Jani Vuolle

Windows Phone 7:n ja Androidin kommunikointi

Opinnäytetyö

Kevät 2013

Tekniikan yksikkö

KCTITE

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Seinäjoen ammattikorkeakoulun Tekniikan yksikkö

Koulutusohjelma: Tietotekniikan koulutusohjelma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Jani Vuolle

Työn nimi: Älypuhelimien kommunikointi

Ohjaaja: Petteri Mäkelä

Vuosi: 2013 Sivumäärä: 44 Liitteiden lukumäärä:

Opinnäytetyössä tutkittiin älypuhelimien kommunikointia ja datan siirtoa. Älypuhelimet siirtävät tietoa palvelimen kautta toisille älypuhelimille, tämä mahdollistaa keskitetyn ja hallittavan tiedonsiirron. Opinnäytetyössä tutkittiin Windows Phone 7- ja Android-älypuhelimia ja palvelin tekniikoista kolmea eri tekniikkaa. Palvelintekniikoista sovellettiin WCF (Windows Communications foundation)-tekniikkaa. REST (Representational State Transfer)-tekniikan käyttöä harkittiin sekä Web Apin käyttöä tutkittiin.

Avainsanat: WCF, REST, Web Api, Windows Phone 7, Android

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Programming

Author: Jani Vuolle

Title of thesis: Smartphone communications

Supervisor: Petteri Mäkelä

Year: 2013 Number of pages:44 Number of appendices:

In this thesis the communication of smart phones and data transfer systems were researched. Smart phones transfer data through servers to other smart phones, which allow manageable transferring. In this thesis Windows Phone 7, Android smart phones and three different server technologies were researched. WCF was chosen to be the core for the server framework. REST technology was considered as an option and the use of Web API was researched.

Keywords: WCF, REST, Web Api, Windows Phone 7, Android

SISÄLTÖ

Kuvio- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

WCF Windows Communication Foundation. Ohjelmointirajapinta, joka toimii kutsumalla funktioita. Käyttää SOAP-formaattia hyväkseen. Voi toteuttaa myös REST-protokollaa. (MS 2012b.)

REST Representational State Transfer. HTTP-protokolla jolla voidaan toteuttaa ohjelmointirajapintoja esimerkiksi www-palveluissa. (Douglas 2013c.)

SOAP Simple Object Access Protocol. XML-viesti formaattia käyttävä protokolla, joka lähettää viestejä erilaisten tasojen kautta kuten http tai tcp. (Douglas 2013b.)

XML Extensible Markup Language. Kieli, jota pystyy suoraan ihminen ja tietokone lukemaan. Pääkohtina yksinkertaisuus, yleisyys ja käytettävyys internetin välityksellä. (RD 2013b.)

HTTP hypertext transfer protocol. Protokolla jolla voidaan jakaa hypermediaviestejä. Käytetään mm. World Wide Webin tiedonsiirron pohjana. (YY 2010.)

TCP Transmission Control Protocol. Protokolla joka hallitsee tiedon siirtoa tietokoneelta toiselle, kun koneet ovat yhteydessä Internetiin. Toimii IP:n kanssa yhteistyössä. (HA 2011.)

JSON JavaScript Object Notation. XML-tapainen kieli, jolla voidaan tallentaa ja lähettää viestejä. JSON on tosin pienempi kuin XML, joten se on nopeampi lukea. (BE 2007.)

URI Uniform Resource Identifier on merkkijono, jolla voidaan esimerkiksi paikantaa www-sivuja. www-sivujen paikannuksia kutsutaan omalla lyhenteellä, URLilla. (AD 2007.)

URL	Uniform Resource Locatoria voidaan käyttää paikantamaan www-sivuja. (AD 2007.)

1 JOHDANTO

1.1 Työn tausta

Mobiilimaailma kehittyy ja uusia tutkimuskohteita tulee uusien käyttöjärjestelmien myötä. Uudet Lumia-älypuhelimet tulivat markkinoille ja niitä haluttiin tutkia, koska niissä on Windows Phone 7 (WP7) -käyttöjärjestelmä. Windows Phone 7 -käyttöjärjestelmä ei ole ollut vielä kovin kauaa markkinoilla. Tavoitteena oli erilaisten kommunikointityylien tutkiminen ja ohjelmoiminen.

1.2 Työn tavoite

Työn tavoitteena oli tutkia miten älypuhelimet kommunikoivat palvelimien kanssa. Päätavoitteena oli tutkia tekniikka, jolla eri älypuhelinten käyttöjärjestelmät pystyvät kommunikoimaan keskenään. Ensiksi tutkittiin Windows Phone 7 -käyttöjärjestelmää. Myöhemmin Android-käyttöjärjestelmä tuli mukaan, koska se on yksi suosituimpia käyttöjärjestelmä älypuhelimissa. Lisäksi pohdittiin millaisia erilaisia palvelinratkaisuja on tarjolla ja valittiin sopivin. Kun palvelimen runko oli valittu, ohjelmoitiin yksinkertainen ohjelma jolla voitiin testata yhteyttä. Palvelimen ohjelmoinnin jälkeen luotiin yksinkertainen Windows Phone 7 –käyttöliittymä, joka yritti muodostaa yhteyden palvelimelle. WP7-järjestelmän jälkeen ohjelmoitiin Androidille samanlainen ohjelma, joka haki tietoa palvelimelta. Yhteyksien toimivuus testattiin ohjelmoinnin jälkeen. Lopuksi palvelimen ohjelmakoodia parannettiin siten, että se hakee paikkatiedon tietokannasta. WP7-ohjelmakoodia muutettiin käyttämään karttapalvelua ja asettamaan paikka-arvoja käyttäjältä.

1.3 Työn rakenne

Luvussa 2 kuvaillaan lyhyesti WP7- ja Android-älypuhelimien ominaisuuksia. Samalla tutustutaan eri vaihtoehtoihin palvelin ympäristössä.

Luvussa 3 vertaillaan palvelintekniikoita keskenään. Vertailun jälkeen suunnitellaan ja ohjelmoidaan palvelimelle ja älypuhelimelle sovellukset, joilla voidaan testata yhteyksien toimivuus.

Luvussa 4 muodostetaan yhteys palvelimen ja älypuhelimien välille ja testataan toimiiko tiedon siirto.

Luvussa 5 päivitetään ohjelmakoodia palvelimella ja uudistetaan WP7-sovellusta. WP7 asetetaan karttajärjestelmä, johon voidaan merkitä ihmisten paikkatietoja.

2 ÄLYPUHELIMET JA KOMMUNIKOINTI

Älypuhelimet valtaavat normaalien matkapuhelimien markkinoita nopeaa tahtia. Älypuhelimet ovat vahvassa asemassa niiden monipuolisuuden ansioista. Opinnäytetyön teon hetkellä kolme suurta älypuhelinten käyttöjärjestelmää olivat: Google Android, Microsoft Windows Phone ja Apple iOS.

2.1 Windows Phone 7

Windows Phone 7 -käyttöjärjestelmä julkaistiin ensimmäistä kertaa vuonna 2010. Ensimmäiset puhelimet eivät kuitenkaan tulleet Nokialta, vaan muilta yrityksiltä (Ricker 2010). Myöhemmin Nokia aloitti yhteistyön Microsoftin kanssa, jolloin syntyi Lumia-malliset älypuhelimet. Niissä pääkäyttöjärjestelmänä toimii Windows Phone. (HS 2011).

Microsoft on asettanut seuraavat vaatimukset puhelimiin, jotta Windows Phone 7 toimisi oikein laitteessa.

- Puhelimessa tulee olla Start-, Search-, Back-painikkeet ja ohjaimet puhelimen sammuttamiseen sekä äänenvoimakkuuden säätöpainikkeet
- WVGA-näyttö jossa on 800 x 480 resoluutio
- Kosketusnäyttö, joka tukee 4-pistetoimintoja puhelimen hallitsemiseksi
- Tuki puhelimen laajakaistaa tai Wi-Fi-yhteyttä varten
- Vähintään 256 Mb RAM-muistia
- Vähintään 8 Gb kovalevytilaa Flash-muistina
- A-GPS
- Kiihdytyksen tunnistusanturi. (MS 2012a.)

Windows Phone 7 -käyttöjärjestelmässä tuetaan moniajoa vain muutamissa ennalta asennetuissa ohjelmissa. Muulloin käytetään tombstone-tekniikkaa tai isolated storage -tekniikkaa. Tombstone-tekniikka ei poista ohjelmaa kokonaan muistista, kun käyttäjä sulkee sen. Ohjelman sulkeutuessa tallentaa ohjelma tilannetiedot muistiin. Kun käyttäjä painaa Takaisin-painiketta puhelimessa, puhelin lataa tombstonesta vanhat tiedot uudelleen ohjelmaan. Tavallaan se siis tallentaa oh-

jelman tilanteen puhelimen takamuistiin ja lataa tiedot tarpeen vaatiessa. Tombstone voi pitää viiden ohjelman tiedot muistissa yhtä aikaa. Jos ohjelmia on enemmän, niin tilannetiedot täytyy tallentaa isolated storageen. (Poznanski 2011.)

2.2 Android

Android on Linux-pohjainen käyttöjärjestelmä, jonka luonti aloitettiin vuonna 2005 ja joka rahoitettiin ja ostettiin Googlen toimesta. Ensimmäinen Android-älypuhelin tuli markkinoille 2008 vuoden lopussa. Alkuperäisen Android-version nimi oli Donut, jonka versio numero on 1.6. Opinnäytetyön tekohetkellä uusin versio oli 4.2, joka on nimetty Jelly Beaniksi. (SS 2013.)

Android-puhelimissa on pienemmät vaatimukset kuin Microsoftin Windows Phone 7 -käyttöjärjestelmässä. Tosin Google yleensä pyytää valmistajia lisäämään hyödyllisiä ominaisuuksia, kuten Micro-USB-portin älypuhelimiin.

Pakolliset vaatimukset ovat:

- Home-, Menu- ja Back-painikkeet aina saatavilla
- QVGA-näyttö, jossa 240 x 320-resoluutio
- kosketusnäyttö
- 92 Mb Ram (vähintään)
- vähintään 150 Mb kovalevytilaa käyttäjälle
- 2-megapikselin kamera. (Dolcourt 2010.)

