OULUN SEUDUN
AMMATTIKORKEAKOULU

Jaakko Partanen

PERSONAL TRAINER -AJANVARAUSJÄRJESTELMÄ

PERSONAL TRAINER -AJANVARAUSJÄRJESTELMÄ

Jaakko Partanen
Opinnäytetyö
Kevät 2011
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

OULUN SEUDUN AMMATTIKORKEAKOULU TIIVISTELMÄ

Koulutusohjelma	Opinnäytetyö	Sivuja	+	Liitteitä
Tietotekniikan koulutusohjelma	insinöörityö	43	+	0

Suuntautumisvaihtoehto	Aika
Sulautetut ohjelmistot	2011

Työn tilaaja	Työn tekijä
Trainer4You Oy	Jaakko Partanen

Työn nimi

Personal Trainer -ajanvarausjärjestelmä

Avainsanat

Qt, MySQL, SQLite, WAMP, kosketusnäyttö, käytettävyys

Insinöörityössä suunniteltiin ja toteutettiin kosketusnäytöllä toimiva yksinkertainen ja helppokäyttöinen ajanvarausjärjestelmä Trainer4You-yritykselle. Käytettävyys oli yksi työn aiheista ja tähän kiinnitettiin huomiota käyttöliittymän kehittelyssä. Ohjelman kautta voi varata aikoja yrityksen eri palveluihin ja samalla yritys saa lisämainosta ja näkyvyyttä itselleen. Ohjelmalle haluttiin myös mahdollisuus käyttää sitä useammassa toimipisteessä.

Toteutukseen käytettiin Qt-ohjelmointiympäristöä, joka pohjautuu C++-kieleen, mutta hyödyntää omaa koodigeneraattoria ja makroja. Aluksi tietokanta oli tarkoitus toteuttaa MySQL:n avulla, mutta ajureiden kanssa ilmenneiden ongelmien takia tästä luovuttiin ja toteutus tapahtui SQLitellä. SQLite antaa melkein kaikki ominaisuudet pienemmässä paketissa kuin MySQL ja on riittävä tämän ohjelman toiminnan kannalta.

Lopulta syntyi toimiva ajanvarausjärjestelmä, jolla on jatkokehitysmahdollisuuksia. Ohjelma toimii tulevassa toimipisteessä sellaisenaan, mutta vaatii joidenkin asetuksien muutoksia, jos se halutaan muualle. Tästä syystä asetusvalikko olisi ollut hyvä tehdä, mutta toteutukselle ei jäänyt aikaa ja se ei kuulunutkaan alkuperäisiin suunnitelmiin. Myös muita jatkokehitysaiheita on ilmennyt ja on ehdotettu tilaajayritykselle.

OULUN SEUDUN AMMATTIKORKEAKOULU ABSTRACT

Koulutusohjelma	Opinnäytetyö	Sivuja	+	Liitteitä
Information Technology	Bachelors Thesis	43	+	0

Suuntautumisvaihtoehto	Aika
Embedded Systems	2011

Työn tilaaja	Työn tekijä
Trainer4You Oy	Jaakko Partanen

Työn nimi

Personal Trainer - Time Reservation System

Avainsanat

Qt, MySQL, SQLite, WAMP, touch screen, usability

Simple and ”easy to use” – time reservation program was planned and made utilizing a big touch screen TV in this thesis. Usability was a major point during whole process and influenced greatly in planning of GUI and inner mechanics of the work. Through the system user could reserve time to any of the provided services by the company (Trainer4You). Also the company gained more advertising space and visibility with this solution. Program was requested to be able to be used in different offices.

Program was made using Qt programming framework which is based on C++ programming language, but it uses it’s own code generator and macros. By original planning database was to be made using MySQL, but drivers in Qt gave lots of problems. Because of this, solution was made using SQLite which provides almost all features of MySQL in smaller package and that was enough for this software.

The final result was time reservation system that has further developement possibilities. Program will be at it’s future office as is, but if taken to other location some settings need to be changed internally. For this reason, creating a settings menu would have been good, but there wasn’t enough time to implement it and it wasn’t included in the original plan. Other ideas were found also and some have been proposed to commissioning firm.

SISÄLTÖ

KÄSITTEET

API Application Programming Interface, ohjelmointirajapinta Ohjelma kommunikoi tätä hyödyntäen muiden ohjelmien kanssa.

Asynkroninen
Synkronisen vastakohta. Ei-reaaliaikainen kommunikointi on kommunikointitapa, jossa kommunikoinnin osapuolet eivät ole ajallisesti toisistaan riippuvaisia eli ovat ajasta (ja paikasta) riippumattomia.

GPL GNU General Public License on vapaiden ohjelmistojen julkaisemiseen tarkoitettu lisenssi. Antaa kenelle tahansa oikeuden käyttää, kopioida, muuttaa ja jakaa edelleen ohjelmia ja niiden lähdekoodia.

GUI Graphical User Interface, graafinen käyttöliittymä.

Hard coded Kovakoodattu eli esimerkiksi jotkin asetukset (palvelin, portti) ovat suoraan koodissa sellaisenaan eikä muuttujana, jota voi vaihtaa. Aiheuttaa sen, että jos muutoksia tulee, ne pitää fyysisesti kirjoittaa koodiin uudelleen ja kääntää lähde toimivaksi ohjelmaksi uudestaan.

HCI Human-Computer Interaction, ihmisen ja koneen vuorovaikutus.

ISP Internet Service Provider, internetpalvelujen tarjoaja.

LGPL GNU Lesser General Public License on ehdoiltaan kevyempi lisenssi. Pääasiallinen eroavaisuus GPL:n ja LGPL:n välillä on se, että LGPL-ohjelmistot voidaan linkittää yhteen ei-GPL-lisensoidun ohjelman osan kanssa.

OpenGL Open Graphics Library, avoin grafiikkakirjasto.

PC Personal Computer, pöytätietokone

QSA	Qt Script for Applications, skriptauskieli sovelluksille. Työkalukokoelma, jonka avulla C++-sovelluksista saadaan tehtyä skriptattavia käyttäen Qt Script-kieltä.
SMTP	Simple Mail Transfer Protocol, yksinkertainen postin välityspalvelu.
SQL	Structured Query Language, jäsennelty kyselykieli. Käytetään tietokantahakuihin esimerkiksi MySQL-tietokantojen kanssa.
SSL	Secure Sockets Layer, tiedonsalausmenetelmä. Koostuu kahdesta avaimesta: julkinen, jonka tietävät kaikki, ja yksityinen, jonka tietää vain viestin oikea vastaanottaja.
TCP	Transmission Control Protocol, lähetyksenhallintaprotokolla
Transaktio	Kokoelma tietokantaoperaatioita, jotka suoritetaan kerralla ja periaatteella ”kaikki tai ei mitään” eli jos jokin osa ei onnistu koko operaatio peruutetaan.
UI	User Interface, käyttöliittymä.
WAMP	Ohjelmakokonaisuus, jonka nimi tulee sen sisältämistä ohjelmista ja käyttöjärjestelmästä: Windows, Apache, MySQL ja PHP, Perl ja/tai Python.

1 JOHDANTO

Työn tarkoituksena oli tehdä Trainer4You-yritykselle ajanvarausjärjestelmä asiakkaille ja mahdollisesti houkutella ratkaisulla uusia asiakkaita. Aiemmin varaukset tehtiin ottamalla yhteys yrityksen henkilöstöön suoraan tai yrityksen esittelypisteellä vierailun yhteydessä. Uudessa järjestelmässä ajanvarauslaitteisto sijoitetaan esittelypisteen läheisyyteen liikuntakeskus Hukassa ja se on käytettävissä myös esittelyaikojen ulkopuolella. Jos järjestelmä toteutetaan hyvin ja se koetaan hyödylliseksi, se voidaan ottaa käyttöön yrityksen muissa toimipisteissä, joita löytyy ympäri Suomea esimerkiksi Helsingistä, Jyväskylästä, Kuopiosta ja Rovaniemeltä.

Järjestelmä koostuu tietokoneesta ja näkyvin osa tulee olemaan suuri kosketusnäyttö. Kokemukset suurien kosketusnäyttöjen ja mainonnan tekemistä ovat myös hyvin esillä nykyisellään kaikenlaisissa tuotteissa ja varsinkin kosketusnäyttöjä tuodaan uusiin tuotteisiin melkein päivittäin. Ohjelman käyttöliittymä on suunniteltu mahdollisimman yksinkertaiseksi ja helppokäyttöiseksi, jotta käyttäjä voi helposti ja mukavasti tehdä varauksen. Jos ohjelma ei ole käytössä, näytöllä näytetään mainoksia, mutta myös houkutellaan käyttäjää koskettamaan sitä ja huomaamaan ajanvarausmahdollisuus. Houkuttelevuuden kannalta ulkoasulla on erittäin suuri merkitys. Toteutukseen valittiin Qt-ohjelmointiympäristö sen monipuolisen käyttöliittymän suunnittelutyökalun takia ja erittäin hyödyllisen ammatillisen kehitysmahdollisuuden takia työelämää varten. Yrityksen hyötynä on Windows-käyttöjärjestelmässä toimiva ohjelma, jonka käyttäminen ei vaadi suurta teknista osaamista. Riittää, että osaa käynnistää tietokoneen ja napsauttaa pikakuvakkeesta ohjelman päälle. Lisäksi yritys saa näkyvyyttä työaikojen ulkopuolella liikuntahallin aukioloaikojen puitteissa.

2 KÄYTETYT TEKNIIKAT

2.1 Qt

Qt on ohjelmointiympäristö, jonka kehitti norjalainen Trolltech-yritys. Qt:llä tehdään ohjelmia pöytäkoneille ja myös sulautetuille laittelle ja sen ominaispiirteisiin ja lähtökohtiin kuuluu toiminta-alustojen rajojen ylittäminen. Toisin sanoen kertaalleen kirjoitettu koodi voidaan kääntää uudestaan eri alustaa varten ja sitä ei tarvitse muokata alusta alkaen. Aiemmin ohjelma piti kirjoittaa melkein kokonaisuudessaan uusiksi, jos tahdottiin siirtää ohjelma toiselle laitteelle. Koska ohjelmat lähtökohtaisesti voidaan tehdä useammalle alustalle, säästetään aikaa integroinnista ja se voidaan käyttää ohjelman parantamiseen. Qt:ssä on myös hyvin kattava dokumentaatio kaikista sen ominaisuuksia. (The Qt FAQ. 2005; Qt framework. 2011.)

Qt mahdollistaa graafisten käyttöliittymien nopean tuottamisen, koska se käyttää jokaisen laitteen omia ominaisuuksia eli API:a grafiikan tuottamiseen ja saa systeemin resurssit kokonaan käyttöönsä. Tämä vähentää tarvittavia koodirivejä, mahdollistaa erinomaisen skaalautuvuuden sekä tuottaa hienon ulkoasun laitekohtaisesti ja hyvän käyttöliittymän. Lisäksi Qt mahdollistaa 3D-grafiikan käytön hyödyntäen OpenGL-grafiikkakirjastoa. Qt:ssä on käytössä monisäikeisyys, eli ohjelman tehdessä esimerkiksi laskutoimituksia käyttäjä voi jatkaa vaikka puhelinluettelon selaamista samaan aikaan, koska tehtävien suorittaminen on jaoteltu ja yksi tehtävä ei saa kaikkia resursseja. Google Earth, Skype, VLC Media Player ja Virtual Box ovat hyviä esimerkkejä ohjelmista, jotka on tehty Qt:llä. (The Qt FAQ. 2005; Qt framework. 2011.)

Tämän lisäksi C++ GUI Programmin with Qt 4 -kirjan alkusanat kiteyttävät erään hyvin oleellisen asian ohjelmoijan näkökannalta. Kehitysympäristön tulee olla looginen, eikä pakottaa tekemään sattumanvaraiselta tuntuvia ratkaisuja. Qt onnistuu myös vähentämään päällekkäisyyksiä ja toistuvuutta, joiden välttäminen nopeuttaa työskentelyä. (Blanchette – Summerfield, 2006, IX.)

Qt:ssä järjestelmän eri osat voivat kommunikoida keskenään ilman tietoa toisistaan. Tämä mekaniikka on nimeltään meta-object system. Siinä on kaksi avainpalvelua: Signals & Slots eli vapaasti suomennettuna viestit ja paikat ja Introspection eli itsetarkkailu. Jokainen signaali voidaan kytkeä yhteen tai useampaan paikkaan ja paikkaan voi olla kytkettynä yksi tai useampi

signaali. Paikat ovat normaaleja C++-funktioita. Erona on vain, että niihin voidaan liittää viesti ja viestin lähtiessä funktiota kutsutaan aina automaattisesti.

