Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Tietokantajärjestelmät

2010

Jarmo Torvinen

Skaalautuvien vektorigrafiikkadiagrammien toteuttaminen käyttämällä SVG-tekniikkaa

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutusohjelma | Tietokantajärjestelmät

Kesäkuu 2010 | 29 sivua

Oskari Kiviniemi

Jarmo Torvinen

Skaalautuvien vektorigrafiikkadiagrammien toteuttaminen käyttämällä SVG-tekniikkaa

Tämän työn tavoitteena oli tutkia, soveltaa ja tehdä malliesimerkkejä SVG-pohjaisesta diagrammeista käyttämällä mobiililaitteita, yrityksen omaa www-palvelinta, tietokantaa ja tässä olevaa dataa pohjana.

Käytettynä menetelmänä oli yhdistää erilaiset standardit yhteen ja luoda näistä erilaisista standardeista dynaamisia diagrammeja yrityksen käyttöön. Käytettävät standardit olivat HTML-, XML-, PHP-, MySql- ja SVG-tekniikka. Nämä yhdessä yhdistettyinä matemaattisiin kaavoihin takaavat laadukkaan lopputuloksen.

Opinnäytetyön tuloksena syntyi toimiva sivusto ja ohjelmapaketti, joka toimii samalla tavalla niin mobiililaitteessa kuin normaalissa tietokoneessakin. Lisäksi lopputulos on selkeä ja nopea. Tavoitteisiin pääsemiseksi tehtiin omaa selvitystyötä sekä oltiin aktiivisia erilaisilla kehitysfoorumeilla. Nämä selvitysten ja yhteistyön lopputulokset on kuvattu työssä.

Lopullinen tuotos on nopea, toimiva ja vakuuttava esitysmedia monenlaiseen yrityskäyttöön, kun halutaan kuvata visuaalisesti reaaliaikaista dataa yrityksen tilasta tai varastojen arvoista. Työssä olevia ohjelmointiesimerkkejä voidaan vähällä vaivalla soveltaa numeeristen tietoaineistojen visualisointiin internetselaimella sekä myös mobiililaitteilla.

ASIASANAT:

SVG, XML, PHP, SQL, diagrammit, relaatiotietokannat

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Database Systems

June 2010 | 29 pages

Oskari Kiviniemi

Jarmo Torvinen

Implementation of the Scaleable Vectordiagrams by Using SVG-technigues

The purpose of this thesis was study, apply and make examples of SVG-based diagrams by using mobile devices, corporate personal web-server, database and the data in it.

Different standards were combined together to create dynamic diagrams in corporate use. Used standards were HTML-, XML-, MySql- and SVG-technique. Combining these techniques with mathematic formulas guarantees good quality results.

In this study the result was working web-pages and software-package, which works in the same way in mobile devices as in normal computer. Moreover, the results are clear and fast to use. These results required background research work and activity in different development forums. Results and collaboration are discusses in this thesis.

The final outcome is a fast, working and convincing presentation media for multipurpose corporate use, when result needs to be visual and real-time data from corporate state or value of stocks. Programming results in this thesis can be used with moderate modifications in different numeric data materials visualization presentation in web-browsers and mobile devices.

KEYWORDS:

PHP, SQL, XML, diagram, relation database

SISÄLTÖ

TERMISTÖ

1 JOHDANTO 1

1.1 Työn tavoitteet 1

1.2 Skaalautuvan vektorigrafiikan taustoja 1

2 SVG:N PERUSARKKITEHTUURI 4

2.1 SVG:n peruselementit 4

2.2 SVG:n <g>-elementti 5

2.3 Pattern-elementti 7

3 SVG-YMPÄRISTÖ 9

3.1 SVG:n palvelinympäristö 9

3.2 Tietokannan kuvaus 10

3.3 Järjestelmän kutsut kaaviossa 12

4 SVG-OHJELMOINTIKIELELLÄ RATKAISTUJA ONGELMIA OHJELMOINTIRAJAPINNALLA 14

4.1 Ratkaisu pylvään korkeuden laskentaan 16

4.2 Ratkaisu viivadiagrammi 19

4.3 Ratkaisu piirakkadiagrammiin 20

5 SVG-NÄKYMIÄ ERILAISILLA NÄYTÖILLÄ 24

6 JOHTOPÄÄTÖKSET 27

LÄHTEET 29

KUVAT

Kuva 1. Kappale 1 & 2, lähdekoodi 6
Kuva 2. Kuva lopputuloksesta 6
Kuva 3. Pattern-koodia 8
Kuva 4. Pattern-esimerkki 8
Kuva 5. Kannan osarakennekuva 10
Kuva 6. Kuva järjestelmän tuottamasta kaaviosta 11
Kuva 7. Sekvenssikaavio generoinnista 12
Kuva 8. Lähdekoodin kuvankaappaus 13
Kuva 9. Tulostus palkkidiagrammista 15
Kuva 10. Pyöristysfunktio 16
Kuva 11. Pituusfunktio 17
Kuva 12. Palkkifunktio 18
Kuva 13. Viivadiagrammin esimerkki 19
Kuva 14. Viivan silmukka 20
Kuva 15. Kaavan ratkaisua 21
Kuva 16. Piirakkadiagrammi 23
Kuva 17. E90-ruutukaappaus, Opera-selaimella 24
Kuva 18. Ruutukaappaus näytöltä, Opera-selaimella 25
Kuva 19. Android-ruutukaappaus 26
Kuva 20. iPad-ruutukaappaus 26

Termistö

LAMP, WAMP ja XAMPP

Nämä ovat kokoelmia avoimia lähdekoodiohjelmia, jotka yhdessä muodostavat WWW-palvelimen ja jonka alla voidaan suorittaa dynaamisia web-sivuja ja tietokantoja. L tarkoittaa Linus-ympäristöä. W tarkoittaa Windows-ympäristöä ja X tarkoittaa kannettavaa, muistitikulla suoritettavaa ympäristöä. (xampp 2010)

Android Mobiililaitteille tarkoitettu käyttöjärjestelmä, joka pohjautuu vahvasti Linux-ytimeen. Google on alun perin kehittänyt kyseisen käyttöjärjestelmän, mutta nykyään sen kehittämisestä vastaa Open Handset Alliance. Android-järjestelmä koostuu 48 laitteisto- ja ohjelmistovalmistajasta sekä teleoperaattorista. Google on julkaissut suurimman osan Androidin koodista Apache-lisenssillä. (Android 2010)

SVG Scalable Vector Graphics, skaalautuva vektorigrafiikka on kaksiulotteisten vektorikuvien kuvauskieli ja toteutus tehdään XML-merkintäkielellä. Tämä mahdollistaa liikkuvien kuvien esittämisen. SVG on World Wide Web Consortiumin kehittämä avoin kuvatiedostostandardi vuodelta 1999. (SVG 2010; SVG tutorial 2010)

SVG tukee kaikkein monipuolisimmin erilaisia mittayksikköjä. Yleisimmät ja käytetyimmät mittayksiköt SVG:ssä ovat em, ex, px, pt, pc, cm, mm, in ja prosentit, joista em, ex, pt, pc, cm, mm ja in ovat pääsääntöisesti painotaloissa käytettyjä pituusmittoja.

View Näkymä, joka koostuu tallennetuista kyselyistä. Sitä käsitellään kuin virtuaalista taulua kannassa, joka esittää kyselyn lopputuloksen. Näkymä ei ole fyysinen osa todellisia tauluja.

1 Johdanto

1.1 Työn tavoitteet

Tämän työn tavoitteena on tehdä toimiva ohjelmistopaketti ja web-sivusto, jossa toimii dynaamisia diagrammeja lähes reaaliajassa. Informaationa diagrammeille toimii yrityksen oma tietokanta, josta haetaan erilaisilla kyselyillä ja näkymillä tiedot ohjelmaan: näin ne käsitellään ja esitetään visuaalisesti erilaisina diagrammeina.

Osana kokeiluna tullaan käyttämään Linuksessa toimivaa virtualisointia, jossa ajetaan kaksi erilaista alustaa. Toinen alustoista on Googlen Android ja toinen Applen iPad. Työssä ei ollut mahdollista käyttää fyysisiä laitteita, koska niiden saatavuus tätä työtä tehdessä oli erittäin heikko. Tästä syystä ei ole mahdollista demonstroida fyysisellä laitteella ja ottaa ruutukaappauksia suoraan laitteesta.

1.2 Skaalautuvan vektorigrafiikan taustoja

Kiinnostukseni skaalautuvan vektorigrafiikan (SVG) tekniikkaan syntyi siitä syystä, että oppilaitoksessamme aihetta on tutkittu vähän. Toinen syyni valita SVG-kieli opinnäytetyöksi oli se, että suomalaisissa kirjoissa ei ole selitetty eikä demonstroitu sitä, miten SVG:llä voisi tehdä dynaamisia pylväs-, viiva- ja piirakkadiagrammeja suoraan yrityksen tietokannoista käyttäen vain PHP-syntaksia sekä dynaamisia web-sivuja.

Suomessa on saatavilla tällä hetkellä yksi kirja, joka referoi 10 % SVG:n käyttömahdollisuuksia ja jättää kokonaan pois juuri sen osa-alueen, josta olen kiinnostunut, eli tietokannat ja dynaamiset taulukot. Tästä syntyi ongelma, koska SVG:stä löytyy vain yksi kirja, jossa kaikki on selitetty ja se on ainoa tietolähteeni tähän eksperimentaaliseen ohjelmointityöhön.

Adobe on kehityttänyt tekstien ja kuvien tulostamista varten PostScript-kielen vuonna 1982. PostScript on tavallaan SVG:n esi-isä, joka pystyi käsittelemään graafisesti skaalautuvia fontteja ja objekteja. Isona haittana pidettiin sitä, että PS-tiedostot kasvoivat niin isoiksi, että niiden tulostaminen kesti kauan eikä niitä ollut suunniteltu internetiä varten.

