OULUN SEUDUN
AMMATTIKORKEAKOULU

Esa Rauman

RADIOKILPAILUN TULOSPALVELUOHJELMISTO

RADIOKILPAILUN TULOSPALVELUOHJELMISTO

Esa Rauman
Opinnäytetyö
4.4.2011
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

OULUN SEUDUN AMMATTIKORKEAKOULU TIIVISTELMÄ

Koulutusohjelma	Projekti	Sivuja	+	Liitteitä
Tietotekniikka	Insinöörityö	44	+	2
Suuntautumisvaihtoehto	Aika			
Ohjelmistotuotanto	2010			
Työn tilaaja	Työn tekijä			
Contest Club Finland ry	Esa Rauman			
Työn nimi				
Radiokilpailun tulospalveluohjelmisto				
Avainsanat				
Ohjelmointi, C++, SQL, ohjelmistokehitys, ohjelmistotuotanto, radioamatööritoiminta				

Projektissa suunniteltiin ja toteutettiin uusi ohjelmisto, jolla pystytään tekemään kaikki radiokilpailujen tulospalvelussa tarvittavat toimenpiteet, kuten esimerkiksi kilpailijoiden yhteyslokien ristiintarkastus, pisteiden laskenta ja lopputulosten raportointi muutamassa eri muodossa. Yksi tavoitteista oli, että ohjelmisto toimisi mahdollisimman monella eri käyttöjärjestelmällä. Tässä projektissa toteutettiin ns. proof-of-concept-versio, jolla tutkittiin, onko kyseisenlaisen ohjelmiston toteuttaminen mahdollista ja mitä vaatimuksia automatisoitu lokien tarkistus aiheuttaa itse kilpailusäännöille ja kilpailijoille.

Ohjelmiston toteutuksessa käytettiin C++-ohjelmointikieltä Qt-kirjastojen avulla, jotta saavutettaisiin tuki mahdollisimman monelle käyttöjärjestelmäalustalle. Suunnittelu ja toteutus tapahtui lähinnä soveltaen agile-menetelmiä, jolloin voitiin joustavasti suunnitella sitä mukaa, kun ohjelmiston varsinainen koodaus eteni. Ohjelmiston tietokantapohjana päädyttiin käyttämään SQLite-tietokantamoottoria sen sulautetun luonteen takia, jolloin se soveltuu erinomaisesti sovelluksen omaksi tiedon tallennusmuodoksi.

Lopputuloksena syntyikin ohjelmisto, joka sai nimekseen ”KVULogChecker”. Työn tilaajalle ehdotettiin myös muutamia asioita sähköisten lokien oikean muotoilun tärkeydestä, jotta ne voitaisiin tarkistaa automatisoidusti, ja tämä saattaa johtaa joissakin kilpailuissa sääntömuutoksiin.

OULU UNIVERSITY OF APPLIED SCIENCES ABSTRACT

Degree prgramme	Thesis	Pages + Appendices
Information technology	Bachelors thesis	44 + 2
Line	Date	
Software engineering	2011	
Commissioned by	Author	
Contest Club Finland ry	Esa Rauman	

Thesis title

Results service software for radioamateur contests

Key words

Programming, C++, SQL, software development, software engineering, Radioamateur contests

In this project a new program was planned, which could handle all necessary operations needed to provide results service, like for example cross checking of contact logs, counting scores and report final results in couple of different forms. One of the objectives for this project was, that the developped software would run on multiple operating systems. In this project so called proof-of-concept - version was implemented to study, is this even possible and what kind of demands this kind of automated log checking would set to contest rules, contest organizers and the compoetitors.

In implementation of this software C++ and Qt-framework was used to achieve the multi-platforms support. Design and implementation was done mainly with agile-methods which were adapted to projects needs, when design and implementation could be done in flexible maner. For database platform SQLite was used, because of it's embedded qualities, so it is a perfect fit for applications own file format.

As a result, a software was implemented, which is called ”KVULogChecker”. Some things were suggested to the projects commisioner, concerning correct formatting of the electronic logs, so they can be checked automatically and these suggestions may lead to some changes in contest rules.

SISÄLLYS

1 JOHDANTO

Opinnnäytetyön tilaajana toimiva Contest Club Finland ry (CCF) on yhdistys, jonka tarkoituksena on toimia suomalaisen radiourheilun katto-organisaationa. Yhdistys on perustettu 24.2.1996 (1).

CCF järjestää radioamatöörikilpailuja, joiden tulospalvelun hoitamiseen tarvitaan uusi sovellus, joka tuottaisi tulospalvelun tarvitsemat dokumentit eli tarkistaisi kilpailijoiden lähettämät yhteyslokit ja laskisi kilpailijoiden lopulliset pisteet. Uuden sovelluksen nimeksi on päätetty KVULogChecker. Ohjelmistosta toteutetaan tämän työn puitteessa ns. proof-of-concept-versio, joka ei välttämättä sisällä kaikkea suunniteltua toiminnallisuutta. Sen pääasiallinen tarkoitus on tutkia, onko tällaisen ohjelmiston luominen mahdollista ja mitä vaikutuksia sillä on suomalaisten radiokilpailujen sääntöihin ja sähköisiin lokitiedostoihin.

Tähän asti tuloslaskenta ja lokien ristiintarkastus on tehty Suomessa lähes puhtaasti käsityönä, joten lopullisten tulosten saaminen julkaisukuntoon on ollut erittäin hidasta. Osallistujia kilpailuihin saattaa olla satoja, ja jokainen loki sisältää helpostikin tuhansia rivejä.

2 RADIOKILPAILU

Radiokilpailussa radioamatööriasemalta, jota operoi yksi tai useampi henkilö, yritetään saada mahdollisimman monta yhteyttä toisiin asemiin yleensä rajatun ajan puitteissa. Monesti kilpailun säännöissä määritellään myös käytettävä lähetelaji (sähkötys, puhe ja erilaiset digitaaliset lähetelajit). Yhteyden sisältö määritellään kilpailun säännöissä. Pääosassa suomalaisia kilpailuja se sisältää rs(t)-raportin, joka kuvaa vasta-aseman luettavuutta ja signaalin laatua. Raportin lisäksi sanomaan kuuluu myös juokseva sarjanumero sekä maakuntatunnus. Jokaisesta onnistuneesta yhteydestä molemmat asemat saavat säännöissä määrätyn määrän pisteitä. Jos vasta-asema sijaitsee alueella, joka on määritelty kilpailun säännöissä kertoimeksi, saa kilpailija lisäpisteitä yhteydestä. Suurimmassa osassa suomalaisia kilpailuja on kertoimeksi määritelty maakuntatunnus. Kertoimen lisäpisteet voi saada taajuusaluekohtaisesti vain kerran.

Kilpailun jälkeen osallistujat lähettävät listan pitämistään yhteyksistä eli lokin kilpailun järjestäjälle, joka tarkistaa lokit ristiin sanomassa olevien virheiden varalta ja laskee lopputulokset.

3 KETTERÄT MENETELMÄT

Vuonna 2001 muutamat ohjelmiston kehittäjät julkaisivat *agile manifesto*n, joka sai aikaan eräänlaisen vallankumouksen ohjelmistoprojektien hallinnossa. Manifestin sisältö on seuraavanlainen:

”Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan:

Yksilöitä ja vuorovaikutusta enemmän kuin prosesseja ja työkaluja
Toimivaa sovellusta enemmän kuin kokonaisvaltaista dokumentaatiota
Asiakasyhteistyötä enemmän kuin sopimusneuvotteluita
Muutokseen reagoimista enemmän kuin suunnitelman noudattamista.

Vaikka oikeallakin puolella on arvoa, me arvostamme vasemmalla olevia asioita enemmän.” (2.)

Näiden sekä manifestissa mainittujen 12 periaatteen pohjalta on syntynyt lukuisia ketteriä menetelmiä, joista tunnetuimpia lienevät Scrum ja Extreme Programming. (3; 4.)

Ketterät menetelmät yrittävät minimoida riskiä tekemällä ohjelmointia yleensä lyhyissä iteraatioissa. Niistä jokainen on kuin miniprojekti, johon sisältyvät kaikki perinteiset ohjelmistotuotannon vaiheet jossain muodossa. Tavoitteena on, että jokaisen iteraation jälkeen on olemassa toimiva julkaisukelpoinen ohjelmisto, johon joka iteraatiossa on lisätty toiminnallisuutta. (2.)

Ketterissä menetelmissä korostetaan suoraa kontaktia ohjelmointitiimin ja asiakkaisiin, ja mielellään vielä niin, että kommunikointi tapahtuisi mahdollisimman paljon kasvotusten. Näissä menetelmissä dokumentaatio ei ole itseisarvo eikä projektin edistymistä mitata dokumentaation määrällä, vaan keskitytään itse ohjelmiston tuottamiseen. Tästä ei pidä kuitenkaan vetää sitä johtopäätöstä, että dokumentointia tai suunnittelua ei tehtäisi lainkaan, vaan niitä tehdään tarvittaessa ja varsinaisen ohjelmoinnin rinnalla, ei kaavamai-

sen prosessina, kuten perinteisessä vesiputousmallissa. Tämä saa myös aikaan sen, että voidaan reagoida ohjelmiston määrityksiin vääjäämättä tuleviin muutoksiin ja lisäyksiin dynaamisemmin. (2.)

3.1 Scrum

Hirotaka Takeuchi ja Ikujiro Nonaka esittivät ensimmäisenä teoksessaan "The New Product Development Game" uudenlaisen, holistisen suhtautumistavan tuotekehitykseen, jossa yksi ryhmä suorittaa tuotteen kehittämisen alusta loppuun vahvasti lomittuneella vaiheistuksella. Tämän tiimin toimintaa verrataan rugby-joukkueesen, jossa kaikki pyrkivät etenemään yksikkönä, siirrellen palloa eteen- ja taaksepäin. Varsinaisen scrum-menetelmän kuitenkin kehittävät Jeff Sutherland, John Scumniotales ja Jeff McKenna. (5.)

Scrum-menetelmässä työskennellään iteratiivisesti ja inkrementaalisesti, lisäten vähitellen ominaisuuksia ja parannellen olemassa olevia, useiden toteutuskierrosten aikana. Scrum-menetelmässä näitä kierroksia kutsutaan pyrähdyksiksi tai englanniksi sprinteiksi. Sprintit kestävät tyypillisesti kahdesta neljään viikkoa. Sprintin alussa tiimi valitsee projektin kannalta tärkeimmät asiat, jotka sprintin aikana pyritään toteuttamaan. Sprintin lopussa taas pidetään esittelytilaisuus, jossa projektiryhmä esittelee konkreettisesti aikaansaannoksiaan. (5.)

Menetelmässä käytetään itsenäisiä tiimejä, joiden koon tulisi olla ihanteellisesti seitsemän henkeä (± 2 henkeä). Tiimi päättää itsenäisesti seuraavan sprintit tavoitteet ja tehtävät sekä sen, miten niihin päästään. Tiimi voi myös itse päättää työskentelytavoistaan. (5.)

Yhden tiimiläisen on toimittava ns. scrum masterina, joka on eräänlainen projektipäällikkö ja tiimin vetäjä. Hänellä tosin ei ole suoranaista valtaa tiimiin, vaan hänen tehtävänsä on lähinnä poistaa työskentelyä haittaavia esteitä sekä dokumentoida tiimin työskentelyä. Hänen vastuullaan on myös varmistaa ja seurata sitä, että tiimin työskentely on tuottavaa ja pelisääntöjä nouda-

tetaan. Jos sprintin tavoitteiden saavuttaminen näyttää epävarmalta, scrum master myös suorittaa tarvittavat toimet, että tavoitteet saavutettaisiin. (5.)

Scrum-tiimin ulkorajapinnassa toimiva tuoteomistaja toimii asiakkaan edustajana. Hänen tehtäviinsä kuuluvat vaatimusten määrittely sekä julkaisujen sisällöstä ja aikataulutuksesta päättäminen. Hän huolehtii myös projektin kannattavuudesta ja valvoo, että sprinteissä tuotettavat asiat ovat tuotteen vaatimusten ja onnistumisen kannalta oleellisia. Tuotteenomistaja ohjaa sprinttien aloituspalavereita ja hyväksyy sprintin lopulliset tuotokset. (5.)

Menetelmässä käytetään kolmea taulukkoa, joilla seurataan ja suunnitellaan projektin edistymistä. Taulukot ovat tuotteen työlista, julkaisun työlista ja toteutusvaiheen työlista. Tuotteen työlista sisältää kaikki vaatimukset, jotka tuotteen tulisi täyttää. Julkaisun työlista sisältää kaikki vaatimukset, jotka seuraavaksi julkaistavan version tulisi täyttää, ja työvaiheen työlista sisältää sprintissä toteutettavat vaatimukset. (5.)

Scrum-menetelmässä suositellaan pidetäväksi päivittäisiä enintään 15 minuuttia kestäviä kokouksia, joissa kysytään projektiin osallistuvilta henkilöiltä seuraavia asioita: mitä olet tehnyt edellisen scrum-palaverin jälkeen, mitä aiot tehdä seuraavaan kokoukseen mennessä sekä mikä on hankaloittanut työtäsi? (5.)

3.2 Extreme Programming (XP)

Extreme Programming -menetelmän on kehittänyt Kent Beck työskennellessään Chryslerilla projektinjohtajana (6). XP-menetelmässä on neljä perusarvoa: kommunikaatio, yksinkertaisuus, palaute ja rohkeus (7).