Viestin ja paikan liittäminen tapahtuu seuraavasti connect()-function avulla. Ensimmäinen parametri on lähettäjä, toisena viesti, kolmantena vastaanottaja ja viimeisenä funktio.

```
connect(sender, SIGNAL(signal), receiver, SLOT(slot));
```

Esimerkiksi voidaan tehdä tekstikenttä, johon voi laittaa numeroarvon. Lisäksi käytössä on erillinen lcd-tyylinen näyttöalue, jolle halutaan päivittää tekstikentän sisältö, jos sitä muutetaan. Silloin lause voisi näyttää tältä:

```
connect(tekstikentta, valueChanged(int), lcd, setValue(int));
```

Signaalien ja slottien käytön mahdollistaa itsetarkkailumekanismi, joka kerää QObject-aliluokilta tarvittavan tiedon eli niiden signaalit ja paikat ja luokan nimen. Mekanismi myös tukee tekstikäännöksiä, jolloin valikot ja muut ohjeet kääntyvät oikealle kielelle kielivalintojen mukaan jos mahdollista. Se myös luo pohjan QSA:lle (Qt Script for Applications). Standard C++ ei tarjoa tukea dynaamiselle meta-informaatiolle, jota Qt:n meta-object system tarvitsee. Qt ratkaisee tämän ongelman käyttämällä erillistä työkalua nimeltään *moc.*
Mekanismi toimii seuraavasti:

- Q_OBJECT-makro asettaa muutaman itsetarkkailufunktion jotka sisällytetään jokaiseen QObject-aliluokkaan: metaObject(), tr(), qt_metacall() ja muutama muu.
- Qt:n moc-työkalu toteuttaa toiminnot funktioille, jotka Q_OBJECT asetti ja kaikille signaaleille.
- QObject jäsenfuktiot kuten connect() ja disconnect() käyttävät itsetarkkailufunktiota omaan työhönsä.

Kaiken tämän qmake, moc ja QObject hoitavat automaattisesti, joten siihen ei yleensä tarvitse kiinnittää huomiota, jolloin riittää yhteyksien luominen connect()-funktion avulla. (Blanchette – Summerfield, 2006, IX.)

2.2 MySQL

Tietokanta on jäsennelty kokoelma tietoa ja se voi olla yksinkertainen kuvagalleria tai pankin asiakasrekisteri. Tietokannan hyödyt tulevat esiin, kun halutaan käsitellä, säilöä ja organisoida suuria määriä dataa. Käytännössä kun työstä tulee liian hankalaa ja aikaanvievää tehdä käsin, on tietokannan käyttö suotavaa. Esimerkiksi pankeilla on tileihin, asiakkaisiin, osakkeisiin jne. liittyviä tapahtumia sadointuhansin tai miljoonittain joka päivä, joten tarvittaisiin käsittämätön määrä työntekijöitä tekemään merkintöjä tapahtumista tai tehokas ja luotettava tietokanta, joka voi käsitellä paljon tietoja nopeasti ja samanaikaisesti. Tiedon käsittelyyn käytetään siihen tehtyjä ohjelmistoja, kuten MySQL (My Structured Query Language), jotka sallivat tiedon haun ja säilönnän mahdollisimman tehokkaasti. MySQL on hyvin yleisesti käytetty avoimen lähdekoodin tietokantojen hallintajärjestelmä, jota kehittää ja ylläpitää Oracle. (DuBois 2003, 3–7; MySQL Reference Manual 5.5, chapter 1.3.1.)

MySQL on nopea, ellei jopa kaikista nopein tietokantojen hallintajärjestelmä ja tiedonkäsittely sillä on näppärää, koska haun järjestys ei ole kiinteä, vaan sen voi määritellä. Lisäksi tulostusmuoto on itse määriteltävissä. Toisaalta MySQL on tehokas, mutta yksinkertainen ja helppo asentaa, varsinkin jos käyttää valmiita paketteja (esim. WAMP). WAMP on ohjelmistokokonaisuus Windowsille ja se pitää sisällään Apachen, MySQL:n ja PHP:n, Perlin ja/tai Pythonin. Lisäksi MySQL:llä on hyvä saatavuus avoimuutensa takia ja samalla se on halpa tai jopa ilmainen. Pienen kokonsa ja hyvän siirrettävyyden ansiosta MySQL toimii hyvin monella alustalla. Se myös tukee useampia yhteyksiä samanaikaisesti useampaan tietokantaan samanaikaisesti ja tukee monia rajapintoja käyttämistä varten (PHP, C, Perl, Java, Python jne.). MySQL on asiakas-palvelinyhdistelmä ja sisältää monisäikeisen SQL-palvelimen, joka tukee monia erilaisia asiakasohjelmia, kirjastoja ja ylläpitotyökaluja. Lisäksi tarjolla on myös palvelinkirjasto sulautetuille laitteille jonka voi linkittää omaan sovellukseen. Palvelinyhdistelmä on turvallinen käyttää omien ominaisuuksien takia ja tukee lisäksi Secure Sockets Layeria (SSL). (DuBois 2003, 3–7; MySQL Reference Manual 5.5, chapter 1.3.1.)

MySQL on relaatiotietokanta eli tiedot voidaan kytkeä toisiin tietohin, jolloin saadaan luotua kokonaisuuksia. Ominaisuus ei ole pakollinen käyttää, mutta helpottaa huomattavasti ylläpidettävyyttä, tietojenkäsittelyä jne. Se on myös yleisin standardisoitu kieli, jota käytetään tietokantojen käsittelyssä ja se on määritelty ANSI/ISO SQL -standardissa. Se on GPL-lisenssin

alainen. Tosin jos tarve vaatii, ostettavissa on kaupallinen lisenssi. MySQL:ää käyttävät esimerkiksi: nokia.com, youtube.com, Wikipedia, Google ja Facebook. (MySQL Reference Manual 5.5, chapter 1.3.1.)

2.3 SQLite ja phpLiteAdmin

SQLite on tietokantajärjestelmä, joka muistuttaa MySQL:ää paljon ja käyttää samaa SQL-kieltä toiminnoissaan. Tosin se ei tarvitse ollenkaan asetuksia, on täysin palvelimesta riippumaton ja tukee transaktioita. SQLite on kevyempi ja pienempi kooltaan, jolloin se ei tue ihan kaikkia käskyjä. Esimerkkeinä taulujen yhteishakuja on karsittu, taulurakenteiden muokkaus on hyvin rajallinen ja käyttäjien oikeuksien tai tunnuksien luonti ei ole mahdollista. Silti suurin osa MySQL:ää vastaavista toiminnoista löytyy ja on samoja ja nämä ominaisuudet riittävät hyvin kehitettävässä ohjelmassa, koska paikallinen tietokanta riittää sen tarpeisiin. Koko tietokanta on yhdessä tiedostossa, joka on alustariippumaton eli sen voi suoraan ottaa käyttöön vaikka Linuxilla, kun alkuperäinen on Windows-koneelta. Ohjelma soveltuu erinomaisesti pieniin tai keskisuuriin web-sovelluksiin, mobiilisovelluksiin sekä testaus- ja demokäyttöön. (Features of SQLite.)

Oleellisinta kuitenkin on, että SQLite on suoraan tuettuna Qt:ssä ja sen ajurit ovat mukana. Tässä työssä lopulta MySQL-ajureiden kääntäminen ei onnistunut ja ajanpuutteen vuoksi päädyttiinkin käyttämään SQLiteä. Samalla vaihtoon meni WAMP-paketin käyttö, johon sisältyi normaali phpMyAdmin ja otettiin samantyylinen phpLiteAdmin käyttöliittymäksi tietokannalle. Mikään tietokantojen hallintaohjelma ei ole täysin pakollinen loppukäytössä, helpotus kylläkin työskentelyvaiheessa.

PhpLiteAdmin on web-pohjainen tietokannan hallintatyökalu SQLite2:lle ja SQLite3:lle. Se koostuu yhdestä tiedostosta, joka kopioidaan tietokannan kanssa samaan paikkaan ja avataan se selaimessa. Asetuksista vaihdetaan salasana ja valitaan tietokanta ja käyttöliittymä on valmis. Taulut voidaan luoda tätä kautta ja katsoa helposti tietoja ja seurata, onko kaikki oikeilla paikoilla. Toisaalta phpLiteAdmin kannattaa pitää ohjelman mukana tässä vaiheessa, jotta voi esimerkiksi siistiä vanhat varaukset pois aina silloin tällöin. Ajavaraus ei tällöin pääse hidastumaan ainakaan tietokannan turhan kasvamisen myötä. (PhpLiteAdmin features.)

3 OHJELMISTOJEN KÄYTETTÄVYYS

3.1 Käytettävyyden kehittyminen

Jo 1970-luvulla alkoi tulla selväksi, että käyttöliittymäsuunnittelusta tulisi iso osa ohjelmistokehitystä. Kun yhä suurempi osa ohjelmistoista kehitettiin interaktiivisiksi eli vuorovaikutteisiksi, alkoi huomio keskittyä loppukäyttäjän tarpeisiin ja mieltymyksiin. Aiemmin suurin osa ohjelmistojen käyttäjistä oli tietojenkäsittelyn ammattilaisia, jotka kehittivät ja ylläpitivät järjestelmiä yrityksissään. Tietokoneiden kehittyessä tehokkaammiksi käyttäjäkunta alkoi kasvaa monipuolisemmaksi, joten sovelluksien asentamista ja muokattavuutta yksinkertaistettiin. Tavalliset toimistotyöntekijät alkoivat saada vastuun käyttää omia sovellusohjelmiaan ja PC:n ilmestymisen myötä oli tyypillistä, että loppukäyttäjä asensi ja hallitsi omia ohjelmistojaan. (Rosson – Carroll 2002, 9.)

Käytettävyyden käsite syntyi käyttäjien monipuolistumisen kautta, koska samalla entistä suurempi osa käyttäjistä ei ollut teknisesti lahjakkaita. Vuorovaikutteisia järjestelmiä alettiin vertailla ja arvioida suhteessa käytettävyyteen: järjestelmän laadukkuus suhteessa käyttämisen oppimisen helppouteen, helppokäyttöisyyteen ja käyttäjän tyytyväisyyteen. (Rosson – Carroll 2002, 9.)

Käytettävyys tuli mukaan ohjelmistokehitysprosessiin molemmista päistä eli alkuun vaatimuksena ja loppuun systeemitestauksessa. Markkinointiryhmät haastattelivat asiakkaita ja analysoivat kilpailevia tuotteita ymmärtääkseen vaatimukset. Laadunvalvontaryhmät testasivat suunniteltuja vaatimuksia järjestelmille, jotta ihminen olisi tehokas järjestelmien käyttäjä. Yleensä tämä testaus oli ”inhimillinen tekijä”-arvioinnin yhteenveto. Valitettavasti markkinointiryhmät puhuivat harvoin varsinaisille käyttäjille vaan saivat tietonsa hallinnolta tai vastaavien tuotteiden tutkimisesta. Myös laadunvalvonnan ongelmana oli, että laadunvalvontatestaus suoritettiin kehitystyön lopussa. Saatettiin löytää oikeita ongelmakohtia, mutta löydökset tulivat liian myöhään, jotta niitä voitiin huomioida. (Rosson – Carroll 2002, 9–10.)

3.2 Käyttöliittymän käytettävyyden määrittely

Sinisalon ja Lehtosen raportissa lainataan Normania, jonka mukaan käytettävyydessä tutkitaan ihmisen ja koneen välistä kommunikaatiota ja sillä määritellään niiden välistä toimintaa. Käytettävyyden mittaaminen on vaikeaa, joten sitä lähestytään kuvailemalla. Tällöin joudutaan hakemaan ominaisuuksia, joista tehdään päätelmiä. Hyvä käytettävyys ilmenee mukavana ja helppona käyttökokemuksena ja tällainen syntyy, kun käyttäjä ei joudu ratkaisemaan ongelmia. (Lehtonen – Sinisalo 2007, 6.)

Optimaalinen käyttöliittymä vaatii systemaattista lähestymistä suunnittelussa. Tästä johtuen käytettävyyden testaus pitää sisällyttää suunnitteluun, jotta varmistetaan tehokkuus. Testauksen kautta saadaan kuva, mitkä asiat toimivat suunnitellusti ja mitkä eivät. Tarvittavien korjauksien jälkeen voidaan käyttöliittymää kutsua optimoiduksi käyttäjän kannalta. (Lehtonen – Sinisalo 2007, 6.)

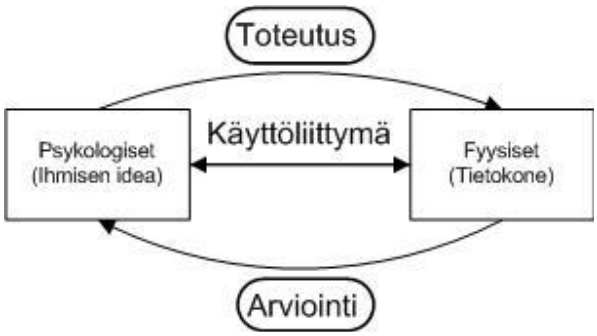
Käytettävyyden toteuttaminen on onnistunut erinomaisesti silloin, kun käyttäjä voi keskittyä työtehtäväänsä ilman että ajattelee työvälinettään. Työvälineen käyttäminen ei siis vaadi ylimääräisiä ponnisteluita, aikaa tai resursseja. Jos käytettävyydessä on epäonnistuttu, siitä aiheutuu yksityishenkilöille, yhteiskunnalle ja yrityksille lisäkustannuksia. Kun laitteen käytettävyys on huono, laite hidastelee ja on epälooginen, käyttämiseen kuluu enemmän aikaa ja resursseja, kuin olisi optimaalista, ja tästä syntyy kuluja. Lisäksi tämä voi vaikuttaa negatiivisesti ihmisen mielialaan, koska käyttäjästä alkaa tuntua, että virheet johtuvat hänen osaamattomuudestaan ja motivaatio voi laskea. Käytettävyydeltään miellyttävä laite taas voi nostaa mielialaa mukavuutensa kautta. (Kuparinen 2008, 30.)

