Poznań University of Technology

Faculty of Control, Robotics and Electrical Engineering

Institute of Robotics and Machine Intelligence

Division of Control and Industrial Electronics

Desktop application C# and WPF

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:

- basics of C# language and .NET framework,
- LINQ technology,
- available methods of network communication in .NET,
- WPF technology and XAML language,
- MVVM design pattern.

Skills

The aim of the course is to acquire skills in:

- building desktop applications using the .NET framework,
- creating GUI using WPF and XAML,
- writing application logic using C# language,
- implementation of client-server communication in C#,
- implementation of a desktop application using the MVVM design pattern.

Social competences

The aim of the course is to develop proper attitudes:

- proper management of the desktop application code base, taking into account chosen design pattern,
- proper division of functionality in accordance with chosen design pattern.
- 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,
- choosing the right technology and programming tools for the given problem,
- testing developed IT system.

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 Sunday, June 6, 2021 by 23:59.

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) Introduction to C# and .NET

C# is a high-level, multi-paradigm general-purpose programming language. It is often referred to as *object-oriented* language, due to its strong reliance on the mechanism of classes and interfaces. C# is tightly integrated with the .NET, which is both a framework and a runtime environment [2]. It was originally created for writing applications for Windows operating systems, but is now a multi plaform programming tool [3]. In C# you can build: window applications, CLI applications, *services* for Windows, libraries and Windows components, mobile and web applications. C# is standardized by ECMA International as the standard *ECMA-334* [4] and by ISO as the standard *ISO/IEC 23270* [5].

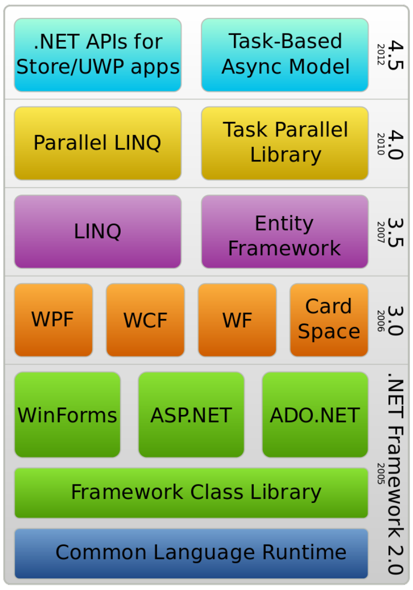

Fig. 1. .NET Framework components.

C# uses the *Common Type System* (CTS). CTS in .NET supports the following five categories of types [6]:

- Class - defines **operations** that an object (i.e. *instance* of a class) can perform (methods, events or properties) and **data** that an object contains (fields). Contains implementations of defined operations.
- Structure - in .NET, all primitive data types like `Char`, `Double`, `Boolean`, `Int32` etc. are defined as structures. Structures also contain data and operation, they differ from classes because they cannot be inherited.
- Enumerations - have a name and contain a set of fields in form of integer values. Do not contain operations.

- Interface - like class, can have properties, methods and events, but all of which are abstract members. Interfaces cannot contain constructors or fields. To implement interface operations, a class inheriting from it must be used.
- Delegates - reference types that are used similar to e.g. function pointers in C++. They are used to handle events and callback functions in .NET.

.NET Framework (pronounced „dot net”) is a programming platform developed by Microsoft that works primarily on Windows. This framework consists of two main components. The first is the class library called *Framework Class Library* (FCL) and provides language interoperability (each language can use code written in other languages) in several programming languages. The second is *Common Language Runtime* (CLR), i.e. a virtual machine that manages the execution of .NET programs, with additional services such as security, memory management and exception handling [7]. Figure 1. shows the key components of the .NET platform.

Language interoperability - which means that any language can use code written in other languages - is a key feature of the .NET platform. Because the Intermediate Language (IL) code produced by the C# language compiler conforms to the CTS specification, it may interact with code generated from Visual Basic, Visual C++, or any of the more than 20 other languages compatible with CTS. A single project can contain multiple modules written in different .NET languages, and types can reference each other as if they were written in the same language [8]. Fig. 2. shows the relationship between the source code in C# and the .NET Framework.

