Kimmo Keronen

Hybridiohjelmointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

16.4.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Kimmo Keronen Hybridiohjelmointi 33 sivua + 1 liite 16.4.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaajat	lehtori Simo Silander systeemiarkkitehti Tuomo Telkkä

Insinöörityössä tutkittiin, minkälaisia etuja voidaan saavuttaa ohjelmoimalla tietokoneohjelma useammalla ohjelmointikielellä. Tutkimuksen tuloksia hyödyntäen toteutettiin julkaisualan ohjelmistoyritys Anygraaf Oy:lle ohjelmamoduuli, jolla voidaan mallintaa visualisesti tietokannassa olevaa dataa kuvaajin ja taulukoin.

Tutkimus aloitettiin selvittämällä, miten ohjelmointikieliä voidaan jaotella ryhmiin ja mitä heikkouksia ja vahvuuksia erilaisilla ohjelmointikielillä on. Tutkimuksesta selvisi esimerkiksi, että joidenkin ohjelmointikielien vahvuus on lopputuloksena syntyvä suorituskykyinen ohjelma mutta kielellä ohjelmointi on raskasta ja hidasta. Joissain ohjelmointikielissä tilanteen havaittiin olevan päinvastainen.

Seuraavaksi tutkittiin menetelmiä joilla tällaisia ohjelmointikielipareja voidaan yhdistää. Havaittiin, että on olemassa niin sanottuja vieraan funktion rajapintoja, jotka mahdollistavat ohjelmointikielien väliset operaatiot. Hybridiohjelmoinnille ja vieraan funktion rajapintojen käytölle löydettiin sovelluskohteita niin mobiiliohjelmista kuin perinteisistä työpöytäsovelluksistakin. Mahdollisia sovelluskohteita ovat muun muassa monialustaiset mobiiliohjelmat, suorituskykykriittiset ohjelmat sekä valmiiseen ohjelmaan jälkikäteen toteutettavat ohjelmalaajennukset.

Insinöörityössä toteutettu ohjelmamoduuli toimii esimerkkinä hybridiohjelmasta. Ohjelmamoduuli toteutettiin C++-kieliseen ohjelmaan, mutta kuvaajien ja taulukoiden luomiseen käytettiin JavaScript-kielistä kirjastoa, jqPlotia. Erikieliset osuudet yhdistettiin tutkimuksessa esitellyin keinoin. Lopputuloksena syntyneellä ohjelmamoduulilla voidaan tuottaa graafista sisältöä esimerkiksi lehtiartikkeleihin tai yrityksen sisäistä seurantaa tukevia kuvaajia.

Avainsanat	ohjelmointi, hybridi, mobiili, monialusta

Author Title Number of Pages Date	Kimmo Keronen Hybrid Programming 33 pages + 1 appendix 16 April 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Simo Silander, Senior Lecturer Tuomo Telkkä, System Architect

The subject of this thesis was to study the advantages of compiling software using several different programming languages. On the basis of the study, a software module able to visualize database data with graphs and tables was developed for Anygraaf Oy.

The study begun by finding out how different programming languages can be sorted in groups and what are the strengths and weaknesses of these groups. The study showed for example that the strength of some of the languages is a result of very fast application but the development process with the language can be heavy and slow. In some cases, the situation was the opposite.

The next step was to find out means to combine a pair of otherwise incompatible programming languages. Special foreign function interfaces exist to enable cross-language operations. Applications for hybrid programming and the foreign function interfaces were found in areas in mobile development and traditional desktop development alike. Possible applications are for example multi-platform mobile applications, performance critical applications and software plugins made for existing applications.

The software module developed in the study is an example of a hybrid program. The module was implemented into an application programmed in the C++ programming language and the software library used for producing the graphs and tables, named jqPlot is programmed in JavaScript. The different parts were combined by means found out in the study. The resulting software module can be used for example to add graphical content to a news paper article or to produce graphs supporting the company's internal monitoring.

Keywords	programming, hybrid, mobile, multi-platform

Sisällys

Lyhenteet ja sanasto

Android	Nykyisin Googlen kehittämä mobiilikäyttöjärjestelmä.
AnyReader	Anygraaf Oy:n kehittämä mobiililukuohjelma.
C	Dennis Ritchien kehittämä imperatiivinen ohjelmointikieli.
C++	Bjarne Stourstrupin C-kielestä jatkokehittämä ohjelmointikieli.
C#	Microsoftin kehittämä oliopohjainen ohjelmointikieli.
CFFI	Common Foreign Function Interface. Common Lispin vieraan funktion rajapinta.
Common Lisp	Lisp-ohjelmointikielen murre, joka tukee useita ohjelmointiparadigmoja.
CSS	Cascading Style Sheets. Kuvauskieli HTML-sivujen ulkoasun määrittämiseen.
Ctypes	Python-ohjelmointikielen vieraan funktion rajapinta.
cURL	Monialustainen tiedonsiirtoon tarkoitettu ohjelmakirjasto.
FFI	Foreign Function Interface. Vieraan funktion rajapinta kahdella eri ohjelmointikielellä tehdyn ohjelmamoduulin yhdistämiseen.
GPS	Global Positioning System. Satelliitteihin perustuva paikannusjärjestelmä.
HTML	Hypertext Markup Language. Nettisivujen määrittelemiseen käytetty kuvauskieli.
iOS	Applen kehittämä mobiilikäyttöjärjestelmä.
Java	Sun Microsystemsin kehittämä ohjelmointikieli.

JavaScript	Web-ohjelmissa käytettävä tulkattava ohjelmointikieli.
JNI	Java Native Interface. Java-kielen vieraan funktion rajapinta.
jqPlot	Chris Leonellon kehittämä kuvaajien piirtämiseen tarkoitettu JavaScript-kirjasto.
jQuery	jQuery Teamin kehittämä yleiskäyttöinen JavaScript-kirjasto.
Neo	Anygraaf Oy:n kehittämä toimitusjärjestelmä.
Next Media	Tivit Oy:n ylläpitämä mediatuotannon projekti, johon Anygraaf Oy kuuluu.
Objective-C	Applen kehittämä oliopohjainen ohjelmointikieli.
OpenGL	Monialustainen grafiikan piirtämiseen tarkoitettu ohjelmakirjasto.
PhoneGap	Nykyisin Adoben kehittämä mobiilikehitysympäristö.
Python	Python Software Foundationin kehittämä tulkattava ohjelmointikieli.
Qt	Nykyisin Digian kehittämä C++-pohjainen ohjelmistokehitysympäristö.
SQL	Structured Query Language. IBM:n kehittämä tietokantojen kyselykieli.
SWIG	Simplified Wrapper and Interface Generator. Ohjelma vieraan funktion rajapintojen luomiseen eri ohjelmointikielille.
Tekes	Teknologian ja innovaatioiden kehittämiskeskus. Suomen valtion virasto, joka rahoittaa tutkimus- ja kehitysprojekteja.
XULRunner	Mozilla Foundationin kehittämä ohjelmointiympäristö.

1 Johdanto

Tämän insinöörityön aiheena on tutkia erilaisia mahdollisuuksia toteuttaa ohjelmia ja niiden osia käyttäen useampaa kuin yhtä ohjelmointikieltä. Tarkoitus on selvittää, voidaanko ohjelmointikieliä yhdistämällä saavuttaa ratkaisuja, joissa kunkin kielen vahvuuksia voidaan hyödyntää ja heikkouksia paikata. Tutkimuksen tuloksia apuna käyttäen toteutetaan Anygraaf Oy:n ohjelmistoihin ohjelmamoduuli, jolla voidaan visualisoida tietokannassa olevaa dataa esimerkiksi kuvaajilla ja taulukoilla. Ohjelmamoduuli halutaan toteuttaa mahdollisuuksien mukaan hyödyntäen valmiita avoimen lähdekoodin ratkaisuja, mikä luo tarpeen insinöörityön aiheena olevalle tutkimustyölle.

Anygraaf Oy on ohjelmistoyritys, joka on erikoistunut julkaisualan ohjelmistoihin, kuten toimitus- ja ilmoitusjärjestelmiin, levikin- ja jakelunhallintaohjelmistoihin sekä monikanavajulkaisemiseen. Anygraafilla on 45 työntekijää, jotka jakautuvat Helsingin ja Turun konttoreihin sekä Ruotsissa ja Yhdysvalloissa sijaitseviin tytäryhtiöihin. Pääkonttori on Helsingissä. Insinöörityössä toteutettava ohjelmamoduuli tulee käyttöön Anygraafin Neo-toimitusjärjestelmään, mutta siitä on tarkoitus tehdä moduuli, jota voidaan tarvittaessa käyttää muissakin Anygraafin ohjelmissa. Onnistuessaan ohjelmamoduulin merkitys Anygraafille ja sen asiakkaille voi olla suuri, sillä Neoa käyttävät Anygraafin asiakkaiden lisäksi myös sen omat työntekijät toiminnanohjausjärjestelmänä. Ohjelmamoduulilla tuotettavia kuvaajia ja taulukoita voidaan käyttää esimerkiksi yrityksen sisäisessä seurannassa tai toimitustyössä lehtiartikkeleita tukevana materiaalina. Projekti kuuluu myös osana viestintäalan tutkimusohjelma Next Mediaan, johon Anygraaf Oy osallistuu. Next Medialla on osittainen Tekes-rahoitus ja sitä koordinoi Tivit Oy [1].

Työn alussa tutkitaan erilaisia ohjelmointikieliä. Kielistä kartoitetaan niiden vahvuuksia, heikkouksia ja soveltumista yhdistettäväksi jonkin toisen kielen kanssa. Kun sopivia ohjelmointikieliä on löydetty, etsitään erilaisia tekniikoita yhdistää niitä ja mietitään hyötyjä tai haittoja näillä tekniikoilla on ohjelman toimivuuden tai esimerkiksi ohjelmakoodin ymmärrettävyyden kannalta. Tämän jälkeen voidaan etsiä kieliyhdistelmille erilaisia sovelluskohteita sekä työpöytäsovelluksista että mobiiliohjelmista ja toteutetaan edellä mainittu ohjelmamoduuli, jonka määrittely-, suunnittelu- ja toteutusprosesseihin paneudutaan syvemmin. Lopuksi pohditaan työn tuloksia ja sen onnistumista ja sitä, mitkä mahdollisista sovelluskohteista ovat mielenkiintoisimpia ja olisiko niissä mahdollisuuksia jatkokehitykselle.

2 Ohjelmointikielet

Erilaisia ohjelmointikieliä on satoja tai jopa tuhansia [2]. On kuitenkin mielekästä rajata tutkittavat kielet yleisimpiin teollisuudessa käytettäviin kieliin. Monissa sovelluskohteissa, kuten mobiilisovelluksissa tämä on myös edellytys, sillä niissä voidaan käyttää vain alustan tukemia kieliä.

Ohjelmointikieliä voidaan jaotella ryhmiin monin eri perustein [3]. Yleisimpiä jaotteluperusteita ovat kielen paradigma eli kielen lähtökohta mallintaa jonkin ohjelmointitehtävän ratkaisu sekä tapa, jolla ohjelmakoodi muutetaan sarjaksi tietokoneen ymmärtämiä käskyjä. Lisäksi ohjelmointikieliä voidaan jakaa luokkiin esimerkiksi ilmaisuvoiman, abstraktiotason, virheenhallinnan ja tiedon tyypittämisen mukaan. Ohjelmointikielien tarkka jaottelu on usein kuitenkin vaikeaa, ja monet kielet ovatkin niin sanottuja hybridikieliä eli ne yhdistävät ominaisuuksia useammista ryhmistä. Yksittäisten kielten ominaisuuksien selvittämistä mielekkäämpää on tutkia edellä kuvattujen jaottelujen mukaisten kieliryhmien ominaisuuksia. Keskitytään erityisesti ryhmien heikkouksiin ja vahvuuksiin. Voidaanko löytää yhdistelmiä, joissa toisen ryhmän vahvuus paikkaa toisen ryhmän heikkouden?

2.1 Ohjelmointiparadigmat

Ohjelmointiparadigma on ohjelmointikielen taustalla oleva periaate tai tapa ajatella ja mallintaa asioita. Ohjelmointikielet voidaan jaotella neljään paradigmaan: oliopohjaiseen ohjelmointiin, proseduraaliseen ohjelmointiin, funktionaaliseen ohjelmointiin ja deklaratiiviseen ohjelmointiin [3]. Ohjelmointikieli voi noudattaa yhtä paradigmaa tai tukea useita paradigmoja.

Olio-ohjelmoinnissa ohjelma koostuu olioista, jotka sisältävät loogista tietoa ja toiminnallisuutta. Oliot voidaan ajatella itsenäisiksi koneiksi, jotka viestivät keskenään. Olio-ohjelmointi on kehitetty helpottamaan ohjelmistojen kehittämistä ja ylläpitoa, ja sen vahvuuksia ovatkin ohjelmakoodin uudelleenkäytettävyys ja helppo ymmärrettävyys todellista maailmaa vastaavien käsitteiden ansiosta [5]. Olio voi olla esimerkiksi ohjelmistossa oleva mallinnus ihmisestä. Sillä voi olla ominaisuuksia, kuten pituus ja paino, sekä toimintoja, kuten kävelemien tai puhuminen. Olio-ohjelmoinnissa tieto kapseloidaan olioiden sisälle, mikä lisää turvallisuutta ja vähentää ohjelmointivirheitä. Oliopoh-

jaisista ohjelmista saattaa tulla monimutkaisia, ja ne ovat usein hitaampia ja käyttävät enemmän muistia muita paradigmoja noudattaviin ohjelmiin verrattuna.

Proseduraalinen ohjelmointi on olio-ohjelmointia perinteisempi paradigma, jossa ohjelma koostuu sarjasta peräkkäin suoritettavia komentoja tai aliohjelmia, jotka muuttavat ohjelman tilaa eli sen tietorakennetta tietyllä ajan hetkellä. Proseduraalinen ohjelma on kaikkein luonnollisin tapa kertoa tietokoneelle, mitä tehdä, sillä myös tietokoneen ymmärtämä konekieli on proseduraalista [6]. Proseduraalinen ohjelma saattaa olla muistin käytöltään tehokkaampi kuin saman asian tekevä oliopohjainen ohjelma. Proseduraalisessa ohjelmassa joudutaan turvautumaan olio-ohjelmaa enemmän globaaliin dataan, mikä vähentää ohjelman turvallisuutta. Uusien tietotyyppien luominen on myös hankalampaa kuin oliopohjaisissa kielissä, mikä vähentää ohjelman laajennettavuutta.

Funktionaalinen ohjelmointi perustuu matemaattisten funktioiden käyttöön. Puhtaasti funktionaalisessa ohjelmassa ei ole tilaa. Koska ohjelmalla ei ole tilaa, sen koostavilla funktioilla ei ole sivuvaikutuksia. Sama funktio aiheuttaa samalla syötteellä aina saman lopputuloksen. Tästä johtuen ohjelman rinnakkainen suoritus monisäikeisesti ei aiheuta ongelmaa, ja ohjelman suorittaminen voi olla todella tehokasta.

