Janne Vierula

Terminaalin viivakoodinlukujärjestelmän

Suunnittelu ja toteutus

Opinnäytetyö
Kevät 2010
Tekniikan yksikkö
Tietotekniikka
Ohjelmistotekniikka

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

Koulutusyksikkö: Tekniikan yksikkö
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto: Ohjelmistotekniikan suuntautumisvaihtoehto

Tekijä: Janne Vierula

Työn nimi: Terminaalin viivakoodinlukujärjestelmän suunnittelu ja toteutus

Ohjaaja: Petteri Mäkelä

Vuosi: 2010 Sivumäärä: 49 Liitteiden lukumäärä: 1

Tässä työssä käydään läpi HahkaWay Oy:lle tehdyn uuden Viivakoodinlukujärjestelmän suunnittelun ja toteutuksen eri vaiheet. Viivakoodinlukujärjestelmän tehtävä on kerätä seurantaan tarvittavia tietoja kuljetusyksiköistä ja lähettää nämä tiedot HahkaWay Oy:n palvelimelle. Viivakoodinlukujärjestelmää käytetään HahkaWay Oy:n terminaalissa, jossa kuljetusyksiköiden lajittelu asiakkaille tapahtuu. Järjestelmän tuottamia tietoja käytetään moniin eri tarkoituksiin, joihin ei tässä työssä syvennytä. Työssä keskitytään uuden järjestelmän suunnittelun eri vaiheisiin ja tietoihin, joita viivakoodinlukujärjestelmä tuottaa.

Aluksi selvitettiin vanhan viivakoodinlukujärjestelmän toiminta ja rakenne, jonka jälkeen suunniteltiin uusi viivakoodinlukujärjestelmä. Suunnittelun tuloksena päätettiin, että vanhassa viivakoodinlukujärjestelmässä olevat työasemat ja laserviivakoodinlukijat korvataan Motorolan valmistamilla MT2090-käsipäätteillä. Käsipäätteisiin suunniteltiin ohjelmisto, joka siirtää kuljetusyksiköistä kerätyt tiedot palvelimelle. Ohjelmisto toteutettiin MT2090-käsipäätteille käyttäen Microsoft Compact Framework -ympäristöä ja Motorolan toimittamaa Enterprise Mobile Development Kit -luokkakirjastoa (EMDK).

Työn tuloksiksi saatiin valmis suunnitelma uudesta viivakoodinlukujärjestelmästä ja suunnitelma ohjelmiston rakenteesta MT2090-käsipäättelle, joka on osaksi toteutettu.

Asiasanat: Viivakoodi, Windows CE, MT2090, Microsoft .NET Compact Framework

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Faculty: School of Technology
Degree programme: Information Technology
Specialisation: Software Engineering

Author: Janne Vierula

Title of the thesis: Planning and Implementation of a Barcode Reading System

Supervisor: Petteri Mäkelä

Year: 2010 Number of pages: 49 Number of appendices: 1

The objective of this thesis is to design and implement a new barcode reading system for HahkaWay Oy, to replace the old reading system due to its aging. The purpose of the barcode reading system is to gather information about transport-units and transmit this information to HahkaWay Oy´s server. The system is used in the HahkaWay Oy terminal, where the sorting of transport-units takes place.

Firstly the operation and design of the old barcode reading system were studied, after that the planning of a new barcode reading system was started. Based on the results of planning it was decided that the new barcode reading system would work with Motorola MT2090 Mobile terminals instead of barcode scanners and workstations which the old barcode reading system relied on. It was also decided that software would be engineered to MT2090 Mobile terminals. The purpose of this software was to transmit information about transport units to HahkaWay Oy`s server. The software was engineered using Microsoft Visual Studio 2008, .NET compact Framework and EMDK, C# was used as programming language.

One result of this thesis was a plan of the new barcode reading system ready to be implemented, and another result was a partially implemented plan of software to MT2090 Mobile terminal.

Keywords: Barcode, Programming, .NET Compact Framework, MT2090, Windows CE

SISÄLLYS

TIIVISTELMÄ **2**

ABSTRACT **3**

KÄYTETYT TERMIT JA LYHENTEET **6**

KUVIO- JA TAULUKKOLUETTELO **8**

1 JOHDANTO **10**

1.1 Tausta 10

1.2 Tavoitteet 10

1.3 Toimeksiantaja 11

1.4 Rakenne 11

2 VIIVAKOODIT **12**

2.1 Historiaa 12

2.2 Rakenne 13

2.3 UPC ja EAN 13

2.4 Code 39 14

2.5 2D-Viivakoodit 14

3 RFID **16**

3.1 Viivakoodit ja RFID 16

3.2 EPC 17

4 MICROSOFT .NET COMPACT FRAMEWORK **18**

4.1 .NET- ja .NET CF -ympäristöjen erot 18

4.2 .NET CF -ympäristön pääkomponentit 19

4.3 Hallittu koodi 20

4.4 Muistinhallinta 22

5 VIIVAKOODINLUKUJÄRJESTELMÄ **23**

5.1 Järjestelmän tarkoitus 23

5.2 Vanhan järjestelmän rakenne 23

5.3 Uuden järjestelmän vaatimusten määrittely 25

5.4 Uuden järjestelmän rakenne 25

6 KÄSIPÄÄTE **27**

6.1 Vaatimukset 27

6.2 Testaus 29

7 OHJELMISTO 30
7.1 Tehtävä 30
7.2 Suunnittelu 31
7.2.1 Tietoliikenneliittymä 31
7.2.2 Käyttöliittymä ja sisäiset tietorakenteet 32
7.2.3 Rinnakkaiset toiminnat 34
7.2.4 Synkronointi 35
7.3 Toteutus 36
7.3.1 Käyttöliittymäluokka 37
7.3.2 Queue-luokka 38
7.3.3 Tunit-luokka 40
7.3.4 Synkronointi 41
7.4 Ohjelmiston toiminta 43
8 TULOKSET 46
8.1 Jatkokehitys 46
9 YHTEENVETO 47
10 JOHTOPÄÄTÖKSET 49
10.1 Tulevaisuuden näkymät 49
LÄHTEET 50

KÄYTETYT TERMIT JA LYHENTEET

Kuljetusyksikkö Minkä tahansa kokoinen pakkauskokonaisuus. (GS1 Finland 2010.)

Pudotusnumero HahkaWay Oy:llä käytössä oleva asiakkaan tunnus, koostuu kahdesta kolmen numeron mittaisesta numerosarjasta, joista ensimmäinen numerosarja määrittää reitin ja toinen reitillä olevan kaupan. (HahkaWay 2009.)

Terminaali Tila, missä hoidetaan elintarvikkeiden lajittelu ja varastointi ennen asiakkaalle kuljettamista, myös sama tila missä tämän opinnäytetyön tuloksena valmistuvaa järjestelmää tullaan käyttämään. (HahkaWay 2009.)

Reittilista HahkaWay Oy:n terminaalityöntekijöillä käytössä oleva lista, josta terminaalityöntekijät näkevät kuinka monta kuljetusyksikköä miltäkin tavaran toimittajalta menee millekin kaupalle. (HahkaWay 2009.)

WLAN Langaton lähiverkko, radioaalloilla toimiva tietoliikenteessä käytettävä langaton tiedonsiirtomekanismi. (Tietopankki 2009.)

HTTP Hypertext Transfer Protocol on tiedonsiirtoprotokolla, jota käytetään internet-selaimen ja palvelimen väliseen www-sivujen siirtämiseen. (About.com 2010.)

XML Tulee sanoista Extensible Markup Language ja on merkkipohjainen yleinen formaatti tiedon esittämiseen. (Msdn 2010c.)

Pikseli Kuvapiste on tietynkokoinen ja tietynvärinen tietokoneiden näytössä ja bittigrafiikassa käytettävä pienin mahdollinen yksikkö, joista kuva muodostuu. (Bittikarttagrafiikka 2008.)

Käskykanta Kaikki prosessorin tuntemat konekieliset käskyt muodostavat prosessorin käskykannan. (Männikkö 2000a.)

Kääntäjä Ohjelma, joka muuntaa ihmisymmärrettävän ohjelmakoodin konekieliseksi koodiksi. (Männikkö 2000b.)

Konekieli Konekieli on binäärimuotoisia käskyjä, joita prosessori suorittaa. (Männikkö 2000a.)

Luokka Olioperusteisessa ohjelmoinnissa käytettävä termi voidaan ajatella esimerkiksi itse suunniteltuna tietotyyppinä. joka sisältää metodeja ja muita tietotyyppejä. (Archer 2001, 9.)

Objekti Luokan perusteella muistiin luotu ilmentymä luokasta. (Archer 2001, 9.)

RoHS-direktiivi Euroopan unionin säännös, joka rajoittaa tiettyjen haitallisten aineden käyttöä sähkö- ja elektroniikkalaitteissa. RoHS-direktiivi kieltää esimerkiksi lyijypitoisen tinan käyttämisen elektroniikan valmistuksessa. RoHS-direktiivi on otettu käyttöön vuonna 2006. (RoHS 2009.)

KUVIO- JA TAULUKKOLUETTELO

Kuva 1. Viivakoodin perusrakenne 13

Kuva 2. ASCII-merkin sisällyttäminen Data Matrix -standardin viivakoodiin 15

Kuva 3. EPC-protokollapino 17

Kuva 4. .NET Framework -ympäristön rakenne 19

Kuva 5. Hallitun koodin suoritusprosessi 21

Kuva 6. Vanhan järjestelmän rakenne 23

Kuva 7. Reittilista 24

Kuva 8. Uuden järjestelmän rakenne 26

Kuva 9. Motorola MT2090-käsipääte 28

Kuva 10. Xml-dokumentin muoto 32

Kuva 11. Käyttöliittymän pääikkuna suunnitteluvaiheessa 33

Kuva 12. Ohjelmiston säikeet 34

Kuva 13. Synkronoinnin periaatekuva 36

Kuva 14. Käyttöliittymäluokka 38

Kuva 15. Queue-luokka 39

Kuva 16. Tunit-luokka 41

Kuva 17. Synkronoinnin toteutusperiaate 42

Kuva 18. MT2090-viivakoodinlukuohjelmiston käyttäjän valinta 43

Kuva 19. MT2090-viivakoodinlukuohjelmiston käyttöliittymän pääikkuna 44

Kuva 20. MT2090-viivakoodinlukuohjelmiston virheilmoitus 45

1 JOHDANTO

1.1 Tausta

Kuljetusyksiköllä tarkoitetaan minkä tahansa kokoista pakkauskokonaisuutta, jota liikutellaan toimitusketjussa. Kuljetusyksiköiden seuranta on tärkeää toimitusketjussa, koska kuljetusyksiköt matkaavat pitkiä matkoja ennen kuin ne päätyvät tavaran valmistajalta kaupan hyllyille.

Tärkeintä kuljetusyksiköiden seuranta on terminaalitiloissa, joissa hoidetaan tavaroiden lajittelu valmistajakohtaisista kuljetusyksiköistä kauppakohtaisiin kuljetusyksiköihin. Tämä tarkoittaa käytännössä sitä, että lavat ja rullakot puretaan ja ne uudelleenlajitellaan kauppakohtaisiksi lavoiksi ja rullakoiksi. Lajitteluvaiheessa virheitä tulee ja nämä virheet maksavat. Toinen kriittinen kohta toimitusketjussa on tavaroiden lastaus kuljetettavaksi. Näiden virheherkkien alueiden virheet pyritään poistamaan kokonaan kuljetusyksiköiden seurannan avulla.

HahkaWay Oy:n terminaalitiloissa on viivakoodinlukujärjestelmä, jota käytetään kuljetusyksiköiden seurantaan. Tämä viivakoodinlukujärjestelmä on kuitenkin tullut käyttöikänsä päähän ja RoHS-direktiivin takia Suomessa ei enää saa myydä varaosia kyseiseen viivakoodinlukujärjestelmään. Tämän takia terminaalitiloihin täytyi suunnitella ja toteuttaa uusi viivakoodinlukujärjestelmä. (HahkaWay 2009.)

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa uusi viivakoodinlukujärjestelmä vanhan järjestelmän korvaajaksi HahkaWay Oy:n terminaalitiloihin. Uuteen viivakoodinlukujärjestelmään tehtiin rakennemuutoksia. Nämä muutokset olivat työasemien poisto terminaalitiloista ja laser-viivakoodinlukijoiden korvaus MT2090-

käsipäätteillä. Näistä rakennemuutoksista johtuen täytyi käsipäätteelle suunnitella ohjelmisto, joka siirtää tiedot luetuista kuljetusyksiköistä HahkaWay Oy:n palvelimelle.

1.3 Toimeksiantaja

Opinnäyteyön toimeksiantajana toimi HahkaWay Oy. HahkaWay Oy on logistiikka-alan yritys, joka hoitaa lämpötilasäädettyä elintarvikejakelua Länsi-Suomen läänin alueella ja ei lämpötilasäädeltyjä kuljetuksia koko maahan sekä kuivakuljetuksia Helsinkiin. Tämän lisäksi HahkaWay Oy:llä on varastointitoimintaa Seinäjoella. HahkaWay Oy on perustettu nykyiseen muotoonsa vuonna 2005, yrityksessä työskentelee noin 70 työntekijää. (HahkaWay 2009.)

1.4 Rakenne

Luvussa 2 käsitellään viivakoodeja, niiden historiaa, rakennetta ja standardeja. Luvussa 3 kerrotaan RFID-tekniikasta ja sen tämänhetkisestä tilanteesta viivakoodin korvaajana. Luvussa 4 kerrotaan .NET Compact Framework -alustasta, jota käytettiin tämän opinnäytetyön tuloksiksi saadun ohjelman kehitykseen.

Luvussa 5 selvitetään viivakoodinlukujärjestelmän tarkoitus HahkaWay Oy:n terminaalitiloissa, vanhan järjestelmän rakenne ja uuden järjestelmän vaatimukset sekä sen suunniteltu rakenne.

Luvussa 6 käsitellään käsipäätteen valintaprosessi, valitun MT2090-käsipäätteen ominaisuudet ja testaus. Luvussa 7 käsitellään käsipäätteen ohjelmiston tehtävä ja ohjelmiston suunnittelu, toteutus sekä ohjelmiston toiminta opinnäytetyön kirjoitushetkellä.

Luvussa 8 käydään läpi työn tulokset ja jatkokehityssuunnitelmat ja luvussa 9 käydään läpi yhteenveto. Luvussa 10 käsitellään johtopäätöksiä ja tulevaisuuden näkymiä.

