Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma

Janne Lanki

TIEDONSIIRTO HUOLTOKIRJA-JÄRJESTELMÄÄN ERÄAJOINA

Opinnäytetyö 2010

TIIVISTELMÄ

Janne Lanki
Tiedonsiirto Huoltokirja-järjestelmään eräajoina, 56 sivua
Saimaan ammattikorkeakoulu, Lappeenranta
Tekniikka, tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö 2010
Ohjaajat: Toimitusjohtaja, Heikki Hepomäki, Cadimage Oy
Lehtori, Martti Ylä-Jussila, Saimaan ammattikorkeakoulu

Opinnäytetyön tavoitteena on toteuttaa ohjelma, joka suorittaa eräajoina tiedonsiirron Excel-tiedostoista Huoltokirja-järjestelmän SQL-tietokantaan. Automaattinen tiedonsiirto säästää manuaalista työtä, lyhentää käyttöönottoaikaa sekä vähentää tiedonsyötön virheitä.

Huoltokirja on kiinteistöhuollon tarpeisiin toteutettu toiminnanohjausjärjestelmä, jonka päätoiminnot ovat kiinteistöihin kohdistuvien huoltotoimenpiteiden sekä kaluston hallinta.

Asiakkaana opinnäytetyöprojektilla on Cadimage Oy. Valmis tuote tulee Cadimage Oy:n asiakkaan, Lappeenrannan kaupungin tilakeskuksen, käyttöön. Valmiin tuotteen on oltava niin luotettava ja helppokäyttöinen, että sen peruskäyttäjäksi voidaan tarvittaessa kouluttaa henkilö, jonka tietotekninen ymmärrys on kansalaisten perustasoa.

Käytettyjä tekniikoita toteutuksessa ovat Microsoft .NET Framework, C#-ohjelmointikieli, Microsoft SQL Server sekä OLE DB -yhteydet.

Järjestelmän kehitystyö on aloitettu huhtikuussa 2010 ja järjestelmä on otettu tuotannolliseen käyttöön marraskuussa 2010.

Asiasanat: järjestelmäintegraatio, tiedonsiirto, eräajo, tietokannat

ABSTRACT

Janne Lanki
Data Transfer to Real Estate Maintenance Management Software by Batch Processing, 56 pages
Saimaa University of Applied Sciences, Lappeenranta
Technology, Information Technology
Software Engineering
Final year project 2010
Instructors: CEO, Heikki Hepomäki, Cadimage Oy
Lecturer, Martti Ylä-Jussila, Saimaa University of Applied Sciences

The purpose of the final year project was to develop an application that performs data transfer via batch processing to Real Estate Maintenance Management Software. The data is read from Excel-files and written to a Microsoft SQL database. The automated data transfer saves manual labor, time and reduces errors in data input.

The Real Estate Maintenance Management Software has been developed for the needs of property maintenance. Its main points are managing real estates' service information and equipment.

The client of the final year project is Cadimage Oy. The finished product is to be used by a client of Cadimage Oy, The Lappeenranta Estates Centre. The product has to be reliable and user-friendly, enough to be used with minimal training and novice computer skills.

The technologies used include Microsoft .NET Framework, C# programming language, Microsoft SQL Server and OLE DB connectivity.

The development of the project began in April 2010. The system has been in use since November 2010.

Keywords: system integration, data transfer, batch processing, databases

SISÄLTÖ

TERMISTÖ

Agile	Joukko ohjelmistotyön menetelmiä, jotka tunnetaan myös nimellä ketterä ohjelmistokehitys. Menetelmissä pyritään minimoimaan turha dokumentointi ja keskittymään tehokkaaseen työskentelyyn ja nopeisiin näkyviin tuloksiin.
C#	Microsoftin kehittämä moderni olio-ohjelmointikieli, joka kehitettiin yhdistämään C++:n tehokkuus ja Javan helppokäyttöisyys.
CLR	Common Language Runtime. .NET Frameworkin osa, joka tarjoaa virtuaaliajoympäristön .NET-yhteensopivan kääntäjän tuottaman välikoodin suorittamiseen.
DataRow	Olio, jonka avulla kuvataan yksittäinen tietorivi.
HyperDoc	Windows-pohjainen piirustusarkisto, joka soveltuu kiinteistöjen ja teollisuuden kunnossapidon dokumenttien hallintaan.
Java	Sun Microsystemsin kehittämä alustariippumaton olio-ohjelmointikieli.
Microsoft SQL Server	Microsoftin kehittämä relaatiotietokantajärjestelmä, joka sisältää tietokannan hallintatyökalut. Microsoft SQL Serveristä on olemassa eri versioita muun muassa kotikäyttäjille (Express-versio) ja yrityskäyttöön (Standard-, Enterprise- ja Datacenter-versiot)
.NET Framework	Microsoftin kehittämä ohjelmistokomponenttikirjasto, joka koostuu luokkakirjastosta ja CLR-ajoympäristöstä.
ODBC	Open Database Connectivity. Standardoitu rajapinta, jonka avulla sovellukset voivat kommunikoida ajurien avulla tietokantojen kanssa.
OLE DB	Microsoftin kehittämä rajapinta tiedon käsittelyyn. OLE DB:n avulla tietoa voidaan käsitellä miltei samalla tavalla riippumatta lähdetiedon muodosta.
Relaatiotietokanta	Tietokantamalli, jossa tiedot esitetään tauluina, joilla on suhteita toisiin tauluihin. Taulujen suhteet, eli relaatiot, määritetään tauluissa olevien pääavainten ja niihin viittaavien viiteavainten avulla. Hyvin suunnitelluissa relaatiotietokannoissa samaa tietoa ei sijaitse useassa paikassa. Tietoa haeta tallennusjärjestyksen perusteella, vaan viittaamalla arvoihin (pääavaimet).
Scrum	Ohjelmistokehityksessä käytettävä projektinhallinnan menetelmä. Scrum kuuluu ketterän ohjelmistokehityksen, eli Agile-menetelmien, piiriin.
SQL	Structured Query Language. Standardoitu kyselykieli, jolla voidaan tehdä erilaisia hakuja, muutoksia ja lisäyksiä relaatiotietokantaan.

Visual Studio	Microsoftin toimittama sovelluskehitysympäristö. Sovelluksia voidaan Visual Studiolla kehittää eri ohjelmointikielillä, joita ovat muun muassa C++, C# ja Visual Basic.
XML	eXtensible Markup Language. Standardi, jolla tiedon merkitys voidaan kuvata tiedon sekaan. Käytetään usein järjestelmienvälisen tiedonvälityksen formaattina.

1 JOHDANTO

Opinnäytetyön tavoitteena on kehittää ohjelma, jolla voidaan tehdä eräajoina tiedonsiirto erilaisista lähteistä Huoltokirja-järjestelmän tietokantaan. Aihe on syntynyt Cadimage Oy:n asiakkaan, Lappeenrannan kaupungin tilakeskuksen, tarpeesta. Järjestelmään tulisi pystyä siirtämään suuria tietomääriä ohjelmallisesti, koska käsin syöttäminen on virhealtista ja hyvin hidasta.

Lappeenrannan kaupunki käyttää kiinteistöhuollon Huoltokirja–järjestelmää, jonka Cadimage Oy on teettänyt pääasiassa oppilastöinä Saimaan ammattikorkeakoulun opiskelijoilla. Tämä työ on jatkokehitysprojekti kyseessä olevaan järjestelmään. Järjestelmään on ideoitu vielä tämänkin työn jälkeen uusia kehitysprojekteja.

Asiakkaan tarpeisiin riittää lyhyellä tähtäyksellä, että tietojen lähdemuoto on Excel-tiedostoissa. Opinnäytetyön vaatimuksia on täten rajattu niin, että hyväksyttäviä lähdemuotoja ovat Excel-tiedostot. Kohteena on asiakkaan Microsoft SQL –tietokanta. Tiedonsiirron välimuotona käytetään XML-kuvauskieltä.

Projektin tavoitteena on kehittää suurten tietomäärien siirtämiseen soveltuva ohjelmisto. Ohjelmiston tulee olla luotettava ja niin helppokäyttöinen, että sen peruskäyttäjäksi voidaan tarvittaessa kouluttaa henkilö, jonka tietotekninen ymmärrys on kansalaisten perustasoa.

2 PROJEKTIN SIDOSRYHMÄT

Projektilla oli vain kaksi sidosryhmää, jotka olivat Cadimage Oy ja Lappeenrannan kaupungin tilakeskus.

2.1 Cadimage Oy

Cadimage Oy on 1990-alussa perustettu imatralainen yritys. Yrityksen toimintana oli alunperin karttojen ja teknisten piirustusten muuntaminen paperimuodosta sähköiseen muotoon. Cadimage Oy tarjoaa myös IT-konsultointia muutamalle pienelle ja keskisuurelle yritykselle. Toiminnan laajentuessa yritys on luonut yhteyksiä HyperDoc-dokumentinhallinta-järjestelmän luoneeseen Tessel Systems AB:hen.

Yrityksen asiakkaita ovat muun muassa Lappeenrannan kaupunki, Lappeenrannan seurakunta ja Savitaipaleen kunta. Cadimage Oy tarjoaa nykyisin myös teknistä tukea HyperDoc-järjestelmälle sekä markkinoi HyperDoc:ia sekä siihen liitettyä Huoltokirja-järjestelmää. Tuotteita esitellään messuilla ja kiinteistöhuoltoalan tapahtumissa.

2.2 Lappeenrannan kaupungin tilakeskus

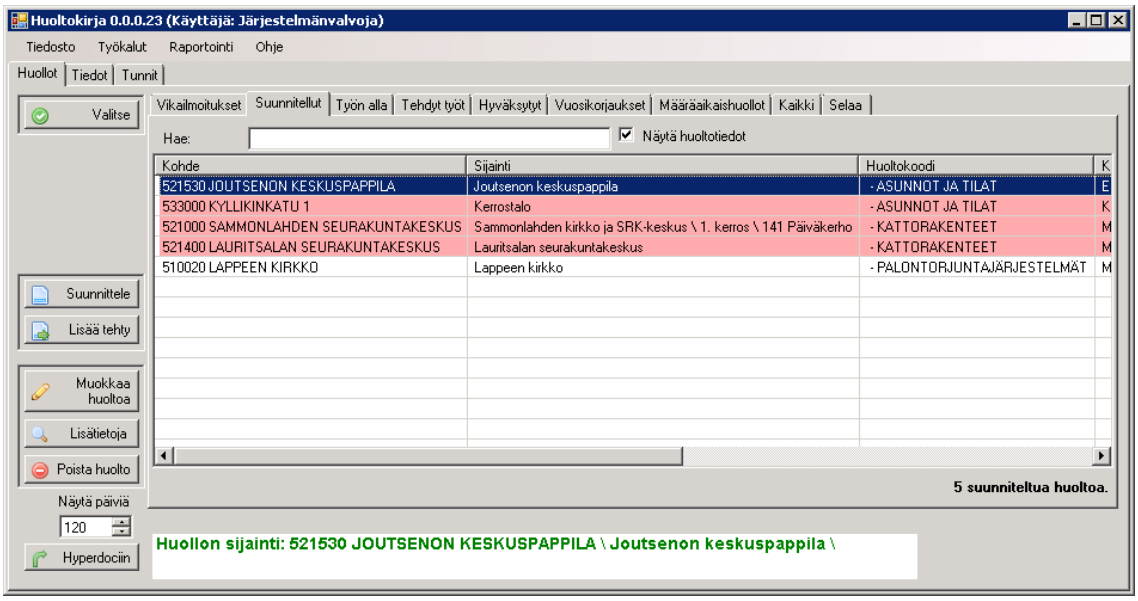
Lappeenranta on kaupunki Etelä-Karjalan maakunnassa Kaakkois-Suomessa. Kaupunki sijaitsee Saimaan vesistöalueen etelärannan ja Venäjän rajan välisellä alueella. Asukasluvultaan Lappeenranta on Suomen 13:nneksi suurin kaupunki, ja siellä asuu 71 720 asukasta. (Väestörekisterikeskus 2010.)

Lappeenrannan kaupungin tilakeskus vastaa siitä, että kaupungin toimialojen käytössä on tarkoituksenmukaiset, riittävät ja kokonaistaloudelliset toimitilat. Tilakeskus täyttää nämä vaatimukset rakennuttamalla, suunnittelemalla ja vuokraamalla tiloja sekä ylläpitämällä niitä järkevän elinkaariajattelun mukaisesti. Edellisen lisäksi Tilakeskus vuokraa kaupungin omistamia tiloja ulkopuolisille sekä vastaa muun muassa toritoiminnan järjestämisestä. (Lappeenranta 2010.)

Cadimage Oy on toimittanut Tessel Systems AB:n kehittämän HyperDoc-järjestelmän Lappeenrannan kaupungin tilakeskukselle. Järjestelmän avulla Lappeenrannan kaupungin tilakeskus hallinnoi kiinteistöjensä ja kalustonsa dokumentteja, joihin kuuluvat muun muassa huoneistojen pohjapiirustukset sekä laitteiden käyttöohjeet. Cadimage Oy on myös toimittanut Lappeenrannan tilakeskukselle Huoltokirja-järjestelmän, joka toimii HyperDoc-järjestelmän kanssa yhteiskäytössä. Huoltokirja-järjestelmällä Lappeenrannan kaupungin tilakeskus voi ylläpitää ja seurata omistamiinsa kiinteistöihin kohdistuvia huoltotoimenpiteitä.

3 HUOLTOKIRJA- JA HYPERDOC-JÄRJESTELMÄT

Huoltokirja on kiinteistöhuollon tarpeisiin toteutettu toiminnanohjausjärjestelmä. Järjestelmässä ovat keskeisinä toimintoina kiinteistöihin kohdistuvien huoltotoimenpiteiden suunnittelu ja seuranta. Huoltokirja toimii asiakas-palvelin-periaatteella. Asiakkaat voivat hakea ja muokata Huoltokirja-järjestelmän tietoja tekemällä pyyntöjä palvelimelle. Tiedot sijaitsevat palvelinkoneen MS SQL –tietokannassa, jonka käyttöä palvelinsovellus hallinnoi. Asiakas kommunikoi palvelinsovelluksen kanssa käyttämällä asiakassovellusta. Seuraavassa on esitelty asiakassovelluksen päätoiminnot. Kuvassa 3.1 on Huoltokirja-sovelluksen päänäkymä.



Kuva 3.1 Huoltokirjan päänäkymä

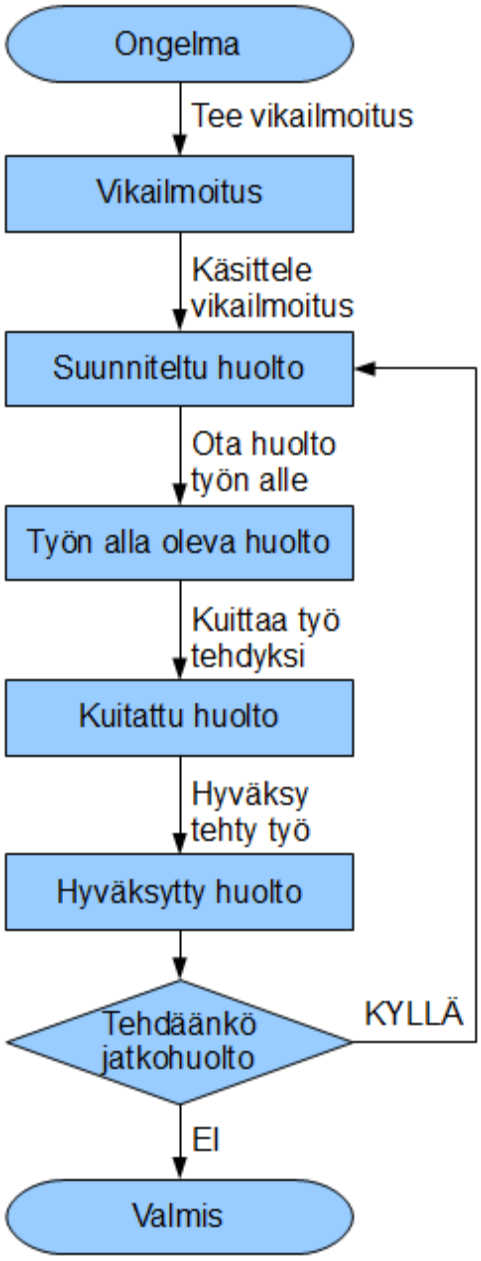
3.1 Huoltoprosessi

Huoltoprosessissa on kuusi vaihetta:

- vikailmoituksen tekeminen
- vikailmoituksen käsittely
- huollon ottaminen työn alle
- huollon kuittaaminen

- huollon hyväksyminen
- jatkohuollon tekeminen.

Kuvassa 3.2 on kuvattu tyypillisen huoltoprosessin eteneminen.