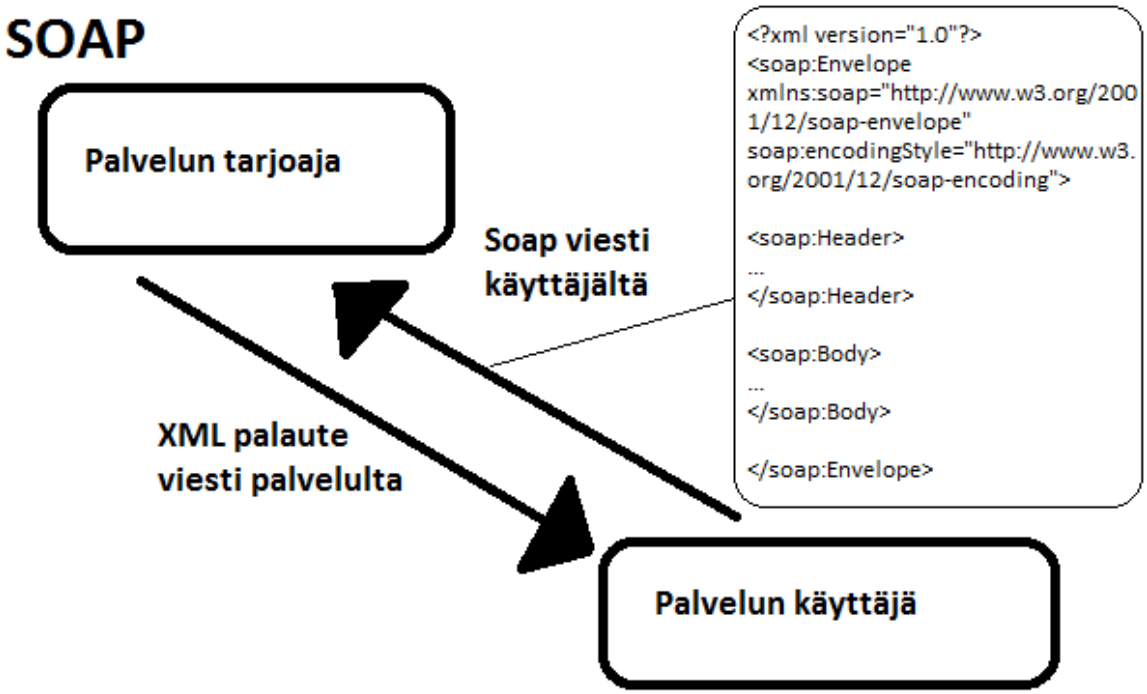
2.3 Palvelintekniikat

Ennen kuin oli vielä keksitty standardia erilaisille datan siirroille, ohjelmoijat tekivät omia ratkaisujaan yhteyksien selvittämiseksi. Nykyään on kuitenkin monia eri tekniikoita, joilla pystytään standardoimaan erilaisia tiedonsiirtotapoja HTTP protokollassa. HTTP (hyper text transfer protocol) on yksi yleisimpiä protokollia, joita käytetään tiedon siirtoon esimerkiksi selaimissa.

2.3.1 WCF service

Windows Communication Foundation (WCF) on runko, jolla voidaan rakentaa palvelu-painotteisia sovelluksia SOAP-arkkitehtuurilla toteutettuna. WCF on joustava kokonaisuus, koska uusien tarpeiden ilmaantuessa voidaan vain tehdä uusi rajapinta, jolloin ei tarvitse muuttaa vanhoja ohjelmia. WCF-palveluita on myös helppo hyödyntää eri alustoilla, koska Microsoft tukee ohjelmiensa yhteensopivuutta. Esimerkiksi Windows Phone -ympäristössä voidaan käyttää WCF-palvelua tiedonsiirtoon. (MS 2012b.)

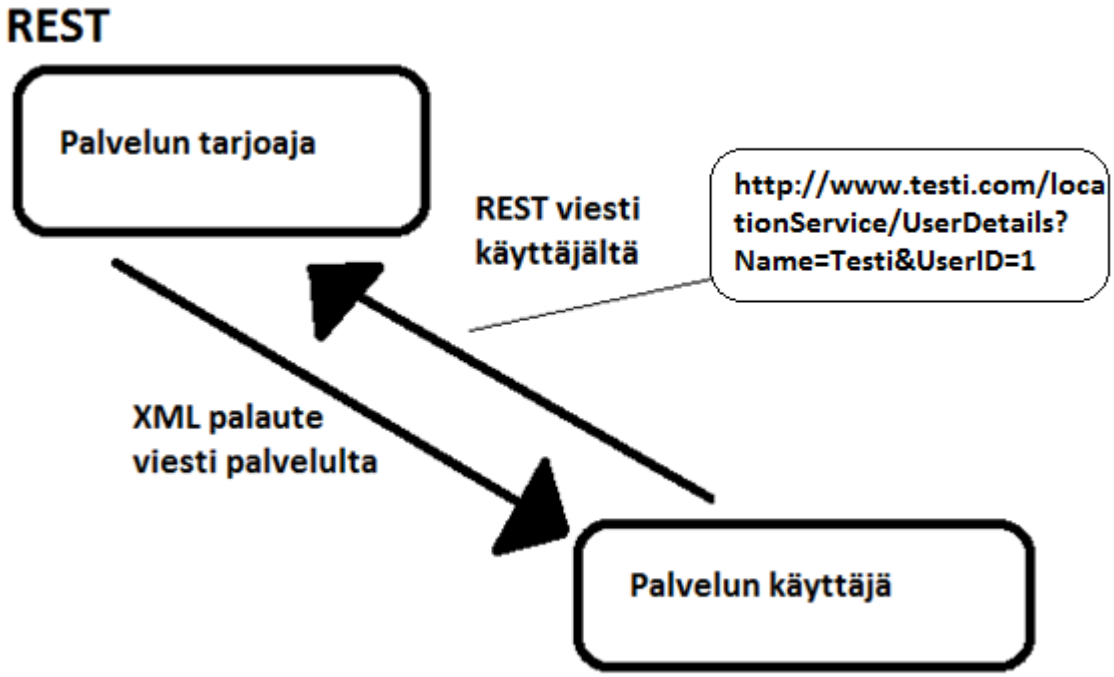
SOAP eli Simple Object Access Protocol on XML-pohjainen protokolla, joka sisältää kolme osaa: Envelope, jossa on valinnainen Header ja pakollinen Body-osa. SOAP on helposti toteutettava protokolla, koska se hyväksyy erilaisia tiedonsiirtomenetelmiä kuten HTTP, SMTP, TCP ja niin edelleen. SOAPin huonona ominaisuutena on kuitenkin sen paketin suuruus. Suuret paketit voivat hidastaa tiedonkulkua. Pakettien suuruus tulee kuitenkin ongelmaksi vasta, kun on kyse suurista määristä paketteja. (Douglas 2013b.)



Kuvio 1. SOAP-viestin rakenne
(Douglas 2013a.)

2.3.2 REST

Representational State Transfer on arkkitehtuurityyli, joka on suurimmalta osin käytössä www-palveluissa. RESTissä on vain yksi perusrajapinta jokaiselle palvelulle. Sen sijaan, että jaetaan kaikki eri palvelut kuten SOAPissa, toteutetaan vain neljä eri palvelumuotoa. Tätä kutsutaan nimellä CRUD (Create, Read, Update, Delete). Toisena liitteenä REST tarvitsee palvelujen URLin, näillä voidaan sitten suorittaa kaikki toiminnot RESTissä lähettämällä haluttu pyyntö haluttuun osoitteeseen. (Hatch 2010.)



Kuvio 2. REST-viestin rakenne
(Douglas 2013a.)

REST-tekniikka on kuitenkin suunnattu pääasiallisesti HTTP-lähetykseen, toisin kuin SOAP-tekniikka. Parempana puolena RESTissä verrattuna SOAP-tekniikkaan on kuitenkin se, että toimintojen nimet ovat aina samanlaisia ja valintana toimii vain muuttujan valitseminen. (Hatch 2010.)

Vaikka WCF on suunniteltu käyttämään SOAP-arkkitehtuuria, voidaan RESTiä käyttää SOAPin korvaajana palvelujen toteutukseen. Microsoft on kehittänyt ADO.NET-datapalveluja niin pitkälle, että REST-palvelun lisääminen on melkein kokonaan automaattista. (MS 2008.)

2.3.3 Web API

Web API on runko, jolla voidaan luoda ohjelmia .NET-rungon ylle. Web API käyttää HTTP-protokollaa yhteyksien muodostamiseen. Se tukee JSON-, XML- formaatteja, joten suurin osa ohjelmista on yhteensopivia sen kanssa. Web API käyttää ohjaimia (controller) hallitsemaan HTTP-pyyntöjä. Jokaisen HTTP-viestin kohdalla Web APIn runko päättää, mikä ohjain ottaa viestin vastaan. (MS 2013.)

Web APIn pääsuunnitelmana on luoda sovellusympäristö, joka toimii millä tahansa alustalla. Se helpottaisi sovellusten toimivuutta huomattavasti, koska ohjelmaa ei ole luotu tietylle käyttöympäristölle. Web API on kuitenkin vielä kehitysvaiheessa ja monia ominaisuuksia puuttuu. Vuoden 2013 alkupuolella on ollut esimerkiksi standardoinnin korjauksen ja ylläpidon ongelmia. (Mozilla 2013).

3 PALVELIMEN JA KÄYTTÖLIITTYMIEN OHJELMOINTI

Työssä oli tarkoituksena saada toimiva yhteys palvelimen ja älypuhelimien välillä, mahdollisesti reaaliaikaisesti. Tietoja lähetetään palvelimelta puhelimille niitä pyydettäessä.

3.1 Eri tekniikoiden vertailu

Vaihtoehtoina palvelintoteutuspohjaksi olivat: WCF SOAP tai REST ja uusimpana Web API. WCF voi kuitenkin olla suhteellisen raskas älypuhelimelle, koska sen tiedon lähetystapa sisältää paljon turhia tietueita, joita olisi voinut leikata pois. REST-tekniikassa tietojen lähetystapaa on yksinkertaistettu mikä mahdollistaa kevyemmän tiedon kulun. REST-tekniikka toimii vain HTTP-lähetystavalla, toisin kuin SOAP-pohjalla toimiva palvelu. (Douglas 2013c).

Työssä valittiin kuitenkin WCF rungoksi palvelimelle, koska siitä oli paljon ohjeita ja esimerkkejä saatavilla. Windows Phone 7 ja WCF ovat molemmat Microsoftin tuotteita, joten yhteensopivuusongelmiakaan ei pitäisi ilmentyä. Web Apista ei ollut vielä paljon tietoa työn aloitushetkellä, joten sitä ei harkittu. REST-pohjaa olisi voitu käyttää, mutta pelkän HTTP:n tukeminen on ehkä liian suppeaa, jos palvelua halutaan laajentaa myöhemmin.

3.2 Vaadittujen järjestelmien asentaminen

WCF on valmiiksi asennettuna Windows Vistassa ja sitä myöhemmissä käyttöjärjestelmissä. WCF vaatii kuitenkin kolme eri resurssia toimiakseen oikein. Ne ovat MSDTC (Microsoft Distributed Transaction Coordinator), IIS (Internet Information Services) ja WAS (Windows Process Activation Service). (MS. 2012c.)

Ohjelmointityökaluna palvelinta ja Windows Phone 7:ää varten käytettiin Microsoft Visual Studio 2010 -ohjelmaa. Visual Studioon asennettiin lisäksi Windows Phone SDK (Source Development Kit), jossa on kaikki tarvittavat resurssit Windows Phonen ohjelmointia varten. (MS 2011.)

Android-sovellusta varten käytettiin vapaan lähdekoodin Eclipse-ohjelmaa. Ohjelman lisäksi otettiin käyttöön kSOAP2-kirjasto, joka tukee SOAP-toimintoja.

