SAVONIA

Soittolistojen hallintatyökalu Avaya Contact Center -järjestelmään

Aleksi Rytkönen

Opinnäytetyö

___. ___. _____ _____

Ammattikorkeakoulututkinto

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ**
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Aleksi Rytkönen	
Työn nimi Soittolistojen hallintatyökalu Avaya Contact Center -järjestelmään	
Päiväys 29.5.2012	Sivumäärä/Liitteet 52
Ohjaaja(t) Markku Halttunen	
Toimeksiantaja/Yhteistyökumppani(t) Call Waves Solutions Finland Oy, Esa Tuovinen	

Tiivistelmä

Insinöörityön aiheena oli kehittää toimeksiantajalle soittolistojen hallintatyökalu Avaya Contact Center –järjestelmään. Pääasiallinen tavoite oli helpottaa puhelinjärjestelmän numeroiden siivoamista, koska siitä oli tullut hidasta ja raskasta. Samalla järjestelmä alkoi olla siivouksen tarpeessa yhä useammin.

Sovellus kehitettiin C# -ohjelmointikielellä käyttäen Microsoftin Visual Studio 2010 –kehitysympäristöä. Tietojen tallennukseen käytettiin Microsoftin SQL Server –tietokantaa ja Microsoft Excel –rajapintaa. Sovellukseen suunniteltiin neljä osiota: soittojärjestelmän siivous, soittolistojen luonti, puhelinnumeroiden siivous ja soittolistojen tilanne. Näistä soittolistojen tilanteen seuraus oli ainoa, joka löytyi järjestelmästä itsestään. Sovelluksen kehitys aloitettiin soittojärjestelmän siivous –osiosta, koska oli tärkeää saada se mahdollisimman nopeasti käyttöön.

Insinöörityön lopputuloksena oli toimiva hallintatyökalu Call Wavesin puhelinjärjestelmään. Työkalun avulla järjestelmän siivoaminen onnistuu yhdeltä henkilöltä muutamassa tunnissa, kun se aikaisemmin vaati useamman henkilön työpanoksen päivien ajalta. Soittolistojen tilanteen seuraus jätettiin lopullisesta työkalusta pois, koska sitä ei koettu tarpeelliseksi.

Avainsanat
Avaya Contact Center, C#, SQL Server

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS**
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Aleksi Rytkönen			
Title of Thesis Call List Management Tool for Avaya Contact Center System			
Date	29.5.2012	Pages/Appendices	52
Supervisor(s) Markku Halttunen			
Client Organisation/Partners Call Waves Solutions Finland Oy, Esa Tuovinen			

Abstract

The topic of this thesis was to develop a call list management tool for Avaya Contact center system. The main objective was to ease the cleaning of the phone numbers in the system because it had become slow and difficult. In the same time the system was in a need of cleaning more often.

The application was developed in C# programming language using the Microsoft Visual Studio 2010 integrated development environment. Microsoft SQL Server database and Microsoft Excel interface were used for saving the application data. The application was planned to consist of four parts: cleaning of the system, creation of call lists, cleaning of the phone numbers and status of the call lists. The system itself had originally only the status of the call lists function. The development was started from the cleaning of the system part because it was the most important part of the application.

The result of this thesis was a functional management tool for the contact center system. Now it was possible to clean the whole system in a few hours by one person only instead of keeping a few employees busy for couple of days. The call list status monitoring did not make it to the final application because it was not considered necessary.

Keywords
Avaya Contact Center, C#, SQL Server

SISÄLTÖ

1 JOHDANTO 7
2 Avaya contact center -puhelinjärjestelmä 8
3 Kehitysvälineet 9
3.1 Microsoft Visual Studio 2010 –kehitysympäristö 9
3.2 Microsoft SQL Server 10
4 Ohjelmiston kehitysprosessi 12
4.1 Olio-ohjelmointi 12
4.1.1 Tietokantayhteyden muodostaminen SQL Server-tietokantaan 14
5 Vaatimukset 15
5.1 Soittojärjestelmän siivous 15
5.2 Soittolistojen luonti 15
5.3 Soittolistan puhelinnumeroiden siivous 16
5.4 Soittolistojen tilanne 16
6 Kehitys 17
6.1 Soittojärjestelmän siivous-osan kehitys 17
6.2 Luo soittolista -toiminnon kehitys 34
6.2.1 Soittolistan manuaalinen luominen 39
6.2.2 Soittolistan luominen mallipohjan avulla 44
6.3 Soittolistan puhelinnumeroiden siivous-osan kehitys 47
6.4 Soittolistojen tilanne 50
7 Yhteenveto 51
LÄHTEET 52

Termit ja lyhenteet

Agentti	Agentti tarkoittaa puhelinalalla työntekijää, joka soittaa ja vastaanottaa puheluita.
Soittolista	Soittolistalla tarkoitetaan puhelinjärjestelmään vietävää toimeksiantajan toimittamaa listaa kontakteista, joille Call Wavesin agentit soittavat toimeksiannon mukaisesti.
Syykoodi	Kontaktin päätöskoodi, jolla määritellään miten asiakaskontakti on päättynyt.
Kehitysympäristö	Ympäristö/Ohjelmisto jolla kehitetään sovelluksia, eli tuotetaan ohjelmakoodia.
Debug	Virheenkorjaus, eli etsitään ohjelmakoodista virheitä.
.NET (Framework)	Sovellusalusta, jonka päällä .NET sovelluksia ajetaan (vrt. Java).
IDE	Integrated Development Environment, eli nippu työvälineitä, joilla voidaan kehittää valmis sovellus.
VoIP	Voice over Internet Protocol tarkoittaa äänen kuljettamista IP-protokollan avulla dataverkkoja pitkin normaalin puhelinlinjan sijaan.

1 JOHDANTO

Tämän insinöörityön aiheena on soittolistojen hallintatyökalun kehitys Call Waves Solutions Oyj:n Avaya Contact Center -puhelinjärjestelmään. Soittolistojen hallinta vie nykyisellään todella paljon Call Wavesin tekniikan osaston työaikaa. Tarve soittolistojen hallintaa helpottavalle työkalulle on näin ollen suuri.

Soittolistojen hallinta voidaan jakaa karkeasti kahteen osa-alueeseen: puhelinjärjestelmän siivoukseen ja soittolistojen luontiin. Näistä kahdesta kriittisempi ja työkalun pääasiallinen käyttökohde on puhelinjärjestelmän siivous. Tässä raportissa käsitellään sovelluksen kehitysvaiheet vaatimusmäärittelyistä kehitystyöhön, ja aina käyttöönottovaiheeseen asti. Johdannon jälkeen käydään läpi puhelinjärjestelmän rakenne teoriassa. Tämän jälkeen esitellään työssä käytetty kehitysympäristö ja keskeisimmät menetelmät, jonka jälkeen tarkastellaan varsinaista sovelluskehitystä ja ohjelmakoodia. Viimeisissä luvuissa pohditaan jatkokehitysideoita ja analysoidaan projektin onnistumista.

2 Avaya contact center -puhelinjärjestelmä

Call Waves Solutions Finland Oyj:llä on käytössään Yhdysvaltalaisen Avayan voip-teknologiaan perustuva prediktiivinen soittojärjestelmä. Call Wavesilla on kahden tyyppistä puheluliikennettä, sisääntulevaa ja ulosmenevää. Tämä työ liittyy enimmäkseen ulosmenevään puheluliikenteeseen. (Avaya, Contact Centers)

Soittolistat ovat tyypillisesti toimeksiantajilta tulevia Excel-muotoisia listoja, jotka sisältävät vaihtelevan määrän kontakteja eli puhelinnumeroita, joihin soittojärjestelmä soittaa. Ongelmia aiheuttaa listojen standardimuodon puuttuminen joten toimeksiantajien toimittamat listat ovat keskenään täysin erilaisia. Lisäksi pahimmillaan saman kampanjan sisällä toimitetut listat ovat keskenään erilaisia.

Soittolistassa keskeisessä osassa ovat luonnollisesti puhelinnumerot. Toimeksiantajalta tulevat listat voivat olla, ja useimmiten ovat, suoraan jonkinlaisesta yrityksen omasta asiakasrekisteristä tulostettuja listoja. Tästä johtuen numerot voivat olla monenlaisessa eri muodossa, ja sisältää niihin kuulumattomia merkkejä. Call Wavesilla käytössä oleva Avaya-järjestelmä on todella tarkka puhelinnumeroiden suhteen. Jos listan jossakin numerossa on yksikin ylimääräinen ei-numeraalinen merkki, järjestelmä ei ota listaa ollenkaan vastaan. Järjestelmässä ei ole itsessään sisäistä puhelinnumeroiden puhdistusohjelmaa, joten tällaiselle ominaisuudelle on myös käyttöä. Parhaimmillaan tällainen ominaisuus säästää useita työtunteja, koska ylimääräisiä merkkejä ei tarvitse metsästää käsin tuhansien numeroiden seasta.

Tärkein tältä insinöörityöltä vaadittava ja samalla Avaya-järjestelmästä kokonaan puuttuva ominaisuus on kuitenkin järjestelmän puhdistussovellus. Puhdistuksella tarkoitetaan yksinkertaisesti jo soitettujen numeroiden siivoamista järjestelmästä. Jos siivousta ei suoriteta aika ajoin, järjestelmän toiminta ja tehokkuus huononee.

3 Kehitysvälineet

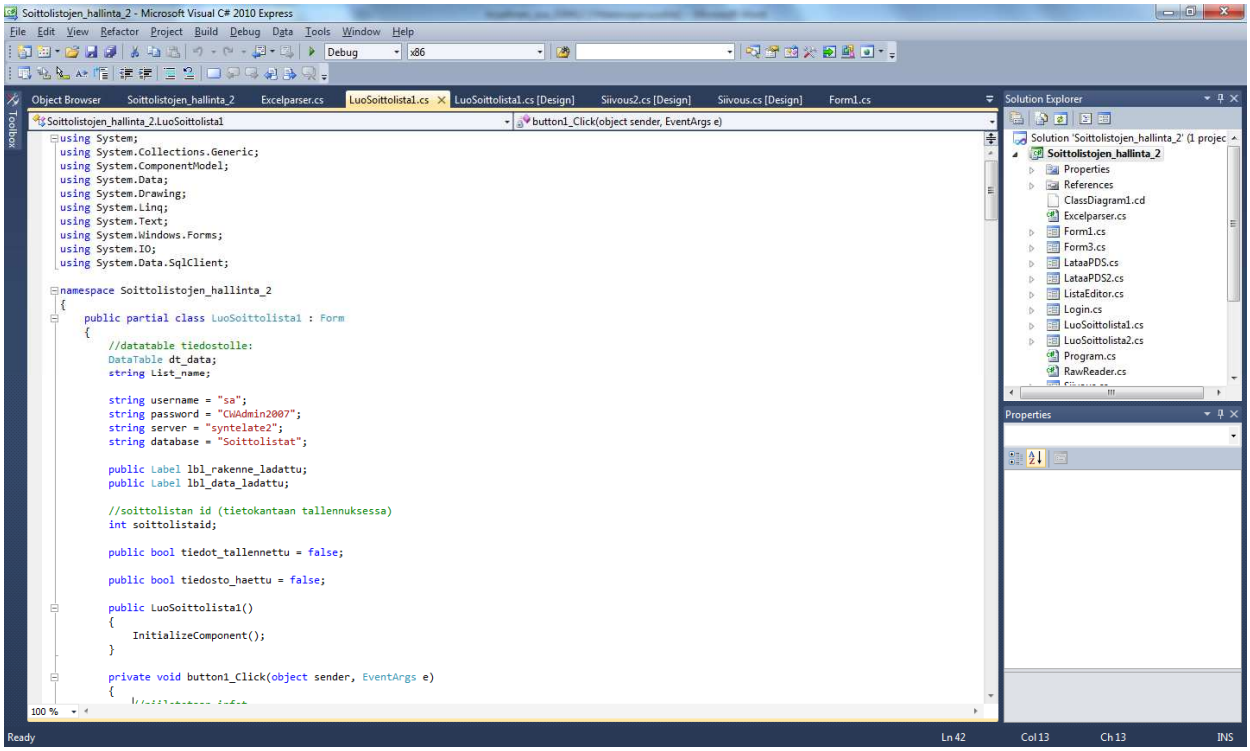
Kehitysympäristöksi projektiin valittiin Microsoftin Visual Studio. Projektin kehitys aloitettiin Visual Studio 2008:lla ja se vaihtui projektin puolessa välissä Visual Studio 2010:een. Kehityskieleksi valittiin C#.NET. Kielen valintaa helpotti aikaisempi kokemus ja tehokkaat tietokantojen käsittelyominaisuudet. Lisäksi suurin osa Call Wavesin tietokantapalvelimista on Microsoftin SQL Servereitä, joten Microsoftin tuottama ohjelmointikieli oli luonnollinen valinta tietokantojen käsittelyyn. Lisäksi ohjelmaa käytetään vain Windows-ympäristössä.

3.1 Microsoft Visual Studio 2010 –kehitysympäristö

Visual Studio -kehitysympäristö oli tuttu jo monen vuoden ajalta. Projekti aloitettiin ensin 2008-versiolla, mutta ympäristö vaihdettiin 2010-versioon sen ensimmäisten versioiden ilmestyttyä. Kehitysympäristön valintaa helpottivat myös Visual Studion tehokkaat debug- eli virheenkorjaustyökalut. Näillä virheenkorjaustyökaluilla on mahdollista porautua todella syvälle ohjelman toimintaan, ja selkeän käytettävyyden ansiosta ohjelmakoodissa olevien virheiden analysointi on helppoa. Visual Studiolla voidaan tuottaa vain .NET-ympäristössä toimivia sovelluksia (pl. jotkin C++-sovellukset). (Järvinen 2008, Visual Studion uusin versio)

Visual Studiosta puhuttaessa törmää lyhenteeseen IDE, eli Integrated Development Environment. IDE tarkoittaa työkalua, jolla voidaan tuottaa sovellus alusta loppuun. Visual Studio sisältää kaikki ohjelmistokehitykseen tarvittavat välineet, joten se on IDE.

C#.NET on Microsoftin ISO-standardoitu olio-pohjainen ohjelmointikieli. Kieli kehitettiin alun perin tavoitteena C++-kielen tehokkuus ja Javan helppokäyttöisyys. Kielen syntaksi muistuttaakin paljon em. kieliä. Kielen suurimmat erot esimerkiksi C++:aan verrattuna ovat lähinnä muistinkäsittelyyn liittyviä. C# pyrkii siivoamaan automaattisesti ohjelman jättämät jäljet muistista, mikä on useissa tapauksissa hyvä asia. Näin ohjelmasta ei jää niin helposti rippeitä muistiin viemään resursseja, kun ohjelma on käynnissä pidempään.