2 VIIVAKOODIT

Viivakoodit ovat laajalti käytössä erilaisissa tiedonkeruujärjestelmissä ympäri maailmaa. Tässä luvussa kerrotaan viivakoodien historiasta, rakenteesta ja käydään läpi yleisimpiä viivakoodistandardeja.

2.1 Historiaa

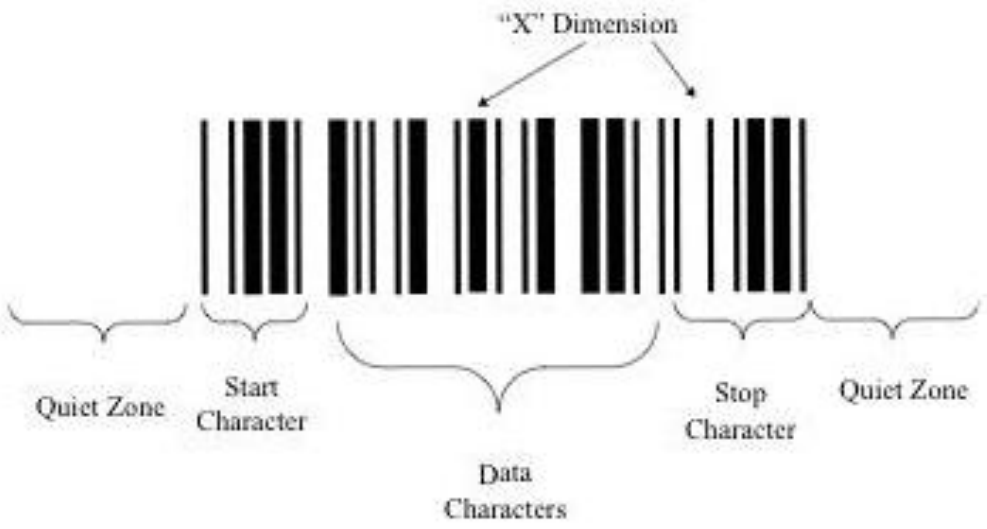
Viivakoodeja käytetään nykyään jokaisessa kaupassa. Ensimmäinen askel viivakoodien kehitykseen otettiin vuonna 1948 Bernard Silverin ja Joseph Woodlandin toimesta. Silver kuuli yliopiston käytävällä tiedekunnan johtajan ja ruokaketjun presidentin välisen keskustelun, että yliopisto alkaisi kehittää automaattista tuotteiden tiedonkeruujärjestelmää kassoille, mistä yliopisto kieltäytyi. Vuonna 1952 Woodland ja Silver kehittivät ensimmäisen viivakoodinlukulaitteen prototyypin, joka oli pöydän kokoinen ja vei 5 Kw sähköä. Keväällä 1969 ensimmäiset viivakoodinlukulaitteet asennettiin General Motorsille ja General Tradingille. Laitteet olivat alkeellisia, sillä ne pystyivät tulkitsemaan vain kahden numeron pituisia viivakoodeja. (Barcoding 2003.)

Vuonna 1971 IBM sai valmiiksi nykyisinkin vielä käytössä olevan Universal Product Code -standardin (UPC), jonka kehityksessä Woodlandillä oli tärkeä rooli. Vuonna 1973 UPC-standardi otettiin käyttöön teollisuudessa. Vuonna 1960 kehitetty laser-tekniikka mahdollisti viivakoodienlukijoiden valmistuksen edullisesti. Tämän seurauksena vuonna 1974 ensimmäinen viivakoodilukijan avulla rekisteröity tuote saatiin myytyä. Myöhemmin Woodland kehitti UPC-standardista hieman laajemman version European Article Numbering System -standardin (EAN), josta on tällä hetkellä tulossa laajimmin käytössä oleva standardi. (Barcoding 2003.)

2.2 Rakenne

Viivakoodi koostu symboleista ja symboli koostuu tietystä määrästä elementtejä. Elementti on musta tai valkoinen tietynpaksuinen pystysuora viiva. Viivakoodeista on monia standardeja. Standardit määrittävät sen, mitä kirjainta tai numeroa tietty symboli vastaa, montako eri paksuista elementtiä symboli saa sisältää ja onko standardin tyyppi erillinen vai jatkuva. Tunnetuimpia standardeja ovat EAN, UPC, Code 39 ja Code 128. Erilaisista standardeista huolimatta viivakoodilla on aina tietyt perusosat, ne ovat hiljainen alue, alku- ja loppumerkit ja data-merkit. (Max 2002, 89 - 100.)

Hiljainen alue kertoo viivakoodinlukijalle sen alueen, mistä elementtien mittaus aloitetaan. Alku- ja loppumerkit kertovat viivakoodinlukijalle kohdan, mistä itse data-merkit alkavat ja mihin ne loppuvat. Data-merkit sisältävät datan, joka voi olla kirjaimia tai numeroita riippuen viivakoodin standardista. Viivakoodin perusrakenne selviää kuvasta (kuva 1). (Max 2002, 89 - 100.)



Kuva 1. Viivakoodin perusrakenne (Max 2002, 94.)

2.3 UPC ja EAN

UPC (Universal Product Code)- ja EAN (European Article Number) -standardit ovat hallittuja viivakoodistandardeja. Tämä tarkoittaa sitä, että molempien viiva-

koodistandardien käyttöä valvotaan tarkasti. Amerikassa käytössä olevan UPC-standardin valvonnan hoitaa GS1-US, joka tunnettiin aiemmin nimellä UCC (Uniform Code Council). Muualla maailmassa käytössä olevan EAN-standardin valvonnasta huolehtii GS1, joka enemmin tunnettiin nimellä EAN International. UPC-standardista on tällä hetkellä olemassa viisi eri versiota ja EAN-standardista kaksi. Kaikki EAN- ja UPC-standardit käyttävät neljää eri elementtileveyttä ja ne pystyvät esittämään vain numeroita. Kaikkien EAN- ja UPC-standardien käyttöön täytyy anoa käyttölupa. (BarCode 1 2009a.)

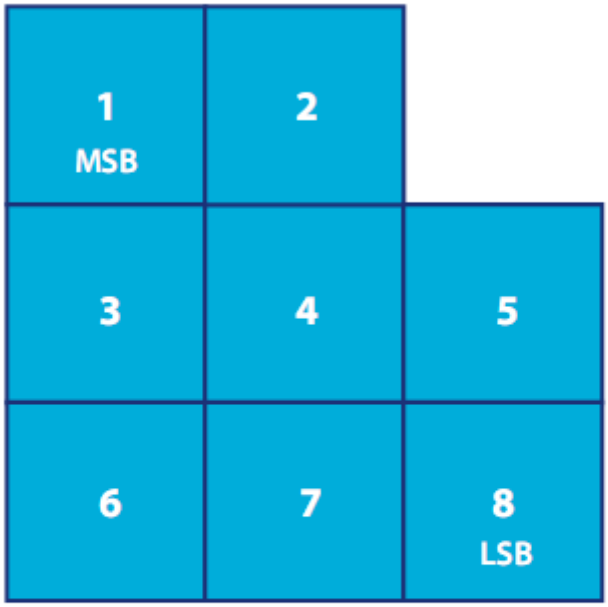
2.4 Code 39

Code 39 -standardin yksi symboli eli merkki koostuu yhdeksästä elementistä, joista kolme on paksuja ja loput kuusi ohuita. Näistä yhdeksästä elementistä viisi on mustia ja neljä valkoisia. Code 39 -standardilla pystytään esittämään numero 0 - 9, isot kirjaimet A - Z ja seitsemän erikoismerkkiä. Näistä seitsemästä erikoismerkistä yhtä käytetään alku- ja loppumerkkinä. (BarCode 1 2009b.)

2.5 2D-Viivakoodit

2D-viivakoodit eroavat normaaleista (1D-viivakoodeista) niin, että kaikissa 2D-viivakoodistandardeissa tieto tallennetaan viivakoodiin matriisimuotoon. 2D-viivakoodien etuja 1D-viivakoodeihin nähden ovat niiden tiedon tiheys, koska 2D-viivakoodeissa tieto tallennetaan pysty- ja vaakasuoraan. Haittoja on virhealttius ja viivakoodinlukunopeus. (BarCode 1 2009c.)

Data Matrix -viivakoodistandardissa elementti on yksi pikseli ja symboli on tietty määrä pikseleitä matriisimuodossa. Pikseleiden väri matriisissa määrää pikselillä esitettävän tiedon. Data Matrix -viivakoodistandardissa pikselit voivat olla joko mustia tai valkoisia. Mustalla pikselillä tarkoitetaan 1-bittiä ja valkoisella 0-bittiä, eli 8-bittiä pitkä ASCII-merkki voidaan esittää 3x3-pikselin kokoisella matriisilla. (GS1 2009, 45 - 62.)



Kuva 2. ASCII-merkin sisällyttäminen Data Matrix -standardin viivakoodiin (GS1 2009, 62.)

Kuvassa (kuva 2) näkyy yksi Data Matrix -viivakoodistandardin yhdeksän elementin kokoinen symboli, jolla pystytään esittämään 1-tavun pituinen ASCII-merkki. vasemmassa yläkulmassa sijaitseva pikseli vastaa eniten merkitsevää bittiä ASCII-tavussa ja oikeassa alakulmassa sijaitseva pikseli vastaa vähiten merkitsevää bittiä.

3 RFID

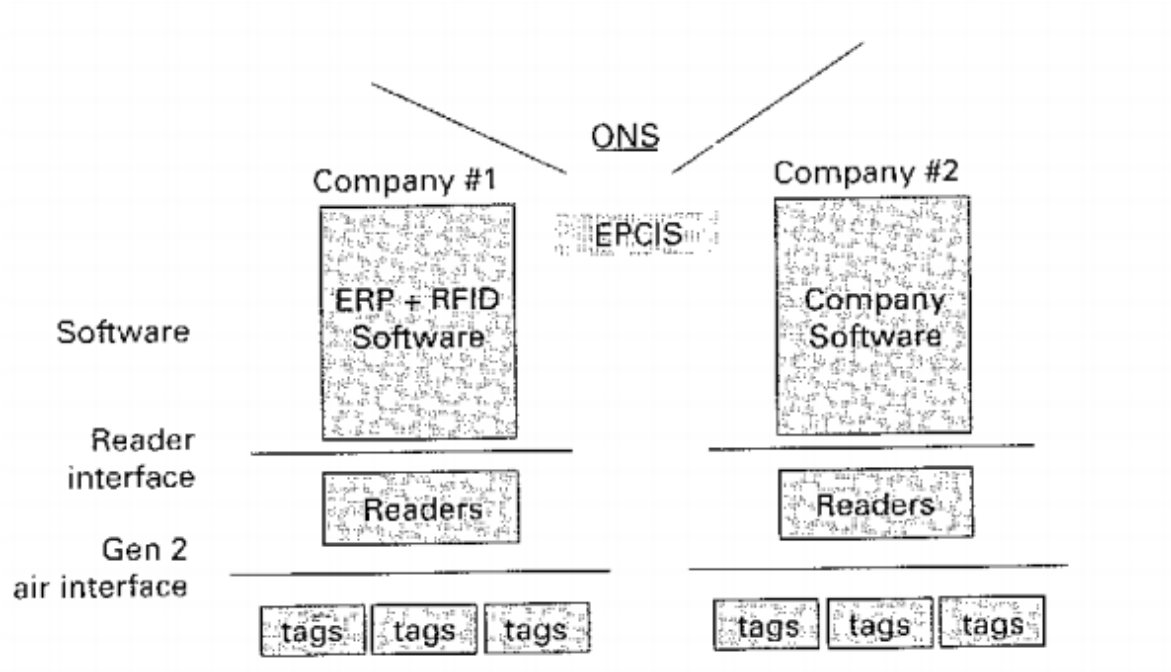
RFID-lyhenne tulee sanoista Radio Freguency IDentification. RFID-lyhenteellä tarkoitetaan yleensä RFID-tageja. RFID-tagi on elektroninen laite, joka sisältää mikroprosessorin, flash-muistin, radiopiirin ja antennin. RFID-tageja käytetään tiedon varastointiin. Tieto voidaan lukea ja kirjoittaa RFID-tagiin langattomasti radiosignaalien avulla. RFID-tagit jaetaan kahteen pääluokkaan, jotka ovat passiiviset ja aktiiviset. Nämä eroavat toisistaan sillä tavalla, että passiiviset RFID-tagit ottavat lukuun tarvitsemansa energian radiomagneettisesta säteilynä RFID-lukijalta. Aktiivisissa RFID-tageissa on erillinen virtalähde ja ne vastaanottavat ja lähettävät tietoa tietyn väliajoin. Suurin ero näiden kahden RFID-tagi tyypin välillä on niiden lukuetäisyys, passiivisessa lukuetäisyys on alle metrin, ja aktiivisissa se on metristä ylöspäin aina kymmeniin metreihin asti riippuen lähetystehosta. (Hedgepeth 2007, 10 - 14.)

3.1 Viivakoodit ja RFID

RFID-tageja on jo pitemmän aikaa suunniteltu viivakoodien korvaajiksi toimitusketjun seurantajärjestelmissä, koska ne ovat joka osa-alueella nopeampia ja helpompia kuin viivakoodit. Ensimmäinen etu RFID-tageilla viivakoodeihin nähden on RFID-tagien lukemisen helppous. Lukemisen helppoudella tarkoitetaan sitä, että RFID-tunnisteita käytettäessä ei tarvita näköyhteyttä toisin kuin viivakoodeja käytettäessä. Tämä etu koskee kuitenkin vain aktiivisia RFID-tageja. Toinen etu on RFID-tagien uudelleenkäyttömahdollisuus. Uudelleenkäyttömahdollisuudella tarkoitetaan, että käytön jälkeen tagia voidaan käyttää jossain toisessa tuotteessa tai kuljetusyksikössä, koska RFID-tagiin tieto voidaan uudelleenkirjoittaa. Kolmas etu RFID-tageilla on niiden kestävyys rankoissa olosuhteissa. Hyvin koteloituna RFID-tagi kestää sellaisia olosuhteita, mitä viivakooditarralle tulostettu viivakoodi ei kestä. RFID-tagit ovat tulevaisuudessa viivakoodien korvaajia, mutta niiden hinnat ovat vielä liian kalliita verrattaessa viivakoodeihin. (Hedgepeth 2007, 4 - 14.)

