Mobile Application Development

PC Remote Controller

Kamil Janowski

Bachelor's Thesis
5.2013

Information Technology
Technology, communication and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s) JANOWSKI Kamil	Type of publication Bachelor´s Thesis	Date DDMMYYYY
	Pages 50	Language English
		Permission for web publication (X)
Title Mobile Application Development		
Degree Programme Information Technology		
Tutor(s) PELTOMÄKI Juha		
Assigned by		

Abstract

The project discusses a remote controller for a PC with Microsoft Windows operating system installed.
The project consists of three applications: server written with Qt Framework, a remote controller client for Android 2.2 operating system and a remote controller client for Windows Phone 7.1.
The server listens to messages sent to it by remote mobile clients via TCP and UDP messages, depending on what kind of operation is to be performed by the server. It can generate system keyboard and mouse events such as mouse click, mouse movement, scroll, keyboard key down and keyboard key up. Both clients contain custom controls resembling the actual PC input devices such as a keyboard and a touchpad, in order to perform the actual input that is supposed to be parsed by the server.
The thesis includes the full implementation of the project as well as the documentation explaining exactly how the three applications work.

Keywords
Java, C#, Qt Framework, Android, Windows Phone

Miscellaneous

TABLE OF CONTENTS

TERMINOLOGY

Client – any application that is able to communicate with the PCRemote Server.
Mobile device – device running either Windows Phone (at least 7.1 version) or Android (at least 2.3 version) operating system with client application installed.
Message Queue – Queue (First-In-First-Out) containing set of NetworkMessage objects.
HSQLDB – database often used for unit tests, as it provides multiple database dialects and it is possible to keep the entire database in memory without necessity of creating actual network connection.
Dalvik – Java Virtual Machine developed by Google, that is used in Android operating system.
TCP – Transmission Control Protocol; a connection communication protocol.
UDP – User Datagram Protocol; a connectionless communication protocol.
Maven – application for managing the project. It allows to generate the base for the project, compile the project as well as link external resources to it.

LIST OF FIGURES

LIST OF TABLES

1 Introduction to the topic

1.1 Purpose

The purpose of the project is to ease user’s access to the computer when showing a multimedia presentation.

Using just a PC mouse and keyboard might often be unhandy when giving the presentation. The user must stay in front of the computer at all times to switch slides and show additional content on the screen, which makes it difficult to make contact with people that listen to the presentation. PCRemote allows avoiding this necessity. The project allows taking the full control over computer cursor, keyboard, media keys and even files stored on the computer from a mobile device that can be simply carried around the room. This way the user can get closer to his listeners and is no longer bound to one place.

1.2 Project background

The whole project consists of three applications: a server written with C++ and Qt Framework that will run on Windows operating system and two clients, one for Android and one for Windows Phone.

The clients connect with the server over the network via TCP (for general data transfer purposes) and UDP (for mouse movement requests, since UDP does not perform any data correction checks, therefore it does not generate any delays) and will take the control over the cursor and the keyboard.

2 Theoretical basis

2.1 Why these technologies?

The reader might wonder why among non-functional requirements for server there is Windows, but not Linux even though it is supposed to be written with Qt Framework which is a multiplatform framework for C++. Well, the problem is, that even though Qt Framework makes it much easier to write applications with graphical user interface, it still does not provide all the features required by the project, such as simulation of mouse or keyboard button click. What is more, Linux native libraries do not provide this functionality either whereas WinAPI does.

Then why is it written in C++ at all? That is because C++ is a very efficient language and efficiency for background applications is very important. Users certainly do not want any background application that does not do anything for most of the time (unless you ask it to) to use up 20 MB of memory. What is more, applications written in C++ are totally stand-alone. Many potential users of the application have very little experience with usage of computers and satisfying a requirement of installing for instance JRE before an application can run, might be an unachievable task for them, especially since users are reluctant to read installation instructions.

As for clients, Android was chosen because it’s currently the most popular mobile operating system, and even though version 2.2 is already pretty old, it’s still the most popular version of Android.

Windows Phone recently started gaining on popularity; however there are still very few native applications for this operating system, which means that, there are very few competitive applications.

2.2 Android

Android is a popular Linux based open source mobile operating system that is currently developed by Google Inc. and is used in many tablets and mobile phones. Although the newest version of it is 4.2 “Jelly Bean”, the most popular one is still 2.3 “Gingerbread”

2.2.1 Basics

An important thing to know about Android is that it breaks the rule promoted by Oracle (creator of Java) “write once, run anywhere”. Android uses its own Java Virtual Machine called Dalvik. It slightly differs from JRE that can be downloaded from Oracle site. Why was it then decided to do that? The answer is simple – performance and improved integration. Unfortunately, if applications are written for one specific virtual machine, run only in Android operating system, portability of the application is lost as it will eventually be able to run only in Android environment.

In Android operating system there are 3 most basic entities: Activities, Services and Widgets. PCRemote actually uses Activities only, since they are most commonly used in Android applications as well as the other two do not really provide any useful functionality for the project.

Activity is the most basic view the user can see right after starting the application. It may contain all kind of controls like buttons, labels and pictures. Single app usually contains a multiple of Activities. All Activities are removed from memory right after the application gets closed.

An Activity has a following lifecycle which is illustrated in the figure below (FIGURE 1).

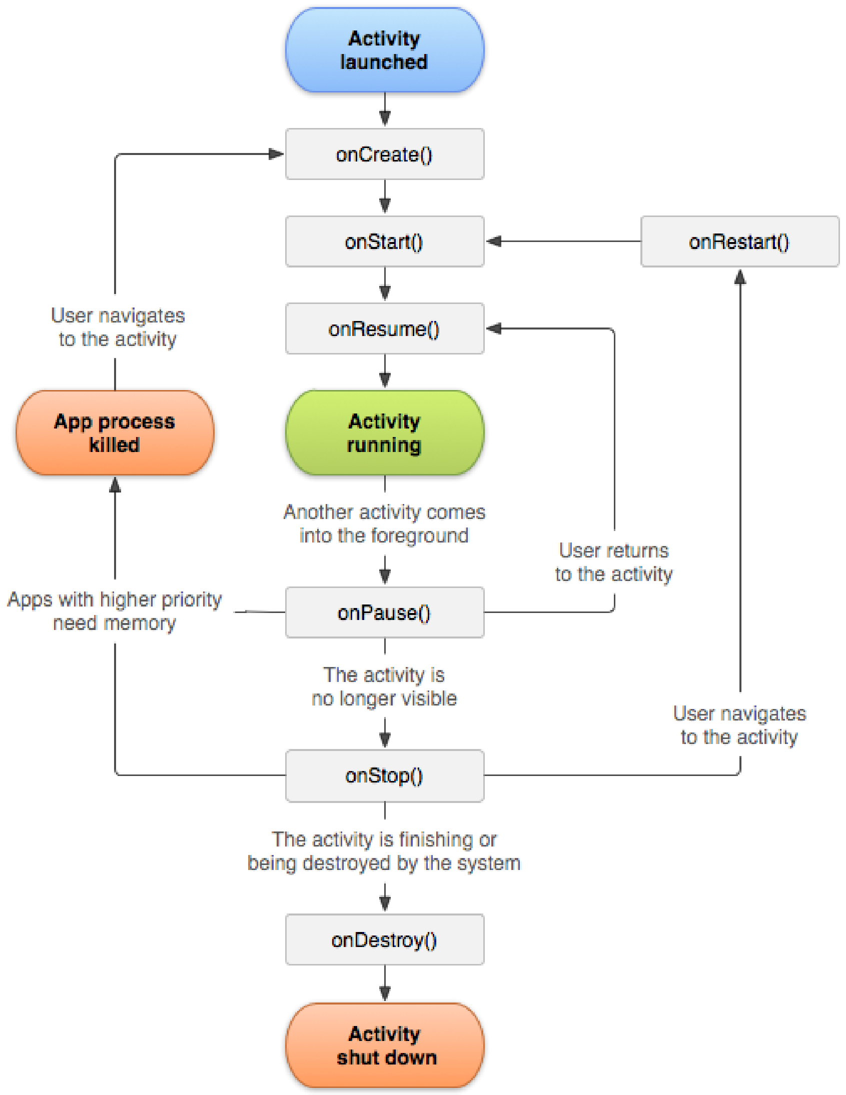

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

FIGURE 1. Android: Activity Lifecycle

An example of activity could be as follows:

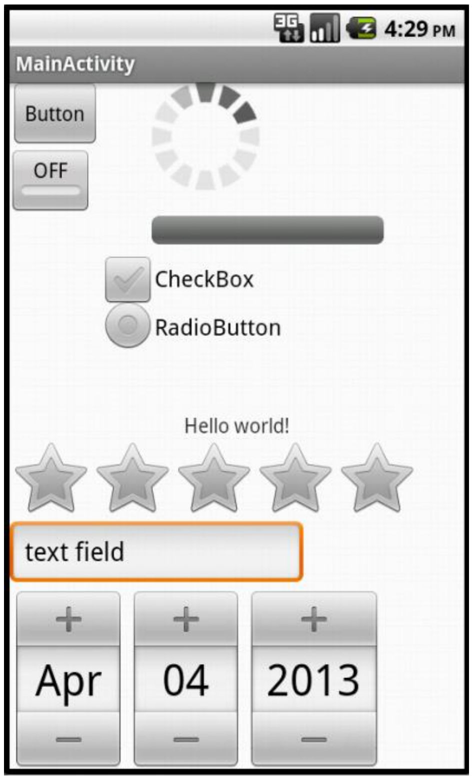


FIGURE 2. Android: Activity Example

Widgets are the functional elements of an application that can be easily added straight to the main desktop of the operating system. Unlike Activities, widgets are active at all times, provided that the user actually puts them on the desktop.

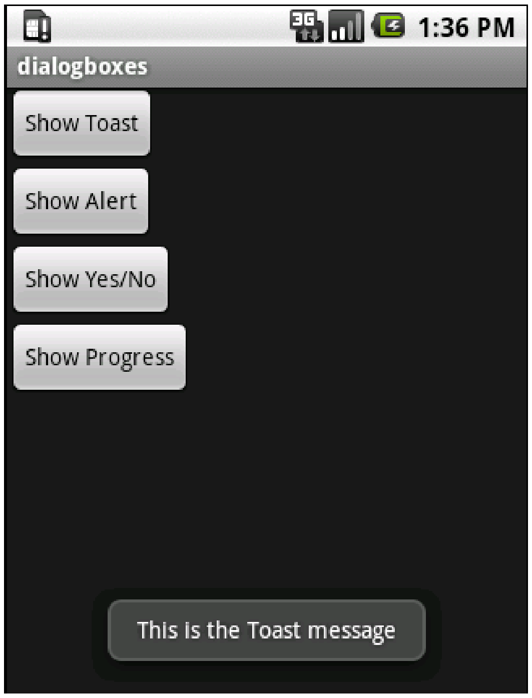
Then there are services which are nothing more than kind of processes that may get started along with the application and then remain turned on even when the application itself is closed. They are most commonly used for data synchronization with a remote server. This way even when the application if turned off, they can show the user some notifications, for instance about a new email waiting in a mailbox.

Every Activity, Widget and Service need to be defined in application manifest.

2.2.2 Notifications

In Android there are basically two different kinds of notifications: Toasts and top screen notifications.

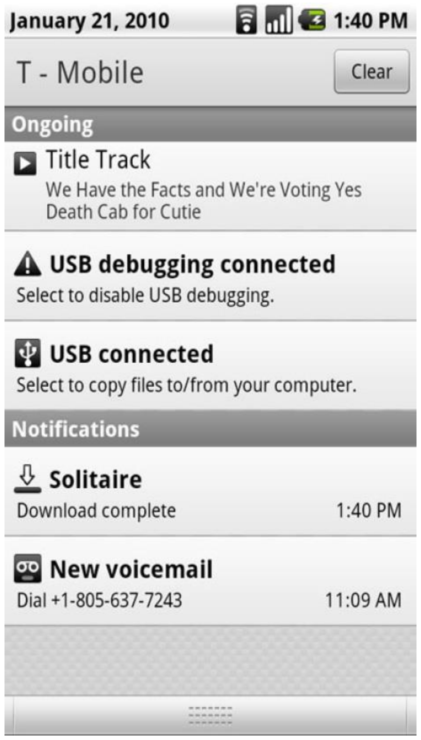
Toasts are usually small pieces of text shown at the bottom of the screen for just a few seconds. Only one toast message can be shown at a time. If the application tries to show two or more toasts at one time, they will be queued and shown one after another, each for the specified amount of time.



This is the Toast message

FIGURE 3. Android: Toast Example

The other notifications can be shown at the top of the screen. These are most often created by services whenever they generate some data worth presenting to the user.


January 21, 2010 1:40 PM

T - Mobile Clear

Ongoing

Title Track
We Have the Facts and We're Voting Yes
Death Cab for Cutie

USB debugging connected
Select to disable USB debugging.

USB connected
Select to copy files to/from your computer.

Notifications

Solitaire
Download complete 1:40 PM

New voicemail
Dial +1-805-637-7243 11:09 AM

FIGURE 4. Android: Top Screen Notifications example

Unlike toasts, top screen notifications can contain multiple different Views that may present multiple kinds of data as well as control the application itself. Also, they may be connected to some Activity. This way when a user clicks one of the notifications, one of the application activities, that would present more data concerning particular notification, may be started.

2.2.3 Unit tests

There are several ways to write unit tests depending on what exactly is to tested. Android documentation advices to use JUnit framework for most basic tests, as Android testing framework is based on it.

JUnit is not an integral part of Android framework nor is it a standard part of Java, therefore it needs to be added to a project manually (some IDEs such as Eclipse can add the proper JUnit JAR automatically, judging from annotations that are used in the code) either by modifying the main project file or, if Maven is used, by adding the following dependency to your POM file:

```
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
</dependency>
```

To write a proper JUnit test, it is important to stick to the following practises:

a) All of the tests should be placed in a separate source directory, preferably called “test”
b) Each test class should be placed in a package that is named the same as the package of the class that it tests
c) Name of a test class should be the same as the name of the class that it tests with additional suffix “Test”
d) Every testing method should be named in the same way as the method that it tests with prefix “test”

This way, if a test for class “src/fi.example.MyClass” is to be created, a file called “test/fi.example.MyClassTest” should be created and if the class “MyClass” contains public method “foo()”, then “MyClassTest” should contain method “testFoo()”.
It is important to stick to these rules, first of all, because the structure of the project is more logical and easier to read for other programmers and also, because many applications for code statistics (such as for instance “Sonar”) require to stick to them in order to calculate test coverage properly.