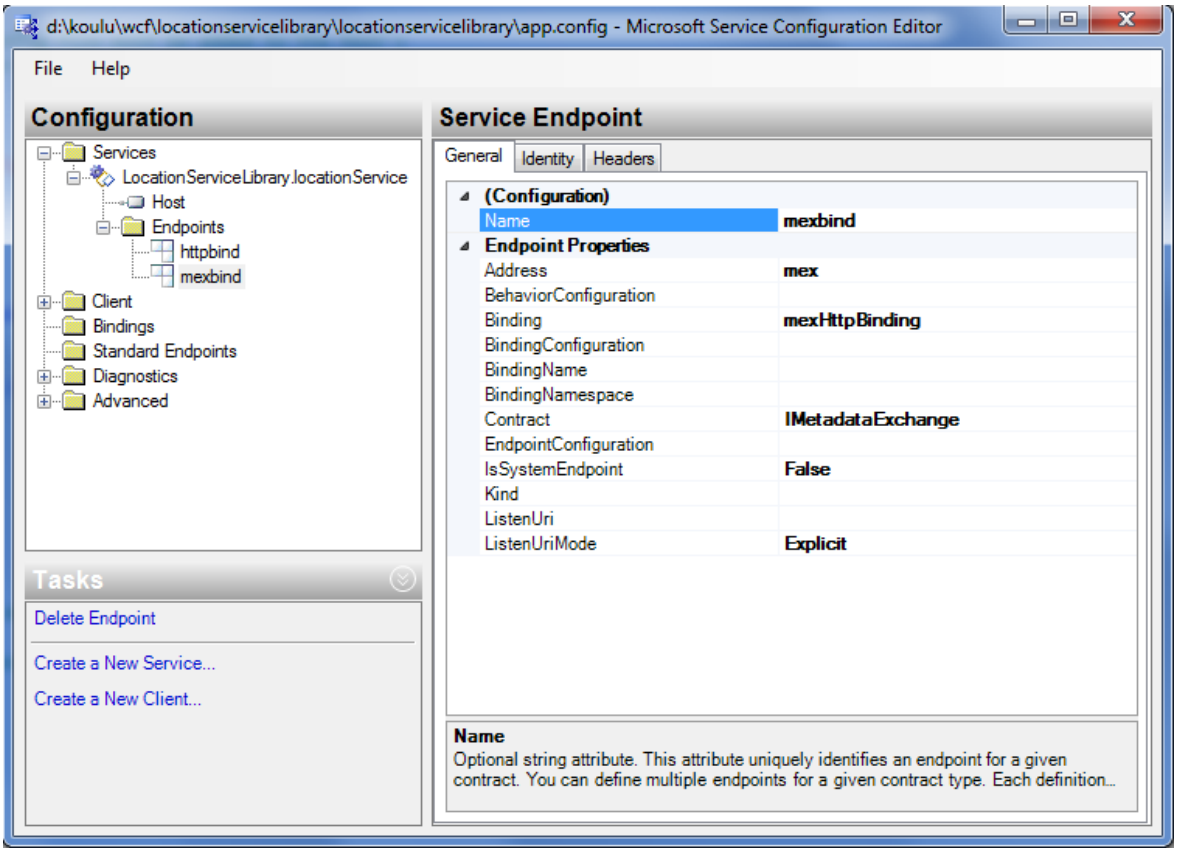
3.3 Ohjelmointi ja suunnittelu

Looginen järjestys toteutukselle oli ensin palvelimen ohjelmoiminen ja testaus, jonka jälkeen ohjelmoitiin Windows Phone 7 -sovellus. Kun kommunikointi toimi moitteettomasti näiden kahden välillä, voitiin ottaa Android-järjestelmä rinnalle kehitykseen.

3.3.1 Palvelimen ohjelmointi

WCF-palvelin toimii pääasiassa lähettämällä funktiokutsuja asiakkaalta palvelimelle SOAP-muodossa. WCF-palvelin pystyy myös lähettämään pyyntöjä asiakkaalle Callback-toiminolla, joten toimintaa pystyy hallitsemaan molempiin suuntiin. Pääsääntöisesti palvelin kuitenkin jakaa palveluita päätepisteistä, joita sille on määritelty. Visual Studio 2010:een on valmiiksi toteutettu Service Configuration Editor, jolla pystyy määrittelemään palveluita ilman että ohjelmakoodiin tarvitsee koskea.

Työssä käytettiin kahta erilaista sidontaa, HTTP- ja MEX-bindingejä. Mex binding on tarkoitettu testausta varten, kun koneelta testataan virtuaalisesti palvelinta. Se täytyy poistaa myöhemmin, kun palvelin otetaan käyttöön. HTTP binding on perussidonta, joka toteuttaa HTTP-kutsuja. (MS 2012d.)



Kuvio 3. Service Configuration Editor
(Microsoft Visual Studio 2010).

Jos ohjelmakoodia haluaa itse muuttaa, niin se onnistuu App.config-tiedostoa muokkaamalla. Helpommin muokkaaminen kuitenkin onnistuu Service Configuration Editorilla. Sopimusten teon jälkeen palvelut vaativat itselleen Contractin eli sopimuksen, millä tavalla palvelu palvelee asiakkaitaan.

Sopimus on yleensä määritelty Interfaceksi eli rajapinnaksi. Rajapinnassa määritellään yksinkertaisesti millaisia funktioita palvelu pitää sisällään. Itse ohjelmakoodille ei kirjoiteta siis rajapintaan. Jos esimerkiksi halutaan palauttaa erilaisia tekstirivejä riippuen siitä missä tilassa palvelin on, niin sille vain luodaan yksinkertainen malli:

```
[ServiceContract]
public interface ILocationService
{
        [OperationContract]
        string checkStatus();
}
```

Rajapinnan yläpuolelle on merkitty ServiceContract-attribuutti, joka kertoo että kyseinen rajapinta on suunnattu palvelun sopimukseksi. Funktioiden yläpuolelle merkitään OperationContract-attribuutti kertomaan, mitä funktioita halutaan näkyvän palvelussa asiakkaille.

Seuraavaksi rajapinta täytyy toteuttaa, joten sille tehdään oma luokka, johon itse koodi ohjelmoidaan. Luokka voidaan yksinkertaisimmillaan nimetä rajapinnan mukaan eli LocationService. Luokkaan täytyy muistaa lisätä rajapinnan toteutus, jotta luokka osaa käyttää rajapinnan funktioita. Tämä onnistuu lisäämällä ”: RajapinnanNimi” luokan nimen loppuun:

```
[ServiceBehavior]
public class LocationService : ILocationService
{
public string checkStatus(){
            return "active";
             }

}
```

Luokan toteutuksen yläpuolelle on lisätty ServiceBehavior-attribuutti merkitsemään, että tämän luokan toiminnot ovat käytettävissä. Itse koodi palauttaa tällä hetkellä aina ”active”-tekstirivin, mutta siihen voidaan myöhemmin lisätä muitakin tiloja.

Luokkia voidaan lisätä myös toisella tavalla palveluihin. Jos esimerkiksi halutaan, että luokka toimii tietuesarjana asiakkaasta, niin luokkaan täytyy lisätä DataContract- ja DataMember-attribuutit merkitsemään, mitkä ovat palvelun julkisia muuttujia.

```
[DataContract]
public class UserData
{
    [DataMember]
    public int userId { get; set; }
    [DataMember]
    public string userName { get; set; }
    [DataMember]
    public double latitude { get; set; }
    [DataMember]
    public double longitude { get; set; }

}
```

Luokkaa voidaan käyttää palvelussa vapaasti kunhan ne kummatkin vain sisältyvät samaan nimiavaruuteen. Asiakaspuolella asiat voivat monimutkaistua, jos palvelimen luokkia käytetään paljon.

Lisätään rajapintaan ja rajapinnan toteutukselle ohjelmakoodi, jossa käytetään uutta luotua luokkaa. Tietueluokan voi asettaa palautusarvoksi, niin kuin tavallisessakin funktiossa.

```
[ServiceContract]
public interface ILocationService
{
    [OperationContract]
    string checkStatus();
    [OperationContract]
    UserData GetLocation(UserData user);

}
```

Toteutuksen puolella asetetaan käyttäjälle jotkin paikka-arvot, jotta ohjelmaa voitaisiin testata.

```
public UserData GetLocation(UserData user)
{
     user.longitude = 20;
     user.latitude = 40;

     return user;
}
```

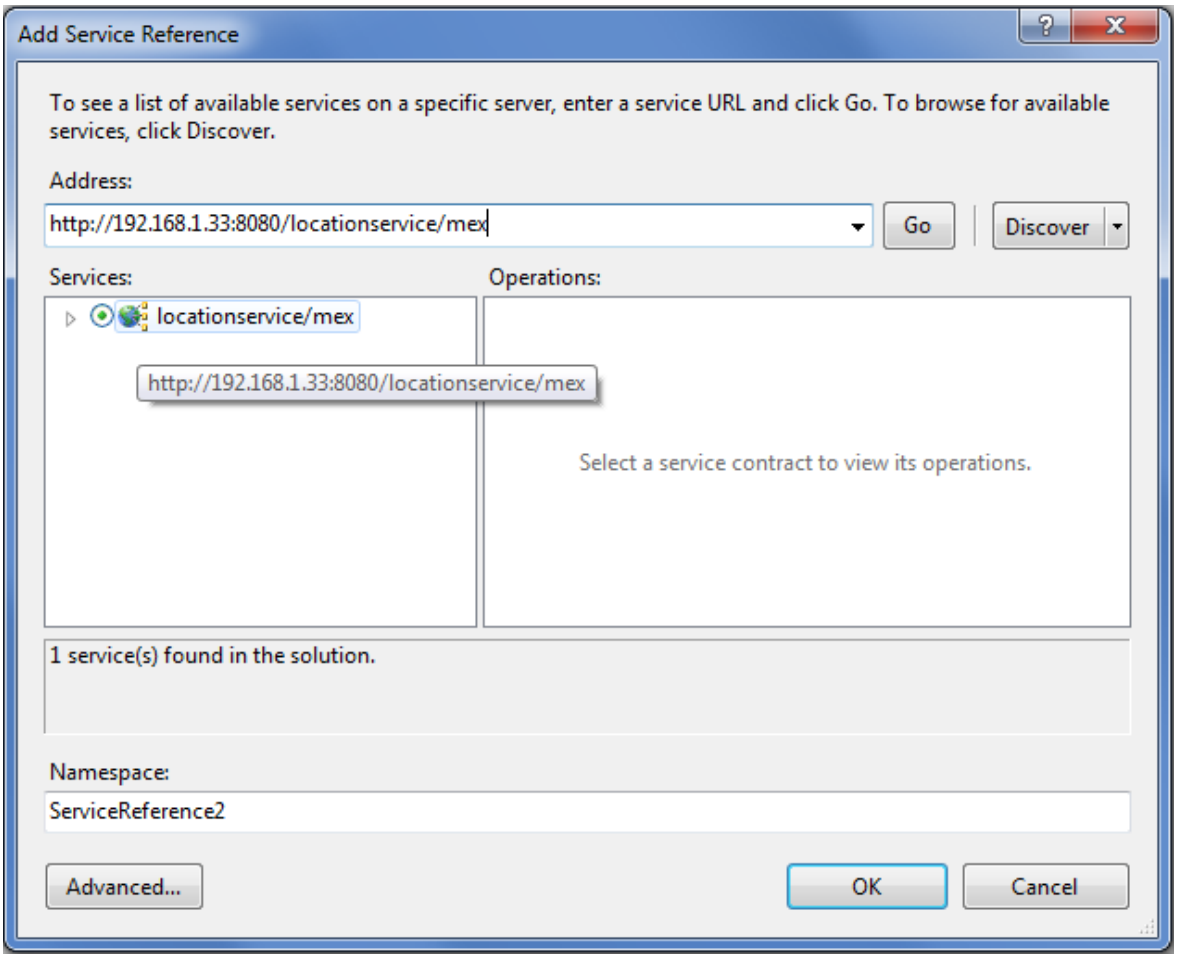
Palvelimen ohjelmointi on nyt valmis testattavaksi. Testauksesta kerrotaan neljännessä luvussa.

