Haaga-Helia
ammattikorkeakoulu Oy

Käsipäätelaitteiden hallintasovelluksen koodaus

Mikko Poikkilehto

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2019

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

22.04.2019

Tekijä(t)
Poikkilehto, Mikko.

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Käsipäätelaitteiden hallintasovelluksen koodaus

Sivu- ja liitesivumäärä
20 + 1

Opinnäytetyön tarkoituksena on tuottaa Windows CE ja Windows Mobile –pohjaisille käsipäätteille etähallintasovellus. Työn lähtökohtana on yrityksen tarve laitteiden etähallintaan, jotta voidaan tehokkaammin hallita suurta laitemäärää. Sovelluksen tavoitteena pitää yllä laiterekisterä sekä muokata ja päivittää käsipäätteiden tiedostoja, jotka määrittelevät asetukset. Suurin osa markkinoiden ratkaisuista tukevat iOS ja Android laitteita erittäin hyvin mutta Windows CE tai Windows Mobile laitteita rajoitetusti tai ei ollenkaan.

Sovelluksen toteuttamiseksi valittiin C# -kieli koska siitä on myös .NET Compact Framework -versio saatavilla, joka toimii Windows CE ja Windows Mobile laitteilla ja sisältää kaikki tarvittavat toiminnot. Valitulla C#-kielellä toteutettiin myös Windows-palvelimella toimiva palvelinsovellus. Projektin tuloksena saatiin asiakassovellus käsipäätteille, palvelinsovellus sekä tietokanta.

Asiasanat
.Net Framework, C#-kieli, Windows CE, ohjelmointi

Sisällys

1 Johdanto

Opinnäytetyön tarkoituksena oli toteuttaa kaupanalan yritykselle sovellus, jolla voidaan seurata ja päivittää myymälöissä olevia käsipäätteitä keskitetysti. Yritys on kaupanalalla toimiva vähittäiskauppaketju, jolla on toimipisteitä ympäri Suomea. Käsipäätteet sijaitsevat myymälöissä ja niitä käytetään myymälöiden taustatoiminnoissa. Lähtötilanteessa käsipäätteille ei ollut etähallintaohjelmistoa vaan ne piti lähettää IT-osastolle päivitettäväksi. Kehitysprojektina oli toteuttaa käsipäätteille hallintaratkaisu, että saataisiin tehostettua vikatilanteiden selvitystä ja muutoksenhallintaa sekä vähennettyä kustannuksia.

Tähän tarpeeseen yritys kartoitti markkinoilla olevia hallintaratkaisuja. Yrityksellä ei ollut tarvetta täysimittaiselle mobiililaitteiden hallintasovellukselle vaan tarpeena oli saada päivitettyä yksittäisiä tiedostoja käsipäätteillä. Sovellusmallin piti olla keskitetty niin, että käsipäätteiden asiakassovellus päivittäisi käsipäätteen asetukset hallintasovelluksen määritysten mukaan. Tarve oli vain Windows CE-käyttöjärjestelmä laitteiden laitehallintaan. Tämän vaatimuksen pohjalta ei ollut perusteltua hankkia täysimittaista mobiililaite hallintasovellusta. Ehdotin yritykselle, että voisin ohjelmoida tarpeeseen sopivan ohjelmiston, millä voidaan lähettää tiedostoja käsipäätteille. Ehdotus hyväksyttiin ja sovellus toteutettiin vaatimusten mukaan. Ohjelman valmistuttua oli se tarkoitus ottaa käyttöön myymälöiden käsipäätteissä.

Työssä kuvataan sovelluksen koodaamiseen tarvittavat ohjelmistot ja työkalut ja kuvataan projektin kulku sekä tuotos. Työn tavoitteena oli ohjelmoida palvelinsovellus ja asiakassovellus sekä tietokanta, jotka ovat yhteydessä toisiinsa. Käsipäätteet tulisi saada listattua palvelimen sovellukseen ja mahdollistaa tiedostopakettien lähetys käsipäätteille. Palvelin oli tarkoitus toteuttaa Windows-palveluna, joka käynnistyy automaattisesti Windowsin käynnistyessä. Näin palvelinta on helpompi hallita ja seurata mahdollisia vikatilanteita. Windows-palvelun hallintaa varten oli tarkoitus tehdä graafinen käyttöliittymä, joka mahdollisti palvelun käyttämisen eri tietokoneilta saman tietoverkon sisällä. Käsipäätteiden asiakassovellus pitäisi pystyä tarpeen tullen päivittämään etänä tietoverkon yli.

2 Sovelluksen suunnittelu ja toteutus

2.1 Tarvekartoitus

Windows CE ja Windows Mobile –laitteille on saatavilla useampia kaupallisia sovelluksia mutta harva enää tukee Windows CE tai Mobile käyttöjärjestelmää. Useimmissa tuki on lopetettu ja ominaisuuksien toimintaa ei taata uudemmissa versioissa. Syynä on se, että Microsoft lopettaa Windows CE käyttöjärjestelmän kehittämisen ja tuki loppuu 10.10.2023 (Microsoft 2019).

VMwaren AirWatch on mobiililaitehallinta sovellus yrityksille, joka tukee Windows CE-käyttöjärjestelmän versiota 6.x, ja Windows Mobile käyttöjärjestelmän versiota 6.5 (VMware 2019). Zebran SOTI mobiililaitehallinta sovellus tukee Windows CE .NET 4.2 sekä Windows Mobile 5.0 sekä myöhempiä versioita käyttöjärjestelmistä (Zebra 2019). Ne ovat täysimittaisi mobiililaitehallinta sovelluksia, mitkä ovat lisensoitu per asiakaspääte. Ohjelmistot kattavat useamman käyttöjärjestelmän ja ovat enemmän suunnattu puhelimien ja tablettien hallintaan.

Windows CE ja Mobile –laitteille on myös tarjolla kaupallisia ftp-sovelluksia millä tiedostoja voidaan lähettää yksittäisille käsipäätteille. Tämä malli todettiin hankalaksi ylläpidon kannalta, kun vaatimuksena oli keskitetty hallinta.

2.2 Arkkitehtuurimalli

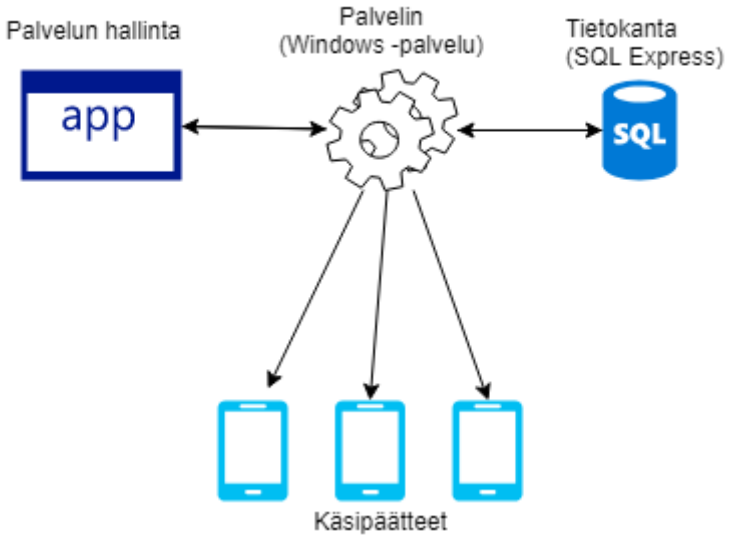
Projektin toteutus alkoi arkkitehtuurin määrittelyllä, arkkitehtuuriksi valittiin palvelin-asiakas malli. Tavoitteena oli voida lähettää tiedostoja käsipäätteille, näin vältyttäisiin käsipäätteiden lähettämisestä huoltoon asetuksien päivitystä varten. Suunnitteluvaiheessa päätettiin palvelimen ja asiakassovelluksen välinen viestintä toteuttaa XML-viesteinä, jotta viesteillä on johdonmukainen rakenne ja viestien sisältöä on helpompi käsitellä.

Käsipäätteen sovellus tahdottiin pitää toiminnoiltaan kevyenä akun keston vuoksi. Asiakassovellukseen toteutettiin myös päivitysominaisuus, jotta voidaan tarpeen tullen päivittää uusia ominaisuuksia verkon. Käsipäätteiden sovellus ottaisi aina käynnistyksessä yhteyden palvelimeen ja tämän jälkeen määrätyin väliajoin. Sovellus tarkistaisi onko päivityksiä saatavilla. Näin saataisiin yksinkertainen tapa pitää laiterekisteriä sekä päivittää laitteiden asetuksia ja tiedostoja.