JUnit framework provides a large variety of annotations that are supposed to help programmers to write precise unit tests fast and easily. The focus of this thesis is on the most basic ones only, which are: @Test, @Before, @After, @BeforeClass and @AfterClass.

Every method that performs a test and should follow the following rules:

a) Should be annotated with @Test
b) Should not be static
c) Should be of type void
d) Should be public
e) Should not take any parameters

This way, if the following method is to be tested

```
public int foo(int a) {
    return a + 1;
}
```

The test method should look like this

```
@Test
public void testFoo() {
    Assert.assertEquals(4,foo(3));
}
```

The Assert class is also a part of JUnit framework. It contains a set of static methods performing simple tests and throwing AssertionException if a test fails. For instance Assert.assertEquals(Object a, Object b) will throw exception if a is not equal to b, which will lead to test failure.

Then there are @Before and @After annotations. They should also be public, non-static and of type void, however, unlike test methods, they do not need to have any specific name. They are used for data preparation of a single test. The method annotated with @Before, will be executed before each test method, and the method annotated with @After will be automatically called right after the return of test method.

Finally there are @BeforeClass and @AfterClass. The methods annotated with these should stick to the following rules:

a) Should be static
b) Should be of type void
c) Should be public
d) Should not take any parameters

```
@BeforeClass
public static void initTest() {

}
```

Unlike methods annotated with @Before or @After, the methods annotated with @BeforeClass and @AfterClass are executed only once during the entire test case. They are usually useful if a programmer wants to initialize some values that will be used

by several test methods, like such as a HSQLDB connection (database often used for unit tests, as it provides multiple database dialects and it is possible to keep the entire database in memory without necessity of creating actual network connection).
Unfortunately JUnit is not enough to test all functionality of the application. For instance, in order to get access to application resources, application context that cannot be acquired manually is still needed. In such a case however it is enough to make the test class inherit AndroidTestCase class (included with Android Test Framework) which provides application Context object.

Tests for your Activities might also be wanted to be created by extending InstrumentationTestCase class. This way programmer can get full control of Activity lifecycle, perform dependency injections (create mock system objects such as Context or Applications and use them to run Activity under test) and simulate user interaction (generate keystrokes and touch events under test).

Finally the Services might need to be tested. In such a case the test case should extend ServiceTestCase (derived from TestCase, which was necessary to inherit in JUnit 3; in JUnit 4 TestCase is still supported but it is not necessary to use it). ServiceTestCase provides programmers with methods like startService() or bindService(). This way service is started only when a programmer explicitly requires it which allows to create initial system mock objects.

2.3 Windows Phone

Windows Phone is a mobile operating system developed by Microsoft Corporation. The version 7 was released in 2010 and it was a direct successor of Windows Mobile.
The newest available version of Windows Phone is 8, released in December 2012.

2.3.1 Basics

The equivalent of Android Activities in Windows Phone are Pages. Every page always contains 2 files: XAML that contains the layout of the Page and C# class that contains the logic of the Page. Unlike in case of Android, the page does not have to be defined in any additional manifest.

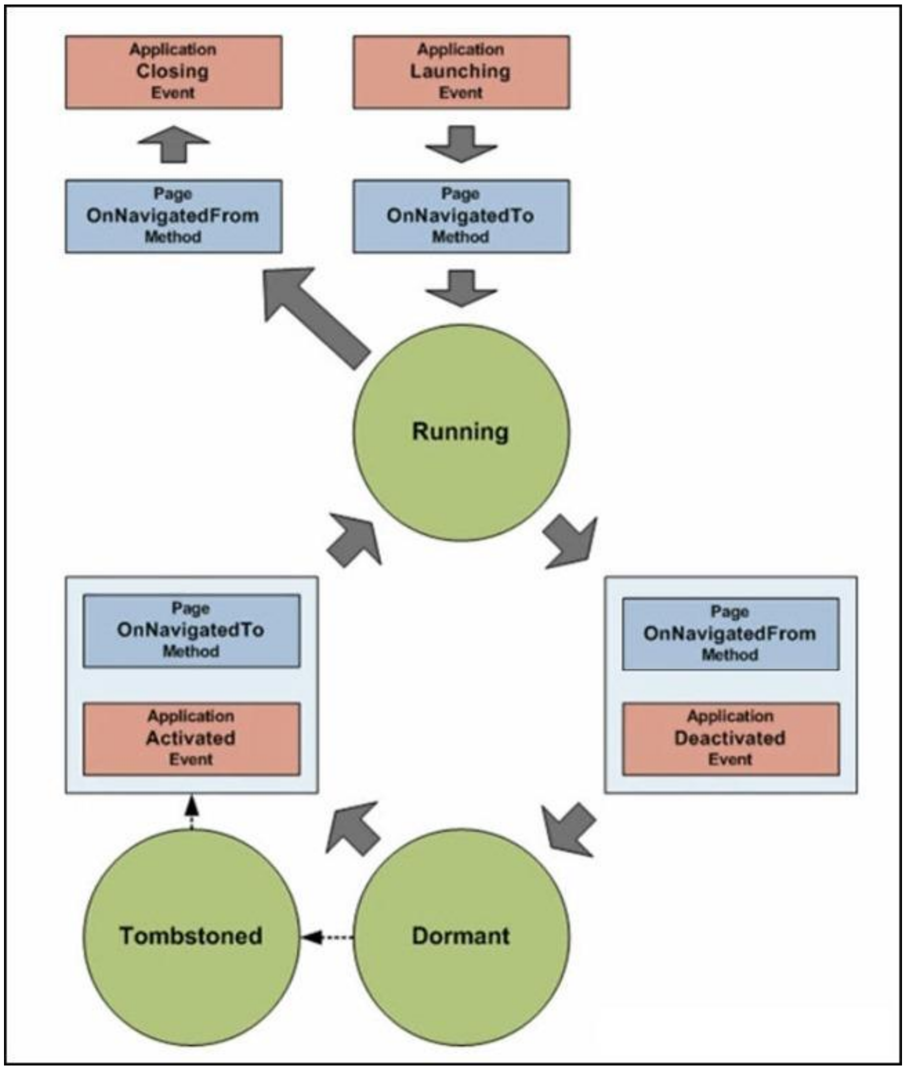

Application Closing Event
Application Launching Event
Page OnNavigatedFrom Method
Page OnNavigatedTo Method
Running
Page OnNavigatedTo Method
Application Activated Event
Page OnNavigatedFrom Method
Application Deactivated Event
Tombstoned
Dormant

FIGURE 5. Windows Phone: Page Lifecycle

2.3.2 Notifications

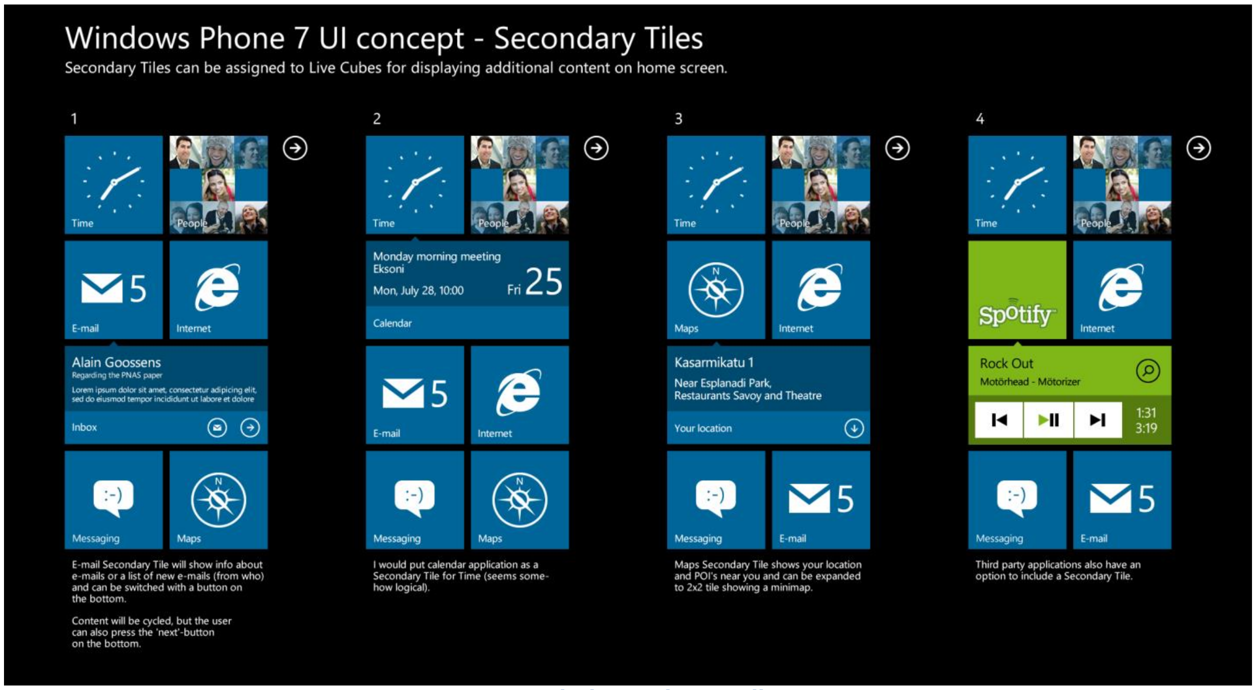
Windows Phone provides two basic types of notifications: tile notifications and toasts. Every application has at least one tile (but may contain multiple of tiles as well). Each tile may reference to a different Page of the application.

A tile except for standard icon may contain a number. It is used to represent for instance a number of unread text messages, or missed calls. Moreover tails may periodically change their content and instead of presenting the application icon, show either some text or image.

In Windows Phone 7.1 there are two kinds of tails: small ones and wide ones. Unfortunately the wide tiles are reserved for system applications only. It is supposed to be changed only in Windows 8.

There are several ways of updating tile content.
A tile can be updated straight from application Page, but then you have to remember that application must be active to perform such operation. In most cases this kind of solution is just useless.

The other way is to create Background Agent. Background Agents can be started on system start up and may perform many different operations (including tile update) even when the application that they are part of, is still not on. It is however sometimes troublesome that agents are executed by operating system only periodically and their work needs to be finished within a certain amount of time. If the agent does not make it to finish the operation in time, it gets automatically killed by the operating system.


Windows Phone 7 UI concept - Secondary Tiles

Secondary Tiles can be assigned to Live Cubes for displaying additional content on home screen.

E-mail Secondary Tile will show info about e-mails or a list of new e-mails (from who) and can be switched with a button on the bottom.

Content will be cycled, but the user can also press the 'next'-button on the bottom.

I would put calendar application as a Secondary Tile for Time (seems somehow logical).

Maps Secondary Tile shows your location and POI's near you and can be expanded to 2x2 tile showing a minimap.

Third party applications also have an option to include a Secondary Tile.

FIGURE 6. Windows Phone: Tiles

The other type of notifications are Toasts. Those are the top screen notifications, that can also activate a certain page once they are clicked by a user.

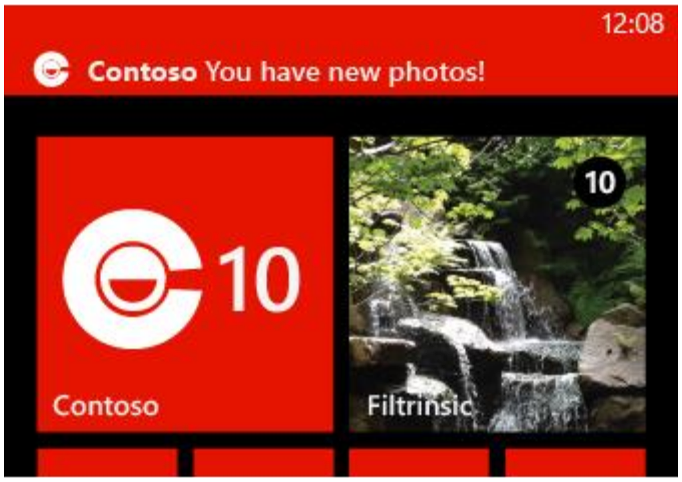


FIGURE 7. Windows Phone: Toast

2.3.3 Framework

Windows Phone applications use several different Microsoft technologies. Most basic ones are obviously C# and .Net Framework as the whole logic of the application is created with them, but then to create a layout, Silverlight is also required.

The problem is, that unlike in case of Android, in Windows Phone there is no access to all of the features of the programming language that are supposed to be used, as the system has only .Net Core implemented. That often causes a great deal of confusion. First, when the user faces the problem, he may think that it is easy to implement as he already knows C# and .Net quite well and then the user slowly starts noticing that most basic features he uses in every application are unavailable in the Windows Phone application. Many operations in Windows Phone need to be executed asynchronously and yet, the programmer does not have access to semaphores, which are the most basic synchronization elements.

2.3.4 Unit tests

Unit tests in C# are created in a slightly different way than in Java. First of all, unlike in Java, in C# project multiple source directories cannot be created, but instead multi-project solution may be used. So what programmer needs to do, is first create a new project called in the same way as project that contains classes that he wants to test, but with a suffix “Tests”.

Within new project programmer needs to create a class and call it in the same way as the class that he wants to test, but including suffix “Tests”. The class should contain attribute [TestClass].

Then, methods in test case should be named exactly the same way as the method that needs to be tested is called. Each test method should contain attribute [Test].

