherwise, the component will simply be transparent.

To help create small applications, you need to make some template project supporting architecture styles. The template should help the developer not to spend a lot of time on the design of the typical small project. The success of the idea is greatly supported by the high standardization of application implementations on WPF technologies.

The template architecture is based entirely on the MVVM pattern. It is a standard, proven and more than recognized solution for writing any project

within WPF technology. Also, this solution will allow you to easily navigate the implementation for third-party developers, which is part of the development goal.

The base class BaseControlViewModel has been created, which must be followed by all application windows. It will provide the window with the most necessary commands and methods: initiating, updating and closing the window, protection against multiple initiations and links to previous and subsequent models to link successors.

Added main window model to support MVVM. There is some implementation of the language collection to further configure the location to change the language of the application. It is now in the basic main window of the application.

Also prepared the integration of the tab component, of course, preserving the supplied MVVM architecture. From the first tab, the display of the main menu can be automatically initiated, or something else at the request of the user. Any of the above designs is easy to modify and expand.

Development of application based on adaptive view models designing

There are two ways to develop this tool: use the tool as a library of interface components or use it as a template.

In the first case, the practice of use is similar to all other frameworks. By inserting the required component into the xaml markup and using its fields. Usage is similar, but the components will be better. The components themselves have excellent performance for all goals.

