Urpo Kähkönen

HOTI MANAGEMENT TOOL

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Luonnontieteiden ala
Tietojenkäsittelyn koulutusohjelma
Syksy 2009

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Luonnontieteiden ala	Koulutusohjelma Tietojenkäsittely
Tekijä(t) Urpo Kähkönen	
Työn nimi HOTI Management Tool	
Vaihtoehtoiset ammattiopinnot Ohjelmistosuunnittelu	Ohjaaja(t) Matti Härkönen
	Toimeksiantaja Luovaliike Oy
Aika Syksy 2009	Sivumäärä ja liitteet 28+0

Tämän opinnäytetyön aiheena oli toteuttaa Luovaliike Oy:lle ylläpitosovellus HOTI kotihoidon tietojärjestelmään. Sovelluksen avulla ylläpitohenkilöstö voi helposti helposti hyödyntää asiakastietoa, jota tarvitaan järjestelmän ylläpidossa.

Opinnäytetyön teoriaosuudessa käsitellään tietoturvametelmiä, käyttöliittymäsuunnittelua ja LINQ-tekniikoita.

Sovelluksen toteuttamiseen käytettiin Visual Studio 2008 ja SQL Server 2005 ohjelmistoja. Ohjelmointiin käytettiin C#-ohjelmointikieltä. Kehitysprosessi aloitettiin luomalla vaatimusmäärittelyn pohjalta tietokantakuvaus. Kun tietokanta oli saatu valmiiksi, voitiin aloittaa ohjelman käyttöliittymän toteutus.

Opinnäytetyön käytännön osuudessa käsitellään ohjelman toteutusvaiheita ja esitellään käyttöliittymä yleisellä tasolla.

Opinnäytetyön tuloksena saatiin kehitettyä toimiva ohjelma, vaikka kaikkia haluttuja ominaisuuksia ei saatu toteutettua. Ohjelman kehitystä jatketaan opinnäytetyön valmistuttua.

Kieli	Suomi
Asiasanat	Tietoturva, Käyttöliittymäsuunnittelu, LINQ
Säilytyspaikka	☒ Kajaanin ammattikorkeakoulun Kaktus-tietokanta ☒ Kajaanin ammattikorkeakoulun kirjasto

Kajaanin ammattikorkeakoulu
University of Applied Sciences

THESIS
ABSTRACT

School Business	Degree Programme Business Information Technology
Author(s) Urpo Kähkönen	
Title HOTI Management Tool	
Optional Professional Studies Programming	Instructor(s) Matti Härkönen
	Commissioned by Luovaliike Oy
Date Fall 2009	Total Number of Pages and Appendices 28+0

The thesis was commissioned by Luovaliike Oy. The purpose of the thesis was to develop software to help the maintenance of the HOTI system. With this maintenance software administrators can easily utilize the customer information that is needed in maintenance tasks.

The theoretical part of the thesis concentrates on different information security methods, graphical user interface design and LINQ technologies.

Visual Studio 2008 and SQL Server 2005 applications and C# programming language were used in the development process. The first step in the development process was to design a database based on the requirements analysis. The programming of the graphical user interface was started after the database was complete.

The empirical part of the thesis presents the different stages of software development and graphical user interface at a general level.

The result of this thesis was operational software, even though some of the desired features could not be accomplished. The development of the software will continue after the thesis.

Language of Thesis	Finnish
Keywords	Information security, graphical user interface design, LINQ
Deposited at	☒ Kaktus Database at Kajaani University of Applied Sciences ☒ Library of Kajaani University of Applied Sciences

SYMBOLILUETTELO

.NET FRAMEWORK	Microsoftin ohjelmistokomponenttikirjasto.
HOTI	Luovaliike Oy:n kehittämä kotihoidon tietojärjestelmä.
LINQ	Language Integrated Query.
RDBMS	Relational Database Management System. Relaatiotietokanta.
SQL	Structured Query Language. Relaatiotietokantojen yhteydessä käytetty kyselykieli.
XML	Extensible Markup Language. Rakenteellinen kuvauskieli

SISÄLLYS

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimi Luovaliike Oy. Luovaliike Oy on kehittänyt kotihoidon tietojärjestelmän (HOTI), jonka ylläpitoa avustamaan tarvittiin erillinen ylläpito-ohjelma (HOTI Management Tool). Aiheen valintaan vaikutti merkittävästi se, että tämän työn tekijä on ollut mukana HOTI:n kehityksessä. Aihe oli haastava, mutta silti laajuudeltaan sopiva yhden henkilön toteutettavaksi.

Opinnäytetyön tavoitteena oli tehdä HOTI:n rinnalle erillinen ohjelma, johon voidaan tallentaa ylläpidossa tarvittavaa tietoa. Työn toteutuksessa käytettiin Visual Studio 2008 ja SQL Server 2005 ohjelmistoja. Ohjelmoinnissa käytettiin C#-ohjelmointikieltä.

Teoriaosassa on käsitelty erilaisia tietoturvamenetelmiä, LINQ-tekniikoita ja käyttöliittymäsuunnitteluohjeita. Nämä aiheet on valittu tukemaan käytännön osuuden toteuttamisprosessia ja kehittämään tämän työn tekijän tietämystä aihealueista.

Käytännön osuudessa käsitellään ohjelman toteutusvaiheita ja käydään läpi käyttöliittymän toiminnallisuus yleisellä tasolla.

2 TIETOTURVAMENETELMÄT

Luvussa käsitellään menetelmiä, joita voidaan käyttää tietoturvan parantamiseen. Usein ohjelmilla käsitellään sellaista tietoa, jonka ei haluta joutuvan vääriin käsiin. Tästä syystä tietoturva-asioiden huomioon ottaminen ohjelmiston kehitysvaiheessa on tärkeää.

2.1 Tiivisteet

Tiiviste on salasanasta tai jostakin muusta tiedosta luotu määrämittainen tarkistussumma, joka luodaan tiivistealgoritmia käyttämällä. Tiivisteen luonti on aina yksisuuntainen operaatio. Olemassa olevaa tiivistettä ei voida enää muuttaa takaisin sitä vastaavaksi merkkijonoksi. Tiivisteen käyttö perustuu siihen, että samasta merkkijonosta samalla algoritmilla luotu tiiviste on aina samanlainen. Tätä tietoa käytetään hyödyksi erilaisissa ohjelmissa käyttäjän kirjautumistietojen tarkastuksessa. Käyttäjän antamasta salasanasta muodostetaan tiiviste ja sitä verrataan vastaavaan tallennettuun arvoon. Mikäli arvot vastaavat toisiaan, on annettu salasana oikea. Kuviossa 1. on listattu yleisesti käytettyjä tiivistealgoritmeja. (Northrup 2009, 581-583.)

Table 12-4 Nonkeyed Hashing Algorithms

Abstract Class	Implementation Class	Description
MD5	*MD5CryptoServiceProvider*	The Message Digest 5 (MD5) algorithm. The hash size is 128 bits.
RIPEMD160	*RIPEMD160Managed*	The Message Digest 160 (MD160) hash algorithm. The hash size is 160 bits.
SHA1	*SHA1CryptoServiceProvider*	The Secure Hash Algorithm 1 (SHA1). The hash size is 160 bits.
SHA256	*SHA256Managed*	The Secure Hash Algorithm 256 (SHA256). The hash size is 256 bits.
SHA384	*SHA384Managed*	The Secure Hash Algorithm 384 (SHA384). The hash size is 384 bits.
SHA512	*SHA512Managed*	The Secure Hash Algorithm 512 (SHA512). The hash size is 512 bits.

Kuvio 1. Tiivistealgoritmit (Northrup 2009, 581-582.)

Pelkkä tiivisteen käyttö salasanojen tallennuksessa ei kuitenkaan ole riittävä turva salasanojen suojaamiseksi. Salasanasta luotu tiiviste on mahdollista murtaa sanakirjahyökkäyksen avulla. Sanakirjahyökkäyksessä käytetään erilaisista sanoista luotua tiivistelistaa. Listalla olevia tiivisteitä verrataan murrettavaan tiivisteeseen. Mikäli murrettavaa tiivistettä vastaava tiiviste löytyy listalta, tiedetään myös sana josta tiiviste on luotu. Mitä yksinkertaisempi alkuperäinen salasana on, sitä helpompaa on löytää vastaava tiiviste. Kyseisen hyökkäyksen vaikeuttamiseksi tiivisteen luonnin yhteydessä salasanaan voidaan lisätä satunnainen merkkijono. (Hamilton 2008, 415-418.)