Kuva 3.2 Huoltoprosessin kulku

Yleisin huollon etenemisprosessi on seuraavankaltainen: Järjestelmä vastaanottaa tietyn kiinteistön edustajalta **vikailmoituksen.** Työnjohtaja käsittelee vi-

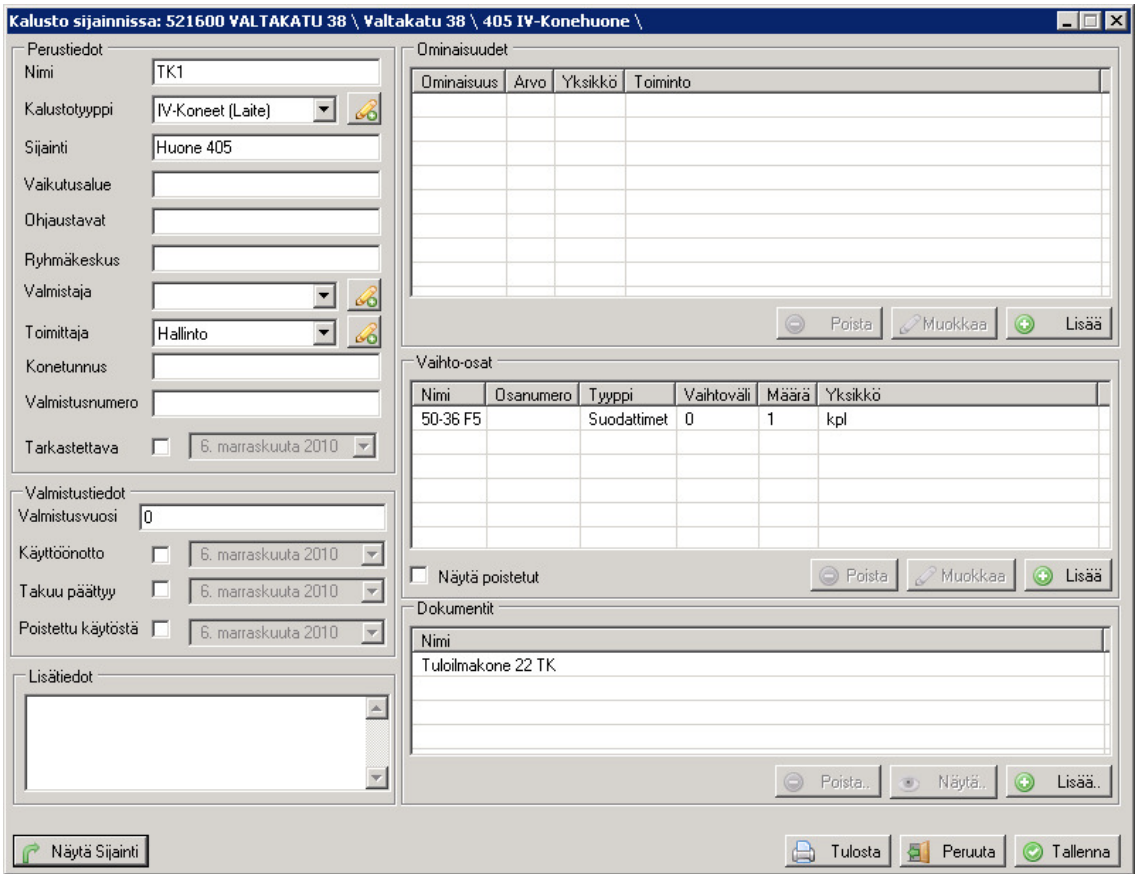
kailmoituksen tekemällä siitä **suunnitellun** huollon, jolle määrätään huollon suorittava henkilö, eräpäivä ja muut perustiedot. Huollon suorittava huoltohenkilö ottaa suunnitellun huollon **työn alle**. Kun huoltotoimenpide on suoritettu, merkitsee huoltohenkilö työn tehdyksi ja kirjaa huoltotyölle tehdyt toimenpiteet, vaihdetut osat ja muut tarvittavat tiedot. Tämän jälkeen työ on **kuitattu**-tilassa. Työnjohtaja käsittelee tehdyn huollon ja merkitsee sille tarvittaessa kustannukset ja muut lisätiedot. Tämän jälkeen huolto on lopullisessa **hyväksytty**-tilassa. Hyväksymisvaiheessa huollosta on mahdollista tehdä jatkohuolto. Mikäli jatkohuolto halutaan tehdä, kopioidaan valmiin huollon perustiedot ja siirrytään kopioidun huollon **suunnittelu**-vaiheeseen. Jokaiseen uuteen huollon vaiheeseen siirryttäessä järjestelmä kirjaa ylös siirtymishetken päivämäärän myöhempää tarkastelua varten.

Eri huollon vaiheet ovat sallittuja eri käyttäjäryhmille. Vikailmoituksia tekevät ainoastaan kiinteistöjen edustajat. Vikailmoituksia käsittelevät työnjohtajat. Huoltohenkilöt sekä ottavat huoltoja työn alle, että kuittaavat työt tehdyiksi. Työnjohtajat hyväksyvät tehdyt työt. Järjestelmänvalvojat voivat puolestaan suorittaa huollon kaikki vaiheet.

Järjestelmä mahdollistaa myös huoltojen tietojen muokkauksen jokaisessa huollon vaiheessa sekä jatkohuollon luomisen tehdystä huollosta. Erilaiset toimenpiteet ovat sallittuja vain tietyille järjestelmän käyttäjäryhmille, joten esimerkiksi vikailmoituksen vastaanottaminen ja huoltohenkilölle määrääminen on mahdollista vain **työnjohtaja**-käyttäjäryhmään kuuluvalle käyttäjälle.

3.2 Kalustonhallinta

Huoltokirja-järjestelmässä voidaan myös muokata, luoda, kopioida ja siirtää kiinteistöihin kuuluvaa kalustoa sekä niihin kuuluvia osia, ominaisuuksia ja dokumentteja. Kuvassa 3.3 on Huoltokirjan Kaluston muokkaus -lomake.

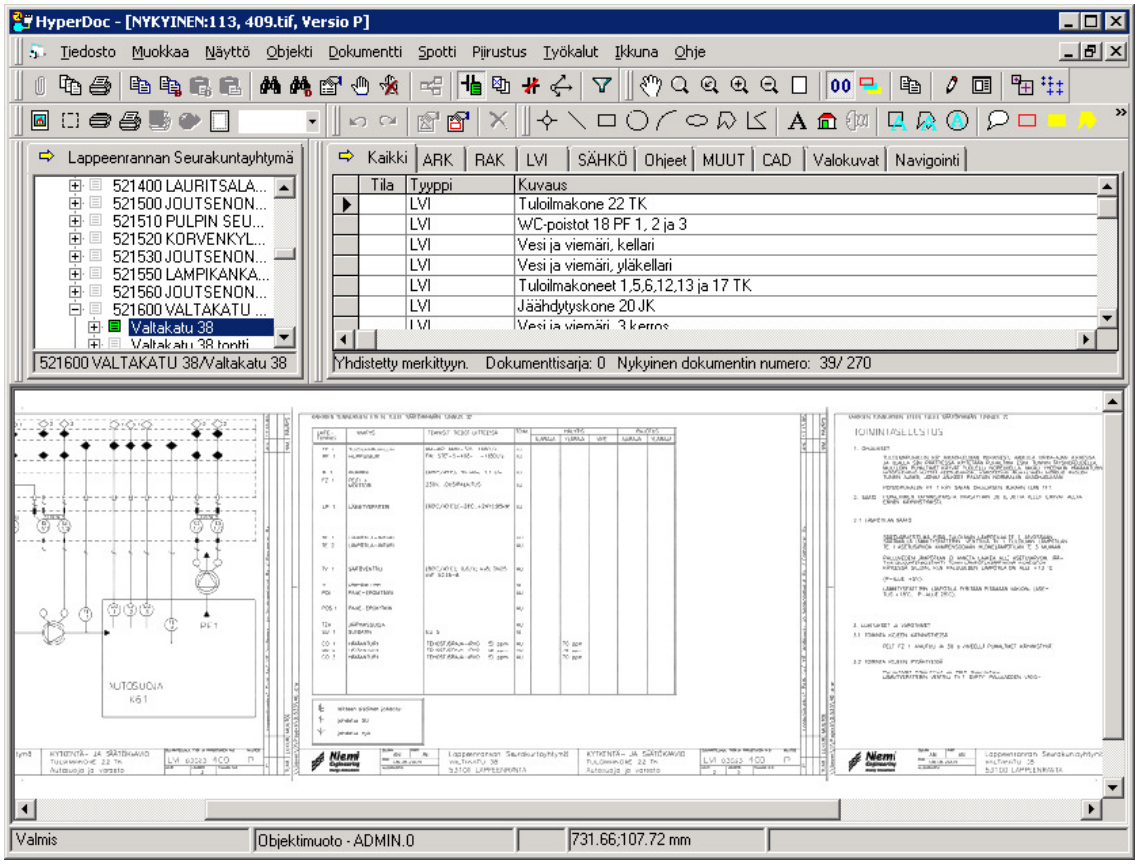


Kuva 3.3 Huoltokirjan Kaluston muokkaus -lomake

Kuvassa 3.3 näkyy kalusto, sen perustiedot, ominaisuudet, vaihto-osat sekä dokumentit. Lomakkeelta voidaan muokata kaluston tietoja, lisätä ja poistaa ominaisuuksia, vaihto-osia ja dokumentteja sekä avata dokumentteja.

Kiinteistöt ja kalusteet on linkitetty Huoltokirja-järjestelmän kanssa yhteiskäytössä olevaan HyperDoc-dokumentinhallintajärjestelmään. Huoltokirja- ja HyperDoc-järjestelmien välillä voidaan siirtyä kiinteistöstä toiseen, selata dokumentteja ja pohjapiirustuksia sekä näyttää kaluston ja huoltojen sijainnit.

Huoltokirjan Kaluston muokkaus -lomakkeelta voidaan avata kalustoon liitetty dokumentti, joka käynnistää HyperDoc-sovelluksen ja näyttää kyseisen dokumentin HyperDoc-sovelluksen selausikkunassa. Kuvassa 3.4 on HyperDoc-dokumentinhallintajärjestelmän päänäkymä.



Kuva 3.4 HyperDoc-dokumentinhallintajärjestelmä

Kuvassa 3.4 on aiemmin Huoltokirja-sovelluksesta valittu dokumentti avattuna HyperDoc-dokumentinhallintajärjestelmään Huoltokirja-sovelluksen kautta.

4 TIEDONMUUNNOKSEN ONGELMAT

Yritysten tietojärjestelmissä käsitellään usein suuria tietomääriä. Yritykset ja instanssit määrittelevät tietojensa tallentamismuodot oman toiminnan ja tarpeidensa mukaisesti. Jos tiedonsiirtoa tarvitaan näiden järjestelmien välillä, tiedon muoto on harvoin määritelty yhdenmukaisesti. Tällöin tieto on muunnettava kohdejärjestelmän ymmärtämään muotoon.

4.1 Tietojen konversio skripteillä

Tietoja voidaan muuntaa eri muotoihin erilaisilla skripteillä. Skriptit ovat ajettavia suoritustiedostoja. Kun tietokannan tietoja halutaan muuttaa toisenmuotoiseen tietokantaan, voidaan luoda konversioskripti, jossa määritellään, minkä nimiset taulut ja yhteydet muokataan vanhasta tietokannasta uuden tietokannan tauluiksi ja yhteyksiksi. Taulujen välille voidaan luoda uusia yhteyksiä, ja niiden välisiä yhteyksiä voidaan purkaa. Myös taulujen tietoja voidaan muokata, lisätä tai poistaa skriptien avulla. Kun skripti on luotu, se suoritetaan, jolloin se tekee muutokset tietokantaan. Skriptin suorittamisen lopputuloksena on joko uusi tietokanta tai vanhan tietokannan päivitetty tila.

4.2 Tietoihin liittyvät ongelmat

Seuraavassa on kuvattu järjestelmien välisen tiedonsiirron keskeiset ongelmat, joita ovat tietojen määrittelyerot, riippuvuusongelmat, päällekkäisyysongelmat sekä syötetietojen tarkistukset. Yleisten ongelmien yhteydessä on kuvattu tähän opinnäytetyöhön liittyvät ongelmat. Esimerkin kuvista on poistettu ylimääräistä tietoa selkeyden vuoksi.

4.2.1 Tiedon määrittelyerot

Järjestelmien tiedot voi olla erilaisissa muodoissa, kuten

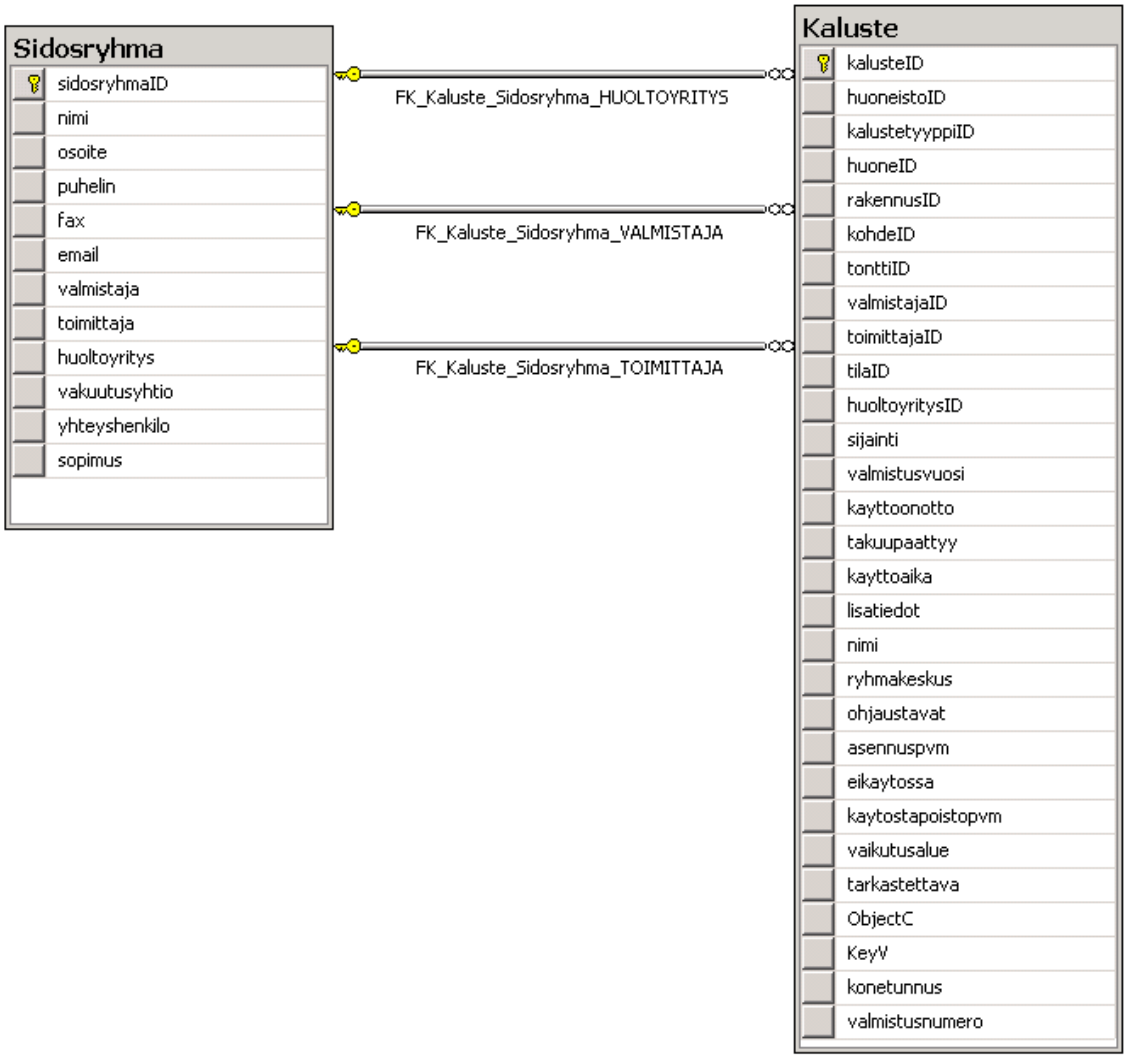
- merkkipohjaisissa peräkkäistiedostoissa
- binääritiedostoissa
- Excel-taulukoissa
- Word-asiakirjoissa

- relaatiotietokannoissa.

Jotta tiedonsiirto eri järjestelmien välillä sujuisi luotettavasti, on järjestelmien välille määriteltävä jonkinlaiset muunnossäännöt. Sääntöjen avulla voidaan tieto muuntaa lähdejärjestelmästä kohdejärjestelmään sopivaksi.

Ongelmaan on kehitetty erilaisia ratkaisuja, joilla on hyvät ja huonot puolensa. Eräs tapa määritellä vastaanotettavan tiedon tyyppi on käyttää XML-määritettyjä skeemoja. Skeemat määrittävät, minkä nimisiä tietokenttiä suuremmat tietokokonaisuudet sisältävät. Skeemat myös määrittävät kelvollisten syötteiden muodon, jotta vääränmuotoisen tiedon vastaanotto voitaisiin välttää ennakkoon.

Kuvassa 4.1 ovat kohdejärjestelmänä toimivan Huoltokirja-järjestelmän Kalusto- ja Sidosryhmä-taulujen tiedot.



Kuva 4.1 Sidosryhmä ja Kaluste-taulut

Jokainen Sidosryhmä yksilöidään **sidosryhmaID**:llä. Jokainen **Kaluste** yksilöidään **kalusteID**:lla. Jokaisella **Kalusteella** voi olla huoltoyritys, valmistaja ja toimittaja, joista jokainen on viittaus **Sidosryhmä**-taulun **sidosryhmaID**-kenttään.

Lähdetiedot ovat esimerkissämme Excel-tiedostossa, jonka Kalusto-välilehdellä ovat kuvan 4.2 mukaiset tiedot.

