GRAAFISEN KÄYTTÖLIITTYMÄN TOTEUTTAMINEN QT-TEKNIIKALLA

Case: Tampereen Bitumikate Oy

Nea Saastamoinen

Opinnäytetyö
Toukokuu 2011
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotanto
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotannon suuntautumisvaihtoehto

SAASTAMOINEN, NEA: Graafisen käyttöliittymän toteuttaminen Qt-tekniikalla: Case Tampereen Bitumikate Oy

Opinnäytetyö 53 s., liitteet 2 sivua.
Toukokuu 2011

Tampereen Bitumikate Oy on Tampereella toimiva vedeneristysalan yritys, joka työllistää noin kymmenen henkilöä kokopäiväisesti. Suurin osa yrityksen toiminnasta perustuu erilaisiin rakennusurakoihin, joista on tarpeen tallentaa tietoja tietojärjestelmiin. Tällaisia tietoja ovat muun muassa urakkamaksuihin liittyvät tiedot. Yrityksellä ei ole käytössään kunnollista järjestelmää, vaan tietoja kirjataan Exceltaulukkoon, joka on erittäin altis esimerkiksi näppäilyvirheille. Yritys haluaisi käyttöönsä paremman ratkaisun tietojen tallentamiseksi.

Opinnäytetyön tavoitteena on helpottaa yrityksessä tehtävien maksukirjausten kirjaamista, vähentää niihin liittyvää näppäilytyötä ja virhealttiutta. Opinnäytetyön tarkoituksena on toteuttaa yrityksen vaatimusten mukainen sovellus, joka mahdollistaa sujuvasti maksukirjausten tekemisen. Opinnäytetyössä keskitytään sovelluksen graafisen käyttöliittymän toteuttamiseen.

Opinnäytetyössä tutkittiin mahdollisuuksia toteuttaa sovellus ja sen käyttöliittymä Java-, PHP- ja C++-kielillä, koska kaikki nämä tarjoavat mahdollisuuden näyttävien käyttöliittymien toteutukseen. Valintaa näiden kielten välillä punnittiin muun muassa alustariippumattomuuden ja tulevaisuutta kestävien näkökulmien kannalta.

Opinnäytetyön tuloksena yritykselle toteutettiin maksujen kirjaussovellus, jossa on graafinen käyttöliittymä. Käyttöliittymä toteutettiin Qt Framework -tekniikalla, joka on C++:n graafinen käyttöliittymäkirjasto ja joka mahdollistaa käyttöjärjestelmästä riippumattomien sovellusten toteuttamisen natiivilla ulkonäöllä.

Qt Framework havaittiin erittäin hyväksi tavaksi toteuttaa sovelluksia ja niiden käyttöliittymiä tehokkaasti ja nopeasti.

Asiasanat: Graafisen käyttöliittymän toteutus, Qt Framework, C++

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology
Option of Software engineering

SAASTAMOINEN, NEA: Creating A Graphical User Interface with Qt Framework: Case Tampereen Bitumikate Oy

Bachelor's thesis 53 pages, appendices 2 pages
May 2011

Tampereen Bitumikate Oy is a roof waterproofing company in Tampere and it employs about ten full-time employees. The majority of the tasks in the company are based on different types of construction work. The company needs to save information about these tasks, such as information about payments concerning the contracts. There is currently no appropriate system in use in the company but the information is kept in an Excel datasheet. The datasheet is very vulnerable to typing mistakes, for example. The company would like to use a better solution to save the information.

The objective of this thesis was to make it easier to save the information concerning the payments and lessen the vulnerability to typing mistakes. The purpose of this thesis was to create an application which meets the criteria set by the company and which makes it easier to save the payment information. The thesis is focused on creating a graphical user interface for the application.

Different kinds of methods and programming languages were investigated to create the application and its graphical user interface. For example Java, PHP and C++ were investigated because they serve a possibility to create an impressive graphical user interface and they are platform independent.

The result of this thesis is a graphical application in which it is easy to save information about the payments. The graphical user interface was created in Qt Framework technology which is a C++ library for graphical user interfaces and which makes it possible to create operating system independent applications with native look-and-feel and user experience.

In this thesis, the Qt Framework was identified as a very good technology to build the application and graphical user interfaces very efficiently and quickly.

Keywords: Creating a Graphical User Interface, Qt Framework, C++

SISÄLTÖ

1 JOHDANTO

Tampereen Bitumikate Oy on Tampereen Sarankulmassa toimiva vedeneristysalan yritys, joka on perustettu vuonna 1985. Yrityksessä työskentelee vakituisesti kymmenisen henkilöä erilaisissa tehtävissä. Suurin osa yrityksen toiminnasta perustuu vaativiin vedeneristys- ja kattoalan rakennusurakoihin. Yrityksellä on tarve tallentaa rakennusurakoita koskevia tietoja muun muassa palkanmaksua varten.

Tällä hetkellä tietoja kirjataan Excel-taulukkoon, jonka käyttäminen vaatii paljon ylimääräistä käsityötä ja papereiden pyörittelyä. Yritys haluaisi käyttöönsä paremman ratkaisun, jotta erittäin oleelliseen maksujen kirjaamiseen liittyvää työtaakkaa saataisiin vähennettyä ja ohjattua toimeksiantajan kannalta tärkeämpiin liiketoiminta-prosesseihin.

Tämän opinnäytetyön tavoitteena on helpottaa ja yksinkertaistaa toimeksiantajan maksukirjausten tekemistä. Opinnäytetyön tarkoituksena on tuottaa toimeksiantajan esittämien vaatimusten mukainen sovellus, joka tekee maksukirjausten merkitsemisen helpoksi ja sujuvaksi. Sovelluksen toteutustapaa punnitaan muutamien mahdollisten vaihtoehtojen välillä.

Tässä opinnäytetyössä paneudutaan myös siihen, mikä käyttöliittymä itse asiassa on ja millainen sen on oltava, jotta sovelluksen käyttö olisi käyttäjälle mieluista ja helppoa. Työn lukijalle suositellaan yleistä, keskitasoista ohjelmointikielten peruskäsitteiden tuntemista olio-ohjelmoinnin, UML-mallinnuksen ja käytettävyyden periaatteiden osalta. Lisäksi suositellaan yleisimpien käyttöliittymäkomponenttien tuntemista.

2 TAUSTA

2.1 Maksujen kirjaaminen Tampereen Bitumikate Oy:ssä

Palkanmaksua varten yrityksen työntekijät ilmoittavat eri työmailla käyttämänsä työtunnit toimistolla työskentelevälle henkilölle. Ilmoitusvälineenä toimii paperimuotoinen asiakirja eli niin sanottu ”tuntilappu”, jonka kukin työntekijä palauttaa tyypillisesti parin viikon välein toimistolle. Näistä tuntilapuista yrityksen toimistotyöntekijä kokoaa tiedot käsin Excel-taulukkoon ja muodostaa näiden tietojen perusteella maksurivejä. Näitä maksurivejä kutsutaan ennakkomaksuiksi.

Tuntilappujen lisäksi työntekijät palauttavat myöhemmin vielä työmaata koskevat mittauspöytäkirjat, jossa on eriteltynä urakan sisältämät erityyppiset työtehtävät aiemmin esiteltyä ”tuntilappua” tarkemmin. Mittauspöytäkirjan perusteella yrityksen toimistotyöntekijä laskee vielä urakkamaksun, joka myös merkitään Exceliin. Maksuja on siis kahdentyyppisiä – ennakko- ja urakkamaksuja – ja näillä on hieman erityyppiset laskentaperusteet. Molemmat maksut muodostavat työntekijälle maksettavan palkan.

2.2 Havaitut ongelmat

Yrityksen edustajan mukaan tämänhetkinen Excel-taulukkoon perustuva ratkaisu ei ole riittävä ja siinä on havaittu selkeitä ongelmia. Käsin täytettävä Excel-taulukko vaatii hyvin paljon ylimääräistä näppäilytyötä esimerkiksi siten, että työntekijän tiedot täytyy syöttää joka kerta uudestaan. Edellä mainitun lisäksi taulukko on erittäin altis muun muassa näppäilyvirheille, koska tekstiä syötetään paljon.

Jatkuva samojen tietojen uudelleen syöttäminen ja mahdollisten virheiden korjaus vie myös tarpeettomasti toimistotyöntekijän työaikaa ja tämän ajan hän voisi käyttää tehokkaamminkin. Yrityksessä siirryttäisiin mielellään käyttämään sovellusta, jolla voisi vaivattomasti tallentaa rakennusurakoita koskevat maksut, mutta tällaista ohjelmaa ei toimeksiantajan mukaan ole saatavilla.

2.3 Aiheen rajaus

Opinnäytetyönä toteutetaan sovellus toimeksiantajan tarpeisiin ja tässä opinnäytetyössä keskitytään sovelluksen käyttöliittymän luomiseen. Sovellukseen luodaan sellaiset toiminnallisuudet, joille löytyy vastine toimeksiantajan käytössä olevasta Excel-taulukosta. Tämä tarkoittaa sitä, että opinnäytetyössä paneudutaan yrityksen käytössä olevan Excel-taulukon sisältämien tietojen ja toiminnallisuuksien parempaan jäsentelemiseen uudessa sovelluksessa. Tarkoituksena ei ole luoda uutta toiminnallisuutta. Sovellus toteutetaan kuitenkin sillä tavoin, että uusien toiminnallisuuksien lisääminen on jälkeenpäin helppoa.

Opinnäytetyön tavoitteena on vähentää toimeksiantajan urakkamaksujen kirjaamiseen liittyvän ylimääräisen näppäilytyön määrää. Tavoitteena on myös kirjausten tekemiseen liittyvän virhealttiuden vähentäminen ja kerätyn aineiston eli datan jatkokehityskelpoisuus myöhempää käyttöä – esimerkiksi tilastointia ja raportointia – varten. Tämän lisäksi opinnäytetyön tavoitteena on vähentää toimistotyöntekijän maksuihin liittyvää yksinkertaista laskemistyötä. Ennen kaikkea sovelluksen on määrä ratkaista toimeksiantajan maksujen kirjaamiseen liittyviä ongelmia ja siten tehostaa tätä toimeksiantajan tärkeää liiketoimintaprosessia.

2.4 Aiempi kokemus

Olen perehtynyt käyttöliittymien toteuttamiseen Tampereen ammattikorkeakoulussa tietojenkäsittelyn opintojeni ohessa. Suuntauduin opinnoissani ohjelmistotuotantoon, jonka ansiosta tutustuin eri ohjelmointikielten ja -tekniikoiden piirteisiin sekä ohjelmistosuunnittelun että -toteutuksen tasoilla. Suunnittelun ja toteutuksen lisäksi olin tutustunut ohjelmistokehitykseen liittyviin käytänteisiin.

Laajahko ohjelmointikokemukseni koostui näin ollen toteutustasolla muun muassa Java-, PHP- ja C++-kielistä ja suunnittelutasolla UML-mallinnuskielestä. Tietokanta-osaamista minulla oli MySQL:stä, PostgreSQL:stä ja Microsoft Accesista. Kokemukseni ansiosta pystyin punnitsemaan sovelluksen toteutustapaa edellä mainittujen ohjelmointikielten ja -tekniikoiden välillä.

2.5 Ratkaisuehdotus

Toimeksiantajan maksujen kirjaaminen helpottuisi paljon, jos tällä olisi käytössään Excel-taulukkoa parempi ratkaisu, esimerkiksi maksujen kirjaussovellus. Tämän opinnäytetyön puitteissa toimeksiantajalle päätettiin luoda sovellus, jonka on tarkoitus vähentää myös yksinkertaista laskemistyötä, sillä sovellukseen on helposti toteutettavissa muun muassa automaattisesti laskettavia kenttiä käyttäjän syöttämistä tiedoista.

3 KÄYTETTÄVÄT MENETELMÄT

3.1 Sovelluksen suunnittelu ja sen tietorakenteet

Toteutettava sovelluksen tietorakenteet perustuvat aiemmin toimeksiantajan käytössä olleeseen Excel-taulukon vastaaviin tietorakenteisiin. Toteuttavaan sovellukseen nämä rakenteet jäsennellään loogisesti paremmin ja siten, että ne ovat helpommin jatko-hyödynnettävissä. Käytännössä tämä tarkoittaa käytössä olevan Excel-taulukon sisältämien reaalimaailman asioiden jaottelua järkeviin kokonaisuuksiin. Asioiden jaotteluun voidaan käyttää erilaisia tapoja.

Sovellukseen tarvitaan toiminnallisuudet työntekijöiden, työmaiden ja maksujen tietojen käsittelemiseen ja tallentamiseen. Maksurivien kirjaamista varten tarvitaan lisäksi muutamia yksinkertaisia nelilaskintasoisia laskutoimituksia. Sovelluksen käyttäjällä pitää olla mahdollisuus tallentaa sovelluksessa käsiteltäviä tietoja esimerkiksi tietokantaan.

Sovelluksessa on tärkeää tehdä ero kahden erityyppisen maksun välillä, koska näiden maksujen laskentaperusteet poikkeavat merkittävästi toisistaan. Maksuja on siis kahdentyyppisiä: ennakko- ja urakkamaksuja. Ennakkomaksut lasketaan parin viikon välein yrityksen työntekijöiden toimittamista asiakirjoista, joita kutsutaan ”tuntilapuiksi”. Urakkamaksut lasketaan urakan päätteeksi mittauspöytäkirjasta. Yrityksen toimistotyöntekijä laskee maksut suurimmaksi osaksi käsin. Hänen mukaansa käsin laskemiseen kuuluu liikaa aikaa ja vaivaa sekä tietojen tallennus on työlästä.

Toteutettavan sovelluksen on tarkoitus olla helposti laajennettavissa esimerkiksi erilaisia tarkempia ja täsmällisempiä rekistereitä kattavaksi kokonaisuudeksi. Siihen voidaan kehittää muuan muassa asiakashallintaa helpottava asiakasrekisteri, joka yhdistetään työmaiden tietoihin. Sovellukseen voidaan kehittää myös henkilöstöpalveluita helpottavaksi työkaluksi työntekijärekisteriksi, joka sisältäisi myös tarkempia työsuhteeseen, työntekijöiden tietoihin sekä taitoihin liittyviä tietoja. Niin ikään olisi helppoa laajentaa sovellusta siten, että se kattaisi myös palkanmaksun. Yrityksellä ei ole kuitenkaan tällä hetkellä tarvetta sovelluksen palkanmaksulaajennukselle, koska palkat maksetaan tällä hetkellä tilitoimiston välityksellä. Sovellus tulee siten aluksi vain yrityksen sisäiseen käyttöön.

3.2 Toteutustavan ja -teknologian valinta

Sovellusta suunniteltiin toteutettavaksi Visual Basic-, Java-, PHP- ja C++-kielillä, koska nämä teknologiat tarjoavat riittävät työkalut sovelluksen toteuttamiseen. Kaikilla näillä teknologioilla voidaan toteuttaa sangen helposti sekä näyttäviä graafisia käyttöliittymiä että niin sanotusti sovelluksen alla olevaa toiminnallisuutta.