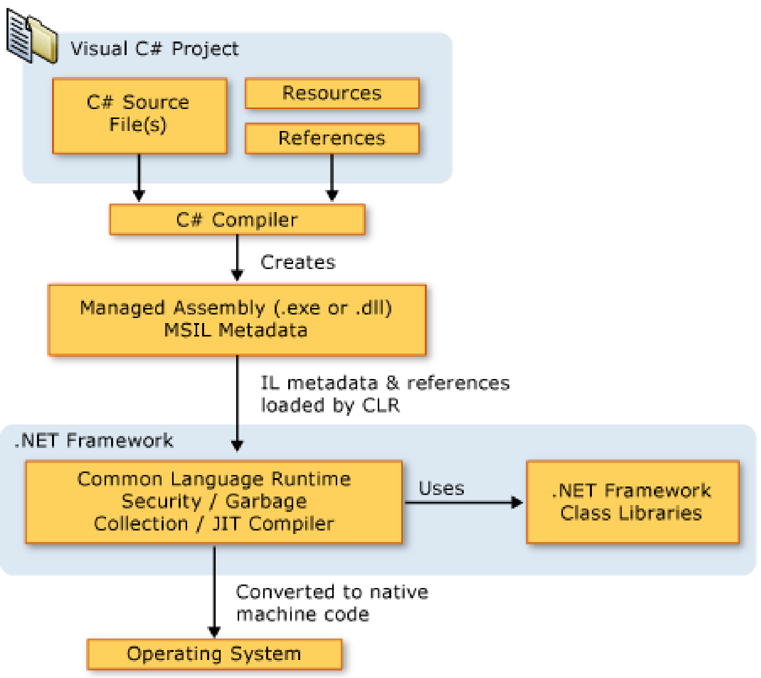
Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

Fig. 2. .NET Framework architecture.

.NET Framework contains a comprehensive library of over 4,000 classes organized in namespaces that provide a wide range of useful funcionalities like network communication, database support, syntactic analysis as well as a wide range of containers using LINQ technology.

The most popular integrated development environment for C# is Visual Studio - available for free download in the *community* version [9].

1. LINQ technology

Language Integrated Query (LINQ, pronounced „link") is an important component of the .NET Framework that adds native query functions to .NET languages. LINQ extends the language by adding a queries similar to those in SQL and can be used to conveniently extract and process data from arrays, enumerable classes, XML and JSON documents, relational databases and other external data sources. It also defines a set of method names (called standard query operators or standard sequence operators), along with the compiler rules to translate queries that use those methods, lambda expressions, and anonymous types [10].

Although originally LINQ was part of the .NET technology, there are currently ports for PHP (PHPLinq) [11], JavaScript (linq.js) [12], and many other languages. Note, however, that ports are not strictly equivalent to LINQ in .NET.

Listing 1. and 2. presents, respectively: sample data in JSON format and an example use of LINQ and the library JSON.NET [13] to read the value of one of data attributes. The advantage of this approach is the ability to dynamically search data without a priori knowledge of its structure.

Listing 1. An example of data in JSON format.

```
{ "id": 321443, "measurement":
  { "envdata":
    { "temp": "20.3C", "press": "1013.2hPa", "hum": "80%" }
  }
}
```

Listing 2. An example of handling JSON using LINQ.

```
JObject jObj = JObject.Parse(jsonText);
var temp = (string)jObj.Descendants()
                        .OfType<JProperty>()
                        .Where(p => p.Name == "temp")
                        .First()
                        .Value;
```

2. HttpWebRequest and HttpClient

Two basic methodologies for performing HTTP requests on .NET use classes `HttpWebRequest` [14] and `HttpClient` [15]. `HttpWebRequest` class is an older solution and is currently less frequently used than `HttpClient` class. The main advantages of former solution are the possibility of very detailed request configuration and the possibility of synchronous execution of requests. Kistings 3. and 4. an examples of HTTP request configuration with GET and POST methods using `HttpWebRequest` are presented. Listing 5. shows the procedure of getting the server response as text (`string`), independent of a request method. The example of the POST request shows that the class `HttpWebRequest` requires precise configuration by the developer.

