TIETOKANTAPOHJAINEN CSS -TEEMAEDITORI

LAHDEN AMMATTIKORKEAKOULU
Mediatekniikan koulutusohjelma
Teknisen visualisoinnin suuntautumisvaihtoehto
Opinnäytetyö
4.5.2009
Anu Törrö

Lahden ammattikorkeakoulu
Mediatekniikan koulutusohjelma

TÖRRÖ, ANU: Tietokantapohjainen CSS -teemaeditori

Teknisen visualisoinnin opinnäytetyö, 85 sivua

Kevät 2009

TIIVISTELMÄ

Tämä opinnäytetyö käsittelee moderneja webtekniikoita ja niiden ominaisuuksia webympäristössä toteutetuissa käyttöliittymissä. Työn jälkimmäisessä osassa käsitellään Rosendahl Digital Networks Oy:n Ajax-portaalin teemaeditori-lisäosan kehittämistä. Yritys tuottaa sovelluksia vaatetusteollisuuden yrityksille.

Teoriaosuudessa käsitellään CSS-tyyliohjeita: mitä ne ovat tänä päivänä ja mihin suuntaan niiden kehitys on menossa. Teoriaosuudessa käsitellään myös yleisemmällä tasolla käytettävyyttä huomioiden websovellusten suunnittelua. Työssä pohditaan myös graafisen käyttöliittymän hyviä ominaisuuksia ja sitä, millä tavoin ja kuinka tärkeää graafinen toteutus on käytettävyyden osalta. Työssä otetaan huomioon myös se, millä tavalla CSS on osa nykyaikaista dynaamista webkehitystä muiden webtekniikoiden kanssa.

Opinnäytetyön empiirinen osuus muodostuu Rosendahl Digital Networks Oy:lle toteutetusta CSS-teemaeditorista. Teemaeditori toteutettiin erillisenä lisäosana yrityksen selainpohjaisia sovelluksia kokoavaan portaaliympäristöön. Editorin tarkoitus on helpottaa ja tehostaa portaalin käyttöliittymän ulkoasuun vaikuttavien teemojen luomista. Johtopäätöksien perusteella modernin websovelluksen toteuttamiseen tarvitaan laaja skaala eri alojen tuntemusta. On otettava myös huomioon websovellusta kehitettäessä, että sille on varattava tarpeeksi aikaa sekä muita tarvittavia resursseja, mikäli halutaan päästä hyvään lopputulokseen.

Avainsanat: CSS, käytettävyys, graafinen käyttöliittymä, Ajax, relaatiotietokannat, SQL, JavaScript, DOM

Lahti University of Applied Sciences
Degree Programme in Media Technology

TÖRRÖ, ANU: Database-based CSS - theme editor

Bachelor´s Thesis in Visualization Engineering, 85 pages

Spring 2009

ABSTRACT

This thesis deals with modern web techniques and their use in graphic user interfaces which are implemented in the web environment. In addition, the thesis deals with a theme editor based on Cascading Style Sheets, which was implemented as an expansion for the Ajax-based portal system of Rosendahl Digital Networks Ltd. The company produces software solutions for the fashion industry.

The theory section is roughly divided into four sections.The first section introduces Cascading Style Sheets and describes how CSS has developed and what future prospects there are. The second and third sections explore usability in general and usability as the basis of designing graphic user interfaces. The fourth section presents the most common modern web techniques.

The empirical part of the thesis deals with a CSS theme editor made for Rosendahl Digital Networks Ltd. The theme editor was produced as an expansion for the company's existing web-based portal. The main purpose of the theme editor was to facilitate and optimize the making of the themes within the portal. As a conclusion, it can be said, that a broad variety of skills from different fields were needed to produce the desired outcome. Issues concerning time, money and other resources also have to be taken into account.