3.3.2 Windows Phone 7 -käyttöliittymän ohjelmointi ja suunnittelu

Ennen palvelun lisäämistä suunniteltiin yksinkertainen käyttöliittymä. Käyttöliittymässä on kaksi painiketta ja kolme tekstikenttää. Painikkeita painamalla aktivoidaan pyyntöjä palvelimelta ja tekstikenttiin tulkitaan sitten palvelimelta saadut palautukset.

Kuvio 4. Windows Phone 7 Designer
(Microsoft Visual Studio 2010).

Palvelun liittäminen WP7-ympäristöön on vaivatonta, sillä kummatkin tuotteet ovat Microsoftilta peräisin. WP7-projektiin palvelun lisääminen Service References-kansioon saadaan valitsemalla Add Service Reference -vaihtoehto. Service Reference-ikkunassa on Discover-painike, joka etsii koneella olevia palveluita, jotka voidaan lisätä helposti valitsemalla palvelu ja nimeämällä se. Mex bindingin ansioista päästään käyttämään palvelua virtuaalisena. Kun palvelu on lisätty, sen tulisi näkyä projektin Service References -kansiossa.



Kuvio 5. Service Referencen lisäikkuna
(Microsoft Visual Studio 2010).

Lisätään puhelimen pääsivun koodille ohjelmakoodi, joka käyttää palvelimelle ohjelmoitua palvelua getStatus.

```
public partial class MainPage : PhoneApplicationPage
{

    ServiceReference1.LocationServiceClient locationClient;
    ServiceReference1.UserData currentUser = new ServiceReference1.UserData();

    public MainPage()
    {
        InitializeComponent();

        locationClient = new
        ServiceReference1.LocationServiceClient("BasicHttpBinding_ILocationServ
        ice");

        locationClient.getStatusCompleted += new
        EventHandler<ServiceReference1.getStatusCompletedEventArgs>(locationCli
        ent_getStatusCompleted);
```

```
        }


        void locationClient_getStatusCompleted(object sender,
ServiceReference1.getStatusCompletedEventArgs e)
        {
            if (e.Result != null)
            {
                if (!e.Cancelled)
                    tbTila.Text = e.Result.ToString();
            }
        }


        private void btnButton1_Click(object sender, RoutedEventArgs e)
        {

            locationClient.getStatusAsync();

        }

    }
```

Aluksi luodaan muuttujat palvelulle ja palvelun käyttäjäluokalle. Tämän jälkeen alustetaan palvelu käyttövalmiiksi ja valitaan, mitä toimintoja halutaan käyttää palvelusta, tässä tapauksessa siis getStatus-funktiota. Samalla kun alustus palvelun funktioon lisätään, luo Visual Studio automaattisesti tapahtumahallintafunktion käyttäjälle. Tätä funktiota käytetään silloin, kun palvelun toiminto on suoritettu loppuun. Puhelin ohjelmoitiin siis tulkitsemaan, mitä palvelu lähetti takaisin. Lopuksi painiketta painamalla kutsutaan palvelun getStatus-funktiota, jonka pitäisi palauttaa ”active”-tekstirivi puhelimelle. Tekstirivi tulostetaan tekstikentän sisälle, jolla voidaan emulaattorissa todeta toimivuus.

Seuraavaksi lisätään toinen palvelufunktio, joka ohjelmoitiin palvelimelle. Toinen palvelufunktio käyttää palvelimella olevaa UserData-luokkaa, joka jo aiemmin alustettiin puhelimen ohjelmakoodiin.

```
    public partial class MainPage : PhoneApplicationPage
    {

        ServiceReference1.LocationServiceClient locationClient;
        ServiceReference1.UserData currentUser = new ServiceReference1.UserData();

        public MainPage()
        {
            InitializeComponent();

             locationClient = new
             ServiceReference1.LocationServiceClient("BasicHttpBinding_ILocationServ
             ice");
```

```
            locationClient.getStatusCompleted += new
            EventHandler<ServiceReference1.getStatusCompletedEventArgs>(locationCli
            ent_getStatusCompleted);

            locationClient.getLocationCompleted += new
            EventHandler<ServiceReference1.getLocationCompletedEventArgs>(locationC
            lient_getLocationCompleted);


        }



        void locationClient_getStatusCompleted(object sender,
ServiceReference1.getStatusCompletedEventArgs e)
        {
            if (e.Result != null)
            {
                if (!e.Cancelled)
                    tbTila.Text = e.Result.ToString();
            }
        }


        private void btnButton1_Click(object sender, RoutedEventArgs e)
        {
            locationClient.getStatusAsync();
        }


        void locationClient_getLocationCompleted(object sender,
ServiceReference1.getStatusCompletedEventArgs e)
        {
            if (e.Result != null)
            {
                if (!e.Cancelled)
                {
                    tbLat.Text = e.Result.latitude.ToString();
                    tbLon.Text = e.Result.longitude.ToString();
                }
            }
        }

        private void btnButton2_Click(object sender, RoutedEventArgs e)
        {
            currentUser.userName = tbName.Text.ToString();
            currentUser.userId = Convert.ToInt32
            (tbID.Text);

            locationClient.getLocationAsync(currentUser);
        }

    }
```

Puhelimen ohjelmakoodi hyödyntää nyt kaikkia palveluita, jotka ovat palvelimella tarjolla ja on valmis testattavaksi. Testauksesta kerrotaan neljännessä luvussa.

3.3.3 Android-käyttöliittymän ohjelmointi ja suunnittelu

Koska Android ei suoraa tue SOAP-yhteyttä, käytetään ksoap2-kirjastoa. Kirjastossa on valmiina ohjelmakoodeja, jotka helpottavat WCF-palvelimen käyttöä. Itse kirjaston lisäämisen lisäksi tarvitsee vain tietää, missä osoitteessa palvelu sijaitsee ja palveluiden nimet. Ohjelmakoodin alkuun siis lisätään nämä osoitteet.

```
private static final String SOAP_ACTION =
"http://tempuri.org/ILocationService/checkStatus";
private static final String METHOD_NAME = "checkStatus";

private static final String SOAP_ACTION2 =
"http://tempuri.org/ILocationService/GetLocation";
private static final String METHOD_NAME2 = "GetLocation";

private static final String NAMESPACE = "http://tempuri.org/";
private static final String URL =
"http://192.168.1.33:8080//locationservice//basic";
```

http://tempuri.org on Microsoftin oma määritelmä palvelimelle, jota käytetään suoraan samalta koneelta testausvaiheessa. Myöhemmin NAMESPACE ja URL täytyy siis vaihtaa oikean palvelimen sijaintiin. SOAP_ACTION ja METHOD_NAME täytetään palvelun funktion nimellä ja rajapinnan nimellä.

Yksinkertainen tiedonsiirto on helppo suorittaa Androidin ja WCF välillä. checkStatus-palvelu vaatii seuraavanlaisen ohjelmakoodin.

```
SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.setOutputSoapObject(request);

HttpTransportSE ht = new HttpTransportSE(URL);
ht.call(SOAP_ACTION, envelope);
final  SoapPrimitive response = (SoapPrimitive)envelope.getResponse();
final String str = response.toString();
```

Ensiksi määritellään SOAP-objekti asettamalla sivun nimiavaruus ja funktion nimi. Tämän jälkeen luodaan kirje, johon asetetaan SOAP-objektin tiedot. Seuraavaksi luodaan HTTP-tiedon siirto, johon asetetaan palvelun osoite. Lopuksi voidaan kutsua palvelua palvelimelta ja palvelun palaute tulkitaan tallentamalla se omaan merkkijonoon.

Kun käytetään tietueluokkia tiedon siirtoon, niin ohjelmakoodia vaaditaan enemmän. Ksoap2-kirjasto ei tue kaikkia muuttujia, joten niille pitää itse ohjelmoida Marshal-toiminto. Esimerkiksi double-muuttujan marshallointi tapahtuu seuraavasti.

```
public class MarshalDouble implements Marshal
{
    public Object readInstance(XmlPullParser parser, String namespace, String
name, PropertyInfo expected) throws IOException, XmlPullParserException {

        return Double.parseDouble(parser.nextText());
    }

    public void register(SoapSerializationEnvelope cm) {
         cm.addMapping(cm.xsd, "double", Double.class, this);
    }

    public void writeInstance(XmlSerializer writer, Object obj) throws
IOException {
           writer.text(obj.toString());
        }
}
```

Ohjelmakoodilla saadaan luettua, kirjoitettua ja rekisteröityä double-muuttujia. Marshalling täytyy toteuttaa ennen SOAP-kutsua, jotta Android ymmärtää miten double-muuttujien kanssa tulee toimia.

Seuraavaksi luodaan vastine palvelimella olevalle UserData-luokalle. Luokka voidaan toteuttaa esimerkiksi seuraavalla tavalla.

```
public class UserData implements KvmSerializable {

    public int userId;
    public String userName;
    public double latitude;
    public double longitude;

    public UserData(){}

    public Object getProperty(int arg0) {

        switch(arg0)
        {
        case 0:
            return userId;
        case 1:
            return userName;
        case 2:
            return latitude;
        case 3:
            return longitude;
        }
```

```
        return null;
    }

public int getPropertyCount() {
        return 4;
    }

public void getPropertyInfo(int index, Hashtable arg1, PropertyInfo info) {
        switch(index)
        {
        case 0:
            info.type = PropertyInfo.INTEGER_CLASS;
            info.name = "userId";
            break;
        case 1:
            info.type = PropertyInfo.STRING_CLASS;
            info.name = "userName";
            break;
        case 3:
            info.type = Double.class;
            info.name = "latitude";
            break;
        case 4:
            info.type = Double.class;
            info.name = "longitude";
            break;
        default:break;
        }
    }

public void setProperty(int index, Object value) {
        switch(index)
        {
        case 0:
            userId = Integer.parseInt(value.toString());
            break;
        case 2:
            userName = value.toString();
            break;
        case 3:
            latitude = Double.parseDouble(value.toString());
            break;
        case 4:
            longitude = Double.parseDouble(value.toString());
            break;
        default:
            break;
        }
    }
}
```

Luokan täytyy toteuttaa KvmSerializable-luokkaa, jotta se osaa vastaanottaa yhteyksistä saatua tietoa. Erikoisin funktio on getPropertyInfo, sillä jokaiseen muuttujaan täytyy täyttää muuttujan tyyppi ja nimitiedot. Perusmuuttujat voidaan lisätä PropertyInfo.MUUTTUJAN_NIMI tavalla, mutta erikoisemmissa täytyy käyttää

omaa marshallointia. Tässä tapauksessa käytetään Double.class merkitsemään, että tietue on double-muotoa.