	A	B	C	D	E	F	G	H
1	ID	Nimi	Konetunnus	Sijainti	Vaikutusalue	Valmistaja	Toimittaja	Huoltoyritys
2	1	Tuloilmakone	TK01	IV konehuone	Koko Rakennus	IV-Produkt	Intervent oy	I-Huolto Asennus oy
3	2	Poistoilmakone	PK01	IV konehuone	Koko Rakennus	IV-Produkt	Intervent oy	I-Huolto Asennus oy
4	3	Huippuimuri	PF02	Vesikatto	Opetuskeittiö	Vallox	I-Huolto Asennus oy	I-Huolto Asennus oy
5	4	Huippuimuri	PF03	Vesikatto	WC tilat+SK	Vallox	I-Huolto Asennus oy	I-Huolto Asennus oy
6	5	Huippuimuri	PF04	Vesikatto	Radon poisto	SK-tuote	I-Huolto Asennus oy	I-Huolto Asennus oy
7	6	Huippuimuri	PF05	Vesikatto	Radon poisto	SK-tuote	I-Huolto Asennus oy	I-Huolto Asennus oy
8	7	Huippuimuri	PF06	Vesikatto	Radon poisto	SK-tuote	I-Huolto Asennus oy	I-Huolto Asennus oy
9	8	liesikupu	Fasad	Opetuskeittiö	Liedet	Lapetek oy	I-Huolto Asennus oy	I-Huolto Asennus oy
10	9	Peltimoottori	FG01	TK01	TK01	Belimo	Realmec oy	Realmec Oy
11	10	Peltimoottori	FG30	PK01	PK01	Belimo	Realmec oy	Realmec Oy
12	11	Peltimoottori	FG50	PK01 kanava	Keittiö poisto	Belimo	Realmec oy	Realmec Oy
13	12	Venttiilimoottori	TV45	TK01	TK01 lämmityspatteri	Belimo	Realmec oy	Realmec Oy
14	13	Säätökeskus	VAK-1	IV konehuone	Koko rakennus	Trend	Realmec oy	Realmec Oy
15	14	Säätöventtiili	TV45	TK01	TK01 lämmityspatteri	Belimo	Realmec oy	Realmec Oy

Kuva 4.2 Excel-tiedoston Kalusto-välilehti

Tiedonsiirtosovellus ei tiedä ilman määrityksiä, mitkä taulukon sarakkeet liittyvät mihinkin tietokannan tauluihin ja sarakkeisiin. Määritysten avulla sovellukselle voidaan kertoa, että Excel-taulukon **ID**-sarake vastaa tietokannan **Kaluste**-taulun **kalusteID**-kenttää, **Valmistaja**-sarake vastaa **Kaluste**-taulun **valmistajaID**-kenttää ja niin edelleen.

On kuitenkin huomattava, että esimerkiksi **valmistajaID**-kenttä tietokannan **Kaluste**-taulussa on kokonaisluku, joka yksilöi valmistajan (**Sidosryhmä**-taulusta). Excel-tiedostossa on kuitenkin **Valmistaja**-sarakkeessa tekstiä, tarkemmin sanottuna valmistajan nimi. Tiedonsiirtosovellukselle voidaan tehdä määritys, joka kertoo, että **Valmistaja**-sarakkeessa olevat tiedot etsitään tietokannan **Sidosryhma**-taulusta **nimi**-tiedon perusteella.

4.2.2 Riippuvuusongelmat

Tilanne monimutkaistuu siinä vaiheessa, kun tietojen välillä on viittauksia. Tietojenväliset viittaukset määritellään tietoja yksilöivillä pääavaimilla ja viiteavaimilla, jotka puolestaan viittaavat yksilöiviin pääavaimiin. Kuvassa 4.3 ovat esimerkin Excel-tiedoston **Vaihto-osat**-välilehden tiedot.

	A	B	C
1	**Kalusto ID**	**Osan nimi**	**Osanumero**
2	1	Suodatin	ELEF-190-F7
3	2	Suodatin	ELEF-190-F6
4	1	Puhallin suorakäyttö	EFA-190-00-FD-E2-0300-0
5	2	Puhallin suorakäyttö	EFA-190-00-FD-E2-0300-0
6	10	Peltimoottori	AF24
7	11	Peltimoottori	AF25
8	14	Venttiilimoottori	NRY24

Kuva 4.3 Excel-tiedoston Vaihto-osat-välilehti

Välilehdellä kerrotaan, mikä osa kuuluu millekin kalusteelle. **Kalusto ID**-sarakkeessa olevat luvut viittaavat Excel-tiedoston **Kaluste**-välilehden **ID**-sarakkeen riveihin. Kun tiedot kopioidaan Excel-tiedostosta tietokantaan, suorittaa tiedonsiirtosovellus ensin **Kaluste**-välilehden tietojen kopioinnin kantaan. Tietokannassa luodaan kuitenkin jokaiselle uudelle luodulle riville uusi uniikki **kalusteID**-arvo, joka ei vastaa Excel-tiedostossa oleviin **Kaluste**-välilehden **ID**-sarakkeen arvoja. Tiedonsiirtosovelluksen on tässä vaiheessa pidettävä kirjaa siitä, mikä oli Excel-taulukon **Kaluste**-taulun **ID**-sarakkeen rivin arvo sekä tietokannan **Kaluste**-tauluun juuri luodun rivin **kalusteID**-arvo. Nämä arvot liittyvät toisiinsa, koska **Vaihto-osat**-välilehden **Kalusto ID** -sarakkeen arvot on muutettava vastaamaan tietokantaan kopioinnin aikana päivitettyjä **kalusteID**-arvoja, jotta viittaukset siirtyisivät tietokantaan oikein.

4.2.3 Päällekkäisyysongelmat

Ongelmana on myös päällekkäisten tietojen välttäminen. Kohdejärjestelmässä on usein olemassa olevia tietoja, joita ei haluta esiintyvän moneen kertaan. Kuvassa 4.4 ovat esimerkin Excel-tiedoston **Sidosryhmä**-välilehden tiedot.

	A	B	C	D	E	F
1	**Nimi**	**Osoite**	**Puhelin**	**Fax**	**Sähköposti**	**Yhteyshenkilö**
2	Intervent Oy	Tampere	+35833485833	+35833485877	www.intervent.fi	Jari Parkkila
3	Vallox Oy	Myllykyläntie 9-11 32200 Loimaa	0107732200		www.vallox.com	
4	Lapetek Oy	Kankiraudantie 1 00700 Helsinki	09-2511030	09-25110351	www.lapetek.com	
5	Realmec oy	Lappeenranta/joutseno	0400421769		kai.pasanen@realmec.fi	Kai Pasanen
6	Belimo Suomi	Insinöörinkatu 7A 00880 Helsinki	0207639500	0207639501	www.belimo.fi	

Kuva 4.4 Excel-tiedoston Sidosryhmä-välilehti

Tietokannan **Sidosryhmä**-taulu saattaa sisältää ennestään yrityksiä joita sinne ollaan lisäämässä. Kun tiedonsiirtosovellus lisää Excel-tiedoston **Sidosryhmä**-välilehdellä olevia rivejä tietokantaan, tarkistaa se ensin, löytyykö jo olemassa oleva rivi tietokannasta. Mikäli rivi löytyy jo tietokannasta, ei lisäystä tehdä. Tietokannasta haetaan kuitenkin löydetyn rivin pääavaimen arvo (**sidosryhmaID**) talteen. Kun Excel-taulun muut välilehdet viittaavat kyseiseen sidosryhmään, käytetään viittauksen arvona tietokannasta löydettyä pääavaimen arvoa. Mikäli taas riviä ei löytynyt tietokannasta, lisätään rivi normaalisti tietokantaan, ja otetaan talteen juuri luodun rivin pääavaimen arvo, kuten kohdassa 4.2.2.

Tiedonsiirtosovelluksen on kuitenkin tiedettävä, mikä määrää sen, löytyykö rivi jo tietokannasta. Toisin sanoen, mikä tieto esimerkiksi yksilöi kunkin **Sidosryhmä**-taulun rivin. Tiedonsiirtosovellukselle voidaan määritellä, että yksilöivä tieto on sidosryhmän **nimi**. Toisaalta, määrittelyssä voidaan ottaa huomioon useampi tieto. Näin voitaisiin määritellä, että **Sidosryhmä**-taulun rivin yksilöllisyys määritellään **nimen** sekä **osoitteen** mukaan, jolloin tietokannan **Sidosryhmä**-tauluun voitaisiin lisätä useampi samanniminen sidosryhmä, kunhan niiden osoitteet eroavat toisistaan.

4.2.4 Syötetietojen tarkistukset

Ongelmana järjestelmien välisessä tietojensiirrossa on myös syötetietojen kelpoisuustarkistukset. Jos lähdejärjestelmässä on tehty syötetietojen kelpoisuustarkistukset jo lähdejärjestelmän tietojen syöttövaiheessa, ongelmia tuskin ilmenee. Mikäli lähdetietoina taas toimii esimerkiksi Excel-tiedosto, ei lähdetiedoille välttämättä ole tehty erikseen muotoilu- tai muita tarkistuksia.

Ongelmana voi olla esimerkiksi Excel-taulukon sarake, johon tulisi syöttää vain päivämäärätietoja. Mikäli Excel-taulukkoon ei ole erikseen tehty makroja, jotka estävät epäkelpojen tietojen syötön, tiedonsyöttäjät voivat syöttää kyseiseen sarakkeeseen mitä tahansa tekstiä. Näin lähdetietoja ei voida muuntaa suoraan kohdejärjestelmään. Ongelma voidaan ratkaista lisäämällä lähdetietojen syöttöön tarkistus (Excel-taulukkoon lisättävät makrot, jotka käsittelevät syötteiden tarkistuksia). Toinen ratkaisu ongelmaan on tehdä ennen tietojen kohdejärjestelmään siirtämistä esitarkistus, jossa varmistetaan, että kaikki siirrettävät tiedot ovat oikeassa muodossa. Mikäli ongelmia ilmenee, tietoja ei siirretä, vaan ongelmista kerrotaan käyttäjälle, jonka vastuulle jää lähdetietojen korjaaminen.

Esimerkissä olevassa Excel-tiedostossa voisi olla sarake, johon tulisi syöttää vain päivämäärätietoja. Tiedonsiirtosovellus voi ilmoittaa siirtovaiheessa käyttäjälle, mikäli jonkin sarakkeen arvo on virheellinen. Käyttäjälle voidaan antaa erinäisiä vaihtoehtoja ongelman käsittelyyn. Käyttäjä voi ohittaa koko rivin, jolloin sitä ei lisätä tietokantaan. Toinen vaihtoehto voisi olla antaa käyttäjän syöttää käsin virheellisen arvon tilalle kelvollinen arvo. Kolmas vaihtoehto olisi käyttää tyhjää arvoa, mikäli sarakkeeseen hyväksyttäisiin tyhjät arvot. Viimeinen vaihtoehto olisi keskeyttää tietojen siirto kokonaan.

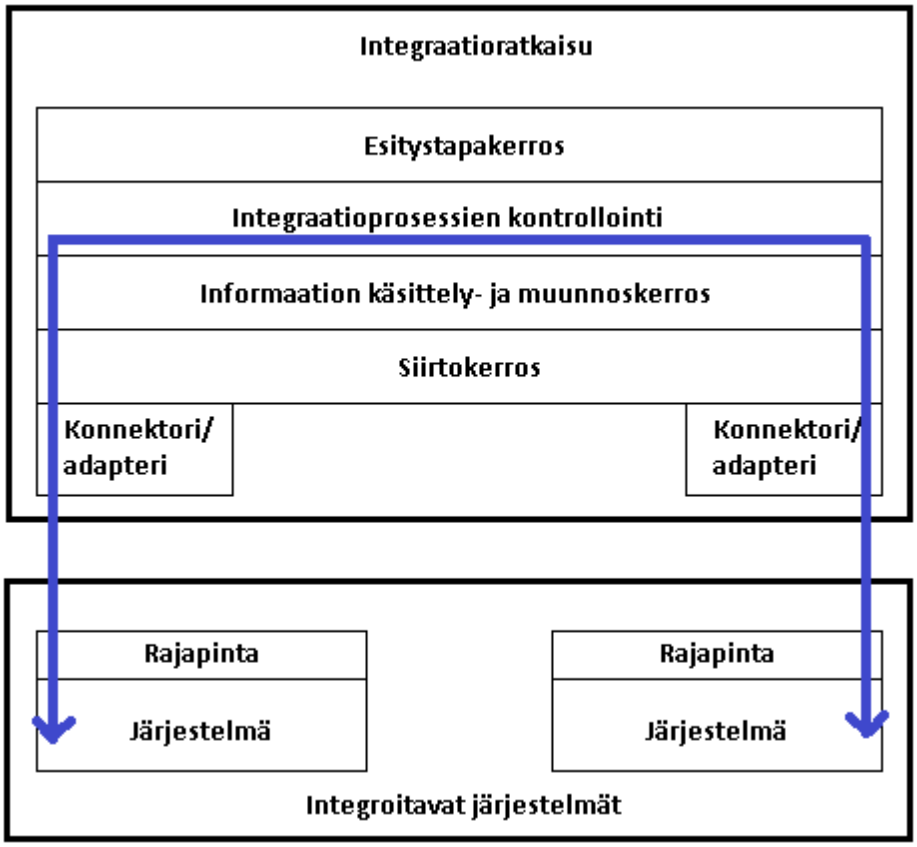
4.3 Tietojärjestelmien integrointi

Järjestelmäintegraatiolla tarkoitetaan tyypillisimmin niitä tapoja ja tekniikoita, joiden avulla muutoin keskenään yhteensopimattomat tietojärjestelmät saadaan kommunikoimaan keskenään. Integraatio on kokoelma toimintatapoja, jotka voivat tehostaa merkittävästi yrityksen toimintaa ja lisätä sen joustavuutta samoin kuin parantaa monitorointia ja raportointia. Järjestelmäintegraatio on määritelty toimintamalleiksi ja tekniikoiksi, joiden avulla voidaan saattaa vähintään kaksi eri toiminnallisuutta tarjoavaa tietojärjestelmää jakamaan informaatiota siten, että informaation siirto ja muunnokset ovat kontrolloitavissa ja monitoroitavissa yhdestä tai useammasta keskitetystä pisteestä. (Tähtinen 2005, 48.)

Integraatioratkaisu koostuu

- informaation siirtämisestä integroitavien järjestelmien välillä
- tietomuunnoksista näiden järjestelmien sisäisten esitysmuotojen välillä, sekä
- kokonaisprosessin (tiedonsiirto sekä tietomuunnokset) kontrolloinnista sekä tähän liittyvästä valvonnasta ja raportoinnista (Tähtinen 2005, 48).

Kuvassa 4.5 on kuvattu integraatioarkkitehtuurin malli:



Kuva 4.5 Integraatioarkkitehtuurin malli (Tähtinen 2005)

Alimpana arkkitehtuurissa ovat integroitavat järjestelmät, jotka eivät sellaisenaan kykene kommunikoimaan keskenään. Jotta ne voisivat keskustella keskenään, ne tarvitsevat ulospäin näkyvät rajapinnat, joita muut järjestelmät voivat käyttää päästäkseen käsiksi järjestelmän tarjoamiin palveluihin.

Mikäli integraatioratkaisua ajatellaan yhtenä toiminnallisena kokonaisuutena, integroitavien järjestelmien rajapintoja vastaan asettuvat erityiset integraatioratkaisun rajapintakomponentit, joista käytetään tyypillisesti nimityksiä konnektori, adapteri tai agentti. (Tähtinen 2005, 71.)

Siirtokerros on tietoverkko tai jokin tietoverkkojen päällä toimiva sanomajärjestelmä, jota pitkin palvelujen kutsuja ja tietoja välitetään. Yritysten sisäisten järjestelmien kommunikoidessa keskenään luonnollinen valinta tietoverkoksi on yrityksen paikallisverkko, Local Area Network (LAN). Yritysten välisessä tiedonsiirrossa käytetään yleensä Internetin välityksellä tapahtuva liikenne. Kun tiedonvälityksessä käytetään jotakin julkista tietoverkkoa kuten Internetiä, on tiedonsiirron tietoturva erittäin tärkeässä asemassa. (Tähtinen 2005, 50.)

Kun halutaan yleiskäyttöistä ja yhtenäistä integraatioratkaisua, joka mahdollistaa periaatteessa minkä tahansa ohjelmistoparin katkottoman kommunikaation, lisätään arkkitehtuurimalliin Informaation käsittely- ja muunnoskerros. Sen tehtävänä on tulkita lähettävän järjestelmän muodostamaa informaatiota sekä muodostaa kyseisestä informaatiosta vastaanottaman järjestelmän ymmärtämä kokonaisuus. Edellytyksenä on, että kerros ymmärtää molempien järjestelmien tietomuodot (formaatit). (Tähtinen 2005, 57-58.)

Integraatioprosessi on tekninen apuväline, jonka avulla liiketoimintaprosesseja voidaan nopeuttaa ja tehostaa. Integraatioprosessien kontrollointi -kerrosta vastaavat yritysten neuvotteluissa käyttämät enemmän tai vähemmän standardoidut tavat ja käytännöt. Järjestelmät voisivat kommunikoida keskenään suoraan rajapintakerroksen avulla kierrättämättä jakamaansa informaatiota tietomuunnos- ja kontrollointikerrosten läpi. Laajojen kokonaisuuksien hallinnassa on kuitenkin tehokkaampaa soveltaa mahdollisimman yleiskäyttöistä mallia. (Tähtinen 2005, 63-64.)

