Jaakko Kilpeläinen

Rakennusautomaatiojärjestelmän hälytyskäsittelyn parantaminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Automaatiotekniikka

Insinöörityö

27.9.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jaakko Kilpeläinen Rakennusautomaatiojärjestelmän hälytyskäsittelyn parantaminen 40 sivua + 5 liitettä 1.10.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Automaatiotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Energia-automaatio
Ohjaaja(t)	tekninen johtaja Jarkko Turunen lehtori Antti Liljaniemi

Insinöörityö tehtiin rakennusautomaatioratkaisuja toteuttavalle Arealtec Oy:lle. Työ tehtiin, koska yrityksen valvomottomissa kohteissa käyttämä ratkaisu näyttää vain voimassa olevat hälytykset. Tarkoituksena on parantaa nykyistä hälytysten käsittelyä yksinkertaisella Windows-käyttöjärjestelmäiselle taulutietokoneelle asennettavalla ohjelmalla, joka näyttää myös poistuneet hälytykset ja kirjoittaa lokia tapahtumista.

Raportin alussa esitellään projektissa käytettävä ohjelmistotuotannon malli, toteutuksessa käytettävä ohjelma, koodauskieli sekä alustavat suunnitelmat. Tämän jälkeen tarkemmat suunnitelmat, ohjelman rakenne ja testaus.

Ohjelma toteutettiin Qt-kehitysympäristöohjelmalla ja kielenä käytettiin C++:aa. Toteutetun ohjelman rakenne noudattaa alussa tehtyä suunnitelmaa, mutta koodi ei noudata täydellisesti olio-ohjelmoinnin olioiden kapseloimisen mallia, vaan se on enemmänkin proseduraalisen kielen ja oliokielen hybridiä. Työn tuloksena saatiin toimiva ja asiakkaan vaatimukset täyttävä ohjelma.

Avainsanat	Qt Creator, C++, Ohjelmointi, Rakennusautomaatio

Author(s) Title Number of Pages Date	Jaakko Kilpeläinen Enhancing of a Building Automation System's Alarm Handling 40 pages + 5 appendices 1 October 2013
Degree	Bachelor of Engineering
Degree Programme	Automation Technology
Specialisation option	Energy Automation
Instructor(s)	Jarkko Turunen, CTO Antti Liljaniemi, Senior Lecturer

The Bachelor's thesis was made for Arealtec Oy which executes building automation solutions. The thesis was made because in buildings without building control software the company's currently used solution for alarm management is imperfect. It does not show exited alarms or provide an acknowledgement feature for customer. The aim is to improve the current alarm handling with simple installable software running on Windows operating system, which shows exited alarms and provides acknowledgement feature.

This report first presents used software production model, coding program, programming language and preliminary plans. After that comes detailed plans, program structure and integration testing.

The program was implemented with Qt development software and the used language is C++. The structure of the published program complies with the initial project plan, but the code does not fully follow the encapsulation rules of object oriented programming. It rather resembles hybrid code, which mixes structures between procedural and object based languages. The result was a functional program that meets the requirements of the client.

Keywords	Qt Creator, C++, Programming, Building Automation

Sisällys

Termit

DEOS	Saksalainen yritys, joka kehittää ja valmistaa laitteita ja sovelluksia rakennusautomaatioon.
Alakeskus	Vapaasti ohjelmoitava laite, johon voidaan liitää tuloja sekä lähtöjä
VAK	Valvonta-alakeskus sisältää alakeskuslaitteet sekä niiden vaatimat apulaitteet johtoteineen.
Telnet	Kaksisuuntainen yhteysprotokolla pääteyhteyksien muodostamiseen palvelimeen internetin tai lähiverkkojen ylitse.
Kääntäjä	Kääntää tekstipohjaisen ohjelmointikoodin tietokoneen ymmärtämään muotoon.
Kirjasto	Kokoelma valmista koodia
Puskurimuisti	Väliaikainen talletuspaikka
Dynaaminen olio	New-komennon avulla luotava olio, jonka ohjelmoija tuhoaa
Staattinen olio	Olio tuhotaan ja sen käyttämä muisti tuhotaan ohjelman suorituksen loputtua
Olio	Looginen joukko yhteenkuuluvia muuttujia ja funktioita
Luokka	Olion rakennuksen pohjapiirustukset
Kapselointi	Pyrkimys luoda uudelleenkäytettäviä luokkia, joita toinen ohjelmoija pystyy käyttämään ilman, että tuntee luokan toteutusta

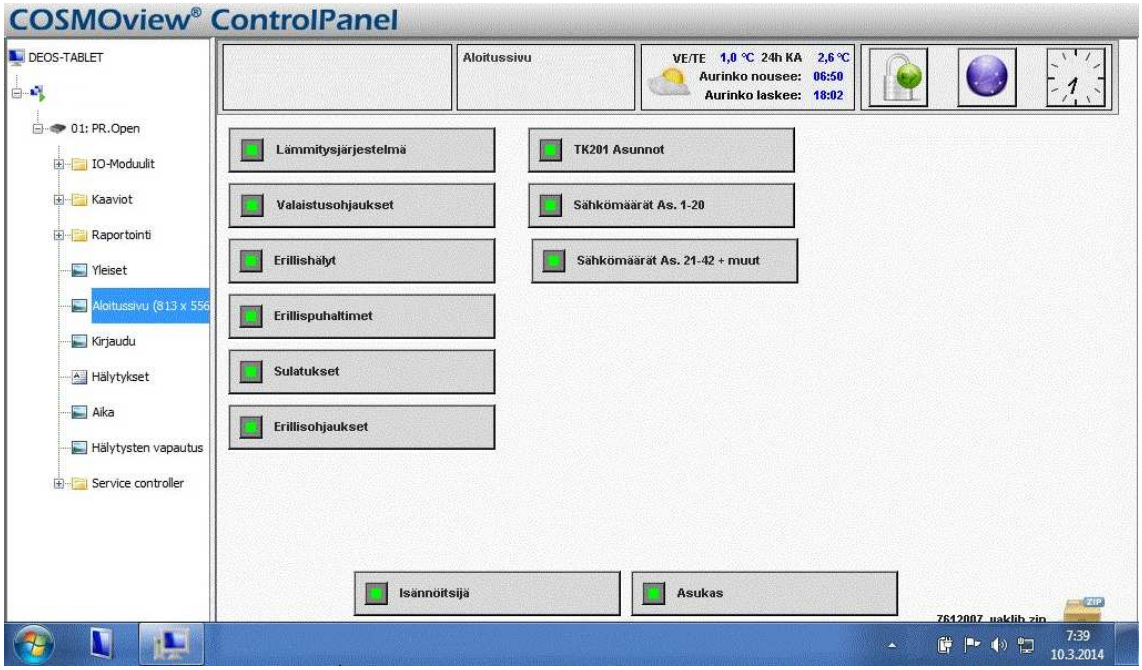
1 Johdanto

1.1 Opinnäytetyö

Työn aiheena on parantaa rakennusautomaatiojärjestelmän hälytysten käsittelyä tietokoneohjelman avulla. Tämän kirjallisen raportin tarkoituksena on antaa lukijalle tietoa käytetyistä ratkaisuista, kertoa miksi niihin päädyttiin ja esitellä ohjelman rakenne.

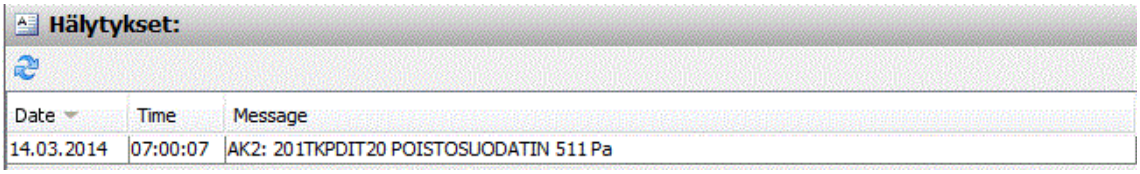
Opinnäytetyö tehtiin Arealtec Oy nimiseen yritykseen. Arealtec Oy on vuonna 1994 perustettu rakennusautomaatiourakoita toteuttava yritys, joka työllistää tällä hetkellä noin 40 työntekijää Vantaalla ja Hyvinkäällä. Arealtec Oy on osa suurempaa EMC Talotekniikka–yhtiötä. EMC-Talotekniikka on puolestaan osa suurempaa eurooppalaista Royal Imtech N.V konsernia. Royal Imtech N.V työllistää noin 29 000 työntekijää ympäri Eurooppaa ja sen vuosittainen liikevaihto on noin 5,5 miljardia euroa. Omistajanvaihdoksista huolimatta Arealtec Oy:n alkuperäinen nimi on päätetty säilyttää.

Arealtec Oy tarjoaa asiakkailleen kahta automaatiojärjestelmää, joista yritykselle uudempaa valmistaa saksalainen DEOS AG. Valmistaja tarjoaa laajemman hälytyskäsittelyominaisuuden vain raskaamman valvomo-ohjelmiston mukana. Järjestelmää kuitenkin pystyy käyttämään kuvan 1 mukaisen DEOS:in kevyemmän COSMOview-ohjelmiston tai selaimen avulla, mutta tällöin hälytysnäkymässä näkyy vain aktiiviset hälytykset. Arealtec Oy käyttää tätä kevyempää järjestelmää erityisesti pienissä kohteissa, joiden rakennusautomaatiojärjestelmää hallitaan kosketusnäytön avulla. Työn tarkoituksena on luoda ohjelma, joka näyttää käyttöliittymässään poistuneet hälytykset, kirjoittaa hälytyslokia ja tarjoaa käyttäjälle mahdollisuuden kuitata hälytyksiä.



Kuva 1. Rakennusautomaatiojärjestelmän käyttöliittymän etusivu

Käytettävä järjestelmä koostuu taulutietokoneesta tai PC:stä ja valvonta-alakeskuksen sisällä sijaitsevasta alakeskuksesta. Tietokoneella otetaan alakeskukseen TCP-IP-yhteys, joko valmistajan ohjelman tai selaimen avulla. Käyttäjälle avautuu tavallinen rakennusautomaatiovalvomon käyttöliittymä, josta pystyy tarkastelemaan muun muassa lämmityksen tai ilmastointikoneiden prosesseja reaaliajassa. Tässä näkymässä on valittavana myös kuvan 2 mukainen valmistajan hälytysnäkymä.



Kuva 2. Valmistajan hälytysnäkymä

1.2 Ohjelmistoprojektien yleiset ongelmat

Nykyiset ohjelmistot ovat valtavia ja monimutkaisia kokonaisuuksia, joita yhden ihmisen on mahdotonta hahmottaa. Eri ihmisten rakentamien ohjelman eri osien pitää toimia saumattomasti keskenään ja niiden tulee olla helppoja liittää toisiinsa.

Eri tietolähteitä tutkiessa törmää vaihteleviin lukuihin ja syihin miksi IT-projektit usein epäonnistuvat. Jotkut tietolähteet antavat epäonnistumisprosentiksi jopa 70 %, mutta keskiarvoa tutkittaessa luku on noin 45 %. Suurimpia syitä synkkään epäonnistumisprosenttiin ovat osapuolten välisen viestinnän puute, ajankäytön ja siitä seuraavan budjetin vaikea ennustettavuus, epäonnistumisprosentin mittausmenetelmä, epäselvä projektin tavoitteiden ja laajuuden määrittely sekä huono projektin seuranta ja kontrollointi. Näitä syitä yritetään karsia alati kehittyvillä ketterillä ohjelmistotuotannon malleilla. Yhteistä ketterillä malleilla on, että ne poikkeavat tavallisen projektin suoraviivaisesta järjestyksestä eli ensin määrittely, sitten suunnittelu ja lopuksi toteutus ja testaus. Tämän vuoksi ketterät menetelmät soveltuvat paremmin isoihin ohjelmistoprojekteihin, koska ne antavat liikkumavaraa ja joustoa. Tässä opinnäytetyössä sovelletaan perinteistä lineaarista mallia, koska kyseessä on pieni ja selkeä ohjelma. [2,3.]

1.3 Projektissa käytettävä ohjelmistotuotannon malli

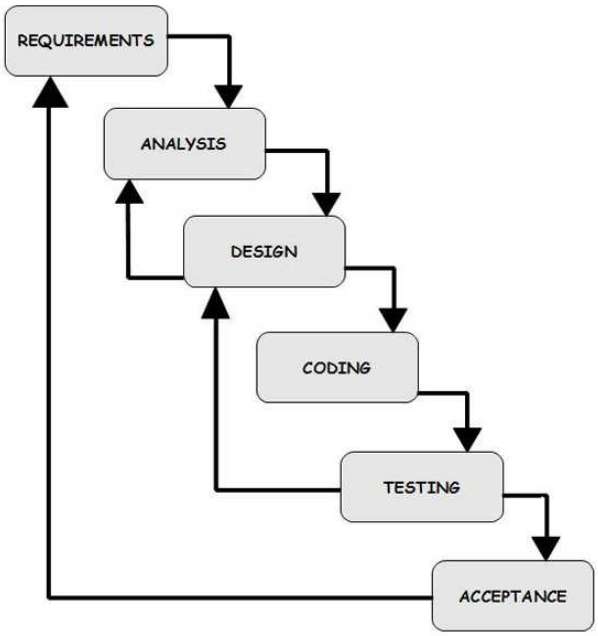
Vesiputousmalliksi kutsutussa kuvan 3. mukaisessa lineaarisessa ehkä kaikkein tunnetuimmassa ohjelmistotuotannon mallissa edetään vaihe vaiheelta eteenpäin ilman, että palataan edelliseen vaiheeseen paikkaamaan virheitä. Tässä toimintamenetelmässä etuna on, että vaihe suunnitellaan huolellisesti ennen kuin siirrytään seuraavaan vaiheeseen, joka todennäköisesti poistaa virheitä ja vähentää ajankäyttöä projektin lopusta sekä projektin kulun selkeä etenemisjärjestys. Vesiputousmallin luonteeseen kuuluu myös suuri dokumentoinnin määrä, joka sopii yhteen opinnäytetyön kanssa. Vesiputousmallin suurin heikkous on, että on yleisesti mahdotonta suunnitella projektia täydellisesti etukäteen ilman, että olisi tarpeen palata edelliseen vaiheeseen. Edellä mainittujen syiden vuoksi vesiputousmalli soveltuu yleisesti vain pieniin ja tarkasti määriteltyihin ohjelmistoprojekteihin. [12,13.]