This means, that if the following class is given

```
namespace Project
{
    public class MyClass
    {
        public int Foo(int a)
        {
            return a + 1;
        }
    }
}
```

the test case for it should look like this:

```
namespace ProjectTests
{
    [TestClass]
    public class MyClassTests
    {

        private static MyClass myClass;

        [ClassInitialize]
        public static void Initialize(TestContext context)
        {
            context.WriteLine("Initialization of test case");
            myClass = new MyClass();
        }

        [TestMethod]
        public void Foo()
        {
            Assert.AreEqual(2, myClass.Foo(1));
        }

        [ClassCleanup]
        public static void CleanUp()
        {

        }
    }
}
```

Note that attributes [ClassInitialize] and [ClassCleanup] are equivalent to @BeforeClass and @AfterClass annotations in JUnit. Additionally, the initialization method takes one parameter of type TestContext. The parameter allows printing out text into test log, performing time measurements for the test in a very easy way as well as provides data about used test environment.

2.4 Technology differences

Although one could think that, since both Android and Windows Phone operating systems were created for a similar purpose, they both should also be similar from programmers' point of view too, they actually are very different one from another.

2.4.1 Touch event consumption

In case of applications like PCRemote, where custom widgets such as touchpad are used, it is important to process all the touch events manually. It is especially important in case of Windows Phone client where a touchpad is placed on a pivot page. If the touch event does not get consumed in a proper way there, users might accidentally switch pages, instead of just moving the cursor.

In Android and Windows Phone touch event consumption is solved in two different ways.

In Android, even without any documentation, the following method may be easily spotted:

boolean onTouchEvent(MotionEvent e)

Even without reading any documentation programmers can easily guess that the method's parameter contains a description of the touch event and then the method should return Boolean value telling whether an event was consumed or not, as the method is pretty self-explanatory.

In the case of Windows Phone it is much more complicated, as there are several methods that require to be implemented in this case to perform the same task:

```
- void MouseMove(object sender, MouseEventArgs e)
- void ManipulationComplete (object sender,
ManipulationCompletedEventArgs e)
- void ManipulationDelta(object sender,
ManipulationDeltaEventArgs e)
- void ManipulationStarted(object sender,
ManipulationStartedEventArgs e)
```

Each of them takes parameters of different types what may cause a great deal of confusion. Also, instead of making this method return the value stating whether the event was consumed, event property Handled needs to be set to true.

```
e.Handled = true;
```

2.4.2 Energy management

Energy management policies in Android and Windows Phone are quite different.
In case of Android the save battery mode is optional. Users may choose to dim the screen or slower the processor clock. This means that energy management policy does not really have any effect on how the applications are executed in the system, however, programmers developing applications for Windows Phone must always remember about energy management policies of this operating system.

In Windows Phone, once the screen gets locked, all applications are paused. Only some background agents have execution rights, but also not all of them. It is quite troublesome even during debugging. When the device screen is locked, the system agent responsible for USB communication is paused and therefore further debugging with Visual Studio becomes impossible.

Also, when the screen gets locked, all network connections get closed. Especially in case of PCRemote, this is really troublesome, as there is really no point in using any background agents to communicate with the server and sometimes detecting if the application is still connected to the server might be difficult.

All in all, Windows Phone energy management policy from the user's point of view may seem to be better, however, from programmer's point of view it is much more difficult to cope with.

2.4.3 Templates

In Android whenever a list of some elements should be presented, there is usually a need to create a container for them somewhere in the layout and then manually add the elements from code. Then whenever user wants to modify one of the records, the layout must be updated manually.

In Windows Phone (and Silverlight) however there is access to templates and observable collections.

Every pattern is first described in XAML file (may be the same as the rest of the layout). Programmers can define names of collections used for building the list already in the template.

```
<ListBox x:Name="listbox" Grid.Row="1" Grid.Column="0">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel>
                <toolkit:ContextMenuService.ContextMenu>
                    <toolkit:ContextMenu>
                        <toolkit:MenuItem Header="Remove" Click="MenuItem_Click_1" />
                        <toolkit:MenuItem Header="Edit" Click="MenuItem_Click" />
                    </toolkit:ContextMenu>
                </toolkit:ContextMenuService.ContextMenu>
                <Button Click="Button_Click" Width="460">
                    <Button.Content>
                        <Grid x:Name="LayoutRoot" Background="Transparent">
                            <Grid.RowDefinitions>
                                <RowDefinition Height="Auto" />
                                <RowDefinition Height="Auto" />
                                <RowDefinition Height="Auto" />
                            </Grid.RowDefinitions>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="Auto" />
                                <ColumnDefinition Width="400" />
                            </Grid.ColumnDefinitions>
                            <TextBlock Text="Name: " Grid.Row="0" Grid.Column ="0"/>
                            <TextBlock Text="{Binding Name}" Grid.Row="0" Grid.Column="1" Name="Name"/>
                            <TextBlock Text="Address: " Grid.Row="1" Grid.Column="0" />
                            <TextBlock Text="{Binding Address}" Grid.Row="1" Grid.Column="1" Name="Address" />
                            <TextBlock Text="Address: " Grid.Row="2" Grid.Column="0"/>
                            <TextBlock Text="{Binding Port}" Grid.Row="2" Grid.Column="1" Name="Port"/>
                        </Grid>
                    </Button.Content>
                </Button>
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Now there is a need to have in the Page class an observable collection called “listbox” (as this is the name that x:Name property indicates). Since the collection is observable, whenever some object is added or deleted from it, it will generate an appropriate event and the view will get automatically updated.

Also, in this case custom multiple-property elements are used. If the view is to get updated automatically whenever a single object gets modified by a user, it is enough that the interface INotifyPropertyChanged gets implemented and event PropertyChanged in each of the setters of the element class gets generated.

```
public event PropertyChangedEventHandler PropertyChanged;

private string name;