2.2 Suolaus

Ennen tiivisteen luontia salasanaan liitetään ”suola”, eli satunnainen merkkijono. Tällä menetelmällä vaikeutetaan tiivisteen murtamista merkittävästi. Tälläisen tiivisteen murtaminen ei onnistu suoraan sanakirjahyökkäyksen tiivistelistan avulla, koska jokainen listan tiiviste täytyy luoda uudestaan suolan avulla. Vasta tämän jälkeen voidaan alkaa etsiä vastinetta murettavalle tiivisteelle. Suolauksessa käytetyn merkkijonon tulee olla yksilöllinen jokaista käyttäjätunnusta kohden. Kuviossa 2. on esitetty tiivisteen laskeminen ja suolaus C#-ohjelmointikieltä käyttäen. (Hamilton. 2008, 415-418; Burnett & Foster. 2004, Chapter 4.)

```
private void hashExample()
{
  string plainText = "This is a secret for hashExample";
  Response.Write("plainText: " + plainText + "<br>");
  // Create strong random byte values
  byte[] saltBytes = new byte[8];
  RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
  rng.GetNonZeroBytes(saltBytes);
  // Create an array to hold plain text and salt
  byte[] plainTextBytes = ASCIIEncoding.ASCII.GetBytes(plainText);
  byte[] plainTextAndSaltBytes =
    new byte[plainTextBytes.Length + saltBytes.Length];

  // Append salt value and create byte array for hash
  for (int x=0; x < saltBytes.Length; x++)
    plainTextAndSaltBytes[x] = saltBytes[x];
  for (int x=0; x < plainTextBytes.Length; x++)
    plainTextAndSaltBytes[saltBytes.Length + x] =
     plainTextBytes[x];
  // Perform hash using SHA1
  HashAlgorithm hash = new SHA1Managed();
  byte[] hashBytes = hash.ComputeHash(plainTextAndSaltBytes);
  Response.Write("Hashed string with salt: " +
    Convert.ToBase64String(hashBytes) + "<br>");
}
```

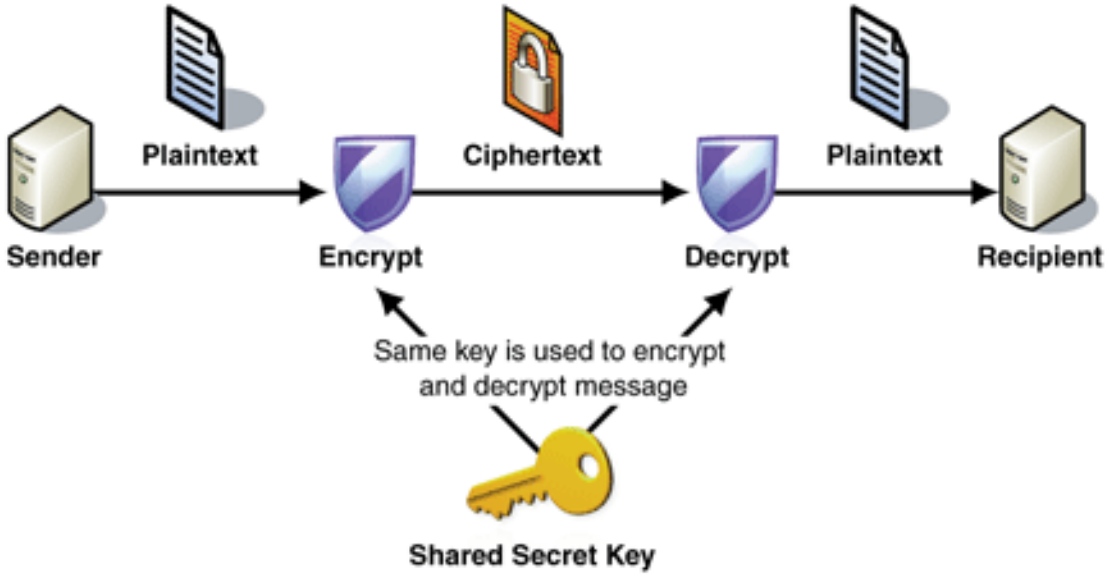
Kuvio 2. Tiivisteen luonti ja suolaus (Burnett & Foster. 2004, Chapter 4.)

2.3 Salausmenetelmät

Erilaiset salausmenetelmät ovat yleensä välttämättömiä, jotta tietoa voidaan käsitellä ja säilyttää turvallisesti. Tässä luvussa esitellään yleisesti käytetyt salausmenetelmät.

a) Symmetrinen salaus

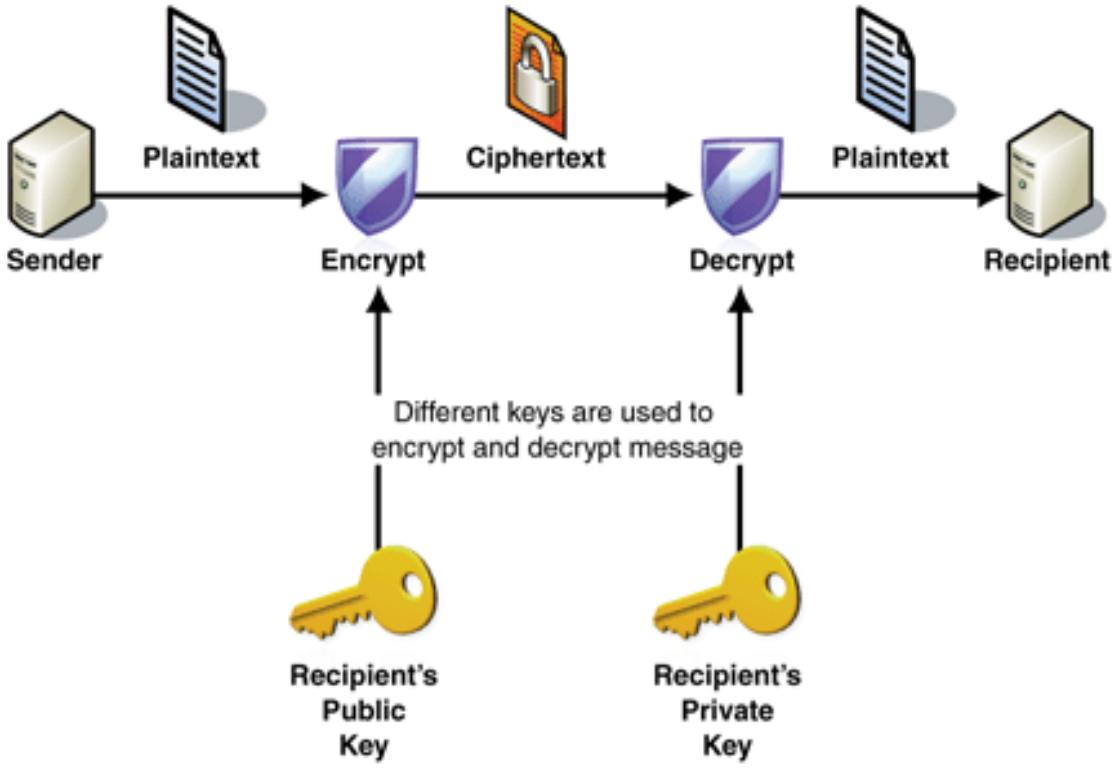
Salaisen avaimen salausta kutsutaan symmetriseksi salaukseksi. Tiedon salaamiseen ja purkamiseen käytetään samaa avainta. Symmetrisen salauksen toiminta on esitetty kuviossa 3. Symmetrinen salaus on nopeaa ja sen murtaminen on vaikeaa. Salausavaimen säilyttäminen on merkittävin ongelmakohta tässä salausmuodossa. Symmetrisen salauksen menetelmiä ovat esimerkiksi salausalgoritmit DES, RC2 ja RC4. (Karttuenn 1999.)



Kuvio 3. Symmetrinen salaus (Microsoft 2005.)

b) Asymmetrinen salaus

Asymmetrisesta salauksesta käytetään nimitystä julkisen avaimen salaus. Tiedon salaamiseen ja purkamiseen käytetään avainparia, julkista ja salaista avainta. Tieto salataan julkisella avaimella ja puretaan salaisella avaimella. Julkisella avaimella salatun tiedon voi purkaa vain avainparin salaisella avaimella. Huono puoli tässä menetelmässä on salauksen hitaus. Asymmetrisen salauksen toiminta on esitetty kuviossa 4. Asymmetristä salausta käytetään pääasiassa käyttöoikeuksien todentamisessa ja sähköisessä allekirjoituksessa. RSA on yksi laajimmalle levinneistä julkisen avaimen salausmentelmistä. (Karttuenn 1999.)



Kuvio 4. Asymmetrinen salaus (Microsoft 2005.)

2.4 Obfuskointi

Obfuskoinnilla tarkoitetaan ohjelman lähdekoodin muuttamista sellaiseen muotoon, että ihmisen on mahdotonta ymmärtää ohjelman sisäistä toimintaa pelkästään koodia katsomalla. Tällä menetelmällä pyritään siis lisäämään tietoturvaa ja suojelemaan tekijänoikeuksia. Obfuskointi tehdään yleensä siihen tarkoitetulla erillisellä ohjelmalla. Obfuskointi ei saa muuttaa suojattavan ohjelman toiminnallisuutta, eikä hidastaa merkittävästi sen toimintaa.

Vaihtoehtona obfuskoinnille on ohjelman salaaminen ja salauksen purkaminen ennen ohjelman ajamista. Salauksen purkaminen kuitenkin hidastaa ohjelman toimintaa, joten tämä ei ole yleisesti käytössä oleva menetelmä. (Tyma 2003; Barak 2001.)