3.2 EPC

EPC tulee sanoista Electronic Product Code, joka on RFID-tekniikan avulla toteutettu standardi tuotteiden seurantaan toimitusketjussa. EPC-standardi on EPCGlobal-yrityksen vielä kehitteillä oleva standardi. EPC-standardin tämänhetkinen toiminta voidaan kuvata kolmekerroksisella EPC-protokollapinolla. Nämä kerrokset ovat ”Software”, ”Reader Interface” ja ”Gen 2 air interface”. Software-kerros hoitaa kommunikoinnin yritysten välillä EPC information services (EPCIS) -standardin avulla. EPCIS-standardi määrittää säännöt sille, missä muodossa tiedot tuotteista siirtyvät yritysten välillä. Reader Interface -kerros hoitaa kommunikoinnin RFID-lukijan ja Software-kerroksen välillä, eli käytännössä se hoitaa tavaroiden tietojen siirron RFID-lukijalta palvelimelle. Gen 2 Air Interface -kerros huolehtii tiedon välityksestä RFID-tagien ja RFID-lukijan välillä. EPC Gen2 -protokolla on kuitenkin vielä kehitteillä ja siitä puuttuu esimerkiksi tiedon salaus ennen lähetystä tagin ja lukijan välillä. (Miles, Sarma & Williams 2008, 16 - 20.)



Kuva 3. EPC-protokollapino (Miles & ym 2008, 17.)

EPCIS-standardista on olemassa vapaanlähdekoodin toteutus Fosstrak EPCIS. Fosstrak EPCIS toteuttaa 1.0.1-version EPCIS -standardista ja se mahdollistaa EPCIS-standardin testauksen yritysten välillä. (Fosstrak 2009.)

4 MICROSOFT .NET COMPACT FRAMEWORK

Micrososft .NET Compact Framework -alusta on yhtenäinen Windows mobile komponentti, jota käytetään ohjelmistokehitykseen. .NET Compact Framework -alustaa kutsutaan yleensä .NET CF -ympäristöksi .NET CF -ympäristö on laitteistoriippumaton ja koostuu kahdesta pääkomponentista Common Language Runtime -virtuaalikoneesta (CLR) ja .NET -luokkakirjastosta. (Msdn 2010a.)

Tässä luvussa kerrotaan .NET CF -ympäristön rakenteesta, koska tämän opinnäytetyön tuloksiksi saatu ohjelmisto kehitettiin .NET CF -ympäristöä käyttäen.

.NET CF -ympäristön rakenne perustuu Microsoftin aikaisemmin Windows -käyttöjärjestelmälle kehittämään .NET Framework -ympäristöön. Vaikka nämä ympäristöt ovat melkein identtisiä toistensa kanssa, niin ei niitä pidä sekoittaa. .NET Framework -ympäristö on suunniteltu käytettäväksi Windows -käyttöjärjestelmillä PC-tietokoneissa, kun taas .NET CF -ympäristö on optimoitu pienitehoisille mobiililaitteille. .NET CF -ympäristö toimii Windows CE- ja Windows Mobile -käyttöjärjestelmissä. (Msdn 2010a.)

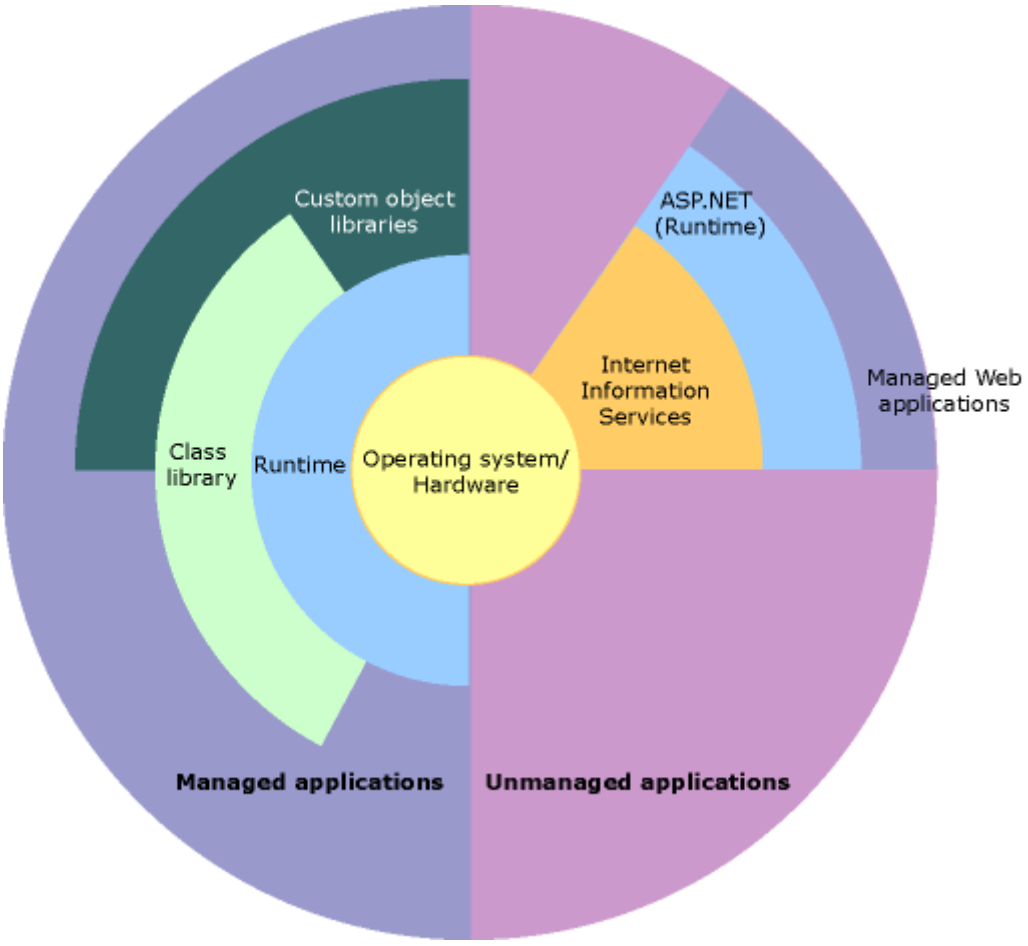
4.1 .NET- ja .NET CF -ympäristöjen erot

Kokonaisuudessa .NET CF -ympäristön koko on vain 8 % .NET Framework -ympäristön koosta. Tämä on saavutettu niin, että .NET CF -ympäristön luokkakirjasto toteuttaa noin 30 % .NET-ympäristön täydestä luokkakirjastosta. CLR-virtuaalikoneen koko on kutistettu 12 %:iin .NET -ympäristön virtuaalikoneen koosta. .NET CF -ympäristön pienempi koko tarkoittaa myös sitä, että joitain ominaisuuksia on jätetty pois. Esimerkkinä pois jätetyistä ominaisuuksista voidaan mainita tuki ASP.NET-tekniikalle. (Msdn 2010d.)

4.2 .NET CF -ympäristön pääkomponentit

Common Language Runtime(CLR) on ohjelma, joka suorittaa .NET-kääntäjien tuottamaa hallittua Microsoft Intermediadate Language(MSIL) -koodia. Suorittaessaan MSIL-koodia CLR-virtuaalikoneen sisältämä kääntäjä tekee MSIL-koodille täsmäkäännöksen. Täsmäkäännöksen aikana MSIL-koodi käännetään laitteen prosessorin käskykannalle sopivaksi konekieliseksi koodiksi, jota sitten suoritetaan laitteen prosessorilla. (Liberty 2005, 3.)

.NET Framework -luokkakirjasto on kokoelma luokkia, jotka sisältävät tarvittavat funktiot ohjelmistokehitykseen (Msdn 2010b). Kuvasta (kuva 4) selviää .NET Framework -ympäristön rakenne, joka vastaan suurimmaksi osaksi .NET CF -ympäristön rakennetta.

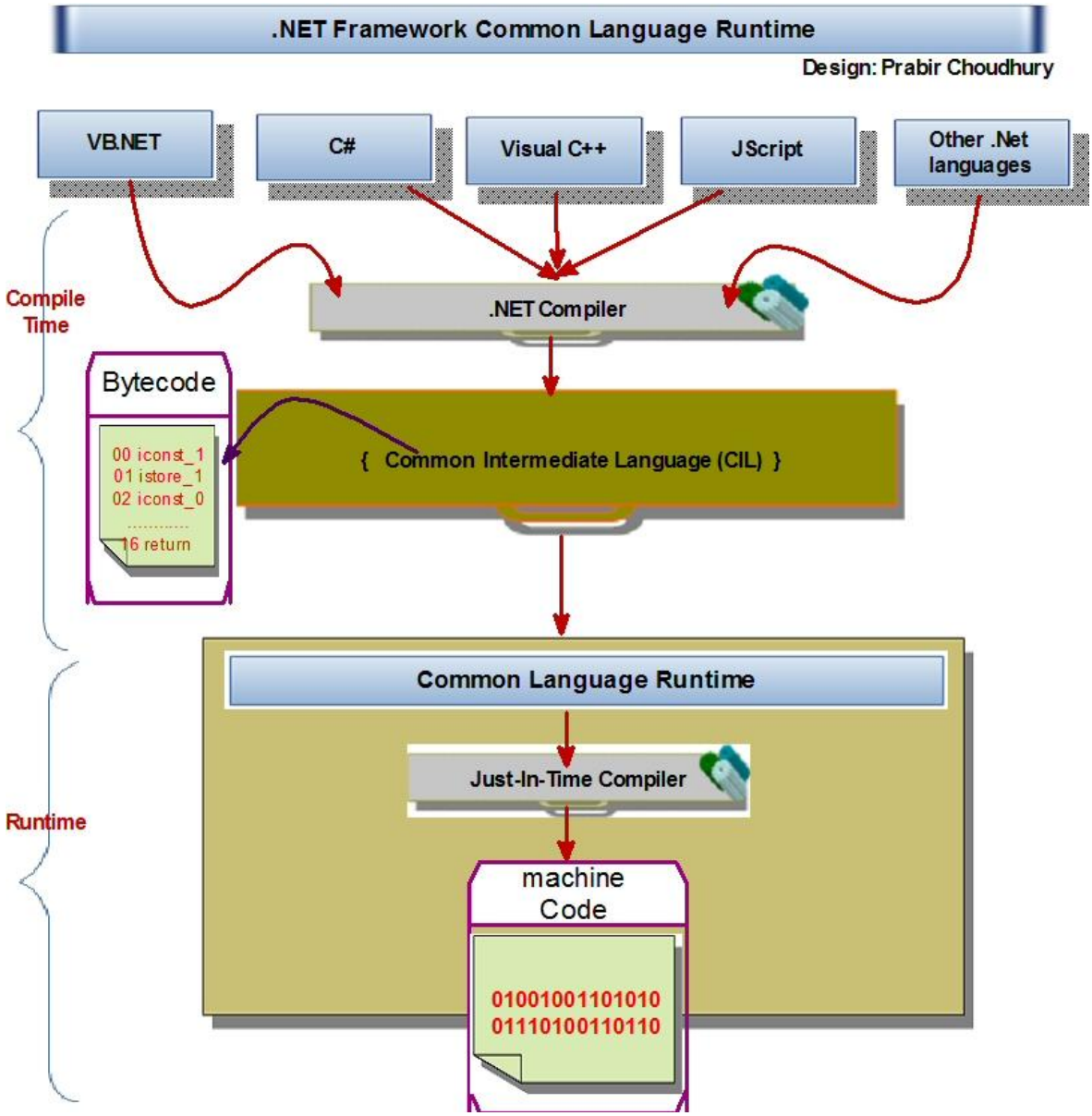


Kuva 4. .NET Framework -ympäristön rakenne (Msdn 2010b.)

Kuvassa (kuva 4) termi “Class library” vastaa .NET Framework -luokkakirjastoa. Vastaavasti termi ”Runtime” on CLR. Termi ”Custom object libraries” vastaa kolmannen osapuolen toimittamia luokkakirjastoja. Termi “Managed applications” vastaa hallittua koodia. Termi ”Unmanaged applications” vastaa ei-hallittua koodia ja termi ”Operating system” vastaa käyttöjärjestelmää. Termit ”Internet Information Services”, ”ASP.NET(Runtime)” ja ”Managed Web applications” voidaan jättää huomioimatta, koska ne eivät sisälly .NET CF -ympäristöön. Kuvasta 4 selviää myös, miten natiivi koodi keskustelee suoraan laitteen käyttöjärjestelmän rajapinnan kanssa, kun taas halitussa koodissa käyttöjärjestelmän kanssa keskustelun hoitaa CLR. (Msdn 2010b.)

4.3 Hallittu koodi

Hallittu koodi on CLR-virtuaalikoneen suorittamaa MSIL-koodia. Hallitun ohjelmakoodin suoritus eroaa natiivin eli ei-hallitun ohjelmakoodin suorittamisesta seuraavalla tavalla: Perinteinen eli natiivi ohjelmakoodi käännetään suoraan prosessorin käskykannalle sopivaksi konekieliseksi koodiksi ohjelmistokehittäjän tietokoneella. Käännöksen jälkeen koodi ladataan laitteen muistiin ja suoritetaan laitteen prosessorilla. Hallittu koodi käännetään kohdelaitteesta riippumatta MSIL-koodiksi ohjelmistokehittäjän tietokoneella. Käännöksen jälkeen MSIL-koodi ladataan laitteelle, jossa CLR-virtuaalikone tarkastaa ja suorittaa koodin. Suorituksen aikana täsmäkääntäjä kääntää koodin laitteen prosessorin käskykannasta riippuen oikeanmuotoiseksi konekieliseksi koodiksi. Täsmäkäännöksen jälkeen natiivi konekielinen koodi suoritetaan laitteen prosessorilla. Kuvasta (kuva 5) selviää hallitun koodin suoritusprosessi. (Boling 2003, 1113 - 1115.)



Kuva 5. Hallitun koodin suoritusprosessi (Choudhury 2008.)

Hallitun koodin etuja natiiviin koodiin ovat sen prosessoriarkkitehtuuririippumattomuus ja tyyppiturvallisuus. Prosessoriarkkitehtuuririippumattomuus tarkoittaa sitä että kerran käännettyä ohjelmaa voidaan suorittaa kaikilla laitteilla, joissa on CLR, vaikka laitteiden prosessoriarkkitehtuurit eroavaiskivatkin toisistaan. Tyyppiturvallisuus tarkoittaa sitä, että muunnokset tietotyyppien välillä tarkastetaan. Tällöin virheellisiä arvomuunnoksia ei pääse tapahtumaan. Haittoja ovat lisääntyneet muistin- ja prosessorinkäyttö. Lisäksi jokaisella laitteella, missä halutaan hallittua koodia suorittaa, on oltava CLR-virtuaalikone. (Boling 2003, 1113 - 1115.)