Suurin osa ohjelmistoprojektien ongelmista johtuu tyypillisesti kommunikaation puutteesta tai sen laadusta. XP-menetelmässä projektiin osallistuvat joutuvat kommunikoimaan toistensa kanssa. Menetelmässä luotetaan näin ollen vähemmän kirjalliseen dokumentaatioon ja enemmän suullisen kommunikaatioon. Yksinkertaisuudella tarkoitetaan sitä, että järjestelmä toteutetaan niin

yksinkertaisella tavalla kuin suinkin mahdollista. Palautetta kerätään aina toteutetun vaiheen jälkeen kaikilta projektiin osallistuvilta, myös asiakkailta. Menetelmän arvoihin kuuluu rohkeus, sen osittaisen radikaaliuden vuoksi ja asiakkaan puolelta myös siksi, että hänelle ei voida antaa selkeää lupausta ohjelmiston hinnasta, vaan hän on itse mukana projektissa vaikuttamassa kustannuksiin. (7.)

Menetelmään kuuluu olennaisena osana pariohjelmointi, jossa toinen ohjelmoijista kirjoittaa koodia ja toinen tarkistaa sitä. Näin suuri osa loogisista kömmähdyksistä ja kirjoitusvirheistä saadaan kiinni jo ennen testausta. Merkittävää menetelmässä on myös se, että siinä korostetaan automatisoidun testauksen merkitystä, eli suositellaan kirjoittamaan ensin automatisoitu testi ja vasta sen jälkeen varsinainen ohjelmakoodi. Testauksen korostaminen ilmenee myös niin, että mitään koodia, jolle ei ole olemassa testiä, ei voida hyväksyä. (6.)

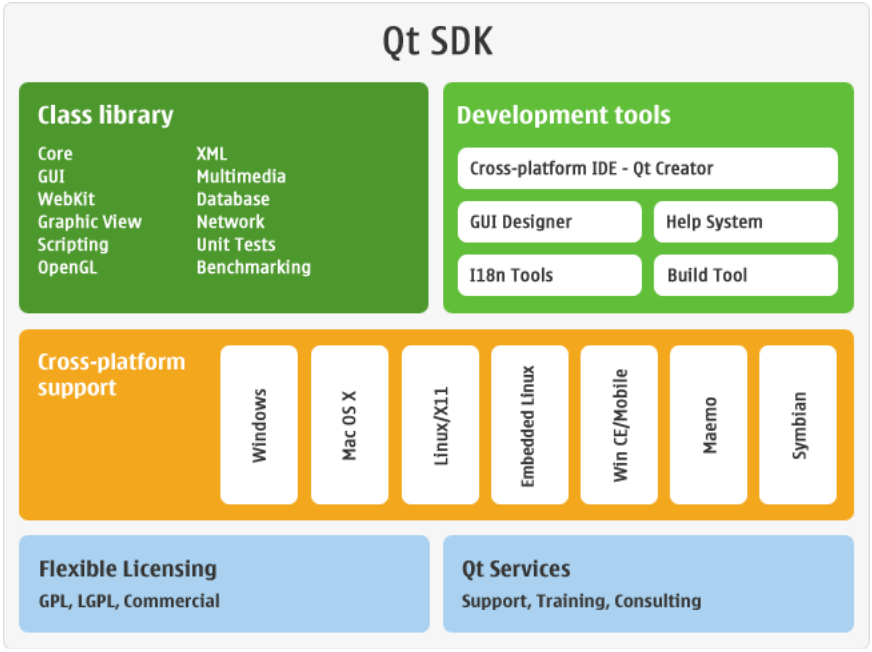
XP-menetelmä kannustaa käyttämään myös ohjelmistokehitystä automatisoivia työkaluja aina, kun se on mahdollista (mm. automatisoitu testaus ja integroiminen). Menetelmässä pyritään myös siihen, että ylitöitä ei tulisi, vaan ohjelmoijilla olisi säännöllinen työaika. (6; 7.)

4 QT-OHJELMISTOVIITEKEHYS

Havaard Nord ja Eirik Chambe-Eng alkoivat kehittää Qt-ohjelmistoviitekehystä vuonna 1991. Kaksi ensimmäistä versiota toimivat vain kahdella käyttöjärjestelmäalustalla (X11/Unix ja Windows). Loppuvuodesta 2001 julkaistiin Qt:n versio 3.0, joka lisäsi tuen myös Mac OSX -käyttöjärjestelmälle. (8.)

Nokia osti Nordin ja Chambe-Engin perustaman Trolltech ASA -yrityksen vuonna 2008, jolloin yrityksen nimi muuttui ensin Qt Softwareksi ja sitten Nokian Qt Development Frameworks -yksiköksi. Siitä lähtien tavoitteena on ollut tehdä Qt:sta pääasiallinen kehitysalusta Nokian laitteille. Nykyään Nokia kehittää Qt:ta yhteistyössä erittäin aktiivisen käyttäjäyhteisön kanssa. (8.)

Qt on ohjelmistoviitekehys, jolla voidaan samalla koodilla tuottaa ohjelmistoja monille erilaisille käyttöjärjestelmäalustoille, kuten Microsoft Windows, Apple Mac OSX ja Linux, sekä erilaisille sulautetuille laitteille, kuten matkapuhelimille (kuva 1).



KUVA 1. Qt-kehitysympäristö (9)

Kuten kuvasta 1 voidaan havaita, Qt koostuu monesta erilaisesta moduulista, joista ohjelmistokehittäjän kannalta tärkeimpiä ovat luokkakirjasto ja kehitystyökalut. (9.)

4.1 Tärkeimmät ominaisuudet

Seuraavassa käydään karkeasti läpi Qt:n tärkeimpiä ominaisuuksia, kuten signaaleja ja slotteja sekä widgettejä. Käsittelen myös työssä merkittävässä osassa olevia Qt:n geneerisiä kontainereita, jotka ovat määriteltyinä Qt:n luokkakirjastossa.

4.1.1 Signaalit ja slotit

Signaalit ja slotit ovat Qt:n keskeisimpiä ominaisuuksia. Niitä käytetään olioiden väliseen kommunikointiin. Signaali-slot-mekanismi on täysin tyyppiturvallinen, koska lähtevän signaalin pitää vastata vastaanottavan slotin tyyppitietoja täydellisesti. (10.)

Signaali lähetetään, kun oliossa tapahtuu jotain, joka voi kiinnostaa muita olioita. Esimerkiksi käyttäjän klikatessa käyttöliittymässä olevaa nappia lähettää painiketta edustava olio signaalin ”clicked()”. Ohjelmoija voi luoda omiin luokkiinsa signaaleita määrittelemällä ne otsikkotiedostoon signals-kohdan alle. Signaaleja ei tarvitse implementoida, vaan niiden olemassa olo on vain ilmoitettava. (10.)

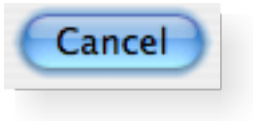
Slotit voivat ottaa vastaan signaaleja, mutta ne ovat myös tavallisia C++-jäsenfunktioita, jotka määritellään luokan otsikkotiedoissa näkyvyyden mukaan private- tai public slots -kohdan alle. Slotin paluutyypin tulee olla aina void, eli se ei voi palauttaa mitään arvoa kutsuvalle funktiolle. (10.)

Qt:n signaalit ja slotit eivät vaadi, että luokat olisivat tietoisia toisistaan. Tämä helpottaa uudelleenkäytettävien luokkien kirjoitusta (11). Esimerkiksi slot ei ole tietoinen siitä, onko siihen kytketty signaalia, ja vastaavasti signaali ei tiedä, onko sitä yhdistetty slottiin. Näin voidaan rakentaa oikeasti toisistaan riippumattomia komponentteja (10).

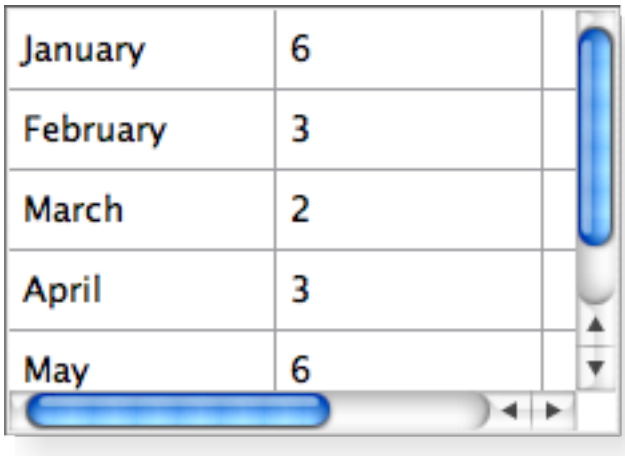
4.1.2 Widgetit

Widgetit ovat käyttöliittymän perusrakennuspalikoita. Jokainen käyttöliittymäkomponentti, joka on sijoitettuna käyttöliittymään, on itsenäinen widget. Käyttöliittymän ikkunakin on widget (12). Jos widgetille ei ole määritelty sen omistavaa parent widgettiä, siitä muodostuu oma ikkuna. Kuvissa 2–4 on esimerkkejä erilaisista käyttöliittymä-widgeteistä.

KUVA 2. QLineEdit (13)



KUVA 3. QPushButton (13)



KUVA 4. QTableWidget (13)

Ohjelmoija voi myös kehittää ja muokata olemassa olevia widgettejä soveltumaan paremmin omiin käyttötarkoituksiinsa tekemällä QWidgetistä tai paranneltavasta widgetistä periytetyn luokan. (11.)

4.1.3 Kontainerit ja iteraattorit

Kontaineria voidaan ajatella arkistokaappina, joka sisältää dokumentteja. Samoin ohjelmoinnissa kontaineri on tiedon tallennuspaikka, johon voidaan tallentaa erilaisia tietorakenteita, kontainerin tyypin mukaan.

Qt tarjoaa yleiskäyttöisiä C++-malliluokkiin perustuvia kontainereita, joihin voi tallentaa määrittelemänsä tietotyypin mukaista dataa. Kontainerit on jaettu kahteen ryhmään, assosiatiivisiin ja sekventiaalisiin, niiden toimintatavan mukaisesti. (14.)

Assosiatiivisiin kontainereihin kuuluvat QMap-, QHash- ja QSet-nimiset rakenteet. QMap<Key, T> ja QHash<Key, T> ottavat C++-malliparametreikseen kaksi tietotyyppiä, joista ensimmäinen toimii avaimena ja toinen arvona. Tästä johtuu myös tämän kontaineriryhmän nimitys: avain on linkitettynä arvoon. Kun tiedetään avain, voidaan hakea siihen liittyvä arvo. QMap ja QHash ovat toiminnaltaan samankaltaisia, mutta eroavat toisistaan algoritmisesti. Pääasiallisena erona on, että QMap tallentaa tiedot avaimen mukaan järjestettynä ja QHash enemmän tai vähemmän satunnaisessa järjestyksessä. Toisena merkittävä erona voidaan myös pitää sitä, että QHashistä tiedon haku on nopeampaa. QSet taas tarjoaa yksiarvoisen matemaattisen joukon, josta hakeminen on hyvin nopeaa. (14; 15.)

Sekventiaalisiin kontainereihin puolestaan kuuluu pääasiassa erilaisia lista- sekä pinotyyppisiä rakenteita. Näistä merkittävimpinä mainittakoon QList<T>, QVector<T>, QStack<T> ja QQueue<T>. QList<T> sopii suurimpaan osaan sovellutuksia joustavuutensa ja helpon käytettävyytensä takia. Jos tarvitaan yhtenäistä muistialuetta, silloin on käytettävä QVector<T>-tyyppistä kontaineria, jossa haut ovat nopeita. Tosin muistialueen yhtenäisyyden vuoksi lisääminen kontainerin keskelle tai alkuun on hitaampaa. QStack<T> ja QQueue<T> ovat pinorakenteita, jotka eroavat toisistaan lähinnä siinä, missä järjestyksessä pinoon tallennetut tiedot saadaan sieltä ulos. QStack on normaali LIFO (Last In, First Out) -pino, jossa viimeisenä tallennettu tieto saadaan ensimmäisenä ulos. QQueue puolestaan on jono eli FIFO (First In, First

Out) -tyyppinen, jossa ensin tallennettu saadaan myös ensimmäisenä ulos. (14.)

Kontainereita voidaan myös sijoittaa sisäkkäin, kuten esimerkiksi QHash<QString, QList<NRAUBalticQSOCC> >, jollaista toteutetussa ohjelmassa käytettiin lokien tarkistuksessa. Avaimena toimi aseman kutsu ja arvona kyseisen aseman loki. Tämä mahdollisti helpon pääsyn halutun aseman lokiin. NRAUBalticQSOCC -tietorakenne on esiteltynä työssä myöhemmin. On tärkeää huomioida, että määrittelyn lopussa olevien >-merkkien välissä on vähintään yksi välilyönti. Muutoin kääntäjä tulkitsisi sen >>-operaattoriksi, jonka tehtävänä on siirtää bittejä oikealle (16,14).

Jos kontainereiden kanssa käytetään itse määriteltyjä tietotyyppejä, niiltä vaaditaan tiettyjen ehtojen täyttämistä. Omien tietotyyppien tulee sisältää konstruktori ja kopiokonstruktori. Tosin jos näitä ei ole, C++-kääntäjä toteuttaa ne oletuskonstruktorilla. Jotkut kontainerit vaativat myös joidenkin vertailuoperaattorien ylikirjoittamista, mikä onkin useasti muutoinkin suotavaa tiedon etsimisen ja vertailemisen helpottamiseksi. (14.)

Kontainereiden yhteydessä on otettava esille myös iteraattorit, joiden avulla voidaan iteroida kontainerin sisältöä. Qt tarjoaa mahdollisuuden käyttää Java- tai STL (C++ Standard Template Library) -tyyppisiä iteraattoreita. Java-tyyppiset ovat kätevämpiä käyttää, mutta hieman tehottomampia kuin STL-tyyppiset.