Toteutusteknologian valintaa pohdittiin tarkoin ja pohdinnan aikana tutustuttiin ohjelmointikielten ominaisuuksiin ja niiden tarjoamiin mahdollisuuksiin toteuttaa toimeksiantajan vaatimusten mukainen sovellus. Valinnassa painotettiin ensisijaisesti käyttöjärjestelmäriippumattomuutta, koska yrityksen toimipisteestä löytyy sekä Windows- että Linux-käyttöjärjestelmällä varustettuja työpisteitä. Lisäksi valinnassa painotettiin kustannusten minimointia ja tulevaisuutta kestävää ratkaisua sovelluksen jatkokehitystä ja laajennusta ajatellen.

Edellä kuvatuista kielistä ainoastaan Visual Basic keskittyy Windows-ympäristöön, mutta niin Java, PHP kuin C++:kin ovat käyttöjärjestelmäriippumattomia ohjelmointikieliä. Suunnitelma PHP:stä toteutuskielenä hylättiin, koska se on tyypillisesti web-ympäristöön tarkoitettu kieli ja siten vaatii WWW-palvelimen, esimerkiksi Apache2:n, toimiakseen. Palvelimen hankinta ja siihen liittyvä ylläpito lisäisivät hyvin todennäköisesti sovelluksen käyttökustannuksia, koska toimeksiantajalla itsellään ei ole palvelimen ylläpitoon vaadittavia tietoja eikä taitoja, vaan nämä pitäisi hankkia ulkopuoliselta toimijalta.

Mahdollisiksi toteutustavoiksi jäivät jäljelle näin ollen Java ja C++. Valinta näiden kahden välillä ei ollut aivan yksinkertainen, sillä molempiin on saatavilla tuttuja käyttöliittymän toteutuskomponentteja. Javaan on saatavilla Swing-käyttöliittymäluokkakirjasto ja C++:aan on saatavilla Qt Framework -ohjelmistokehys. Molemmat kielet tukevat tunnetusti myös olioperusteista ohjelmointia ja molempien olemus tietotyyppeineen ja funktioineen on suhteellisen samanlainen.

Merkittävin ero näiden kahden kielen välillä lienee kuitenkin se, että Java on tulkattava kieli ja C++ käännettävä kieli. Tämän vuoksi Javalla toteutettua lähdekoodia on hitaampaa kääntää kuin C++:lla toteutettua lähdekoodia. Tämä johtuu siitä, että Java-koodi täytyy ensin kääntää tavukoodiksi ja vasta sen jälkeen se tulkataan

erityisellä Java Virtual Machinellä käyttöjärjestelmän ymmärtämään muotoon. C++-kielessä lähdekoodi käännetään vain kerran ja käännöksen tuloksena syntyy suoraan käyttöjärjestelmälle sopiva käännös sovelluksesta.

Vähemmän merkittävä ero löytyy myös kielten käyttämistä muistinhallintatavoista. Javassa on automaattinen roskienkeräys eli garbage collection, joka vapauttaa muistia sitä mukaa, kun sitä ei enää tarvita. Javasta poiketen C++ tukee esimerkiksi delete-operaattoria, hajotinta eli destruktoria ja osoittimia, jotka ovat järkevän muistin käytön kannalta erittäin tehokkaita ja käyttökelpoisia komponentteja.

Sovelluksen ja käyttöliittymän toteutustavaksi valittiin C++, joka varustettiin Qt-ohjelmistokehyksellä. Qt lienee C++:n tunnetuin luokkakirjasto graafisten käyttöliittymien toteuttamiseksi. Sovelluksessa käsiteltävät tiedot päätettiin tallentaa SQLite-tietokantaan, koska sille löytyy suoraan tuki Qt:stä sekä desktop- että mobiilipuolella. Lisäksi SQLite-tietokannat ovat pienikokoisia esimerkiksi Microsoft Accesilla tehtyihin tietokantoihin nähden, eivätkä ne ole riippuvaisia tietokanta-palvelimesta, kuten esimerkiksi MySQL ja PostgreSql.

4 KÄYTTÖLIITTYMÄN SUUNNITTELEMINEN

4.1 Mikä on käyttöliittymä?

Tietoteknisessä kirjallisuudessa esiintyvä käyttöliittymän määritelmä ei ole täysin vakiintunut, vaan lähes jokaiselta löytyy oma määritelmänsä. Yleensä määritelmässä esitetään käyttöliittymän olevan käyttäjän käsittelemä osa, jonka välityksellä tämä on vuorovaikutuksessa järjestelmän kanssa. Käyttöliittymiin kuuluu sekä ohjelmoinnilla saavutettuja rakenteita eli esimerkiksi sovelluksia sekä fyysis-konkreettisia laitteita ja välineitä. Alla on muutamia käyttöliittymän määritelmiä.

Käyttöliittymällä tarkoitetaan käyttäjälle näkyvää ohjelmiston osaa, jonka välityksellä käyttäjä antaa käskyjä järjestelmälle (Alaterä & Halttunen, 2003).

”Käyttöliittymä mahdollistaa ohjelman ja käyttäjän välisen vuorovaikutuksen: esimerkiksi tietojen syöttämisen, komentojen antamisen ja ohjelman palautteen.” (Visio-paja, 2008).

Käyttöliittymä voidaan helposti mieltää pelkäksi merkkipohjaiseksi tai graafiseksi näytöllä näkyväksi kuvavirraksi, mutta käyttöliittymää edustavat myös välineet, joilla käyttöliittymää käytetään, esimerkiksi raha-automaatin näppäimet (Wikipedia, 2011).

Käyttöliittymä voidaan jakaa myös kolmeen tasoon, jotka ovat teknis-fysiologis-ergonominen taso, käsitteellis-havainnollinen taso ja toiminnallis-kontekstuaalinen taso. Näistä ensimmäiseksi mainittu edustaa fyysistä laitetta ja välineistöä, jotka tarvitaan vuorovaikuttamiseen, ja kaksi jälkimmäistä sovelluksen loogista tasoa, jotka saavutetaan esimerkiksi ohjelmoinnilla ja jotka ovat abstrakteja ja joita ei siten voi fyysisesti kosketella. Käyttäjä ei välttämättä tee eroa eri tasojen välillä, vaan mieltää koko järjestelmän yhdeksi käyttöliittymäkokonaisuudeksi. (Kuutti, 1999.)

Käyttöliittymät liittyvät kiinteästi käytettävyyteen, joka on ohjelmistotuotannossa yksi ensisijaisimmista tavoitteista. Käyttöliittymä on parhaimmillaan silloin, kun käyttäjä kokee sovelluksen käytön olevan sujuvaa ja sulavaa, eikä edes välttämättä huomaa käyttävänsä mitään erityistä käyttöliittymää tai tietoteknistä vempainta.

Tästä voidaan helposti vetää johtopäätös, että hyvä käyttöliittymä edustaa hyvää käytettävyyttä ja käyttöliittymä on eräs käytettävyyden ilmenemismuoto. Tarkemmin hyvään käyttöliittymään tutustutaan kappaleessa ”4.5 Millainen on hyvä käyttöliittymä?”.

4.2 Käyttöliittymät yleensä

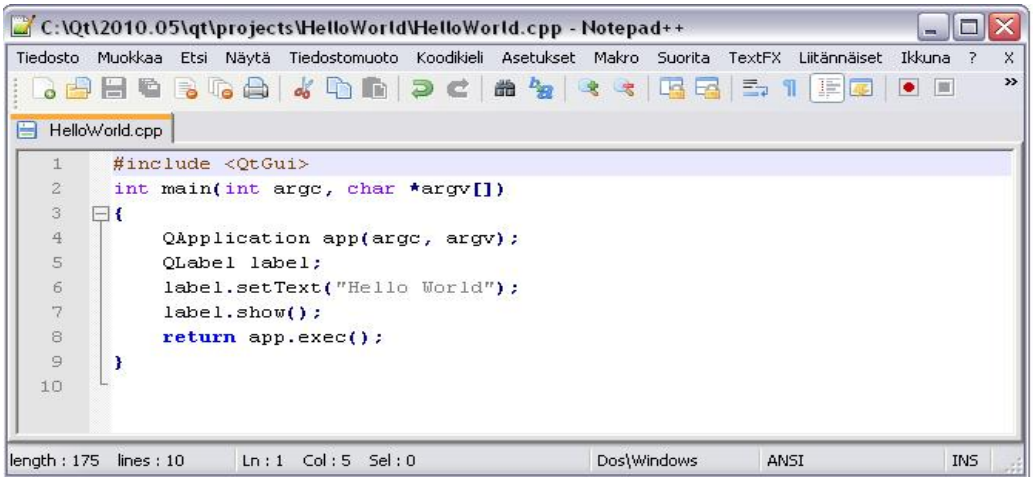
Käyttöliittymät perustuvat vuorovaikutusmalleihin ja käyttöliittymien ovat vain yksi osa sovellusta. Sovellukset voidaan jakaa sen mukaan, minkä tyyppisestä sovelluksesta ja vuorovaikutuksesta on kyse. Sovellukset voidaan esimerkiksi jakaa kahteen tyyppiin, jotka ovat välinesovellus ja palvelusovellus. Välinesovellukset ovat pää-sääntöisesti yhden käyttäjän välineitä jonkin asian työstämiseksi ja palvelusovellukset yleensä useamman käyttäjän, laajoihin kokonaisuuksiin liittyviä välineitä. (Wiio, 2004, 126–127.)

	Palvelusovellus	Välinesovellus
Suorakäyttö	Tuotannonohjaus	Tekstit Julkaisut Kuvat Laskelmat Suunnitelmat
Keskustelukäyttö	Pankkiautomaatti Kirjanpito Webbikauppa	Käyntikorttiautomaatti

KUVIO 1. Eri sovellustyyppejä (Wiio 2004, 126)

Kuvio 1:stä voidaan loogisesti päätellä, että suorakäyttöiset välinesovellukset, joissa objekteja käsitellään nimensä mukaisesti suoraan, ovat tyypillisempiä kuin suorakäyttöiset palvelusovellukset. Sama pätee myös keskustelukäyttöisten väline-

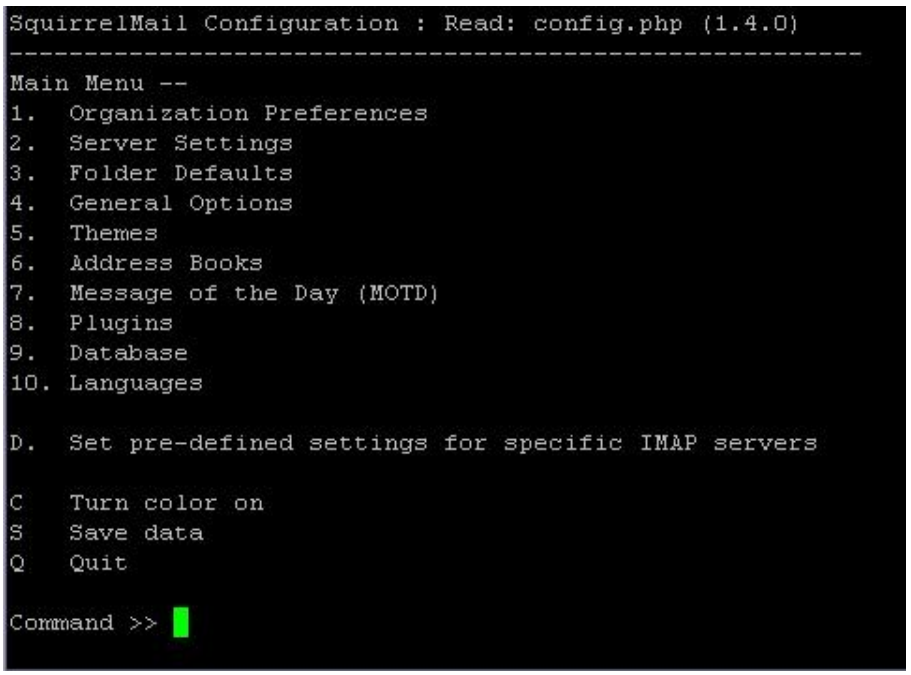
sovellusten ja keskustelukäyttöisten palvelusovellusten välillä, mistä viimeiseksi mainittu näyttää olevan tyypillisempi. Wiion mukaan suorakäyttöisissä sovelluksissa käyttäjä voi tarttua olioihin ja kohdistaa näihin toimenpiteitä, joiden tulos nähdään näytöllä välittömästi. Sen sijaan keskustelukäyttöisissä sovelluksissa olioita ei käsitellä suoraan, vaan välissä toimivat esimerkiksi lomakkeet. (Wiio, 2004, 125, 144.)



KUVIO 2. Tekstieditori Notepad++ edustaa suorakäyttöistä välinesovellusta, jossa objektia eli tässä tekstiä muokataan siten, että muutokset näkyvät välittömästi.

Lisäksi käyttöliittymät voidaan jakaa niiden käytön mukaan pelkästä tekstistä koostuviin merkkipohjaisiin ja graafisiin käyttöliittymiin. Nykyään käytössä olevat sovellukset ovat yleensä graafisella käyttöliittymällä varustettuja. Kuviossa 2 on esimerkkinä pankkiautomaatin keskustelukäyttöisen palvelusovelluksen graafinen käyttöliittymä. (Alaterä, Halttunen, 2003; Automatia Pankkiautomaatit Oy 2011).

KUVIO 3. Otto. -pankkiautomaatti edustaa tyypillistä keskustelukäyttöistä palvelusovellusta. (Automatia Pankkiautomaatit Oy, 2011).



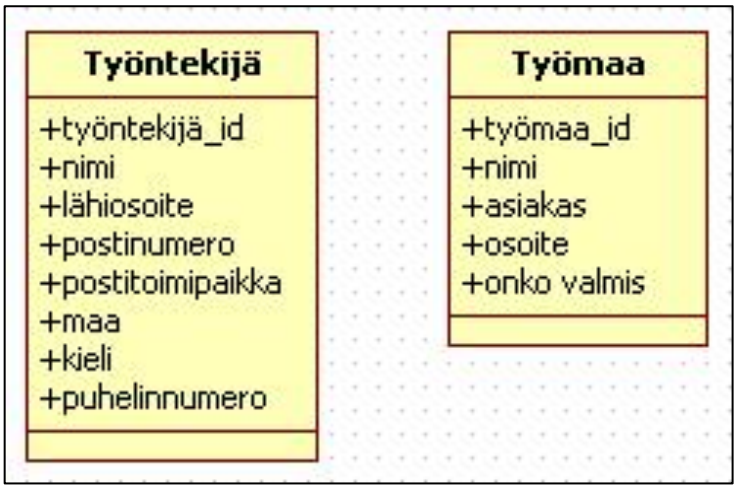
KUVIO 4. Squirrelmail-webmailin hallinta ja konfigurointi tehdään tekstipohjaisella työkalulla, joka on myös keskustelukäyttöinen palvelusovellus.

4.3 Käyttöliittymien suunnittelumenetelmät

Sovelluksen ja sen käyttöliittymän suunnittelu on hyvä aloittaa tunnistamalla käyttäjän pyrkimys ja siihen tarvittavat prosessit. Pyrkimyksiä varten voidaan luokitella käyttäjiä ja tunnistaa kaikille käyttäjille yhteisiä pyrkimyksiä. Seuraava tärkeä osa sovelluksen suunnitteluvaihetta on yksilöidä ja jäsennellä sovellukseen liittyviä prosesseja. (Wiio 2004, 93–94.)