Pääohjelman puolella täytyy myös alustaa luokka ja asettaa marshallingit erikoisille muuttujille. Koska getLocation-palvelu palauttaa tietueluokan, täytyy sekin huomioida ohjelmakoodissa.

```
SoapObject request2 = new SoapObject(NAMESPACE, METHOD_NAME2);

UserData U = new UserData();

U.userId = 1;
U.userName = "Testi";
U.latitude = 0;
U.longitude = 0;

PropertyInfo pi = new PropertyInfo();
pi.setName("UserData");
pi.setValue(U);
pi.setType(U.getClass());
request2.addProperty(pi);


SoapSerializationEnvelope envelope2 = new
SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope2.dotNet = true;
envelope2.setOutputSoapObject(request2);


envelope2.addMapping(NAMESPACE, "UserData",new UserData().getClass());
```

Tietueluokan lisääminen tapahtuu lisäämällä tiedot PropertyInfo-luokkaan. PropertyInfot asetetaan sitten SOAP-objektin tietoihin, josta se osaa tulkita ne oikein. Tämän jälkeen täytyy lisätä marshalling double-muuttujille.

```
MarshalDouble latitude = new MarshalDouble();
latitude.register(envelope2);
MarshalDouble longitude = new MarshalDouble();
longitude.register(envelope2);
```

Jäljelle jää enää palvelun kutsuminen ja tulkitseminen. Lisätään palvelun palaute omaan UserData-luokkaan.

```
HttpTransportSE ht2 = new HttpTransportSE(URL);

ht2.call(SOAP_ACTION2, envelope2);
SoapObject response2 = (SoapObject)envelope2.getResponse();
U.userId = Integer.parseInt(response2.getProperty(0).toString());
U.userName = response2.getProperty(1).toString();
U.latitude = Double.parseDouble(response2.getProperty(2).toString());
U.longitude = Double.parseDouble(response2.getProperty(3).toString());
```

```
final String str = response2.toString();
```

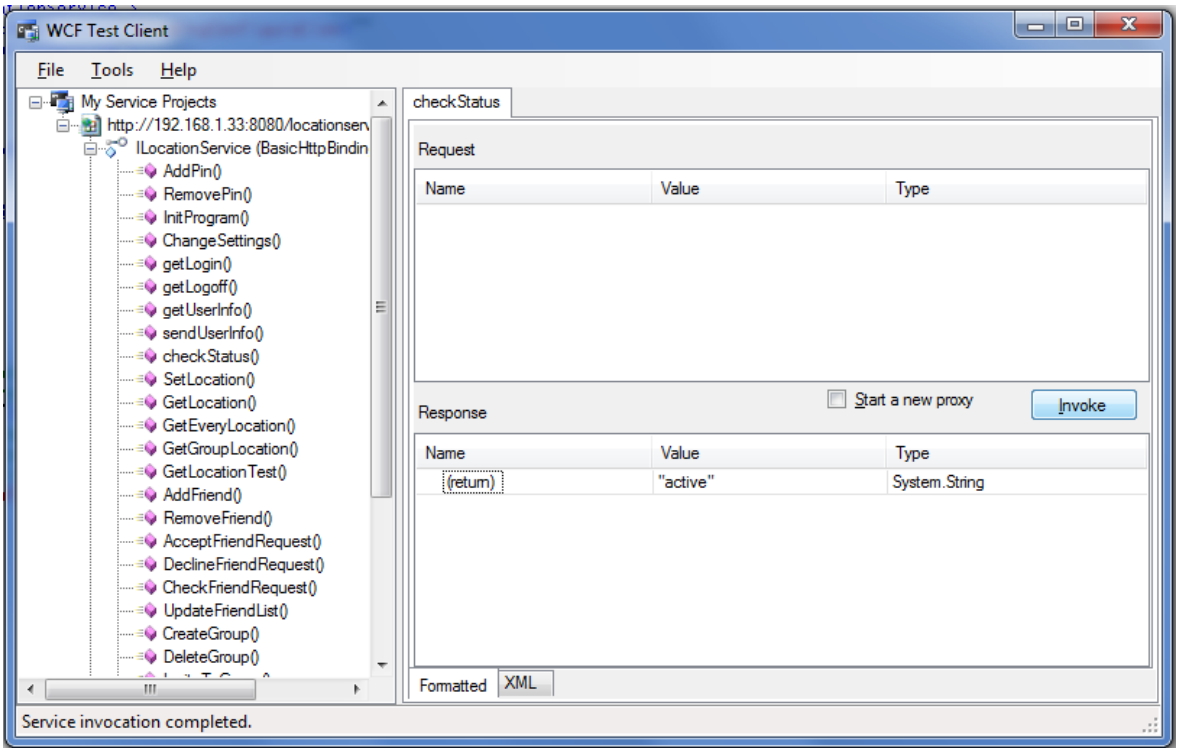
Muuttujat eivät kuitenkaan tule aina palvelimelta samassa järjestyksessä takaisin, kuin mitä itse luokkaan on määritelty. Tästä syystä tulisi aina ennen arvojen asettamista tarkistaa, missä järjestyksessä tiedot saapuvat perille.

4 YHTEYKSIEN TESTAAMINEN

Koska testauksessa ei ollut fyysistä laitetta palvelimen toteuttamista varten, täytyi testaus suorittaa virtuaalisesti. Virtuaalinen palvelimen testaus onnistui helposti Microsoftin omalla ohjelmalla. Windows Phonen testaukseen käytettiin omaa emulaattoria, kuten myös Androidin tapauksessa.

4.1 Palvelimen virtuaalinen testaus

Kun palvelimen ohjelmakoodi kääntyy ilman virheitä, voidaan palvelimen toteutus testata virtuaalisesti Microsoftin WCF Test Client -ohjelmalla. Testiohjelma käynnistyy automaattisesti kun palvelinta debugataan. Ohjelmassa näkyy kaikki tarpeellinen palvelimen funktioiden testaamiseen. Funktioita pystyy testaamaan valitsemalla funktio, syöttämällä siihen tarvittavat muuttujat ja painamalla Invoke-painiketta.



Kuvio 6. WCF Test Client -ikkuna
(Microsoft Visual Studio 2010).

Palvelimen ohjelmakoodiin voi normaalisti lisätä breakpointteja, kun halutaan testata tietyn funktion toimivuutta palvelimessa. Funktion paluuarvot näkyvät Response-ikkunassa. Test Clientillä on aina helpoin todeta, onko palvelimessa jotain ongelmia. Jos ohjelmissa on ongelmia puhelimien puolella, niin ensimmäisenä täytyy testata, toimiiko palvelin vielä oikein.

4.2 Windows Phone 7in virtuaalinen testaus

Windows Phonen testauksessa käytetään hyväksi emulaattoria. Sillä voidaan todeta, miten ohjelma toimii puhelimessa ilman, että käytettäisiin fyysistä laitetta testaukseen. Muutoin testaus on samanlaista kuin normaalisti ja breakpointteja voidaan lisätä ohjelmakoodiin haluttuihin paikkoihin. Puhelimessa tulee testata vain puhelimen ominaisuuksia, sillä jos palvelin ei toimi, niin puhelimeen ohjelmoidut palvelimelta kutsutut funktiot eivät toimi. Emulaattorin testausta haittaa se, että kosketusnäyttöä käytetään joskus useammalla kuin yhdellä sormella (esim. zoomtoimintoa käytettäessä).

Emulaattorissa voidaan kuitenkin todeta, saadaanko yhteys palvelimeen ja toimiiko se halutulla tavalla. Ohjelmakoodin mukaan testataan palvelimen checkStatus- ja getLocation-funktioita painamalla kahta eri painiketta. Jos funktiot toimivat oikein ja puhelimen ohjelmakoodissa ei ole mitään vikaa, niin tekstiriveille pitäisi tulla ”active”-merkkijono ja satunnaisesti asetetut luvut.

Kuvio 7. Windows Phone 7 -emulaattori
(Microsoft Visual Studio 2010).

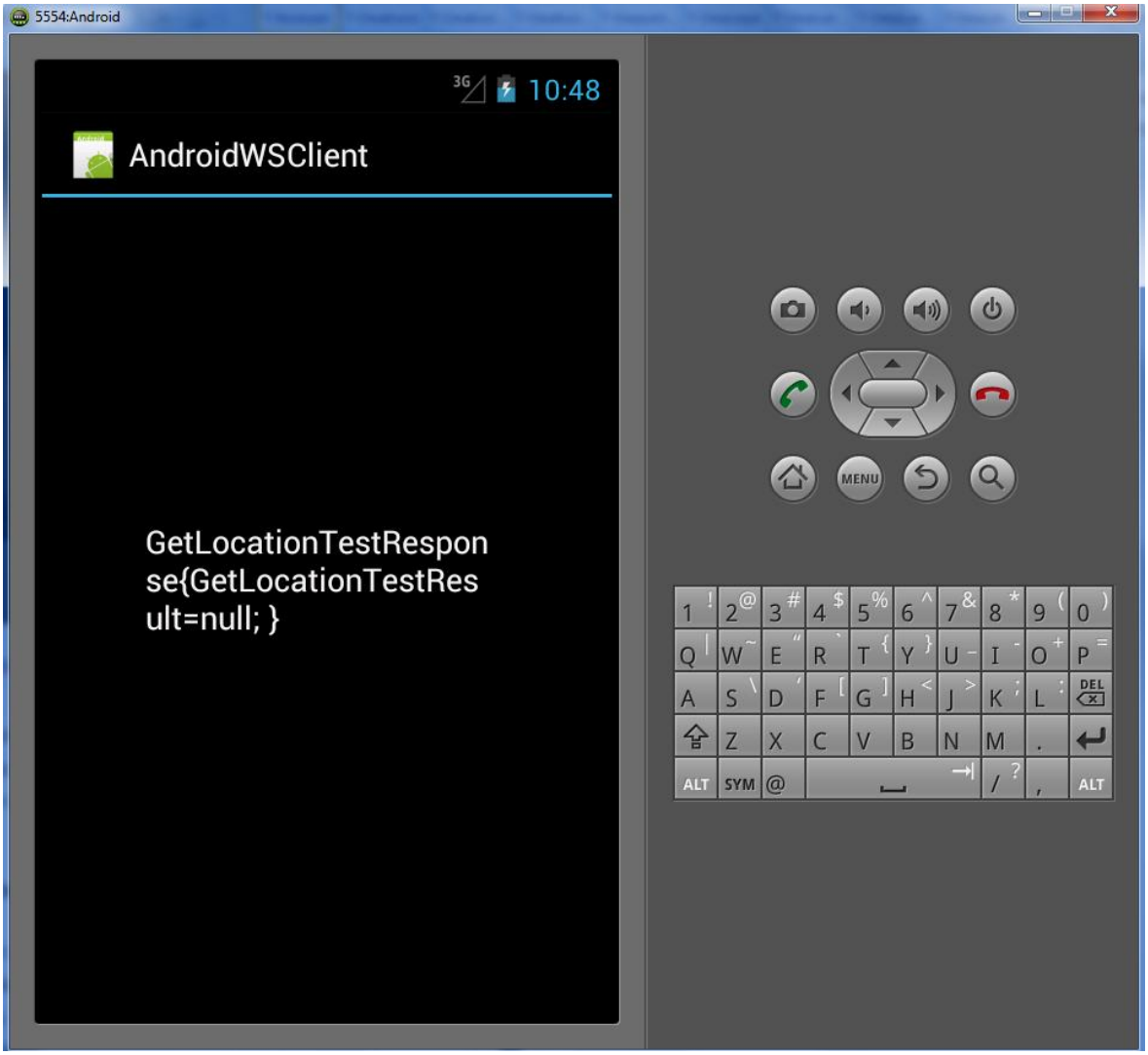
Emulaattorissa saa näppäimistöllä kirjoittamisen käyttöön painamalla PageUp-painiketta kahdesti näppäimistössä. Tämä helpottaa etenkin silloin, kun on ohjelmoitu esimerkiksi käyttäjän kirjautuminen.

4.3 Androidin virtuaalinen testaus

Androidin testauksessa haluttiin vain todeta, saadaanko data puhelimelle asti. Ohjelmakoodin mukaan puhelimelle pitäisi tulla näkyviin kaikki käyttäjän tiedot saman tien käynnistyksen yhteydessä. Näitä tietoja ei ole aseteltu erillisiin tekstikenttiin, vaan raaka data näytetään suoraan.