Kuva 3. Perinteinen vesiputousmalli

Tässä projektissa käytetään kuvan 4 mukaista sovellettua vesiputousmallia paikkaamaan perinteisen mallin heikkouksia. Sovellettua vesiputousmallia käyttäen projektin loppupuolella mahdollisesti esiintyvät komplikaatiot on mahdollista poistaa helpommin iteroimalla vaiheiden välillä.

- Mallin ensimmäisessä vaiheessa luodaan projektin vaatimusmäärittely, yleensä yhdessä asiakkaan kanssa. Tässä projektissa asiakas tiesi tarkkaan mitä halusi, joten hän loi tämän raportin yksin (opinnäytetyön 2.luku).
- Seuraavassa vaiheessa ohjelmoija etsii pintapuoliset järjestelmätason ratkaisut asiakkaan vaatimuksiin ja esittelee nämä hänelle. Esitettyjen ratkaisujen sopiessa asiakkaalle siirrytään seuraavan vaiheeseen, muutoin palataan takaisin etsimään ja suunnittelemaan sopivia ratkaisuja (opinnäytetyön 3.luku).
- Hyväksynnän jälkeen suunnitellaan ohjelman rakenne ja toiminnot tarkemmin (opinnäytetyön 4.luku).
- Neljäntenä alkaa itse koodin tuottaminen (opinnäytetyön 5.luku).
- Testausvaiheessa testataan lähes valmiin ohjelman soveltuvuutta asiakkaan toimintaympäristöön. Mahdollisten ongelmakohtien löytyessä palataan takaisin suunnitteluvaiheen kautta koodausvaiheeseen. Tätä kiertoa iteroidaan, kunnes on päästy osapuolten väliseen yhteisymmärrykseen (opinnäytetyön 6.luku).
- Viimeisenä vaiheena on asiakkaan hyväksyntä, jossa valmista tuotosta verrataan vaatimusmäärittelyyn (opinnäytetyön 7.luku).

Ohjelmistotalon tuottamassa projektissa viimeisimpänä vaiheena on kuvan 3 mukainen ylläpito, joka jää tästä projektista pois projektin kertaluonteisuuden vuoksi.



Kuva 4. Muokattu vesiputousmalli [7.]

2 Asiakkaan vaatimukset ohjelmalle

Asiakas laati ohjelmalle haluamistaan ominaisuuksista tarkemman vaatimusmäärittelyn. Alla on tiivistettynä ja selkeytettynä asiakkaan toiveet. Tämä alkuperäinen vaatimusmäärittely löytyy liitteenä (liite 1) työn lopusta.

Ohjelmointikieli on vapaasti valittavissa, mutta kuitenkin niin, että ohjelman täytyy toimia itsenäisesti ilman erikseen asennettavia kirjasto- tai alustasovelluksia. Ohjelman tulee toimia Windows 7 sekä Windows 8 käyttöjärjestelmien 32- ja 64-bittisissä versioissa. Se osaa tunnistaa käytettävän resoluution, eli tunnistaa automaattisesti onko käytössä pöytäkone vai kosketusnäyttö ja skaalata ikkunan kokonsa sen mukaisesti.

Uuden tapahtuman sattuessa ohjelma tulee voida automaattisesti asettaa päällimmäiseksi sovellukseksi. Ohjelma näyttää hälytyksen käyttöliittymässään eri väreillä, jos se on voimassa oleva tai käyttäjä ei ole kuitannut hälytystä. Ohjelmassa näytettävien hälytysten järjestystä voidaan muuttaa, esimerkiksi aakkosjärjestykseen tai kronologiseen järjestykseen. Oletuksena tähän on, että uusin hälytys on ylimpänä. Ohjelma osaa tunnistaa poistuneen hälytyksen ja sitä vastaavan tapahtuneen hälytyksen indeksin perusteella. Tällöin käyttöliittymän Poistunut-sarakkeeseen kirjoitetaan tapahtuman kohdalle poistuneen hetken ajankohta ja rivin väriksi vaihdetaan vihreä. Jos alakeskus ilmaisee poistuneen hälytyksen, mutta hälytystiedoissa ei löydy tapahtunutta hälytystä, ohjelma näyttää uuden rivin, jossa on merkitty sekä tapahtunut- että poistunut-aikaleima samoiksi (tällöin ei tiedetä, milloin tieto tapahtuneesta hälytyksestä on tullut).

Ohjelman tulee tallentaa automaattisesti hälytyslokia kronologisessa järjestyksessä omaan hakemistorakenteeseen. Tämän tallennusmuotona on txt-muotoinen tekstitiedosto, jossa jokainen hälytys tallennetaan omalle rivilleen muodossa: Aikaleima<TAB>teksti. Lisäksi ohjelman tulee kirjoittaa omaa kirjanpitoa näytettäviä hälytyksiä varten ja jotta uudelleenkäynnistyksen jälkeen ohjelma palautuu samaan tilaan kuin ennen sammuttamista. Kirjanpidossa täytyy olla ainakin seuraavat kentät: tapahtuman indeksi, tapahtuman aikaleima, onko tapahtuma tapahtunut tai poistunut, tapahtuman teksti ja onko tapahtuma kuitattu. Kirjanpidosta poistetaan rivi, jos tapahtuma on poistunut sekä kuitattu.

Ohjelman käyttämät kohdekohtaiset parametrit, kuten alakeskuksen IP-osoite, portti, Kohde-teksti ja VAK-teksti, tulee olla helposti asetaltavissa ja muutettavissa. Tähän sopii esimerkiksi INI-muotoinen tiedosto.

Ohjelman näyttää käyttöliittymässään, onko yhteys alakeskukseen muodostettu. Jos yhteyttä ei ole, ilmaistaan tämä punaisella värillä. Tällöin ohjelma yrittää muodostaa yhteyden alakeskukseen 20 sekunnin välein.

3 Toteutusratkaisuihin päätyminen

3.1 Kehitystyökalun valinta

Ohjelmaa tullaan käyttämään pääasiassa kosketusnäytöllä, joten tämä on hyvä ottaa huomioon toteutusohjelman valinnassa ja käyttöliittymän suunnittelussa. Tällaisen ratkaisun tarjoaisi todennäköisimmin ohjelma, jolla pystyy toteuttamaan alustariippumattomia ohjelmia kosketusnäytöllisten älypuhelimien OS-mobiilikäyttöjärjestelmille. Toteutusohjelman on hyvä sisältää valmiita graafisia komponentteja, kuten paino- ja valintanappeja, sekä listoja, joilla toteuttaa helpommin kosketusnäytöillä käytettäviä käyttöliittymiä. Lisäksi käyttöliittymän testausta helpottaisi taulutietokoneen resoluution huomioiva simulointiohjelma.

Ratkaisun tähän tarjoaa ennen Nokian ja nykyään Digian omistama Qt-kehitysympäristö. Qt on avoimen lähdekoodin ohjelma eli se on ilmainen ja käyttäjät pystyvät kehittämään sitä. Sillä on toteutettu muun muassa tunnettu VLC media player. Se on alustariippumaton työkalu erityisesti graafisten käyttöliittymien toteutukseen, joka tarjoaa sisäänrakennetun tuen C++-kielelle. Ohjelmointi onnistuu myös C#-, Java-, Python-, Ruby, PHP-kielillä, sekä korkeamman tason JavaScript-pohjaisella deklaratiivisella Qt Meta Language (QML)-kuvauskielellä. QML:llä on helpompi toteuttaa näyttäviä ratkaisuja, kuin perinteisillä ohjelmointikielillä. Tulevaisuudessa QML:llä tulevat käyttämään ainakin Jollan Sailfish- ja Ubuntu Mobile -mobiilikäyttöjärjestelmät. [10]

3.2 Ohjelmointikielen valinta

3.2.1 Oliopohjainen ohjelmointi

Maailmassa kaikki voidaan kuvata olioiden avulla ja ihmisellä on luonnollista hahmottaa asioita jakamalla kokonaisuus osiin. Esimerkiksi oppikirjat pyritään muotoilemaan siten, että kokonaisuus on jaettu pienempiin loogisiin osiin, joita on helpompi sisäistää. Olio-ohjelmointi perustuu tähän ajatukseen.

Ohjelmistokustannukset voidaan jakaa kuvan 5 mukaisesti kahteen osaan: rakennuskustannuksiin ja ylläpitokustannuksiin. Oliopohjainen ohjelmointi mahdollistaa sen, että

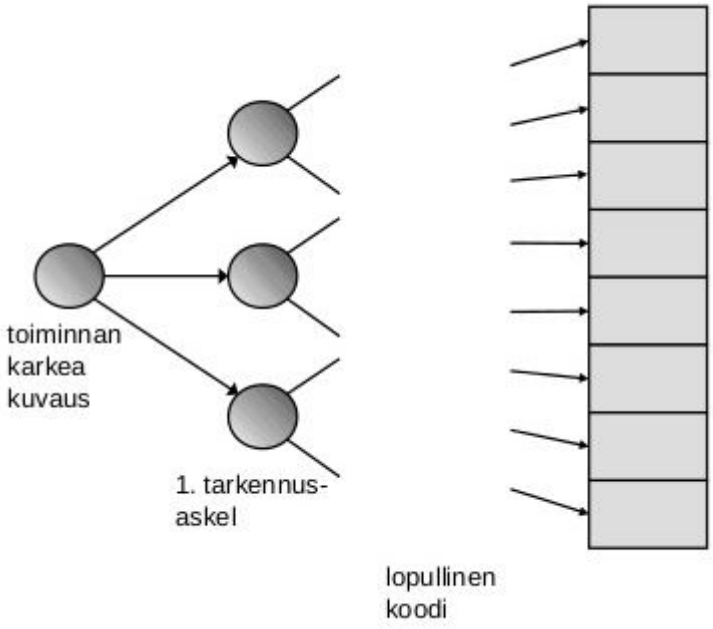
rakennuskustannuksista 15 % koodista koostuu uudesta koodista ja jäljelle jääneeseen 85 %:iin koodista voidaan käyttää muokattua vanhaa koodia. Oliopohjaisesti tehty ohjelma on myös helpommin ylläpidettävää, mikä leikkaa kustannuksia, kun puolet tai enemmän kustannuksista tulee julkaisun jälkeisestä ylläpidosta. Tämä näkyy opinnäytetyön ohjelmassa kehitysympäristön kirjastojen valmiiden funktioiden käytössä, jotka vähensivät ajankäyttöä ja itse kirjoitetun koodin määrää huomattavasti. [6 kalvo...10]

Kuva 5. Ohjelmiston kustannusten jakautuminen. [9, kalvo 10]

Ohjelmoinnilla tarkoitetaan yleisesti tietokoneelle tai jollekin muulle systeemille suunniteltavia ja toteutettavia toimintaohjeita, joka mukaan sen tulisi suorittaa haluttu tehtävä. Tosielämään tämän voi mallintaa ajatuksella, että joku suunnittelee ja kirjoittaa reseptin ruoan valmistukseen, jonka sitten joku toinen valmistaa. Resepti koostuu vaiheista, jonka mukaan tulee toimia:

1. Tarvitset 3 dl riisiä ja 6 dl vettä.
2. Kuumenna vesi kiehuvaksi.
3. Lisää riisi.
4. Keitä 20min välillä sekoittaen, jonka jälkeen vesi on haihtunut ja riisi valmista.

Tavallisilla proseduraalisilla ohjelmointikielillä (kuva 6), kuten C:llä, nämä tietokoneelle annettavat ohjeet ovatkin ylhäältä alaspäin etenevän listan muodossa. Tämä aiheuttaa sen, että pieni muutos ohjelman ylätasolla vaikuttaa suuresti alemmille tasoille. Lisäksi räätälöityä koodia on hankalaa käyttää uudelleen ja sen ylläpito on vaikeampaa. [5,6]



Kuva 6. Proseduraalisen kielen rakenne [6 kalvo 5]

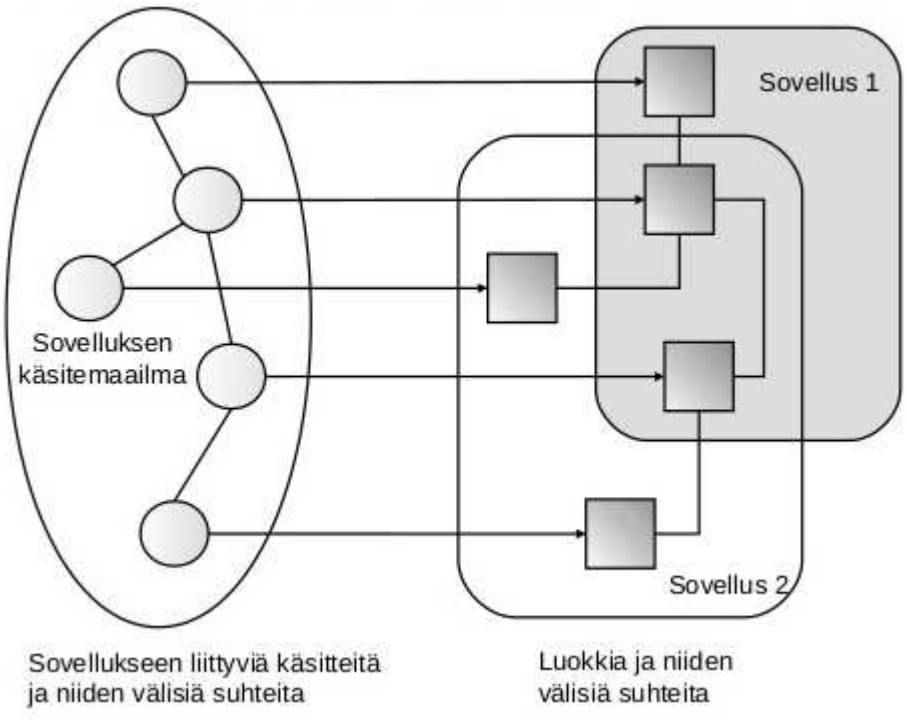
Olio-ohjelmoinnin (kuva 7) pääajatuksena on näiden ongelmien kuvaaminen pysyvämpien itsenäisten osien avulla. Ohjelman eri osista pyritään tekemään mahdollisimman riippumattomia toisistaan. Tämä asioiden abstraktimpi tarkastelu lisää koodin uudelleenkäytettävyyttä ja ylläpitoa, eikä ylemmälle tasolle tehtävä muutos vaikuta voimakkaasti alemmille tasoille. Oliopohjaisen ohjelmoinnin suurimpana heikkoutena on, että suunnittelu vie enemmän aikaa, koska koodin uudelleenkäytettävyys tulee pitää koko-ajan mielessä läpi ohjelmointiprosessin.