3.3 Ihmisen ja koneen kommunikaatio

Sinisalon ja Lehtosen raportissa ihmisen ja koneen vuorovaikutusta (HCI eli Human-Computer Interaction) lähestytään Normanin toiminnan teorian kannalta. Teorian mukaan ihmiset luovat ideoita ja kone vain suorittaa tehtävän eli se on työkalu. Ihminen antaa alkuperäisen idean, jolla on haluttu lopputulos, ja käynnistää toiminnon koneesta ja kone vastaa ohjelmoidulla tavallaan.

Tätä tulosta verrataan siihen, mitä alun perin haluttiin tehdä. Tulkintana tämä on hyvinkin osuva omasta mielestäni. (Lehtonen – Sinisalo 2007, 6–7.)

Edellä kuvattu kommunikaatio ihmisen ja koneen välillä tapahtuu käyttöliittymän kautta. Käyttöliittymä (UI eli user interface) on laitteen tai ohjelman osa, jota kautta käyttäjä antaa tehtävän kyseiselle tuotteelle. Esimerkkeinä voidaan lyhyesti mainita kahvinkeittimen virtakatkaisin, jolla kerrotaan keittimelle milloin kahvia pitää keittää, tai pyykkikone, jolle voidaan kertoa pesulämpötila, rummun pyörimisnopeus jne. Tätä työtä lähempänä on graafinen käyttöliittymä (GUI eli graphical user interface), jossa vuorovaikutus ihmisen ja koneen välillä tapahtuu tekstin, kuvien, ikoneiden ja käyttöliittymäelementtien avulla. Kuvassa 1 on esitetty yksinkertainen vuorovaikutustilanne. (Lehtonen – Sinisalo 2007, 6–7.)



KUVA 1. HCI (Lehtonen – Sinisalo 2007, 7.)

Normanin teoriaan kuuluu myös ihmisen luoma henkinen malli. Käyttäjän tehokkuus paranee sitä mukaa, mitä paremmin rakennettu henkinen malli vastaa kyseistä käyttäjää. Tätä kautta mahdollistuu tiedon soveltaminen ratkaisuja etsittäessä. Jos käyttäjä pystyy hahmottamaan graafisen käyttöliittymän henkisen mallin päässään, sitä paremmaksi hänen tehonsa kasvaa koneen käytössä. Tästä syystä henkinen malli kannattaa huomioida käyttöliittymäsuunnitellussa ja rakentaa siitä mahdollisimman yksinkertainen ja yhtenevä. Esitetyillä ikoneilla on myös oltava yhteys reaalimaailmaan. Käytettävyysongelmia syntyy varmasti, jos ikoni näyttää samalta kuin jokin oikea asia, mutta toimii eri tavoin. Kärjistetty esimerkki voisi olla roskakorin kuvake, joka toimiikin tulostiminena. Jos suunnittelija rakentaa käyttöliittymän tälle pohjalle, samalla hän avustaa käyttäjää luomaan henkisen mallin järjestelmästä. Käyttäjien luomat omat mielikuvat henkisistä malleista rakentuvat myös toisten järjestelmien pohjalta, ja tämä kertoo siitä, että

henkinen malli saa vaikutteita järjestelmistä monelta suunnalta. (Kuva 2.) (Lehtonen – Sinisalo 2007, 7.)

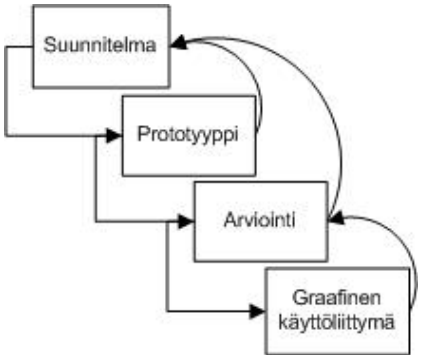
KUVA 2. Henkiseen malliin vaikuttavat tekijät (Lehtonen – Sinisalo 2007, 7.)

Voisi siis olettaa, että mallien käyttö graafisten käyttöliittymien suunnitellussa olisi helppoa, mutta näin ei kuitenkaan ole. Syynä tähän on mallin aineettomuus, koska ne ovat vain käyttäjän tai suunnittelijan pään sisäisiä rakenteita. Erilaiset ihmiset ajattelevat asian eri tavoin, jolloin pieniä logiikkaeroja ilmenee. Siten tarkkaa mallia, rakennetta tai sisältöä on mahdoton saada aikaiseksi, mutta jotain yhteneväisyyksiä ja yksinkertaistuksia voidaan tehdä. Osittain mallien käyttö on alitajuista psykologisesti arvioiden, joten se on hankala hahmotella ulospäin. (Lehtonen – Sinisalo 2007, 8.)

3.4 Käytettävyys ja GUI-suunnittelu

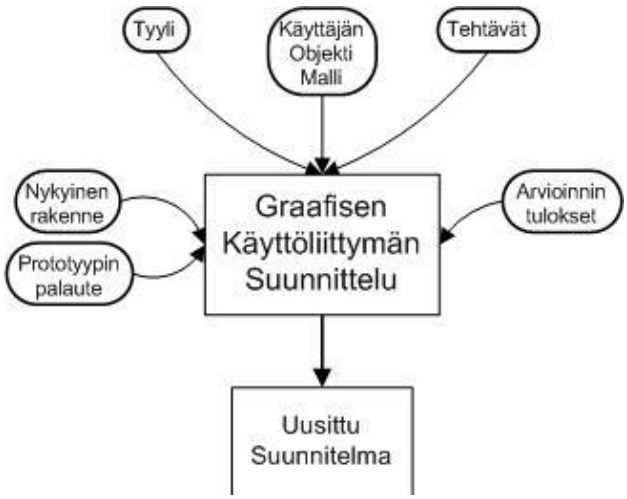
Edellä mainittujen asioiden lisäksi GUI-suunnittelussa on otettava huomioon kognitiivisia tekijöitä. Käyttäjää ei saa laittaa asemaan, jossa pitää muistaa monia yksityiskohtia, vaan pitää käyttää tunnistusta eikä mieleenpalauttamista. Jonkin toiminnon loputtua on myös ilmoitettava tästä käyttäjälle selkeästi. Tämän tyyliset seikat liittyvät ihmisen muistinkäyttöön. Toiseksi pitää muistaa, että ihmiset tekevät virheitä ennemmin tai myöhemmin. Suunnitellaan käyttöliittymä siten, että se voi ennakoida virheitä esimerkiksi estämällä mahdollisten virhetilanteiden synty. Tästä huolimatta virheitä tapahtuu, joten tällaisessa tilanteessa annetaan hyödyllinen virheilmoitus. Ihmiset ovat luonnostaan uteliaita ja sopeutuvaisia uusiin asioihin, joten käyttöliittymän olisi hyvä tukea tätä. Tällainen prosessi onkin tuttu jo monesta perusohjelmasta eli Un-Do (peruuta) ja Re-Do (tee uudelleen) joilla voidaan peruuttaa tehty toiminto tai tehdä uudelleen peruutettu toiminto. (Lehtonen – Sinisalo 2007, 8.)

Raportin mukaan käyttöliittymien suunnitteluun on vaikuttanut kolme asiaa: käyttäjän objektimalli, tehtävien malli ja sovelluksen tyyli. Näistä muodostuu ensimmäinen suunnitelma eli prototyyppi. Kun prototyyppi on luotu, se arvioidaan, ja jos se täyttää vaatimukset voidaan luoda itse graafinen käyttöliittymä. Toisaalta, jos käyttöliittymän prototyyppi todetaan vajaaksi, voidaan joutua palaamaan takaisin suunnitelmaan tai arviointiin kuvan 3 vesiputousmallin mukaisesti. (Lehtonen – Sinisalo 2007, 8.)



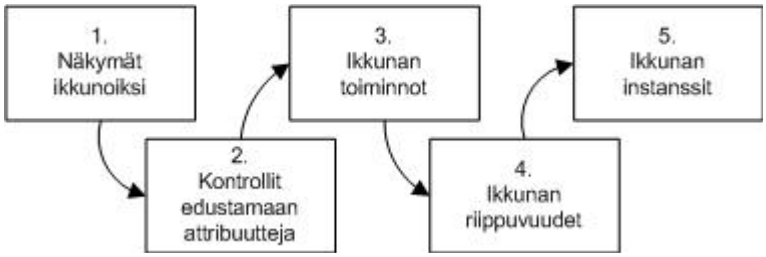
KUVA 3. Osa GUI:n vesiputousmallia (Lehtonen – Sinisalo 2007, 9.)

Kun ensimmäinen vedos on julkaistu, voidaan aloittaa käyttöliittymän kehitys. Graafisen käyttöliittymän kokonaisuutta on tässä vaiheessa luomassa jo enemmän kuin kolme tekijää. Kehitystyön jatkuessa tästä eteenpäin on huomioitava jo valmis käyttöliittymä, prototyypin palaute ja arvioinnin tulokset. GUI-suunnittelua käytetään siis prototyypissä, arvioinnissa ja koko järjestelmän toteutuksessa ja testauksessa. Kuva 4 mallintaa näitä tekijöitä. (Lehtonen – Sinisalo 2007, 8.)



KUVA 4. GUI:n kehittämisen tekijät (Lehtonen – Sinisalo 2007, 9.)

Seuraavana tarkastelun alla on itse toteuttamisen prosesseja, jotka voisivat olla kehityksen kohteena. Prosessiluokkia on kolme: ensimmäinen määrittelee käyttäjän objektien näkymät, toinen kuvastaa tietoa ja toimintoja, jotka käyttäjä tarvitsee tietyissä tehtävissä, ja viimeinen prosessi on olennainen interaktiivisuuden luontiin käyttäjälle. Kehitetävän järjestelmän kannalta kaikki luokat ovat tärkeitä. Luokan mallinnus on kuvassa 5. (Lehtonen – Sinisalo 2007, 9.)



KUVA 5. Suunnitelman prosessit toteutukseen (Lehtonen – Sinisalo 2007, 9.)

Prosessi etenee seuraavasti: Luodaan pääikkuna, ja jos tarvetta, mahdolliset lisänäkymät. Liiallisen tiedon kertyminen yhdelle näkymälle on vältettävä, mutta toisaalta myös lisänäkymien suurta määrä vältettävä, joten mietitään tarkkaan onko tieto tarpeellinen. Muutetaan attribuutit käyttäjän käyttämiksi kontrolleiksi kysymyksellä: miten attribuutti tulisi esittää käyttäjälle kontrollina ja myös silloin, kun sitä ei ole mahdollista käyttää? Tarkistetaan, että käyttäjällä on työkalut tehdä sitä, mitä heidän tarvitsee ohjelmalla milläkin hetkellä suorittaa. Jotta käyttäytyminen olisi hyvin määritelty, turhat ikkunoiden suhteet toisiinsa tulee poistaa. Esimerkkinä, jos sulkee ikkunan A, mitä tapahtuu jo auki olevalle ikkunalle B? Tämä prosessin vaihe nostaa käyttöliittymän käytettävyyttä. Lopuksi tässä prosessissa tulee määritellä onko käyttäjällä mahdollista olla enemmän kuin yksi instanssi auki. Määrittely on tärkeää, koska se vaikuttaa kehittämiseen ja vaikutus näkyy käyttöliittymän monimutkaisuuden vähenemisenä. (Lehtonen – Sinisalo 2007, 9–10.)

3.5 Käytettävyyden arviointi

3.5.1 Menetelmät

Käyttäjä vaikuttaa ohjelman soveltuvuuteen tehtäväänsä. Jos käyttäjää ei tunneta, ei voida tuntea käyttäjän yleisiä toimintatapoja tai arvomaailmaa. Nämä asiat pitää ottaa huomioon

käytettävyyden suunnittelussa ja arvioinnissa unohtamatta käyttöympäristön asettamia vaatimuksia. (Kuparinen 2008, 16–17.)

Käytettävyydestä tehdään yleensä käytettävyystutkimus, joita voidaan tehdä usealla tavalla. Tutkimusten on todettu vähentävän käyttökustannuksia ja käyttövirheitä sekä parantavan tehokkuutta ja nostavan järjestelmän ominaisuuksien käyttöastetta. Esittelen tässä kaksi toisiaan muistuttavaa tapaa, joissa osa-alueet ovat hyvin samanlaisia, mutta testaajat eroavat toisistaan. Syynä tähän myös on se, että molempia tapauksia on tarkoitus käyttää tässä työssä hyödyksi. (Kuparinen 2008, 34–36.)