public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs("Name"));
    }
}
```

2.5 Alternative technologies

Nowadays programmers are not really forced to create native applications designed for one specific mobile operating system only. There are many technologies to choose from, allowing the programmers to code their applications only once and run it on several mobile operating systems. For instance applications written with HTML5 and JavaScript gain more and more on popularity these days as every device can run internet site and every mobile framework contains a web component that can present such a site. Then there are JavaScript frameworks like for instance jQeury Mobile, designed specifically for portable mobile application development. Unfortunately capabilities of such applications are still very limited, as they cannot access most of the device specific functionality such as vibrations, geolocalisation, system specific notification, camera, system configuration etc. They can only present data, download something and store application configuration, which means, that they cannot be used for most applications.

3 IMPLEMENTATION OF THE PROJECT

One of the first steps that every software developer should take before starting the implementation of the application is creating a detailed design of the project, where all the details about all of the features are stated. Such design usually contains some user stories, where the general functionality of the application is presented. Based on user stories it is possible to create functional and non-functional requirements as well as use cases, that are necessary to create the overall class and data flow diagrams that are required for programmers to write the efficient, easy to read code faster.

3.1 User stories

User stories are usually sets of very overall ideas for what users might want to have in the applications. They are usually the base for creating more advanced specification for the application.

3.1.1 Server user stories

1 As a user of a server, I want to access all its features through a tray icon.
2 As a user I want the server to support at least all basic functions of controlling my computer, such as full control over the cursor, just the way I control it with touchpad, control over keyboard and basic media keys (play, previous, next).
3 As a user of a server I want to be able to set it up, so that it refused connections from certain clients.
4 As a user I want to server to be able to provide me instructions of how I should use it and its clients.
5 As a user I want to be able to choose whether my server can be detected automatically by a client.
6 As a user of a server I want to be able to restrict access to it with a password.

3.1.2 Android client user stories

1 As a user I want to be able to add information about a server manually.
2 As a user I want to be able to control my cursor as if my device was a real touchpad.
3 As a user I want to be able to control my keyboard via device with client installed.
4 As a user I want to be able to control my media buttons: play, next, previous, volume up, volume down.
5 As a user I want to be able to choose sensitivity of touchpad.
6 As a user I want to be able to choose sensitivity of scrolls.
7 As a user I want the application to support multiple languages.
8 As a user I want to be notified when the server rejects my attempt to connect.
9 As a user I want to be able to enter password when it is required to start the connection with the server.
10 As a user I want to be able to download files from my PC to my mobile device.

3.1.3 Windows Phone client user stories

1 As a user I want to be able to add information about a server manually.
2 As a user I want to be able to control my cursor as if my device was a real touchpad.
3 As a user I want to be able to control my keyboard via device with client installed.
4 As a user I want to be able to control my media buttons: play, next, previous, volume up, volume down.

5 As a user I want to be able to choose sensitivity of touchpad.
6 As a user I want to be able to choose sensitivity of scrolls.
7 As a user I want the application to support multiple languages.

3.2 Functional requirements

Functional requirements are often based on user stories detailed lists of features that are going to be implemented in the application. They tend to be really helpful for further application design.

3.2.1 Android client

1 It is possible to use mobile device like a touchpad to control cursor and scrolls
2 It is possible to use mobile device to control the keyboard (all keys that can be found on English keyboard)
3 It is possible to add and store data about a new server
4 It is possible to connect to the server
5 It is possible to choose sensitivity of a cursor
6 It is possible to define the time of a long click (how long the control needs to be pressed to call it a long click)
7 It is possible to define sensitivity of a scroll
8 It is possible to control media keys
9 It is possible to download files from computer to mobile device

3.2.2 Windows client

1 It is possible to use mobile device like a touchpad to control cursor and scrolls
2 It is possible to use mobile device to control the keyboard (all keys that can be found on English keyboard)
3 It is possible to add and store data about a new server
4 It is possible to connect to the server
5 It is possible to choose sensitivity of a cursor
6 It is possible to define the time of a long click (how long the control needs to be pressed to call it a long click)

3.2.3 Server

1 Server notifies of each new client connection
2 Server notifies of each client disconnection
3 Server processes all client requests concerning cursor movement or key press
4 Has a tray icon that allows to control the application
5 It is possible to change default port for listening
6 It has an option for closing the application from tray icon menu