Palvelinsovellus päätettiin toteuttaa Windows –palveluna, jotta sovellus voitiin määrittää käynnistymään automaattisesti ja tietovarastoksi valittiin Microsoft SQL Express –kanta,

jota palvelu hallitsee. Tietokannan avulla sovellus voi palvella useita satoja käsipäätteiden asiakassovelluksia samanaikaisesti. SQL Express on Windowsille tarkoitettu ilmainen tietokantaohjelmisto ja riittävä ominaisuuksiltaan. SQL Express tietokannassa on rajoitettu muistinkäyttöä yhteen gigatavuun ja kannan koko on rajoitettu kymmeneen gigatavuun (Microsoft 2019). Sovelluksen vaatimusmäärittelyssä ja testeissä todettiin, ettei yllä mainitut rajoitukset tule vastaan sovelluksessa. SQL Expressin lisäksi käytettiin Microsoftin SQL Management Studio –ympäristöä tietokannan ja taulujen luonnissa sekä tietokyselyiden luonnissa. SQL Management studio on ilmaiseksi saatavilla Microsoftin omalla lisenssillä. (Microsoft 2019). Tarkoituksena oli myös luoda graafinen käyttöliittymä ja hallintapaneeli.

Palvelun sovellus kuuntelee porttia mihin asiakassovellukset ottavat yhteyttä saman tietoverkon sisällä. Laiterekisteri tallennetaan SQL-kantaan ja siitä luodaan graafinen hallintanäkymä. Sovelluksen toimintaperiaate koostuu palvelimesta, asiakassovelluksesta, käyttöliittymästä sekä tietokannasta, toimintakaavio on esitelty kuvassa 1.

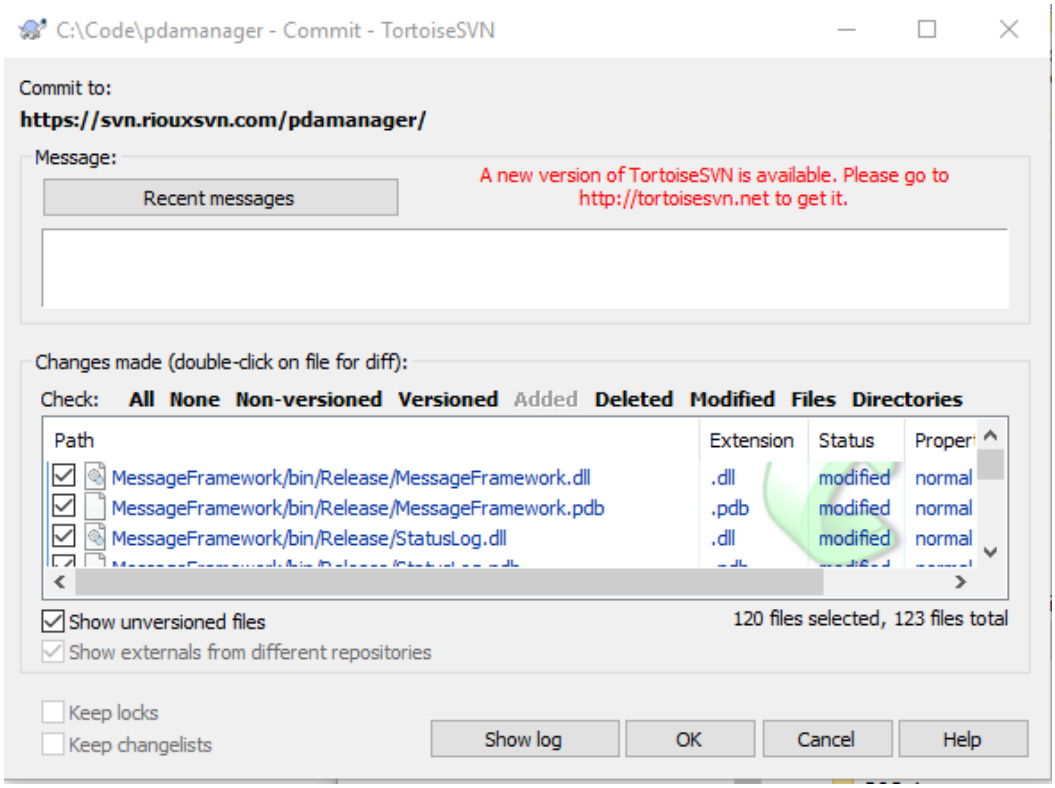


Kuva 1. Sovelluksen toimintaperiaate

2.3 Versionhallinta

Yleisimpiä versionhallintajärjestelmiä ovat SVN ja Git. Tässä tapauksessa valittiin versionhallinnaksi SVN. Se on Apache Software Foundationin kehittämä keskitetty versionhallintajärjestelmä, jolla voidaan hallita ja seurata tiedostojen muutoksia (Apache 2019). SVN versiohallinnan asiakasohjelmaksi TortoiseSVN. Se on graafinen hallintatyökalu SVN versionhallinnalle (Tortoise 2019). SVN valittiin koska siitä oli aiempaa kokemusta ja siihen löytyi ilmaisia palveluntarjoajia. Versionhallinnan avulla ohjelmakoodista oli varmuuskopio

palveluntarjoajalla paikallisen version lisäksi ja koodin ylläpito sekä päivittäminen ovat helpompaa. Koodinkansion versiopäivitys TortoiseSVN ohjelmalla on esitetty kuvassa 2.



Kuva 2. TortoiseSVN käyttöliittymä

SVN palveluntarjoajaksi valittiin RiouxSVN. Se tarjoaa yksityishenkilöille ilmaista tallennustilaa viidelle ohjelmistoprojektille (RiouxSVN 2019). Ohjelmakoodi ladattiin RiouxSVN –palveluun Windowsille tehdyn Tortoise SVN-asiakassovelluksen avulla. Näin voitiin hallitusti päivittää toimiva versio palvelimelle ja tarvittaessa palata aiempiin versioihin virheiden sattuessa.

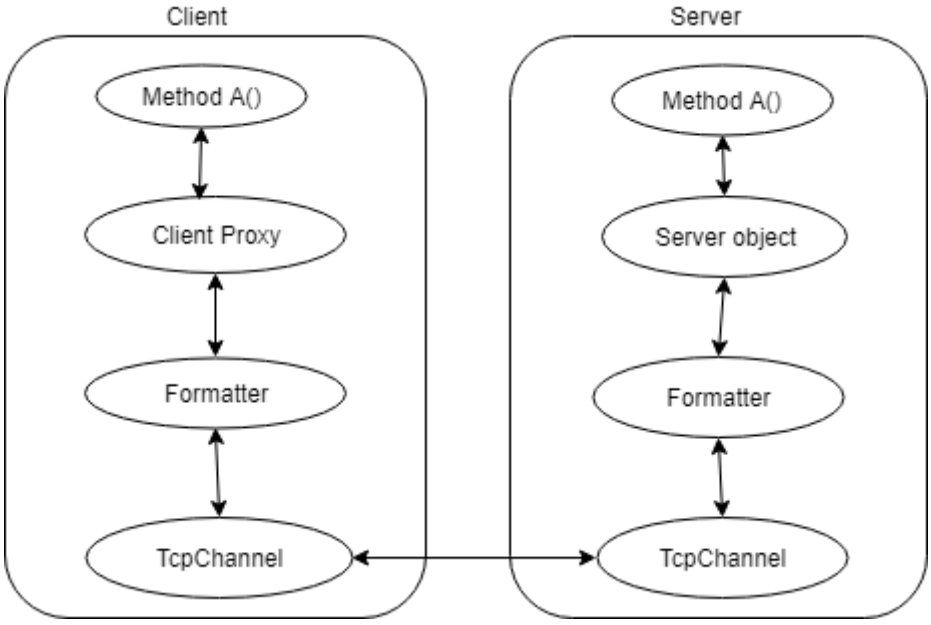
2.4 Ohjelmointikieli

Sovelluksen ohjelmointi toteutettiin Microsoftin C# -kielellä, se on moderni ja vahvasti tyypitetty oliopohjainen kieli, jolla on saman tyyppinen syntaksi kuin C-kielessä. Sillä ohjelmoidut sovellukset vaativat .NET Framework –ajoympäristön toimiakseen. Tämä takaa sen, että ohjelmat toimivat samalla tavalla käyttöjärjestelmästä riippumatta. (Petzold, C. 2007.)