Java-tyyppisten iteraattoreiden käyttö on suurin piirtein samanlaista kaikille kontainerluokille. Iteraattoriluokan nimi on sama kuin kontainerluokan, mutta sen loppuun lisätään Iterator (esim. QListIterator<T>). Java-tyyppisillä iteraattoreilla käydään kontainerin sisältö läpi tyypillisesti while-silmukassa, jonka ehtona on iteraattorin hasNext()-funktio, eli kun kontainerilla ei ole enää seuraavaa alkiota, poistutaan while-silmukasta. STL-tyyppisessä ratkaisussa iterointi taas tapahtuu pääosin for-silmukassa, joka alkaa iteraattorin begin()-funktiosta ja päättyy sen end()-funktioon. (14.)

Qt tarjoaa myös omat Java-tyyliset iteraattorit kontainereille, joiden sisältöä muutetaan iteroitaessa. Nämä iteraattorit ovat tunnistettavissa mutable-sanasta iteraattorin nimessä. Iteroitavaa kontaineria ei saa suoraan muuttaa, vaan on käytettävä iteraattorin tarjoamia funktioita. (14.)

4.2 Qt:n tarjoamat kehitystyökalut

Kuten kuvasta 1 nähdään, kuuluu Qt:hen myös erilaisia kehitystyökaluja. Näistä merkittävin on Qt Creator, joka tarjoaa integroidun kehitysympäristön sovelluksen kehittämiseksi. Creator koostuu tekstieditorista, jolla koodi kirjoitetaan, käyttöliittymäeditorista (joka on itse asiassa Qt Designer, josta myöhemmin), ohjelmistoprojektin hallinnasta ja debuggerin käyttöliittymästä.

Qt Designer toimii käyttöliittymän suunnittelun graafisena apuvälineenä. Designerissa luodaan käyttöliittymän ikkuna, jolle voidaan vetää ja pudottaa Qt:n vakiowidgettejä, joista puolestaan muodostuu varsinainen käyttöliittymälomake. Designerin widgettivalikoimaan on myös mahdollista liittää itse tehtyjä käyttöliittymäwidgettejä. Qt Designeria on mahdollista käyttää erillisenä ohjelmana tai Qt Creatoriin integroituna, jolloin voi siirtyä suoraan varsinaisessa koodissa oleviin käyttöliittymän toimintaa toteuttaviin slot-funktioihin pikavalinnalla. Designer muodostaa käyttöliittymästä xml-muotoisen .ui-päätteisen tiedoston, joka käännetään C++-koodiksi käyttöliittymäkääntäjällä (uic). Tämä tapahtuu automaattisesti käytettäessä Qt Creatoria.

Qt tarjoaa myös kansainvälistystyökalut, joilla voidaan samasta ohjelmasta luoda pienillä muutoksilla käännöksiä useille eri kielille. Käännöstyökalut ovat siinä määrin helppoja käyttää, että työn tekemisessä ei tarvitse olla minkäänlaista ohjelmointikokemusta.

Qt Assistant -järjestelmällä saadaan näkyviin koodia kirjoitettaessa Qt:n referenssidokumentaatio, mutta myös oman ohjelman käyttöohjeistus voidaan toteuttaa näillä työkaluilla.

Viimeisenä osana Qt:n työkaluissa ovat tietenkin varsinaiset käännöstyökalut, joihin kuuluvat mm. metaobjekti-kääntäjä (moc) ja käyttöliittymäkääntäjä (uic).

5 SQLITE- SULAUTETTU TIETOKANTA

SQLite on serveritön, transaktioita tukeva sulautettu tietokanta. Se ei käytä erillisiä palvelimia tai palvelinprosesseja kuten suuremmat tietokantajärjestelmät. SQLiteä ei tarvitse myöskään konfiguroida tai asentaa millään tavalla, vaan se on heti valmis käytettäväksi.

Tietokanta muodostuu yksittäisestä tiedostosta, jolloin sitä voi ajatella enemmänkin ohjelmiston omana tiedostomuotona kuin tietokantajärjestelmänä. Sitä voidaan käyttää myös pelkästään muistissa toimivana kantana, jolloin siitä saa nopean tilapäistallennuspaikan tiedoille. (17.)

Sulautetun tietokannan luonteen takia SQLite ei toteuta ihan täysin ANSI SQL -standardia. Esimerkiksi viite-eheyksien valvominen on tullut SQLiteen vasta versiossa 3.6.19 ja se ei edelleenkään ole oletusarvoisesti käytössä, vaan on otettava erikseen käyttöön komennolla ”PRAGMA foreign_keys = ON;” (18). Muita oleellisia puutteita ovat taulujen muokkaamiseen (ALTER TABLE) liittyvien komentojen puute (DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT), sekä se, että näkymiin (VIEW) ei voida kirjoittaa (19). Tämä tulikin työn aikana monesti esiin, kun kyselyt eivät toimineet kuten suuremmissa tietokantajärjestelmissä. SQLiten ”INSERT INTO”-lauseessa ei voi olla myöskään useita ”VALUES”-lauseita, vaan tiedot on tallennettava rivi kerrallaan.

SQLite valittiin sovelluksen tietokantamoottoriksi, koska se täyttää liitteessä 1 olevan vaatimuksen FA1-1 (Järjestelmällä pitää pystyä tekemään oma tiedosto jokaista kilpailua varten) sekä tarjoaa SQL:n vahvuudet tietojen haussa ja vertailussa. SQLite toimii myös useilla käyttöjärjestelmäalustoilla, joten se täyttää myös liite 1:ssä mainitut yleiset vaatimukset.

6 ABSTRAKTI C++-LUOKKA

Abstrakti luokka on luokka, jossa on vähintään yksi täysin virtuaalinen metodi. Toki luokassa voi olla normaaleja jäsenmuuttujia ja ei-virtuaalisia metodejakin. (20.)

Abstrakteja luokkia käytetään lähinnä pelkästään toisten luokkien yliluokkina. Esimerkkinä tällaisesta luokasta voisi käyttää vaikka muoto-nimistä luokkaa. Muoto on itsessään jo abstrakti käsite eikä siitä ole järkevää muodostaa olioita, mutta kaikille muodoille yhteisiä toimintoja voidaan kyllä määritellä metodeiksi, kuten esimerkiksi piirrä ja pyöritä. Tosin niiden toteuttamiseen ei muoto-luokassa voida ottaa kantaa (21;20).

C++:ssa täysin virtuaalinen metodi määritellään, kuten normaali metodi, mutta perään lisätään ”=0” (esimerkiksi ”void piirra(int x, int y)=0”). Tämä on syntaksina melko outo, mutta johtuu siitä, että kielen kehittäjä ei halunnut ottaa kieleen uutta avainsanaa, jonka lisääminen olisi todennäköisesti viivästyttänyt toiminnon saamista C++-kieleen. Metodi siis vain määritellään, mutta toteutus jätetään alaluokille. (20.)

Rajapinnat ovat täysin abstrakteja luokkia, jotka sisältävät vain abstrakteja metodeita. Joissakin ohjelmointikielissä (esim. Java) on olemassa oma avainsana ja syntaksi täysin abstrakteille luokille. Sallimalla ainoastaan rajapintojen moniperintä voidaan näin välttää moniperinnän pahimpia ongelmia (20).

Rajapinta antaa takuun siitä, että liittymä on mahdollisimman abstrakti. Näin se mahdollistaa muun muassa riippumattomuuden käytetystä toteutuskielestä tai käyttöjärjestelmästä. (20.)

7 SÄÄNNÖLLISET LAUSEKKEET

Säännölliset lausekkeet ovat merkkijonoja, joilla etsitään kyseisen merkkijonon tarkoittamia vastaavuuksia jostain toisesta tekstistä. Niillä voidaan suorittaa myös tiettyjen ehtojen täyttävien merkkijonojen korvausta toiseen merkkijonoon. Säännöllisiin merkkijonoihin viitataan useimmiten niiden englanninkielisellä termillä ”Regular Expression” tai lyhennettynä ”RegEx”. (22.)

Säännölliset lausekkeet ovat saaneet alkunsa tietojenkäsittelytieteessä käsiteltävien automaattiteorian ja formaalin kieliteorian piiristä. SNOBOL-ohjelmointikielessä oli alustavasti toteutettu merkkijonojen etsiminen, mutta toteutus ei ollut identtinen säännöllisten lauseiden kanssa. Varsinaisen alkunsa säännöllliset lausekkeet saivat Unix-käyttöjärjestelmistä hyvin tutun grep-toiminnon myötä, jossa voidaan käyttää säännöllisiä lausekkeita tekstin etsimiseen. Perl-ohjelmointikieli toi vielä mukanaan joitain laajennuksia lausekkeisiin. (23.)

Säännöllisiä lausekkeita voidaan nykyään käyttää melkein kaikkien ohjelmointikielien kanssa sekä monissa kehittyneissä tekstieditoreissa. Niistä onkin suuri hyöty esimerkiksi etsittäessä tietynlaista dataa tai validoitaessa käyttäjän syötteitä. Säännölisten lausekkeiden suurin voima ja samalla heikkous on niiden syntaksissa, jolla saadaan tehtyä todella monimutkaisia hakuja muutamalla merkillä verrattuna siihen, että sama toteutettaisiin perinteisillä rakenteilla, joissa haettavaa tekstiä iteroitaisiin for tai while silmukassa if-else-rakenteilla. Ohjelmiston ylläpidettävyyden kannalta syntaksi muodostuu kuitenkin melko suureksi ongelmaksi epäselvyytensä ja huonon luettavuutensa takia. Monesti lausekkeet ovatkin ”koodaa kerran ja älä ikinä koske”-tyyppisiä, ellei niiden toimintaa ja tarkoitusta ole lähdekoodeihin kunnolla dokumentoitu.

Tässä raportissa ei käsitellä säännöllisten lausekkeiden syntaksia, koska internetistä on helposti löydettävissä useita oppaita niiden muodostamiseen.

Projektissa toteutetussa ohjelmassa säännöllisiä lausekkeita käytettiin cabrillo-tiedostojen oikeellisuuden tarkistamiseen ja käsittelemiseen, mikä muutoin olisi ollut erittäin työlästä.

8 ABSTRAKTI TEHDAS -SUUNNITTELUMALLI

Abstraktiolla tarkoitetaan yleisesti tiedon piilottamista, mutta sitä voidaan myös ajatella tiedon yleistämisenä. Abstraktin tehtaan tarkoitus on tarjota rajapinta, jonka avulla voidaan luoda toisiinsa liittyviä tai toisistaan riippuvia olioperheitä määrittelemättä niiden varsinaista luokkatyyppiä. (24; 25.)

Mallia voidaan ajatella vaikka analogiana autotehtaaseen, joka valmistaa pakettiautoja: Asiakkaalle ei ole merkitystä, kuinka ja missä kyseinen auto on rakennettu, kunhan sillä pääsee ajamaan. Oletetaan, että jossain vaiheessa tehdas päättää aloittaa henkilöautojen valmistuksen. Niiden valmistus ei tietenkään onnistu samalla tehtaalla, joten henkilöautoja varten avataan uusi tehdas. Edelleenkään ei ole asiakkaalle merkityksellistä, millä tehtaalla auto on valmistettu, kunhan hän pääsee autoa käyttämään. (24.)

Ohjelmoinnissa abstrakti tehdas luo oliota muista luokista (jotka liittyvät jollain tavalla toisiinsa) asiakkaalle. Ohjelmaan määritellään tarvittavan olion tyypiksi tämän abstrakti yläluokka, ja tehdas palauttaa yläluokan olion tilalle oikean alaluokan olion. Menetelmällä saadaan ratkaistua ongelma, joka ilmenee, kun on luotava ajonaikaisesti olioita, joiden tyyppiä ei voida kiinteästi kiinnittää ohjelmakoodiin. Näin vältytään myös saman koodin uudelleenkirjoittamiselta. Jos tietotyyppi olisi kiinteä, sama funktio jouduttaisiin kirjoittamaan uudelleen kaikille samaa toimintoa käyttäville, mutta eri tietotyypeille, mikä hankaloittaisi huomattavasti ohjelmakoodin ylläpitoa ja lisäisi myös virheiden riskiä. (24.)

Toteutetussa sovelluksessa käytettiin abstraktia tehdasta ratkaisemaan monia ajonaikaisen olion tyypin valintaan sekä ohjelmiston laajennettavuuteen liittyviä ongelmia.

9 CABRILLO-TIEDOSTOMUOTO

Cabrillo on tiedostomuoto, jota suurin osa kilpailulokin täyttämiseen käytetyistä ohjelmista tuottaa. Se on saavuttanut tärkeimmän tiedostomuodon aseman, jolla lokit palautetaan kilpailun järjestäjille. (26.)

Tiedon fyysinen tallennus cabrillo-muotoiseen lokiin tapahtuu ASCII-muodossa (ASCII, American Standard Code for Information Interchange). ASCII on digitaalisten merkkien tallennustapa, joka on avoin ja standardoitu. Näin varmistetaan, että tallennettua informaatiota voidaan käyttää mahdollisimman monessa eri ohjelmassa. (26.)

Fyysisesti standardoitu tiedostomuoto ei kuitenkaan riitä, vaan tiedoston sisältämät loogiset rakenteet on myös standardoitava (26). Cabrillo-tiedostomuodon suunnittelussa on pyritty siihen, että tiedoston sisältö olisi helposti ihmisen ja koneen luettavissa (verrattuna esimerkiksi xml-tiedostoon joka on vain lähinnä koneen luettavissa) (27).

Cabrillo-loki jakaantuu kahteen osaan: yhteenveto-osaan (ns. Summary sheet), joka sisältää kilpailijan tiedot (kuten esimerkiksi kutsun, kilpailun nimen, kilpailuluokan), ja varsinaiseen lokiosaan, jossa on listattuna kilpailijan pitämät yhteydet sanomineen (26).