Jos Android-sovellus ei saa yhteyttä palvelimen palveluihin, palauttaa se soapFault-virheviestin, kun funktiota yritetään kutsua. Virheviesti tulee siis ohjelmakoo-

dissa, kun SoapObjectiin yritetään asettaa palvelun palautusarvo. Eclipse tulkitsee virheen kuitenkin omalla tavallaan kertomalla, että arvo on null. Yleisempinä ongelmina palvelimen yhteyden hakemisessa on vaihtunut IP-osoite tai palomuuri, joka estää yhteyden saannin.



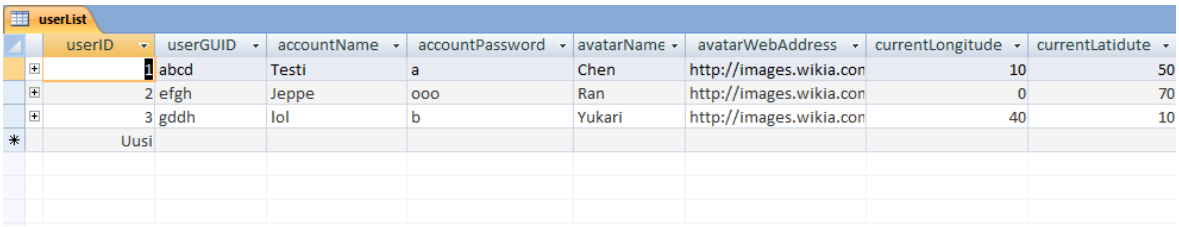
Kuvio 8. Android-emulaattori
(Android 4.1 Eclipse).

5 YHTEYKSIEN KÄYTTÄMINEN

Koska tarjolla oli yksi Lumia 800 -älypuhelin, päätettiin jatkaa ohjelmakoodia sitä varten. Haluttiin, että ohjelmassa näkisi kartalla missä eri henkilöt liikkuvat keskenään. Samalla myös rakennettiin callback-kutsu, jonka tulisi päivittää henkilön sijainnin muille, kun sijaintia vaihdetaan. Tietokantaan tallennetaan henkilön paikkatieto.

5.1 Palvelimen muokkaaminen

Yksi tapa testata tietokantaa on luoda tietokanta tiedosto Microsoftin Access -ohjelmalla. Luodaan käyttäjästä malli tietokantaan ja lisätään muutama käyttäjä tietokantaa, jotta sitä voitaisiin testata.



userList

userID	userGUID	accountName	accountPassword	avatarName	avatarWebAddress	currentLongitude	currentLatidute
1	abcd	Testi	a	Chen	http://images.wikia.con	10	50
2	efgh	Jeppe	ooo	Ran	http://images.wikia.con	0	70
3	gddh	lol	b	Yukari	http://images.wikia.con	40	10
Uusi							

Kuvio 9. Microsoft Access -tietokannan käyttäjä tietue (Microsoft Access 2007).

Ensiksi lisättiin palvelimen app.config tiedoston alkuun tietoja tietokannasta. Tietokannalle täytyy asettaa muutama oleellinen arvo kuten yhteyden muodostus tapa, tietokannan sijainti ja nimi.

```
  <connectionStrings>
    <add name="YhteysString" connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D://userList.mdb" providerName="System.Data.OleDb" />
  </connectionStrings>
```

UserData-luokkaan täytyy myös lisätä muutama muuttuja lisää. currentUri kertoo missä käyttäjä sijaitsee palvelulle ja avatarWebAddress määrittää kuvakkeen käyttäjälle netistä.

```
[DataContract]
    public class UserData
    {
```

```
        [DataMember]
        public int userId { get; set; }
        [DataMember]
        public string userAvatarName { get; set; }
        [DataMember]
        public double latitude { get; set; }
        [DataMember]
        public double longitude { get; set; }
        [DataMember]
        public string avatarWebAddress { get; set; }
        [DataMember]
        public string userPassword { get; set; }
        [DataMember]
        public string CurrentUri { get; set; }
    }
```

Locationservice-luokan alkuun täytyy muistaa alustaa yhteys tietokantaan ennen kuin siihen voidaan ottaa yhteyttä. Tietokannan yhteyden nimi määriteltiin app.config-tiedostossa ja sillä määritellään, mitä yhteyttä halutaan käyttää, jos yhteyksiä on useampia.

```
  public class locationService : ILocationService
    {
        OleDbConnection connectionString =
new
OleDbConnection(ConfigurationManager.ConnectionStrings["YhteysString"].ConnectionStr
ing);
```

Tämän jälkeen muutetaan GetLocation-funktiota siten, että se hakee paikkatiedon tietokannasta. Funktioon syötetään halutun käyttäjän tiedot. Käyttäjän tiedot päivitetään ja palautetaan päivittämisen jälkeen takaisin.

```
public UserData GetLocation(UserData user)
        {
            using (connectionString)
            {
                OleDbCommand cmd = connectionString.CreateCommand();

                cmd.CommandText = "SELECT * FROM userList WHERE userID =" +
                user.userId;

                try
                {
                    OleDbDataReader reader;
                    connectionString.Open();

                    reader = cmd.ExecuteReader();

                    while (reader.Read())
                    {
                       user.longitude = Convert.ToInt32(reader["currentLongitude"]);
                        user.latitude = Convert.ToInt32(reader["currentLatidute"]);
```

```
                }
                connectionString.Close();
            }

            catch (Exception ex)
            {
                connectionString.Close();
            }
        }

        return user;
    }
```

Aluksi alustetaan OleDbCommand-luokka valmiiksi komentoa varten. Tämän jälkeen komennoksi asetetaan tietueen valinta tietokannasta, jossa userID vastaa annettun käyttäjän userIDtä. Sitten avataan yhteys ja yritetään päivittää sijantitiedot käyttäjälle. Jos päivitys ei onnistu, niin tulkitaan virhe catch-haarassa. Virhekoodia ei ole vielä kuitenkaan ohjelmoitu, joten jos virhe sattuu, niin yhteys vain suljetaan. Lopuksi palautetaan käyttäjän tiedot takaisin käyttäjälle.

Ohjelmakoodiin voidaan myös lisätä callback-toiminto, jolloin palvelin lähettää tiedon käyttäjille että kyseisen henkilön paikka on vaihtunut. Callbackin viestimuodon voi itse valita haluamakseen. Yksinkertaisin on kenties raaka (raw)-viesti, jossa on pelkästään tekstiä. Tämän tekstin voi sitten tulkita käyttäjän päässä omalla tavallaan tai käyttämällä esimerkiksi XML-muotoa.

```
int receiverID = 1;

try
            {
            cmd.CommandText = "SELECT userID FROM userList WHERE userID =" + us-
            er.userID;

            OleDbDataReader reader;
            connectionString.Open();

            reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                        try
                        {
                        string subscriptionUri =
                         reader["lastUri"].ToString());

                        HttpWebRequest sendNotificationRequest =
                        (HttpWebRequest)WebRequest.Create(subscriptionUri);

                        sendNotificationRequest.Method = "POST";
```

```
                string rawMessage = "<?xml version=\"1.0\" encoding=\"utf-
                8\"?>" +  "<message type='locationchange'>" +
                "<userID>" + user.userID + "</userID>" +
                "<Longitude>" + user.longitude + "</Longitude>" +
                "<Latitude>" + user.latitude + "</Latitude>" +
                "</message>";

                byte[] notificationMessage =
                 Encoding.Default.GetBytes(rawMessage);

                sendNotificationRequest.ContentLength =
                notificationMessage.Length;
                sendNotificationRequest.ContentType = "text/xml";
                sendNotificationRequest.Headers.Add("X-NotificationClass",
                "3");


                using (Stream requestStream =
                 sendNotificationRequest.GetRequestStream())
                {
                requestStream.Write(notificationMessage, 0,
                notificationMessage.Length);
                }

                HttpWebResponse response =
                (HttpWebResponse)sendNotificationRequest.GetResponse();
                string notificationStatus =
                response.Headers["X-NotificationStatus"];
                string notificationChannelStatus =
                response.Headers["X-SubscriptionStatus"];
                string deviceConnectionStatus =
                response.Headers["X-DeviceConnectionStatus"];

                }
            catch (Exception ex)
            {
            //TextBoxResponse.Text =
             "Exception caught sending update: " + ex.ToString();
            }
```

Ensiksi siis haetaan käyttäjä userId:n perusteella, jolle halutaan lähettää callback-kutsu. Tämän jälkeen avataan yhteys ja luetaan henkilön tiedot tietokannasta. Tietokannasta luetaan viimeksi annettu Uri käyttäjältä ja asetetaan se pyynnön osoitteeksi. Sitten asetetaan viestille teksti, tässä tapauksessa XML-muotoinen viesti ja lasketaan sen pituus tavu muodossa. Pituus asetetaan viestin pituudeksi ja asetetaan viestille sen tyyppi arvot. Viestissä on nyt kaikki valmiina, jotta sen voisi lähettää käyttäjälle. Jos siinä ei onnistuta, niin lähetetään virheviesti palvelimelle.

Seuraavaksi käyttöä helpottamaan ohjelmoidaan kaksi uutta funktiota. SetLocation ja GetEveryLocation. Näillä voidaan asettaa paikkatietoja ja lukea kaikkien käyttäjien tiedot. Näillä komennoilla voidaan puhelimessa toteuttaa käyttäjien paikantaminen.

```
    public bool SetLocation(UserData userLocation)
        {
            bool statusSet = false;

            using (connectionString)
            {
                OleDbCommand cmd = connectionString.CreateCommand();

                cmd.CommandText = "UPDATE userList set currentLongitude=" +
userLocation.longitude + ", currentLatidute=" + userLocation.latitude +
                    " WHERE accountName ='" + userLocation.userAccountName + "' AND
accountPassword ='" + userLocation.userPassword + "'";

                try
                {
                    connectionString.Open();
                    cmd.ExecuteNonQuery();
                    statusSet = true;
                    connectionString.Close();
                }

                catch (Exception ex)
                {
                    //Response.Write("VIRHEVIESTI: " + ex.Message);
                    connectionString.Close();
                }
            }

            return statusSet;

        }
```

SetLocation on suhteellisen yksinkertainen. Se yrittää päivittää tietokantaan sijainnin ja jos onnistutaan, niin true-arvo palautetaan käyttäjälle. True-arvo kertoo käyttäjälle, että päivitys onnistui.

```
public List<UserData> GetEveryLocation()
        {
            List<UserData> userList = new List<UserData>();

            using (connectionString)
            {
                OleDbCommand cmd = connectionString.CreateCommand();

                cmd.CommandText = "SELECT * FROM userList";

                try
                {
                    OleDbDataReader reader;
                    connectionString.Open();

                    reader = cmd.ExecuteReader();

                    while (reader.Read())
                    {
                        if (Convert.ToInt32(reader["userID"]) > 0)
                        {
                            UserData user = new UserData();
```

```
                            user.userAccountName = reader["accountName"].ToString();
                            user.avatarWebAddress = read-
er["avatarWebAddress"].ToString();
                            user.longitude = Con-
vert.ToInt32(reader["currentLongitude"]);
                            user.latitude = Con-
vert.ToInt32(reader["currentLatidute"]);
                            userList.Add(user);
                        }

                    }
                    connectionString.Close();
                }

                catch (Exception ex)
                {
                    //Response.Write("VIRHEVIESTI: " + ex.Message);
                    connectionString.Close();
                }
            }


            return userList;
        }
```

GetEveryLocation hakee kaikki käyttäjät tietokannasta käyttäjälistaan ja lähettää ne lopuksi takaisin niitä pyytävälle käyttäjälle.