Deklaratiivinen ohjelmointi on proseduraalisen ohjelmoinnin vastakohta. Siinä ei suoraan sanota tietokoneelle, mitä sen pitäisi tehdä, vaan mitä pitäisi saada aikaan. Funktionaalisen ohjelmoinnin tapaan deklaratiivinen ohjelmointi sopii hyvin rinnakkaislaskentaan.

Taulukko 1. Ohjelmointikielten tukemat ohjelmointiparadigmat.

Ohjelmointikieli	Olio-ohjelmointi	Proseduraalinen ohjelmointi	Funktionaalinen ohjelmointi	Deklaratiivinen ohjelmointi
C		X		
C#	X	X	X	
C++	X	X	X	
Haskell			X	
Java	X	X		
JavaScript		X	X	
Lisp			X	
Lua	X	X	X	
Objective-C		X	X	
Pascal		X	X	
Perl	X	X	X	
PHP	X	X		
Python	X	X	X	
Ruby	X	X	X	
SQL				X
Visual Basic		X		

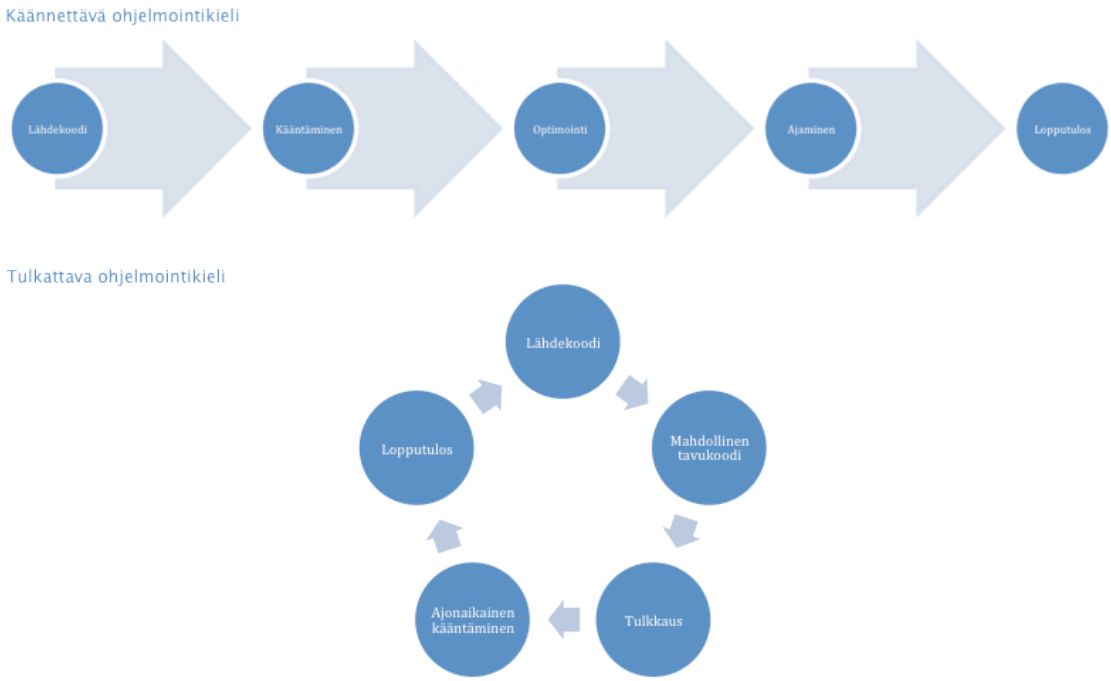
Taulukosta 1 voidaan havaita, että suurin osa yleisimmistä ohjelmointikielistä tukevat useampaa kuin yhtä paradigmaa. Monet kielet esimerkiksi sisältävät funktionaalisen ohjelmoinnin piirteitä, vaikka eivät kykenekään puhtaaseen funktionaaliseen ohjelmointiin.

2.2 Käännettävät ja tulkattavat kielet

Ohjelmointikielet voidaan jaotella sen mukaan, käännetäänkö ohjelmakoodi tietokoneen ymmärtämäksi konekieleksi etukäteen vai tehdäänkö se vasta ohjelman ajamisen aikana. Molempiin tapoihin liittyy hyviä ja huonoja puolia, mutta kuten ohjelmointiparadigmojen yhteydessä, käännettävien ja puhtaasti tulkattavien skriptikielien välinen raja on hämärtynyt.

Perinteisesti käännettävillä kielillä tehdyt ohjelmat ovat nopeampia suorittaa kuin tulkattavilla kielillä. Kääntäjä optimoi konekielisen ohjelman toimimaan jollakin tietyllä tietokonearkkitehtuurilla. Tulkattavalla kielellä koodattu ohjelma ajetaan niin sanotun tulkkiohjelman sisällä, joka myös kääntää ohjelman konekieliseksi ajon aikana. Tästä syntyy ylimääräistä ajonaikaista rasitetta. On myös olemassa kieliä, kuten Java, jotka

käännetään etukäteen tavukoodiksi, joka on jotakin selkokielisen ohjelmakoodin ja konekielen väliltä, mutta ajo tapahtuu silti niin sanotussa virtuaalikoneessa, joten raja kahden tavan välillä ei aina ole selkeä. Kuvasta 1 selviää periaatteellinen ero käännettävän ja tulkattavan ohjelmointikielen välillä. Käännettävällä ohjelmointikielellä toteutetun ohjelman matka lähdekoodista ajettavaksi ohjelmaksi on varsin suoraviivainen, kun taas tulkattavalla kielellä sama lähdekoodirivi saatetaan kääntää konekieliseksi uudestaan ja uudestaan ajon aikana.



Kuva 1. Käännettävän ja tulkattavan ohjelmointikielen prosessikaaviot.

Tietokoneiden suorituskyky on kasvanut viime vuosien aikana huomattavasti, ja skriptikielien tulkit ovat kehittyneet niin hyviksi, että käännöstavan vaikutus suorituskykyyn tavanomaisessa työpöytäkäytössä on vähäinen [7].

Ohjelmointi skriptikielellä voi olla huomattavasti nopeampaa ja helpompaa kuin käännettävällä kielellä. Riittää, että ohjelmakoodiin tehdään muutos ja ohjelma käynnistetään uudestaan. Ohjelman kääntäminen etukäteen voi olla verrattain pitkäkestoinen prosessi, jos kyseessä on iso ohjelma.

Taulukko 2. Käännettäviä ja tulkattavia ohjelmointikieliä.

Ohjelmointikieli	**Käännettävä kieli**	**Tulkattava kieli**
C	X	
C#	X	X
C++	X	
Java	X	X
JavaScript		X
Lua		X
Objective-C	X	
PHP		X
Python		X

Taulukossa 2 luetellaan käännettäviä ja tulkattavia ohjelmointikieliä. C# ja Java ovat jotakin tältä väliltä, sillä niillä tehdyt ohjelmat käännetään etukäteen virtuaalikoneen ymmärtämään muotoon, jonka jälkeen ajo tapahtuu virtuaalikoneessa.

2.3 Abstraktiotaso

Ohjelmointikielen abstraktiotasolla tarkoitetaan sitä, kuinka kaukana kielen käsitteet ovat tietokoneen ymmärtämästä konekielestä [8]. Puhutaan korkean ja matalan tason ohjelmointikielistä. Voidaan ajatella, että korkean tason ohjelmointikielet ovat lähempänä ihmisen luonnollista käsitemaailmaa ja täten helpompia ymmärtää ja käyttää. Tietokoneelle tärkeät asiat kuten muistinhallinta saattaavat olla automatisoitua tai kätketty kokonaan ohjelmoijalta. Korkean tason kielissä käsitellään muuttujia ja olioita, kun taas matalan tason kielissä voidaan tehdä suoria operaatioita muistiin.

Perinteisesti matalan tason ohjelmointikieli tuottaa tehokkaampia ja kompaktimpia ohjelmia kuin korkean tason kieli, sillä koodin muuttaminen konekieleksi on huomattavasti suoraviivaisempaa. Vaikka tietokoneiden suorituskyky on kasvanut huimasti, tämä saattaa silti olla merkittävä tekijä paljon suorituskykyä vaativissa ohjelmissa, kuten esimerkiksi peleissä ja ympäristöissä, joissa suorituskyky on rajallinen kuten mobiililaitteissa.

2.4 Vahvan ja heikon tyypityksen kielet

Neljäs tässä yhteydessä kiinnostava ohjelmointikieliä jakava ominaisuus on, kuinka vahvasti ne tyypittävät käytettäviä muuttujia. Vahva tyypittäminen tarkoittaa sitä, että jokaisella muuttujalla on jokin tyyppi, ja ne voivat saada ainoastaan samantyyppisiä arvoja. Kielissä joissa on vahva tyypitys, kääntäjä huomaa, jos ohjelmoija yrittää asettaa muuttujalle vääränlaisen muuttujan, ja aiheuttaa käännösaikaisen virheen. Näin tällaiset ohjelmointivirheet eivät voi päätyä ajettavaan ohjelmaan. Heikon tyypityksen kielissä tällaisia tarkastuksia ei ole, ja oikeiden tyyppien käyttäminen on ohjelmoijan vastuulla. Heikko tyypitys nopeuttaa ohjelmakoodin kirjoittamista, mutta yleisesti vahvan tyypityksen hyötyjä pidetään suurempina kuin heikon [9].

Taulukko 3. Vahvan ja heikon tyypityksen ohjelmointikieliä.

Ohjelmointikieli	**Tyypitys**	**Tyypin tarkistus**
C	Heikko	Staattinen
C#	Vahva	Staattinen
C++	Vahva	Staattinen
Haskell	Vahva	Staattinen
Java	Vahva	Staattinen
JavaScript	Heikko	Dynaaminen
Lua	Heikko	Dynaaminen
Objective-C	Heikko	Dynaaminen (staattisuus valinnainen)
Pascal	Vahva	Staattinen
Perl	Heikko	Dynaaminen
PHP	Heikko	Dynaaminen
Python	Vahva	Dynaaminen
Ruby	Vahva	Dynaaminen
Visual Basic	Vahva	Staattinen

Taulukosta 3 nähdään että yleisesti käännettävillä kielillä on myös vahva tyypitys ja staattinen tyypityksen tarkistus. Staattinen tarkistaminen tarkoittaa kääntämisen yhteydessä tapahtuvaa, ja dynaaminen ajon aikana tapahtuvaa tarkistamista. Monet skriptikielet ovat niin sanottuja heikon tyypityksen kieliä, jotka korostavat niiden käytön helppoutta.

2.5 Hybridiohjelmointiin sopivat kielet

Ohjelmointikielten jaotteluissa toistuviksi teemoiksi osoittautuivat kielen ymmärrettävyys sekä helppokäyttöisyys ja toisaalta taas lopputuloksena syntyvän ohjelman suorituskykyisyys. Kaikki luetellut asiat ovat tavoiteltavia, mutta ne eivät usein yhdisty samassa ohjelmointikielessä. Tästä syystä ne ovat kuitenkin hyviä valintaperusteita hybridiohjelmointiin sopivia kieliä etsittäessä. Ominaisuudet voidaan saavuttaa yhdistämällä esimerkiksi jokin abstraktiotasoltaan matala käännettävä kieli, kuten esimerkiksi C, jolla saavutetaan suorituskykyinen ohjelma, ja jokin korkean tason tulkattava skriptikieli, jolla ohjelmointi on helppoa ja nopeaa, kuten esimerkiksi Python, Lua ja JavaScript.

C ja C++ tarjoavat mahdollisuuksia matalan tason ohjelmointiin ja sen tuomaan suorituskykyyn. Ne ovat laajasti tuettuja eri sovellusalustoilla, joten ne soveltuvat hyvin hybridiohjelmointiin. C++ on myös tässä työssä toteutettavan esimerkin alustaohjelman ohjelmointikieli. Mobiilialustoja tutkittaessa mukaan on valittava myös kunkin alustan natiivi ohjelmointikieli. Esimerkiksi Java Android -sovelluksissa ja Objective-C Applen iOS-tuoteperheen sovelluksissa. Lopullinen käytettävien kielten valinta riippuu siis tutkittavasta sovelluskohteesta.

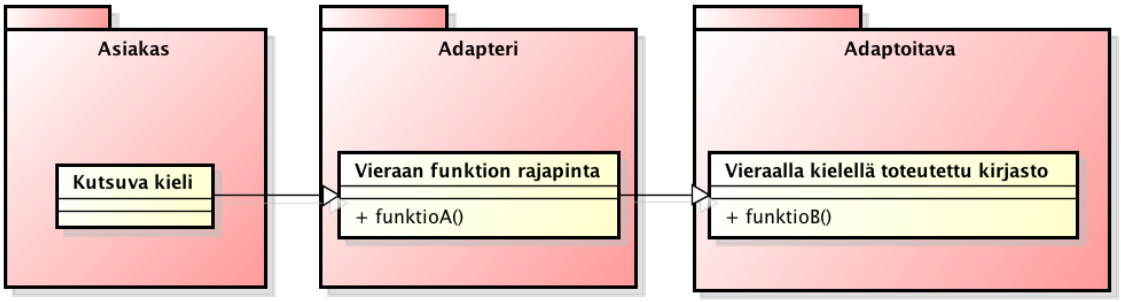
2.6 Ohjelmointikielten yhdistäminen

Sellaisia ohjelmointikieliyhdistelmiä, joissa toinen kieli on johdettu toisesta, voidaan usein käyttää suoraan ilman minkäänlaisia lisätoimenpiteitä. Esimerkiksi C++-ohjelmassa voidaan kutsua C:llä kirjoitettuja funktioita. Useimmiten tilanne on kuitenkin monimutkaisempi ja yhdistettävien kielien välille tarvitaan jonkinlainen rajapinta, jota molemmat kielet ymmärtävät. Tällaista rajapintaa kutsutaan termillä foreign function interface (FFI) eli vieraan funktion rajapinta [10].