Luokka-käsitteellä tarkoitetaan olio-ohjelmoinnissa pohjapiirustusta, jolla määritellään millaisia olioita muodostetaan. Proseduraalisessa ohjelmoinnissa *attribuutteja* ja *operaatioita* käsitellään kahtena erillisenä rakenteena. Olio-ohjelmoinnissa nämä kaksi käsitettä on yhdistetty. Oliolle määritetään luokassa *attribuutit* eli olion ominaisuudet ja *operaatiot* eli olion toiminnot. Riisin valmistuksen esimerkistä voidaan muodostaa luokka kokki. Kokilla on tosielämässä samat *attribuutit* kuin ihmisellä, eli pituus, paino, ikä,

työaika ja niin edelleen. Kokilla on myös paljon *operaatioita,* joista yksi on keittäminen. Kokki luokasta muodostettu *olio*, jolla on *operaatio* keittäminen, voisi näyttää olio-kielellä seuraavalta:

- Markku.keitä(riisi);

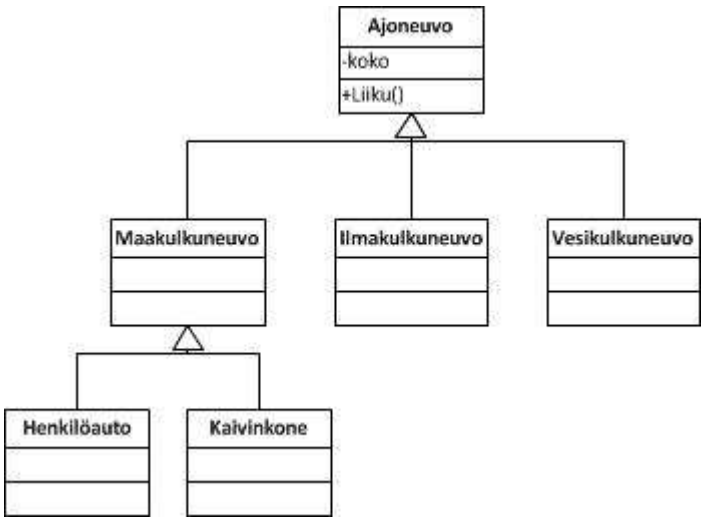
Esimerkissä Markku on *olio*, keitä *toiminto*, jonka Markku suorittaa ja riisi *attribuutti*. Yksinään tämänlainen toteutus ei vähennä koodin määrää verrattuna proseduraalisiin kieliin, mutta esimerkiksi muodostettaessa ravintolaa ei koko koodia tarvitse kopioida jokaisen työntekijän kohdalla. Kokki-luokasta voidaan muodostaa lukematon määrä uusia kokkeja, joilla on samat ominaisuudet. Lisäksi samaa koodia voitaisiin hyödyntää luotaessa ravintolapäällikköä ja kylmäkköä. Tietokoneesta hyvä esimerkki on näytössä esiintyvä pikseli. Kaikilla pikseleillä sijainnin kertovat *attribuutit* x ja y, sekä *operaatio* halutun värin aikaansaamiseksi. Suuri määrä pikseleitä pystytään luomaan yhden luokan kautta, joka säästää koodin määrää ja helpottaa muokkausta.



Kuva 7. Oliopohjaisen kielen rakenne [6 kalvo 8]

Perinnällä tarkoitetaan olio-ohjelmoinnissa, että perivä luokka perii kantaluokan ominaisuudet, kuten kuvan 8. mukaiset koon tiedot ja liikkumisominaisuuden. Esimerkiksi

ajoneuvo-luokasta voisi loogisesti periyttää luokat maakulkuneuvo, vesikulkuneuvo ja ilmakulkuneuvo, joista voisi taas periyttää hienojakoisempia luokkia, kuten henkilöauto ja kaivinkone, joilla on samoja ominaisuuksia kuin ryhmän muilla luokilla. Tämä ominaisuus vähentää koodin määrän ja parantaa uudelleenkäytettävyyttä, kun jokaiselle luokalle ei tarvitse erikseen kirjoittaa samoja ominaisuuksia.



Kuva 8. Perintäesimerkki

3.2.2 C++

C++ on yksi tämän hetken suosituimmista ohjelmointikielistä. Se lasketaan olio-ohjelmointikieleksi, vaikka C-kielen kaltainen proseduraalinenkin ohjelmointi on mahdollista. C++ ei ole uusi kieli vaan uusilla ominaisuuksilla varustettu C-kieli. TIOBE programming community index:iä pidetään yhtenä luotettavimmista tavoista määrittää kuinka suosittu ohjelmointikieli on. Suosio muodostuu suurimpien hakukoneiden hakujen perusteella käyttäen tietynlaisia matemaattisia malleja. C++ sijoittuu tässä mittarissa neljänneksi, samoin kuin edellisvuonna (kuva 9). Sitä ennen ovat suurella erolla muihin C ja Java, sekä Applen suosima C-kielestä jatkettu oliopohjainen Objective-C. Lyhyesti voidaan todeta, että mitä suositumpi kieli on, sitä enemmän sille löytyy käyttäjiä ja hyviä sovelluksenkehitystyökaluja. Vähemmän suositut kielet voivat taas soveltua paremmin johonkin tiettyyn käyttötarkoitukseen, kuten esimerkiksi verkkosivujen ohjelmointiin. [11]

Position Oct 2013	Position Oct 2012	Delta in Position	Programming Language	Ratings Oct 2013	Delta Oct 2012	Status
1	1	=	C	17.246%	-2.58%	A
2	2	=	Java	16.107%	-1.09%	A
3	3	=	Objective-C	8.992%	-0.49%	A
4	4	=	C++	8.664%	-0.60%	A
5	6	↑	PHP	6.094%	+0.43%	A
6	5	↓	C#	5.718%	-0.81%	A
7	7	=	(Visual) Basic	4.819%	-0.30%	A
8	8	=	Python	3.107%	-0.79%	A
9	23	↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑	Transact-SQL	2.621%	+2.13%	A
10	11	↑	JavaScript	2.038%	+0.78%	A

Kuva 9. TIOBE:n ohjelmointikielten sijoitustaulukon kymmenen parasta

Tanskalainen vuonna 1950 syntynyt professori Bjarne Stroustrup alkoi kehittämään C-kieltä vuonna 1979. Projekti kulki nimellä ”C with classes”. Projekti valmistui vuonna 1983, jolloin nimi vaihdettiin C++:ksi. Kieleen on lisätty vuoden 1983 jälkeen lisäominaisuuksia, joista virallisen standardin muodossa vuodesta 1998. Viimeisin standardi ISO/IEC 14882:2011 on julkaistu vuonna 2011. [1 kalvo 17]

Oppikirjojen suosituin esimerkki esittää ohjelmointikielen rakenne on ”hello world”-ohjelman koodi. Tämä siksi, että aloittelijan on riittävän helppo ymmärtää koodia ja kokeneempi saa siitä riittävästi tietoa uuden ohjelmointikielen rakenteesta. Hello world-ohjelma tulostaa ruudulle tekstin:

Hello, world!

C++ kielellä ohjelma näyttää kuvan 10. mukaiselta.

```
# include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
```

Kuva 10. Hello world ohjelman koodi C++-kielellä

Hello world-ohjelma alkaa rivillä: ” #include <iostream>”. Tässä #include-komennolla esitellään kääntäjälle, että halutaan käyttää C++-standardin mukaisia virrankäsittely-operaatiota (iostream), toisin sanoen virrankänkäsittelyfunktiot otetaan käyttöön omalle ohjelmalle.

Funktion suoritus alkaa hakasulusta ja loppuu hakasulkuun, eikä välilyönneillä tai rivinvaihdoilla ole merkitystä ohjelman toimintaan. Toisella rivillä oleva: ”int main()” on pääfunktio, josta ohjelman suorittaminen yleensä alkaa.

Nimiavaruudella tarkoitetaan yleisesti toisiinsa yhteen liitettyjä ohjelmakokonaisuuksia, jotka voivat olla mitä tahansa ohjelmakoodia, joita käyttäjä voi käyttää ja luoda itse. Nimiavaruuksia käytetään helpottamaan erityisesti isoissa ohjelmistoissa syntyviä nimien päällekkäisyyksiä, joita syntyy ohjelmiston eri moduulien välille, sekä jakamaan ohjelmaa loogisiin osiin. Näkyvyystarkenninoperaatiota (::) käytetään osoittamaan, että viitataan ulkoisen nimiavaruuden- tai luokan operaatioon. Käyttäjä ikään kuin kertoo, minkä kokonaisuuden alkiota hän haluaa käyttää.

Hello world esimerkissä ”std::cout”-määreellä halutaan käyttää C++-standardikirjaston tarjoamaa std-nimiavaruuden cout-alkiota, joka tulostaa halutun lainausmerkkien sisässä olevan tekstin. Virran suunnan osoittamiseen käytetään <<-merkintää. Vastaavasti, jos haluttaisiin lukea näppäimistön syötettä käytettäisiin sen lukemiseen cin-alkiota ja suunnan osoittamiseen >>-merkintää. Lainausmerkkien sisällä oleva \n-merkintä tarkoittaa, että kursori vaihtaa riviä saavutettuaan kyseisen merkkiyhdistelmän. Viimeisenä koodia lukeva kääntäjä törmää puolipisteeseen (;), joka tarkoittaa sille, että on tultu lauseen loppuun ja siirrytään suorittamaan seuraavaa osaa koodista.

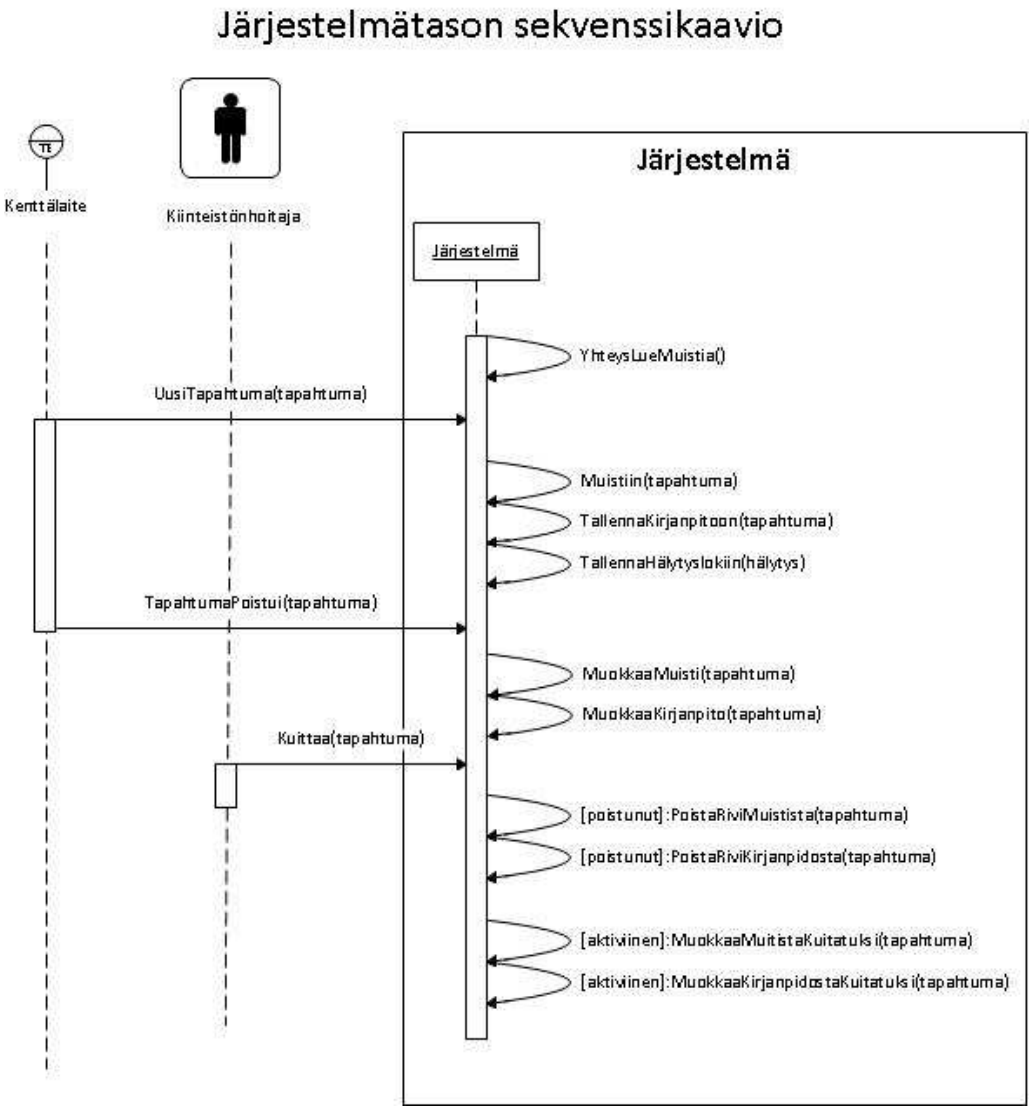
Esimerkistä kannattaa myös huomioida koodin lukua selkeyttäviä seikkoja. Koodista on eri asiat eroteltuna eri väreillä, loogiset kokonaisuudet on eroteltuna rivinvaihdoin omiksi lohkoikseen ja itse tapahtuma on sisennettynä muusta koodista.