Ylimpänä arkkitehtuurissa on esitystapakerros. Se toimii rajapintana integraatioratkaisun ja ihmisten välillä. Se tarkoittaa näkymiä, joiden avulla yrityksen johto ja työntekijät saavat tietoa integraatioprosessien tilasta. Tämän avulla saadaan tietoa yrityksen liiketoiminnan tilasta. Nämä näkymät ja käyttöliittymät voivat olla integraatioratkaisun sähköpostitse lähettämiä raportteja, mutta myös esimerkiksi monipuolisempia vuorovaikutteisempia web-lomakkeita. (Tähtinen 2005, 71.)

5 SYSTEEMITYÖMALLIT

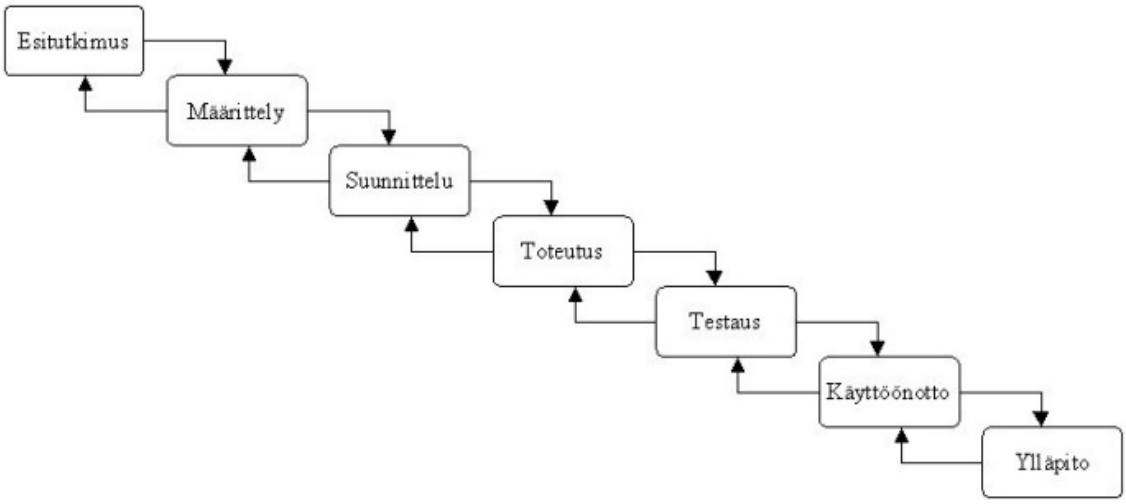
Sovelluskehityksessä käytetään usein hyväksi erilaisia systeemityömalleja. Varsinkin suuremmissa projekteissa systeemityömallit auttavat projektitiimiä näkemään asiat samalla tavalla sekä työskentelemään tehokkaasti yhdessä.

Eri systeemityömalleilla on etunsa ja haittapuolensa. Tässä luvussa esitellään lyhyesti yleisimpiä systeemityömalleja.

5.1 Vesiputousmalli

Vesiputousmalli kehitettiin jo 1960- ja 70-luvun vaihteessa. Sen kehittäjänä mainitaan usein W.W. Royce, joka julkaisi artikkelin aiheesta vuonna 1970. Mallissa tietojärjestelmän kehitys nähdään tasaisesti alaspäin virtaavana vesiputouksena, jossa taaksepäin palaaminen on työlästä. Ideaalitapauksessa työvirta soljuu luontevasti läpi putouksen ja tietojärjestelmä on valmis, mutta näin hyvin harvoin tapahtuu. Vesiputousmalli ei kuvasta kehityksen iteratiivisuutta kovinkaan hyvin, sillä monesti asioita joudutaan korjaamaan eli palaamaan aiempiin vaiheisiin, koska virheitä paljastuu myöhemmissä vaiheissa. Mallin ongelma on myös se, että yleisesti tarkastuspisteet ja dokumentit kiinnitetään eri vaiheiden rajapintoihin, eli toisin sanoen vaiheen loppudokumentti toimii syötteenä seuraavalle vaiheelle. Myöskään varsinaisia tuloksia ei voida asiakkaalle esitellä kuin vasta hyvin myöhäisessä vaiheessa, mikä johtaa siihen, että asiakkaan huomatessa virheitä järjestelmässä, niiden korjaaminen on kallista ja työlästä. (Reinikka 2007.)

Vesiputousmallin eri vaiheet tuottavat dokumentteja. Esitutkimus tuottaa esitutkimusraportin, määrittelyvaihe projektisuunnitelman ja määrittelyraportin, suunnitteluvaihe suunnitteludokumentteja, toteutusvaihe lähdekoodia, testausvaihe testaussuunnitelman ja testausraportit, käyttöönotto käyttöönottosuunnitelman ja käyttöohjeet sekä ylläpito ylläpito-ohjeet. Kuvassa 5.1 on esitetty vesiputousmallin rakenne.



Kuva 5.1 Vesiputousmalli

5.2 Prototyyppimalli

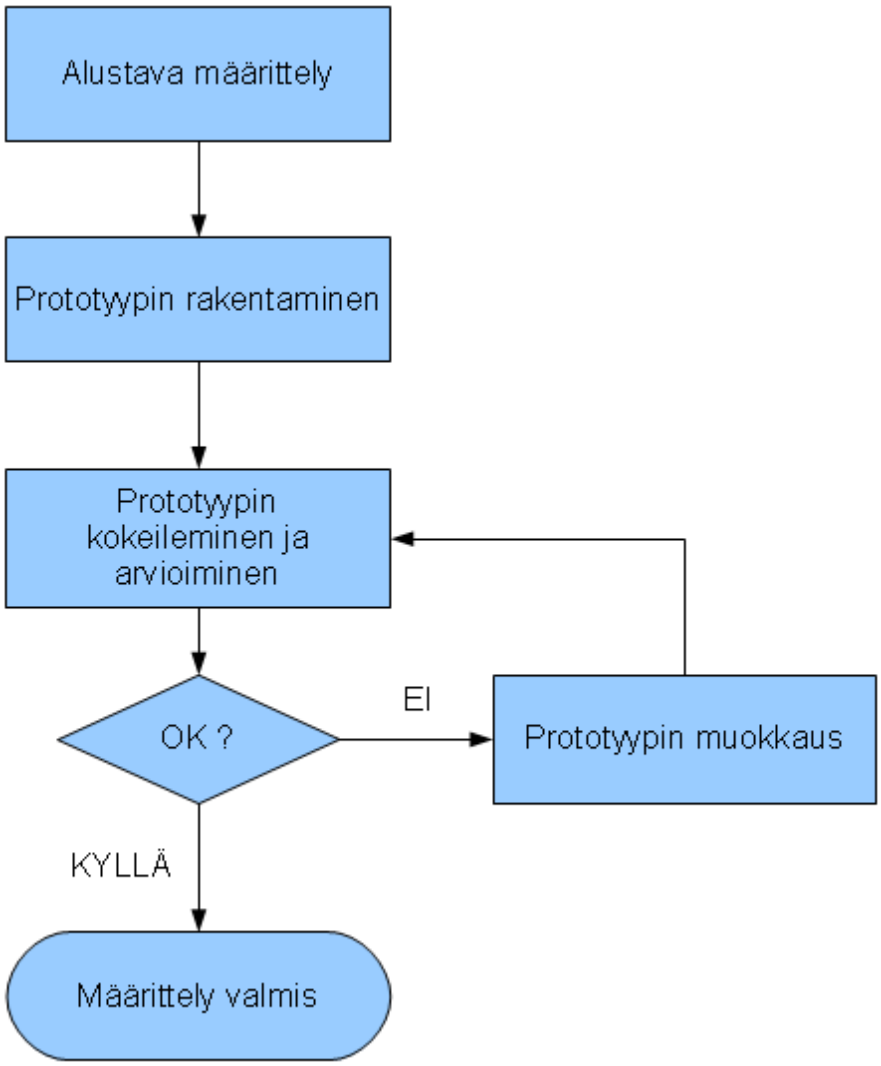
Vesiputousmalli soveltuu huonosti silloin, jos asiakas haluaa nähdä tuloksia nopeasti, ja silloin, kun ei olla aivan täysin varmoja mitä halutaan. Tämän tyyppisiin tilanteisiin soveltuu prototyyppimalli huomattavasti paremmin. (Reinikka 2007)

Malli kierrättää käytännössä elinkaaren viittä keskimmäistä vaihetta iteratiivisesti läpi nopealla tahdilla. Toisin sanoen, kun on tutkittu, mitä asiakas haluaa, ryhdytään määrittelemään, suunnittelemaan ja toteuttamaan ensimmäistä prototyyppiä järjestelmästä. (Reinikka 2007.)

Kun ensimmäinen prototyyppi on valmistettu, voidaan se näyttää asiakkaalle ja keskustella, mitkä asiat ovat oikein, mitkä väärin ja mitä puuttuu. Tämän jälkeen aloitetaan kierros uudestaan ja luodaan toinen prototyyppi ja niin edelleen, kunnes tietojärjestelmä on valmis. Prototyyppimallikaan ei ole ongelmaton, koska monen prototyypin teko kuluttaa resursseja. On myös mahdollista, että prototyypit eivät paljasta kaikkia mahdollisia virheitä, vaikka niitä testattaisiinkin. Olennaisen tärkeää onkin testata lopullinen tuote läpikotaisin. Tosin tämä johtaa siihen, että mahdollisia virheitä löydetään vielä kehityksen loppuvaiheessa, joka tarkoittaa kustannusten kasvua kuten aiemmin on mainittu. (Reinikka 2007.)

Kyseinen malli on määrittelyn tukena toimiva prototyyppimalli. Kun prototyyppi on valmis, luodaan valmis tuote prototyyppiä ohjenuorana käyttäen alusta alkaen.

Kuvassa 5.2 on esitelty määrittelyn tukena toimivan prototyyppimallin rakenne.

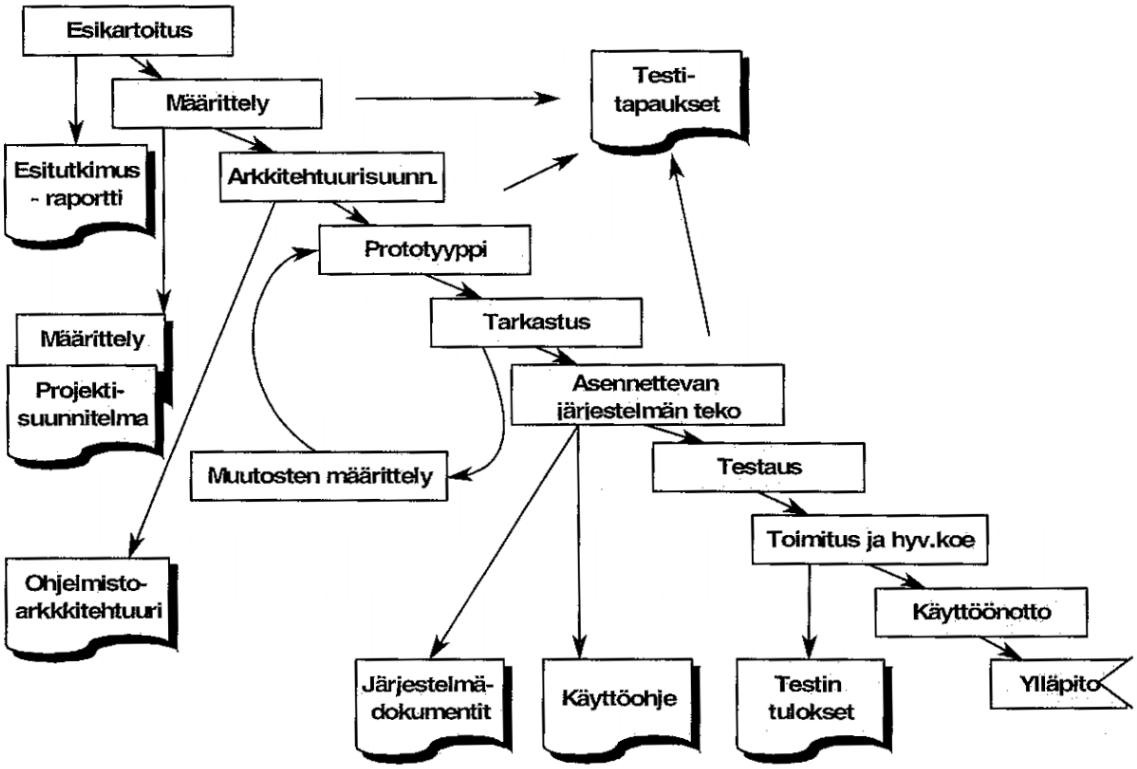


Kuva 5.2 Prototyyppimalli määrittelyn tukena

Prototyyppimallista on myös toinen versio, joka edustaa iteroivaa kehitystä. Tällöin prototyyppi kehitetään valmiiksi järjestelmäksi. Mallissa prototyyppiin rakennetaan oleelliset osat. Joitakin toiminnallisia ja käyttäjälle näkymättömiä

osia, kuten virheentarkistus, ohjeistukset ja optimoitu toiminta, jätetään toteuttamatta. Kun prototyyppi on riittävän pitkällä ja asiakas hyväksyy prototyypin, siirrytään varsinaiseen toteutusvaiheeseen, jolloin puuttuvat osat toteutetaan.

Ongelmaksi protoilussa tulee helposti se, että asiakas saattaa luulla lähes oikealta näyttävän järjestelmän olevan jo käytännöllisesti katsoen valmis, vaikka valtaosa työstä olisi vielä tekemättä. Tämän takia prototyyppiä ei aina kannata tehdä mahdollisimman viimeistellyn näköistä ja tuntuista. Asiakkaan on huomattava, että järjestelmä on vielä keskeneräinen. (Haikala 2000.)

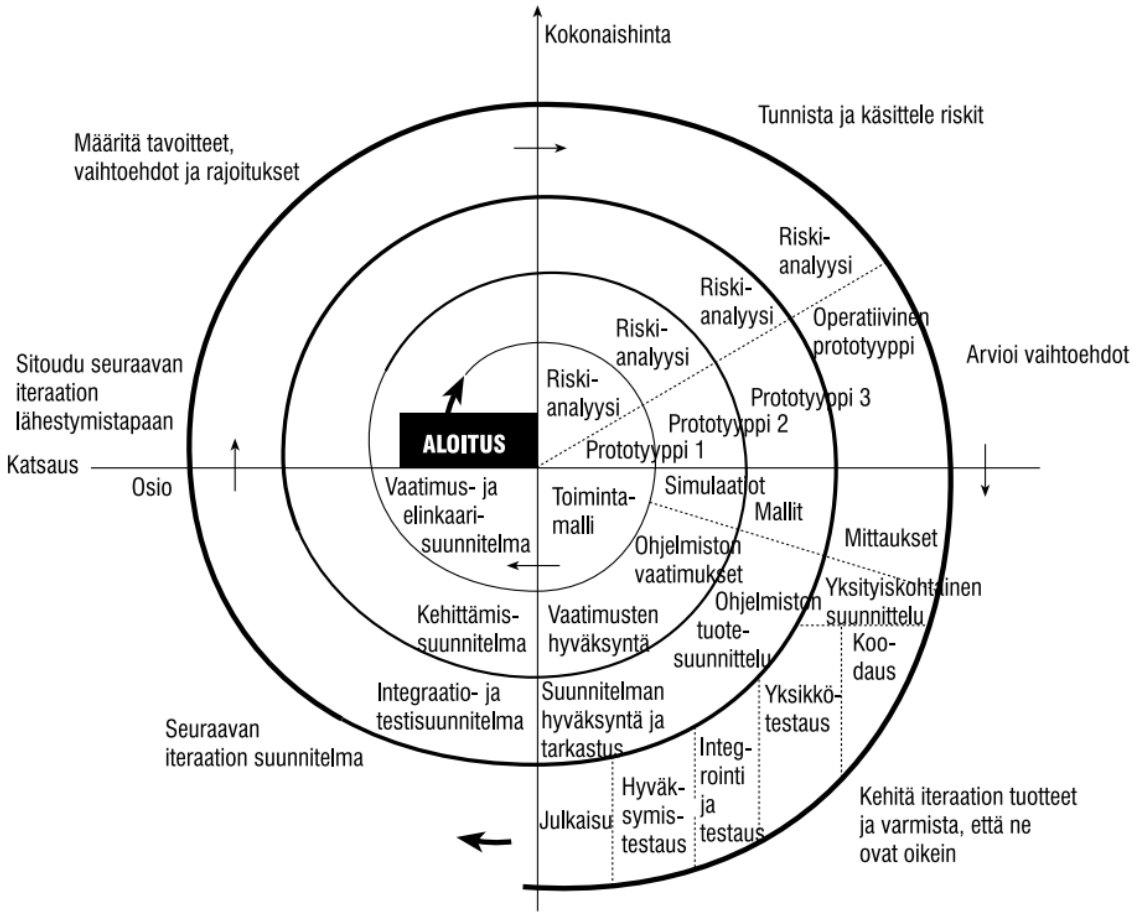


Kuva 5.3 Prototyyppimalli iteroivan kehityksen osana (Haikala 2000)

5.3 Spiraalimalli

Spiraalimalli on riskeihin suuntautunut elinkaarimalli, joka pilkkoo ohjelmistoprojektin miniprojekteiksi. Jokainen miniprojekti keskittyy yhteen tai useampaan riskiin, kunnes kaikki pääriskit on hoidettu. Pääriskien käsittelyn jälkeen spiraali-

malli päättyy kuten vesiputousmallikin. Kuvassa 5.4 on kuvattu spiraalimalli. (McConnell 2002)



Kuva 5.4 Spiraalimalli (McConnell 2002)

Spiraalimallissa aloitetaan pienellä mittakaavalla (kuvan keskusta), tutkitaan riskit, tehdään riskienkäsittelysuunnitelma ja sitoudutaan seuraavan iteraation lähestymistapaan. Kukin iteraatio vie projektin suurempaan mittakaavaan. Spiraalia kierretään yksi kerros kerrallaan. (McConnell 2002.)

