POLITECHNIKA KRAKOWSKA

IM. TADEUSZA KOŚCIUSZKI

WYDZIAŁ FIZYKI MATEMATYKI I INFORMATYKI

KIERUNEK INFORMATYKA

KAMIL JANOWSKI

**Programowanie aplikacji mobilnych na przykładzie aplikacji zdalnie sterującej komputerem dla Androida i Windows Phone'a**

**Developing mobile applications in an example of application for PC remote controlling created for Android and Windows Phone**

PRACA INŻYNIERSKA

STUDIA STACJONARNE

Promotor: Dr inż. Paweł Jarosz

Kraków 2012



PCRemote

Dokumentacja

Spis treści

[1. Wstęp 6](#_Toc344506936)

[1.1. O projekcie 6](#_Toc344506937)

[1.2. Cel 6](#_Toc344506938)

[2. Analiza wymagań i projekt 6](#_Toc344506939)

[2.1. Wymagania funkcjonalne 6](#_Toc344506940)

[2.1.1. Wymagania funkcjonalne dla klientów 6](#_Toc344506941)

[2.1.2. Wymagania funkcjonalne dla serwera 7](#_Toc344506942)

[2.2. Wymagania niefunkcjonalne 7](#_Toc344506943)

[2.2.1. Wymagania niefunkcjonalne serwera 7](#_Toc344506944)

[2.2.1. Wymagania niefunkcjonalne dla klienta 7](#_Toc344506945)

[2.2. Historie użytkowników 7](#_Toc344506946)

[2.2.2. Swerwer PCRemote 7](#_Toc344506947)

[2.2.3. Klienci 8](#_Toc344506948)

[2.3. Diagramy 9](#_Toc344506949)

[2.3.2. Serwer PCRemote 9](#_Toc344506950)

[2.3.3. Klient PCRemote dla Windows Phone’a 12](#_Toc344506951)

[2.3.4. Klient PCRemote dla Androida 16](#_Toc344506952)

[2.4. Projekty okien 18](#_Toc344506953)

[2.4.2. PCRemote Server 18](#_Toc344506954)

[2.4.3. Klient PCRemote dla Windows Phone’a 19](#_Toc344506955)

[2.4.4. Klient PCRemote dla Androida 23](#_Toc344506956)

[3. Wybór i opis technologii 28](#_Toc344506957)

[3.1. Wybór technologii 28](#_Toc344506958)

[3.2. Android 28](#_Toc344506959)

[3.2.1. Podstawy 28](#_Toc344506960)

[3.2.2. Zasoby 29](#_Toc344506961)

[3.2.3. Kontrolki (View) 31](#_Toc344506962)

[3.2.4. Powiadomienia 31](#_Toc344506963)

[3.2.5. Framework 32](#_Toc344506964)

[3.3. Windows Phone 33](#_Toc344506965)

[3.3.1. Języki 33](#_Toc344506966)

[3.3.2. Powiadomienia 33](#_Toc344506967)

[3.3.3. Strony 35](#_Toc344506968)

[3.3.4. Menu 35](#_Toc344506969)

[3.4. Ważne różnice w technologiach 37](#_Toc344506970)

[3.4.1. Pochłanianie zdarzeń dotyku 37](#_Toc344506971)

[3.4.2. Zarządzanie energią 38](#_Toc344506972)

[3.4.3. Wzorce 38](#_Toc344506973)

[3.5. Technologie alternatywne 39](#_Toc344506974)

[4. Implementacja 40](#_Toc344506975)

[4.1. Wiadomości sieciowe 40](#_Toc344506976)

[4.1.1. Wiadomości przesyłane przy pomocy protokołu TCP 40](#_Toc344506977)

[4.1.2. Wiadomości przesyłane przy pomocy protokołu UDP 41](#_Toc344506978)

[4.1.2.2. Wiadomości wysyłane przez serwer 41](#_Toc344506979)

[4.2. Kody przycisków 41](#_Toc344506980)

[4.3. Testy 43](#_Toc344506981)

[4.3.1. Android 43](#_Toc344506982)

[4.3.2. Windows Phone 44](#_Toc344506983)

[4.4. Urządzenia testowe 45](#_Toc344506984)

[4.5. Narzędzia użyte do tworzenia aplikacji 46](#_Toc344506985)

[5. Podsumowanie 46](#_Toc344506986)

[6. Słowniczek 47](#_Toc344506987)

[7. Bibliografia 47](#_Toc344506988)

[Dodatek 1 – Rozwój projektu 48](#_Toc344506989)

[Dodatek 2 – Źródła 49](#_Toc344506990)

PCRemote

Dokumentacja

# Wstęp

## O projekcie

Projekt składa się z trzech aplikacji: serwera napisanego przy pomocy C++, Qt Framework i WinAPI, który będzie uruchamiany w systemie operacyjnym z rodziny MS Windows oraz dwóch klientów, jeden napisany dla Androida, drugi dla Windows Phone’a.

Klienci komunikują się z serwerem przy pomocy protokołu TCP (dla ogólnych rozwiązań jak np. przesyłanie informacji o wciśniętym klawiszu klawiatury) oraz UDP (dla przesyłania informacji o ruchu kursora, jako że w przypadku protokołu UDP nie jest przeprowadzane żadne sprawdzanie poprawności danych, a dzięki temu uzyskuje się dużo większą wydajność, która w przypadku poruszania kursora jest bardzo ważna) w celu przejęcia pełnej kontroli nad kursorem oraz klawiaturą komputera, na którym uruchomiony jest serwer.

## Cel

Projekt ma przede wszystkim ułatwić prezentowanie prezentacji multimedialnych. Dzięki aplikacji użytkownik podczas swojej prezentacji nie jest już zmuszony do ciągłego stania w pobliżu komputera. Dzięki zdalnej obsłudze kursora jest w stanie wskazywać elementy na kolejnych slajdach, a także zmieniać same slajdy, zaś obsługa klawiatury oraz przycisków medialnych ułatwiają przedstawianie dodatkowych demonstracji, np. gdy prezentacja zakłada faktyczne uruchomienie jakiejś strony internetowej, której adres należy najpierw wpisać w przeglądarkę.

# Analiza wymagań i projekt

## Wymagania funkcjonalne

### Wymagania funkcjonalne dla klientów

1. Istnieje możliwość, żeby używać urządzenie mobilne jako touchpad w celu kontrolowania kursora i scrolli.
2. Istnieje możliwość, aby kontrolować wszystkie przyciski klawiatury, które znajdują się w standardowym angielskim układzie.
3. Istnieje możliwość, żeby dodawać i zapisywać na stałe informacje o serwerze.
4. Istnieje możliwość połączenia się z serwerem przez protokół TCP
5. Istnieje możliwość konfiguracji czułości kursora.
6. Istnieje możliwość wybrania czasu, po którym przyciśnięcie touchpada będzie uznane za “przyciśnięcie długie”, co potem jest tłumaczone jako przyciśnięcie prawego przycisku.
7. Istnieje możliwość konfiguracji czułości scrolla.

### Wymagania funkcjonalne dla serwera

1. Serwer wyświetla komunikaty o ustanowieniu nowego połączenia.
2. Serwer wyświetla komunikaty o zamknięciu połączenia z klientem.
3. Serwer przetwarza wszystkie żądania klientów dotyczące jakichkolwiek zdarzeń związanych z kursorem lub klawiaturą.
4. Posiada ikonę tray.
5. Jest w stanie podać wszelkie informacje jakie są konieczne, żeby ustanowić połączenie z nim.
6. Istnieje możliwość zmiany domyślnego portu nasłuchiwania.
7. Istnieje możliwość zamknięcia aplikacji poprzez menu ikony tray.

## Wymagania niefunkcjonalne

### Wymagania niefunkcjonalne serwera

1. System operacyjny Windows Vista / 7 / 8
2. Bezprzewodowe połączenie z internetem.

### Wymagania niefunkcjonalne dla klienta

1. System operacyjny Windows Phone (z serii 7.x) lub Android (co najmniej w wersji 2.2)
2. Bezprzewodowe połączenie z internetem.

## Historie użytkowników

### Serwer PCRemote

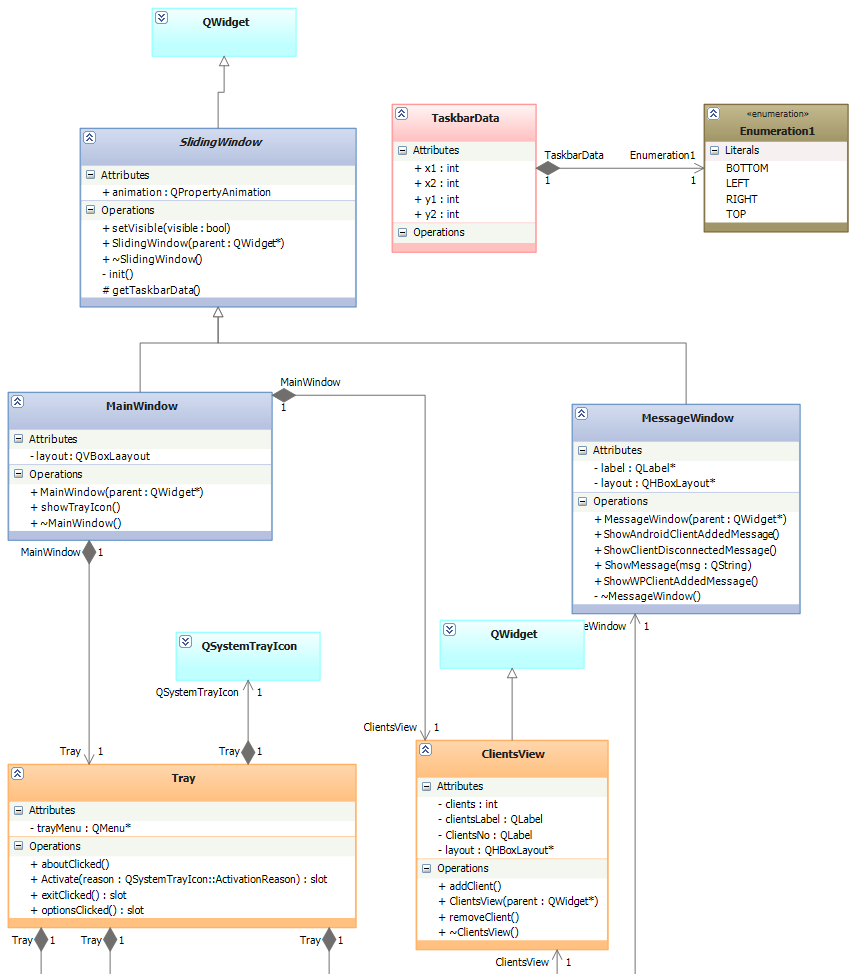
1. Jako użytkowniku aplikacji serwerowej, chcę mieć prosty dostęp do wszystkich jego funkcji poprzez ikonę tray.
2. Jako użytkownik, chcę aby moja aplikacja serwerowa umożliwiała mi zdalną kontrolę myszki, wszystkimi jej przyciskami, oraz klawiatury (łącznie z przyciskami medialnymi takimi jak play, next, previous).
3. Jako użytkownik serwera, chcę móc skonfigurować go tak, aby mógł odrzucać połączenia od konkretnych typów klientów (Android lub Windows Phone).
4. Jako użytkownik chcę, aby aplikacja serwerowa była w stanie dostarczyć instrukcji co do używania zarówno samego serwera jak też i poszczególnych klientów.
5. Jako użytkownik aplikacji serwerowej chcę mieć możliwość wyboru, czy klient może automatycznie wykryć mój serwer.
6. Jako użytkownik serwera chcę mieć możliwość ograniczenia dostępu do niego poprzez wymaganie hasła.
7. Jako użytkownik chcę by moja aplikacja serwerowa była w stanie sama utworzyć bezprzewodową sieć Ad Hoc w przypadku jeśli żadne inne sieci bezprzewodowe nie są dostępne.