KUVA 1. Visual Studio 2010:n käyttöliittymä

3.2 Microsoft SQL Server

Microsoft SQL Server on Microsoftin vastine Oraclen ja muiden kilpailijoiden tietokantapalvelinohjelmistoille. Sen uusin versio on 2008. Käytännössä kaikki tieto Call Wavesilla on tallennettu SQL Servereillä sijaitseviin tietokantoihin. Näitä tietokantoja käytetään apuna soittojärjestelmän siivouksessa. Tässä projektissa käytettävät tietokannat ovat SQL Server 2008 -ympäristössä.

Microsoft SQL Server -tietokannat ovat relaatiomallin mukaisia. Relaatiomalli on tietokantamalleista yksinkertaisin ja lisäksi se on hyvin joustava. Relaatiomallin mukaisessa tietokannassa tiedot ovat tauluissa jotka ovat toisiinsa relaatiossa viiteavaimen avulla. Taulujen rivejä kutsutaan tietueiksi. Jokaisella taulun rivillä on yhtä monta saraketta. Relaatiomallissa tietoja haetaan tauluista niiden oikeiden arvojen perusteella, eikä niiden sijainnin perusteella. Jokaisella taulun rivillä on yksikäsitteinen perusavainkenttä, joka useimmiten on juoksevalla numeroinnilla varustettu numerokenttä. Perusavaimen avulla jokainen taulun sisältämä rivi on uniikki. Relaatiomallin mukaisesta taulusta voidaan hakea tietoa ainakin ilmoittamalla taulun nimi, perusavaimen arvo ja haettavan tiedon sisältävän kentän nimi. Relaatiomallin mukaisesta taulusta voidaan hakea tietoa myös kentän sisältämän arvon mukaan, esimerkiksi päivämää-

rät joltain tietyltä aikaväliltä. Relaatiomallin huono puoli on sen vaatima laskentateho, mutta nykyaikaisilla palvelimilla se ei ole kovin suuri ongelma. (Relaatiotietokannat. Informaatioteknologia, Jyväskylän yliopiston IT-tiedekunta ja avoin yliopisto)

4 Ohjelmiston kehitysprosessi

Ohjelmiston kehityksessä voidaan soveltaa monenlaisia eri malleja ja menetelmiä. Yleisesti ottaen ohjelmiston kehitysprosessi voidaan jakaa viiteen osaan, joita sitten sovelletaan eri tavoin. Nämä viisi osaa ovat:

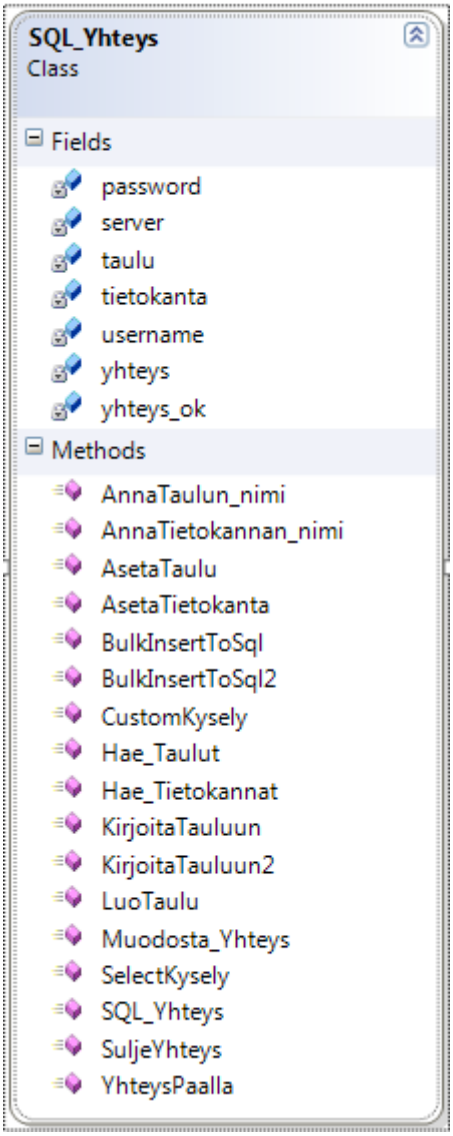
- Vaatimusten määrittely
- Ohjelmiston suunnittelu
- Totetutus
- Testaus
- Käyttöönotto, ylläpito

Tässä dokumentissa ei keskitytä näiden kaikkien vaiheiden määrittelyyn, vaan tarkoitus on keskittyä lähinnä ohjelmiston toteutuksessa käytettyihin eri menetelmiin. Sovelluksen vaatimuksista kerrotaan lisää seuraavassa luvussa. (Kinnunen Jukka, 2004. Ohjelmistojen laatu ja testaaminen.)

Nykyaikaista ohjelmointitapaa mukaillen tämän projektin sovellus on kehitetty käyttäen olio-ohjelmoinnin näkökulmaa. Olio-ohjelmoinnissa pyritään mallintamaan todellisen asiat niin sanotuiksi olioiksi. Olio-ohjelmointi helpottaa koodin ylläpitämistä ja luettavuutta, koska ohjelmakoodi on tavallaan jaettu pienempiin palasiin, eli olioihin. C# on puhdas oliokieli.

4.1 Olio-ohjelmointi

Ohjelma on jatkuvasti yhteydessä tietokantaan, joten tietokantayhteyttä varten tarvittiin luokka. Tällaisen luokan kehittäminen vähentää ohjelmointityötä myöhemmin huomattavasti, koska tietokantayhteyttä varten ei tarvitse joka kerta kirjoittaa samoja koodirivejä uudelleen. Olio-ohjelmoinnilla on myös omat huonot puolensa. Luokassa oleva virhe vaikuttaa kaikkiin luokkaa käyttäviin palveluihin. Niin sanotussa perinteisessä ohjelmoinnissa virheen vaikutus on paikallinen. Oikeaoppinen olio-ohjelmointi vaatii laajaa analysointia ennen ohjelman kehityksen aloittamista. Tällaisessa analyysissa pyritään määrittelemään tarvittavat oliot ja kuka niitä tarvitsee. Analysoinnin tärkeys moninkertaistuu kehitettävän ohjelman kasvaessa. Näin ohjelmiston kehittämisen aloittaminen on helpompaa, kun olioiden väliset suhteet ovat selvillä.



KUVA 2. SQL_Yhteys luokkakaavio

Oheisessa kaaviossa on esitetty SQL_Yhteys-luokan metodit ja kentät. Luokan tärkeimmät metodit ovat yhteyden muodostus (Muodosta_Yhteys) ja vapaamuotoisen kyselyn kirjoitus (CustomKysely). Tällaisessa ratkaisussa on helppoa kirjoittaa luokalle lisää metodeja ja muita ominaisuuksia, eivätkä ne (välttämättä) vaikuta jo olemassaolevaan ohjelmakoodiin.

SQL_Yhteys-oliota kutsutaan ohjelmakoodissa näin:

```
SQL_Yhteys sqlConn = new SQL_Yhteys(username,password,server);
```

Oliolle välitetään tekstimuuttujissa tieto serveristä, johon yhdistetään sekä käyttäjätunnuksista.

4.1.1 Tietokantayhteyden muodostaminen SQL Server-tietokantaan

Tietokantayhteyden muodostamiseen SQL Server -tietokantaan on monta eri tapaa. Helpoin tapa lienee yhdistää tietokantaan Visual Studion tarjoamalla Wizard-tyyppisellä yhdistämissovelluksella. Tässä sovelluksessa käytetään hieman manuaalisempaa tapaa, jotta yhteyden hallitseminen olisi selkeämpää ja helpompaa. Käytännössä SQL Server -tietokantaan yhdistämiseen tarvitaan kaksi objektia; perinteinen string-tietotyyppiä oleva Connection String ja SqlConnection-luokan olio jolle Connection String annetaan parametrina.

Connection String määrittelee serverin johon yhdistetään ja lisäksi käytettävän turvallisuuskontekstin. Turvallisuuskontekstina voidaan käyttää esimerkiksi Windows-tunnuksia tai sitten haluttu tunnus voidaan antaa suoraan Connection Stringin mukana. Yhteys tietokantaan voidaan muodostaa esimerkiksi näin:

```
using System.Data.SqlClient;

SqlConnection           conn           = new SqlDbConnection();
conn.ConnectionString                                          =
              "Data            Source=ServerName;"             +
              "Initial         Catalog=DataBaseName;"          +
              "User               id=UserName;"                +
              "Password=Secret;";
conn.Open();
```

SQL_Yhteys-luokassa edellä esitelty koodi suoritetaan aina, kun luodaan uusi SQL_Yhteys-olio. Tämä vähentää jo huomattavasti kirjoitettavan koodin ja työn määrää.

5 Vaatimukset

Kaikenkattavaa soittolistojen hallintasovellusta olisi näillä resursseilla mahdotonta kehittää. Näin ollen sovellus päätettiin rajata neljään eri osaan, joista tärkein on soittojärjestelmän siivous. Muut ovat:

- Soittolistojen luonti
- Soittolistan puhelinnumeroiden puhdistus
- Soittolistojen tilanne

Käsiteltävät datamäärät ovat suuria, joten se asettaa ohjelmakoodin suorituskyvylle tiettyjä vaatimuksia.

5.1 Soittojärjestelmän siivous

Asiakkailta tulevat soittolistat muunnetaan eräällä työkalulla raakaan tekstimuotoon, jollaisena lista viedään soittojärjestelmään. Kaikki soittojärjestelmän sisälle viedyt listat muodostavat yhden ison listan, joka ajan myötä sitten kasvaa niin suureksi, että soittojärjestelmän toiminta alkaa hidastua. Kriittinen raja alkaa tulla vastaan noin 300 000 – 400 000 puhelinnumeron paikkeilla. Ongelmana on, että järjestelmä ei sellaisenaan osaa poistaa jo soitettuja ja tarpeettomia puhelinnumeroita listalta. Tällöin soitettavien puhelinnumeroiden haku alkaa vähitellen hidastua ja tehokkuus heikkenee.

Siivoaminen itsessään oli aikaisemmin hankalaa ja kallista työtä, koska siivous jouduttiin suorittamaan kokonaan käsin. Soittojärjestelmästä haettiin tieto soitetuista numeroista, joita sitten verrattiin Excelin avulla alkuperäiseen soittolistaan. Tämä aiheutti useimmiten suuren määrän virheitä, jolloin soittoon menikin vahingossa jo soitettuja numeroita. Lisäksi tällainen siivous työllisti useampaa työntekijää pahimmillaan monen työpäivän ajan, virheiden takia puhuttiin jopa työviikoista. Oli selvää, että tähän oli saatava muutos.

5.2 Soittolistojen luonti

Toinen suurempi kokonaisuus on soittolistojen luonti, mihin päätettiin yrittää kehittää jonkinlaista aputyökalua. Ongelmana soittolistojen luomisessa on se, että ne eivät asiakkaalta tullessaan noudata koskaan mitään sovittua muotoa. Käytännössä listat ovat useimmiten Excel-muodossa, mihin yhteneväisyydet sitten loppuvatkin. Kuitenkin listat on vietävä järjestelmään aina samassa muodossa, jolloin listoille täytyy suorittaa tiettyjä toimenpiteitä, kuten sarakkeiden siirtäminen oikeisiin kohtiin ja uniikin numerotunniste-sarakkeen lisääminen. Nämä (toimenpiteet) tehdään nykyisellään siis käsin ja avuksi tarvittaisiin jonkinlainen työkalu. Ongelman helpottamiseksi oli jo kehitetty joitakin prototyyppejä, joita voitiin ottaa käyttöön soittolistojen hallintatyökaluun.

5.3 Soittolistan puhelinnumeroiden siivous

Luonnollisesti soittojärjestelmän virheetön toiminta vaatii, että puhelinnumerot ovat oikeassa muodossa. Jos soittolistan puhelinnumerot sisältävät tiettyjä ei-numeraalisia merkkejä, soittolistan vienti järjestelmään epäonnistuu. Järjestelmä ei osaa siivota itse puhelinnumeroita. Jotkut asiakkaat saattavat toimittaa listoja, jotka on tulostettu suoraan esimerkiksi heidän web-palvelunsa postituslistoilta. Käytännössä tämä tarkoittaa sitä, että käyttäjät ovat syöttäneet puhelinnumeronsa itse, jolloin ne voivat sisältää paljon ei-numeraalisia merkkejä. Tämä aiheuttaa ongelmia ja listan luonnin viivästymistä Call Wavesin päässä. Tätä varten tarvitaan työkalu, joka siivoaa puhelinnumerot ylimääräisistä merkeistä.

5.4 Soittolistojen tilanne

Soittojärjestelmä tallentaa soitetuista puhelinnumeroista paljon erilaista tietoa. Näihin tietoihin ei tätä ennen ollut helposti pääsyä. Call Wavesin raportointijärjestelmän kautta saadaan tällä hetkellä tieto agenttien merkkaamista kontaktien päätöskoodeista. Jos asiakas ei jostain syystä vastaa tai linjalla tapahtuu jokin muu virhe, agentti ei pääse merkkaamaan tällaisessa tapauksessa asiakkaalle mitään päätöskoodia. Tällöin soittojärjestelmä merkitsee itse kyseiselle asiakkaalle jonkin koodin, kuten *väärä numero*, *linja katkesi* ja niin edelleen. Nämä koodit tallentuvat soittojärjestelmän omaan tietokantaan, jonka tietoihin ei tällä hetkellä päästä helpolla käsiksi.

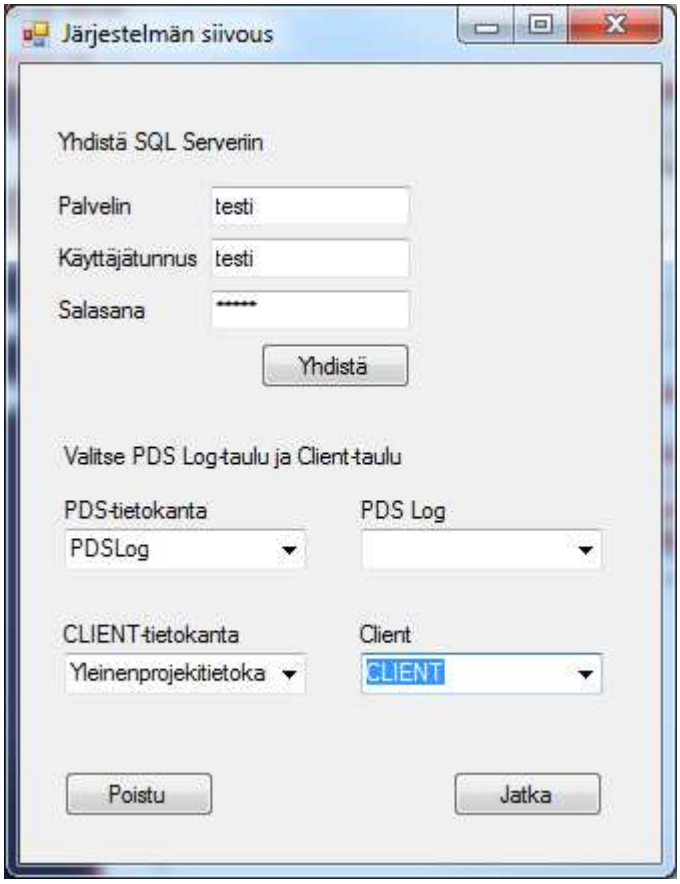
6 Kehitys

Sovelluksen kehitys aloitettiin yksinkertaisesta alkuvalikosta, josta päästään käsiksi haluttuun toimintoon. Jokaisesta napista käynnistyy oma itsenäinen lomakkeensa, joista pääsee etenemään toiminnoissa eteenpäin.