7 It can restrict access with password for Android clients

3.3 Non-functional requirements

Non-functional requirements are requirements for the environment that the application will be run in, such as operating system, or details concerning the hardware.

3.3.1 Clients

1. Wi-Fi connection to the internet
2. A mobile device running either Windows Phone 7 or Android (at least version 2.3) depending on a version of client application.

3.3.2 Server

1. Windows Vista / 7 / 8 operating system
2. Wireless internet connection

3.4 Diagrams

During the development process it is good to create some basic diagrams representing the way the application should work. It does not take a long time to design those, but still, it enhances the work of a group of programmers, as this way they know from the very beginning how to integrate each part of the application that will be worked on separately, as well as it largely increases the quality of the final project.

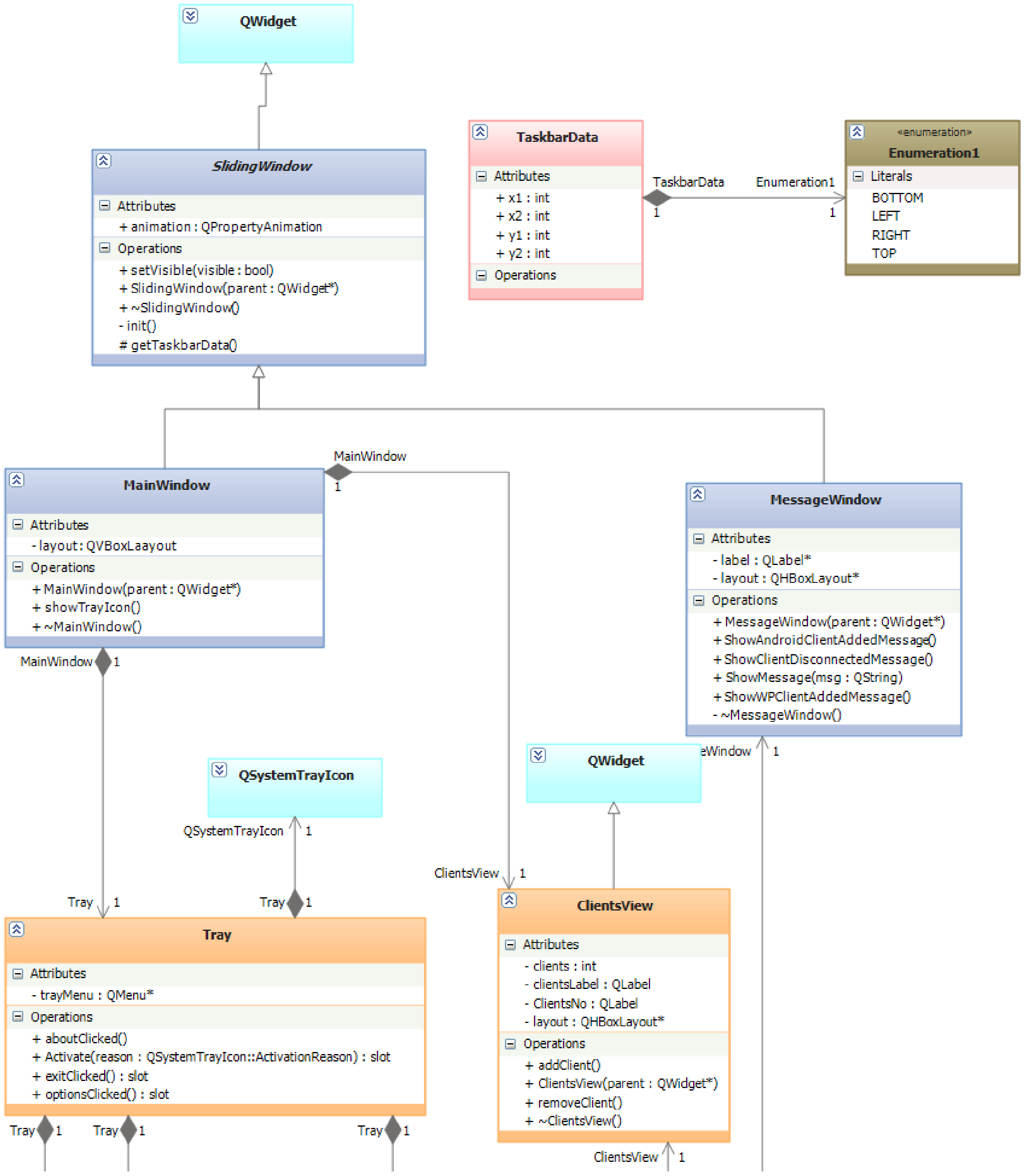
Among the UML diagrams that are most often used for application design are class diagrams, use case diagrams and sequence diagrams.

Class diagrams represent the overall structure of the application. They show classes, that the application will contain, as well as relationships between them and the most crucial methods that will be included into classes.

Sequence diagrams represent the order, that the most crucial functions will be called in. Use case diagrams are often used to represent the relationship between views in the user interface of the application. They can also contain information about access rights for multiple types of users.

3.4.1 Server class diagram

The main part of the whole application is Tray class. Even though it is still just a part of MainWindow, it is visible all the time, while MainWindow is visible only after Tray is activated (tray icon is clicked by a user). Tray contains a set of AbstractCommunicators, although in this example only NetworkCommunicator (inherits AbstractCommunicator) that allows communication with the clients via TCP (general events) and UDP (mouse movement event) is used. This is also Tray that calls the Listen method in each AbstractCommunicator. Then Tray also contains the menu (visible after clicking tray icon with right mouse button) for managing the whole application.



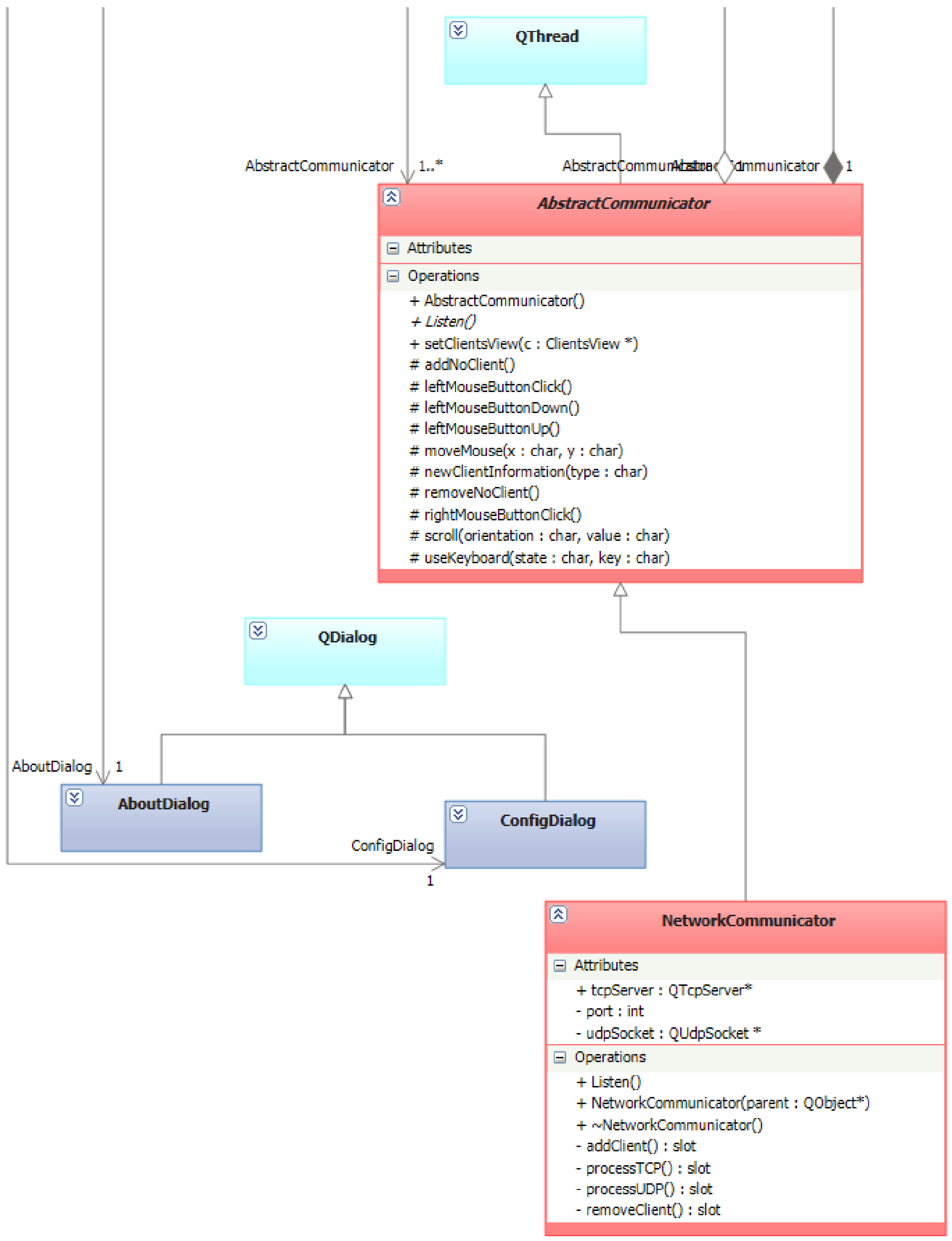


FIGURE 8. Server: Class diagram

3.4.2 Server use case diagram

As a matter of fact, ClientsView is nothing more than just a Widget consisting of two labels: one that says “Number of clients” and the other one that actually represents the number.

From tray icon users can simply go to MainWindow just by performing left click on it, as well as they can open Configuration window or close the application if they right click the tray icon and choose the right option from the popup menu.

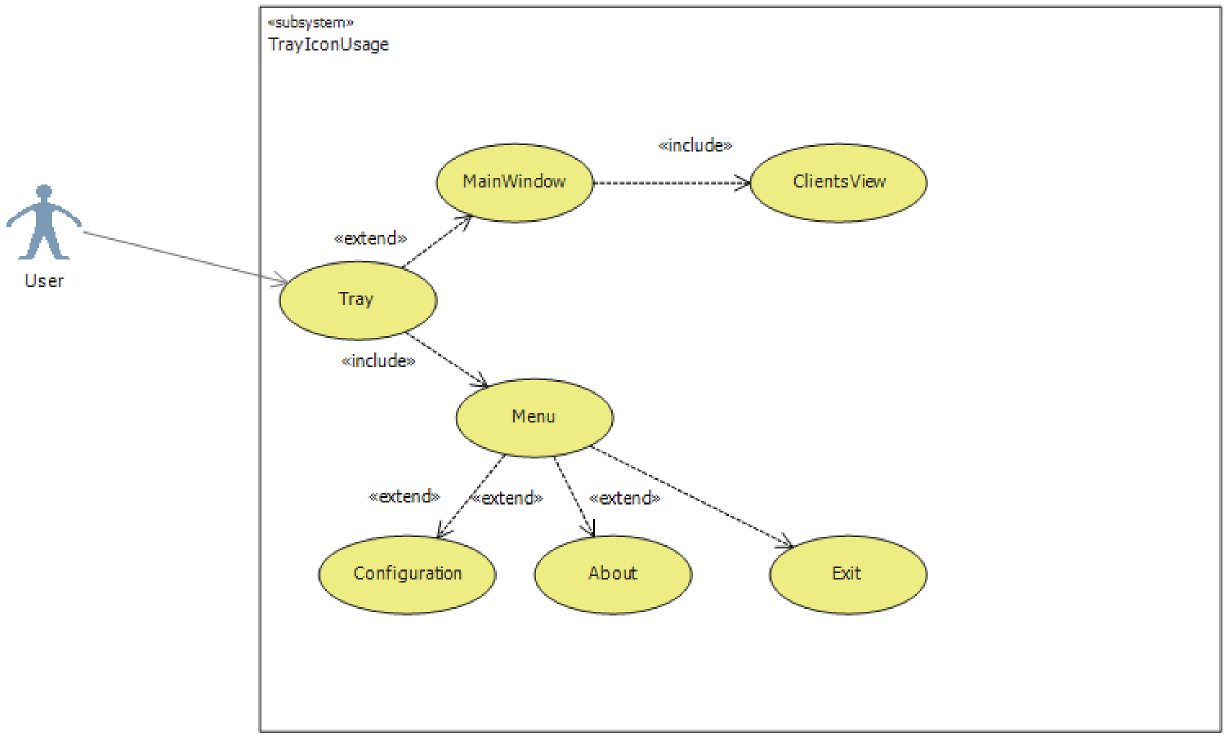


FIGURE 9. Server: use case diagram

3.4.3 Server sequence diagram

The Tray class initializes listening for incoming connections. Once the client connects to the server, a separate thread is created and conversation between client and server starts.

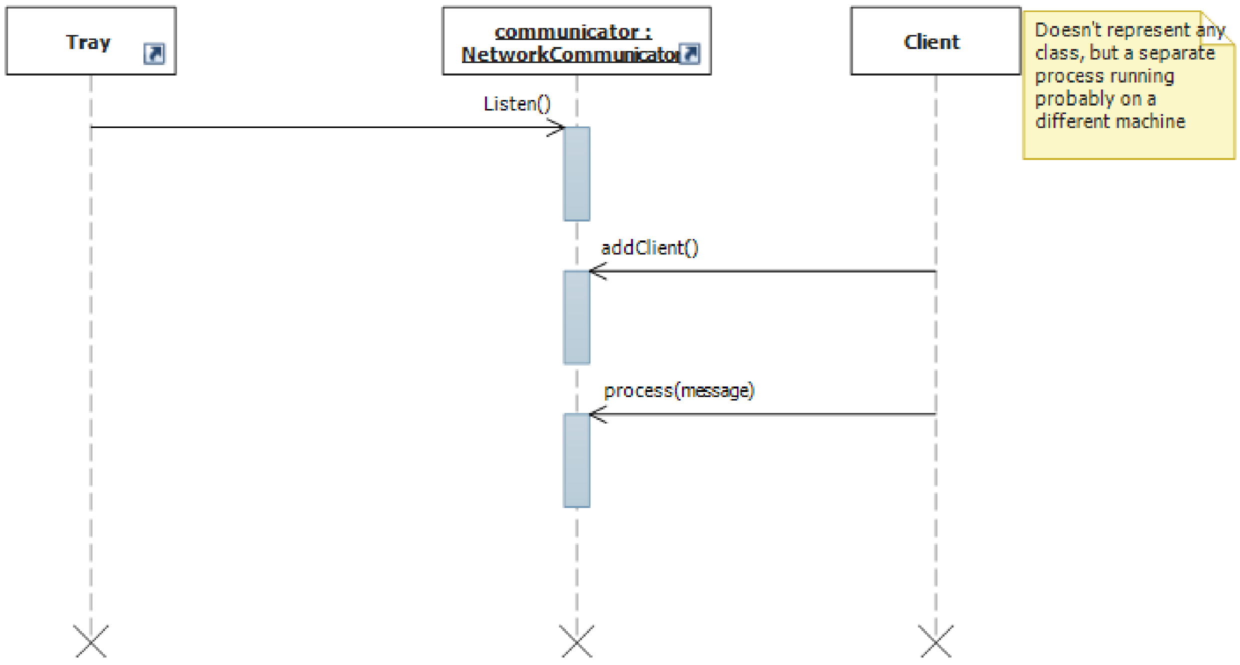

Tray
communicator : NetworkCommunicator
Client
Doesn't represent any class, but a separate process running probably on a different machine
Listen()
addClient()
process(message)

FIGURE 10. Server: sequence diagram

3.4.4 Android client class diagrams

All Activities are divided basically into two types: the ones that are supposed to present only information stored locally on the mobile device and the ones that are used to communicate with the server. The second ones instead of inheriting the class Activity, inherit ControllerActivity that provides them with all methods required for communication with a server as well as the navigation between them all.

Optionally Activity can implement interface ContextResponseListener. This will allow the Activity to receive and process response from the server.

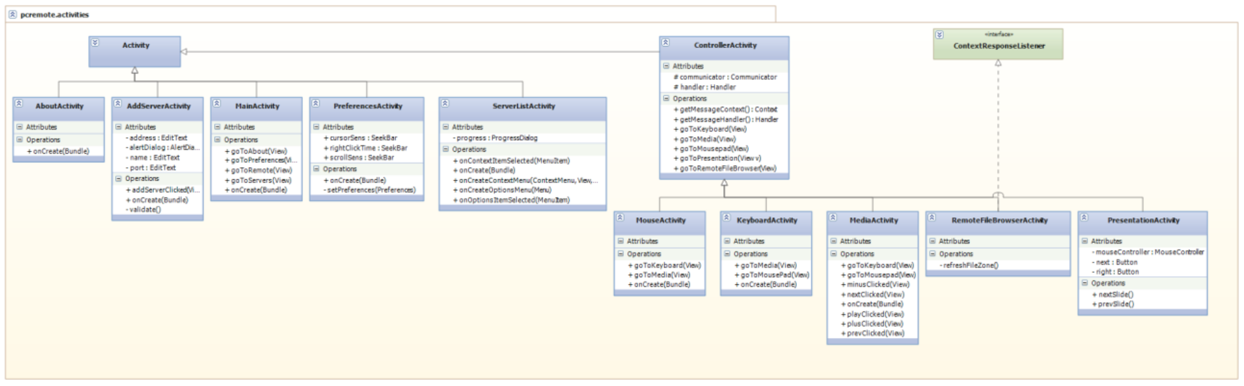


FIGURE 11. PCRemote Android: Activities

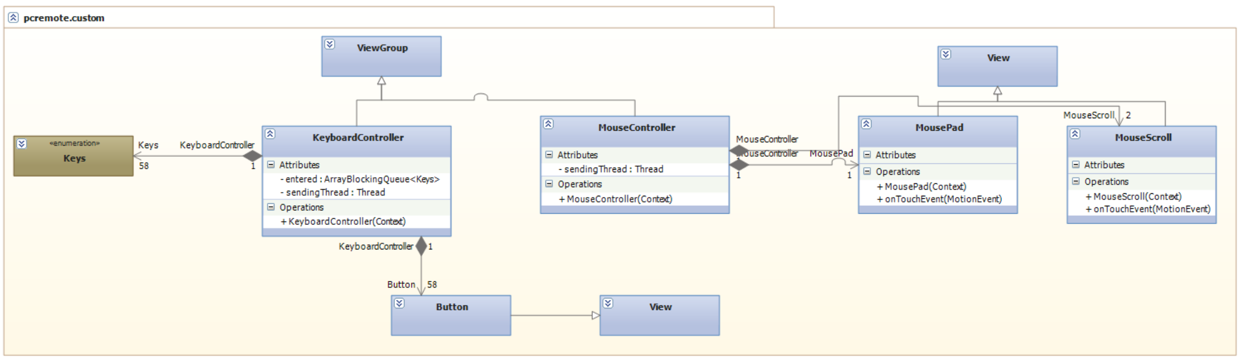


FIGURE 12. PCRemote Android: Custom views

Package pcremote.custom contains custom Views that generate events that will later be transformed into network message and sent to the server. One of them is for instance MouseController that may generate events like CoursorMovementMessage or LeftClickMessage.

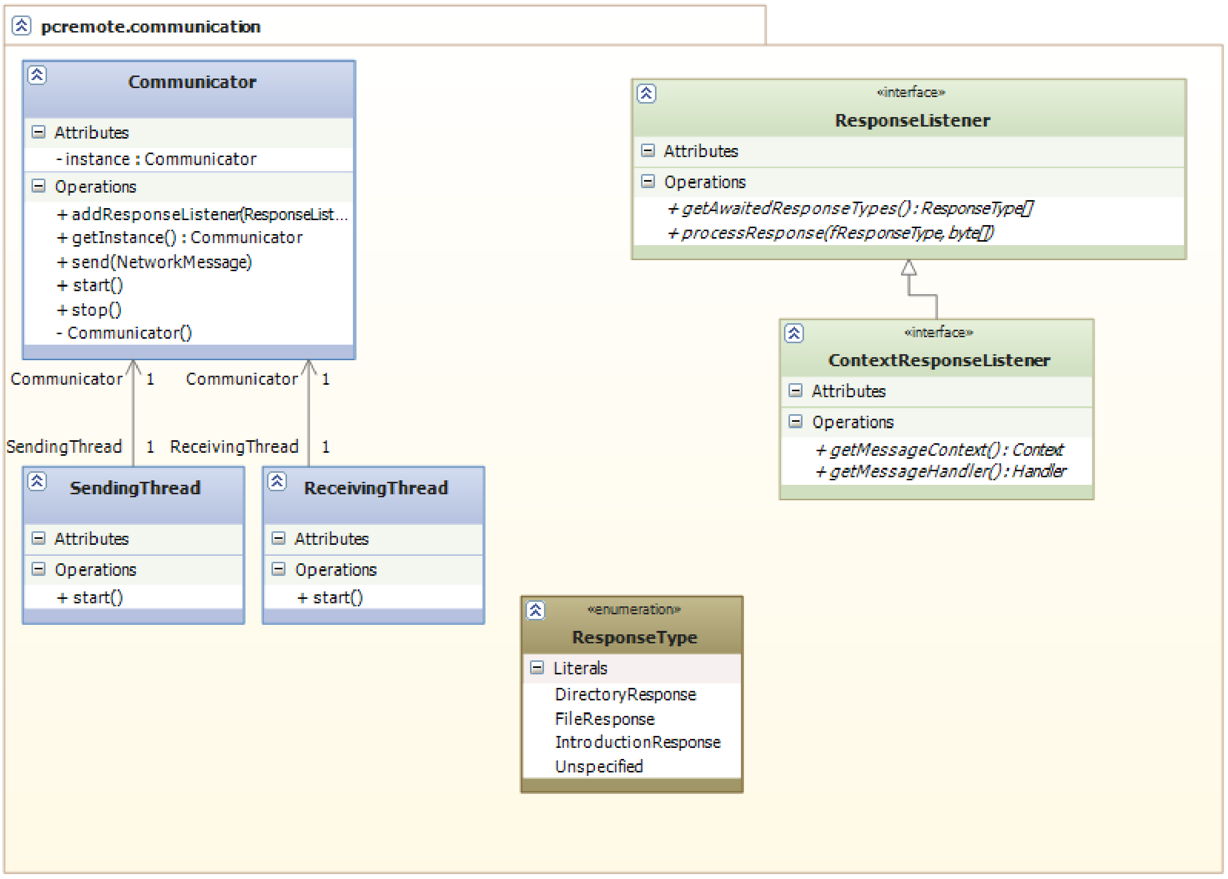


FIGURE 13. PCRemote Android: Communication

No part of the application should access SendingThread or ReceivingThread directly. Rather than that, they should use Communicator class that manages those two, providing easy to use programming interface.

Communicator enables to send messages, as well as register objects that will listen to server responses.

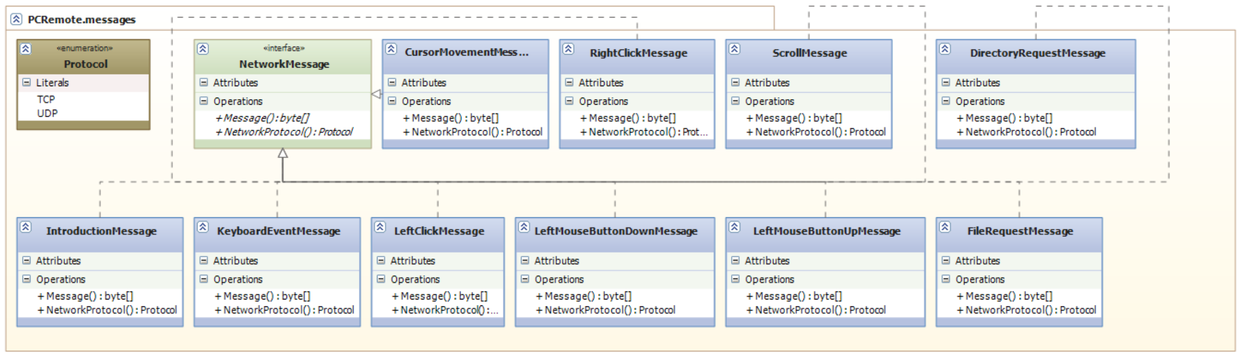


FIGURE 14. PCRemote Android: Network messages

All network messages implement interface NetworkMessage as they all need to contain properties returning the proper protocol type (TCP or UDP) that will be used for sending as well as the message represented by set of bytes itself.

There is a separate network message class for each event and data request.

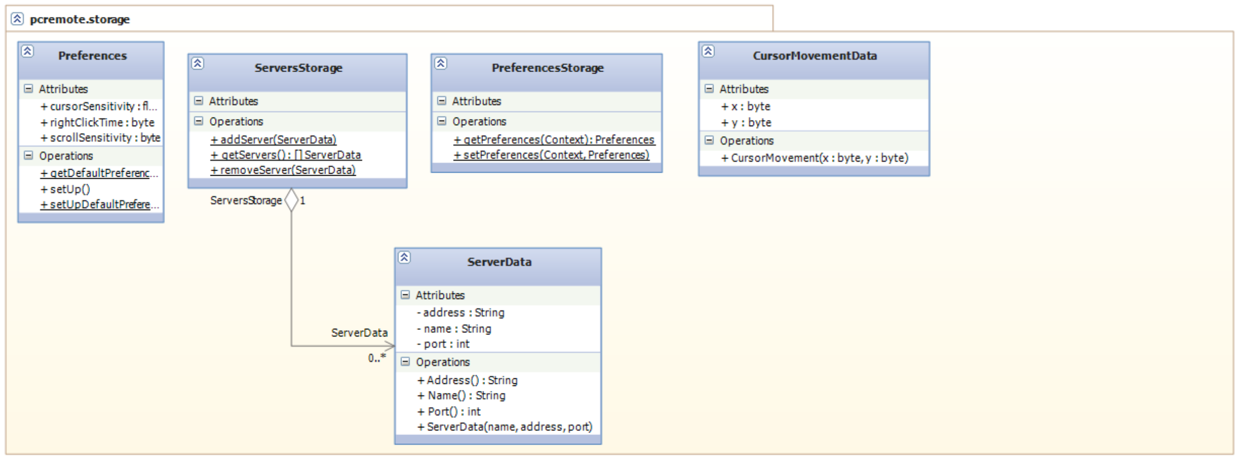


FIGURE 15. PCRemote Android: Storage

3.4.5 Android client use case diagram

The first Activity the users can access right after starting the application is MouseActivity as it is the most often used. From that one, the users can easily access any other activity for communication with server.

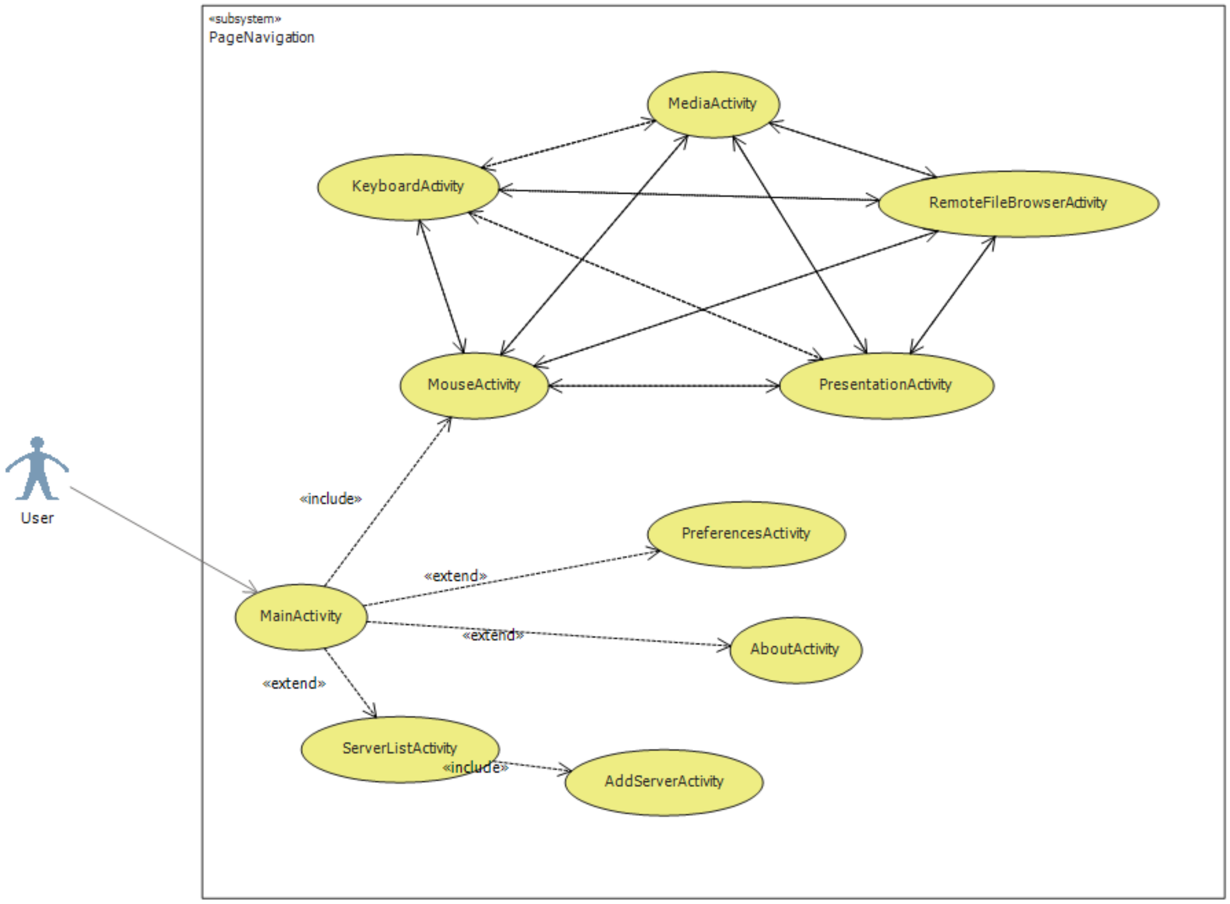


FIGURE 16. PCRemote Android: Use case diagram

3.4.6 Windows Phone client class diagrams

In case of Windows Phone, there are fewer network messages, as there are fewer functional requirements for this client. In this case, there are only messages representing controller events.