3.2.3 Johtopäätökset

Ohjelman toteutus onnistuisi todennäköisesti kaikilla Qt:n tukemilla kielillä, mutta ohjelmointikieleksi valittiin C++. Se on tarpeeksi alkeellinen ohjelmointikieli, jotta pystytään varmasti täyttämään asiakkaan ohjelmalle asettamat vaatimukset, lisäksi se oli tutuin ohjelman toteuttajalle. Ohjelman ei myöskään tarvitse sisältää näyttäviä animaatioita, joita QML:llä pystyisi helpommin luomaan. [10]

3.3 Toteutettavan ohjelman järjestelmätason mallit

Vesiputousmallin toisen kohdan mukaisesti tulee ennen tarkempaan suunnitteluun siirtymistä luoda asiakkaalle hyväksyttävä korkean tason suunnitelma ohjelman rakenteesta. Näitä malleja jatketaan ja tarkennetaan siirryttäessä tarkempaan suunnitteluvaiheeseen. Luokkia voi tulla lisää ja niitä voi karsiutua. Malleja tarkennetaan ohjelman toiminnan kannalta oleellisilla toiminnoilla. Asiakkaan tarkan vaatimusmäärittelyn pohjalta pystyi luomaan kuvan 11. mukaisen hahmotelman järjestelmätason sekvenssikaaviosta, joka kuvaa viestin kulkua ohjelman sisällä kronologisessa järjestyksessä ylhäältä alaspäin. Tapahtuman kestoa kuvaa suorakulmio ja tapahtuman suuntaa nuoli, jonka vieressä on selitys toiminnasta. Kuvan alapuolella on selitettynä tekstillä ohjelman toimintatilanteet.



Kuva 11. Viestin kulkeutumisen hahmotelma järjestelmätason sekvenssikaavion avulla.

Ohjelman yleiset ominaisuudet:

- Käyttöliittymä tarjoaa mahdollisuuden, jolla ohjelma voidaan asettaa ponnahtamaan päällimmäiseksi uuden tapahtuman saapuessa.
- Käyttöliittymällä näkyy alakeskusyhteyden tila.
- Käyttöliittymän tulee tunnistaa käytettävä resoluutio ja skaalautua sen mukaisesti.

Ohjelman käynnistys:

- Ohjelma lukee käynnistyessä käyttäjän asettamat asetukset (IP-osoitteen, portin, VAK ja kohdetiedon), ini-muotoisesta tiedostosta.

- Ohjelma lukee omasta kirjanpidostaan käyttäjän kuittaamien hälytysten tiedot ja asettaa nämä kuitatut hälytykset näkyville, jos ne ovat voimassa.

- Ohjelma kuuntelee kokoajan käyttäjän asettamaa porttia (3060).

Tapahtuman saapuminen alakeskukselta:

- Uuden tapahtuman saapuessa tallennetaan viesti ohjelman muistiin.

- Tapahtuma muokataan haluttuun muotoon ja tallennetaan hälytyslokiin omassa muodossa, sekä ohjelman omaan kirjanpitoon omassa muodossa.

- Tapahtuma näkyy käyttöliittymässä mustalla värillä.

Tapahtuman poistuminen:

- Tapahtuman tiedon, esimerkiksi lämpötilan ylityksen poistuttua, sitä ei vielä poisteta ohjelmasta, vaan olemassa olevaa tietoa tapahtumasta muokataan lokeihin ja grafiikalla väri vaihdetaan vihreäksi, sekä merkataan poistuneen hetken ajankohta tapahtuman kohdalle.

Kuittaus:

- Tapahtuman ollessa aktiivinen kuittaus merkataan grafiikalle ja päivitetään omaan kirjanpitoon.

- Poistuneen hälytyksen kuittaus poistaa sen ohjelman kirjanpidosta sekä grafiikalta.

Virheensieto:

- Poistuneen tapahtumatiedon saapuessa ilman, että ohjelma löytää kirjanpidostaan vastaavaa tapahtunutta hälytystä, merkataan tapahtunut ja poistunut sarakkeiden tapahtuma-ajat samoiksi.

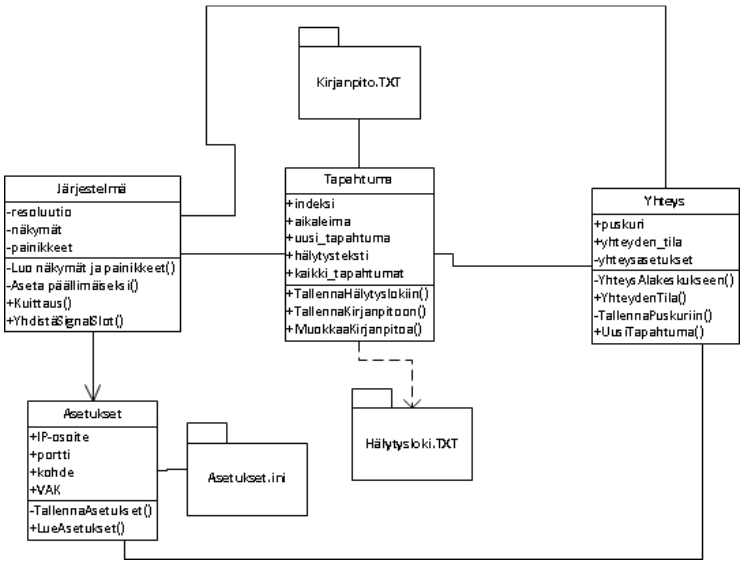
- Epäonnistuneen yhteysyrityksen jälkeen yrittää ohjelma yhdistää uudelleen 20 sekunnin välein.

Luokkakaaviolla kuvataan ohjelman rakenne, toiminnot ja luokkien väliset riippuvuudet. Laatikon ylin rivi kertoo luokan nimen, toinen rivi luokan attribuutit ja kolmas toiminnot, jotka luokka toteuttaa. Attribuuttien ja toimintojen edessä oleva plus(+)-, tai (-)miinusmerkki kertoo näkyvyyden muille luokille, eli voivatko muut luokat hyödyntää tätä toimintoa. Näkyvyyden rajoittamisella pakotetaan ohjelmoija ajattelemaan tarkemmin to-

teutettavan ohjelmakomponentin toimintaa ja rajoittamaan virheiden syntyä. Olio-ohjelmoinnin hyvien periaatteiden mukaisesti ohjelmien tulisi olla selkeitä ja mahdollisimman riippumattomia toisistaan. Tätä ideaa kutsutaan olioiden kapseloinniksi. Luokkien tulisi olla myös hajautettuna eri kerroksille uudelleenkäytettävyyden ja riippuvuuksien minimoinnin takaamiseksi. Kerrostetussa ohjelmassa ylempi kerros ei näy alemmille kerroksille. Yleensä kerrokset jaetaan kolmeen: käyttöliittymäkerros, sovelluslogiikka ja tietokantakerros.

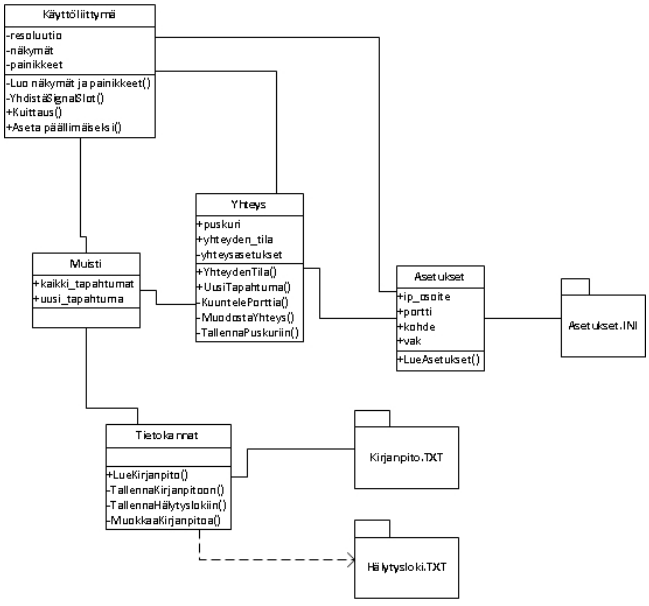
Määrittelyvaiheessa luotiin kaksi vaihtoehtoista luokkakaavioehdokasta toteutettavan ohjelman rakenteelle, joista toinen karsiutuisi pois suunnitteluvaiheessa. Ensin luotiin loogisesti päätelty kaavio järjestelmätason sekvenssikaavion perusteella, minkä jälkeen vertailuksi toinen kaavio vaatimusmäärittelykaaviosta kerättyjen substantiivien pohjalta. Yleisesti voidaan todeta, että syntyneissä luokkakaavioissa esiintyy samanlaisia luokkia ja ohjelma olisi todennäköisesti mahdollista toteuttaa järkevästi molempien mallien avulla.

Toinen käytetyistä tavoista perustuu ajatukseen, että yleensä luokat esiintyvät vaatimusmäärittelyssä substantiiveina. Aluksi kerätään kaikki substantiivit vaatimusmäärittelystä. Osa substantiiveista voidaan karsia heti pois, osa muodostuu loogisesti luokiksi ja osa luokkien attribuuteiksi. Alla on tämän periaatteen perusteella muodostettu määrittelyvaiheen luokkakaavio, johon on lisättynä ohjelmalle välttämättömiä attribuutteja ja toimintoja, joita oli ilmaantunut ensimmäistä kaaviota hahmoteltaessa.



Kuva 12. Määrittelyvaiheen luokkakaavio vaatimusmäärittelyn substantiivien pohjalta.

Toinen tapa muodostaa ohjelman rakenne oli looginen päättely järjestelmätason sekvenssikaavion pohjalta. Tällä tavoin saatiin paremmin jaettua ohjelma kerroksiin ja riippuvuuksia vähennettyä.



Kuva 13. Määrittelyvaiheen luokkakaavio järjestelmätason sekvenssikaavion pohjalta.

4 Ohjelman tarkempi suunnittelu

Tarkemman suunnittelun kohdalla ei nähty järkeväksi piirtää luokista laajennettuja kaavioita, koska ohjelman vaatimusmäärittelyn pohjalta pystyttiin tekemään melko tarkat suunnitelmat ohjelman rakenteesta.

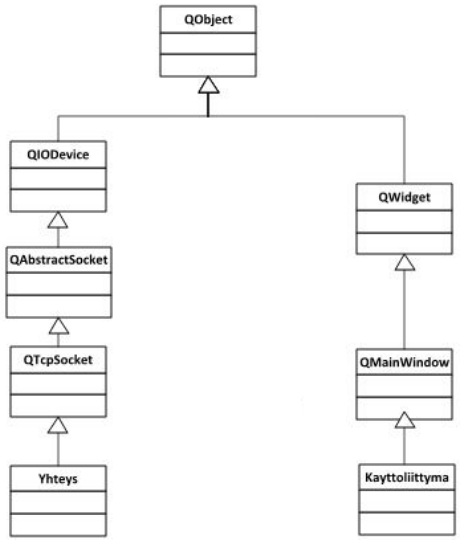
4.1 Toteutus Qt:lla

Kaikki luokat ohjelman perivät QObject-kantaluokan, jonka avulla saadaan käyttöön sen tarjoama connect-funktio. Qt:ssa pystyy luomaan tämän ominaisuuden avulla signal-slot-yhteyksiä, joilla yhdistetään syy-seuraus-suhteita toisiinsa. Signaalin tapahtuessa kutsutaan slottia alla olevan kuvan 14 mukaisesti. Valmiista ohjelmasta otetussa esimerkissä (kuva 14) näkyvän QTcpSocket luokasta muodostetun socket-olion valmiit signaalit yhdistetään omiin funktio-slotteihin. Esimerkissä socket ilmaisee signaalin stateChanged yhteyden tilan muuttuessa, jolloin ohjelmaa kutsuu YhteydenTilaMuuttunut()-slottia. Tätä rakennetta apuna käyttäen pystyvät funktiot välittämään signaaleiksi määriteltyjä funktioita emit-käskyllä ja ohjelmasta pystytään rakentamaan tapahtumapohjainen.

```
6   Yhteys::Yhteys(QObject *parent) :
7       QTcpSocket(parent)
8   {
9       /*Oliota luotaessa yhdistetään socket ja timer olioiden valmiit signaalit
10      omiin funktioihin. Lisäksi luonnissa ladataan asetukset ja kutsutaan yhdistä funktiota.*/
11
12      connect(&socket, SIGNAL(readyRead()),
13              this, SLOT(OdotaLukemista()));
14
15      connect(&socket, SIGNAL(stateChanged(QAbstractSocket::SocketState)),
16              this, SLOT(YhteydenTilaMuuttunut()));
17
18      connect(&socket, SIGNAL(error(QAbstractSocket::SocketError)),
19              this, SLOT(Yhteysvirhe(QAbstractSocket::SocketError)));
20
21      connect(&timer, SIGNAL(timeout()),
22              this, SLOT(TarkastaYhteys()));
23
24      LataaAsetukset();
25      Yhdista();
26  }
27
28  Yhteys::~Yhteys()
29  {
30      // Ei dynaamisia olioita
31  }
```

Kuva 14. Yhteysluokan muodostin, jossa luodaan signaalit ja slotit

Qt:n kirjastot tarjoavat myös valmiiksi yläluokista periytettyjä, tiettyyn tarkoitukseen räätälöityjä aliluokkia, jotka perivät aina ylemmän luokan ominaisuudet. Näitä ovat esimerkiksi TCP-yhteyksiin tarkoitettu QTcpSocket-luokka, sekä käyttöliittymää varten räätälöity QMainWindow-luokka, joita tullaan toteutettavassa ohjelmassa hyödyntämään (kuva 15.).