Sovelluksen ohjelmointiin käytettiin NET Framework 3.5 sekä .NET Compact Framework 3.5 ajoympäristöversioita. Windows CE käsipäätteiden sovellus toteutettiin NETCF 3.5

ohjelmistokomponenttikirjastolla. Se on rajoitetuille resursseille optimoitu versio täydestä .NET Framework 3.5 kirjastosta. Käytetty .NET Framework Compact Framework tukee mm. Windows CE ja Windows mobile käyttöjärjestelmiä ja toimii myös Windowsilla. Windows CE sovellusta varten käytettiin myös Power Toys for .NET Compact Framework 3.5 lisäosaa, joka tuo lisäominaisuuksia sovellusten valvontaan, suorituskyvyn seurantaan ja virheenkorjaukseen (Microsoft 2019).

C#-kielessä on monia sisäänrakennettuja ominaisuuksia, jotka helpottavat palvelin-asiakas sovellusten ohjelmointia. Ohjelmointikielen standardikirjastoihin kuuluu TCP-yhteysluokkia, jolla voidaan TCP-yhteyksien luonti ja hallinta toteuttaa tehokkaasti ilman alemman tason TCP-yhteyshallintaa (Blum 2013). Sovelluksessa hyödynnettiin .NET Frameworkin Remoting ominaisuutta, sen arkkitehtuurimalli on esitetty kuvassa 3. Remoting-arkkitehtuurin avulla voidaan luoda proxy-olioita. Proxy-olio on asiakassovelluksen olio, joka esittää palvelimella sijaitsevaa luokkaa. Proxyn avulla voidaan suorittaa palvelimen metodeja proxy-olion avulla. (Rammer. I & Szpuszta M. 2019.). Sovelluksen palvelimen ja graafisen käyttöliittymän kommunikointi on toteutettu Remoting-arkkitehtuurilla.



Kuva 3. Remoting arkkitehtuuri

NET-Framework sisältää valmiit luokat TCP-yhteyksien käsittelyyn ja säikeiden hallintaan. Sovelluksen TCP-palvelimessa yhteyksien vastaanottaminen ja hallinta on toteutettu sisäänrakennettujen luokkien avulla. Asiakassovelluksessa käytetään myös samoja ohjelmaluokkia yhteyksien hallintaan. Näiden ominaisuuksien avulla saatiin ohjelmointia ja kehitystä nopeutettua huomattavasti.

Käsipäätesovelluksen ohjelmoinnissa käytettiin lisäksi OpenNetCF –kirjastoa, joka tuo Compact Frameworkiin lisää ominaisuuksia Windows CE ja Windows Mobile käyttöjärjestelmiä varten (Github 2019). Sen avulla saatiin helpommin käsipäätteestä laitetietoja, kuten laitteen malli, valmistaja ja MAC-osoite, tietojen lukemiseen käytettyä koodia on esitelty taulukossa 1. Ilman edellä mainittua kirjastoa laitetietojen luku olisi pitänyt toteuttaa C-kielellä.

Taulukko 1. Laitetietojen lukeminen laitteesta

```
if (!clientInfo.OSVersion.Equals("Win32NT"))
{
     //Luetaan laitteen yksilöllinen tunniste
     deviceId = OpenNETCF.WindowsCE.DeviceManagement.GetDeviceID();
}
if (!clientInfo.OSVersion.Equals("Win32NT"))
{   //Luetaan laitemalli lähetettävään viestiin
    sendMessage.clientInfo.Model =
OpenNETCF.WindowsCE.DeviceManagement.OemInfo;
}
```

2.5 Ohjelmointiympäristö

Ohjelmointiympäristönä käytettiin Visual Studio 2008, joka on uusin versio mikä tukee .NET Compact Framework 3.5 sekä .NET Framework 3.5 sekä tarvittavia laite emulaattoreita. Visual Studio 2008 tukee C# ohjelmointikieltä ja tarjoaa hyvän ajonaikaisen virheenkorjaus tuen (Microsoft 2019).

Siinä on myös tuki sovellusten asentamiseen ja asennuspakettien luomiseen Windows CE ja Windows Mobile –laitteille (Microsoft 2019). Visual Studioon voi asentaa emulaattorit Windows CE ja Windows Mobile käyttöjärjestelmille, jolloin ohjelman toimintaa voidaan testata ilman fyysistä laitetta. Emulaattori nopeuttaa ohjelman kehittämistä ja testaamista. Kuvassa 4 on esitetty emulaattorin näkymä käyttöjärjestelmästä.

Kuva 4. Windows Mobile emulaattori

Visual Studio 2008 tukee Windows-palvelu projekteja ja MSI -asennuspaketti projekteja (Microsoft 2019). Samalla ohjelmointiympäristöllä saatiin tehtyä kaikki sovelluksen osat ja asennusohjelmat. Ilman asennusohjelmaa palvelinsovellus on hankala luoda Windows-palveluksi.

Ohjelmointiympäristön avulla saatiin ohjelmoitua palvelin, käyttöliittymä sekä asiakassovellus ja suuri osa koodista voitiin jakaa ohjelmien välillä. Käyttöliittymä luotiin graafisella Windows Forms editorilla, mikä nopeutta kehitystä. Windows Forms on ohjelmakirjastokokoelma .NET Frameworkille, jonka avulla voidaan luoda visuaalisia lomakkeita ja lisätä niihin ohjaustoimintoja. Se yksinkertaistaa yleisiä sovelluksen toimintoja, kuten tiedostojen lukemista ja tiedostoihin kirjoittamista. (Microsoft 2019.)

Windows Mobile Device Center -ohjelmaa käytettiin käsipääte laitteiden yhdistämiseen tietokoneeseen. WMDC:n avulla Windows CE ja Windows Mobile –laitteita voidaan yhdistää tietokoneeseen USB-johdolla, sarjakaapelilla tai Bluetoothin kautta. Ohjelman kautta tiedostoja voidaan kopioida ja muokata sekä voidaan asentaa CAB-asennuspaketteja laitteelle. WMDC:n avulla voidaan myös jakaa tietokoneen internetyhteys käsipäätteen kanssa.

2.6 Toteutus

Sovelluksesta tehtiin ensin ”Proof of Concept” versio. Se on versio, jolla voidaan todentaa sovelluksen arkkitehtuurimalli toteuttamiskelpoiseksi. Tämän version avulla voitiin varmistaa, että TCP-yhteydet toimivat käsipäätteiden ja palvelinsovelluksen välillä sekä valitun formaatin XML-viestit säilyvät ehjinä verkon yli siirrettynä.

XML -viestin formaattiin luotiin eri viestiluokkia, joiden mukaan palvelin ja käsipääte viestivät. Viestiluokat määrittävät mikä on nykyinen toiminta tai tila ja niiden avulla voidaan valita seuraava toiminto. Aina kun käsipääte ottaa palvelimeen yhteyttä se lähettää ensin tarkistusviestin, että palvelimen ja käsipäätteen välinen TCP-yhteys toimii. Viestien tyypit on selitetty taulukossa 2.

Taulukko 2. Message.cs

```
public enum MessageType
{
    BYE,                 //yhteys lopetetaan
    CHECK_UPDATES,       //tarkistetaan onko päiviyksiä
    NOUPDATES,           //ei päivityksiä saatavilla
    NEW_UPDATE_PACKAGES,//uusia päivityksiä on saatavilla
    GET_PACKAGE,         //lataa paketti
    CHECK_ALIVE,         //tarkistetaan tcp-yhteys
    ISALIVE,             //palvelin on tavoitettavissa
    READY,               //valmis seuraavaan toimintaan
    IDLE,                //odottaa seuraavaa toimintaa
    DONE,                //toiminta valmis
    BUSY,                //toiminto kesken
    HANDLE_DATA,         //tiedoston datavirran käsittely kesken
    CLIENT_WAITING_DATA,//odottaa dataa palvelimelta
    SERVER_SENDING_DATA,//palvelin lähettää dataa
    FINISHED,            //toiminto päättynyt
    ERROR                //virhe
};
```

Verkkoprotokollaksi valittiin TCP ja vapaavalintainen portti. Ohjelmasta tehtiin ensin palvelimen TCP-ohjelma mikä vastaanottaa XML-viestejä asiakkaalta, .NET Frameworkista löytyy valmiit kirjastot datan muuntamiseen XML-formaattiin ja sen lähettämiseen. C# -kielessä on XmlSerializer-luokka, jolla voidaan kielen objekteja viedä XML-tiedostomuotoon ja luoda sen pohjalta myös uusia luokkia. Palvelin luo viestin XML-muotoon ja lähettää sen TCP-viestinä asiakassovellukselle, joka muuttaa viestin C#-objektiksi ja käsittelee tiedot. XML-tiedostomuotoa käytetään myös palvelinsovelluksen ja käsipäätesovelluksen asetuksien tallentamisessa, jolloin asetusten muuttaminen on helpompaa eikä niitä tarvitse sisällyttää koodiin. Asiakasohjelman asetustiedosto on esitelty taulukossa 3 ja palvelinsovelluksen asetustiedosto taulukossa 4.