2.5 Vahva salasana

Tärkeä osa monissa ohjelmistoissa on salasanaturvallisuus. Tällä termillä voidaan tarkoittaa monia asioita. Osa salasanaturvallisuutta on se, kuinka salasanat tallennetaan ja miten niitä käsitellään ohjelman sisällä. Salasanojen oikea käsittely ei kuitenkaan riitä mikäli käyttäjien salasanat ovat helposti arvattavissa tai murrettavissa siihen tarkoitetuilla ohjelmilla. Vahvat salasanat ovat käytännössä merkkijonoja, jotka koostuvat erityylisistä merkeistä, kuten numeroista, kirjaimista ja erikoismerkeistä. Salasanan pituus on myös tärkeä tekijä sen vahvuuden määrittämisessä. (Oh 2008.)

Vahvan salasanan luontiin löytyy monia ohjeita. Seuraavaan listaan on koottu näistä yleisimpiä (Oh 2008.):

- Salasanan tulee olla vähintään 8 merkkiä pitkä.
- Käytä erikoismerkkejä, kuten: @#$%^&.
- Vältä yleisiä sanoja ja merkkijonoja.
- Käytä isoja ja pieniä kirjaimia.
- Käytä numeroita.

Vahvojen salasanojen käyttö on hyvä asia tietoturvallisuuden kannalta, mutta niillä on myös huonot puolensa. Käyttäjillä on usein vaikeuksia muistaa ulkoa monimutkaisia salasanoja. Ongelma korostuu silloin kun salasanoja vaihdetaan usein. Ongelman ratkaisemiksi voidaan käyttää menetelmää, jossa salasana luodaan jostakin helposti muistettavasta merkkijonosta. Tässä menetelmässä salasanan luonti aloitetaan valitsemalla joku henkilökohtainen merkkijono, esimerkiksi tuttu lause tai oma nimi. Seuraavaksi merkkijonon kirjaimia tai muita merkkejä korvataan numeroilla tai erikoismerkeillä. Tässä vaiheessaa on tärkeää opetella muistamaan millä erikoismerkillä kukin alkuperäisen merkkijonon merkeistä on korvattu. Lopputuloksena saatu merkkijono on vahva salasana, jos myös muita salasanan luontiohjeita on noudatettu. Tällä tavalla voi luoda ja muistaa helposti useita erilaisia salasanoja. Tärkeintä on muistaa säännöt joilla salasanat on luotu. (Oh 2008.)

3 LINQ

LINQ eli Language Integrated Query on osa Microsoftin .NET Framework 3.5 ohjelmistokomponenttikirjastoa. Perinteisesti tiedon käsittely eri tietolähteistä vaatii erityisen ohjelmointityökalun tai ohjelmakirjaston käyttöä riippuen käytetystä tietolähteestä. LINQ:n tehtävä on helpottaa tiedon käsittelyä ohjelmoinnissa, sekä luoda yhtenäinen ja selkeä tapa tiedon siirtämiseen eri tietolähteistä. (Eichert 2008, 4-5.)

LINQ:n suunnittelussa on ollut useita tavoitteita. Seuraavassa on listattu keskeisimpiä kehitystavoitteita (Mueller 2008, 14-15.):

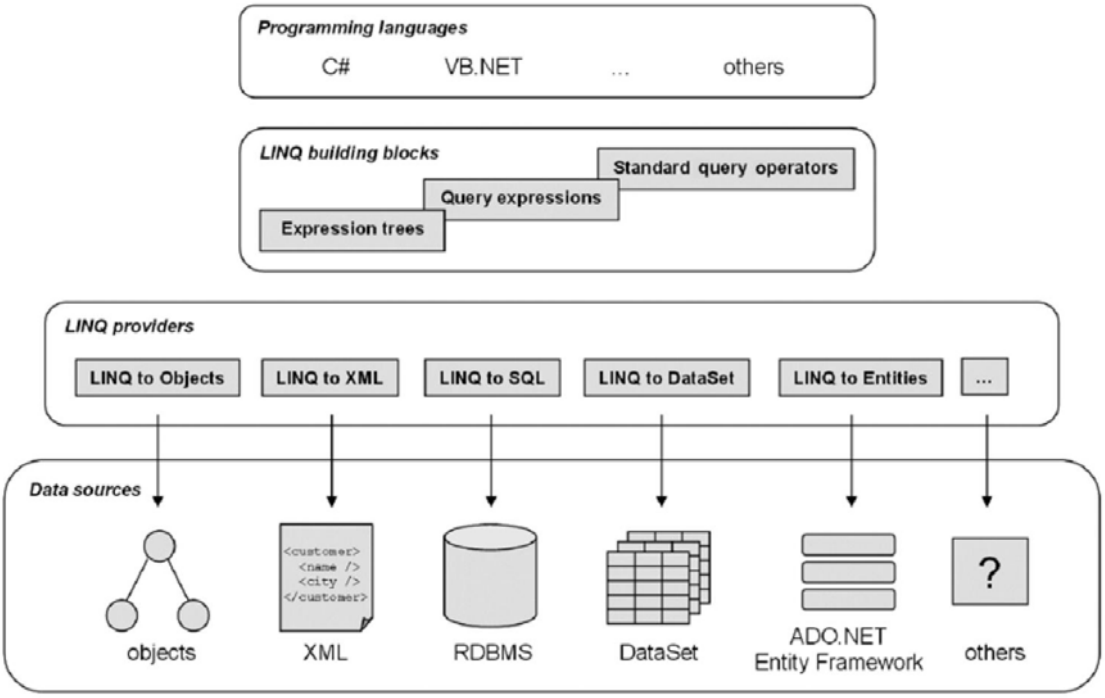
- Tietolähteeseen muodostetun yhteyden yksinkertaistaminen. Yksi suurimmista ongelmista ohjelmistokehityksessä on se, että ohjelmistokehittäjän täytyy käyttää tietolähteestä riippuen useita erilaisia tiedonhakumenetelmiä. Esimerkiksi tiedon hakeminen tietokannasta on merkittävästi erilaista XML-dokumenttien käsittelymenetelmiin verrattuna.

- Tiedon käsittelyn yksinkertaistaminen. Kun tietoa haetaan esimerkiksi tietokannasta, täytyy ottaa huomioon tietolähteen rakenne. Erilaiset taulurakenteet, indeksoinnit ja muut tietokantaelementit johtavat helposti monimutkaisiin tiedonkäsittelyrakenteisiin perinteisiä menetelmiä käytettäessä.

- Tiedon muuntaminen muodosta toiseen. Tiedon siirto tietolähteestä toiseen on yleensä ongelmallista. Esimerkiksi XML-dokumentin sisältämän tiedon siirtäminen tietokantaan vaatii monia toimenpiteitä. Lisäksi tiedon muuntaminen muodosta toiseen tavallisilla mentelmillä ei aina tuota toivottuja tuloksia. LINQ yksinkertaistaa muunnosta merkittävästi tekemällä itse varsinaisen muutostyön.

- Kielen laajennettavuus. LINQ tarjoaa valmiiksi mahdollisuuden hyödyntää useita erilaisia tietolähteitä. Kuitenkin on olemassa tilanteita, jolloin valmiit ratkaisut eivät riitä. Tämän vuoksi LINQ on haluttu kehittää mahdollisimman muokattavaksi.

LINQ voidaan jakaa osiin, jolloin sen rakenne on helpommin ymmärrettävissä. LINQ:n perusta koostuu kolmesta osasta: hakuoperaattorit (query operators), hakulausekkeet (query expressions) ja lausekepuut (expression trees). Näiden kolmen peruskomponentin avulla on luotu laaja valikoima ohjelmointityökaluja, joita voidaan hyödyntää erilaisia tietolähteitä käsiteltäessä. Kirjallisuudessa näistä ohjelmointityökaluista käytetään englanninkielistä nimitystä LINQ providers. Kuviossa 5 on esitetty LINQ:n perusosat ja eri tietolähteisiin käytetyt ohjelmointityökalut. (Eichert 2008, 5-7.)



Kuvio 5. LINQ:n rakenne (Eichert 2008, 7)

3.1 LINQ-kyselyt

Ulkoasultaan LINQ-kyselyt muistuttavat paljon SQL-kyselyitä. Avainsanat from, where, orderby ja select ovat tuttuja tietokantojen parissa työskenteleville.