Listing 3. An example of HTTP GET request configuration using `HttpWebRequest` class.

```
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(uri);
// GET request configuration
request.Method = "GET"; // default value
```

Listing 4. An example of HTTP POST request configuration using `HttpWebRequest` class.

```
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(uri);
// POST request data
var requestData = "filename=chartdata";
byte[] byteArray = Encoding.UTF8.GetBytes(requestData);
// POST request configuration
request.Method = "POST";
request.ContentType = "application/x-www-form-urlencoded";
request.ContentLength = byteArray.Length;
// Wrire data to request stream
Stream dataStream = request.GetRequestStream();
dataStream.Write(byteArray, 0, byteArray.Length);
dataStream.Close();
```

Listing 5. An example of sending an HTTP request and reading server response using `HttpWebRequest` class.

```
using(HttpWebResponse response=(HttpWebResponse)await request.GetResponseAsync())
using(Stream stream = response.GetResponseStream())
using(StreamReader reader = new StreamReader(stream))
{
    responseText = await reader.ReadToEndAsync();
}
```

`HttpClient` class is a newer and currently more commonly used solution. This class significantly facilitates communication with a server with REST API. The main limitation of this solution is entirely asynchronous implementation of requests. In practice, however, this is a desirable property in most applications. On listings 6. and 7. an examples handling HTTP request with GET and POST methods using `HttpClient` are presented.

Listing 6. An example of HTTP GET request using `HttpClient` class.

```
using (HttpClient client = new HttpClient())
{
  responseText = await client.GetStringAsync(uri);
}
```

Listing 7. An example of HTTP POST request using `HttpClient` class.

```
using (HttpClient client = new HttpClient())
{
    // POST request data
    var requestDataCollection = new List<KeyValuePair<string, string>>();
    requestDataCollection.Add(new KeyValuePair<string, string>("key", "value"));
        ...
    var requestData = new FormUrlEncodedContent(requestDataCollection);
    // Sent POST request
    var result = await client.PostAsync(uri, requestData);
    // Read response content
    responseText = await result.Content.ReadAsStringAsync();
}
```

c) Introduction to WPF and XAML

Windows Presentation Foundation (WPF) is a graphical user interface (GUI) framework that is used to build client applications. The WPF programming platform supports a wide set of programming functions, including the application model, resources, controls, graphics, composition layout, security, and *data binding* between GUI and application logic. This framework is part of .NET. WPF uses Extensible Application Markup Language (XAML) to provide *declarative* application programming model [16]. WPF allows to create applications using both markup (XAML) and code (e.g. C#). Typically, XAML tags are used to implement layout of the application, and general-purpose programming languages (so-called *code-behind*) to manage its behavior [17].

XAML simplifies the creation of a user interface for .NET applications. XAML directly instantiates objects in the specified type set. This is different from most other markup languages, which are usually interpreted languages without such a direct link to the object system [18]. XAML code can be written in two basic ways: using attributes (properties) as components of the element (listing 8.) nd using attributes as separate sub-elements (listing 9.).

Listing 8. XAML syntax: attributes as part of the element definition.

```xml
<Button
    Height="30"
    Width="150"
    Margin="30,10,0,10"
    HorizontalAlignment="Left">
    CLICK ME!
</Button>
```

Listing 9. XAML syntax: hierarchical definition of attributes within an element body.

```xml
<Button>
    CLICK ME!
    <Button.Width>
        150
    </Button.Width>
    <Button.Height>
        30
    </Button.Height>
    <Button.HorizontalAlignment>
        Left
    </Button.HorizontalAlignment>
    <Button.Margin>
        30,10,0,10
    </Button.Margin>
 </Button>
```

In the Visual Studio environment, GUI creation using WPF and XAML is significantly simplified by a set of tools that allow *drag and drop* of controls from toolbox to dynamically created preview of an application window and configuration of *attributes* ordered in the form of a list. The XAML code is generated automatically.

d) MVVM design pattern

Model-View-ViewModel (MVVM) is a software design pattern that makes it easier to separate the development of the graphical user interface (*View*) - whether through markup language or direct GUI code - from the development of the application logic (*Model*), so that the view is not dependent on any particular model implementation. The *View Model* is a data converter, which means that it is responsible for converting data objects from Model in such a way that the objects are easily managed and presented in View. In this respect, the View Model is „more a model than a view" and supports most, if not all, logic of displaying the view. Fig. 3 presents a conceptual relationship between design elements.