Jokainen iteraatio sisältää spiraalin ulkolaidoilla merkityt askeleet, jotka ovat:

- tavoitteiden, vaihtoehtojen ja rajoitusten selvitys
- riskien tunnistaminen ja selvittäminen
- vaihtoehtojen puntarointi

- iteraatioon kuuluvien toimitusten kehitys ja oikeellisuusvarmistukset
- seuraavan iteraation suunnittelu
- seuraavan iteraation lähestymistapaan sitoutuminen tarvittaessa (McConnell 2002).

Spiraalimallissa aikaisemmat iteraatiot ovat halvimpia. Toimintaperiaatteiden kehitykseen kuluu vähemmän aikaa kuin vaatimusten määrittelyyn ja vaatimuksiin kuluu vähemmän aikaa kuin suunnitelmaan, tuotteen toteuttamiseen ja testaamiseen. (McConnell 2002.)

5.4 Ketterä ohjelmistokehitys (Agile-menetelmät)

Ketterissä menetelmissä (Agile methods) pyritään järkevään suhteeseen dokumentoinnissa, suunnittelussa ja koodaamisessa niin, että pääpaino on asiakkaan tyytyväisyydellä ja toimivalla ohjelmalla.

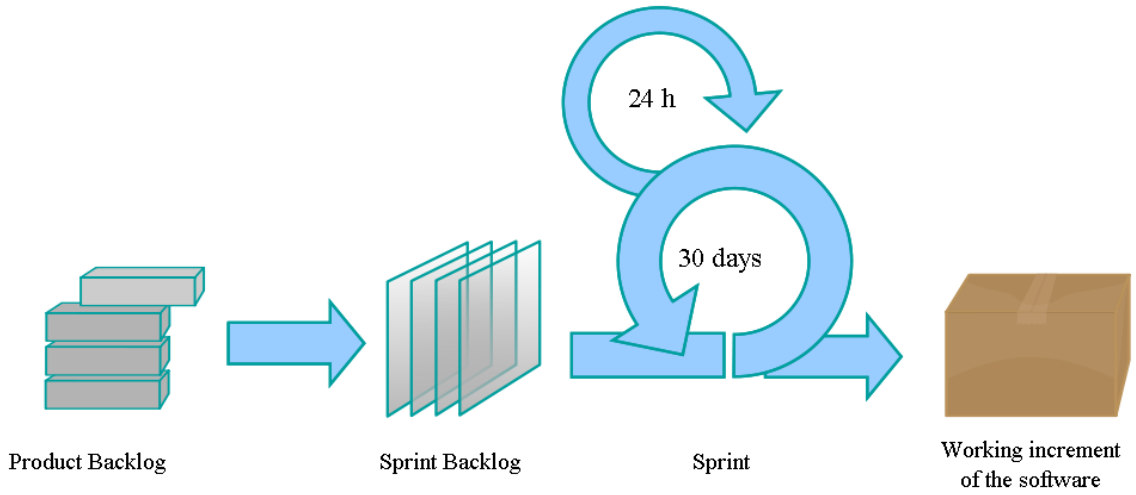
Ketterät menetelmät perustuvat pääasiassa seuraaviin määriteltyihin periaatteisiin (Kosonen 2005):

1. Tärkeintä on täyttää asiakkaan vaatimukset julkaisemalla jatkuvasti ja aikaisin uusia hyödyllisiä versioita ohjelmistosta.
2. Hyväksytään ja otetaan vastaan muuttuvat vaatimukset, jopa kehityksen loppuvaiheessa. Ketterät menetelmät valjastavat muutoksen asiakkaan kilpailueduksi.
3. Luovutetaan toimivia versioita kehitettävästä ohjelmistosta säännöllisesti, mielellään lyhyin väliajoin muutamasta viikosta muutamaan kuukauteen.
4. Liiketoiminnan ammattilaisten ja kehittäjien täytyy työskennellä päivittäin yhdessä koko projektin ajan.
5. Rakennetaan projektit motivoituneiden yksilöiden ympärille ja annetaan heille ympäristö ja tuki, jota he tarvitsevat, sekä luotetaan, että he saavat työn tehtyä.

6. Kaikkein tehokkain tapa välittää tietoa kehitystiimille ja kehitystiimissä on kasvokkain tapahtuva keskustelu.
7. Toimiva ohjelmisto on ensisijainen edistymisen mitta.
8. Ketterät menetelmät suosivat kestävää kehitystä. Rahoittajien, kehittäjien ja käyttäjien tulisi kyetä pitämään jatkuvasti yllä tasainen työtahti.
9. Jatkuva huomion kiinnittäminen tekniseen laatuun sekä hyvään rakenteeseen ja suunnitteluun, lisää ketteryyttä.
10. Yksinkertaisuus - taito maksimoida työn määrä, jota ei tarvitse tehdä - on olennaista.
11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat nousevat itseorganisoituvista tiimeistä.
12. Tasaisin väliajoin tiimi miettii, miten voisi tulla entistä tuottavammaksi, ja sitten säätää ja muokkaa toimintaansa sen mukaisesti.

Yksi yleisimmistä Agile-menetelmistä on Extreme Programming, jonka päällimmäisiä arvoja ovat kasvokkain tapahtuva kommunikointi, ohjelmakoodin ja rakenteen yksinkertaisuus, jatkuva palautteen kerääminen asiakkaalta sekä rohkeus nähdä virheet silloin kun kehitysprosessi on ajautumassa väärään suuntaan. Toinen yleinen Agile-mentelmä on Scrum, jossa lähtökohtana on hallita ohjelmistotuotantoprosessia sekä hallita ja tukea muuttuvia vaatimuksia.

Kuvassa 5.5 on kuvattu Scrum-menetelmä.



Kuva 5.5 Scrum-menetelmä (Wikipedia: Scrum process 2010)

Scrum-menetelmässä käytetään kolmea eri työlistaa, jotka ovat tuotteen työlista (product backlog), julkaisun työlista (release backlog) sekä toteutusvaiheen työlista (sprint backlog).

Scrumissa työskennellään iteratiivisesti ja inkrementaalisesti, tavoitteena oleva tuote rakentuu pikku hiljaa täydellisemmäksi ja valmiimmaksi useiden toteutuskierrosten aikana. Toteutuskierrosta kutsutaan pyrähdykseksi (Sprint), eli sprintiksi. Sprintin kesto on noin 2—4 viikkoa. Jokaisen sprintin sisältö sovitaan ennen periodin aloitusta, ja tehtäväksi valitaan vain niitä asioita, joilla on sillä hetkellä suurin merkitys projektin onnistumisen kannalta. Sprintin lopuksi järjestetään demo-tilaisuus, jossa projektitiimi konkreettisesti esittelee sprintin saavutukset (esim. uusin versio ohjelmistosta). (Wikipedia: Scrum 2010).

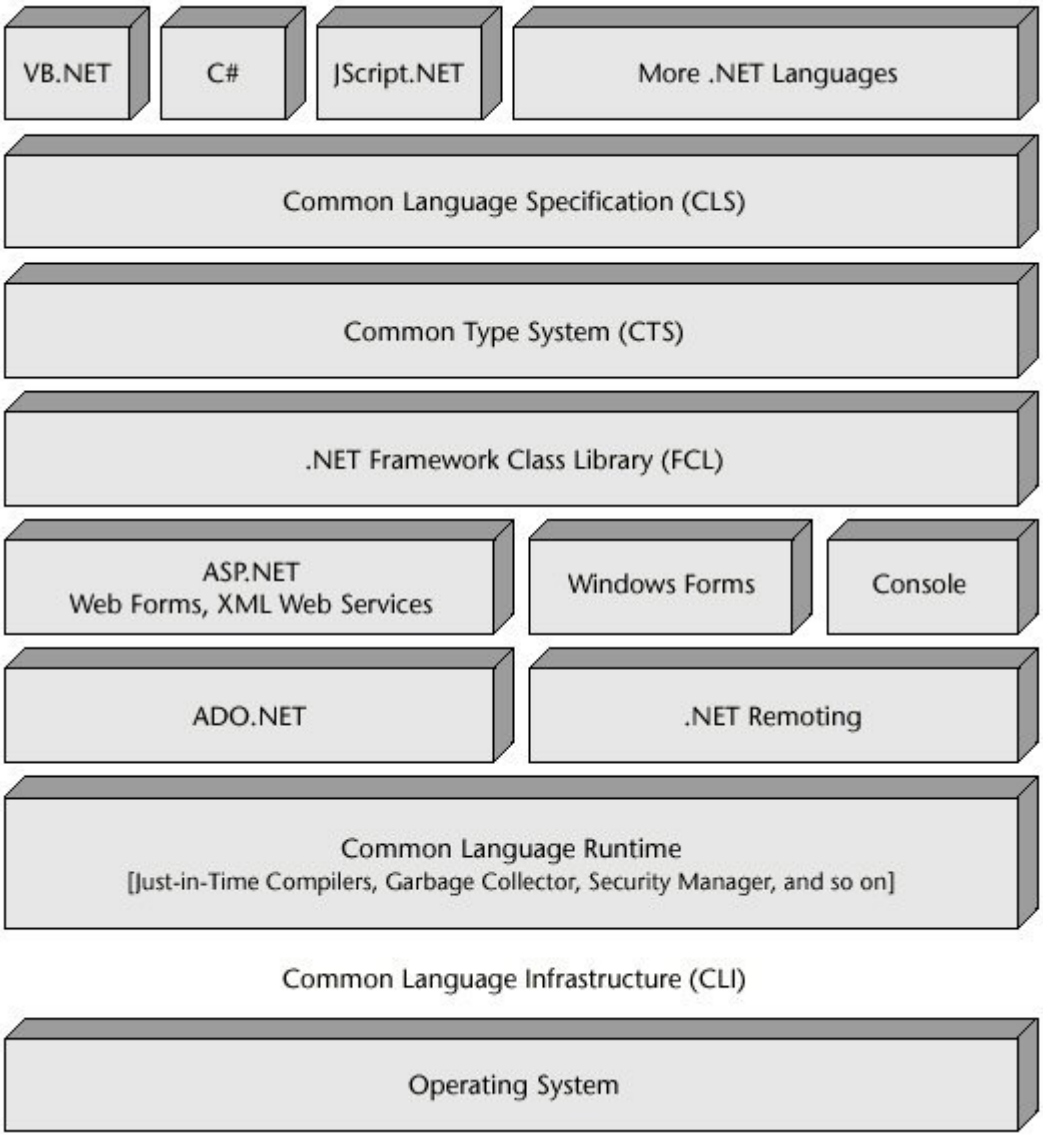
6 KÄYTETYT TEKNIIKAT

Projektin toteutuksessa oli käytössä tässä luvussa esiteltyjä ohjelmistokehitykseen liittyviä tekniikoita ja apuvälineitä.

6.1 .NET Framework

.NET Framework on Microsoftin luoma ohjelmistokomponenttikirjasto, jota Visual Studion .NET yhteensopivat versiot käyttävät. Se koostuu kahdesta osasta, luokkakirjastosta ja Common Language Runtime -ajoympäristöstä (CLR). CLR on kehitetty monien erilaisen ohjelmointikielien saumatonta yhteistyötä varten. .NET ympäristössä kaikki koodi lähdekielestä riippumatta käännetään välikoodiksi, joka suoritetaan CLR:n tarjoamassa virtuaalikoneessa. Virtuaalikone kääntää välikoodin JIT-kääntäjällä lopulliseen koneen ymmärtämään binäärimuotoon. CLR vastaa muistinhallinnasta, tietoturvasta ja poikkeusten käsittelystä. .NET Frameworkin viimeisin versio on 4.0, joka julkaistiin huhtikuussa 2010. (Wikipedia: .NET Framework 2010).

Kuvassa 6.1 on kuvattu .NET Frameworkin kerroksittainen rakenne.



Kuva 6.1 .NET Frameworkin rakenne (CodeGuru 2004)

6.2 C#

C# on Microsoftin .NET-projektiaan varten kehittämä olio-ohjelmointikieli. C# on sen johdosta vahvasti sidottu .NET-ympäristöön ja sen kääntäjä tuottaa välikoodia, joka suoritetaan CLR:n tarjoamassa virtuaalikoneessa. Kieli on läheisintä sukua C++:lle, mutta myös Javasta ja muista nykyaikaisista kielistä on otettu vaikutteita. C# on vahvasti tyypitetty, siinä on automaattinen roskien keruu ja se ei normaalisti tue suoria muistiviittauksia. Vahva tyypitys tarkistuksineen varmistaa, että käyttäjä ei voi tehdä hallitsemattomia tyyppimuunnoksia tai osoittaa taulukon ulkopuolelle. Tarvittaessa voidaan kuitenkin esitellä luokka tai metodi

vaaralliseksi (*unsafe*), joka mahdollistaa C++:sta tuttuja ominaisuuksia, kuten osoittimet ja staattisesti varatut taulukot. Roskien keruu ei toimi vaaralliseksi esiteltyjen osien kanssa vaan muistinhallinta on ohjelmoijan vastuulla niiltä osin. (Wille 2001.)

Kuvassa 6.2 on näyte C#-ohjelmakoodista.

```
//C#Program
static public void Main()
{
    StreamWriter sw=new StreamWriter("date.txt ",true);
    DateTime dt=DateTime.Now;
    string datestring=dt.ToShortDateString()+" "+
    dt.ToShortTimeString();
    sw.WriteLine(datestring);
    sw.Close();
    StreamReader sr=new StreamReader("date.txt ");
    string filetext=sr.ReadToEnd();
    sr.Close();
    Console.WriteLine(filetext);
}
```

Kuva 6.2 C#-ohjelmakoodia (CodeGuru 2004)

6.3 Microsoft SQL Server 2005

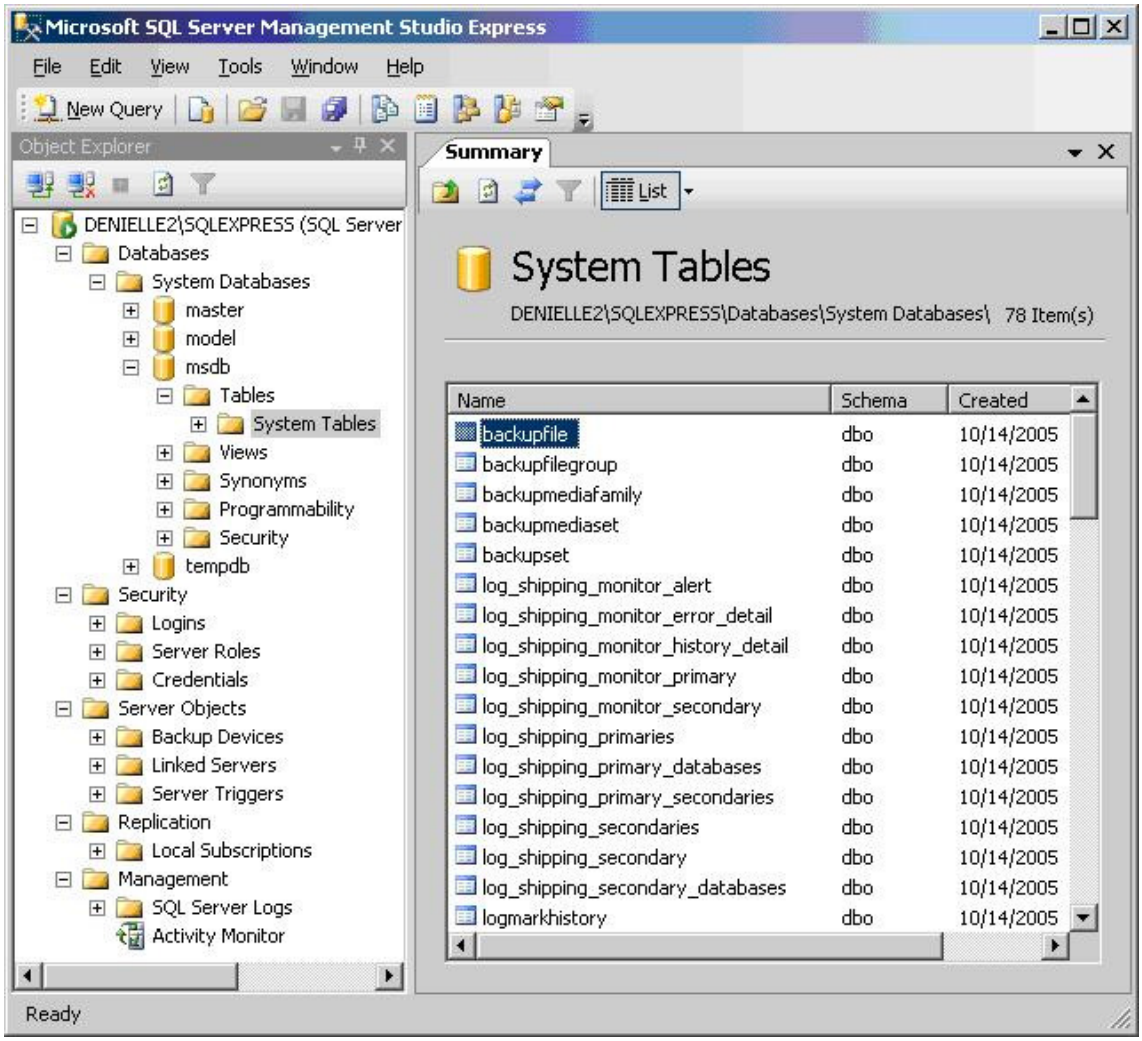
Microsoft SQL Server on relaatiotietokantojen hallintaan tarkoitettu palvelinohjelmisto. Se tukee Transact-SQL -laajennusta, joka mahdollistaa uusia toimintoja SQL-kielen standardiversioon verrattuna. Uusia ominaisuuksia ovat paikalliset muuttujat, laajennettu tuki merkkijonojen ja päivämäärien käsittelyyn sekä laajennetut Insert- ja Delete-lauseet. (Kettinen 2008.)