Alla esimerkki lokin yhteenveto-osasta, joka on tiedoston alussa:

```
START-OF-LOG: 2.0
CREATED-BY: KVULog 0.6
CALLSIGN: OH6KVU
CONTEST: CQ-WW-SSB
CATEGORY: SINGLE-OP 10M LOW
CLAIMED-SCORE: 131784
NAME: Esa Rauman
OPERATORS: OH6KVU
SOAPBOX: Nice contest...
```

Cabrillo-muodon suurin heikkous on, että siinä olevan varsinaisen lokiosuuden muoto muuttuu kilpailussa käytetystä sanomasta riippuen. Näin monen eri kilpailun tukeminen luo omat haasteensa lokin koneistetulle tulkinnalle.

Lokin varsinainen yhteysrivi alkaa aina ”QSO:”-määreellä ja koko loki loppuu ”END-OF-LOG:”-määreeseen.

Esittelen seuraavassa suomalaisissa kisoissa käytetyn NRAU Baltic -muodon ja Skandinavian aktiviteetti -kilpailussa käytettävän SAC-muodon.

Kuvassa 5 on määriteltynä standardin mukainen NRAU Baltic -muotoinen rivi selventävillä apumerkinnöillä varustettuna.

```
                            --------info annettu---- -------info saatu-------
QSO: freq  mo päivämäärä aika kutsu         rst nr# pr kutsu         rst nr# pr
QSO: ***** ** vvvv-kk-pp nnnn ************* nnn nnn aa ************* nnn nnn aa
QSO:  3500 CW 2002-12-29 0530 OH6QU         599 001 PO OH6BG         599 001 PO
000000000111111111122222222223333333333444444444455555555556666666667777777777
123456789012345678901234567890123456789012345678901234567890123456789012345678
```

KUVA 5. NRAU Baltic -formaatti (13)

Kuvassa 5 Freq tarkoittaa taajuutta, mo lähetelajia (CW sähkötyksellä ja PH puheella), rst tarkoittaa kuuluvuusraporttia, nr# yhteyden numeroa ja pr maakuntatunnusta. (26.)

Kuvassa 6 on määriteltynä standardin mukainen NRAU Baltic -muotoinen rivi selventävillä apumerkinnöillä varustettuna.

```
                            --------info annettu---- -------info saatu-------
QSO: freq  mo päivämäärä aika kutsu         rst sanoma kutsu         rst sanoma t
QSO: ***** ** vvvv-kk-pp nnnn ************* nnn ****** ************* nnn ****** *
QSO: 14000 CW 2002-09-21 1048 7S3A          599    001 4K6GF         599    116 0
000000000111111111122222222223333333333444444444455555555556666666666777777777788
123456789012345678901234567890123456789012345678901234567890123456789012345678901
```

KUVA 6. SAC-formaatti (13)

Kuvissa 6 Freq tarkoittaa taajuutta, mo lähetelajia (CW sähkötyksellä ja PH puheella) ja rst tarkoittaa kuuluvuusraporttia. Sanomana on yhteyden numero ja t tarkoittaa käytetyn lähettimen numeroa. (26.)

10 OHJELMISTON SUUNNITTELU JA TOTEUTUS

Projekti toteutettiin ketteriä kehitysmenetelmiä soveltaen. Perinteisessä vesiputousmallissa käytettävä suunnitteluvaihe jätettiin minimiin. Suunnittelua tehtiin ohjelmoinnin yhteydessä sekä selventämään monimutkaisimpia asiakokonaisuuksia. Yksinkertaisimmissa komponenteissa ei siis tehty varsinaista suunnittelua ollenkaan. Tärkeimpinä suunnitteludokumentteina käytettiin liitteinä olevia vaatimusmäärittelyä (liite 1) sekä käyttäjätarinoita (liite 2). Suunnittelussa pyrittiin huomioimaan jonkin verran myös tulevaisuuden tarpeita, esimerkiksi uusien kilpailujen ja erilaisten tiedostomuotojen (varsinkin eri kilpailujen käyttämien cabrillo-muotojen) suhteellisen helppo lisääminen.

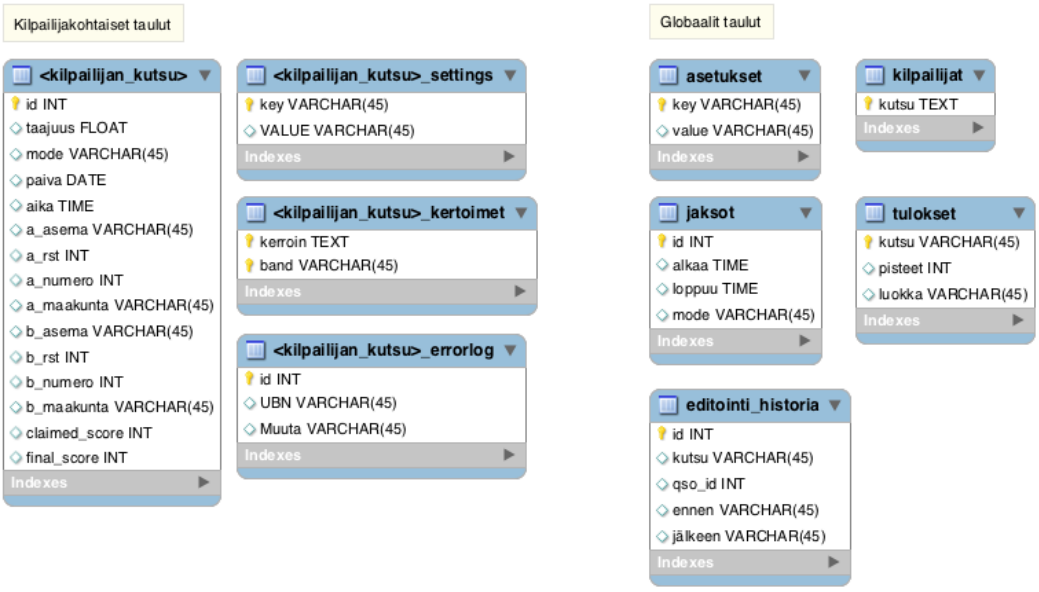
Toteutuksesta sovittiin tilaajan kanssa proof-of-concept-version minimivaatimustaso. Minimivaatimuksiin kuului lähinnä yhden kilpailutyypin (CCF Joulukilpailu) lokien tarkastaminen, pisteiden laskenta ja raportointi sekä kaikki oleelliset näihin toimintoihin liittyvät asiat. Joitain toimintoja toteutettiin ylitse minimivaatimuksien, että saatiin luotua rungot useammille lokityypeille ja kilpailuille.

10.1 Tietokantarakenne

Tietokantarakenteessa päädyttiin ratkaisuun, jossa jokaisella kilpailuun osallistujalla on omat taulunsa yhteyslokille, virhelokille ja muutamille muille kilpailijaa koskeville asioille, kuten kuvasta 7 voi havaita. Lisäksi luotiin muutamia globaaleja tauluja, joihin tallennetaan tietoa mm. tuloksista ja kilpailuun liittyvistä asetuksista. Tähän päädyttiin siksi, että haluttiin säilyttää kilpailijoiden lokit edelleenkin erillisinä, itsenäisinä ja mahdollisimman lähellä alkuperäistä cabrillo-muotoa. Tämä myös helpottaa ristiintarkistuksen iterointia, koska vasta-aseman loki löytyy helposti. Normaalimuotojen käyttöä olisi myös hankaloittanut jonkin verran SQLiten viite-eheyksien valvonnan puutteellisuus.

Kuvan 7 taulun nimeämiskäytännössä <kilpailijan_kutsu> korvataan kilpailijan omalla kutsumerkillä, eli jos kilpailijan kutsu on OH6KVU, korvataan ”<kilpailijan_kutsu>” OH6KVU:lla.

Kuvasta 7 voidaan myös havaita, että taulujen välillä ei ole relaatioita, koska kaikki SQLite-versiot eivät tue viite-eheyksien tarkastusta. Tällöin relaatioiden käyttö on jokseenkin merkityksetöntä. Toisaalta ainut relaatio, jonka olisi voinut luoda, olisi ollut <kilpailijan_kutsu>- ja <kilpailijan_kutsu>_errorlog-taulujen välillä.



KUVA 7. Tietokantarakenne

10.1.1 <kilpailijan_kutsu>-taulu

<kilpailijan_kutsu>-taulu sisältää varsinaisen kilpailijakohtaisen lokiaineiston. A_ -alkuiset sarakkeet tarkoittavat yhteyden a-osapuolta eli kilpailijan lähettämiä tietoja ja b_ -alkuiset taas b-osapuolta eli vasta-asemaa ja siltä saatuja tietoja.

10.1.2 <kilpailijan_kutsu>_settings-taulu

<kilpailijan_kutsu>_settings-tauluun asetetaan joitain kilpailijakohtaisia tietoja avain-arvo-tyyppisesti.

Käytettäviä avaimen arvoja ovat

IMPORTED-FROM
LOG-TYPE
NAME
ADDRESS
OPERATORS
CATEGORY
CREATED-BY
CLAIMED-SCORE
CLAIMED-QSOS
CLAIMED-MULTIP
FINAL-SCORE
CONFIRMED-QSOS
CONFIRMED-MULTIP
CONFIRMED-QSO-POINTS.

”IMPORTED-FROM”-avaimella voi olla vain seuraavia arvoja: ”Cabrillo”, ”PaperLog”, ”UUTest”, ”SynthLog” tai ”Other”. Tällä avaimella siis voidaan kertoa, mistä lokiaineisto on peräisin, ja muodostaa tilastotietoa myöhempään käyttöön eri lokityyppien yleisyydestä.

”LOG-TYPE” ilmaisee, onko loki mukana kilpailussa vai käytetäänkö sitä pelkästään tarkastuslokina. Mahdollisia arvoja tällä avaimella ovat ”ContestLog”, ”CheckLog” tai ”SynthLog”. ContestLog tarkoittaa, että kilpailija on mukana kilpailussa normaalisti. CheckLogilla tarkoitetaan lokia, jonka joku kilpailuun osallistunut on lähettänyt tarkastusta varten, mutta ei halua osallistua varsinaiseen kilpailuun. SynthLog luodaan ohjelmallisesti (synteettisiä lokeja käsitellään tarkemmin luvussa 8.5).

”CATEGORY”-avaimen arvoksi annetaan kilpailijan kilpailuluokka.

”CLAIMED-SCORE” -avaimen arvoksi lasketaan lokin tuonnin jälkeen kilpailijan pistemäärä ilman ristiintarkastusta muiden kilpailijoiden lokien kanssa eli ns. ennakkotulos ja ”FINAL-SCORE” -avaimen arvoksi vastaavasti lopullinen pistemäärä.

Kentät, kuten ”ADDRESS” ja ”OPERATORS”, joiden arvoissa esiintyy useampia rivejä tai kutsuja, erotetaan toisistaan pilkulla.

10.1.3 <kilpailijan_kutsu>_kertoimet-taulu

<kilpailijan_kutsu>_kertoimet-taulu toimii aputauluna helpottamaan kertoimien laskentaa sekä erittelemään, mitkä kertoimet asema on saanut työskenneltyä. Kertoimen nimi ja taajuus toimivat molemmat pääavaimina, jolloin kertoimen on pakko olla uniikki taajuusaluekohtaisesti. Tästä on siis helppo laskea sql-lauseilla rivien määrä, joka on samalla kerrointen määrä. Menetelmä osoittautui varsinaisen ohjelmiston luonnin aikana jokseenkin huonoksi suunnitteluratkaisuksi kilpailussa, jossa on useita jaksoja, koska saman kertoimen voi silloin työskennellä kahteen kertaan. Ongelma kierrettiin laskemalla kerrointen määrä suoraan lokirivien määrästä, joilla pistemäärä on suurempi tai yhtä suuri kuin kertoimesta saatava pistemäärä.

10.1.4 <kilpailijan_kutsu>_errorlog-taulu

<kilpailijan_kutsu>_errorlog-tauluun tallennetaan ns. UBN- raportti (Unique/Bad/Not in log), jossa määritellään syykoodi, miksi yhteydestä on vähennetty pisteitä tai se on hylätty. UBN-kentän mahdollisia arvoja ovat siis U, B tai N. U tarkoittaa uniikkia kutsua, jota ei löydy mistään muusta lokista, B tarkoittaa virhettä kutsussa tai sanomassa ja N tarkoittaa, että yhteyttä ei ole vasta-aseman lokissa tai vasta-aseman lokia ei ole olemassa. (28.)

Muuta -kenttään tallennetaan tarkempi tieto siitä, mikä kyseisessä yhteydessä oli vikana.

Errorlog-taulun id:ksi asetetaan sama numero kuin varsinaisen lokitaulun rivin id, jolloin virheet on helppo kohdistaa oikeaan yhteyteen.

10.1.5 Asetukset-taulu

Asetustaulu on globaali taulu, johon tallennetaan kyseiseen kilpailuun liittyviä asetuksia. Taulu on avain-arvo-tyyppinen ja mahdollisia avaimen arvoja sille ovat kilpailun_tyyppi, nimi, päivämäärä ja aika. Kilpailun tyypillä tarkoitetaan kilpailussa käytettäviä sääntöjä. Mahdolliset arvot ovat Kesakisa, Kinkkukisa, Sainio ja Syysottelu. Jos ohjelmisto alkaa myöhemmin tukea muita kilpailutyyppejä, on se huomioitava myös tässä määrityksessä.

10.1.6 Jaksot-taulu

Jaksot-tauluun määritellään kilpailussa käytettävät jaksot, tai jos jaksoja ei käytetä, määritellään koko kilpailuaika yhdeksi jaksoksi. Tietoa käytetään, kun tutkitaan, onko yhteydet pidetty säädettyjen aikarajojen puitteissa. Jos kilpailussa on useampia jaksoja, se vaikuttaa myös pistelaskentaan, koska aseman kanssa voi pitää yhteyden vain kerran joka jaksolla ja molemmilla taajuusalueilla. Jos jaksoja on kaksi, voidaan vasta-aseman kanssa pitää maksimissaan neljä yhteyttä, kaksi molemmilla taajuusalueilla.