Vieraan funktion rajapinnat

Vieraan funktion rajapinta on ohjelmistotekniikan mekanismi, jossa jollain ohjelmointikielellä kirjoitettu ohjelma voi kutsua toisella kielellä kirjoitetun kirjaston funktioita [10]. Vieraan funktion rajapinnalla tarkoitetaan sitä ”liimakoodia”, jonka ainoa tehtävä on sovittaa nämä muuten yhteensopimattomat rajapinnat. Suunnittelumalliltaan vieraan funktion rajapinta on adapteri-mallin mukainen [11]. Sen vastuulla on rajapintojen lisäk-

si myös tietotyyppien muuntaminen kutsuvan kielen ymmärtämään muotoon. Usein vieraan funktion rajapinta määritellään korkeamman tason ohjelmointikielessä, josta halutaan kutsua jotakin matalamman tason ohjelmointikielen toiminnallisuutta. Monissa dynaamisissa korkean tason ohjelmointikielissä on sisäänrakennettu vieraan funktion rajapinta, kuten Javan Java Native Interface (JNI), Pythonin ctypes-kirjasto ja Common Lispin CFFI. Nämä rajapinnat on kehitetty mahdollistamaan suorat järjestelmäkutsut, jotka on toteutettu käyttöjärjestelmästä riippuen usein jollain muulla kielellä kuin kutsuvalla korkean tason kielellä. Järjestelmäkutsujen mahdollistamisen lisäksi vieraan funktion rajapinnalla on merkittävä rooli hybridiohjelmoinnissa, koska vain harvoin kielet ovat suoraan yhteensopivia ja jokin sovittava rajapinta tarvitaan.



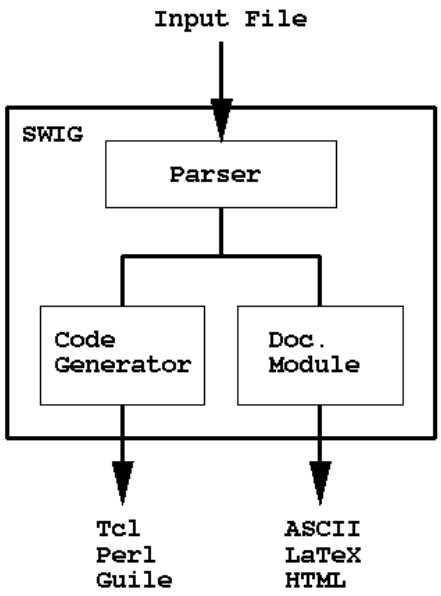
Kuva 2. Adapteri-suunnittelumallin mukaisesti toteutettu vieraan funktion rajapinta.

Kuvassa 2 hahmotetaan vieraan funktion rajapinnan toimintaa. Jollakin korkean tason ohjelmointikielellä toteutettu asiakasohjelma voi kutsua toisella kielellä toteutettua funktiota B kutsumalla vieraan funktion rajapinnan funktiota A. Vieraan funktion rajapinnan toiminnallisuus ei välttämättä rajoitu pelkästään funktiokutsuihin, vaan se voi myös adaptoida olioita ja niiden metodeja sekä mahdollistaa ohjelmointikielten välisen perinnän. Rajapinta voi myös olla kaksisuuntainen, jolloin myös kutsuttava kieli voi käyttää isäntäkielen toimintoja.

Vieraan funktion rajapintojen käytön haittapuolena voidaan pitää ohjelman arkkitehtuurin lisääntynyttä kompleksisuutta. Ohjelmakoodista voi tulla vähemmän ymmärrettävä vieraan funktion rajapinnan käytöstä aiheutuvan ylimääräisen koodin takia. Myös ohjelman virheiden etsintä, eli debuggaus muuttuu vaikeammaksi, koska jollekin ohjelmointikielelle suunniteltujen kehitysympäristöjen virheenetsintätyökalut eivät osaa tulkita toisen kielen symboleita, tai käytettävä kirjasto voi olla valmiiksi käännetty, jolloin lähdekoodia ei edes voida nähdä. Erilailla toteutettu muistinhallinta, roskien keruu ja virheviestien välittäminen saattaavat aiheuttaa myös hankaluuksia. Esimerkiksi Javan

virtuaalikoneen roskien keruu ei ylety Javan vieraan funktion rajapinnan, JNI:n kautta ajettavaan ohjelmaan [12].

Eräs vieraan funktion rajapintojen luontiin tarkoitettu ohjelmistokehitystyökalu on SWIG (Simplified Wrapper and Interface Generator) [13]. SWIG osaa luoda ohjelmoijan valitsemista C- ja C++-ohjelmista rajapinnan valitulle kohdekielelle. Tuettuja kohdekieliä ovat ainakin Python, Perl, PHP, Lua, Java ja Ruby.



Kuva 3. SWIGin rakenne. [14]

Kuva 3 selventää SWIGin toimintaperiaatetta. Sisään tulevat C- ja C++-tiedostot parsitaan ja niiden sisältämistä funktioista luodaan rajapinta valituille kohdekielille. SWIG osaa myös automaattisesti luoda dokumentaation määritellystä rajapinnasta.

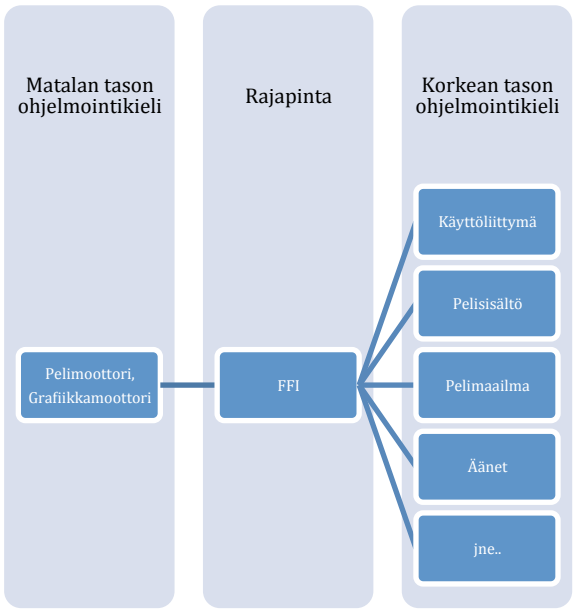
3 Sovelluskohteet

Hybridiohjelma on ohjelma, jonka ohjelmointiin on käytetty useampaa kuin yhtä ohjelmointikieltä. Termi ei käsitä sellaisia ohjelmia, joissa eri kielten välinen rajapinta on ohjelman ulkopuolinen. Esimerkiksi Javalla kirjoitettu ohjelma, joka kommunikoi PHP:lla kirjoitetun palvelinohjelman kanssa internetin välityksellä, ei tässä työssä ole hybridiohjelma. Ohjelman loppukäyttäjälle ei välttämättä näy mitenkään, että ohjelmointiin on käytetty useita ohjelmointikieliä. Monikielisyys voi myös olla ohjelman ominaisuus, jota

loppukäyttäjätkin voivat hyödyntää. Skriptikielien ajonaikainen tulkinta luo mahdollisuuden esimerkiksi laajentaa ohjelman toimintaa loppukäyttäjän määrittelemällä skriptillä, joka suoritetaan jossakin tietyssä tilanteessa.

3.1 Suorituskykykriittiset ohjelmat

Hybridiohjelmien sovelluskohteita voidaan etsiä sekä perinteisistä työpöytäsovelluksista että mobiiliohjelmista. Kategorioilla on myös yhteisiä haasteita, kuten suorituskykyisyys, joita voidaan ratkaista yhdistämällä ohjelmointikieliä. Ohjelmakoodin kohtia, joita suoritetaan hyvin usein tai jotka vaativat runsaasti prosessointiaikaa, sanotaan suorituskykykriittisiksi. Suorituskykykriittiset osuudet voidaan ohjelmoida matalan tason ohjelmointikielellä, jolla ohjelmoija voi hallita ohjelmakoodin suorituskykyisyyttä korkean tason ohjelmointikieltä paremmin. Hyviä esimerkkejä ovat tietokoneohjelmat, joissa tapahtuu paljon matemaattista laskentaa, sekä graafisesti kehittyneet tietokonepelit. Grafiikan piirtämisen ja pelimekaniikan moottorit ovat hyvin suorituskykykriittisiä. Pelin ohjelmistoarkkitehtuuri voidaan suunnitella siten, että grafiikka- ja pelimoottorit ohjelmoidaan matalan tason kielellä kuten C:llä ja muu osuus pelistä voidaan toteuttaa jollakin kevyemmällä korkeamman tason kielellä kuten Lualla, jolla kehitystyö on helpompaa ja nopeampaa. Kuvassa 4 nähdään yksinkertaistettu malli tietokonepelistä, joka hyödyntää hybridiohjelmointia.



Kuva 4. Malli tietokonepelin ohjelmistoarkkitehtuurista.

3.2 Uudelleenkäytettävä ohjelmakoodi

Suorituskykykriittisyyden lisäksi toinen keskeinen haaste ohjelmistokehityksessä, johon hybridiohjelmointi voi vastata, on koodin uudelleenkäytettävyys. On mielekästä, että esimerkiksi jotakin hyvin toteutettua ohjelmakirjastoa voidaan käyttää uudelleen mahdollisimman monessa paikassa. Tilanteesta riippuen käytettävä ohjelmointikieli voi kuitenkin olla eri kuin se, jolla kirjasto alun perin on ohjelmoitu. Sen sijaan, että kirjasto ohjelmoitaisiin uudestaan kohdekielellä, voidaan käyttää vieraan funktion rajapintaa kutsuvan kielen ja vanhan kirjaston välillä. Monesti käyttöjärjestelmien tarjoamat ohjelmistorajapinnat on ohjelmoitu matalan tason ohjelmointikielellä kuten C:llä [15]. Niiden kutsuminen jollain muulla ohjelmointikielellä vaatii siis vieraan funktion rajapinnan käyttämistä. Tästä syystä ohjelmointikielien sidostekniikoita on standardoitu ISO-työryhmässä JTC1/SC22/WG11 [16].

3.3 Monialustasovellukset

Kilpailukykyisten mobiilialustojen määrä aiheuttaa alati kasvavia haasteita mobiilikehittäjille. Mobiiliohjelma halutaan julkaista mahdollisimman monella alustalla laajan asiakaskunnan saavuttamiseksi. Eri alustoille toteutettavien versioiden ohjelmointi ja ylläpitäminen voi kuitenkin olla työlästä. Mobiilisovellukset ohjelmoidaan ensisijaisesti samalla ohjelmointikielellä, jolla mobiilialustan käyttöjärjestelmä on toteutettu, sillä mobiilialustan rajapinnat laitteen toimintojen käyttämiseksi on saatavilla vain tällä ohjelmointikielellä. Taulukossa 4 on esitetty eri mobiilialustojen tukemia ohjelmointikieliä.

Taulukko 4. Mobiilialustojen tukemat ohjelmointikielet.

Mobiilialusta	**Ohjelmointikieli**	**Tuetut kielet**
Android	Java	C++
iOS	Objective-C	Objective-C++
Symbian	Qt	C++
Windows Phone 8	C#	C++

Eri mobiilialustojen ylläpitämisen työläydestä aiheutuva ongelma voidaan ratkaista toteuttamalla mahdollisimman suuri osa ohjelmasta ohjelmointikielellä, joka on tuettu kaikilla alustoilla. Tällaisia kieliä ovat ainakin C++ ja JavaScript selainnäkymän kautta.

Selainnäkymät

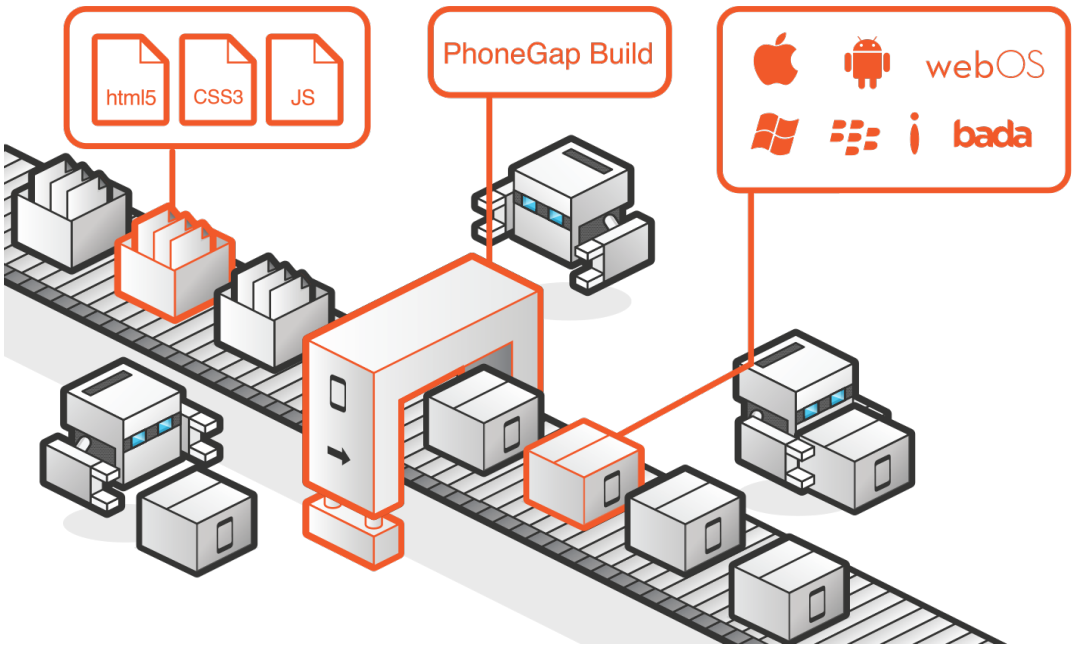
Ohjelmointirajapintojen tarjoamat selainnäkymät on eräs tapa hyödyntää ohjelmakoodin uudelleenkäytettävyyttä. Selainnäkymä on ohjelman käyttöliittymäkomponentti, jossa voidaan näyttää HTML-sivuja aivan kuten tavallisessa nettiselaimessa. Selainnäkymä ei ole täydellinen selain, koska se ei sisällä esimerkiksi osoiteriviä tai eteen- ja taakse-painikkeita. Ohjelmoija voi kuitenkin toteuttaa kyseiset toiminnot halutessaan itse. Koska selainnäkymä sisältää vain itse HTML-sivun, se on helppo sulauttaa ohjelman muiden käyttöliittymäkomponenttien joukkoon.

Internetin yleistymisen ansiosta web-teknologiat ovat nykyaikana hyvin laajasti tuettuja. Yhtä esimerkiksi HTML5:ttä ja JavaScriptiä käyttäen ohjelmoitua internetohjelmaa voidaan käyttää eri käyttöjärjestelmien omissa nettiselainohjelmissa ja monissa kolmannen osapuolen selaimissa. Monet selaimet käyttävät samaa selainmoottoria, mikä lisää yhteensopivuutta. Tätä ominaisuutta voidaan hyödyntää myös muissa tietokoneohjelmissa. Ohjelmaan voidaan lisätä niin sanottu selainnäkymä, jossa nettisivu tai -ohjelma voidaan näyttää aivan kuten nettiselaimessa. Selainnäkymät käyttävät myös samoja selainmoottoreita kuin itsenäiset selainohjelmat. Selainnäkymällä voidaan toteuttaa esimerkiksi ohjelman käyttöliittymä, jolloin se näyttää samalta kaikilla käyttöjärjestelmillä. Web-näkymän toiminnot ja tapahtumat voidaan kytkeä muuhun ohjelmaan niiden sisältämän vieraan funktion rajapinnan avulla. Näytettävä HTML-sisältö voidaan tuottaa dynaamisesti muussa ohjelmakoodissa, tuoda esimerkiksi tietokantataulusta tai linkittää suoraan internetistä. Näin voidaan hyödyntää valmiita web-teknologioihin pohjautuvia ohjelmakirjastoa myös muissa ohjelmointikielissä. Selainnäkymien käyttäminen käyttöliittymässä lisää ohjelman ulkoasun räätälöitävyyttä, koska selainnäkymässä näytettävän HTML:n ei tarvitse välttämättä olla ohjelmassa valmiina. Esimerkiksi internetistä ladattava HTML mahdollistaa ulkoasun muuttamista ilman, että ohjelmaa tarvitsee päivittää.