Eräs tapa jäsennellä näitä prosesseja on sovelluksen jäsenteleminen olio-ohjelmoinnin peruskäsitteiden mukaisesti luokkiin ja attribuutteihin. Tällaista luokittelua varten hyvänä sovelluskehityksenä työkaluna toimii Unified Modeling Language eli UML. (Wiio 2004, 105–107.)

Opinnäytetyönä toteutettavaan sovellukseen kuuluvia luokkia ovat esimerkiksi työntekijä, työmaa ja maksu. Näillä luokilla on erilaisia ominaisuuksia eli attribuutteja kuten nimi ja osoite. Wiion (2004, 107) mukaan UML:n luokkamallityyppinen jaottelu on hyödyllinen ohjelmistojen suunnittelumenetelmänä vaikkapa sovelluksen alla toimivan tietokannan rakenteen suunnittelussa.



KUVIO 5. Työntekijän ja työmaan tiedot UML-luokkakaaviossa

Toinen, hieman edellistä abstraktimpi tapa lähestyä sovellusta ja sen käyttöliittymää on hahmotella käyttöliittymästä erilaisia malleja. Esimerkkinä tällaisesta hahmoteltavasta mallista on ajatusmalli, englanniksi ”mental model”, joka on yleisesti käytettävyyttä ja käyttöliittymiä koskevan kirjallisuuden mukaan hyvin keskeinen termi. Ajatusmalli on käyttäjän ajatuksiin rakentuva mielikuva siitä, mitä objekteja sovelluksessa voidaan käsitellä ja mitä suorituksia näillä objekteilla on mahdollista tehdä. (Wiio 2004, 151, 167.)

Esimerkiksi taulukkolaskentasovelluksessa sovelluksen käyttäjän ajatusmallin sisältämiä objekteja voivat olla muun muassa solu, rivi, sarake, kaava ja aritmeettinen operaattori, kuten + ja -. Opinnäytetyönä toteutettavassa sovelluksessa objekteja ja niitä hyödyntäviä toimenpiteitä voivat taas olla maksu ja sen kirjaaminen, palkan-maksukausi sekä työntekijän tietojen muokkaaminen.

Ajatusmallin lisäksi on olemassa myös käyttäjän käsitemalli. Ajatusmallin ja käyttäjän käsitemallin ero on se, että ajatusmalli on käyttäjän mieleen perustuva malli ja käyttäjän käsitemalli on suunnittelijan tekemä malli, jonka hän haluaa välittyvän käyttäjälle sovelluksen toimintalogiikan perusteella. Toisinaan nämä mallit voivat poiketa merkittävästi toisistaan, mutta tähän on ratkaisuna käyttäjäkeskeinen suunnittelu. (Wiio 2004, 152, 167.)

4.4 Käyttöliittymän suunnitteluprosessi

Sovelluksen käyttöliittymä kannattaa laatia yhteistyössä sovelluksen oletettujen käyttäjien kanssa, jotta heidän kannaltaan saavutetaan sovelluksen optimaalinen käytettävyys. Tällaista suunnittelua kutsutaan käyttäjäkeskeiseksi suunnitteluksi. Käyttäjä-keskeisestä suunnittelusta huolimatta myös sovelluksen suunnittelijalla on vapaus ja sen myötä vastuu tehdä valintoja käyttöliittymän rakenteen suhteen. Käyttäjäkunnasta voidaan kerätä esimerkiksi henkilöhaastattelujen ja kyselyjen avulla tausta-tutkimusaineistoa, jota analysoimalla ja jalostamalla voidaan kartoittaa sovellusta ja käyttöliittymää. Taustatutkimusaineiston prosessoinnin tuloksena voi syntyä sovellusta koskevia vaatimuksia ja toiveita sekä käyttöliittymää koskevia valintoja. (Ekman, Ermi, Jäppinen, Kirvesmäki, Lankoski & Nummela 2002, 7–9, 56.)

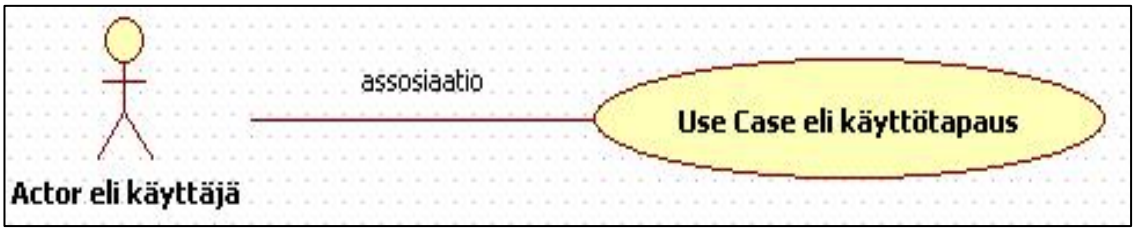
Käyttöliittymää koskevia valintoja tehdään muun muassa käyttötarkoituksen ja esitetyn tietosisällön suhteen. Nämä valinnat vastaavat kysymyksiin ”mihin sovellusta käytetään tai tarvitaan?” ja ”miten sovelluksessa on esitelty sen sisältämät objektit?”. Toisenlaisia käyttöliittymään liittyviä valintoja edustavat termit, joita käyttöliittymä sisältää. Käyttöliittymässä olisi hyvä käyttää vain sellaisia ilmaisuja, jotka ovat käyttäjälle luontevia ja mahdollisesti ennestään tuttuja. Käyttäjälle tutut sanavalinnat auttavat käyttäjää yhdistämään tutut termit niille ominaisiin, tuttuihin objekteihin ja prosesseihin. Kuitenkin kaikkien lopulliseen käyttöliittymään valittujen ratkaisujen on tärkeää olla tasapainossa sekä suunnittelijoiden että käyttäjien suhteen, koska muutoin sovelluksen käytössä voi ilmetä erilaisia ongelmia. (Ekman ym. 2002, 9–10, 25, 27.)

Käyttäjäkeskeistä prosessia jatketaan yleensä iteratiivisesti eli toistaen käyttäjien kanssa läpi koko sovelluksen suunnittelu- ja toteutusprosessin, koska kaikkea aineistoa on vaikea saada koottua yhdellä kertaa. Toisaalta on järkevää keskittyä yhteen sovelluksen kokonaisuuteen kerrallaan ja paneutua siihen riittävän tehokkaasti. Myös käyttäjäkunnasta kummunneiden muutostoiveiden ja -ehdotusten lopputulokset on hyvä käydä käyttäjän kanssa läpi. (Ekman ym. 2002, 9.)

Käyttöliittymän suunnittelun tueksi voidaan koota myös käyttötilannemalleja, jotka muodostuvat käyttäjän tavoitteista eli siitä, mihin päämäärään käyttäjällä on

pyrkimys päästä sovelluksen avulla. Käyttötilannemallit voidaan laatia vapaamuotoisena tekstinä, numeroiduin toimenpideaskelein tai käytön kuvauksena tila-automaattina. Sovelluksen käyttötilannemallit saadaan koottua sovelluksen käyttäjiltä tunnistamalla kaikki heidän pyrkimyksensä, jotka sovellukseen liittyvät. Yleensä käyttötilannemalleista on karsittu enimmät yksityiskohdat pois ja niissä keskitytään enemmänkin johonkin tiettyyn, yleistettyyn käyttötilanteeseen. Myöhemmin käyttötilannemalliin voidaan kartoittaa vielä prosessiin liittyvät poikkeukset. Käyttötilannemalli ei suoraan ota kantaa käyttöliittymän rakenteeseen, mutta se toimii erinomaisesti käyttöliittymä-rakenteen suunnittelun tukena. (Wiio 2004, 111, 115–116, 120–121.)

Myös mallinnuskieli UML tarjoaa muutamia lähestymistapoja käyttötilannemalleille. Niitä voidaan havainnollistaa esimerkiksi käyttötapauskaaviona (englanniksi use case diagram). Tällaiseen kaavioon kootaan sovelluksessa olevat käyttäjät tai toimijat (englanniksi actor) ja heidän suorittamansa toimenpiteet eli käyttötapaukset (englanniksi use case) sovelluksessa. Jokainen käyttötapaus assosioidaan sen käyttäjän kanssa yhteen viivalla ja näin syntyy käyttötapauskaavio. Käyttötapauskaavio on aina sen mukainen, millaisena käyttäjä näkee järjestelmän, mutta tunnetusti sovellukseen liittyy paljon myös sellaista toiminnallisuutta, joka on sen kuoren alla. Esimerkki tällaisesta toiminnallisuudesta on sovelluksen apufunktiot, jotka tukevat varsinaisten käyttötapausten suorittamista.



KUVIO 6. Käyttötapauskaavio, jossa käyttäjällä (actor) on yksi käyttötapaus (use case).

4.5 Millainen on hyvä käyttöliittymä?

Käyttöliittymiä koskevassa kirjallisuudessa annetaan paljon tavoitteita, jotka hyvän käyttöliittymän tulisi toteuttaa tai millainen sen tulisi olla. Kaikki nämä tavoitteet

ovat ihmislähtöisiä ja tyypillisesti ne liittyvät ihmisen kognitiivisiin kykyihin, kuten muistiin, oppimiseen, ajatteluun, havaitsemiseen, tarkkaavaisuuteen, luovuuteen sekä ongelmanratkaisuun.

Wiio (2004, 38) luettelee seuraavat ominaisuudet, millainen käyttäjäystävällinen sovellus on: esteettisesti miellyttävä, kattava, vaivaton, ymmärrettävä.

Alaterä ja Halttunen (2003) täsmentävät hyvän käyttöliittymän ominaisuuksia muun muassa esittämällä, että käyttöliittymässä pitäisi olla vaikea tehdä virheitä, mutta virheen tapahduttua sen pitäisi olla helposti korjattavissa.

Ermi (2002, 57–66) painottaa käytettävyydessä enemmän tunteita ja kokemuksia, kuten helppokäyttöisyyttä, miellyttävyyttä, hauskuutta, kauneutta, yksinkertaisuutta, tuttuutta ja turvallisuutta sekä hallinnantunnetta ja kompetenssia.

Edellä esitellyt tavoitteet voidaan jakaa pääpiirteittäin kahteen tyyppiin: sovellukseen liittyviin objektiivisiin ominaisuuksiin ja käyttäjän kokemiin subjektiivisiin asioihin. Tietynlaista objektiivisuutta edustaa se, että sovellus voi olla tehokas ja kattava sekä se voi olla tehty sellaiseksi, että siinä on vaikea tehdä virheitä. Sen sijaan käyttäjän tunteita ja kokemuksia ovat hauskuutta, kauneutta ja ymmärrettävyyttä koskevat seikat. Tietynlaista käyttäjän kokemaa subjektiivisuutta on myös käytön helppous, joka voi perustua osittain käyttäjä empiirisiin eli kokemusperäisiin havaintoihin ja rutiineihin.

Ekmanin ym. (2002, 10, 19) mukaan ”Ihmisten tulisi pystyä käyttämään – – digitaalisia palveluja ilman, että he kokisivat käyttävänsä jotakin käyttöliittymää, laitetta tai palvelua. Palvelu, jonka käyttöliittymä vaatii ponnistelua tai jonka opetteleminen tuntuu työläältä, jää yleensä käyttämättä.” Lisäksi on myös tärkeää valita, mitä tietoa rajallisella näytöllä esitetään ja missä järjestyksessä tieto esitetään.

Lankoski ym. (2002, 19) siteeraa Shedroffia (1994) luetellen informaation järjestämis-tapoja aakkosten, ajan, paikan, jatkumon, numerojärjestyksen ja kategorian sekä satunnaisesti. Informaation järjestämistavat kannattaa huomioida erityisesti käyttöliittymän valikkorakenteissa, koska se vaikuttaa valikon käytön nopeuteen ja ymmärrettävyyteen (Wiio 2004, 182–187).

Asioiden järjestäminen ei liity ainoastaan valikoihin ja listoihin, vaan esimerkiksi myös komentopainikkeiden järjestyksellä on väliä. Vakiintunut tapa järjestää painikkeet on sijoittaa ensiksi toimintaa edistävät painikkeet, seuraavaksi poistavat ja hävittävät toiminnot ja mahdolliset lopuksi ikkunakohtaiset painikkeet. Tyypillinen toimintaa edistävä painike on OK tai joissakin tapauksissa Sulje. Poistamista ja hävittämistä edistävä painike on yleensä nimeltään Peruuta tai lyhemmin Peru. Länsimaissa totuttu tapa tiedon järjestämiseen on vasemmalta oikealle ja ylhäältä alas. (Sinkkonen & Tuominen 1996, 156.)

Käytettävyysasiantuntijana erittäin tunnettu Jakob Nielsen on koonnut listan kymmenestä yleispätevästä käyttöliittymäsuunnittelun tavoitteesta. Listalla on paljon asioita, jotka koskevat sitä, millä tavoin hyvä käyttöliittymä on vuorovaikutuksessa käyttäjän kanssa.

1. Kommunikoi sopivalla tavalla, sopivassa ajassa siitä, mitä kulloinkin tapahtuu.
2. Puhu käyttäjän kieltä siten, että reaalimaailma ja järjestelmämaailma kohtaavat.
3. Anna käyttäjälle mahdollisuus olla vapaa ja mahdollisuus tehdä virheitä, mutta tehdyt virheet on myös kyettävä perumaan tai niistä toipumaan.
4. Ole yhdenmukainen ja noudata standardeja siten, että käyttäjän ei tarvitse arvailla, tarkoittavatko eri sanat ja toiminnot kenties samaa asiaa.
5. Suunnittele järjestelmä siten, että ongelmatilanteita ei esiinny.
6. Minimoi käyttäjältä vaadittava muistikapasiteetti.
7. Pyri joustavuuteen ja tehokkuuteen.
8. Ole esteettinen ja hyödynnä minimalistisia ratkaisuja.
9. Auta käyttäjiä tunnistamaan, diagnosoimaan ja palautumaan vikatilanteista.
10. Tarjoa hyvää dokumentaatiota tai ohjeita, vaikkei hyvä järjestelmä niitä vaatisikaan. (Nielsen, 2005.)

4.6 Erityishuomioita

Entä liittyykö käyttöliittymäin suunnitteluun joitakin erityishuomioita, jotka tulisi tiedostaa käyttöliittymäsuunnittelussa? Käyttöliittymän havainnollistavuutta ja ymmärrettävyyttä voidaan optimoida erityiskäyttäjäryhmiä varten esimerkiksi valitsemalla selkeitä käyttöliittymätekstejä, käyttämällä riittävän suurta kirjasinta, ottamalla käyttöön järkeviä värivalintoja, hyödyntämällä riittävää kontrastia ja vaikkapa soveltamalla ääniopasteita.

Esimerkiksi voimakasta punaista väriä voidaan käyttää varoitus- ja huomiovärinä. Ja jos tiettyä väriä käytetään käyttöliittymässä ilmentämään jotakin asiaa tai ilmiötä, tässä värin käytössä tulisi olla johdonmukainen ja käyttää väriä koko sovelluksen kattavasti samassa tehtävässä. Liian samansävyisiä värejä on syytä välttää. (Nummela 2002, 100.)