10.1.7 Editointi_historia-taulu

Editointihistoriaan tallennetaan käsin lokiin tehtyjen muutosten historia. Historiasta voidaan nähdä, mitä on muokattu ja mitä kyseisen yhteysrivin sisältö on ollut ennen ja jälkeen muokkauksen. Koska luodussa proof-of-concept-versiossa ei ole vielä olemassa lokin editointimahdollisuutta, editointihistoria -taulu on tehty lähinnä tulevaisuuden tarpeisiin.

10.1.8 Kilpailijat

Kilpailijat-taulu toimii ohjelman aputauluna. Siihen tallennetaan kaikkien kilpailijoiden kutsumerkit, jolloin on helpompi käsitellä dynaamisesti nimettyjä tauluja. Tästä taulusta haetaan myös kilpailijoiden kutsumerkit käyttöliittymän kilpailijalistaan, mukaan lukien tarkastuslokit ja synteettiset lokit.

10.1.9 Tulokset

Tulokset-tauluun tallennetaan kilpailun lopputulokset kilpailuluokittain. Varsinaisessa ohjelmiston luonnissa tästä ajatuksesta kuitenkin luovuttiin. Tulokset on helppo hakea kaikkien kilpailijoiden omista settings-tauluista, joissa on määriteltynä tulokset ja kilpailijan luokka. Tämä taulu kuitenkin säilytetään, jos sille tulee tulevaisuudessa käyttöä.

10.2 Erilaisten tiedostomuotojen tuki

Suunnittelussa huomioitiin, että sovelluksen tulee pystyä lukemaan sekä NRAU Baltic -muotoisia cabrillo-tiedostoja että SAC-tiedostoja, joita tarvitaan tulevaisuudessa.

Liitteessä 1 olevassa vaatimusmäärittelyssä on mainittuna mahdollinen tuki UUTest- ja Fintest-ohjelmistojen tuottamille tiedosto muodoille (FA1-4 ja FA1-5). Myös se päätettiin toteuttaa, koska näitä lokimuotoja käyttäviä kilpailijoita oli melko paljon ohjelmiston testaamiseen käytetyssä lokiaineistossa.

Alun perin suunniteltiin CSV-muotoisten tiedostojen tukemista, excel-lokien tuonnin helpottamiseksi, mutta tämä osoittautui käytännössä mahdottomaksi ja erittäin työlääksi, koska kaikki, excel-lokin lähettäneet, olivat täyttäneet taulukon eri tavalla. Näin ollen tästä vaatimuksesta päätettiin luopua kokonaan.

10.3 Käyttöliittymä

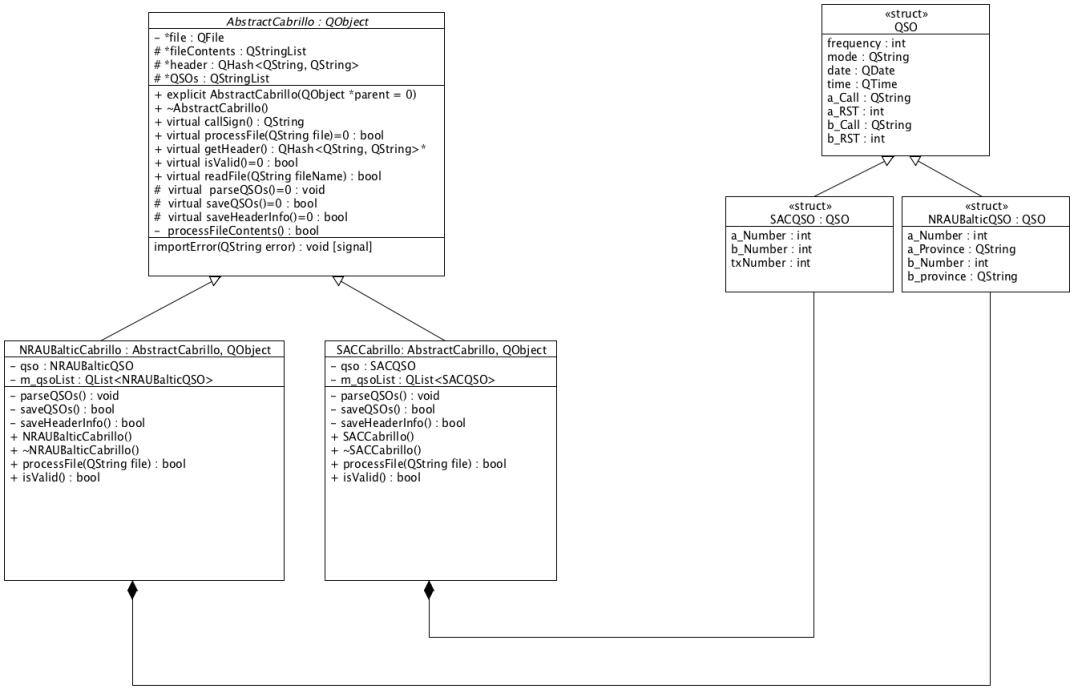
Käyttöliittymässä pyrittiin mahdollisimman selkeään ja hyvään käyttäjäkokemukseen. Tämän ansiosta käyttäjä pystyy intuitiivisesti käyttämään ohjelmistoa ilman, että hänen tarvitsee lukea käyttöohjeita.

Kuvassa 8 on käyttöliittymän pääikkuna. Kuvan tilastotiedoissa joillakin riveillä lukee TextLabel. Kyseiset statistiikat toteutetaan myöhemmissä versioissa, koska ne eivät olleet erityisen oleellisia proof-of-concept-version toiminnan kannalta.

KUVA 8. Käyttöliittymän pääikkuna

10.4 Luokkarakenne

Tässä työssä käytettiin abstrakteja luokkia ja rajapintoja helpottamaan tulevaisuudessa lisättävien uusien kilpailusääntöjen ja lokimuotojen implementointia ja vähentämään saman koodin useaan kertaan kirjoittamista. Esimerkkinä voidaan mainita muun muassa cabrillo-lokiformaatin eri muotojen implementointi, jossa abstrakti yläluokka toteuttaa yhteiset toiminnot ja pakottaa alaluokkia toteuttamaan niille spesifiset toiminnot (kuten esimerkiksi lokin muotoilun validiuden tarkastus). Tietokantaluokka jouduttiin myös toteuttamaan samantapaisesti, juuri cabrillon eri muotojen takia. Kuvasta 9 nähdään cabrillo-luokkien välinen perintähierarkia. Tämä laajennettavuus tuo myös mukanaan ongelman, koska käytettävä luokka on valittava ajonaikaisesti. Tämä ongelma oli helpoiten ratkaistavissa käyttämällä abstrakti tehdas -suunnittelumallia.



KUVA 9. Cabrillo-luokkien rakenne

Työssä käytetään myös soveltaen ns. model-view-controller-arkkitehtuuria, jossa käyttöliittymä (view) erotetaan toimintalogiikasta (controller) ja tiedon käsittelystä (model) (29). Tyypillisesti käyttöliittymän ikkuna siis koostuu kolmesta luokasta. Ensimmäinen liittyy widgettien näyttämiseen ja ulkoasuun, toinen varsinaiseen toiminnallisuuteen ja kolmas tiedon tallentamiseen ja hakemiseen. Tämän sovelluksen tapauksessa model-luokalla käsitellään pääasiassa tietokantaa.

Seuraavassa esitellään lyhyesti ohjelmassa käytetyt luokat ja niiden pääasialliset tarkoitukset. Käyttöliittymä (view)-luokat jätetään esittelemättä, koska ne on generoitu automaattisesti Qt:n Designer-työkalulla.

10.4.1 AbstractCabrillo-luokka

AbstractCabrillo-luokka toimii abstraktina yläluokkana cabrillo-luokille ja toteuttaa niille yhteiset funktiot, jotka liittyvät muun muassa tiedoston lukemiseen ja alustavaan otsikko- ja yhteysrivien erottamiseen.

10.4.2 AbstractCabrilloFactory- ja CabrilloFactory-luokat

AbstractCabrilloFactory -luokka toimii abstraktin tehtaan toteuttavana abstraktina yläluokkana (rajapintana). Tällä rajapinnalla on tarkoitus luoda cabrillo -luokkien olioita. CabrilloFactory taas toteuttaa edellä mainitun rajapinnan ja varsinaisen olioiden luonnin. Nämä luokat yhdessä siis toteuttavat abstrakti tehdas -mallin mukaisen rakenteen.

10.4.3 AbstractContest-luokka

AbstractContest-luokka toimii kilpailuluokkien abstraktina yläluokkana. Se määrittelee rajapinnan kilpailuluokkien toteuttamiselle ja toteuttaa kilpailuluokille yhteisiä funktioita.

10.4.4 AbstractContestFactory- ja ContestFactory-luokat

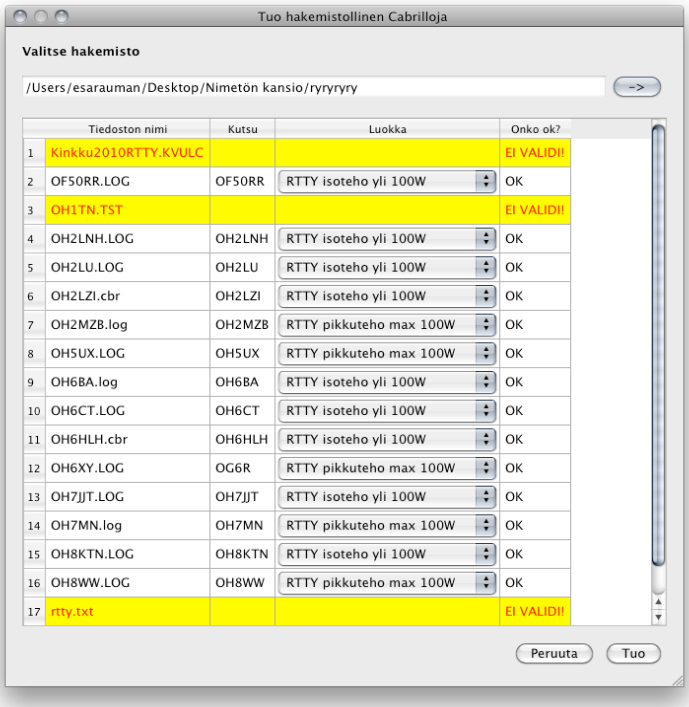
AbstractContestFactory ja ContestFactory toteuttavat yhdessä abstrakti tehdas -mallin mukaisen rakenteen, jolla luodaan kilpailuluokkien olioita (kesakisa, kinkkukisa, sainio ja syysottelu).

10.4.5 ASCIIReport-luokka

ASCIIReport-luokalla luodaan ascii-muotoinen raportti kilpailun lopputuloksista, jossa tulokset on lajiteltu kilpailuluokittain.

10.4.6 CabrilloDirectoryImport-luokka

CabrilloDirectoryImport-luokalla luodaan toiminnallisuus, jolla voidaan tuoda tietokantaan hakemistollinen cabrillo-tiedostoja. Kuvassa 10 nähdään toimintoon liittyvä käyttöliittymä. Kuvasta voi havaita, että jokaisen lokitiedoston validius tarkistetaan, jolloin epäkelpoa dataa ei voi päätyä tietokantaan.



KUVA 10. Tuo hakemistollinen cabrilloja -dialogi

10.4.7 DB-luokka

DB-luokka toimii model-luokkana, jonka pääasiallinen tehtävä on tietokanta-toimintojen toteuttaminen. DB toimii myös yläluokkana DB_NRAUBaltic- ja DB_SAC-luokille.

DB-luokan määrittelytiedosto db.h sisältää myös kaksi struct-tyyppistä tietorakennetta (Jakso ja ReportRow). Jakso-tietorakennetta käytetään välittämään yhden jakson tiedot kilpailuluokalle ja ReportRow sisältää puolestaan tulosraportoinnissa käytetyn rakenteen, johon on tallennettu yksi tulosrivi. Varsinaisesti nämä sisällytetään erilaisiin kontainereihin, jolloin kyseinen kontaineri sisältää kaikki kyseisen tietorakenteen ilmentymät.

10.4.8 DB_NRAUBaltic- ja DB_SAC-luokat

Koska lokien muoto on erilainen NRAU Baltic- ja SAC-muotoa käyttävissä kilpailuissa, luokat DB_NRAUBaltic ja DB_SAC on luotu suorittamaan tehtäviä, jotka ovat erikoistuneet näille lokimuodoille. Näiden luokkien pääasiallisia

tehtäviä ovat kilpailijoiden ja synteettisten lokitaulujen luonti tietokantaan, yhteysrivien vienti kantaan sekä lokien tuonti kannasta muiden luokkien (olioiden) käsiteltäväksi.

10.4.9 DBFactory-luokka

DBFactory-luokka toteuttaa yksinkertaistetun muodon abstrakti tehdas -mallista. DBFactoryllä voidaan luoda tietokantaluokkien olioita.

10.4.10 FinTest-luokka

FinTest-luokka toimii vanhalla dos-pohjaisella FinTest -ohjelmalla luotujen lokien tuonnissa model-luokkana, jonka pääasiallinen tehtävä on parseroida kyseisen muotoinen loki ja viedä sen sisältämät yhteydet tietokantaan. Tämä toiminnallisuus on tehty helpottamaan lokien tarkastajan taakkaa, koska tämänmuotoisia lokitiedostoja käytetään vielä jonkin verran. Käytössä on kuitenkin noudatettava varovaisuutta, koska lokitiedoston rakenteen oikeellisuutta ei voi varmistaa mitenkään.