Taulukko 3. Asiakassovelluksen asetustiedosto

```
<?xml version="1.0" encoding="utf-8"?>
<ClientConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <LogLevel>NONE</LogLevel>
  <LogPath>\Temp</LogPath>
  <ClientVersion>1.0.0.1</ClientVersion>
  <UpdatePending>1</UpdatePending>
  <ConnectionInterval>21600000</ConnectionInterval>
  <ServerName>server</ServerName>
  <Port>12345</Port>
  <ServerIP>127.0.0.1</ServerIP>
  <ConfigVersion>1</ConfigVersion>
  <GroupId>0</GroupId>
</ClientConfig>
```

Taulukko 4. Palvelinsovelluksen asetustiedosto

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <logLevel>NONE</logLevel>
  <configVer>1</configVer>
  <serverPort>12345</serverPort>
  <connectionString>Data Source=servername;Integrated Security=False;Initial Cata-
log=PDAdb;User ID=user;Password=password;Connect
Timeout=15;Encrypt=False;TrustServerCertificate=False</connectionString>
</Configuration>
```

Palvelinsovelluksesta tehtiin Windows-palvelu ja siihen erillinen visuaalinen käyttöliittymä, joka hyödyntää C#-kielen Remoting-ominaisuuksia. Näin Windows-palvelun ja käyttöliittymän ei tarvitse olla samalla tietokoneella vaan palvelua voidaan käyttää saman lähiverkon tietokoneilta. Palvelulle ohjelmoitiin myös graafinen käyttöliittymä, joka ottaa yhteyden Windows-palveluun.

Palvelin toteutettiin monisäikeisenä, jotta se pystyy käsittelemään useita satoja asiakkaita samaan aikaan. Käynnistyessään palvelin aloittaa uuden säikeen, jossa TCP-porttia kuunnellaan, näin ohjelma vastaa muihinkin toimintoihin samalla ja TCP-palvelinta voi-

daan ohjata. Säie joka kuuntelee TCP-porttia ohjaa tulevat yhteydet erilliseen säiepooliin. TCP-porttia kuunteleva sovelluksen osa on kuvattu taulukossa 5.

Taulukko 5. ServerCore.cs

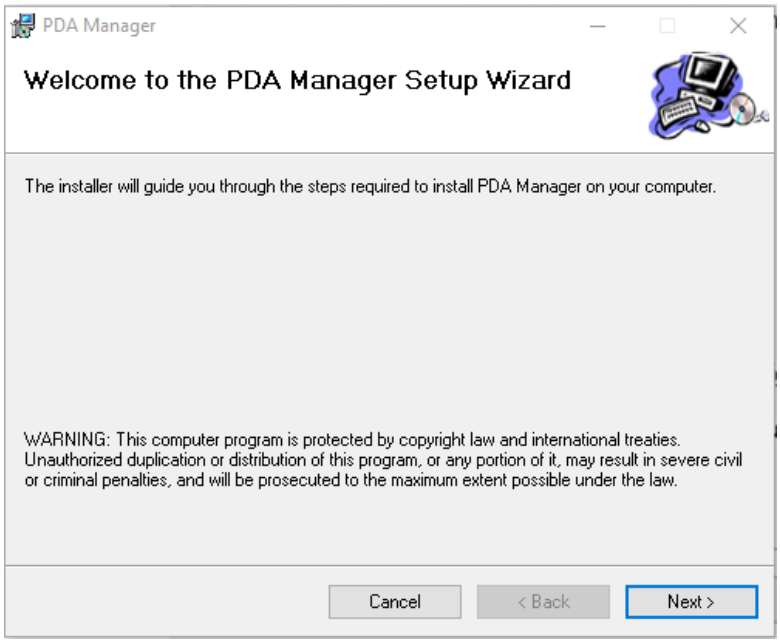
```
do
{
    if (!clientLis.Pending())
    {
       Thread.Sleep(50);
       continue;
    }
    TcpClient client = clientLis.AcceptTcpClient();
    ConnectionThread newconnection = new ConnectionThread(dbHandler,
    client, serializerObjMessage, logger);
    } while (ServerRunning);
```

Säiepooliin löytyi C#-kielestä valmis ”ThreadPool”-luokka, jolla voidaan eri metodeja suorittaa asynkronisesti. Tulevat yhteydet luodaan säiepooliin, mistä voidaan asiakkaita palvella ja käyttää säikeitä uudelleen yhteyden katkettua. Näin säikeitä ei tarvitse erikseen luoda ja hallita vaan C#-tekee sen automaattisesti. TCP-yhteyksien vastaanotto ja säiepooliin lisääminen on kuvattu taulukossa 6.

Taulukko 6. ConnectionThread.cs

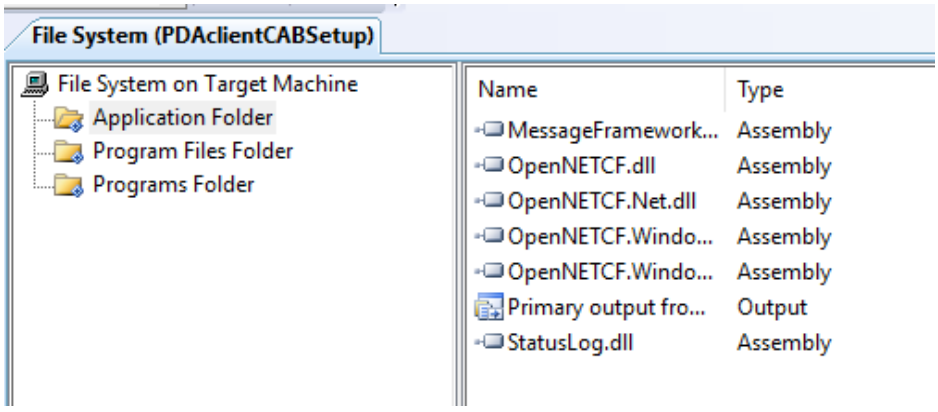
```
public ConnectionThread( IDeviceRepository dbHandler,TcpClient
lisClient, XmlSerializer serializerObjMessage,  ILogging logger)
{
     this.log = logger;
     this.serializerObjMessage = serializerObjMessage;
     this.dbHandler = dbHandler;
     ThreadPool.QueueUserWorkItem(obj => HandleConnection(lisClient));
}
```

Palvelinohjelmistosta tehtiin kaksi eri MSI-asennuspakettiprojektia: ”PDA Manager Setup” ja ”PDA Manager Gui Setup”. Näistä ensimmäinen asentaa palvelinsovelluksen sekä Windows-palvelun. Näin sovellus toimii kaikilla Windows-versiolla, jotka tukevat .NET 3.5 versiota ja ohjelma käynnistyy automaattisesti. Windows palveluna sovellusta voidaan hallita paremmin ja virheet jäävät Windowsin lokiin sekä sovelluksen omaan lokitiedostoon. Jälkimmäinen projekti asentaa sovelluksen graafisen käyttöliittymän. Palvelun asennusohjelman käyttöliittymä on esitelty kuvassa 5, sen avulla voidaan ohjelma asentaa haluttuun tiedostopolkuun.



Kuva 5. Palvelinsovelluksen MSI-asennusohjelma

Käsipäätteen asiakassovellusta varten tehtiin oma Compact Framework –projekti Visual Studioon, se käyttää osin samoja jaettuja kirjastoja kuin palvelinversio. Näin saatiin keskitetysti luotua viestiformaatit ja millä sovelluksen kommunikoi. Käsipäätteelle tehtiin myös Cabinet-asennuspaketti, mikä asentaa sovelluksen käsipäätteelle. Cabinet-tiedostoformaatti on Microsoftin luoma tiedostoformaatti, jolla voidaan pakata useita tiedostoja ja kansioita kompressoituna yhteen cabinet-tiedostoon (Microsoft 1997). Cabinet asennuspaketin luonti on esitelty kuvassa 6. Siinä määritetään asennettavat tiedostot, asennuspolku ja pikakuvakkeet ohjelmalle.



Kuva 6. Cabinet asennus-projekti

Käsipäätteen asiakassovelluksen rinnalle toteutettiin päivitysohjelmisto, millä voidaan myös päivittää itse asiakassovellus. Palvelinsovelluksessa luodaan uusi asiakassovellus-paketti, jonka käsipääte lataa tilapäiskansioon, sulkee asiakassovelluksen ja käynnistää päivitysohjelman. Päivitysohjelma siirtää sovellustiedostot tilapäiskansioista ohjelmakan-

sioon ja käynnistää asiakassovelluksen uudelleen. Näin myös sovelluksen päivitykset voidaan tehdä hallitusti etänä. Päivityssovelluksen prosessien hallinta on kuvattu taulukossa 7. Toiminnon toteutus oli hankala ja vaati paljon kokeiluja, miten päivityssovellus voi käynnistää toisen prosessin ja sulkea oman prosessinsa sen jälkeen.

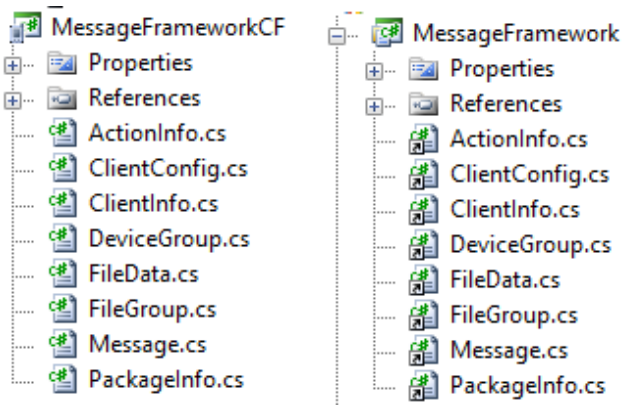
Taulukko 7. Asiakassovelluksen päivityssovellus