Kuva 5. AnyReader hyödyntää selainnäkymiä käyttöliittymässään.

Tutkitaan esimerkkinä Anygraaf Oy:n mobiliohjelmaa AnyReaderia. AnyReader on ohjelma sanoma- ja aikakausilehtien lukemiseen mobiililaitteessa. AnyReaderissa käytetään selainnäkymiä muun muassa käyttöliittymien toteutuksessa. Kuvassa 5 on eri laitteilla otettuja kuvakaappauksia AnyReaderin kirjastonäkymästä. Kirjastonäkymä on HTML-sivu, jota näytetään AnyReaderin sisällä selainnäkymässä, ja se on kytketty rajapinnan kautta muuhun ohjelmakoodiin. Samaa kirjastonäkymää käytetään jokaisessa laitteessa, mikä säästää ohjelmointityötä.

Eräs maksuton selainnäkymiä hyödyntävä ratkaisu on PhoneGap Build -niminen mobiiliohjelmistokehys [17]. PhoneGapilla sovellus ohjelmoidaan käyttäen HTML5:tä CSS3:a ja JavaScriptiä, jotka ovat tuettuja kaikissa mobiililaitteissa. Valmis ohjelma ei kuitenkaan ole web-sovellus, sillä se pakataan itsenäiseksi mobiiliohjelmaksi ja sitä voidaan jaella normaalisti mobiilialustojen ohjelmistokaupoissa. PhoneGap tarjoaa JavaScript-rajapinnan mobiililaitteiden ominaisuuksien, kuten kameran, GPS:n ja muiden sensorien sekä tiedostojärjestelmän käyttämiseen ohjelmassa.

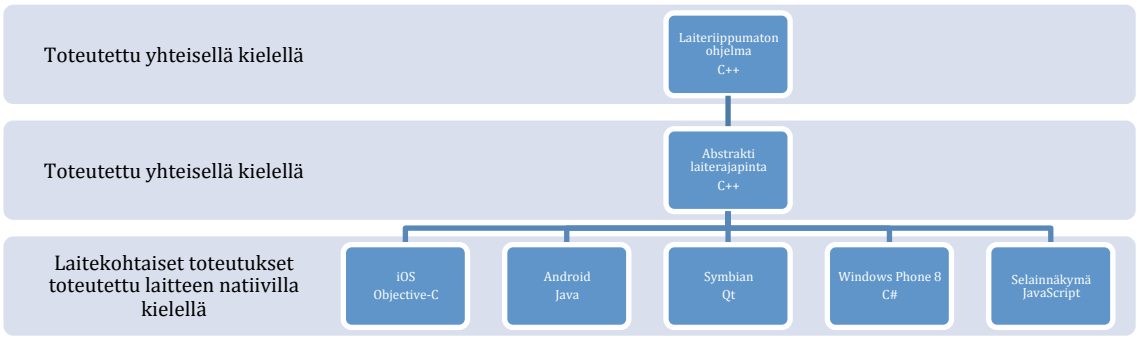


Kuva 6. PhoneGap Build mobiiliohjelmisto-kehyksen toimintaperiaate. [17]

Kuvassa 6 ohjelmoijan web-teknologioilla toteuttama sovellus paketoidaan PhoneGap Buildilla valmiiksi mobiiliohjelmiksi eri alustoille. Ohjelmistokehittäjän ei tarvitse luoda versioita erikseen jokaiselle laitteelle, vaan PhoneGap suorittaa tämän toimenpiteen. Selainnäkymät ja PhoneGap ovat helppoja tapoja toteuttaa monialustaisia sovelluksia, mutta pelkkä web-teknologioiden käyttö voi olla liian rajoittavaa esimerkiksi pelikehityksessä. Joustavampi ratkaisu on toteuttaa monialustasovellus ohjelmointikieliä yhdistämällä. Sovelluksen runko voidaan ohjemoida valittujen alustojen yhteisesti tukemalla kielellä kuten C++:lla ja laiterajapinnat alustan ensisijaisella ohjelmointikielellä (katso taulukko 4).

Yhdistelmäohjelmat

Ohjelmointikieliä yhdistämällä toteutettu monialustasovellus voidaan koostaa siten, että mahdollisimman suuri osa ohjelmasta ohjelmoidaan alustariippumattomasti kohteena olevien alustojen yhteisesti tukemalla ohjelmointikielellä. Alustakohtaiselta ohjelmakoodilta ei kuitenkaan voida täysin välttyä, sillä ohjelman mahdollisesti tarvitsemat järjestelmärajapinnat on toteutettu alustakohtaisella ohjelmointikielellä. Tällä tavalla toteutettu monialustasovellus on siis työläämpi ohjelmoitava kuin selainnäkymillä toteutettu monialustaisuus.



Kuva 7. Monialustaisen mobiilisovelluksen arkkitehtuuri.

Kuvassa 7on esitetty hybridiohjelmoinnin periaatetta hyödyntävän monialustaisen mobiiliohjelman arkkitehtuuri. Ydinohjelma on ohjelmoitu C++:lla eikä siinä saa olla mitään laiteriippuvaista ohjelmakoodia. Laiteriippumattomuus mahdollistaa ydinohjelman kääntämisen jokaiselle ohjelmistoalustalle. Ohjelman ja laitteen välille ohjelmoidaan virtuaalinen laite, joka abstrahoi laitteen järjestelmäkutsurajapinnan. Ohjelma voi käyttää laitteen ominaisuuksia kuten syöte- ja tulostevirtoja, internetyhteyttä ja tiedostojärjestelmää abstraktin laiterajapinnan kautta. Yksinkertaisimmillaan virtuaalilaite on puhtaasti virtuaalinen luokka, jonka jäsenfunktiot toteutetaan laitekohtaisissa ohjelmakoodeissa. Mikään ei estä käyttämästä myös tällaisessa ratkaisussa selainnäkymiä käyttöliittymän räätälöintiin.

Abstraktin laiterajapinnan monimutkaisuus ja alustakohtaisen ohjelmakoodin määrä riippuvat siitä, kuinka monia eri laiterajapinnan toimintoja ohjelma käyttää. Esimerkiksi mobiiliohjelma, joka käyttää matkapuhelimen kameraa, GPS:ää, liikesensoreita, internetiä ja laitteistokiihdytettyä piirtämistä, vaatii paljon laitekohtaista ohjelmakoodia. Tilanteesta riippuen ongelma voidaan ratkaista tekemällä laiterajapinnasta vain osittain virtuaalinen. Osa laiterajapinnan toiminnoista voidaan jollakin kaikkien laitteiden yhteisesti tukemalla ohjelmakirjastolla. Jos esimerkiksi ohjelmointikieli on C tai C++, voidaan käyttää C-kielisiä kirjastoja esimerkiksi internetyhteyksiin (libcurl [18]) ja grafiikan piirtämiseen (OpenGL [19]). Aiemmin mainittu mobiililukuohjelma AnyReader on toteutettu monialustaisesti tällä periaatteella.

3.4 Ohjelmalaajennukset

Työpöytäsovellukset on useimmiten ohjelmoitu matalan tason ohjelmointikielellä kuten C:llä tai C++:lla niillä tuotettujen ohjelmien suorituskykyisyyden vuoksi. Suorituskykyisyyden varjopuolena näillä kielillä kehitetyt ohjelmat eivät ole kovin joustavia muutoksille, kun ohjelmakoodi on kerran käännetty. Pienikin muutos ohjelmaan vaatii suhteellisen raskaan käännösprosessin ajamista uudelleen. Tästä on haittaa esimerkiksi tilanteessa, jossa halutaan ohjelman olevan muokattavissa tai laajennettavissa muiden ohjelmistokehittäjien tai loppukäyttäjien toimesta. C- ja C++-ohjelmiin voi lisätä dynaamisesti C:llä tai C++:lla kirjoitettuja ohjelmalaajennuksia, mutta niiden mukaan liittäminen vaatii ohjelman kääntämisen ja linkkaamisen uudelleen, ja niiden virheenetsintä voi olla hankalaa.

Parempi ratkaisu on lisätä ohjelmaan tuki jollekin skriptikielelle. Yleisimmät tällaiseen tarkoitukseen sopivat skriptikielet ovat Python ja Lua sekä Visual Basic Microsoftin ohjelmissa [20]. Ohjelmakoodiin voidaan upottaa valitun skriptikielen tulkki, joka tulkitsee ja suorittaa skriptikielistä ohjelmakoodia ajon aikana. Ohjelmointikielien välinen kommunikaatio tapahtuu vieraan funktion rajapinnan kautta, ja kommunikaatio voi olla kaksisuuntaista. C- tai C++-koodista voidaan suorittaa mitä tahansa skriptikielistä koodia ja Lua- tai Python-koodista voidaan suorittaa ohjelmoijan vieraan funktion rajapinnassa määrittelemiä C- tai C++-funktioita. Koska ohjelmoijalla on valta määritellä, mitä ydinohjelman funktioita skriptikielestä voidaan kutsua, menetelmä sopii hyvin myös loppukäyttäjien toteuttamiin ohjelmalaajennuksiin. Jos skripteistä ei päästä käsiksi ohjelman kannalta kriittisiin toimintoihin, ei ole riskiä että loppukäyttäjän määrittelemä skripti sotkisi ohjelman toimintaa.

Skriptilaajennuksia voidaan käyttää minkä tahansa ohjelman toiminnallisuuden yhteydessä, jossa mahdollisuudesta toiminnon räätälöimiseen koetaan olevan hyötyä. Esimerkkejä skriptikielisiä ohjelmalaajennuksista ovat esimerkiksi selainohjelmien laajennukset [21] ja tietokonepelien modit, eli käyttäjien luomat muunnokset alkuperäisestä pelistä. Esimerkiksi nettiroolipeli World of Warcraftissa pelaaja voi muokata pelin käyttöliittymää XML-kuvauskielellä ja Lua-skriptikielellä [22].

Kuva 8. World of Warcraftin alkuperäinen käyttöliittymä. [23]

World of Warcraftin alkuperäisessä käyttöliittymässä (kuva 8) käyttöliittymäkomponentit on sijoiteltu ympäri ruutua. Komponentit ovat kaukana toisistaan, joten niiden hahmottaminen ja käyttäminen saattaa olla hankalaa. Kuvassa 9 pelaaja on muokannut Lualla käyttöliittymän itselleen sopivaksi. Käyttöliittymäkomponentit ovat tiiviimmässä paketissa ja itse pelialueelle jää enemmän tilaa ruudulla.

Kuva 9. Lua-kielellä muokattu käyttöliittymä World of Warcraftissa. [24]

Python- tai Lua-tuen lisääminen C-ohjelmaan on verrattain yksinkertaista, sillä varsinkin Lua on kehitetty tällaista käyttöä silmällä pitäen [25]. Molemmista kielistä löytyy vakiona ohjelmointirajapinta C-kielelle sekä useita kolmannen osapuolen ohjelmointirajapintoja C++:lle.

4 Esimerkkisovellus

Osana insinöörityötä toteutettiin Anygraaf Oy:lle tietokantadataa visualisoiva ohjelmamoduuli, jossa hyödynnetään insinöörityössä havaittuja hybridiohjelmoinnin etuja. Ohjelmamoduulin tuotantoprosessi noudatti löyhästi asteittain kehittyvän tuotantoprosessin mallia. Ohjelmamoduulin kuvaus oli lähtötilanteessa hyvin yleisluontoinen, joten sen määrittely ja suunnittelu tarkentuivat moduulia toteutettaessa ja käytössä olevien työkalujen mahdollisuuksien selvetessä. Asteittain kehittyvässä ohjelmistotuotantoprosessissa tuotetaan prototyyppejä ohjelmasta, ja prototyyppien perusteella muutetaan ohjelman määrittelyä haluttuun suuntaan, jonka jälkeen suunnitellaan ja toteutetaan seuraava prototyyppi [26].

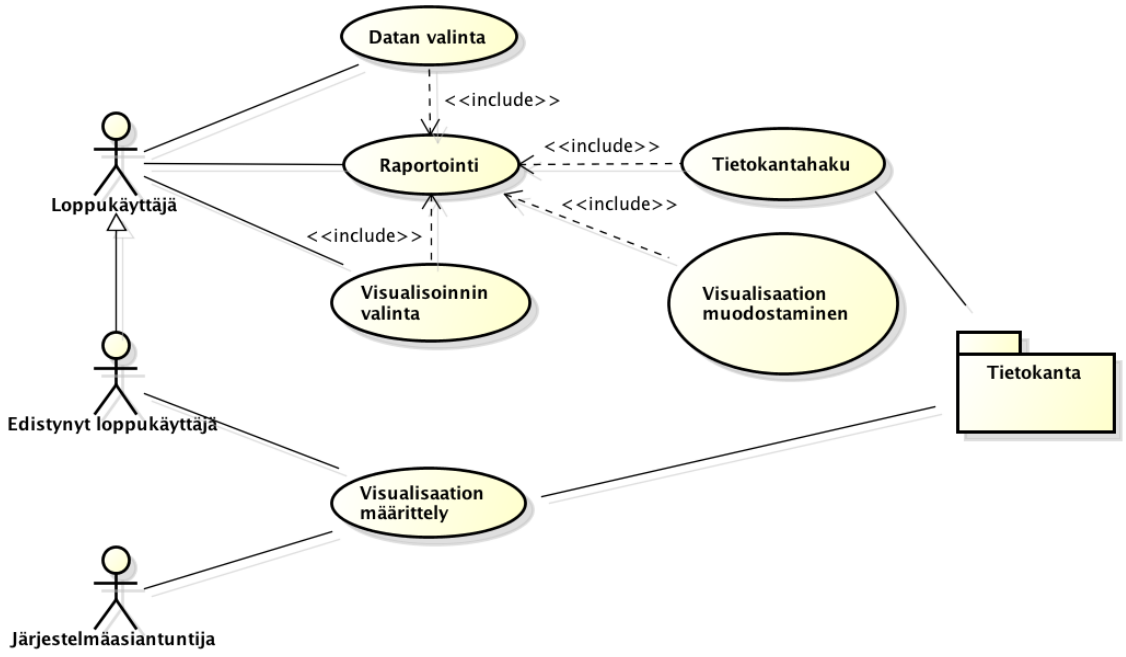
Ohjelmamoduuli kehitettiin osaksi Anygraafin Neo-toimitusjärjestelmää. Neoa käytetään sekä henkilöresurssien että sisällöntuotannon hallintaan toimitustyössä [27]. Neossa on otettu huomioon nykyaikainen monikanavajulkaiseminen. Artikkelin kuva- ja tekstisisältö voidaan Neolla julkaista helposti niin printtilehdessä, nettisivuilla kuin tablet-tietokoneessa ja kännykässäkin. Artikkelien luokitteluun ja metatietoihin on panostettu. Metatiedot tallennetaan tietokantaan ja näin ollen niitä voidaan hyödyntää esimerkiksi yrityksen sisäisessä seurannassa. Tietokantadataa visualisoimalla voidaan esimerkiksi seurata miten artikkelien määrät ovat aiheittain kehittyneet ajan kuluessa. Tiedon perusteella yrityksen voi olla helpompi jakaa resursseja eri aiheiden toimitustyöhön tulevaisuudessa.