10.4.11 HTMLReport-luokka

HTMLreport on ASCIIReport-luokan alaluokka, jossa luodaan samansisältöinen raportti kuin ASCIIReportissa, mutta tallennusmuotona on html. Html-raportti tallennetaan hakemistoon, ja siinä käytetään lisänä Mainoshuone Forcen luomia css-tyylitiedostoja sekä kuvia luomaan siistimpi ilme raportille. Css-tiedostot myös mahdollistavat sen, että varsinainen html-tiedosto sisältää lähinnä datat, ei muotoiluja.

10.4.12 ImportFintest-luokka

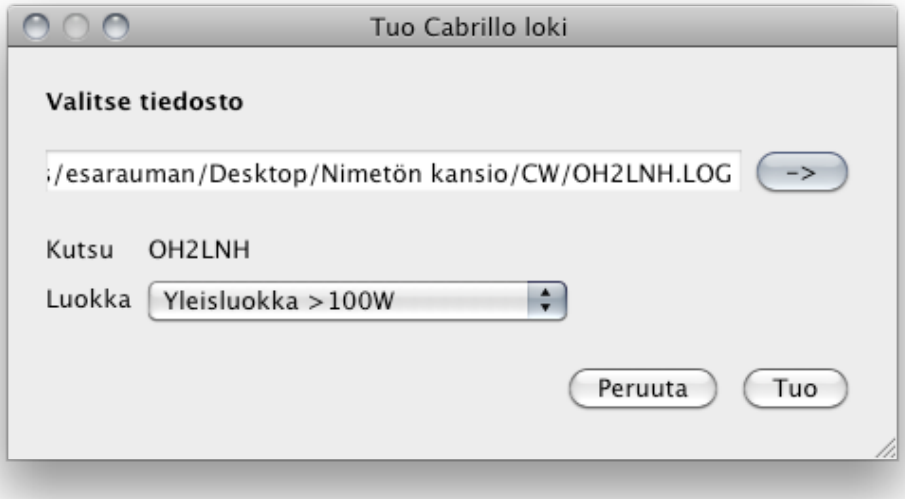
ImportFintest-luokka toteuttaa FinTest-ohjelmalla luotujen lokien tuonnin käyttöliittymän toiminnallisuuden, eli se on ns. controller-luokka.

Kuvassa 11 on esitettynä toimintoon liittyvä käyttöliittymä. Koska FinTest-ohjelman tiedostomuodosta ei selviä mistään a-osapuolen kutsua eikä lähetelajia, on ne määriteltävä käyttöliittymään.

KUVA 11. FinTest-lokin tuonnin käyttöliittymädialogi

10.4.13 ImportSingleCabrillo-luokka

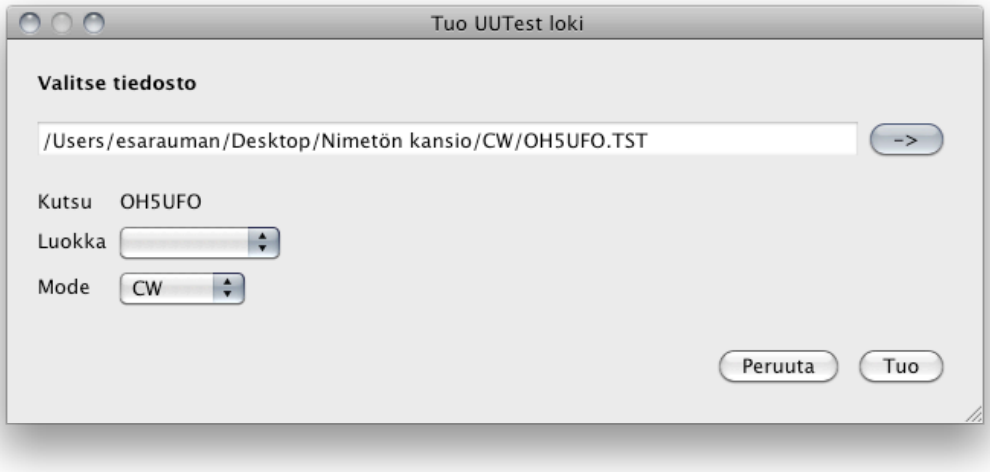
ImportSingleCabrillo toteuttaa yksittäisen cabrillo-lokin tuontitoiminnallisuuden. Luokka toimii controller-luokkana varsinaiselle käyttöjärjestelmäluokalle. Kuvassa 12 on tähän luokkaan liittyvä käyttöliittymä.



KUVA 12. Yksittäisen cabrillo-tiedoston tuontidialogi

10.4.14 ImportUUTest-luokka

ImportUUTest-luokka toteuttaa käyttöliittymän controller-luokan, jolla voidaan tuoda yksittäisiä UUTest-ohjelmalla luotuja lokitiedostoja. Esimerkki käyttöliittymästä on kuvassa 13.



KUVA 13. UUTest-lokin tuontidialogi

10.4.15 Kesakisa-, Kinkkukisa-, Sainio- ja Syysottelu-luokat

Kesakisa-, Kinkkukisa-, Sainio- ja Syysottelu-luokat toimivat AbstractContest-luokan aliluokkina ja toteuttavat varsinaiset lokien tarkastuksiin liittyvät toimenpiteet. Niissä määritellään myös kilpailusäännöistä saatavat kilpailuluokat ja yhteyksistä saatavat pisteet.

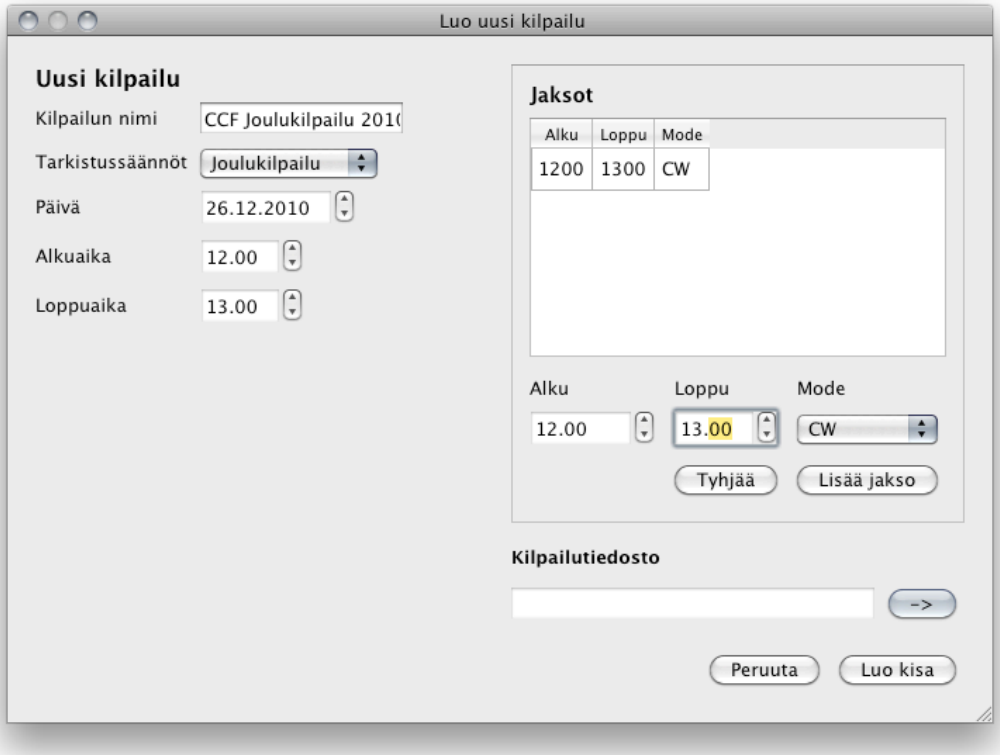
Näistä luokista toteutetussa versiossa luotiin lähinnä kinkkukisa-luokka, koska tavoitteena oli CCF Joulukilpailun lokien tarkastaminen. Muista luokista luotiin rungot, joihin toiminnallisuus on myöhemmin helppo lisätä.

10.4.16 KVULogChecker-luokka

KVULogChecker-luokka toimii ohjelmiston pääikkunan controller-luokkana. Tämän luokan avulla suoritetaan kaikki toiminnallisuus, joka liittyy pääikkunaan. Pääikkuna on nähtävillä kuvassa 8.

10.4.17 NewContest-luokka

NewContest-luokan avulla luodaan uusi kilpailu. Luokka toimii controller-luokkana varsinaiselle käyttöliittymäluokalle ja toteuttaa tarvittavat toiminnot. Kuvassa 14 nähdään, millainen kyseinen lomake on. Vaikka kilpailussa ei olisikaan erikseen määriteltyjä jaksoja, käyttäjän on silti luotava vähintään yksi jakso koko kilpailun ajaksi.



KUVA 14. Uuden kilpailun luontidialogi

10.4.18 NRAUBalticCabrillo- ja SACCabrillo-luokat

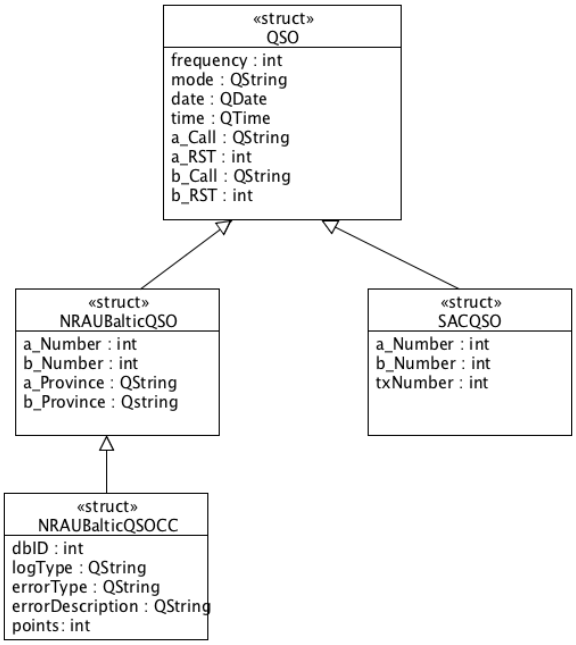
NRAUBalticCabrillo- ja SACCabrillo-luokat periytyvät AbstractCabrillo-luokasta ja toteuttavat lokityypin mukaisen varsinaisen lokin käsittelyn ja tallentamisen tietokantaan. SACCabrillo-luokan toiminnallisuutta ei ole täysin toteutettu, vaan se toteutetaan tulevaisuudessa, kun SAC-kilpailun tuki varsinaisesti lisätään ohjelmaan.

10.4.19 Qso.h

Qso.h ei ole varsinaisesti luokka, mutta sisältää muutamia tärkeitä ohjelmassa käytettyjä tietorakenteita, kuten rakenteet erilaisten lokityyppien yhteysriveille. Tietorakenteissa on käytetty perintää, jolloin vältytään saman asian moneen kertaan kirjoittamiselta.

Kuvassa 15 esitetään tietorakenteiden hierarkia, joka jakaantuu kahteen eri lokityyppiin (NRAUBaltic ja SAC). NRAUBalticQSOCC-tietorakennetta käytetään lokien ristiintarkistuksen yhteydessä, kun tallennetaan väliaikaisesti kyseisen yhteyden tietokanta id, lokin tyyppi, virheen tyyppi (U, B tai N), virheen tarkempi kuvaus sekä yhteydestä annettavat pisteet.

Tässä on jo valmistauduttu myöhempää SAC-kilpailun tukea varten luomalla siinä käytettyä lokityyppiä vastaava struktuuri. Tämä luultavasti tarvitsee myös alleen vielä NRAUBalticQSOCC-tapaisen alastruktuurin pisteiden ja muun sellaisen väliaikaista tallentamista varten.



KUVA 15. QSO-tietorakenteet

10.4.20 UUTestImport-luokka

UUTestImport-luokalla toteutetaan varsinainen uutest-ohjelmalla luotujen lokitiedostojen käsittely ja tietojen tallennus tietokantaan.

10.5 Tiedoston tuonti tietokantaan

Cabrillo-tiedosto luetaan ensin muistiin riveittäin, jonka jälkeen saatu QStringList jaetaan erillisiin otsikko- ja yhteysosiin. Seuraavaksi tarkistetaan lokin oikeellisuus. Tämä tapahtuu vertaamalla ensimmäistä lokiriviä cabrillo-muodon spesifikaatioihin (kuten kenttien lukumäärä, kenttien sisältö ja päivämäärä). Jos lokin muotoilussa on virhe, lopetetaan lokin käsittely ja palautetaan virheilmoitus. Validoinnin jälkeen kilpailijalle luodaan tietokantaan omat lokitaulut sekä lisätään hänet tietokannan kilpailijat-tauluun. Tämän jälkeen käsitellään varsinainen yhteysdata ja tallennetaan se tietokantaan. Lopuksi vielä otsikkodata muotoillaan ja tallennetaan tietokantaan.

FinTest- ja UUTest-lokit käsitellään suurin piirtein samalla periaatteella, mutta niissä ei ole varsinaista otsikkodataa. Näiden lokimuotojen käytössä on oltava varovainen ja tarkistettava käsin, että muodot vastaavat niiden formaattien spesifikaatioita, koska varsinaista validointia näille ei tehdä.

10.6 Lokien tarkastus

Lokien tarkastus tapahtuu varsinaisissa kilpailuluokissa (kinkkukisa, sainio, kesakisa, syysottelu). Tähän ”proof-of-concept”-versioon luotiin CCF Joulukilpailun sääntöjä vastaava kilpailuluokka (kinkkukisa), jonka tehtävänä on luoda synteettiset lokit ja laskea ennakkotulos. Ennakkotulos on pistemäärä ennen lokien ristiintarkastusta, samalla myös kyseisen lokin maksimipistemäärä. Ennakkotuloksen laskemisen jälkeen tehdään varsinainen ristiintarkastus ja kertoimien uudelleenlaskenta, koska hylätyistä yhteyksistä ei voi saada pisteitä (vrt. ennakkotulos, jossa kaikki yhteydet saavat pisteitä). Lopuksi tallennetaan vielä pistemäärät settings-tauluun, josta ne ovat haettavissa raportoinnissa.