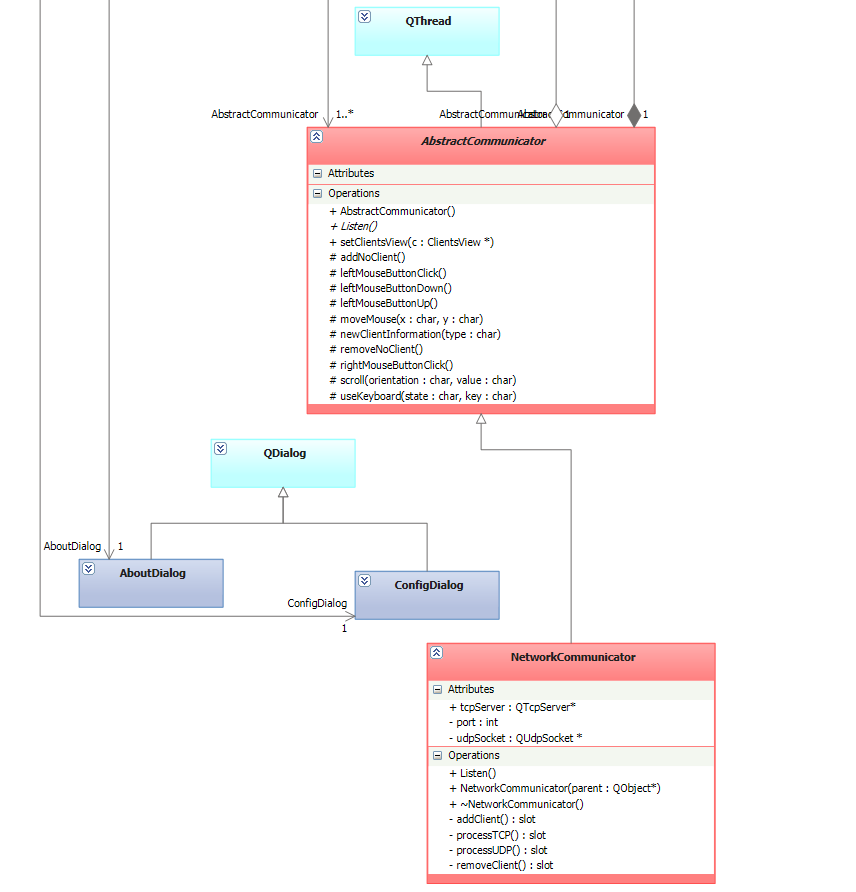
### Klienci

1. Jako użytkownik aplikacji klienckiej chcę mieć możliwość automatycznego wykrycia wszystkich serwerów PCRemote działających w sieci (pod warunkiem, że ich konfiguracja zezwala na automatyczne wykrywanie).
2. Jako użytkownik chcę mieć możliwość dodawania informacji o serwerach ręcznie, jako że konfiguracja niektórych serwerów może nie pozwalać na autowykrywanie.
3. Jako użytkownik chcę mieć możliwość kontrolowania mojego kursora tak samo jak w przypadku zwykłego touchpada.
4. Jako użytkownik chcę, chcę mieć możliwość zdalnej obsługi klawiatury na komputerze na którym zainstalowana jest aplikacja serwerowa PCRemote.
5. Jako użytkownik chcę mieć możliwość zdalnego symulowania przycisków mediów takich jak: play, next, previous oraz manipulacji głośnością.
6. Jako użytkownik chcę mieć możliwość wyboru czułości touchpada na moim urządzeniu mobilnym.
7. Jako użytkownik chcę mieć możliwość konfigurowania czułości scrolli na moim urządzeniu mobilnym.

## Diagramy

### Serwer PCRemote





Główną częścią aplikacji jest klasa Tray. Pomimo faktu, że jest ona tylko częścią klasy MainWindow, jest ona widoczna przez cały czas, podczas gdy MainWindow ujawnia się tylko podczas zdarzenia activated wywołanego na klasie Tray (kiedy ikona zostanie kliknięta lewym przyciskiem myszy). Tray zawiera zestaw obiektów typu AbstractCommunicator. Co prawda w tym przypadku aplikacja komunikuje się tylko przy pomocy połączenia sieciowego (wykorzystuje tylko klasę NetworkCommunicator dziedziczącego po AbstractCommunicator), to jednak w ten sposób łatwo będzie w późniejszym etapie dodać inne sposoby komunikacji jak np. poprzez bluetooth.

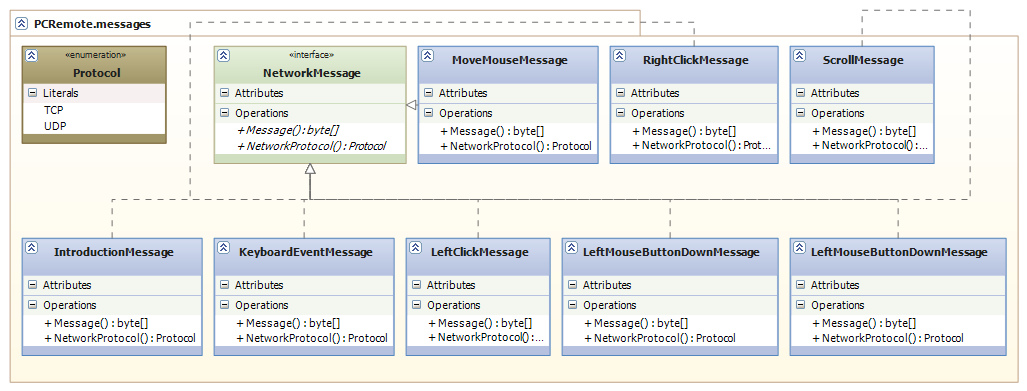
Tray też zawiera menu, które pozwala na obsługę całej aplikacji.



ClientsView jest niczym innym jak tylko kolejną kontrolką zawierającą w sobie dwa obiekty QLabel. Jeden z nich pełni zwykłą funkcję informacyjną, drugi przedstawia liczbę aktualnie podłączonych klientów.



### Klient PCRemote dla Windows Phone’a



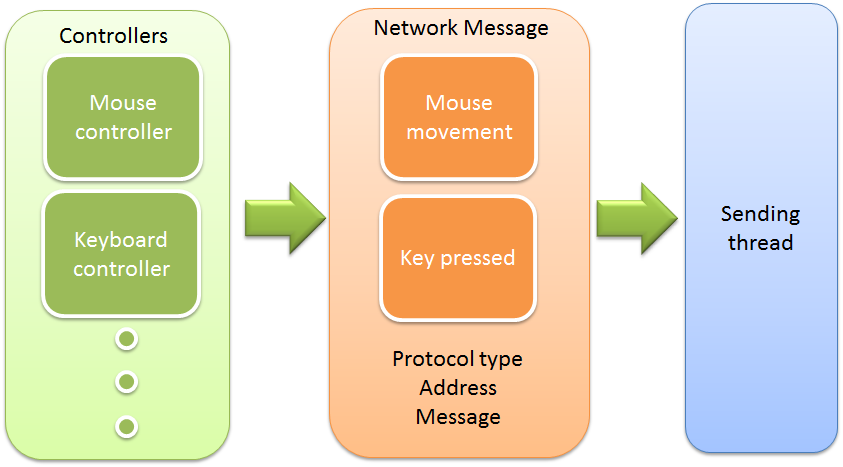
Klasy implementujące interfejs NetworkMessage reprezentują wiadomości przesyłane do serwera. Każda z nich zawsze zawiera dwie funkcje: Message(), która zwraca tablicę typu byte, reprezentującą wiadomość, która ma zostać przesłana oraz NetworkProtocol, zwracającą Protocol.TCP lub Protocol.UDP w zależności od tego jaki protokół ma zostać użyty do wysłania danej wiadomości.

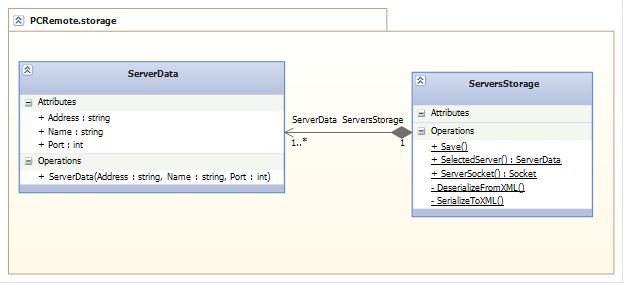
O ile nazwy większości komunikatów mówią same za siebie do czego służą, o tyle IntroductionMessage może pozostawiać pewne wątpliwości. Jest to wiadomość wysyłana do serwera automatycznie zaraz po nawiązaniu połączenia TCP. Zawiera ona informacje o wersji klienta (numer wersji oraz wersję systemu operacyjnego, na którym działa).

Gdy użytkownik przeprowadzi jakąś interakcję z kontrolką reprezentującą dane urządzenie wejścia, wątek główny jedynie generuje odpowiedni obiekt klasy implementującej interfejs NetworkMessage i dodaje go do kolejki wiadomości oczekujących na wysłanie. Istnieje dodatkowy osobny wątek, który wciąż sprawdza, czy w kolejce aktualnie znajdują się jakieś wiadomości do wysłania. Jeśli takowa tam trafia, wątek tła rozpoczyna jej wysyłanie.

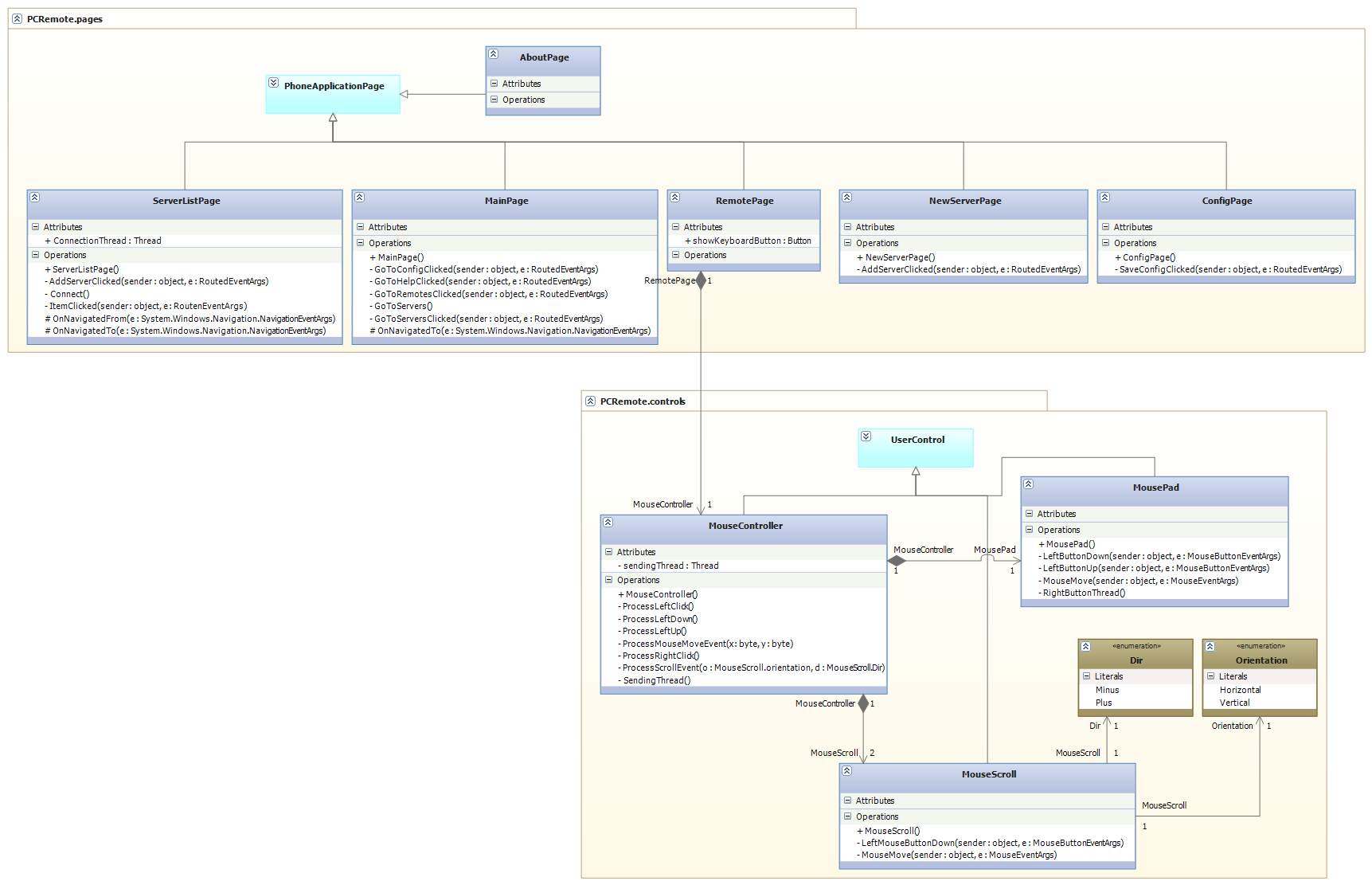
W Javie (i tym samym w Androidzie) istnieją kolejki blokujące, dzięki którym dopóki w kolejce nie ma żadnych elementów, dopóty wątek ją czytający będzie blokowany i zostanie wznowiony dopiero w chwili dodania nowego elementu do kolejki.