4.1 Määrittely

Idea projektille syntyi yrityksen omassa kehitystiimissä, joten projektilla ei ollut varsinaista asiakkaan roolia. Tästä johtuen projektin määrittely oli melko vapaata ja muotoutui projektin edetessä. Aluksi oli vain tiedossa tavoite kyetä visualisoimaan tietokantadataa. Tietokanta muodostuu kokoelmasta tietoja, joilla on yhteys toisiinsa [28] ja tieto-

kantadatan luonteesta johtuen siitä voidaan helposti muodostaa ainakin taulukoita sekä graafisia kuvaajia. Koska haluttu tapa visualisoida tietoa riippuu hyvin paljon käyttötapauksesta, tavan määritellä erilaisia taulukoita ja kuvaajia täytyy olla joustava. Toiminnon tulisi kuitenkin olla sellainen, että sitä osaa käyttää tyypillinen Neon loppukäyttäjä, joka ei välttämättä ole tietotekniikan asiantuntija. Toiminnon tulisi siis pohjautua mahdollisimman paljon helppoihin esimääriteltyihin valintoihin.

Neossa oli jo ennen ohjelmamoduulin luomista raportointitoiminto, jossa edistynyt loppukäyttäjä tai järjestelmän ylläpitäjä voi luoda käsityönä erilaisia raportteja tietokantadatasta. Koska projektin vaatimuksena oli turhan kehitystyön välttämiseksi hyödyntää mahdollisuuksien mukaan jo olemassa olevaa ohjelmakoodia, edellä kuvattu Neon toiminto otettiin tietokantahakujen ja käyttöliittymän osalta alustaksi myös tähän projektiin. Projektin karkeaksi määritelmäksi alkoi siis muotoutua loppukäyttäjälle helppokäyttöinen toiminto jolla voidaan muodostaa tietokantadatasta erilaisia taulukoita sekä graafisia kuvaajia. Taulukoiden ja kuvaajien määrittely tulee olla joustavaa ja toteutuksessa tulisi hyödyntää mahdollisuuksien mukaan valmiita toteutuksia.



Kuva 10. Ohjelmamoduulin käyttötapauskaavio

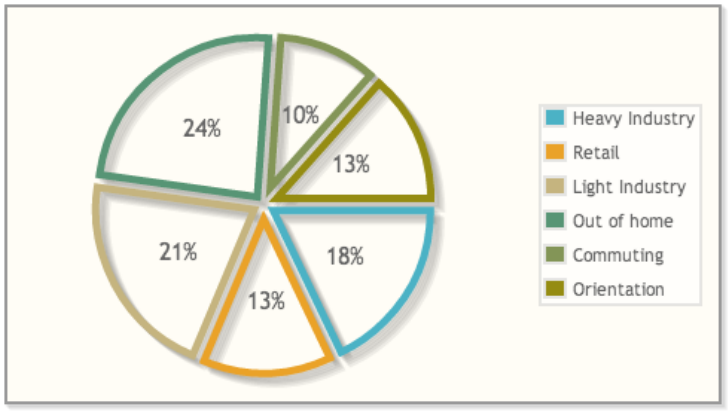
Ohjelmamoduulin määrittelyn perusteella muodostetusta käyttötapauskaaviossa (kuva 10) edistynyt loppukäyttäjä ja järjestelmäasiantuntija voivat määritellä tietokantaan eri-

laisia visualisaatioita tietokannasta muodostettaville raporteille. Loppukäyttäjä voi muodostaa raportteja valitsemalla halutun datan sekä jonkin tietokantaan määritellyistä visualisaatiomäärittelyistä. Uuden ohjelmamoduulin tehtävä on visualisaation muodostaminen.

4.2 Suunnittelu

Esitutkimus

Suunnittelun ensimmäinen vaihe oli tehdä esitutkimusta mahdollisista valmiista avoimen lähdekoodin visualisaatioratkaisuista. Määrittelyn perusteella visualisaatio tulisi toteuttaa jollakin skriptikielellä mahdollisimman joustavan muokattavuuden saavuttamiseksi. Internethaulla selvisi, että tarjolla on paljon JavaScriptillä toteutettuja kuvaajien piirtämiseen tarkoitettuja kirjastoja [29]. Monet kirjastoista ovat liitännäisiä jQuery JavaScript -kirjastoon. jQuery on helpon syntaksinsa ansiosta suosittu avoimen lähdekoodin kirjasto, joka on suunniteltu siten, että se toimii kaikissa selaimissa [30]. Lisenssiehdoiltaan ja ominaisuuksiltaan sopivaksi kirjastoksi osoittautui jqPlot. Erityisen kiinnostava ja muista erottuva ominaisuus on mahdollisuus tallentaa kuvaajat kuvatiedostoina. Tämä avaa mahdollisuuden luoda kuvaajia helposti myös web-ympäristön ulkopuolelle.



Kuva 11. jqPlotilla luotu esimerkkikuvaaja. [31]

jqPlotilla on mahdollista luoda viiva-, pylväs- ja ympyrädiagrammeja (katso kuva 11). Kuvaajiin voidaan lisätä interaktiota, kuten datapisteen huomiokorostus hiiren osoitta-

essa sitä ja kuvaajan akselien skaalauksen muuttaminen dynaamisesti niin, että jotakin kuvaajan kohtaa voidaan tarkastella lähemmin. Teknisesti jqPlotin käyttäminen on hyvin yksinkertaista: vakiomuotoiselle funktiolle annetaan parametreina vain HTML-dokumentin osa, johon kuvaaja piirretään, datasarja, josta kuvaaja muodostetaan, sekä kokoelma kuvaajan ulkoasun määritelmistä (katso koodiesimerkki 1).

```
$.jqplot('kuvaaja', [[[1,2], [3,5], [6, 10]]]);
```

Koodiesimerkki 1. Kuvaajan muodostaminen jqPlotilla.

Tavallisten taulukoiden luomiseen ei välttämättä tarvita mitään JavaScript-koodia, sillä HTML:ssä on sisäänrakennettuna taulukkoelementit. JavaScriptiä voidaan kuitenkin käyttää lisäämään taulukoihin toiminnallisuutta. Tablesorter on jqPlotin tapaan jQuery-liitännäinen, jolla voidaan lisätä HTML-taulukkoon rivien järjestämistoiminto halutun sarakkeen tai sarakkeiden mukaan [32].

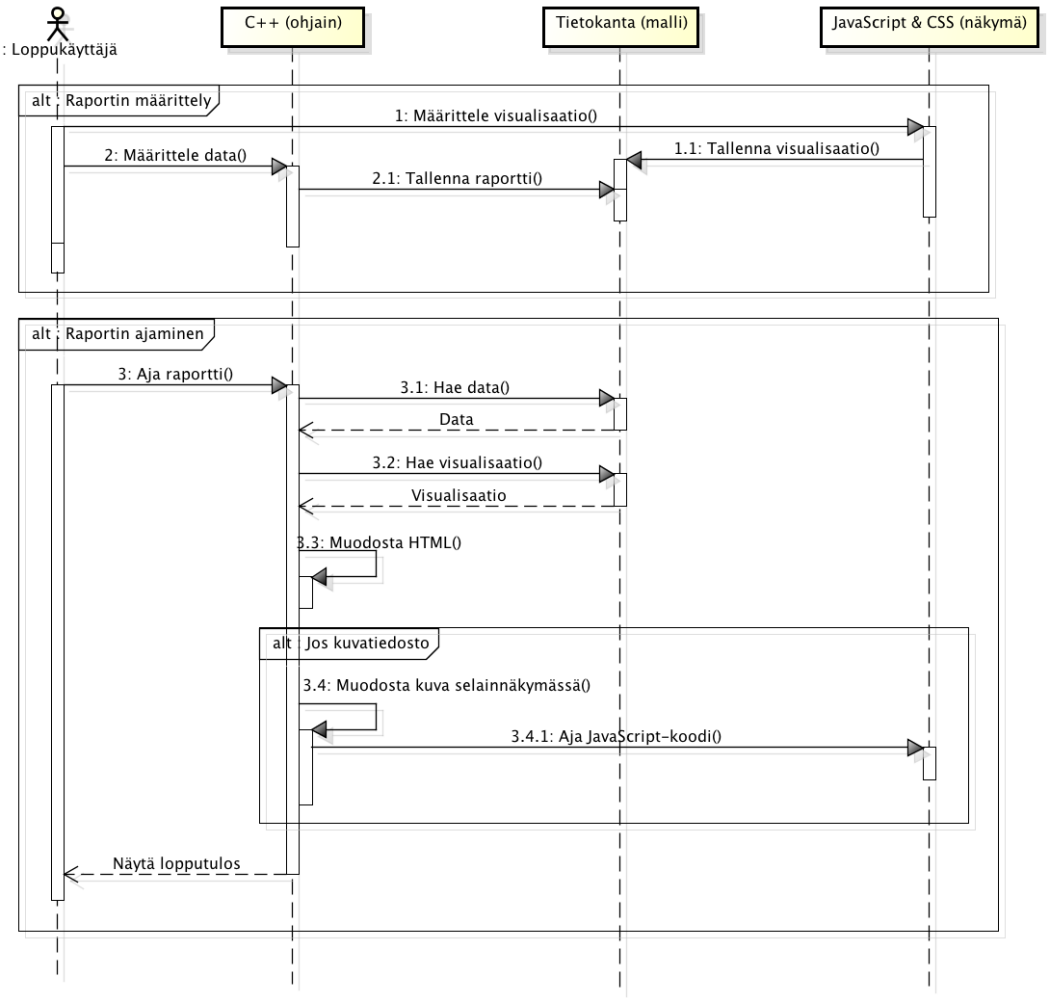
Arkkitehtuuri

Ohjelmamoduulin arkkitehtuurin suunnitteluun vaikutti merkittävästi koodiympäristö, jossa sitä käytetään. Määrittelyssä tehtiin päätös sovittaa uusi ohjelmamoduuli Neossa jo olevan raportointitoiminnon yhteyteen. Uuden ohjelmamoduulin tulisi siis sopia rajapinnaltaan yhteen vanhan raportoinnin kanssa, jolloin uusi ohjelmamoduuli aiheuttaa mahdollisimman vähän muutoksia sitä kutsuvaan ohjelmakoodiin. Voidaan ajatella, että uusi ohjelmamoduuli on uusi tietokantadatan näkymä model-view-controller ohjelmistoarkkitehtuurimallissa. Koska määrittelyn perusteella ohjelmamoduulista pitäisi tulla mahdollisimman joustava, tämä täytyy ottaa huomioon arkkitehtuurin suunnittelussa. Koska visualisointiin valittujen ohjelmakirjastojen ohjelmointikieli on JavaScript ja muun ohjelman ohjelmointikieli on C++, uusi ohjelmamoduuli jakautuu C++-osuuteen ja JavaScript-osuuteen.

Insinöörityössä tehdyn tutkimuksen tulosten mukaan JavaScript on joustavampi ohjelmointikieli kuin C++. On siis mielekästä toteuttaa mahdollisimman iso osuus ohjelmamoduulista JavaScriptillä. Minimissään C++-osuus toimii eräänlaisena ohjaimena, joka näyttää käyttöliittymän valintojen tekemiseen, muokkaa haetun tietokantadatan JavaScriptin ymmärtämään muotoon, luo HTML-sivun, johon JavaScript-osuus liitetään, ja näyttää sen ohjelman käyttäjälle. Yksinkertaisin ratkaisu on näyttää HTML-sivu käyt-

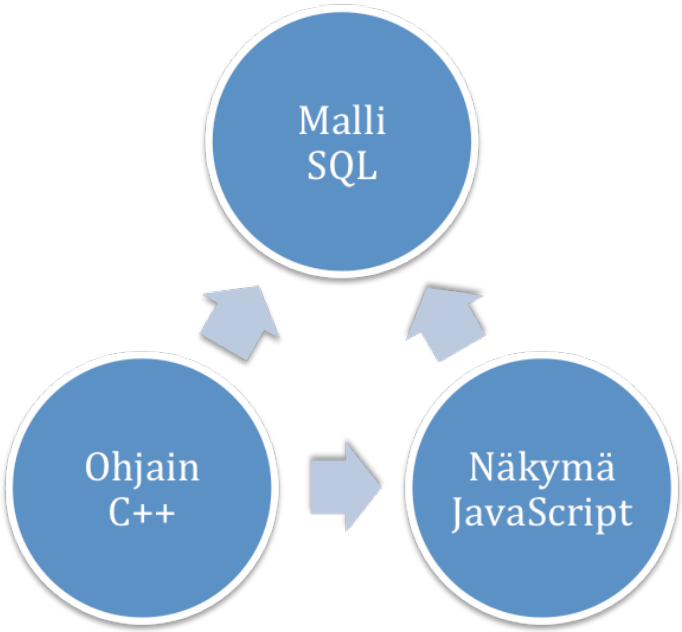
töjärjestelmän oletusselaimessa. Jos kuvaaja halutaan tallentaa kuvatiedostona HTML:n sijaan, täytyy HTML-sivu kuitenkin avata johonkin, jotta kuva voidaan muodostaa. Tätä varten ohjelmamoduuli voi avata HTML-sivun selainnäkymässä, jota ei kuitenkaan näytetä käyttäjälle. Selainnäkymä tallentaa kuvaajan kuvatiedostoon, jota voidaan katsella ohjelman sisäisessä esikatseluikkunassa.

JavaScript-osuuden tehtävä on määritellä ja muodostaa visualisaatio annetusta datasta. Tällainen arkkitehtuuri maksimoi järjestelmän joustavuuden. Vaikka suunnittelu tehtiin ajatellen valittuja JavaScript-kirjastoja, arkkitehtuuri mahdollistaa muidenkin JavaScript-kirjastojen käytön, sillä tietokantadatan lopullinen käsittely tapahtuu vasta määritellyssä JavaScriptissä.