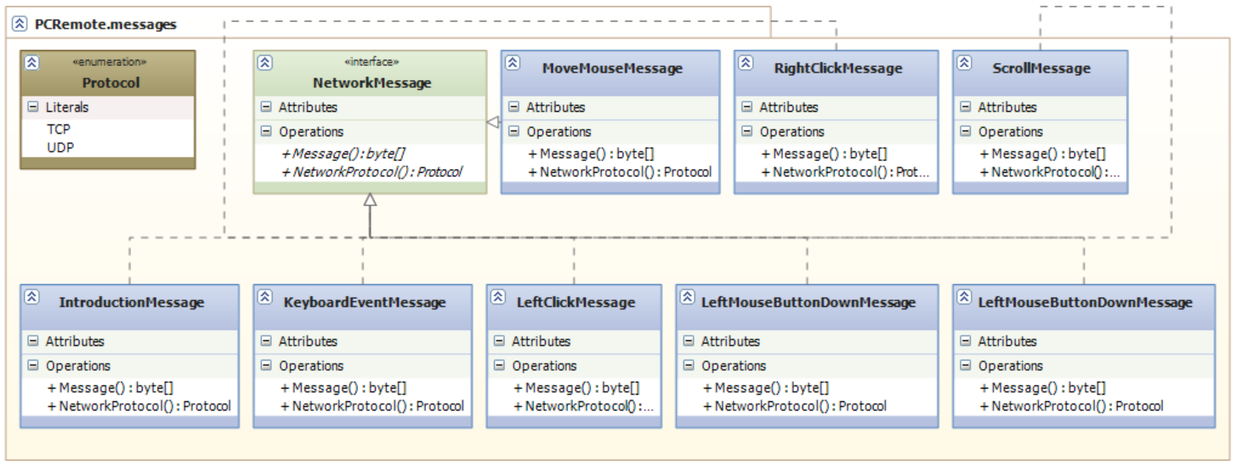


FIGURE17. Windows Phone: messages

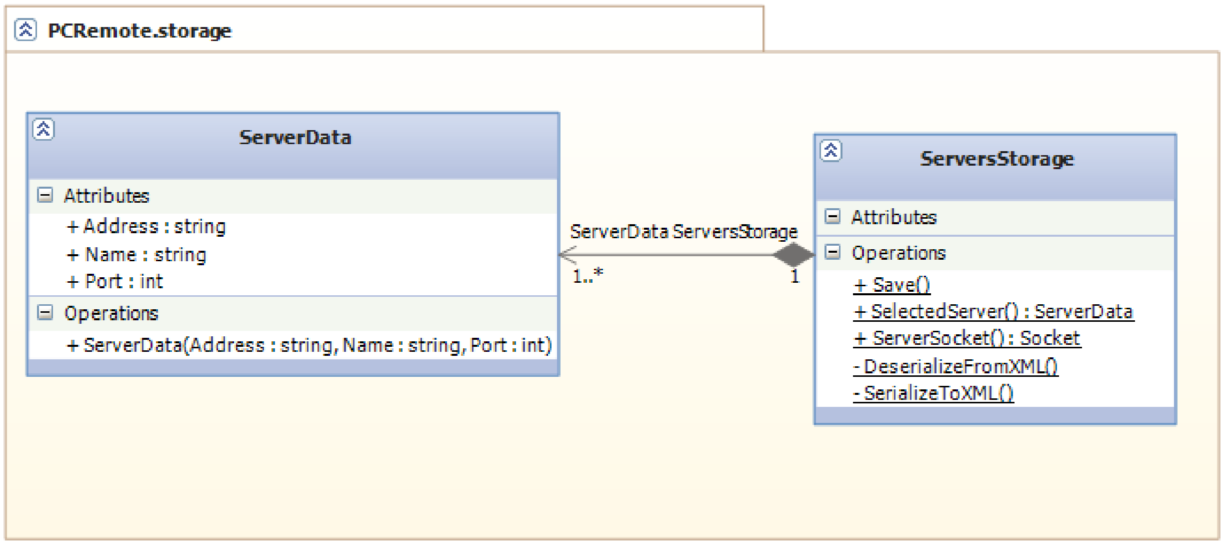


FIGURE 18. Windows Phone: data storage

The data contained and generated by ServersStorage are common for all classes across the application, therefore it is much easier to make the entire class static, rather than pass a reference to it to all the objects used in the application.

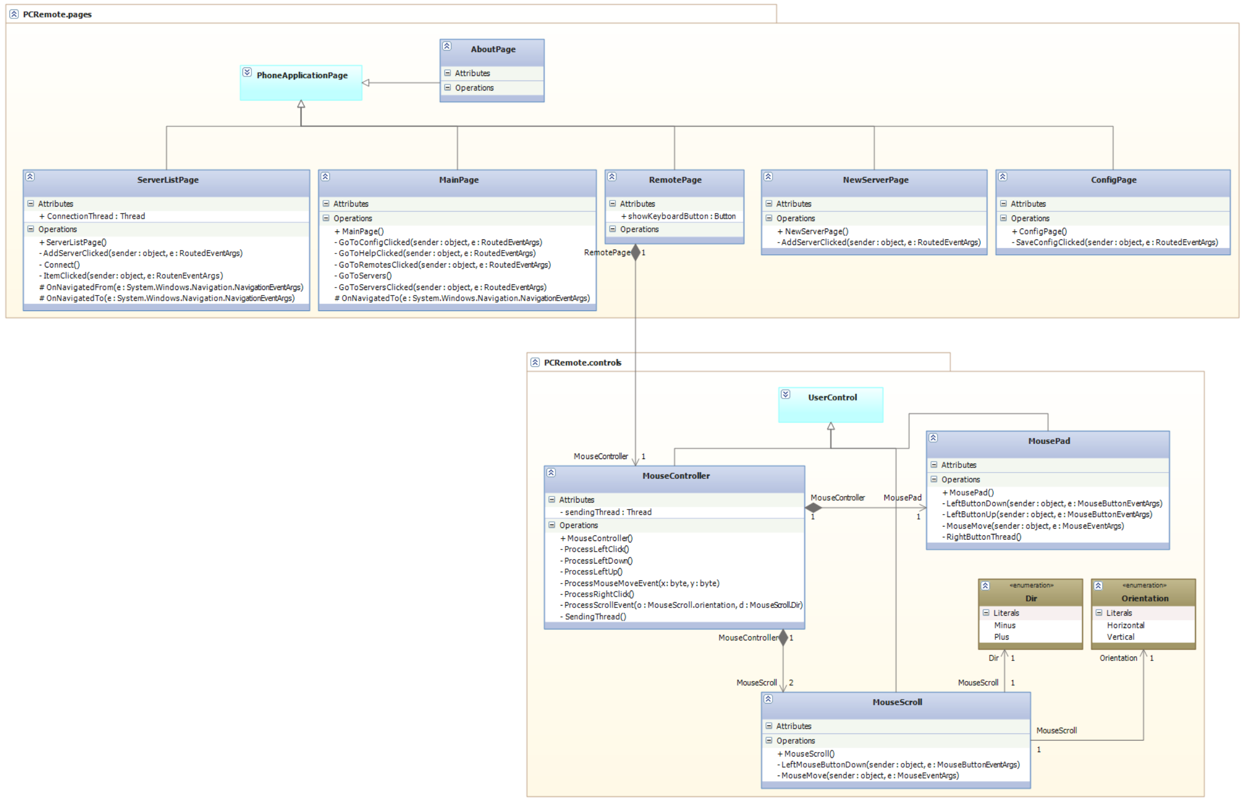

PCRemote.pages
AboutPage
Attributes
Operations
PhoneApplicationPage
ServerListPage
Attributes
+ ConnectionThread : Thread
Operations
+ ServerListPage()
- AddServerClicked(sender : object, e : RoutedEventArgs)
- Connect()
- ItemClicked(sender: object, e: RoutenEventArgs)
# OnNavigatedFrom(e : System.Windows.Navigation.NavigationEventArgs)
# OnNavigatedTo(e : System.Windows.Navigation.NavigationEventArgs)
MainPage
Attributes
Operations
+ MainPage()
- GoToConfigClicked(sender : object, e : RoutedEventArgs)
- GoToHelpClicked(sender : object, e : RoutedEventArgs)
- GoToRemotesClicked(sender : object, e : RoutedEventArgs)
- GoToServers()
- GoToServersClicked(sender : object, e : RoutedEventArgs)
# OnNavigatedTo(e : System.Windows.Navigation.NavigationEventArgs)
RemotePage
Attributes
+ showKeyboardButton : Button
Operations
RemotePage 1
NewServerPage
Attributes
Operations
+ NewServerPage()
- AddServerClicked(sender : object, e : RoutedEventArgs)
ConfigPage
Attributes
Operations
+ ConfigPage()
- SaveConfigClicked(sender : object, e : RoutedEventArgs)
PCRemote.controls
UserControl
MouseController 1
MouseController
Attributes
- sendingThread : Thread
Operations
+ MouseController()
- ProcessLeftClick()
- ProcessLeftDown()
- ProcessLeftUp()
- ProcessMouseMoveEvent(x: byte, y : byte)
- ProcessRightClick()
- ProcessScrollEvent(o : MouseScroll.orientation, d : MouseScroll.Dir)
- SendingThread()
MouseController
MousePad
1
1
MousePad
Attributes
Operations
+ MousePad()
- LeftButtonDown(sender : object, e : MouseButtonEventArgs)
- LeftButtonUp(sender : object, e : MouseButtonEventArgs)
- MouseMove(sender : object, e : MouseEventArgs)
- RightButtonThread()
«enumeration»
Dir
Literals
Minus
Plus
«enumeration»
Orientation
Literals
Horizontal
Vertical
MouseController 1
MouseScroll 2
Dir 1
MouseScroll 1
Orientation 1
MouseScroll
Attributes
Operations
+ MouseScroll()
- LeftMouseButtonDown(sender : object, e : MouseButtonEventArgs)
- MouseMove(sender : object, e : MouseEventArgs)
MouseScroll
1

FIGURE 19. Windows Phone: Pages

Mouse controller is a UserControl that contains two other user controls: MousePad and MouseScroll.

Each time MousePad or MouseScroll recognize a gesture, they raise an event that is captured later on by MouseController. Then depending on the event MouseController generates a proper NetworkMessage and puts it into a messageQueue. Within MouseController there is a separate thread working in the background, that continuously reads the messageQueue and sends data to the server.

3.4.7 Windows Phone client use case diagram

Unlike Android client, Windows Phone client contains only one page that will contain all of the controllers. Remote Page in this case is a Pivot Page. This way it is easy to fit all of the Controls on one page only.

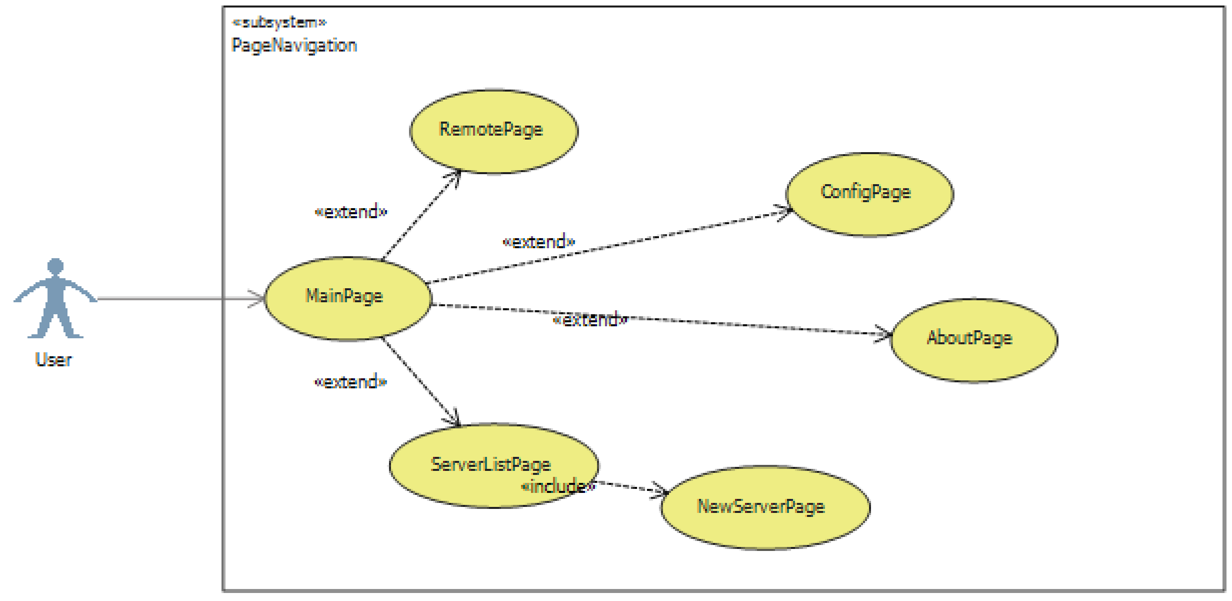


FIGURE 20. Windows Phone: General use case diagram

3.4.8 Windows Phone client sequence diagram

Whenever user slides over touchpad, a proper event is generated, that is passed to MouseController, that contains MousePad. Then MouseController generates an appropriate NetworkMessage and puts it into message queue, where it will be eventually read from by the sending thread and sent to the server.

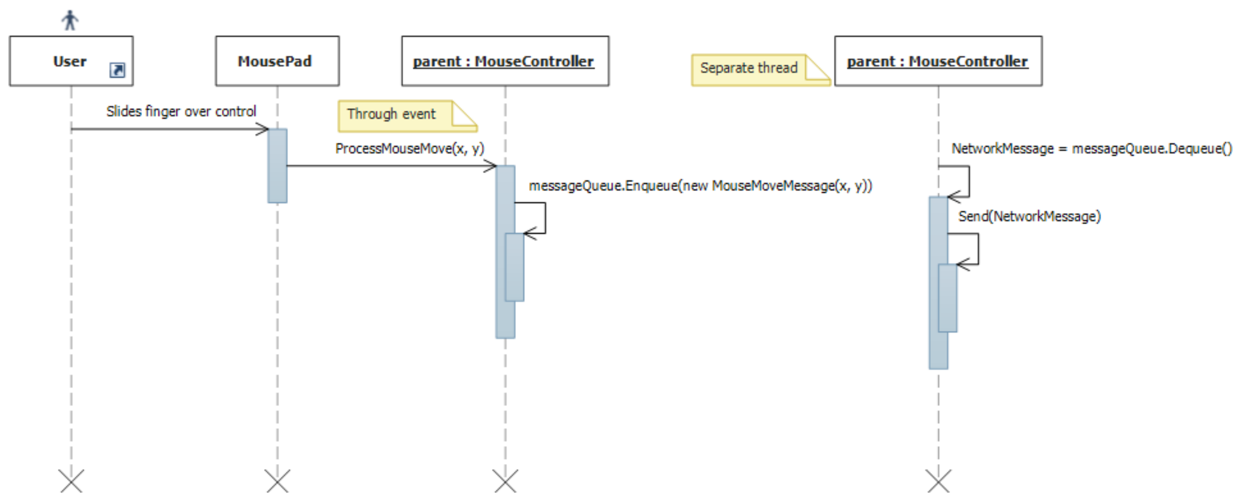


FIGURE 21. Windows Phone: Tiles

3.5 Layout design

It is important to create a general design of each view that will be shown in the application. This way programmers instead of spending a lot of time trying to think how the application should look like, can focus on programming the actual functionality only. This also enables to reuse similar user interface elements across the application, what obviously increases the speed of writing the project.

Layout design is also useful for determining the general look and feel of the application, what is important for increasing the ease of use of the final application.

3.5.1 Server Main Window

The main window contains only information about number of connected clients and provides basic media manipulation. On the screen there is also visible the tray icon.

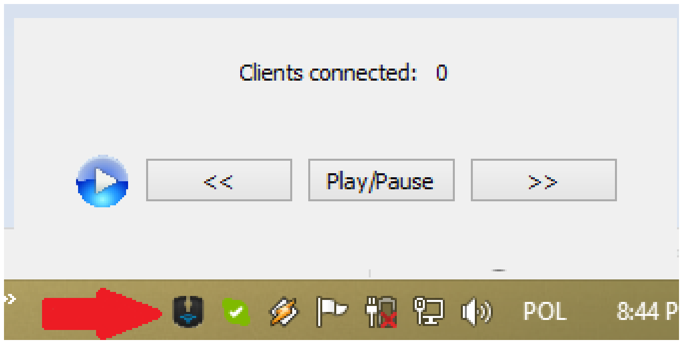


FIGURE 22. Server: Main window

3.5.2 Server popup menu

The popup menu shows up when the user performs right click on the tray icon.

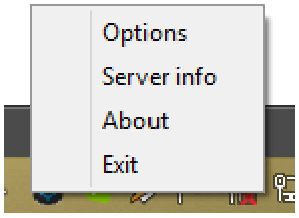
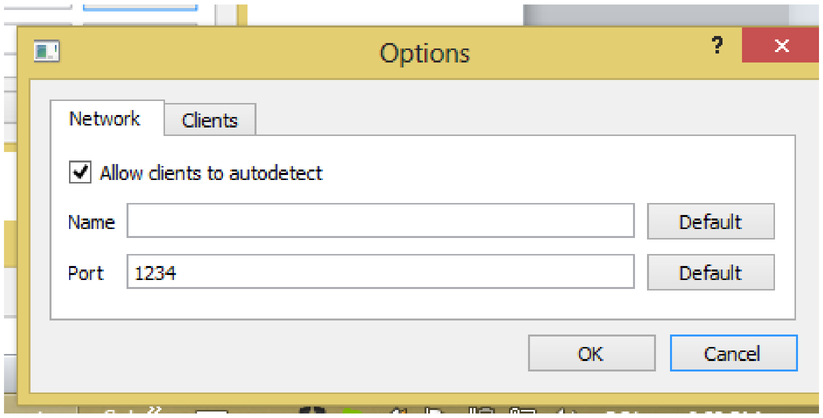


FIGURE 23. Server: Menu

3.5.3 Server configuration window

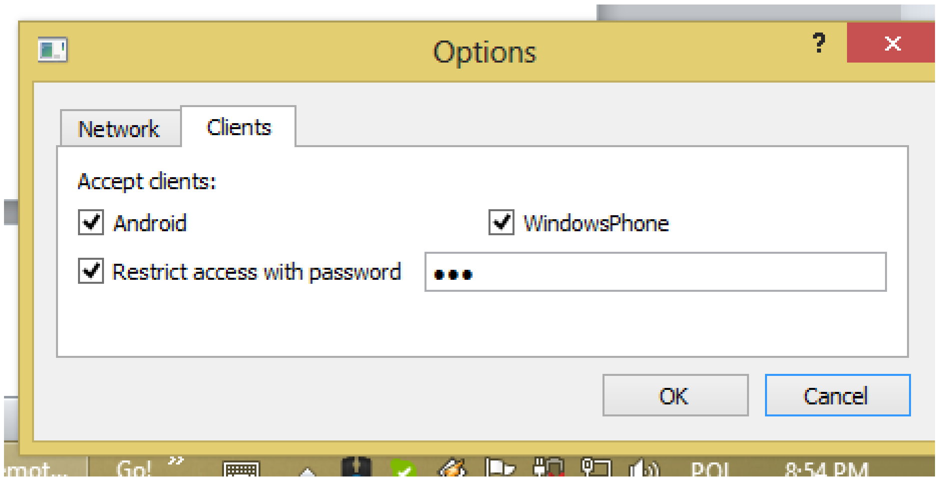
Server configuration window contains two tabs: Network and Clients.
In network tab, user can define name for the server (by default the actual name of the computer, the server application runs on) and port that the server will listen on for incoming connections.

Then in the Clients tab the user can define restrictions for the clients such as the version of client application or password that is required for connection establishment.



☑ Allow clients to autodetect
Name
Port 1234

FIGURE 24. Server: Network configuration window



Accept clients:
☑ Android ☑ WindowsPhone
☑ Restrict access with password •••

FIGURE 25. Server: Client configuration window

3.5.4 Server info

Server info pop-up contains all necessary information about the host required for initialization of connection with the server.

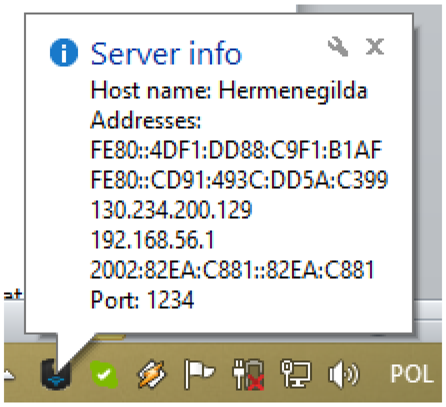

Server info
Host name: Hermenegilda
Addresses:
FE80::4DF1:DD88:C9F1:B1AF
FE80::CD91:493C:DD5A:C399