Yksi LINQ:n uusista ominaisuuksita on var-tietotyyppi. Var tyyppisen muuttujan varsinainen tietotyyppi määritetään ohjelman ajon aikana. Kuviossa 6 on esitetty kuinka vähän ohjelmakoodia tarvitaan, kun LINQ-kyselyn avulla haetaan tietoa tietokannasta ja luodaan haetusta tiedosta XML-dokumentti. (Ferracchiati 2008, 9-10; Eichert 2000, 8)

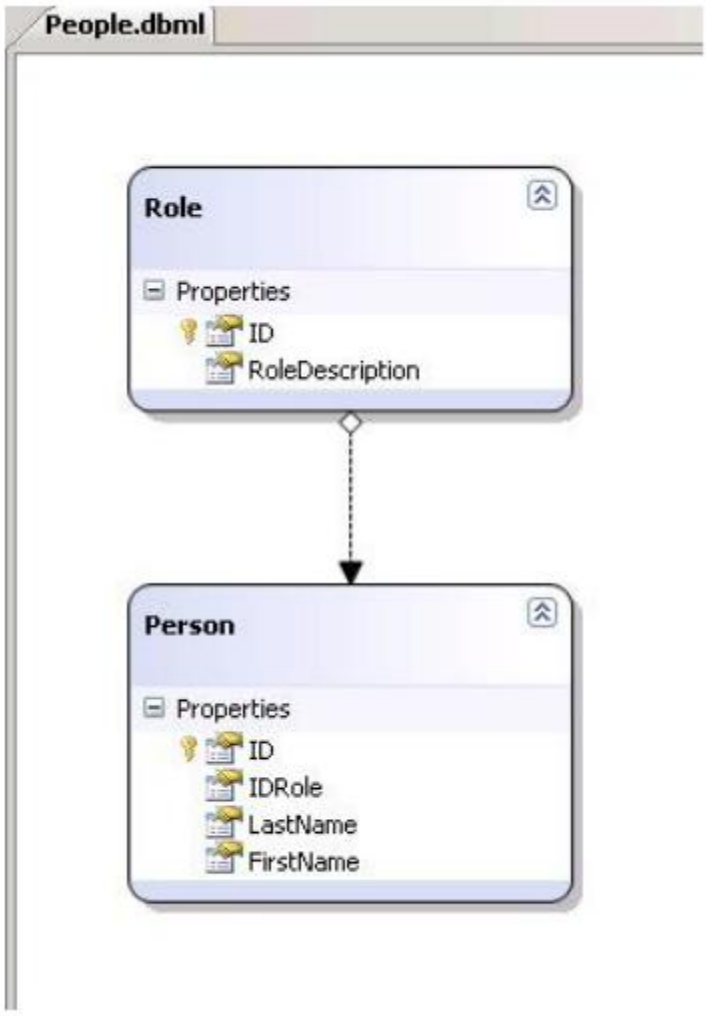
```
var contacts =                          <-- Retrieve customers from database
  from customer in db.Customers
  where customer.Name.StartsWith("A") && customer.Orders.Count > 0
  orderby customer.Name
  select new { customer.Name, customer.Phone };

var xml =                               <-- Generate XML data
  new XElement("contacts",                  from list of customers
    from contact in contacts
    select new XElement("contact",
      new XAttribute("name", contact.Name),
      new XAttribute("phone", contact.Phone)
    )
  );
```

Kuvio 6. LINQ-kysely tietokantaan ja XML-dokumentin luonti (Eichert 2000, 8)

3.2 Yhteyden muodostaminen SQL Server tietokantaan

Visual Studio 2008 ohjelmistoon on lisätty LINQ:n perustoiminnallisuuden lisäksi monia hyödyllisiä työkaluja. Yksi näistä työkaluista on LINQ to SQL Classes Designer (Kuvio 7). Tämän työkalun avulla on mahdollista luoda nopeasti yhteys suoraan olemassa olevaan tietokantaan. LINQ luo automaattisesti tietokannan tauluista valmiit luokkarakenteet, joita voidaan käyttää ohjelmassa. Yhteyden muodostamisen jälkeen tietokannan sisältöä voidaan käyttää suoraan LINQ-kyselyiden avulla. (Ferracchiati 2008, 126-136.)



Kuvio 7. LINQ to SQL Classes Designer (Ferracchiati 2008. 135)

4 KÄYTTÖLIITTYMÄSUUNNITTELU

Käyttöliittymä on yleensä käyttäjän kannalta merkittävin ohjelman osa. Huono käyttöliittymä voi helposti rajoittaa muuten ominaisuuksiltaan hyvää ohjelmaa. Hyvän graafisen käyttöliittymmän toteuttaminen on haasteellista. Mikäli ohjelma sisältää paljon toiminnallisuutta, tulee käyttöliittymästä helposti sekava. Toisaalta käytettävyyden parantaminen tarkoitaa yleensä ohjelmakoodin lisääntymistä ja monimutkaistumista. (Kalimo 1996, 9)

Käyttöliittymän suunnitteluun ja arviointiin on olemassa monia ohjeita ja suosituksia. Hyödyllisimmät ohjeistot ovat yleensä lyhyitä ja helposti muistettavia. Seuraavassa on esitelty kymmenen heuristista sääntöä, joita voidaan käyttää apuna käyttöliittymän suunnittelussa (Kalimo 1996, 38-44.):

1) Käytä yksinkertaista ja luonnollista dialogia

   Käyttöliittymässä tulee esittää vain tehtävän kannalta oleellinen tieto. Harvemmin tarvittava tieto ja toiminnallisuus voidaan siirtää erilliseen ikkunaan tai valikkoon, johon käyttäjä voi tarvittaessa helposti siirtyä. Tiedon esitys tulee toteuttaa luonnollisessa ja loogisessa järjestyksessä.

   Värejä voidaan käyttää apuna ryhmiteltäessä käyttöliittymän eri osia. Käyttöliittymässä tulee kuitenkin välttää käyttämästä turhaan räikeitä värejä.

2) Käytä käyttäjien omaa kieltä

   Käyttöliitymässä käytetyn kielen tulisi olla käyttäjän omaa äidinkieltä ja ammattitermistöä. Käyttäjälle esitetyt viestit tulisi esittää turhaa tietokonetermistöä välttäen. Tekstin lisäksi erilaisten kuvakkeiden ja ikonien valintaan kannattaa kiinnittää huomiota. Ikoneilla luodaan vertauskuvia esimerkiksi erilaisista ohjelman toiminnoista. Silloin kuin se on mahdollista, tulisi esimerkiksi valikoissa ja painikkeissa käyttää samaa sanastoa kuin käyttöjärjestelmässä. Esimerkiksi Windows-käyttöjärjestelmän sanastosta voi ottaa mallia, mikäli ohjelmaa tullaan käyttämään tässä ympäristössä.

3) Minimoi käyttäjän muistikuorma

Käyttäjän muistikuormaan voidaan helpottaa monilla tavoilla. Esimerkiksi kannattaa välttää saman tiedon kysymistä käyttäjältä ohjelman eri osissa. Valintatilanteissa käyttäjälle kannattaa esittää selkeästi vaihtoehdot, joista hän voi valita. Valikoilla voidaan vähentää käyttäjän tarvetta muistaa erilaisia komentoja. Erilaisissa syötteissä voidaan käyttää oletusarvoisia vastauksia. Tällä tavalla käyttäjälle esitetään missä muodossa vastaus tulee antaa. Numeerisia syötteitä käytettäessä käyttäjälle tulee ilmoittaa syötteen raja-arvot ja käytetty yksikkö.

4) Tee käyttöliittymästä kauttaaltaan yhdenmukainen

Käyttöliittymän yhdenmukaisuudella tarkoitetaan monia asioita. Ohjelmassa esiintyvät painikkeet ja ikkunat tulee sijoitella eri näkymissä samalla tavalla. Erilaisilla termeillä ja toimenpiteillä tulee tarkoittaa koko käyttöliittymässä samaa asiaa, muuten yhdenmukaisuus rikkoontuu. Yhdenmukaisuus helpottaa uusien toimintojen ja ominaisuuksien oppimista, koska käyttäjä tunnistaa osan toiminnallisuudesta esimerkiksi jostakin toisesta ohjelman osa-alueesta.

5) Anna käyttäjälle palautetta toiminnoista

Palautteen antaminen käyttäjälle on tärkeää, jotta hän kokee hallitsevansa järjestelmää. Palautetta tulisi antaa kaikista käyttäjän tekemistä toimenpiteistä, eikä vain virhetilanteista. Käyttäjä näkee palautteen avulla, että tehdyt valinnat ja annetut syötteet ovat tuottaneet halutun tuloksen. Lyhyissä toiminnoissa pelkkä tuloksen esittäminen voi riittää palautteeksi. Mikäli käyttäjän käynnistämä toiminto aiheuttaa useita sekunteja kestävän viiveen, pitää käyttäjälle ilmoittaa, että jotain on tapahtumassa. Pitkään kestävissä toiminnoissa edistymistä voidaan kuvata esimerkiksi palkilla, joka täyttyy toiminnon edetessä.

6) Anna selkeä poistumistapa eri tiloista ja toiminnoista

Ohjelman kaikissa osioissa tulisi olla selkeä poistumistapa. Tämä helpottaa käyttöliittymän opettelemista, koska käyttäjä voi helpommin kokeilla ohjelman eri toimintoja. Mikäli käyttäjä on vahingossa valinnut jonkun toiminnon, on tärkeää, että poistumiseen löytyy helppo ja nopea tapa. Poistumisella voidaan tarkoittaa esimerkiksi jonkun toiminnon pe-

rumista tai dialogi-ikkunasta poistumista. Mikäli toiminto on sellainen, ettei sitä voida suorittamisen jälkeen peruuttaa, kannattaa ennen toiminnon suoritusta pyytää erillinen varmistus käyttäjältä.

