Tommi Lehtinen

SOVELLUKSIEN KEHITTÄMINEN RADIOTAAJUISEN ETÄTUNNISTUKSEN TARPEISIIN

Automaation koulutusohjelma

2012

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SOVELLUKSIEN KEHITTÄMINEN RADIOTAAJUISEN ETÄTUNNISTUKSEN TARPEISIIIN

Lehtinen, Tommi
Satakunnan ammattikorkeakoulu
Automaation koulutusohjelma
Syyskuu 2012
Ohjaaja: Peltonen, Kauko
Sivumäärä: 44

Asiasanat: RFID, etätunnistus, seuranta, ohjelmointi

Opinnäytetyön aiheena oli ensimmäisessä vaiheessa tutustua radiotaajuiseen etätunnistustekniikkaan (RFID), työssä käytettävään lukijalaitteeseen ja sen rajapintoihin. Toisena vaiheena oli tehdä kaksi ohjelmistoa, toinen radiotaajuisen etätunnistuksen esittelykäyttöön ja toinen tutkimuskäyttöön. Työn tilaajalla oli tarve tällaisille ohjelmille, koska aiemmin radiotaajuisen etätunnistuksen mahdollisuuksia ei pystytty esittelemään helposti käytännössä sekä tutkimustyöhön kaivattiin ohjelmaa tilaajan haluamilla ominaisuuksilla.

Työn ensimmäinen vaihe toteutettiin hankkimalla ja tutustumalla aihetta käsittelevään kirjallisuuteen ja laitteistoihin. Tästä muodostui opinnäytetyöraportin alkuosa. Toisessa vaiheessa tehtiin kaksi ohjelmistoa, jotka käyttävät toiminnassaan hyväksi radiotaajuista etätunnistusta. Näiden ohjelmien vaiheista muodostui opinnäytetyöraportin loppuosa.

DEVELOPING APPLICATIONS FOR THE NEEDS OF RADIOFREQUENCY IDENTIFICATION

Lehtinen, Tommi
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Automation Engineering
September 2012
Supervisor: Peltonen, Kauko
Number of pages: 44

Keywords: RFID, identification, tracking, programming

The purpose of this thesis was to in the first step to explore radio frequency identification (RFID), reader device and its interfaces. The second step was to make two applications, one for demonstrating the use of RFID and the other for helping research use. Client needed such programs, because RFID possibilities could not be easily demonstrated in practice and customized program was needed for research use.

The first step was put into practice by obtaining and reviewing of relevant literature and hardware. This formed the first part of the thesis report. In the second phase two programs that use radio frequency identification as a part of the programs features were made. The development phases of these programs form the rest of the thesis report.

SISÄLLYS

1 JOHDANTO

1.1 Opinnäytetyön tarkoitus

Opinnäytetyön tarkoituksena oli toteuttaa Satakunnan ammattikorkeakoulun tekniikka ja merenkulku Porin toimipisteeseen radiotaajuisen etätunnistustekniikan eli RFID:n käyttömahdollisuuksia ja toimintaa esittelevä ohjelmisto ja RFID:n parissa tehtävää tutkimustyötä tukeva ohjelmisto. RFID-tekniikan toimintaa esittelevää ohjelmistoa on tarkoitus käyttää erilaisissa messu- ja esittelytilaisuuksissa ja tutkimustyötä tukeva ohjelmiston on tarkoitus tulla toimipisteellä radiotaajuisen tunnistamisen parissa työskentelevien avuksi.

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli saada tuntemusta ja osaamista RFID-tekniikasta, ohjelmistoprojektin eri vaiheista ja projektin dokumentoinnista. Lisäksi tavoitteena opinnäytetyössä oli, että tehtävät ohjelmat täyttävät tilaajan vaatimukset ja niitä käytetään jatkossakin tämän opinnäytetyön valmistumisen jälkeen.

2 YLEISTÄ

2.1 RFID

”RFID (Radio Frequency Identification) on yleisnimitys radiotaajuuksilla toimiville tekniikoille, joita käytetään tuotteiden ja asioiden havainnointiin, tunnistamiseen ja yksilöintiin. Teknologian toiminta perustuu tiedon tallentamiseen RFID-tunnisteeseen ja sen langattomaan lukemiseen RFID-lukijalla radioaaltojen avulla.” (RFID Lab Finland ry 2012.) Yksinkertaisesti tuotteeseen tai asiaan kiinnitetään RFID-tunniste ja siihen kirjoitetaan halutut tiedot, kuitenkin vähintään tunnistekoodi. Sen jälkeen tiedot voidaan lukea etänä RFID-lukijan avulla. Luettuja tietoja voidaan käyttää taustajärjestelmän avulla halutulla tavalla esim. paketoidun tuotteen tunnistamiseen tai sen etenemisen seurantaan logistiikassa. RFID on hyvin samankaltaista kuin viivakoodi. Etuna viivakoodiin on, että RFID ei kuitenkaan tarvitse näköyhteyttä tunnistettavaan kohteeseen ja RFID-tunnisteen tietoja voidaan muuttaa jälkeenpäin, kun taas jo tulostettua viivakoodia ei pystytä muuttamaan. Lisäksi myös likainen RFID-tunniste pystytään usein lukemaan toisin kuin likainen viivakoodi. (RFID Lab Finland ry 2012; Bhuptani & Moradpour 2005, 36–37.)

2.1.1 Käyttökohteita teollisuudessa

Teollisuudessa RFID-teknologia on yleisesti käytössä logistiikassa, varastonhallinnassa ja tuotannossa. Tuotteita seurataan tuotannossa ja läpikäydyistä prosesseista pidetään kirjaa. Tuotteen ulkonäkö voi muuttua täysin matkalla, esimerkiksi maalauksen tai komponenttien lisäysten vuoksi. Lukemalla ja kirjoittamalla tuotteelle tehdyt muutokset RFID-tunnisteeseen prosessin eri vaiheissa pysytään ajan tasalla mitä vaiheita tuote on jo käynyt läpi ja tiedetään tuotteen tarkka koostumus. (RFID Journal 2012; Bhuptani & Moradpour 2005, 16–21.)

Varastonhallinnassa varastotilanne saadaan pidettyä ajan tasalla esimerkiksi lukemalla kaikki lähtevät ja tulevat tuotteet kuljettamalla tuotelava sisään varastoon RFID-portin ohi. Jokaista tuotetta ei tarvitse lukea erikseen, eikä paketteja avata tai purkaa lavalta, vaan lukutapahtuma on hyvin nopea ja lähes yhtäaikainen. Lisäksi lukutapah-

tuma voi laukaista taustajärjestelmässä useita eri tapahtumia, esimerkiksi tuotteiden automaattisen laskutuksen tai päivittää tuotteiden varastotilanteen verkkosivuille. (Bhuptani & Moradpour 2005, 36–37.)

Logistiikassa yleisin RFID:n käyttökohde on omaisuuden seuranta. Esimerkiksi ruokakuljetusten ja rahtikonttien seuranta helpottuu RFID:tä käytettäessä. Omaisuus löytyy nopeammin ja sen sijainnista ollaan jatkuvasti tietoisia. Air Canada käyttää RFID-teknologiaa seuratakseen ruokakärryjensä liikettä lentokentillä ja säästää aikaa ja rahaa, kun kärryjä hukkuu vähemmän ja kirjanpito helpottuu. (RFID Journal 2012.)

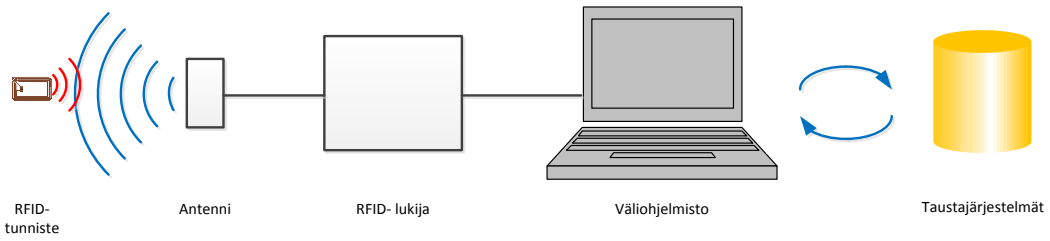
2.1.2 Käyttökohteita palveluissa

Kuluttajille RFID-teknologia näkyy kaupoissa, erilaisissa kulkukorteissa ja autoissa. Kaupoissa vaatteisiin kiinnitetyt muoviset palikat ovat RFID-tunnisteita, joilla pyritään estämään tuotteen varastaminen. Kassojen jälkeen tai ovien läheisyydessä on RFID-lukijat, jotka tunnistavat niiden ohi tai läheisyyteen viedyt RFID- tunnisteet ja antavat hälytyksen henkilökunnalle. (Bhuptani & Moradpour 2005, 7–8, 13.)

RFID-tunnisteen sisältäviä kulkukortteja ja erilaisia avaimenperiä käytetään yhä laajemmin kaupunkien julkisessa liikenteessä matka- ja pääsylippuina ja toimistorakennuksissa, parkkihalleissa ja autoissa kulunvalvontaan. RFID-tunniste voi sisältää esimerkiksi oikeudet päästä rakennuksen aulaan ja toimistoihin, mutta ei välttämättä varastoihin. (Bhuptani & Moradpour 2005, 8; Roussos 2008, 15–16.)

Autoissa RFID-teknologia on käytössä esimerkiksi auton käynnistyksenestossa. Auton avaimeen on sijoitettu RFID-tunniste ja auton ohjauspyörässä tai lukossa on RFID-lukija. Kun autoa yritetään käynnistää, on avaimesta luettavan tunnisteen oltava oikea tai auto ei käynnisty. (RFID Journal 2012.) Toinen RFID-teknologian käyttökohde autoilussa on erilaiset tietullit. Auton ikkunalle laitetaan RFID-tunniste ja tullissa on RFID-lukija. Auton ajaessa tullista portti aukeaa automaattisesti tai asiakasta laskutetaan automaattisesti. (RFID Journal 2012; Bhuptani & Moradpour 2005, 27.)

2.2 RFID järjestelmä



Kuva 1. RFID-järjestelmän osat.

RFID-järjestelmän osia ja niiden paikkaa järjestelmässä esitellään kuvassa 1. RFID-järjestelmä vaatii toimiakseen RFID-tunnisteen, eli tagin ja lukijalaitteen antenneineen tai kytkeytymiselementteineen. Näiden avulla pystytään huomaamaan ja tunnistamaan tagitettu esine tai asia. Yleensä kuitenkin lukija on vielä yhdistettynä tietokoneeseen, jolla on tarvittava äly lukijalta saatavan datan jatkokäsittelyä varten. Usein tietokone on vielä osana suurempaa tietokoneverkkoa tai kytkettynä johonkin taustajärjestelmään tiedonhakua tai tallentamista varten. (Bhuptani & Moradpour 2005, 36.)

RFID:n toimintaperiaate on, että lukija kuulustelee tageja lähettämällä signaalin kytkeytymiselementin avulla tageille, jotka saavat signaalista energiaa. Tagi vastaa heijastamalla osan tästä energiasta takaisin ja lähettää oman signaalinsa. Lukija tulkitsee signaalin ja lähettää tiedot eteenpäin tietokoneelle, jossa väliohjelmisto hoitaa omat toimenpiteensä, esimerkiksi hakee tagia vastaavat tuotetiedot tietokannasta ja näyttää ne tietokoneen ruudulla käyttäjälle. (Bhuptani & Moradpour 2005, 37–38.)

2.2.1 RFID- tunniste eli tagi

Kuva 2. RFID-tunnisteita ja kotelointeja.

RFID-tunnisteita eli tageja löytyy hyvin monenlaisia. Muutamia eri tyyppejä on esillä kuvassa 2. Tagi sisältää lukijalle lähetettävän datan lukijan kuulustellessa tagia. Tagin sisältämän datan määrä riippuu tagin sisältämän mikrosirun tyypistä ja muistin määrästä. Tagi kuitenkin yleensä sisältää vähintään tagin ID- numeron tai EPC- koodin. (Bhuptani & Moradpour 2005, 39, 41.)