130.234.200.129
192.168.56.1
2002:82EA:C881::82EA:C881
Port: 1234

FIGURE 26. Server: Server info

3.5.5 Android client Main Window

From the main window users can easily navigate to each part of the application. At start up, the application automatically navigates from the main window to server list as this is essential to connect to the server before using most features such as remote touchpad or remote keyboard.

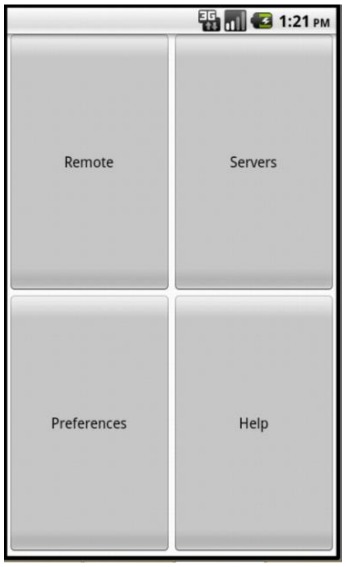


FIGURE 27. Android: Main Window

3.5.6 Android client adding new server

The view allows to manually enter the name for a server, that will be displayed on the list of servers later on as well as the address and port of a server that will be used for the connection. The same view is also used for editing data server.

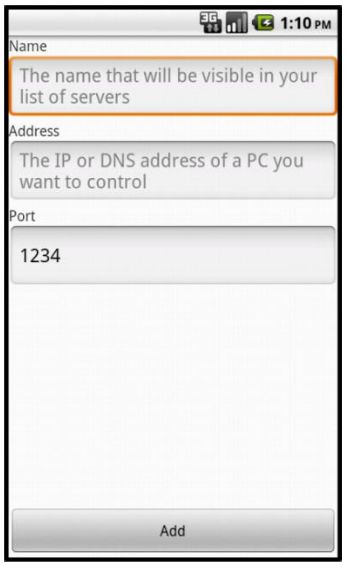

1:10 PM
Name
The name that will be visible in your list of servers
Address
The IP or DNS address of a PC you want to control
Port
1234
Add

FIGURE 28. Android: New server window

3.5.7 Android client configuration

Configuration window allows to configure how long the user will have to press the touchpad, for the touch to be evaluated as Long Press Event and eventually mapped as right mouse button click event as well as define sensitivity of cursor and scroll bars.

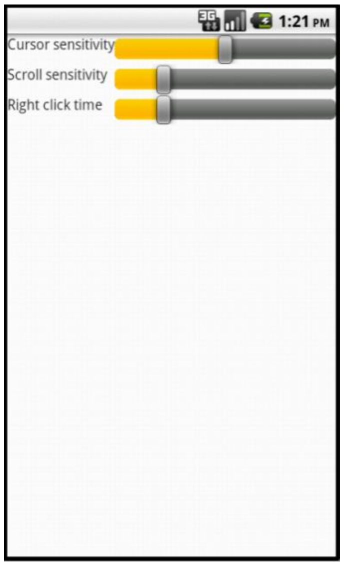
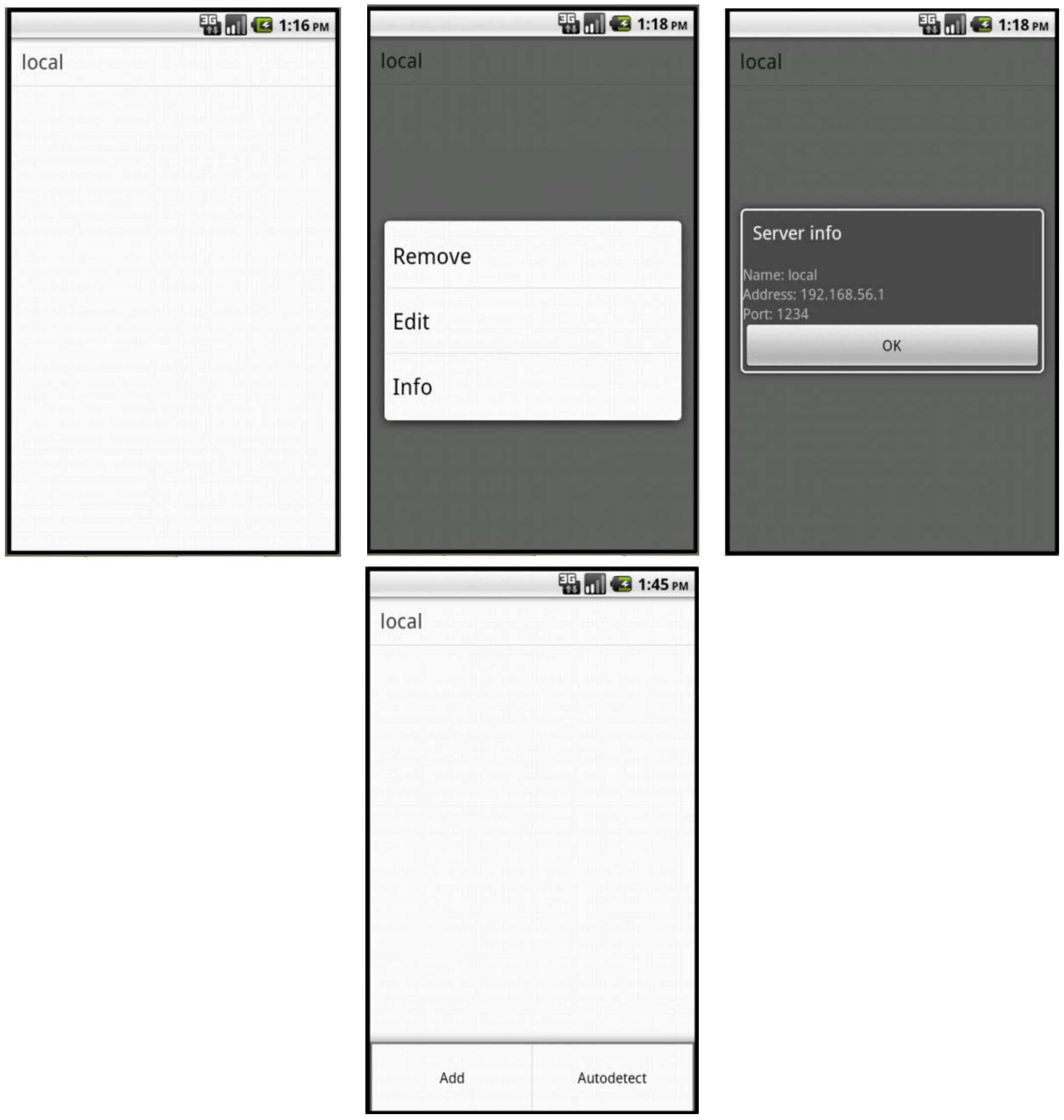


FIGURE 29. Android: Configuration window

3.5.8 Android client server list

The server list window allows the user to add, remove and edit data about available servers.

When user presses the standard menu button, a menu is shown at the bottom of the screen that enables to add more servers to the list.



Name: local
Address: 192.168.56.1
Port: 1234

FIGURE 30. Android: Server list

When user performs long press on one of already existing servers in the list, a pop-up menu is shown that enables the user removal, edition and printing out the information about the server.

Finally, when the user touches an item in the list shortly, the application attempts to establish a connection with the server.

3.5.9 Android controllers

The windows above allow the user to perform the input operations such as cursor movement and keyboard events. Note that some of them contain the same kind of controllers. For instance, first two windows presented in FIGURE 29 both contain touch pad controller. The bigger the touch pad is, the easier it is for the User to control the cursor. The second window however contains two additional buttons for switching the slides of a presentation.

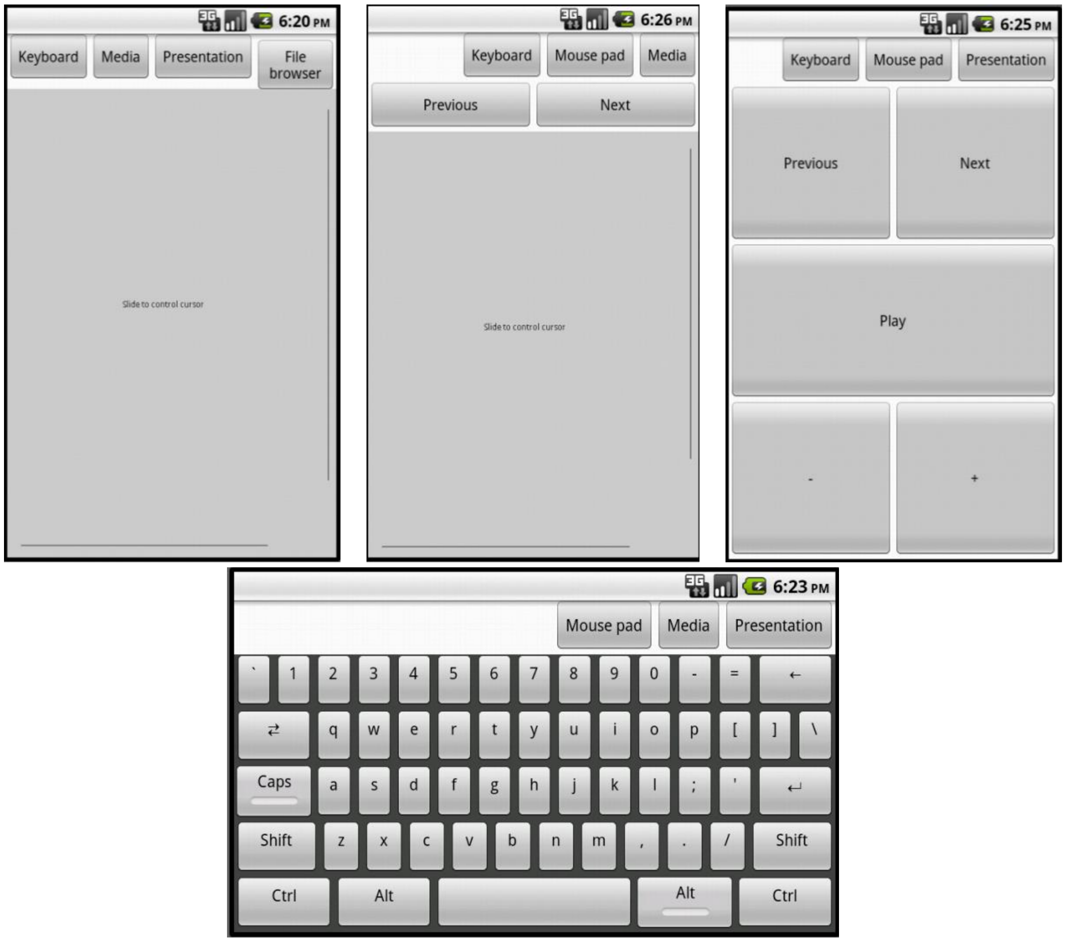
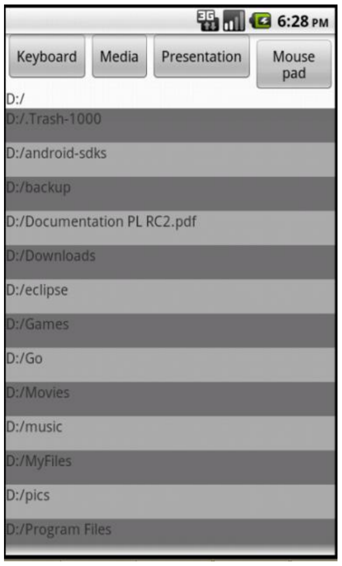


FIGURE 31. Android: Controllers

The other two windows both contain keyboard buttons, but they group the keys by their functionality.

3.5.10 Android client Remote File Browser Activity

Finally there is also Remote File Browser. When started, it contains a list of all drives detected in the hosting system. Once some directory gets clicked by the user, a request for the information about the content of a directory is sent to the server and eventually the whole content of a list is replaced with information found in the response from the server.



D:/
D:/.Trash-1000
D:/android-sdks
D:/backup
D:/Documentation PL RC2.pdf
D:/Downloads
D:/eclipse
D:/Games
D:/Go
D:/Movies
D:/music
D:/MyFiles
D:/pics
D:/Program Files

FIGURE 32. Android: Remote File Browser

When user clicks an item in the list representing a file, it gets automatically downloaded to the mobile device.

3.5.11 Windows Phone

Most windows in Windows Phone client look exactly the same, with only one small exception – all controllers are placed on one Page only. Windows Phone provides the programmer with Pivot Pages that makes it easy and more intuitive for the user to switch between controllers by sliding through the views.

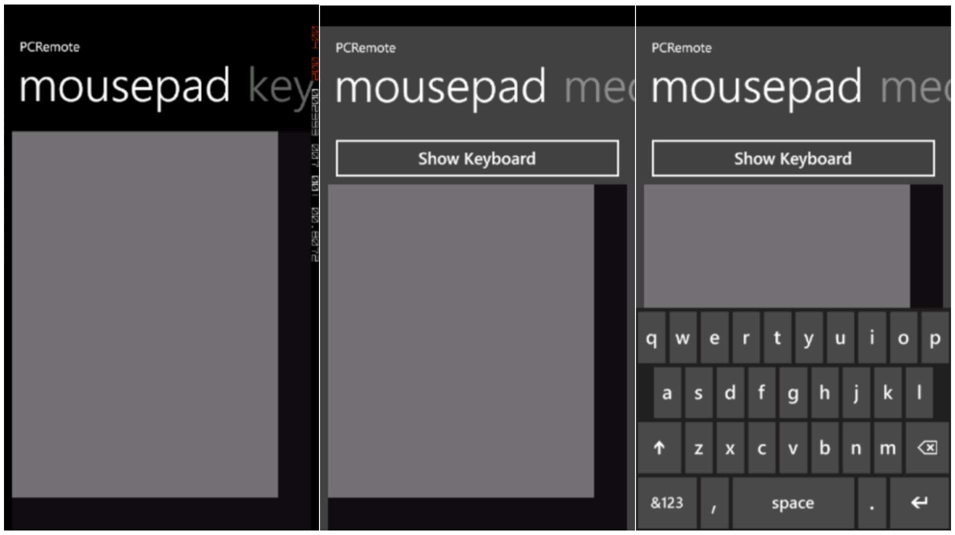


FIGURE 33. Windows Phone: Controllers

3.6 Network messages

In the project two kinds of transport protocols are used: TCP and UDP.
The reason is because TCP generates small delays (since it performs multiple data integrity checks) that could decrease the comfort of using the application, as well as it requires to establish connection, what is impossible during for instance server detection (it is easier to simply send one UDP packet to broadcasting address and then wait for responses).

Whenever the user performs some action with a controller, a new event is generated in the system. The event is then mapped into an appropriate Network Message.
In order to prevent problems with freezing the user interface, both sending and receiving the network messages must be done asynchronously (in separate threads). Whenever a Network Message is generated, it should be passed to the Communicator (the class that provides simple programming interface for managing sending and receiving threads). The Communicator puts all the messages into a message queue and then starts a thread responsible for sending the message.

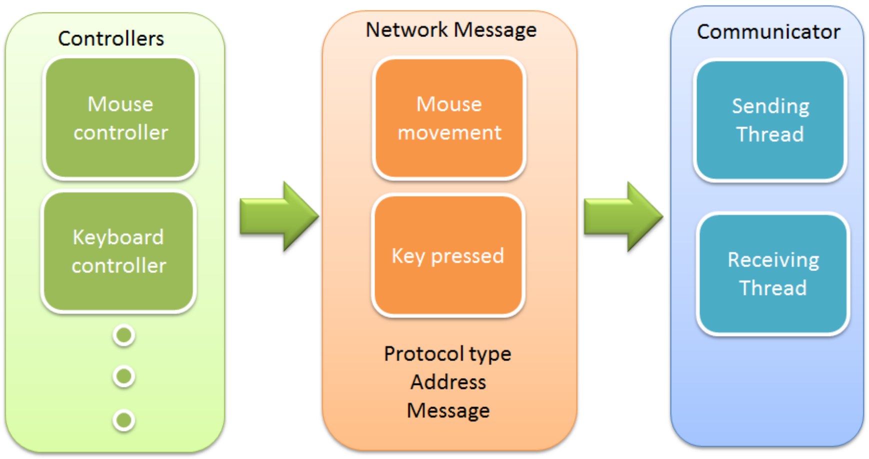


FIGURE 34. Network message generation

The receiving thread in the controller is blocked until it receives a message from the server. Once it happens, an appropriate event is called on all registered in the Communicator response listeners.

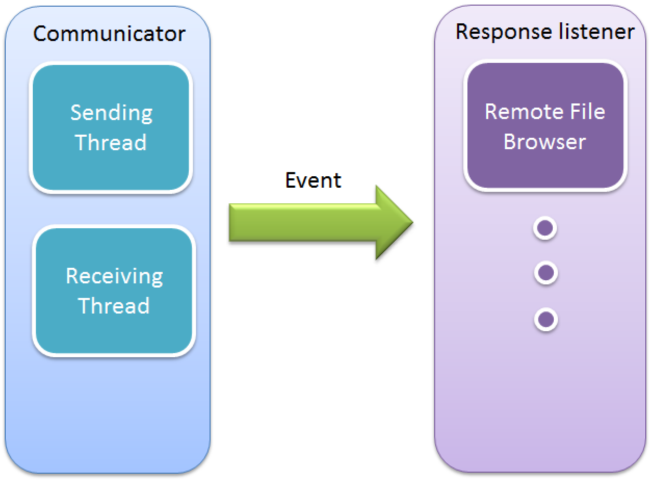


FIGURE 35. Network message processing

3.6.1 Messages sent to server with TCP protocol

Very often there are more than one programmers working on one project in the same time. Because of this, it is important to define in advance how different parts of project will communicate with each other.

The following are messages that clients can generate and send to client with TCP protocol:

TABLE 1: Messages sent to server with TCP protocol

Message name	Number	Bytes description

	of bytes	Operation identifier	Operation	Operation