4.4 Muistinhallinta

Hallitun koodin toinen ominaisuus on automaattinen muistinhallinta, jota kutsutaan roskienkeruuksi. .NET CF -ympäristössä roskienkeruun hoitaa CLR-virtuaalikone. Roskienkeruu tarkoittaa sitä, että CLR-virtuaalikone vapauttaa automaattisesti sovelluksen käyttämästä muistista ne osat, joita sovellus ei enää tarvitse. (Yao & Durant 2009, 19 - 26.)

CLR-virtuaalikoneen suorittaman sovelluksen muisti voidaan jakaa kolmeen eri osaan. Nämä osat ovat metatieto, natiivikoodimuisti ja kekomuisti. Metatieto-muisti sisältää metatiedot jokaisesta ohjelmakoodissa määritellystä luokasta ja muuttujatyypistä. Natiivikoodimuisti sisältää suoritusaikana MSIL-koodista natiiviksi koodiksi käännetyn koodin. Kekomuisti sisältää luotujen objektien tiedot. (Yao & Durant 2009, 19 - 26.)

Käännösaikana .NET-kääntäjä luo sovellustiedostoon metatiedot jokaisesta ohjelmakoodissa käytetystä luokasta. Suoritusaikana, kun CLR-virtuaalikone kohtaa ohjelmakoodissa luokkamäärityksen se luo luokasta metatietojen perusteella kuvauksen metatieto-muistiin. Metatieto-muistissa olevan luokkakuvauksen perusteella CLR varaa myöhemmin tarvittavan määrän kekomuistia luokasta luodulle objektille. Tämän jälkeen, kun kyseisen objektin jotain metodia kutsutaan, CLR-virtuaalikoneen sisältämä täsmäkääntäjä kääntää metodin MSIL-ohjelmakoodin natiiviksi ohjelmakoodiksi ja sijoittaa sen natiivikoodimuistiin. (Yao & Durant 2009, 19 - 26.)

Roskienkerääjä huolehtii siitä, että sovelluksen muistinkäyttö pysyy asetetuissa rajoissa. Roskienkerääjällä pyrkii saavuttamaan tämän tavoitteen seuraavilla tavoilla. Ensimmäiseksi roskienkerääjä etsii ja poistaa kekomuistista sellaisia objekteja, joita sovellus ei enää käytä. Toiseksi roskienkerääjä järjestää kekomuistia, lisätilan saavuttamiseksi. Kolmanneksi roskienkerääjä vapauttaa muistia natiivikoodimuistista, tätä kuitenkin vältetään viimeiseen asti. Naiivikoodimuistin vapauttamista vältetään viimeiseen asti, koska sillä on negatiivinen vaikutus sovelluksen suorituskykyyn. (Yao & Durant 2009, 19 - 26.)

5 VIIVAKOODINLUKUJÄRJESTELMÄ

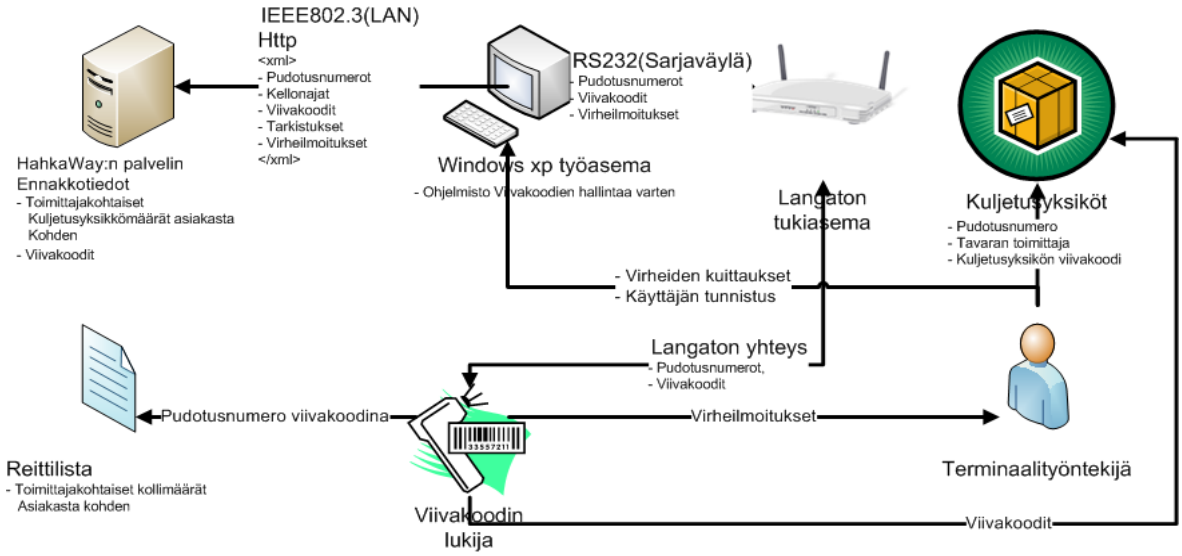
Viivakoodinlukujärjestelmä on HahkaWay Oy:llä käytössä oleva tiedonkeruujärjestelmä, minkä tuottamaa informaatiota käytetään kuljetusyksiköiden seurantaan.

5.1 Järjestelmän tarkoitus

Viivakoodinlukujärjestelmän tehtävä HahkaWay Oy:n terminaalissa on kuljetusyksiköiden tietojen kerääminen ja siirto palvelimelle. Järjestelmän käyttäjänä toimivat terminaalityöntekijät. Jokaisesta kuljetusyksiköstä kerätään kuljetusyksikön viivakoodi, asiakkaan tunniste eli pudotusnumero, kellonaika, käyttäjä ja mahdollinen lavaviivakoodi. Näitä tietoja käytetään hyväksi esimerkiksi todellisten kuljetusyksikkömäärien laskennassa.

5.2 Vanhan järjestelmän rakenne

Ensimmäisenä selvitettiin vanhan viivakoodinlukujärjestelmän rakenne, joka selviää kuvasta 6.



Kuva 6. Vanhan järjestelmän rakenne

Vanha järjestelmä koostuu työasemasta, langattomasta tukiasemasta, reittilistasta ja tietystä määrästä laser-viivakoodinlukijoita.

Työasema sisältää viivakoodienhallintaan käytettävän ohjelmiston, joka hoitaa viivakoodien, kellonaikojen ja pudotusnumeroiden väliaikaisen varastoinnin. Tämän lisäksi viivakoodienhallintaohjelmisto hoitaa näiden tietojen vastaanottamisen RS232-sarjaväylästä, tietojen muuntamisen xml-muotoon ja näiden xml-tiedostojen lähettämisen palvelimelle. Ohjelmistoa käytetään myös mahdollisissa virhetilanteissa virheiden kuittaamiseen.

Langaton tukiasema hoitaa viivakoodien ja pudotusnumeroiden vastaanottamisen viivakoodinlukijoilta ja niiden välittämisen RS232-sarjaväylää pitkin työasemalle. Viivakoodinlukijat välittävät tiedot luetuista viivakoodeista langattomalle tukiasemalle ja ilmoittavat terminaalintyöntekijöille mahdollisista virheistä.

Reittilistan tehtävä on kertoa terminaalityöntekijälle toimittajakohtaiset kuljetusyksikkömäärät kaupoittain. Reittilistalla on myös kaupan pudotusnumero viivakoodimuodossa. Kuvasta 7 selviää reittilistan rakenne.

Reittilista pvm 10.2 reitti 121 Auto HW 40

Pudotus numero	Asiakas nimi	M = Muovi/Liha, E = Eines, P = Muu/Pahvi			Yhteensä		Muilla reiteillä
1210410 MA;TI;KE;TO;PE	M-MARKET TERVAJOKI	Saarioinen	5 E	19 KG 1 #	**9** M **6** E	**218** KG 8 #	
		Atria Oyj		127 KG 1 #			
		Järvi Suomen Portti	1 M	10 KG 1 #			
		HK-Ruokatalo	3 M	20 KG 1 #			
		Oy Snellman Ab	3 M	19 KG 1 #			664 Tuko
		Pouttu Oy	1 M	12 KG 1 #			
		Lihajaloste Korpela	1 M	8 KG 1 #			
		Apetit Kala Oy	1 E	3 KG 1 #			
1210417 MA;TI;KE;TO;	ISOKYRÖN KOULUKESKUS	Saarioinen	1 E	12 KG 1 #	**1** E	**58** KG 2 #	321 Kesko/Pakkanen
		Atria Oyj		46 KG 1 #			
1210420 KE	RKOYJ ISOKYRÖX	Atria Oyj		7 KG 1 #		**7** KG 1 #	
1210426 KE;TO;PE;	ISONKYRÖN VANHAINKOTI	Atria Oyj		73 KG 1 #		**73** KG 1 #	221 Ruokakesko 321 Kesko/Pakkanen
1210429 KE	MÄKYNEN VELJEKSET	Atria Oyj		9 KG 1 #		**9** KG 1 #	

Kuva 7. Reittilista (HahkaWay 2009.)

Reittilistalla näkyy vasemmassa reunassa kaupan pudotusnumero, sen oikealla puolella näkyy kaupan nimi. Kaupan nimen alapuolella näkyy pudotusnumero vii-

vakoodimuodossa ja sen oikealla puolella näkyvät toimittajakohtaiset kuljetusyksikkömäärät.

5.3 Uuden järjestelmän vaatimusten määrittely

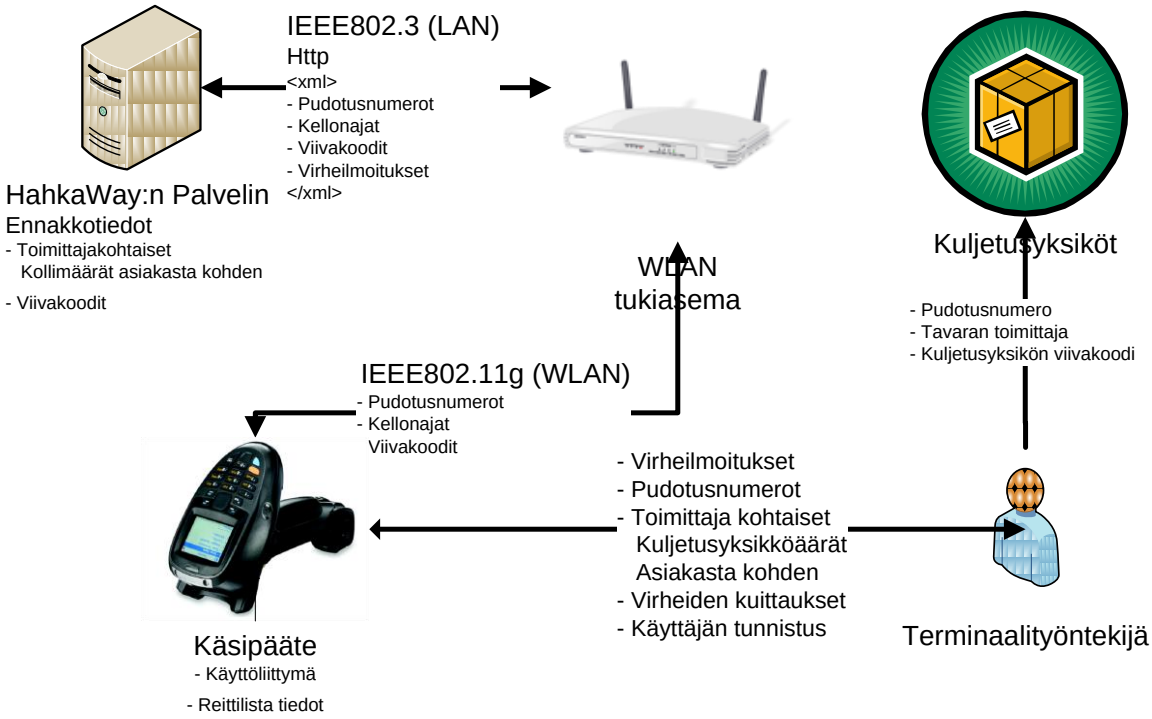
Vaatimusten määrittely aloitettiin tutustumalla terminaalityöntekijöiden tekemään työhön. Seurannassa keskityttiin erityisesti viivakoodien lukemiseen. Tutustumisen tuloksena päädyttiin siihen, että viivakoodienluku on pieni mutta tärkeä osa terminaalityöntekijöiden työtä varsinkin virheselvittelyn kannalta.

Tutustumisen jälkeen kyseltiin mielipiteitä uuteen järjestelmään tarvittavista muutoksista. Ensimmäinen muutosehdotus oli, että kaupan pudotusnumeron reitti- ja pudotusosa syötetään tulevaisuudessa suoraan viivakoodinlukijaan numeronäppäimillä. Toinen uudistamisvaatimus oli, että terminaalitiloista poistetaan työasemat. Työasemien poisto terminaalitiloista tarkoitti sitä, että virheet täytyy pystyä tulevaisuudessa kuittaamaan suoraan viivakoodinlukijasta. Kolmas uudistamisehdotus oli, että tällä hetkellä reittilistalla näkyvät toimittajakohtaiset kuljetusyksikkömäärät näkyvät uudessa järjestelmässä viivakoodinlukijan näytöllä.

5.4 Uuden järjestelmän rakenne

Järjestelmän vaatimusten perusteella suunniteltiin uuden viivakoodinlukujärjestelmän rakenne, siten että se kattoi mahdollisimman hyvin vaatimukset.

Uudessa järjestelmässä viivakoodinlukijat ja työasemat korvattiin käsipäätteillä, joihin suunniteltiin ohjelmisto. Käsipäätteet siirtävät tiedot kuljetusyksiköistä HahkaWay Oy:n palvelimelle WLAN-yhteyden avulla.



Kuva 8. Uuden järjestelmän rakenne

Kuvasta 8 selviää, että uusi viivakoodinlukujärjestelmä on rakenteeltaan yksinkertaisempi kuin vanha. Tämä saatiin aikaan sillä, että työasemat ja viivakoodinlukijat korvattiin käsipäätteillä. Työasemien poisto terminaalitiloista aiheutti sen, että käsipäätteisiin täytyi suunnitella ja toteuttaa ohjelmisto, joka siirtää tiedot kuljetusyksiköistä HahkaWay Oy:n palvelimelle ja ilmoittaa virheistä käyttäjälle. Tämän ohjelmiston suunnittelua ja toteutusta käsitellään luvussa 7.

Uuden järjestelmän määrittelyn jälkeen siirryttiin määrittämään käsipäätteen vaatimuksia ja etsimään kyseisiä vaatimuksia täyttäviä käsipäätettä.

6 KÄSIPÄÄTE

Tässä luvussa kerrotaan uuteen viivakoodinlukujärjestelmään valitun käsipäätteen valintavaiheet ja kyseisen käsipäätteen ominaisuudet.