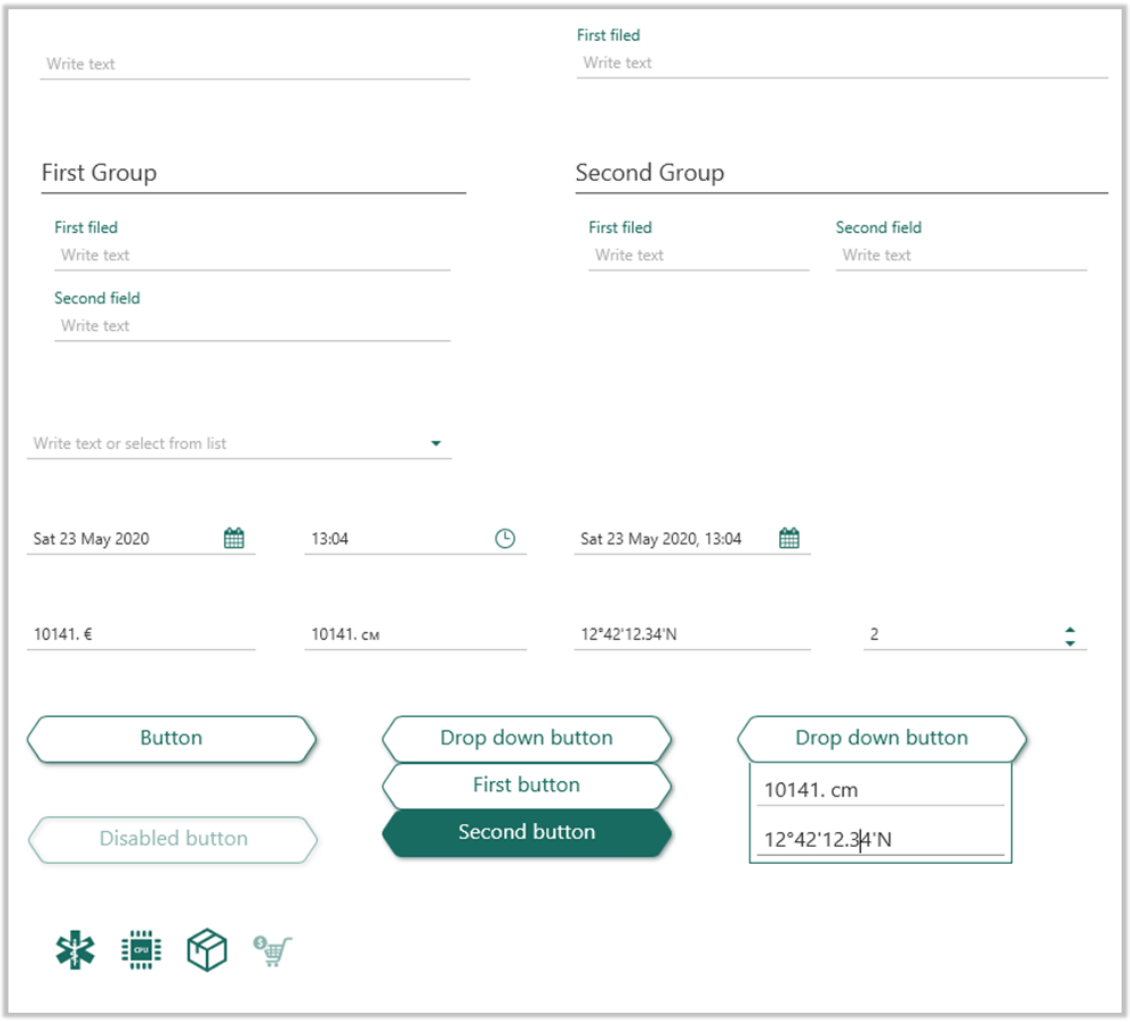

Figure 5. View controls library

The second case provides many more differences from similar means. It is necessary to first analyze the existing architecture to understand how not to violate it and follow it.

There is a DI container with the class name DIContainer.cs. To work with which you must first register the implementation of the interface. To do this, we need a module that is a class with a description of the implementations. By default, the MainModule class was created in this project. This is done through the RegisterType command.

Full registration looks like this:

```
builder.RegisterType<SomeClass>()
        .As<ISomeClass>()
        .SingleInstance();
```

There can be many modules, but they must be registered when starting the program via method DIContainer.Register.

Dynamic localization

A detailed study and implementation of changing the language of the application without restarting the application. It can be used not only when using a template, but it is necessary to fully understand the principle of operation.

First, let's start with the fact localization is saved by separate dictionaries in XAML format. Static ".resx" extension files greatly complicate the situation. This will allow you to use Dynamic resources if possible. It will also provide code on how to get a localized value programmatically.

A ResourceDictionary is created for each language and a language tag is added to the name for convenience, for example, lang.en.xaml. They will all have strings with the same keys, but with a value in the appropriate language. Here is one such dictionary in which the value Name is added:

```
<ResourceDictionary
   <v:String x:Key="Name">Name</v:String>
</ResourceDictionary>
```

It is necessary to make a mechanism for changing the language. We recommend saving the current language selected by the user in the App.config file within the appSettings tag. The language change will take place in three steps:

1. Write the new CultureInfo to App.config and assign to App.CurrentCultureInfo.
2. Call the substitution mechanism ResourceDictionary.
3. Call event to which all methods that update the values in the open window are signed.

To do all this, make a manager:

```
public class LanguageManager
{
   public event EventHandler CultureChangedEvent;
   public void SetCurrentCulture(CultureInfo cultureInfo)
   {
      AppConfig.LangCulture = cultureInfo.TwoLetterISOLanguageName;
      SetLanguage(cultureInfo);
      OnCultureChangeEvent();
   }
   private void OnCultureChangedEvent()
   {
      CultureChangedEvent?.Invoke(this, EventArgs.Empty);
   }
}
```

Now only the SetLocalization mechanism is missing, which will replace ResourceDictionary. It is necessary having received a desirable language, to establish it for current thread and to replace the used dictionary in Application. You should also run this code when you run the application, such as App.xaml.cs, to set the language that the user selected in the previous session. The method looks as follows:

```
private static void SetLanguage(CultureInfo culture)
{
   Thread.CurrentThread.CurrentUICulture = culture;
   Thread.CurrentThread.CurrentCulture = culture;
```

```
    var path = $"Resources/lang.{culture.TwoLetterISOLanguageName}.xaml";
    var currentDict = new ResourceDictionary
    {
        Source = new Uri(path, UriKind.Relative)
    };
    var previeous = Current.Resources.MergedDictionaries
          .FirstOrDefault(dictionary =>
          dictionary.Source.OriginalString.StartsWith("Resources/lang."));
    if (previeous != null)
    {
        var index = Current.Resources.MergedDictionaries.IndexOf(previeous);
        Current.Resources.MergedDictionaries.Remove(previeous);
        Current.Resources.MergedDictionaries.Insert(index, currentDict);
    }
    else
    {
        Current.Resources.MergedDictionaries.Add(currentDict);
    }
}
```

To improve this, use the Singleton design pattern manager and make the instance of User control static. Additionally, you can make an interface for the manager to support the DI container.