7) Anna käyttäjälle mahdollisuus käyttää oikopolkuja

Oikopolut ovat vaihtoehtoisia tapoja suorittaa ohjelman toimintoja. Erilaisia oikopolkuja tarvitaan, kun käyttäjä oppii ohjelman käytön ja haluaa hyödyntää sitä mahdollisimman tehokkaasti. Graafisessa käyttöliittymässä oikopolkuja voidaan luoda käyttämällä hyväksi erilaisia komentovalikoita, hiiren näppäimiä ja näppäinyhdistelmiä. Joissakin tapauksissa on mahdollista tallentaa käyttäjän antamat komennot ja tarjota niitä käyttäjälle myöhemmin, jolloin hän voi helposti toistaa haluamansa toiminnon. Ohjelman käyttöä voidaan nopeuttaa myös antamalla käyttäjällä oletusarvoja erilaisiin syötekenttiin. Esimerkiksi päiväyksen valinnassa voidaan asettaa kuluva päivä oletusarvoksi.

8) Anna virhetilanteista selkeät virheilmoitukset

Mahdollisissa virhetilanteissa käyttäjälle tulee näyttää selkeä ja mahdollisimman täsmällinen ilmoitus käyttäjän omalla kielellä. Ohjelman pitäisi pystyä esittämään eritasoisia virheilmoituksia käyttäjän pyynnöstä. Esimerkiksi ohjelman ylläpitäjä haluaa virhetilanteessa mahdollisimman tarkan tiedon virheestä, mutta tavalliselle käyttäjälle riittää yksinkertaisempi ilmoitus.

9) Vältä virhetilanteita

Osa virhetilanteista on mahdollista välttää kokonaan erilaisia menetelmiä käyttämällä. Kirjoitusvirheitä on mahdollista välttää joissakin tilanteissa antamalla käyttäjälle lista vaihtoehdoista sen sijaan, että käyttäjä itse kirjoittaa haluamansa vaihtoehdon tai komennon. Varmistuksen kysyminen käyttäjältä on suotavaa silloin kun käyttäjä on suorittamassa toimintoa, jota ei voida perua. Ohjelman toimintatilojen vaihtuminen kannattaa esittää käyttäjälle selkeästi. Tällöin vältytään tilanteelta, jossa käyttäjä suorittaa toiminnon joka ei ole hyväksyttävä valitussa tilassa.

10) Anna riittävä ja selkeä apu sekä dokumentaatio

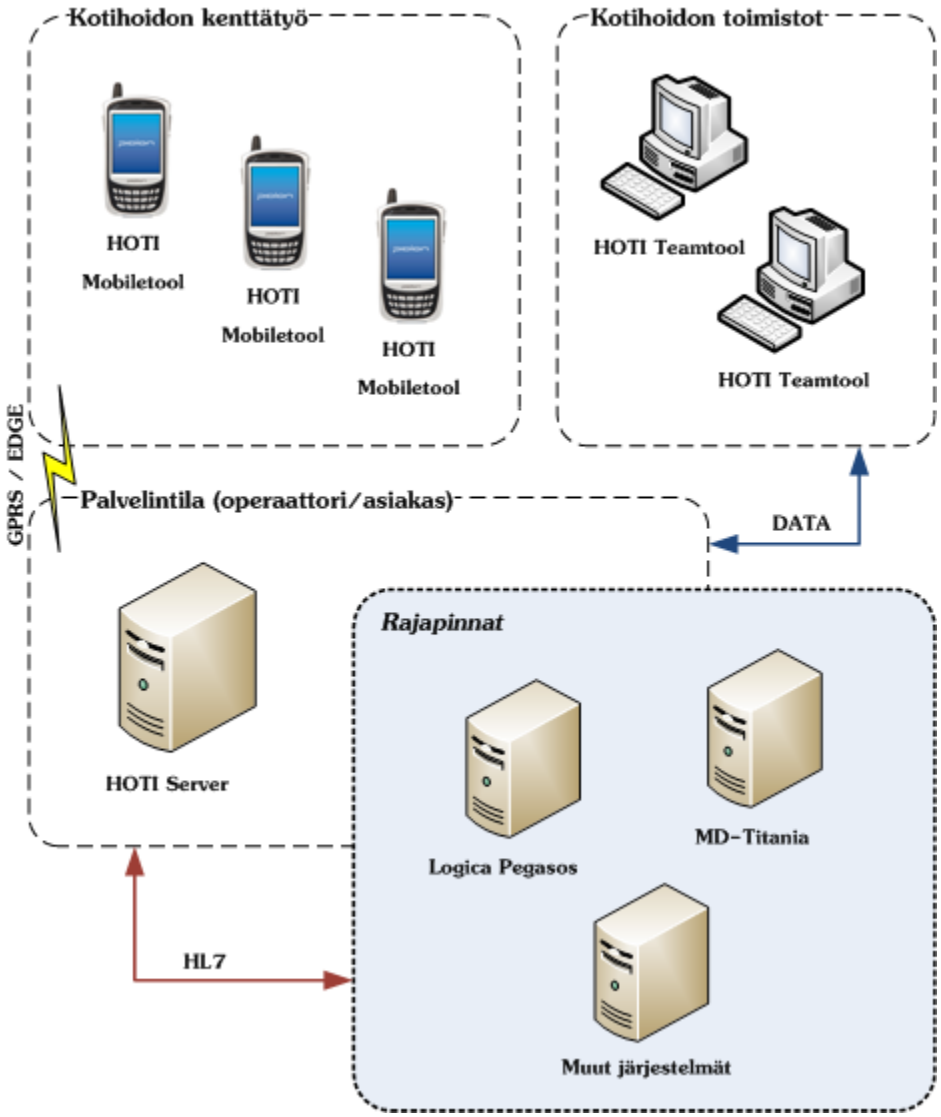
Käyttöliittymä on harvoin sellainen, että käyttäjä pystyy käyttäämään sitä ilman erillisiä ohjeita. Tästä syystä käyttäjää pitäisi pystyä opastamaan ohjelman käytössä erilaisilla tavoilla. Ohjelmaan voidaan esimerkiksi rakentaa sisäinen opaste, josta käyttäjä voi katsoa tarvittaessa ohjeita. Käyttäjää voidaan myös avustaa ns. puhekuplaohjeiden avulla. Puhekuplaohje voi toimia esimerkiksi siten, että käyttäjä vie hiiren kursorin näppäimen päällä, jolloin näytölle ilmestyy hetken kuluttua seloste näppäimen toiminnasta.

5 HOTI MANAGEMENT TOOL

Tämän opinnäytetyön tavoitteena oli kehittää ominaisuuksiltaan ja käytettävyydeltään toimiva ohjelma, jota voidaan hyödyntää HOTI:n ylläpidossa. Ohjelma tulee pääasiassa Luovaliike Oy:n käyttöön, mutta myöhemmin käyttäjinä voi toimia esimerkiksi asiakkaan ylläpitohenkilöstö. Ohjelman nimeksi annettiin HOTI tuoteperheen mukaisesti HOTI Management Tool.

5.1 Projektin taustaa

Luovaliike Oy on kehittänyt HOTI kotihoidon tietojärjestelmän (Kuvio 8). HOTI on työkalu kotipalvelun ja kotisairaanhoidon työntekijöille. Järjestelmän pääasialliset sovellukset ovat HOTI Mobiletool ja HOTI Teamtool. HOTI Mobiletool on nimensä mukaisesti mobiililaitteella toimiva ohjelma. HOTI Teamtool on työpöytäsovellus, jonka avulla voidaan suorittaa mm. työn ohjausta ja seurantaa. Lisäksi järjestelmään kuuluu erinäisiä palvelinsovelluksia, joiden tehtävinä on huolehtia järjestelmän sisäisestä tiedonsiirrosta ja yhteyksistä asiakkaan taustajärjestelmiin.



Kuvio 8. HOTI kotihoidon tietojärjestelmä

Ylläpidon kannalta ajateltuna HOTI on haasteellinen kokonaisuus. Asiakkaalla voi olla käytössä esimerkiksi useita kymmeniä mobiililaitteita. Tällöin laitteista tulee olla tallessa tarkat tiedot, jotta ylläpitotoimenpiteet voidaan suorittaa nopeasti ja tehokkaasti.

5.2 Vaatimusmäärittely

Työn toteutus aloitettiin vaatimusmäärittelyn pohjalta, jossa kuvattiin mitä tietoja ohjelmassa käsitellään ja tallennetaan tietokantaan. Erilaisia tietokonaisuuksia oli useita, joista jokainen sisälsi kohtuullisen määrän erilaista tietoa.

Vaatimusmäärittelyn mukaiset tietokokonaisuudet olivat seuraavanlaiset:

- Asiakkaan tiedot. Asiakkaan tiedoista haluttiin tallentaa sopimustyyppi ja ylläpitoa koskevat tarkennukset. Lisäksi tarvittiin tiedot asiakkaiden palveluksessa olevista henkilöistä, joihin täytyy olla yhteydessä erilaisiin ylläpitotehtäviin liittyen.

- Laitetiedot. Laitteista tarvittiin mahdollisimman kattavat tiedot, jotta ylläpito voidaan hoitaa mahdollisimman sujuvasti. Lisäksi piti pystyä kirjaamaan laitteiden huoltohistoriatietoja.

- Sovellustiedot. Asiakkaan käytössä olevista sovelluksista tallennetaan versiotiedot ja muita tarkentavia tietoja.