6.1 Vaatimukset

Käsipäätteen vaatimukset määriteltiin osaksi järjestelmän vaatimusten perusteella ja osaksi vanhan järjestelmän rakenteen perusteella. Vaatimusten määrittelyn tuloksena päädyttiin siihen, että käsipäätteessä täytyy olla seuraavat ominaisuudet:

- laser-viivakoodinlukija
- näyttö
- numeronäppäimet
- tietoliikenneyhteydet WLAN
- IP-luokitus vähintään IP54
- laitteen muoto pistoolimainen
- hyvät ohjelmistokehitysmahdollisuudet.

IP-luokitus määrittää minkä tahansa sähköisen laitteen koteloinnin tiiviyden, eli kuinka hyvin elektroninen laite on suojattu vedeltä ja pölyltä. IP-luokkanumero koostuu kahdesta numerosta, joista ensimmäinen määrää laitteen suojauksen pölyä vastaan ja toinen numero suojauksen vettä vastaan. Mitä suuremmat numerot ovat, sen paremmin laite on suojattu. (Taloon.com 2010.)

Käsipäätteen valinta suoritettiin pääasiassa internetistä valmistajien sivulta. Valmistajien sivuilta etsittiin vaatimuksia täyttäviä laitteita, joita sitten pyydettiin testattavaksi. Ensimmäinen laite, joka vastaisi määrittelyn asettamia vaatimuksia, oli Motorolan valmistama MT2090-käsipääte. Tästä käsipäätteestä hankittiin testattavaksi kameraviivakoodinlukijalla varustettu versio, koska maahantuojalla ei ollut

varastossa laser-viivakoodinlukijalla varustettua versiota. Kuvassa 9 näkyy MT2090-käsipääte.

Kuva 9. Motorola MT2090-käsipääte (Motorola 2009.)

Käsipäätteen ominaisuudet olivat seuraavat:

- kameraviivakoodinlukija
- 320 pikseliä korkea ja 240 pikseliä leveä värinäyttö
- tietoliikenneyhteydet WLAN ja Bluetooth
- IP-luokka IP54
- käyttöjärjestelmä Windows CE 5.0. (Motorola 2009.)

Käsipäätteen tarkemmat tekniset tiedot löytyvät **liitteestä 1.**

6.2 Testaus

Käsipäätteen testaus suoritettiin testaamalla sen viivakoodinlukunopeutta, yleistä käyttönopeutta ja tietoliikenneyhteyksien toimivuutta. Tämän lisäksi terminaalityöntekijöiltä kyseltiin mielipiteitä laitteesta. Testauksen päätyttyä päädyttiin seuraaviin tuloksiin.

MT2090-käsipäätteen viivakoodinlukunopeus oli tietenkin liian hidas, johtuen siitä että testattavana olevassa mallissa oli kameraviivakoodinlukija. Laitteen yleinen käyttönopeus oli suhteellisen hyvä ja WLAN-yhteys vaikutti todella hyvältä. Tämän lisäksi laite osoittautui fyysiseltä olemukseltaan ja ohjelmistokehitysmahdollisuuksiltaan sopivaksi. Ainoa huono puoli laitteessa oli, että näyttö oli liian pieni.

Tulosten perusteella päätettiin, että ohjelmistoa lähdetään kehittämään kyseiselle kasipäätteelle pienistä puutteista huolimatta, koska markkinoilla ei ollut muita vastaavia laitteitta. MT2090-käsipäätteestä tilattiin laserviivakoodinlukijalla varustettu versio, johon ohjelmisto suunniteltiin.

7 OHJELMISTO

Luvussa 5.4 mainittujen rakennemuutosten takia täytyi MT2090-käsipäätteeseen suunnitella ja toteuttaa ohjelmisto, joka siirtää tiedot kuljetusyksiköistä HahkaWay Oy:n palvelimelle.

Tässä luvussa käydään ensin läpi, MT2090-käsipäätteeseen suunnitellun ohjelmiston tehtävä. Tämän jälkeen käydään läpi ohjelmistonkehitysvaiheet ja ohjelmiston toiminta opinnäytetyön kirjoitushetkellä.

Ohjelmistokehityksen päävaiheet olivat ohjelmiston suunnittelu ja ohjelmiston toteutus. Ohjelmisto kehitettiin Microsoftin C#-ohjelmointikielellä .NET Compact Framework -ympäristöä ja Microsoft Visual Studio 2008 käyttäen.

Näiden työkalujen lisäksi käytettiin Motorolan toimittamaa Enterprise Mobile Development Kit (EMDK) -luokkakirjastoa, joka sisältää laitekohtaiset rajapinnat esimerkiksi käyttöliittymien luomiseen laitteen näytölle.

7.1 Tehtävä

Ohjelmiston tehtävä MT2090-kasipäätteessä määriteltiin järjestelmän vaatimusten perusteella.

Ohjelmiston tehtävänä on kerätä jokaisesta luetusta kuljetusyksiköstä seuraavat tiedot:

- kuljetusyksikön viivakoodi
- lavan viivakoodi
- asiakkaan pudotusnumero (reitti- ja pudotusosa)
- kellonaika, koska kuljetusyksikön viivakoodi on luettu
- käyttäjän nimi, kuka on kuljetusyksikön lukenut.

Näiden tietojen keräämisen jälkeen ohjelmisto muuntaa tiedot xml-muotoon ja lähettää tiedot HahkaWay Oy:n palvelimelle. Lähetyksen jälkeen ohjelmisto vastaanottaa synkronisesti xml-muotoisen vastauksen luettujen tietojen oikeellisuudesta ja ilmoittaa virheistä käyttäjälle. Virhe tapahtuu silloin, kun luetun viivakoodin pudotusnumero ei vastaa palvelimella olevaa tietoa, jolloin ohjelmisto pyytää käyttäjää tarkastamaan tiedon oikeellisuuden ja kuittaamaan, mitä kuljetusyksikölle tehdään.

7.2 Suunnittelu

Ohjelmiston suunnittelun aikana selvitettiin ohjelmiston tietoliikenneliittymä HahkaWay Oy:n palvelimeen, suunniteltiin käyttöliittymä sekä ohjelmiston rakenne.

7.2.1 Tietoliikenneliittymä

Tietoliikenneliittymänä päädyttiin käyttämään samaa liittymää, mitä vanha järjestelmä käytti. Liittymän toiminta oli seuraavanlainen: Xml-tiedosto, joka sisältää luvussa 7.1 mainitut tiedot, lähetetään http-protokollaa käyttäen palvelimelle. Palvelin käsittelee tiedon ja muuntaa kenttien ”ProvinetStatus”, ”Customer” ja ”CustomerName” arvot. Palvelin vastaa samanmuotoisella xml-tiedostolla ohjelmistolle. Vastaanotetun xml-dokumentin ProvinetStatus-kentän arvosta ohjelma päättelee onko virheitä tapahtunut. Jos ProvinetStatus-kentän arvo on eri kuin OK, niin silloin jokin lähetetystä kuljetusyksikön tiedosta on virheellinen ja ProvinetStatus-kenttä sisältää palvelimen antaman virheilmoituksen. Kuvasta 10 selviää xml-dokumentin muoto.

```
<?xml version="1.0" standalone="yes" ?>
- <File>
  - <File>
      <User>Käyttäjä1</User>
      <Customer>1130212</Customer>
      <Box>900160298228</Box>
      <Pallet>214105</Pallet>
      <DateTime>2010211134656</DateTime>
      <AreYouSure>0</AreYouSure>
      <ProvinetStatus />
      <CustomerName />
    </File>
  - <File>
      <User>Käyttäjä1</User>
      <Customer>1130212</Customer>
      <Box>900166711475</Box>
      <Pallet>214105</Pallet>
```

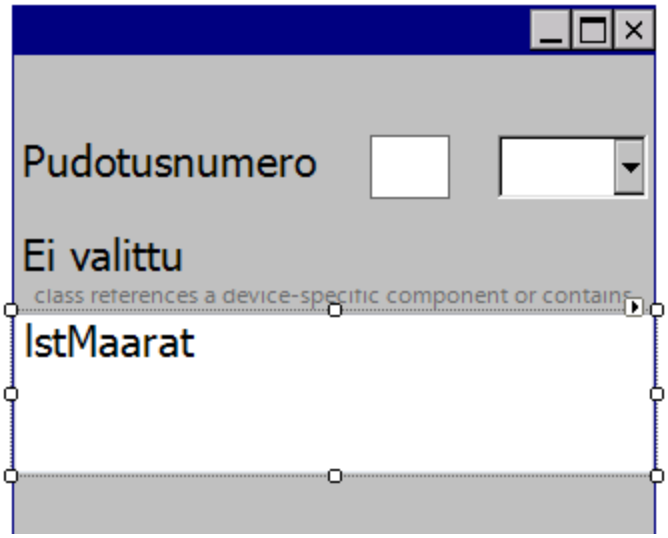
Kuva 10. Xml-dokumentin muoto

Tietoliikenneliittymän suunnittelun jälkeen siirryttiin suunnittelemaan käyttöliittymää.

7.2.2 Käyttöliittymä ja sisäiset tietorakenteet

Käyttöliittymä suunniteltiin järjestelmän vaatimusten perusteella niin, että siitä näkyy toimittajakohtaiset kuljetusyksikkömäärät asiakasta kohden ja asiakkaan nimi. Tämän lisäksi käyttöliittymään pystyy syöttämään kaupan pudotusnumeron niin että se on jaoteltu reitti- ja pudotusosaan.

Käyttöliittymä suunniteltiin käyttäen Microsoft Visual Studio 2008 -ohjelman Form-designer-osaa. Käyttöliittymän ulkoasu muodostui suunnitteluvaiheessa kuvan 11 näköiseksi.



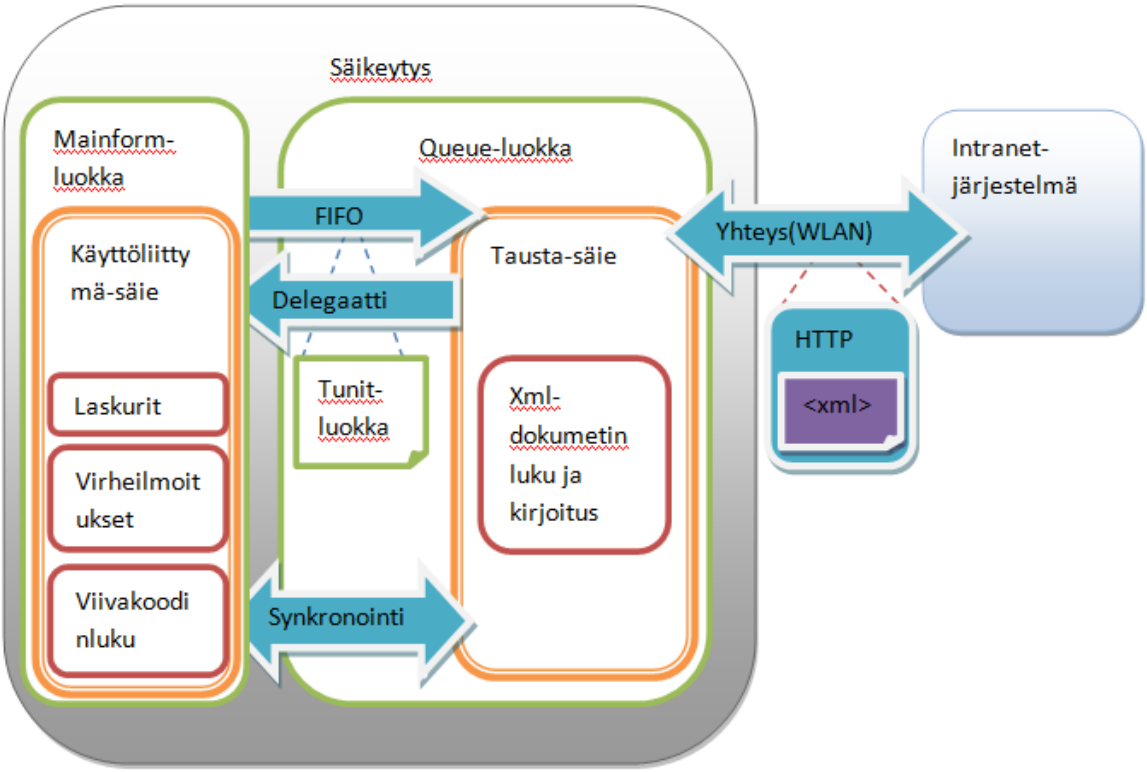
Kuva 11. Käyttöliittymän pääikkuna suunnitteluvaiheessa

Kuvassa 11 näkyy käyttöliittymän niin kuin se Visual Studio 2008 -ohjelman Formdesigner-osassa näkyi. Tämä kuva ei vastannut lopullista käyttöliittymää, koska Formdesigner-osa ei osannut esittää EMDK-luokkakirjaston tuomia graafisia ominaisuuksia. Käyttöliittymässä näkyy pudotusnumero, joka on jaettu kahteen osaan, reittiosaan ja pudotusosaan. Pudotusnumeron alapuolella näkyy kyseistä pudotusnumeroa vastaavan kaupan nimi. Kaupan nimen alla näkyy listamaisesti toimittajakohtaiset kuljetusyksikkömäärät ja niiden tyypit. Käyttöliittymän suunnittelun jälkeen siirryttiin suunnittelemaan tietorakennetta kuljetusyksiköiden tietojen varastointiin.

Kuljetusyksikön tietojen varastointiin laitteen muistiin suunniteltiin Tunit-luokka, joka sisälsi luvussa 7.1 määritellyt tiedot. Käyttöliittymän, tietoliikenneliittymän ja Tunit-luokan suunnittelun jälkeen todettiin, että tietoliikenneliittymän aiheuttavat odotukset eivät saa näkyä käyttöliittymän toiminnassa. Tämän ongelman ratkaisemiseksi päätettiin, että käyttöliittymää ja tietoliikenneliittymää suoritetaan omissa säikeissään. Tämän jälkeen alettiin suunnittelemaan käyttöliittymän ja tietoliikenneliittymän rinnakkaisia toimintoja.