Kuva 15. Yhteys- ja käyttöliittymäluokan perintähierarkia

4.2 Käyttöliittymä

Qt tarjoaa graafisen työkalun ohjelman käyttöliittymän suunnitteluun. Hiiren avulla voidaan raahata drag&drop-periaattella erilaisia käyttöliittymän rakennuspalikoita, kuten painonappeja ja listaobjekteja, joiden ominaisuuksia pystytään muokkaamaan erilaisten muokkausta helpottavien valikoiden kautta. Tästä näkymästä muodostetaan xml-muotoinen tekstipohjainen rakenne, joka käännösvaiheessa yhdistetään ohjelmakoodin kanssa. Myös palikoiden toiminnallisten yhteyksien, kuten napin painalluksen yhdistämistä haluttuun tapahtumaan, on helpotettu tämän valikon kautta, joka vähentää kirjoitettavan koodin määrä. Ohjelman käyttöliittymä on mahdollista luoda myös kokonaan kirjoittamalla.

Toteuttavan ohjelman kohdalla graafista ominaisuutta ei tulla suuresti tarvitsemaan, koska kaikki ominaisuudet pyörivät yhden pääikkunan ympärillä, joka näyttää kaiken tarvittavan. Objektien sijoitus ikkunassa haluttuun kohtaan on kuitenkin tätä ominaisuutta apuna käyttäen helpompaa.

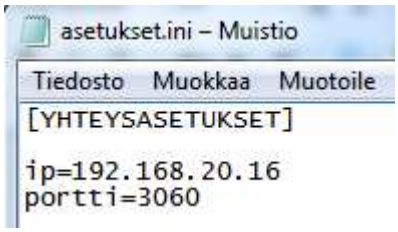
5 Toteutus

Toteutusvaiheessa dokumentoinnin määrä pidettiin mahdollisimman pienenä ja keskityttiin ohjelman kirjoittamiseen. Ohjelma rakennettiin lohko kerrallaan, jotta liian suuresta hahmotettavasta kokonaisuudesta syntyvät sekaannukset ja ohjelmointivirheet minimoitaisiin. Lohkon riittävän toimintavarmuuden ja vaaditunlaisen toteutuksen täytyttyä siirryttiin rakentamaan seuraavaa lohkoa, joka yhdistettiin vanhaan kokonaisuuteen. Seuraavaksi kerrottavien luokkien järjestys noudattaa ohjelman loogista toimintajärjestystä.

5.1 Asetukset

Kuva 16. Asetusluokka

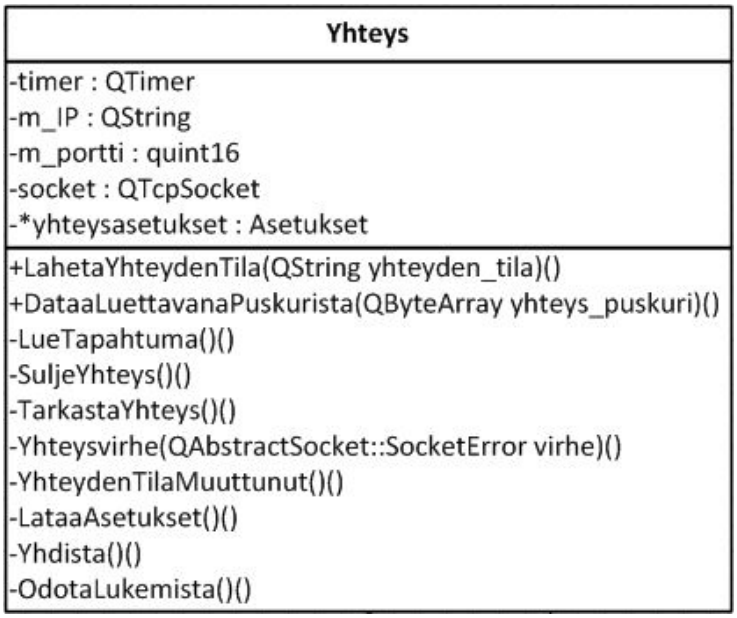
Vaatimusmäärittelyssä toivottiin asetusten, eli IP:n ja portin, lukua esimerkiksi ini-tiedostosta (kuva 17.). Tähän tiedostoon syöttäisi yrityksen edustaja rakennusvaiheessa oikeat asetukset, jonka jälkeen käyttäjän ei niihin tarvitsisi koskea.



Kuva 17. Yhteysasetustiedoston sisältö

Asetusten luku haluttiin eriyttää muista luokisti ohjelman selkeyttämisen kannalta vaikka luokkana se on pieni. Tämä siksi, jotta testaus olisi helpompaa ja ohjelma olisi loogisemmin jaoteltu. Asetusluokka sisältää vain tiedot palauttavan funktion PalautaYhteysasetukset(), joka lukee tiedot ini-tiedostosta ja palauttaa nämä listan muodossa. Virhetilanteen tapahtuessa palautetaan lista, joka sisältää arvot ”0” ”0”.

5.2 Yhteys alakeskukseen



Kuva 18. Yhteyden luokkakaavio

Yhteysluokka pitää huolen yhteyden kunnosta ja viestien lukemisesta alakeskuksesta. Muodostimessa yhdistetään staattisena luotavan QTcpSocket-luokan olion socket signaaleihin omat slotit, sekä ladataan yhteysasetukset tiedostosta Asetukset-luokan olion avulla. Esimerkki yhteysluokan muodostimesta esiintyy ylempänä kuvassa 14.

Olion luonnin yhteydessä kutsutaan myös Yhdista()-funktiota, joka yhdistää alakeskukseen ja käynnistää yhteyden tilaa tarkkailevan ajastimen. Ajastin kutsuu funktiota TarkastaYhteys(), joka tarkastaa sekunnin välein kuuluuko valittu verkko tietokoneen yhdistettyinä oleviin verkkoihin. Jos näin ei ole, ohjelmalle kerrotaan, että yhteys on poikki ja sen tulee ensin katkaista yhteys. Tämän jälkeen yritetään yhdistää uudelleen. Tämä ratkaisu tarvittiin, koska socket-olio ei tarjoa automaattisesti yhteyden tilaa tarkkailevaa funktiota, vaan sille pitää kertoa, koska yhteys on todella poikki.

Viestien lukeminen alakeskuksesta tapahtuu LueTapahtuma()-funktion avulla (kuva 19). Muodostimessa yhdistetään socket-olion signaali readyRead(), joka ilmaisee kun dataa on luettavana, slottiin OdotaLukemista(), jossa odotetaan kaksi sekuntia ennen kuin siirrytään lukemaan tapahtumia LueTapahtuma()-funktion avulla. Odottamista tarvitaan erityisesti onnistuneessa yhdistystilanteessa, jossa alakeskus lähettää kaikki

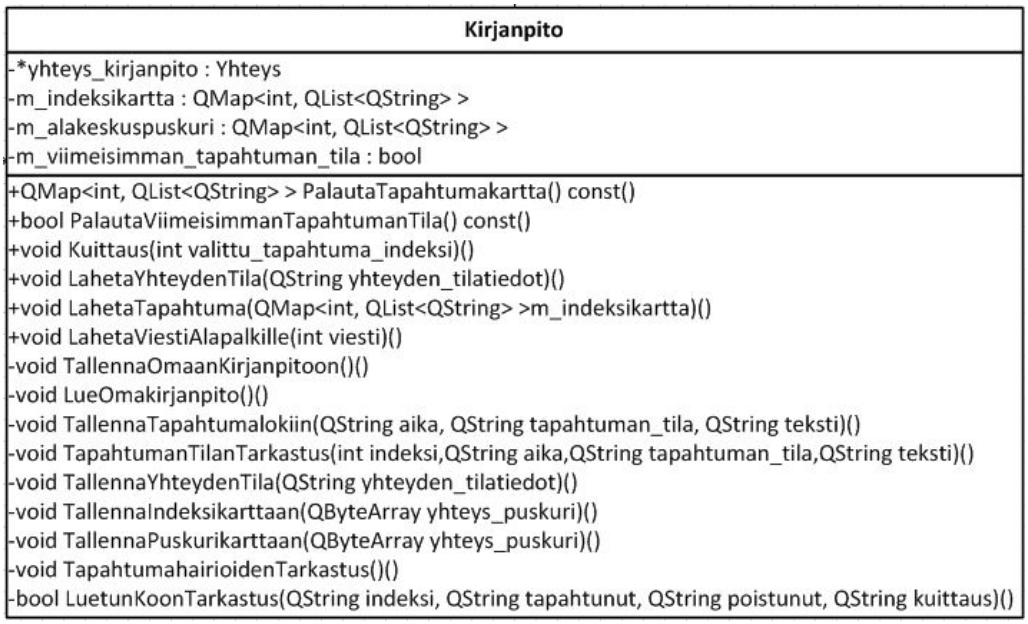
voimassa olevat tapahtumat luettavaksi, jotta kaikki tapahtumat luetaan yhdellä kerralla. Itse lukeminen tapahtuu tallettamalla kaikki luettu QByteArray-taulukkoon ja lähettämällä tapahtumat eteenpäin.

```
void Yhteys::OdotaLukemista()
{
    /*Yhdistettäessä ohjelma ilmaisee ensin yhden tapahtuneen,jonka jälkeen loput tapahtuneet.
    Odottamalla hetken kaikki tapahtumat ehtivät saapua luettaviksi.*/

    QTimer::singleShot(2000,this,SLOT(LueTapahtuma()));
}


void Yhteys::LueTapahtuma()
{
    /*Havaittaessa liikennettä tallennetaan kaikki data muuttujaan
    ja lähetetään eteenpäin.*/

    QByteArray yhteys_puskuri;

    while(!socket.atEnd())
    {
        yhteys_puskuri = socket.readAll();
    }

    emit DataaLuettavanaPuskurista(yhteys_puskuri);
}
```

Kuva 19. Tapahtumien lukeminen

Viesti tilan muutoksesta välitetään seuraavalle luokalle samaan tapaan muodostimessa yhdistetyn socket-olioon yhdistetyn signal-slot-rakenteen avulla. Yhteyden tilan muuttuessa lähetetään QString-tyyppisenä tekstirakenteena yhteyden tilatiedot eteenpäin.

5.3 Tapahtumatietojen ylläpito



Kuva 20. Kirjanpidon luokkakaavio

Tietoa tapahtumista pitää yllä Kirjanpito-luokka. Sen periaatteena on ylläpitää kahta QMap<int, QList<QString>-muotoista karttaa. Toiseen talletetaan kuitatut ja poistuneet hälytykset ja toiseen pelkät voimassa olevat tapahtuneet, eli jälkimmäinen kartta peilaa alakeskuksen muistia (kuva 21). Karttaan talletetaan int-tyyppiseen muuttujaan tapahtuman indeksi ja QList-listamuuttujaan tiedot tapahtumasta. Tapahtuman saa siis valittua kartasta sen avaimen, eli indeksin perusteella. Näitä kahta karttaa vertailemalla erilaisten ehtolauseiden avulla pystytään omakirjanpidosta poistamaan haamutapahtumat, jotka muuten jäisivät ilmaisemaan väärää tilaa tapahtuman tilan muuttuessa ohjelman ollessa kiinni tai yhteyden ollessa poikki. Vertailemalla pystytään tarkastamaan myös, että lokitiedostoon tallennettavia tapahtumia ei kirjoiteta aina uudelleenyhdistettäessä uudestaan, koska yhdistettäessä uudelleen alakeskus ilmaisee ulos kaikki voimassa olevat tapahtumat.

```
void Kirjanpito::TallennaPuskurikarttaan(QByteArray yhteys_puskuri)
{
    QByteArray muunto = yhteys_puskuri;
    muunto.replace('\0', ";");
    QTextCodec *codec = QTextCodec::codecForName("IBM 850");
    QString s = codec->toUnicode(muunto);

    QList<QString> lista = s.split(";");
    lista.removeAll("");

    QListIterator<QString> iter(lista);
    while (iter.hasNext())
    {
        int indeksi = iter.next().toInt();
        QString aika = iter.next();
        QString tapahtuman_tila = iter.next();
        QString teksti = iter.next();

        // Tapahtuma poistui
        if(m_alakeskuspuskuri.contains(indeksi) == true && tapahtuman_tila == "0")
        {
            m_alakeskuspuskuri.remove(indeksi);

            // Poistutaan, jotta alempia ehtolauseita ei suoriteta
            return;
        }

        // Ei sisällä tapahtumaa (tapahtuma uusi)
        if(m_alakeskuspuskuri.contains(indeksi) == false)
        {
            QString tapahtunut = aika;
            QString poistunut = "";
            QString kuittaus = "";
            QList<QString> uusi_tapahtuma;
            uusi_tapahtuma << tapahtunut << poistunut << kuittaus << teksti;
            m_alakeskuspuskuri.insert(indeksi,uusi_tapahtuma);
        }
    }
}
```

Kuva 21. Tallennus alakeskuksen muistia peilaavaan karttaan m_alakeskuspuskuri

Saapuvista viesteistä tarkastetaan, että viestin viimeinen merkkiyhdistelmä on valmistajan valitsema \0, ennen kuin ne lisätään karttoihin. Vääränlaisen viestin saapuessa ohjataan virheellinen data tekstitiedostoon ja ilmaistaan käyttäjälle virheilmoitus ponnahdusikkunassa.