W Windows Phonie jest niestety zaimplementowany tylko .NET Core, a co za tym idzie, brakuje w nim dostępu do semaforów, a bez semaforów nie można stworzyć kolejki blokującej. Tym samym, ten sam problem w przypadku Windows Phone’a musi być rozwiązany w sposób nieblokujący. Nie jest bardzo poważny problem z punktu widzenia programistycznego, ale powoduje większe zużycie energii, co ma poważne znaczenie w przypadku urządzeń mobilnych.



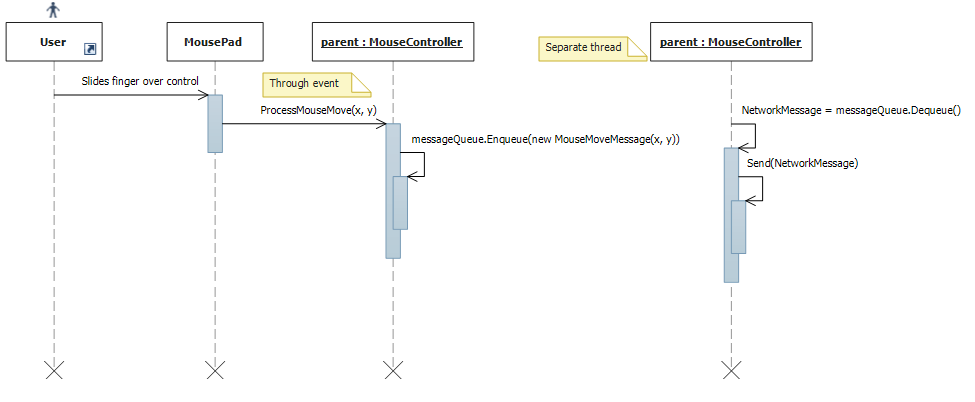


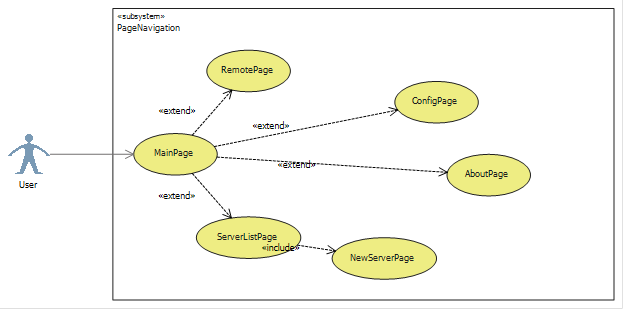
Dane zawarte w klasie ServersStorage są wspólne dla wszystkich elementów całej aplikacji, toteż dużo łatwiej jest zaimplementować tą klasę, jako klasę statyczną, niż przekazywać referencję do pojedynczego jej obiektu do wszystkich Stron programu.



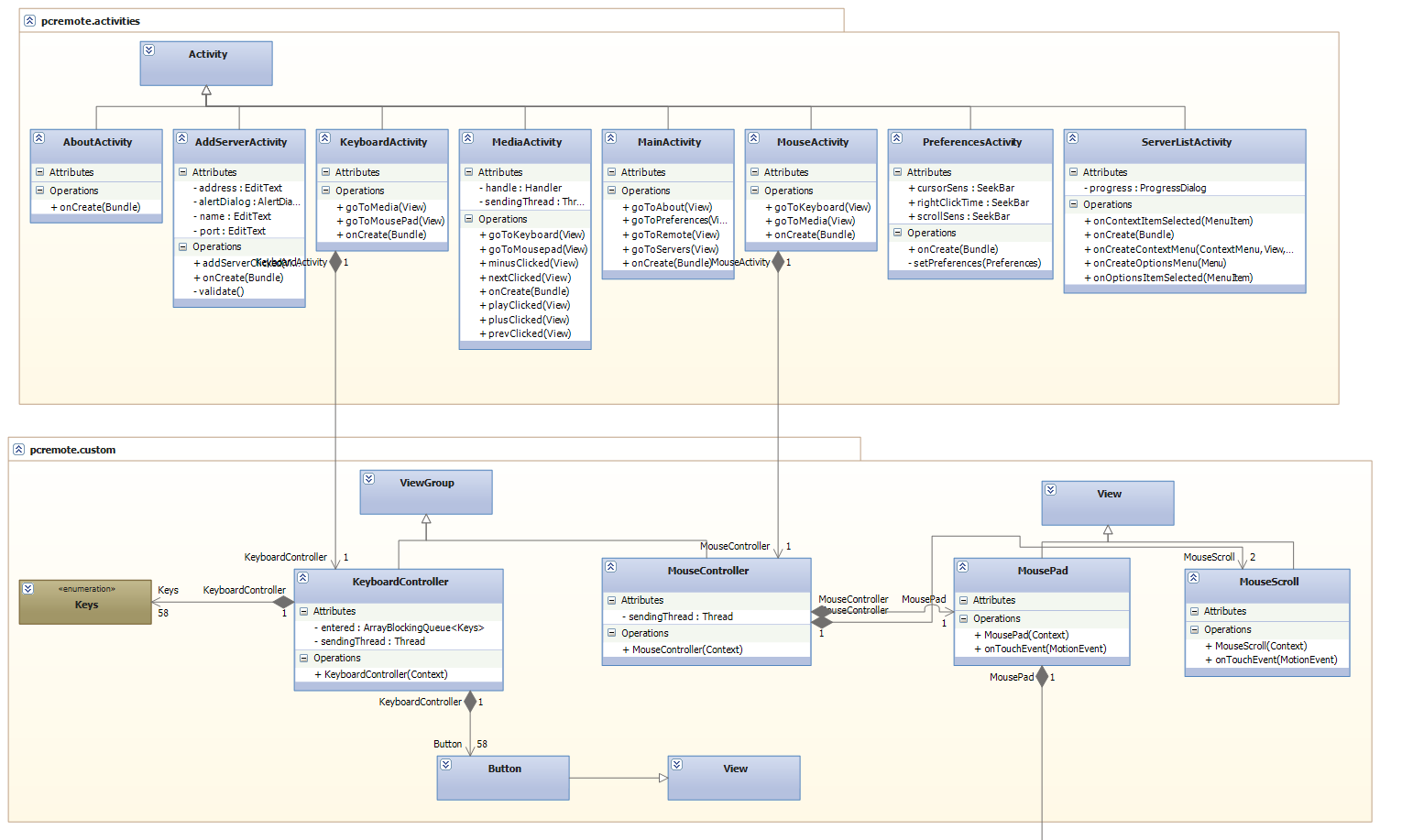
MouseController jest klasą dziedziczącą klasę UserControl, która enkapsuluje 2 kontrolki: MousePad oraz MouseScroll co pozwala jej spokojnie na odgrywanie roli touchpada.

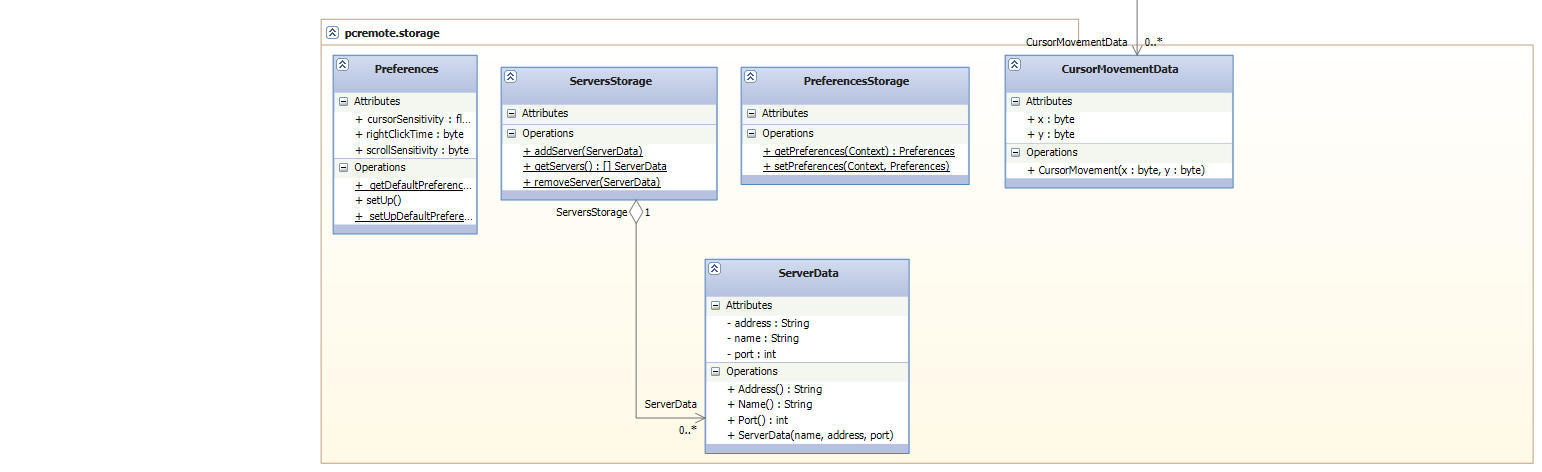
Za każdym razem, gdy MousePad lub MouseScroll rozpoznają gest, generują zdarzenie, które w następnej kolejności może zostać przechwycone przez Stronę, która powinna następnie wygenerować obiekt implementujący interfejs NetworkMessage oraz wstawić go do kolejki wiadomości (messageQueue), która ciągle jest odczytywana przez wątek wysyłający, który jeśli zobaczy, że w kolejce znajduje się jakaś wiadomość, wysyła ją od razu do serwera.





### Klient PCRemote dla Androida



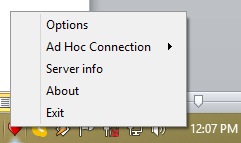


Pozostałe diagram dla klienta na Androida wyglądają analogicznie do tych na Windows Phone’a.

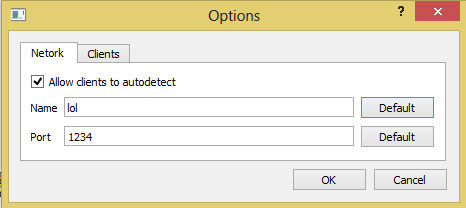
## Projekty okien

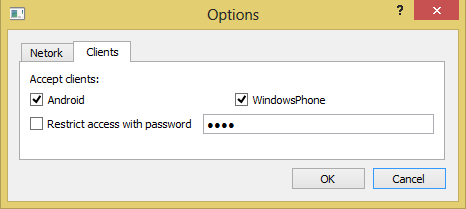
### PCRemote Server

#### Główne menu

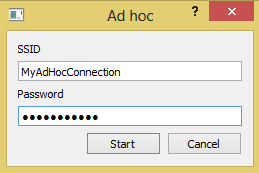


#### Okno opcji



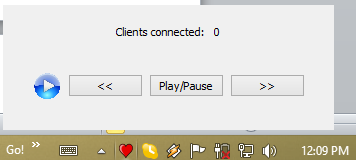


#### Okno połączenia Ad hoc



#### Główne okno

(Pojawia się, gdy przyciśnięta jest ikona tray)



### Klient PCRemote dla Windows Phone’a

#### Strona główna



Główna strona umożliwia nawigację do wszystkich innych stron.