7.2.3 Rinnakkaiset toiminnat

Ohjelmiston toiminta jaettiin kahdelle eri säikeelle. Näiden säikeiden nimet olivat käyttöliittymä- ja taustasäie. Käyttöliittymäsäie vastaa käyttöliittymän palvelemisesta ja viivakoodien vastaanotosta käsipäätteen viivakoodinlukijalta. Taustasäie vastaa kuljetusyksiköiden tietojen muunnoksista xml-dokumentiksi ja xml-dokumentin siirrosta palvelimen ja ohjelmiston välillä. Ohjelmiston toiminnan säilyttämiseksi käyttöliittymä- ja taustasäikeen täytyy pystyä kommunikoimaan toistensa kanssa ja käyttöliittymän täytyy pystyä välittämään kuljetusyksiköiden tiedot taustatäikeelle. Käyttöliittymän- ja taustasäikeen väliseen tiedonvälitykseen suunniteltiin Queue-luokka. Queue-luokan tehtävä on kaksisuuntaisen tiedonvälityksen hoitaminen käyttöliittymä- ja taustasäikeen välillä. Kuvassa 12 näkyy ohjelmiston säikeytyksen rakenne.



Kuva 12. Ohjelmiston säikeet

Kuvasta 12 selviää säikeiden tehtävät ja se, että miten niiden välinen kommunikointi on toteutettu. Käyttöliittymäsäikeen tehtävät ovat:

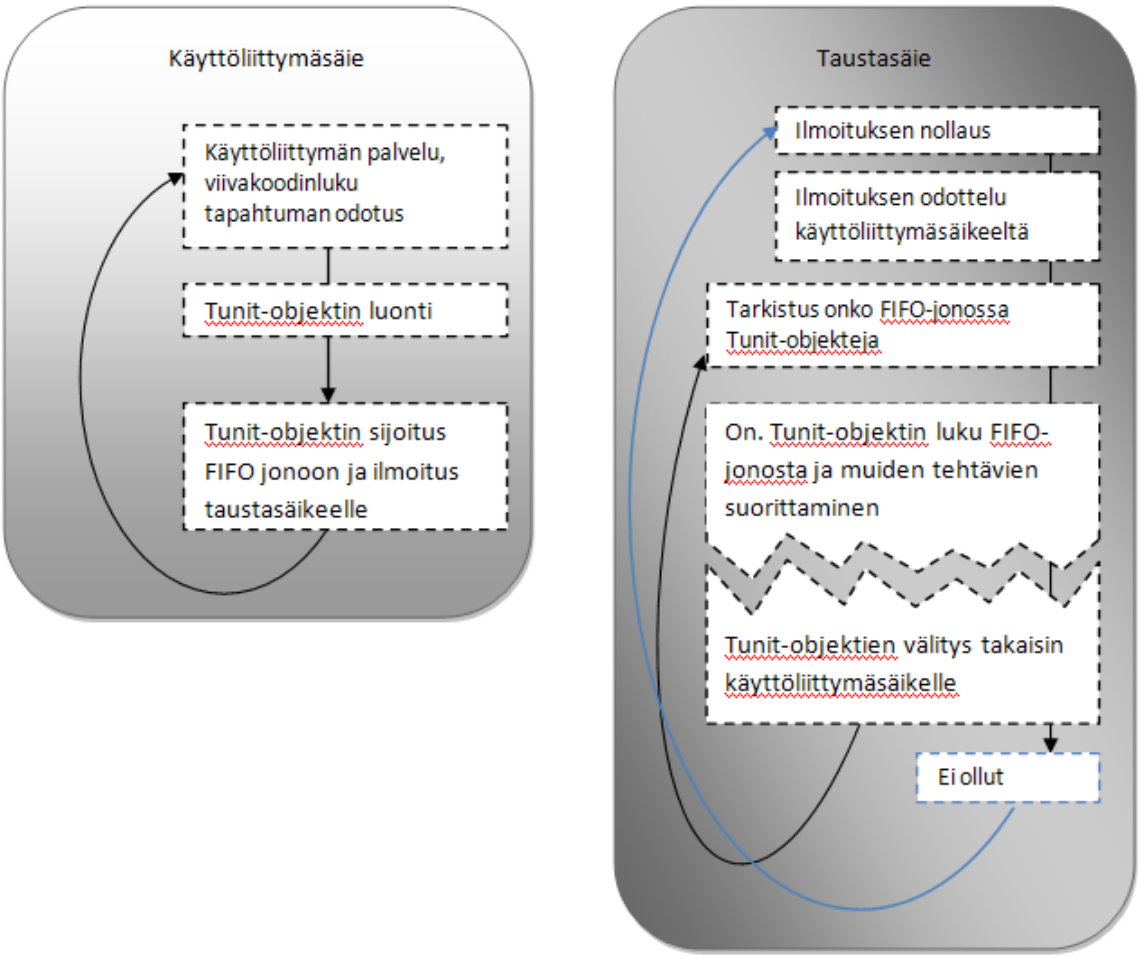
- Palvella käyttöliittymää.
- Vastaanottaa viivakoodit käsipäätteen viivakoodinlukijalta.
- Luoda Tunit-objektit ja tallettaa niihin viivakoodit, pudotusnumerot ja lavanumerot, sekä välittää kyseiset Tunit-objektit taustasäikeelle First In First Out (FIFO) -jonorakenteen avulla.
- Käsitellä taustasäikeeltä takaisin saadut Tunit-objektit.
- Ilmoittaa virheistä käyttäjälle ja päivittää kuljetusyksikkölaskurit taustasäikeeltä vastaanotettujen Tunit-objektien perusteella.

Taustasäikeen tehtävät ovat:

- Purkaa FIFO-jonorakennetta.
- Muuntaa puretut Tunit-objektit xml-muotoon.
- Välittää xml-muotoinen tieto palvelimelle ja vastaanottaa vastaus palvelimelta.
- Muuntaa vastaanotettu xml-dokumentti takaisin Tunit-objekteiksi.
- Välittää Tunit-objektit käyttöliittymälle delegaatin avulla.

7.2.4 Synkronointi

Käyttöliittymäsäikeen ja taustasäikeen välinen toiminta synkronoitiin. Synkronoinnilla taattiin, että tausta- ja käyttöliittymäsäikeen välinen tiedonvälitys on turvallista. Eli taattiin se, että käyttöliittymäsäie on varmasti lopettanut Tunit-objektin kirjoituksen FIFO-jonoon ennen kuin taustasäie aloittaa lukemaa Tunit-objektia jonosta. Synkronoinnilla taattiin myös se, että taustasäie ei pollaa FIFO-jonoa turhaan. Kuvasta 13 selviää synkronoinnin suunniteltu toiminta.



Kuva 13. Synkronoinnin periaatekuva

Kuten kuvasta 13 selviää taustasäie tarkastaa vielä ilmoituksen saatuaan, onko FIFO-jonossa oikeasti Tunit-objekteja. Tämän jälkeen taustasäie purkaa FIFO-jonosta yhden Tunin-objektin kerrallaan ja prosessoi sen. Tätä toimintaa jatketaan niin kauan, kunnes FIFO-jono on tyhjä, jonka jälkeen palataan odottelemaan uutta ilmoitusta käyttöliittymäsäikeeltä.

7.3 Toteutus

Seuraavassa neljässä kappaleessa käydään läpi käyttöliittymän ja ohjelmiston sisäisten tietorakenteiden toteutuksen kannalta tärkeimmät luokat. Tämän lisäksi

käydään läpi käyttöliittymä-luokka, Queue-luokka ja Tunit-Luokka sekä tausta- ja käyttöliittymäsäikeen synkronoinnin toteutus.

7.3.1 Käyttöliittymäluokka

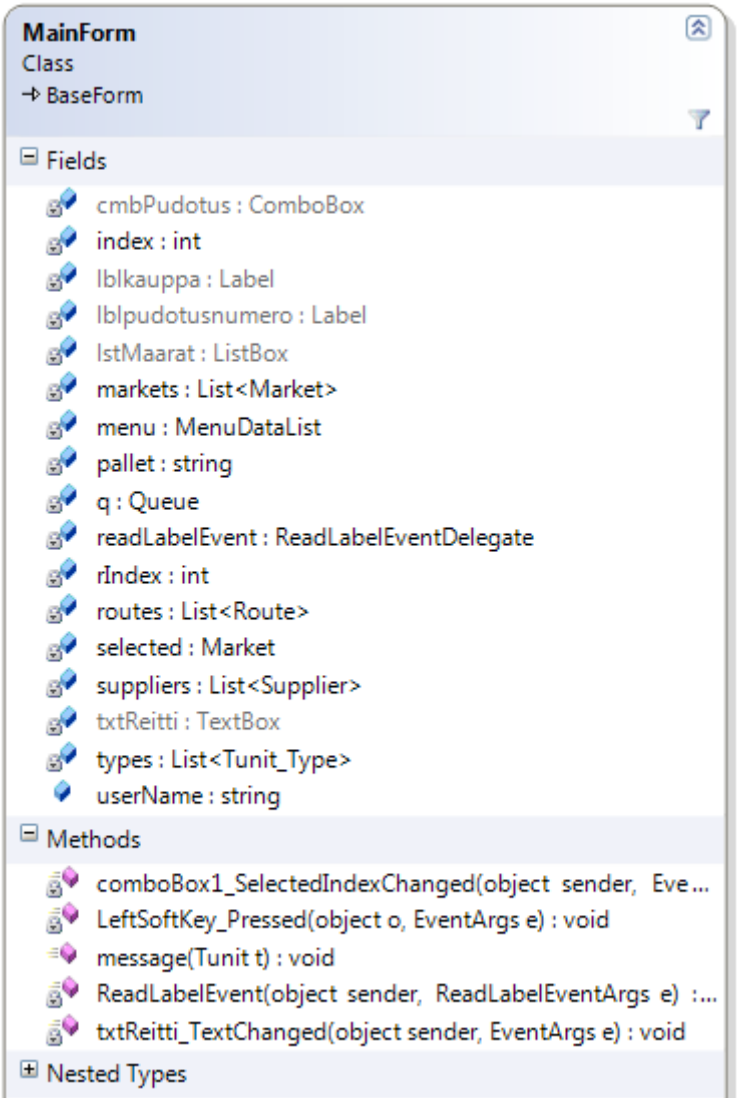
Käyttöliittymän toteutuksen kannalta tärkeimmät luokat, jotka löytyivät EMDK-luokkakirjaston Symbol.MT2000-nimiavaruudesta, olivat:

- QuestionForm-luokka sisältää tarvittavan toiminnallisuuden kysymyslomakkeen näyttämiseen MT2090-käsipäätteen näytöllä.
- MsgBox-luokka sisältää tarvittavat toiminnallisuuden ilmoituslomakkeet näyttämiseen käsipäätteen näytöllä.
- BaseForm-luokka sisältää tarvittavan toiminnallisuuden pääikkunan näyttämiseen laitteen näytöllä.

Näiden luokkien lisäksi tarvittiin myös seuraavat luokat .NET Compact Framework -luokkakirjastosta:

- TextBox-luokka sisältää tarvittavat toiminnallisuuden tekstikentän lisäämiseen käyttöliittymään ja kyseisen tekstikentän hallinnointiin.
- ComboBox-luokka sisältää tarvittaman toiminnallisuuden alasvetovalikon lisäämiseen käyttöliittymään.
- ListBox-luokka sisältää tarvittaman toiminnallisuuden listan lisäämiseen käyttöliittymään.

Käyttöliittymä toteutettiin käyttäen kyseisiä luokkia, käyttöliittymän pääikkunan luokkakaavio näkyy kuvasta 14.



Kuva 14. Käyttöliittymäluokka

Käyttöliittymäluokka periytettiin EMDK-luokkakirjastosta löytyvästä BaseForm-luokasta. Periyttämisellä saatiin käyttöön EMDK-luokkakirjaston sisältämät graafiset ominaisuudet ja SoftKey-näppäimien käyttömahdollisuus.

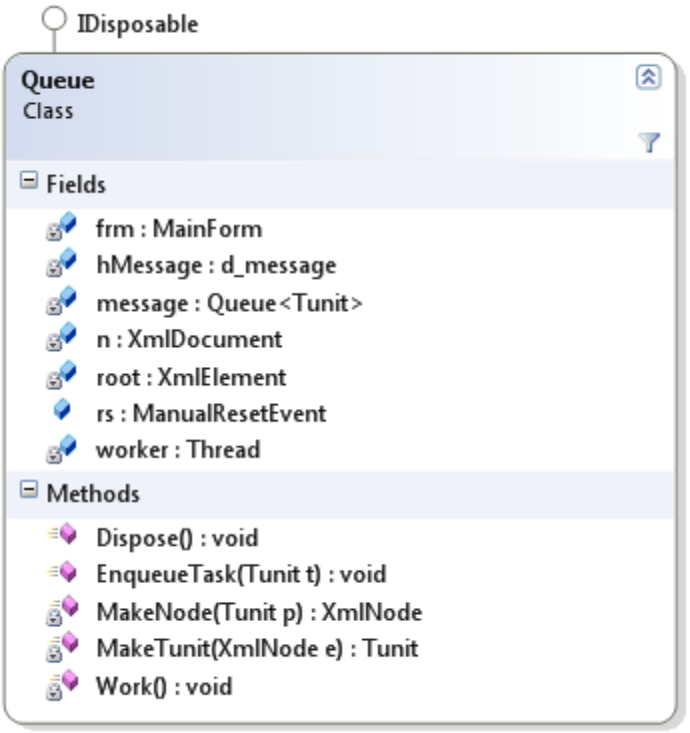
7.3.2 Queue-luokka

Queue-luokan toteutuksen kannalta tärkeät luokat, jotka löytyvät .NET Compact Framework -luokkakirjaston System-nimiavaruudesta olivat seuraavat:

- Threading.Thread-luokka sisältää tarvittavat metodit säikeiden luomiseen tuhoamiseen ja hallintaan.

- Xml.XmlDocument-luokka sisältää tarvittavat metodit Xml-mutoisen tiedon kirjoittamiseen ja lukemiseen.
- Threading.ManualResetEvent-luokka sisältää tarvittavat metodit säikeiden väliseen synkronointiin.
- Collections.Generic.Queue-luokka sisältää tarvittavat metodit FIFO-jonon käsittelyyn.

Queue-luokka toteutettiin käyttäen kyseisiä luokkia ja sen luokkakavio muodostui kuvan 15 mukaiseksi.



Kuva 15. Queue-luokka

Queue-luokka sisältää tarvittavat metodit käyttöliittymä- ja taustasäikeen väliseen kommunikointiin. Tämän lisäksi se sisältää taustasäikeen toiminnallisuuden. Queue-luokan metodien käyttötarkoitukset ovat seuraavat:

- EnqueueTask-metodia käytetään Tunit-objektin välittämiseen käyttöliittymäsäikeeltä taustasäikeelle.
- MakeNode-metodia käytetään Tunit-objektin muuntamiseen xml-muotoon.

- MakeTunit-metodia käytetään vastaanotetun xml-muotoisen tiedon muuntamiseen Tunit-objektiksi.
- Work-metodi sisältää taustasäikeen toiminnallisuuden. Taustasäie suorittaa Work-metodia niin kauan, kunnes ohjelma sammutetaan.