Käydään aluksi läpi heuristinen arviointi, jota suorittaa käytettävyysasiantuntija tai vaikka suunnittelija itse. Tähän on olemassa avuksi kymmenen säännön lista Nielseniltä ja Molichinilta.

1) Käytä yksinkertaista ja luonnollista dialogia
2) Käytä käyttäjien omaa kieltä
3) Minimoi käyttäjän muistikuorma
4) Tee käyttöliittymästä kauttaaltaan yhdenmukainen
5) Anna käyttäjälle palautetta toiminnoista
6) Anna selkeä poistumistapa eri tiloista ja toiminnoista
7) Anna käyttäjälle mahdollisuus käyttää oikopolkuja
8) Anna virhetilanteista selkeät virheilmoitukset
9) Vältä virhetilanteita
10) Anna riittävä ja selkeä apu ja dokumentaatio
(Riihiaho, 5.)

Tavoitteena on asettua käyttäjän asemaan ja arvioida käyttökokemusta hänen näkökulmastaan. Arvioinneissa tuotteen suunnittelija tai muu asiantuntija arvioi tuotetta pyrkien arvioinnissaan asettumaan todellisen käyttäjän asemaan. Järjestelmän kaikki toiminnot käydään läpi ja samalla koetetaan touteuttaa kaikki mahdolliset käyttötapaukset. Kun suunnittelija itse tekee arvioinnin on mahdollista, että hän saa harhaanjohtavaa dataa, koska ei osaa ajatella samoin kuin uusi käyttäjä. Esimerkkinä voi yksinkertaisesti toimia valikkorakenne, sillä suunnittelija on sen tehnyt ja nähnyt monta kertaa, kun taas uusi käyttäjä ei tiedä, mistä mikin toiminto löytyy. (Kuparinen 2008, 34–36.)

Toisaalta käytettävyyttä voidaan testata järjestämällä mahdollisimman tavallisia käyttötilanteita, joita suorittavat kohderyhmästä valitut henkilöt. Koska suurin osa käytettävyysongelmista tulee esiin vasta todellisissa käyttötilanteissa, käyttämällä tavallisia koehenkilöitä saadaan yleensä parempaa dataa. Tämän tehokkaamman ja suuremman katsannon lisäksi tätä tapaa pidetään kustannustehokkaana saatuun hyötyyn nähden. Tällaisessa testissä voidaan lisänä käyttää ääneen ajattelun metodia, jossa käyttäjiä pyydetään kertomaan ääneen ajatuksensa. Tällöin saadaan käyttäjän ajatusmallista kuva ja ongelmien kuvaukset heidän näkökulmastaan. Lopuksi saadun palautteen yleensä analysoi käytettävyysasiantuntijat ja analyysin jälkeen tehdään tarpeellisia muutoksia ja korjauksia. (Kuparinen 2008, 34–36.)

Lisäksi on olemassa kyselylomakkeita käytettävyyden testaamiseen, mutta ne soveltuvat paremmin täydentämään jo mainittuja keinoja. Parhaiten lomakkeilla voidaan kartoittaa ongelmien yleisyyttä eikä niinkään tarkkaa ongelmaa. (Kuparinen 2008, 34–36.)

3.5.2 Käytettävyyden arvioinnin osa-alueet

Tässä luvussa on jo aiemmin mainittu tiettyjä osa-alueita (käyttäjän virheiden sieto, informatiivisuus jne.), joita hyödynnetään arvioinnissa. Seuraavana kokeilen avata näitä käsitteitä lyhyin kuvauksin. Tässä käsitellyt eivät ole kaikki mahdolliset arvioitavat asiat, mutta aiheen laajuutta ne jo kuvastavat.

Intuitiivisuus

Intuitiivisyys koostuu käytännössä muista käytettävyyden tekijöistä, ja jos nämä ominaisuudet on toteutettu hyvin, se johtaa itsessään hyvään intuitiivisyyteen. Käytettävyyden ollessa erinomainen, järjestelmän käyttämisen voi parhaassa tapauksessa opetella itse, jolloin koulutusta tai ohjekirjaa ei tarvita. Tämä ei tietenkään poista näiden tarvetta, mutta vähentää käyttäjän turhautumista, koska ei tarvitse aina tukeutua ohjeisiin. (Kuparinen 2008, 17–18.)

Luotettavuus ja virheettömyys

Luotettavuus ja virheettömyys käsitetetään alitajuisesti itsestään selviksi perusasioiksi. Puutteellisuuden näissä huomaa aina, mutta ei välttämättä tule ajatelleeksi asiaa ollenkaan, kun

ne ovat kohdallaan. Toisin sanoen ohjelman tulee olla teknisesti toimiva. Käynnistys ja latautuminen ei saa kestää liian kauan, tuhlata tai sitoa tietokoneen resursseja haitallisesti ja virheitä pitäisi olla mahdollisimman vähän. Nämä seikat edesauttavat käyttäjätyytyväisyyttä hyvin suurella painoarvolla ja lisäksi taloudellisuutta, joka myös on käytettävyyden osa-alue. Käyttäjän aika ei kulu teknisten heikkouksien kanssa kamppailuun tai tukipalveluissa, jolloin itse työnteolle jää enemmän aikaa. (Kuparinen 2008, 18.)

Esimerkkinä kamppailusta voisi olla tekstinkäsittelyohjelma, joka kaatuu vähän väliä ja samalla korruptoi auki olleen tiedoston. Tästä seuraa, että tehokas ja tarkoituksenmukainen käyttö vähenee, kun käyttäjän aika menee varmuuskopioiden ottamiseen ja jatkuvaan tallenteluun, jotta vältetään suuremmat tietojen menetykset. Tämä tottakai karsii käyttäjän luottamusta ja saa hänet etsimään parempaa vaihtoehtoa. (Kuparinen 2008, 31–32.)

Käyttäjän virheiden sieto

Järjestelmän pitää sietää käyttäjän tekemiä virheitä ja odottamatonta toimintaa. Ohjelman olisi hyvä opastaa käyttäjää aina oikeaan suuntaan ja vaikka käyttäjä poikkeaisi normaalista toiminnasta esimerkiksi täyttämällä kysymyskenttiä väärässä järjestyksessä, se ei saisi kaatua. Ihmiset tulevat aina tekemään virheitä, mutta määrä vähenee, jos laite on helppokäyttöinen. Hyvä järjestelmä voi auttaa virheiden välttämisessä, esimerkiksi tutuilla kysymyksillä haluatko varmasti poistaa jotain tai haluatko varmasti sulkea ohjelman tallentamatta muutoksia. Toisaalta tässäkään ei saa mennä liiallisuuksiin, jolloin käyttäjä voi vähintäänkin ärsyyntyä.

Kuparinen mainitsee esimerkkinä Microsoftin Windows Vista -käyttöjärjestelmän, joka usein ohjelmia ja järjestelmäasetuksia avattaessa pysäyttää toimintansa, kunnes käyttäjä antaa vastauksen ”haluatko varmasti” -tyyppiseen kysymykseen. Tähän totean todenneeni saman, ja pysähtelyn haittaa vähentäisi jo huomattavasti pelkästään se, että pysäytys koskisi vain kyseistä ohjelmaa, eikä koko tietokonetta. (Kuparinen 2008, 19.)

Informatiivisuus

Informatiivinen ja selkeä järjestelmä ei jätä käyttäjälle epäselvyyttä toimintatavoistaan. Pahimmillaan kryptinen varoitus järjestelmältä, esimerkiksi ”järjestelmä sammui virhekoodilla -

217535”, voi aiheuttaa vaaratilanteita tai ihmishenkien menetyksiä (ydinvoimala, lääketeollisuus). Selkeät ja lyhyet ilmoitukset helpottavat tilanteen hahmottamista ja ymmärtämistä. Jos virheen ymmärtäminen vaatii suurempaa tietoa, lyhyen kuvauksen lisänä voi olla linkki lisäinformaatioon. Ymmärrettävyyttä edistetään muun muassa välttämällä vierasta- ja ammattikieltä, mutta tätä käsitellään erikseen myöhemmin. (Kuparinen 2008, 19–20.)

Käyttäjä voi myös turhautua, jos järjestelmä haluaa tietoja käyttäjältä ilman syytä eli käyttäjä ei näe tarpeellisuutta tai merkitystä asiassa. Käyttäjä kaipaa myös tietoa tehtävien ja tapahtumien edistymisestä. Esimerkiksi latauspalkin kasvaminen viestii tämän. Tällöin ihminen voi seurata tilannetta ja tehtävän valmistuttua vapauttaa oman muistinsa asiasta. (Kuparinen 2008, 19–20.)

Muistin kuormittavuus

Ihmisellä on kolmenlaista muistia: pitkäkestoista eli säilömuistia, lyhytkestoista eli työmuistia sekä aistimuistia, johon tallentuu aistien kautta saatu informaatio. Lyhytkestoinen muisti on oleellisin, kun puhutaan käytettävyydestä. Lyhytkestoista muistia hyödynnetään ohjelman peruskäytössä ja erityisesti uuden ohjelman opetteluvaiheessa. Lyhytkestoiseen muistiin tallentuvat muun muassa käyttöliittymässä näkyvät objektit ja niiden käyttötavat. Myös silloin, kun ohjelma on jo tuttu ja asiat ovat pitkäkestoisessa muistissa, sen käyttöönotto vaatii asian tuomista ensin takaisin lyhytkestoiseen muistiin. Käytettävyydeltään hyvä tuote ei kuormita käyttäjän muistia liiaksi. (Kuparinen 2008, 20–22.)

Kuparisen teksissä viitataan tutkimukseen jossa todetaan, että ihminen pystyy kerralla muistamaan 5–9 asiaa. Käyttöliittymässä ei siis kannata ylittää tätä arvoa jos mahdollista.
Jos käyttäjän muistia ei kuormiteta liiaksi helpotetaan samalla myös järjestelmän opittavuutta. Tähän vaikuttaa lisäksi yksinkertaisuus ja ymmärrettävyys, kuten elementtien sijoittelu muualta tutuille paikoille tai yleiseen loogiseen järjestykseen ja ymmärrettävä kielen ja termien käyttö. Toinen käsitelty tutkimus toteaa todeksi vanhan sanonnan ”yksi kuva on enemmän kuin 1000 sanaa”. Ihminen muistaa graafisen datan helpommin kuin tekstin. Ikonien ja symoblien käyttö on siis suotavaa ja myös oleellistä graafisissa käyttöliittymissä. Esimerkkiksi roskakori tarkoittaa asioiden poistamista tulevaisuudessa, kun se tyhjennetään. Sieltä voi kuitenkin vielä palauttaa asian, jos se ei kuulu sinne. (Kuparinen 2008, 20–22.)

Värit, sijoittelu, koko, muoto ja ääni

Käyttöliittymien värit tulisi valita hyvän kontrastin mukaan, mutta välttää ääripäitä ja lisäksi kannattaa muistaa näkörajoitukset, kuten puna-vihervärisokeus. Myös yleiset käytännöt kannattaa muistaa, kuten punainen mielletään yleensä varoittavana tai vaaran värinä ja nykyaikana tietokoneiden kautta sininen mielletään linkiksi tai lisäinfon mahdollisuudeksi. Liiallistä värien käyttöä kannattaa välttää, koska se vaatii liikaa huomiota ja tekee asian mahdollisesti sekavaksi. Esimerkiksi tekstissä valkoinen-musta on paras yhdistelmä. Huonot värivalinnat voivat ärsyyntymisen lisäksi aiheuttaa epämukavuutta, päänsärkyä ja pahoinvointia. (Kuparinen 2008, 23–24.)

Myös toimintojen ja ikoneiden paikat kannattaa pitää samanlaisina eri näkymissä. Esimerkiksi Windows-sovelluksissa on totuttu valikkoriviin ohjelman yläreunassa. Käyttöliittymäsuunnittelussa kannattaa huomioida myös tekstin koko, jotta se on oletuksena tarpeeksi suuri kaikille ja kuvakkeet tai ikonit ovat tarpeeksi suuria käytön helpottamiseksi. Lisäksi ääntä voidaan käyttää merkkinä jostakin, ja yksi yleinen tapaus on virheilmoitus. Käyttäjillä on taipumus reagoida äänisignaaliin nopeammin kuin visuaaliseen, mutta silti ääniä on käytettävä säästeliäästi, sillä se on myös häiriötekijä. (Kuparinen 2008, 23–24.)

Kosketusnäytöissä tulee ensisijaisesti huomoida käyttöliittymän koko, jotta kaikkia kontrollit ovat tarpeeksi suuria isosormisillekin. Ulkoasulla, muotoilulla ja väreillä voidaan ohjata ihmisiä toimintoihin ja pitämällä ulkoasussa samankaltaisuuksia tuoteperheissä saadaan luotua mielikuva valmistajasta. Tällä saadaan kilpailuetua jos käytettävyys on onnistunut. Vanhat mielikuvat jostain merkistä vaikuttavat jo ennen uuden hankkimista positiivisesti tai negatiivisesti.