RFID-järjestelmät ja -tunnisteet voidaan kategorisoida monella tavalla. Kaksi tyypillisintä tapaa on jakaa tagit niiden toimintataajuuden mukaan tai tunnisteen mikrosirun energian saannin perusteella. Yleisimpiä toimintataajuuksia ovat low frequency (LF), high frequency (HF), ultra high frequency (UHF) sekä mikroaaltotaajuiset järjestelmät. Toimintataajuudet järjestelmillä ovat LF 135 kHz tai alle, HF 13,56 MHz, UHF:llä tyypillisesti 860–960 MHz ja mikroaaltotaajuisilla 2,45 GHz. (Bhuptani & Moradpour 2005, 44; Roussos 2008, 45–48.) Tässä työssä keskitytään UHF-taajuuksiin.

Tageja on kolmea eri tyyppiä: passiivisia, puoli-passiivisia ja aktiivisia. Passiivinen tagi saa kaiken kommunikointiin vaadittavan energiansa lukijan lähettämästä signaalista ja ei siksi tarvitse omaa akkua. Passiiviset tagit ovat yleisimpiä ja niitä voidaan valmistaa hyvin halvalla. Ne koostuvat yleensä vain antennista ja mikrosirusta. Tyypillisiä käyttökohteita passiiviselle tagille ovat kulunvalvonta ja omaisuuden valvonta. (Bhuptani & Moradpour 2005, 39–40; Roussos 2008, 5.)

Puoli-passiiviset tagit sisältävät antennin ja mikrosirun lisäksi akun, jota käytetään antamaan energiaa mikrosirulle yksinkertaisien tehtävien suorittamiseen. Puoli-passiivinen tagi käyttää kuitenkin lukijalta tulevaa energiaa aktivoituakseen ja datan lähettämiseen. Tagi käyttää akkua vain kun se on aktiivisena ja tämän vuoksi puoli-passiiviset tagit kestävät yleensä vuosia. Tyypillinen käyttökohde puoli-passiiviselle tagille on elektroninen tullin kerääminen. (Bhuptani & Moradpour 2005, 40; Roussos 2008, 5.)

Aktiiviset tagit sisältävät oman akun, josta tagi saa energiansa. Aktiivinen tagi ei tarvitse lukijan lähettämän signaalin energiaa toimiakseen. Aktiivisella tagilla lukuetäisyys on pidempi, tagilla voidaan suorittaa monimutkaisemman tiedon vaihtamista lukijan kanssa ja sitä käytettäessä saavutetaan parempi tarkkuus. Huonoja puolia aktiivisissa tageissa on niiden elinikä, joka on riippuvainen akun varauksesta, kallis hinta ja koko. Aktiivista tagia käytetään esimerkiksi armeijan kalliiden varusteiden jäljittämiseen ja tarkkailuun. (Bhuptani & Moradpour 2005, 40; Roussos 2008, 5.)

Tageja koteloidaan hyvin monella eri tapaa riippuen käyttökohteesta. Kotelointi on hyvin tärkeä osa luodessa toimivaa RFID-järjestelmää. Tageja löytyy esimerkiksi tarroina, painettuna eri materiaaleille kuten muoville, muovisissa koteloinneissa kuten avaimenperissä ja muovikorteissa, sekä lasikoteloituina eläimen tai ihmisen ihon alle sijoittamista varten. (Bhuptani & Moradpour 2005, 40–41.)

2.2.2 EPC

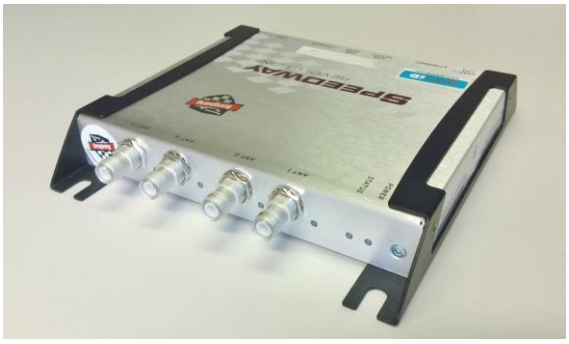
Electronic Product Code (EPC), eli sähköinen tuotekoodi on merkintärakenne, jolla esineille ja asioille voidaan antaa yksilöllinen tunniste. Sitä voi verrata viivakoodin sisältämään numerosarjaan. EPC koostuu alkutunnisteesta, hallinnointinumerosta, objektiluokasta ja sarjanumerosta. Alkutunniste kertoo EPC:n version, hallinnointinumero määrittää koodia käyttävän yhtiön, objektiluokka kertoo tuoteluokan ja sarjanumero määrittelee yksilöllisen ilmentymän objektiluokassaan. (Bhuptani & Moradpour 2005, 45.)

2.2.3 Antenni

Kuva 3. Antenneja.

RFID-lukijan antenni toimii kanavana tagin ja lukijan väliselle kommunikoinnille lähettäen lukijan tehoa tageille ja ottamalla tagin lähettämän vastauksen vastaan. Kuten tageja, myös antenneja löytyy lukuisia erilaisia (Kuva 3) käyttökohteen ja tarkoituksen mukaan. Lineaarisella antennilla saavutetaan pidempi lukuetäisyys kuin pyöreän lukualueen omaavalla antennilla, mutta se ei tunnista yhtä hyvin tageja, jos niiden asennot ovat vaihtelevia. (Bhuptani & Moradpour 2005, 47–48.) Antenneja voi olla lukijaan yhdistettynä yksi tai useampia kuten RFID- portissa.

2.2.4 Lukija



Kuva 4. RFID-lukija.

RFID-lukijan (Kuva 4) tehtävänä on lukea tagiin tallennettu data ja kommunikoida tietokoneen ja väliohjelmiston kanssa. Tätä tehtävää varten lukija sisältää komponentit energian tuottamiseksi antenneille, tietokoneen kanssa kommunikointiin ja tietojen käsittelyyn lukijan sisällä. Mitä älykkäämpi lukija on, sitä enemmän se sisältää eri osia. Älykkäät lukijat pystyvät toimimaan useilla eri radiotaajuuksilla, prosessoimaan

samanaikaisesti ja tehokkaasti suuria tagimääriä, tukevat monia eri liityntätapoja ja osaavat antaa tietoa omasta tilastaan ja toiminnastaan. (Roussos 2008, 37–39.)

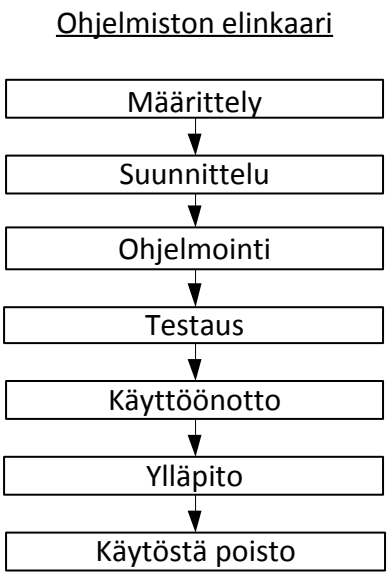
2.2.5 Väliohjelmisto ja taustajärjestelmä

Väliohjelmana toimii yleensä tietokoneelle asennettu ohjelma, joka hallinnoi lukijoita ja niiltä saatua tietoa. Väliohjelma suorittaa lukijalta saadulle datalle ennalta määrätyt toimenpiteet ja antaa siitä informaatiota käyttäjälleen. Ohjelma voi olla yrityksessä jo olemassa oleva, kuten varaston hallintaan tai inventaarioon tarkoitettu sovellus. Tällainen sovellus yhdistetään usein vielä taustajärjestelmään, joka voi olla esimerkiksi yrityksen tuotetietokanta. (Bhuptani & Moradpour 2005, 53–54.)

Esimerkiksi varaston hallinnassa ohjelma voi antaa lukijalle käskyn tagien kuulustelun aloittamiseksi uusien tuotteiden saapuessa. Lukija hoitaa tuotteiden tagien lukemisen ja palauttaa väliohjelmalle niiden tiedot. Väliohjelma hakee tietoja vastaavat tuotteet tietokannasta, merkkaa ne saapuneiksi, päivittää varastotilanteen ja näyttää käyttöliittymässä päivitetyn tilanteen.

2.3 Ohjelmistotuotanto

Ohjelmistotuotannon voi vapaasti tulkita tarkoittavan ohjelmistotyötä, jonka seurauksena syntyvät järjestelmät täyttävät niille annetut kohtuulliset vaatimukset (Haikala & Märijärvi 2006, 16). Ohjelmistotuotanto voidaan jakaa edelleen pienempiin osa-alueisiin, jotka ovat laatujärjestelmä, hankkeiden hallinta ja projektin hallinta. Näistä tärkeimpänä osana omassa työssäni oli projektin hallinta ja sen osa-alueet, ohjelmiston kehitysprosessi ja sen tukitoiminnot laadunvarmistus sekä dokumentointi. (Haikala & Märijärvi 2006, 16.)



Kuva 5. Ohjelmiston elinkaari.

Kehitysprosessiin voidaan erottaa seuraavat vaiheet (Kuva 5): määrittely, suunnittelu, ohjelmointi, testaus, käyttöönotto ja ylläpito. Tämä kuvastaa samalla ohjelman elinkaarta. Tapoja, joilla elinkaari jaetaan vaiheisiin, on useita ja niitä sanotaan vaihejakomalleiksi. Vaihejakomalleja ovat esimerkiksi vesiputousmalli, suihkulähdemalli ja prototyyppimalli. Kaikilla näillä on omat erityispiirteensä kehitysprosessin etenemisen kannalta. (Haikala & Märijärvi 2006, 35–37.)

3 SUUNNITTELU

3.1 Aikataulu ja resurssit

Käytettävänäni työssä oli ammattikorkeakoululta valmiiksi löytyvät laitteet ja ohjelmistot, joista sain valita käytettävän kokoonpanon itse työhön sopivaksi. Opinnäytetyön käytännön osuuden valmiiksi saamiseen tarvittavaa aikaa oli hyvin vaikea arvioida etukäteen työn luonteen vuoksi. Alustavasti aikatauluksi tehtävien ohjelmistojen valmistumiseen varattiin neljä kuukautta.

3.2 Käytettävä RFID laitteisto

Yhtenä opinnäytetyön osana oli tutustua valmiiksi hankitun RFID-laitteiston osiin. Lukijalaitteita löytyy kaksi, jotka molemmat ovat malliltaan Impinj SpeedWay R420. Lukijasta löytyy liitännät neljälle antennille ja useita eri liitäntämahdollisuuksia ulkoiseen järjestelmään.

Antenneja koululla on kolmea eri mallia. Suurimmat erot antennien välillä löytyvät niiden koosta ja kaapeloinnista. Teknisesti antennit ovat hyvin samankaltaisia suorituskyvyiltään. Kahdessa antennimallissa antennikaapeli kytketään antennin taakse erillisellä antennikaapelilla ja yhdessä mallissa antenninkaapeli on valmiiksi kytkettynä antennin sivulle. Koska suuria eroja ei antennien välillä ole, päädyin valitsemaan mallin, missä antennikaapeli tulee antennin sivulta. Tämä helpottaa antennin siirtelyä paikasta toiseen ja antennin kiinnittämistä eri paikkoihin sen vaatiessa vähemmän tilaa antennin taakse. Ohjelmointia varten sain käyttööni kannettavan tietokoneen, jonka suorituskyvyn pääteltiin riittävän hyvin tarpeisiini. Lisäksi sain käyttööni tilaa koulun palvelimelta tietokantoja varten.

3.3 Ohjelmointikieli

Aikaisempaa ohjelmointikokemusta minulla oli erilaisien automaatiossa käytettävien logiikoiden ja robottien ohjelmoimisesta sekä PHP-, HTML- ja Delphi-

ohjelmoinnista. Tältä pohjalta ohjelmointikielellä ei ollut minulle merkitystä ja olin valmis opettelemaan uuden ohjelmointikielen.