Kuva 12. Ohjelmamoduulin sekvenssikaavio.

Kuvassa 12 olevassa sekvenssikaaviossa hahmotetaan ohjelmamoduulin arkkitehtuuria sekä toimintaa. Ohjelmamoduuli noudattaa löyhästi malli-näkymä-ohjain-ohjelmistoarkkitehtuurityyliä. Puhtaassa malli-näkymä-ohjain-arkkitehtuurissa malli edustaa järjestelmän dataa, näkymä määrittää käyttöliittymän ulkoasun ja ohjain vastaanottaa käyttäjän syötteen sekä muuttaa mallia sekä näkymää sen mukaisesti. Kuvasta 13 selviää malli-näkymä-ohjain-arkkitehtuurin riippuvuussuhteet. Malli ei riipu näkymästä eikä ohjaimesta, joten samasta mallista voidaan luoda useita näkymiä. Ohjaimen riippuvuus näkymästä on myös vähäinen.



Kuva 13. Malli-näkymä-ohjain-arkkitehtuuri.

Mallina voidaan ajatella olevan tietokanta, johon on tallennettu raportin määrittely. Karkeasti määrittely sisältää SQL-lauseen jolla data tuotetaan, JavaScriptin jolla visualisaatio tuotetaan sekä CSS:n jolla visualisaation ulkoasu määritellään. JavaScript ja CSS siis muodostavat näkymän ja C++ toimii ohjaimena mallin ja näkymän välissä.

4.3 Toteutus ja jatkokehitys

Tietokanta ja käyttöliittymä

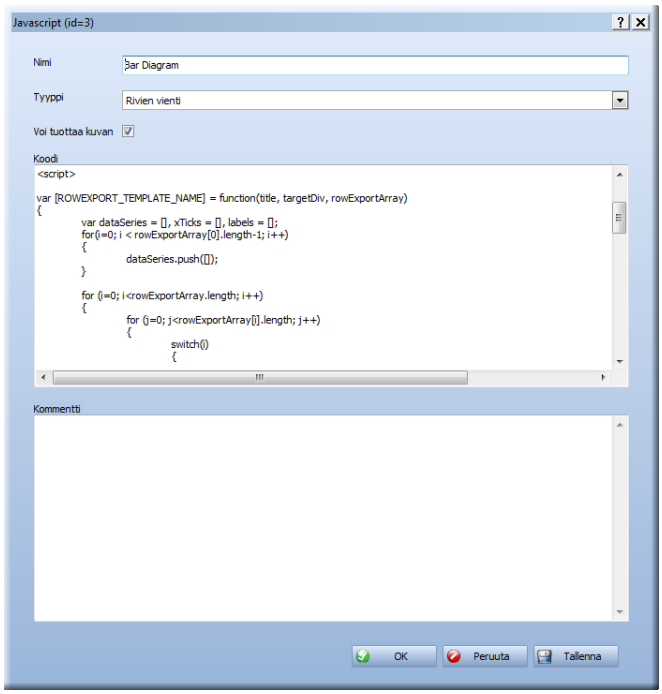
Ohjelmamoduulin toteuttaminen aloitettiin luomalla tietokantaan taulut, joihin määritellyt JavaScriptit ja CSS:t voidaan tallentaa. Anygraafin ohjelmistokehyksellä onnistuu tieto-

kantataulujen määrittely suoraan ohjelmakoodissa. Ohjelmistokehys tekee määritellyn taulun tietokantaan ja luo käyttöliittymän tauluun liittyvän datan käsittelyä varten.

```
BEGIN_DB_CLASS()
class JavaScript
{
public:
        JavaScript();
        ~JavaSrript();
private:
        int id;
        string nimi;
        selection tyyppi;
        boolean voi_olla_kuva
        string koodi;
        string kommentti;
};
END_DB_CLASS()
```

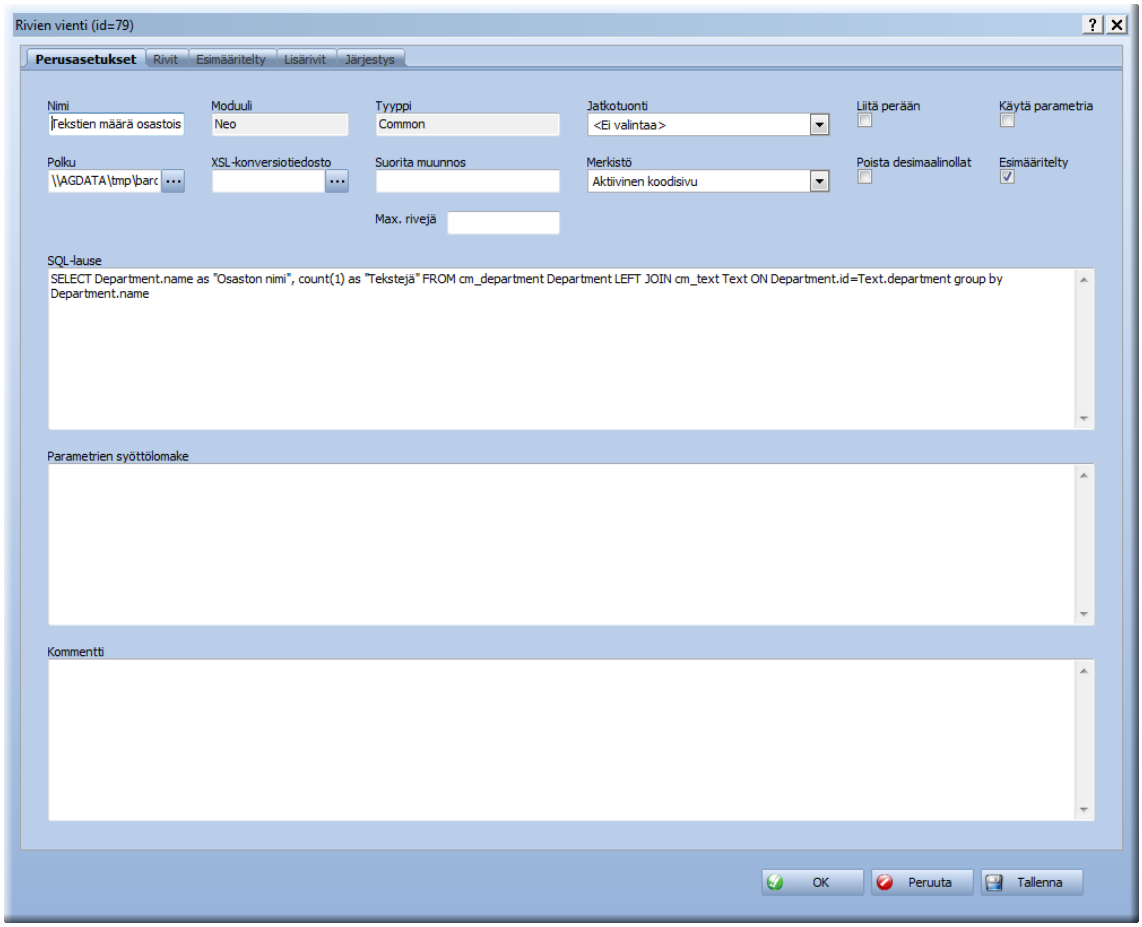
Koodiesimerkki 2. JavaScript-luokan määrittely.

Koodiesimerkissä 2 on ohjelmamoduuliin liittyvän JavaScript-luokan yksinkertaistettu määrittely C++-tyylisenä pseudokoodina. Alun ja lopun makroilla kerrotaan ohjelmisto-kehykselle että luokasta halutaan luoda tietokantataulu. Luokkaa voidaan käsitellä normaalin C++-luokan tapaan ja sen tila voidaan halutessa tallentaa tietokantaan tai ladata sieltä.



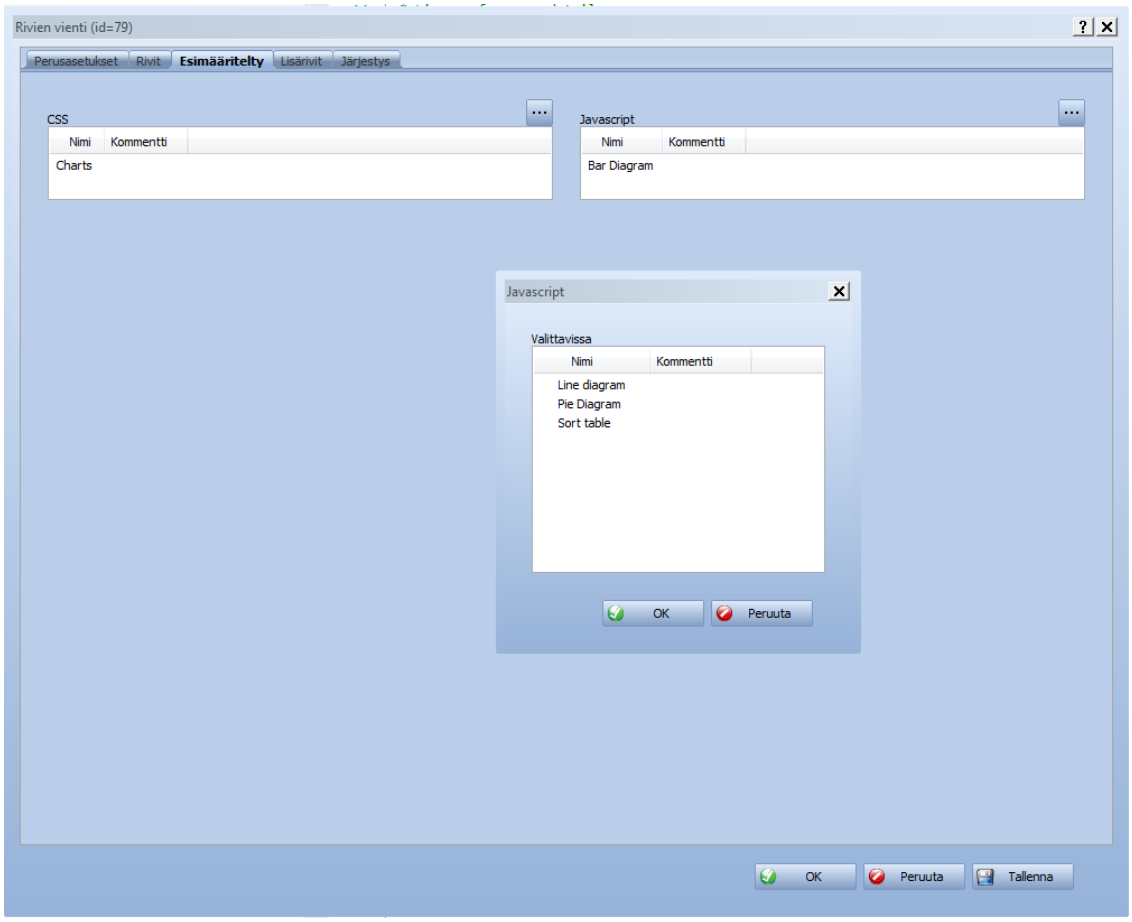
Kuva 14. JavaScript-tietokantataulun editori.

Kuvassa 14 ohjelman käyttäjä editoi JavaScript-tietokantataulun tietuetta, jonka id on 3. Käyttäjä voi muokata ohjelmistokehyksen luomassa editorissa kaikkia koodiesimerkissä 2 määriteltyjä muuttujia. Luokkaan lisättiin myös valinta, jolla määritellään, kykeneekö kyseinen JavaScript luomaan kuvaajasta kuvatiedoston. Käyttäjä ei voi tehdä epäsopivia yhdistelmiä valitsemalla tallennusmuodoksi kuvatiedoston, jos kuvatiedoston luominen ei ole mahdollista.



Kuva 15. Raportin määrittely

Raportinmäärittelyn käyttöliittymään (kuva 15) lisättiin valinta, halutaanko raportin muodostuksessa käyttää esimääriteltyä visualisaatiota. Raportin määrittelijän on käsin muodostettava SQL-lause, jolla raportin data haetaan tietokannasta. Raporttien määrittely vaatii siis jonkin verran teknistä tietämystä sekä käytettävän tietokannan tuntemusta. Määrittelijä valitsee ”Polku”-kentässä, missä muodossa ja mihin raportti tallennetaan. Jos polku jätetään tyhjäksi, käyttäjältä kysytään polkua raportin ajovaiheessa.

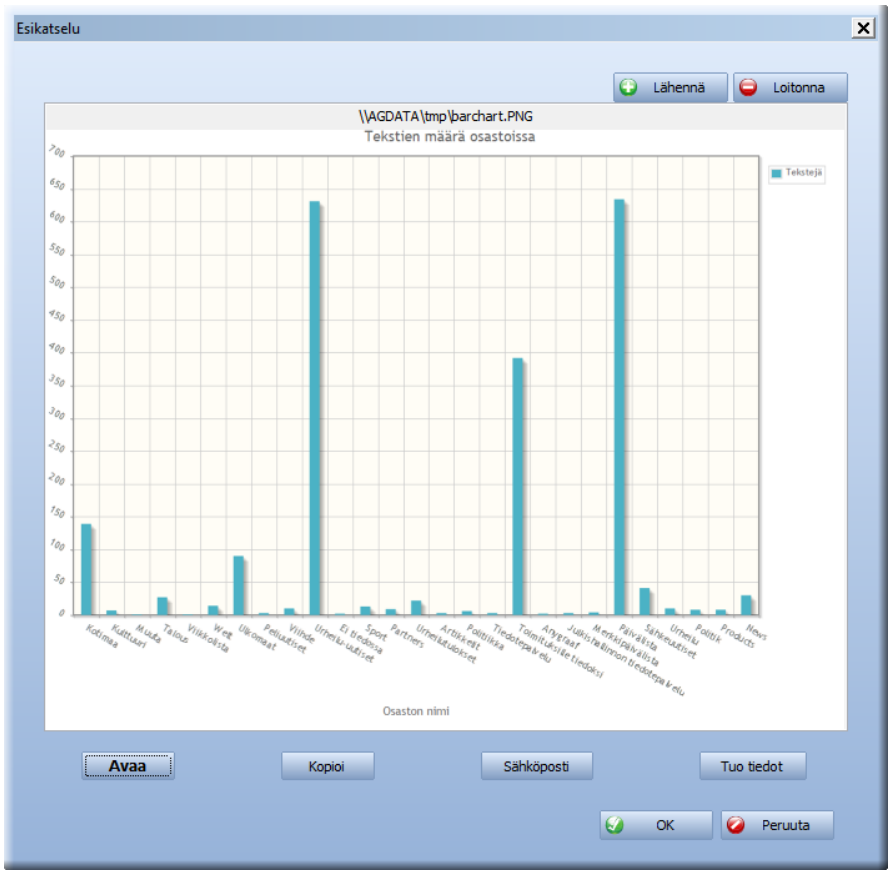


Kuva 16. Esimäärittelyjen valitseminen raportoinnissa.