Helppokäyttöisyyden luominen voi mennä pieleen jos yksinkertaistetaan väärin, esimerkiksi käytetään liikaa valikoita. Halu saada isot painikkeet on pahasta, jos alivalikoiden määrä voitaisiin supistaa neljästä kahteen lisäämällä 1 tai 2 päävalikkoa ja tinkimällä hieman painikekoosta. Varsinkin perustoiminnot pitää saada käyttöön nopeasti ja helposti, kun taas harvemmin vaihdettavat asetukset voivat olla alivalikon takana.

Käyttöohjeet ja tuen saatavuus

Ideaalisesti järjestelmä olisi niin helppo opetella itse, ettei se tarvitsisi ollenkaan ohjeita, opastusta tai koulutusta. Lisäksi se olisi niin virheetön, ettei tarvetta tukipalvelutarpeita olisi. Käytännössä tämä on kuitenkin melko mahdoton toteuttaa. (Kuparinen 2008, 25–26.)

Nykyisellään varsinaiset ohjekirjat ovat ainakin osittain jo kadonneet, vaikka hyvin tehty ohjekirja omasta mielestäni ratkaisee yleisimmät ongelmat parhaiten. Neuvot ja ohjeet ovat siirtyneet enemmän ohjelmien sisälle, internettiin, sähköposteihin ja puhelintukeen. Näitä kautta on hankala kuvailla juuri tapahtunut virhe, kun manuaalissa voisi olla tilanteesta hyvä kuva selventämässä. Joskus toki etähallinnan kautta saa parasta apua ja puhelinpalveluissa voi keskustella suoraan ihmiselle, mikä on monille helpompaa kuin ohjelman tarjoamien vinkkien seuraaminen. Ratkaisu voi sisältyä niihin, mutta mentaalinen malli ei ole samanlainen käyttäjän kanssa. Ironisimpana esimerkkinä voin mainita omasta kokemuksesta PSOASin vikailmoitukset, joista pitää tehdä sähköposti-ilmoitus, myös kun kyseessä on internetliittymän ongelma. (Kuparinen 2008, 25–26.)

Kieli ja termistö

Ohjelmassa käytetty kieli ei saisi olla esteenä ohjelman käyttämiselle. Vaikka yleensä saatavilla on erikielisiä versioita ohjelmista, silti vastaan tulee ohjelmia, joissa ainoa käyttökieli on englanti. Silti käännöksissä usein joitain virheilmoituksia ym. jää kääntämättä. Tämän lisäksi termistössä on monesti englannin kieltä joka tapauksessa. Tämän takia komentojen ja valikoiden yhdenmukaisuus korostuu, jolloin ohjelmaa voi käyttää ainakin osittain vaikka kielitaitoa ei olisikaan. Toiminnot pitää kuvata ymmärrettävästi ja välttää alakohtaisia sanoja ja lyhenteitä, ellei ohjelma ole juuri sen alan ongelmiin tehty. Esimerkiksi IT-alan ammattitermistö on tyypillisesti ollut varsin rikas, ja siinä vilisee lyhenteitä. Alan koko sanastoa ei hallitse yksikään IT-osaaja. Toisena ääripäänä tutkielmassa on esitetty sukututkimukseen tehty ohjelma, jolloin on sallittua käyttää sen omaa erikoissanastoa, koska käyttäjä tuntee käsitteet. Silloin tällöin ammattisanat saattavat levitä yleiseen käyttöön, jolloin niitä voidaan käyttää (esim. Internet, modeemi, sähköposti). (Kuparinen 2008, 26–27.)

Psykologian kannalta ajateltuna tiedetään, että kun ihminen kohtaa vierasta kieltä tai termejä, joita hän ei ymmärrä, hän herkästi jättää asian huomiomatta ja asia jää ymmärtämättä. Tämän välttämiseksi pitää pyrkiä käyttämään vain käyttäjälle tuttua kieltä sekä termejä, jotka ovat käyttäjän ymmärrettävissä. Jos on pakkottava tarve käyttää yleiskielestä poikkeavia termejä, on aina pyrittävä käyttämään edes termejä, jotka ovat samantyyppisissä tilanteissa ja ohjelmissa muutenkin käytössä. (Kuparinen 2008, 27–28.)

Yhteensopivuus

Jos mahdollista, järjestelmien tulisi olla yhteensopivia muiden markkinoilla olevien järjestelmien kanssa esimerkiksi tallennusmuodoiltaan, jolloin dataa voidaan siirtää järjestelmästä toiseen. Tätä edistää standardien ja vakiintuneiden käytänteiden noudattaminen. Toisaalta joskus tarvitaan niin suppean alan järjestelmiä, että on turhaa käyttää aikaa yhteensopivuuden aikaansaamiseksi, jos sille ei ole mitään syytä. Samalla säästetään jopa laitteen resursseja, kun ohjelma voidaan tehdä kevyemmäksi. (Kuparinen 2008, 28.)

Käytön miellyttävyys ja esteettisyys

Käytettävyys tavallisesti johtaa käytön miellyttävyyteen, mutta se voidaan ajatella myös omaksi käytettävyyden osa-alueeksi. Käytön miellyttävyyteen voidaan sisällyttää myös esteettisyys. Helppokäyttöinen ja ergonominen järjestelmä on emotionaalisesti miellyttävä, mutta nämä ominaisuudet eivät vielä yksin riitä käyttäjän mielihyvän maksimoimiseen. Emotionaalisesti täysin miellyttävässä järjestelmässä käyttöliittymä on myös esteettisesti kaunis. Ihmiset kokevat positiivisia tunteita (innokkuus, tyytyväisyys, mukaansatempaavuus, mielenkiintoisuus, turvallisuus ja rentous) käyttäessään käyttöliittymältään esteettisesti kaunista järjestelmää. Tutkimusten mukaan käyttöliittymän esteettisyys vaikuttaa suoraan myös käyttäjien näkemykseen järjestelmän helppokäyttöisyydestä ja käyttäjät ovat taipuvaisia luottamaan järjestelmän esteettisesti miellyttävään ulkoasuun. Samaisen tutkimuksen mukaan käyttäjät ajattelevat laadukkaan ulkoasun suunnittelijoiden osanneen suunnitella myös muun palvelun laadukkaasti. (Kuparinen 2008, 28–29.)

4 AJANVARAUSJÄRJESTELMÄ

4.1 Lähtökohdat

Projektissa visioitiin suunniteltavaksi järjestelmä, jonka tarkoitus on houkutella uusia asiakkaita ja antaa hyvä käyttökokemus ja -mukavuus. Houkutelevina tekijöinä ovat suuri koko näytöltä ja sopivat mainokset ja lausahdukset. Näytön kooksi kaavailtiin minimissään 42-tuumaista laajakuvatelevisiota. Yksinkertaisuudessaan muutamalla kosketuksella saisi tietoa, mitä kaikkea on tarjolla ja mitä nämä palvelut sisältävät. Esimerkiksi täytettäviä kohtia ei saa olla liiaksi ja tarjontajärjestyksellä on suuri merkitys. Niinpä rakenteessa päädyttiin suureen etusivuun eri palveluiden kera. Palvelun valinnan jälkeen saa lyhyen kuvauksen sisällöstä ja kalenterin tarkasteltavaksi. Aikojahan on hyvä tarjota alkuun, koska epäröivä voi päättää kokeilla palvelua helpommin, jos sopiva aika onkin heti tarjolla. Vasta tämän jälkeen pyydetään mahdolliset tarvittavat henkilö- ja esitiedot. Myös muita mainoksia voi esittää samalla, sillä näytön koko on tarpeeksi iso ja näin myös huomioidaan yksityisyyttä, koska keholla voi peittää kohtalaisesti henkilötiedot. Toisaalta järjestelmän pitäisi olla helposti asennettava ja ylläpidettävä, jottei vähemmällä teknisellä osaamisella tulisi ongelmia. Tavoitteena oli siis mahdollisimman vähän asetusvaihtelua ja lisäohjelmia.

Näyttävyyden pohjalta päädyttiin käyttämään Qt:tä, koska graafisen ulkoasun luonti olisi helppoa ja muutokset kohtalaisesti tehtävissä. Myös pienet animoinnit voisivat olla mahdollisia aikataulun puitteissa. Päävalikkoon taustaksi valittiin jo käytössä oleva mainostaulu, koska sen ulkoasu soveltui sellaisenaan valikoksi. Siinä oli sopivan yksinkertainen mainos, johon vain lisätään kosketusominaisuus (kuvan 6 tausta). Muu laittesto koostuu PC:stä ja Windows-käyttöjärjestelmästä, koska suurin osa nykyihmistä osaa käyttää näitä ja ylläpito helpottuu. Tekstikenttien täyttö tapahtuu normaalin näppäimistön tai virtuaalinäppäimistön kautta.

Tietokannan käsittelyyn käytetään MySQL-tietokannan hallintaohjelmistoa ja WAMP-ohjelmistoa paikallisena palvelimena. Ohjelmiston asentaminen ei ole vaikeaa ja tarpeen vaatiessa perusasetuksien muutokset ovat myös kohtuudella tehtävissä. Tietokantaan ei voida linkittää jo käyttössä olevia kalentereita, koska yritys käyttää normaaleja paperikalentereita organisointiin, joten kannan suurin merkitys on vain ehkäistä päällekkäiset varaukset. Joka tapauksessa

varaukset pitää käydä läpi ja sovitella aikatauluihin ottamalla yhteys asiakkaaseen. Tästä syystä aikoihin otettiin minimissään kolmen arkipäivän etäisyys kuluvasta päivästä.

4.2 Työn aloitus

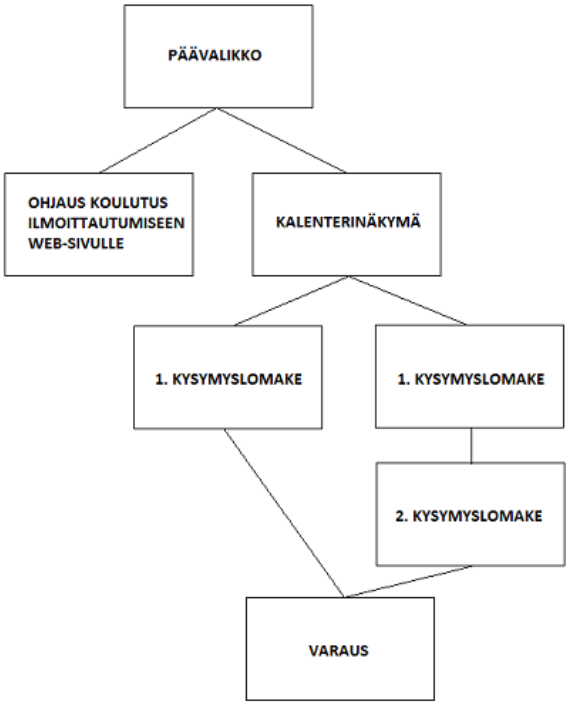
Alkuvaiheessa pidettiin muutamia palavereita ja näiden pohjalta saatiin suunta tavoitteiden toteuttamiselle ja ohjelmalle asetettavat vaatimukset. Vaatimuksia täydennettiin projektin aikana ja samalla ilmeni joitain väärinymmärryksiä aiemmin esitetyistä vaatimuksista ja ominaisuuksista. Lisäksi tein selvitystä Qt:n ominaisuuksista ja mahdollisuuksista ja sen yhteensopivuudesta MySQL:n kanssa, sillä Qt oli suuremmassa mittakaavassa uusi tuttavuus. Totesin kaiken kuitenkin mahdolliseksi toteuttaa näillä työkaluilla ja projekti pääsi eteenpäin. Lopulta MySQL:n käyttö osoittautui oletettua hankalammaksi. Tätä selvennän luvuissa 4.7 Tietokanta ja 5 Jatkokehitys.

4.3 Päävalikko

Ensimmäisenä suunnittelin päävalikon. Vaatimuksissa määriteltiin päävalikolle (kuva 6) kuusi valintaa, jotka ovat Kehon koostumusanalyysi, Kehon ikä - Mittaus, Kuntosaliohjelmat, Personal trainer tapaaminen, Ravintovalmennus ja Ilmoittaudu koulutukseen. Koska valintoja oli yhtä monta kuin eräässä yrityksen käytössä olevassa suuressa mainostaulussa, päädyttiin käyttämään samaa ulkoasua pohjana ohjelmassa. Jokaiselle valinnalle tulee oma painike, joihin saa pienen aiheeseen liittyvän kuvan. Kun kuvaa käytetään painikkeena, painikkeen koko kasvaa, ja tämä onkin suotavaa kosketusnäytöllä. Alusta asti omana ideana oli, että pieni animointi toisi käyttömukavuutta ja palautetta käyttäjälle toiminnoista, mutta tähän ei ollut lopulta aikaa, joten se voi jäädä kehitysmahdollisuudeksi.