```
try
{
//Kopioidaan tiedostot sovelluskansioon
foreach (string newPath in Directory.GetFiles(CurrentDir +
@"\PDAclientUpdate", "*.*"))
                    File.Copy(newPath, newPath.Replace(CurrentDir +
@"\PDAclientUpdate", CurrentDir), true);
//Poistetaan tilapäistiedostot
Directory.Delete(CurrentDir + @"\PDAclientUpdate", true);

//Käynnistetään asiakassovellus
System.Diagnostics.ProcessStartInfo processStartInfo = new
System.Diagnostics.ProcessStartInfo();
                processStartInfo.FileName = CurrentDir +
@"\PDAclient.exe";
System.Diagnostics.Process.Start(processStartInfo);
}
Application.Exit(); // Suljetaan päivityssovellus
```

Sovellukselle tehtiin Visual Studioon projektit ”PDAManager”, jonka alle tehtiin yhteensä 14 projektia, joista kolme on asennuspaketti projekteja. Projektit ovat kuvattu liitteessä 1.

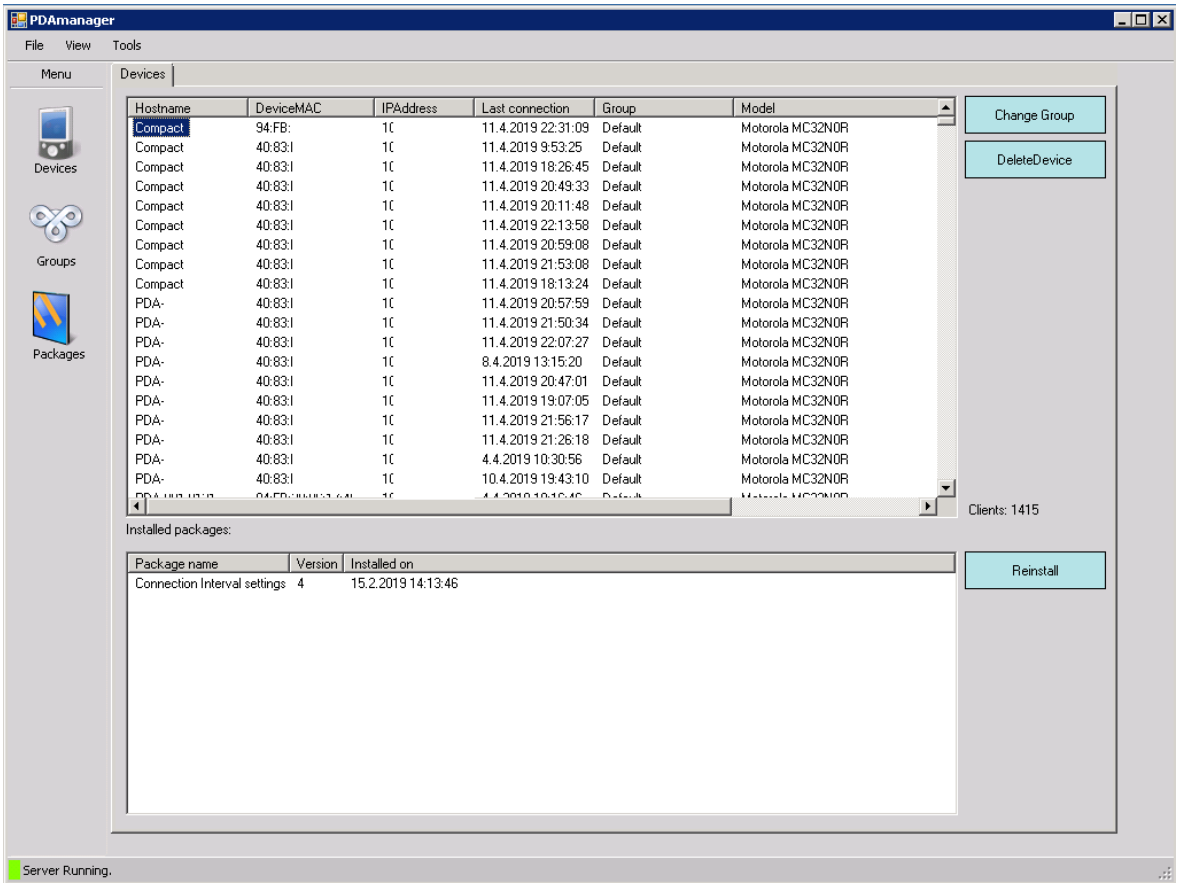
Viestintää varten tehtiin Visual Studioon kaksi projektia, toinen käsipäätteille ja toinen .NET täysversioon. Ohjelmaluokat ovat projektissa ”MessageFrameworkCF” ja niihin viitataan projektissa ”MessageFramework”, näin samaa koodia voitiin käyttää käsipäätesovelluksessa ja palvelinsovelluksessa. Toteutus esitetään kuvassa 7, lähdekoodi jaettiin vastaavasti myös tapahtumalokitieto –projekteissa ”StatusLog” ja ”StatusLogCF”.



Kuva 7. Lähdekoodin jakaminen projektien kesken

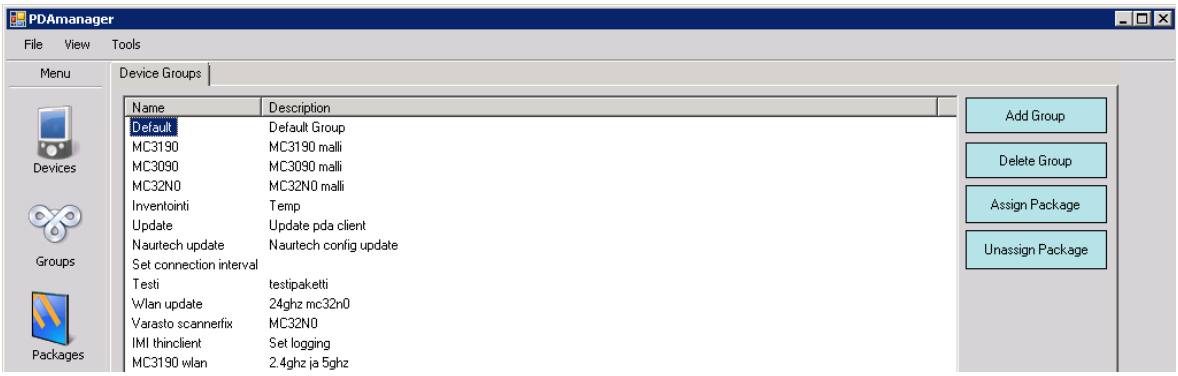
Käsipäätteen sovellus tehtiin mahdollisimman huomaamattomaksi, se ei näy käyttäjälle. Sovellus käynnistyy ja aloittaa säikeen, joka ottaa määrätyin väliajoin yhteyden palvelimeen ja käsittelee viestit. Sovellus kirjoittaa tapahtumalokia tiedostoon ja sen tarkkuutta voidaan vaihdella. Tapahtumalokin avulla on helpompi selvittää mahdollisia vikatilanteita.

Käyttöliittymä toteutettiin Windows Forms –sovelluksena, sillä on helppo luoda ulkoasu sovellukselle Visual Studion graafisella käyttöliittymällä. Sovelluksen käyttöliittymään suunniteltiin kolme näkymää: Devices, Groups ja Packages. Näistä ensimmäisessä Devices -näkymässä, listataan kaikki palvelimeen äskettäin yhteydessä olleet laitteet. Näkymässä on myös laitteen ip-osoite, laitemalli, ryhmä, mac-osoite ja laitenimi. Näkymästä voi myös vaihtaa laitteiden ryhmiä, poistaa laitteita tai asentaa tiedostopaketti uudelleen. Device-näkymän käyttöliittymä on esitetty kuvassa 8.



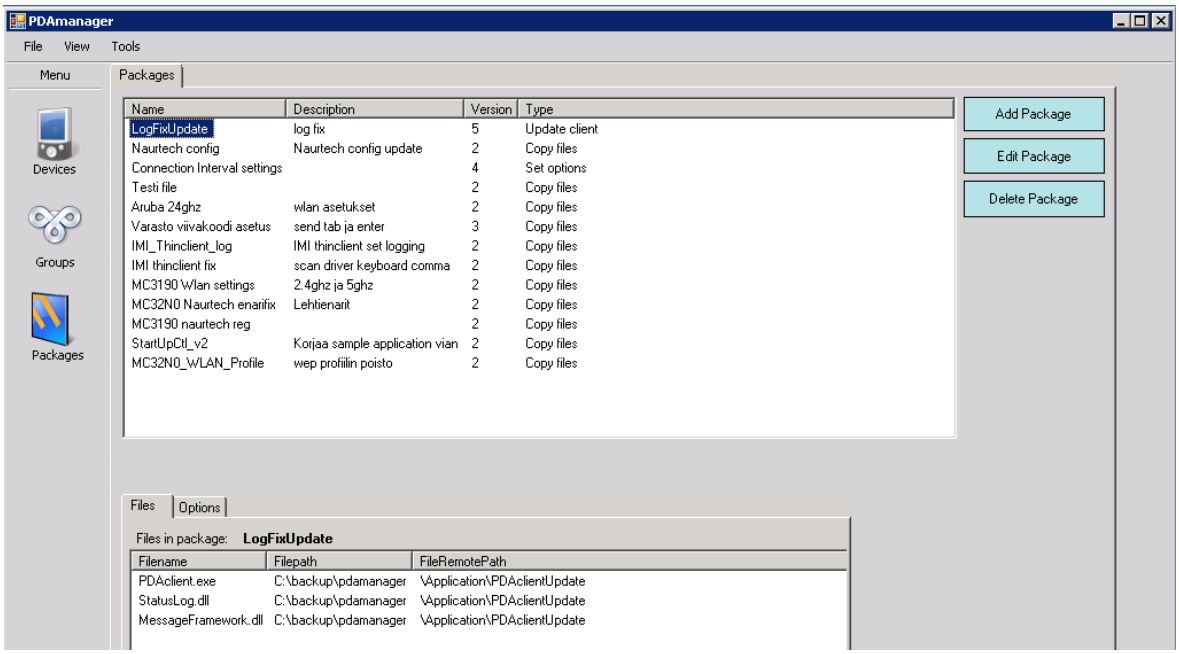
Kuva 8. PDA-manager Devices-näkymä

Groups-näkymässä voidaan luoda tai poistaa ryhmiä laitteille sekä liittää tai poistaa niitä asennuspaketeista. Ryhmiin liitetyt asennuspaketit lähetetään käsipäätteille, jotka kuuluvat kyseiseen ryhmään. Näin voidaan erotella eri tarpeisiin ja eri laitemalleille omat ryhmät. Groups-näkymä esitellään kuvassa 9.



Kuva 9. PDA Manager, Devices-näkymä.

Packages-näkymässä hallitaan asetuspaketteja, se esitellään kuvassa 10. Ohjelmassa on kolmen tyyppisiä paketteja: ”Copy files”, jolla voidaan lähettää tiedostoja laitteelle. Paketti ”Set options” määrittää asiakassovelluksen yhteys aikavälin ja palvelimen osoitteen. Paketti ”Update client” päivittää itse asiakassovelluksen laitteella. Paketeista pidetään versiontia ja aina kun pakettia muutetaan, se lähetetään sille asetetun ryhmän käsipäätteille uudelleen.



Kuva 10. PDA Manager, Packages-näkymä.