5.2 Windows Phone 7 -käyttöliittymän päivitys

Ensimmäisenä muutoksena on service referencen päivittäminen uuteen palvelukoodiin. Yksinkertaisin tapa on poistaa vanha service reference ja lisätä se uudelleen samalla nimellä. Service reference tulisi aina päivittää, jos palvelimen ohjelmakoodiin tehdään muutoksia. GPS-toimintaa varten voidaan käyttää Microsoftin GeoCoordinateWatcher-luokkaa. Pushpin-luokalla voidaan lisätä merkkejä karttaan, joten sillä voidaan merkitä myös ihmisten sijainteja.

```
GeoCoordinateWatcher geoWatcher;
List<Pushpin> userLocationPins = new List<Pushpin>();
Pushpin pin = new Pushpin();
```

Pushpin-listaan lisätään kaikkien käyttäjien merkinnät karttaan. Niitä muunnellaan sitten, kun callbackistä tulee kutsu tai jos käyttäjä itse pyytää päivitystä. Seuraavaksi asetetaan geoWatcherille sopivat asetukset, jotta kartta toimisi halutulla tavalla.

```
if (geoWatcher == null)
{
            geoWatcher = new GeoCoordinateWatcher(GeoPositionAccuracy.High)
            {
            MovementThreshold = 10
            };

            geoWatcher.PositionChanged += new
            EventHandler<GeoPositionChangedEventArgs<GeoCoordinate>>(watcher_Positi
            onChanged);

            geoWatcher.StatusChanged += new
            EventHandler<GeoPositionStatusChangedEventArgs>(watcher_StatusChanged);
            geoWatcher.Start();

            map1.ZoomBarVisibility = System.Windows.Visibility.Visible;
            map1.Mode = new Microsoft.Phone.Controls.Maps.AerialMode();

}
```

geoWatcherille asetetaan tarkkuudeksi korkein mahdollinen ja se päivittää kohdetta 10 metrin välein. Tämän jälkeen asetetaan kaksi tapahtuman käsittelijää, jotka hoitavat paikan vaihdoksen ja tilan päivityksen. Sitten geoWatcherin on valmis käynnistettäväksi. Kartan asetukset voi päivittää myös geoWatcherin käynnistyessä.

Karttapalvelu täytyy myös alustaa sivun .xaml-tiedostoon. Ensiksi se täytyy alustaa ennen itse ohjelmakoodia ja luoda se sivun käyttöliittymään.

```
shell:SystemTray.IsVisible="True" xmlns:my="clr-
namespace:Microsoft.Phone.Controls.Maps;assembly=Microsoft.Phone.Controls.Maps"
Loaded="PhoneApplicationPage_Loaded" d:DesignHeight="696" d:DesignWidth="480">

<!--Pivot item one-->
<controls:PivotItem Header="Map">
            <Grid>
            <my:Map Name="map1" />
            </Grid>
</controls:PivotItem>
```

Asetetaan toiselle pivot sivulle kaksi tekstikenttää ja kaksi painiketta joilla voidaan manuaalisesti vaihtaa käyttäjän paikkaa ja päivittää käyttäjien paikat.

```
<!--Pivot item two-->
<controls:PivotItem Header="Set stuff">
            <Grid>
            <StackPanel>
            <TextBox Name="tbLon" Width="200" Text="0" TextAlignment="Right" />
            <TextBox Name="tbLat" Width="200" Text="0" TextAlignment="Right" />
            <Button Name="btnSetLocation" Content="OK" Click="btnSetLocation_Click"
/>
```

```
            <TextBlock Height="30" Name="tbStatus" Text="TextBlock" />
            <Button Name="btnrefresh" Content="Refresh" Click="btnrefresh_Click" />
            </StackPanel>
            </Grid>
</controls:PivotItem>
```

Sivun alustuksen lopuksi lisätään HttpNotificationChannel, joka hallitsee callback-kutsuja. Kanavalle voidaan antaa oma nimi, jolla käyttäjä saa yhteyden oikeaan kanavaan.

```
string channelName = "RawSampleChannel";

pushChannel = HttpNotificationChannel.Find(channelName);

if (pushChannel == null)
{
            pushChannel = new HttpNotificationChannel(channelName);

            pushChannel.ChannelUriUpdated += new
EventHandler<NotificationChannelUriEventArgs>(pushChannel_ChannelUriUpdated);
            pushChannel.ErrorOccurred += new
EventHandler<NotificationChannelErrorEventArgs>(pushChannel_ErrorOccurred);
            pushChannel.HttpNotificationReceived += new
EventHandler<HttpNotificationEventArgs>(pushChannel_HttpNotificationReceived);

            pushChannel.Open();

}
else
{
            pushChannel.ChannelUriUpdated += new
EventHandler<NotificationChannelUriEventArgs>(pushChannel_ChannelUriUpdated);
            pushChannel.ErrorOccurred += new
EventHandler<NotificationChannelErrorEventArgs>(pushChannel_ErrorOccurred);
            pushChannel.HttpNotificationReceived += new
EventHandler<HttpNotificationEventArgs>(pushChannel_HttpNotificationReceived);

            System.Diagnostics.Debug.WriteLine(pushChannel.ChannelUri.ToString());
            MessageBox.Show(String.Format("Channel Uri is {0}",
            pushChannel.ChannelUri.ToString()));

}
```

Kanavalle täytyy ohjelmoida kaksi eri toimintoa, koska jos kanava on jo luotu niin käyttäjä saattaa jumittua. Käyttäjän kanavan URI (Uniform Resource Identifier) tulostetaan käyttäjälle testausta varten. Se täytyy poistaa myöhemmin kun ohjelmaa aletaan käyttää. Kummassakin tapauksessa alustetaan kanavalle kolme tapahtuman käsittelijää.

Seuraavaksi luodaan tapahtuman käsittelijöille ohjelmakoodit, jotta ne tietävät mitä tehdä, kun viesti saapuu palvelimelta tai kun kanava vaihtuu ja niin edelleen.

```
void pushChannel_ChannelUriUpdated(object sender, NotificationChannelUriEventArgs e)
{

            Dispatcher.BeginInvoke(() =>
            {
            System.Diagnostics.Debug.WriteLine(e.ChannelUri.ToString());
            MessageBox.Show(String.Format("Channel Uri is {0}",
            e.ChannelUri.ToString()));

            });
}
```

Kun Uri päivitetään, niin se vain tulostetaan uudestaan käyttäjälle. Myöhemmin tähän voidaan lisätä käyttäjän laitteen vaihto.

```
void pushChannel_ErrorOccurred(object sender, NotificationChannelErrorEventArgs e)
{
            Dispatcher.BeginInvoke(() =>
            MessageBox.Show(String.Format("A push notification {0} error occurred.
{1} ({2}) {3}",
            e.ErrorType, e.Message, e.ErrorCode, e.ErrorAdditionalData))
            );
}
```

Samoin kuin Urin päivittämisen tapahtuma hallinnassa, virheen hallintakin vain tulostaa käyttäjälle kaikki virheviestit.

```
void pushChannel_HttpNotificationReceived(object sender, HttpNotificationEventArgs
e)
{
string message;

            using (System.IO.StreamReader reader =
            new System.IO.StreamReader(e.Notification.Body))
            {
            message = reader.ReadToEnd();
            }

            using (XmlReader reader = XmlReader.Create(new StringReader(message)))
            {
                        reader.ReadToFollowing("message");
                        reader.MoveToFirstAttribute();
                        string type = reader.Value;

                        if (type == "locationchange")
                        {
                        reader.ReadToFollowing("UserID");
                        string value1 = reader.ReadElementContentAsString();

                        reader.ReadToFollowing("Longitude");
                        double value2 = reader.ReadElementContentAsDouble();

                        reader.ReadToFollowing("Latitude");
                        double value3 = reader.ReadElementContentAsDouble();
```

```
                        Dispatcher.BeginInvoke(() =>
                        MessageBox.Show(String.Format("Someone has changed their
                        location : {0} , {1} , {2}", value1, value2, value3))
                        );

                        }

            }
}
```

Kun palvelin lähettää callback-kutsun käyttäjälle, yritetään sitä tulkita XML-viestinä. Ensiksi katsotaan millainen viestityyppi XML-viestillä on. Jos se on locationchange-tyyppiä, niin luetaan lähetetyt arvot. Lopuksi ne tulostetaan käyttäjälle, myöhemmin kuitenkin tähän lisätään ohjelmakoodi, joka päivittää Pushpin-merkinnän kartalla kyseisellä käyttäjällä.

Asetetaan geoWatcherin tapahtumahallitsijoille seuraavat ohjelmakoodit:

```
void watcher_StatusChanged(object sender, GeoPositionStatusChangedEventArgs e)
{
            if (e.Status == GeoPositionStatus.Initializing)
            {
                Debug.WriteLine("Initializing");
            }
            else if (e.Status == GeoPositionStatus.Ready)
            {
                Debug.WriteLine("Ready");
            }
}
```

Kun geoWatcherin tila vaihtuu käynnistyksestä valmiustilaan, niin siitä kerrotaan vain ohjelman testaajalle. Myöhemmin tähänkin voidaan lisätä käyttäjälle viesti siitä, että koska puhelin on valmis vastaanottamaan tietoa.

```
void watcher_PositionChanged(object sender,
GeoPositionChangedEventArgs<GeoCoordinate> e)
{
            Debug.WriteLine("({0},{1})", e.Position.Location.Latitude,
            e.Position.Location.Longitude);

            map1.Center = new GeoCoordinate(e.Position.Location.Latitude,
            e.Position.Location.Longitude);

}
```

Kun käyttäjä vaihtaa paikkaa, niin keskitetään kartta siihen pisteeseen, missä hän sillä hetkellä on. Tämän ohjelmakoodin voisi tosin asettaa vaihtoehtoiseksi, jos käyttäjä ei halua seurata häntä lähellä tapahtuvaa toimintaa.