KUVA 3. Päävalikko

6.1 Soittojärjestelmän siivous-osan kehitys

Soittojärjestelmän siivoukseen päästään sovelluksen päävalikosta valitsemalla *Järjestelmän siivous*.



KUVA 3. Järjestelmän siivous

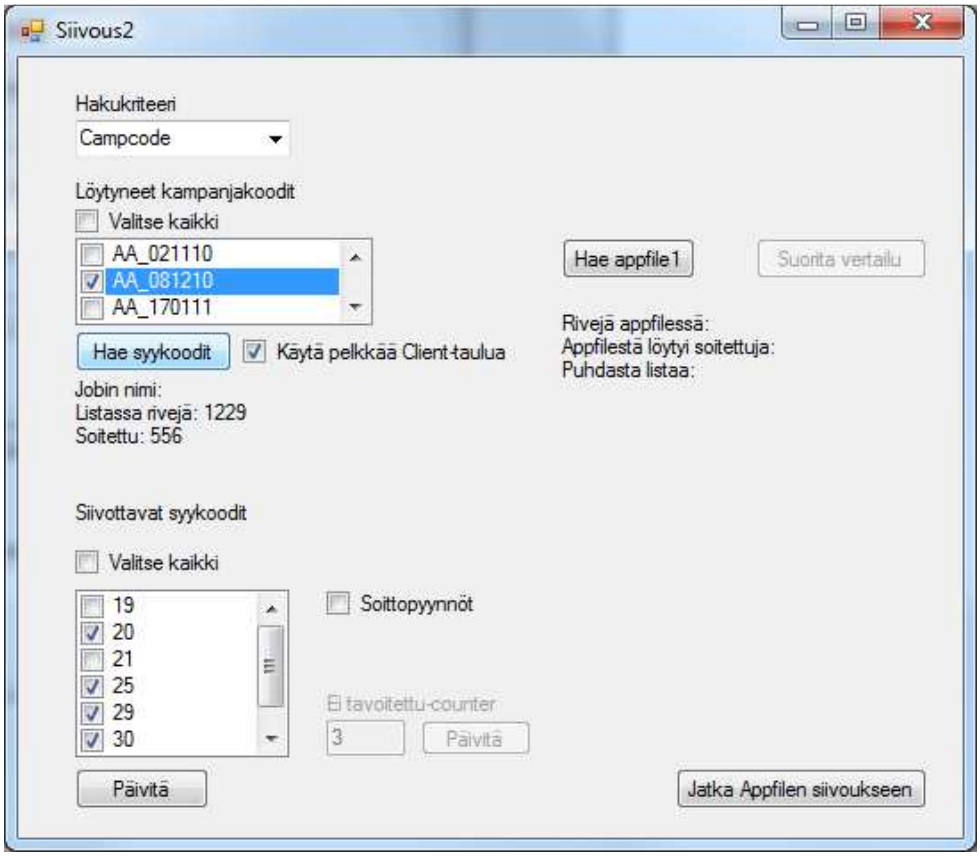
Lomakkeelle syötetään palvelin, käyttäjätunnus ja salasana. Kun yhteys on muodostettu onnistuneesti, haetaan pudotusvalikkoihin PDS-tietokanta ja CLIENT-tietokanta lista palvelimella olevista tietokannoista. PDS-tietokanta sisältää kaikki puhelinjärjestelmässä olevat puhelinnumerot, ja tiedon siitä, onko niihin jo soitettu. CLIENT-tietokanta sisältää kaiken soittolistalla olevan tiedon, kuten asiakkaan nimen, osoitteen ja niin edelleen. Kaikkeen liikenteeseen palvelimen ja sovelluksen välillä käytetään itse tehtyä SQL_Yhteys-luokkaa. SQL_Yhteys-luokalla on metodi, joka palauttaa kaikki palvelimelta löytyvät tietokannat List-tyyppisenä listana. Metodi näyttää seuraavanlaiselta:

```
public List<string> Hae_Tietokannat()
{
      /////tämä metodi hakee annetun sql-palvelimen tietokannat
      //string-muotoinen lista tietokantojen nimille
      List <string> tietokanta_lista = new List<string>();

      //tarkistetaan ensin, että onko yhteys olemassa
```

```
if (yhteys_ok == true)
{
      //jos yhteys on päällä:

      //luodaan sql komento-olio
       System.Data.SqlClient.SqlCommand     komento     =     new     Sys-
       tem.Data.SqlClient.SqlCommand();

      //annetaan sille yhteydeksi aikaisemmin luotu yhteys
      komento.Connection = yhteys;
      komento.CommandType = CommandType.StoredProcedure;
         komento.CommandText = "sp_databases";

         System.Data.SqlClient.SqlDataReader SqlDR;
         SqlDR = komento.ExecuteReader();

         while (SqlDR.Read())
         {
             tietokanta_lista.Add(SqlDR.GetString(0));
         }

         SqlDR.Close();
         SqlDR.Dispose();

     }
     return tietokanta_lista;
 }
```

Metodi käyttää SQL Serverin valmista sp_databases-funktiota (Stored procedure) joka palauttaa tietokantojen nimet. Tietokantojen nimet lisätään while-silmukassa palautettavaan listaan. Kun lista tietokannoista on saatu haettua, saadaan listan sisältö lisättyä pudotusvalikkoon kätevästi käyttämällä Combobox-luokan Items-kokoelman Add-metodia:

```
//käydään lista läpi for eachilla ja syötetään tiedot comboboksiin Add
metodilla:

foreach (string s in tietokanta_lista)
{
                  combo_dblista1.Items.Add(s);
}
```



KUVA 4. Soittolistan valinta

Tässä näkymässä valitaan siivottava soittolista. Käytännössä listan siivoaminen tehdään vertailemalla soitettuja asiakkaita alkuperäiseen soittojärjestelmään vietyyn soittolistaan. Soittojärjestelmään vietyä listaa kutsutaan niin sanotuksi Appfileksi. Kyseessä on tekstimuotoinen tiedosto, joka sisältää jokaiselle puhelinnumerolle uniikin id-numeron, puhelinnumeron, kampanjakoodin ja postinumeron. Soittojärjestelmään ei siis viedä kaikkea tietoa asiakkaasta, vaan loput tiedot haetaan em. id-numeron perusteella tietokannasta. Tietokantaan tallentuu tieto siitä, onko asiakas kontaktoitu vai ei. Tämän perusteella verrataan tietokannasta löytyviä kontaktoituja id-numeroita Appfilestä löytyviin id-numeroihin.

Tietokannassa on luonnollisesti paljon soittolistoja vuosien varrelta. Kokonainen soittolista erotellaan muista niin sanotun kampanjakoodin (Campcode) perusteella. Tämän vuoksi ohjelmaan on sisällytetty mahdollisuus valita tietokannasta tietty lista, mihin Appfileä sitten verrataan. Tämä kampanjakoodilista haetaan lomakkeen käynnistyessä. Joissakin tapauksissa helpointa on vain valita kaikki listat tietokannasta, jonka vuoksi myös valintamahdollisuuden poistamista on harkittu käytön selkeyttämiseksi. Yksittäisen listan valinnasta on kuitenkin hyötyä, koska tällaisissa tapauksissa

vertailu voi kestää hieman kauemmin. Kun käyttäjä on valinnut halutut listat ja painanut *Hae syykoodit* -nappia, haetaan lista syykoodeista, joita kontaktoiduille numeroille on merkitty. Haku suoritetaan HaeSyykoodit()-funktiolla, joka täyttää syykoodi-valikon löytyneillä koodeilla. Haku on yksinkertainen tietokantakysely:

```
//haetaan syykoodit client-taulusta
string query = "";

//kasataan query:
if (valitut_campcodet.Count > 0)
{

                int j, rows = valitut_campcodet.Count;
        query   =   "SELECT   LKTL_COMPLETECODE   FROM   "   +   kampan-
        ja_conn.AnnaTaulun_nimi() + " WHERE " + kriteeri + " in (";

        for (j = 0; j < rows; j++)
                {
                query = query + "'" + valitut_campcodet[j] + "',";
      }

        query = query.Remove(query.Length - 1, 1);
        query  =  query  +  ") AND LKTL_COMPLETECODE IS NOT NULL GROUP BY
        LKTL_COMPLETECODE";
}

syykoodit = kampanja_conn.CustomKysely(query);
```

Kysely kasataan for-silmukan avulla. Käytännössä koodissa käydään läpi käyttäjän valitsemat kampanjakoodit, ja lisätään ne String-muotoiseen tietokantakyselyyn. Lopullinen kysely siis näyttäisi em. tapauksessa suunnilleen tältä:

SELECT LKTL_COMPLETECODE FROM TESTITAULU WHERE CAMPCODE IN ('testikoodi1','testikoodi2') AND LKTL_COMPLETECODE IS NOT NULL GROUP BY LKTL_COMPLETECODE

Kysely annetaan SQL_Yhteys-luokan metodille CustomKysely, joka palauttaa kyselyn tulokset Datatable-oliossa. CustomKysely-metodi on kätevä ja tehokas tapa hakea tietokannasta tietoa vapaamuotoisella kyselyllä. Seuraavaksi voidaan tarkastella

hieman metodin ohjelmakoodia, koska se on laajalti käytössä lähes jokaisessa sovelluksen vaiheessa.

```
//Metodi custom-kyselyitä varten, metodille annetaan query kokonaisuudessaan string muuttujassa
//Query suoritetaan automaattisesti, ja metodi palauttaa tulokset datatablena

public DataTable CustomKysely(string q)
{
      //ensin muodostetaan datatable, johon kyselyn tulokset tallennetaan
      DataTable Results = new DataTable();
      string query = "";

     //määritellään kyselylle tietokanta, jota käytetään
     query = query + "USE " + tietokanta;
     query = query + " " + q;

     bool query_onnistui = true;

     //yritetään hakea dataa
     try
     {
              System.Data.SqlClient.SqlCommand komento = new System.Data.SqlClient.SqlCommand();
                  komento.Connection = yhteys;
                  komento.CommandText = query;

                  Results.Load(komento.ExecuteReader());

                  komento.Dispose();
     }
     catch
     {
         query_onnistui = false;
     }


     if (query_onnistui == true)
     {
         //palautetaan datatable
```

```
            return Results;
        }
        else
        {
            //jos haku epäonnistui, ilmoitetaan käyttäjälle
            MessageBox.Show("Tietojen haku epäonnistui.");
            return Results;
        }
    }
```

Metodille annetaan alkuparametrina tietokantakysely String-muodossa. Käyttäjän on itse huolehdittava, että kysely on validi. Metodi lisää kyselyyn automaattisesti käytettävän tietokannan, joka on määritelty SQL_Yhteys-olio luotaessa. Tämän jälkeen yritetään hakea tietoa tietokannasta Try-Catch virheenkäsittelyn avulla varmistetulla haulla. Haku suoritetaan käyttämällä .NET-ympäristön valmista SqlCommand-luokkaa. Luokalle annetaan parametreina aikaisemmin luotu yhteys ja käyttäjän kirjoittama kysely. Kyselyn palauttamat tiedot saadaan tallennettua Datatableen kutsumalla Datatable-luokan Load() metodia, jonka sisällä kutsutaan SqlCommand-luokan ExecuteReader()-metodia. Tietojen lataus tapahtuu täysin automaattisesti, joten minkäänlaista tietojen kirjoittamista rivi riviltä muistiin ei tarvita. Jos haku onnistui, niin palautetaan Datatable käyttäjälle. Epäonnistumisesta ilmoitetaan virheilmoituksella.

Nyt kun lista syykoodeista on saatu Datatable-tyyppisenä tauluna, voidaan se ladata käyttäjän näkyville valikkoon. Lataamiseen käytetään normaalia for-silmukkaa.

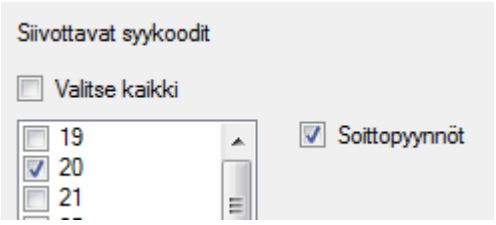
```
//lisätään syykoodit (jos löytyi)
for (i = 0; i < rivilkm; i++)
{

     checked_syykoodit.Items.Add(syykoodit.Rows[i][0].ToString());
}
```

Lisäys on lähes identtinen aikaisemmin esitellyn tietokantalistan kokoamisen kanssa. Datatable käydään for-silmukan avulla läpi ja sen sisältämät tiedot lisätään Add()-metodilla listaan.