Tietokanta toteutettiin siten että siihen voidaan tallentaa käsipäätteiden laitetietoja, tietoja asennetuista paketeista ja milloin laitteet ovat viimeksi ottaneet palvelimeen yhteyttä. Käsipäätteiden MAC-osoite toimii yksilöivänä tunnisteena, jolla tiedot yhdistetään käsipäätteeseen. Tietokantaan suunniteltiin myös jatkokehitystä varten taulu, johon voidaan luoda

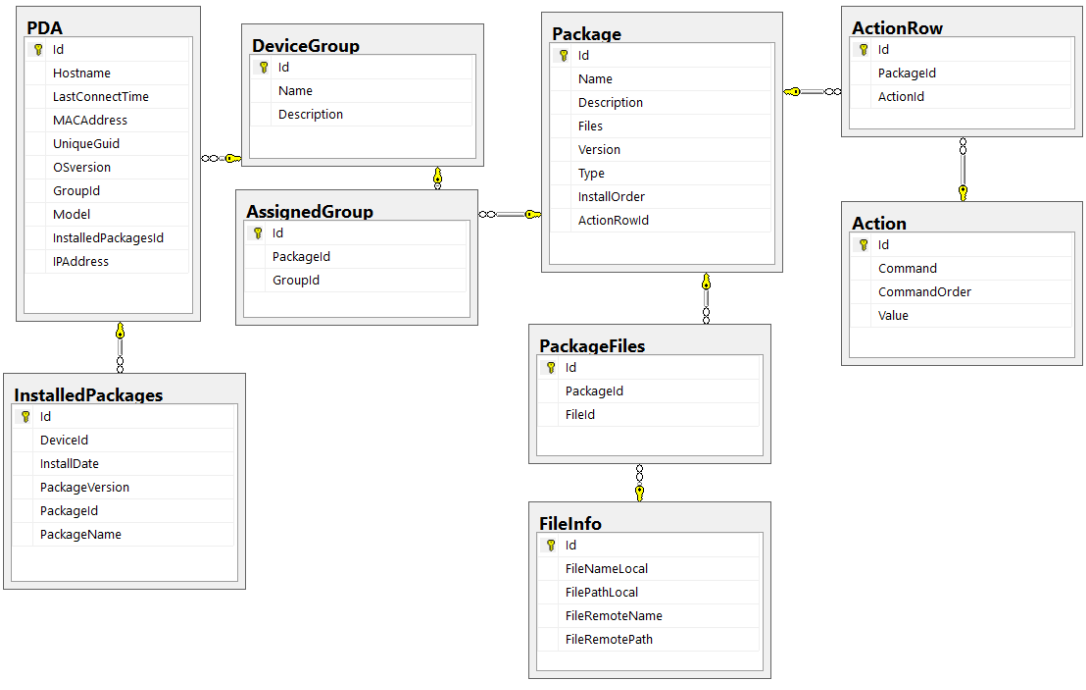
käsipäätteillä ajettavia komentoja. Komentojen suorituksen toteutus jätettiin projektin ulkopuolelle työmäärän vuoksi.

SQL –tietokannan taulut luotiin vaiheittain, kun edellinen toiminto oli saatu valmiiksi ja testattu. Lopulliseen versioon tuli yhteensä 9 taulua. Taulut ovat esitetty taulukossa 8.

Taulukko 8. SQL taulut.

PDA	Käsipäätteen tiedot sisältä taulu.
DeviceGroup	Laiteryhmä, joka määrittää asennettavat paketit.
AssignedGroup	Ryhmälle määritetyt asennuspaketit.
Package	Paketti-taulu, sisältää lähetettävät tiedostot tai asetukset.
PackageFiles	Paketin sisältämät tiedostot.
FileInfo	Yhden tiedoston tiedot ja kohdepolku.
ActionRow	Käsipäätteellä suoritettavat komennot .
Action	Komento, joka suoritetaan ja järjestys.
InstalledPackages	Asennetut paketit ja niiden versiot.

Sovelluksen tärkein taulu on PDA-taulu, se sisältää laitetiedot, sekä viimeisen yhteysajan. Taulujen relaatiot on esitetty kuvassa 11.



Kuva 11. SQL-taulujen relaatiot.

Sovelluksen tietokannasta luotiin SQL-skripti, joka luo automaattisesti sovelluksen käyttämän tietokantarakenteen. Tietokannan yhteysasetukset ovat määritettävissä sovelluksen ohjelmakansion xml-asetustiedostossa.

2.7 Tuotos

Projektin tuotoksena tuli lopulta neljä eri komponenttia. Windows -palvelu, joka hallitsee laitteiden tietojen tallennusta ja hakua SQL-tietokannasta. Graafinen käyttöliittymä palvelun hallitsemiseen sekä käsipäätteen asiakassovellus. Käsipäätteen sovellus ottaa palvelimeen yhteyttä aina käynnistyksen yhteydessä, sekä määrätyin väliajoin. Sovellus tarkistaa palvelimelta onko sille saatavilla uusia tiedostopaketteja. Mikäli käsipäätteelle on tiedostopaketti saatavilla, se lataa paketin tiedostot ja tallentaa ne paketin määrittämiin kansioihin. Näin voidaan päivitettyjä tiedostoja jakaa käsipäätteille hallitusti ilman, että niitä tarvitsee lähettää huoltoon ohjelmistopäivitystä varten. Sovellus on tuotantokäytössä ja se palvelee yli tuhatta käsipäätettä, joissa asiakassovellus on asennettuna.

Sovelluksen palvelin toteutettiin niin, että se voi palvella mahdollisimman monta asiakassovellusta samaan aikaan. Palvelin toimii monisäikeisenä ja luo aina uuden säikeen jokaista yhteyttä ottavaa asiakasta varten. Säikeiden luonnissa hyödynnettiin C# kielen valmista säikeidenhallinta ThreadPool-luokkaa. ThreadPool –luokka käyttää Windowsin säikeitä, jolloin säikeiden luonti ja hallinta ovat automaattisia. Luokan staattisilla metodeilla voidaan luoda delegaatteja säikeille. (Blum 2013.)

Sovellukseen toteutettiin komponentti tapahtumien tallentamiseen tiedostoon, virheenkorjausta varten. Toiminnossa hyödynnettiin myös XMLSerializer –toimintoa, jolla saatiin TCP-viestien sisällöt luettavaan muotoon. Kirjattavien tapahtumien raportointitarkkuudelle luotiin neljä eri tasoa, jotta voidaan tarvittaessa seurata hyvin tarkasti ohjelman toimintaa. Raportointitasot ovat kuvattu taulukossa 9. Näin saatiin nopeasti selville missä kohdassa ohjelmaa mahdolliset virheet tapahtuivat.

Taulukko 9. Raportointitasot