Kirjanpito tallentaa omaa kirjanpitoaan txt-muotoiseen tekstitiedostoon, jotta uudelleenkäynnistettäessä ollaan samassa tilanteessa kuin suljettaessa. Tapahtumalokin kirjoitus tapahtuu käyttäen lähes samanlaista yksinkertaista tekstitiedostoon kirjoittavaa rakennetta (kuva 22). Lisäksi kirjoitettiin samanlainen funktiorakenne yhteyden tilan viesteille. Tästä rakenteesta voisi olla hyötyä esimerkiksi, rakennusautomaatioyrityksen vikaa korjaamaan tulleelle henkilölle, joka näkisi milloin yhteys alakeskukseen on ollut poikki.

```
void Kirjanpito::TallennaTapahtumalokiin(QString aika, QString tapahtuman_tila, QString teksti)
{
    //Tallennetaan tapahtumaloki omaan kansioon, vuoden ja kuukauden mukaisiin tekstitiedostoihin.

    QDir kansio(QCoreApplication::applicationDirPath() + "/Tapahtumaloki/");
    if(!kansio.exists())
    {
        kansio.mkpath(".");
    }

    QString tapahtumalokin_tiedoston_polku = QCoreApplication::applicationDirPath() + "/Tapahtumaloki/"
            + QDate::currentDate().toString("MM.yyyy") + ".txt";
    QFile f(tapahtumalokin_tiedoston_polku);

    if(f.open(QIODevice::WriteOnly | QIODevice::Append | QIODevice::Text))
    {
        //Selkeyden vuoksi 0 ja 1 tekstimuotoon
        if(tapahtuman_tila == "0")
        {
            tapahtuman_tila = "poistui";
        }

        if(tapahtuman_tila == "1")
        {
            tapahtuman_tila = "tapahtui";
        }

        QTextStream ulos_syotto(&f);
        ulos_syotto  << aika << " " << tapahtuman_tila << " " << teksti<< "\n";
    }
    f.close();
}
```

Kuva 22. Tapahtumalokin tallennus

Omakirjanpidon luku tiedostosta tapahtuu kuten tiedostoon kirjoittaminen, mutta virran suunta on erilainen. Selattaessa omakirjanpidon tekstitiedostoa voi sinne harhapainalluksista johtuen eksyä ohjelman rakennetta sekoittavia merkkejä. Näiden kulkeutuminen käytettäväksi asti on estetty aikarakenteiden kokoa, sekä aikarakenteiden ja indeksin muotoa tarkkailevien validaattorien avulla.

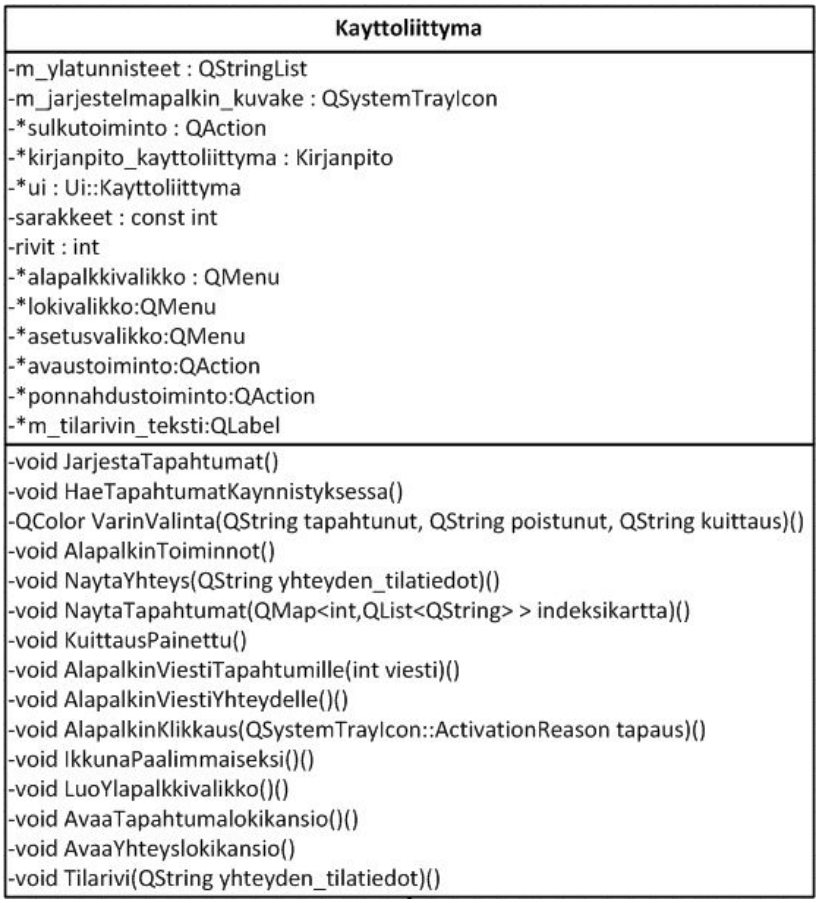
Tapahtumien kuittaus ominaisuus kirjoitettiin kirjanpitoluokalle. Funktio saa arvokseen käyttäjän käyttöliittymästä valitseman tapahtuman indeksin arvon, jonka perusteella tapahtuma valitaan kartasta käsiteltäväksi. Tapahtuman ollessa poistunut ja kuitattu poistetaan se omakirjanpidosta ja ohjelman muistista. Voimassa olevalle tapahtumalle merkataan kuittauskenttään kuittaushetken ajankohdan arvo.

Tapahtumakirjanpito lähettää seuraavalle luokalle, eli käyttöliittymälle, kolmea signaalia tapahtuman mukaan. Näitä tietoja ovat yhteyden tilan muutos tallennuksen jälkeen, poistuneet ja kuitatut tapahtumat sisältävän kartan omakirjanpitoon tallennuksen jälkeen, sekä saapuneen viestin tyypin ja tiedot Windowsin järjestelmäpalkille näytettäväksi (kuva 23).

```
signals:
    void LahetaYhteydenTila(QString yhteyden_tilatiedot);
    void LahetaTapahtuma(QMap< int, QList<QString> > m_indeksikartta);
    void LahetaViestiAlapalkille(int viesti, QString tapahtuman_teksti);
```

Kuva 23. Kirjanpidon lähettämät signaalit

5.4 Käyttöliittymä



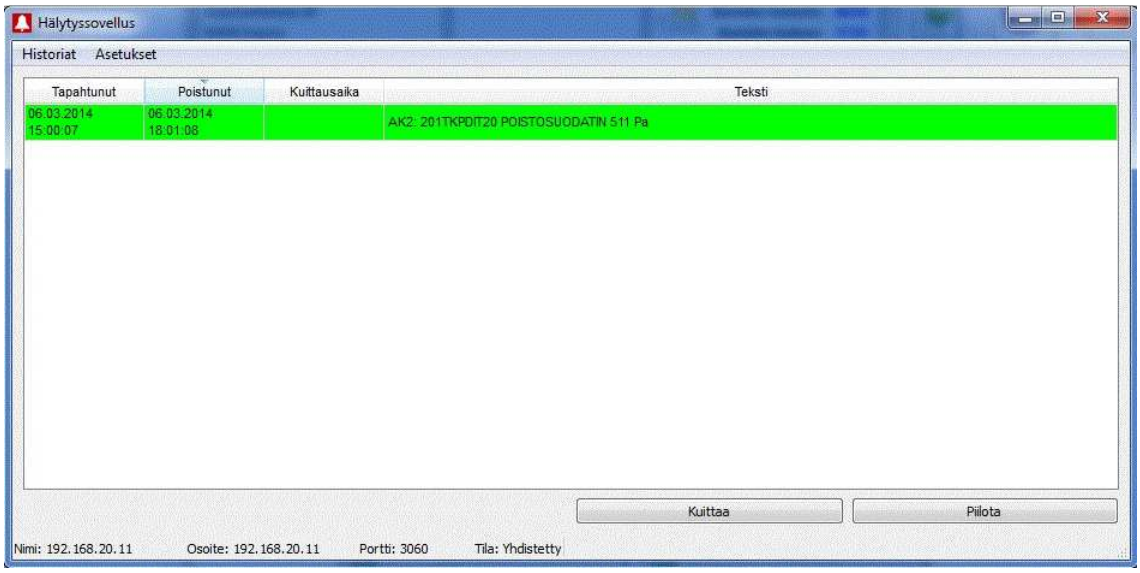
Kuva 24. Käyttöliittymän luokkakaavio

Tässä ohjelmassa graafista työkalua ei tarvittu kuin päänäkymän ulkoasun hahmotelmaan, koska ohjelmassa on vain yksi graafinen ikkuna ja loput ominaisuudet oli selkeämpää luoda koodin avulla. Luotaessa oliota käyttöliittymäluokasta ajetaan läpi myös graafisen käyttöliittymän xml-koodi (kuva 25). Muodostimessa luodaan myös muut näkyvät rakenteet, kuten ohjelman kuva näkymään järjestelmäpalkkiin, toimintojen painikkeet ja signal-slot syy-seuraus-suhteet.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Kayttoliittyma</class>
 <widget class="QMainWindow" name="Kayttoliittyma">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>931</width>
    <height>431</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Hälytyssovellus</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout">
    <item row="1" column="1">
     <layout class="QHBoxLayout" name="horizontalLayout">
      <item>
       <widget class="QPushButton" name="kuittauspainike">
        <property name="text">
         <string>Kuittaa</string>
        </property>
       </widget>
      </item>
```

Kuva 25. Otos ohjelman xml-koodista

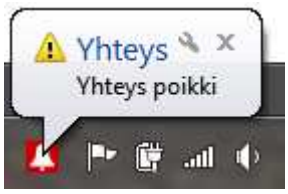
Hyväksytyn lisäämisen jälkeen kirjanpito-oliolta tapahtumakartta lähetetään käyttöliittymälle näytettäväksi. Käyttöliittymän funktio NaytaTapahtumat(QMap<int, QList<QString> > m_indeksikartta) purkaa kartan sisältämät tapahtumat osiin ja piirtää ne ruudukon muotoiseen rakenteeseen siten, että yksi rivi muodostuu järjestyksessä indeksi, tapahtunut, poistunut, kuittaus ja tapahtuman teksti, jonka jälkeen siirrytään piirtämään seuraava rivi. Käyttäjältä on piilotettuna indeksisarake (kuva 26). Tehtävien sekoittuminen ruudukkoon piirron yhteydessä estetään poistamalla mahdollisuus järjestellä tapahtumia, järjestämällä ne oletusjärjestykseen ja palauttamalla ominaisuus takaisin päälle funktion suorituksen lopussa. Rivin värin valinta tehdään vertailemalla tapahtuman aikatietoja ruudukkoon piirron yhteydessä. Esimerkiksi poistuneen ajankohdan ollessa tapahtunut myöhemmin kuin tapahtunut ajankohta, merkataan rivi vihreällä poistuneen merkiksi.



Kuva 26. Ohjelman käyttöliittymä

Tapahtuman kuittaus tehdään kaksoisklikkaamalla tapahtumaa tai erillisen painikkeen avulla. Käyttäjän valitessa avautuvasta ponnahdusikkunasta ”kuittaa tapahtuma”-valinta kutsutaan kirjanpitoluokan kuittaus-funktiota. Kuittauksen jälkeen tehdään samat toimenpiteet, kuin uuden viestin saapuessa, eli piirretään taulukko uudestaan.

Tapahtuman saapuessa, yhteyden tilan muutoksesta tai kuittauksesta näytetään käyttäjälle järjestelmän alapalkissa ilmoitus tapahtumasta (kuva 27). Esimerkiksi uuden tapahtuman saapuessa ikkunan otsikko on ”Uusi tapahtuma” ja viestinä näytetään tapahtuman tiedot. Poistuneessa vastaavasti otsikkona poistunut.



Kuva 27. Alapalkin viesti

5.5 Pääohjelma