#### Strona nowego serwera



Pozwala na ręczne wprowadzenie danych o serwerze.

#### Lista serwerów



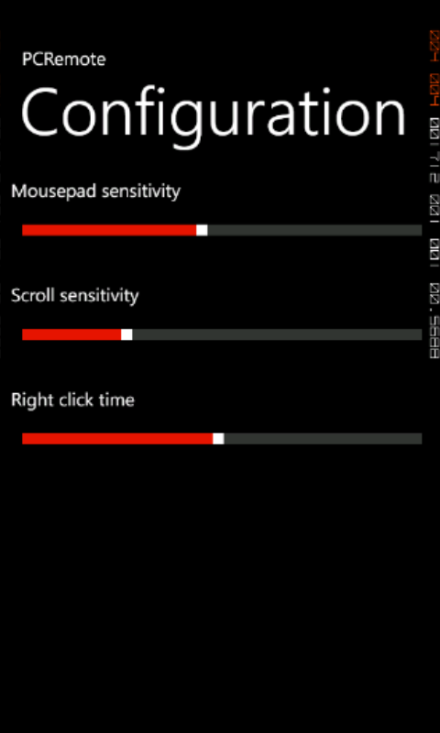
Pozwala na przeglądanie listy wszystkich zapisanych serwerów, połączenie się do jednego z nich, jak też i przejście do ręcznego dodawania serwerów lub ich automatycznego wykrywania.

#### Lista serwerów + menu kontekstowe



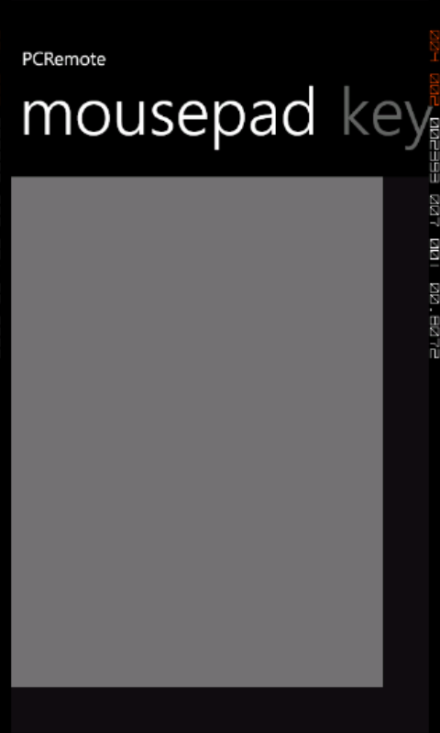
Jeśli użytkownik przyciśnie dłużej dany serwer, pojawia się menu kontekstowe, które pozwala na edycję lub usunięcie wybranego serwera.

#### Strona konfiguracji



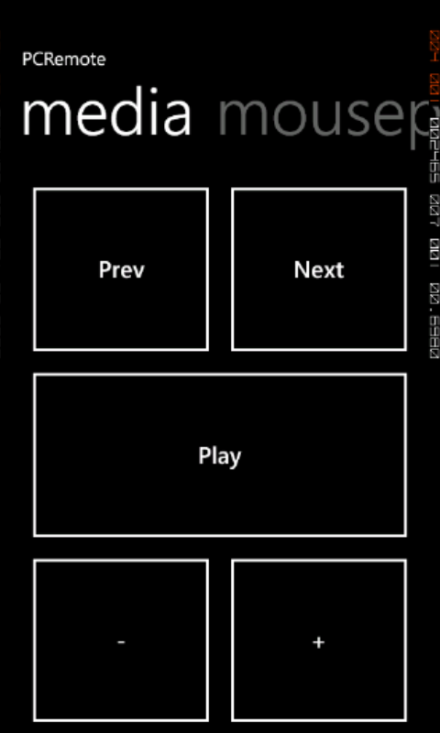
Okno konfiguracji pozwala wybrać czułość touchpada I scrolli jak również czas po którym przyciśnięcie touchpada będzie uznane za „przyciśnięcie długie” co wygeneruje zdarzenie wciśnięcia prawego przycisku myszy.

#### Zdalny touchpad



Umożliwia poruszanie kursorem i przewijanie scrolli. W tym przypadku szary obszar reprezentuje gładzik, podczas gdy dwa czarne pasy po prawej i na dole służą do przewijania scrolli.

#### Zdalne media



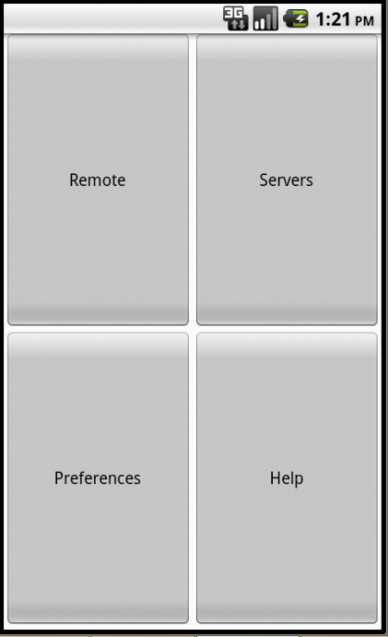
Strona mediów pozwala na zdalne sterowanie przyciskami klawiatury do sterowania mediami.

### Klient PCRemote dla Androida

Jako, że specyfikacja Windows Phone’a i Androida różnią się dość znacząco od siebie ich GUI też nie może być dokładnie takie samo.

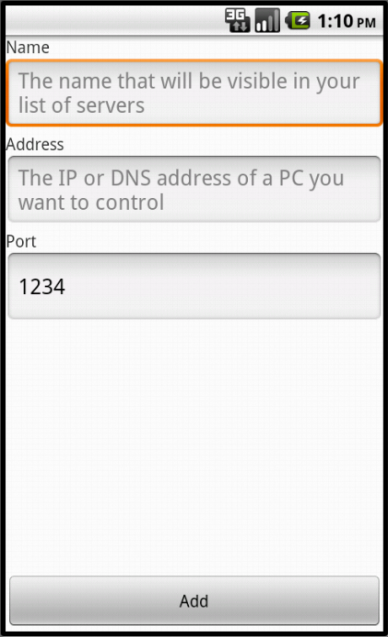
Na przykład wersjach Androida przed wersją 3.0 istnieje wymóg wstawiania fizycznego przycisku w urządzeniu, który służy do uruchamiania menu, podczas gdy Windows Phone nigdy takiego wymogu nie posiadał. W Windows Phonie istnieją tak zwane Pivot Pages podczas gdy w Androidzie 2.2 nie ma nic, co by działało w analogiczny sposób. Istnieją co prawda zakładki, ale wyglądają one zupełnie inaczej i nie działają równie wydajnie. Do tego dochodzą różne ograniczenia we frameworkach, które różnią się w obu przypadkach i tym samym narzucają nieco inne podejścia do tych samych problemów.

#### Strona główna



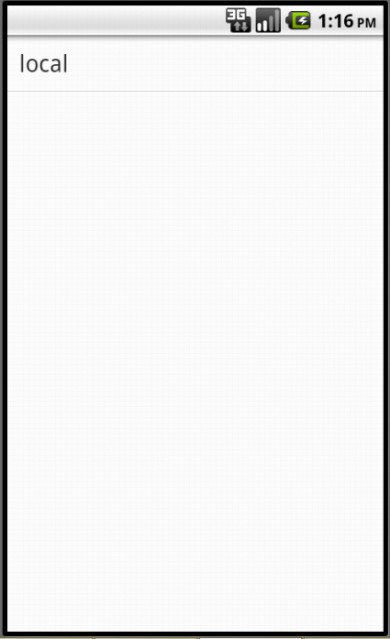
Zapewnia nawigację do wszystkich pozostałych Aktywności.

#### Strona nowego serwera



W tej aktywności możemy właściwie zobaczyć pierwszą różnicę w GUI, w porównaniu do Windows Phone’a. Podstawowe TextBoxy w Androidzie pozwalają na umieszczanie podpowiedzi, co powinno być w nie wpisane, podczas gdy Windows Phone nie posiada ani tej funkcjonalności, ani też żadnej dobrej alternatywy dla niej.

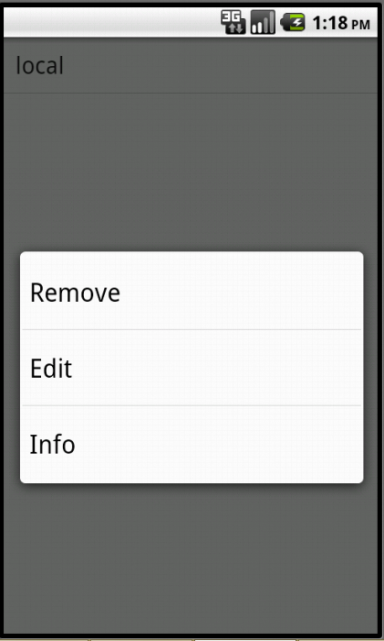
#### Lista serwerów



Zawiera listę serwerów.

Jeśli serwer przyciśnie krótko jedną z opcji na liście, aplikacja rozpoczyna połączenie z serwerem.

#### Lista serwerów – menu kontekstowe

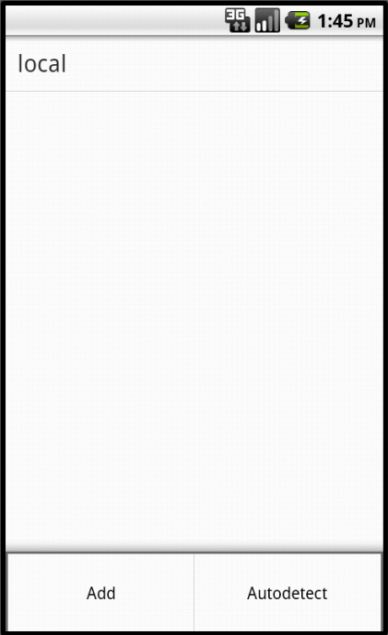


Jeśli użytkownik przyciśnie jedną z opcji na liście dłużej, pojawia się menu kontekstowe umożliwiające usunięcie serwera z listy, jego edycję oraz wyświetlenie informacji o danym serwerze.

#### Lista serwerów – informacje o serwerze

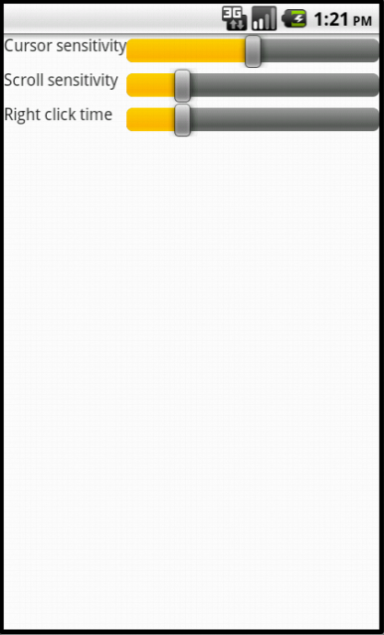


#### Lista serwerów – menu standardowe

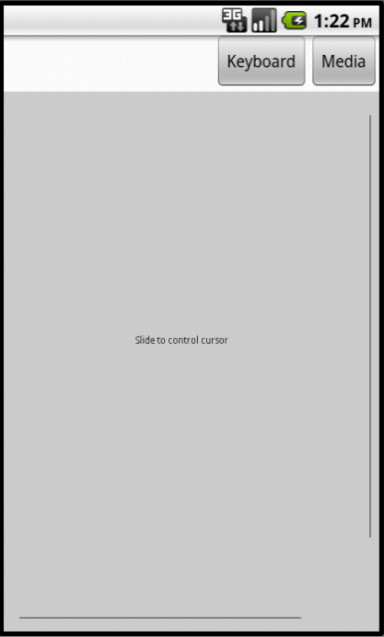


Jeśli użytkownik przyciśnie fizycznie wbudowany w urządzenie przycisk menu, na ekranie pojawia się menu dające możliwość dodania nowego serwera, lub ich automatycznego wykrycia ich w sieci.