Tietynlaisina erityisryhminä voidaan pitää myös henkilöitä, joilla on näkövamma. Tällöin voi olla aiheellista harkita käyttöliittymän selkiyttämistä entisestään. Näkövammaiset hyötyvät erityisesti sovelluksessa käytetyistä ääniopasteista ja korkeasta kontrastista.. Nykyään on myös hyvät tekniset valmiudet toteuttaa vaikkapa puhesynteesi heikkonäköisten ja sokeiden henkilöiden avuksi. (Ollikainen 2002, 106.)

5 QT FRAMEWORK

5.1 Qt:n historia

Qt on C++-kielen käyttöliittymäkirjasto, jolla voidaan luoda näyttäviä sovellusten käyttöliittymiä helposti ja nopeasti. Qt:n kehitystyö aloitettiin vuonna 1991, ja sen takana voidaan pitää kahta norjalaista tietojenkäsittelijää, Haavard Nordia ja Eirik Chambe-Engiä. Heidän yhteisenä päämääränään oli luoda alustariippumattomalle C++-kielelle maailman paras käyttöliittymäohjelmistokehys

Nord ja Chambe-Eng perustivat yrityksen, joka nimettiin ensin Quasar Technologiesiksi ja tämän jälkeen Troll Techiksi. Myöhemmin yrityksen nimi vaihdettiin vielä Trolltechiksi. 1990-luvun alkupuoliskolta lähtien Qt laajeni tukemaan eri alustoja ja käyttöjärjestelmiä. (Blanchette, Summerfield, 2008, xix; Qt (framework), 2011.)

Vuoden 2008 kesäkuuhun mennessä Nokian kiinnostus Linux-puhelimiin ja Qt:hen oli kasvanut niin paljon, että Nokia päätti ostaa Trolltechin. Kaupan myötä alustariippumattoman Qt:n kehitys siirtyi Qt Development Frameworksille. (Lehtinen, 2008; Qt, 2011; Qt Development Frameworks, 2010).

Nykyään Qt:tä käytetään monissa eri käyttökohteissa, muun muassa matkapuhelinten ja tietokoneiden sovelluksissa. Alustariippumattomuutensa ansiosta Qt:llä tuotettuja natiiveja sovelluksia löytyy niin matkapuhelimista kuin PC- ja Mac-ympäristöistäkin. Tietokoneella tunnetuimpia Qt-sovelluskohteita ovat Skype, Opera-selain ja Google Earth. Qt:tä löytyy myös Linux-puolelta muun muassa KDE-sovelluksista. Mobiilipuolella Qt:tä on muun muassa MeeGo-nimisessä Linux-pohjaisessa käyttöjärjestelmässä ja ohjelmistoalustassa. Lisäksi Nokian ensimmäiseen Linux-pohjaiseen Maemo-puhelimeen eli N900:aan on saatavilla Qt-tuki, joka voidaan asentaa näppärästi laitteen omasta pakettienhallinnasta.

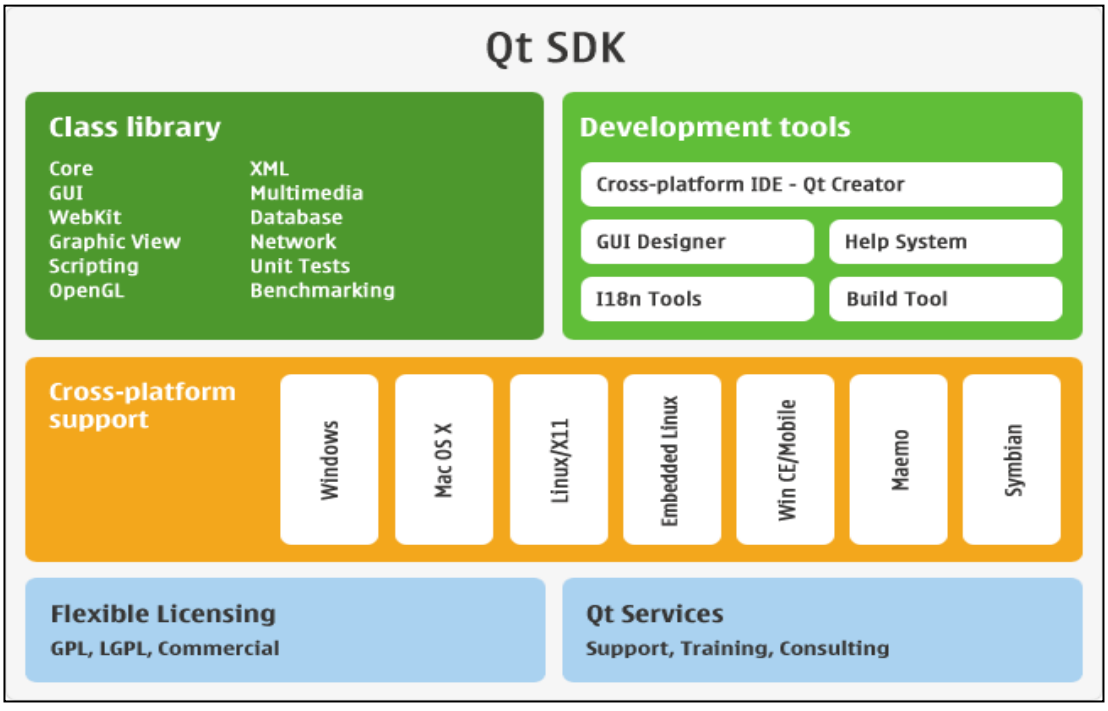
Qt:n kehitys on ollut kiitettävän aktiivista, mikä kertoo sen elinvoimaisuudesta ja suosittuudesta. Tämän opinnäytetyöprosessin aikana Qt:stä ehti ilmestyä kaksi stabiilia uutta versiota: versio 4.7.1 julkaistiin marraskuussa 2010 ja versio 4.7.2 julkaistiin maaliskuussa 2011 (McDonald, 2010; Shaw, 2011).

5.2 Qt-työkalut

Qt-sovellusten kehitystä varten on useita eri ohjelmistokehitysvaiheeseen räätälöityjä kehitystyökaluja. Työkalut ovat saatavilla Nokian sivuilta ladattavassa Qt SDK:ssa ja näistä kehitystyön kannalta välttämättömimpiä ovat:

- Qt Creator sovelluksen suunnitteluun ja kehitykseen,
- Qt Designer käyttöliittymien suunnitteluun ja toteutukseen,
- Qt Linguist sovellusten kääntämiseen eri kielille,
- Qt Assistant verkossa olevan dokumentaation esittämiseen ja
- Qt Simulator sovellusten testaamiseen simuloiduilla mobiililaitteilla.

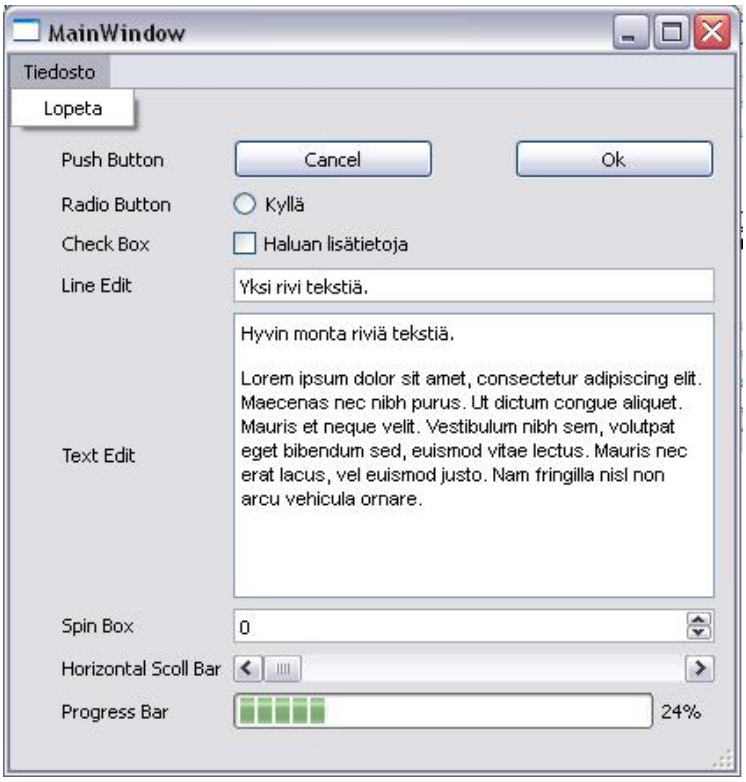
Qt:n tuorein versio 4.7.2 on saatavilla Qt SDK -ohjelmistokehityskokonaisuudesta, joko GPL:n, LGPL:n tai kaupallisen lisenssin mukaan. Se on ladattavissa Nokian Qt-sivuilta, osoitteesta http://qt.nokia.com/downloads/ . (Nokia, 2011.)



KUVIO 7. Qt SDK pähkinänkuoressa (Nokia, 2011)

5.3 Qt-käyttöliittymän peruskomponentit ja moduulit

Qt:stä löytyvät käyttöliittymäsuunnittelijalle kaikki tärkeimmät käyttöliittymäkomponentit, kuten painikkeet, dialogit ja tekstikentät. Qt:ssä näitä komponentteja kutsutaan yleisesti widgeteiksi ja niitä voi myös laatia itse, mikäli sopivaa valmista widgetiä ei ole vielä olemassa.



KUVIO 8. Erilaisia widgetejä aseteltuna Form Layoutiin eli kahteen palstaan

Qt:ssä on myös paljon sellaista, mitä sovelluksen käyttäjä ei päällisin puolin suoraan näe. Tällaisia ominaisuuksia kutsutaan moduuleiksi, joita Qt:stä löytyy muun muassa tietokantojen, verkkotoimintojen, vektorigrafiikan käsittelyyn. Sovelluksessa tarvittavat lisämoduulit täytyy ladata kuitenkin erikseen käyttöön. On myös mahdollista, että osa moduuleista täytyy ”kääntää” käsin Qt:n lähdekoodista.

5.4 Qt perustuu C++:aan

Koska Qt on C++:n graafinen luokkakirjasto, Qt-sovelluksissa voidaan käyttää C++-kielestä tuttuja ominaisuuksia, kuten osoittimia. Niillä on näppärää viitata keskus- eli

stack-muistissa sijaitsevan muuttujan arvoon. Tavallisesta muuttujasta poiketen osoitinmuuttuja tallennetaan niin sanottuun pino- eli heap-muistiin. Jos stack-muistiin luodaan paljon olioita tai muuttujia, on vaarana, että se pirstaloituu eli fragmentoituu tai loppuu kesken.

Sen sijaan heap-muistiin luotavat osoittimet ovat muistinkäytön kannalta tehokkaita ratkaisuja, koska niihin viittaaminen on nopeaa ja ne kuormittavat vähemmän prosessoria kuin tavalliseen muuttujaan viittaaminen. Tällainen dynaaminen muistinhallinta on erityisen huomionarvioista mobiililaitteille suunnitelluissa sovelluksissa niiden rajallisen muistin määrän vuoksi. Osoitinmuuttujan dynaamisesti varaama muisti on helposti vapautettavissa muuhun käyttöön delete-operaattorilla.

```
/*
 * Alla luodaan EmployeeDialog-luokan olioon viittaava
 * osoitin ”e”. Osoittimen tunnistaa asteriskista
 * eli *-merkistä.
 */

     EmployeeDialog* e = new EmployeeDialog(3);

/*
 * Käynnistetään e-olion suorittaminen
 */

     e->exec();

/*
 * Suorittamisen päätyttyä voidaan olio poistaa delete-
 * operaattorilla. Samalla vapautuu olion varaama
muisti.
 */

     delete e;
```

Koodiesimerkki 1. Osoittimen dynaaminen muistinhallinta ja vapauttaminen delete-operaattorilla

C++:n perinteisiä perustietotyyppejä ovat bool, char, double, int ja string. Qt:ssä on näitä vastaavien, Q-kirjainalkuisten perustietotyyppien – esimerkiksi QDouble – ohella erityinen QVariant-tietotyyppi. QVariant-muuttujiin voidaan sijoittaa kaikkien viiden edellä mainitun perustyypin arvoja. QVariantin sisältämän varsinaisen tyypin

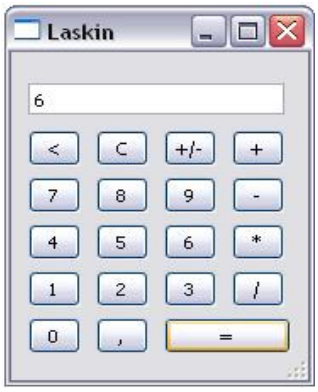
saa selville type()-funktiolla ja muuttuja on helposti konvertoitavissa tyypistä toiseen convert()-funktiolla. (Nokia, 2010.)

5.5 Signal-slot-mekanismi

Puhtaasti Qt-lähtöinen ajatus on tapahtumienkäsittelyyn tarkoitettu signal-slot-mekanismi, jonka on ideoinut Qt:n kehittäjä Eirik Chambe-Eng. Tämän Qt:lle keskeisen mekanismin perusajatuksena on, että jokin sovelluksen objekti voi lähettää tietyn tapahtuman yhteydessä signaalin ja tämä signaali kytketään johonkin slotiin. Slotit ovat aivan tavallisia funktioita, joihin signaali kytketään erityisellä connect()-funktiolla. Tyypillisiä signal-slot-käyttökohteita ovat nappien painallukset ja valikoissa tehdyt klikkaukset. (Blanchette, Summerfield, 2008.)

```
connect(ui->pushbutton_on_yhta_suuri, SIGNAL(clicked()), this, SLOT(laske()));
```

Koodiesimerkki 2. Yhtäsuuruuspainikken clicked()-signaali yhdistetään connect()-funktiolla laske()-funktioon eli slottiin.



KUVIO 9. Qt:llä toteutettu esimerkkisovellus ”Laskin”, jonka yhtäsuuruuspainiketta klikattaessa kutsutaan laske()-funktiota. Yhtäsuuruuspainikkeen klikkaamista kuvaava triggered()-signaali käynnistää siis laske()-funktion suorittamisen.

5.6 Useille alustoille kääntäminen ja kansainvälistäminen

Yli 400 luokasta koostuvan Qt:n erityispiirteinä voidaan pitää sen monipuolisuutta ja kykyä toimia sulavasti eri alustoilla ja käyttöjärjestelmissä. Siitä löytyvät työkalut

muun muassa multimedian, tietoverkkotoimintojen, XML:n, kolmiulotteisen grafiikan ja tietokantojen käsittelyyn. Qt:llä kertaalleen kirjoitettu lähdekoodi voidaan kääntää mille tahansa Qt-tuetulle laitteelle tai käyttöjärjestelmälle ilman että lähdekoodia tarvitsee muuttaa (Blanchette, Summerfield, 2008, xv).

Käytännössä tämä optimaalinen, Qt:n motostakin – Write Once, Run Everywhere – tunnettu tilanne ei täysin aina toteudu, sillä esimerkiksi työpöytäympäristöille suunniteltu ja toteutettu sovellus saattaa tarvita joitakin muutoksia toimiakseen mobiililaitteessa. Sovelluskehitykseen kuluva aika lyhenee kuitenkin merkittävästi, koska sovelluksen koodista suurin osa toimii suoraan.