Vektorigrafiikkaa internetiin ei ollut uusi idea. Adoben, IBM:n, Netscapen ja Sunin yhteistyötiimi esitti idean W3C:lle XML-pohjaisesta PGML:stä (Precision Graphics Markup Language) huhtikuussa 1998. Samaan aikaan yhteistyötiimi HP, Macromedia, Microsoft ja Visio esitti W3C:lle idean XML-pohjaisesta VML:stä (Vector Markup Language) toukokuussa 1998. Lopputuloksena kahdesta ideasta syntyi SVG-kehitystyöryhmä, joka julkaisi vaatimuksia SVG:lle lokakuussa 1998. (Campesato 2004, xvii – xviii.; SVG-wiki 2010)

Ensimmäiset vedokset SVG:stä tulivat julki helmikuussa 1999, ja siitä päivästä lähtien on tullut julkaisuun monta päivitystä ja muutamia kokeellisia täytäntöönpanoja.

SVG-tiedostot ja niiden käyttäytyminen on määritelty XML-tekstitiedossa. Nämä voidaan hakea, indeksoida, skriptata ja tarvittaessa pakata. Koska ne ovat XML-tiedostoja, SVG-kuvat voidaan luoda ja muokata millä tahansa tekstieditorilla, mutta niitä varten on myös SVG-pohjainen piirustusohjelma, joilla niitä voidaan luoda ja muokata.

Kaikki modernit internetselaimet Microsoft Internet Exploreria (IE8) lukuun ottamatta tukevat suoraan ja suorittavat SVG-ohjelmaskriptejä. Suosituin SVG:tä tukeva selain on Firefox, paras SVG:n skaalaavuuden testaukseen Opera sekä pluginin kautta IE8.

Vaikka SVG on ensisijaisesti vektorigrafiikan kuvauskieli, niin sitä käytetään yleisesti erilaisissa teollisuussovelluksissa. Näistä suurimmat ovat graafinen teollisuus, jossa Canon, Adobe ja Corel ovat tehneet SVG-alalajin nimeltään SVG-Print.

Esimerkkinä SVG:n käytöstä normaalissa tilanteessa on painoalalla: posterin taikka muun erittäin ison mainoksen painaminen niin, ettei sen lopullinen laatu kärsisi. Tämä tapahtuu niin, että bittikuvat ja tekstit muutetaan vektoreiksi ja tällä tavalla skaalataan isommaksi kyseistä kuvaa taikka mainosta. Kaikki skaalautuu lineaarisesti, ja lopputulos on samanlainen kuin alkuperäinen. Ainoa poikkeus tähän on fraktaalisuurennus, jota käytetään niin sanotussa lehtipainossa, joiden lopullinen kuva taikka mainoskoko ei ole A2:ta suurempi.

Tämänhetkisten tietojen mukaan maailman suurimmat SVG:n käyttäjät ovat normaalit ihmiset, jotka eivät ole huomanneet, että normaalit matkapuhelimet ovat SVG-alustoja. Nokialta on esimerkiksi SVG-tuki ollut ensimmäisestä Symbian-puhelimesta saakka eli vuodesta 2003. Tämä on luonut markkinoille uusia sovelluksia, jotka ovat nopeita ja kevyitä. Hyvänä esimerkkinä voidaan pitää matkapuhelimien gps-navigaatiosovellukset, autojen navigaatiolaitteet ja internetin karttasovellukset sekä animoidut soittoäänet ja animoidut taustat.

2 SVG:n perusarkkitehtuuri

2.1 SVG:n peruselementit

SVG:ssä on kahdeksan peruselementtiä:

- **Line**, suora viiva kahden koordinaattipisteen välillä (Nykänen 2007, 81–82).
- **Path**, polku, kaikkein vaikein elementti käyttää. Käytetään piirtäessä monimutkaisia kuvioita yhdistäen viivoja ja kaaria. Sisältäen kaikki viivan aliominaisuudet, mm. kvadraattisen Bézier-käyrän. (Nykänen 2007, 86–90.)
- **Circle**, ympyrä, kaksi koordinaattia ja säteen pituus (Nykänen 2007, 79).
- **Ellipse**, ellipsi, kuin ympyrä mutta kaksi muuta koordinaattia määrittää ellipsin muodon (Nykänen 2007, 80).
- **Rectancle**, neliö, pituus ja leveys sekä kulmanpyöristys (Nykänen 2007, 77–78).
- **Polyline**, murtoviiva; kuin viiva mutta jatkaa edellisestä pisteestä seuraavaan annettuun pisteeseen (Nykänen 2007, 83–84).
- **Polygon**, monikulmio, kun halutaan piirtää enemmän kuin kolmisivuisia kappaleita. Ainoa ero murtoviivaan on se, että monikulmio piirtää automaattisesti viimeisestä annetusta koordinaatista viivan ensimmäiseen annettuun koordinaattiin. (Nykänen 2007, 84–85.)
- **Text**, teksti; kirjoitettu teksti, joka sisältää monia aliominaisuuksia, mm. fontit, koon ja tyyli (Nykänen 2007, 108–110).

2.2 SVG:n <g>-elementti

Työssäni käytin pääsääntöisesti SVG:n <g>-elementtiä. Tämä vastaa jollakin tavalla Java-ohjelmointikielen Graphics-luokkaa, tosin monipuolisemmin. <g>-elementtiä käytetään pääsääntöisesti ryhmittämään yksittäisiä SVG-elementtejä, jotka muodostavat yhdessä yhden <g>-elementin, jota voidaan käsitellä erilaisilla komennoilla. Tämä antaa nopeutta ja käsiteltävyyttä: jokaista yksittäistä SVG-elementtiä ei tarvitse jokaiselle kertoa samaa käskyä. Tarvitaan vain yksi käsky ja kyseinen <g>-elementti tekee suoritettavan käskyn yhdellä kertaa. Käsky, jolla käsketään <g>-elementtiä, on *Transform*. (Campesato 2004, 5–6.; Nykänen 2007, 25.)

Kuvan 1 esimerkissä kummallekin kappaleelle on annettu samat perustiedot, viiva, laatikko ja teksti ”Ryhmitelty Teksti”.

- Kappale 1 (vasen) annettiin käsky Rotate (kallistaa) 45 astetta myötäpäivään
- Kappale 2 (oikea) annettiin käsky Rotate (kallistaa) –60 astetta vastapäivään.

Muita käskykomentoja g-ryhmälle ovat seuraavat: siirto (translate), kallistus (rotate), skaalaus (scale), venytys (skewX) ja venytys (skewY).

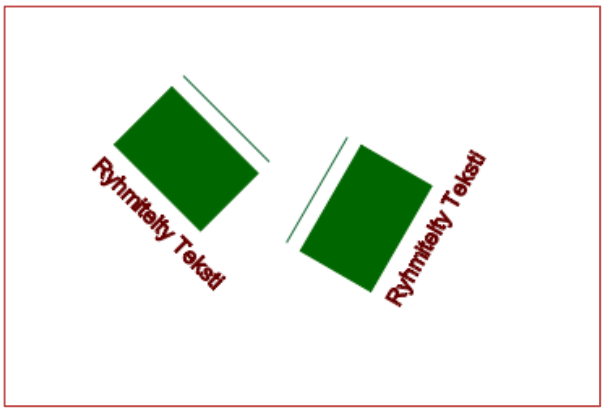
```
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">

    <g id="Kappale 1" transform="rotate(45)">
      <line x1="10" y1="10" x2="85" y2="10"
          style="stroke: #116633;"/>
      <rect x="10" y="20" height="50" width="75"
          style="stroke: #006600; fill: #006600"/>
      <text x="10" y="90" style="stroke: #660000; fill: #660000;
      font-family: Arial; font-size  : 14px;">
        Ryhmitelty Teksti</text>
    </g>


    <g id="Kappale 2" transform=" rotate(-60)" >
      <line x1="10" y1="10" x2="85" y2="10"
          style="stroke: #116633;"/>
      <rect x="10" y="20" height="50" width="75"
          style="stroke: #006600; fill: #006600"/>
      <text x="10" y="90" style="stroke: #660000; fill: #660000;
      font-family: Arial; font-size  : 14px;">
        Ryhmitelty Teksti</text>
    </g>

</svg>
```

Kuva 1. Kappale 1 & 2, lähdekoodi



Kuva 2. Kuva lopputuloksesta

2.3 Pattern-elementti

Näiden peruselementtien lisäksi on paljon muitakin elementtejä, joista tähän työhön oli pakollinen Pattern-elementti.

Pattern-elementti tarvitsee muutamia asioita toimiakseen jouhevasti. Pattern-tagi pitää olla <defs>-tagin sisällä, ”defs” tulee sanasta definitions, määritelmät. Jokaisella <defs> sisällä olevalla <pattern>-elementille pitää olla yksilöivä ID-tunniste, tässä kyseisessä tapauksessa se on nimetty ”kolmio”. (Campesato 2004, 24–25.; Nykänen 2007, 103–104.)

Itse määrittelemille yksiköille patternUnits on aina arvo ”userSpaceOnUse” ja käyttäjä kutsuu tarvittaessa patternia. Kutsu esimerkkitapauksessa ellipsi halutaan täyttää kolmiolla ja kutsu tapahtuu Fill-proseduurissa kutsumalla url(#kolmio). Tämä on lähes samanlainen kutsu kuin normaalissa HTML-koodissa tai XML-koodissa. Tässä työssä käytetään Pattern- ja Defs-tageja luotaessa diagrammeille taustaristikko ja kuviointi.

Kuvassa 3 näkyy oikea järjestys, millä tavalla SVG-standardi vaatii määrittelemään ennen mitään kutsuja ja muotoja kaikki mahdolliset määritelmät. Mikäli <defs>-tagi olisi <g>-tagin jälkeen, niin lopputuloksena olisi pitkä ja tylsä XML-pohjainen virheilmoitus, josta ei saa selvää, missä virhe on oikeasti. Kuvasta 3 myös näkee riviltä 1 sen, että esimerkki on puhdasta SVG-koodia ilman PHP-syntaksia. (Campesato 2004, 24–25.; Nykänen 2007, 103–104.)