Now for use as a dynamic resource it is enough to address on a line key with a necessary word.

Let's move on to the third point of the framework - subscribing to a language change event. When the label is defined from code, not XAML markup, you must subscribe to the LanguageManager.CultureChangedEvent language change event. It is convenient to subscribe to a language change event for a Load event and unsubscribe from an Unload event. Now that the language has changed, it is possible to respond with appropriate actions, such as resetting the value of another language.

To set a new value it must be obtained from the created ResourceDictionary directly from the code. These values are really easy to access using the basic Application.Current.TryFindResource method. Note that it would be much better to do a separate method for this.

```
private static string GetResourceByName(string resourceName)
{
   return Application.Current.TryFindResource(resourceName) as string;
}
public static string Name => GetResourceByName("Name");
```

Now it will be extremely easy to get the value. When used, each time will be given an actual value relative to the current language of the application. Simply substitute the value instead of the current one. When using MVVM, the model needs to be called RaisePropertyChanged (nameof (NameTitle)). This will notify the XAML markup that the value of this field has changed. In turn, Binding will also track changes, so you need to correctly define Mode (not one way to source).

At this stage, you can stop, as this implementation is enough, but there is another interesting case. This most often occurs when using frameworks with ready-made components.

Imagine a situation where there is some control that takes a value in the form of a large number. It also has a mask that places the appropriate characters after thousands and at the end of the integer. Thus, the mask depends on the installed application language. Depends on the fact that in some countries use commas after a thousand and a period after a whole part, and in other countries on the contrary, or with other punctuation marks.

When the language changes, the number and mask remain the same, they have not changed their meanings. The display itself must be updated. In this case, the control will most likely have a field for transmitting the current localization language. You can of course transmit it in response to a language change event, as described above, but there is a much better way.

You must use static fields that can trigger a StaticPropertyChanged event. This will be very easy to use, as everything will happen automatically. To do this, create a static field, an event to update it and subscribe once to the language change event:

```
public static class LanguageManagerInstance
{
   private static CultureInfo _currentThreadCulture;
   private static readonly PropertyChangedEventArgs
CurrentMaskCulturePropertyEventArgs = new
PropertyChangedEventArgs(nameof(CurrentThreadCulture));

   public static event PropertyChangedEventHandler StaticPropertyChanged;
   public static ILanguageManager Instance { get; }
   public static CultureInfo CurrentThreadCulture
   {
      get => _currentThreadCulture;
      set
      {
         _currentThreadCulture = value;

StaticPropertyChanged?.Invoke(null,CurrentMaskCulturePropertyEventArgs);
      }
   }
   static LanguageManagerInstance()
   {
      Instance = DI.Resolve<ILanguageManager>();
      Instance.CultureChangedEvent += OnLanguageChanged;
   }
   private static void OnLanguageChanged(object sender, EventArgs e)
   {
      CurrentThreadCulture = Thread.CurrentThread.CurrentCulture;
   }
}
```

It is important to pay attention to the name of how StaticPropertyChanged is called. This is the main essence of the update. Be especially careful with arguments. Unsubscribe from the language change event, of course, is not required, as the mechanism always works and is in a static class.

The use will be analyzed on the example of the DevExpress framework. In it the text editor is called TextEdit and has the MaskCulture field in which it is necessary to transfer actual CultureInfo. The proposed design is used very easily, you just need to refer to this field:

```
<dxe:TextEdit Mask="###,###,###.#"

MaskCulture="{BindingPath=(r:LanguageManagerInstance.CurrentThreadCulture)}"/>
```

Conclusion

Template based approach for adaptive view models designing for desktop application is proposed in this paper. Proposed approach based on simple mathematical calculations that become a part of framework supporting mechanisms of resining user control after receiving notification about user to change the language, scale or layouts. Proposed framework covers all requirements, represented in the table 1. Using proposed approach lets obtain the next advantages in comparison with Telerik and DevExpress frameworks: (i) easy support; (ii) flexible modification of set of user control; (iii) free license; (iv) convenience for the development of any scaling project. Investigation of factors influenced on effective usage of the proposed framework make a conclusion that time for getting experience of the confident works is less that DevExpress and Teleric. Development details and architectural design patterns allows to design a workflow for performing all frameworks action to get flexible solution needed for different DevOps activities.