Qt:n ansioksi voidaan lukea myös edistynyt sovelluksen kääntämismahdollisuus kielestä toiseen QtSDK:n mukana tulevan Qt Linguistin avulla. Perinteinen ohjelmointitapa on käyttää maailmanlaajuisesti hyvin ymmärrettyä englantia muuttujien, funktioiden ja luokkien nimissä sekä kommentoida sovellus tarpeellisin osin englanniksi. Kun kaikki käyttöliittymässä näkyvät tekstit sijoitetaan tr()-funktion sisään, ne voidaan kääntää myöhemmin Qt Linguistillä esimerkiksi suomeksi.

```
QPushButton* cancel = new QPushButton(tr("Cancel"));
```

Koodiesimerkki 3. Peruuttamispainikkeen luominen ja käyttöliittymätekstin sijoittaminen tr()-funktioon.

5.7 Qt:n heikot puolet

Qt:stä on vaikeaa löytää ongelmakohtia, ellei tällaiseksi lasketa sitä, että yksinkertaisetkin Qt:llä tehdyt sovellukset vievät suhteellisen paljon tilaa. Qt-ohjelmointi vaatii yleensä myös C++-osaamista, ja C++ mielletään usein vaikeaksi kieleksi oppia. Toisaalta, kun on C++:n oppinut, saa hyvät valmiudet myös muihin kieliin, vaikka C++:kin tarjoaa jo erittäin kattavasti mahdollisuuksia laajojenkin sovellusten laatimiseen.

Opinnäytetyön kirjoitushetkellä Qt:n vuonna 2008 ostanut Nokia teki mittavan laitteistoalustaa koskevan strategisen päätöksen siirtyä yhteistyöhön Microsoftin kanssa. Tämän seurauksena Qt:n tulevaisuutta mobiilipuolella on kyseenalaistettu, koska Nokia linjasi, että Windows Phone 7:sta tulee sen ensisijainen

älypuhelinalusta, eikä se näytä olevan saamassa Qt-tukea. Pari viikkoa myöhemmin eli 7.3.2011 Digia ilmoitti ostavansa Qt:n lisenssoinnin ja palvelut Nokialta. Kaupasta huolimatta Nokia jatkaa Qt:n kehittämistä, mutta Qt:n vastuulle siirtyvien 3500 asiakasyrityksen myötä Digia alkaa tarjota laajemmin Qt:tä erilaisissa laitteissa. (Ballmer, Elop 2011; Digia, 2011; Räihä, 2011.)

6 KÄYTTÖLIITTYMÄN TOTEUTTAMINEN QT:LLÄ

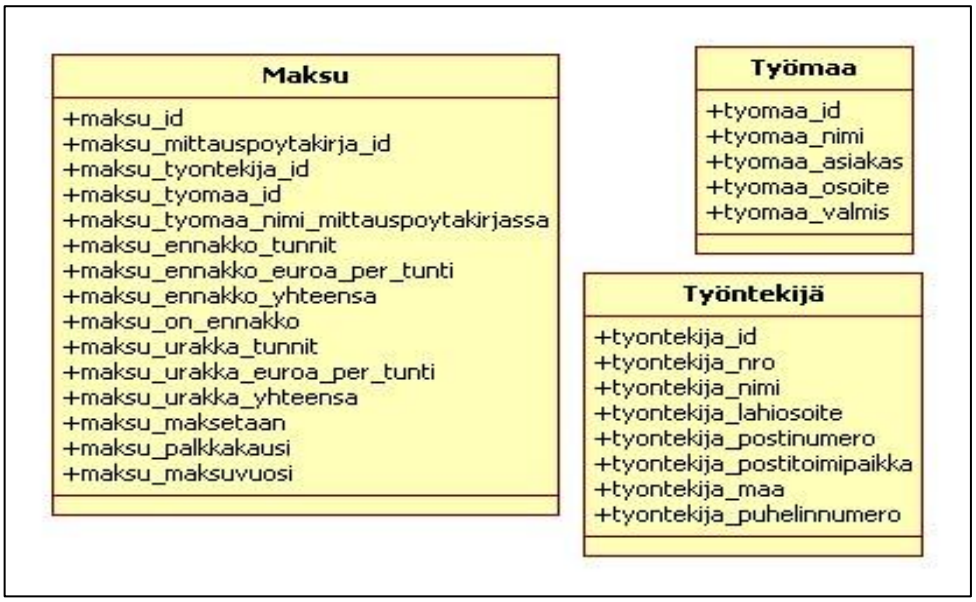
Tässä luvussa käytetään kaavio-, koodi- ja käyttöliittymäkuvaesimerkkeinä opinnäytetyönä toteutettua maksujen kirjaussovellusta. Opinnäytetyönä toteutettu sovellus vastaa tyypillistä keskustelukäyttöistä palvelusovellusta, koska sovelluksessa suoritetut toimenpiteet toteutetaan erityisten lomakkeiden avulla, eikä sovelluksessa voi muokata objekteja suoraan. Keskustelukäyttöisyyden ansiosta käyttäjän syötteitä voidaan käsitellä ja siten ennaltaehkäistä ongelmien syntymistä.

6.1 Käyttöliittymän suunnitteluvaihe

Ennen kuin käyttöliittymäprosessi on edennyt siihen pisteeseen, että varsinainen toteutus voi alkaa, on täytynyt kerätä oletetusta käyttäjäkunnasta aineistoa siitä, minkälaisia reaalimaailman asioita sovellus sisältää ja millaisia käyttäjän pyrkimykset sovelluksen suhteen ovat. Tämä aineiston kerääminen suoritettiin pitämällä muutamia vapaamuotoisia palavereja yhdessä toimeksiantajan kanssa. Näissä palavereissa tutustuttiin muun muassa käytössä olleeseen Excel-taulukkoon sekä sen käyttämiseen ja ongelmiin.

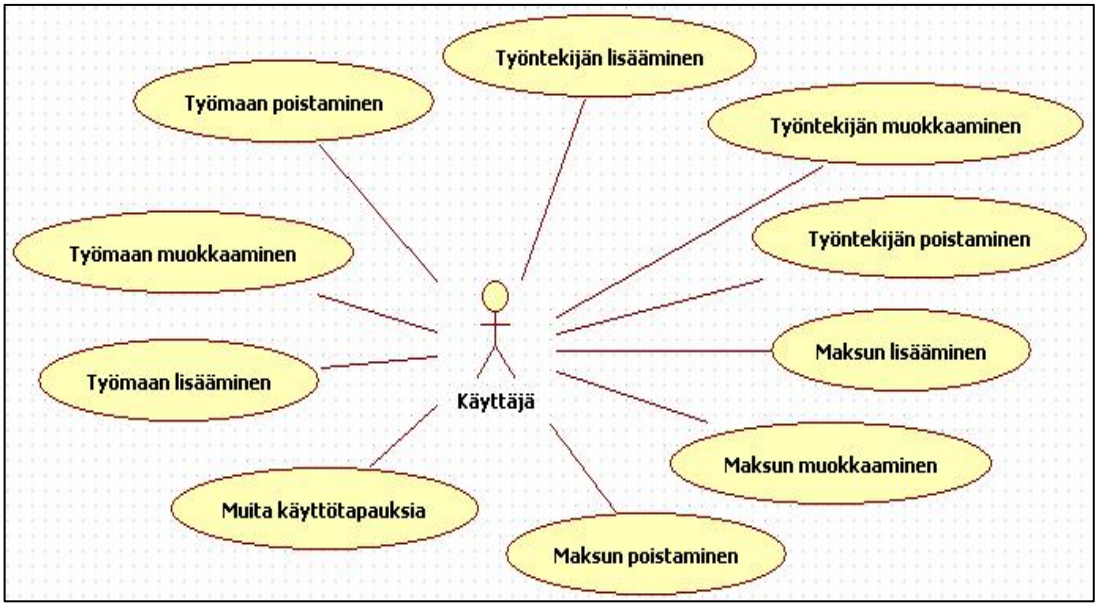
Lisäksi käytiin läpi, miten erityyppiset maksut lasketaan ja kirjataan Excel-taulukkoon ja minkälaisia ongelmia maksujen kirjaamisessa on. Palavereissa hahmoteltiin käyttöliittymän rakennetta paperimuotoisilla käyttöliittymän prototyypeillä ja päätettiin, että sovelluksessa olisi hyvä noudattaa Excelistä tuttua taulukkomaista rakennetta maksurivien visuaalisessa toteutuksessa.

Palaverien pohjalta muodostui ajatus sovelluksen jakamisesta kolmeen selkeään loogiseen kokonaisuuteen, jotka ovat: ”maksu”, ”työntekijä” ja ”työmaa”. Näitä kokonaisuuksia voidaan kutsua UML:n termein luokiksi ja ne voidaan sijoittaa luokka-kaavioon (Kuvio 10). Yleensä luokkakaavioissa kootaan yhteen tietyssä skenaariossa olevia asioita hyvin yleisellä tasolla sekä näihin liittyviä ominaisuuksia (attribuutteja) ja toimintoja (metodeja/funktioita).



KUVIO 10. Sovelluksen luokkakaavio

Näihin kaikkiin kolmeen kokonaisuuteen eli luokkaan voidaan sijoittaa käyttäjän tärkeimmät pyrkimykset, mitä tällä on sovelluksen suhteen. Käyttäjän pyrkimyksiä voidaan helposti lähestyä myös UML:stä tutun käyttötapauskaavion avulla. Sovelluksen kolmella loogisella kokonaisuudella on kolme pääasiallista käyttötapausta, jotka ovat lisäys, muokkaus ja poisto. Nämä vastaavat sovelluksen käyttäjän tavanomaisimpia pyrkimyksiä.



KUVIO 11. Maksujen kirjaussovelluksen tärkeimmät käyttötapaukset liittyvät aina tiedon lisäämiseen, muokkaamiseen tai poistamiseen.

Sovelluksen kolme selkeää kokonaisuutta käyvät selvästi ilmi myös sovelluksen pääikkunasta (KUVIO 9), jossa on sekä pikakomentopainikkeet että looginen valikko-rakenne. Komentopainikkeissa on käytetty kuvakkeita, jotka yksilöivät kutakin käyttö-tapausta ja joiden perusteella käyttäjän on helppo navigoida sovelluksessa. Tällainen jaottelu erilaisiin toiminnallisiin osiin on tärkeää käytettävyyden kannalta, koska se ihminen jäsentää maailmaa myös toisiinsa kuuluviin kokonaisuuksiin.

Kuvakkeita on käytetty johdonmukaisesti siten, että lisäyskuvakkeissa on vihreä plus-merkki, muokkauskuvakkeissa suurennuslasi ja poistokuvakkeissa punainen rasti. Valinnassa pyrittiin perinteiseen sovelluslogiikkaan ja kuvakkeiden semanttisiin ominaisuuksiin, esimerkiksi vihreä plus-merkki on helposti miellettävissä johonkin lisäävään toiminnallisuuteen, jossa jotakin tietoa tai ominaisuutta kartutetaan. Punainen rasti sen sijaan liitetään usein tietojen hylkäämiseen ja poistamiseen.

KUVIO 12. Sovelluksen pääikkuna

6.2 Käyttöliittymän toteutuksen esitoimenpiteet

Kun sovellus on suunniteltu sen sisältämän tietosisällön ja loogisen rakenteen osalta, voidaan aloittaa sovelluksen toteutusvaihe. Haluttaessa käyttää Qt Frameworkia

toteutustapana, on ensin täytynyt ladata Qt SDK Nokian sivuilta ja asentaa paketin mukana tulevat Qt-työkalut, minkä jälkeen käynnistetään Qt Creator.

Sovelluksen käyttöliittymän toteutus aloitetaan Qt Creatorissa luomalla uusi projekti. Uutta projektia luotaessa pitää valita, minkä tyyppistä sovellusta ollaan luomassa. Vaihtoehtoja ovat:

- Qt Gui Application eli graafinen sovellus,
- Mobile Qt Application eli mobiilisovellus ja
- Qt Console Application eli tekstipohjainen sovellus

Tyypillisimmin valitaan graafinen sovellus tai mobiilisovellus. Seuraavaksi projektille annetaan nimi ja valitaan, mitä Qt-versioita halutaan käyttää. Valittavissa on yleensä ainakin Desktop-, Maemo- ja Symbian Device -versioita. Valinnan jälkeen projektiin luodaan tyypillisesti MainWindow-luokka, joka toimii sovelluksen pääikkunana. Sovellus koostuu yleensä yhdestä pääikkunasta ja tähän liittyvistä dialogeista, joihin liikutaan käyttäjän tekemien valintojen mukaan.

Projektin luomisen jälkeen projektihakemistossa on .pro-tiedosto, joka sisältää qmake-kääntäjälle tiedot projektin kääntämisestä. Projektitiedostossa määritellään muun muassa, onko kyse graafisesta vai merkkipohjaisesta sovelluksesta, mitä mahdollisia lisämoduuleita sovellus tarvitsee ja mitkä tiedostot kuuluvat projektiin. Lisäksi projektihakemistossa on .ui-päätteisiä lomakkeita, .h-päätteisiä otsikko-tiedostoja eli headereita ja tärkeimpänä varsinaisia luokkia, jotka tunnistaa .cpp-päätteestä.

6.3 Tyypillinen Qt-sovellus noudattaa luokkamallista rakennetta

Kuten edellisessä kappaleessa todettiin, Qt-sovellus koostuu yleensä yhdestä Main-Window-luokasta ja määrittelemättömästä määrästä dialogeja. MainWindow'sta eli ohjelman pääikkunasta on – yksinkertaisia sovelluksia lukuun ottamatta – yleensä tarpeen siirtyä dialogiin, jossa tapahtuu tietyn toimintokokonaisuuden suorittaminen. Dialogit on helpointa luoda QtCreatorin Edit-näkymässä, jossa projektin tiedosto-

puusta valitaan Forms. Tämän jälkeen klikataan Forms-kokonaisuutta hiiren oikealla painikkeella ja valitaan Add New.

Seuraavaksi valitaan template eli pohja, jonka päälle aletaan dialogia rakentaa. Käytännössä on nopeinta valita Qt Designer Form Class, koska tällöin Qt Creator luo valmiiksi kaikki tarvittavat tiedostot. Valinnan jälkeen pitää vielä valita sopiva lomake-tyyppi. Valittavana on muun muassa dialogeja erilaisin komentopainike-asetteluin, dialogi ilman komentopainikkeita, pääikkuna ja widget. Kun sopiva pohja on valittu, annetaan luotavalle dialogille kuvaava nimi ja klikataan Next ja tämän jälkeen Finished. Nyt dialogi ja siihen liittyvät luokka- ja otsikkotiedostot on luotu.