Jotta ohjelman käyttö olisi sujuvampaa, ohjelma valitsee automaattisesti tyypillisimmät koodit jotka siivotaan soittolistalta pois. Yleisimmät koodit, joita ei siivota, ovat

soittopyynnöt ja ei tavoitetuiksi merkityt kontaktit. Käytännössä siis kaikki koodit väliltä 20 ja 30 merkataan siivottaviksi. Sekä kampanjakoodi-listassa ja syykoodi-listassa käytetään CheckedListBox-kontrollia. CheckedListBox-kontrollin sisältämiä vaihtoehtoja voidaan merkitä valituiksi käyttämällä kontrollin SetItemChecked-metodia. Metodille annetaan parametreiksi halutun vaihtoehdon indeksi (alkaa nollasta) ja Boolean-tyyppinen muuttuja. Jos vaihtoehto halutaan valituksi, Boolean-muuttujan arvoksi laitetaan true.

```
        //sitten valitaan vielä valmiiksi yleisimmät poistettavat syykoo-
        dit:
        for (i = 0; i < checked_syykoodit.Items.Count; i++)
        {
                    try
                    {
                    int syykoodi = Con-
                    vert.ToInt32(checked_syykoodit.Items[i].ToString());

                    if (syykoodi >= 20 && syykoodi <= 30 && syykoodi !=
                    21)
                    {
                    checked_syykoodit.SetItemChecked(i, true);
                    }
                    }

         catch (Exception err)
        {
        ////virhe
        }
    }
```

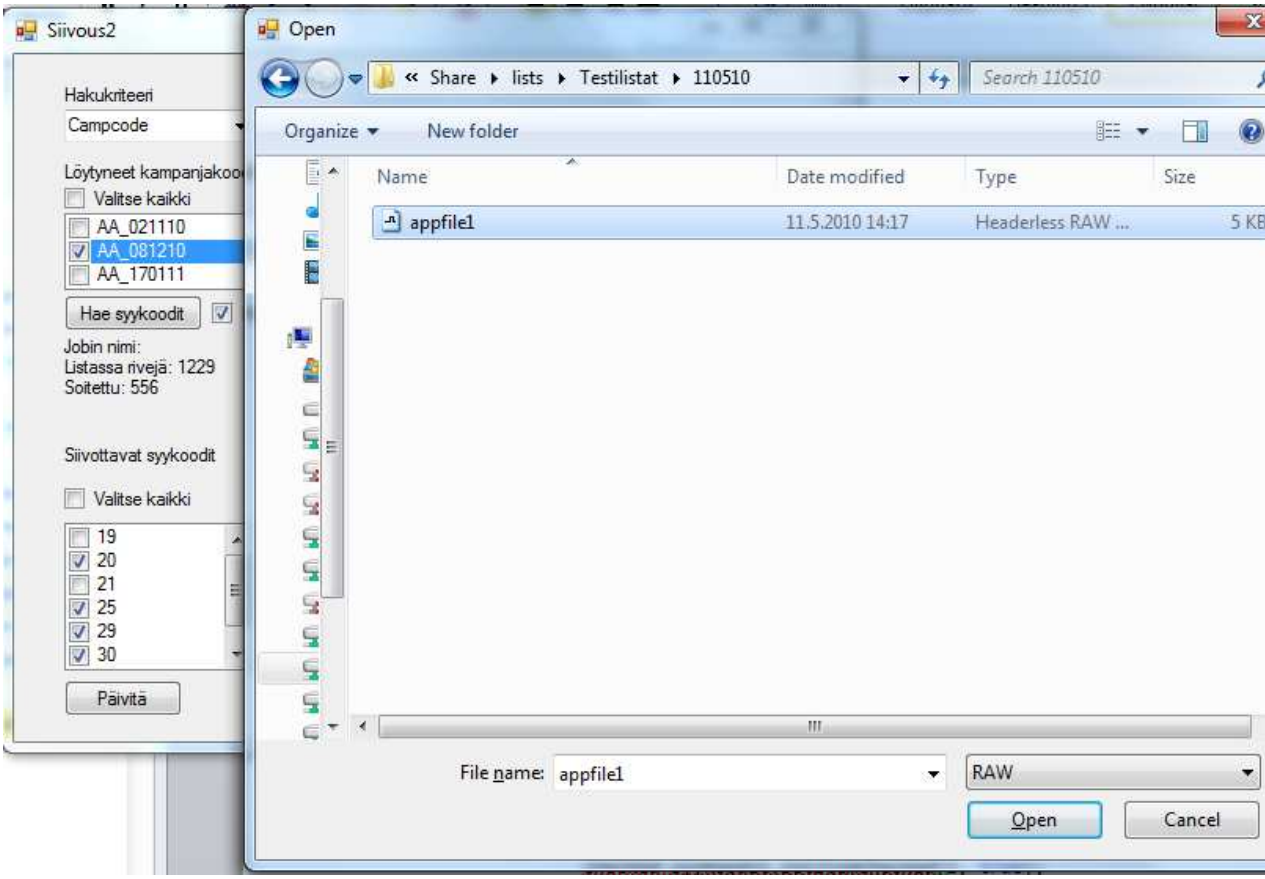


KUVA 5. Soittopyynnöt-valinta

Käyttäjä voi toimia soittopyyntöjen kanssa kahdella eri tavalla. Jos soittopyynnöt halutaan normaalisti soittoon niin sanotusti puhtaana, käyttäjän pitää jättää Soittopyynnöt-

valinta valitsematta. Tällöin soittopyynnöt tulevat normaalisti muun listan seassa. Käyttäjä voi myös valita Soittopyynnöt-valinnan, jolloin soittopyyntöihin merkitty soittopäivä lisätään siivottavaan listaan. Tällöin soittojäjestelmä pyrkii soittamaan asiakkaalle merkittynä päivänä.

Seuraava vaihe listan siivouksessa on oikean Appfilen avaaminen. Oikean Appfilen valitseminen on käyttäjän vastuulla.



KUVA 6. Appfilen valitseminen

Tiedoston valitseminen suoritetaan normaalilla tiedoston avaus-dialogilla, joka on tuttu lukuisista Windows-ohjelmista. Dialogi näytetään, kun käyttäjä klikkaa Hae appfile1-nappia.

```
//Haetaan appfile1

polku = "";

//luodaan OpenFileDialog luokasta tavallinen browse file dialogi
OpenFileDialog fd = new OpenFileDialog();
```

```
    //filteroidaan näytettävät tiedostot:
    fd.Filter = "RAW|*.raw";

    //tuodaan dialogi esiin
    fd.ShowDialog();
```

Tiedoston avaus-dialogi saadaan .NET-ympäristössä kutsuttua helposti OpenFile-Dialog-luokkaa käyttämällä. Dialogi luodaan käyttämällä new-komentoa. Dialogille voidaan määritellä näytettävät tiedostomuodot Filter-ominaisuuden avulla. Appfile on raw-muotoinen tekstitiedosto, joten suodattimeksi laitetaan ainoastaan raw-päätteiset tiedostot. Dialogi saadaan käyttäjälle näkyviin ShowDialog()-komennolla.

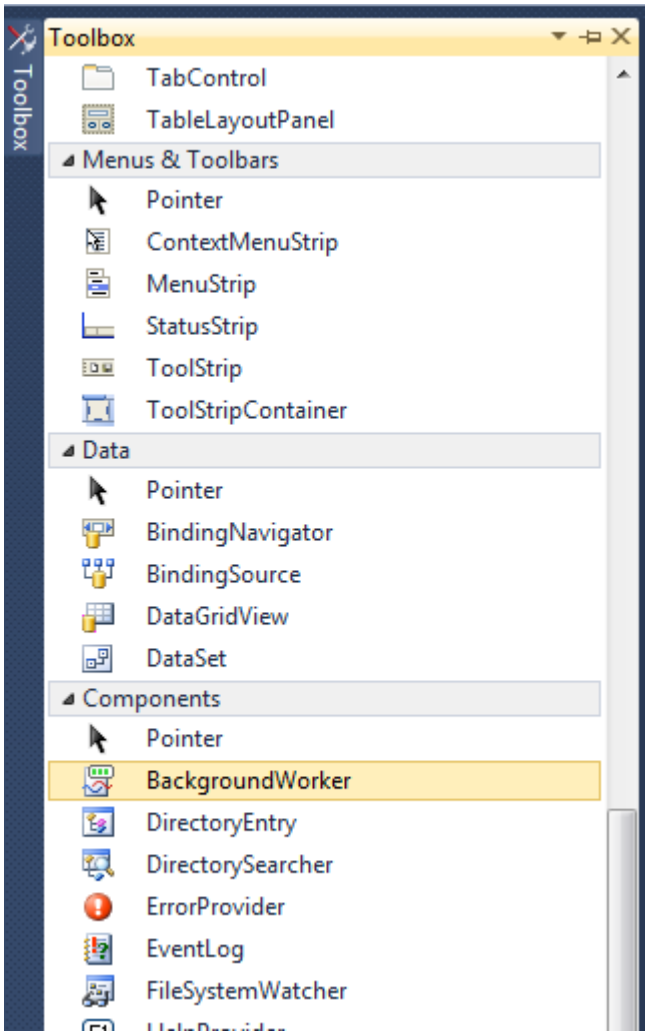
```
  //tarkistetaan, että dialogista palautui tiedostonimi, eli käyttäjä valitsi
  jonkin tiedoston, eikä poistunut cancelilla
  if (fd.FileName != "")
  {
      polku = fd.FileName;
      RawReader raw_file = new RawReader(polku);

      //otetaan raw_fileltä appfile stringlist muodossa:
      list_appfile = raw_file.AnnaLista();

      //lasketaan montako riviä appfilessä oli
      appfile_rivilkm = list_appfile.Count;

     //päivitetään näkymää
            lbl_appfilerivilkm.Text   =   "Rivejä   appfilessä:   "   +   appfi-
le_rivilkm.ToString();

            appfile_haettu = true;
            btn_vertaa.Enabled = true;
  }
```

Jos käyttäjä valitsi jonkin tiedoston, eikä poistunut Cancel-napilla, luetaan tiedoston sisältö String-tyyppiseen List-muuttujaan (List<string>). Tapaus, jossa käyttäjä on poistunut dialogista Cancel-napilla, voidaan tarkistaa helposti. OpenFileDialog-luokka palauttaa valitun tiedoston polun FileName-ominaisuutta kutsuttaessa. Jos käyttäjä poistui dialogista esimerkiksi Cancel-napilla, on FileName-ominaisuuden arvo tyhjä.

```
//Tämä metodi lukee raw tiedoston string listiin
int riviLkm = 0;

//luodaan StreamReader-olio joka lukee raw-tiedoston rivi riviltä
using (StreamReader reader = new StreamReader(fp, Encoding.Default))
{
      string line;
      //luetaan tiedostoa niin pitkään kun readerin palauttama rivi ei
      ole null
      while ((line = reader.ReadLine()) != null)
      {
                  //lisätään rivin sisältö listaan
                  list_raw.Add(line);
                  riviLkm++;
      }
}
```

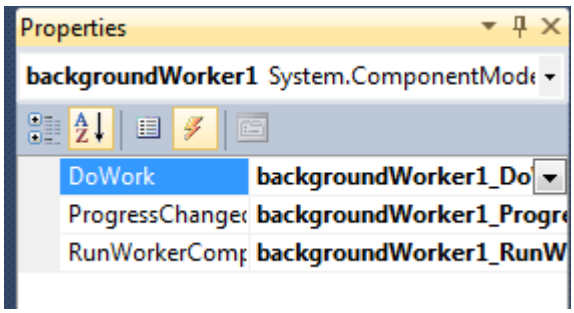
Tiedoston sisällön lukeminen suoritetaan itsetehdyllä RawReader-luokalla. Luokalle annetaan alkuparametrina tiedoston polku, jolloin luokka hoitaa tiedoston lukemisen automaattisesti List<string>-muotoon. Em. ohjelmakoodi on otettu RawReader-luokan tiedoston luku-metodista. Tiedoston lukemiseen käytetään .NET-ympäristöstä löytyvää StreamReader-luokkaa. Tiedosto käydään läpi rivi kerrallaan while-silmukan avulla. Silmukan joka kierroksella kyseinen rivi lisätään aina List<string>-tyyppiseen muuttujaan. Silmukkaa suoritetaan niin pitkään kunnes StreamReaderin palauttama rivi on tyhjä. List<string>-muodossa oleva lista saadaan RawReader-luokalta kutsumalla AnnaLista()-metodia.

Kun käyttäjä on saanut Appfilen valittua, hän voi suorittaa tietokannasta löytyvän listan ja appfilen sisältämien numeroiden vertailun. Ohjelma näyttää tässä vaiheessa käyttäjälle appfilestä löytyneiden soitettujen numeroiden määrän. Listan vertailu voi kestää sen koosta riippuen jopa muutaman minuutin. Tämän vuoksi vertailu suoritetaan omassa prosessissaan, jolloin ohjelma ei jumitu vertailun ajaksi. Tehtävän siirto toisen prosessin suoritettavaksi on helppo toteuttaa .NET-ympäristössä BackgroundWorker-luokan avulla.



KUVA 7. BackgroundWorker työkalulaatikossa

Tässä projektissa BackgroundWorker otettiin käyttöön yksinkertaisesti raahaamalla se Visual Studion työkalulaatikosta lomakkeelle.



KUVA 8. BackgroundWorker-luokan tapahtumankäsittelijät

BackgroundWorker-luokalla on kolme tapahtumankäsittelijää, joita käytetään apuna tehtävän suorittamisessa. Tärkein tapahtumankäsittelijä on DoWork, jonka sisälle tässä projektissa laitettiin em. listojen vertailu.

```
private void backgroundWorker1_DoWork(object sender, DoWorkEven-
tArgs e)
{
            SuoritaVertailu();
}
```

DoWork-tapahtuma kutsuu SuoritaVertailu()-metodia. Tässä vaiheessa koodin suoritus siirtyy BackgroundWorker-luokan suoritettavaksi. BackgroundWorker-luokka huolehtii siitä, että tehtävän suoritus tapahtuu omana prosessinaan, jolloin muu osa ohjelmasta on edelleen käytettävissä.

SuoritaVertailu()-metodin tehtävänä on käytännössä vertailla valittua tietokantaa Appfilen sisältöön, ja ottaa samalla id-numerolla varustetut rivit talteen normaaliin taulukkoon.

```
//otetaan soitettujen datatablesta acctnumit tavalliseen string-
taulukkoon.
//näin sitä on nopeampi käsitellä, kuin viittaamalla datatablessa ole-
vaan acctnumiin:
string[,] soitetut_acctnumit = new string[soitetut_rivilkm, 2];

for (i = 0; i < soitetut_rivilkm; i++)
{
      soitetut_acctnumit[i,          0]          =          soite-
      tut.Rows[i]["ACCTNUM"].ToString();
      soitetut_acctnumit[i,          1]          =          soite-
      tut.Rows[i]["LKTL_RETRYAFTER"].ToString();
}
```

Soitetut kontaktit ovat aikaisemmin tietokannasta haetussa Datatablessa. Vertailun nopeuttamiseksi Datatablesta luetaan id-numero (Soittojärjestelmän ympäristössä ACCTNUM) ja mahdolliset soittopyyntöjen päivämäärät tavalliseen String-tyyppiseen taulukkoon, jossa on kaksi saraketta. Ohjelman testauksessa kävi ilmi, että arvojen vertailu taulukon avulla on huomattavasti nopeampaa kuin suoraan Datatablessa olevaan tietoon viittaaminen. Datatablessa oleva arvo saadaan muuttujaan:
string arvo = dt_esim[rivinumero]["sarakkeen_nimi"].ToString();

Kun arvot on saatu taulukkoon, voidaan aloittaa soitettujen ja alkuperäisen puhtaan listan vertailu.

```
for (i = 0; i < appfile_rivilkm; i++)
```

```
{
      //ekaksi otetaan appfilestä pelkkä acctnum, pituudeltaan se on aina
      19 merkkiä, ja se        on joka rivin alussa.
      //tämä onnistuu kutsumalla sille omistettua funktiota:
      appfile_acctnum = SuodataAcctnum(list_appfile[i]);

      string appfile_backup = appfile_acctnum;
```

Vertailuun käytetään luonnollisesti for-silmukkaa. Ensimmäisenä tarvitaan alkuperäisen listan rivin id-numero (ACCTNUM). Acctnum on aina 19 merkkiä pitkä ja se on Appfilen jokaisen rivin alussa. Koodin selkeyttämiseksi acctnumin saamiseksi rivin alusta käytetään SuodataAcctnum()-funktiota. SuodataAcctnum()-funktiolle annetaan teksti-muotoinen appfile-rivi parametrina, jolloin funktio palauttaa halutun acctnumin. SuodataAcctnum()-funktio näyttää kaikessa yksinkertaisuudessaan tältä:

```
private string SuodataAcctnum(string s)
{
      //tämä funktio suodattaa sille annetusta appfile rivistä acctnumin
      string acctnum = s.Substring(0, 19);
      return acctnum;
}
```

Acctnumin suodattamiseen käytetään String-luokan Substring-funktiota. Substring-funktiolle annetaan parametreina halutun tekstin aloituskohta ja pituus. Tässä tapauksessa halutaan siis tekstin alusta ensimmäiset 19 merkkiä.