10.6.1 Ennakkotulosten laskenta

Ennakkotulokset lasketaan käymällä lokin kaikki rivit läpi jaksoittain ja merkitsemällä lokiriville pistemäärä, jonka siitä voi maksimissaan saada.

10.6.2 Synteettiset lokit

Moniin kilpailusääntöihin oli alun perin kirjattu, että yhteydestä aseman kanssa, joka ei ollut palauttanut lokiaan, mutta joka ilmeni vähintään tietyssä määrässä lokeja, annetaan puolet normaaleista yhteyspisteistä. Sääntö aiheutti tarpeen generoida lokit ohjelmallisesti, että yhteydet voidaan tarkistaa ja pisteyttää normaalisti.

Synteettiset lokit ja niiden tietokantarakenne eivät eroa mitenkään muuten normaalista, kuin siten, että niiden settings-taulun asetuksiin on määritelty SynthLog. Synteettisiä lokeja ei myöskään sisällytetä tuloslaskelmiin eikä niihin lasketa pisteitä.

Nämä lokit luodaan taulukoimalla kaikkien yhteyksien b-osapuolet, jotka esiintyvät riittävän monta kertaa, mutta eivät kuitenkaan ole kilpailijattaulussa. Tässä vaiheessa merkitään myös ns. uniikit kutsut eli radioamatööritunnukset, jotka esiintyvät b-osapuolina vain kerran.

Taulukoinnin jälkeen luodaan varsinaiset lokirivit kääntämällä a- ja b-osapuolten datojen paikkaa (eli a:sta tule b-osapuoli ja päinvastoin), ja näin saatu loki viedään kantaan.

10.6.3 Lokien ristiintarkastus

Ristiintarkastus aloitetaan jakamalla lokit kilpailun luonnissa määriteltyjen jaksojen mukaisiin osalokeihin, jonka jälkeen ne iteroidaan yksitellen läpi ja tehdään varsinainen tarkastus. Synteettiset ja tarkastuslokit jätetään tarkistamatta, koska niillä ei osallistuta varsinaiseen kilpailuun.

Ristiintarkistuksessa verrataan a- ja b-osapuolten lokeista samaa yhteyttä toisiinsa. Näin voidaan havaita, ovatko yhteydessä vaihdetut sanomat siirtyneet oikein molemmille osapuolille.

Yhteydessä voi olla virhe kellonajoissa, kutsussa tai sanomassa. Kellonajalla varmistetaan, että kyseessä on varmasti oikea yhteys, ja siitä myös huomataan jos kilpailijan lokissa on systemaattinen virhe ajassa. Aikaikkunaksi CCF Joulukilpailussa on määritelty ± 15min. Jos yhteydestä löytyy yksi tai useampi virhe, pisteytetään yhteys sen mukaisesti ja merkitään tietokantaan kilpailijan errorlog-tauluun virheen tyyppi, tarkempi kuvaus ja yhteysrivin id-numero.

Ristiintarkistuksen jälkeen on myös tarpeellista tarkistaa kilpailijalle annettavat kertoimet, jotka jakaantuvat taajuusalueen ja jakson mukaan. Kerroinpisteet lisätään tietokantaan riveittäin ja kertoimen antavat maakuntatunnukset lisätään kertoimet-tauluun.

Hylätyistä (virhekoodi U tai N) yhteyksistä ei luonnollisestikaan voida antaa kerroinpisteitä. Poikkeuksen muodostaa virhekoodi B, jonka mukaan yhteyden sanomassa on virhe. Tässä tapauksessa kilpailijan on kuitenkin saatava täydet kerroinpisteet. Virhekoodit on selvennetty tarkemmin luvussa 9.1.4.

11 SÄÄNTÖMUUTOSEHDOTUKSET

Yhtenä suurena esteenä täysin automatisoidulle lokien tarkistukselle on se, että kilpailujen säännöt sallivat useiden erilaisten lokityyppien käytön. Esimerkkinä voisi mainita viitosten syysottelun sääntöjen kohdan ”Lokiformaatti ja lokien tarkastus”: ”Mikä tahansa yleisesti käytetty ASCII-kilpailuformaatti, jossa on yksi yhteys yhdellä rivillä ja kaikki erilliset yhteystiedot, alkavat allekkain samasta kohtaa riviä --” (30). Tämäntyyppiset säännöt aiheuttavat ongelman, koska tällaista geneeristä ASCII-tiedostoa ei ole mitenkään standardoitu, ja näin ollen sitä on mahdoton käsitellä automatisoidusti. Myös paperilokit aiheuttavat samankaltaisen ongelman, koska ne on käsin syötettävä järjestelmään, jotta tuloslaskenta toimisi oikein. Ehdotankin sääntöihin muutoksen, että vain oikein muotoiltu NRAUBaltic Cabrillo -tiedosto olisi hyväksytty tiedostomuoto, jossa kilpailija voi palautta yhteyslokinsa kilpailun järjestäjälle. Cabrillo-tiedostojen käytölle ei pitäisi olla mitään esteitä, koska suurin osa kilpailuissa käytetyistä lokiohjelmista tuottaa oikein muotoiltuja cabrillo -tiedostoja. Lisäksi on olemassa ns. offline cabrillo -editoreita, jolla voidaan tuottaa kyseisenlaisesti muotoiltu tiedosto. Esitänkin siis muutamia muutoksia kilpailuiden sääntöihin, joiden avulla automaattinen lokien tarkastus mahdollistuisi:

1. Sallitaan vain NRAU Baltic -muotoiltujen cabrillo-lokien käyttö.

2. Säännöissä ei ole määriteltynä aikaikkunaa eli sitä, paljonko eri osapuolten kellot voivat olla väärässä. Aikaikkuna tulisi määritellä ohjelmistoon toteutetun ± 15 min mukaiseksi.

3. Sääntöihin tulee määritellä selkeämmin, että kilpailija vastaa itse toimittamansa lokin muotoilun ja sisällön oikeellisuudesta.

Kohdassa 1 esittämäni muutos on herättänyt melkoisesti keskustelua harrastepiireissä puolesta ja vastaan. Monella on vielä käytössä vanhentuneet, jo aikansa eläneet kilpailulokiohjelmat, joilla ei voi tuottaa cabrillo-muotoisia lokitiedostoja. Joidenkin mielestä tämä ehdotus myös vähentäisi kilpailuun

osallistuvien amatöörien määrää. Uskon kuitenkin, että kyseessä on lähinnä normaali muutosvastarinta, joka häviää, kun kilpailijat ja varsinkin kilpailuiden järjestäjät huomaavat uuden järjestelmän mukanaan tuomat edut perinteiseen, käsityönä tehtyyn lokien tarkastukseen verrattuna. Kilpailijoiden ei tarvitse välttämättä edes luopua vanhoista tutuista ohjelmistaan, jos he käyttävät aiemmin mainittuja offline cabrillo editor -ohjelmia.

Muutos on oikeastaan pakko tehdä, jos tulevaisuudessa aiotaan siirtyä käyttämään tulospalveluportaalia, kuten tällä hetkellä suunnitelmissa on. Tämä portaali mahdollistaisi ainoastaan lokien palauttamisen cabrillo-muodossa (tai käsin syötettynä www-lomakkeella).

Kohdan 2 muutos tulee määritellä lähinnä asian virallistamiseksi ja selkeyttämiseksksi.

Testiaineistona käytetyssä lokimateriaalissa ilmeni suuria virheitä lokitiedostojen muotoiluissa. Monet kilpailijat olivat muokanneet lokitiedostojaan siten, että niitä ei voitu enää koneella lukea, vaan kilpailun järjestäjän oli korjattava käsin tiedostot oikeanlaisiksi. Perinteisesti kilpailun tuomarit ovat korjanneet virheet ns. ”ham spiritin” nimissä, jonka voisi määritellä radioamatöörien väliseksi veljeydeksi, yhteistyöksi, hyväksi hengeksi ja avunannoksi. En ole missään muussa kilpailulajissa nähnyt, että tuomarit korjaisivat kilpailijoiden virheitä, joten tässä asiassa pitäisi mielestäni vastuu säilyttää kilpailijalla itsellään, eli virheellinen loki olisi joko hylättävä tai palautettava kilpailijan korjattavaksi (viitaten muutosehdotukseen numero 3).

Näiden muutosten jälkeen päästään vasta oikeasti kehittämään kilpailujen tulospalvelua lähemmäs nykyaikaista tasoa, jossa ennakkotulokset ovat saatavilla heti lokien palautuksen jälkeen ja lopulliset tulokset muutaman päivän kuluttua lokien palautusajan päätyttyä. Muutoksista saattaa myös saada kuvan, että haluaisin hankaloittaa kilpailuihin osallistumista, mutta tarkoituksena on kuitenkin helpottaa lokien tarkastajien työtä ja siten nopeuttaa tulosten julkaisemista. Kilpailijankin kannaltakin on huomattavasti parempi, että nämä asiat ovat selkeästi ja yksiselitteisesti säännöissä määriteltyjä.

12 YHTEENVETO JA JÄRJESTELMÄN TULEVAISUUDENNÄKYMÄT

Työn tarkoituksena oli luoda uusi ohjelmisto, jolla voidaan suorittaa radiokilpailun tulospalvelussa tarvittavat toimenpiteet automatisoidusti ja samalla helpottaa ja nopeuttaa kilpailunjärjestäjän työtaakkaa. Kyseessä oli suomalaisessa radiokilpailumaailmassa uraauurtava työ, ja sen vaikutukset näkyvät vielä pitkälti tulevaisuudessa.

Suurimmat suunnitteluun ja toteutukseen liittyvät haasteet liittyivät ohjelmiston tulevaisuudennäkymiin eli siihen, miten luodaan riittävät abstraktiot, joilla saavutetaan ohjelmiston helppo laajentaminen (esimerkiksi uusien kilpailujen tuen luominen). Toinen suuri, lähinnä suunnittelullinen, ongelma oli varsinainen lokien tarkastus, jonka toteuttamiseen olisi ollut monta erilaista vaihtoehtoa.

Koska kyseessä on vasta ohjelmiston ensimmäinen versio, jolla tutkitaan, onko kyseinen lokien automaattinen tarkastus mahdollista ja mitä ongelmia se tuo mukanaan, kehitystä jatketaan saatujen käyttäjäkokemusten ja toiveiden perusteella. Ohjelmaan lisätään tukea useille erilaisille kilpailuille sekä toteutetaan määritellyt toiminnot, joita ei tähän versioon vielä toteutettu.

Tulevaisuudessa järjestelmää täydennetään web-portaalilla, johon kilpailun osallistujat voivat palauttaa omat lokinsa. Paperilokien tai muiden yhteensopimattomien lokimuotojen käyttäjät voivat puolestaan syöttää omat yhteytensä helppokäyttöisellä lomakkeella, joka muodostaa oikein muotoillun cabrillo-tiedoston. Portaalilla saadaan siten estettyä suurin osa ongelmista, jotka johtuvat virheellisesti muotoilluista lokitiedostoista. On myös mahdollista, että koko sovellus kirjoitetaan uudelleen internet-sovellukseksi, jolloin on mahdollista nähdä tulosten kehittyminen välittömästi, kun kilpailijat lähettävät lokejaan järjestelmään. Tämän toteuttaminen ja täydellinen hyödyntäminen vaatii edellisessä luvussa ehdottamieni sääntömuutosten tekemistä.

LÄHTEET

1. CCF in brief. 2010. Saatavissa: http://contestclubfinland.com/CCF/index.php?option=com_content&task=view&id=20&Itemid=2. Hakupäivä 5.11.2010.
2. Ketterä ohjelmistokehitys. 2011. Wikipedia. Saatavissa: http://fi.wikipedia.org/wiki/Ketter%C3%A4_ohjelmistokehitys. Hakuväivä 26.2.2011.
3. Highsmith, J. 2001. History: The Agile Manifesto. Saatavissa: http://agilemanifesto.org/history.html. Hakuväivä 26.2.2011.
4. Beck, K. - Beedle, M. - Bennekum, Av. - Cockburn, A. - Cunningham, W. - Fowler, M. - Grenning, J. - Highsmith, J. - Hunt, A. - Jeffries, R. - et al. 2001. Manifesto for Agile Software Development. Saatavissa: http://agilemanifesto.org/. Hakupäivä 26.2.2011.
5. Scrum. 2011. Wikipedia. Saatavissa: http://fi.wikipedia.org/wiki/Scrum. Hakupäivä 26.2.2011.
6. Extreme Programming. 2011. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Extreme_Programming. Hakupäivä 26.2.2011.
7. Lindberg, H. 2003. Extreme Programming. Tampereen yliopisto. Pro Gradu -työ.
8. Qt (framework). 2010. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Qt_(framework). Hakupäivä 16.10.2010.
9. What is Qt. 2010. Nokia. Saatavissa: http://qt.nokia.com/?currentflipperobject=04fb871fbc28d8e8e46c51fe89ef8d67. Hakupäivä 16.10.2010.
10. Signals & Slots. 2010. Nokia. Saatavissa: http://doc.qt.nokia.com/4.7/signalsandslots.html. Hakupäivä 16.10.2010.
11. Qt Whitepaper. Nokia. Saatavissa: http://developer.qt.nokia.com/wiki/QtWhitepaper. Hakupäivä 16.10.2010.