Dialogikokonaisuus kirjoitetaan tyypillisesti olio-ohjelmoinnista tutulla tavalla luokan muotoon, ja se jakautuu kolmeen erimuotoiseen tiedostoon. QtCreator ei kuitenkaan pakota käyttämään luokkamuotoista jaottelua, eikä se ole järkevääkään yksin-kertaisissa sovelluksissa, mutta laajemmissa kokonaisuuksissa suositellaan käytettäväksi luokkamuotoista jaottelua tiedostoissa. Opinnäytetyönä toteutettuun sovellukseen luotiin neljä luokkaa, joilla jokaisella on .h-otsikkotiedosto, .cpp-luokkatiedosto ja .ui-käyttöliittymäluokkatiedosto. Tiedostot on nimetty seuraavasti:

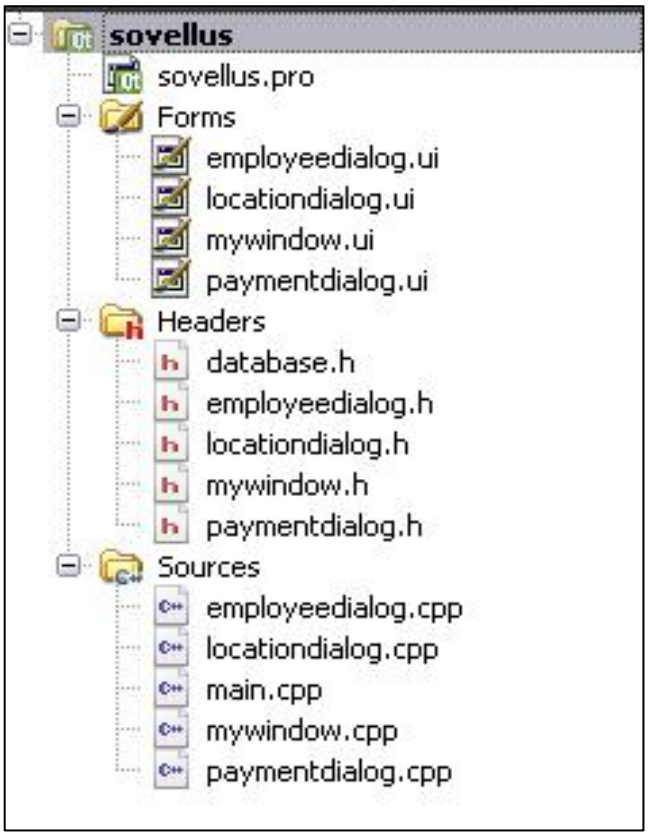
1. Ohjelman pääikkuna eli mywindow koostuu tiedostoista mywindow.h, mywindow.cpp, mywindow.ui.

2. Työntekijöiden hallinta tapahtuu employeedialog-luokassa, joka sisältää tiedostot employeedialog.h, employeedialog.cpp ja employeedialog.ui.

3. Työmaita koskevat tiedot käsitellään locationdialog-luokassa ja tässäkin luokassa on tyypilliset .h-, .cpp- ja .ui-tiedostot.

4. Maksuja kirjataan luokassa nimeltä paymentdialog käyttäen apuna tiedostoja paymentdialog.h, payment-dialog.cpp ja paymentdialog.ui.

Edellä mainittujen neljän käyttöliittymäluokan lisäksi sovelluksessa on Main.cpp-tiedosto, joka sisältää sovelluksen käynnistävän Main-funktion, ja yksinkertainen database.h-otsikkotiedosto, jossa on tietokantayhteyttä koskevia määrityksiä.



KUVIO 13. Sovellukseen kuuluvat projektitiedostot

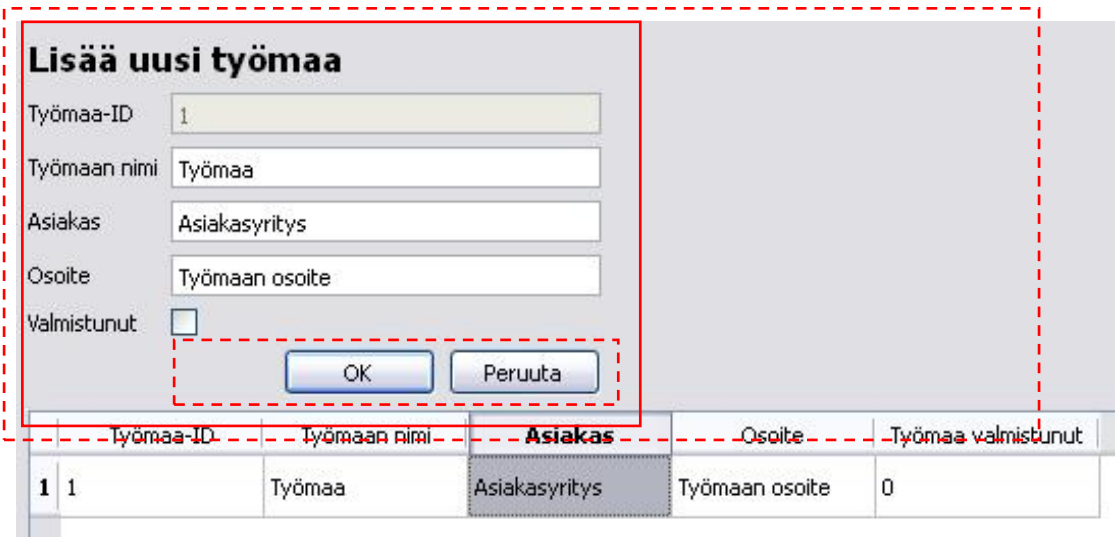
6.4 Asettelun valinta

Käyttöliittymä voidaan toteuttaa kahdella tapaa Qt Creatorissa: käyttöliittymän komponenteille voidaan määrittää kokonaan käsin tai sitten voidaan käyttää Qt Creatorin mukana tulevaa helppokäyttöistä Design-näkymää. Design-näkymää käyttämällä voi helposti säästää useita työtunteja. Käyttöliittymän toteuttaminen Qt Creatorilla kannattaa aloittaa valitsemalla sopiva asettelu kullekin ikkunalle ja dialogille. Tarvittaessa voidaan käyttää useita asetteluja yhtä aikaa. Asetteluja kutsutaan Qt:ssä layoutiksi, ja niitä on valittavissa neljä erilaista:

- vertical layout eli vaakasuuntainen asettelu,
- horizontal layout eli pystysuuntainen asettelu,
- grid layout eli ruudukkotyyppinen asettelu ja
- form layout eli lomaketyyppinen asettelu

Asettelun käyttö ei ole pakollista, koska Qt tarjoaa myös kaksi muuta tapaa widgetien asetteluun. Nämä tavat ovat absoluuttinen asettelu ja manuaalinen asettelu. Asettelun käyttöä kuitenkin suositellaan, koska niiden ansiosta dialogi mukautuu eri fonttien, kielten ja alustojen mukaan. Asettelun puuttumisen vaarana on, että esimerkiksi nimiötekstistä jää osa näkymättä. (Blanchette, Summerfield, 2008, 141.)

Samassa ikkunassa voidaan käyttää myös useita eri asetteluja yhtäaikaisesti. Kuvio 14:n sisemmässä katkoviivalla rajatussa alueessa eli komentopainikkeille varatussa tilassa on käytetty käytetty vertical layoutia ja painikkeiden vasemmalla puolella on niin sanottu ”horizontal spacer”. Ehjällä viivalla rajatussa alueessa on käytetty form layoutia ja koko kokonaisuus on koottu horizontal layoutiin. Lopputuloksena on visuaalisesti selkeä lomakekokonaisuus.



KUVIO 14. Useita asetteluja voidaan käyttää yhtä aikaa.

6.5 Widgetien sijoittaminen dialogiin

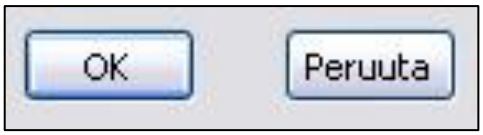
Layoutin asettelun jälkeen voi aloittaa komponenttien eli widgetien siirtämisen dialogiin. Siirtäminen on erittäin helppoa ja se toimii tutulla drag and drop -tyylillä. Erittäin tyypillinen form layoutiin eli lomaketyyppiseen asetteluun perustuva dialogi koostuu yleensä kahdesta palstasta: vasen palsta pitää sisällään QLabel-luokasta perityt nimiöt ja oikea palsta näihin nimiöihin liittyvät syötekentät-, valinta- ja komento-painikkeet. Widgetejä voivat olla esimerkiksi:

- painikkeita (QPushButton) ja valintoja (QRadioButton, QCheckbox),
- tekstisyötekenttiä (QTextEdit, QLineEdit),
- askellusruutuja (QSpinBox, QDoubleSpinBox) ja
- tiettyjä erityiskenttiä, kuten muun muassa päivämäärä- ja kellonaikakenttiä sekä web-näkymäkenttä

Widgetit ladotaan layoutiin yleensä vasemmalta oikealle ja ylhäältä alas, aivan kuten silmä liikkuu kirjaa lukiessa. Lopuksi dialogiin lisätään vielä sen sisältämien toimintojen hyväksymistä, hylkäämistä ja tietojen tyhjennystä koskevia komentopainikkeita, jotka on peritty QPushButton-luokasta.

6.6 Komentopainikkeet

Tyypillisiä painikkeita ovat hylkäämistä ja perumista kuvaava Peruuta/Cancel, tyhjentämistä ilmentävä Tyhjennä/Clear sekä tallennusta ja toteutusta ilmentävä OK-painike. Toisinaan OK-painikkeen tilalla käytetään Sulje/Close-painiketta, jos dialogilla tehtävät toimenpiteet suoritetaan sitä mukaa, kun käyttäjä niitä syöttää. OK-painiketta käytettäessä muutokset suoritetaan yhdellä kertaa painikkeen painamisen jälkeen. Edellä mainitusta voidaan päätellä, että Sulje- ja OK-painikkeet ovat yleensä toisensa pois sulkevia ja tietyssä dialogissa esiintyy niistä vain toinen. Se, kumpi painike lomakkeella on, riippuu tietojen suorittamistavasta. Komentopainikkeita ei tarvitse aktivoida välttämättä, sillä näppäimistön Esc-näppäimen painallus vastaa Peruuta-painikkeen aktivointia ja Enter-näppäimen painallus vastaa OK-painikkeen aktivointia. (Wiio, 2004, 187.)



KUVIO 15. Tyypilliset komentopainikkeet hyväksymistä ja hylkäämistä varten vastaavat käyttöliittymässä OK- ja Peru-painikkeita.

Sovelluksessa käytettiin kaikissa dialogeissa painikkeita tässä järjestyksessä ja tällaisin käyttöliittymätekstein, mikä luo käyttäjälle hyvin tärkeän yhteneväisyyden ja loogisuuden tunteen. Painikkeiden välissä on myös hieman tyhjää tilaa, mikä estää tehokkaasti virheklikkauksia. Järjestys perustuu perinteiseen painikkeiden järjestämistapaan eli siihen, että toimintoa edistävä painike on dialogissa ensimmäisenä ja tämän jälkeen toimintoa hylkäävä painike.

6.7 Syötteiden tarkistus eli validointi

Yleensä on järkevää, että käyttäjän syötteitä jollain tapaa validoidaan eli tarkistetaan, että kentissä täyttyvät sovellukseen määrittellyt kriteerit esimerkiksi käyttäjäsyötteen muodolle. Syötteen validointia koskevat säännöt kannattaa kirjoittaa koodiin käsin ja tämä vaatii QtCreatorissa siirtymisen Edit-näkymään, jossa avataan haluttu .cpp-luokkatiedosto.

Syötteiden validointia varten Qt:ssä on olemassa QRegExp- ja QValidator-luokat. QValidator on myös yläluokka muun muassa QIntValidator-, QDoubleValidator- ja QRegExpValidator-luokille. Nämä nimensä mukaisesti tarkastavat kokonaisluku- ja desimaalilukusyötteitä sekä säännönmukaisin lausekkein tarkasteltavia syötteitä. Tyypillisesti validaattoreita käytetään syötekentissä, kun halutaan rajoittaa käyttäjän antaman syötteen muotoa. Esimerkiksi kenttään, johon on kytketty QIntValidator, ei voi syöttää muita merkkejä kuin numeroita 0–9.

```
// esitellään desimaalilukuvalidaattori
QDoubleValidator*   doubleValidator;

// esitellään kokonaislukuvalidaattori
QIntValidator* intValidator;

doubleValidator     = new QDoubleValidator(this);
intValidator        = new QIntValidator(this);

// sallitaan vain desimaaliluvut kentässä "double_field"
ui->double_field->setValidator(doubleValidator);

// sallitaan vain kokonaisluvut kentässä "int_field"
ui->int_field->setValidator(intValidator);
```

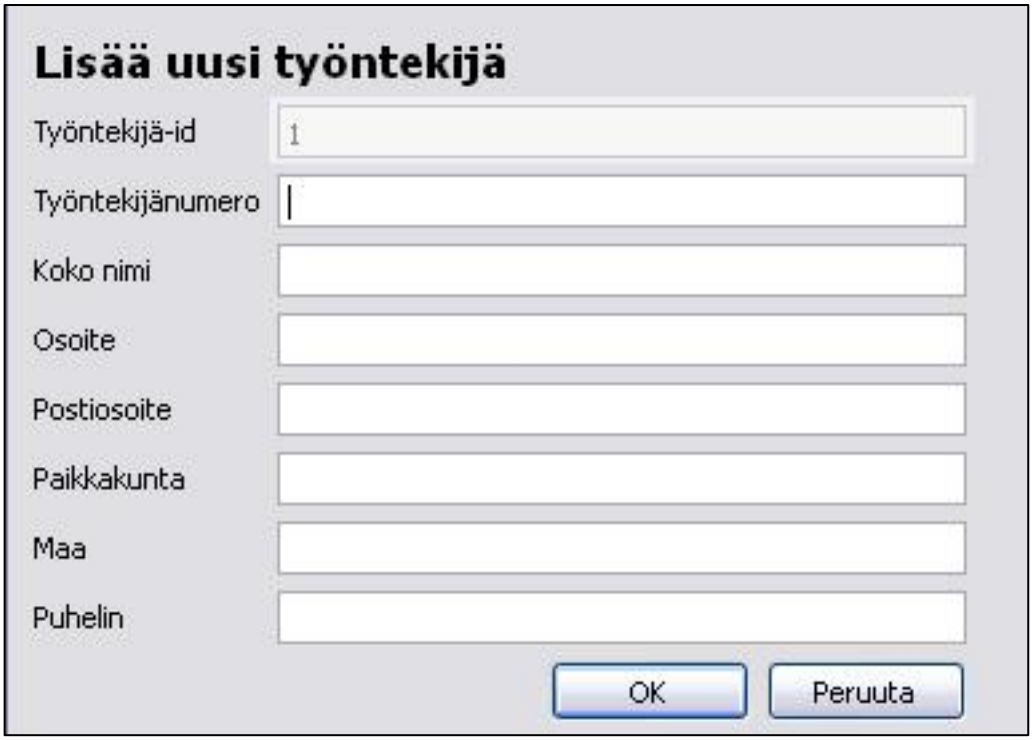
Koodiesimerkki 4. Yksinkertaiset validaattorit kokonais- ja desimaalilukujen tarkistamiseksi.