```
1  #include "kayttoliittyma.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7
8      Kayttoliittyma w;
9      w.show();
10
11     return a.exec();
12 }
```

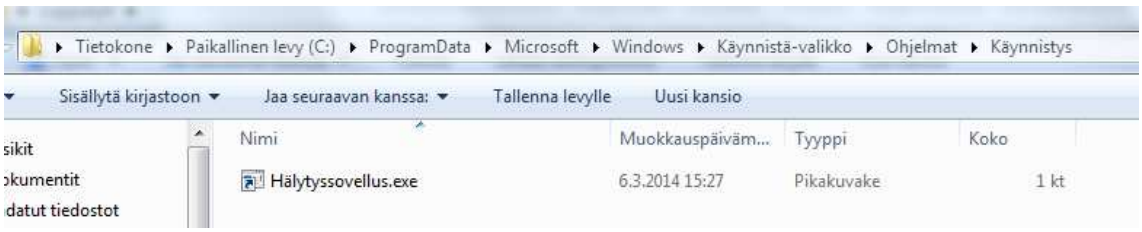
Kuva 28. Pääohjelman rakenne

Ohjelman rakenteessa kaikki muut luokat on muodostettu käyttöliittymäluokasta luotavan olion alle, kuvan 28 mukaisesti. Tällöin suljettaessa käyttöliittymä vapautetaan kaikki ohjelman muisti, jolla pyritään estämään mahdolliset muistivuodot. Käyttöliittymäolio luodaan pääohjelman main-funktiossa.

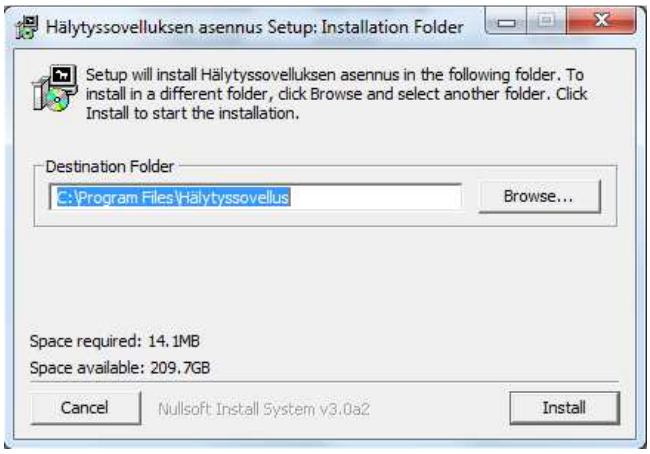
6 Integrointitestaus

Koodia pyrittiin kääntämään mahdollisimman usein toteutusvaiheessa, jotta erinäköiset ongelmat tulisi heti huomattua ja virheitä ei pääsisi syntymään. Testausta tehtiin myös yhdisteltäessä luokkia toisiinsa, jolloin koodista tarkasteltiin hitaasti ajatuksen kanssa funktioiden rakenne ja tapahtumien halutunlainen kulku funktiolta toiselle, jolla pyrittiin löytämään virheitä. Lisäksi koodausvaiheessa ohjelmalla testattiin erilaisia ongelmallisia perustilanteita, kuten viestin tapahtuminen yhteydettömässä tilassa, virheellisten merkkien syöttämistä omaan kirjanpitoon ja verkkokaapelin irrottamista, joten ohjelman tiedettiin selviytyvän muutamista perusongelmista.

Ohjelman asennusta varten luotiin asennuspaketti yksinkertaisella vapaaseen lähdekoodiin perustuvalla Nullsoft Scriptable Install System(NSIS)-ohjelmalla. Asennuspaketti kysyy käyttäjältä minne ohjelma halutaan asentaa. Hakemistoon asennetaan itse ohjelman, oletusasetuksinen ini-muotoinen yhteystiedosto ja tyhjä omakirjanpitotiedosto. Ohjelma luo ensimmäisen käynnistyksen yhteydessä kansiorakenteet, joihin tallennetaan tapahtuma-, yhteys- ja virhelokia. Luomalla pikakuvake ohjelmatiedostosta ja asentamalla se Windowsin tarjoamaan ”Käynnistys” kansioon saadaan ohjelma aukeamaan tietokoneen uudelleenkäynnistyksen yhteydessä, esimerkiksi sähkökatkon jälkeen (kuva 29).



Kuva 29. Käynnistyksessä avautumisen polku



Kuva 30. Asennuspaketti

Ohjelman käyttöliittymä on niin selkeä, että sen testausta fyysisesti kiinteistönhoitajalla ei nähty järkeväksi. Integrointitestauksessa ohjelma asennettiin etäyhteyden kautta kerrostalossa sijaitsevaan taulutietokoneeseen. Ohjelma asentui ja käynnistyi niinkuin sen kuuluikin, joten se jätettiin toimimaan ja tilaa tarkkailtiin kerran päivässä kahden viikon ajan. Heti kuitenkin huomattiin, että Windows 7-alustaisella kannettavalla tietokoneella ohjelma kuluttaa muistia jopa 10000 kilotavua, kun Windows 7-pohjaisella taulutietokoneella muistin kulutus putosi noin 4000 kilotavuun.

Testijakson aikana huomattiin kaksi ongelmaa. Ensimmäisenä muutettiin tapahtuneet aktiiviset viestit näyttämään aina vain tapahtuneen hetken aikaa, kun ennen näkyi uudestaan tapahtuneilla viesteillä myös poistuneen hetken aika. Toisena huomattiin yhteyslokin perusteella, että yhteys sulkeutuu ja yhdistetään heti uudelleen 24 tunnin syklissä onnistuneen yhdistämisen jälkeen, vaikka väylässä on esiintynyt liikennettä. Kirjoittamalla testikoodi huomattiin, että juuri ennen yhteyden katkeamista socket ilmaisee virheen ”The remote host closed the connection”. Tämä virhesanoma viittaa siihen, että alakeskus katkaisisi yhteyden, eikä vika olisi toteutetussa sovelluksessa. Ongelmaa yritettiin ratkaista muun muassa löytämällä alakeskuksen asetuksista tämän toiminnon estävää toimintoa, mutta sellaista ei löytynyt. Ratkaisuksi kehitettiin ajastin, joka ilmoittaa käyttäjälle yhteyden muutokset, vasta kun yhteys on ollut poikki yli kolme sekuntia. Tällä tavoin lokia ei kirjoiteta tai ilmoituksia näytetä, lyhyiden katkosten jälkeen.

7 Yhteenveto

7.1 Toteutuksen vertaus suunniteltuun

Ohjelman lopullisesta rakenteesta tuli lähes vaatimusmäärittelyn sekvenssikaavion pohjalta rakennetun luokkakaavion mukainen. Muutoksena suunnitelmissa esiintyneen muistiluokan putoaminen pois. Luokan jättäminen tai muiden luokkien lisääminen olisi voinut olla järkevää ohjelman selkeyden kannalta, joidenkin luokkien melko suuren koon vuoksi. Ohjelman toimintasuunta on kuitenkin suoraviivainen, joten ymmärryksen kannalta tästä ei välttämättä olisi hyötyä. Viestien kulkeutuminen ohjelmassa noudattaa kuitenkin hyvin alussa suunniteltua sekvenssikaaviota viestin kulusta ohjelman sisällä.

Olio-ohjelmoinnin käytön ehkä tärkein syy oli koodin uudelleenkäytettävyys. Siihen, että itse kirjoitettuja funktioita voisi jatkossa sellaisenaan käyttää toisissa ohjelmissa, ei juurikaan päästy. Toteutetun ohjelman rakenteessa tämä tarkoittaisi pitkien funktioiden jakamista pienempiin abstrakteihimpiin osiin. Qt:n kirjastojen valmiita funktioita pyrittiin kuitenkin käyttämään mahdollisimman paljon, mikä lisää ohjelman selkeyttä ja uudelleenkäytettävyyttä.

Ohjelma on käytännössä yksi exe-muotoinen tiedosto. Valmiin ohjelmatiedoston koko on reilu 14000 kt, joka on suuri suhteutettuna ohjelman yksinkertaiseen rakenteeseen ja komponentteihin. Ohjelman kokoa yritettiin pienentää poistamalla riippuvuuksia ohjelmakirjastoihin, mutta tämä ei auttanut. Kokoa verrattiin muihin Qt:lla tehtyihin esimerkkiohjelmiin ja niiden koot olivat myös samaa kokoluokkaa. Ohjelman koko ei kuitenkaan ole suuri ongelma muistin ollessa nykyisin halpaa.

7.2 Asiakkaan vaatimuksien täyttyminen

Asiakas halusi ohjelmalta seuraavia ominaisuuksia:

- Ohjelman tulee toimia Windows 7/8 (32/64bit)-käyttöjärjestelmissä.
- Ohjelman tulee toimia itsenäisesti ilman erikseen asennettavia alustasovelluksia.
- Uuden tapahtuman saapuessa voidaan ohjelma asettaa käyttöliittymässä päällimmäiseksi sovellukseksi.

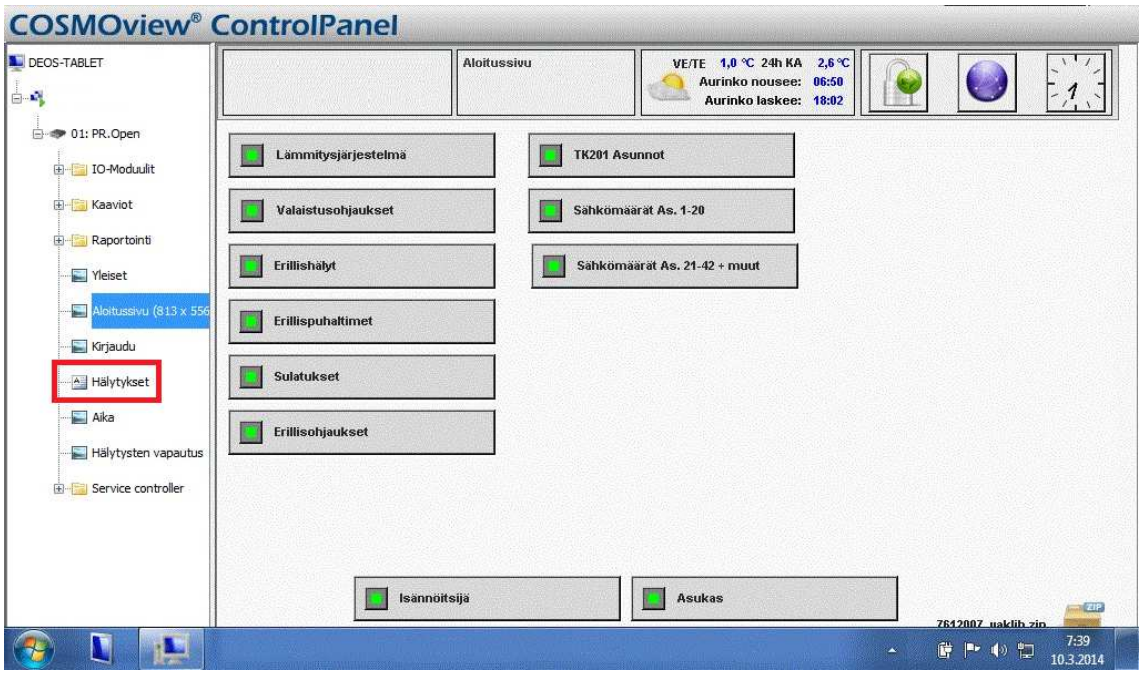
- Kenttien järjestystä voidaan muuttaa esimerkiksi uusin tapahtuma ylimpänä. Oletuksena uusin ylimpänä.
- Ohjelma osaa näyttää aktiivisen- ja poistuneen hälytyksen eri väreillä.
- Ohjelma osaa kirjoittaa hälytyslokia.
- Tapahtumien kuittaus.
- Käyttöliittymässä näytetään voimassa olevat ja kuittaamattomat tapahtumat.
- Käyttöliittymässä näytetään tilarivillä yhteyden tila alakeskukseen. Yhteydettömässä tilassa valikon väri on punainen.
- Poistuneen hetken aika on näkyvillä.
- Ohjelman tulee kirjoittaa omaa kirjanpitoa, jotta uudelleenkäynnistyksessä palataan vanhaan tilaan.
- Yhteysasetukset (alakeskuksen IP-osoite, portti, Kohde-teksti, VAK-teksti) voidaan asettaa ini-tiedostosta.
- Ohjelman tulee skaalautua automaattisesti käytettävän järjestelmän mukaan.
- Uudelleenyhdistys 20 sekunnin välein.
- Jos alakeskus ilmaisee poistuneen hälytyksen, mutta hälytystiedoissa ei löydy tapahtunutta hälytystä, ohjelma näyttää uuden rivin, jossa on merkitty sekä tapahtunut- että poistunut-aikaleima samoiksi. (Tällöin ei tiedetä, milloin Tapahtunut-hälytys on tullut).

Ohjelma täytti asiakkaan sille asettamat vaatimukset. Asetuksissa aseteltava kohde- ja VAK-tekstit jätettiin pois toteutettavasta ohjelmasta asiakkaan toiveen mukaisesti. Ohjelmaikkunan skaalausta ei lähdetty muuttamaan käytettävän käyttöjärjestelmän mukaan, koska käytetty koko sopi oletusasetuksilla myös taulutietokoneelle.

Ohjelman tuomiseksi päällimmäiseksi tulee ohjelma sulkea tai painaa ohjelman piilotapainiketta, jolla on sulkemisen kanssa ominaisuus, eli ohjelma piilotetaan järjestelmäpalkkiin. Ikkunan ollessa taustalla ikkuna vain aktivoituu, mutta ei ponnahda päällimmäiseksi ja ohjelman kuvake alkaa vilkkua järjestelmäpalkissa. Tämä johtuu ilmeisesti Windowssin ominaisuudesta, joka estää ilman rekisteriin tehtäviä muutoksia aukinaisen ohjelmaprosessin asettamisen päällimmäiseksi ikkunaksi. Ikkuna olisi voitu myös asettaa näkymään aina päällimmäisenä, jolloin ikkuna ei voisi vahingossa jäädä muiden ikkunoiden alle.