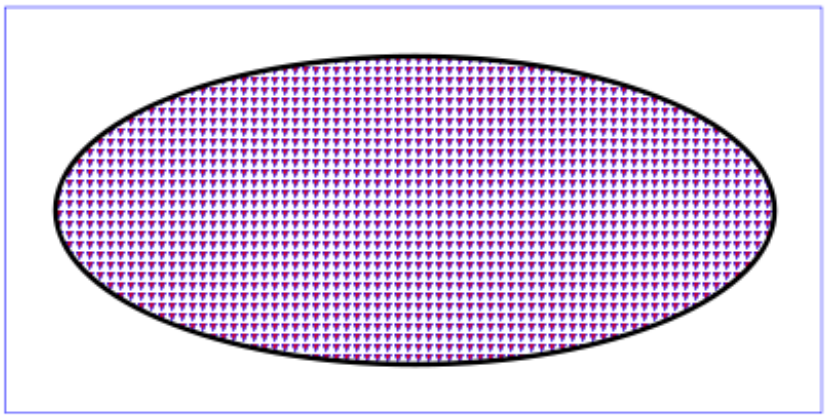
```
<svg version="1.0" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://ww


<defs>
<pattern id="kolmio" patternUnits="userSpaceOnUse"
              x="0" y="0" width="10" height="10"
              viewBox="0 0 10 10" >
      <path d="M 0 0 L 7 0 L 3.5 7 z" fill="red" stroke="blue" />
</pattern>
</defs>

<g transform="translate(10 10 ) scale(0.5)">
<rect fill="none" stroke="blue" x="1" y="2" width="795" height="395" />
<ellipse     fill="url(#kolmio)" stroke="black" stroke-width="5"
             cx="400" cy="200" rx="350" ry="150" />
</g>
</svg>
```

Kuva 3. Pattern-koodia

Kuvassa 4 on käytetty kuvan 3 mukaisia kutsuja sekä määritelmiä ja lopputuloksena on oikeaoppinen ellipsi kolmiotäytteellä.



Kuva 4. Pattern-esimerkki

3 SVG-ympäristö

3.1 SVG:n palvelinympäristö

Tässä työssä käytin pääsääntöisesti Windows-ympäristöä ja siinä Gnu-lisenssillä toimivaa Xampp-ohjelmistoa. Xampp-ohjelmistopaketti on yhdistelmä seuraavista ohjelmistoista:

- Apache www-palvelinohjelmisto
- Mysql-tietokantaohjelmisto
- PHP-ohjelmointikielirajapinta.

Olisin voinut myös käyttää Lampp-ohjelmistopakettia, joka on ihan samanlainen kuin Xampp mutta Linux-ympäristössä. En nähnyt tarpeelliseksi tehdä testausta Linux-ympäristössä, koska lopputulos on www-sivuilla toimiva, alustavapaa koodi. (Apache 2010; MySql 2010; PHP 2010; Xampp 2010)

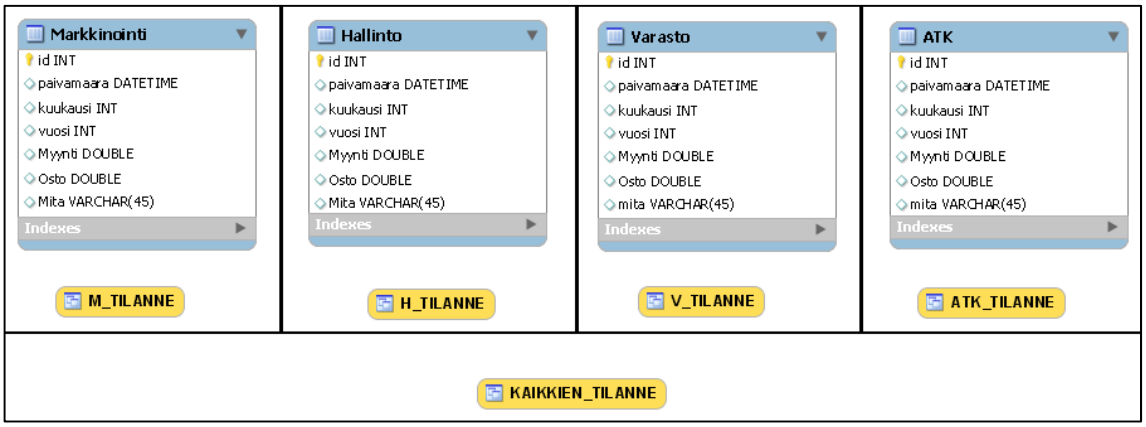
Raudan osalta SVG-ohjelmointi PHP-syntaksilla yhdistettynä MySql-tietokantaan ei vaadi suurta laskennallista tehoa. Tätä työtä tehdessäni palvelimeni kulki portaattomasti USB-muistitikulla, johon oli asennettu Xampp-ohjelmisto.

Tämä antaa uudenlaisen käsityksen mainoslauseesta ”tehdä työtä siellä, missä kone on”. Esittäessäni tätä pientä innovaatiota koulun tietojenkäsittelyn muille opiskelijoille kaikki innostuivat asiasta. Nyt he voivat tehdä ohjelmointia ja testausta siellä, missä muistitikku sopii koneeseen, ja käynnistää oman palvelimen muistitikulta.

3.2 Tietokannan kuvaus

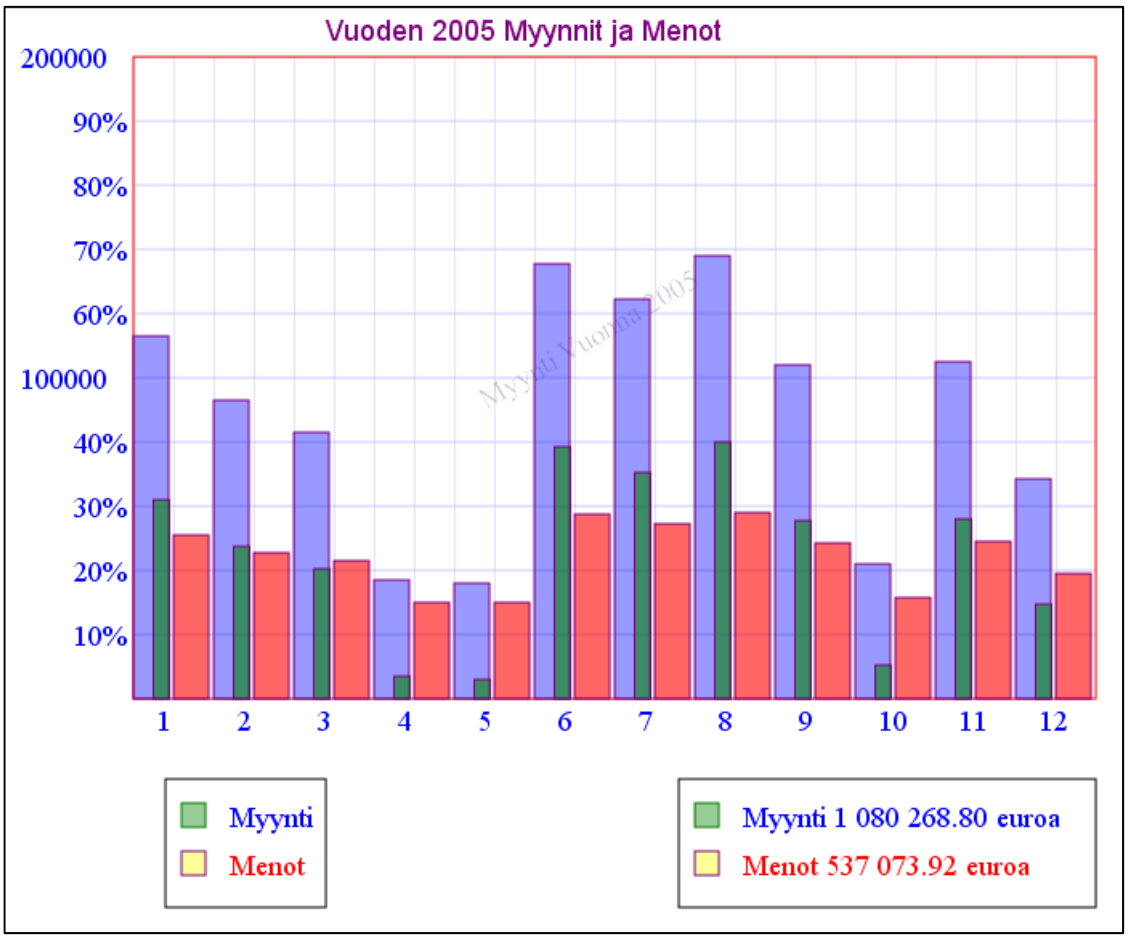
Käytettävä palvelin on jakautunut neljään eri tietokantaan sekä yhteen yleiseen tietokantaan, jossa on käyttäjäprofiilit. Luodessani kantoja suunnittelin sen niin, että jokaisella segmentillä on oma tietokantansa, joissa kuitenkin on yksi yhteisesti samanrakenteinen taulu, josta sitten haetaan data muokattavaksi.

Jokaiselle tietokannalle on oma SQL-näkymä (view, kuvassa keltainen), joka kertoo juuri kyseisen kannan segmentin kuluista, sekä yksi universaali SQL-näkymä, joka laskee kaikista neljästä kannasta tulot ja menot yhteensä sekä katteen. Samoille yksittäisille näkymille voidaan välittää parametri ”vuosi” ja ”vuosinumero”, jolloin näkymä palauttaa kyseisen vuoden tiedot.



Kuva 5. Kannan osarakennekuva

Perusideana oli tehdä yksi lomake www-sivuille, joka vaatii kirjautumisen. Tällä lomakkeella henkilö syöttää esimerkiksi joka päivä myynnit ja ostot. Kaikki kirjautuu aikaleimalla oikeaan paikkaan ja oikeaan kantaan. Näin näkymä päivittää itse itseään automaattisesti, aina kun sitä kutsutaan sivuilla. Toisella termillä sitä kutsutaan reaaliaikaiseksi näkymäksi, jolloin sivulle tuleva palkkidiagrammi on kuvan 6 näköinen.