Lukijalaitteen valmistajalta oli saatavilla aloitusopas ohjelmoinnin aloittamisen helpottamiseksi. Oppaassa ohjelmointikielenä oli käytössä C# (eng. C sharp). Visual C# on oliopohjainen ohjelmointikieli, joka perustuu C++-kieleen, sisältää Java-kielen piirteitä ja yhdistää C++:n laskentatehokkuuden ja Visual Basicin helppokäyttöisyyden (Moghadampour 2009, 14). Helpottaakseni omaa tulevaa ohjelmointia valitsin C#-kielen käytettäväksi ohjelmieni koodaamisessa.

3.4 Ohjelmointiympäristö

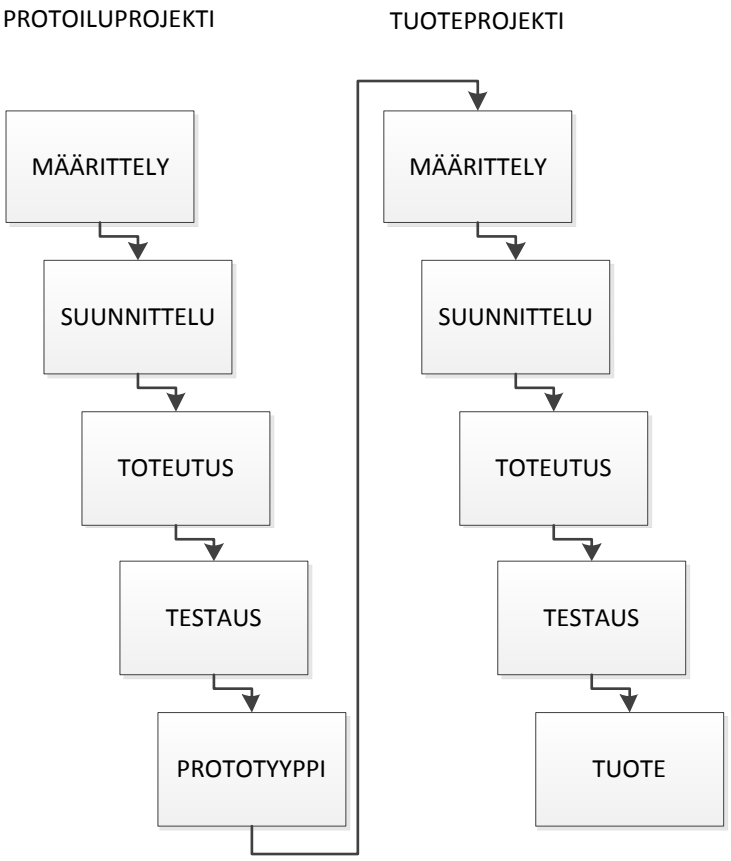
Saamassani aloitusoppaassa ohjelmointiympäristönä oli käytössä Microsoft Visual Studio 2010. Visual Studio tukee useita ohjelmointikieliä kuten Visual C++, C#, Visual Basic.NET ja Visual J#. Päädyin käyttämään samaa ohjelmointiympäristöä, koska ohjelmasta oli saatavilla ilmainen versio opiskelijoille, mikä mahdollisti ohjelmoinnin kotona omalla koneella. Ammattikorkeakoululla oli valmiina lisenssi ohjelman asentamiseksi ja käyttämiseksi laboratoriossa olevassa koneessa.

3.5 Liitynnät

Lukijalaitteesta on RJ45-, USB- ja GPIO-liitynnät. Näistä päätin käyttää RJ45-liitäntää, koska tällöin lukijaan voidaan yhdistää verkon yli tai kytkemällä lukija Ethernet-kaapelilla suoraan tietokoneeseen.

3.6 Ohjelmakehityksen vaihejakomalli

Työn alussa ei vielä ollut täysin selvää kuvaa millainen valmis ohjelma tulisi olemaan. Tutkittuani eri ohjelmistotuotannon vaihejakomalleja, päädyin valitsemaan käyttööni prototyyppimallin sen sopiessa hyvin projektini luonteeseen.



Kuva 6. Esimerkki prototyyppimallista (Haikala & Märijärvi 2006, 42.)

Prototyyppimallilla (Kuva 6) tarkoitetaan työskentelymallia, jossa tuotteen jotain ominaisuutta tai piirrettä kokeillaan ennen lopullisen tuotteen rakentamista. Mallissa edetään syklisesti luoden uusi prototyyppi, jonka asiakas tarkistaa. Näin varmistetaan, että järjestelmä sisältää kaikki asiakkaan tarvitsemat toiminnot ja että käyttöliittymä on mieluinen. Prototyyppimallin etuna on myös, että muutokset järjestelmään voidaan tehdä helpommin ja nopeammin verrattuna koko järjestelmän uudelleen ohjelmointiin. (Haikala & Märijärvi 2006, 42–44; Arnowitz, Arent & Berger 2007, 3–4.)

3.7 Rajaus ja riskit

Opinnäytetyötä varten saamani työnkuvaus oli jätetty tarkoituksella hyvin avoimeksi. Näin pääsin itse vaikuttamaan mahdollisimman paljon työn toteutustapaan ja sain kokemusta laajan projektin toteuttamisesta. Työn rajaus oli tehtävä hyvin tiukaksi

varsinkin tehtävien ohjelmien osalta, muuten projektista olisi helposti tullut liian laaja ja sen loppuun saattaminen venyttänyt aikataulua.

Suurimpana riskinä ohjelmistotuotannossa ovat aikataulujen ja työmäärien väärä arviointi. Syynä tähän on ohjelmistojen luonteeseen liittyvän toteuttamistyön arvioinnin vaikeus. (Haikala & Märijärvi 2006, 25.)

Tässä vaiheessa riskinä omassa työssäni oli aikataulun mahdollinen väärä arviointi ja rajausten jääminen hyvin yleispiirteisiksi, mutta oli mahdotonta tehdä tarkempaa rajausta ennen tutustumista käytettävään RFID-laitteistoon ja ohjelmointiympäristöön. Rajauksia pystytään tarkentamaan ohjelmaa suunniteltaessa ja määriteltäessä.

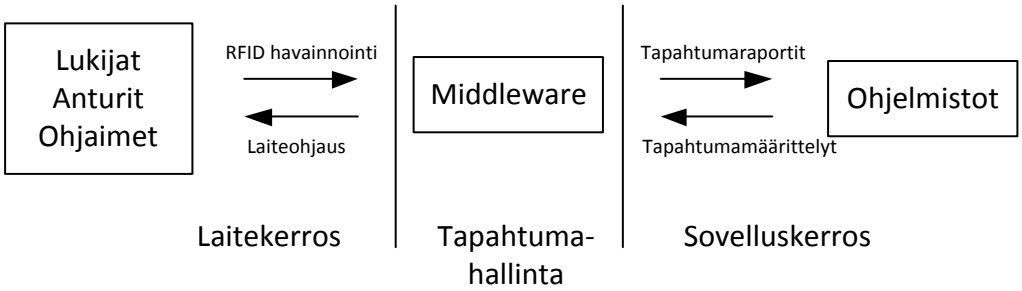
4 TYÖN ALOITUS

Opinnäytetyötä aloittaessani oli RFID-laitteisto jo valmiiksi hankittuna. Aluksi RFID oli minulle lähes tuntematon käsite. Aloitin työn tekemisen tutustumalla RFID-laitteistoon ja sen toimintaan valmistajan toimittaman ohjelman avulla. Lisäksi aloin etsimään kirjallisuutta ja tietoa radiotaajuisesta tunnistamisesta. Seuraavana vaiheena tutustuin ohjelmointikieleen ja ohjelmointiympäristöön tarkemmin erilaisien C#-ohjelmointiharjoitusten avulla. Tämän jälkeen lähdin suunnittelemaan ja toteuttamaan opinnäytetyöni osana olevia ohjelmia.

4.1 Laitteistoon ja RFID tekniikkaan tutustuminen

Aloitin koulun RFID-laitteistoon tutustumisen valmistajan toimittaman ohjelman avulla. Se oli visuaalisesti hyvin yksinkertaisen näköinen, mutta sisälsi silti paljon toiminnallisuuksia. Ohjelma oli tarkoitettu esittelemään RFID-lukijan eri ominaisuuksia. Kokeilin erilaisia laitteistokokoonpanoja ja asetuksia tehden samalla muistiinpanoja erilaisien lukijan asetusten vaikutuksista lukutapahtumaan. Tämä helpotti työtäni myöhemmässä vaiheessa. Ymmärtääkseni radiotaajuista etätunnistusteknologiaa syvemmin hain samalla tietoa kirjoista ja internetistä.

4.2 Middleware



Kuva 7. RFID middleware ja sen rooli.

RFID middleware toimii välikerroksena laitteiden ja sovelluskerroksen välillä (Kuva 7). Sen tehtävänä on suodattaa, järjestellä ja ottaa selvää lukijan saamista tagien tiedoista, jotta niitä voidaan käyttää ohjelmistoissa hyväksi. Middleware voi, esimer-

kiksi suodattaa tagien tiedot, jotka vastaavat yksittäistä tuotetta, kun halutaan vain kuormalavojen tiedot. (Roussos 2008, 99–101.)

Middleware välittää myös ohjelmistoilta tulevat käskyt lukijoille. Tällaisia käskyjä ovat, esimerkiksi tagien lukemisen aloittaminen ja lukijan asetusten muuttaminen. (Roussos 2008, 100–101.)

4.3 Ohjelmointikieleen ja –ympäristöön tutustuminen

Microsoft Visual Studio ja C#-kieli olivat minulle tässä vaiheessa vielä hyvin uutta, joten aloitin niihin tutustumisen käytännön ohjelmointiharjoituksilla. Tätä varten päädyin lainaamaan C#-ohjelmointikirjan, joka sisälsi ohjelmointiharjoituksia perusteista lähtien ja ohjelmointiympäristönä toimi myös Microsoft Visual Studio. Visual Studion ja C#-kielen käytön sisäisti nopeasti aikaisemman ohjelmointi kokemuksen ja harjoitusten avulla.

Seuraavaksi siirryin tutustumaan valmistajan toimittamaan lukijalaitteen ohjelmoinnin aloitusoppaaseen. Siinä oli eriteltynä erillinen esimerkki jokaiselle lukijan päätoiminnolle. Kävin esimerkit läpi kokeilemalla samalla erilaisia muutoksin koodissa ja niiden vaikutusta lukijan toimintaan. Esimerkit läpikäytyäni päätin alkaa suunnittelemaan ensimmäistä ohjelmaani. Tässä vaiheessa olin jo saanut hyvän kuvan työn mahdollisuuksista ja rajoitteista.

5 WORKERS- OHJELMA

5.1 Määrittely ja rajaus

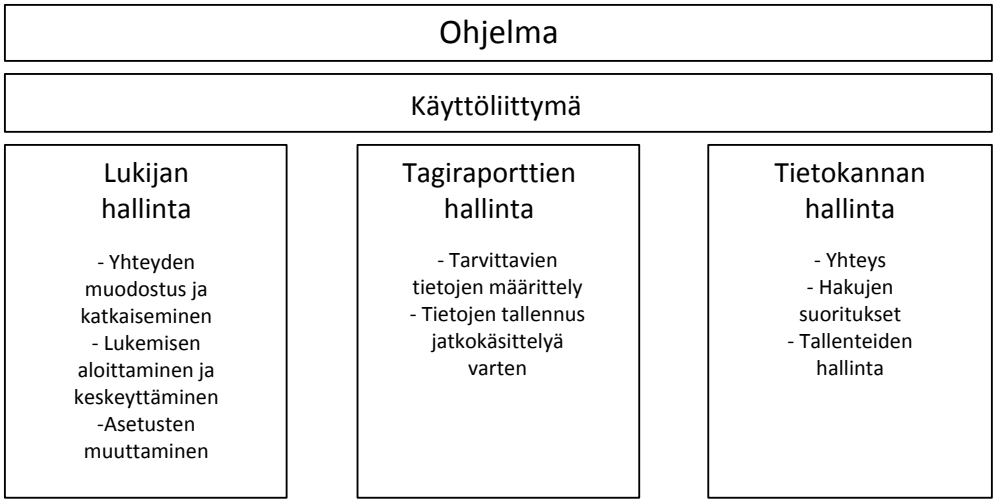
Ensimmäisen ohjelman osalta ohjeistuksenani oli tehdä ohjelmisto ja tietokanta, joiden avulla tiettyyn tunnisteeseen olisi mahdollista yhdistää lisätietoja kuten, esimerkiksi nimi ja kasvokuva. Tunnistekoodia luettaessa nämä kyseiset tiedot ilmestyisivät tietokoneen ruudulle. Ohjelmaa pystyy tällöin käyttämään esimerkiksi kulunvalvonnan esittelyyn RFID:tä käytettäessä.