- Tiedot ylläpitohenkilöistä. Ohjelma tulee ylläpitohenkilöstön käyttöön, joten henkilöistä tallennetaan perustiedot ja heille luodaan käyttäjätunnukset.

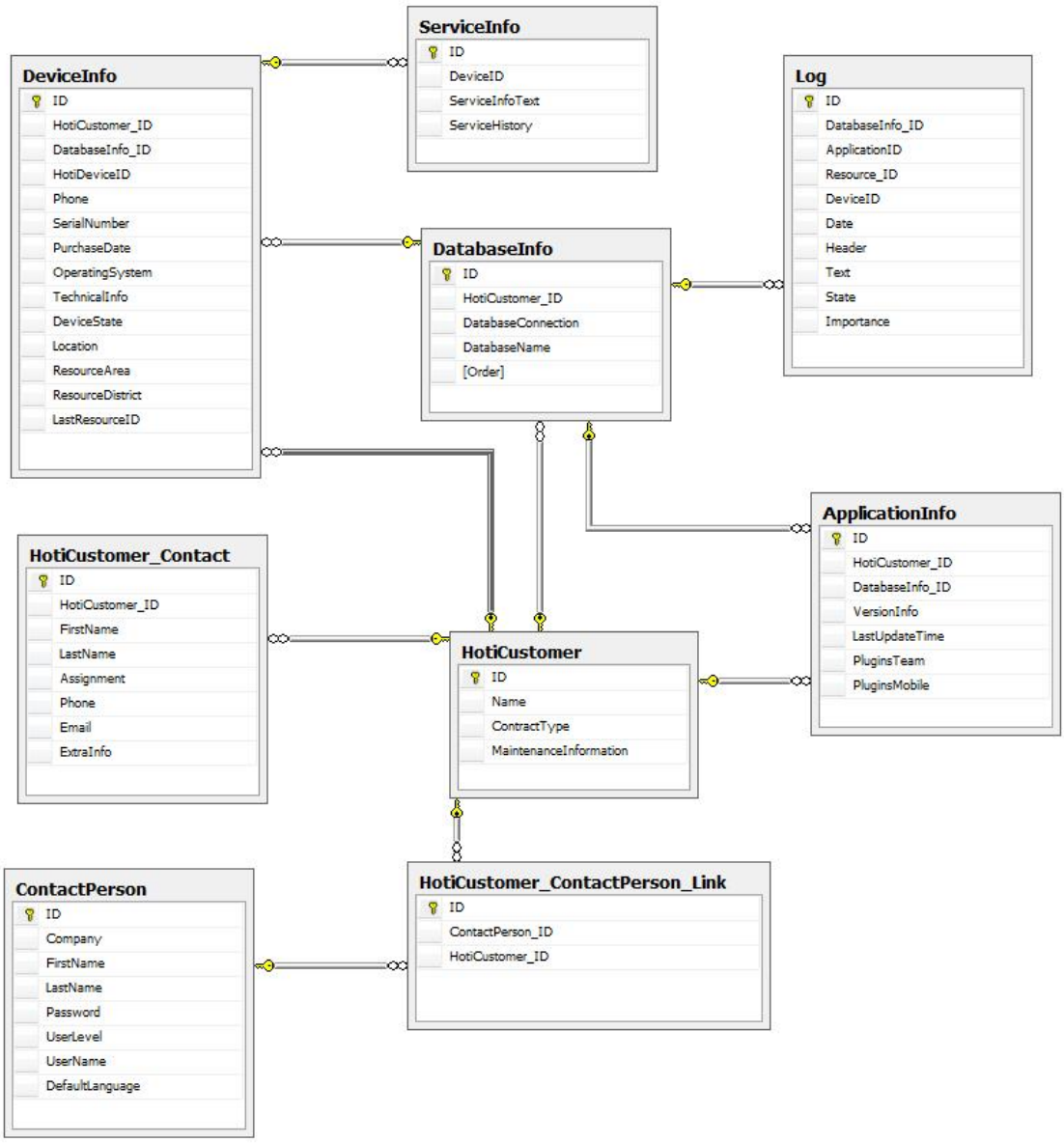
Käyttöliittymän osalta alkuperäisessä vaatimusmäärittelyssä ei ollut useita vaatimuksia. Vaatimuksia kuitenkin tarkennettiin käyttöliittymän ja muun toiminnallisuuden osalta projektin edetessä. Yhtenä käyttöliittymää koskevana vaatimuksena oli, että eri näkymiä voi rajoittaa käyttäjätasojen mukaisesti. Käyttäjätasojen avulla osa näkymistä piti pystyä piilottamaan kokonaan tai rajaamaan siten, että tietojen muokkaus ei ole sallittua.

5.3 Toteutus

Ohjelman toteutuksessa käytettiin Visual Studio 2008 ja SQL Server 2005 ohjelmistoja. Ohjelmointiin käytettiin C#-ohjelmointikieltä ja ohjelmistoalustana toimi .NET Framework 3.5. Kaikki ohjelmassa käytetyt tietokantakyselyt on tehty käyttäen LINQ-tekniikoita. Ohjelman obfuskointiin käytettiin Xenocode Postbuild ohjelmaa.

Ensimmäinen vaihe ohjelman toteutuksessa oli suunnitella ja toteuttaa tietokantarakenne (Kuvio 9). Kun tietokanta oli saatu valmiiksi, oli mahdollista suunnitella, kuinka tietoja esitetään ja käsitellään käyttöliittymässä.

Projektin edetessä pidettiin palavereita, joissa arvioitiin käyttöliittymän ja ohjelman toiminnallisuuden sen hetkistä tilannetta. Tällä tavoin voitiin muokata alkuperäisiä suunnitelmia, mikäli jokin valittu toimintatapa ei tuntunut toimivalta.



Kuvio 9. Tietokantarakenne

5.4 Tietoturva

Ohjelmalla tullaan käsittelemään luottamuksellista tietoa, joten on tärkeää, että ohjelman tietoturvaominaisuudet ovat kunnossa. Salasanojen käsittelyn yhteydessä ohjelmassa on käytetty tiivisteitä ja suolausta. Nämä menetelmät on esitelty kappaleissa 2.1 ja 2.2. Lopullisia ohjelman käyttäjiä tulisi opastaa käyttämään vahvoja salasanoja, jotta voitaisiin olla varmoja salasanakäytäntöjen turvallisuudesta. Mahdollista olisi myös pakottaa valitsemaan käyttäjälle vahva salasana uuden käyttäjän luonnin yhteydessä. Tämä toiminto on kuitenkin vielä ideatasolla, eikä sen lisäämisestä ohjelmaan ole tehty päätöstä.

Ohjelman asetustiedosto on salattu käyttäen kappaleessa 2.3 esitettyä symmetristä salausta. Tarvittaessa myös tietokantaan tallennetut tiedot voitaisiin erikseen salata. Tietojen salausta ei kuitenkaan ole vielä toteutettu.

5.5 Käyttöliittymä

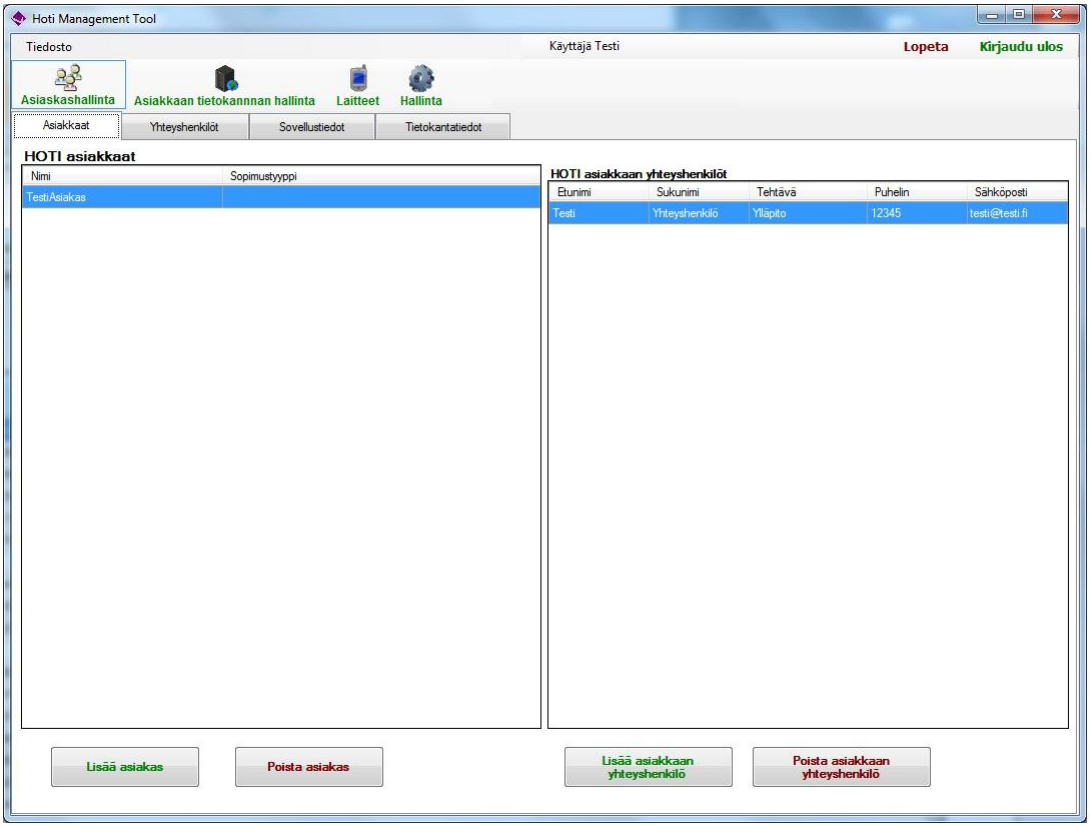
Käyttöliittymä oli haastava kokonaisuus opinnäytetyön tekemisessä. Osa-alueita ja hallittavaa tietoa oli paljon. Käyttöliittymän ulkoasun ja toimintojen osalta ohjelmassa päätettiin käyttää HOTI Teamtool sovelluksessa hyväksi koettuja käytäntöjä.