Kun acctnum-numero on saatu suodatettua tekstin joukosta, tehdään sille vielä yksi varmistus.

```
appfile_acctnum = appfile_acctnum.Trim();
```

Acctnum määritellään listaa tehdessä manuaalisesti, joten virheen mahdollisuuskin on olemassa. Ohjelma käyttää String-luokan Trim()-funktiota, jolla poistetaan muuttujan lopusta ylimääräiset välilyönnit, jos acctnum onkin ollut pituudeltaan vähemmän kuin 19 merkkiä. Tällöin vältytään virheiltä vertailutilanteessa.

Seuraavaksi ohjelma siirtyy varsinaisen vertailun suorittamiseen.

```
//kun acctnum on otettu, täytyy sitä verrata koko soitettujen datatableen:
            for (j = 0; j < soitetut_rivilkm; j++)
            {
                        //verrataan if-lauseella acctnumeja
                        if (appfile_acctnum == soitetut_acctnumit[j, 0])
                        {
                        //jos acctnumit täsmää, tallennetaan acctnum string-
                        taulukkoon
                        acctnum_soitetut[acctnum_counter, 0] = appfile_backup;

                        if (soitetut_acctnumit[j, 1] != "")
                        {
                        acctnum_soitetut[acctnum_counter,    1]    =    muotoi-
                        lePvm(soitetut_acctnumit[j, 1]);
                        }
                        else
                        {
                        acctnum_soitetut[acctnum_counter,    1]    =    soite-
tut_acctnumit[j, 1];
                        }
                           acctnum_counter++;
                       }
            }
```

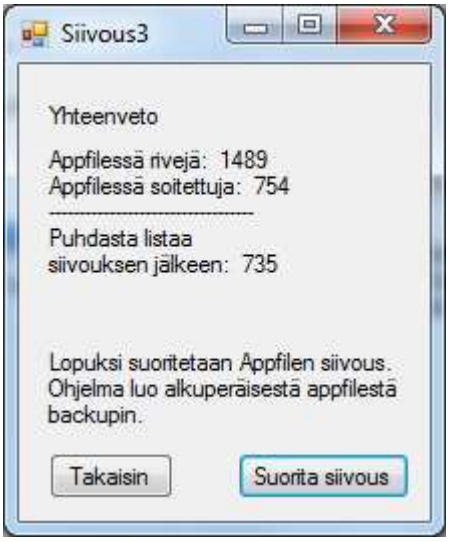
Vertailun lisäksi tässä vaiheessa suoritetaan jo soittopyyntöjen soittopäivämäärien valmistelu, jos ohjelman käyttäjä haluaa ne listalle mukaan. Jos acctnumit täsmäävät, niin kyseinen acctnum lisätään soitettujen acctnumien listalle. Tämä lista on siis käytännössä lista id-numeroista, jotka pitää poistaa alkuperäisestä appfile1-tiedostosta. Samalla vertailun lomassa ohjelma muotoilee soittopyynnön halutun soittopäivämäärän oikeaan muotoon, koska appfile1-tiedostoon vaadittu päivämäärämuoto on erilainen kuin tietokannassa oleva päivämäärämuoto. Muotoiluun käytetään muotoilePvm()-funktiota.

```
      private string muotoilePvm(string pvm)
      {
            //tekee päivämäärästä (muotoa 10.5.2010) 2010/05/10 muotoisen
            string []paivamaara = pvm.Split('.');
```

```
        paivamaara[2] = paivamaara[2].Substring(0, 4); //pelkkä vuo-
        si,skipataan kellonaika kokonaan

                    StringBuilder uusi = new StringBuilder();

                    if (paivamaara[0].Length < 2)
                    {
                    paivamaara[0] = "0" + paivamaara[0];
                    }
                    if (paivamaara[1].Length < 2)
                    {
                                paivamaara[1] = "0" + paivamaara[1];
                    }

                    uusi.Append(paivamaara[2] + "/" + paivamaara[1] + "/"
                    + paivamaara[0]);

                    return uusi.ToString();
    }
```

Päivämäärän sisältävä string-muuttuja pilkotaan ensin osiin Split()-funktiolla, jolle annetaan parametrina piste. Koska tietokannassa olevassa päivämäärässä on myös kellonaika mukana, se pitää siivota pois käyttäen Substring-funktiota. Kellonaika joutuu samaan taulukon alkioon vuoden kanssa. Seuraavaksi tarkastetaan, onko päivä tai kuukausi yksinumeroinen, jolloin sen eteen tarvitaan nolla, esim. 22.3.2011 on 22032011. Lopuksi käytetään StringBuilder-luokkaa tekstin kasaamiseen. StringBuilder-luokka valittiin siksi, koska se on huomattavasti nopeampi tekstimuotoisen datan käsittelyssä kuin normaali String-muuttuja, tilanteissa joissa silmukka pyörii tuhansia kertoja.

Kun listalle on tehty vertailu ja poistettavat acctnumit on löydetty, käyttäjä voi siirtyä varsinaisen appfile1-tiedoston siivoamiseen. Siivouksen voi aloittaa painamalla *Jatka appfilen siivoukseen* -nappia. Napin painalluksen jälkeen avautuu pieni yhteenveto-ruutu.



KUVA 9. Siivouksen loppuyhteenveto

Appfilen siivous tehdään painamalla *Suorita siivous*-nappia. Ennen kuin appfile-tiedostoon tehdään muutoksia, siitä otetaan varmuuskopio kaiken varalta. Varmuuskopiointia varten kutsutaan TeeBackup()-funktiota.

```
private void TeeBackup(string p)
{
        //tekee backupin tiedostopolusta löytyvästä tiedostosta.

        //ensin otetaan tiedostonimi talteen:
        string filename = Path.GetFileName(p);
```

Funktio saa alkuparametrina polun, jossa tiedosto sijaitsee. Esimerkiksi C:\testi\testi_appfile.raw. Tästä tiedostopolusta otetaan tiedoston nimi talteen. Nimi saadaan otettua polusta erilleen .NET-ympäristön Path-luokan funktiolla GetFileName(). GetFileName() palauttaa tiedoston nimen string-muodossa.

```
        //otetaan alkuperäinen polku talteen
        string original_path = p;

        //tehdään backup-polku uudella tiedostonimellä
        //käytetään string-luokan Replace metodia, jolla etsitään vanha fi-
        lename ja                              korvataan se uudella
        string new_path = p;
        new_path = new_path.Replace(filename, "backup_appfile1.raw");
```

Seuraavaksi otetaan vanha tiedostopolku talteen original_path-muuttujaan. Tämän jälkeen voidaan tehdä uusi polku, jonka tiedostonimi korvataan uudella varmuuskopio-nimellä käyttäen String-luokan Replace()-funktiota.

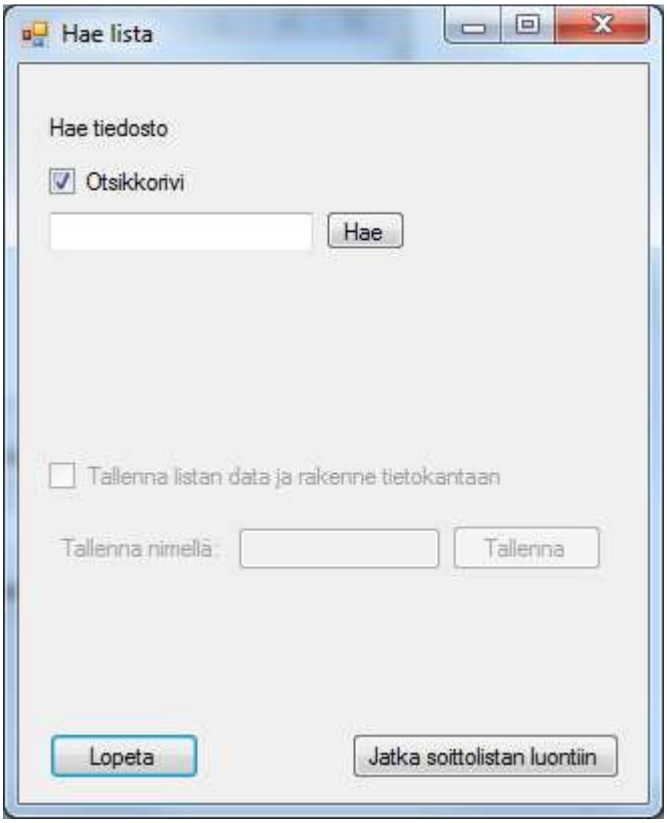
```
            //sitten tehdään varsinainen backup
            //jos backup on jo olemassa, poistetaan se ensin ja kirjoitetaan
            uusi

            if (File.Exists(new_path) == true)
            {
                File.Delete(new_path);
                File.Copy(original_path, new_path);
            }
            else
            {
                File.Copy(original_path, new_path);
            }
    }
```

Ohjelma katsoo aina ennen varmuuskopion tekemistä, onko kansiossa jo olemassa tiedosto samalla nimellä. Edellinen varmuuskopio poistetaan aina uuden alta. Tiedoston olemassaolon toteamiseen käytetään .NET-ympäristön File-luokan Exists()-funktiota. Tiedoston kopioiminen onnistuu helposti em. File-luokan Copy()-funktiolla. Copy()-funktiolle annetaan parametreiksi vanha polku, missä kopioitava tiedosto sijaitsee ja lisäksi uusi polku johon tiedosto kopioidaan uudella nimellä.

6.2 Luo soittolista -toiminnon kehitys

Tässä luvussa käydään lyhyesti läpi Luo soittolista -toiminnon kehitystä. Toimintoa ei koskaan otettu varsinaisesti käyttöön ensimmäisten testien jälkeen. Testeissä huomattiin, että asiakkailta tulevan materiaalin suuri vaihtelevuus aiheuttaa ongelmia, joita ei olisi saatu ratkaistua annetulla aikataululla. Parhaimmillaan yksinkertaiset soittolistat saatiin luotua muutamalla napinpainalluksella, mutta monimutkaisissa tapauksissa oli nopeampaa tehdä soittolista käsin Exceliä käyttäen. Luo soittolista -toimintoon päästään sovelluksen päävalikosta valitsemalla *Luo soittolista*.

Seuraavaksi aukeaa näkymä, jonka avulla valitaan aineisto josta soittolista luodaan. Tiedoston tietokantaan tallentaminen on tässä vaiheessa vielä poistettu käytöstä, koska käyttäjä ei ole valinnut tiedostoa.



KUVA 10. Hae lista

Listan haku -dialogi aukeaa *Hae*-napista painamalla. Haku-dialogi luodaan Windowsin OpenFileDialog-luokkaa käyttämällä. Dialogi-luokalle voidaan määritellä Filter-asetuksen avulla tiedostot, jotka käyttäjän on mahdollista dialogissa nähdä. Tässä tapauksessa käyttäjälle näytetään ainoastaan .xls- ja .csv-tiedostot. ShowDialog-metodilla dialogi näytetään käyttäjälle.

```
//Käyttäjä hakee excel-tiedoston
string filepath = ""; //koko tiedostopolku
OpenFileDialog fd = new OpenFileDialog();

fd.Filter = "Excel files|*.xls|CSV|*.csv";

fd.ShowDialog();
```

Seuraavaksi tarkistetaan, valitsiko käyttäjä jonkin tiedoston. Jos käyttäjä poistui dialogista Cancel-napin avulla, Dialogi-luokan FileName-ominaisuus palauttaa tyhjän arvon. Tämä tarkistetaan if-rakenteen avulla. Jos käyttäjä on valinnut tiedoston, tarkistetaan tiedoston pääte Path-luokan GetExtension-metodilla.

```
//katsotaan tuliko sieltä tiedostoa
if (fd.FileName != "")
{
      filepath = fd.FileName;

      txt_filename.Text = fd.FileName;

       //katotaan onko tiedosto csv vai xls
      string extension = Path.GetExtension(filepath);

      //katsotaan onko otsikkorivi-valinta ruksattu
      bool header;

      if (cb_otsikko.Checked == true)
      {
                  header = true;
      }
      else
      {
                  header = false;
      }
```

Tiedosto jäsennellään DataTable-formaattiin itsetehdyllä Excelparser-luokalla. Luokalle välitetään parametreina tiedostopolku, tiedostopääte ja tieto otsikkorivin olemassaolosta. Luokka palauttaa annetun tiedoston DataTable-formaatissa AnnaDatatable-metodin avulla. Lopuksi otetaan tiedoston tietokantaan tallennusmahdollisuus käyttöön.

```
      //muutetaan tiedosto datatableksi Excelparser luokan avulla:

      Excelparser data = new Excelparser(filepath, extension, header);

      dt_data = data.AnnaDatatable;

      //sitten annetaan käyttäjälle infoa:
```

```
            NaytaInfo(dt_data, filepath);

            tiedosto_haettu = true;
            check_tallennus.Enabled = true;

        }
```

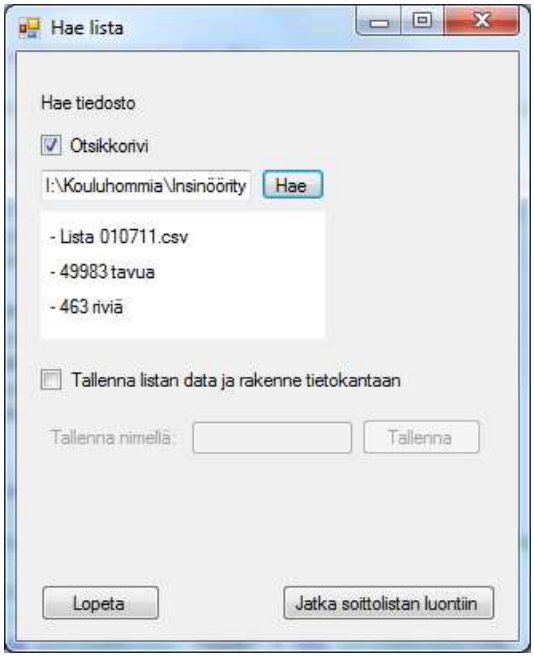
KUVA 11. Tiedoston haku –dialogi

Kun tiedosto on haettu, päivitetään käyttäjän näkymään joitakin tietoja tiedostosta. Tiedoston koko saadaan FileInfo-luokan avulla, jolle annetaan parametrina tiedoston polku.

```
FileInfo fi = new FileInfo(polku);
long len = fi.Length;
```

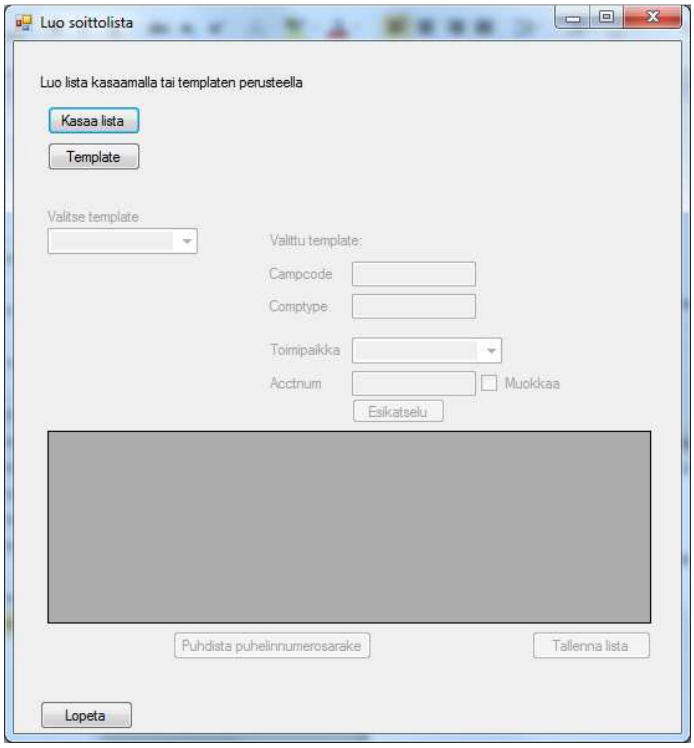
Tiedoston rivien lukumäärä otetaan DataTablesta kutsumalla luokan Rows-kokoelman Count-ominaisuutta, joka palauttaa rivien lukumäärän.

```
int rivilkm = dt.Rows.Count;
```



KUVA 12. Lista haettu.