Kuva 6. Kuva järjestelmän tuottamasta kaaviosta

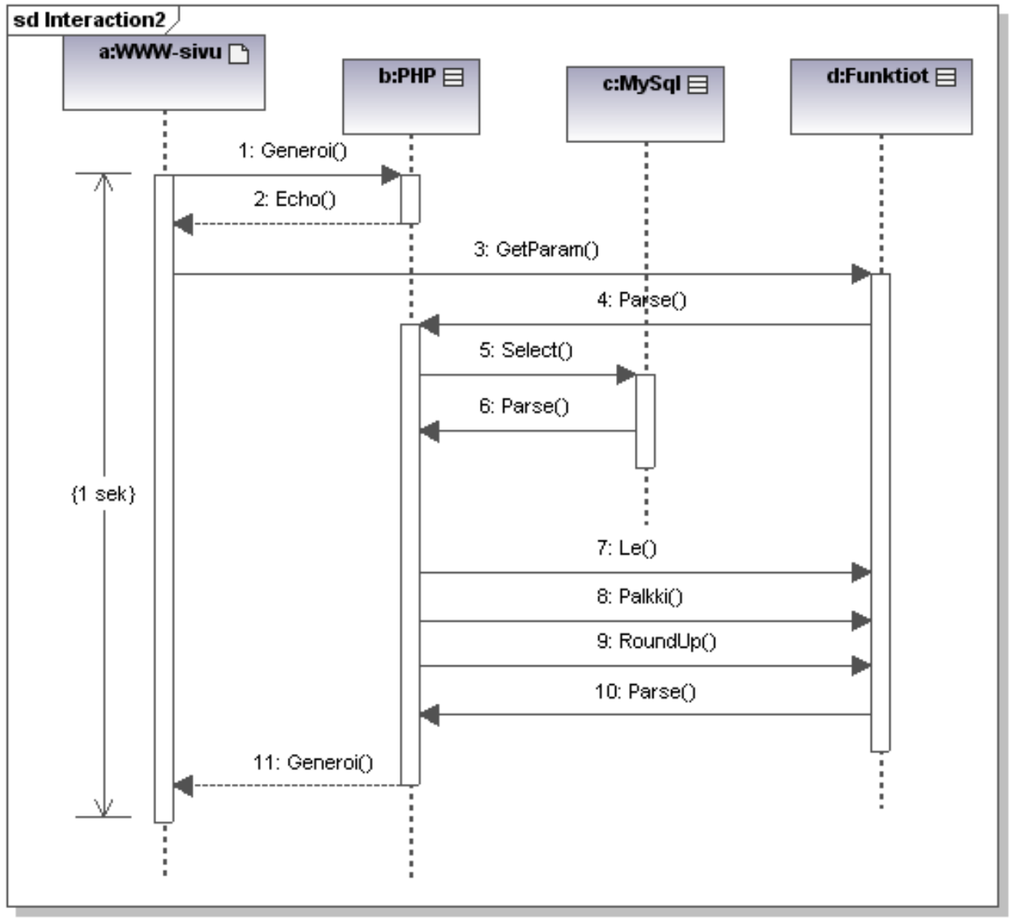
3.3 Järjestelmän kutsut kaaviossa

Vieressä olevasta kaaviosta saadaan selville, millä tavalla ja kuinka nopeasti sivulle tulevasta kutsusta saadaan luotua SVG-diagrammi. Keskimääräinen aika verkon yli on noin 1 standardisekunti.

Sivulle tullaan ulkopuolisesta osoitteesta, kirjaudutaan sisälle ja ohjataan eteenpäin. Käyttäjä valitsee haluamansa toiminnon, esimerkiksi hallinnon vuoden 2008 tulot ja menot. Hän napsauttaa linkkiä ja prosessi etenee näin kuvassa 7 olevan sekvenssikaavion mukaisesti:

1. WWW-sivu kutsuu PHP-oliota ja pyytää generoimaan SVG-kaavion.
2. PHP-olio palauttaa echo-proseduurilla ”ok” ja välittää parametreja GetParam-funktiolla eteenpäin.
3. GetParam saa parametrit ja palauttaa PHP-oliolle Parse-funktiolle.
4. Parse-funktio käsittelee ja tekee tietokantaan kyselyjä Select-funktiolla.
5. Vastaus tulee kannasta ja Parse-funktio käsittelee ne eteenpäin.
6. PHP-olio pyytää luodusta Funktiot-luokasta kolme funktiota palauttamaan parametrit takaisin antamilleen parametreille.
7. Parse-funktio saa kaikki pyydetyt tiedot ja suorittaa PHP-oliolla Generoi-funktion.
8. PHP-olio on generoinut SVG-koodin ja palauttaa näkyville saadut tulokset.

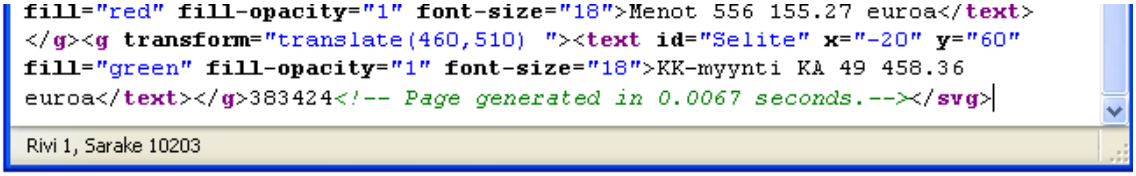
Luotu SVG-diagrammi ei poikkea kutsuiltaan ja käytökseltään mitenkään normaalista PHP-sivusta. Ainoa poikkeus on se, että näyttöön palautuva tulos on kuvaa sekä tekstiä, ei vain kuvaa taikka tekstiä.



Kuva 7. Sekvenssikaavio generoinnista

Kuva 7 selvittää suuripiirteisesti www-sivun, PHP:n, tietokannan ja funktion väliset kutsut ja antaa käsityksen siitä, kuinka tiivis paketti on kyseessä. Kutsuttaessa esimerkiksi matkapuhelimella diagrammia, tiedon palauttamiseen takaisin matkapuhelimeen diagrammiksi on se standardi 1 sekunti, joka on nykyaikana erittäin nopea tulos. Tehtäessä sama testi tietokoneella, palautusaika diagrammille oli 0,0067 sekuntia.

Kuvassa 8 on lähdekoodin kuva sivun luomiseen käytetystä ajasta.



Kuva 8. Lähdekoodin kuvankaappaus

Generoitu suoritusaika merkitsee näinä päiviä erittäin paljon. Nykytrendien mukaan kaiken pitää toimia mobiililaitteessa ja olla saatavilla mahdottoman nopeasti. Huonona esimerkkinä voidaan pitää erilaisia tabloidi-lehtien sivustoja, joiden latautuminen matkapuhelimella on erittäin hidasta.

Tästä on esimerkkinä Iltasanomien kotisivujen latautuminen E90 kommunikaattorilla, käyttäen WLAN-verkkoa ja Opera selainta kestää noin 1 minuuttiin, kun taas Iltalehden mobiilisivut latautuvat noin 10 sekunnissa.

Muita hyötyjä on toisena esimerkkinä kokous, jossa halutaan esittää vuoden liikevaihdonkehitys graafisena käyränä. Normaalisti tähän tarvitaan joku sihteeri taikka avustaja etsimään papereista ja taloustoimistosta tiedot, syöttämään saadut tiedot Excel-taulukkoon ja tekemään manuaalisesti viivadiagrammi.

Kokouksessa johtaja voi vain napsauttaa linkkiä ja ohjelma hakee suoraan tietokannoista tarvittavan datan ja tulostaa näytölle viivadiagrammin alle sekunnissa.

4 SVG-ohjelmointikielellä ratkaistuja ongelmia ohjelmointirajapinnalla

Saatuani käsiini kirjan nimeltään ***Fundamentals of SVG Programming: Concepts to Source Code*** aloin opiskella SVG:n kirjoittamista ja tutkia mukana ollutta CD:tä. Saatuani tarpeeksi tietoa aloin käsikirjoittaa SVG-koodia. Puhdas SVG-koodin kirjoittaminen oli helppoa ja yksinkertaista. (Campesato 2004)

Käynnistin serverin ja ensimmäinen testisivun, joka sisälsi PHP-syntaksia sekä SVG-testikoodia. Se ei toiminut ollenkaan. Kirjasta puuttuu tietoa siitä, miten pystytään kirjoittamaan PHP-kielellä SVG-koodia.

Tästä syntyi ensimmäinen tutkimusongelma. Pitkien keskustelujen myötä SVG-koodin kehittäjiltä tuli ratkaisukoodi ongelmaan ja samaa ratkaisua käytetään yhä tänäkin päivänä luotaessa dynaamista PHP/SVG-koodia.

PHP-sivun alkuun pitää laittaa kolme riviä koodia, jotta PHP-kääntäjä ja apache osaa lukea sisältöä oikein ja palauttaa tarvittavan datan oikeassa muodossa.

```
header ('Content-Type: image/svg+xml')
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg version="1.0" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" >
```

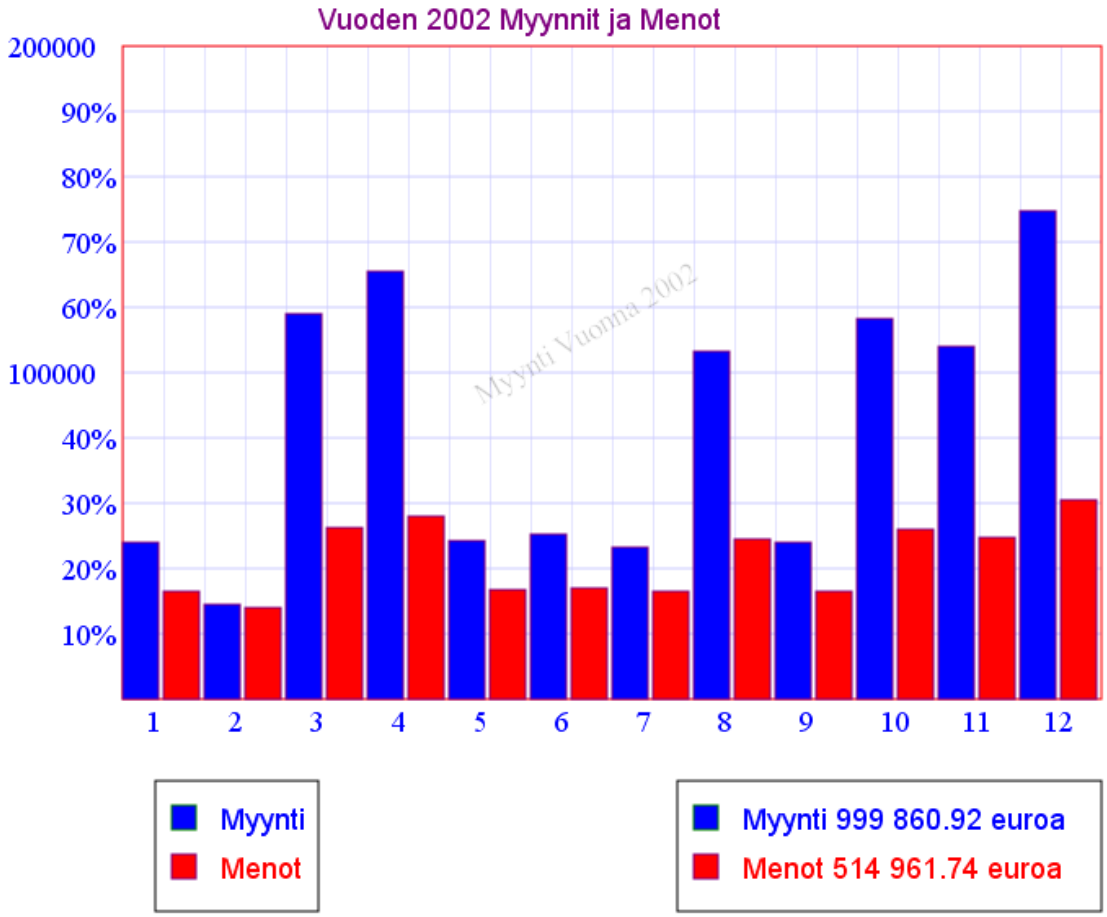
Testiympäristössä datan puskurointi erilaisessa tilanteessa normaalisti suoritetaan php-koodin puolella *echo*- taikka *print*-käskyllä. Koodin ollessa svg-muotoinen *echo* tai *print* komennon käyttäminen aiheuttaa aina automaattisesti virheen. Tästä syystä puskurointi pitää tapahtua aina kommenttiin tulostamisella, esimerkkinä tästä on ***echo ”<!-- $arvo --> ”;.***