Vaatimusmäärittely sisälsi paljon erilaisia tietokokonaisuuksia, joten alkuun täytyi suunnitella millaisiin osiin käyttöliittymä voidaan jakaa. Ohjelman toiminnallisuus jaettiin toisistaan riippumattomiksi loogisiksi kokonaisuuksiksi, jotka näkyvät ohjelman käyttäjälle erilaisina näkyminä. Käyttäjä voi liikkua näkymien välillä ohjelman yläpalkissa olevia ikoneita klikkaamalla (Kuvio 10). Käyttöliittymän suunnittelussa on pyritty noudattamaan kappaleessa 4 esiteltyjä suunnitteluohjeita.

Kuvio 10. Ohjelman päänäkymä

5.5.1 Asiakashallinta

Asiakashallinta-näkymään (Kuvio 11) on koottu omille välilehdilleen asiakkaaseen liittyvät tiedot. Asiakkaat välilehdellä käyttäjä voi selata ja muokata asiakkaan perustietoja sekä asiakkaan yhteyshenkilötietoja. Seuraava välilehti sisältää tiedot ylläpitohenkilöistä ja mitkä asiakkaiden järjestelmät ovat heidän vastuualueenaan. Sovellustiedot välilehdellä on esitetty tiedot asiakkaiden käyttämistä sovellusversioista. Sovellusversiotiedot on mahdollista päivittää suoraan asiakkaan omasta tietokannasta. Viimeisellä välilehdellä hallitaan asiakkaiden tietokantayhteystietoja.

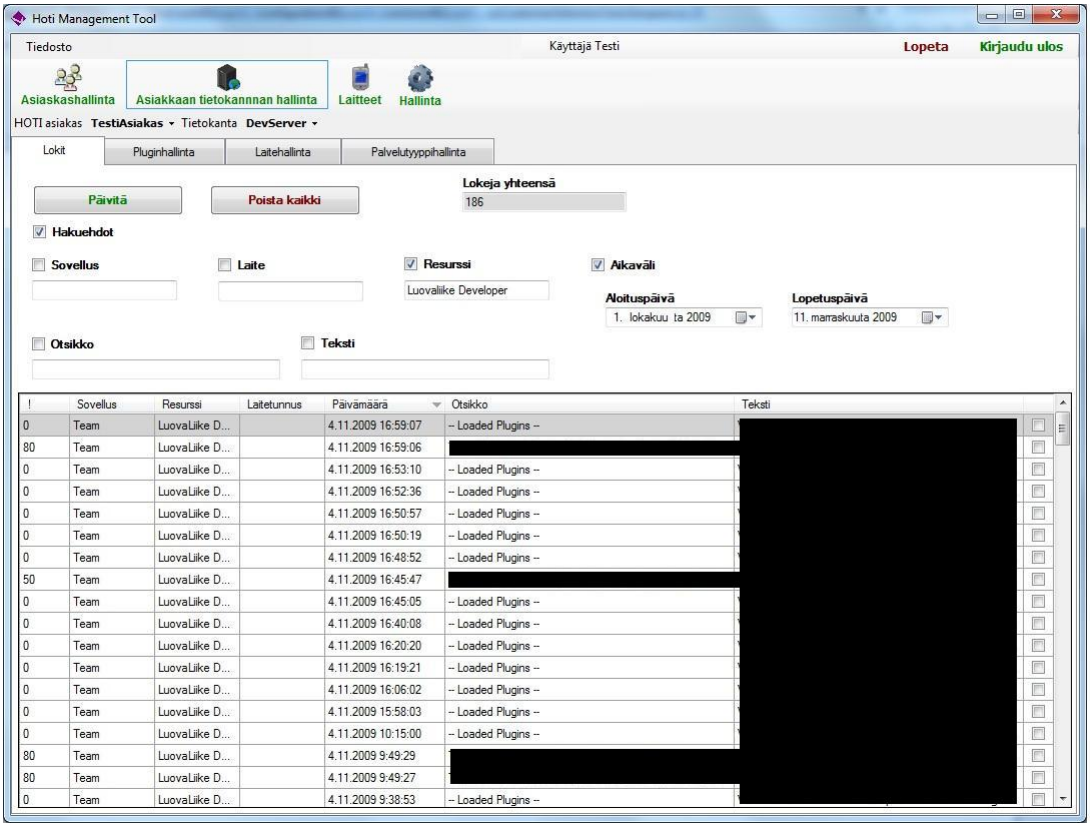


Kuvio 11. Asiakashallinta

Yhteyshenkilöt-välilehdellä sijaitsevat henkilöt ovat sovelluksen käyttäjiä, joten uuden yhteyshenkilön luonnissa tallennetaan perustietojen lisäksi käyttäjätiedot. Käyttäjätietoja ovat käyttäjänimi, salasana, käyttäjätaso ja käyttäjän kieli.

5.5.2 Asiakkaan tietokantojen hallinta

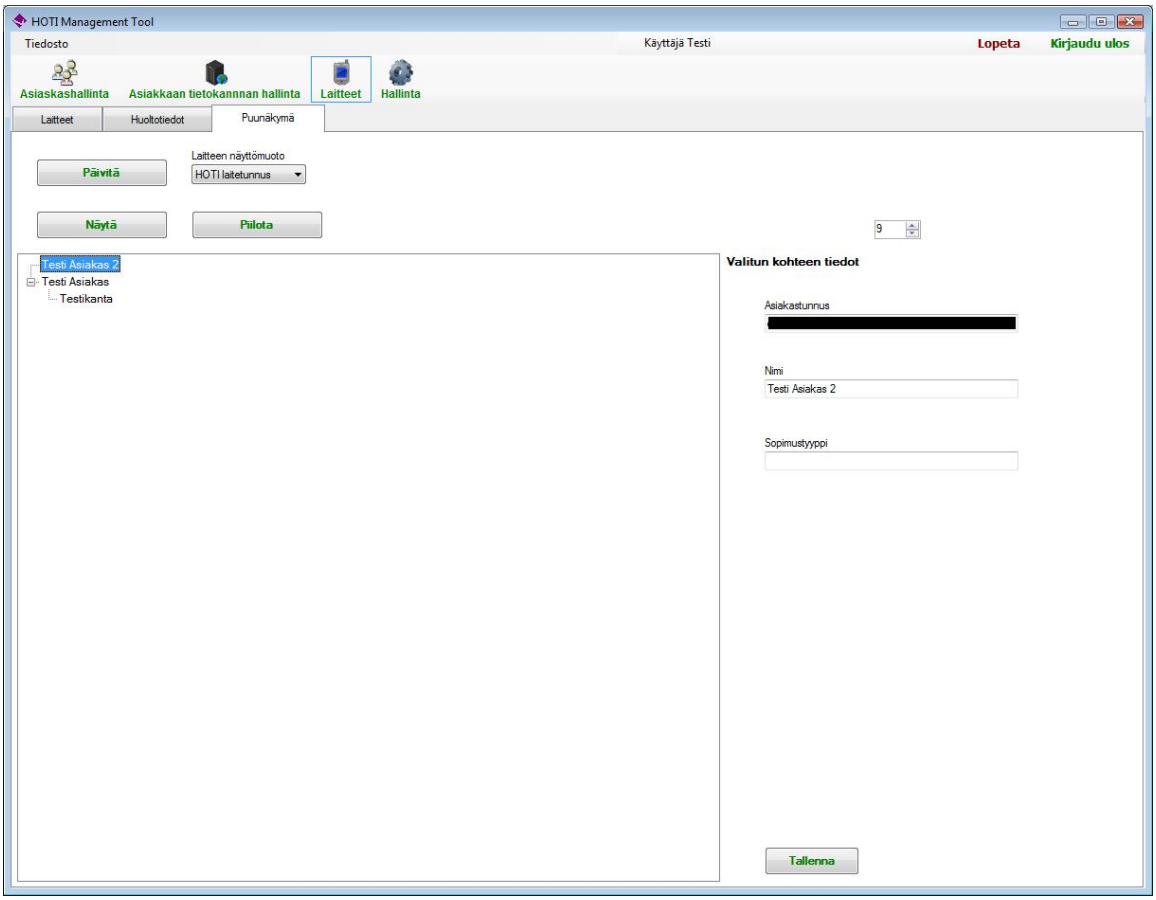
Asiakkaan tietokantojen hallinta -näkymässä (Kuvio 12) käyttäjä voi yhdistää haluamansa asiakkaan tietokantaan. Lokit-välilehdellä voidaan tarkastella HOTI:n tuottamia lokitietoja. Näytettäviä lokeja voidaan suodattaa erilaisten hakuehtojen avulla. Näitä hakuehtoja ovat mm. lokin luontipäivämäärä ja sovellus, josta lokit on luotu. Pluginhallinta-välilehdellä on mahdollista tarkastella ja tarvittaessa muuttaa asiakkaan käyttämän HOTI Teamtool ohjelman asetuksia. Laitehallinta-välilehdellä näytetään valitun asiakkaan mobiililaitetietoja. Tällä välilehdellä olevat tiedot ovat suppeampia verrattuna ylläpito-ohjelman tietokannassa säilytettäviin mobiililaitetietoihin.