Ennen kuin painikkeiden toiminnot ohjelmoidaan, täytyy lisätä SetLocation ja GetEveryLocationin käyttö ohjelmakoodiin. Alkuun täytyy ensin alustaa kummatkin komennot samalla lailla, kuin kahdessa aiemmassa palvelinfunktiossa.

```
locationClient.SetLocationCompleted += new
EventHandler<ServiceReference1.SetLocationCompletedEventArgs>(locationClient_SetLoca
tionCompleted);
locationClient.GetEveryLocationCompleted += new
EventHandler<ServiceReference1.GetEveryLocationCompletedEventArgs>(locationClient_Ge
tEveryLocationCompleted);
```

Alustamisen jälkeen ohjelmoidaan niiden toiminnot.

```
void locationClient_SetLocationCompleted(object sender, Sys-
tem.ComponentModel.AsyncCompletedEventArgs e)
        {

            if (e.Result == true)
            {
                tbStatus.Text = "Location sent";
            }
            else
            {
                tbStatus.Text = "unable to reach server";
            }

        }
```

SetLocationin toiminta on yksinkertainen. Jos päivitys onnistui, niin palvelin lähettää käyttäjälle true-arvon. Tällöin vaihdetaan tekstiksi Location sent, joka kuvaa käyttäjälle, että lähetys onnistui. Jos lähetys ei onnistunut, niin kerrotaan käyttäjälle, että palvelimeen ei saatu yhteyttä.

```
void locationClient_GetEveryLocationCompleted(object sender,
ServiceReference1.GetEveryLocationCompletedEventArgs e)
        {
            List<ServiceReference1.UserData> tmpList = new
List<ServiceReference1.UserData>();
            map1.Children.Clear();
            userLocationPins.Clear();
            tmpList = e.Result.ToList();

            foreach (ServiceReference1.UserData User in tmpList)
            {
                Pushpin newPin = new Pushpin();
                newPin.Location = new GeoCoordinate(User.latitude, User.longitude);

                ImageBrush image = new ImageBrush()
                {
                    ImageSource = new System.Windows.Media.Imaging.BitmapImage
```

```
                                        (new Uri(User.avatarWebAddress))
                };
-
                newPin.Content = new Ellipse()
                {
                    Fill = image,
                    StrokeThickness = 10,
                    Height = 64,
                    Width = 64

                };

                map1.Children.Add(newPin);

                userLocationPins.Add(newPin);
            }

        }
```

GetEveryLocationin kohdalla toteutetaan karkea päivitys käyttäjien paikkojen päivitykseen. Koko vanha merkkilista poistetaan ja vaihdetaan uusiin paikkoihin. Tämä toiminto tulisi päivittää järkevämmäksi myöhemmin, mutta testitarkoitukseen se soveltuu.

Viimeiseksi täytyy vielä toisella pivot-sivulla oleville napeille ohjelmoida toiminnot, jotta paikkaa voitaisiin vaihtaa manuaalisesti.

```
private void btnSetLocation_Click(object sender, RoutedEventArgs e)
        {
            currentUser.longitude = Convert.ToInt32(tbLon.Text);
            currentUser.latitude = Convert.ToInt32(tbLat.Text);
            locationClient.SetLocationAsync(currentUser);

        }
```

Paikan vaihtaminen napin painalluksesta on yksiselitteinen. Tekstikentistä päivitetään ensin tieto käyttäjäluokkaan ja käyttäjän tiedot lähetetään sitten palvelimelle.

Päivitysnapin painallukseen tarvitsee vain asettaa GetEveryLocationin kutsu.

```
  private void btnrefresh_Click(object sender, RoutedEventArgs e)
        {
            locationClient.GetEveryLocationAsync();
        }
```

Päivitysnappia painamalla haetaan kaikkien käyttäjien viimeisimmät paikat tietokannasta. Kun kutsut on luotu, niin täytyy vielä päivittää ohjelmakoodit, kun palvelin palauttaa arvoja takaisin käyttäjälle.

```
void locationClient_GetLocationCompleted(object sender,
ServiceReference1.GetLocationCompletedEventArgs e)
{
             Pushpin newPin = new Pushpin();

            if (currentUser.userId == e.Result.userId)
            {
                currentUser.latitude = e.Result.latitude;
                currentUser.longitude = e.Result.longitude;
                pin.Location = new GeoCoordinate(currentUser.latitude,
currentUser.longitude);
                map1.Children.Add(newPin);
            }

            else if( e.Result.userId > 0)
            {
                foreach (UserInfo user in userList)
                {
                    if (user.ID == e.Result.userId)
                    {
                        user.name = e.Result.userAvatarName;
                        newPin.Location = new GeoCoordinate(e.Result.latitude,
e.Result.longitude);
                        user.location = newPin;
                        map1.Children.Add(user.location);

                        ImageBrush image = new ImageBrush()
                        {
                            ImageSource = new Sys-
tem.Windows.Media.Imaging.BitmapImage(new Uri(e.Result.avatarWebAddress))
                        };

                        newPin.Content = new Ellipse()
                        {
                            Fill = image,
                            StrokeThickness = 10,
                            Height = 64,
                            Width = 64

                        };
                    }
                }
            }


        }
```

Paikkatiedon saavuttua palvelimelta tutkitaan ensin onko paikkatieto puhelimen käyttäjän paikkatieto vai jonkin toisen henkilön. Jos paikkatieto on puhelimen käyttäjän oma, niin se voidaan suoraa päivittää karttaan. Ennen kuin toisen henkilön paikkatieto päivitetään, täytyy oikea henkilö löytää käyttäjälistasta.

5.3 Parannetun järjestelmän testaaminen

Ohjelmoinnin jälkeen saatiin ohjelma testausvaiheeseen asti. Vaikka ohjelma olikin karkea, niin se silti toteutti sitä mitä haluttiin. Älypuhelimessa on kuitenkin tärkeää, että ohjelma on mahdollisemman optimoitu, jotta se säästää virtaa ja ei lähetä turhia pyyntöjä palvelimelle.

Kuvio 10. Emulaattorin viimeinen testaus (Microsoft Visual Studio 2010).

6 TULOKSET

Työssä onnistuttiin siinä, mitä haluttiin toteuttaa. Yhteydet puhelimien välillä toimivat ja palvelimen funktioita pystytään kutsumaan. Windows Phone 7 -järjestelmässä saatiin käyttäjien paikantaminen toimivaksi, tosin ohjelmakoodia täytyy hioa paremmaksi, sillä itse koodi on erittäin karkea ja vie turhan paljon resursseja.

Android tuotti hieman ongelmia toteutuksen kannalta, mutta ongelmista selvittiin. Androidin isoimpana ongelmana oli yhteyden saaminen palvelimeen ja sieltä tiedon hakeminen. Androidissa täytyy myös itse hallita kaikki mahdolliset null-arvot, jotka vaikeuttavat yhteyden saantia monesti.

Tietokantaan tallentaminen sujui ongelmitta, tosin joskus se aiheutti timeout-virheitä, mutta vain harvoin. Timeoutit täytyy kuitenkin ottaa huomioon, sillä normaalissa tilanteessa tiedon hakemisessa kestää paljon kauemmin kuin virtuaalisessa ympäristössä.

7 YHTEENVETO

Työssä tutkittiin älypuhelimien kommunikointia. Windows Phone 7:n ja Androidin kommunikointia selvitettiin virtuaalisen palvelimen avulla. Tulokseksi saatiin toimiva kommunikointi älypuhelimien ja palvelimen välillä. Windows Phone 7:n ohjelmaa kehitettiin käyttämään kartta palvelua, jolla paikannettiin käyttäjiä. Palvelimelle lisättiin uusia palveluita, joilla saatiin paikkatietojen luku tietokannasta toteutettua.

LÄHTEET

AD. 2007. URI vs. URL: What's the difference? [www-dokumentti]. Almaer Dion. [Viitattu 22.5.2013]. Saatavissa: http://ajaxian.com/archives/uri-vs-url-whats-the-difference

BE. 2007. Using JSON to Exchange Data. [www-dokumentti]. Better Explained. [Viitattu 22.5.2013]. Saatavissa: http://betterexplained.com/articles/using-json-to-exchange-data/

Dolcourt. 2010. The hardware guts of your Android phone. [www-dokumentti]. Jessica Dolcourt. [Viitattu 16.4.2013]. Saatavissa: http://www.cnet.com/8301-19736_1-20021553-251.html

Douglas. 2013a. Web Services Explained. [www-dokumentti]. Douglas K Barry. [Viitattu 17.4.2013]. Saatavissa: http://www.service-architecture.com/web-services/articles/web_services_explained.html

Douglas. 2013b. SOAP. [www-dokumentti]. Douglas K Barry. [Viitattu 17.5.2013]. Saatavissa: http://www.service-architecture.com/web-services/articles/soap.html

Douglas. 2013c. Representational State Transfer (REST). [www-dokumentti]. Douglas K Barry. [Viitattu 17.5.2013]. Saatavissa: http://www.service-architecture.com/web-services/articles/representational_state_transfer_rest.html

Hatch. 2010. REST Services Explained. [www-blogi]. [Viitattu 18.2.2013]. Saatavissa: http://blogs.geniuscode.net/RyanDHatch/?p=29

HA. 2011. TCP/IP Protocol Fundamentals Explained with a Diagram. [www-dokumentti]. Himanshu Arora. [Viitattu 22.5.2013]. Saatavissa: http://www.thegeekstuff.com/2011/11/tcp-ip-fundamentals/

HS. 2011. Nokia ja Microsoft laajaan yhteistyöhön. [www-dokumentti]. Helsinging Sanomat. [Viitattu 17.5.2013]. Saatavissa: http://www.hs.fi/talous/artikkeli/Nokia+ja+Microsoft+laajaan+yhteistyöhön/1135263725824

Mozilla. 2013. WebAPI/PlannedWork. [www-dokumentti]. Mozilla. [Viitattu 3.5.2013]. Saatavissa: https://wiki.mozilla.org/WebAPI/PlannedWork

MS. 2008. A Guide to Designing and Building RESTful Web Services with WCF 3.5. [www-dokumentti]. Microsoft. [Viitattu 14.4.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/dd203052.aspx

MS. 2011. Windows Phone SDK 7.1. [www-dokumentti]. Microsoft. [Viitattu 19.2.2013]. Saatavissa: http://www.microsoft.com/en-us/download/details.aspx?id=27570

MS. 2012a. Hardware Specifications for Windows Phone. [www-dokumentti]. Microsoft. [Viitattu 12.2.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/ff637514%28v=vs.92%29.aspx

MS. 2012b. What Is Windows Communication Foundation. [www-dokumentti]. Microsoft. [Viitattu 12.2.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/ms731082.aspx

MS. 2012c. Operating System Recources Required by WCF. [www-dokumentti]. Microsoft. [Viitattu 19.2.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/aa702743.aspx

MS. 2012d. How to: Publish Metadata for a Service Using a Configuration File. [www-dokumentti]. Microsoft. [Viitattu 18.5.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/ms734765.aspx

MS. 2013. Your First ASP.NET Web API. [www-dokumentti]. Microsoft. [Viitattu 11.4.2013]. Saatavissa: http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api

Poznanski. 2011. Ways to Load Applications Faster – A Windows Phone Recipe. [www-dokumentti]. Arik Poznanski. [Viitattu 16.4.2013]. Saatavissa: http://blogs.microsoft.co.il/blogs/arik/archive/2011/03/25/ways-to-load-applications-faster-a-windows-phone-recipe.aspx

RD. 2013a. SOAP Syntax. [www-dokumentti]. Refsnes Data. [Viitattu 18.4.2013]. Saatavissa: http://www.w3schools.com/soap/soap_syntax.asp

RD. 2013b. Introductions to XML. [www-dokumentti]. Refsnes Data. [Viitattu 22.5.2013]. Saatavissa: http://www.w3schools.com/xml/xml_whatis.asp

Ricker. 2010. Microsoft announces ten Windows Phone 7 handsets for 30 countries: October 21 in Europe and Asia, 8 Novemeber in US. [www-dokumentti]. Thomas Rickers. [Viitattu 16.4.2013]. Saatavissa: http://www.engadget.com/2010/10/11/microsoft-announces-ten-windows-phone-7-handsets-for-30-countrie/

SS. 2013. A Brief History of Android. [www-dokumentti]. Spice Stellar. [Viitattu 17.5.2013]. Saatavissa: http://visual.ly/brief-history-android

YY. 2010. HTTP Explained: What Does HTTP Stand for, What is HTTP Meaning and HTTP's Definition? [www-dokumentti]. Yang Yang. [Viitattu 22.5.2013].

Saatavissa: http://www.kavoir.com/2009/03/http-explained-what-does-http-stand-for-what-is-http-meaning-and-https-definition.html

LIITTEET