Tietokannan yhdistäminen ei tuottanut minkäänlaista ongelmaa edes tietoturvan kannalta. Ainoa tietoturvaa heikentävän ratkaisun tein siinä, että data lähetetään GET-metodilla eikä Post-metodilla. GET-metodilla on helpompaa käsitellä liikkuvia parametrejä ja nopeampaa paikallistaa virheet.

Kun tiedetään millainen pylväsdiagrammi halutaan esittämään tietokannassa olevaa dataa, tulee seuraavaksi ongelmaksi

- suhteellisen korkeuden laskeminen
- oikeaan paikkaan sijoittaminen
- maksimiarvon määrittäminen dynaamisesti.

Kuvassa 9 on esimerkki lopputuloksesta. SVG käyttää käännettyä karteesista koordinaattijärjestelmää, jossa vasemmassa ylänurkassa on 0-piste, oikealle ja alaspäin on positiiviset luvut. Kuva 9 esittää vuoden 2002 myynnit ja menot.



Kuva 9. Tulostus palkkidiagrammista

4.1 Ratkaisu pylvään korkeuden laskentaan

Määriteltäessä kaavion korkeutta sovitaan, että palkin maksimikorkeus on 400 pikseliä. Näin saadaan yksi vertailukorkeus, jota käytetään suhteellisen korkeuden määrittämiseksi.

Ennen kuin saadaan laskettua suhteellista korkeutta, pitää pyöristää aina vuoden tai jakson suurin luku seuraavaan lähimpään tasalukemaan. Kuvassa se olisi 200 000.

Koska tavoitellaan pyöristystä ylöspäin lähimpään tasalukuun kaikilla luvuilla, niin pyöristyssääntöä pitää muokata omaksi uudeksi pyöristysfunktioksi. Tämä uusi pyöristysfunktio ottaa huomioon luvun pituuden ja näin saadaan aina oikea arvo. Ylöspäin olevan funktion matemaattista kaavaa ei voida suoraan esittää matemaattisella kaavalla, joten kuvassa 10 on pyöristysfunktio esitettynä PHP-ohjelmointikielellä.

```
26 function roundup ($value)
27 {
28     //return ceil($value*pow(10, $dp))/pow(10, $dp);
29
30     if(is_int($value))
31     {
32         $S = Le($value);
33         $val = ceil($value*pow(10, $S))/pow(10, $S);
34         return number_format($val,0,".","");
35     }
36
37     elseif(is_float($value))
38         {
39         // Kutsutaan apuparametri pilkkua varten
40         $T = Le($value)+3;
41         //+3 arvolla saadaan arvoja 100000, 200000,
```

Kuva 10. Pyöristysfunktio

Kuvassa 11 on ohjelmallinen määritelmä funktiossa *roundup* käytetystä *Le*-funktiosta. Le-funktio laskee annetun arvon pituuden merkkeinä, mikä on tärkeää ohjelmoinnin ja tulevien funktioiden toimivuuden kannalta.

```
 4
 5 function Le ($value)
 6 {
 7     //$value = $value(9,0);
 8     $R = strlen($value)-1;
 9     $R = $R-(2*$R);
10     $luku = $R;
11     return $luku;
12     /*
13     LE funktio laskee saadun arvon pituuden,
14     esim 123 456 789 ja 123456789 ovat eripituisia.
15     siksi onkin tarkkaa mikä arvo tietokannasta
16     tuodaan muutettavaksi ja missä muodossa.
17     onko arvolla desimaaleja
18     */
19 }
```

Kuva 11. Pituusfunktio

Käyttämällä näitä kahta funktiota saadaan ratkaistua palkin suhteellinen korkeus.

Kaavassa

$$\boldsymbol{h = korkeus\ pikselein}\text{ä}, \qquad \mathbf{M}_{\boldsymbol{\rho}} = \boldsymbol{maksimiarvo\ arvoista\ \rho_{\eta}},$$

$$\boldsymbol{\rho_{\eta}} = \boldsymbol{joukko\ arvoja}\ (\boldsymbol{\rho})\ , \boldsymbol{lukum}\text{ää}\boldsymbol{r}\text{ä}\ (\boldsymbol{\eta}),$$

$$\boldsymbol{\rho = laskettava\ arvo}$$

$$h = 400 * \left(1 - \frac{\mathrm{M}_{\rho_\eta} - \rho}{\mathrm{M}_{\rho_\eta}}\right) = 400 * \left(1 - \frac{160201 - 124557}{160201}\right) = 311$$

Kuvassa 12 on edellä oleva kaava esitettynä ohjelmointikielellä.

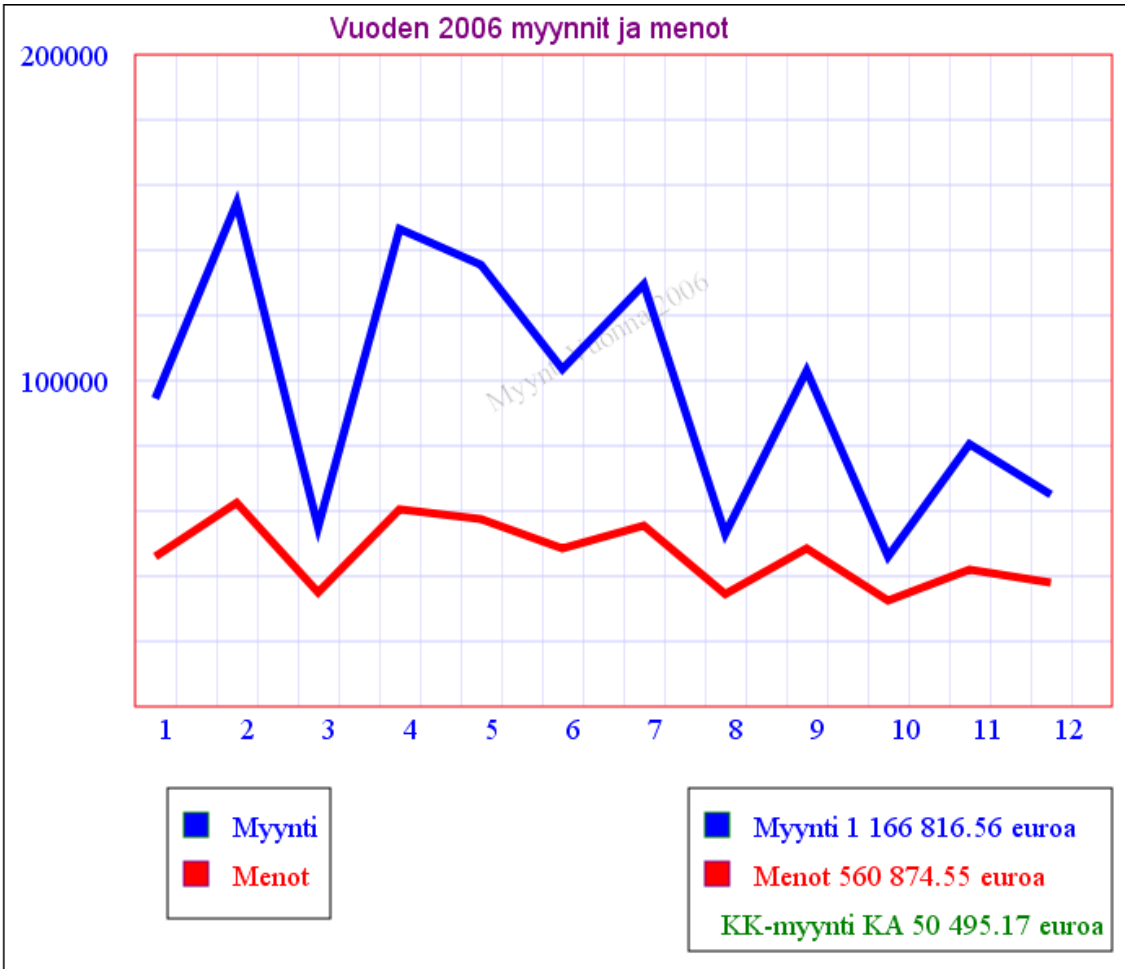
```
55
56 function palkki($MaxArvo, $Karvo)
57 {
58     /*
59     Laskee Annetun Max-arvon ja kannasta tulevan
60     arvon suhteen ja palauttaa arvon pikselimäärän
61     sekä prosenttuaalisen arvon esim 317 pix ja 51%
62     1027 * 768  == 400 korkeus
63     1200 * 1024 == 600 korkeus
64     */
65     $MA = intval($MaxArvo);
66     $KA = intval($Karvo);
67     //echo $MA . " " . $KA;
68     $Pro = 1-($MA-$KA)/$MA;
69    // echo "<br>" . $Pro . "<br>";
70     $Pix = 400 * $Pro;
71     Return round($Pix,0);
72 }
73
```

Kuva 12. Palkkifunktio

Näin ollen on ratkaistu kaksi ongelmaa kolmesta. Kolmannen eli oikeaan paikkaan sijoittaminen ratkaisu on suoraan saatujen arvojen summa. Koska palkin korkeus on maksimissaan 400 pikseliä korkea, sen alimmainen kohta on 400 pikseliä; käytettäessä palkkifunktiota se palauttaa palkinkorkeuden pikseleinä saatua arvoa 311 pikseliä. Peruskoulun matematiikalla pärjää tässä laskussa: 400 – 311 = 89 pikseliä. Joten palkin aloituskohta ylhäältä on 89 pikseliä ja pituus 311 pikseliä.