12. Widgets tutorial. 2010. Nokia. Saatavissa: http://doc.qt.nokia.com/4.7/widgets-tutorial.html. Hakupäivä 5.11.2010.
13. Macintosh Style Widget Gallery. Nokia. Saatavissa: http://doc.trolltech.com/4.7/gallery-macintosh.html. Hakupäivä 5.11.2010.
14. Container Classes. Nokia. Saatavissa: http://doc.qt.nokia.com/latest/containers.html. Hakupäivä 26.2.2011.
15. QMap class reference. Nokia. Saatavissa: http://doc.qt.nokia.com/latest/qmap.html. Hakupäivä 26.2.2011.
16. Operators in C and C++. 2011. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B. Hakupäivä 27.2.2011.
17. Features of SQLite. Saatavissa: http://www.sqlite.org/features.html. Hakupäivä 13.10.2010.
18. SQLite Foreign Key support. Saatavissa: http://www.hwaci.com/sw/sqlite/foreignkeys.html. Hakupäivä 5.11.2010.
19. SQL Features That SQLite Does Not Implement. Saatavissa: http://www.sqlite.org/omitted.html. Hakupäivä 5.11.2011.
20. Kainulainen, H. 2002. Moniperintä vastaan rajapinnat. Saatavissa: http://www.mit.jyu.fi/opetus/opinnayte/LuK/MoniperintaVastaanRajapinnat/. Hakupäivä 5.11.2010.
21. Stroustrupp, B. 1997. The C++ programming language 3rd edition. Massachusetts: Addison-Wesley.
22. Goyvaerts, J. - Levithan, S. 2009. Regular Expressions cookbook. 1. painos. Sebastobol (CA): O'Reilly Media
23. Regular Expression. 2011. Wikipedia. Saatavissa: http://en.wikipedia.org/wiki/Regular_expression. Hakupäivä 26.2.2011
24. Nimimerkki ”Nitin”. 2010. Abstract Factory Pattern. Saatavissa: http://xeon2k.wordpress.com/2010/11/28/abstract-factory-pattern/. Hakupäivä 25.2.2011
25. Gamma, E. - Helm, R. - Johnson, R. - Vlissides, J. Design Patterns CD. CD-ROM. Addison-Wesley.

26. Perkiömäki, J. 2008. Cabrillo - kontestilokin rakennuskieli. Saatavissa: http://www.oh6aa.com/tr/cabrillo.html. Hakupäivä 5.11.2010.
27. Garlough, R. 2000. Cabrillo V2.0 FAQ. Saatavissa: http://www.kkn.net/~trey/cabrillo/faq.txt. Hakupäivä 5.11.2010.
28. CQ WW Contest Committee. 2003. CQ World Wide DX Contest. Saatavissa: http://www.cqww.com/cqwwubn.htm. Hakupäivä 17.11.2010.
29. Reenskaug, T. 1979. Models - Views - Controllers. Saatavissa: http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf. Hakupäivä 29.3.2011.
30. Saimaan viitoset ry OH5AB. 2010. Viitosten syysottelu 2010 kilpailukutsu ja säännöt HF. Saatavissa: www.sral.fi/kilpailut/saannot/syysottelu10.pdf. Hakupäivä 29.3.2011.

KVULogChecker

Vaatimusmäärittely

Yleiskuvaus

- Ohjelmiston tarkoituksena on automatisoida radioamatöörikilpailuun osallistuneiden kilpailijoiden sähköisessä tai paperimuodossa lähettämien lokien tarkastus ja pisteidenlaskenta.
- Tulosten laskennan jälkeen ohjelmasta voidaan tulostaa erilaisia raportteja

Yleiset rajoitukset

- Ohjelma toteutetaan Qt 4.6-ohjelmistokehystä käyttäen. Ohjelmiston tulee toimia Mac OSX, Linux ja Windows -ympäristöissä.

Toiminnot

FA1 Tiedostonhallinta

- FA1-1 järjestelmällä pitää pystyä tekemään oma tiedosto jokaista kilpailua varten.
- FA1-2 järjestelmällä voidaan lukea aiemmin tallennettuja tiedostoja.
- FA1-3 järjestelmän tulee voida tuoda CABRILLO-muotoinen tiedosto tietokantaan
- FA1-4 järjestelmä voi tukea UUTest-lokiformaattia
- FA1-5 järjestelmä voi tukea Fintest-ohjelman lokiformaattia

FA2 Paperilokien käsittely

- FA2-1 käyttäjän on voitava syöttää kilpailijan paperimuotoisena lähettämä logi järjestelmään
- FA2-2 järjestelmään voidaan tallennetaan tieto, kuka paperilokin on lähettänyt tilastointia varten

FA3 Lokien tarkastus

- FA3-1 järjestelmän tulee tarkistaa yhteyksien sanomien oikeellisuus
- FA3-2 järjestelmän tulee laskea kilpailijan pisteet kyseisen kilpailun sääntöjen mukaisesti
- FA3-3 järjestelmän tulee hallita kilpailun säännöistä aiheutuvat poikkeustilanteet pistelaskuun
- FA3-4 järjestelmän tulee pitää kilpailijakohtaista virhelokia, joka sisältää kilpailijan hylätyt rivit, rivit joilla virheellinen sanoma, syykoodi ja paljonko pisteitä vähennetty kyseisestä yhteydestä
- FA3-5 järjestelmän tulee tukea käsintarkastusta ja editointia
- FA4-6 järjestelmään tuotava loki on voitava merkitä tarkistus-lokiksi

FA4 Raportointi

- FA4-1 järjestelmän pitää pystyä muotoilemaan lopputuloksista raportti pdf, html ja tekstitiedosto-muodossa.

- FA4-2 järjestelmän pitää pystyä muodostamaan kilpailijakohtaisista virhelokeista yhteenveto
- FA4-3 käyttäjän tulee voida tulostaa raportteja (esim. ei lokia, tarkistuslokin lähettäneet)
- FA4-4 järjestelmästä on voitava tulostaa / katsella yhden tai monen kilpailijan lokia/virhelokia
- FA4-5 käsin editoidut rivit (alkup. ja editoitu) viedään järjestelmän yleiseen editointilokiin

Ulkoiset liittymät

- Ohjelmistolla ei ole muita ulkoisia liittymiä kuin tulostin.

Muut ominaisuudet

- Ohjelmiston tulisi olla helppokäyttöinen sekä toimia monella käyttöjärjestelmäalustalla (Mac OSX, MS Windows, Linux)

KVULogChecker

Käyttäjätarinat

Tarina 1

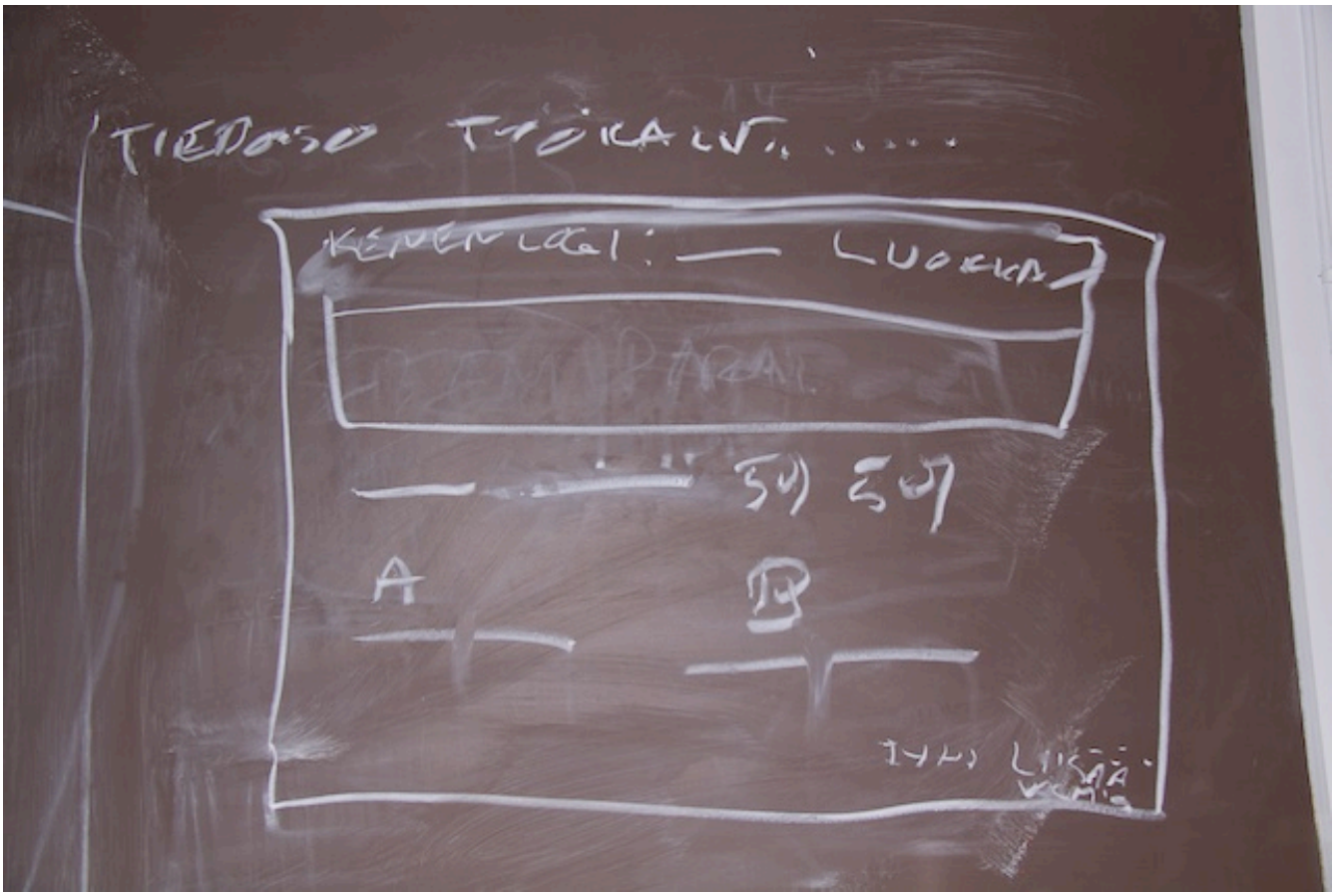
Klikkaan Tiedosto / Uusi kilpailu.
Avautuu dialogi, jossa määritellään kilpailun tiedot

Tarina 2

Klikataan Tiedosto / Tuo hakemisto (cabrillo). Valitaan hakemisto jossa cabrillo-tiedostot ovat. Järjestelmä hakee kaikista hakemistossa olevista cabrillo-tiedostoista kilpailijan kutsun ja tarkistaa lokin validiuden. Käyttäjä määrittelee kutsun perään kyseisen kilpailijan kilpailuluokan (kutsun perässä on combobox jonka oletus on tyhjä).
Klikataan tuo, jonka jälkeen järjestelmä tuo lokit kantaan ja laskee claimed scoret.
Lopuksi näytetään claimed scoret ko. lokeille

Tarina 3

Halutaan tuoda paperilogi
Valitaan Työkalut / Paperilogin syöttö, josta avautuu uusi dialogi, jossa paperilokin tiedot voidaan syöttää kantaan.
Lisää-napilla lisätään qso kantaan.
Valmis-napilla toiminto lopetetaan.
Selausikkunaa tuplaklikkaamalla tuodaan rivi editoitavaksi (Lisää-napista päivitä nappi ja tyhjää napista peruuta-nappi).
Tyhjää-napilla tyhjätään editointirivit.
Lisää napin yhteydessä kasvatetaan kilpailijan sanoman sarjanumeroa yhdellä ja maakuntatunnus pysyy samana. (a-puoli kuvassa 1). Muut kentät tyhjennetään.



Kuva 1 Alustava piirrelmä paperilokien syöttöikkunasta

Tarina 4

Yksittäinen cabrillo voidaan lisätä kilpailuun valitsemalla valikosta Tiedosto/Tuo Cabrillo, josta aukeaa dialogi, johon valitaan tiedosto ja määritellään kilpailijan tiedot.
klikataan import, jonka jälkeen loki tuodaan kantaan ja lasketaan sille claimed score ja näytetään se.

Tarina 5

Kun logit on tuotu kantaan laitetaan kone laskemaan tulokset valitsemalla Työkalut / laske tulokset.
Näytetään progress dialog.
Laskennan valmistuttua näytetään tulokset luokittain, puolikusolokit, puuttuvat lokit, tarkistuslokit.
Virhelokeihin pääsee aseman tuloksen perässä olevasta napista.

Tarina 6

UUTest-lokin tuonti valitsemalla valikosta Tiedosto/Tuo UUTest, josta aukeaa dialogi johon valitaan tiedosto ja määritellään kilpailijan tiedot.
Klikataan import, jonka jälkeen loki tuodaan kantaan ja lasketaan sille claimed score ja näytetään se.

Tarina 7

FinTest-lokin tuonti valitsemalla valikosta Tiedosto/Tuo Fintest, josta aukeaa dialogi, johon valitaan tiedosto ja määritellään kilpailijan tiedot.
Klikataan import, jonka jälkeen loki tuodaan kantaan ja lasketaan sille claimed score ja näytetään se.

Tarina 8

Valitsemalla valikosta tiedosto/avaa aukeaa tiedoston avausdialogi, josta valitaan .kvulc tiedosto. Klikataan open, niin tiedosto avataan.

Tarina 9

Klikkaamalla hiiren oikealla näppäimellä pääikkunassa olevassa osallistujat-listassa osallistujan kutsua saadan esille kontekstivalikko, josta seuraavat valinnat:

- Avaa virheloki
- Näytä loki
- Näytä kilpailijan tiedot
- Poista kilpailija
- Päivitä lista

Em. toiminnot avaavat uuden ikkunan, jossa toimintoja pääsee toteuttamaan.
Tuplaklikkaamalla kilpailijan kutsua, avataan kyseisen kilpailijan loki tarkistusta / muokkausta varten.
“Poista kilpailija” varmistaa poistamisen erillisellä messageboxilla.

Tarina 10

Kilpailijan voi poistaa järjestelmästä valitsemalla Työkalut / Poista kilpailija, valitaan avautuvasta dialogista kilpailija joka halutaan poistaa. Klikkaamalla “poista” poistetaan kilpailija tietokannasta, klikkaamalla Peruuta; peruutetaan toiminto.