#### Strona konfiguracji



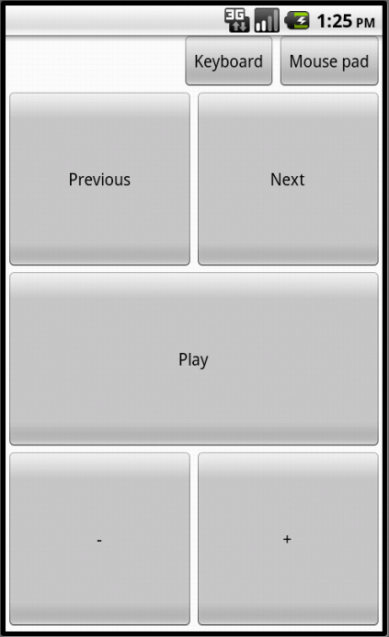
#### Zdalny touchpad



#### Klawiatura



#### Zdalne media



# Wybór i opis technologii

## Wybór technologii

Możesz się zastanawiać, dlaczego w wymaganiach niefunkcjonalnych aplikacji serwerowej znajduje się wzmianka o Windowsie, a o Linuxie nie, pomimo faktu, że jest on napisany przede wszystkim przy pomocy Qt Frameworka, który umożliwia tworzenie wieloplatformowych aplikacji w C++. Problem jednak polega na tym, że pomimo faktu, że Qt Framework jak najbardziej ułatwia tworzenie aplikacji okienkowych, które potem łatwo jest przystosować do pracy w różnych systemach operacyjnych, to jednak nie posiada możliwości wygenerowania zdarzeń systemowych takich jak kliknięcie przycisku myszy, czy przyciśnięcie przycisku na klawiaturze. Co więcej, większość dystrybucji linuksowych też nie dostarcza żadnych standardowych bibliotek, które dostarczałyby tego typu funkcjonalność, podczas gdy jest ona zaimplementowana w WinAPI.

W takim razie dlaczego w ogóle używać C++’a? Aplikacje napisane w tym języku są bardzo wydajne, a wydajność w przypadku aplikacji, które mają działać w tle bez przerwy od momentu uruchomienia komputera, jest bardzo ważna. Z pewnością żaden użytkownik nie chce, żeby aplikacja działająca w tle, która przez większość czasu i tak nic konkretnego nie robi, zużywała powiedzmy 20 MB pamięci. Co więcej aplikacje napisane w C++ nie potrzebują wcześniejszej instalacji żadnego dodatkowego oprogramowania. Większość potencjalnych użytkowników aplikacji prawdopodobnie cechuje się stosunkowo niewielką wiedzą z dziedziny informatyki i dodatkowy wymóg wcześniejszej instalacji np. JRE mógłby być dla nich dosyć kłopotliwy.

Jeśli chodzi o aplikacje klienckie, Android został wybrany dlatego, że jest on obecnie najbardziej rozpowszechnionym systemem operacyjnym dla urządzeń mobilnych i choć mimo wersja 2.2 jest już troszkę podstarzała, to wiąż jest najpopularniejszą wersją tego właśnie systemu.

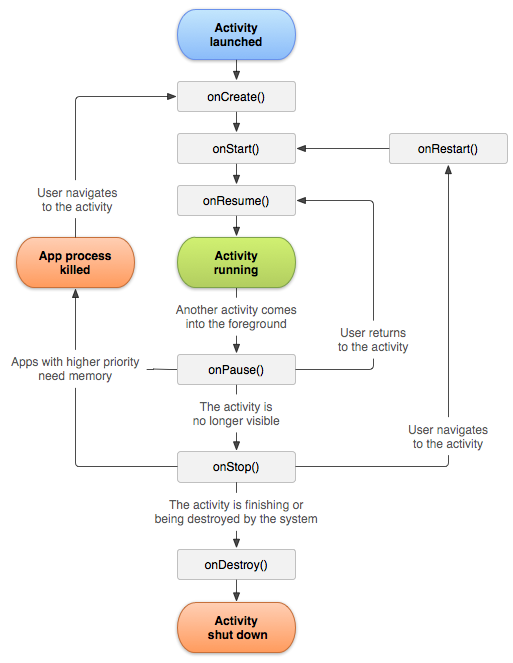
Windows Phone również ostatnimi czasy zyskuje coraz więcej na popularności, a w Microsoft Store wciąż znajduje się bardzo niewiele aplikacji natywnych dla tego systemu operacyjnego.

## Android

### Podstawy

W systemie Android istnieją 3 podstawowe encje: aktywność, serwis oraz widget. Co prawda PCRemote wykorzystuje tylko aktywności, ale pozostałe są niemal równie często używane w aplikacjach Androidowych.

Aktywność jest podstawowym okienkiem, które widzimy w aplikacji i charakteryzuje się następującym cyklem życia:



Widgety jak łatwo można się domyślić, są to elementy aplikacji, które łatwo można dodawać do głównego pulpitu systemu, zaś serwisy, to tak naprawdę osobne aplikacje mogące działać w tle nawet gdy nasza aplikacja wydaje się być wyłączona.

Dodatkowo każdy widget, serwis i aktywność muszą być ręcznie zdefiniowane przez programistę w manifeście (AndroidManifest.xml). Jeśli się o tym zapomni, w momencie gdy aplikacja wywoła encję, zostanie zgłoszony błąd i aplikacja zostanie zakończona przez system.

### Zasoby

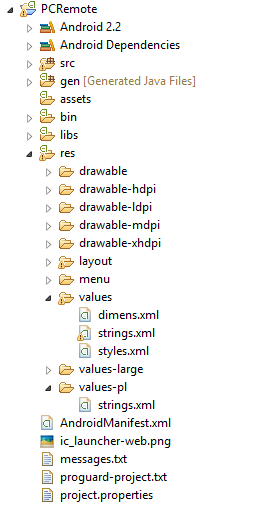
#### Języki

Android udostępnia bardzo prosty i wydajny mechanizm tworzenia aplikacji wielojęzykowych, opierający się w zasadzie na nazewnictwie katalogów.

Podstawowy katalog nazywa się values i zawiera wszelkie domyślne stringi i style, które będą używane w aplikacji. Możemy jednakowoż utworzyć dodatkowy katalog, którego nazwa przestrzega następującej konwencji:

values-<dwuliterowy skrót kraju>

Dla przykładu załóżmy, że mamy katalog values-pl. Jeśli więc stworzymy w nim plik XML zawierający stringi, które mają takie same identyfikatory jak w przypadku stringów umieszczonych w katalogu values, to jeśli użytkownik ma język systemu ustawiony na polski, wszelkie stringi jakie zostaną wyświetlone w aplikacji zostaną zaczerpnięte z katalogu values-pl (chyba, że jakiś nie został przetłumaczony; wtedy użyta jest wartość domyślna z katalogu values)



#### Obrazy

W trakcie budowania aplikacji, kiedy chcemy wyświetlić jakiś obraz, możemy skorzystać z jednej z dwóch opcji: wstawić ów obraz raz w wysokiej rozdzielczości, co zapewni jego ostrość niezależnie od tego, na jak dużym ekranie aplikacja zostanie wyświetlona, lub wstawienie tego samego obrazu do aplikacji kilka razy w różnych rozdzielczościach. Jak się okazuje, pierwsze rozwiązanie nie zawsze jest optymalne. Skalowanie obrazu o wysokiej rozdzielczości wymaga dość sporej mocy obliczeniowej, co mogłoby powodować znaczne opóźnienia w działaniu aplikacji zwłaszcza jeśli chcielibyśmy wyświetlać kilka obrazów na raz.

Z pomocą na szczęście przychodzi nam framework. W katalogach drawable-xhdpi, drawable-hdpi, drawable-mdpi oraz drawable-ldpi możemy wstawić kopie tego samego obrazu w różnych rozdzielczościach (odpowiednio w bardzo dużej rozdzielczości, dużej, średniej, lub niskiej), a w zależności od urządzenia, na którym będzie uruchomiona aplikacja, system sam podejmie decyzję o tym, który obraz powinien zostać wczytany.

Ważną zasadą, której trzeba tutaj przestrzegać jest nazewnictwo plików. Obrazy, które chcemy dodać do naszej aplikacji nie mogą zawierać żadnych białych znaków ani dużych liter.

#### Układy

W katalogu layout możemy znaleźć pliki XML definiujące zawartość danej aktywności. Jednakowoż każda aktywność domyślnie może być przedstawiona zarówno pionowo jak i poziomo, co w przypadku małego ekranu, który posiada większość urządzeń z Androidem, sprawia, że w co najmniej jednym z tych położeń stworzony przez nas układ kontrolek może się wydawać mało czytelny, dlatego też istnieje prosta opcja umożliwiająca stworzenie osobnych układów dla pojedynczej aktywności w zależności od tego jak zostanie odwrócony ekran. Aby uzyskać taki efekt, wystarczy stworzyć dodatkowe dwa katalogi layout-portrait oraz layout-horizontal, gdzie będą się znajdować pliki XML o tych samych nazwach, ale mogą definiować nieco inny układ kontrolek, który w dowolnej pozycji pozwoli na optymalne wykorzystanie przestrzeni ekranu.

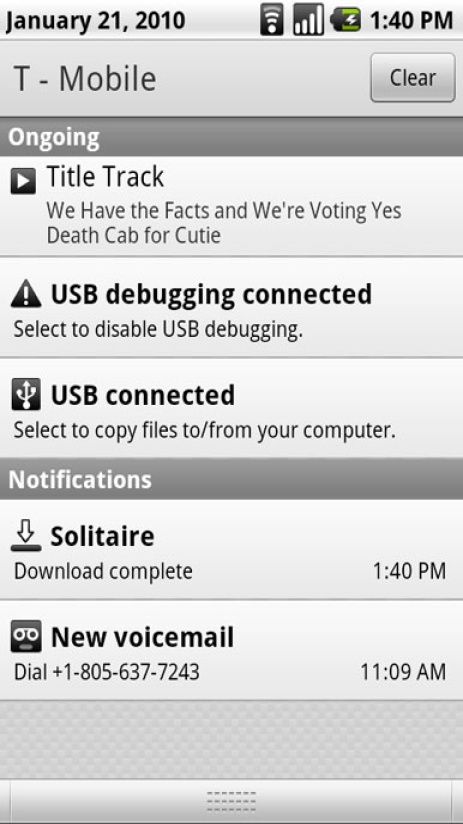
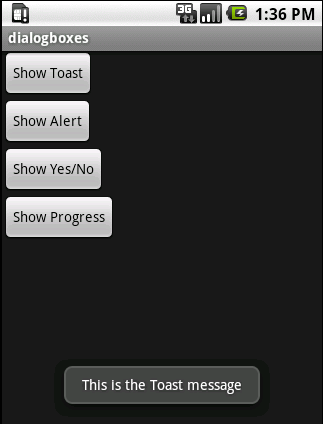
### Kontrolki (View)

#### Zakładki

Jedną z bardziej charakterystycznych kontrolek pojawiających się w Androidzie, a których nie ma w Windows Phonie są zakładki. Niestety są one jednak tworzone w dość mało wydajny sposób. Stworzenie pojedynczej aktywności wymaga wielu zasobów, zaś w przypadku zakładek jest tworzona jedna aktywność przechowująca zakładki i kolejne osobne dla zawartości każdej z zakładek. Ponadto, jako że Android został stworzony też z myślą o słabszych urządzeniach, które mogą mieć dosyć niewielką ilość pamięci podręcznej, w momencie przełączenia zakładki pamięć po nich jest automatycznie zwalniana, co oznacza, że programista ręcznie musi zapisywać stan aktywności a potem go przywracać. Powoduje to też pewien dyskomfort w przypadku niektórych urządzeń, jako że przy przełączaniu się między zakładkami trzeba chwilę odczekać zanim zostanie ona na nowo załadowana do pamięci.

### Powiadomienia

W Androidzie istnieją 2 typy powiadomień: powiadomienia typu toast oraz powiadomienia na „szufladce”.