Kuva 7 esittää ohjelman toiminnan rakennetta hieman yksinkertaistettuna. Päävalikosta on kaksi etenemistietä: avataan koulutukseen ilmoittautuminen web-selaimeen tai avataan kalenteri ajanvaraukseen. Ajanvarauksesta edetään valitun palvelun kysymyksiin, joissa on eroja palvelun mukaan, ja lopulta tehdään varsinainen varaus.

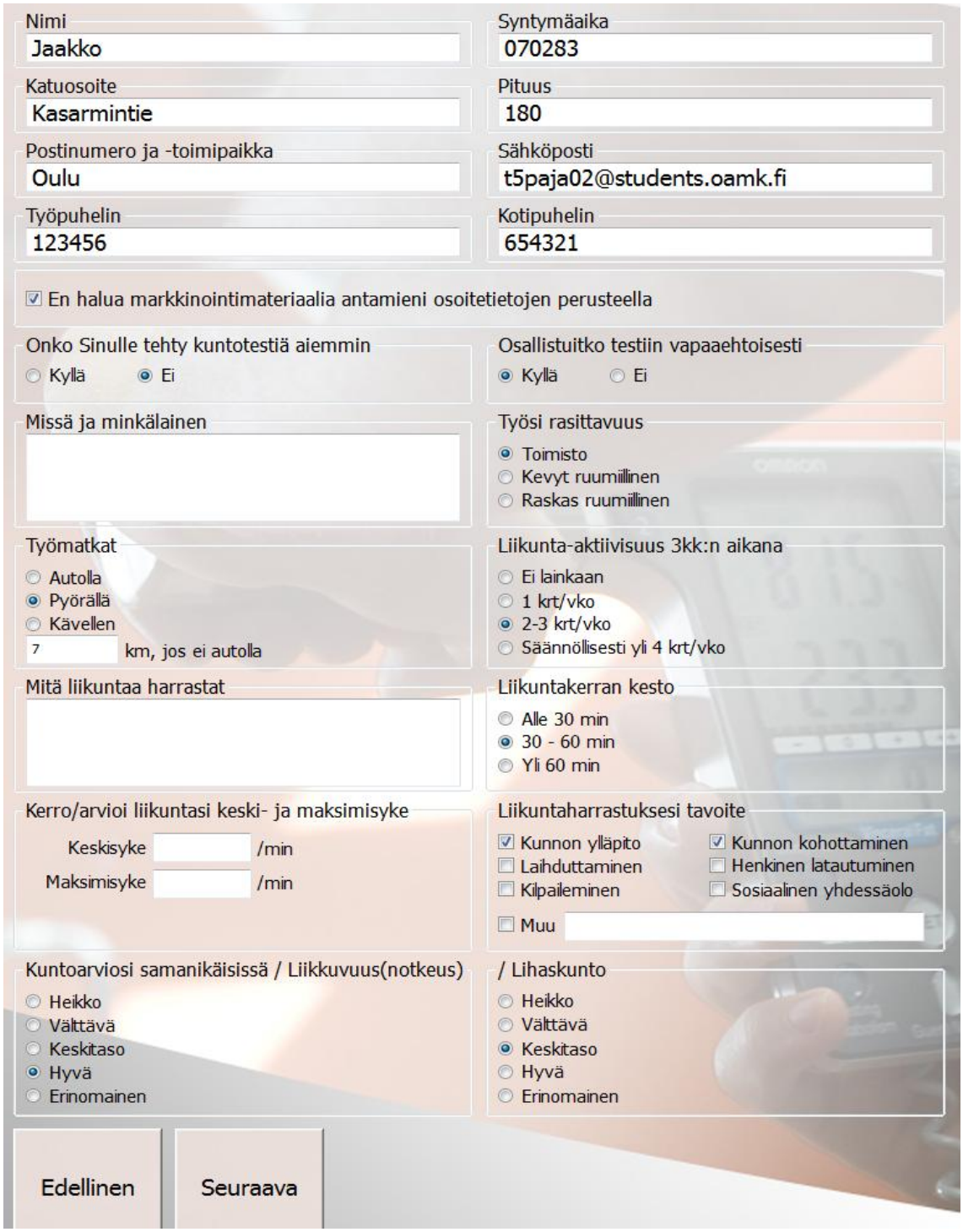
KUVA 6. Päävalikko



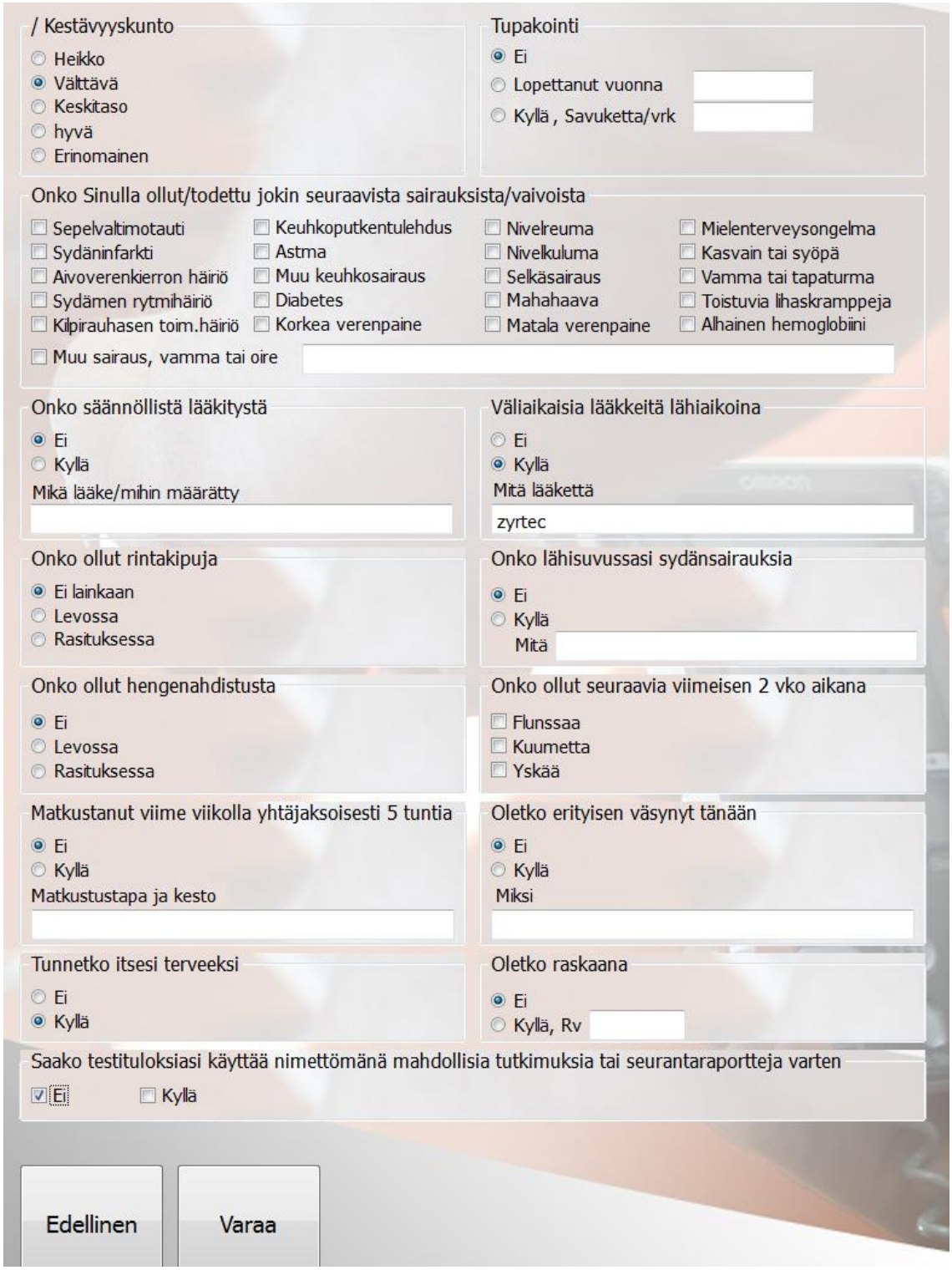
KUVA 7. Ohjelman etenemisen rakenne

4.4 Palveluvalikot

Seuraavassa vaiheessa siirryin kehittelemään peruspohjaratkaisua (kuva 7 ja 8) palveluille. Jonkin verran aikaa tuhlaantui esitietolomakkeiden saantiin tilaajayritykseltä, mutta osan ajasta pystyin käyttämään pohjan suunnitteluun ja täyttämään kysymykset myöhemmin.



KUVA 7. Varauslomake kehon koostumusanalyysiin s. 1/2



KUVA 8. Varauslomake kehon koostumusanalyysiin s. 2/2

Perusrakenteen malli on seuraava: näytön vasen puoli käytetään kysymyksiin ja oikea puoli ohjeisiin ja mahdollinen lopputila mainoksille. Rakenteeseen ladataan ensimmäisissä viidessä palvelussa oikeat kysymykset ja kuvaukset. Jos esitietokysymyksiä on paljon, ne jatkuvat uudelle sivulle, jonka pohjarakenne on samanlainen. Kummassakin tapauksessa tarkistetaan, että

tarpeelliset tiedot on täytetty, kun lopullista varausnappia painetaan. Kaikissa vaiheissa on myös paluupainike, jolla voidaan palata yksi askel taaksepäin lomakkeissa tai vaikka päävalikkoon asti. Erona on 6. palvelu eli koulutus, jonka valinnasta avataan yrityksen web-sivusto koulutuksen kohdalta. Tämä poikkeus tehtiin siksi, että prosessi koulutukseen on erilainen ja se tehdään jo olemassaolevien työkalujen ja mekaniikoiden kautta.

Teknisesti ratkaisu toimii siten, että kalenteri antaa lomakenäkymän luonnin yhteydessä tiedon, mitä palvelua halutaan. Lomakenäkymä rakennetaan tämän mukaan joko yhdeksi lomakkeeksi tai tarpeen vaatiessa luodaan seuraava-painike, joka käytännössä luo uuden lomakkeen. Kaikki tiedot kerätään QStringList-luokan (kuva 9) muuttujaan ja se välitetään varauksen tekoon tai seuraavalle lomakkeelle, jossa sitä täydennetään ja sieltä varauksen tekoon.

Kuvassa 9 nähdään yksityisen muutujan luonti Forms-luokalle tiedostossa Forms.h rivi 22. Tähän taulukkoon lisätään henkilön nimi, jos kenttä ei ole tyhjä (rivit 113–116), ja riveiltä 892–897 nähdään uuden lomakkeen luonti ja sen saamat parametrit. Toinen lomake saa siis yläluokan osoittimen (this), palvelun numeron (number) ja taulukon (emaildata). Tällainen käsittely on jokaiselle tiedolle. Lisäksi kuvassa on pakollisten kenttien tarkastus eli jos yksikin pakollinen tieto on tyhjä, valmis-muuttuja muutetaan epätodeksi (rivit 109–112) ja ohjelma ei etene seuraavaan lomakkeeseen, vaan näytetään muistutus pakollisista kentistä (rivit 898–902).

```
 22      QStringList emaildata;
100      bool valmis = true;
101      if(number == 1)
102      {
104          while(emaildata.size() > 2)
105          {
106              emaildata.removeLast();
107          }
108
109          if(ui->nimi->toPlainText().isEmpty() == true)
110          {
111              valmis = false;
112          }
113          else
114          {
115              emaildata.append(ui->nimi->toPlainText());
116          }
892          if(valmis == true)
893          {
894              Forms2 *second = new Forms2(this, number, emaildata);
895              second->show();
896              repaint();
897          }
898          else
899          {
900              QMessageBox msgBox;
901              msgBox.setText("Pakolliset kentät pitää täyttää!");
902              msgBox.exec();
```

KUVA 9. Esimerkki tietojen tarkistuksesta ja keräyksestä QStringList-luokan muuttujaan

Osittaiseksi ongelmaksi muodostui Qt Designerissa olevien pohjasijoittelumallien käyttö, sillä lopulta sisäkkäisiä ratkaisuja tuli tehtyä paljon hyvän ulkoasun saavuttamiseksi. Toisena ongelmana ilmeni Qt Designerin omat virhetilat. Jos siirsi yhden elementin toisaalle, koko näkymän alue venähti yli näyttörajojen ilman, että vierityspalikat ilmestyivät näytölle tai pystyi muuttamaan asetuksista kokoa oikeaksi. Yleensä peruutus tai ohjelmointiympäristön uudelleenkäynnistys pelasti tilanteen, mutta kerran tai kaksi jouduin aloittamaan alusta varmuuskopioista. Lisäksi olin PHP-kielen kanssa työskennellessä unohtanut, että C++ pohjaiset kielet ovat hyvin tarkkoja tietotyypeistä ja piti soveltaa erinäisiä muuntolauseita, joita Qt:ssä on paljon, jotta kaikki tiedot saatiin QStringListiin talteen.