Ohjelmaa tarkemmin suunnitellessani päädyin rajaamaan ohjelman toiminnallisuuden siten, että sillä pystyy luomaan yhteyden lukijalaitteeseen, lukemaan tageja, hallinnoimaan tietokannan tallenteita ja näyttämään tageihin liitettyjen tallenteiden tiedot ruudulla. Lisäksi ohjelman käyttöliittymän pitää olla tarpeeksi helppo käyttää ja selkeä asettelultaan, jotta sen käyttämisen opetteluun ei mene liikaa aikaa. Käyttöliittymän sisältämät osat selviäisivät tarkemmin prototyyppejä kokeiltaessa ja käyttäjien antaman palautteen avulla.

5.2 Ohjelman suunnittelu

Ensimmäistä ohjelmaa aloittaessani oli idea ohjelman ulkoasusta ja sisällöstä selvä. Olin päättänyt tehdä työntekijöiden kulunvalvontaa esittelevän ohjelman, joka näyttäisi työntekijän tiedot kuvaruudulla suorakaiteen muotoisessa ikkunassa. Päätin näytettäviksi tiedoiksi henkilön nimen, työnkuvan, tervehdystekstin ja valokuvan. Käyttöliittymän suunnittelussa päädyin alussa käyttämään mallina suomalaista ajokorttia. Päädyin kuitenkin muuttamaan käyttöliittymän asettelua myöhemmässä vaiheessa toimivammaksi.

Minulla oli jo hyvä käsitys tarvittavista ohjelmakoodista tekemieni harjoituksien ja esimerkkien pohjalta. Päädyin jakamaan ohjelman koodauksen kuvassa 8 näkyvällä tavalla.



Kuva 8. Ohjelman osat.

Jaoin ohjelman neljään selkeään eri osaan: käyttöliittymä, lukijan hallinta, tagiraporttien hallinta ja tietokannan hallinta. Käyttöliittymä kulki koko ajan ohjelmakoodin osien mukana, koska sen avulla hallitaan ohjelman toimintaa ja näytetään informaatiota käyttäjälle. Ilman käyttöliittymää ohjelmaa ei voida käyttää.

Lukijan hallinnan jaoin vielä pienempiin osiin: yhteyden muodostus sekä katkaiseminen, lukemisen aloittaminen sekä keskeyttäminen ja asetusten muuttaminen. Näistä muodostui selkeyttäviä osia ohjelmaa tehdessä. Tagiraporttien hallinnan alle muodostui kaksi osaa: tagin lukutapahtuman käsittely ohjelmassa ja tietojen tallennus jatkokäsittelyä varten.

Tietokannan hallinnan jaoin kolmeen eri osaan: yhteydet, hakujen suorittaminen ja tallenteiden hallinta. Tämän ohjelmarakennekuvan avulla pystyin aloittamaan ohjelman koodaamisen ja tarkemman suunnittelun sekä miettimään tietokannan rakennetta.

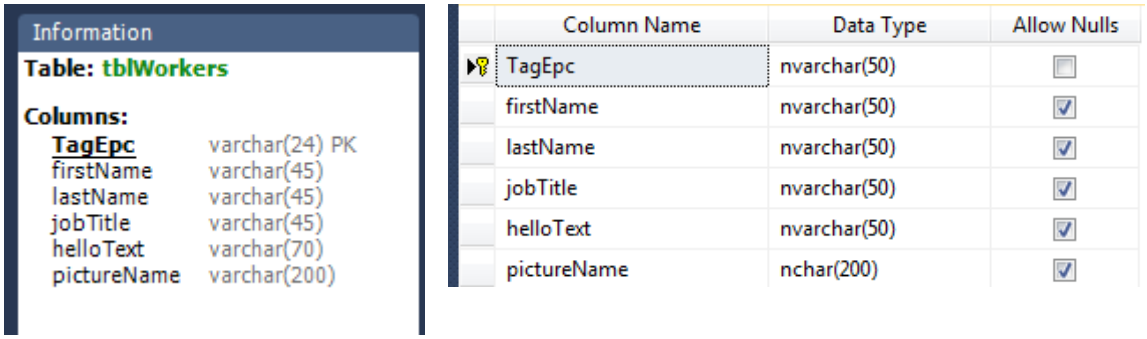
5.3 Tietokanta

Tietokannan toteuttamiseen ohjelmalle minulla oli kaksi vaihtoehtoa. Ensimmäinen oli tehdä tietokanta paikallisesti koneelle ohjelman sisälle ja toinen oli tietokannan sijoittaminen ulkoiselle palvelimelle. Aluksi päätin, että teen ohjelman käyttämään

ainoastaan palvelimelle tulevaa tietokantaa. Kuitenkin myöhemmin selvisi, että saan yhteyden palvelimeen ainoastaan ammattikorkeakoulun lähiverkossa. Tämän vuoksi päädyin tekemään ohjelmasta myös version, jossa tietokanta on integroituna ohjelmaan.

Palvelimella tietokanta on toteutettu MySQL (wiki rajapinta) relaatiotietokantaohjelmistolla. MySQL sisältää suoraan rajapinnan C#:lle ja siksi valitsin sen käytettäväksi omassa työssäni. Ohjelman sisäisen tietokannan päätin tehdä suoraan Visual Studion omalla tietokantatyökalulla, joka käyttää Microsoft Server Express tietokantaa. Työkalun käyttäminen ja tietokannan integroiminen projektiin oli tehty helpoksi.

Molemmissa toteutustavoissa loin tietokannan nimeltä MyWorkers ja sen alle taulun tblWorkers. Siihen lisättiin kentät tarvittaville tiedoille (Kuva 9).



Kuva 9. Tietokantojen rakenne.

- TagEpc: tagin EPC-koodi, taulukon pääavain (Primary Key).
- firstName: Etunimi.
- lastName: Sukunimi.
- jobTitle: Työnkuva.
- helloText: Tervehdysteksti. Myöhemmin muutettu company kentäksi: Yritys.
- pictureName: Kuvan sijainti ja nimi.

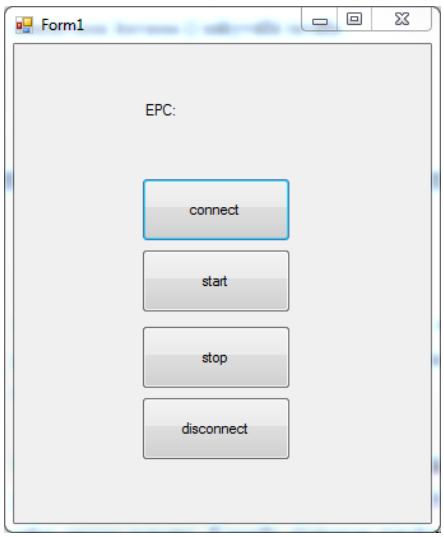
Täytin tauluihin valmiiksi yhden tallenteen tiedot ohjelman toiminnan kokeilemista varten (Kuva 10).

	TagEpc	firstName	lastName	jobTitle	helloText	pictureName
	546F5053686F700000001A09	Tommi	Lehtinen	Projektityöntekijä	Huomenta!	C:\Users\TKI-TOMMI...
*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

Kuva 10. Tallenne lisättynä tietokantaan.

5.4 Ensimmäinen prototyyppi

Koodatessani ensimmäistä prototyyppiä ohjelmasta päätin pitää homman yksinkertaisena. Lähdin tekemään ohjelmaa, joka pystyisi ainoastaan ottamaan yhteyden lukijaan, lukemaan tageja ja näyttämään luetun tagin EPC-koodin.



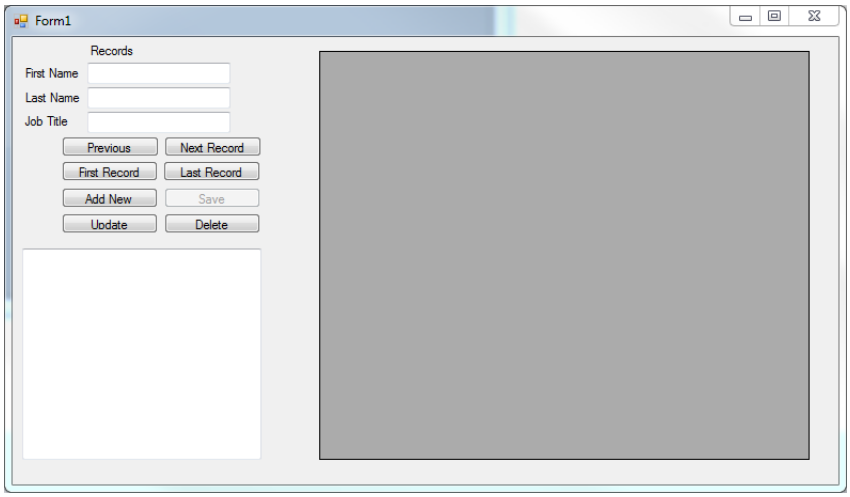
Kuva 11. Ensimmäisen prototyypin käyttöliittymä.

Käyttöliittymä (Kuva 11) ei sisältänyt muuta, kuin painikkeet lukijan yhdistämiseen, lukemisen aloittamiseen, pysäyttämiseen, yhteyden katkaisuun ja kentän näyttämään luetun tagin EPC:n. Ohjelmakoodia oli vielä suhteellisen vähän ja se oli kaikki sijoitettuna yhteen luokkaan.

Ensimmäinen prototyyppi valmistui nopeasti. Testausvaihe osoitti prototyyppiin tehdyt ominaisuudet toimiviksi. Niitä voitaisiin käyttää ohjelman myöhemmissä vaiheissa.

5.5 Toinen Prototyyppi

Toisella prototyypillä halusin kokeilla tietokannan hallintaa ohjelman kautta. Tätä varten tein käyttöliittymän (Kuva 12), johon lisäsin kentät näyttämään tallenteen tiedot ja painikkeet tietokannan hallinnoimiseksi.



Kuva 12. Toisen prototyypin käyttöliittymä.

Käyttöliittymästä oli nähtävillä tallenteiden määrä ja monesko ruudulla näkyvä tallenne on tietokannassa. Etunimelle, sukunimelle ja työnimikkeelle oli syöttökentät, jotka toimivat samalla tallenteiden näyttämiseen. Painikkeet löytyivät tallenteiden välillä navigoimiseen, uuden tallenteen lisäämiseen ja poistamiseen. Vasemmassa alareunassa oli kenttä, johon ilmestyi riveittäin tietokantaan tehdyt muutokset, kuten tallenteen lisääminen ja tietokantaan yhdistäminen. Oikeassa reunassa oli paikka kuvalle, joka liitetään tallenteeseen.

Tämä Prototyyppi vaati hieman enemmän aikaa valmistuakseen kuin ensimmäinen. Tietokantaan yhdistämiseen vaadittava koodi vaati paljon tutustumista sen ymmärtämiseksi ja soveltamiseksi. Yksinkertaistettuna tallenteiden saamiseksi tietokannasta ohjelmassa avataan yhteys tietokantaan, avataan se, haetaan tietyn taulun tallenteet ohjelman muistiin ja suljetaan yhteys. Aina, kun tietokantaan halutaan tehdä muutoksia, muutosten siirtymiseksi tarvitsee yhteys tietokantaan avata.

Koodin valmistuessa prototyyppi toimi muuten hyvin, mutta kuvan näyttäminen ruudulla jäi kesken. Tässä vaiheessa ei ollut vielä varmaa lisättäisiinkö kuvat suoraan tietokantaan, vai vain kuvan tiedostopolku ja kuville tehtäisiin oma kansionsa ohjelmaan.

5.6 Kolmas prototyyppi

Kahden ensimmäisen prototyypin jälkeen minulla oli tulevan ohjelman tärkeimmät toiminnot tehtynä. Seuraavaksi päätin lähteä yhdistämään niitä samaan prototyyppiin ja samalla hioa käyttöliittymää lopullisempaan suuntaan. Lisäksi ajattelin lisätä ohjelmaan lukijan asetuksien muuttamista varten oman osansa.