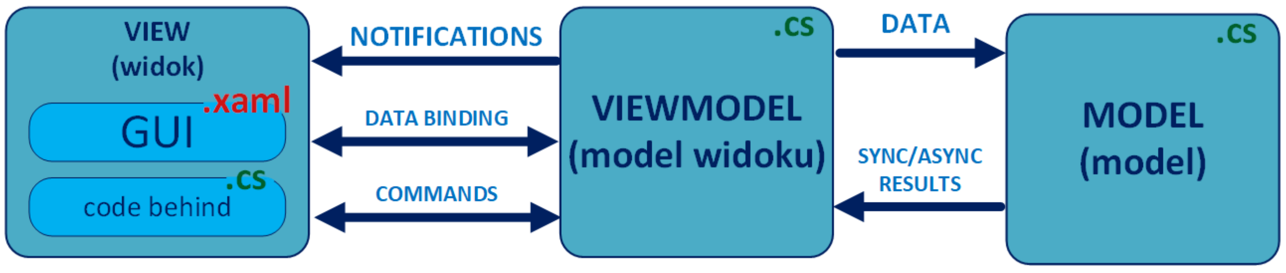

Fig. 3. The relationship between a View, a View Model, and a Model in the MVVM design pattern.

Important concepts in implementing the MVVM pattern in .NET using WPF and C# are notifications, commands and the mechanism of data binding. Notifications and commands are typically implemented by classes inheriting from the dedicated interfaces: `INotifyPropertyChanged` [19] and `ICommand` [20]. *Data binding* uses the special class `Binding` [21] directly in the XAML code. On the listing 10. is presented an example of a GUI element (`TextBox`) which attribute `Text` is binded with the property `Input` of class `SimpleViewModel`. `SimpleViewModel` class, presented in listing 11., implements the interface `INotifyPropertyChanged`, therefore it contains handler of the event `PropertyChanged` and standard function that calls this handler. This function is used when value of the `Input` property was changed.

Listing 10. An example of an XAML element using data binding mechanism.

```
<Window ... >
    <Window.DataContext>
        <myviewmodel:SimpleViewModel/>
    </Window.DataContext>
    ...
    <!-- Text input -->
    <TextBox Text="{Binding␣Input}"/>
   ...
</Window>
```

Listing 11. An example of a simple class implementing a View Model with the `Input` *property associated with a GUI element.*

```
public class SimpleViewModel : INotifyPropertyChanged
{
    #region Properties
    private string input; // privte FIELD
    public string Input   // public PROPERTY with 'get' and 'set'
    {
        get
        {
            return input;
        }
        set
        {
            input = value;
            OnPropertyChanged("Input");
        }
    }
    #endregion
    ...
    #region PropertyChanged
    public event PropertyChangedEventHandler PropertyChanged;

    /**
      * @brief Simple function to trigger event handler
      * @params propertyName Name of SimpleViewModel property as string
      */
    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
            handler(this, new PropertyChangedEventArgs(propertyName));
    }
    #endregion
}
```

IV. Scenario for the class

a) Teaching resources

Hardware	• computer,
Software	• Visual Studio IDE (Windows) or MonoDevelop IDE (multiplatform),
	• web server (eg. XAMPP, Lighttpd),

b) Tasks

Familiarize yourself with examples of desktop applications available in GitHub repository. Create a desktop application for selected the operating system (e.g. Windows, Mac, Linux). Use e.g. .NET platform: C# for application logic and XAML to create user interface. Use the MVVM architectural pattern to separate your code into a view, a view model, and a model. Apply additional XAML sheets to format *theme* (style) of the document (*optional*).