Kuvassa 16 näkyvään Neon raportoinnin käyttöliittymään lisättiin valinnat esimääritellyn raportin muodostavalle CSS:lle ja JavaScriptille. Valintojen lisääminen tehtiin myös raportoinnin tietokantataulun määrittelyä muokkaamalla. Koska raportin data muodostetaan vapaamuotoisella SQL-lauseella tehdyllä tietokantahaulla, vastuu datan ja esimääritellyn visualisoinnin yhteensopivuudesta jää raportin määrittelijälle. Tämä on toiminnon helppokäyttöisyyttä huonontava tekijä, jota voitaisiin parantaa toimintoa jatkokehittämällä. Huomattava parannus olisi SQL-lauseen muodostin, jolla käyttäjä voisi määritellä visualisoitavan datan tekemällä valintoja. Tällöin raportin määrittelijän ei välttämättä tarvitsisi tuntea tietokannan rakennetta etukäteen. Raportin muodostin voisi myös näyttää koko ajan reaaliaikaisen esikatselukuvan kuvaajasta.

Kuvatiedostojen esikatseluikkunaan lisättiin toimintoja, joita raportin ajaja saattaa haluta tehdä kuvaajalle. Tiedoston voi avata käyttöjärjestelmän oletuskuvankatseluohjelmassa, kopioida leikepöydälle, lähettää sähköpostin liitteenä tai viedä tietokantaan. Kuvassa 15 Neon käyttäjä on ajanut raportin tekstien määrästä osastoittain. Raporttia

visualisoidaan pylväsdiagrammilla joka on tallennettu kuvatiedostoon. Liittessä 1 on esitetty lisää esimerkkejä ohjelmamoduulilla tuotetuista kuvaajista ja taulukoista.



Kuva 17. Tuotetun kuvaajan esikatseluikkuna.

Ohjelmakoodi ja vieraan funktion rajapinta

Ohjelmamoduulissa vieraan funktion rajapintana on Anygraafin ohjelmakirjastoon toteutettu selainnäkymä CAgWebControl. Selainnäkymän selainmoottorina toimii Mozilla Foundationin luoma XULRunner, jota hyödyntävät myös Mozillan omat selaimet sekä Googlen Chrome-selain [33]. Selainnäkymä itsessään on ohjelmoitu C++:lla, mutta se kykenee suorittamaan JavaScript-kielisiä ohjelmia luoden vieraan funktion rajapinnan ohjelmamoduuliin. Rajapintaan ei toteutettu minkäänlaista virheenkäsittelyä. Virheenkäsittelyn toteuttaminen voisi olla hyvä jatkokehitysidea ohjelmamoduulille.

Ohjelmamoduulin toiminta ohjelmoitiin kuvan 12 sekvenssikaaviossa olevan pseudokoodin mukaisesti. C++-ohjelma generoi HTML-sivun ja liittää siihen valitut CSS:t ja JavaScriptit. Tietokannasta haetut rivit muutetaan JavaScript-taulukoksi ja tallennetaan

HTML:ään JavaScript-muuttujana. JavaScriptissä taulukko käsitellään niin, että siitä voidaan muodostaa kuvaaja. Koska kaikki visualisaatiokohtainen datan käsittely tapahtuu JavaScript-määrityksissä, C++-ohjelma voi tallentaa datan vakiomuodossa eikä siellä tarvitse käsitellä eri kuvaajatyyppien tarpeita. Lopuksi valmis HTML-dokumentti tallennetaan valittuun tiedostoon ja avataan esikatseltavaksi joko käyttöjärjestelmän oletusselaimeen tai kuvassa 17 olevaan esikatseluikkunaan, jos valittu tiedosto on kuvatiedosto.

Helppokäyttöisyyden ja virheenkäsittelyn lisäksi ohjelmamoduulia voisi jatkokehittää esimerkiksi luomalla mahdollisuuden ajastaa raporttien ajaminen. Jos raportteja käytetään esimerkiksi yrityksen sisäiseen seurantaan, olisi mielekästä, että kuvaajista olisi aina ajantasainen versio saatavissa ilman, että raportti tarvitsee käydä manuaalisesti ajamassa. Lukuun ottamatta jatkokehitysmahdollisuuksina lueteltuja puutteita ohjelmamoduulin toteutus onnistui hyvin. Hybridiohjelmoinnin seurauksena JavaScriptillä tehty ohjelmamoduuli voitiin valjastaa C++-käyttöön. Kuten johdannossa mainittiin, ohjelmamoduuli liittyi myös viestintäalan Next Media -projektiin, johon Anygraaf Oy osallistui. Ohjelmamoduulin prototyyppiä esiteltiin projektin tulosseminaarissa [34].

5 Yhteenveto

Insinöörityö koostui tutkimustyöstä, jonka lopputuloksista saa hyvän yleiskuvan, miten tietokoneohjelma voidaan koostaa eri ohjelmointikielillä ohjelmoiduista osasista ja mitä etuja kyseisillä tekniikoilla voidaan saavuttaa. Tutkimuksesta selvisi, että usein paras mahdollinen ohjelma ei synny käyttämällä vain yhtä ohjelmointikieltä. Ohjelmoijalle on hyödyllistä tuntea erilaisia ohjelmointikieliä ja niiden heikkouksia ja vahvuuksia, jotta ohjelmoija osaa valita oikean ohjelmointikielen jokaiseen tilanteeseen.

Työssä johdatettiin lukija hybridiohjelmointiin esittelemällä ensin erilaisten ohjelmointikielten ominaisuuksia. Havaittiin, että parhaimmillaan ohjelmointikieliä voidaan yhdistää siten, että toinen ohjelmointikieli paikkaa vahvuuksillaan toisen kielen heikkouden. Esimerkiksi tehokkaita ohjelmia tuottavalla C-kielellä tehdyn ohjelman toimintaa on vaikea muuttaa ohjelman ollessa käynnissä, mutta sitä voidaan paikata jollakin ajon aikana tulkittavalla kielellä kuten Lualla. Hybridiohjelmoinnista havaittiin olevan hyötyä muissakin vastaavissa tilanteissa, joissa yhdellä ohjelmointikielellä tehtävä kehitys ei riitä saavuttamaan haluttua lopputulosta. Hyvä esimerkki on nykypäivän ohjelmisto-

maailmassa yhä kasvavassa roolissa oleva mobiilikehitys. Yhdellä ohjelmointikielellä tehtyä ohjelmaa voidaan käyttää vain yhdessä laitteessa, kun taas hyvin toteutettua hybridiohjelmaa voidaan käyttää monissa laitteissa.

Osana insinöörityötä toteutettiin Anygraaf Oy:lle tietokantadataa visualisoiva ohjelmamoduuli hyödyntäen tutkimuksen tuloksia. Ohjelmamoduuli ohjelmoitiin yhdistämällä C++:aa ja JavaScriptiä tutkimustyössä tutuksi tulleella selainnäkymällä. Ohjelmamoduulin ja tutkimustyön tekeminen oli opettavaista sillä suurin osa asioista oli minulle uusia. Toivon insinöörityön tarjoavan lukijalle uusia näkökulmia työssä käsiteltyjen ohjelmistotekniikan ongelmien ratkaisuun.

Lähteet

1 Next Media. Verkkodokumentti. Tivit Oy. 2013. <http://www.nextmedia.fi/573139/fi/read/Mediakokemus_muuttuu_digitaaliseksi?history=573139>. Luettu 17.3.2013.

2 Luettelo ohjelmointikielistä. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Luettelo_ohjelmointikielist%C3%A4>. Luettu 17.3.2013.

3 Categorial list of programming languages. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Categorical_list_of_programming_languages>. Luettu 17.3.2013.

4 Programming paradigm. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Programming_paradigm>. Luettu17.3.2013.

5 Olio-ohjelmointi. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Olio-ohjelmointi>. Luettu 23.3.2013.

6 Procedural programming. Verkodokumentti. Chipkidz. 2009. <http://chipkidz.wordpress.com/2009/08/07/procedural-programming/>. Luettu 23.3.2013.

7 Mischook, Stefan. 2005. Scripting vs. Programming: is there a difference? Verkkodokumentti. <http://www.killersites.com/blog/2005/scripting-vs-programming-is-there-a-difference/>. Luettu 24.3.2013.

8 High-level programming language. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/High-level_programming_language>. Luettu 24.3.2013.

9 Advantages of Weak Typing over Strong Typing. Verkkodokumentti. Stackoverflow. 2011. <http://stackoverflow.com/questions/6944522/advantages-of-weak-typing-over-strong-typing>. Luettu 24.3.2013.

10 Foreign function interface. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Foreign_function_interface>. Luettu 26.3.2013.

11 Wrapper library. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Wrapper_library>. Luettu 26.3.2013.

12 Java native interface. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Java_native_interface>. Luettu 26.3.2013.

13 SWIG. Verkkodokumentti. SWIG. 2013. <http://www.swig.org/>. Luettu 29.3.2013.

14 SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++. Verkkodokumentti. David M. Beazley. 1996. <http://www.swig.org/papers/Tcl96/tcl96.html>. Luettu 7.4.2013.

15 Language binding. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Language_binding>. Luettu 26.3.2013.

16 JTC1/SC22/WG11 – Binding Techniques. Verkkodokumentti. ISO. 2011. <http://www.open-std.org/JTC1/SC22/WG11/>. Luettu 30.3.2013.

17 PhoneGap Build. Verkkodokumentti. PhoneGap. 2013. <http://phonegap.com/>. Luettu 1.4.2013.

18 cURL. Verkkodokumentti. cURL. 2013. <http://curl.haxx.se/>. Luettu 8.4.2013.

19 OpenGL. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/OpenGL>. Luettu 8.4.2013.

20 Extending a C++ application with embedded scripting. Verkkodokumentti. Stackoverflow. 2012. <http://stackoverflow.com/questions/8872743/extending-a-c-application-with-embedded-scripting>. Luettu 10.4.2013.

21 Browser Extensions. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Browser_extension>. Luettu 10.4.2013.

22 User interface customization guide. Verkkodokumentti. WoWWiki. <http://www.wowwiki.com/User_interface_customization_guide>. Luettu 10.4.2013.

23 World of Warcraft kuvakaappaus. Verkkokuva. <http://www.elvenrunes.com/>. Katsottu 10.4.2013.

24 World of Warcraft kuvakaappaus. Verkkokuva. <http://www.xdsign.dk/wow/UI2007b.jpg>. Katsottu 10.4.2013.

25 Lua. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Lua_(programming_language)>. Luettu 10.4.2013.

26 Iterative and incremental development. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Iterative_and_incremental_development>. Luettu 11.4.2013.

27 Anygraafin Neo-toimitusjärjestelmä. Verkkodokumentti. Anygraaf Oy. <http://www.anygraaf.fi/main/rightcol_fin/neo_publishing_solution_68.html>. Luettu 11.4.2013.

28 Tietokanta. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Tietokanta>. Luettu 11.4.2013.

29 Hilario, Leonel. 2010. 5 Top jQuery Chart Libraries for Interactive Charts. Verkkodokumentti. <http://www.1stwebdesigner.com/css/top-jquery-chart-libraries-interactive-charts/>. Luettu 13.4.2013.

30 jQuery. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Jquery>. Luettu 13.4.2013.

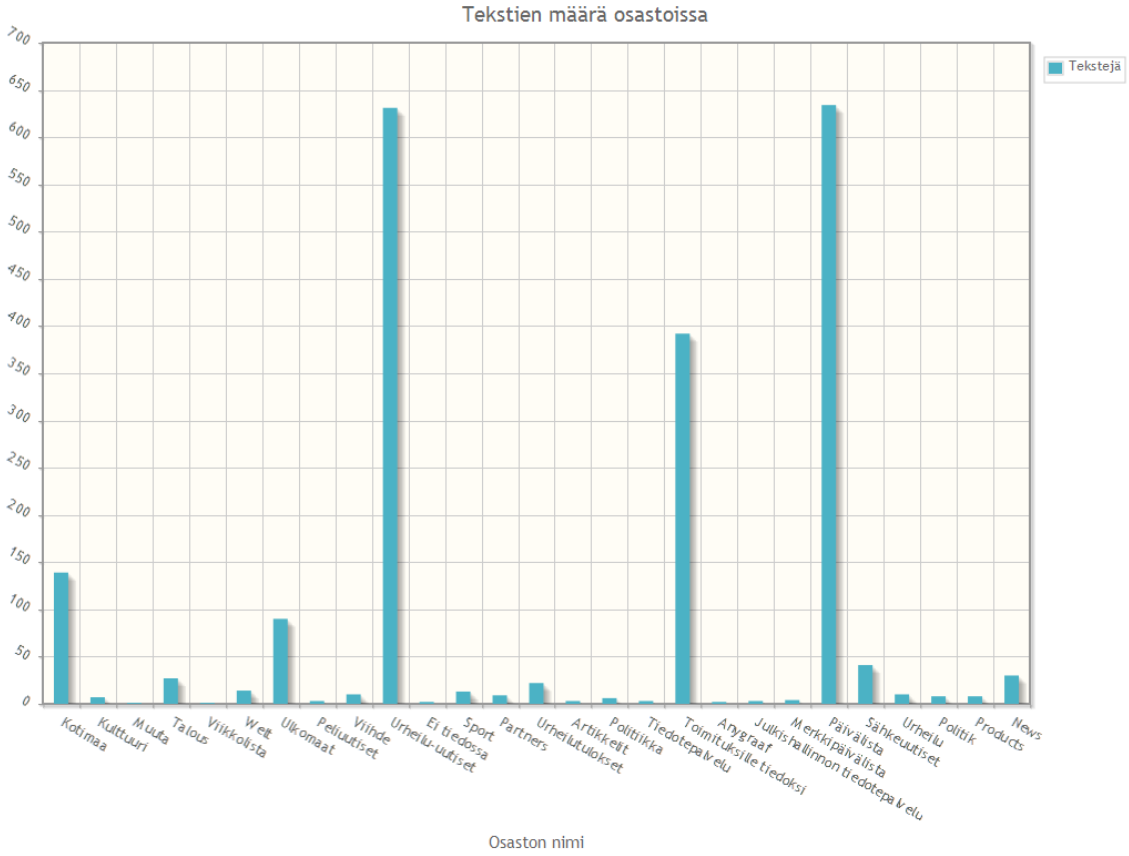
31 Leonello, Chris. jqPlot. Verkkodokumentti. <http://www.jqplot.com/>. Luettu 13.4.2013.