Serverin toiminnot jakautuvat kolmeen kerrokseen. Alin SQLOS-kerros vastaa yleensä käyttöjärjestelmälle kuuluvista toiminnoista. Näitä ovat säikeiden vuorotus, muistinhallinta ja lukkiutumisen estäminen. Keskimmäisenä oleva Relation Engine -kerros toteuttaa alemman kerroksen palvelujen avulla varsinaisen relaatiotietokannan. Ylimpänä toimiva protokollakerros vastaa kommunikoinnista muiden järjestelmien kanssa. (Kettinen 2008.)

MS SQL Server on toiminnoiltaan kehittyneempi kuin kevyeen tietokantakäyttöön tarkoitettu Microsoft Access. Samanaikaisuuden hallinta on tehokkaampaa, sillä ainoastaan tarvittavat rivit lukitaan eikä kokonaisia tauluja. SQL Server tarjoaa myös laajempia palveluja helpottamaan sen käyttöä. Näitä ovat varmuuskopioiden hallinta, datan analysointityökalut sekä raportointi- ja integrointipalvelut. (Kettinen 2008.)

Järjestelmästä on olemassa ilmainen Express Edition. Ilmaisversio pitää sisällään ainoastaan keskeisimmät toiminnot. Sen tehoa on laskettu rajoittamalla laitteistotuki yhteen prosessoriin ja neljään gigatavuun keskusmuistia. Express Edition sisältää ainoastaan karsitun version tietokannan hallintasovelluksesta. (Kettinen 2008.)

Kuvassa 6.3 on Microsoft SQL Server Express -version hallintatyökalu Microsoft SQL Server Management Studio Express.



Kuva 6.3 Microsoft SQL Server Management Studio Express (Global Classroom Training 2009)

6.4 OLE DB

Object Linking and Embedding, Database (OLE DB) on Microsoftin kehittämä rajapinta tiedon käsittelyyn. Tietoa voidaan käsitellä samankaltaisesti riippumatta tiedon muodosta, kunhan sille on olemassa ajuri. Käsiteltävä tieto on yleensä tietokantamuodossa, mutta OLE DB-ajureita on myös esimerkiksi tekstitiedostojen, Excel-taulukoiden ja kolmansien osapuolten tietojärjestelmien käsittelyyn. (Microsoft 2010.)

7 PROJEKTIN KULKU

Opinnäytetyöni toteutettiin eri vaiheissa, jotka tapahtuivat osittain päällekkäin. Työ toteutettiin omalla vapaalla toteutusmallilla, jolla on joitakin yhtäläisyyksiä Agile-menetelmien kanssa, mutta joka ei ollut puhdas Agile-menetelmä.

Projektia seurattiin ja toteutettiin säännöllisillä ohjauspalavereilla, joihin osallistuivat itseni lisäksi asiakas Heikki Hepomäki sekä opinnäytetyön ohjaaja Martti Ylä-Jussila. Projektiorganisaatio koostui yllämainituista henkilöistä. Palavereissa käytiin läpi edellisen palaverin jälkeen tehdyt työt, senhetkinen tilanne sekä käsiteltiin mahdollisesti eteen tulleet ongelmat ja / tai uhat.

Projektin vaiheet olivat seuraavat:

- vaatimusten kartoitus
- Huoltokirja-järjestelmän opiskelu
- valmiiden sovellusten kartoitus ja ratkaisujen tutkiminen
- toteutuksen suunnittelu
- toteutus ja testaus
- käyttöönotto.

7.1 Vaatimusten kartoitus

Ensimmäisessä vaiheessa kartoitin vaatimuksia kehitettävälle sovellukselle. Vaatimusten kartoitusta tehtiin projektiorganisaation kesken palavereissa. Tässä vaiheessa kartoitettiin pääasiassa toiminnallisia vaatimuksia, mutta myös joitakin erinäisiä teknisiä vaatimuksia tuli jo tässä vaiheessa eteen. Vaatimuskartoituksessa ei kuitenkaan menty toteutuksen yksityiskohtiin. Käyttöliittymältä vaadittiin helppokäyttöisyyttä, koska tietojen siirtoa tehdään harvoin ja tekijät eivät ole tietotekniikan ammattilaisia. Tämä vaihe oli suhteellisen nopea, koska vaatimuksia ei käytännössä ollut kovinkaan paljoa. Vaatimuksena oli tuottaa käytännössä Excel-tiedostoja lukeva ja tiedot tietokantaan kirjoittava sovellus, joka hoitaisi päällekkäisyyksien hallinnan ja muutamien yleisten syötevirheiden tulkinnan. Vaatimuksista laadittiin tekstimuotoinen tiedosto.

7.2 Huoltokirja-järjestelmän opiskelu

Toisessa vaiheessa opiskelin Huoltokirja-järjestelmän, joka toimi tietojensiirron kohdejärjestelmänä. Koska olin ollut jo mukana Huoltokirja-järjestelmän ylläpito- ja toteutusvaiheissa, järjestelmä oli jo tuttu ennestään. Tämä vaihe toimi lähinnä järjestelmän toimintojen kertaamisena ja oli nopeasti ohi.

7.3 Valmiiden sovellusten kartoitus ja ratkaisujen tutkiminen

Kolmas vaihe työssä oli tutkia, onko ongelmaan jo olemassa olevia yleispäteviä sovelluksia. Sovelluksissa oli myös oleellista se, etteivät ne olisi liian kalliita yrityksen käyttöön. Koska riittävän selkeitä ja edullisia vaihtoehtoja ei löytynyt, tutkin olemassa olevia ratkaisuja Internet-lähteistä ja kirjasin ideoita ylös myöhempää käyttöä varten.

7.4 Toteutuksen suunnittelu

Neljännessä vaiheessa suunnittelin itse tiedonsiirron toteutusta. Kolmas ja neljäs vaihe kulkivat osittain päällekkäin, koska suunnittelin sovellusta samalla kuin tutkin aiheita. Olin ehtinyt jo suunnitella suuren osan sovelluksesta, kun huomasin, että olin tullut pitkälti samoihin johtopäätöksiin hahmotelmassani kuin Internet-lähteet. Tämä lisäsi varmuutta siitä, että sovellus olisi mahdollista kehittää, eikä suurempia ongelmia luultavasti tulisi eteen.

Suunnittelu tapahtui käytännössä kirjaamalla tekstitiedostoon toteutusideoita sitä mukaa kuin niitä ilmeni. Kun ideoita oli mielestäni riittävästi valmiin toteutuksen kunnolliseen suunnitteluun, tein tarkempaa suunnittelua erilaisten kuvitteellisten esimerkkitapausten pohjalta.

7.5 Toteutus ja testaus

Viides vaihe oli sovelluksen toteutus ja testaus, joihin käytin suurimman osan ajasta. Toteutuksessa otin päämääräkseni tehdä sovelluksen yhden osa-alueen kerrallaan. Osa-alueiksi hahmottelin seuraavat:

- käyttöliittymän alustava toteutus
- tietokannan taulujen rakenteen lukeminen
- Excel-tiedoston rakenteen lukeminen

- tiedon kohde- ja lähdemuodon keskinäisten yhteyksien määritys
- tiedon luku Excel-tiedostosta
- tiedon muunto lähdemuodosta XML-muotoon yhteysmääritysten pohjalta
- tiedon siirto XML-muodosta kohdemuotoon (SQL-tietokantaan).

Aluksi toteutin alustavan käyttöliittymä ilman toiminnallisuutta. Kun käyttöliittymä oli valmis, siirryin toiminnallisuuden lisäykseen.

Jokaisen osa-alueen toteutin ensin erikseen C#-ohjelmointikielellä. Kun osa-alue oli valmis, testasin sen. Jos ongelmia ilmeni, palasin toteutusvaiheeseen ja toistin testauksen. Jatkoin prosessia, kunnes osa-alue oli toimiva. Jokaisen osa-alueen kohdalla myös täydensin tai muokkasin tarvittaessa käyttöliittymää.

7.6 Käyttöönotto

Kuudes ja viimeinen vaihe oli sovelluksen käyttöönotto, joka tapahtui syyslokakuun vaihteessa 2010. Käyttöönottoa oli jo ennakoitu ottamalla asiakkaan tietokannasta kopio ja ajamalla sovelluksella eräajoja siihen. Eräajotietojen ajo tietokannan kopioon todettiin onnistuneeksi, ja samojen tietojen ajo suoritettiin myös asiakkaan varsinaiseen tietokantaan ongelmitta. Asiakas tarvitsee vielä todennäköisesti koulutusta sovelluksen käyttöön, tai vaihtoehtoisesti asiakas teettää tulevat siirtotyöt Cadimage Oy:llä.

7.7 Projektissa käytetyt systeemityömallit

Projektin kehitysprosessissa yhdisteltiin elementtejä useasta eri systeemityömallista. Kehitystyö sujui osin vesiputousmallin mukaisesti. Ensin oli tutkittava, onko järjestelmää mahdollista rakentaa ja millaisia ongelmia eteen saattaa tulla. Tämän jälkeen projekti siirtyi määrittelyvaiheeseen. Koska olin sovelluskehitystiimin ainoa ohjelmoija ja määrittelijä, ei projektin eteneminen edellyttänyt paljoakaan kommunikointia tai asiakastarpeiden kartoittamista alkuvaiheen asiakaspalavereiden jälkeen.

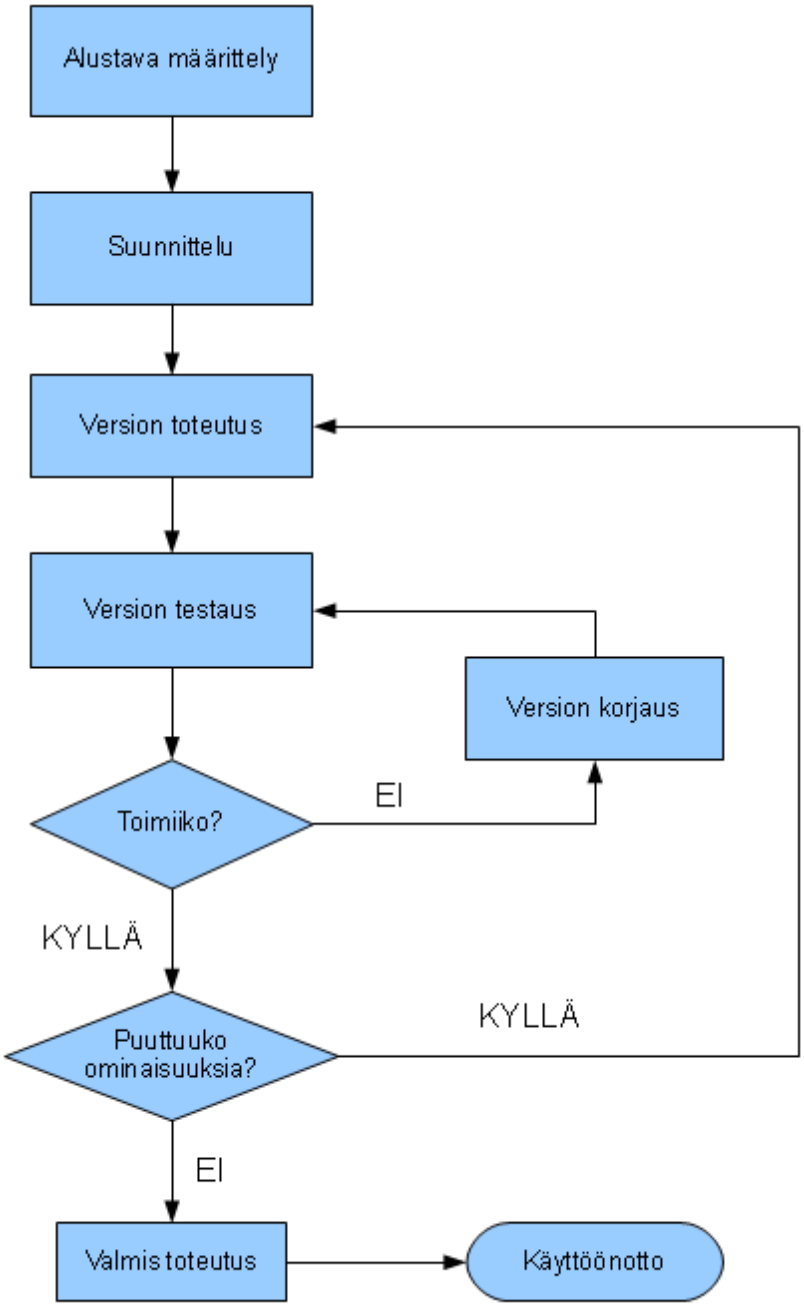
Suunnittelu-, toteutus- ja testausvaiheessa projektin eteneminen ei enää noudattanut täysin vesiputousmallia, koska näitä kolmea vaihetta toistettiin monta

kertaa. Vaikka kaikki tärkeimmät ominaisuudet oli määritelty jo aiemmin, tuli tässä vaiheessa eteen silti uusia vaatimuksia ja ominaisuuksia, joita tuli suunnitella, toteuttaa ja testata vuorotellen.

Kun järjestelmän vaaditut ominaisuudet oli toteutettu, jätettiin toissijaiset ominaisuudet toistaiseksi toteuttamatta ja siirryttiin käyttöönottovaiheeseen, jolloin vesiputousmalli jatkui loppuun asti.

Työskentelytavoissa dokumentoinnin osuus oli minimoitu. Dokumentointia oli lähinnä kehitysideoiden vapaa ylöskirjaaminen sekä runsas ohjelmakoodin kommentointi. Projektista ei syntynyt muita virallisia dokumentteja kuin alun esitutkimus ja projektisuunnitelma. Minkäänlaista määrittelydokumenttia projekti ei tuottanut. Projektin edetessä pidin kuitenkin työpäiväkirjaa, johon kaikki projektin edetessä tulevat asiat kirjattiin ylös.

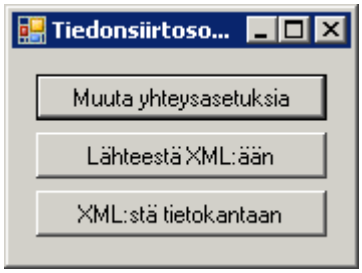
Kuvassa 7.1 on kuvattu karkeasti projektini kulku.



Kuva 7.1 Oman projektin kulku

8 VALMIIN JÄRJESTELMÄN TOIMINNOT

Tässä luvussa esitellään valmiin järjestelmän toiminta eri vaiheissa. Kuvassa 8.1 on tiedonsiirtosovelluksen aloitusvalikko.

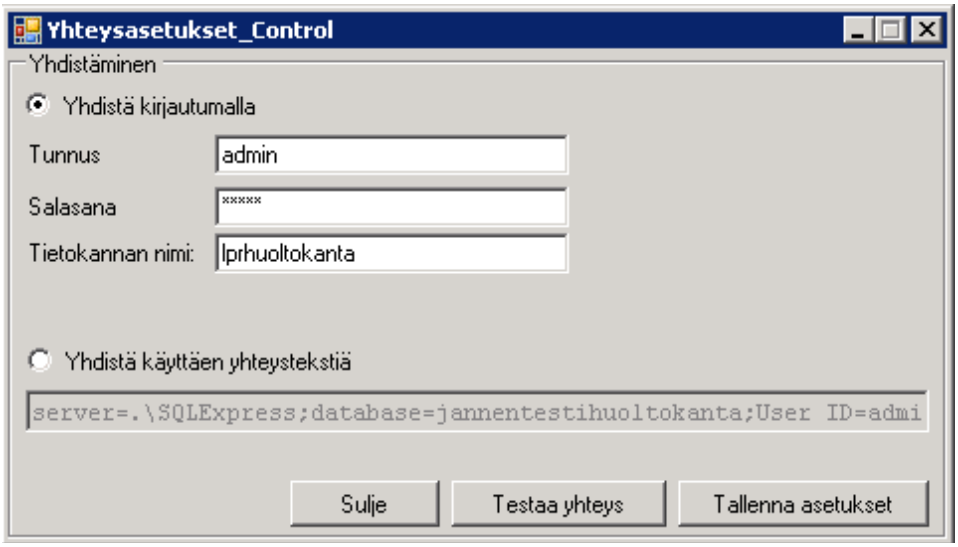


Kuva 8.1 Tiedonsiirtosovelluksen aloitusvalikko

Valikosta voidaan muuttaa yhteysasetuksia, kopioida tietoja lähteestä XML-tiedostoon sekä XML-tiedostosta tietokantaan.

8.1 Yhteysasetusten muuttaminen

Muuta yhteysasetuksia -painikkeesta päästään muokkaamaan yhteysasetuksia, joita käytetään, kun kopioidaan tietoja XML-tiedostosta tietokantaan. Painike avaa kuvan 8.2 mukaisen lomakkeen.