Kuvio 12. Asiakkaan tietokannan hallinta

5.5.3 Laitetiedot

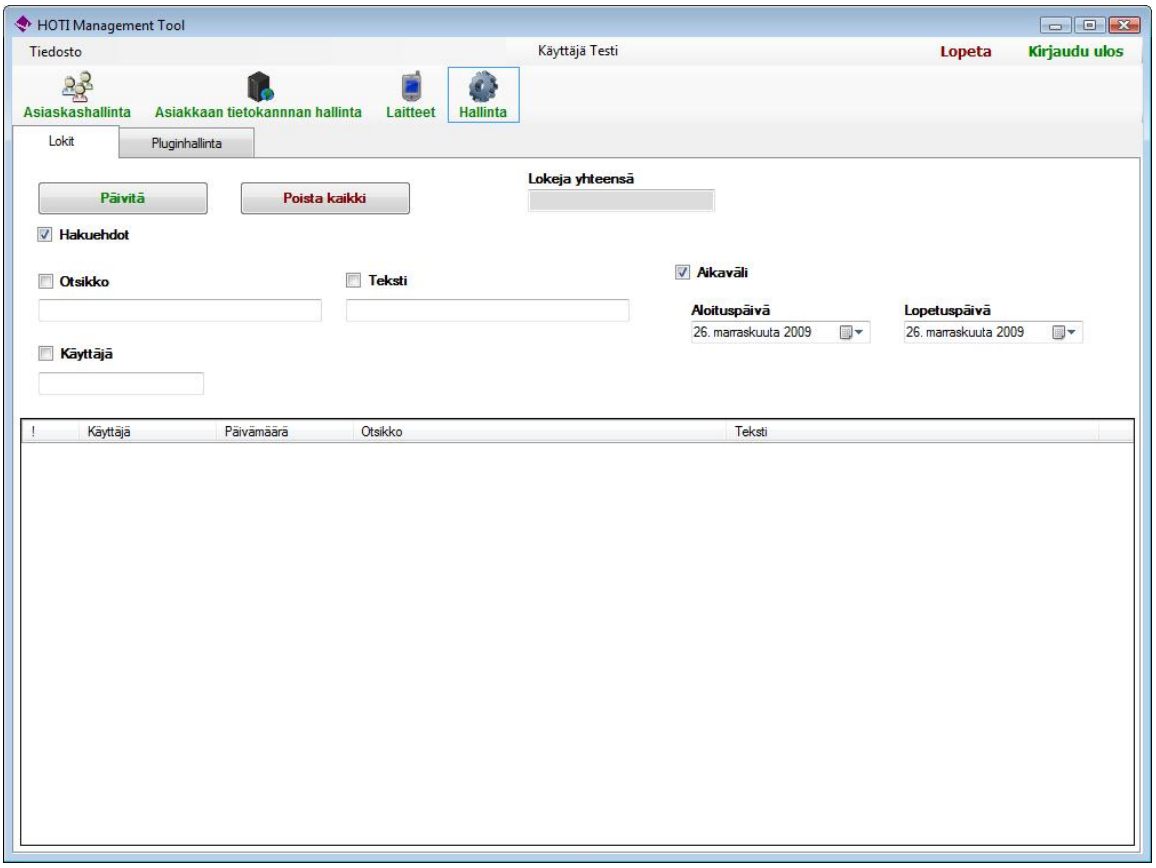
Laitteet näkymä (Kuvio 13) sisältää kolme välilehteä. Laitteet-välilehdellä on listattu kaikki laitteet taulukkomaisesti. Näytettäviä laitteita voidaan suodattaa erilaisten hakuehtojen avulla. Huoltotiedot-välilehdellä voidaan tutkia ja muokata laitteiden huoltohistoriaa. Viimeisellä välilehdellä laitteet näytetään puumaisessa rakenteessa, jossa laitteet on jaoteltu asiakkaittain ja käyttäjäryhmittäin. Käyttäjä voi tarvittaessa muokata laitetietoja.



Kuvio 13. Laitteet

5.5.4 Ohjelman hallinta

Viimeissä ohjelman näkymässä (Kuvio 14) hallitaan itse ohjelman asetuksia. Tämä näkymä on yleensä käytössä vain päätason käyttäjille. Näkymä sisältää kaksi välilehteä. Ensimmäisellä välilehdellä voidaan tarkastella ohjelman lokitietoja samaan tapaan kuin lokit-välilehdellä asiakkaan tietokannan hallinta -näkymässä. Toisella välilehdellä voidaan muuttaa ohjelman asetuksia. Käyttäjä voi muuttaa esimerkiksi mitä näkymiä tai välilehtiä näytetään eri käyttäjätasojen käyttäjille.



Kuvio 14. Hallinta

6 POHDINTA

Tietoturvamenetelmien käsittely teoriaosuudessa oli tarpeellista, koska oma tietotaito kyseiseltä osa-alueelta oli ainakin osittain puutteellista. Täysin uutena aihealueena opinnäytetyössä olivat LINQ-tekniikat. LINQ-kyselyiden käytön oppiminen oli kuitenkin odotettua helpompaa. Tietokantarakenteen huolellinen suunnittelu ennen varsinaisen ohjelmointityöskentelyn aloittamista osoittautui tärkeäksi. Tietokannassa olevien taulujen ja tiedon määrän vuoksi muutosten tekeminen tietokantaan aiheuttaa helposti paljon lisätyötä ohjelmointipuolella. LINQ:n ansiosta muutosten tekeminen tauluihin ei kuitenkaan aiheuta niin paljon lisätyötä kuin vanhempia menetelmiä käytettäessä.

Työn tavoitteena oli kehittää ominaisuuksiltaan ja käytettävyydeltään toimiva ohjelma, jota voidaan hyödyntää HOTI:n ylläpidossa. Tavoitteen saavuttamisessa onnistuttiin kohtuullisen hyvin. Joitakin ohjelmaan ideoituja ominaisuuksia jouduttiin karsimaan kehitysvaiheessa, mutta ohjelmaan saatiin toteutettua ne ominaisuudet, minkä takia kehitystyö alun perin aloitettiin. Ohjelman kehitystä jatketaan opinnäytetyön valmistuttua.

LÄHTEET

Barak Boaz. 2001. Can We Obfuscate Programs? Web-dokumentti. Saatavilla: http://www.cs.princeton.edu/~boaz/Papers/obf_informal.html (Luettu 20.10.2009)

Burnett Mark, Foster James. 2004. Hacking the Code: ASP.NET Web Application Security. Syngress Publishing. ISBN 1932266658.

Eichert Steve, Marguerie Fabrice, Wooley Jim. 2008. LINQ in Action. Manning Publications Co. ISBN: 1-933988-16-9.

Ferracchiati Fabio Claudio. 2008. LINQ for Visual C# 2008. Apress.
ISBN 978-1-4302-1580-6.

Hamilton Bill. 2008. ADO.NET 3.5 Cookbook. O'Reilly Media. ISBN 9780596101404

Kalimo Anna. 1996. Graafisen käyttöliittymän suunnittelu. Tietotekniikan kehittämiskeskus TIEKE ry. ISBN 9517623828

Karttuenn, E. 1999. Kryptologia. Web-dokumentti. Saatavilla: http://www.seepia.org/html/seepia1/kryptologia/kryptologia.shtml (Luettu 20.10.2009).

Microsoft. 2005. Data Confidentiality. Web-dokumentti. Saatavilla:
http://msdn.microsoft.com/en-us/library/aa480570.aspx (Luettu: 4.11.2009)

Mueller John Paul. 2008. LINQ For Dummies. Wiley Publishing, Inc. ISBN 9780470277942.

Northrup Tony. 2009. Microsoft .NET Framework-Application Development Foundation. Microsoft Press. ISBN 978073562619.

Oh Damien. 2008. How To Create Strong Passwords That You Can Remember Easily. Web-dokumentti. Saatavilla: http://www.makeuseof.com/tag/how-to-create-strong-password-that-you-can-remember-easily/ (Luettu: 10.11.2009)

Tyma Paul. 2003. Encryption, Hashing, and obfuscation. Web-dokumentti. Saatavilla: http://news.zdnet.com/2100-9595_22-128604.html (Luettu 20.10.2009)