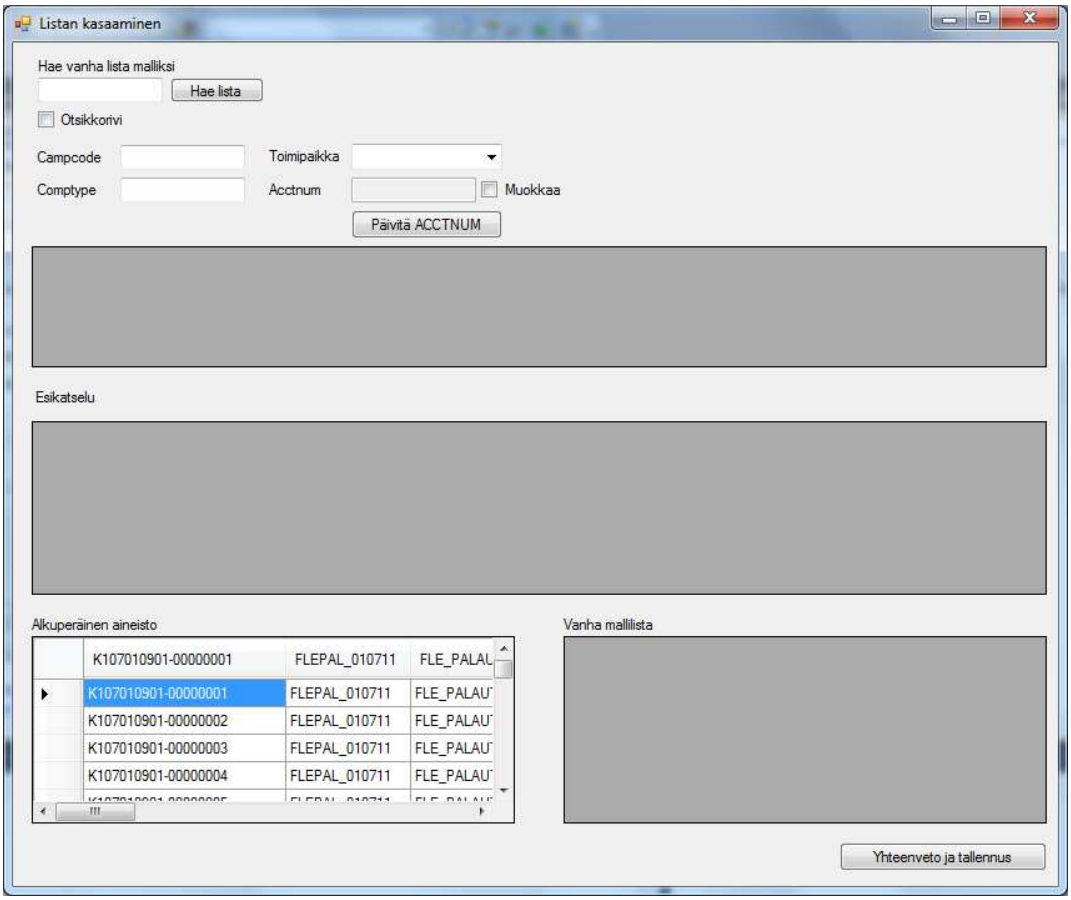
Sitten listan luontia jatketaan painamalla *Jatka soittolistan luontiin* -nappia. Seuraavassa näkymässä käyttäjä valitsee, haluaako luoda listan manuaalisesti, vai käyttää tallennettua mallipohjaa. Käydään ensimmäisenä läpi manuaalinen luonti.



KUVA 13. Listan kasaamistavan valinta.

6.2.1 Soittolistan manuaalinen luominen

Soittolistan luontieditori on seuraavan kuvan mukainen. Ensimmäisenä käyttäjä valitsee editorin avulla vanhemman soittolistan, jota käytetään mallina. Lista haetaan käyttämällä aikaisemmin esiteltyjä OpenFileDialog- ja Excelparser-luokkia.



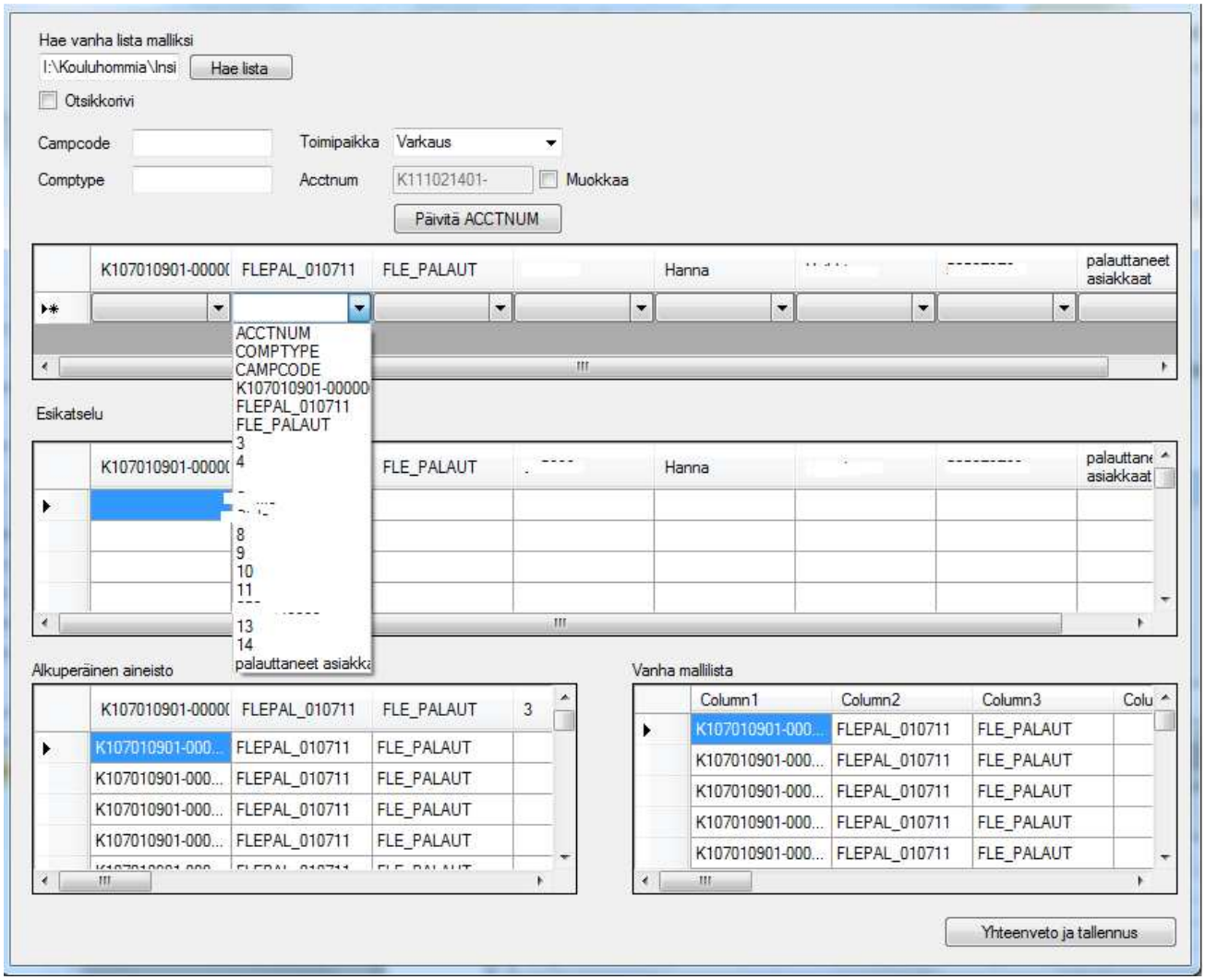
KUVA 14. Soittolistan manuaalinen luominen

Kun käyttäjä poistuu Tiedoston haku-dialogista, haetaan Toimipaikka-pudotusvalikkoon tietokannasta käytettävissä olevat toimipaikat. Hakuun käytetään aikaisemmin esiteltyä itsetehtyä SQL-luokkaa.

```
string query = "Select Toimipaikka from Toimipaikat Group by
Toimipaikka";
```

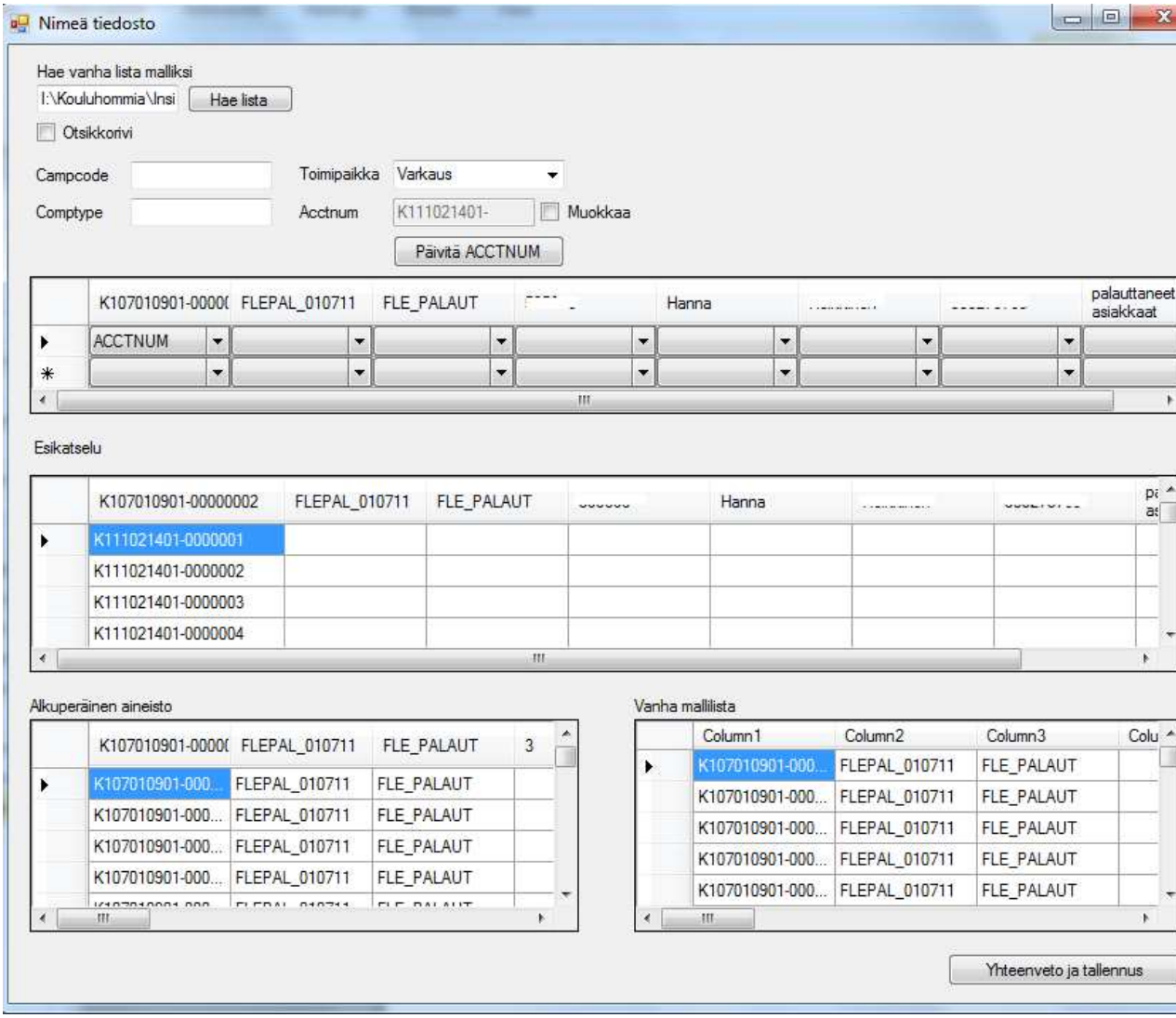
Kun käyttäjä valitsee jonkin toimipaikan, generoidaan sille automaattisesti oikeanlainen jokaiselle soittolistan riville tuleva uniikki toimipaikkakohtainen id-numero.

Listan muokkauksessa käytetään DataGridView-luokkaa, joka toimii kätevästi yhteen DataTablen kanssa, joka on sovelluksen käyttämä käsittelyformaatti Excel-tiedostoissa. Kun käyttäjä on valinnut vanhan mallitiedoston, ladataan kyseisestä tiedostosta ensimmäinen rivi malliksi editorin ylempään DataGridView-näkymään otsikkoriville. Ideana on, että käyttäjä laittaa käsittelyssä olevan soittolista-aineiston sarakkeet oikeille paikoilleen vertaamalla vanhempaan soittolistaan. Tähän käytetään otsikkorivin alapuolella olevia pudotusvalikoita, joissa on soittolista-aineiston sarakkeiden otsikoita.