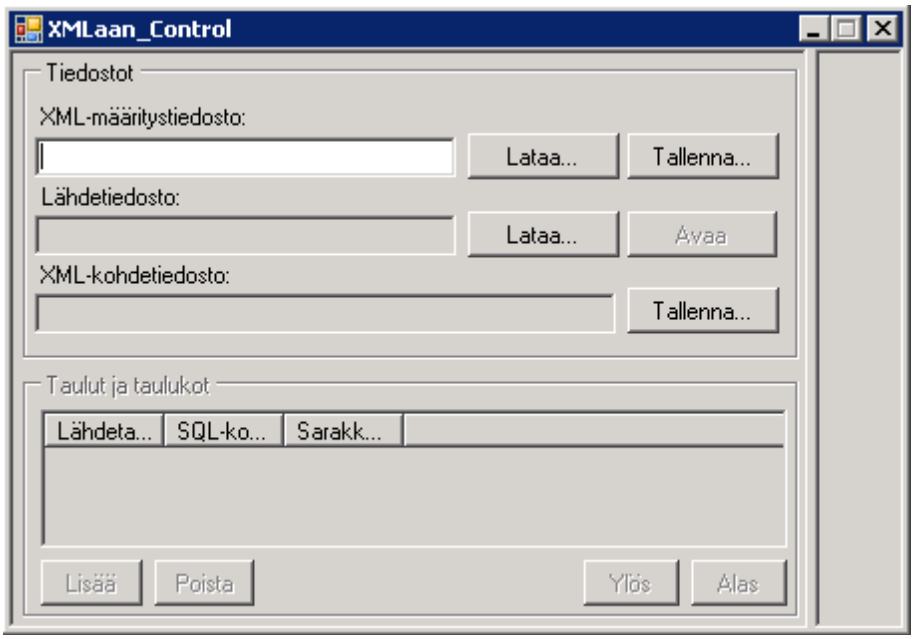
Kuva 8.2 Yhteysasetusten muuttaminen

Lomakkeelta voidaan asettaa tietokantayhteys määrittämällä tunnus, salasana sekä tietokannan nimi. Vaihtoehtoisesti yhdistäminen tietokantaan voidaan määritellä käyttäen yhteystekstiä (connection string). Yhteyden määrityksen jälkeen yhteyden toiminta voidaan testata **Testaa yhteys** -painikkeella ja asetukset tallennetaan **Tallenna asetukset** -painikkeella.

8.2 Määritystiedoston luominen

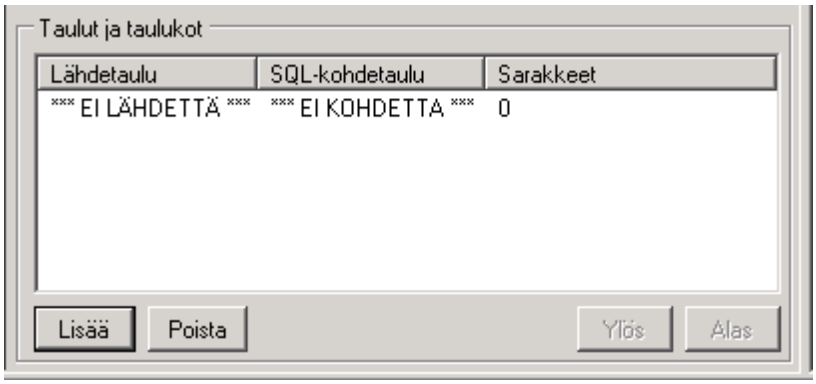
Ennen kuin tietoa voidaan lukea lähteestä XML-tiedostoon, on määritettävä säännöt sille, miten tietoa luetaan, ja mihin muotoon se tallennetaan. Jos määritystiedostoa ei ole luotu, se on tässä vaiheessa luotava.

Määritystiedosto luodaan painamalla päävalikosta **Lähteestä XML:ään** -painiketta, jonka jälkeen näytölle avautuu kuvan 8.3 mukainen lomake.



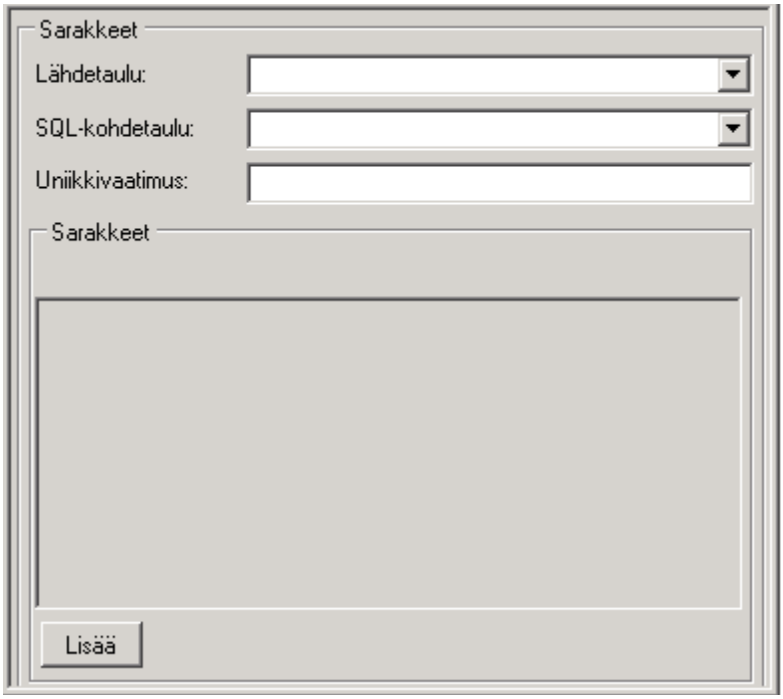
Kuva 8.3 XML-määrityksen luominen (vaihe 1)

Koska määritystiedostoa ei vielä ole, painetaan **Lähdetiedosto**-tekstilaatikon vieressä **Lataa**-paniketta ja valitaan tiedoston avaus -ikkunasta Excel-lähdetiedosto. Seuraavaksi lisätään lähteen ja kohteen väliset yhteydet. Yksittäisen tietokantataulun ja Excel-tiedoston välilehden välinen yhteys lisätään painamalla lomakkeen vasemmassa alalaidassa olevaa **Lisää**-painiketta. Tämän jälkeen **Taulut ja taulukot** -listalle tulee uusi rivi kuvan 8.4 mukaisesti.



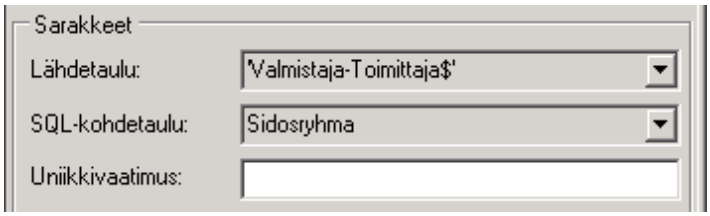
Kuva 8.4 XML-määrityksen luominen (vaihe 2)

Kuten kuvasta 8.4 näkyy, valitulla yhteydellä ei ole määritelty Excel-lähdetaulua eikä tietokannasta kohdetaulua. Seuraavaksi valitaan luotu rivi, jolloin ikkunan oikealle puolelle tulee näkyviin määrityksen luominen kuvan 8.5 mukaisesti.



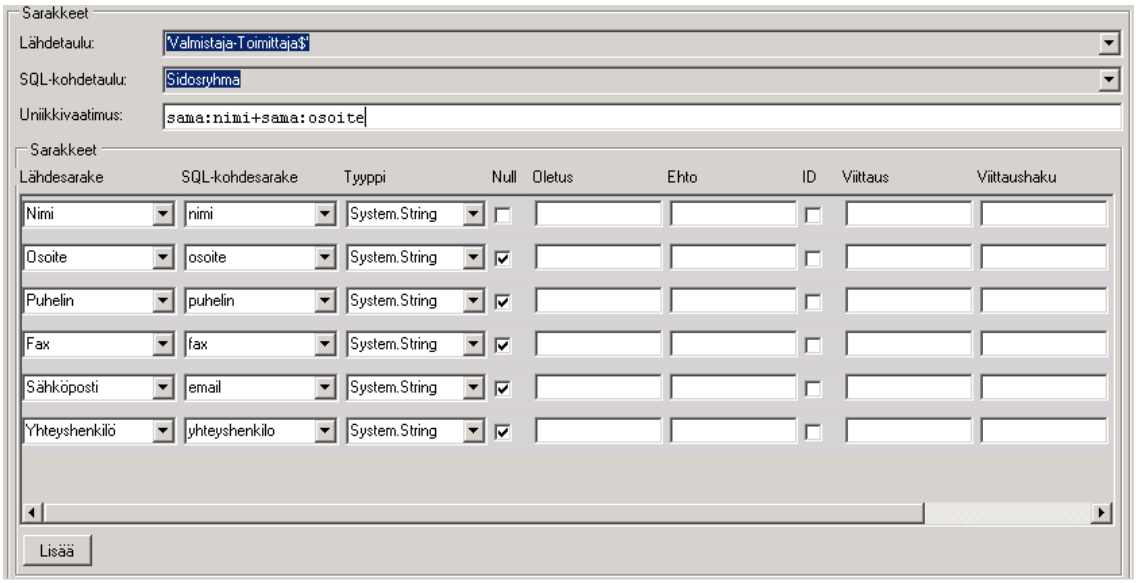
Kuva 8.5 XML-määrityksen luominen (vaihe 3)

Seuraavaksi valitaan Excel-tiedoston välilehti (**Lähdetaulu**) sekä tietokannan taulu, johon kyseinen välilehti liittyy (**SQL-kohdetaulu**). Esimerkissä valitsemme Lähdetauluksi '**Valmistaja-Toimittaja$**'-rivin, joka sisältää Excel-tiedostossa sidosryhmät. SQL-kohdetauluksi valitsemme **Sidosryhma**-rivin.



Kuva 8.6 XML-määrityksen luominen (vaihe 4)

Kun tauluyhteys on luotu, on lisättävä lähde- ja kohdetaulujen väliset sarakeyhteydet. Ne lisätään yksitellen painamalla lomakkeen alareunassa olevaa **Lisää**-painiketta. Kuvassa 8.7 näkyvät esimerkkimääritykseemme lisätyt sarakeyhteydet (jotka on lisätty käsin).



Kuva 8.7 XML-määrityksen luominen (vaihe 5)

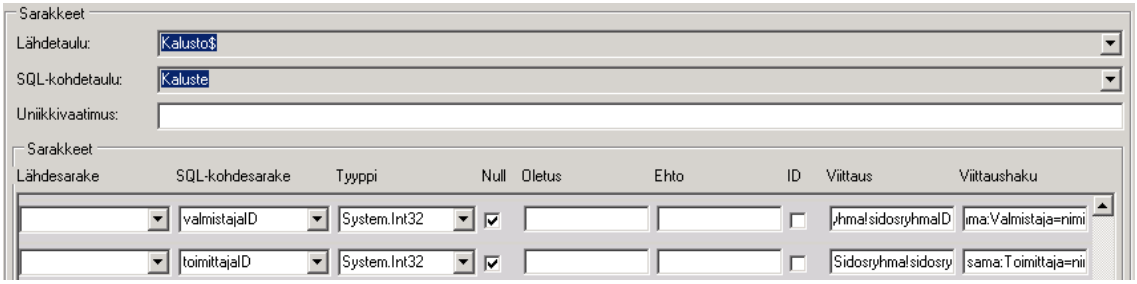
Yllä olevan kuvan mukaisesti määritys sisältää tällä hetkellä tavan, jolla oikein muodostetusta Excel-tiedostosta luetaan tiedot tietokannan **Sidosryhma**-tauluun. Määritystiedot ovat yllä olevassa tapauksessa melko yksiselitteiset. Excel-tiedoston **Nimi**-sarake on yhteydessä tietokannan **Sidosryhma**-taulun **nimi**-tietoon, **Osoite osoite**-tietoon **Puhelin puhelin**-tietoon, **Fax fax**-tietoon, **Sähköposti email**-tietoon sekä **Yhteyshenkilö yhteyshenkilo**-tietoon. Kaikki kyseiset tiedot ovat yksinkertaisessa tekstimuodossa, ja ainoastaan **nimi**-sarake on pakollinen, eli se ei siis saa olla tyhjä. Lisäksi Sidosryhma-taulun rivin

uniikkivaatimus on määritelty seuraavasti: **sama:nimi+sama:osoite**. Tämä kertoo, että kahden erillisen rivin **nimi**-tieto on oltava täsmälleen sama sekä **osoite**-tieto oltava täsmälleen sama, jotta rivit tulkittaisiin samaksi.

Esimerkki ei tietenkään sellaisenaan ole kovin hyödyllinen, koska määritys sisältää vain tietokannan yhden taulun tietojen (**Sidosryhma**) kirjoittamisen. Yhteyksiä voidaan kuitenkin lisätä tähän samaan määritystiedostoon painamalla lomakkeen vasemmassa alalaidassa olevaa **Taulut ja taulukot** -kohdan **Lisää**-painiketta ja toistamalla siitä seuraavat vaiheet.

8.2.1 Viittausyhteyden luominen

Seuraavassa kerrotaan viittausarvon luominen ja sen muotoilu. Otetaan esimerkiksi tietokannan **Kaluste**-taulu, ja sen **toimittaja**- ja **valmistaja**-tiedot. Kuvassa 8.8 on esimerkille oleelliset sarakeyhteydet Excel-tiedoston **Kalusto$**-välilehden sekä tietokannan **Kaluste**-taulun välille.



Kuva 8.8 XML-määrityksen luominen (vaihe 6)

Kuvassa ei ole lainkaan määritelty Lähdesaraketta, koska arvoa ei haeta sellaisenaan Excel-tiedoston **Kalusto$**-välilehden sarakkeesta. Arvo etsitään sen sijaan kuvan **Viittaus**-kentän mukaisen hakutoiminnon **Sidosryhma!sidosryhmaID** sekä **Viittaushaku**-kentän mukaisen hakutoiminnon **sama:Valmistaja=nimi** avulla. Hakutoiminnot kertovat yhdessä, että Excel-tiedoston **Kalusto$**-välilehden **Valmistaja**-sarakkeen arvo on oltava sama kuin **Sidosryhma**-taulun **nimi**-tieto. Lisäksi sarakkeen arvoksi asetetaan **Sidosryhma**-taulusta löydetyn rivin **sidosryhmaID**-arvo. Tämä voi vaikuttaa sekavalta, jonka takia määritystiedostojen luomiseen kannattaa käyttää aikaa eikä sitä suositella tehtäväksi ilman aiheeseen perehdytystä.

Kun määritystiedosto on valmis, se tallennetaan painamalla lomakkeen vasemman puoliskon **XML-määritystiedosto**-kentän vierellä olevaa **Tallenna**-painiketta.

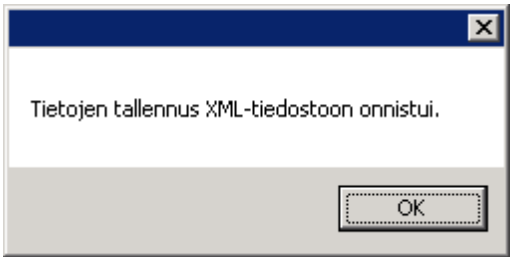
8.3 Excel-tiedoston tallentaminen XML-tiedostoon

Kun määritystiedosto on tehty, ladataan se painamalla lomakkeen vasemman puoliskon **XML-määritystiedosto**-kentän vieressä olevaa **Lataa**-painiketta.

Seuraavaksi ladataan haluttu Excel-tiedosto (jonka on oltava määritysten mukainen) painamalla lomakkeen vasemman puoliskon **Lähdetiedosto**-kentän vieressä olevaa **Lataa**-painiketta.

Lopuksi XML-tiedosto tallennetaan painamalla lomakkeen vasemman puoliskon **XML-kohdetiedosto**-kentän vierellä olevaa **Tallenna**-painiketta ja valitsemalla tallennettavan tiedoston nimi.

Mikäli tiedoston tallennus onnistui, ilmestyy ruudulle kuvan 8.9 mukainen ikkuna, jonka voi sulkea.