4.2 Ratkaisu viivadiagrammi

Luotaessa viivakaaviota kaavat pysyvät samanlaisina, paitsi sisäisen loopin rakenteeseen joudutaan huijaamaan loopin ulkopuolinen aloituskohta, edellisen pisteen vertauskohta ja loopin jälkeen lopetuslauseke.



Kuva 13. Viivadiagrammin esimerkki

Kuvassa 14 on ohjelmakoodista tehty ruutukaappaus, josta nähdään että ohjelmaa pitää hieman huijata, kun käytetään <path>-elementtiä silmukan sisällä. Tästä syystä yhdessä silmukassa voi tehdä vain yhden <path>-elementin. Kuvassa 14 nähdään rivillä 145 <g>-tagin aloitus silmukan ulkopuolella sekä rivillä 161 <path>-tagin muut määritelmät sekä lopetus kummallekin tageille.

```
135 // alustetaan VIIVA aloituskohta
136 // Pitää huijata sen verran konetta että loopin ulkopuolella
137 // haetaan ensimmäinen arvo josta mennään eteenpäin.
138
139 $aloituskohta = 0;
140 $max = roundup($MX);
141 $aloituskohta = palkki($max,$huijaus); //pikselimäärä
142 $pix = (400 - $aloituskohta); //saadaan aloituskohta
143
144 // VIIVA 1 ALKAA !!! eli myynnit
145 echo '<g transform="translate(80,30)"><path d="m12.5, '. $pix;
146
147 $prev = $pix;    // Tämä on Viivan ensimmäinen ja
148                  // loopin ulkopuolinen muuttuja ja arvo
149 while($row = mysql_fetch_row($result))
150 {
151     $myynti = $row[0];
152     $max    = roundup($MX);
153     $a      = palkki($max, $myynti);
154     $Viiva  = (400-$a) - $prev;
155     $Vertaus= (400-$a);
156
157 echo ' 50, '.$Viiva;
158 $prev = $Vertaus;
159 //prev on arvo johon viitataan aina vertauksessa edelliseen viivan kohtaan
160 }//while
161 echo '" style="fill:none;stroke:blue;stroke-width:5;fill-opacity:0;"/></g>';
162
163 // tähän päättyi VIIVA-koodi
```

Kuva 14. Viivan silmukka

4.3 Ratkaisu piirakkadiagrammiin

Alustavasti voisi olettaa, että ympyränmuotoisen diagrammin piirtäminen olisi helppoa kuin yrittäisi kopioida tekstiä netistä. Mutta todellisuus on totuutta ihmeellisempää. Ennen kuin mietitään ympyrän ongelmaa ohjelmoinnin kannalta, niin tähän kohtaan kuuluu seuraava Rick Cookin lausahdus (Cook 1989, 69):

"The three most dangerous things in the world are a programmer with a soldering iron, a hardware type with a program patch and a user with an idea."

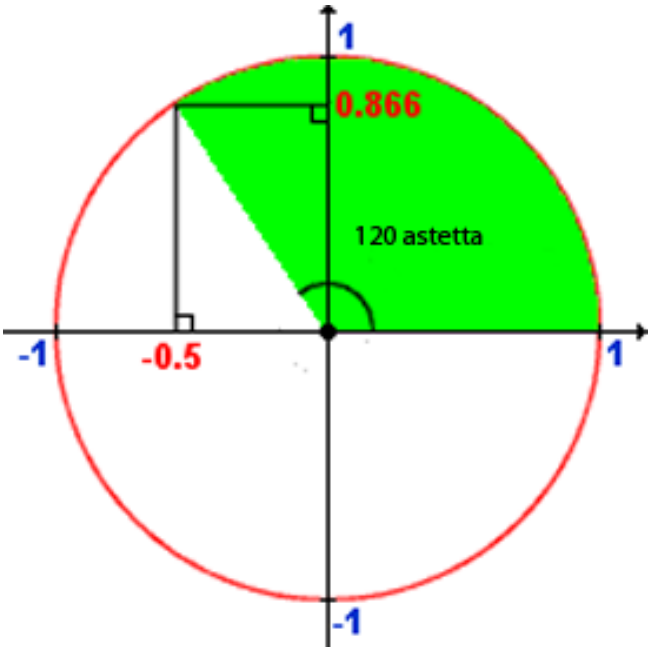
Sama ongelma koskee myös ympyränmuotoista diagrammia. Syy siihen on se, että laskettaessa piirrettävä viiva X- ja Y-suunnassa ei mene perusmatematiikan mukaisesti, vaan siihen pitää soveltaa korkeampaa trigonometristä matematiikkaa.

Kaavassa (ks. Majaniemi 1999, 39–40)

$$\boldsymbol{\alpha = sade, \beta = kulma, \pi = pii, \sum \beta = kulmien\ summa}$$

$$\boldsymbol{X = \alpha * \cos(\sum \beta * \frac{\pi}{180})}\ ,\ \boldsymbol{Y = \alpha * \sin(\sum \beta * \frac{\pi}{180})}$$

Kuvan 14 esimerkissä on haluttu saada tietää aloituskohdasta 120 astetta olevan kulmapisteen koordinaatit, että voidaan piirtää oikean mittainen kaari oikeaan paikkaan.



Kuva 15. Kaavan ratkaisua

Alkaessani tutkia piirakkadiagrammin tekemistä minulla ei ollut muuta lähdekoodia kuin käyttämässäni kirjassa oleva yksi malliesimerkki. Tämä malliesimerkki oli tehty ECMA-script nimisellä ohjelmoinnilla, jolla ei saa luotua dynaamista piirakkaa vaan ainoastaan staattisia piirakoita. Tämä ratkaisu ei sopinut ohjelmointityöhön. Muutaman päivän lukiessani ECMA:a aloin ymmärtää sen rakennetta ja miten se voitaisiin siirtää puhtaasti PHP-ohjelmointikielelle.

Monien kokeilujen ja tulosteiden jälkeen sain ratkaistua matemaattisen kaavan sekä nipun pienempiä ongelmia, mm. parametrien välittämistä ja laskemista. Muita ohjelmointia hidastavia ongelmia piirakkadiagrammiin liittyen oli luoda funktio, joka laskee annetusta parametrista kutsun tietokantaan, tiettyyn tauluun ja palauttaa sieltä vaaditut arvot takaisin. Funktio laskee niistä arvoista kokonaissumman sekä palauttaa arvojoukon kulma-asteita takaisin ohjelmakoodiin siihen kohtaan, joka sitä kutsui ja kykeni laskemaan ohjelmaa eteenpäin matemaattiseen kaavaan oikeat ratkaisut.

Jotta saadaan oikea kulma-aste, joudutaan saatuja arvoja muuttamaan tasakulmiksi. Esimerkiksi arvo 38.8881 pyöristetään arvoon 38.89. Tämä johtuu siitä, että saadaan toiminnot nopeiksi ja vältetään liian pitkän desimaaliluvun joutumista kaavaan. Tämä aiheuttaisi enemmän pyöristysvirhettä suuntaan tai toiseen, mistä voisi koitua isossa piirakkadiagrammissa värien välille valkoista tai viivojen symmetrian rikkoutuminen.

Kaavassa $\delta = osajoukon\ arvo,\ \sum \delta = osajoukon\ summa$

$$\boldsymbol{\alpha = 360 * (1 - ((\sum \delta - \delta) / \sum \delta))}$$

$$\alpha = 360 * (1 - \left(\frac{11449655{,}55 - 98640{,}79}{11449655{,}55}\right)) = 38{,}89$$

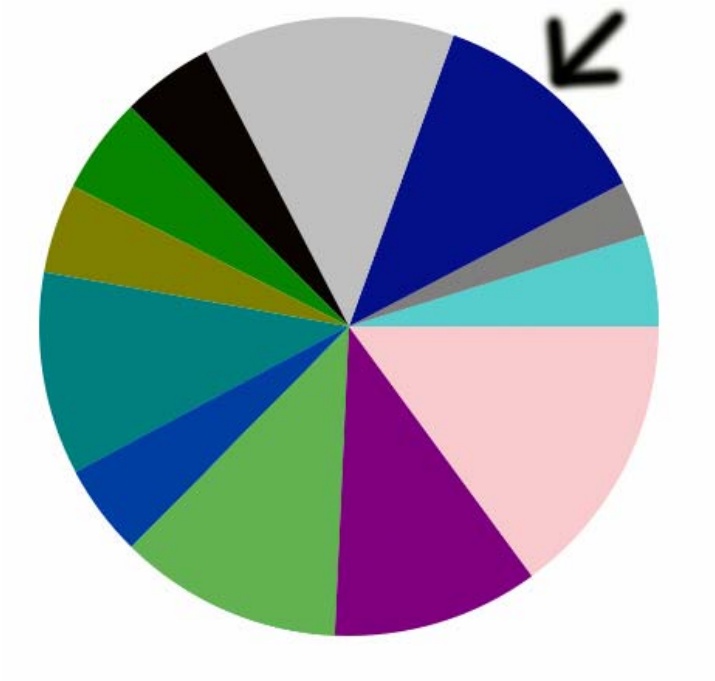
Pohdittuani, millä värillä mikäkin piirakansiivu olisi, niin helpottaakseni ohjelmointityötä loin yhden kiinteän arrayn-sisällön. Tämä sisältää 16 erilaista väriä ja näin ei tarvitse kuin kutsua funktiota, joka piirtää piirakan. Näin ollen ohjelma aloittaa värit samassa järjestyksessä. Käyttötarkoitukseen sopivat värit ovat sitten toinen ongelma, mutta se ei vaikuta ohjelmafunktion toimivuuteen.