Kuva 13. Kolmannen prototyypin käyttöliittymä.

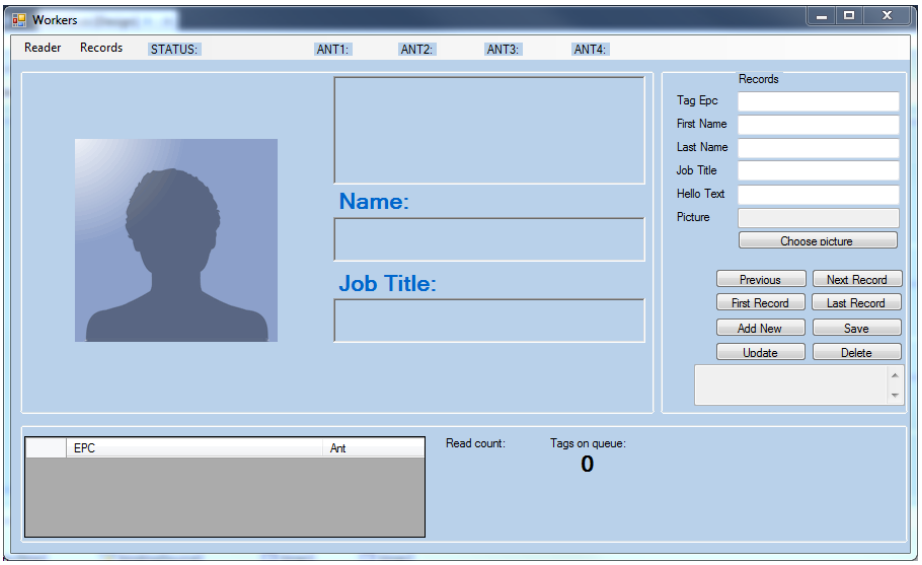
Uuden prototyypin käyttöliittymä (Kuva 13) sai päättämäni henkilökortin ilmeen ja väreiksi valitsin sinisen eri sävyt. Lisäksi ohjelma sai yläreunaan oman valikkopalkkinsa. Valikoista löytyi lukijaa ja tietokantaa varten omat osansa. Lukijavalikon alta löytyi lukijan hallintaan tarvittavat toiminnot ja tietokantavalikon alta saatiin tietokannan hallinta näkyviin ja piilotettua. Yläreunassa näkyi nyt myös lukijan yhteyden tila: Connected, Singulating ja Disconnected. Lukijaan antennit näkyivät OK-tekstinä kytketyn antennin kohdalla. Edelliseen tietokantaprototyyppiin lisäyksenä tein myös kentän tervehdystekstille nimi- ja työnimikekentän yläpuolelle.

Prototyyppiin lisätystä lukijavalikosta sisältä löytyi asetusvalikko. Asetusvalikosta päästiin antamaan yhdistyksen kohteena olevan lukijan IP-osoite ja lukijan lähetysteho päästiin syöttämään antennikohtaisesti valmistajan suositteleman 10–30 dBm välillä.

Prototyyppi sisälsi huomattavasti paljon enemmän koodia kuin aikaisemmat prototyypit. Työmääräkin oli huomattavasti suurempi verrattuna aikaisemmin toteuttamiini ohjelmointiprojekteihin. Koodi oli vieläkin kaikki yhdessä luokassa ja kommentointi oli vielä vajavaista. Seuraavan prototyypin yksi osa oli siistiä ja jaotella koodia selkeämmäksi alustavien suunnitelmieni mukaan.

Testauksessa kokeiltiin lukijan asetusten muuttamista ja koodia, joka vertasi luetun tagin EPC:tä tietokannassa oleviin ja toi tiedot näkyviin näytölle. Ongelmaksi osoittautui tilanne, missä lukijan kantaman alueella oli useampi tagi. Tällaisessa tilanteessa, kun tageja luetaan hyvin nopeaan tahtiin, ohjelma ehti näyttämään yhden tallenteen tietoja vain pienen hetken. Näytöllä ehti näkyä vain vilaus yhden tallenteen tiedoista ennen kuin ne vaihtuivat uusiin. Lisäsin seuraavaan prototyyppiin tehtäviin muutoksiin tämän ongelman ratkaisemisen. Muuten prototyyppi oli onnistunut ja askel ohjelmiston kehityksessä eteenpäin.

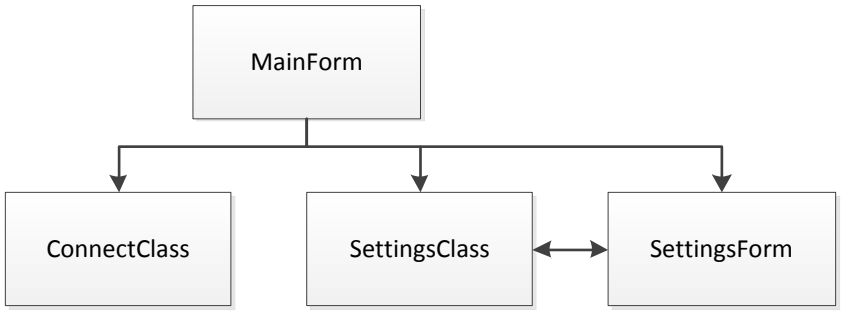
5.7 Neljäs protyyppi



Kuva 14. Neljännen protoyypin käyttöliittymä.

Neljännessä prototyypissä kokeiltiin edellisen prototyypin pohjalta tehtyjä parannuksia. Prototyypin käyttöliittymään (Kuva 14) tuli uutena alareunaan tila taulukolle, joka näyttää luettujen tagien jonon. Lisäksi käyttäjälle näytettiin informaatiota jonon pituudesta ja luettujen tagien määrästä.

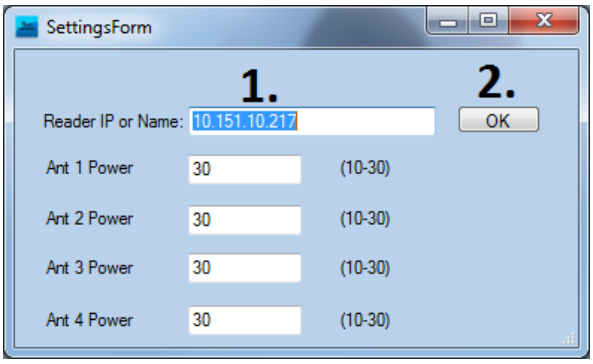
Ongelman tallenteiden tietojen nopeasta vaihtumisesta ratkaisin tekemällä erillisen taulukon, johon luetut tagit lisättiin lukemisjärjestyksessä. Samalla ohjelman koodiin tehtiin lisäys, joka tarkistaa löytyykö tagi jonosta. Tagin ollessa jonossa sitä ei huomioida, muuten se lisätään jonon perään. Samalla varmistettiin, ettei yhden tagin tiedot pääse valtaamaan koko jonoa. Jokaista tietoa näytetään 5 sekuntia. Tämän ajan todettiin riittävän henkilön tietojen lukemiseen riittäväksi. Ajan kuluttua loppuun, ohjelma poistaa näytetyn tiedon jonotaulusta ja siirtyy näyttämään seuraavaa.
Ohjelmakoodin jäsentelin ja kommentoin uusiksi ohjelmaa selkiyttääkseni. Osa koodista sai omat luokkansa ja eri toimintojen suorituskykyä paranneltiin.



Kuva 15. Ohjelmakoodin jako.

Jaoin tekemäni koodin neljään osaan (Kuva 15). Vähempikin olisi riittänyt näin pienessä ohjelmassa, mutta päädyin tähän ratkaisuun, koska se selkiytti mielestäni koodia. MainForm sisältää ohjelman käyttöliittymän toiminnallisuuksien, tietokantaan yhdistämisen, tagien tarkastukset ja jonotuksen hallinnan. ConnectClass koostuu kaikista lukijan operoimiseksi tarvittavista osista, kuten yhdistäminen, lukeminen ja asetusten muuttaminen. SettingsClass toimii käyttäjän tekemien asetusten väliaikaisena säilytyspaikkana ja SettingsForm sisältää asetusikkunan toiminnallisuuksien koodin.

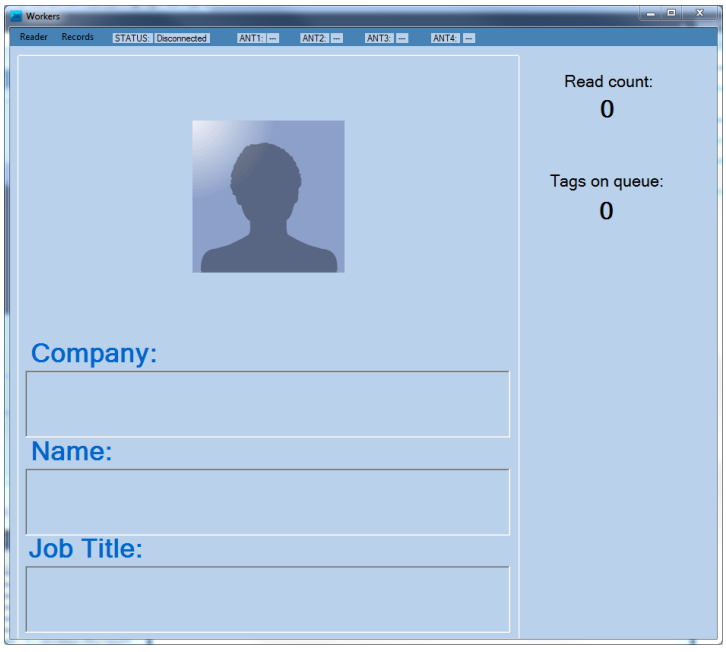
Tässä prototyypissä lisäsin asetuksille oman ohjelmaikkunansa (Kuva 16), jonka saa auki lukijavalikon alta. Tässä vaiheessa en ollut vielä lisännyt erillistä painiketta asetusten muuttamisen peruuttamiseen, vaan se tehtiin sulkemalla asetusikkuna.



Kuva 16. Asetusikkuna.

Prototyyppiä testattaessa ei toiminnallisuudesta löytynyt uusia parannuksen kohteita. Kuitenkin ohjelman käyttöliittymäikkunan todettiin olevan liian pieni. Sitä päätettiin suurentaa, jotta kuva ja tiedot olisivat nähtävissä kauempaa. Lisäksi tervehdystekstin päätin muuttaa kertomaan yrityksen nimen, koska se sopi paremmin ohjelman teemaan. Nämä parannukset olivat seuraavassa vaiheessa pääasiana.

5.8 Valmis ohjelma

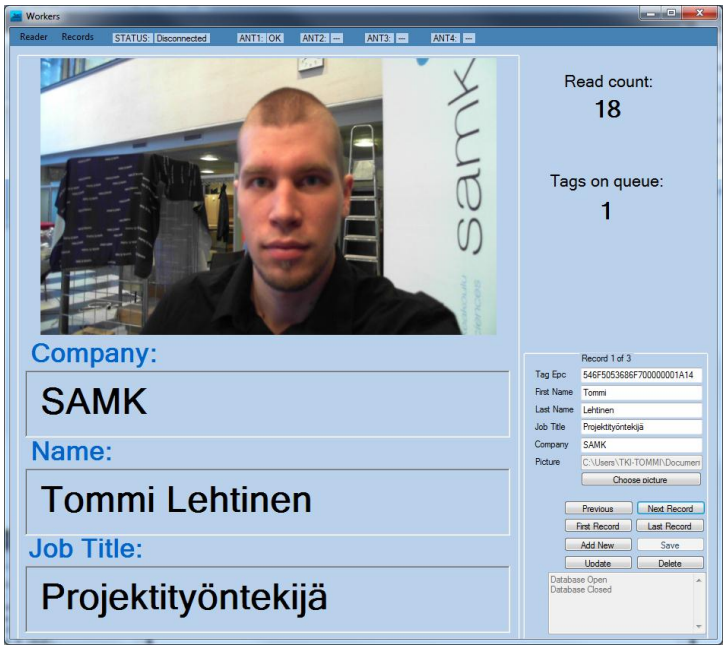


Kuva 17. Valmis ohjelma.