4.5 Varaustoiminto

Varauksen tekeminen ei itsessään näy asiakkaalle juuri mitenkään, lukuunottamatta kuittausviestiä varauksesta, joten sillä ei ole ulkoasukuvausta. Kuittausviesti myös nollaantuu itsestään 10 sekunnin kuluessa ja ohjelma palaa päävalikkoon. Tällä taataan, että käyttäjän vastaukset tai joissain tapauksissa jopa nimi eivät jää muiden nähtäville. Samalla voi keskittyä lukemaan mahdollisia valmistautumisohjeita, jotka avautuvat samaan aikaan.

Toteutusidea tähän oli hyvin yksinkertainen, mutta lopulta ei yhtä helppo toteuttaa. Kun kaikki tarvittava data on kerätty ja tarkistettu, QStringList-luokan taulukko kootaan yhdeksi pitkäksi merkkijonoksi valitun palvelun perusteella ja välitetään Email-oliolle. Email-olio hoitaa varsinaisen lähetysvaiheen. Samalla, kun varausnappia on painettu ja tämä toiminto käynnistyy, ohjelma käynnistää Adobe Acrobat Readerin ja avaa siihen varattavaa palvelua vastaavan valmistautumisohjeen, jos sellainen on tarpeen. Viimeisenä tehdään merkintä tietokantaan varatusta ajasta, jolloin muut eivät voi varata tismalleen samaa aikaa.

Järjestelmä käyttää yksinkertaisinta ratkaisua sähköpostin lähetykseen eli TCP-pohjaista SMTP:tä, jolla voi vain lähettää viestejä. Sähköpostin lähetykseen vaadittavat palvelin ja muut tiedot ovat tässä vaiheessa ”hard coded”, joten suosittelen jatkokehitystä, jonka aikana tehdään asetuksille oma valikko. Ongelmia aiheutuu, jos järjestelmän käyttämä ISP (internet service provider eli internetpalvelujen tarjoaja) vaihtuu, koska nämä koodirivit pitää korjata ja kääntää ohjelma uudelleen. Itse viestin rakentamiseen oli kaksi vaihtoehtoa. Tässä tapauksessa päätin käydä taulukon läpi manuaalisesti jokaiselta kohdalta. Vaihtoehtona olisi silmukka, joka pyörisi niin kauan että taulukko tyhjenisi. Jälkimmäinen sietää virheitä paremmin, mutta postista tulisi sekavampi. Toiminnan rakentaminen siis kesti kauemmin, mutta tällä saadaan selkeyttä viestiin ja lopputulos on paremmin luettavissa. Vastaamattomat tai tarpeettomat kohdat pitää tämän takia merkata jotenkin, ja valitsin (-)-merkin tähän. Lopuksi kutsutaan kalenterin tuhoajaa, jolloin päävalikko ilmestyy näkyviin ja kaikki muut vaiheet on suljettu (kalenteri rakennettiin myöhemmässä kehitysvaiheessa).

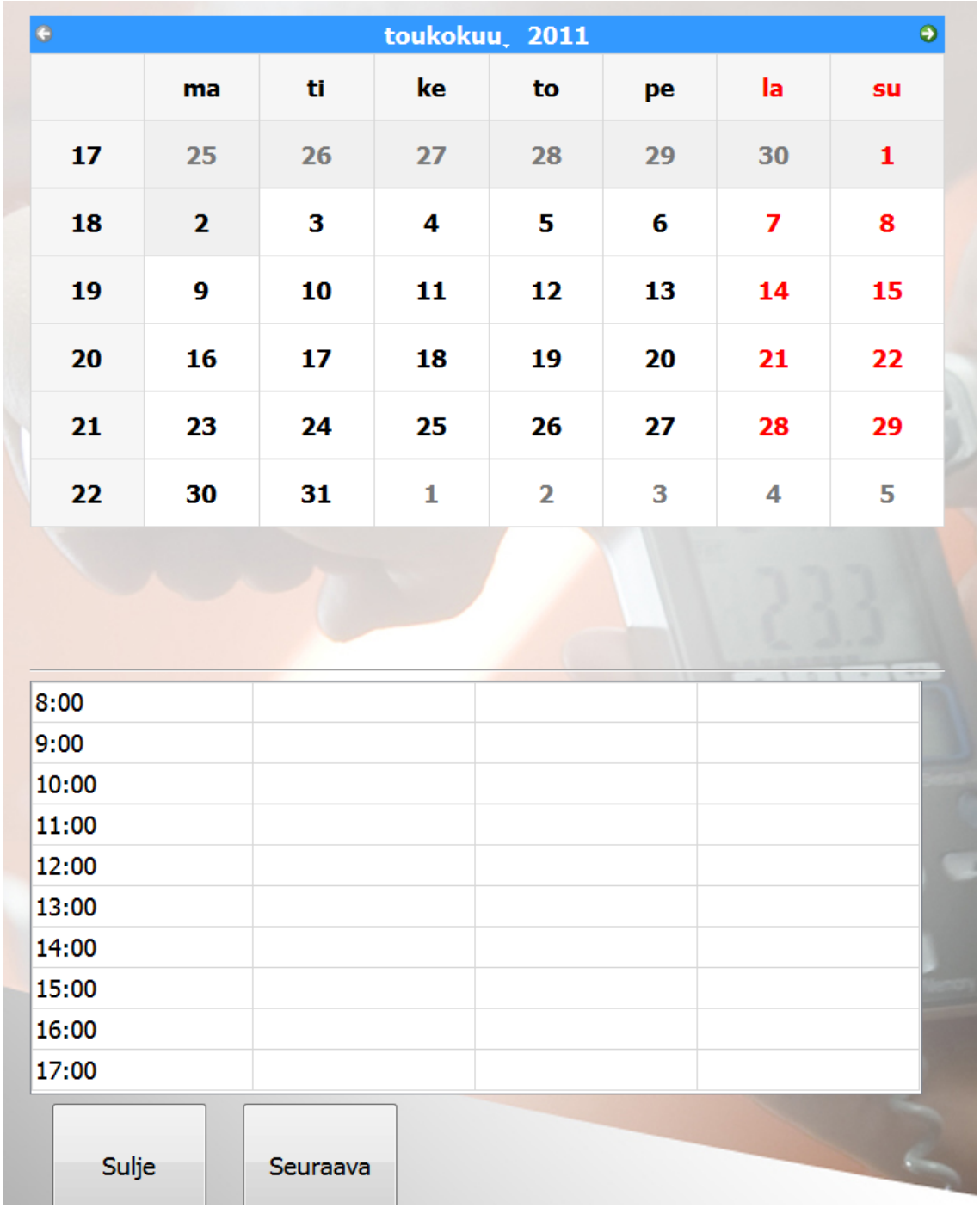
Sähköpostin lähettäminen oli yksi suurimmista ongelmista koko projektista. Ensinnäkin aiheesta oli huonosti malleja ja ohjeita saatavilla ja dokumentaatiostakaan ei ollut suurta apua. Tuorein malliesimerkki oli Qt 3:n ajoilta ja käytössä oli Qt 4. Tästä johtuen esimerkin käyttämien luokkien muuttuessa se ei enää toiminut uudessa versiossa sellaisenaan. Tämän pohjan muokkaus vei paljon aikaa ja sitä tuhlaantui turhiin korjailuihin. Lopulta selvisi nimittäin, että tämän mallinen ratkaisu oli asynkroninen ja johti siihen, että järjestelmä suoritti käskyn nopeammin, kuin sen suoritus kokonaisuudessaan vaatisi, ja päädyttiin virhetilanteeseen. Virhe kokonaisuudessaan tapahtui seuraavasti: ohjelma luo Email-olion, jonka tehtävänä on viestinlähetys, ja se avaa yhteyden palvelimelle ja antaa viestin sen välitettäväksi. Tämä vei aikaa tilanteesta riippuen sekunnista muutamaan, kun taas ohjelma suoritti koko työn paljon nopeammin ja palasi

pääohjelmaan suorittamaan seuraavaa riviä, jossa poistetaan Email-olio. Kun olio olikin jo tuhottu ja palvelin ei vielä ollut saanut koko viestiä, viestiä ei koskaan lähetetty.

Ongelma korjataan viiveellä, jotta viesti ehtii perille asti, ja vasta sitten poistetaan olio. Jostain syystä oliota ei voitu poistaa operaation jälkeen ilman, että ohjelma kaatuisi, joten olio jätettiin muistiin ja luotetaan Qt:n omaan roskankeräimeen muistin vapautuksessa. Varauksen suorittamisen jälkeen palataan päävalikkoon sulkien kalenteri, jolloin Qt:n pitäisi poistaa olio automaattisesti. Ideaalitapauksessa toiminnolle tehtäisiin oma suorittava säie, jolloin ongelman pitäisi poistua, mutta tämän toteutukselle ei ollut aikaa tietokantaongelmien vuoksi.

4.6 Kalenteri

Kalenterin (kuva 10) toteuttaminen jäi loppupuolelle, vaikkakin sen paikaksi tarkentui ilmestyminen heti päävalikon jälkeen. Tällaiseen ratkaisuun päädyttiin käytettävyyden takia: on mukavampaa selata sopivia aikoja aluksi ja vasta sen jälkeen vastata kysymyksiin. Tämä saattaa myös houkuttaa epävarman asiakkaan varaamaan ajan, kun sopiva aika onkin heti tarjolla.



KUVA 10. Ajanvarausksen kalenteri ja kellonajan valinta

Näkymä on yhdenmukainen lomakkeiden kanssa eli vasemmalle puolelle tulee kalenteri ja kellonajan valinta, kun oikealla on tarjotun palvelun lyhyt kuvaus ja mahdollinen mainos. Kalenterin alapuolelle näkymään sijoitetaan taulukko, jossa on kellonajat joille aikoja voi varata. Sopivan päivän ja ajan valinnan jälkeen painetaan Seuraava-painiketta ja päästään eteenpäin.

Pohjaratkaisusta päästiin nopeasti muokkaamaan kalenterin ulkoasua, josta ensimmäiseksi vaihdettiin viikon rakenne amerikkalaisesta suomalaiseksi eli maanantaista sunnuntaihin, sitten rajattiin kuluneet päivät pois valinnosta. Myös seuraavat kolme arkipäivää rajataan pois ja viikonloput myös piti suodattaa pois, jotta kalenterit ehditään yhdenmukaistaa. Tietokanta tulee mukaan kuvaan, ja aina kun asiakas valitsee jonkin päivän kalenterista, haetaan palvelua vastaavasta taulusta sen päivän varatut ajat. Jos niitä on, nämä ajat rajataan pois valinnoista päällekkäisyyksien välttämiseksi. Varatut ajat näkyvät punaisella tekstillä ja vaaleammalla sävyllä Windowsin tapaan, kun jotain asiaa ei voi enää valita. Jokainen aika sijoitetaan taulukon soluun ja aikojen määrä eli pituus vaihtelee palvelun mukaan. Päivämäärä ja aika välitetään eteenpäin lomakkeille QStringList-muuttujassa. Tässä hyödynnetään erittäin näkyvästi Qt:n ”viesti ja paikka” -ominaisuutta (kuva 11).

```
25      QObject::connect(ui->timetable, SIGNAL(itemClicked(QTableWidgetItem*)), this, SLOT(time_selected()));
26      //QObject::connect(ui->kalenteri, SIGNAL(clicked(QDate)), this, SLOT(reset()));
27      QObject::connect(ui->kalenteri, SIGNAL(selectionChanged()), this, SLOT(date_changed()));
249 void Appointment::date_changed()
250 {
251     QDate temp_date = ui->kalenteri->selectedDate();
252     QStringList varatut;
253     int j = 0;
254     int k = 0;
255     QTableWidgetItem *temp;
256
257     if (QSqlDatabase::database().open())
258     {
259         QSqlQuery query;
260         query.exec("select * from koostumus where date='" + temp_date.toString("yyyy-MM-dd") + "'");
261         // iteraattori aloittaa aina ensimmäisestä edeltävästä paikasta, joten next osuu ensimmäiseen oikeaan arvoon
262         // palautttaa falsen kun uutta tietoa ei enään ole
263         while (query.next())
264         {
265             ui->testdata->append(query.value(2).toString());
266             varatut.append(query.value(2).toString());
267         }
268         QSqlDatabase::database().close();
269         int varaukset = varatut.size();
270
271         if(varatut.isEmpty())
272         {
273             //QObject::connect(ui->kalenteri, SIGNAL(clicked(QDate)), this, SLOT(reset()));
274             QObject::connect(this, SIGNAL(refresh()), this, SLOT(reset()));
275             emit refresh();
276         }
277         else
278         {
```

KUVA 11. Viesti ja paikka -mekanismi, aikojen päivitys kun valittu päivä vaihtuu

Aikataulukon täyttäminen vaatii erikoisemman silmukan, sillä jokainen tieto taulukossa on itse asiassa oma olio. Tämän selvittäminen ja toiminnallisuuden teko vei hieman aikaa. Koska jokainen tieto on oma olio, tietojen käsittelystä tuli vähän monimutkaisempaa. Toinen silmukka tarvittiin tarkistamaan onko kyseinen aika jo varattu. Varatut ajat rajattiin pois valinnoista ja

erillinen resetointisignaali yhdistettiin tähän, etteivät varatut ajat jää kummittelemaan, jos valittua päivää jälleen vaihdetaan. Kalenterin asetuksista viikonloppujen poistaminen oli vaikein, koska kyseessä on iso valmis komponentti, jonka perusmekaniikan muuttaminen on vaikeaa. Tämä toteutettiin lopulta ohjeistuksella, että viikonloppuisin ei ole aikoja. Jos lauantai tai sunnuntai yritetään valita, annetaan virheilmoitus ja pyydetään valitsemaan uusi päivä viikolta.