W przypadku powiadomień typu toast zawsze jest to zwykły ciąg znaków pojawiający się na dole ekranu. Jest to dobry typ powiadomienia, jeśli chcemy uświadomić użytkownika na przykład o tym, że połączenie z serwerem zostało nawiązane, lub zerwane.

Drugi typ powiadomień jest nieco bardziej złożony. Jest wykorzystywany głównie przez serwisy i może w zasadzie zawierać w sobie dowolną ilość różnych kontrolek. Do tego, tego typu powiadomienie można powiązać z aktywnością. W ten sposób, jeśli użytkownik dotknie powiadomienia, zostanie od razu przeniesiony do aktywności, która jest w jakiś sposób związana z tym, czego powiadomienie dotyczyło.

### Framework

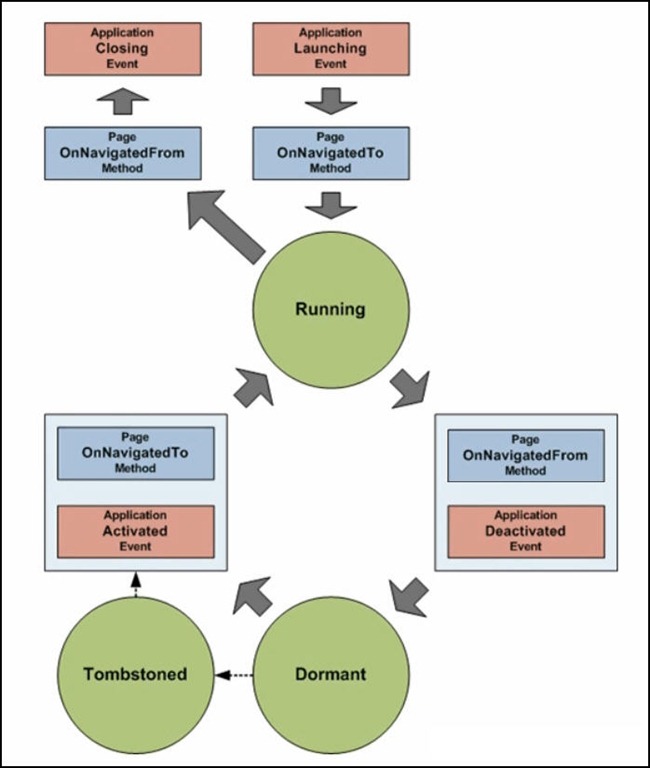
Android 2.2 bazuje teoretycznie na Javie 1.5, ale nie wszystkie funkcje z tej wersji JDK zostały zaimplementowane w Dalviku.

W obecnie używanych wersjach Javy metoda stop() w wątkach jest uważana za przestarzałą, ale w JDK 1.5 jeszcze za taką uważa nie była. Nie zmienia to jednak faktu, że maszyna wirtualna Javy w Androidzie 2.2 po prostu pomija tą funkcję, generując w logach jedynie informację o jej pominięciu.

Nie zostały też zaimplementowane niektóre kontenery, co jednak nie stanowi większego problemu, jako że to akurat zawsze można sobie dopisać samemu.

## Windows Phone

Odpowiednikiem Androidowych aktywności w Windows Phone są strony (Page).



### Języki

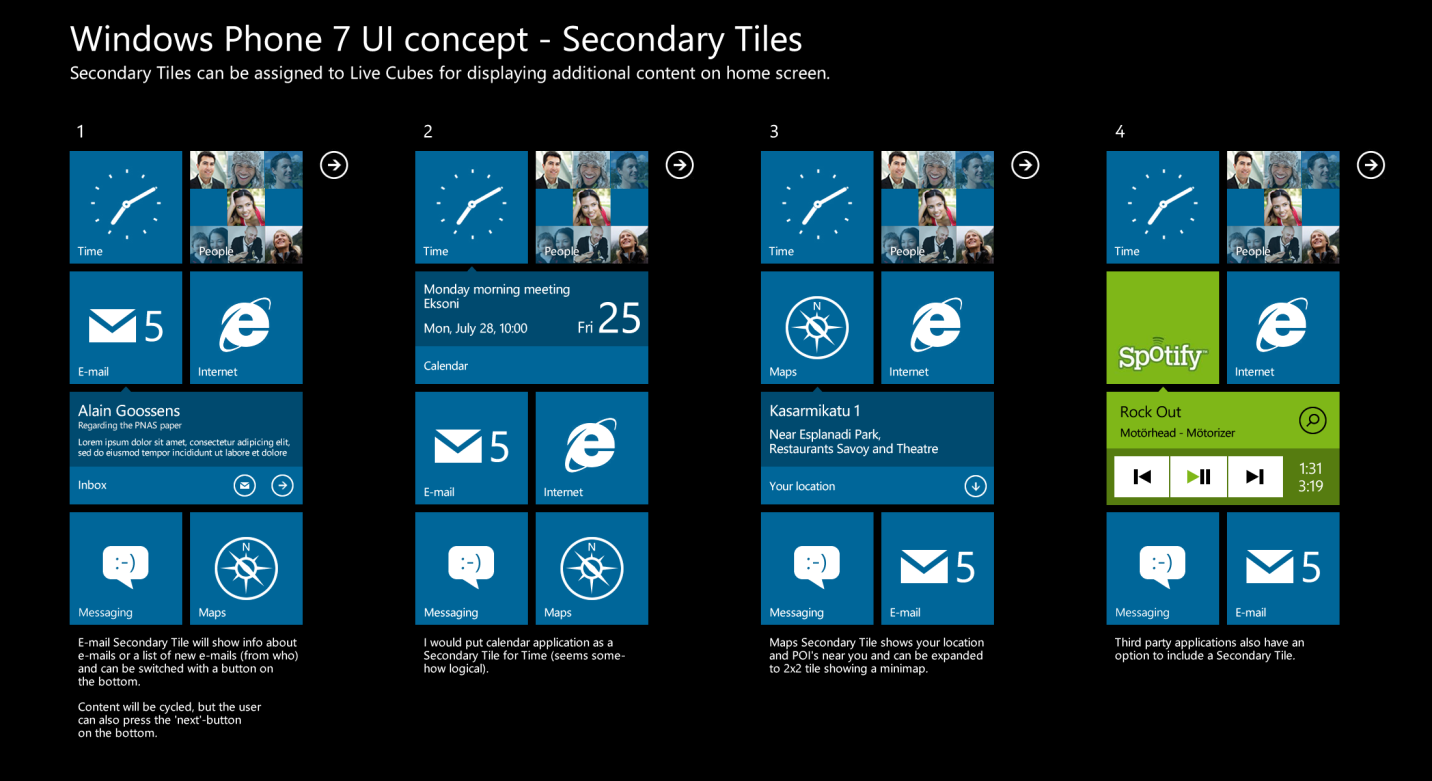
Stworzenie aplikacji wielojęzykowej w Windows Phonie jest dużo bardziej skomplikowane niż w Androidzie.

Aby stworzyć aplikację wielojęzykową konieczne jest przeprowadzanie zmian w konfiguracji, a nawet wyłączenie na chwilę Visual Studio i ręczne edytowanie niektórych plików projektu, jako że Visual Studio nie potrafi tego zrobić sam.

### Powiadomienia

Windows Phone udostępnia przede wszystkim jeden typ powiadomień – powiadomienia na kafelkach.

Każda aplikacja może mieć dowolną większą od zera liczbę kafelków, z czego każdy z nich może przedstawiać inne informacje i prowadzić do innej strony w aplikacji.



Oprócz standardowej ikonki, na kafelku może wyświetlać się liczba. Jest to wykorzystywane np. przy wyświetlaniu ilości nieprzeczytanych wiadomości tekstowych, lub nieodebranych połączeń. Ponadto kafelki mogą periodycznie zmieniać swoją zawartość i wówczas zamiast ikonki przez chwile może być wyświetlany obraz lub tekst.

W systemie Windows Phone istnieją 2 typu kafelków: szerokie i wąskie. Niestety w tej wersji systemu szerokie kafelki zarezerwowane są tylko i wyłącznie dla aplikacji systemowych. Jest to zmienione dopiero w Windowsie 8.

Istnieje też kilka sposobów na aktualizowanie zawartości kafelków. Jednym z nich i najprostszym zarazem jest zwykłe ich aktualizowanie przy pomocy stron aplikacji. Jednakże niewygodnym wymogiem takiego rozwiązania jest fakt, że aplikacja musi być włączona i aktywna, żeby kafelek mógł zostać aktualizowany. W większości wypadków w takim razie, takie rozwiązanie mija się z celem.

Innym, lepszym sposobem na aktualizację informacji wyświetlanych na kafelku jest robienie tego poprzez agentów tła (background agent). Agenci mogą być uruchamiani razem z systemem i mogą wykonywać wszelkie operacje, które często są konieczne do wykonania, gdy sama aplikacja jest wciąż wyłączona (łącznie z aktualizacją kafelków). Problemem jednak jest fakt, że agenci uruchamiani są periodycznie, a ich działanie musi się zakończyć w określonym przedziale czasowym (zależnym od typu agenta). Jeśli operacja wykonywana przez agenta nie zdoła się zakończyć w wyznaczonych ramach czasowych, jest ona automatycznie zabijana przez system.

### Strony

W Windows Phonie 7 istnieje wiele różnych typów stron. Jednym z ważniejszych typów jest na pewno PivotPage. Jest on pewnego rodzaju odpowiednikiem zakładek w Androidzie, z tą różnicą, że zawartość wszystkich zakładek znajduje się na jednej stronie, a tym samym wszystkie zakładki są ładowane na raz.

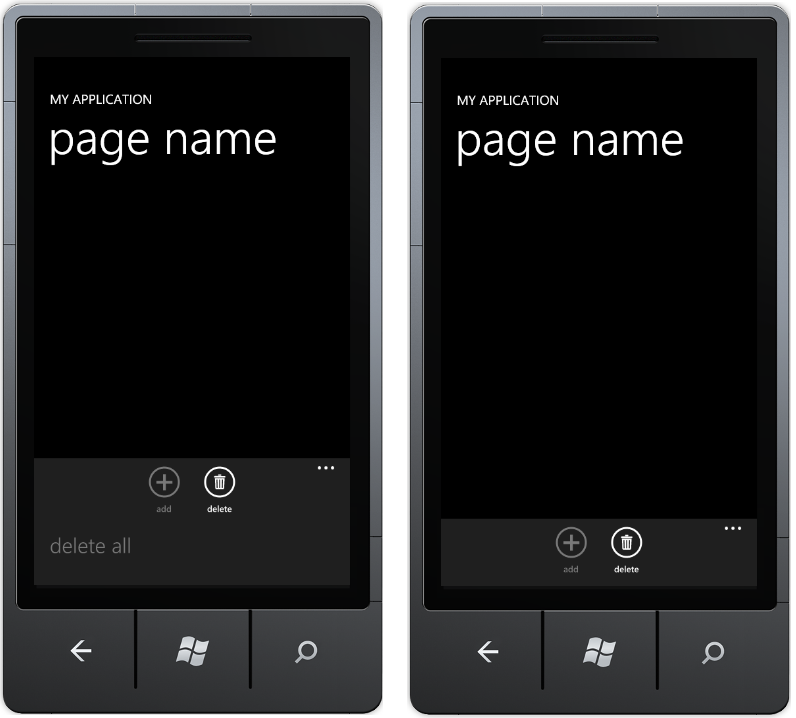
Przejście pomiędzy kolejnymi zakładkami jest wykonywane poprzez wykonanie gestu przesunięcia w lewo lub w prawo.



### Menu

Jak było to wspomniane wcześniej, w systemie Android do wersji 2.3 włącznie istnieje wymóg, by urządzenie posiadało fizyczny przycisk uruchamiający menu standardowe, podczas gdy w Windows Phonie wymóg taki nigdy nie istniał. Było to pokierowane faktem, że fizyczny przycisk menu może być mało intuicyjny. Użytkownik nigdy nie jest informowany o tym, kiedy może użyć przycisku menu, a kiedy nie, poza tym skupia całą swoją uwagę na ekranie, gdzie spodziewa się znaleźć wszystkie ważne opcje i raczej się nie spodziewa, że musi sięgnąć gdzieś poza sam ekran, żeby skorzystać z funkcji, której właśnie poszukuje.