Valmiissa ohjelmassa (Kuva 17) kuvan paikka siirtyi tiedoissa päällimmäiseksi ja sen ala kasvoi huomattavasti. Ohjelmaikkunan kokoa ja tietojen näyttämisessä käy-

tettävän fontin kokoa suurennettiin. Tervehdysteksi muutettiin kertomaan yritys, missä henkilö työskentelee.

Valmiin ohjelman ensimmäisen version lopullisesti valmiiksi saamiseksi kirjoitin sen käytölle vielä pikaoppaan ohjelman käyttöön. Siinä kerrotaan ohjelman toiminta ja opastetaan lyhyesti ohjelman käyttö. Lopuksi loin vielä Visual Studion avulla ohjelmasta asennuspaketin, jonka avulla ohjelma saadaan asennetuksi käytettävälle tietokoneelle. Asennuspaketti sisältää myös ohjelman pikaoppaan. Valmis ohjelma toiminnassa on kuvassa 18. Seuraavaksi pääsin kokeilemaan ohjelmaa käytännössä.



Kuva 18. Workers-ohjelma toiminnassa.

5.9 Testaus

Sain tilaisuuden testata ohjelmaa Satakunnan ammattikorkeakoululla pidetyssä Innopäivässä. Innopäivän tarkoituksena oli esitellä SAMK:issa tehtävää tutkimus- ja kehitystyötä eri tahoille messutyyppisessä tapahtumassa. Yhtenä esittelyn kohteena oli RFID-tekniikka. Paikalle oli asennettuna RFID-laitteisto ja ohjelmani oli käynnissä tietokoneella. Innopäivän kävijöistä otettiin heidän halutessaan kuva web-kameralla ja henkilön tiedot lisättiin tietokantaan. Tämän jälkeen henkilölle annettiin tarratagi kiinnitettäväksi vaatteisiin. Tagitettujen henkilöiden liikkuessa tunnistusalueen sisäl-

lä heidän tietonsa menivät ohjelmassa jonoon ja sitä kautta näkyville kävijöille suunnattuun näyttöön.

Ohjelma toimi tapahtuman ajan moitteettomasti. Monia kiinnosti mihin järjestelmän toiminta perustuu. Tästä voidaan päätellä, että valmis ohjelma oli onnistunut ja sopi sille määriteltyyn tarkoitukseen hyvin. Erityisesti henkilön kuvan näkyminen herätti kiinnostusta.

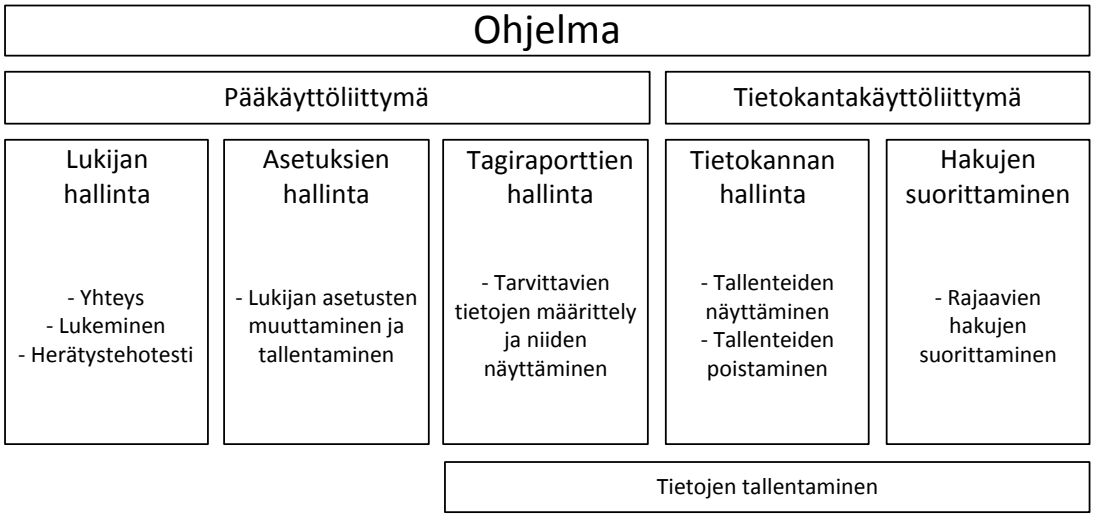
6 TAG READER -OHJELMA

6.1 Määrittely ja rajaus

Toisen ohjelman ohjeistuksena oli tehdä ohjelmisto ja tietokanta, joiden avulla tallennettaisiin automaattisesti vain tietyt lukijalta tulevat tagin lukutapahtuman tiedot. Tilaajan tarpeellisiksi näkemät tiedot olivat: tagin EPC, lähetysteho, vastaanottoteho ja aikaleima. Koska ohjeistuksessa ei otettu muuten kantaa ohjelman toimintaan, tein rajauksen itse. Rajasin ohjelman toiminnallisuuden niin, että ohjelmalla pystyy luomaan yhteyden lukijaan, lukemaan tageja, tallentamaan tagien tiedot tietokantaan, luomaan erilaisia rajaavia hakuja tietokannasta ja tallentamaan hakujen tulokset edelleen XML-tiedostoksi (Extensible Markup Language) jatkokäsittelyä varten.

6.2 Ohjelman suunnittelu

Tuleva ohjelma olisi huomattavasti laajempi toiminnoiltaan kuin ensimmäisenä tekemäni. Ensimmäisestä ohjelmasta saamani kokemuksen perusteella tein heti alustavan suunnitelman ohjelman rakenteesta. Lopullinen rakenne muodostuu prototyyppien avulla.

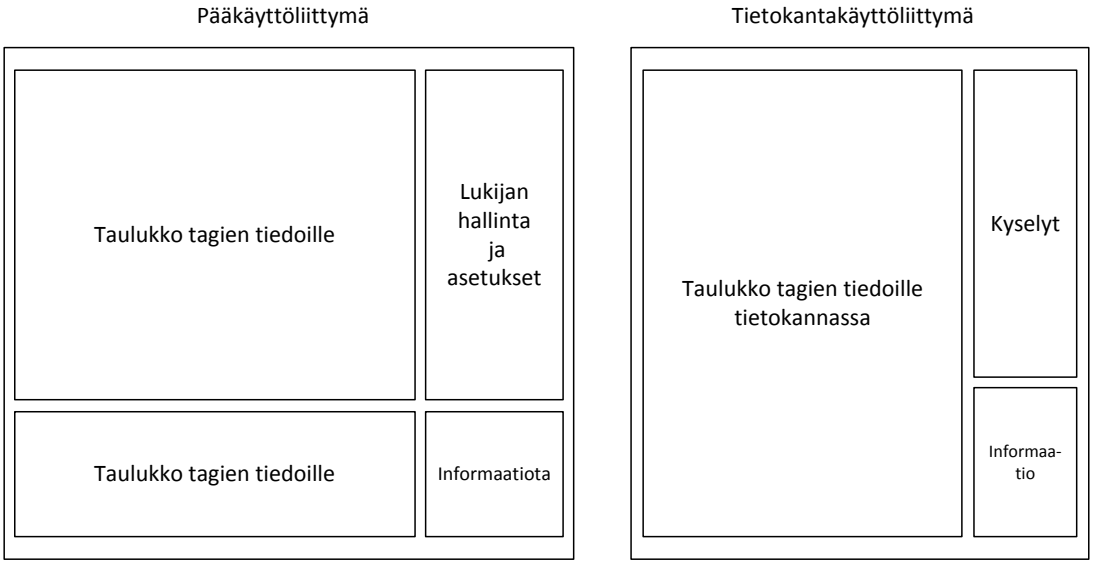


Kuva 19. Ohjelman osat.

Ohjelman jaoin kahteen pääosaan: pääkäyttöliittymä ja tietokantakäyttöliittymä (Kuva 19.). Näin sain luontevan jaon ohjelman toimintojen jakamiseen. Pääkäyttöliittymän alle tuli lukijan hallinta, asetuksien hallinta ja tagiraporttien hallinta. Tietokantakäyttöliittymä jakautui tietokannan hallintaan ja hakujen suorittamiseen. Lisäksi tietojen tallentamisesta tein oman lohkon.

Käyttöliittymiä (Kuva 20.) suunnitellessa järkeväksi tavaksi näyttää tagien tiedot päädyin taulukkoon. Siihen tiedot ovat helppo jäsennellä ja ne ovat selkeästi luettavissa. Piirsin suunnitelmassa kaksi taulukkoa pääkäyttöliittymään, joista toinen näytti kaikki tagien lukutapahtumat ja toinen näytti jokaisen eri EPC-koodin sisältävän tagin.

Koska ohjelmalla on tarkoitus suorittaa erilaisia mittauksia ja lukijan asetuksia muutetaan usein, päätin lisätä asetusten muuttamiseen vaadittavat osat suoraan pääkäyttöliittymään. Käyttöliittymässä halusin näyttää myös informaatiota ohjelman tapahtumista ja lukijan tilasta.



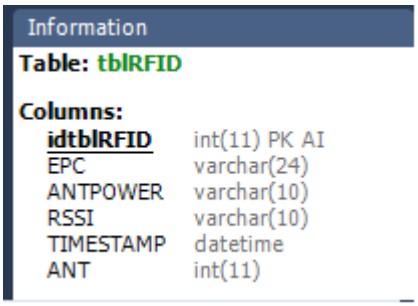
Kuva 20. Käyttöliittymien asettelusuunnitelma.

Tietokantakäyttöliittymään tein oman taulukon tietokannassa olevien tagien näyttämiseen ja sen viereen piirsin paikan tietokannan hallintaa ja kyselyitä varten. Tietokantakäyttöliittymä näyttäisi myös ohjelman tapahtumista informaatiota käyttäjälle.

6.3 Tietokanta

Ohjelmaa on tarkoitus käyttää koulun tiloissa, joten päädyin tekemään tietokannan koulun palvelimelle. Tämä mahdollistaa sen, että tallennettuihin tietoihin päästään käsiksi kaikilta koneilta, joihin ohjelma on asennettuna. Mittaukset pystytään suorittaamaan laboratoriossa olevalla koneella ja tallennetun datan analysointi eri koneella käyttäjän niin halutessa.

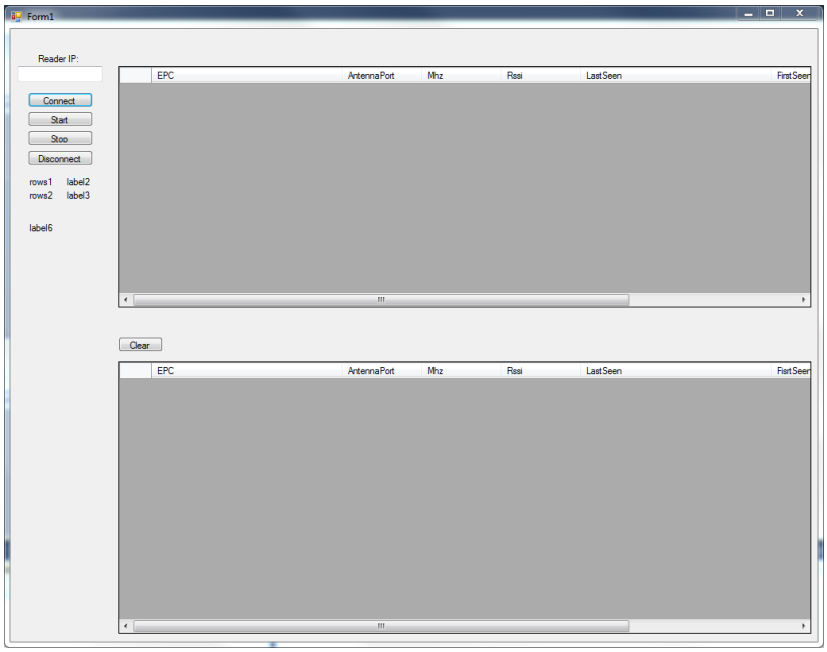
Tietokantaan tallennettaviksi tiedoiksi valittiin tagin EPC, antennin numero, lähetysteho, takaisinsirontateho ja aikaleima (Kuva 21.). Lisäksi tein sarakkeen automaattiselle tallenteen numeroinnille ja se toimii samalla pääavaimena. Ensimmäisessä ohjelmassani pääavaimena oli tagin EPC-koodi, mutta tässä tapauksessa se ei ollut mahdollista. Taulukkoon tallentuu samasta tagista useita eri lukukertoja ja pääavaimen täytyy olla yksilöllinen. Tietokannan ja taulukon luomiseen käytin MySQL Workbench -ohjelmaa.