7.3 Kehitysideat

Vanha hälytysnäkymä on edelleen käyttäjän valittavana valvontaohjelman puunäkymässä. Tämän valinnan kautta olisi hyvä saada näkymään toteutettu ohjelma tai valintapainike olisi hyvä edes piilottaa käyttäjältä. Toinen vaihtoehto nähdä hälytykset vanhalla tavalla on ottaa yhteys selaimella alakeskukseen, jolloin nykyisen ohjelman voisi liittää jolloin lailla osaksi selainta.



Kuva 31. Painike, josta pääsee käsiksi vanhaan hälytysnäkymään

Tapahtuneet tapahtumat näkyvät käyttäjälle vain tekstin muodossa. Tämä ei ole kovinkaan visuaalisesti havainnollistava tapa tarkasteltaessa esimerkiksi lokitiedostoja. Havainnollistavampi tapa esittää tapahtumaloki olisi piirtää siitä jonkinlainen diagrammi, josta näkisi esimerkiksi, mitkä tapahtumat ovat tapahtuneet ja poistuneet useaan otteeseen.

Nykyinen tekstitiedostoon lokeja kirjoittava rakenne on melko riskialtis työskenneltäessä suurten tapahtumamäärien kanssa. Modernimpi ja varmempi tapa olisi tallettaa tapahtumat tietokantaan.

Ohjelman omakirjanpidon txt-muotoiseen tekstitiedostoon pääsee helposti käsiksi. Tämän tiedostomuodon tilalle olisi hyvä vaihtaa salatumpi tiedostomuoto, jonka rakennetta ei voisi käyttäjä muuttaa.

Säikeistyksellä tarkoitetaan ohjelman eri prosessien jakamista omiksi osikseen. Windowsissa eri säikeet näkyvät tehtävienhallinnassa saman ohjelman useampana ilmentymänä. Esimerkiksi tämän ohjelman osalla olisi ollut hyvä asettaa yhteyden tilaa tarkkaileva funktio omaksi osakseen, jolloin olisi päästään myös nopeampiin vastausaikoihin.

Lähteet

1 C-kielestä C++-kieleen (www-materiaalia). <https://noppa.aalto.fi/noppa/kurssi/as-0.3301/luennot/AS-0_3301_as-0.3301-lecture1.pdf.pdf> Luettu 21.10.2013

2 Epäonnistumisien syyt (www-materiaalia). <http://www.ibmsystemsmag.com/power/Systems-Management/Workload-Management/project_pitfalls/?page=3> Luettu 2.11.2013

3 Epäonnistuminen ohjelmistoprojekteissa (www-materiaalia> <http://www.tietoviikko.fi/kaikki_uutiset/article909109.ece> Luettu 2.11.2013

4 Hello world ohjelma (www-materiaalia) <http://en.wikipedia.org/wiki/Hello_world_program> Luettu 29.10.2013

5 Johdatus olio-ohjelmointiin (www-materiaalia). Tekijä Jussi Pohjolainen, Senior Lecturer at Tampere University of Applied Sciences. <http://www.slideshare.net/pohjus/johdatus-olioohjelmointiin-presentation> Luettu 19.10.2013

6 Johdatus olio-pohjaiseen ajatteluun (www-materiaalia). <http://www.slideshare.net/PassoK/johdatus-oliopohjaiseen-ajatteluun> Luettu 19.10.2013

7 Muokattu vesiputousmalli (www-materiaalia) <http://www.buzzle.com/images/diagrams/waterfall-model-diagram.jpg > Luettu 1.11.2013

8 Olioiden ohjelmointi C++:lla, Matti Rintala ja Jyke Jokinen. (vuosi. 2000, 3. painos)

9 Pygame lopputyö <http://publications.theseus.fi/bitstream/handle/10024/21239/kuisma_joentakanen.pdf?sequence=1>

10 Skrolli QML (www-materiaalia) <http://www.skrolli.fi/qml-1> Luettu 2.11.2013

11 TIOBE (www-materiaalia) <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> Luettu 28.10.2013

12 TTY Hypermedia vesiputousmalli (www-materiaalia) <http://hlab.ee.tut.fi/hmopetus/vpsist-oppimateriaali/4-menetelmia-ja-malleja/4-3-suunnittelumalleja/4-3-1-ohjelmistotuotannon-malli> Luettu 2.11.2013

13 Vesiputousmalli (www-materiaalia) <http://www.cs.helsinki.fi/u/taina/opol/k-2009/pdf/luku-6_2.pdf> Luettu 3.11.2013

Liite 1

1. Insinöörityö Jaakko Kilpeläinen

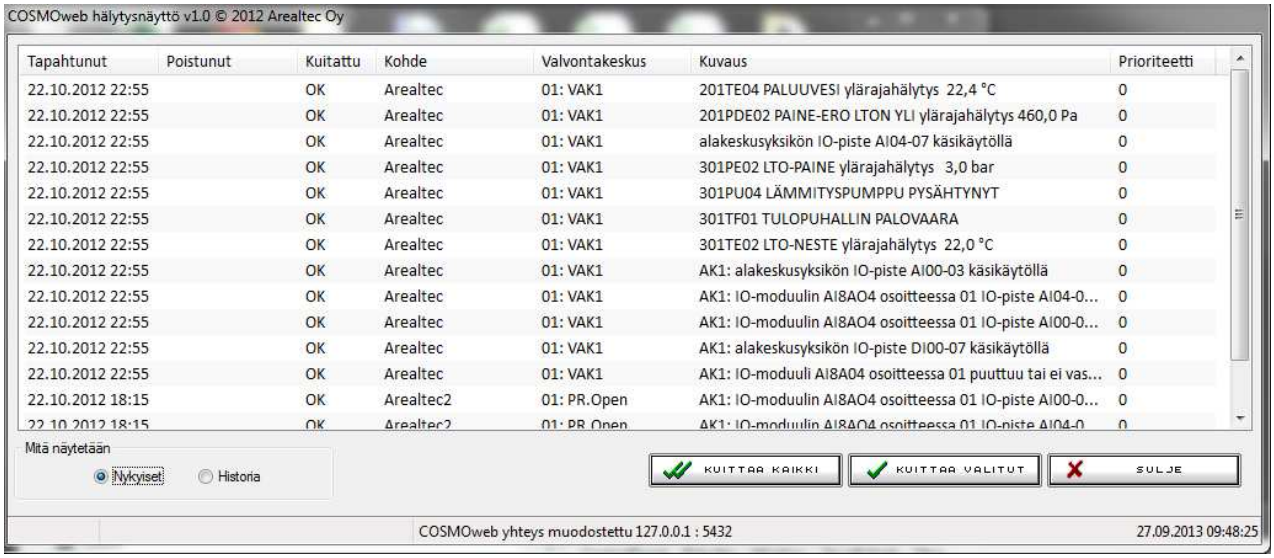
1.1 Ohjelmisto ja sen tavoitteet

Tämän insinöörityön tarkoituksena on toteuttaa yksinkertainen hälytyskäsittelyohjelmisto DEOSin alakeskukselle. Ohjelman tulee toimia Windows 7/8 (32/64bit) käyttöjärjestelmissä. Ohjelman pääasiallinen käyttötarkoitus tulee olemaan VAK luokse asennetussa kosketusnäytössä ja parantaa olemassa olevaa hälytyskäsittelyä. Nykyinen alakeskuksen hälytyskäsittely osaa ainoastaan näyttää voimassa olevat hälytykset.

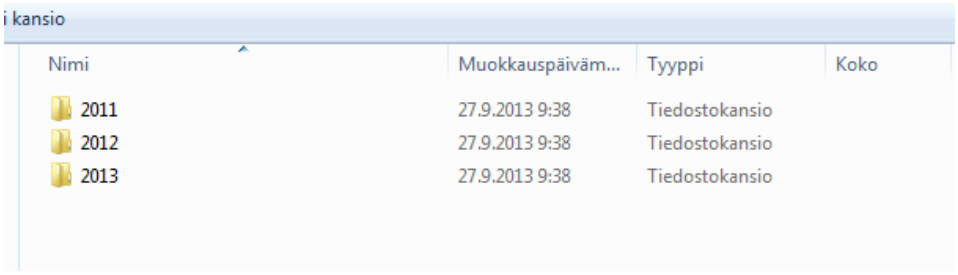
Ohjelma osaa näyttää aktiivisen hälytyksen, poistuneen hälytyksen eri väreillä sekä kirjoittaa hälytyslokia. Lisäksi käyttäjä voi kuitata hälytyksen huomioiduksi (tapahtumakirjanpito). Ohjelma näyttää hälytyksen käyttöliittymässään, jos a) se on voimassa oleva b) sitä ei ole kuitattu.

Ohjelma osaa tunnistaa poistuneen hälytyksen ja sitä vastaavan tapahtuneen hälytyksen indeksin perusteella. Tällöin hälytyksen aikaleima kirjoitetaan Poistunut –sarakkeeseen sekä rivin tekstit väritetään vihreällä. Tapahtunut väri on musta.

Esimerkki käyttöliittymästä. Prioriteetti-kenttää ei esitetä tehtävässä sovelluksessa. Kohde sekä valvontakeskus-kenttä asetetaan ohjelman asetuksissa (esim. ini-tiedosto).



Hälytysloki tallennetaan kronologisesti omaan hakemistorakenteeseen automaattisesti. Tallennusmuotona teksti (txt) ja jokainen hälytys tallennetaan omalle riville muodossa: Aikaleima<TAB>teksti



1.2 Käsitteitä

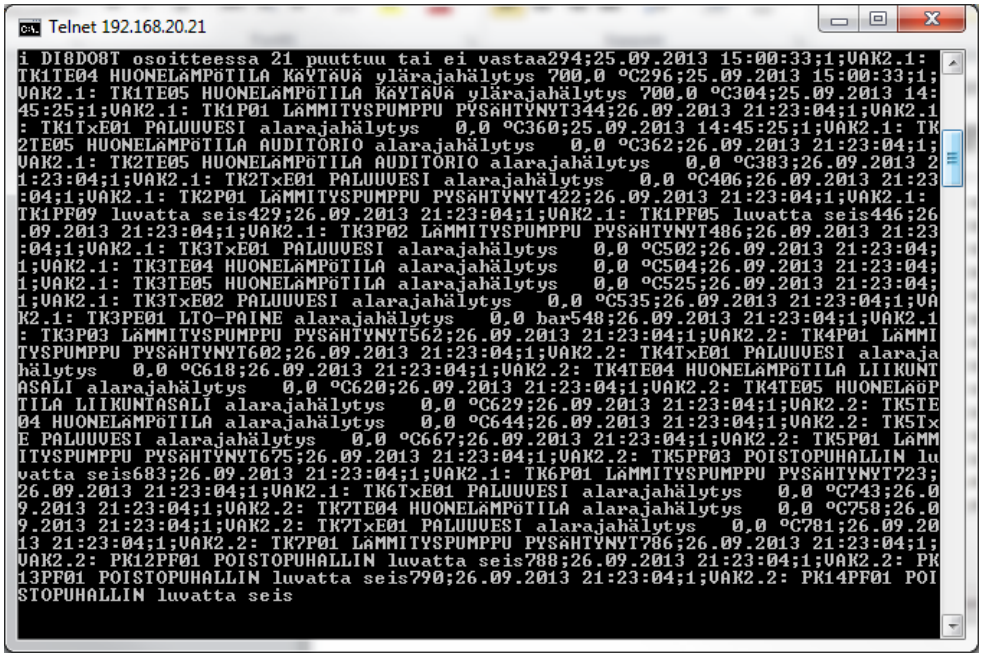
DEOS
Saksalainen yritys, joka kehittää ja valmistaa laitteita sekä sovelluksia rakennusautomaatioon.

Alakeskus
Vapaasti ohjelmoitava laite, johon voidaan liittää tuloja sekä lähtöjä (IO-pisteitä).

VAK
Valvonta-alakeskus sisältää alakeskuslaitteet sekä niiden vaatimat apulaitteet johtoteineen.

1.3 Tarvittavat tiedot

Ohjelma kuuntelee alakeskuksen TCP-porttia 3060, joka ilmaisee kaikki tapahtuneet hälytykset TCP-ASCII muodossa.



Komentoriviltä TELNET <alakeskuksen IP-osoite> 3060

Hälytystiedon muoto on seuraava:
indeksi [0…1983];aikaleima [dd.mm.yyyy hh:mm:ss];tapahtunut/poistunut [0/1];hälytysteksti [0…255 merkkiä]

1.4 Vaatimukset

Ohjelmointikieli on vapaasti valittavissa, mutta kuitenkin niin, että ohjelman täytyy toimia itsenäisesti ilman erikseen asennettavia kirjasto- tai alustasovelluksia.

Kun hälytysnäyttösovellus havaitsee uuden tapahtuman (tapahtunut/poistunut hälytys) ohjelma voidaan asettaa käyttöliittymän päällimmäiseksi sovellukseksi.

Hälytysnäytön järjestystä voidaan muuttaa klikkaamalla otsikkoa, esim. aakkosjärjestykseen, uusin ylimpänä. Oletusjärjestys ohjelman käynnistyessä on, että uusin tapahtunut hälytys on ylimpänä.

Ohjelma ilmaisee yhteyden alakeskukseen tilirivillä esim. ”Yhteys muodostettu alakeskukseen 192.168.20.11:3060”. Jos yhteyttä ei ole, ohjelma ilmaisee tämänkin punaisella tekstillä: ”Ei yhteyttä alakeskukseen 192.168.20.11:3060”.

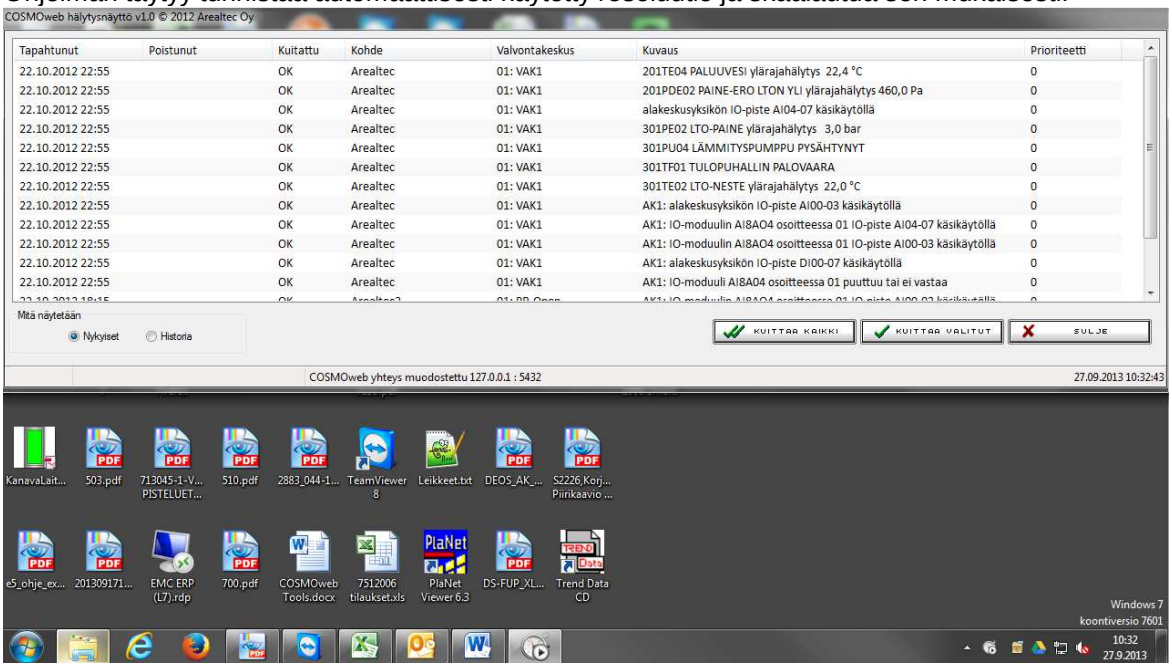
Ohjelman täytyy kirjoittaa omaa kirjanpitoa näytettäviä hälytyksiä varten. Kirjanpidossa täytyy olla ainakin seuraavat kentät:

- Tapahtuman indeksi
- Tapahtuman aikaleima
- Tapahtunut/poistunut (0/1)
- Tapahtuman teksti
- Onko kuitattu (0/1)

Kirjanpidosta poistetaan rivi, jos tapahtuma on poistunut sekä kuitattu.

Ohjelman yhteysasetukset (alakeskuksen IP-osoite, portti, Kohde-teksti, VAK-teksti) asetella esimerkiksi initiedostosta.

Ohjelman täytyy tunnistaa automaattisesti käytetty resoluutio ja skaalautua sen mukaisesti.



1.5 Vikasietoisuus

Jos yhteyttä alakeskukseen ei voida muodostaa, ohjelma yrittää esim. 20 sekunnin välein muodostaa yhteyttä.

Jos alakeskus ilmaisee poistuneen hälytyksen, mutta hälytystiedoissa ei löydy tapahtunutta hälytystä, ohjelma näyttää uuden rivin, jossa on merkitty sekä tapahtunut- että poistunut-aikaleima samoiksi. (Tällöin ei tiedetä, milloin Tapahtunut-hälytys on tullut).

Jos ohjelma käynnistetään uudelleen, täytyy näkymän palautua samaan näkymään ennen ohjelman sammutamista oman hälytyskirjanpitonsa avulla.