```
    public enum LoggingLevel {
        NONE = 0, // ei tallenneta lokia
        FATAL = 1, //Vain vakavat virheet kirjataan
        NORMAL = 2, //Vakavat virheet ja sovelluksen käynnistyminen
        FULL = 3, //Kaikki tapahtumatiedot
        DEBUG = 4 //Kaikki tapahtumatiedot sekä xml-viestisisältö
    }
```

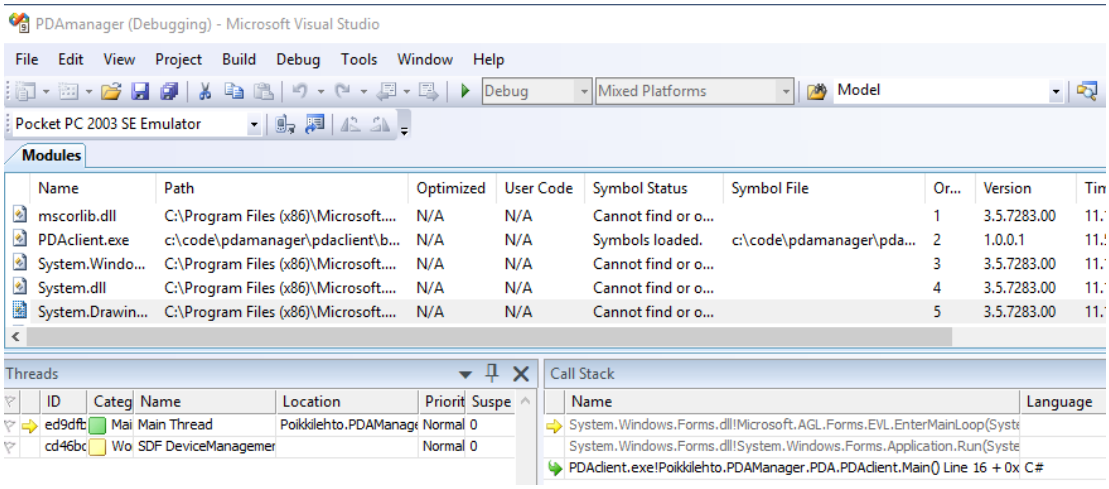
Ohjelmoinnin apuna käytettiin Visual Studion PocketPC-emulaattoria sekä Zebran käsipäätelaitteita, joilla sovellusta testattiin. Sovelluksen palvelinohjelmaa testattiin ensin Windows 10 tietokoneella, johon oli asennettu SQL Express. Asiakassovellusta kehitettiin alkuvaiheessa myös Windows 10 tietokoneella, että saatiin perustoiminnallisuudet valmiiksi. Tämän jälkeen ohjelmaan lisättiin Windows CE- käyttöjärjestelmällä toimivat osuudet ja kehitystä jatkettiin emulaattorilla sekä fyysisellä laitteella.

2.8 Mahdolliset ongelmat ja riskit

Sovelluksen koodaaminen on monivaiheinen prosessi ja sen kehitysvaiheessa muutosten hallinta on tärkeää, jotta voidaan palata tiettyyn versioon tarvittaessa. Sovelluskehityksessä käytetään yleisesti versionhallinta järjestelmiä tavoitteen saavuttamiseksi.

Sovellusta testattiin ensin paikallisella asennuksella, jolloin palvelin asennettiin kannettavalle tietokoneelle ja asiakasohjelma käynnistettiin samalla tietokoneella. Näin voitiin testata, että sovellukset toimivat. Pilotti vaiheessa palvelinsovellus asennettiin Windows 2008 –palvelimelle ja asiakassovellus muutamaan käsipäätteeseen. Testin jälkeen todettiin, että sovellus toimii ja se asennettiin kaikille käsipäätteille.

Sovelluksen ohjelmoinnin ja testauksen aikana kerättiin lokiin tarkkaa tietoa. Niin voitiin vahvistaa, että sovellus toimii oikein ja virheiden sattuessa saatiin nopeasti selville mistä virhe johtuu. Virheiden tutkintaan ja selvitykseen käytettiin Visual Studion virheenkorjausominaisuuksia, käyttöliittymä on esitelty kuvassa 12. Virheenkorjauksessa voidaan tarkastella sovelluksen ajonaikaisia dll-kirjastoja ja virheen tapahtuma hetkellä olleita muuttujien arvoja sekä kutsuttuja metodeja.



Kuva 12. Visual Studion virheenkorjaus käyttöliittymä

Testausvaiheessa tuli ilmi muutama puute asiakassovelluksessa, jotka korjattiin ja päivitykset lähetettiin käsipäätteille. Testausjakson jälkeen sovelluksessa ei todettu virheitä tai puutteita ja se otettiin käyttöön.

3 Pohdinta

Opinnäytetyön tavoitteena oli ohjelmoida työnantajalleni sovellus käsipäätteiden hallintaan sekä laajentaa omaa ohjelmointi osaamista. Saatavilla oleva materiaali .NET Compact Frameworkista on aika rajoittunutta ja useamman vuoden takaa tai ovat tarkoitettu uudemmille versioille. Käytettävää ohjelmointikieltä rajasi se, että harva ohjelmointikieli tukee Windows CE-käyttöjärjestelmää. Ohjelmointikieli vaihtoehdot olivat C# tai C -kieli, joten C# valittiin koska se tukee käytettyjen laitteiden käyttöjärjestelmiä sekä Windows-käyttöjärjestelmiä. Aiempaa ohjelmointikokemusta minulla oli enemmän Java- ja C++ -ohjelmointikielistä sekä VBscript –skriptaukset mutta C#-kielellä ohjelmoinnista ei juurikaan. Tästä syystä sovelluksen tavoitellut toiminnot olivat alussa hyvin yksinkertaiset mutta projektin ja osaamisen edetessä toimintoja lisättiin.

3.1 Pohdinta tehdystä

Ennen projektin aloittamista toteutin useamman konsolipohjaisen palvelin-asiakas TCP-ohjelman, jotta ymmärsin tarvittavat TCP-yhteyksien ja viestien välityksien toimintaperiaatteet. Pohjatietojen kerääminen ja käytettävien työkalujen opetteleminen veivät aikaa enemmän kuin oli alun perin ajateltu. Projekti lähti muutaman esimerkin pohjalta kasvamaan ja lisäsin sovellukseen osia sitä mukaan, kun aiemmat todettiin toimiviksi. Kehitys sujui aluksi hitaasti ja varsinkin tiedostopakettien lähetys toimintojen ohjelmoinnissa meni odotettua pidempään. Sovelluksen toteuttaminen pienemmissä osissa tehosti ohjelmointia ja koodin jakaminen eri osien välillä nopeutti ohjelmointia huomattavasti. Jouduin soveltamaan eri lähteitä, jotta sain osan koodista toimimaan käsipäätteen sovelluksessa. Osan .NET Frameworkille tehdyistä ohjeista pystyi toteuttamaan myös Compact Frameworkilla. Versionhallinta helpotti lähdekoodin hallintaa ja sen avulla oli nopea ladata lähdekoodi eri työasemille projektin aikana.

Ohjelmoinnissa kului paljon aikaa virheiden selvittämiseen ja ohjelmointiluokkien sekä metodien opettelemiseen. Ohjelmointia helpotti C#-kielen automaattinen muistinhallinta mikä vähentää sovelluksen muistivuotoja. Visual Studio tarjosi hyvät työkalut virheiden etsimiseen ja tapahtumatietojen tallennus tiedostoihin auttoi virheiden tutkimisessa. Käsipäätesovelluksen ohjelmoinnissa emulaattori nopeutti ohjelmointia, kun sovelluksen testaamiseen ei tarvittu aina fyysistä laitetta. Graafisen käyttöliittymän viimeistelyyn ja hiomiseen kului myös paljon aikaa mutta käyttöliittymästä tuli lopulta selkeä ja yksinkertainen.

3.2 Projektin hyödyt