Kun näin saadut kaksi arvoa ja osajoukon summa ovat selvillä, ne voidaan sijoittaa <path>-elementtiin yhdeksi osamääritelmäksi kokonaislausetta. Muiden osaelementtien arvoja voidaan laskea suoraan saatujen kahden arvon perus-

teella, jotka ovat yhteen ja vähennyslaskua. Lopullinen <path>-elementin sisältö näyttää esimerkiksi tältä:

```
<path d="M250,250 l177.15978795601, -92.813843425338
A200,200 0 0 0 317.7147400828,61.812024891282 L250,250 z"
style="fill: rgb(0, 0, 128);"/>
```

Kyseinen polku on kuvassa 16 tummansininen alue oikeassa yläkulmassa.

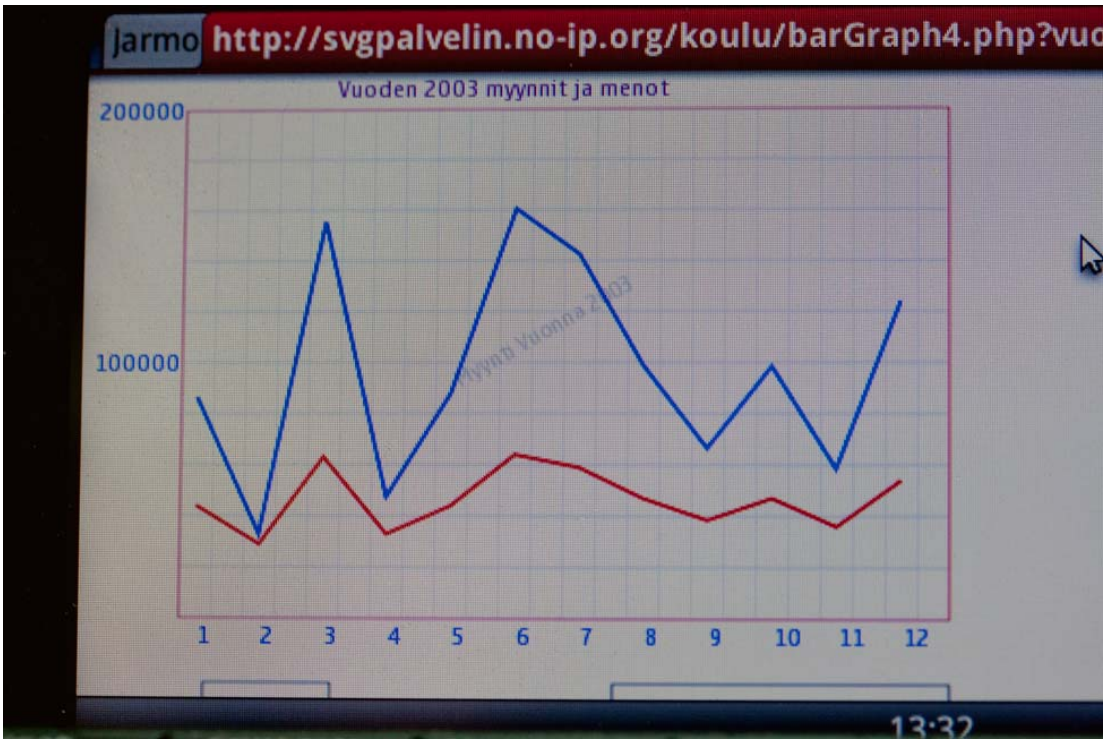


Kuva 16. Piirakkadiagrammi

5 SVG-näkymiä erilaisilla näytöillä

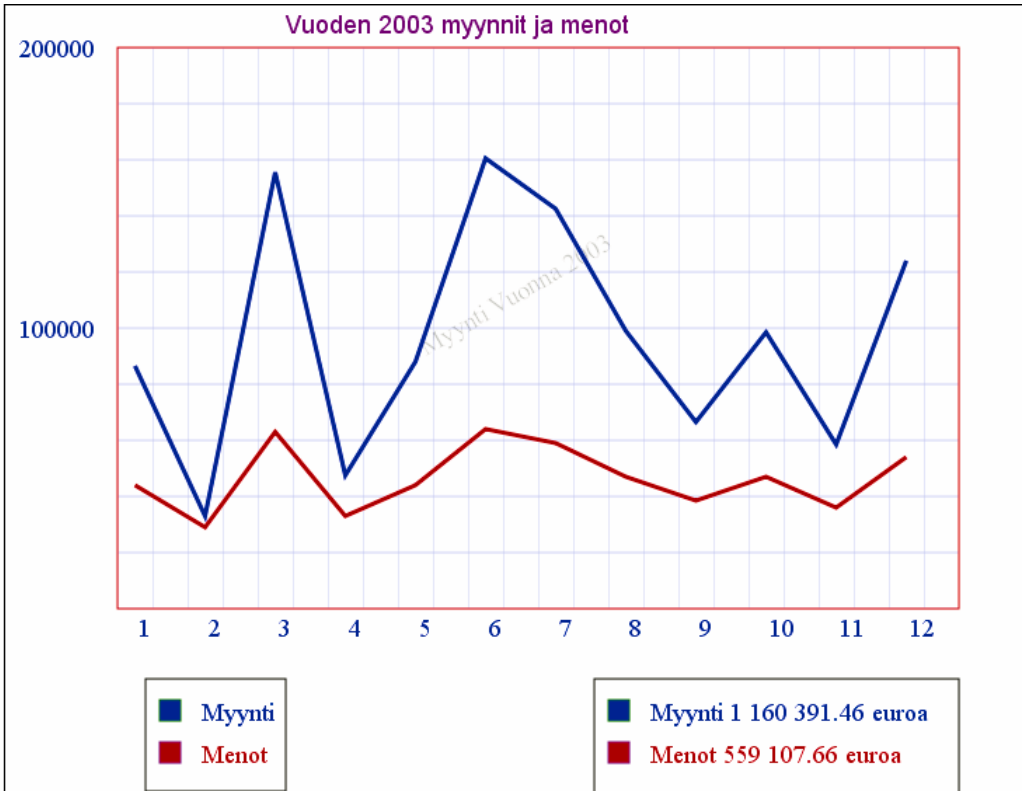
Kun aloin tehdä työtäni, päätin jo alussa, että tehdyt työt pitää näkyä myös melkein tavallisissa matkapuhelimissa. Nykypäivänä suurin osa valmistettavista matkapuhelimista on multimediapuhelimia ja osaltaan sitä kautta SVG-yhteensopivia, kuten Nokia Symbian-puhelimet, Apple iPhone ja muut Symbian-puhelimet. On olemassa muitakin matkapuhelinkäyttöjärjestelmiä, jotka tukevat SVG-standardia tai jotakin sen muunnelmaa. Erikoisin ja odotetuin OS on matkapuhelimeen Googlen kehittämä Android, mutta sen julkaisupäivä on epävarma ja siksi vain mainitsemisen arvoinen.

Kuvassa 17 on otettu Nokia E90-kommunikaattorista näyttökuva dokumenttikameralla. Nettiselaimena oli käytössä Opera 10.50, jolla on paras mobiilituki tällä hetkellä. Valintani Opera internetselaimeen on yksin se, että se on ainoa kunnollinen selain, joka toimii E90 matkapuhelimessa niin kuin sen pitää. Toistaiseksi Nokialla on heikko tuki muihin selaimiin, mutta kehitystä on tapahtunut.



Kuva 17. E90-ruutukaappaus, Opera-selaimella

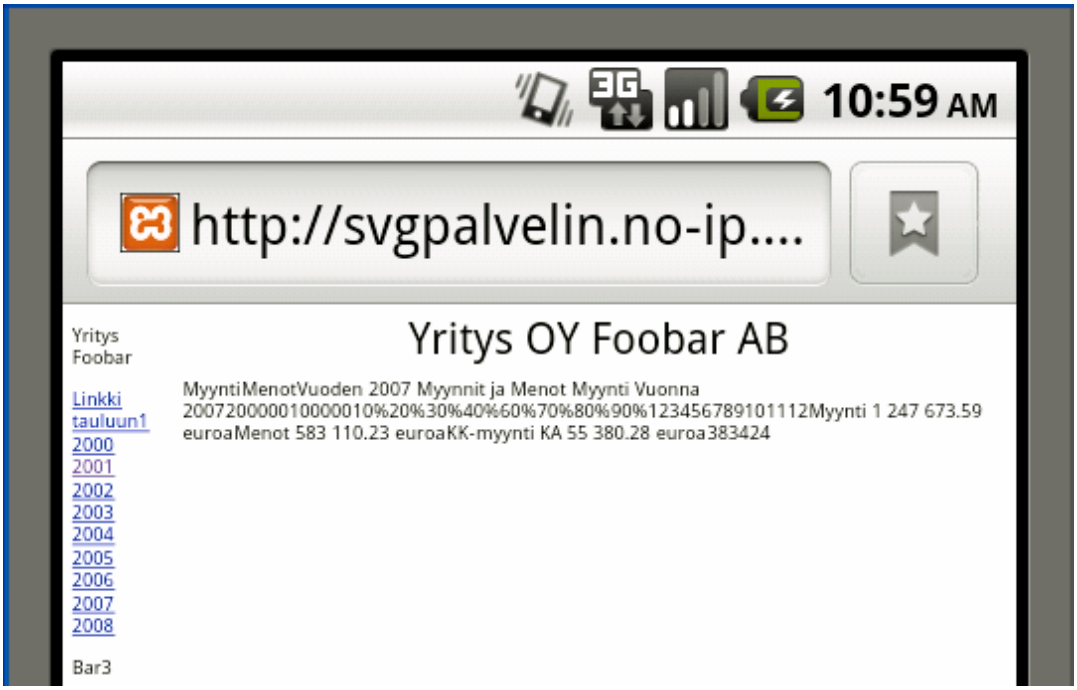
Kuvassa 18 on sama ruutukaappaus 22”:n näytöltä otettuna.