7.3.3 Tunit-luokka

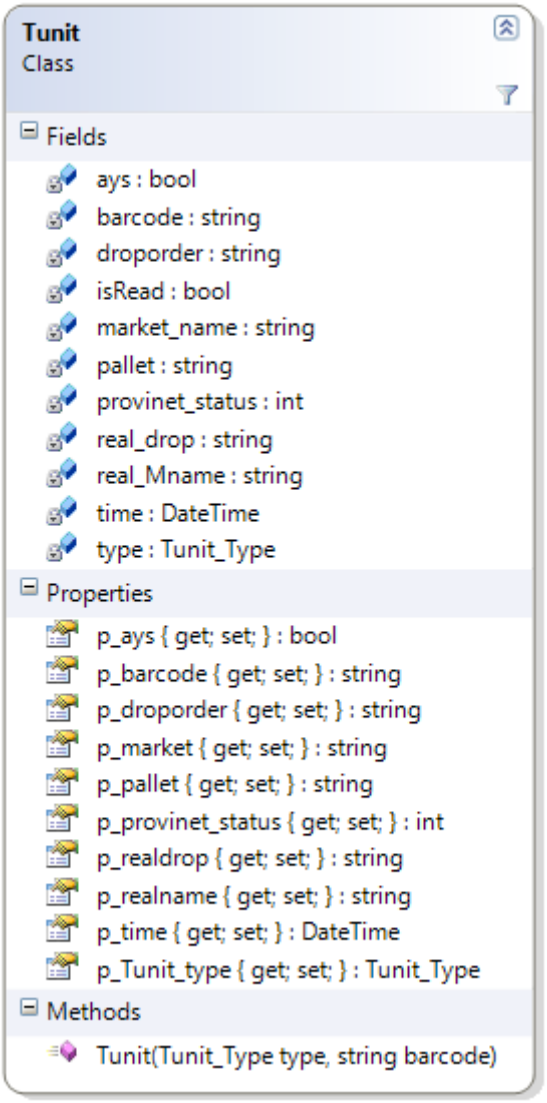
Tunit-luokan toteuttamisen kannalta tärkeimmät luokat, jotka löytyivät .NET compact Framework -luokkakirjaston System-nimiavaruudesta, olivat:

- DateTime-luokka sisältää tarvittavat metodit ajan esittämiseen ja tallentamiseen laitteen muistiin.
- string-luokka sisältää tarvittavat metodit merkkijonon käsittelyyn.

Luokkien lisäksi tarvittiin .NET Compact Framework-luokkakirjaston perustietotyypeistä seuraavat tietotyypit:

- int-tietotyyppi vastaa 32-bittistä kokonaislukua.
- bool-tietotyyppi vastaa totuusarvoa, voi olla ”true” tai ”false”.

Tunit-luokka toteutettiin kyseisiä luokkia ja tietotyyppejä käyttäen ja sen luokkakaavio muodostui kuvan 16 mukaiseksi.



Kuva 16. Tunit-luokka

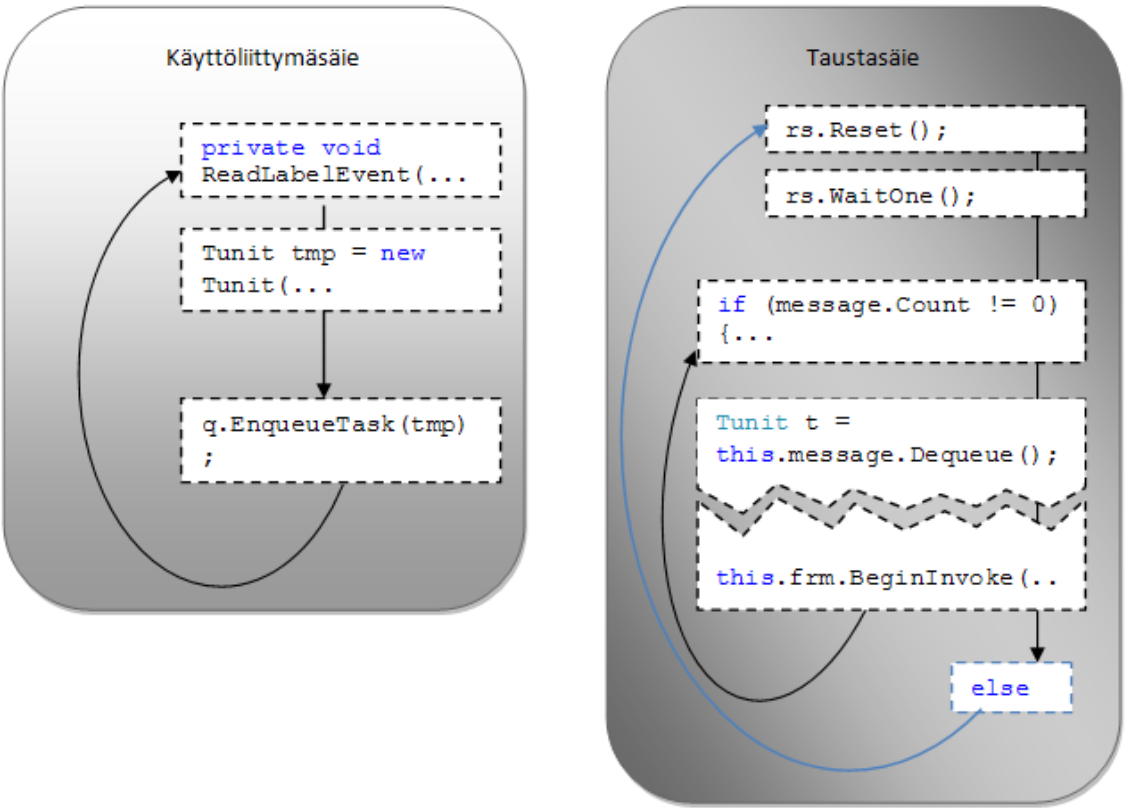
Tunit-luokasta luotuihin objekteihin tallennetaan kuljetusyksiköiden tiedot.

7.3.4 Synkronointi

Käyttöliittymä- ja taustasäikeen välinen synkronointi toteutettiin osaksi käyttöliittymä-luokkaan ja osaksi Queue-luokkaan.

Synkronointi toteutettiin käyttäen .NET Framework -luokkakirjaston ManualResetEvent-luokkaa. ManualResetEvent-luokka sisältää toiminnallisuuden kahden säikeen väliseen synkronointiin.

ManualResetEvent-objektilla voi olla kaksi eri tilaa: signaloitu tai ei-signaloitu. ManualResetEvent-objektin tilaa hallinnoidaan kahdella metodilla, jotka ovat Set ja Reset. Set-metodilla asetetaan ManualResetEvent-objektin tila signaloiduksi, Reset-metodilla tila asetetaan ei-signaloiduksi. Säie, jonka toimintaa ManualResetEvent-objektilla ohjataan, kutsuu WaitOne-metodia ManualResetEvent-objektilta. Riippuen ManualResetEvent-objektin tilasta säie saa jatkaa toimintaansa tai se joutuu pysähtymään. ManualresetEvent-objektin tilan ollessa ei-signaloitu joutuu säie pysähtymään odottamaan, kunnes ManualResetEvent-objektin tila muuttuu signaloiduksi. Käyttöliittymä- ja taustasäikeen välinen synkronointi toteutettiin kuvan 13 mukaiseksi ja se muodostui seuraavanlaiseksi.



Kuva 17. Synkronoinnin toteutusperiaate

Kuva 17 esittää ohjelmakoodissa olevat funktiot jotka toteuttavat kuvassa 13 kuvatut toiminnallisuudet. ManualResetEvent-objektin Set-metodin kutsua ei näy kuvassa, koska sen kutsu on sisällytetty EnqueueTask-metodiin. EnqueueTask-metodi myös lukitsee FIFO-jonon ennen Tunit-objektin kirjoitusta jonoon, taatakseen kirjoituksen valmistumisen ennen kuin taustasäie lukee Tunit-objektin jonosta.

7.4 Ohjelmiston toiminta

Ohjelmiston tilanne tämän opinnäytetyön kirjoitushetkellä oli seuraavanlainen: Ohjelmistolla pystyi tunnistamaan käyttäjän, lukemaan viivakoodeja ja liittämään niitä lavakoodiin. Ohjelmisto tekee oikeanmuotoisen xml-dokumentin viivakoodeista ja tallentaa sen päivänmäärän nimellä MT2090-käsipäätteen flash-muistille. Vastaanotetun xml-tiedoston muuntaminen Tunit-objekteiksi on tehty. Puuttuvia ominaisuuksia on vielä xml-tiedoston siirto http-protokollaa käyttäen palvelimen ja ohjelmiston välillä.

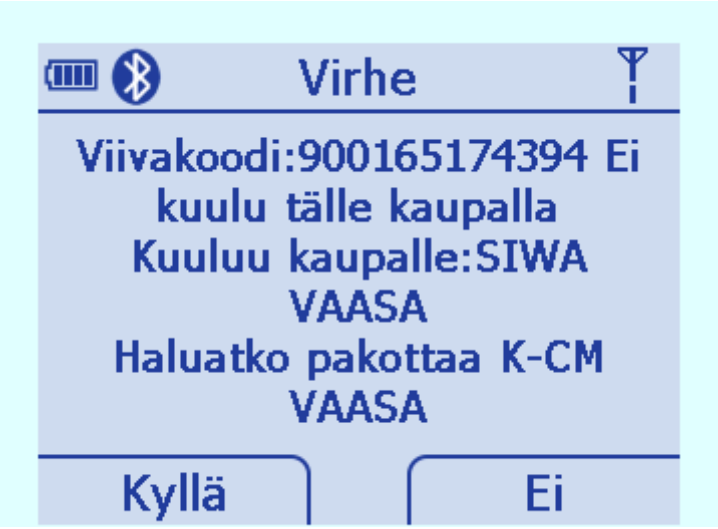
Kuva 18. MT2090-viivakoodinlukuohjelmiston käyttäjän valinta

Ohjelman käynnistymisen jälkeen se pyytää valitsemaan käyttäjän. Käyttäjä valitsee haluamansa käyttäjän ja painaa ”Kirjaudu”-näppäintä, tämän jälkeen ohjelma siirtyy pääikkunaan, missä itse viivakoodienluku tapahtuu.

Kuva 19. MT2090-viivakoodinlukuohjelmiston käyttöliittymän pääikkuna

Kuvassa 19 näkyy ohjelmiston pääikkunan käyttöliittymä. Käyttöliittymän toiminta on seuraavanlainen: Käyttäjä syöttää haluamansa reitin laitteen numeronäppäimiä käyttäen, jonka jälkeen ohjelma listaa kaikki kyseisellä reitillä olevat pudotusnumerot oikeassa reunassa olevaan alasvetovalikkoon ja näyttää kaupan nimen ”Pudotusnumero”-tekstin alapuolella. Kaupan nimen alapuolella näkyy toimittajakohtaiset kuljetusyksikkömäärät kyseiselle kaupalle ja ne päivittyvät sitä mukaa kun käyttäjä lukee viivakoodeja. Lavan viivakoodi näkyy näytön oikeassa reunassa, jos viivakoodeja liitetään lavaan.

Virheellisen viivakoodinluvun tapahtuessa, eli kun viivakoodi ja pudotusnumero eivät vastaa palvelimella olevaa tietoa, ohjelmisto antaa seuraavanlaisen virheilmoituksen, joka näkyy kuvassa 20.



Kuva 20. MT2090-viivakoodinlukuohjelmiston virheilmoitus

Käyttäjä tarkastaa tiedon oikeellisuuden ja valitsee kyllä tai ei. Kun käyttäjä valitsee kyllä, ohjelmisto pakottaa kuljetusyksikön valitulle kaupalle ja lähettää tiedot uudestaan palvelimelle. Käyttäjän valitessa ei, ohjelmisto ei lähetä tietoja uudelleen palvelimelle.

8 TULOKSET

Tämän opinnäytetyön tuloksiksi saatiin valmis suunnitelma uudesta viivakoodinlukujärjestelmästä HahkaWay Oy:n terminaalitiloihin ja suunnitelma MT2090-käsipäätteen ohjelmistosta, joka on osaksi toteutettu. Uusi järjestelmä tulee korvaamaan vanhan järjestelmän kokonaan, kunhan se saadaan valmiiksi ja käyttöönotettua. MT2090-käsipäätteen ohjelmistosta puuttuu vielä kuljetusyksiköiden tietojen siirto http-protokollaa käyttäen palvelimelle.

Uudesta järjestelmästä tulee olemaan hyötyä HahkaWay Oy:lle siinä mielessä, että sen käyttöönoton jälkeen kaikki kuljetusyksiköt ehditään lukemaan, koska käsipäätteitä tulee olemaan riittävästi. Lisäksi uusi viivakoodinlukujärjestelmä on RoHS-direktiivin mukainen.

Järjestelmä ei kuitenkaan vielä toteutuksen jälkeenkään tule vastaamaan suunnittelussa määriteltyä järjestelmää, koska reittilistat tullaan jättämään uuteenkin järjestelmään. Tämä tehdään sen takia, koska palvelimen ja ohjelmiston välillä ei saada vielä siirrettyä toimittajakohtaisia kuljetusyksikkömääriä ja vielä ei ole tiedossa tapaa, kuinka ohjelmisto osaisi tunnistaa kuljetusyksikön toimittajan automaattisesti.

8.1 Jatkokehitys

Ohjelmiston kehitystä jatketaan http-yhteyden kehittämisen parissa. Http-yhteyden toimintaan saattamisen jälkeen ohjelmaa aletaan testata vanhan järjestelmän rinnalla yhden käsipäätteen avulla. Testauksen aikana pyritään kitkemään mahdolliset ohjelmointivirheet pois ohjelmasta. Testauksen jälkeen järjestelmä käyttöönotetaan ja vanha järjestelmä puretaan. Tavoitteena on kehittää uutta järjestelmää niin, että tulevaisuudessa terminaalitiloista voitaisiin jättää pois reittilistojen paperiversiot kokonaan.

9 YHTEENVETO

Tässä työssä käsiteltiin uuden viivakoodinlukujärjestelmän suunnittelua ja toteutusta HahkaWay Oy:n terminaalitiloihin vanhan viivakoodinlukujärjestelmän korvaajaksi. Vanha viivakoodinlukujärjestelmä täytyi korvata uudella, koska se oli tullut käyttöikänsä päähän eikä vanha järjestelmä ollut RoHS-direktiivin mukainen.