4.7 Tietokanta

Järjestelmän tietokanta ei ole monimutkainen, koska jokaiselle palvelulle tehdään oma taulu, johon merkataan varatut ajat. Taulut eivät tarvitse mitään liitoksia ja yhteneväisyyksiä. Jokaiseen tauluun tulee kolme saraketta, joista kaksi on merkitseviä. Päivämäärä ja kellonaika ovat merkitsevät sarakkeet ja id-sarake on vain kirjanpidollinen lisä. Tauluja tulee siis yhteensä viisi, koska koulutusilmoittautumiset hoidetaan yrityksen web-sivujen kautta ja tällöin touteutettava ohjelma ei tee varauksia. Aina kun joku selaa vapaita aikoja, tauluista tarkistetaan varatut ajat ja rajataan ne pois mahdollisista ajoista. Näin vähennetään muutostyön määrää, kun varaukset pitää vahvistaa ja sovittaa paperikalentereihin. Tämä käytännön takia ei vaadita myöskään erillistä tietokantapalvelinta, vaan tällainen pieni ja kevyt tietokanta hoidetaan paikallisesti WAMP-paketin avulla käyttäen samaa tietokonetta palvelimena, jossa muu ohjelmisto on käynnissä.

Toteutuksen ongelmakohdaksi nousi MySQL-ajureiden saaminen Qt:lle. Ajureita ei tule LGPL-version mukana, vaan ainoastaan kaupallisessa versiossa, joten ainoa mahdollisuus on ajureiden kääntäminen itse. Tähän on tarjolla ohjeet, jotka omasta mielestäni eivät ole kovin selkeitä ja joissa vaiheita ei selitetä tarpeellisella tarkkuudella. Lisäksi rakentamisperiaate muuttuu julkaistun Qt-version mukaan, jolloin oikean ohjeen arvailu lisääntyy. Tämän selvittelyyn ja yrittämiseen tuhlaantui aivan liikaa aikaa ja lopulta, vaikka sain ajurit käännettyä muutamilla eri tavoilla, yksikään niistä ei toiminut tai kehitysympäristö ei löytänyt niitä.

Tästä johtuen viime metreillä MySQL vaihdettiin SQLiteen, jonka suorituskyky riittää tässä tapauksessa varsinkin huomioiden jatkokehitysaiheet. SQLitelle ajurit löytyvät myös ilmaisversiosta, joten piti vain opetella SQLiten käyttäminen. Tästä johtuen myös WAMP-

serveripaketti jäi pois käytöstä ja tilalle otin PhpLiteAdminin (kuva 12), jonka käyttöliittymä on samanlainen muttei vaadi läheskään yhtä paljon asetuksien muuntelua.

KUVA 12. PhpLiteAdmin-käyttöliittymä

5 JATKOKEHITYS

Ohjelmaa on mahdollisesti tarkoitus käyttää useammassa toimipisteessä, joten tämän edesauttamiseksi olisi hyvä lisätä ja muuttaa joitakin ominaisuuksia. Lisäksi samalla voidaan korjata mahdollisia virheitä ja esteettisiä asioita ja tehdä pieniä muutoksia. Suurimpana uutena ominaisuutena kannattaisi tutkia mahdollisuutta synkronoida työntekijöiden kalenterit ohjelman kanssa, jolloin käsipelillä tehtävästä yhdenmukaistuksesta pääsisi eroon. Näistä syistä jatkokehityskierros olisi järkevää toteuttaa.

Käsittelen ensin kalenterisynkronaation. Ohjelman kehityksen aikana kävikin ilmi, ettei kyseessä ollut ihan perinteinen paperikalenteri kokonaisuudessaan, vaan käytössä oli Google Calendar -kalenteri. Tätä voitaisiin hyödyntää suoraan Googlen APIn läpi, jolloin varaukset saataisiin reaaliajassa näkyviin ihmisten kalentereihin tai johonkin yleiseen kalenteriin turvallisuus- ja valintamenettelysyistä. Googlen APIa tarkemmin tuntematta, muut tiedot saattaisivat lähteä edelleen sähköpostissa tai mahdollisuuksien mukaan siirtyä linkitettynä kalenteriin. Välivaihe varauksien järjestelyssä kevenisi, päällekkäiset ajat voitaisiin heti karsia ja varauksien minimietäisyyttä kuluvasta päivästä voitaisiin mahdollisesti pienentää, mikä johtaisi tehokkaampaan ajan käyttöön. Tarvittaisiin tarkistus, että kalenterimerkintä tuli varmasti tehtyä tai tehdään myöhemmin jos esimerkiksi nettiyhteys on poikki. Tämän toteutuksen hankaluudesta ei ole kokemusta, joten en suosittelisi muita suuria muutoksia samaan jatkokehitysprojektiin.

Toinen tärkeä asia olisi asetukset-valikko, joka suojattaisiin salasanalla tai olisi erillinen käynnistettävä osa ohjelmaa, jotta ulkopuoliset eivät voi niitä muuttaa. Tähän valikkoon tulisi sähköpostiosoite tai useampi vaadittaessa ja palvelimen asetusmääritykset. Suurin etu tässä on sovelluksen helpompi ja parempi siirrettävyys uusille toimipisteille, mutta valikko helpottaa muutoksia myös alkuperäisessä toimipisteessä, jos ISP vaihtuu. Samoihin asetuksiin lisättäisiin tarvittavat Google Calendar -asetukset. Käytettävyyden kannalta valikko olisi myös selvä parannus helppouden ja selkeyden kannalta esimerkkinä. Kokonaisuutena tämä on pieni asia lisätä ja alunperin idea tuli mieleen jo tämän työn aikana, mutta ylimääräistä aikaa ei jäänyt toteutukseen.

Lisäksi voisi tehdä pienempiä muutoksia ja korjauksia kuviin, sähköpostin ulkoasuun ym. tarpeen mukaan ja samalla katsoa voisiko kuvien päivittämistä helpottaa. Kysymysten määrää kannattaa

miettiä uudelleen lomakkeissa, varsinkin vapaasti täytettävissä tekstikenttissä, koska näiden karsiminen tekisi käytön sujuvammaksi ja miellyttävämmäksi. Ehdotin kysymysten läpikäyntiä jo alkuperäisen kehityksen aikana, mutta silloin ei ainakaan tullut muutoksia tähän. Valmistautumisohjeet testeihin olisi parempi näyttää ohjelman sisäisesti, mutta pdf-tiedostojen avaaminen Qt:ssä vaatii enemmän työtä, joten tämän paikka on jatkokehityksessä. Vaihtoehtoisesti tiedot voisi muuntaa helpommin käsiteltävään muotoon. Myös sähköpostin lähetyksen toteutusta voi tutkia ja muokata paremmaksi mahdollisuuksien mukaan.

Automaattinen vanhojen aikojen poiston lisääminen tietokantaan voisi olla hyvä idea, jolloin ohjelma ei hidastu ajan myötä, kun tietokanta kasvaa turhan suureksi vanhan datan takia. Toki kantaa voi tyhjentää manuaalisesti PHPMyAdminLiten kautta, mutta käytettävyyden kannalta tämä olisi parempi automaattisena. Tietokannat voisivat mielestäni pysyä paikallisina, koska työntekijät ja paikat ovat erit joka tapauksessa. Tosin jos eri alueen asiakkaan on tarkoitus pystyä varaamaan aika toiseen toimipisteeseen, pitää miettiä, miten tämä toteutettaisiin, ja tässä tapauksessa MySQL kannattaisi saada toimimaan.

6 YHTEENVETO

Opinnäytetyön päämääränä oli touteuttaa ajanvarausjärjestelmä, jota mahdolliset uudet asiakkaat voivat käyttää liikuntahallien yleisissä tiloissa. Samalla järjestelmä toimisi suurena mainoksena ja näkyvyyden kasvuna yritykselle. Koska järjestelmä hyödyntää suurta kosketusnäyttöä, käytettävyyteen kiinnitettiin paljon huomiota, ettei näppäimistöön tarvisi kajota kovin usein.

Käytettävyyden kannalta ohjelman toteutus onnistui, vaikka aina pieniä parannuksia on mahdollista tehdä. Kyselylomakkeiden tekstikenttien määrää olisi hyvä supistaa, jos vain mahdollista, ja ohjelman asetuksille tehdä hallintamekanismi. Käyttöliittymän painikeet ovat tarpeeksi suuria ja monivalinnatkin saa valittua ilman ongelmia, joten käyttäjän ei pitäisi turhautua näistä syistä.

Qt:n ilmaisversion vuoksi tietokantaongelmia oli enemmän kuin odotin ja lopulta ongelmat ratkaistiin vaihtamalla toteutustekniikkaa. Turhautumista aiheuttivat myös sähköpostin kanssa ilmenneet ongelmat, vaikka aihealue olikin tuttu muista tekniikoista. Kuitenkin Qt:n toteutustapa oli erilainen ja dokumentoinnin esimerkit olivat vanhasta versiosta. Graafisesti Qt:llä on kohtalaisen vaivatonta tehdä hienoa jälkeä ja siinä on paljon valmiita komponentteja, joita hyödyntää. Toisaalta jos samoja pohjia haluaa käyttää, kannattaa siirtyä Designerin käytöstä perinteiseen koodausmalliin, jolloin pohjan voi kopioida ja muokata käyttöön helpommin.

Työn aikana ilmeni jotain muutoksia ja väärinkäsityksiä, kuten se, että paperikalenterit tarkoittivatkin Googlen kalenteria. Pienet muutokset tehtiin jo opinnäytetyön aikana ja ohjelma on toimiva kokonaisuus, jota voi jatkokehittää paremmaksi. Alun perin oli tarkoitus, että järjestelmä olisi saatu testikäyttöön jo kehitysvaiheessa ja sitä olisi voitu mukauttaa palautteen mukaan. Ajan ja laitteistohankintaongelmien takia tämä jäi tekemättä. Toisaalta samanlainen testaus voidaan suorittaa ennen mahdollista jatkokehitystä ja huomioida saatu palaute siinä vaiheessa.

LÄHTEET

Blanchette, Jasmin – Summerfield, Mark, 2006. C++ GUI Programming with Qt 4. USA: Prentice Hall. Saatavissa: http://digg.com/news/technology/Free_e_book_C_GUI_Programming_with_Qt_4_Zip_with_PDF. Hakupäivä 11.4.2011.

DuBois, Paul, 2003. MySQL: The definitive guide to using, programming and administering MySQL 4. Inidianapolis, USA: Sams Publishing.

Features of SQLite. Saatavissa: http://www.sqlite.org/features.html. Hakupäivä 9.5.2011.

Kuparinen, Liisa 2008. Käytettävyyden merkitys ohjelmiston valinnassa. Saatavissa: https://jyx.jyu.fi/dspace/bitstream/handle/123456789/18307/URN_NBN_fi_jyu-200805215349.pdf?sequence=1. Hakupäivä 11.4.2011.

Lehtonen, Jyri – Sinisalo, Juha 2007. Graafisen käyttöliittymän suunnittelu ja käytettävyys tietojärjestelmän kehittämisessä. Saatavissa: http://www.peikkoluola.net/portfolio/files/2007_TJK_GUI_21.pdf. Hakupäivä 11.4.2011.

MySQL Reference Manual 5.5, chapter 1.3.1. Oracle. 2011. Saatavissa: http://dev.mysql.com/doc/refman/5.5/en/what-is-mysql.html. Hakupäivä 7.2.2011.

PhpLiteAdmin features. Saatavissa: http://www.sqlite.org/cvstrac/wiki?p=ManagementTools. Hakupäivä 9.5.2011.

Qt framework. 2011. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Qt_%28framework%29. Hakupäivä 7.2.2011.

Riihiaho, Sirpa. Käytettävyyden arviointi ilman käyttäjiä. Saatavissa http://www.soberit.hut.fi/T-121/T-121.600/asiantuntija-arviot.pdf. Hakupäivä 11.4.2011

Rosson, Mary Beth – Carroll John M. 2002. Usability Engineering: Scenario Based Developement of Human-Computer Interaction. USA: Academic Press.

The Qt FAQ.Trolltech. 2005. Saatavissa: http://doc.qt.nokia.com/3.3/faq.html. Hakupäivä 7.2.2011.