Scroll	3	1	1- Vertical scroll 2- Horizontal scroll	Value scrolled
Mouse click event	3	2	1- Click 2- Button down 3- Button up	1- Left button 2- Right button 3- Middle button
Introduction (tells server what kind of client it is)	4	5	1- Android 2- Windows Phone	Two bytes for version number (optional)
Keyboard event	3	6	1- Button down 2- Button up 3- Button clicked	Button code
Directory request	n	8	Number of bytes of the path (16 bytes)	Path of directory
File request	n	9	Number of bytes of the path (16 bytes)	Path of file

3.6.2 Messages sent to server with UDP protocol

As some features, that the project should support, require immediate reaction of the server, but yet it is not that important that every single packet makes it to its destination, some network messages generated by the client are sent via UDP protocols. Those are the following:

TABLE 2: Messages sent to server with UDP protocol

Message name	Number of bytes	Bytes description		
		Operation identifier	Operation	Operation
Mouse move	3	1	X coordinate	Y coordinate

3.6.3 Messages sent by server with TCP protocol

Whenever client requests a piece of data, the programmer has to make sure, that the response will always get delivered properly, therefore all responses sent by server to client use TCP protocol.

TABLE 3: Messages sent by server with TCP protocol

Message name	Number of bytes	Bytes description	
		Operation identifier	Content
Directory response	n	1	16 bytes - number of records Record: - 16 bytes – size of path - 1 byte determines if it is file or directory (0 – file, 1 – directory) - x bytes – path
File response	n	2	32 bytes – file size x bytes – file content

3.6.4 Key codes

Since Android, Windows Phone and Windows use slightly different key codes, it is necessary to create a separate set of values representing keys that will be common for all three applications of PCRemote project.

TABLE 4: Key codes

Key name	Value
Left Shift	1
Right Shift	2
Left Alt	3
Right Alt	4
Space	5
Right Control	6
Left Control	7
A	8
B	9
C	10
D	11
E	12
F	13
G	14
H	15
I	16
J	17
K	18
L	19
M	20
N	21

O	22
P	23
Q	24
R	25
S	26
T	27
U	28
V	29
W	30
X	31
Y	32
Z	33
, <	34
. >	35
/ ?	36
; :	37
‘ “	38
[}	39
] {	40
\ |	41
~ `	42
1	43
2	44
3	45
4	46
5	47
6	48
7	49
8	50
9	51
0	52
- _	53
+ =	54
Backspace	55
Enter	56
Tab	57
Caps Lock	58
Volume Up	59
Volume Down	60
Next Track	61

Previous Track	62
Play Track	63

4 SUMMARY

Creating three separate applications that would communicate through network messages is not an easy task, but it certainly is worth the effort.

Now, thanks to the PCRemote project, it is easier to give presentations, as the user no longer needs to stay close to the computer just to perform input required for switching slides, pointing some data with cursor, or entering text.

Thanks to this project I was able to learn that Qt, despite of being a really good portable framework for C++, still lacks a lot of features, what forces the programmers to use the standard system libraries, such as WinAPI, what makes the code more complex and less portable. I learnt that programming Windows Phone applications is not as easy as it may seem, since a lot of features of the .NET Framework are simply not accessible for mobile applications. Finally, I learnt that although Java is a great programming language, applications written in it are not always as portable as they originally were meant to be.

Both Android and Windows Phone applications have their advantages and disadvantages. In Android it is really easy to perform any kind of network communication and thread synchronisation, as standard Java provides a very simple and easy to use framework that supports this kind of features, however creating the user interface for applications designed for this operating system is most of the time rather complicated. In case of Windows Phone, creating the clear user interface is extremely easy, but implementing some logic working in the background often causes a lot of problems, since framework forces the programmer to perform a lot of operations asynchronously, but it still doesn't provide basic mechanisms for thread synchronisation. This way some functions need to work in non-blocking way, what results in poorer performance of the application as well as faster battery drain.

REFERENCES

Android developers – official documentation - http://developer.android.com - 5.2013

Android basic tutorial - http://developer.android.com/training/basics/firstapp/starting-activity.html, 5.2013

Android touch event consumption - http://developer.android.com/training/graphics/opengl/touch.html, 5.2012

Android notifications - http://developer.android.com/guide/topics/ui/notifiers/notifications.html, 5.2013

Android shares information - http://developer.android.com/about/dashboards/index.html, 5.2013

JUnit tests - http://www.oracle.com/technetwork/articles/adf/part5-083468.html, 5.2013

Microsoft Developer Network (MSDN) - http://msdn.microsoft.com/en-US/ - 5.2013

Oracle Technetwork- http://www.oracle.com/technetwork/ - 5.2013

Qt Project - http://qt-project.org/doc/qt-4.8/tutorials.html - 5.2013

Windows Phone general information - http://en.wikipedia.org/wiki/Windows_Phone, 5.2013

Windows Phone programming basic tutorial - http://channel9.msdn.com/Series/Windows-Phone-7-Development-for-Absolute-Beginners, 5.2013

Windows Phone touch event consumption - http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj207076(v=vs.105).aspx, 5.2013)

Windows Phone unit tests - http://msdn.microsoft.com/en-us/library/ms182532.aspx, 5.2013)

Figure 1 reference http://developer.android.com/reference/android/app/Activity.html, 5.2013)

Figure 5(http://jesseliberty.com/2011/07/27/page-statestep-by-step/, 5.2013)

Figure 6 - http://www.concept-phones.com/nokia/windows-phone-7-concept-ui-nokia-relies-cubes-brilliant/, 5.2013

Figure 7http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662938(v=vs.105).aspx, 5.2013)