W Windows Phone problem ten został rozwiązany w zupełnie inny sposób. Za każdym razem, gdy użytkownik ma możliwość skorzystania z menu standardowego, w prawym dolnym rogu ekranu wyświetlane są trzy kropeczki po których przyciśnięciu rozwija się lista menu. Oprócz tego na tym samym pasku mogą się znajdować ikony najczęściej używanych opcji, które również są widoczne przez cały czas działania konkretnej strony aplikacji.



Nieco gorzej się ma jednak sytuacja, jeśli chcemy stworzyć menu kontekstowe. Jak się okazuje, .NET Core nie zawiera żadnych klas, które umożliwiałyby stworzenie i obsługę takiego typu menu. W tym celu można próbować używania standardowych funkcji rysujących, żeby stworzyć swoje własne menu, lub prościej, zainstalować Silverlight Toolkit, który już wszelkie tego typu klasy zawiera. Tak więc o ile z punktu widzenia użytkownika wykorzystywanie menu kontekstowego może się wydawać proste, intuicyjne i wykorzystywane tak samo jak w Androidzie (pojawia się po dłuższym przyciśnięciu elementu), o tyle z punktu widzenia programisty jest to nieco bardziej skomplikowane, jako że wymaga instalacji dodatkowych pluginów które, wydawać by się mogło, powinny się znajdować w standardzie frameworka.

## Ważne różnice w technologiach

### Pochłanianie zdarzeń dotyku

W przypadku aplikacji PCRemote, gdzie używana jest niestandardowa kontrolka taka jak touchpad, ważne jest właściwe przeprowadzenie pochłonięcia (konsumpcji) zdarzenia dotyku. Duże znaczenie ma to zwłaszcza w przypadku aplikacji klienckiej dla Windows Phone’a, gdzie touchpad jest umieszczony na stronie typu pivot page. Jeśli zdarzenie nie zostanie pochłonięte w takim przypadku, kiedy użytkownik będzie chciał przesunąć kursor w prawo lub w lewo, jednocześnie będzie zmieniał zakładkę, jako że zdarzenie dotyku najpierw zostanie wysłane do kontrolki, a następnie jeszcze raz będzie przekazane do jej kontenera, czyli w tym przypadku strony.

W Androidzie i w Windows Phonie pochłanianie zdarzenia dotyku rozwiązane jest w zupełnie inny sposób.

W Androidzie na samym początku, gdy tylko chcemy przetwarzać zdarzenie dotyku od razu napotykamy się na metodę:

boolean onTouchEvent(MotionEvent e)

W tym przypadku dokumentacja od razu mówi, że argument przekazywany do metody mówi, jakiego typu jest to zdarzenie (początek dotyku, jego kontynuacja, lub koniec), a wartość zwracana przez metodę mówi, czy dotyk został pochłonięty ( <http://developer.android.com/reference/android/view/View.html#onTouchEvent(android.view.MotionEvent> ). Jest to bardzo proste i szybkie rozwiązanie, gdzie wszystko, co może interesować programistę znajduje się właściwie w jednym miejscu w dokumentacji. W przypadku Windows Phone’a jest to jednak bardziej skomplikowane. Istnieje cała seria różnych zdarzeń, które trzeba zaimplementować, a są to:

- MouseMove

- ManipulationComplete

- ManipulationDelta

- ManipulationStarted

O ile zdarzenie MouseMove mogłoby spokojnie zastąpić swoją funkcjonalnością metodę onTouchEvent w Androidzie, o tyle zaimplementowanie samego tego jednego zdarzenia nie jest wystarczające, żeby pochłonąć zdarzenie dotyku. Dlatego konieczna w tym przypadku jest implementacja wszystkich powyższych zdarzeń. Każde z nich dostaje odpowiedni dla siebie argument opisujący zdarzenie i w każdym z tych argumentów, w celu wykonania pochłonięcia należy ustawić następującą właściwość:

event.Handled = true

### Zarządzanie energią

W obu systemach operacyjnych polityka zarządzania energią jest rozwiązana w zupełnie inny sposób.

W Androidzie tryb oszczędnościowy jest opcjonalny. Sprowadza się on jednak tylko do zmniejszenia jasności ekranu oraz do spowolnienia zegara procesora. Oznacza to, że polityka zarządzania energią w tym systemie nie ma żadnego większego wpływu na działanie pojedynczych zainstalowanych na urządzeniu aplikacji. Jednakże programiści tworzący swoje aplikacje dla Windows Phone’a muszą mieć zawsze na względzie tamtejsze rozwiązania dotyczące oszczędzania energii.

Pierwszym problemem jest fakt, że gdy tylko ekran zostanie zablokowany, wszystkie aktualnie uruchomione aplikacje zostają zatrzymane. Prawo na dalsze działanie mają tylko niektórzy agenci, ale też nie wszyscy. Jest to widoczne nawet w momencie testowania aplikacji. Jeśli ekran urządzenia jest zablokowany (a tym samym agent odpowiedzialny za komunikację poprzez port USB jest zatrzymany), a spróbujemy przeprowadzić instalację naszej aplikacji poprzez Visual Studio, otrzymamy jedynie komunikat, że instalacja nie może zostać przeprowadzona i że telefon musi zostać odblokowany.

Drugim dość poważnym utrudnieniem jest też fakt, że w momencie zablokowania urządzenia, zrywane są wszelkie połączenia sieciowe. W ten sposób wszyscy agenci, którzy korzystają z połączenia sieciowego stają się bezużyteczni, aż do momentu odblokowania telefonu (przy czym konieczne jest wtedy nawiązywanie połączenia od nowa). Jest to też znaczne utrudnienie dla aplikacji PCRemote jako, że po odblokowaniu telefonu, konieczne jest na nowo tworzenie sesji.

### Wzorce

Wszelkie technologie Microsoftu charakteryzują się tym, że do granic możliwości upraszczane w nich jest przedstawianie wszelkiego rodzaju danych. Windows Phone nie stanowi w tym przypadku wyjątku.

O ile w Adroidzie, kiedy tworzymy listę elementów, często musimy najpierw stworzyć kontener na liczne obiekty, a potem ręcznie dodawać reprezentację każdego z tych obiektów do listy widzianej przez użytkownika, co często jest rozwiązane przez adaptery, które bywają trudne w użytkowaniu, o tyle w Windows Phonie mamy możliwość skorzystania z bardzo wygodnych kolekcji obserwowalnych oraz wzorców.

Najpierw każdy wzorzec musi zostać opisany w pliku XAML opisującym zawartość strony. Już we wzorcu możemy podawać nazwy zmiennych zawartych w klasie, której kolekcja obiektów potem zostanie przekazana do listy.



W następnym kroku jedyne co należy zrobić to przekazać do naszej listy (ListBox, dostępny z kodu C# pod nazwą „listbox” jako, że taka jest wartość atrybutu x:Name) obserwowalną kolekcję zawierającą obiekty dowolnej klasy, pod warunkiem, że zawiera ona zmienne o nazwach, które są używane przy słowie kluczowym „Binding” w naszym XAMLu.

Obserwowalna kolekcja jest znowuż niczym innym jak listą, która generuje odpowiednie zdarzenia za każdym razem, gdy jest do niej dodawany lub usuwany element. Jest wskazanym, by każda klasa, której obiekty są umieszczane w kolekcji obserwowalnej, też była obserwowalna (implementuje interfejs INotifyPropertyChanged i przy setterze każdej właściwości generuje zdarzenie PropertyChanged). Dzięki temu, jeżeli któraś z wartości obiektu zostanie zmieniona, zmiana jest automatycznie aktualizowana też w liście.

## Technologie alternatywne

Obecnie na rynku istnieje wiele rozwiązań alternatywnych dla zwykłych technologii natywnych. Jednym z najpopularniejszych jest na przykład jQuery Mobile, dzięki któremu można tworzyć aplikacje w zasadzie na każdą platformę przy użyciu HTMLa i JavaScripta. W wielu wypadkach mogłoby się to wydawać genialne rozwiązanie, kiedy chcemy tworzyć naszą aplikację tylko raz i uruchamiać ją na każdej platformie, jednakże tego typu rozwiązania zawsze mają swoje ograniczenia w porównaniu do rozwiązań natywnych.

W jQuery Mobile nigdy nie uda nam się uruchomić wibracji telefonu, nie da się korzystać z nawigacji, ani też z akcelerometru, żyroskopu, czy termometru ani wielu innych funkcji, których się raczej nie wykorzystuje podczas pisania stron internetowych. Do tego dochodzi fakt, że cała logika aplikacji jest stworzona w JavaScripcie, który jest uruchamiany na telefonie. Po pierwsze utrudnia to znacznie debugowanie aplikacji. Nawet jeśli funkcja działa poprawnie w przeglądarce komputera, nie znaczy to, że będzie ona działała równie poprawnie na telefonie, jako że interpreter na telefonie często miewa ograniczoną funkcjonalność. Po drugie JavaScript jest językiem interpretowanym, a co za tym idzie, znacznie wolniejszym niż skompilowane aplikacje stworzone przy pomocy technologii natywnych.

Różne firmy próbują naprawić ten problem tworząc bardziej złożone frameworki. Tak powstał np. APM Studio (rozwijany przez firmę Antenna Software), dzięki któremu możliwe jest stworzenie aplikacji na każdą platformę przy pomocy jedynie Javy, za pośrednictwem bardzo wygodnego w użyciu API. Dzięki temu, ograniczenia, które pojawiają się przy rozwiązaniach przeglądarkowych są znacznie zredukowane, programowanie staje się dużo szybsze i łatwiejsze niż w przypadku tworzenia aplikacji natywnych (zaimplementowane API zawiera bardzo wiele funkcjonalności znacznie ułatwiających pisanie zwłaszcza aplikacji sieciowych), ale sama technologia nie jest dostępna dla przeciętnego amatora programowania, gdyż jej wykorzystanie kosztuje wiele tysięcy dolarów.

# Implementacja

## Wiadomości sieciowe

W projekcie zostały użyte dwa różne protokoły warstwy transportowej: TCP i UDP.

Powodem dla którego zostało to rozwiązane właśnie w ten sposób jest fakt, że w przypadku TCP przeprowadzane jest wiele operacji mających na celu zapewnienie, że dane zostały przesłane we właściwy sposób, co jednak generuje poważne opóźnienia w dostarczeniu wiadomości do aplikacji, a co za tym idzie, poruszanie kursorem mogłoby się stać wyjątkowo niewygodną czynnością.

Dlatego też ta pojedyncza operacja została rozwiązana przy pomocy protokołu UDP.

### Wiadomości przesyłane przy pomocy protokołu TCP

All messages presented below are to be sent from client to server only.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Message name | Number of bytes | Bytes description | | |
| 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 |

### Wiadomości przesyłane przy pomocy protokołu UDP

#### Wiadomości wysyłane przez klienta

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Message name | Number of bytes | Bytes description | | |
| Operation identifier | Operation | Operation |
| Mouse move | 3 | 1 | X coordinate | Y coordinate |
| Server scan request | 2 | 0 | 1. Android 2. Windows Phone | - |

### Wiadomości wysyłane przez serwer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Message name | Number of bytes | Bytes description | | |
| Operation identifier | Operation | Operation |
| Server scan response | n | 0 | 0 | <Name of serverer>;<port> |

## Kody przycisków

|  |  |
| --- | --- |
| 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 |

## Testy

### Android

Dokumentacja Androida doradza, by do testów jednostkowych używać frameworku JUnit, który jest najpopularniejszym frameworkiem do tworzenia testów dla Javy ( <http://developer.android.com/tools/testing/testing_android.html> ).