Kuva 8.9 XML-tiedostoon tallentaminen

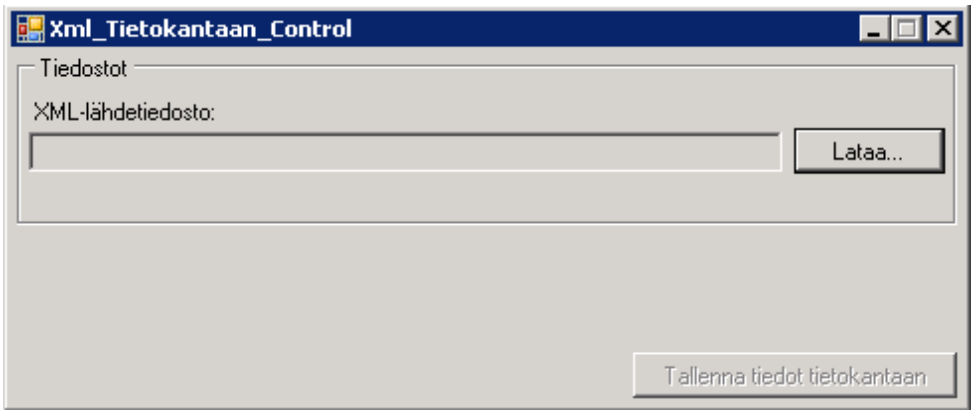
Ennen yllä olevaa ikkunaa, voi ruudulle ilmestyä erinäisiä käyttäjälle kohdistettuja kysymyksiä sen mukaan, vaatiiko jokin siirtotoimenpide käyttäjän päätöksiä. Kuvassa 8.10 on näyte sovelluksen tuottamaa XML-koodia.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Root>
  - <Taulutiedot kohdetyyppi="SQL">
    - <Taulu kohde="Sidosryhma" lahde="'Valmistaja-Toimittaja$'" uniikkivaatimus="sumea:[nimi]">
        <sarake kohde="sidosryhmaID" tyyppi="System.Int32" salliNull="False" id="True" />
        <sarake kohde="nimi" lahde="Nimi" tyyppi="System.String" salliNull="True" />
        <sarake kohde="osoite" lahde="Osoite" tyyppi="System.String" salliNull="True" />
        <sarake kohde="puhelin" lahde="Puhelin" tyyppi="System.String" salliNull="True" />
        <sarake kohde="fax" lahde="Fax" tyyppi="System.String" salliNull="True" />
        <sarake kohde="email" lahde="Sähköposti" tyyppi="System.String" salliNull="True" />
        <sarake kohde="valmistaja" tyyppi="System.Boolean" salliNull="False" oletus="True" />
        <sarake kohde="toimittaja" tyyppi="System.Boolean" salliNull="False" oletus="True" />
        <sarake kohde="huoltoyritys" tyyppi="System.Boolean" salliNull="False" oletus="True" />
        <sarake kohde="vakuutusyhtio" tyyppi="System.Boolean" salliNull="False" oletus="False" />
        <sarake kohde="yhteyshenkilo" tyyppi="System.String" salliNull="True" />
      </Taulu>
    - <Taulu kohde="Kaluste" lahde="Kalusto$">
        <sarake kohde="kalusteID" lahde="ID" tyyppi="System.Int32" salliNull="False" id="True" />
        <sarake kohde="kohdeID" tyyppi="System.Int32" salliNull="False" oletus="16" />
        <sarake kohde="tonttiID" tyyppi="System.Int32" salliNull="True" oletus="" />
        <sarake kohde="rakennusID" tyyppi="System.Int32" salliNull="False" oletus="490" />
        <sarake kohde="tilaID" tyyppi="System.Int32" salliNull="True" oletus="" />
        <sarake kohde="huoneistoID" tyyppi="System.Int32" salliNull="True" />
        <sarake kohde="huoneID" tyyppi="System.Int32" salliNull="True" />
        <sarake kohde="kalustetyyppiID" lahde="Kalusto-tyyppinumero" tyyppi="System.Int32" salliNull="True" />
        <sarake kohde="valmistajaID" tyyppi="System.Int32" salliNull="True" viittaus="Sidosryhma!sidosryhmaID"
          viittaushaku="sama:Valmistaja=nimi" />
        <sarake kohde="toimittajaID" tyyppi="System.Int32" salliNull="True" viittaus="Sidosryhma!sidosryhmaID"
          viittaushaku="sama:Toimittaja=nimi" />
        <sarake kohde="huoltoyritysID" tyyppi="System.Int32" salliNull="True" viittaus="Sidosryhma!sidosryhmaID"
          viittaushaku="sama:Huoltoyritys=nimi" />
        <sarake kohde="sijainti" lahde="Sijainti" tyyppi="System.String" salliNull="True" />
        <sarake kohde="valmistusvuosi" lahde="Valmistus-vuosi" tyyppi="System.Int32" salliNull="True" />
        <sarake kohde="kayttoonotto" lahde="Käyttöön-otto päivä" tyyppi="System.DateTime" salliNull="True" />
```

Kuva 8.10 Tuotettua XML-koodia

8.4 XML-tiedoston tallentaminen tietokantaan

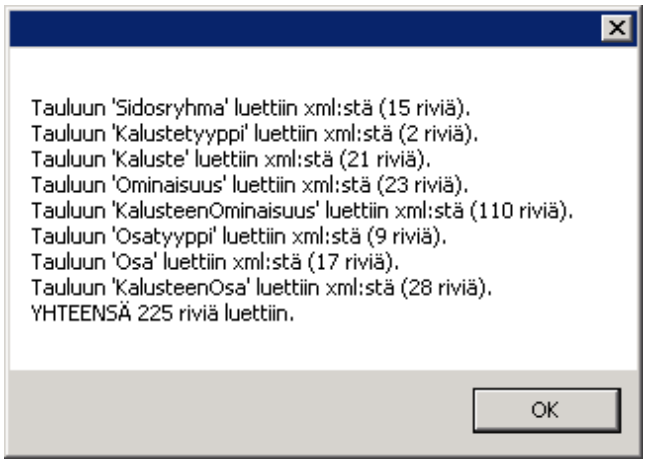
Tuloksena saatu XML-tiedosto voidaan kirjoittaa tietokantaan. Tietokantaan kirjoitus tehdään vaiheittain. Ensin painetaan sovelluksen aloitusvalikon **XML:stä tietokantaan** -painiketta, jolloin kuvan 8.11 mukainen lomake avautuu ruudulle.



Kuva 8.11 XML:stä tietokantaan tallentaminen (vaihe 1)

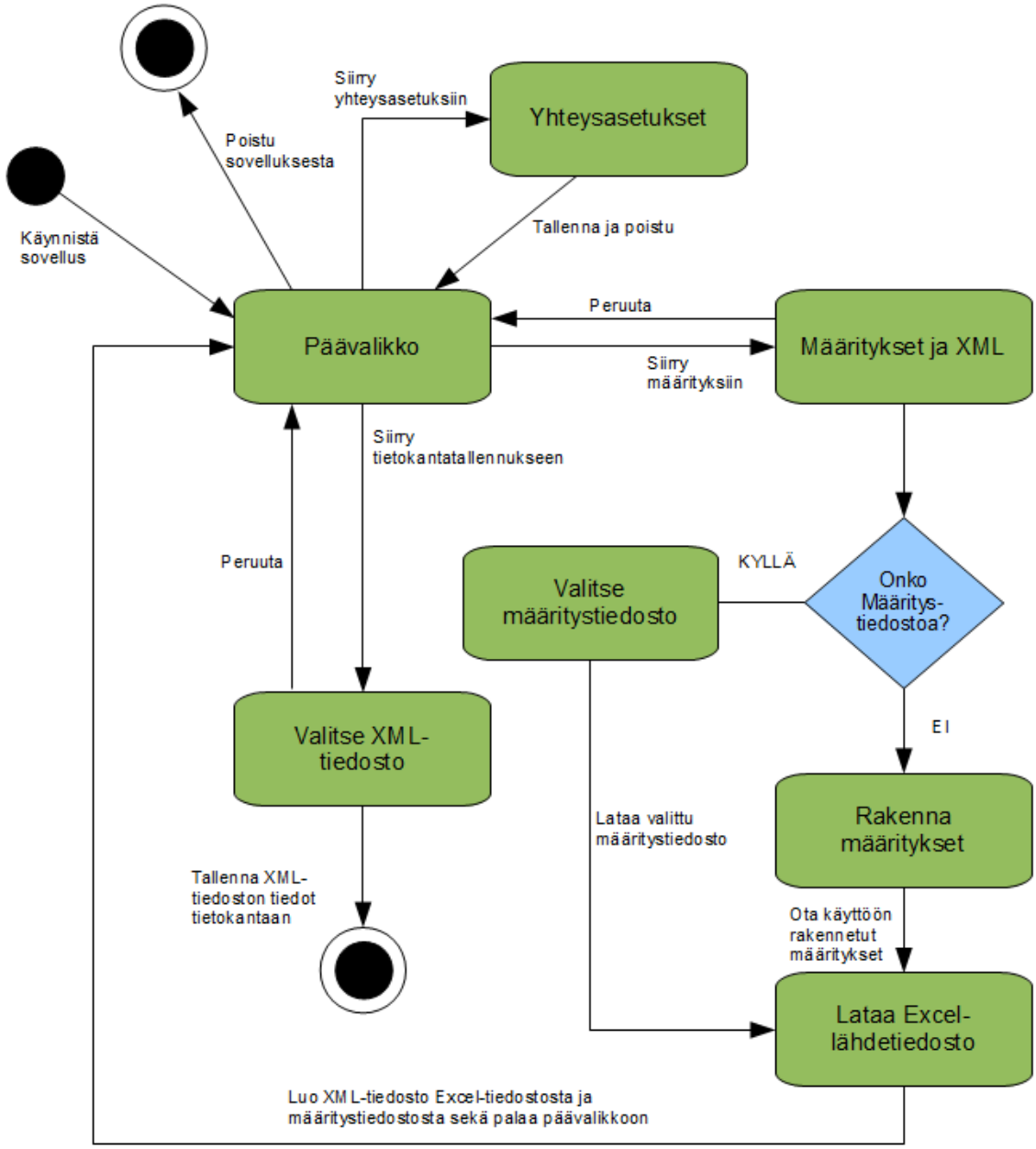
Tallennettava XML-tiedosto ladataan ensin sovelluksen muistiin painamalla **XML-lähdetiedosto**-kentän vieressä olevaa **Lataa**-painiketta ja valitsemalla

avautuvasta tiedoston lataus -ikkunasta haluttu XML-tiedosto. Kuvan 8.12 mukainen ikkuna näytetään ruudulla.



Kuva 8.12 XML:stä tietokantaan tallentaminen (vaihe 2)

Ikkuna kertoo, kuinka monta riviä tietoja luettiin ja mihin tietokannan tauluihin tiedot on tarkoitus kirjoittaa. Lopuksi tiedot tallennetaan tietokantaan painamalla lomakkeen **Tallenna tiedot tietokantaan** -painiketta. Kuvassa 8.13 on kuvattu sovelluksen toiminnan rakenne kaaviona.



Kuva 8.13 Sovelluksen toimintokaavio

9 YHTEENVETO

Seuraavassa on yhteenveto projektin kulusta, kohdatuista ongelmista, onnistumisista ja epäonnistumisista, hylätyistä ratkaisuvaihtoehdoista, jatkokehitysideoista sekä opituista asioista.

Projektin kulku sujui mielestäni onnistuneesti, vaikka oletin aluksi aikataulun venyvän. Projekti pysyi kuitenkin aikataulussaan, eikä toteutusongelmia ollut ennakoitua enempää. Työtahti oli myös melko hyvä, vaikka aina itse ohjelmointityöhön ei ehtinyt paneutumaan. Muut työtehtävät veivät toisinaan melko paljon aikaa, mutta työt oli priorisoitava kulloinkin vallitsevan tilanteen mukaan. Mielestäni sain rajattua projektin jo alkuvaiheessa sopivan kokoiseksi. Olen huomannut aiemmissa projekteissani projektien paisumisen liian suureksi sitä mukaa, kun sovellukseen keksitään uusia ominaisuuksia. Tällä kertaa sain kuitenkin hillittyä ylimääräisten ominaisuuksien toteutusta ja tyydyin kirjaamaan kyseiset toisarvoiset ominaisuudet jatkokehitysideoiksi. Työpäiväkirjan säännöllinen päivittäminen projektin edetessä auttoi hahmottamaan aina hyvin senhetkisen tilanteen. Koska kirjasin työpäiväkirjaan eteen tulleet ongelmat ja tekemäni korjaukset, omaan muistiin ei tarvinnut luottaa niin paljoa. Ongelmien ja ideoiden kirjaaminen ylös sitä mukaa kuin ne ilmaantuivat auttoi myös tämän raportin kirjoittamisessa.

Ongelmia projektissa kohtasin muutaman kerran enimmäkseen käyttöliittymätoteutuksen osalta. Koska käyttöliittymää ei oltu määritelty etukäteen lainkaan, oli sen toteuttaminen paikoitellen haasteellista. Käyttöliittymä koki monta eri kehitysvaihetta ja versiota ennen viimeisintä versiota. Käyttöliittymä saattoi silti jäädä lopulta hieman haasteelliseksi tietotekniikkaan perehtymättömille. Toisaalta sovelluksen peruskäyttöön on mahdollista perehdyttää henkilö, jolla on peruskansalaisen tietotekniset taidot.

Parhaiten onnistuin mielestäni itse tietojen käsittelyyn tarvittavien ominaisuuksien määrittelyssä. Koska olin panostanut riittävästi aikaa määrittelyvaiheessa tietojen yhdistelyn eri ongelmien ratkaisemiseen, itse toteutusvaihe eteni tasaisen varmasti. Jälkikäteen ajateltuna määrittelyvaiheessa olisi pitänyt miettiä myös

käyttöliittymän suunnittelemista niin pitkälle kuin mahdollista. Tämä olisi helpottanut toteutuksen loppuvaihetta ja tehnyt mahdollisesti ohjelmasta helpommin käytettävän.

Hylättyjä ratkaisuvaihtoehtoja sovelluksen edetessä oli standardoitujen XML-skeemojen käyttö määriteltäessä tietojen rakennetta. Päädyin käyttämään omia tietojen rakenteen määritelmiäni ajanpuutteen takia. Standardien XML-skeemojen opiskelu ja käyttöönotto olisi vienyt liiaksi aikaa niistä saataviin hyötyihin nähden.

Jatkokehitysideana on lisätä eri tietomuotoja lähde- ja kohdetiedoiksi. Alkuperäisten vaatimusten mukaan lähdetiedon tulisi olla Excel-tiedostossa ja kohdetiedon Microsoft SQL -tietokannassa. Sain kuitenkin jo toteutusvaiheen loppupuolella lisättyä lähdemuotojen valikoimaan Access-tietokantojen tuen. Access-tuen lisääminen osoittautui yllättävän yksinkertaiseksi OLE DB -tekniikan ansiosta, eikä lisäystarve vaatinut käytännössä kuin muutaman koodirivin lisäämisen sovellukseen.

Lähdetiedostot ovat tällä hetkellä Excel-tiedostoissa, eikä Excel sisällä kovinkaan paljon mekanismeja syötteiden oikeellisuuden tarkistamiseen. Toinen jatkokehitysidea onkin tehdä yksinkertainen tietojensyöttösovellus olemassa olevien määritystiedostojen pohjalta. Näin voitaisiin varmistaa, että tietojen syöttäjät eivät voisi vahingossakaan syöttää epäkelpoa tietoa järjestelmään. Tietoa ei siis enää syötettäisikään Excel-tiedostoihin, vaan tietojen syöttöön varta vasten kehitetyn sovelluksen avulla. Tämä kehitysidea tuli mieleeni jo toteutusvaiheessa. Käyttöönottovaiheessa asiakas ehdotti täsmälleen samaa ideaa. Kehitysidea on hyvin varteenotettava ja se tullaan luultavasti toteuttamaan.

Kaiken kaikkiaan opin entistä enemmän panostamaan sovelluskehityksen alkuvaiheisiin, eli esitutkimukseen, määrittelyyn ja suunnitteluun. Myös oman työskentelyrytmin opettelu ja ylläpitäminen auttoi projektin toteuttamisessa ja aion jatkossakin täyttää työpäiväkirjaa ja kirjata eteen tulleita ongelmia ja ideoita talteen.

KUVAT

LÄHTEET

CodeGuru 2010. The .NET Architecture. http://www.codeguru.com/csharp/sample_chapter/article.php/c8245 (Luettu 15.11.2010)

Global Classroom Training 2009. The SQL Server for Database Administrators Video Training DVD. http://globalclassroomtraining.com/site_listings/The%20SQL%20Server%202005%20For%20Database%20Administrators%20Video%20Training%20DVD.html (Luettu 15.11.2010)

Haikala, I. 2000. Ohjelmistotuotanto. Pieksämäki: Rt-Print oy.

Kettinen, J. 2008. Huoltokirja – Kiinteistöhuollon toiminnanohjausjärjestelmä. Opinnäytetyö. Etelä-Karjalan ammattikorkeakaoulu.

Kosonen, S. 2005. Ohjelmoinnin opetus Extreme Programming -hengessä. Jyväskylän yliopisto.

Lappeenranta. Tilakeskus. http://www.lappeenranta.fi/?deptid=11919 (Luettu 6.11.2010)

Microsoft 2010. Microsoft OLE DB. http://msdn.microsoft.com/en-us/library/ms722784%28VS.85%29.aspx (Luettu 6.11.2010)

McConnell, S. 2002. Ohjelmistotuotannon hallinta. Helsinki: Edita Prima Oy.

Reinikka, A. 2007. Tietojärjestelmän kehittäminen markkinaehtoisten sähkösopimusten hallintaan. Diplomityö. Lappeenrannan tekninen yliopisto. https://oa.doria.fi/bitstream/handle/10024/38317/nbnfi-fe200805221430.pdf?sequence=3 (Luettu 6.11.2010)

Tähtinen, S. 2005. Järjestelmäintegraatio. Helsinki: Talentum Media Oy.

Väestörekisterikeskus 2010. Kuntien asukasluvut aakkosjärjestyksessä. http://vrk.fi/vrk/files.nsf/files/CA29780B1934541BC22577B90044D341/$file/20100930.htm (Luettu 6.11.2010)

Wikipedia: .NET Framework 2010. http://en.wikipedia.org/wiki/.NET_Framework (Luettu 9.11.2010)

Wikipedia: Scrum process 2010. http://upload.wikimedia.org/wikipedia/commons/5/58/Scrum_process.svg (Luettu 9.11.2010)

Wikipedia: Scrum 2010. http://fi.wikipedia.org/wiki/Scrum (Luettu 9.11.2010)

Wille, C. 2001. C#. Jyväskylä: Docendo Finland Oy.