Kuva 21. Tietokannan rakenne.

6.4 Ensimmäinen prototyyppi

Ensimmäistä prototyyppiä lähdin toteuttamaan tekemällä sille yksinkertaisen käyttöliittymän lukijan hallinnointiin ja taulukoiden toiminnan kokeilemiseksi. Ainoat toiminnot tässä vaiheessa olivat lukijan hallinta ja tagiraporttien hallinta.



Kuva 22. Ensimmäisen prototyypin käyttöliittymä.

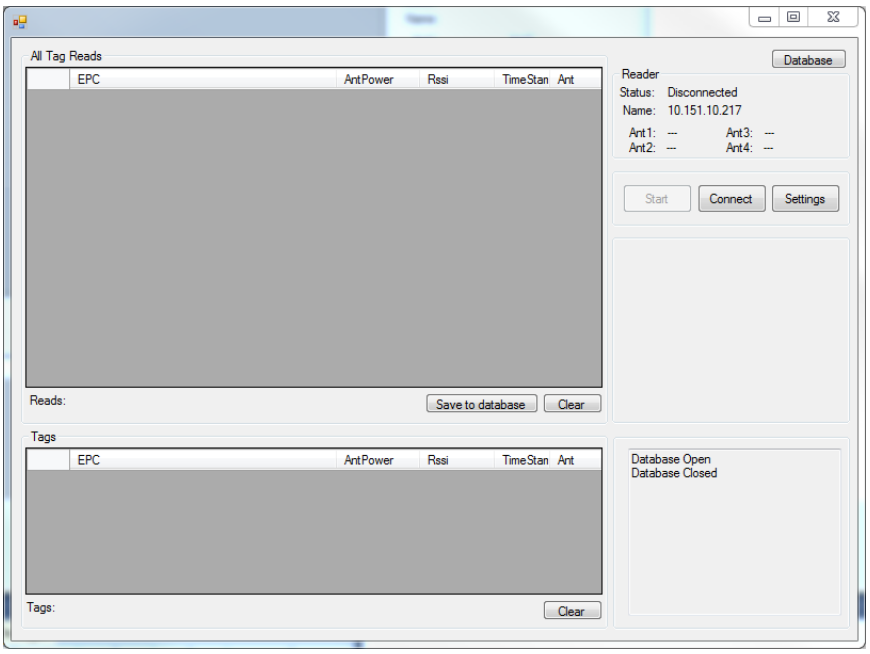
Käyttöliittymässä (Kuva 22.) oli painikkeet lukijan yhteyttä sekä tagien lukemista varten, kenttä lukijan osoitetta varten, rivimäärien näytöt ja kaksi taulukkoa tagien tietoja varten. Taulukoihin tein vielä tässä vaiheessa sarakkeet kaikille mahdollisille tiedoille, joita tagiraporteista oli mahdollista saada. Ylimääräiset tiedot voitiin helposti tiputtaa pois ennen lopullista ohjelmaa.

Ongelmallisin kohta prototyypin toteuttamisessa oli toiseen tauluun haluttujen vain eri EPC-koodin omaavien tagien lisääminen. Ratkaisuna tein koodiin osan, joka tarkistaa taulun jokaisen rivin EPC-solusta löytyykö luettu tagi jo taulusta. Jos tagia ei löydy, tauluun lisätään uusi rivi, muuten löydetty rivi korvataan uusilla tagiraportista saaduilla arvoilla.

Testeissä prototyyppi oli toimiva kokeiltavilta osin. Seuraavaan prototyyppiin päätettiin karsia ylimääräiset taulukoiden sarakkeet pois ja näkyviin jäisi ainoastaan määrittelyssä mainitut tiedot: EPC-koodi, lähetysteho, takaisinsirontateho, aikaleima ja antennin numero.

6.5 Toinen prototyyppi

Toiseen prototyyppiin muokkasin käyttöliittymän vastaamaan paremmin suunnitelmia. Uusiksi kokeiltaviksi ominaisuuksiksi tuli tietokantaan tallennus, lukijan asetusten muokkaus ja ohjelman tapahtumien näyttäminen käyttäjälle.



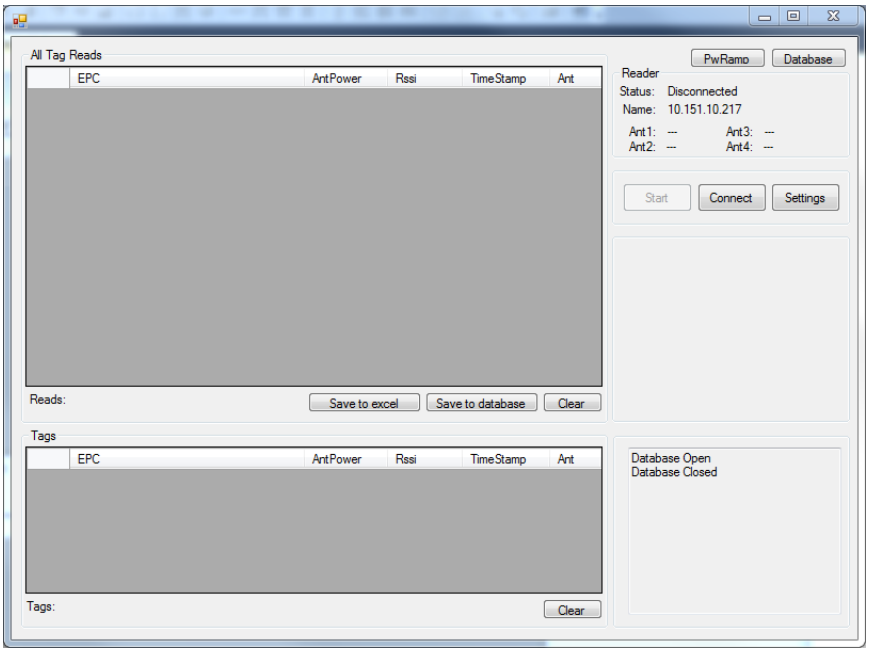
Kuva 23. Toisen prototyypin käyttöliittymä.

Käyttöliittymässä (Kuva 23.) oli nyt taulukot otsikkoineen ja sarakkeineen, kentät näyttämään ohjelman tapahtumia sekä painikkeet tietojen tallentamiseksi tietokantaan, lukijan hallinnointiin ja asetusten näyttämiseen. Yläkulmassa oli myös painike tietokantaan siirtymiseksi, mutta se ei vielä sisältänyt mitään toiminnallisuutta.

Rivien lisääminen tietokantaan oli suurin uudistus prototyypissä. Tässä vaiheessa se tapahtui ohjelmassa avaamalla yhteys tietokantaan, lisäämällä yhden rivin tiedot tietokantaan ja sen jälkeen sulkemalla yhteys. Tämä toimi hyvin pienillä tietomäärillä.

Testauksissa käyttöliittymän asettelu todettiin hyväksi, joten päätin lähteä toteuttamaan seuraavaa prototyyppiä. Siihen lisättäviä toimintoja ovat taulukoiden tietojen tallennus tiedostoksi ja automaattinen toiminto, jolla haetaan tagien aktivoitumiseen vaadittavaa lähetystehoa.

6.6 Kolmas prototyyppi



Kuva 24. Kolmannen prototyypin käyttöliittymä.

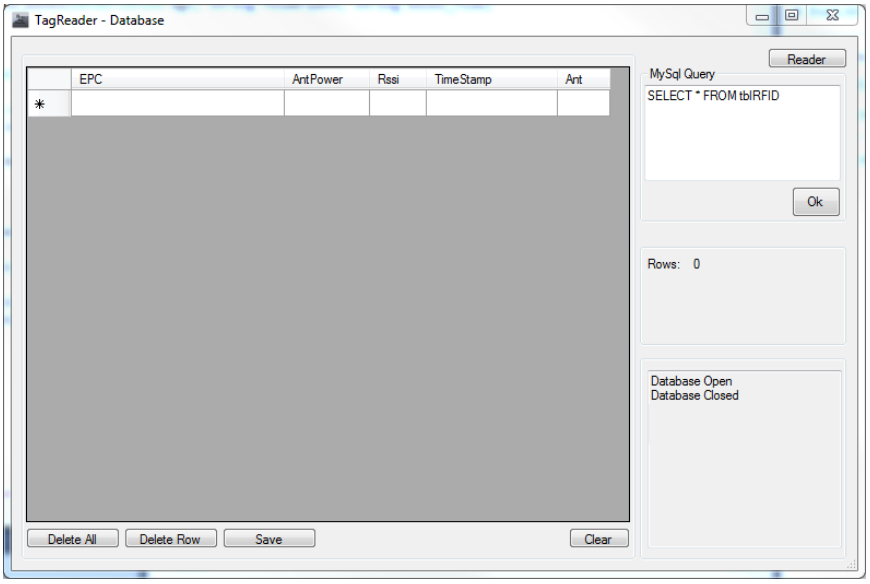
Prototyypin käyttöliittymä (Kuva 24.) sai nyt kaksi uutta painiketta uusien toimintojen käynnistämiseksi. Toinen oli taulukoiden tietojen tallentamiseksi XML-tiedostoksi ja toisella käynnistettiin tagien herätystehojen hakeminen.

Tallentaminen tiedostoksi onnistui käyttämällä ohjelman koodissa StreamWriter luokkaa ja sen metodeja. Metodien avulla luodaan uusi tiedosto, johon taulukon rivit kirjoitetaan solu kerrallaan. Samalla tein erillisen ikkunan, joka kysyy tiedostolle annettavaa nimeä käyttäjältä.

Tagien herätystehojen hakeminen onnistui luomalla kokonaan oma luokkansa, joka hoiti automaattisesti herätystehojen hakemiseen vaadittavat toimenpiteet sitä kutsuttaessa ohjelman sisällä. Nyt kaikki tapahtui automaattisesti nappia painettaessa. Ohjelma loi yhteyden lukijaan ja aloitti tagien lukemisen pienimmällä lähetysteholla. Pienin lukijan lähetysteho on 10 dBm, josta tehoa nostettiin 2 sekunnin välein +1 dBm ja lopetettiin lukeminen lähetystehon ollessa 31 dBm. Jokaisella lähetysteholla luetut tagit lisättiin ylempään taulukkoon ja ensimmäistä kertaa tagia luettaessa se lisättiin myös alempaan taulukkoon. Näin nähtiin helposti millä teholla tagi on vas-

tannut ensimmäisen kerran. Pääkäyttöliittymän toiminnot olivat nyt valmiita. Seuraavaksi siirryin tietokantakäyttöliittymän toteuttamiseen.

6.7 Neljäs prototyyppi



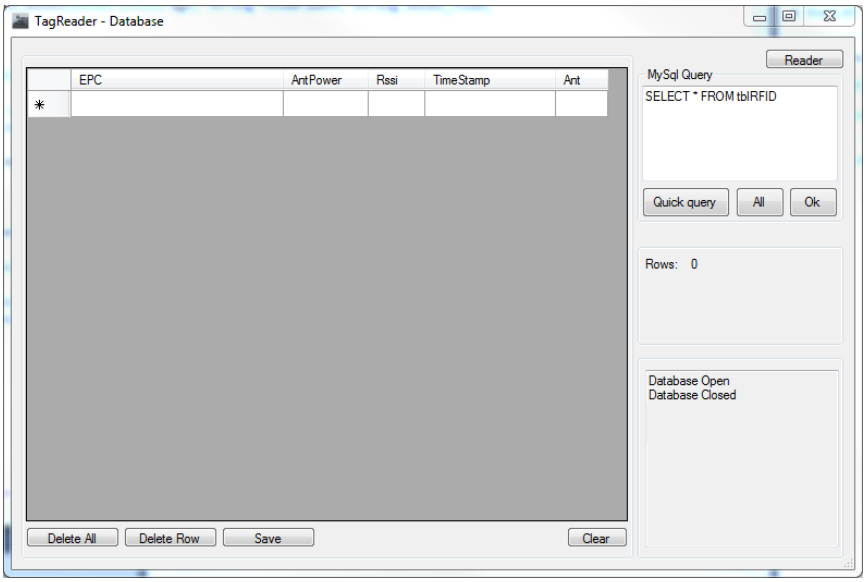
Kuva 25. Ensimmäinen tietokantakäyttöliittymä.