6.8 Näkyvyys- ja käytössä oloasetukset

Aina ei ole järkevää pitää kaikkia dialogin komponentteja näkyvillä tai aktiivisina ja käyttöönotettuina eli niin sanotusti ”enabloituina”. Käyttöliittymästä voidaan piilottaa mikä tahansa komponentti käyttämällä setVisible()-funktiota. Tämä funktio saa parametrinään arvon ”true” tai ”false” sen mukaan, halutaanko komponentin olevan näkyvillä.

Komponentin voi myös lukita käyttäjältä siten, ettei syötteitä tähän kenttään sallita. Tämä onnistuu setEnabled()-funktiolla, joka myös saa parametrinään arvon ”true” tai ”false”. Lukitsemisen käyttöä kannattaa harkita sellaisissa kentissä, jotka esimerkiksi lasketaan tai päätellään automaattisesti käyttäjän aiemmista syötteistä tai joihin käyttäjällä ei ole pääsyä ennen kuin tietty tapahtuma tai tietyt ehdot toteutuvat.

Kuviossa 16 on dialogi, jossa työntekijä lisätään sovellukseen. Dialogissa nähdään automaattisesti määrittyvä eli niin sanottu inkrementaali kokonaisluku, joka toimii yksi-löivänä tunnisteena eli työntekijän id:nä. Koska id:tä ei ole tarpeen muuttaa, id-kenttä on lukittu dialogissa setEnabled(false)-funktiota kutsumalla. Valitettavasti jouduin kä-sittelemään kuvaa korostaakseni setEnabled()-funktion vaikutusta id-kentässä.



KUVIO 16. Kaikilla sovellukseen lisättävillä objekteilla on uniikki tunniste eli id.

6.9 Valikot ja valikkojen toiminnot

Sovellusten pääikkunoissa on yleensä valikko. Tyypillisiä valikoita ovat esimerkiksi Tiedosto-, Muokkaa- ja Ohje-valikot. Qt Creatorissa valikotkin voidaan laatia joko Design-näkymässä tai Edit-näkymässä käsin kirjoittaen. Suosittelen käyttämään Edit-näkymää monimutkaisemmissa ohjelmissa, koska valikoissa ei oikeastaan ole muuta visuaalista käyttöliittymän osaa kuin itse valikkoteksti ja koska käsin kirjoitettuna valikko on helpommin hallittavissa silloin, kun sen näkee suoraan tekstinä Edit-näkymässä.

Valikoita varten täytyy luoda QAction-luokasta ilmentymä eli olio, jolla voidaan kytkeä valikossa oleva tekstielementti tiettyyn toimintoon. Lisäksi tarvitaan QMenu-luokasta tehty ilmentymä eli valikko-olio, johon QAction-tyyppiset toiminnot lisätään. Tämän jälkeen tarvitsee vielä kytkeä signal-slot-mekanismilla hiiren klikkaus eli käyttäjän tekemän valinta ja tiettyyn dialogiin siirtyminen tai muu toiminnallisuus toisiinsa (Koodiesimerkki 5, Kuvio 17).

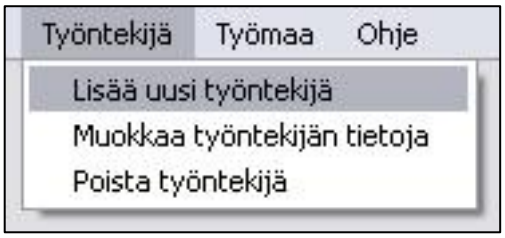
```
// Luodaan addEmployeeAction-toiminto.
QAction* addEmployeeAction =
         new QAction(tr("Lisää uusi työntekijä"), this);

// Luodaan empMenu-valikko.
QMenu* empMenu = employeeMenu =
         menuBar()->addMenu(tr("Työntekijä"));

// Lisätään toiminto valikkoon.
employeeMenu->addAction(addEmployeeAction);

// Yhdistetään toiminto connect()-funktiolla
// addEmplyoee()-slottin.
connect(addEmployeeAction, SIGNAL(triggered()),
         this, SLOT(addEmployee()));
```

Koodiesimerkki 5. QAction-luokasta tehdyn olion (addEmployeeAction) lisääminen valikkoon ”employeeMenu” ja toiminnon ”addEmployeeAction” kytkeminen funktiolla ”connect()” slotiin nimeltä ”addEmployee()”.



KUVIO 17. Työntekijä-valikkoon on luotu käyttötapausten mukaisia toimintoja. Tästä kuvasta voidaan huomata myös natiivin Windows-sovelluksen mukainen ulkoasu.

6.10 Tietojen tallentaminen

Suurimmassa osassa sovelluksia on tarpeen jossain vaiheessa lukea sovelluksen ulkopuolista tietoa eli dataa ja tallettaa tästä koostettua tai käyttäjän syöttämää uutta tietoa. Eräs hyvä tapa tällaisen tiedon tallentamiseen on käyttää tietokantaa, koska tietokannat mahdollistavat hyvinkin suurten tietomäärien säilyttämisen ja hallinnan sujuvasti. On olemassa myös muita tapoja, kuten tekstitiedostoon tallentaminen.

Opinnäytetyönä toteutettu maksujen kirjaussovellus käyttää tietokantana SQLitellä tehtyä tietokantaa ja tämän vuoksi sovellukseen toteutettiin tietokantayhteydettä koskevat määritykset. Tietokantayhteyttä varten luotiin database.h-otsikkotiedosto (Koodiesimerkki 6), joka sisältää staattisen funktion oletustietokantayhteyden avaamiseksi. Tietokantayhteyden ottamista varten ei ollut tarpeen luoda omaa luokkaansa, sillä Qt Framework itsessään tarjoaa tähän hyvät työkalut. Tietokannan rakenne on kuvattu liitteessä numero 1.

Jotta tietokannassa olevien tietojen käsittely olisi helppoa, jokaisella sinne tallennetulla rivillä eli tietueella on uniikki tunnisteensa eli id. Tunnisteen käyttö helpottaa oikean tietueen valitsemista erityisesti muokkausta ja poistoa koskevissa käyttötapauksissa.

```
#ifndef DATABASE_H
#define DATABASE_H
#include <QSqlDatabase>
#include <QDebug>
```

```
static bool createConnection()
{

 QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
 db.setDatabaseName("tbk.db");

    if (db.open()) {
        qDebug() << "Database opened!";

        return true;
    }

    else
    {
        qDebug() << "Error occurred while opening the
                     database!";

        return false;
    }

}

#endif // DATABASE_H
```

Koodiesimerkki 6. Tietokantayhteys avataan sovelluksen käynnistämisen yhteydessä staattisella createConnection()-funktiolla, joka palauttaa onnistumisen yhteydessä bool-tyyppisen arvon ”true” ja epäonnistumisen yhteydessä arvon ”false”.

6.11 Käyttäjän pyrkimyksen tunnistaminen

Sovelluksessa käsiteltävien asiakokonaisuuksien – työntekijän, työmaan ja maksun tietojen – hallitseminen tapahtuu kutakin kokonaisuutta koskevassa dialogissa. Ei ole järkevää luoda jokaiselle käyttötapaukselle omaa dialogiaan, vaan yhden kokonaisuuden dialogi voidaan parametrein mukauttaa käyttötapauksen mukaiseksi. Tällaista mukauttamista on esimerkiksi dialogin ja ikkunan otsikon määrittäminen vastaamaan käyttötapausta ja painikkeiden tekstien mukauttaminen tilanteeseen sopivin, käyttäjälle tutuin termein.

Kun ohjelman pääikkunasta valitaan esimerkiksi valikon kautta jotakin käyttötapahtumaa vastaava dialogi, tämä dialogi luodaan siten, että sille välitetään parametrinä käyttötapausta kuvaava int-tyyppinen kokonaislukumuuttuja. Kokonaislukumuuttujia käytettiin johdonmukaisesti kaikissa sovelluksen luokissa siten, että luku

yksi (1) tarkoittaa lisäämistä, luku kaksi (2) muokkaamista ja luku kolme (3) poistoa (Koodiesimerkki 6–8).

```
explicit LocationDialog(int type, QWidget *parent = 0);
```

Koodiesimerkki 6. Työmaadialogin eli LocationDialog-luokan muodostimen runko. Tälle dialogille välitetään parametrinä käyttötapausta kuvaava kokonaisluku, int type, joka voi olla arvoltaan joko 1 (lisäys), 2 (muokkaus) tai 3 (poisto).

```
LocationDialog::LocationDialog(int type, QWidget
*parent) : QDialog(parent),

[…]

    if (type == 1)
    {
        setWindowTitle(tr("Lisää uusi työmaa"));
        ui->label_location_title->setText(tr("<h2>Lisää
        uusi työmaa</h2>"));

[…]

    }

    if (type == 2)
    {
        setWindowTitle(tr("Muokkaa työmaan tietoja"));
        ui->label_location_title-
>setText(tr("<h2>Muokkaa
        työmaata</h2>"));
[…]
```

Koodiesimerkki 7. Kokonaislukuparametrin int type käsittelyä LocationDialog-luokan muodostimessa.

```
void MyWindow::modifyLocation() {
    LocationDialog* l = new LocationDialog(2);
    l->exec();
    delete l;
}
```

Koodiesimerkki 8. Kun pääikkunasta, siirrytään työmaan lisäykseen, luodaan LocationDialog-olioon viittaava osoitin ”2” ja tämä olio saa parametrinään kokonaisluvun 2. Kun dialogin toiminta on suoritettu, olion varaama muisti vapautetaan delete-operaattorilla.

Välitetyn int type –parametrin mukaan siis dialogille sopiva otsikko (Kuvio 18). Esimerkiksi, kun välitetään kokonaisluku 2, LocationDialog-olion otsikko on ”Muokkaa työmaata” (Koodiesimerkki 6, Kuvio 8).

KUVIO 18. Työmaan muokkaaminen dialogissa

6.12 Käyttötapaus vaikuttaa myös komentopainikkeisiin

Kaikissa kolmessa dialogissa on lisäksi OK-, ja Peruuta-painikkeet. Peruuta-painike toimii dialogista ja käyttötapauksesta riippumatta aina samalla tavalla eli sen aktivoimalla käyttäjä peruu tekemänsä muutokset ja sulkee diaogin. OK-painikkeen toiminta riippuu siitä, mistä käyttötapauksesta on kyse, toisin sanoen, mikä parametri dialogille on välitetty. Käyttötapauksen tyyppi – eli onko kyseessä lisäys, muokkaus vai poisto – vaikuttaa muun muassa siihen, millainen kysely tietokantaan lähetetään.

Lisäyksessä tietokantaan lähetetään INSERT-kysely, muokkauksesta UPDATE-kysely ja poistossa DELETE-kysely. Näistä ensin mainittu INSERT-kysely on yksinkertaisin, koska siinä ei vaadita tietueen tunnistamista uniikin id:n perusteella. Sekä UPDATE- että DELETE-kyselyssä tietokannassa oleva tietue tunnistetaan ensin sitä kuvaavan uniikin id:n perusteella ja vasta tämän jälkeen suoritetaan kysely tähän tietueeseen kohdistuvat toimenpiteet.

Tietokantaan lähetettävät sql-standardia noudattavat kyselyt käynnistyvät dialogin OK-painikkeesta, joka lähettää hiiren klikkausta tai Enter-näppäimen painallusta

vastaavan triggered()-signaalin. Tämä signaali on kytketty slotiin, jossa määritellään suoritettava toiminnallisuus.

```
void LocationDialog::on_location_dialog_ok_clicked()
{
if (type == 1)
    {
        QSqlQuery* query = new QSqlQuery;
        query->prepare("INSERT INTO tyomaa(tyomaa_id,
                                           tyomaa_nimi,

tyomaa_asiakas,
                                           tyomaa_osoite,

tyomaa_valmis)"
                       "VALUES (?, ?, ?, ?, ?)");
query->addBindValue(ui->field_location_id->text());
query->addBindValue(ui->field_location_name>text());
query->addBindValue(ui->field_location_client->text());
query->addBindValue(ui->field_location_address->text());

if (ui->field_location_finished->checkState() ==
                                             Qt::Checked)
        {
            query->addBindValue(1);
        }
        else
        {
            query->addBindValue(0);
        }

        query->exec();
[…]
        delete query;
    }
[…]
}
```

Koodiesimerkki 9. Kun LocationDialogin OK-painiketta klikataan, sovellus käynnistää käyttötapauksen mukaisen käsittelyn on_location_dialog_ok_clicked()-funktiossa.

7 POHDINTA

Tämän opinnäytetyön puitteissa toteutettiin sovellus Tampereen Bitumikate Oy:lle. Sovelluksella oli tarkoitus pystyä kirjaamaan yrityksen toimintaan oleellisesti liittyvien rakennusurakoiden maksutietoja. Sovelluksen toteutustavaksi valittiin Qt Framework, joka on C++:n käyttöliittymäkirjasto graafisten käyttöliittymien luomiseksi.

Opinnäytetyönä toteutetun sovelluksen kehitysprosessi oli hyvin tyypillinen ohjelmistotuotannon prosessi. Aluksi kerättiin taustatutkimusaineistoa muun muassa sovelluksen käyttäjää haastatellen ja näin saatiin tietoa muun muassa sovelluksen rakenteesta ja käyttöliittymän ilmeestä. Suunnitteluvaiheessa tutustuttiin käytössä olleeseen Excel-taulukkoon ja siihen, mitä kaikkea sillä on mahdollista tehdä ja mitkä ovat sen puutteita. Yhteiset palaverit havaittiin hyödyllisiksi, koska niissä päästiin konkreettisesti käsiksi käytössä olleen Excel-taulukon ongelmiin ja koska sekä toimeksiantaja että työn toimeksisaaja olivat läsnä samassa tilanteessa. Palavereissa käytiin myös keskustelua siitä, millainen sovelluksen käyttöliittymä olisi sovelluksen käyttäjän mielestä hyvä. Keskustelun tukena ja havainnollistamiskeinoina käytettiin yksinkertaisesti paperia ja kynää, joilla hahmoteltiin käyttöliittymän prototyyppejä.

Keskustelun pohjalta ja alan kirjallisuuteen tutustumalla punnittiin tarkoin sovelluksen ja sen käyttöliittymän toteutustapaa. Toteutustavaksi valittiin Qt Framework, koska se tarjoaa käyttöjärjestelmäriippumattoman tavan sovelluksen toteuttamiseksi ja sillä toteutettu sovellus on helposti käännettävissä useimmille tunnetuille alustoille ja käyttöjärjestelmille.

Koko sovelluskehitysajan toimeksiantajaa pidettiin aktiivisesti läsnä muun muassa lähettämällä noin parin viikon välein ohjelman demoversio, johon toimeksiantaja tutustui ja josta toimeksiantajalta pyydettiin palautetta. Tutustuttuaan demoon toimeksiantajan edustaja lähetti sähköpostitse kirjalliset kommentit ja nämä muutokset pyrittiin ottamaan käyttöön seuraavaan demoversioon mennessä. Tällaisia muutospyyntöjä oli muun muassa käyttöliittymätekstien suomentaminen aiemmin käytetyn englannin kielen asemesta. Opinnäytetyönä toteutetun maksujen kirjaus-

sovelluksen ohjelmistokehitysprosessia voidaan siten pitää käyttäjäkeskeisenä, minkä ansiosta sovellus vastaa käyttäjän asettamiin vaatimuksia.