32 Bach, Christian. Tablesorter. Verkkodokumentti. <http://tablesorter.com/docs/>. Luettu 14.4.2013.

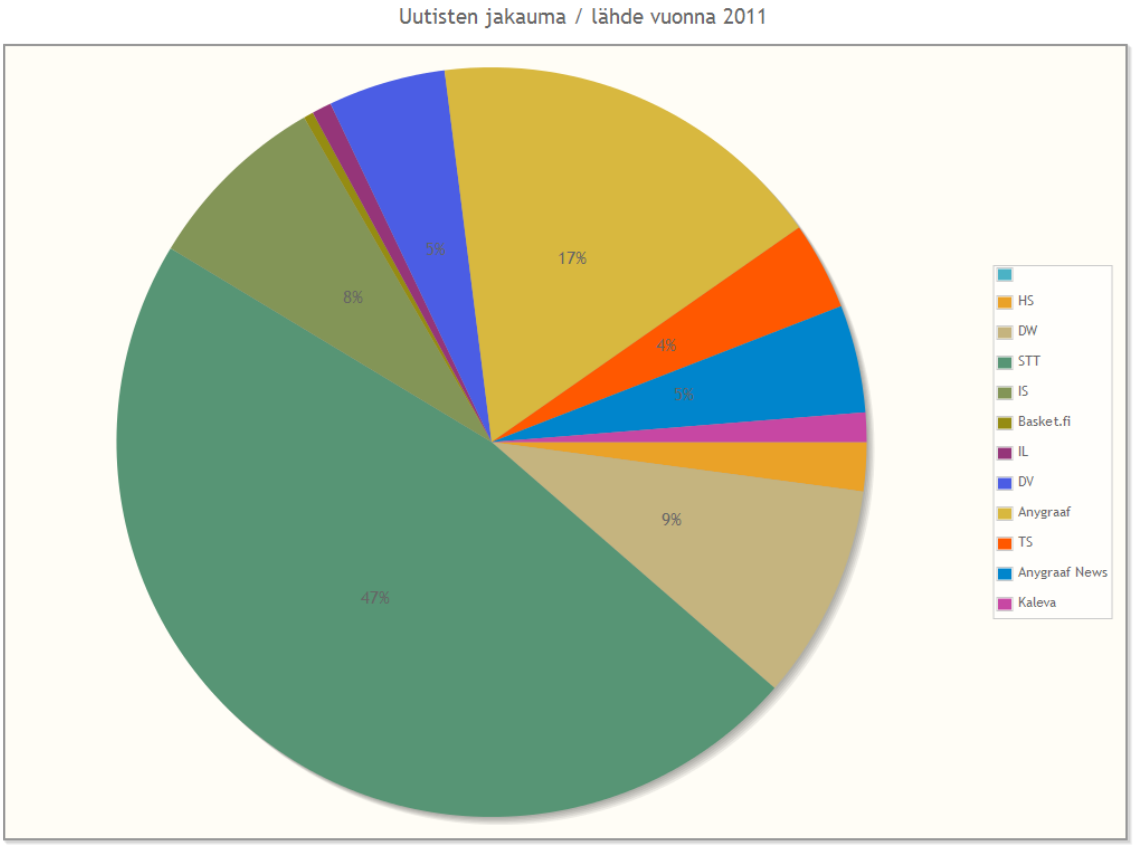
33 XULRunner. Verkkodokumentti. Mozilla Foundation. <https://developer.mozilla.org/en-US/docs/XULRunner>. Luettu 15.4.2013.

34 Telkkä, Tuomo. Torri, Hanna. 2013. Product database in editorial systems. Verkkovideo. <http://vimeo.com/album/2260291/video/59521407>. Katsottu 16.5.2013.

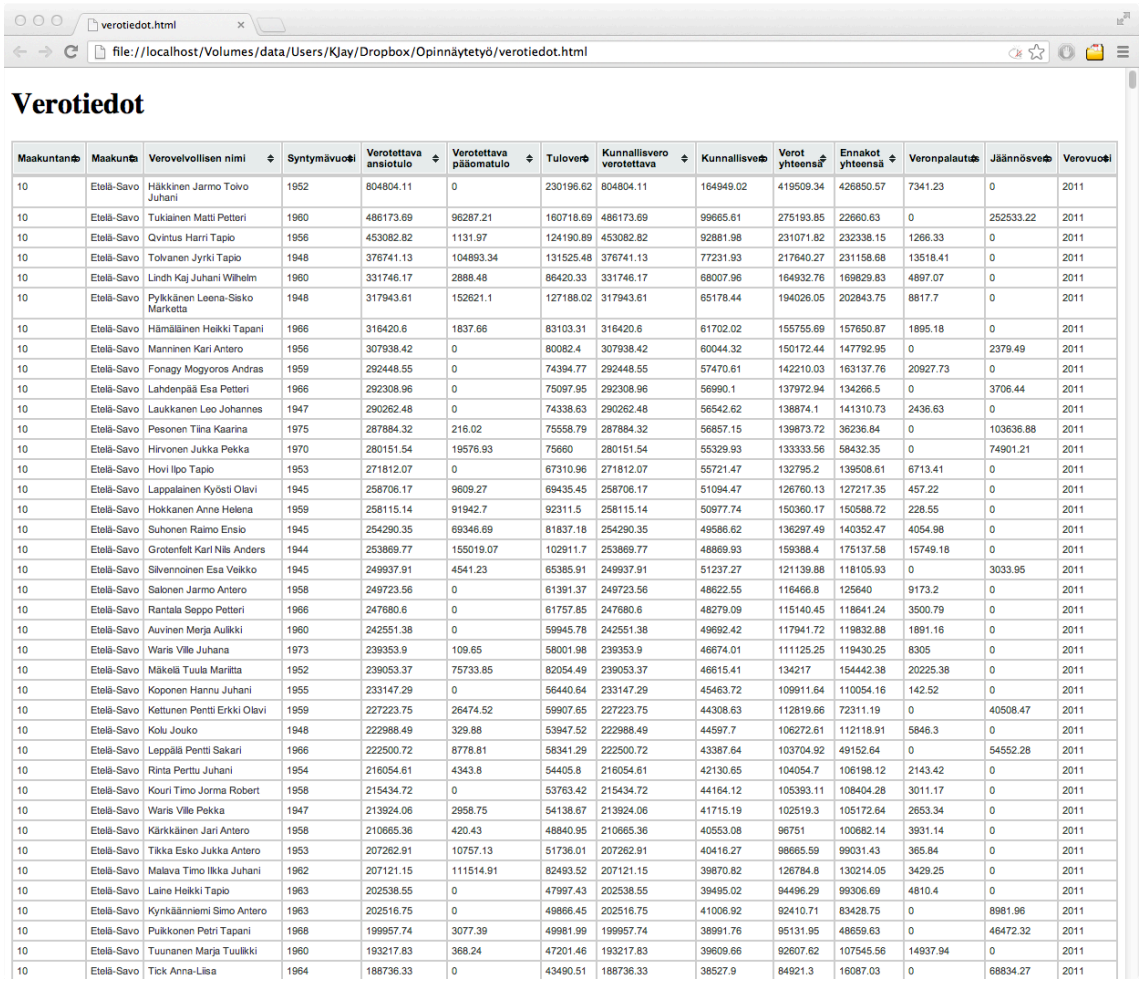
Esimerkkiohjelmalla tuotettuja kuvaajia



Kuva 1. Pylväsdiagrammi lehtijuttujen jakaumasta osastoittain.



Kuva 2. Ympyrädiagrammi eri lähteistä tulleiden uutisaiheiden jakaumasta.



verotiedot.html

file://localhost/Volumes/data/Users/KJay/Dropbox/Opinnäytetyö/verotiedot.html

Verotiedot

Maakuntano	Maakunta	Verovelvollisen nimi	Syntymävuosi	Verotettava ansiotulo	Verotettava pääomatulo	Tulovero	Kunnallisvero verotettava	Kunnallisvero	Verot yhteensä	Ennakot yhteensä	Veronpalautus	Jäännösvero	Verovuosi
10	Etelä-Savo	Häkkinen Jarmo Toivo Juhani	1952	804804.11	0	230196.62	804804.11	164949.02	419509.34	426850.57	7341.23	0	2011
10	Etelä-Savo	Tukiainen Matti Petteri	1960	486173.69	96287.21	160718.69	486173.69	99665.61	275193.85	22660.63	0	252533.22	2011
10	Etelä-Savo	Qvintus Harri Tapio	1956	453082.82	1131.97	124190.89	453082.82	92881.98	231071.82	232338.15	1266.33	0	2011
10	Etelä-Savo	Tolvanen Jyrki Tapio	1948	376741.13	104893.34	131525.48	376741.13	77231.93	217640.27	231158.68	13518.41	0	2011
10	Etelä-Savo	Lindh Kaj Juhani Wilhelm	1960	331746.17	2888.48	86420.33	331746.17	68007.96	164932.76	169829.83	4897.07	0	2011
10	Etelä-Savo	Pylkkänen Leena-Sisko Marketta	1948	317943.61	152621.1	127188.02	317943.61	65178.44	194026.05	202843.75	8817.7	0	2011
10	Etelä-Savo	Hämäläinen Heikki Tapani	1966	316420.6	1837.66	83103.31	316420.6	61702.02	155755.69	157650.87	1895.18	0	2011
10	Etelä-Savo	Manninen Kari Antero	1956	307938.42	0	80082.4	307938.42	60044.32	150172.44	147792.95	0	2379.49	2011
10	Etelä-Savo	Fonagy Mogyoros Andras	1959	292448.55	0	74394.77	292448.55	57470.61	142210.03	163137.76	20927.73	0	2011
10	Etelä-Savo	Lahdenpää Esa Petteri	1966	292308.96	0	75097.95	292308.96	56990.1	137972.94	134266.5	0	3706.44	2011
10	Etelä-Savo	Laukkanen Leo Johannes	1947	290262.48	0	74338.63	290262.48	56542.62	138874.1	141310.73	2436.63	0	2011
10	Etelä-Savo	Pesonen Tiina Kaarina	1975	287884.32	216.02	75558.79	287884.32	56857.15	139873.72	36236.84	0	103636.88	2011
10	Etelä-Savo	Hirvonen Jukka Pekka	1970	280151.54	19576.93	75660	280151.54	55329.93	133333.56	58432.35	0	74901.21	2011
10	Etelä-Savo	Hovi Ilpo Tapio	1953	271812.07	0	67310.96	271812.07	55721.47	132795.2	139508.61	6713.41	0	2011
10	Etelä-Savo	Lappalainen Kyösti Olavi	1945	258706.17	9609.27	69435.45	258706.17	51094.47	126760.13	127217.35	457.22	0	2011
10	Etelä-Savo	Hokkanen Anne Helena	1959	258115.14	91942.7	92311.5	258115.14	50977.74	150360.17	150588.72	228.55	0	2011
10	Etelä-Savo	Suhonen Raimo Ensio	1945	254290.35	69346.69	81837.18	254290.35	49586.62	136297.49	140352.47	4054.98	0	2011
10	Etelä-Savo	Grotenfelt Karl Nils Anders	1944	253869.77	155019.07	102911.7	253869.77	48869.93	159388.4	175137.58	15749.18	0	2011
10	Etelä-Savo	Silvennoinen Esa Veikko	1945	249937.91	4541.23	65385.91	249937.91	51237.27	121139.88	118105.93	0	3033.95	2011
10	Etelä-Savo	Salonen Jarmo Antero	1958	249723.56	0	61391.37	249723.56	48622.55	116466.8	125640	9173.2	0	2011
10	Etelä-Savo	Rantala Seppo Petteri	1966	247680.6	0	61757.85	247680.6	48279.09	115140.45	118641.24	3500.79	0	2011
10	Etelä-Savo	Auvinen Merja Aulikki	1960	242551.38	0	59945.78	242551.38	49692.42	117941.72	119832.88	1891.16	0	2011
10	Etelä-Savo	Waris Ville Juhana	1973	239353.9	109.65	58001.98	239353.9	46674.01	111125.25	119430.25	8305	0	2011
10	Etelä-Savo	Mäkelä Tuula Maritta	1952	239053.37	75733.85	82054.49	239053.37	46615.41	134217	154442.38	20225.38	0	2011
10	Etelä-Savo	Koponen Hannu Juhani	1955	233147.29	0	56440.64	233147.29	45463.72	109911.64	110054.16	142.52	0	2011
10	Etelä-Savo	Kettunen Pentti Erkki Olavi	1959	227223.75	26474.52	59907.65	227223.75	44308.63	112819.66	72311.19	0	40508.47	2011
10	Etelä-Savo	Kolu Jouko	1948	222988.49	329.88	53947.52	222988.49	44597.7	106272.61	112118.91	5846.3	0	2011
10	Etelä-Savo	Leppälä Pentti Sakari	1966	222500.72	8778.81	58341.29	222500.72	43387.64	103704.92	49152.64	0	54552.28	2011
10	Etelä-Savo	Rinta Perttu Juhani	1954	216054.61	4343.8	54405.8	216054.61	42130.65	104054.7	106198.12	2143.42	0	2011
10	Etelä-Savo	Kouri Timo Jorma Robert	1958	215434.72	0	53763.42	215434.72	44164.12	105393.11	108404.28	3011.17	0	2011
10	Etelä-Savo	Waris Ville Pekka	1947	213924.06	2958.75	54138.67	213924.06	41715.19	102519.3	105172.64	2653.34	0	2011
10	Etelä-Savo	Kärkkäinen Jari Antero	1958	210665.36	420.43	48840.95	210665.36	40553.08	96751	100682.14	3931.14	0	2011
10	Etelä-Savo	Tikka Esko Jukka Antero	1953	207262.91	10757.13	51736.01	207262.91	40416.27	98665.59	99031.43	365.84	0	2011
10	Etelä-Savo	Malava Timo Ilkka Juhani	1962	207121.15	111514.91	82493.52	207121.15	39870.82	126784.8	130214.05	3429.25	0	2011
10	Etelä-Savo	Laine Heikki Tapio	1963	202538.55	0	47997.43	202538.55	39495.02	94496.29	99306.69	4810.4	0	2011
10	Etelä-Savo	Kynkäänniemi Simo Antero	1963	202516.75	0	49866.45	202516.75	41006.92	92410.71	83428.75	0	8981.96	2011
10	Etelä-Savo	Puikkonen Petri Tapani	1968	199957.74	3077.39	49981.99	199957.74	38991.76	95131.95	48659.63	0	46472.32	2011
10	Etelä-Savo	Tuunanen Marja Tuulikki	1960	193217.83	368.24	47201.46	193217.83	39609.66	92607.62	107545.56	14937.94	0	2011
10	Etelä-Savo	Tick Anna-Liisa	1964	188736.33	0	43490.51	188736.33	38527.9	84921.3	16087.03	0	68834.27	2011

Kuva 3. Taulukko vuoden 2012 verotuksesta Etelä-Savossa.