Sovelluksesta tuli lopulta paljon laajempi ja työläämpi kuin alun perin ajattelin. Projektin aikana opin paljon C#-kielen ja .NET Frameworkin toiminnasta sekä ohjelmointimenetelmistä. Sen ansioista opin luomaan enemmän uudelleenkäytettävää koodia mitä voisin mahdollisesti hyödyntää tulevaisuudessakin. Vaikka sovellus toteutettiin vanhemmalla .NET versiolla, niin suurin osa toiminnoista on säilytetty uudemmissakin versioissa tai niiden tilalle on tehty tehokkaampia ja helpompia toteutustapoja. Sovelluksesta tuli toimiva ja se on tuotantokäytössä edelleen. Sovellus on helpottanut käsipäätelaitteiden ylläpitoa, kun asetuksia voidaan päivittää etänä. Se on myös tehostanut laitteiden elinkaarenhallintaa, tietokannan laiterekisterin avulla voidaan arvioida laitteiden uusimistarpeet paremmin. Projektilla ei ollut tarkkaa aikataulua ja sen takia sovelluksen edistyminen oli ajoittain hidasta, käyttöönotto sujui kuitenkin ongelmitta ja projekti täytti sille asetetut vaatimukset.

3.3 Jatkokehitys

Sovelluksen jatkokehitys rajoittuu mahdollisten virheiden korjaukseen. Toiminnallisuutta voidaan lisätä, jos ilmenee tarvetta uusille ominaisuuksille. Tulevaisuutta ajatellen sovellukseen jätettiin valmius jatkokehitykselle, päivityspakettien toiminnallisuutta voidaan laajentaa toteuttamalla lisää eri toimintoja mitä suoritetaan käsipäätteellä. Laitteista luettavia tietoja voidaan myös lisätä, käsipäätteistä on mahdollista saada luettua akun varauksen taso ja wlan-signaalin vahvuus. Sovelluksen käyttämä XML-viestiformaatti mahdollistaa myös sovelluksen yhdistämiseen muihin rajapintoihin tarvittaessa. Yhtenä kehitysvaihtoehtona olisi laajentaa sovellus toimimaan myös Android-käyttöjärjestelmässä.

Lähteet

Apache 2018. Apache Subversion Features. Luettavissa: https://subversion.apache.org/features.html. Luettu: 3.4.2019.

Blum, R. 2003. C# Network Programming. ISBN: 0782141765.

Github 2019. CStacke. OpenNetCF SmartDeviceFramework. Luettavissa: https://github.com/ctacke/sdf . Luettu:1.3.2019.

Microsoft. 2019. .NET Compact Framework 3.5 Redistributable. Luettavissa: https://www.microsoft.com/en-us/download/details.aspx?id=65. Luettu: 1.2.2019.

Microsoft 2019. Features Supported by the Editions of SQL Server 2012. Luettavissa: https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2012/cc645993(v=sql.110)#CrossBoxScale. Luettu: 29.3.2019.

Microsoft 1997. Microsoft Cabinet Format. Luettavissa: https://docs.microsoft.com/en-us/previous-versions//bb267310(v=vs.85). Luettu: 1.5 2019.

Microsoft 2019. Microsoftin elinkaari käytänto. Luettavissa: https://support.microsoft.com/fi-fi/lifecycle/search/1143. Luettu: 15.3.2019.

Microsoft. 2019. Power Toys for .NET Compact Framework 3.5. Luettavissa: https://www.microsoft.com/en-us/download/details.aspx?id=13442. Luettu: 4.4.2019.

Microsoft 2019. SQL Server Management Studio - License Terms. Luettavissa: https://docs.microsoft.com/en-us/legal/sql/sql-server-management-studio-license-terms. Luettu: 10.3.2019.

Microsoft 2008. Visual Studio 2008 Product Comparison. Luettavissa: https://www.microsoft.com/en-us/download/confirmation.aspx?id=7940&6B49FDFB-8E5B-4B07-BC31-15695C5A2143=1. Luettu: 3.5.2019.

Microsoft 2007. Windows Forms overview. Luettavissa: https://docs.microsoft.com/en-us/dotnet/framework/winforms/windows-forms-overview. Luettu: 5.5.2019.

Petzold C., .NET Book Zero. 2007. Luettavissa: http://www.charlespetzold.com/dotnet/DotNetBookZero11.pdf. Luettu: 19.4.2019.

RAMMER, I., SZPUSZTA M. 2005. Advanced .NET Remoting, Second Edition. Luettavissa: http://read.pudn.com/downloads124/ebook/526071/Apress%20-%20Advanced%20DotNET%20Remoting,%202nd%20Edition.pdf. Luettu: 21.4.2019.

RiouxSVN. 2019. Free Subversion Hosting. Luettavissa: https://riouxsvn.com/. Luettu: 1.1.2019

TortoiseSVN. 2019. About TortoiseSVN. Luettavissa: https://tortoisesvn.net/. Luettu: 10.1.2019.

VMware 2018. AirWatch. Supported Devices, OS, and Agents. Luettavissa https://docs.vmware.com/en/VMware-AirWatch/9.1/vmware-airwatch-guides-91/GUID-AW91-Before_Prod.html. Luettu: 1.4.2019.

Zebra. 2019. Supported Devices. Luettavissa: https://www.soti.net/mc/help/v14.1/en/setup/installing/supported_devices.html. Luettu: 10.4.2019.

Liitteet

Liite 1. Visual Studio -projektirakenne

MessageFramework	Palvelinsovelluksen xml-viesti luokka, projekti käyttää ”MessageFrameworkCF”-projektin lähdekoodia.
MessageFrameworkCF	Käsipäätteen xml-viestiluokka, kaikki sovellusten väliset viestit ohjelmoitiin tähän.
PDA.Repository	Käytettävien tietovarastojen rajapinnat ja toteutukset, mahdollistaa useamman tietokantaohjelman käyttämisen. Ainoastaan SQL-rajapinta on toteutettuna.
PDAclientUpdater	Käsipäätteen sovelluksen päivitysohjelma
PDAserver	Palvelinsovellus
PDAServerRemote	Palvelinsovelluksen etäkäyttö lähdekoodi
PDAServerGui	Palvelinsovelluksen käyttöliittymä
PDAclient	Käsipäätteen sovellus, projekti on Compact Framework alustalle.
PDAServerService	Palvelinsovelluksen Windows-palvelun lähdekoodi
StatusLog	Tapahtumaloki –luokat, lähdekoodit viittaavat StatusLogCF-projektiin
StatusLogCF	Tapahtumaloki –luokat, määritetty rajapinta ”ILogging” ja sen toteuttava luokka LogAppend
PDA Manager Gui Setup	Palvelun graafisen käyttöliittymän asennusprojekti. Projektilla luodaan MSI-asennuspaketti, jolla sovellus voidaan helposti asentaa Windowsille.
PDAClientCABSetup	Asiakassovelluksen asennusprojekti Windows CE ja Windows Mobile käyttöjärjestelmille. Projekti luo CAB-asennustiedoston, jolla sovellus voidaan asentaa käsipäätteille.