Tietokantakäyttöliittymä sai heti alustavan suunnitelmani mukaisen asettelun. Käyttöliittymään (Kuva 25.) tuli taulukko näyttämään tietokannassa olevat tiedot, painikkeet rivien poistamiseen, kyselyiden suorittamiseen ja taulukon tietojen tallentamiseen XML-tiedostoksi. Oikeassa yläkulmassa oli painike pääkäyttöliittymään siirtymiseksi ja kenttä SQL-kyselyiden suorittamiseen. Oikeasta alakulmasta löytyi kenttä, joka näytti ohjelman tapahtumia kuten tietokannan avaamisen, sulkemisen ja poistettujen rivien määrän.

Ohjelmaa testattiin nyt ensimmäistä kertaa kokonaisena pakettina muutamien testikäyttäjien avulla tarvittavien parannusten ja lisäysten selvittämiseksi. Käyttäjien antaman palautteen perusteella ohjelmaan kaivattiin mahdollisuutta muuttaa tiedostopolku, johon XML-tiedosto luotiin. Lisäksi tietokannasta tehtävien yleisimpien hakujen suoritusta haluttiin helpommaksi ja ohjelmassa oli muutamia ohjelman kaatumisen aiheuttavia virheitä. Näiden tietojen pohjalta lähdin toteuttamaan seuraavaa prototyyppiä.

6.8 Viides prototyyppi

Uudessa prototyypissä tietoja tallennettaessa annettiin käyttäjälle mahdollisuus valita sen tallennuspaikka erillisen hakemistoikkunan avulla. Tietokannan käyttöliittymä (Kuva 26.) sai kaksi painiketta lisää ja uuden ominaisuuden kyselyiden suorittamiseksi.



Kuva 26. Toinen tietokantakäyttöliittymä.

Uusista painikkeista toisella saatiin haettua kaikki tallenteiden tiedot tietokannasta ja toinen oli uutta kyselyiden tekemistä varten. Nyt käyttäjällä oli mahdollisuus suorittaa kyselyitä valitsemalla taulukosta solu, jonka sisältämällä arvolla tietoja haluttiin rajata. Esimerkiksi valittaessa jokin EPC-solu, niin kyselypainiketta painaessa taulukkoon tulee näkyviin vain saman EPC:n omaavat tallenteet.

Erilaiset ohjelman kaatumisongelmat tulivat myös korjatuksi. Testeissä ongelmaksi osoittautui tilanne, jossa käsiteltiin tuhansien tagien tietoja. Tällöin ohjelman toiminta hidastui huomattavasti. Lisättäessä tietokantaan esimerkiksi yli 10000 tagin lukutietoa lisäämisprosessi kesti aivan liian kauan. Sama ongelma oli tietokannasta yhtäaikaisesti useita tallenteita poistettaessa. Koska ohjelma oli kuitenkin toiminut muuten hyvin, päätin lähteä toteuttamaan lopullista ohjelmaa ja samalla parantaa ohjelman suorituskykyä.

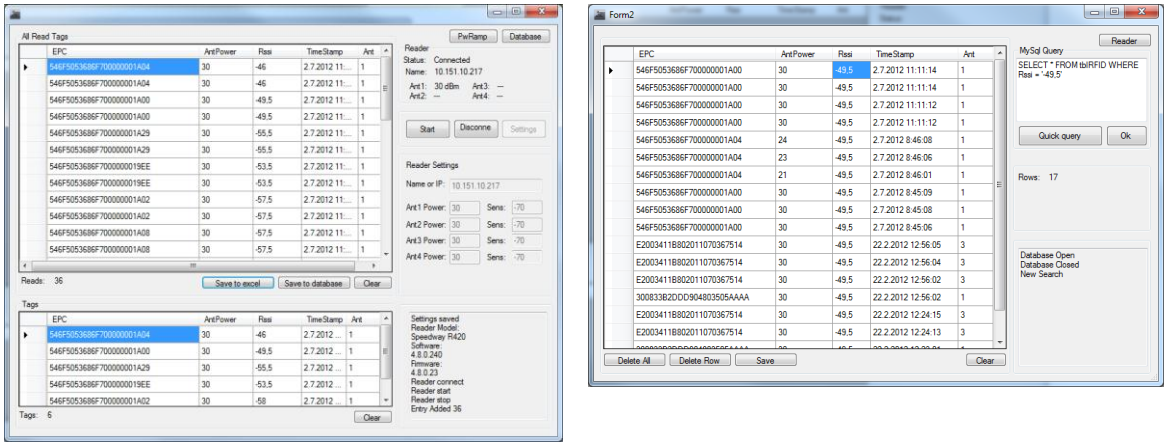
6.9 Valmis ohjelma

Viimeinen versio ohjelmasta valmistui prototyyppien pohjalta nopeasti. Sen suoritus kykyä sain parannettua huomattavasti. Tallenteiden lisäys tietokantaan nopeutui, kun koodissa muutin tietokantaan yhdistämisen ja yhteyden katkaisun tapahtuvaksi vain kerran lisäysprosessin aikana. Nyt yhteys tietokantaan avattiin tallenteiden lisäysten alkaessa ja katkaistiin vasta, kun kaikki lisäykset oli tehtynä.

Tallenteiden poistaminen nopeutui muuttamalla poistolauseke muotoon, jossa kaikki poistettavat tallenteet nidottiin yhteen lauseeseen ja se suoritettiin yhdellä kertaa. Ennen poistaminen tapahtui tallenne kerrallaan ja se kuormitti ohjelmaa turhaan.

Ohjelmakoodiin tuli tässä vaiheessa uusi osa, joka tarkisti suoritettavat SQL-lauseet. Lauseet, joilla voitaisiin tuhota tietokanta tai taulu tietokannassa, karsittiin pois. Näin tietämätön käyttäjä ei pääse vahingossa tuhoamaan tallenteitaan.

Kuten ensimmäiseen ohjelmaani, tähänkin kirjoitin vielä pikaoppaan ohjelman käyttämiseen. Ohjelmalle tein asennus paketin Visual Studion avulla ja pikaoppaan liitin siihen mukaan. Valmis ohjelma toiminnassa on kuvassa 27.



Kuva 27. Valmis ohjelma toiminnassa.

6.10 Testaus

Testikäyttäjät suorittivat valmiin ohjelman testauksen. Ohjelmaan tehtyihin parannuksiin oltiin hyvin tyytyväisiä ja uusia ongelmia ei löytynyt. Suurimman kiitoksen ohjelma sai nopeudesta ja helpoista tietokantahauista.

Ohjelman todellinen testi on kuitenkin se, kun sitä aletaan käyttää tilaajan toimesta säännöllisesti. Silloin ohjelman soveltuvuus tutkimustyön avuksi selviää ja ohjelman toimivuudesta saadaan mielipiteitä laajemmalta käyttäjämäärältä.

7 POHDINTA

7.1 Aikataulu

Opinnäytetyön aikataulu oli hyvin vaikea arvioida työn alussa, koska ohjelmistoprojektin työmäärää ei osattu arvioida aiemman kokemuksen puuttuessa. Tehtyjen ohjelmistojen valmistumisen kannalta aikataulu venyi alustavaa suunnitelmaa pidemmäksi. Molemmat ohjelmistot valmistuivat noin 2–3 viikkoa myöhässä. Myös opinnäytetyöraportin kirjoittamiseen kului hieman enemmän aikaa kuin olin aluksi suunnitellut. Aikataulun venyminen oli mielestäni kuitenkin pieni työmäärään nähden.

7.2 Opinnäytetyön lopputulos

Työn tavoitteena oli oppia ymmärtämään RFID-tekniikkaa ja toteuttaa sen toimintaa esittelevä ohjelma sekä tutkimustyössä apuna toimiva ohjelma. RFID osoittautui nopeasti hyvin laajaksi käsitteeksi ja sen rajaaminen opinnäytetyötä varten oli aluksi hankalaa. Kuitenkin aineistoja lukiessa ja oman tietämyksen laajentuessa rajauksen pystyin tekemään. Tässä raportissa päädyin kertomaan RFID-tekniikasta vain pintapuolin ja niiltä osin kuin työn ymmärtämisen kannalta on tärkeää.

En ollut aikaisemmin tehnyt mitään laajempaa ohjelmistoprojektia ja sen toteuttaminen ei ollut juurikaan tuttua minulle. Työtä aloittaessa jouduin perehtymään hyvin paljon ohjelmistoprojektin eri osiin ja niiden työvaiheisiin ja dokumentointiin. Alussa minulla oli tarkoituksena tehdä kattavaa dokumenttia kaikista projektin vaiheista, mutta työn edetessä vähensin dokumentoinnin määrää ja tein sitä vain tämän työn kannalta tärkeiltä osilta. Keräsin dokumentaatiota lähinnä eri ohjelma versioihin tehdyistä muutoksista ja ohjelmien rakenteesta. Tämä osoittautui hyväksi ratkaisuksi, koska muuten työn määrä olisi kasvanut luultavasti liian suureksi.

Ohjelmistoprojektin vaihejakomallin jouduin valitsemaan vain sen pohjalta, mitä olin kirjoista lukenut ja valitsemalla tähän projektiin mielestäni parhaiten sopivan. Prototyyppimalli osoittautui nopeasti hyväksi valinnaksi, koska ohjelmien kehitys tapahtui

pieni osa kerrallaan ja käytännön testien avulla. En kuitenkaan osaa sanoa olisiko jokin toinen vaihtoehto ollut parempi.

Työssä valmistuneiden ohjelmien tekeminen oli mielestäni mielenkiintoisin ja haastavin osuus. Uuden ohjelmointikielen opettelu, ohjelmien testaaminen ja ongelmien ratkaisu piti työn mielekkäänä, vaikka välillä jonkin ohjelman osan toteuttaminen tuntui mahdottomalta. Ohjelmakoodin lukeminen kävi kuitenkin jatkuvasti helpommaksi ja luontevammaksi työn edetessä ja ongelmia oppi ratkaisemaan monella eri tavalla. Valmistuneet ohjelmat täyttävät mielestäni niille annetut vaatimukset hyvin ja olen itse tyytyväinen tekemääni työhön. Jatkossa ohjelmia voisi kuitenkin kehittää tekemällä niihin mahdollisuuden käyttää useita lukijoita samaan aikaan ja lisäämällä ainakin Tag Reader -ohjelmaan tagien kirjoitus mahdollisuuden.

LÄHTEET

Arnowitz, J., Arent, M. & Berger, N. 2007. Effective Prototyping for Software Makers. San Francisco: Elsevier inc.

Bhuptani, M. & Moradpour, S. 2005. RFID Field Guide: Deploying Radio Frequency Identification System. Upper Saddle River (NJ): Sun Microsystems/Prentice Hall.

Haikala, I. & Märijärvi, J. 2006. Ohjelmistotuotanto. 11. painos. Helsinki: Talentum Media Oy ja tekijät.

Moghadampour, G. 2009. C# ohjelmointi. Helsinki: WSOYpro.

RFID Lab Finland ry. RFID- tietoutta. Viitattu 5.7.2012. http://www.rfidlab.fi/rfid-tietoutta

RFID Journal. RFID Business Applications, 1-4. Viitattu 5.7.2012. http://www.rfidjournal.com/article/view/1334/1

RFID Journal. RFID Consumer Applications and Benefits, 1-2. Viitattu 5.7.2012. http://www.rfidjournal.com/article/view/1332/1

Roussos, G. 2008. Networked RFID: Systems, Software and Services. New York: Springer