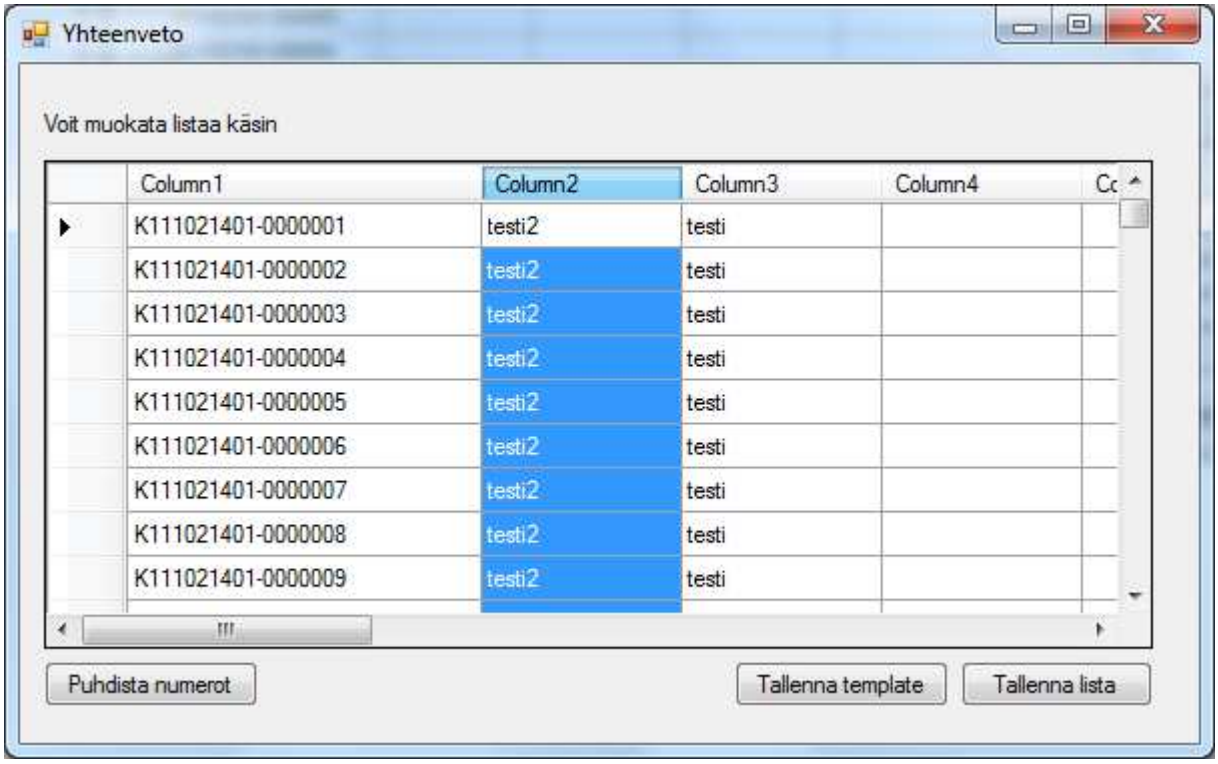
KUVA 15. Sarakkeiden kohdistaminen.

Kun käyttäjä valitsee jonkin sarakkeen, muutos näytetään heti alemmassa esikatselunäkymässä. Näin muokkauksen lopputulos näkyy koko ajan reaaliaikaisena.



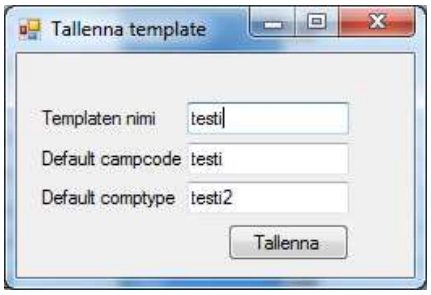
KUVA 16. Esikatselu.

Valmis soittolista saadaan tallennettua *Yhteenveto ja tallennus* -napista. Käyttäjän näkyville tuodaan vielä yhteenveto, jossa näkyy valmis soittolista DataGridView-näkymässä. Tässä vaiheessa käyttäjä pystyy vielä muokkaamaan yksittäisiä soluja kaksoisklikkaamalla haluamaansa solua. Käyttäjä voi myös suorittaa niin sanotun puhelinnumeroiden puhdistuksen, jossa puhelinnumero-sarake puhdistetaan ei-numeraalisesta tiedosta. Tämän funktion toiminnallisuus käydään myöhemmin tarkemmin läpi.



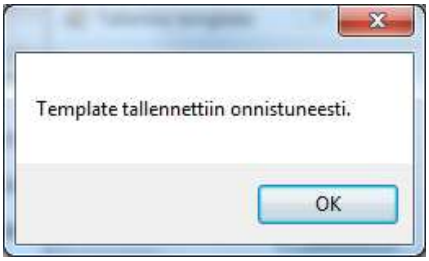
KUVA 17. Yhteenveto.

Tallenna template -napista voidaan tallentaa soittolistasta mallipohja, jota voidaan käyttää seuraavalla kerralla samanlaisen soittolistan luomisessa. *Tallenna template* -näkymässä mallipohjalle annetaan nimi ja määritetään oletusarvot ns. campcode ja comptype tiedoille, joita puhelinjärjestelmä käyttää listan erottelussa. Mallipohjasta tallennetaan tietokantaan nimi, campcode, comptype ja sarakkeiden paikat Excelissä.



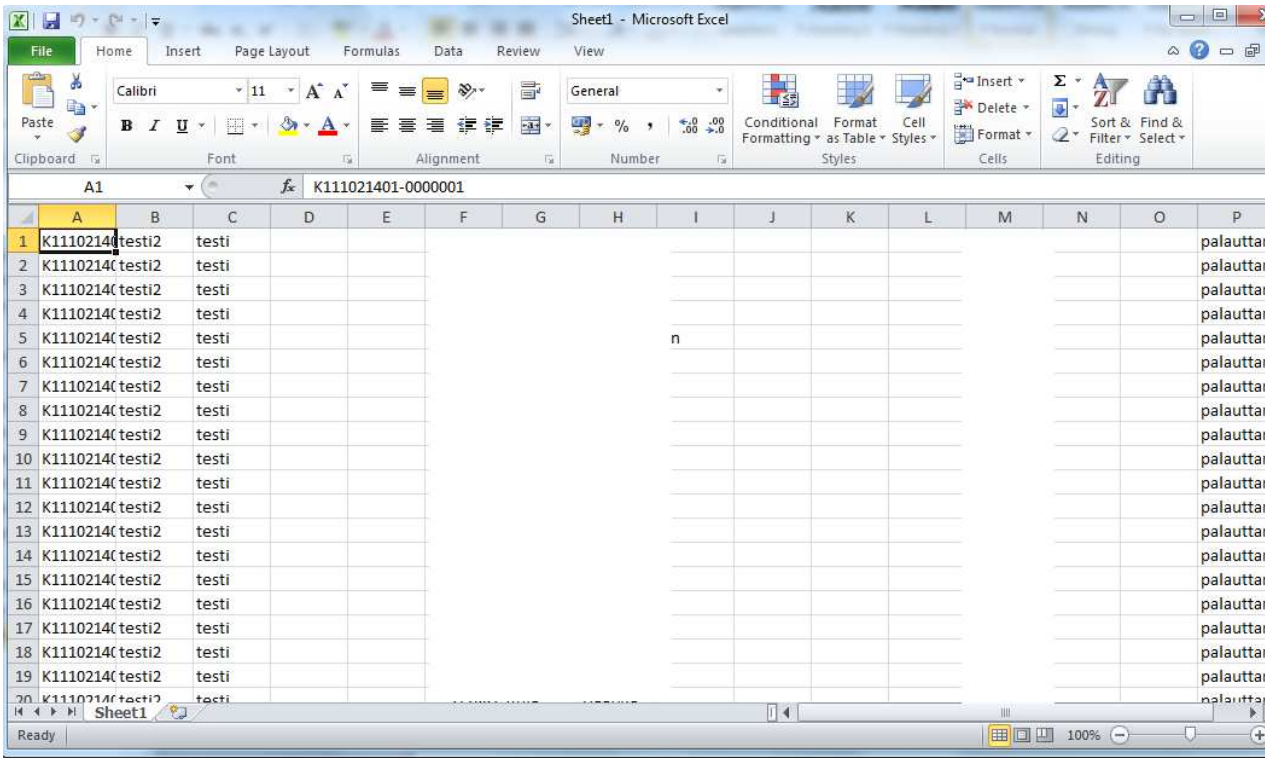
KUVA 18. Mallipohjan tallennus.

Onnistuneesta tallennuksesta ilmoitetaan käyttäjälle kuvan 19 mukaisella ilmoituksella.



KUVA 19. Mallipohjan onnistunut tallennusilmoitus.

Tallenna lista-napista luotu soittolista avataan suoraan Excel-ohjelmaan, josta käyttäjä voi sen itse tallentaa haluamaansa muotoon ja sijaintipaikkaan.



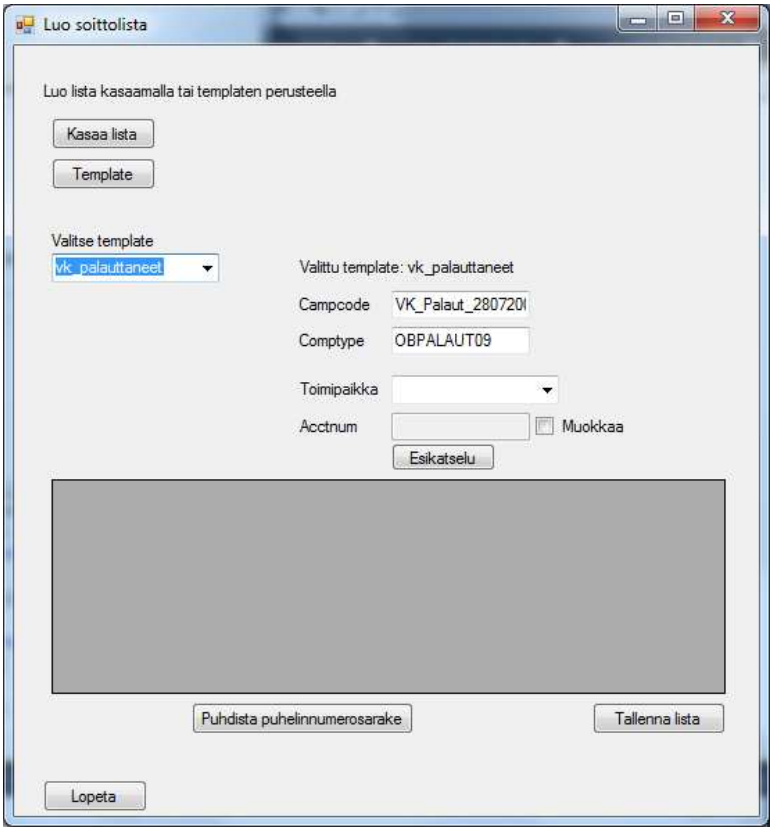
KUVA 20. Valmis lista Excelissä.

DataTable avataan Excelissä muuttamalla se ensin geneeriseksi Object-tyyppiseksi taulukoksi. Muuntamiseen käytetään MuunnaDt-metodia, joka yksinkertaisesti käy DataTablen rivi riviltä läpi ja lukee sen Object-tyyppiseen taulukkoon. Sitten kutsutaan ApplicationClass-luokkaa, josta luodaan Workbook-objekti. Workbook-objektille luodaan Worksheet-objekti, jolle voidaan antaa suoraan arvona äsken luotu Object-taulukko. Tällainen menetelmä on todella nopea, koska Exceliin ei tarvitse kirjoittaa mitään solu kerrallaan. Lopuksi Excel-sovellus tuodaan käyttäjän näkyville asettamalla ApplicationClass-luokan Visible-muuttujalle arvoksi true.

```
Object[,] lista = MuunnaDt(uusi_lista);
//sitten exceli pystyyn

ApplicationClass objExcel = new ApplicationClass();
Workbooks objBooks = objExcel.Workbooks;

System.Globalization.CultureInfo oldCI = Sys-
tem.Threading.Thread.CurrentThread.CurrentCulture;

System.Threading.Thread.CurrentThread.CurrentCulture = new Sys-
tem.Globalization.CultureInfo("en-US");

Workbook objBook = objBooks.Add(XlWBATemplate.xlWBATWorksheet);
Sheets objSheets = objBook.Worksheets;
_Worksheet objSheet = (_Worksheet)objSheets.get_Item(1);
Range objRange = null;
object objOpt = Type.Missing;
string col = GetLastColumnName(uusi_lista.Columns.Count) + uu-
si_lista.Rows.Count;
objRange = objSheet.get_Range("A1", col);
objRange.Value2 = lista;
objExcel.Visible = true;
```

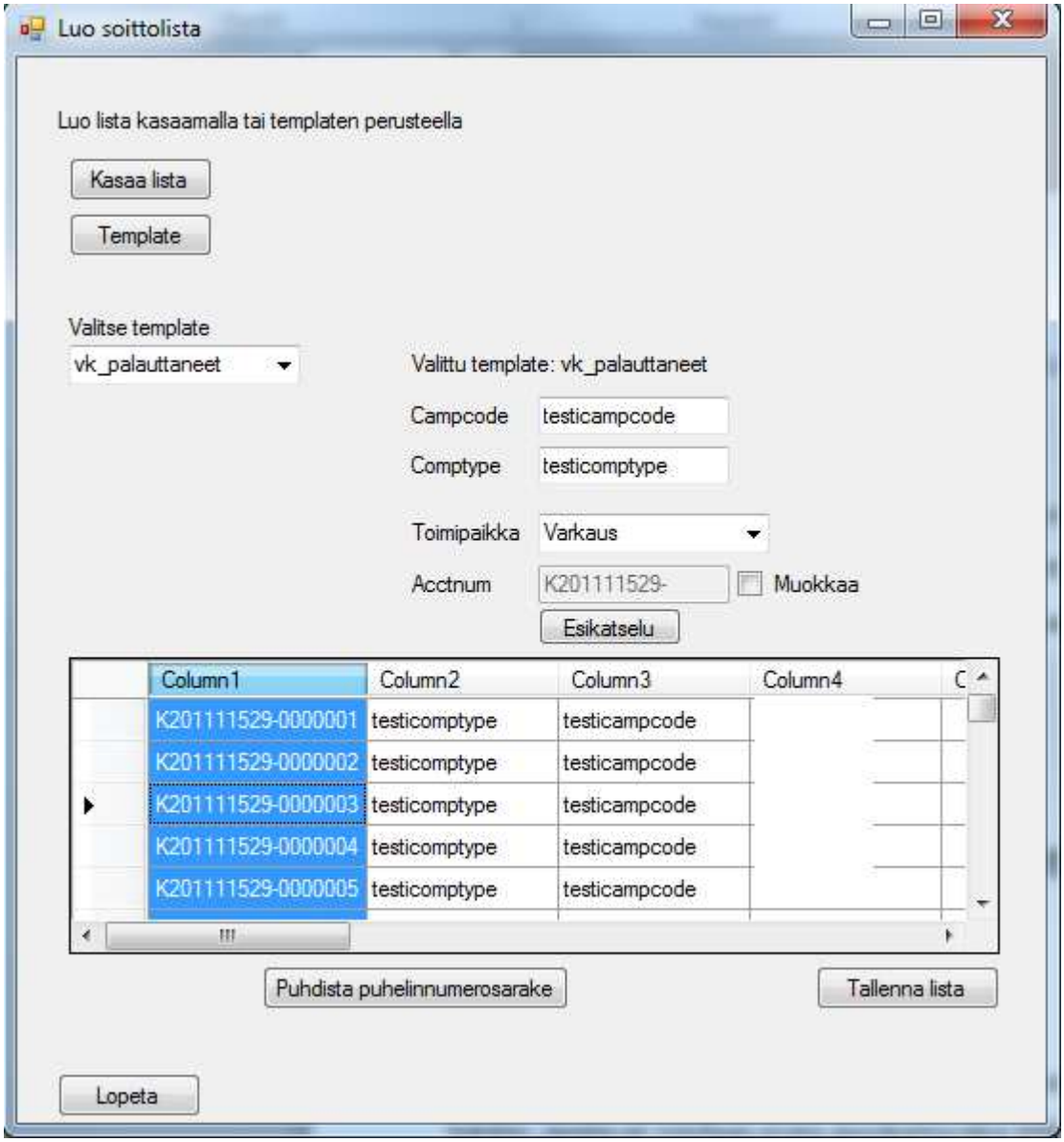
6.2.2 Soittolistan luominen mallipohjan avulla

Soittolista voidaan luoda nopeasti mallipohjan avulla, jos samanlaisella rakenteella tehdystä listasta on aikaisemmin tallennettu mallipohja. Mallipohjan asetukset saadaan näkyviin painamalla *Template*-nappia. Mallipohjat haetaan lomakkeen latautuessa tietokannasta.