Kuva 18. Ruutukaappaus näytöltä, Opera-selaimella

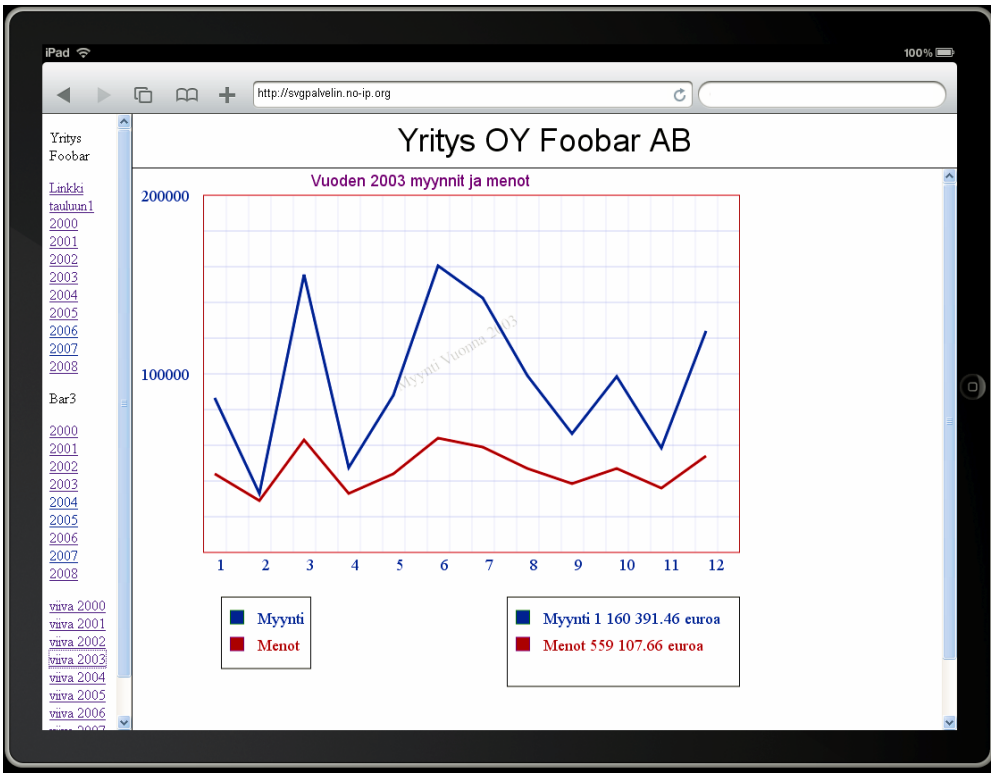
Kuvissa 19 ja 20 on käytetty Linus-käyttöjärjestelmää ja sen ohjelmaa nimeltään VirtualBox. VirtualBoxissa ajetaan kahta virtuaaliympäristöä, Googlen Androidia ja Applen iPadiä. Ajettaessa virtuaaliympäristöä voidaan testata näiden kahden laitteen ominaisuuksia ilman fyysistä laitetta, mutta ei voida tietää, toimiiko ja käyttäytyykö fyysinen laite testissä samalla tavalla kuin virtuaalinen laite, ei voida tietää. Mahdollisuutena on myös ajaa Applen iPhonea, mutta teknisistä ongelmista johtuen tätä mahdollisuutta ei ollut käyttää tätä testiä tehtäessä.

Kuvassa 19 on virtuaalisen Android-puhelimen ruutukaappaus. Toistaiseksi Android-puhelin ei tue SVG-tekniikkaa, mutta saatujen tietojen perusteella se on kehitteillä. Julkaisuajankohtaa tuelle ei ole kerrottu toistaiseksi kehitysfoorumeilla.



Kuva 19. Android-ruutukaappaus

Kuvassa 20 on virtuaalinen näkymä Applen iPad-laitteesta, jossa testasin SVG-testikoodia. iPad tukee uusimmalla käyttöjärjestelmällä suoraan SVG-koodia.



Kuva 20. iPad-ruutukaappaus

6 Johtopäätökset

Aloittaessani opinnäytteeni tyhjältä paperilta edeten SVG-tekniikalla toimivaan tietokantaohjelmistoon, joka piirtää lähes reaaliajassa (viivytyksettä) erilaiset diagrammit näytölle kuin näytölle, on vaatinut ponnistelua. Monenlaisten ongelmien ratkaiseminen ilman mitään kunnollista kirjallisuutta oli haastavaa.

Suurimpana haasteena pidin sitä, että SVG-standardi muuttui työn aikana useasti, tai siinä olevien luokkien rakenteet ja kutsut muuttuivat erilaisiksi. Tästä johtuen vielä edelleenkin standardin muuttuessa erilaiseksi, pitää työtä muuttaa ja korjata tarvittaessa. Suurin yksittäinen ongelma oli standardien muututtua saada PHP ymmärtämään SVG-sisältöistä dataa. Itse aktiivisella osallistumisella kehittäjien välisiin keskusteluihin eri foorumeilla tuotti tulosta ja nykyään käytössä oleva menetelmä ja standardi ovat osaksi minun kehittelemääni.

PHP toi eteen erilaisia haasteita. Syynä oli kaksi erilaista kuvaamisrakennetta: SVG:n tiukka XML-hierarkia ja PHP:n avoin hierarkia. Nämä rakenteet piti saada keskustelemaan keskenään ja parametrien piti liikkua näytöltä tietokantaan ja takaisin. Haasteena oli saada tämä yhdistelmä toimimaan moitteetta, ja monien kokeilujen jälkeen onnistuin. Suurimmaksi PHP:n haasteeksi voisin sanoa piirakkadiagrammin matemaattisten kaavojen muuttamista ohjelmakieleksi ja toisinpäin.

Kuvat 17 ja 18 sekä osittain kuva 20 todistavat, että SVG-tekniikka toimii samalla tavalla niin mobiilialustoilla kuin pöytätietokoneissa. Ainoa poikkeavuus on se, että pöytätietokoneissa on toistaiseksi isompi ja laajempi tuki erilaisille toiminnoille. Tämä erilaisuuden tietokoneiden ja mobiililaitteiden välillä huomaa, kun vertaillaan selaimia keskenään.

Tämän opinnäytetyöprosessin aikana opin, kuinka yhdistellä erilaisia ohjelmointikieliä ja kuinka saada aikaan haluttu lopputulos. Mielekkäimpänä oppina olen pitänyt sitä, että olen saanut tutustua muihin tämän ohjelmointikielen pioneereihin sekä olla vaikuttamassa standardin kehitykseen PHP:n puolella.

Ohjelmoinnin osalta olen oppinut lukemaan sekä kehittämään tilanteisiin sopivia funktioita, joilla voi helpottaa ohjelman tekemistä, sekä muokkaamaan jälkikäteen paremmin vallitsevaan ympäristöön.

Uutena oppina vanhassa ympäristössä on ollut erilaisten tietokantojen näkymien luominen ja käyttäminen tässä ohjelmointityössä. Se on vaatinut harjoittelua ja erilaisten työkalujen hankkimista sekä niiden tehokasta käyttämistä. Näistä työkaluista on ollut hyötyä myös opinnäytetyön ulkopuolisessa ohjelmoinnissa.

Mikäli alkaisin nyt tehdä tämän opinnäytetyön uusiksi, niin minun olisi helpompi ja nopeampi tehdä se valmiiksi. Se johtuisi siitä, että minun ei tarvitsisi käydä kirjeenvaihtoa sähköpostitse erilaisille tahoille, jotka päättävät ja kehittävät SVG-standardia. Voisin vain kirjoittaa web-sivun alkuun ne vaadittavat tiedot ja lopputulos olisi SVG-koodia. Sain itse olla mukana tässä kehityksessä, ja yhteistyössä saatiin aikaiseksi standardi, joka on vakiintunut sekä PHP- että SVG-ympäristössä. Mikäli voisin tehdä työn eri lailla, niin ottaisin vertailuun enemmän erilaisia matkapuhelimia. Mielenkiintoinen tuttavuus olisi tuleva Android-puhelin fyysisesti sekä iPhone, kuinka työni lopputulos näkyisi. Osittain sain testattua virtuaalialustan kautta iPad ja Androidin toimintaa. Tämä antaisi myös kokeilumahdollisuuksia käyttää Ajax-ohjelmointia ja tehdä vieläkin nopeampia ja tehokkaampia tietokantaratkaisuja verkon ylitse.

Itse toivoisin, että lähitulevaisuudessa joku tekisi SVG-kielellä osittain samantyylisen opinnäytetyön, jossa testiympäristössä olisi enemmän mobiililaitteita, mm. iPhone, Androidi, iPad, Nokian N800 Internet Tablet sekä muita e-ink-tuotteita. Opinnäytteen tekijä voisi testata niiden nopeutta ja käytettävyyttä sekä testata sitä, miten erilaiset tietokannat vaikuttavat tuloksiin.

LÄHTEET

Android 2010. Viitattu 20.4.2010 http://en.wikipedia.org/wiki/Android_(operating_system).

Apache 2010. Viitattu 25.5.2010 http://www.apache.org/.

Campesato, Oswald 2004. FUNDAMENTALS OF SVG PROGRAMMING: CONCEPTS TO SOURCE CODE. Hingham Massachusetts : Charles River Media.

Cook, Rick 1989. The Wizardly Compiled. Riverdale, NY: Baen Publishing Enterprises.

Majaniemi, Antti 1999. Geometria: geometriaa, trigonometriaa ja vektorilaskentaa, Kotka: Tieto-kotka.

MySql 2010. Viitattu 1.3.2008 http://www.mysql.com/about/.

Nykänen, Oskari 2007. SVG - skaalautuva vektorigrafiikka. Jyväskylä: Docendo.

Scalable Vector Graphics (SVG) 2010. Viitattu 1.3.2010 http://www.w3.org/Graphics/SVG/.

SVG-tutorial 2010. Viitattu 3.3.2010 http://www.w3schools.com/svg/default.asp.

SVG-wiki 2010. Viitattu 10.1.2010 http://en.wikipedia.org/wiki/Scalable_Vector_Graphics.

Xampp 2010. Viitattu 1.3.2010 http://www.apachefriends.org/en/xampp.html

XML 2010. Viitattu 24.5.2010 http://en.wikipedia.org/wiki/XML.