Toteutettu sovellus vastaa opinnäytetyössä määriteltyyn tavoitteeseen ja tarkoitukseen mielestäni hyvin. Sovelluksella voidaan suorittaa vastaavia toimenpiteitä kuin aiemmin käytössä olleella Excel-taulukollakin. Koska sovellukseen syötettävä aineisto on pääosin määrämuotoista validoitujen syötekenttien ansiosta, on virheen tekeminen ja ”väärän” datan syöttäminen tehty vaikeaksi. Sovellus myös laajentaa toimeksiantajan liiketoimintaprosessia siten, että sovellukseen voidaan kirjata tarkempia tietoja työn-tekijöistä, työmaista ja rakennusurakoihin liittyvistä maksuista kuin mitä Excel-taulukkoon teknisistä rajoituksista johtuen oli mahdollista kirjata. Opinnäytetyön tavoitteen toteutuminen, eli maksujen kirjaamisen helppous, täsmentyy vielä sen myötä, kun toimeksiantaja on ottanut sovelluksen tuotantokäyttöönsä ja käyttänyt sitä jonkin aikaa.

Toimeksiantajan ensivaikutelma ja palaute sovelluksesta on ollut pääasiassa positiivista. Toimeksiantajan mukaan sovellus helpottaa maksujen kirjaamista ja vähentää tekemisen virheiden mahdollisuutta. Sovellus ei merkittävästi vähennä työhön käytettävää aikaa, mutta se on kokonaisuutena parempi kuin aiemmin käytössä ollut taulukkoon perustuva ratkaisu, joka toimeksiantajan mukaan oli myös sekava. Oikeastaan tällaista prosessin nopeuttamista on kyseenalaista edes pitää erityisen ensisijaisena tavoitteena, koska nopeutta tärkeämpää on tietojen oikeamuotoisuus ja virheiden välttäminen. Sovelluksen käytön nopeus ja viipeet kylläkin vaikuttavat käyttäjän kokemukseen sovelluksen käytettävyydestä, mutta tässä tapauksessa on kyse pikemminkin siitä, miten nopeasti käyttäjä pystyy syöttämään tietoja sovellukseen. Käytännössä sovelluksessa ei siten ole sellaisia teknisiä rajoituksia, jotka estäisivät tiedon syöttämistä nopeammin. Lisäksi yrityksen edustajan mukaan sovelluksesta voivat hyötyä kaikki sellaiset yritykset, jotka käyttävät urakkapalkkausta, eli sovelluksella voi olla lukuisia käyttökohteita muun muassa rakennusalan yrityksissä.

Opinnäytetyöprosessin aikana toimeksiantajan kanssa tehtiin päätös sovelluksen jatko-kehityksestä ja siihen lisättävistä ominaisuuksista. Erääksi tällaiseksi lisättäväksi ominaisuudeksi sovittiin tietojen vienti halutussa formaatissa palkanmaksua varten. Lopullisesta palkanmaksusta vastaa toimeksiantajan valitsema tilitoimisto, ja

ainakaan tällä hetkellä tilitoimistolla ei ole pääsyä sovelluksen käyttämään tietokantaan. Sovellukseen kuitenkin ehdittiin lisätä alustava tietojen viennin mahdollistava toiminnallisuus jo opinnäytetyöprosessin viime hetkillä ja tätä toiminnallisuutta onkin tavoitteena vielä viimeistellä jälkeenpäin.

Sovelluksessa on hyödynnetty Qt:n tarjoamia tekniikoita esimerkiksi tietojen syöttämisessä siten, että tietokantaan pyritään tallentamaan oikeamuotoista tietoa. Tämä tarkoittaa käyttöliittymässä olevien kenttien tietojen validointia. Sovelluksessa estettiin tietokannan tietueiden suora muokkaus siten, että tietojen muokkaamiseen käyttäjä joutuu käyttämään kuhunkin käyttötarkoitukseen optimoitua lomaketta. Suurimmassa osassa lomakkeiden kentistä on jonkinlainen validaattori, jonka mukaan käyttäjän syötteitä tarkistetaan. Esimerkiksi palkanmaksukautta tarkentavaan vuosi-kenttään voi syöttää vain kokonaislukuja, koska vuosiluvut koostuvat tunnetusti pelkistä kokonaisluvuista ja siten ei ole tarpeellista sallia esimerkiksi aakkosiin kuuluvia merkkejä. Validaattoreiden ansiosta tietojen syöttäminen väärin tai tahaton virheen tekeminen on tehty sovelluksessa tältä osin vaikeaksi, mutta täysin aukoton ei tällainenkaan ratkaisu ole.

Kuten IT-projekteissa yleensäkin, on mahdollista, että sovellukseen jää ”bugeja” ja kauneusvirheitä. Vaikka sovellus vastaakin tässä opinnäytetyössä määriteltyyn tavoitteeseen ja tarkoitukseen, niin siitä löytyy ainakin näin toteuttajan näkökulmasta vielä muutamia hienosäädettäviä ja viimeisteltäviä kohtia. Yksi tällainen seikka on muuttujien johdonmukainen nimeäminen, mutta tämä ei vaikuta mitenkään sovelluksen käyttäjän kokemukseen, eikä muutenkaan näy sovelluksessa päällepäin.

Olen tyytyväinen opinnäytetyönä toteutettuun sovellukseen, mutta koen, että olisi kannattanut entistä tarkemmin rajata kehitetyn sovelluksen ominaisuudet ennen toteutuksen aloittamista. Vanha sanonta ”hyvin suunniteltu on puoliksi tehty” piti siis paikkansa tässäkin projektissa. Todennäköisesti täytyisi tehdä vielä enemmän taustatyötä ja kartoittaa tarkemmin toimeksiantajan tarpeita ja pyrkimyksiä, jotka sovellukseen liittyvät. Lisäksi jonkin verran aiheutti ongelmia se, että yritin ensin luoda sovellusta Qt Framework 4.7.1 Snapshot –versiolla, ennen kuin ymmärsin, ettei tällaisessa Snapshot-versiossa ole välttämättä kaikkia Qt:n tarjoamia toiminnallisuuksia mukana. Versiolla 4.7.1 tehtyjen kokeilujen jälkeen siirryin täysimääräiseen

4.7.0-versioon, jolla sovelluksen teko onnistui luontevasti ja ilman suurempia ongelmia.

Yleisesti ottaen tutustuttuani Qt:hen olen sitä mieltä, että sillä voi olla edessä loistava tulevaisuus IT-alan eri käyttökohteissa, niin mobiili- kuin desktop-puolella. Tulevaisuus on kuitenkin hyvin riippuvainen siitä, millaisia ratkaisuja Qt:n omistaja, Nokia, tekee. Koska QtSDK tarjoaa erinomaiset työkalut käyttöliittymien ja sovellusten nopeaan kehittämiseen, olisi sääli, jos Qt kuihtuisi pois. Koin Qt:n mieluisimmaksi ja loogisimmaksi ohjelmointitekniikaksi, mihin olen tähän mennessä tutustunut. Edellä mainitun seikkojen ohella Qt-yhteisö vaikuttaa elinvoimaiselta, joten sovelluksen toteuttaja löytää helposti vertaistukea, mikäli kohtaa ongelmia. Lisäksi olen vahvasti sitä mieltä, että Qt Framework kaikessa monipuolisuudessaan ja nokkeluudessaan poikii vielä useita innovaatioita. Olen erittäin tyytyväinen sovelluksen toteuttamiseen Qt-tekniikalla ja se on varmasti edistänyt mahdollisuuksia IT-alan ammattilaisena.

LÄHTEET

Alaterä, A. & Halttunen, K. 2003. Tiedonhakujärjestelmien tarjoamat hakumenetelmät. Luettu 24.2.2011. http://internetix.fi/opinnot/opintojaksot/0viestinta/informaatiotutkimus/po2/tiedonha.htm.

Automatia Pankkiautomaatit Oy . Virtuaali-Otto. Luettu 27.4.2011. http://otto.fi/?ca=virtuaali_otto&an=default.

Ballmer, S. & Elop, S., 2011. Open Letter from CEO Stephen Elop, Nokia and CEO Steve Ballmer, Microsoft. Luettu 1.3.2011. http://conversations.nokia.com/2011/02/11/open-letter-from-ceo-stephen-elop-nokia-and-ceo-steve-ballmer-microsoft/.

Digia. 2011. Digia ostaa Qt-ohjelmistojen kaupallisen lisensointi- ja palveluliiketoiminnan Nokialta. Luettu 8.3.2011. http://www.digia.com/C2256FEF0043E9C1/0/405002251?opendocument&lang=fi.

Ekman, I., Ermi, L., Jäppinen, A., Kirvesmäki, L., Lankoski, P. & Nummela, J., 2002. Suunnitteluperusteita henkilökohtaiseen navigointiin. Teoksessa Kirvesmäki, L. & Lankoski, P. (toim.) Henkilökohtainen navigointi: periaatteita käyttösuunnitteluun. Studies in Information Sciences, SISCI. Tampere: Tampere University Press.

Ermi, L. 2002. Tavoitteena emotionaalisesti miellyttävä käyttökokemus. Teoksessa Kirvesmäki, L. & Lankoski, P. (toim.) Henkilökohtainen navigointi: periaatteita käyttösuunnitteluun. Studies in Information Sciences, SISCI. Tampere: Tampere University Press.

Kuutti, K. 1999. Mikä on "käyttöliittymä"?. Luettu 27.4.2011. http://www.tol.oulu.fi/kurssit/tkop/tkop5_1.html.

Käyttöliittymä. Päivitetty 24.2.2011. Luettu 8.3.2011. http://fi.wikipedia.org/wiki/K%C3%A4ytt%C3%B6liittym%C3%A4.

Lehtinen, J. 2008. Nokia hakee vauhtia ohjelmistoihin Linux-yhtiöstä. Julkaistu 28.1.2008. Luettu 28.2.2011. http://www.digitoday.fi/data/2008/01/28/nokia-hakee-vauhtia-ohjelmistoihin-linux-yhtiosta/20082663/66.

McDonald, J. 2010. Qt 4.7.1 Released. Luettu 8.3.2011. http://labs.qt.nokia.com/2010/11/09/qt-4-7-1-released/.

Nielsen, J. 2005. Ten Usability Heuristics. Luettu 2.3.2011. http://www.useit.com/papers/heuristic/heuristic_list.html.

Nummela, J. 2002. Visuaalinen suunnittelu pieninäyttöisiin laitteisiin. Teoksessa Kirvesmäki, L. & Lankoski, P. (toim.) Henkilökohtainen navigointi: periaatteita käyttösuunnitteluun. Studies in Information Sciences, SISCI. Tampere: Tampere University Press.

Nokia, 2010. QVariant Class Reference. Luettu 11.3.2011. http://doc.qt.nokia.com/latest/qvariant.html.

Nokia, 2011. A tour to the Qt (for Symbian). Luettu 28.2.2011. http://wiki.forum.nokia.com/index.php/A_tour_to_the_Qt_(Qt_for_Symbian).

Nokia, 2011. Qt Creator IDE and tools. Luettu 28.2.2011. http://qt.nokia.com/products/developer-tools.

Ollikainen, J. 2002. Erilaisia erityisryhmiä – erilaisia tarpeita. Teoksessa Pilke, E (toim.) Aktiivinen käyttöliittymä 2000. Hypermedialaboratorio. Tampereen yliopisto. Tampere: Jäljennepalvelu.

Qt. Päivitetty 19.1.2011. Luettu 28.2.2011. http://fi.wikipedia.org/wiki/Qt.

Qt Development Frameworks. Päivitetty 29.12.2010. Luettu 28.2.2011. http://fi.wikipedia.org/wiki/Qt_Development_Frameworks.

Qt (framework). Päivitetty 28.2.2011. Luettu 28.2.2011. http://en.wikipedia.org/wiki/Qt_(framework).

Räihä, K. 2011. Millainen on Qt-kehitysympäristön tulevaisuus puhelinpuolella. Luettu 1.3.2011. http://www.puhelinvertailu.com/uutiset.cfm/2011/02/13/millainen_on_qt-kehitysympariston_tulevaisuus_puhelinpuolella.

Shaw, A. 2011. Qt 4.7.2 has been released! Luettu 8.3.2011. http://labs.qt.nokia.com/2011/03/01/qt-4-7-2-has-been-released/.

Shedroff, N., 1994. Information Interaction Design: A Unified Field Theory of Design. Luettu 1.3.2011. http://www.nathan.com/thoughts/unified/unified.pdf.

Sinkkonen, I. & Tuominen, J. 1996. Käyttöliittymän visuaalinen suunnittelu. Teoksessa Kalimo, A. (toim.) Graafisen käyttöliittymän suunnittelu. Opas ohjelmistojen käytettävyyteen. Espoo: Suomen ATK-kustannus Oy.

Visiopaja Oy. 2008. Ensimmäinen ATK-sanasto. Luettu 8.3.2011. http://sano.se/suomeksi/index.php?do=hae&one=1&haku=k%E4ytt%F6liittym%E4.

Wiio, A. 2004. Käyttäjäystävällisen sovelluksen suunnittelu. Helsinki: Edita.

LIITE 1

TIETOKANNAN RAKENNE

```
--Table: maksu
--DROP TABLE maksu;
CREATE TABLE maksu (
  maksu_id                                integer PRIMARY KEY,
  mittauspoytakirja_id                    numeric,
  maksu_tyontekija_id                     numeric,
  maksu_tyomaa_id                         numeric,
  maksu_tyomaa_nimi_mittauspoytakirjassa  text,
  maksu_ennakko_tunnit                    numeric,
  maksu_ennakko_euroa_per_tunti           numeric,
  maksu_ennakko_yhteensa                  numeric,
  maksu_on_ennakko                        numeric,
  maksu_urakka_tunnit                     numeric,
  maksu_urakka_euroa_per_tunti            numeric,
  maksu_urakka_yhteensa                   numeric,
  maksu_maksetaan                         numeric,
  maksu_palkkakausi                       text,
  maksu_maksuvuosi                        numeric
);

--Table: tyomaa
--DROP TABLE tyomaa;
CREATE TABLE tyomaa (
  tyomaa_id       integer PRIMARY KEY,
  tyomaa_nimi     text,
  tyomaa_asiakas  text,
  tyomaa_osoite   text,
  tyomaa_valmis   numeric
);
```

```
--Table: tyontekija
--DROP TABLE tyontekija;
CREATE TABLE tyontekija (
  tyontekija_id                integer PRIMARY KEY,
  tyontekija_nro               numeric,
  tyontekija_nimi              text,
  tyontekija_puhelinnumero     text,
  tyontekija_lahiosoite        text,
  tyontekija_postinumero       text,
  tyontekija_postitoimipaikka  text,
  tyontekija_maa               text
);
```

LIITE 2

TIETOKANTAYHTEYDET

Tietokannan ”Maksu”-taulun kentässä “maksu_tyontekija_id” viitataan ”Tyontekija”-taulun ”tyontekija_id”-kenttään. Samoin ”Maksu”-taulun kentässä nimeltä ”maksu_tyomaa_id” viitataan ”Tyomaa”-taulun ”tyomaa_id”-kenttään. Tällä tavoin saadaan id:n perusteella haettua työntekijän ja työmaan nimi.