Bibliography

(Andrade, 2007) Andrade Chris, Livermore Shawn, Meyers Mike, Van Vliet Scott. .NET Development with the Windows Presentation Foundation. Professional WPF Programming. Wiley Publishing Inc., Indianapolis, Indiana, 2007, 452 p.

(Ethan, 2011) Ethan Marcotte. Responsive Web Design. — A Book Apart, 2011. — 143 c. — ISBN 978-0-9844425-7-7.

(Devexpress, 2019) [electronic resource] / https://www.devexpress.com/

(IFML, 2016) [electronic resource] / Wikipedia – Mode of access https://en.wikipedia.org/wiki/Interaction_Flow_Modeling_Language

(Griffiths, Tonya, Barclay and Peter J, Teallach, 2001) Griffiths, Tonya, Barclay and Peter J, Teallach: A model-based user interface development environment for object databases, Interacting with Computers,vol.1, 2001,pp.31-68.

(Krzysztof Gajos and Daniel S. Weld, 2004) Krzysztof Gajos and Daniel S. Weld, SUPPLE: Automatically Generating User Interfaces, in Proceedings of IUI'04, Funchal, Portugal,2004, pp.83-100

(Lu, Xudong, and Jiancheng Wan., 2007) Lu, Xudong, and Jiancheng Wan. "Model Driven Development of Complex User Interface." *MDDAUI*. 2007

(Marco Brambilla, Pietro Fraternali, 2015) Marco Brambilla, Pietro Fraternali. Interaction Flow Modeling Language. Model-Driven UI Engineering of Web and Mobile Apps with IFML, 2015.

(Mitchell K. et. al, 1996) K. Mitchell, J. Kennedy, P. Barclay, A Framework for User Interfaces to Databases (DRIVE), in Proceeding of AVI, ACM Press, 1996.

(Mori G. et. al., 2004) Giulio Mori, Fabio Paterno and Carmen Santoro, Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, 30(8), 2004, pp.1-14.

(Nunes, N.J. at. el., 2005) Nunes, N.J. and J.F.e. Cunha, Wisdom – A UML based architecture for interactive systems, in Proceeding of DSV-IS, Limerick, Ireland, 2000, pp.191-205.

(Pedro J. Molina et. al., 2002) Pedro J. Molina, Santiago Meliá and Oscar Pastor, JUST-UI: A User Interface Specification Mode, in Proceedings of CADUI 2002, Valenciennes, France, 2002, pp.63-74

(Puerta A. and Eisenstein J., 1999) Angel Puerta and Jacob Eisenstein, Towards a General Computational Framework for Model-Based Interface Development Systems (MOBI-D), in Proceeding of ACMIUI 1999, Redondo Beach CA USA

(Telerik, 2019) electronic recourse https://www.telerik.com/

(Weil, 2016) Weil Arnaud. XAML, C# and the MVVM pattern. Learn WPF MVVM. Leanpub, 2016, 165 p.

(Nathan, 2015) Nathan Adam. Universal Windows Apps with XAML and C#. Person Education Inc., 2015, 723 p.

Authors' Information

Olena Chebanyuk – Software Engineering Department, National Aviation University, Kyiv, Ukraine,

Major Fields of Scientific Research: Domain Engineering, Model-Driven Architecture, Model-Driven Development, Software Architecture, Mobile development, Software development,

e-mail: chebanyuk.elena@ithea.org, chebanyuk.elena@gmail.com

Vladyslav Pashystyi – *Middle Developer of Softheme company, Kyiv, Ukraine. Student of Software Engineering Department, National Aviation University, Kyiv, Ukraine.*

Major Fields of Scientific Research: Architectural programming, Responsive user interfaces.

E-mail: *pvladvoland@gmail.com*