KUVA 21. Mallipohjan valinta

Kun käyttäjä valitsee haluamansa mallipohjan, käydään hakemassa tietokannasta kyseisen mallipohjan campcode- ja comptype-tieto. Tämän jälkeen käyttäjä voi muokata kyseisiä tietoja haluamakseen. Tämän jälkeen valitaan toimipaikka samaan tapaan kuin uutta soittolistaa luotaessa, jolloin ohjelma generoi automaattisesti kyseiselle toimipaikalle sopivan acctnum-numeron. Kenttä on oletuksena lukittu, mutta se voidaan avata muokattavaksi klikkaamalla Muokkaa-valintalaatikkoa. Seuraavaksi käyttäjä voi painaa Esikatselu-nappia, jolloin aineistosta luodaan mallipohjan mukainen soittolista, ja se tuodaan esikatselunäkymään näkyville.



KUVA 22. Esikatselunäkymä.

Esikatselussa ohjelma asettelee sarakkeet samalla tavalla kuin aikaisemmin käsin luodussa listassa. Sarakkeiden paikat on tallennettu tietokantaan. Esikatselunäkymässä käyttäjä voi vielä muokata haluamiaan tietoja käsin, ja suorittaa puhelinnumerosarakkeelle puhdistuksen. Puhelinnumerosarakkeen puhdistus on erittäin tarpeellinen ominaisuus, koska asiakkaiden toimittamissa soittolistoissa on usein muitakin merkkejä kuin numeroita. Lopuksi soittolista voidaan tallentaa CSV:ksi. Tallennusnappia painettaessa ohjelma kysyy perinteisellä Windows-dialogilla polun mihin tiedosto tallennetaan. Itse CSV-tiedoston luominen suoritetaan itsetehdyllä funktiolla.

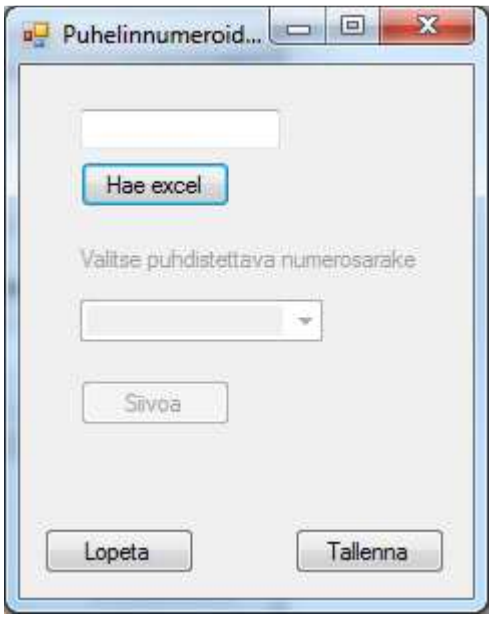
```
string polku = sd.FileName;
```

```
System.IO.StreamWriter writer = new System.IO.StreamWriter(polku, true,
System.Text.Encoding.Default);
int rowcount = luotu_lista.Rows.Count;
int colcount = luotu_lista.Columns.Count;
int i, j;
string rivi = "";

      for (i = 0; i < rowcount; i++)
      {
                  rivi = "";

                  for (j = 0; j < colcount; j++)
                  {
                              rivi       =       rivi       +       luo-
                              tu_lista.Rows[i][j].ToString() + ";";
                   }

                  writer.WriteLine(rivi);
      }

writer.Close();
MessageBox.Show("Tiedosto tallennettu, " + polku);
```

Funktio käyttää hyväkseen kehitysympäristön valmista StreamWriter-kirjastoa, jolla voidaan nopeasti kirjoittaa tiedostoja. Luodun listan DataTable käydään yksinkertaisesti for-silmukalla läpi rivi ja sarake kerrallaan, ja samalla jokaisen sarakkeen väliin lisätään puolipiste.

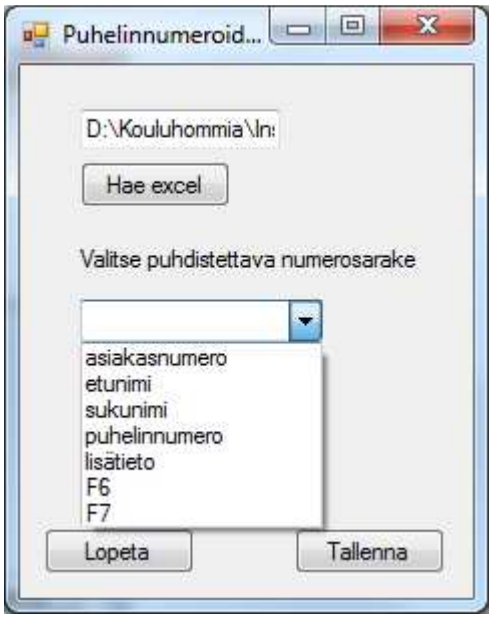
6.3 Soittolistan puhelinnumeroiden siivous-osan kehitys

Soittolistan puhelinnumeroiden siivoaminen on soittolistojen hallinnassa yksi erittäin tärkeä toiminto. Asiakkaiden toimittamissa aineistoissa voi olla puhelinnumeroiden seassa muitakin merkkejä kuin numeroita. Soittojärjestelmä itsessään ei osaa siivota numeroita ja pahimmassa tapauksessa virheellisiä merkkejä sisältävä numero voi estää koko aineiston lataamisen järjestelmään. Puhelinnumeroiden siivoukseen pääsee Soittolistojen hallinta-ohjelman päävalikosta. Siivoustyökalu on hyvin yksinkertainen ja nopea käyttää.



KUVA 23. Siivoustyökalu

Aluksi varsinanen siivousosio on poistettu käytöstä. Siivousosio tulee käyttöön, kun käyttäjä on valinnut Excel-tiedoston, joka siivotaan. Excel-tiedoston avaamisessa käytetään samaa tekniikkaa kuin aiemmassa soittolistan luonti-osiossa.

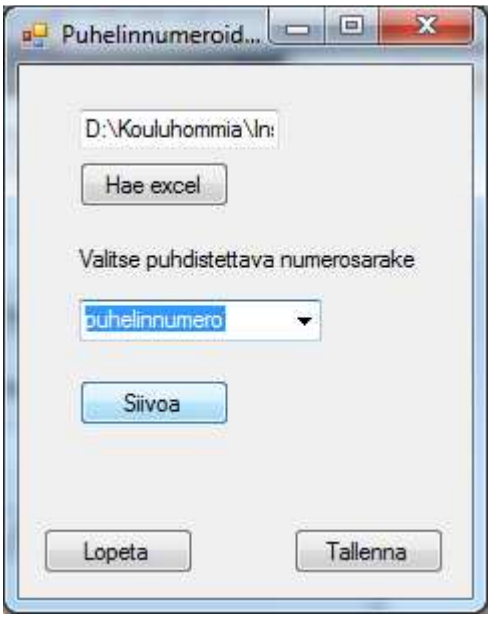


KUVA 24. Aktiivinen siivousosio

Kun käyttäjä on valinnut siivottavan Excel-tiedoston, täytetään siivousosion pudotusvalikko Excelin sarakkeiden nimillä. Käytettävä Excelparser-luokka palauttaa Excel-tiedoston datatablena, jolloin sitä on helppo käsitellä. Datatable käydään läpi for-silmukan avulla ja jokaisella kierroksella lisätään sarakkeen otsikko pudotusvalikkoon.

```
Excelparser data = new Excelparser(tiedosto,
Path.GetExtension(tiedosto), true);

xltable = data.AnnaDatatable;

int rivilkm = xltable.Rows.Count;

int colLkm = xltable.Columns.Count;
int i = 0;


for (i = 0; i < colLkm; i++)
{
                    comboBox1.Items.Add(xltable.Columns[i].ColumnName);
}
panel1.Enabled = true;
```

Valittuaan haluamansa sarakkeen, käyttäjä voi aloittaa sarakkeen siivouksen painamalla *Siivoa*-nappia.



KUVA 22. Sarake valmiina siivottavaksi.

Siivous suoritetaan käymällä jokainen valitussa sarakkeessa oleva solu läpi for-silmukalla ja lähettämällä joka kierroksella solun sisältö tekstimuotoisena funktiolle PuhdistaNumerot().

```
public String PuhdistaNumerot(string numero)
{
      if (numero.Length > 1)
      {
                  char[] nums = new char[numero.Length];

                  int pos = 0;
                  foreach (char ch in numero)
                  {
                              if (Char.IsDigit(ch))
                              nums[pos++] = ch;
                  }

                  return new String(nums, 0, pos);
      }

return "0";

}
```

Funktio käy saamansa tekstimuotoisen muuttujan merkki merkiltä läpi ja tarkastaa Char.IsDigit()-metodilla onko kyseinen merkki numero vai ei. Numeromerkeistä kasataan uusi tekstimuuttuja joka palautetaan lopuksi kutsujalle. Käyttäjä voi tallentaa aineiston painamalla työkalun Tallenna-nappia, jolloin siivottu datatable avataan Excelissä ja käyttäjä voi hoitaa tallennuksen Excelin tallennustoimintoa käyttäen.

6.4 Soittolistojen tilanne

Soittolistojen tilanne -osiosta luovuttiin projektin aikana kokonaan, eikä sen kehitystä edes aloitettu. Osion hyödyllisyys suhteessa kehitystyön määrään olisi ollut liian pieni, sillä Avaya-soittojärjestelmä tarjoaa kyseisen toiminnon, joskin erittäin suppeana.

7 Yhteenveto

Tämän insinöörityön tavoitteena oli kehittää Call Wavesille Windows-pohjainen työkalu, jolla on mahdollista suorittaa tiettyjä Avaya-puhelinjärjestelmästä puuttuvia toimintoja. Näistä tärkein ja halutuin toiminto oli puhelinjärjestelmän siivous, jossa jo soitetut puhelinnumerot siivotaan pois järjestelmästä. Muita haluttuja ominaisuuksia olivat soittolistojen luonti, soittolistojen puhelinnumeroiden puhdistus ja soittolistojen tilan seuranta. Näistä ominaisuuksista soittolistojen luontiin ja puhelinnumeroiden puhdistukseen oli jo paljon valmista koodia, joten kehitystä ei tarvinnut aloittaa täysin puhtaalta pöydältä.

Varsinainen kehitystyö ja koodin kirjoitus sujui koko projektin ajan melko hyvin, eikä suuria suunnitteluvirheitä ilmennyt ohjelman kehityksen aikana. Ensimmäisenä valmiiksi kehitettiin puhelinjärjestelmän siivous-osio, koska Call Wavesin it-osastolla oli sille suuri tarve. Puhelinjärjestelmän siivous-osion valmistuttua työkalu siirtyi heti käyttöön. Koska työkalu tuli kehittäjän ja muun it-osaston omaan käyttöön, saatiin työkalussa ilmenneet virheet nopeasti korjattua. Aluksi työkalua käytettiinkin suoraan Visual Studion virheenkorjaus-tilassa, jotta mahdolliset virheet saataisiin heti kiinni. Puhelinjärjestelmän siivous-osion jälkeen kehitettiin puhelinnumeroiden puhdistus-osio. Kyseisen osion kehitys ja liittäminen osaksi työkalua tapahtui nopeasti, koska valmista koodia oli paljon. Soittolistojen luonti-osiosta oli myös olemassa jo valmis runko, jota lähdettiin kehittämään eteenpäin. Tuloksena saatiin suhteellisen toimiva soittolistojen luonti-työkalu, jolla etenkin yksinkertaiset tietyiltä asiakkailta tulevat soittoaineistot saadaan nopeasti käsiteltyä. Soittolistojen tilan seuranta-osio jätettiin lopullisesta ratkaisusta kokonaan pois, koska sille ei koettu olevan käyttöä. Vastaava toiminto löytyy jo Avaya-puhelinjärjestelmän omista hallintatyökaluista.

Projektin tuloksena syntyi yksinkertainen työkalu, joka nopeasti maksoi siihen käytetyn kehitysajan takaisin säästyneinä työtunteina. Nyt puhelinjärjestelmän siivous on mahdollista toteuttaa yhden henkilön toimesta muutamassa tunnissa. Tavoitteet siis saavutettiin odotuksien mukaisesti. Sovelluksen ulkoasu ja komponenttien asettelu kaipaa jatkokehitystä.

LÄHTEET

Avaya, Contact Centers. [Viitattu 10.10.2011]. Saatavissa: http://www.avaya.com/usa/solutions/portfolio--contact-centers

Järvinen Jani, 2008. Visual Studio 2008-käsikirja (verkkokirja), WSOYPro, Visual Studion uusin versio-luku. Saatavissa: https://aapeli.amkit.fi

Kinnunen Jukka, 2004. Ohjelmistojen laatu ja testaaminen. Moniste. Ei kustantajaa. Varkaus: Savonia-ammattikorkeakoulu.

Relaatiotietokannat. Informaatioteknologia, Jyväskylän yliopiston IT-tiedekunta ja avoin yliopisto. [Viitattu 10.1.2012]
Saatavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/tietokannat/index1.html