Przy tworzeniu tego typu testów należy stworzyć osobny katalog ze źródłami o nazwie „tests”. W katalogu powinny się znajdować paczki o analogicznych nazwach do tych, w których znajdują się klasy, które chcemy testować. Plik zawierający testy powinien mieć nazwę taką samą jak klasa, którą chcemy testować z sufiksem „Test”. Tak więc jeśli chcemy testować klasę XXX, klasa zawierająca testy dla niej, powinna się nazywać XXXTest. Metody zawarte w klasie również powinny mieć nazwy analogiczne do tych, które mają testować, z tą drobną różnicą, że powinny zawierać prefiks „test”. Na przykład Jeśli chcemy testować metodę foo(), to metoda testująca działanie funkcji foo() powinna się nazywać testFoo().

Testy pisane przy użyciu frameworka JUnit charakteryzują się własnym cyklem życia.

Pierwsza w klasie uruchamiana jest metoda posiadająca anotację @BeforeClass. Jest ona pewnego rodzaju konstruktorem klasy, w której powinniśmy przygotować wszystkie testy do poprawnego działania. Metoda zawierająca anotację @BeforeClass powinna być statyczna.

@BeforeClass public static void konstruktor(){}

Następnie uruchamiane są testy. Każdy z nich powinien zawierać anotację @Test. Anotacja opcjonalnie może zawierać argumenty expected oraz timeout. Expected reprezentuje wyjątek, który powinien zostać zwrócony. Jeśli żaden wyjątek nie zostanie zwrócony, lub wyjątek jest inny niż w założeniu, test jest zakończony niepowodzeniem.

@Test (expected = NullPointerException.class) public void foo(){}

Drugim opcjonalnym parametrem anotacji @Test jest timeout. Reprezentuje on liczbę milisekund, podczas którego to czasu metoda powinna się zakończyć. Jeśli jej wykonanie trwa dłużej, test kończy się niepowodzeniem.

@Test (timeout = 1000) public void foo(){}

Jeżeli do anotacji nie zostaną przekazane żadne dodatkowe parametry, test zostanie zakończony powodzeniem tylko wtedy, jeśli metoda się zakończy bez wyrzucania żadnych wyjątków.

Na samym końcu konieczne jest zwolnienie zasobów, które przygotowaliśmy sobie na samy początku przy pomocy metody z anotacją @BeforeClass. Służy do tego statyczna metoda z anotacją @AfterClass.

@AfterClass public static void destruktor(){}

### Windows Phone

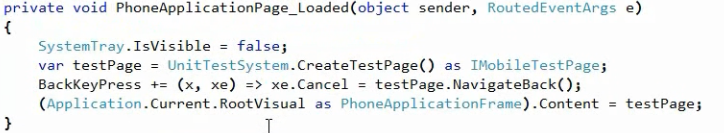
Windows Phone jest oparty tak naprawdę o 2 różne technologie Microsoftu: C# (i tym samym .NET Framework) oraz Silverlight.

Powoduje to, że aby stworzyć profesjonalny test dla aplikacji Windows Phonowej, potrzebujemy odpowiedniego frameworka, który stworzony został właśnie dla technologii Silverlight – Silverlight Unit Test Framework.

Aby rozpocząć tworzenie testów dla naszej aplikacji musimy najpierw stworzyć w naszej solucji osobny projekt dla Windows Phone’a, dodać do niego referencje do aplikacji, którą chcemy testować, oraz do plików dll frameworka testowego.

Przechodzimy do źródła głównej strony nowo utworzonej aplikacji i dodajemy metodę, która powinna zostać uruchomiona wraz z załadowaniem strony (należy ją przypisać do zdarzenia Loaded strony).

Metoda powinna wyglądać mniej więcej w następujący sposób:



W ten sposób zostanie załadowana domyślna strona testowa.

Następnie w tej samej przestrzeni nazw należy umieścić klasę dziedziczącą klasę SilverlightTest i posiadającą atrybut [TestClass], która zawiera publiczne metody, z których każda jest typu void i posiada atrybut [TestMethod]. Każda metoda powinna posiadać taką samą nazwę jak metoda, którą testuje wraz z prefiksem „Test”. Tak więc metoda testująca metodę o nazwie Foo() powinna się nazywać TestFoo().

Jeśli dana metoda ma wyrzucać wyjątek, istnieje możliwość dodania dodatkowego atrybutu „ExpectedException”.

[TestMethod]

[ExpectedException(typeof(NullReferenceException))]

public void TestFoo(){}

Po uruchomieniu projektu (z wyłączony debuggerem), wszystkie testy zostaną automatycznie wykonane i wynik zostanie wyświetlony na pierwszej stronie.

## Urządzenia testowe

Choć emulatory Androida i Windows Phone’a są często bardzo przydatne podczas tworzenia aplikacji, konieczne jest zawsze przetestowanie aplikacji na fizycznych urządzeniach jako, że czasami może się ona zachowywać na nich nieco inaczej niż w emulatorze.

PCRemote został testowany na następujących urządzeniach:

1. Nokia Lumia 710



|  |  |
| --- | --- |
| System operacyjny | MS Windows Phone 7.5 Mango |
| Procesor | Qualcomm MSM8255T |
| Zegar procesora | 1,4 GHz |
| RAM | 512 MB |
| Pamięć wbudowana | 8GB |
| Ekran | 480 x 800 px (3,7‘’) |

1. Samsung Galaxy Ace



|  |  |
| --- | --- |
| System operacyjny | Android 2.3 Gingerbread |
| Procesor | Qualcomm MSM7227 |
| Zegar procesora | 800 MHz |
| RAM | 384 MB |
| Pamięć wbudowana | 158 MB |
| Ekran | 320 x 480 px (3,5‘’) |

## Narzędzia użyte do tworzenia aplikacji

W celu stworzenia projektu zostało użyte wiele różnych narzędzi pozwalających na tworzenie w łatwy sposób aplikacji w danym języku wymaganym do konkretnej aplikacji.

1. Qt Creator – użyty do stworzenia serwera. Pozwala na łatwe pisanie programów w C++ wraz z Qt Frameworkiem
2. Visual Studio 2010 – użyty do stworzenia klienta dla Windows Phone’a. Choć istnieją narzędzia alternatywne do tworzenia Aplikacji dla Windows Phone’a, to jest najlepiej udokumentowane i najprostsze w użyciu.
3. Eclipse – użyty do stworzenia aplikacji klienckiej dla Androida. Jest to bardzo popularne i proste w użyciu środowisko do programowania w Javie. Oficjalny tutorial dla Androida bazuje na tym narzędziu.

# Podsumowanie

Stworzenie trzech osobnych aplikacji, które będą się potrafiły komunikować w sieci nie jest łatwe, ale z pewnością warte podjęcia próby.

Teraz dzięki projektowi można łatwiej przedstawiać prezentacje multimedialne. Aplikacja pozwala na pełną kontrolę kursora, dzięki czemu można w łatwy sposób wskazywać ważniejsze elementy umieszczone na kolejnych slajdach oraz samą zmianę slajdów. Pozwala też na łatwe przedstawianie treści umieszczonych poza prezentacją, takich jak strony internetowe, lub nagrania audio i wideo.

Należy nadmienić, że praca nad projektem pozwoliła mi na znaczne poszerzenie mojej wiedzy z zakresu programowania aplikacji dla Windows Phone’a oraz Androida, a także tworzenia testów jednostkowych.

# Słowniczek

1. Klient – każda aplikacja zdolna do komunikacji z serwerem PCRemote.
2. Urządzenie mobilne – dowolne urządzenie działające pod systemem operacyjnym Windows Phone (w wersji 7.x) lub Android (w wersji co najmniej 2.2) z zainstalowaną aplikacją kliencką PCRemote.
3. Kolejka wiadomości – kolejka (First-In-First-Out) zawierająca obiekty klas implementujących interfejs NetworkMessage.
4. Komunikat przedstawienia się – komunikat reprezentowany przez klasę IntroductionMessage. Jest to pierwszy komunikat wysyłany przez klienta automatycznie zaraz po nawiązaniu połączenia TCP z klientem. Zawiera on przede wszystkim informacje o wersji klienta (numer wersji oraz kod systemu operacyjnego pod którym działa klient).
5. Przyciski mediów – przyciski pojawiające się tylko na niektórych klawiaturach. Służą do rozpoczęcia lub wstrzymania odtwarzania ścieżki oraz przejścia do następnej lub poprzedniej ścieżki (gdzie ścieżka jest aktualnie odtwarzanym plikiem audio lub wideo).

# Bibliografia

1. Android developers – oficjalna dokumentacja - <http://developer.android.com> (zawiera bardzo wiele informacji ogólnych dotyczących Javy oraz pełną dokumentację wraz z przykładami dotyczącymi Androida)
2. Oficjalna dokumentacja JUnit - <http://www.junit.org/>
3. Microsoft Developer Network (MSDN) - <http://msdn.microsoft.com/en-US/> (zawiera pełną dokumentację C#, .NET Frameworku, Silverlight, Windows Phone, Silverlight Unit Test Framework wraz z przykładami)
4. Qt Project - <http://qt-project.org/doc/qt-4.8/tutorials.html> (zawiera oficjalną dokumentację oraz tutoriale dla Qt Frameworka)

# Dodatek 1 – Rozwój projektu

|  |  |
| --- | --- |
| Data | Komentarz |
| 1.08.2012 | Rozpoczęcie pracy nad klientem dla Androida |
| 3.08.2012 | Rozpoczęcie pracy nad serwerem; stworzenie możliwości nawiązania połączenia klienta i serwera; stworzenie ikony tray serwera |
| 6.08.2012 | Stworzenie klasy SlidingWindow dla serwera i utworzenie pierwszych powiadomień o nawiązaniu połączenia |
| 7.08.2012 | Stworzenie pierwszej wiadomości pomiędzy serwerem i klientem – przedstawianie się klienta |
| 12.08.2012 | Stworzenie aktywności dodawania serwerów do listy; stworzenie listy serwerów |
| 14.08.2012 | Trwałe zapisywanie listy serwerów |
| 15.08.2012 | Dodanie menu standardowego i kontekstowego do listy serwerów |
| 21.08.2012 | Stworzenie touchpada dla Androida, odpowiednich komunikatów dla serwera, funkcje przetwarzające komunikaty na serwerze (wszystko przy pomocy protokołu TCP; widoczne znaczne opóźnienia w przesyłaniu, utrudniające użytkowanie aplikacji) |
| 23.08.2012 | Utworzenie klawiatury dla Androida wraz z całą komunikacją z serwerem |
| 24.08.2012 | Utworzenie przycisków do obsługi mediów |
| 26.08.2012 | Stworzenie aktywności konfiguracji |
| 10.09.2012 | Rozpoczęcie prac nad klientem dla Windows Phone’a; stworzenie ekranu głównego oraz listy serwerów i możliwość dodawania do niej elementów |
| 12.09.2012 | Trwałe zapisywanie listy serwerów (serializacja do XML’a); nawiązywanie połączenia z serwerem oraz przesyłanie komunikatu przedstawienia się |
| 18.09.2012 | Dodanie MainWindow do serwera. Okno wyświetla liczbę aktualnie połączonych klientów oraz zawiera przyciski obsługi mediów |
| 30.09.2012 | Dodanie obsługi protokołu UDP do serwera oraz przeniesienie obsługi touchpada na ten właśnie protokół w kliencie dla Androida (kursor zaczyna się zachowywać zgodnie z założeniami) |
| 10.10.2012 | Dodanie touchpada do klienta Windows Phone’a |
| 13.10.2012 | Dodanie funkcji automatycznego ponownego nawiązania połączenia po obudzeniu procesu |
| 23.10.2012 | Dodanie obsługi klawiatury dla klienta WP (rozwiązane w inny sposób niż w przypadku klienta. Wymagało dodania zmian na serwerze) |
| 24.10.2012 | Dodanie obsługi przycisków mediów |
| 19.11.2012 | Ukończenie konfiguracji klienta dla Windows Phone’a |
| 28.11.2012 | Zakończenie prac nad bugami w projekcie |
| 1.12.2012 | Zakończenie prac nad elementami graficznymi |
| 23.12.2012 | Zakończenie prac nad dokumentacją do projektu |

# Dodatek 2 – Źródła

Wszystkie źródła aplikacji, są umieszczone i regularnie aktualizowane na serwerze Gita (w tym przypadku konkretnie GitHuba):

<https://github.com/kamiljano/PCRemote.git>