1. Create REST client applications for a personal computer 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.

   (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 PC with selected operating system. 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 module (view, e.g. a new tab) 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 PC with selected operating system. 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 your application with module (view, e.g. a new tab) to configure and save user settings.

   (a) The configuration page should contain basic application settings, such as port number, server API version (convenient for the developer), requests sample time, maximum number of saved samples, etc.

   (b) Configuration information should be saved locally on **disk** as a JSON text file.

   (c) After reloading web page, saved user's settings should be read from the file.

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. BILLWAGNER. *Dokumenty C# — wprowadzenie, samouczki, informacje.* [Online]. [N.d.] [visited on 2020-05-04]. Available from: https://docs.microsoft.com/pl-pl/dotnet/csharp/. Library Catalog: docs.microsoft.com.
3. *Supported Platforms | Mono* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://www.mono-project.com/docs/about-mono/supported-platforms/.
4. *Standard ECMA-334* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://www.ecma-international.org/publications/standards/Ecma-334.htm.
5. 14:00-17:00. *ISO/IEC 23270:2003* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/67/36768.html. Library Catalog: www.iso.org.
6. THRAKA. *Wspólny system typów* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://docs.microsoft.com/pl-pl/dotnet/standard/base-types/common-type-system. Library Catalog: docs.microsoft.com.
7. DOTNET-BOT. *Dokumentacja platformy .NET* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://docs.microsoft.com/pl-pl/dotnet/standard/. Library Catalog: docs.microsoft.com.
8. BILLWAGNER. *Wprowadzenie do języka C# i systemu .NET Framework* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://docs.microsoft.com/pl-pl/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. Library Catalog: docs.microsoft.com.
9. *Visual Studio 2019 | Pobierz bezpłatnie* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://visualstudio.microsoft.com/pl/vs/. Library Catalog: visualstudio.microsoft.com.
10. BILLWAGNER. *Zapytanie zintegrowane z językiem (LINQ) (C#)* [online]. [N.d.] [visited on 2020-05-04]. Available from: https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/concepts/linq/. Library Catalog: docs.microsoft.com.
11. *CodePlex Archive* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://archive.codeplex.com/. Library Catalog: archive.codeplex.com.
12. CIURARU, Mihai. *mihaifm/linq* [online]. 2020 [visited on 2020-05-05]. Available from: https://github.com/mihaifm/linq. original-date: 2012-04-07T11:39:29Z.

13. *Json.NET Schema - Newtonsoft* [online]. [N.d.] [visited on 2020-05-05]. Available from: https://www.newtonsoft.com/jsonschema.
14. *HttpWebRequest Klasa (System.Net)* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/dotnet/api/system.net.httpwebrequest. Library Catalog: docs.microsoft.com.
15. *HttpClient Klasa (System.Net.Http)* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/dotnet/api/system.net.http.httpclient. Library Catalog: docs.microsoft.com.
16. TERRYGLEE. *Co to jest platforma WPF? - Visual Studio* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/visualstudio/designers/getting-started-with-wpf. Library Catalog: docs.microsoft.com.
17. THRAKA. *Wprowadzenie do WPF* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/dotnet/framework/wpf/introduction-to-wpf. Library Catalog: docs.microsoft.com.
18. THRAKA. *Przegląd XAML - WPF* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/dotnet/desktop-wpf/fundamentals/xaml. Library Catalog: docs.microsoft.com.
19. *INotifyPropertyChanged Interfejs (System.ComponentModel)* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/dotnet/api/system.componentmodel.inotifypropertychanged. Library Catalog: docs.microsoft.com.
20. *ICommand Interfejs (System.Windows.Input)* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/dotnet/api/system.windows.input.icommand. Library Catalog: docs.microsoft.com.
21. *Binding Klasa (System.Windows.Data)* [online]. [N.d.] [visited on 2020-05-06]. Available from: https://docs.microsoft.com/pl-pl/dotnet/api/system.windows.data.binding. Library Catalog: docs.microsoft.com.