Aluksi kerrottiin teoriaa Viivakoodeista ja verrattiin niiden ominaisuuksia RFID-tekniikkaan. Tämän jälkeen käytiin läpi .NET Compact Framework -ympäristön perusrakenne, jonka jälkeen siirryttiin käymään läpi itse työvaiheita.

Ensimmäisenä käytiin läpi vanhan järjestelmän toiminta, jonka jälkeen käytiin läpi uudistamistoiveet. Uudistamistoiveet olivat työasemien poisto terminaalitiloista ja pudotusnumeron syöttömahdollisuus numeronäppäimiä käyttäen suoraan viivakoodinlukijaan sekä toimittajakohtaisten kuljetusyksikkömäärien näkyminen viivakoodinlukijan näytöllä.

Tämän jälkeen käytiin läpi uuden järjestelmän suunnittelu, jonka tuloksena päätettiin, että työasemat ja viivakoodinlukijat korvattaisiin käsipäätteillä. Tästä seurasi, että käsipäätteisiin täytyi suunnitella ohjelmisto. Uuden järjestelmän suunnittelun jälkeen valittiin sopiva käsipääte. Käsipäätteen valinnan tuloksina päädyttiin Motorolan valmistamaan MT2090-käsipäätteeseen. Käsipäätteen valinnan jälkeen siirryttiin ohjelmiston suunnittelu- ja kehitysvaiheisiin. Ohjelmiston kehitysvaiheissa käytiin läpi käyttöliittymän, tietoliikenneliittymän ja sisäisten tietorakenteiden suunnittelu ja toteutus.

Lopuksi selvitettiin suunnitellun ohjelmiston toiminta opinnäytetyön kirjoitushetkellä ja tulokset sekä jatkokehityssuunnitelmat. Ohjelmiston toiminta opinnäytetyön kirjoitushetkellä vastasi muuten suunnittelussa määriteltyä toimintaa, mutta siitä puuttui vielä tietoliikenneliittymä HahkaWay Oy:n palvelimelle.

Tuloksissa selvitettiin, että ilman muutoksia HahkaWay Oy:n palvelimen ohjelmistoon, todellisia toimittajakohtaisia kuljetusyksikkömääriä ei saada näkyviin käsipäätteen näytöllä. Tuloksissa käytiin läpi myös uuden järjestelmän tuomat hyödyt. Näitä hyötyjä olivat, että uusi järjestelmä on RoHS-direktiivin mukainen ja että uuden järjestelmän käyttöönoton jälkeen käsipäätteitä on riittävästi.

Jatkokehityssuunnitelmissa selvitettiin, että ohjelmiston kehitystä jatketaan tietoliikenneliittymän toteuttamisen parissa. Tietoliikenneliittymän toteuttamisen jälkeen uutta järjestelmää aletaan testata ja testauksen jälkeen se käyttöönotetaan. Jatkokehityksessä sivuttiin myös sellaista mahdollisuutta, että käyttöönoton jälkeen ohjelmiston kehitystä jatketaan. Lisäkehityksen tavoitteena olisi kehittää ohjelmistoa ja viivakoodinlukujärjestelmää niin, että tulevaisuudessa reittilistojen paperitulosteet voitaisiin jättää kokonaan pois terminaalista.

10 JOHTOPÄÄTÖKSET

Tämän opinnäytetyön teon aikana opittiin paljon ohjelmiston suunnittelusta ja taustatiedon keräämisen tärkeydestä, ennen työn suunnittelun aloittamista. Suurempaan järjestelmään liittyviä ohjelmistoja suunniteltaessa on ensin tiedettävä mihin suurempi järjestelmä kykenee, ennen kuin ohjelmiston ominaisuuksia lähdetään suunnittelemaan.

10.1 Tulevaisuuden näkymät

C# on ohjelmointikielenä suhteellisen tuore ja varsinkin mobiililaitteissa se on yleistynyt .NET Compact Framwork -alustan avulla vuonna 2002. Tämän lisäksi Microsoft tulee varmasti kehittämään .NET Compact Framework -ympäristöä parempaan suuntaan. Näiden asioiden perusteella voidaan todeta, että tulevaisuudessa kun ohjelmistoon täytyy tehdä muutoksia tai päivityksiä, ei se tuota ongelmia. Toinen hyvä asia on .NET-alustan järjestelmäriippumattomuus. Kun jossain vaiheessa siirrytään RFID-tekniikkaan ja uusiin laitteisiin, ohjelmistoa ei tarvitse räätälöidä kokonaan uudestaan, vaan parhaimmassa tapauksessa ohjelmistoon tarvitsee muuttaa vain RFID-tunnisteenluku-osa.

RFID-tekniikka ja sen avulla toteutettu UPC-standardi ovat vasta alkutekijöissään. RFID-tagien hinnat ovat vielä niin paljon korkeammat verrattaessa viivakooditarroihin, että niiden käyttöönotto ei ole vielä kannattavaa. Kaikkein suurin syy kutienkin on se, että muutosten tekeminen suurien yritysten järjestelmiin on hidasta ja kallista. Tällä perusteella RFID-tekniikan käyttöönottoa koko toimitusketjussa saadaan odottaa vielä ainakin 15 vuotta puhumattakaan sen saumattomasta toiminnasta.

LÄHTEET

About.Com. 2010. http. [WWW-dokumentti]. Bradley Mitchell. [22.02.2010]. Saatavana: http://compnetworking.about.com/od/networkprotocols/g/bldef_http.htm

Archer. T. 2001. Inside C#:Microsoft .NET. Suomentaja: Jussi Arola. Helsinki: Edita Oyj.

BarCode 1. 2009a. All About UPC Barcode & EAN Barcode. [WWW-dokumentti]. Adams Communications. [22.02.2010]. Saatavana: http://www.adams1.com/upccode.html

BarCode 1. 2009b. All About Code 39 Barcode. [WWW-dokumentti]. Adams Communications. [23.02.2010]. Saatavana: http://www.adams1.com/39code.html

Barcoding. 2003. Barcodes Sweep the World. [WWW-dokumentti]. Tony Seideman. [6.11.2009]. Saatavana: http://www.barcoding.com/information/barcode_history.shtml

Boling, B. 2003. Programming Microsoft: Windows CE .NET. 3rd Edition. Washington: Microsoft Press.

Bittikarttagrafiikka. 2008. Digitaalisen median perusteet 2009. [WWW-dokumentti]. Creative Commons. [25.1.2010]. Saatavana: http://www.tol.oulu.fi/kurssit/dmp/eeva/bittikartta.html

Choudhury, P. 2008. Common Language Runtime (CLR). [WWW-dokumentti]. Prabir Choudhury's blog. [25.1.2010]. Saatavana: http://prabirchoudhury.wordpress.com/2008/12/15/common-language-runtime-clr/

GS1. 2009. GS1 DataMatrix. [Verkkokirja]. GS1. [Viitattu 23.2.2010]. Saatavana: http://www.gs1.org/docs/barcodes/GS1_DataMatrix_Introduction_and_technical_overview.pdf

GS1 Finland. 2010. GS1 Kuljetusyksiköiden merkinnät. [WWW-dokumentti]. GS1 Finland. [Viitattu 7.3.2010]. Saatavana: http://www.gs1.fi/gs1-palvelut/kuljetusyksikoiden-merkinnat

HahkaWay. 2009. Neuvotteluissa ja työntekijöiden haastatteluissa hankittuja tietoja.

Hedgepeth, WO. 2007. RFID METRICS:Decision Making Tools for Todays's Supply Chains. Broken Sound:Taylor & Francis Group.

Liberty, J. 2005. Programming C#: Building .NET applications. 4th Edition. Gravenstein: O'Reilly Media, Inc.

Max, M. 2002. Essentials of inventory management. [Verkkokirja]. AMACOM. [Viitattu 9.11.2009]. Saatavana Ebrary-tietokannasta: Vaatii käyttöoikeuden.

Miles, S. B., Sarma, S. E. & Williams, J. R. 2008 RFID:Technology and Applications. Cambridge: Cambridge University.

Msdn. 2010a. .NET Compact Framework. [WWW-dokumentti]. Microsoft. [13.1.2010]. Saatavana: http://msdn.microsoft.com/en-us/library/f44bbwa1.aspx

Msdn. 2010b. .NET Framework. [WWW-dokumentti]. Microsoft. [13.1.2010]. Saatavana: http://msdn.microsoft.com/en-us/library/zw4w595w.aspx

Msdn. 2010c. XML. [WWW-dokumentti]. Microsoft. [25.1.2010]. Saatavana: http://msdn.microsoft.com/en-us/library/aa286548.aspx

Msdn. 2010d. Differences between the .NET Compact Framework and the .NET Framework. [WWW-dokumentti]. Microsoft.[22.3.2010]. Saatavana: http://msdn.microsoft.com/fi-fi/library/2weec7k5%28en-us%29.aspx

Motorola. 2009. MT2000 Series Handheld Mobile Terminals. [WWW-dokumentti]. Motorola. [19.1.2010]. Saatavana: http://www.motorola.com/Business/US-EN/Business+Product+and+Services/Bar+Code+Scanning/Bar+Code+Scanners/General+Purpose+Scanners/MT2000+Series+Handheld+Mobile+Terminals_US-EN

Mänikkö, T. 2000a. Johdatus ohjelmointiin: konekieli. [PowerPoint-esitys]. Jyväskylän yliopisto. [26.1.2010]. Saatavana: http://myy.haaga-helia.fi/~ict1td001/tito/K06/Konekieli.ppt

Männikkö, T. 2000b.Johdatus ohjelmointiin: Konekielet. [WWW-dokumentti]. Jyväskylän yliopisto. [26.1.2010]. Saatavana: http://www.mit.jyu.fi/opetus/Ciao/ciao016.htm

RoHS. 2009. RoHS-direktiivi. [WWW-dokumentti]. Sähköala.fi. Saatavana: http://www.sahkoala.fi/kohderyhmat/ammattilaiset/ymparisto/fi_FI/RoHS-direktiivi/

Taloon.com, 2010. IP-luokitus. [WWW-dokumentti]. Saatavana: http://kauppa.taloon.com/facebox/ip.html

Tietopankki. 2009. Langaton verkko. [WWW-dokumentti]. Buffalo. [25.1.2010]. Saatavana: http://wlan.dacco.fi/langaton.htm

Yao, P. & Durant, D. 2009. Programming .NET Compact Framework 3.5. 2nd Edition. U.S.:Corporate and Goverment Sales.

LIITE 1: MT2000-sarjan tekniset tiedot

MT2000 Specifications

Physical Characteristics	
Dimensions:	7.8 in H x 3.1 in. W x 5.0 in. D 19.8cm H x 7.8 cm W x 12.7cm D
Weight:	13.5 oz/378 gm
Display:	320 X 240 QVGA color graphic display
Battery:	2400 mAh Li-Ion @3.7Vdc
Communication Interface:	USB 2.0 Host and RS-232
Keypad:	21-key single stroke numeric and shifted alpha, 4-way navigation button with 2 soft keys for application control
Color:	Dark Gray
Performance Characteristics	
CPU:	IntelXScale PXA270 @312MHz
Operating System:	Microsoft® Windows CE 5.0
Memory:	64MB/64MB
Application Development Environment:	Microsoft Windows client CE EMDK; Motorola EMDK; MCL Designer V 3.0 (MCL Client pre-loaded on both the MT2070 and the MT2090; pre-licensed on the MT2070 only)
User Environment	
Operating Temperature:	-4° to 122° F/-20° TO 50° C
Storage Temperature:	-40° to 160° F/-40° TO 70° C
Humidity:	5% to 95% non-condensing
Drop Specification:	6 ft./1.8 m drop to concrete over the operating temperature range
Tumble Specification:	250 3.2 ft./1m tumbles (500 drops)
Environmental Sealing:	IP54
ESD:	±15kVDC air discharge, ±8kVDC direct/indirect discharge
Data Capture Options	
Options:	4 configurations: 1D Laser, 1D/2D Imager, 1D/2D HD Imager, DPM Imager
Symbology Decode Capability	
1D:	UPC/EAN (UPCA/UPCE/UPCE1/EAN-8/EAN-13/ JAN-8/JAN-13 plus supplements, ISBN (Bookland), Code 39 (Standard, Full ASCII, UCC/EAN-128, ISBT-128 Concatenated), Code 93, Codabar/ NW7, MSI Plessey, I 2 of 5 (Interleaved 2 of 5 / ITF, Discrete 2 of 5, IATA, Chinese 2 of 5), GS1 DataBar (Omnidirectional, Truncated, Stacked, Stacked Omnidirectional, Limited, Expanded, Expanded Stacked, Inverse), Base 32 (Italian Pharmacode)
2D:	TLC-39, Aztec (Standard, Inverse), Maxicode, DataMatrix/ECC 200 (Standard, Inverse), QR code (Standard, Inverse, Micro)
PDF417:	PDF417 (Standard, Macro), Composite Codes (CC-A,CC-B,CC-C)
Postal:	U.S. Postnet and Planet, U.K. Post, Japan Post, Australian Post, Netherlands KIX Code, Royal Mail 4 State Customer, upu fics 4 State Postal, USPS 4CB
Wireless LAN Data Communications	
Radio:	Tri-mode IEEE® 802.11a/b/g
Output Power:	100mW US and international
Antenna:	Internal
Data Rates Supported:	802.11b – 11MBPS; 802.11a – 54MBPS; 802.11g – 54MBPS
Frequency Range:	Country dependant: typically 2.4 to 2.5 GHz for 802.11b/g and 5.15 to 5.825GHz for 802.11a
Cordless Data Communications	
Bluetooth:	2.4Ghz class I radio – 300 feet (91.44 m) open air
Peripherals and Accessories	
Cradles:	Single slot charge only cradle, Single slot multi-interface with Bluetooth cradle, Single slot charge only fork lift cradle, 4-slot charge only cradle, 4-slot Ethernet cradle
Chargers:	Four slot battery charger
Other Accessories:	Belt holster, desk cup, Intellistand
Printers:	Supports Motorola approved 3rd party printers
Regulatory	
Electrical Safety:	Certified to UL60950-1, CSA C22.2 No. 60950-1, EN60950-1
EMI/RFI:	North American: FCC Part 2 (SAR), FCC Part 15 Subpart B - Class B, RSS210, EU: EN 301-489-1, 489-17
Laser Safety:	IEC Class 2/FDA Class II in accordance with IEC60825-1/EN60825-1

Warranty

The MC2000 Series is warranted against defects in workmanship and materials for a period of 36 months from date of shipment, provided that the product remains unmodified and is operated under normal and proper conditions

Recommended Services	
Customer Services:	Service from the Start with Comprehensive Coverage

See Decode Zone charts on back...