Key words: CSS, usability, graphic user interface, Ajax, relational database, SQL, JavaScript, DOM

SISÄLLYS

1 JOHDANTO 1

2 CSS -MÄÄRITELMÄT 2
2.1 Johdatus tyylikieliin ja CSS:n historiaan 2
2.2 Miksi CSS:ää tulisi käyttää 3
2.3 Määrittelyt ja keskeiset periaatteet 4
2.4 Selaintuki 14
2.5 Mediasääntö ja -tyypit 15
2.6 CSS3 -luonnos 18
2.6.1 Kehitys 18
2.6.2 Modulaarisuus 20

3 DYNAAMISET WEBTEKNIIKAT 21
3.1 JavaScript 21
3.2 Ajax 22
3.2.1 Ajaxin määrittely ja sen ongelmallisuus 22
3.2.2 Ajaxin käyttäminen 23
3.2.3 Toimintaidea 25
3.2.4 XML 26
3.2.5 XMLHttpRequest-objekti 27
3.2.6 DOM-malli 28
3.2.7 JavaScript ja CSS 29
3.3 PHP 29
3.4 Relaatiotietokannat 31
3.4.1 Yleisesti 31
3.4.2 Suunnittelu 31
3.4.3 SQL 34
3.4.4 Toteutus 36

4 KÄYTETTÄVYYS OSANA OHJELMISTON SUUNNITTELUA 37
4.1 Käytettävyyden tarpeellisuus 37
4.2 Heuristinen arviointi 39
4.3 Käytettävyys visuaalisen suunnittelun osana 40
4.4 Käyttäjän tunteminen 40
4.5 Käyttäjäkeskeiset suunnittelumenetelmät 41

5 GRAAFISEN KÄYTTÖLIITTYMÄN OMINAISUUDET 42
5.1 Määrittely, Graphic User Interface 42
5.2 Kuva vai teksti 42
5.3 Värit 43
5.4 Käyttäjän huomion ohjaaminen 44
5.5 Sommittelu 45
5.6 Visuaalinen tasapaino 45
5.7 Prototyypit 46
5.8 Testaaminen 48

6 CSS-EDITORI -SOVELLUKSIA 49

7 CASE: ROSENDAHL DIGITAL NETWORKS OY - CSS - TEEMAEDITORI 57
7.1 Tietoja yrityksestä 57
7.2 Toimintaympäristö 57
7.2.1 Portaali 57
7.2.2 Lähtökohdat CSS-teemoille ja kansiorakenteelle 59
7.2.3 Teemat 60
7.3 Vaatimusmäärittely 61
7.4 Työn kulku 65
7.5 Tekninen toteutus 67
7.6 Casen arviointi 68

8 YHTEENVETO 69

LÄHTEET 72

LIITTEET 78

LYHENTEET JA TERMIT:

ActiveX	Nimitys Microsoftin Windows-käyttöjärjestelmässä käytetylle Component Object Model (COM)-tekniikalle. COM-komponentit ovat uudelleenkäytettäviä ohjelmistokomponentteja.
Ajax	Asynchronous JavaScript and XML, eri webtekniikoista koottu tekniikka, jonka avulla luodaan interaktiivisia websovelluksia.
ASCII	American Standard Code for Information Interchange, tietokoneiden merkistö, joka sisältää englannin kielen kirjaimet, numerot, joukon välimerkkejä ja ohjauskoodeja.
Binääri	Binäärijärjestelmä, lukujärjestelmä , jonka kantaluku on kaksi.
CSS	Cascading Style Sheets, erityisesti WWW-dokumenteille kehitetty tyyliohjeiden laji.
Chrome	Googlen kehittämä avoimen lähdekoodin WWW-selain.
DHTML	Dynamic HTML, kokoelma eri tekniikoista, joiden avulla voidaan luoda interaktiivisia WWW-sivuja.
DOM	Document Object Model, ohjelmointirajapinta.
HTML	Hyper Text Markup Language, avoimesti standardoitu kuvauskieli, jonka avulla voidaan rakentaa WWW-sivuja.
ID	Identifier, tietoteknisessä merkityksessä tarkoittaa yksilöivää tunnistetta.
ISO	International Organization for Standardization, kansainvälinen standardismijärjestö.
Iteratiivinen	Tuotantoprosessin malli (tässä yhteydessä), missä suunnittelu ja toteutus tehdään pienissä osissa toistuvasti.
MIME	Multipurpose Internet Mail Extension, määrittelee kattavan joukon sisältötyyppejä, joita käytetään muun muassa http:ssa ilmaisemaan välitetyn datan muotoa.
MySQL	Avoimen lähdekoodin SQL-tietokannan hallintajärjestelmä.
Pannaus	Panorointi, sivuttaissuuntainen kameraliike, toisin sanoen liikuttamista sivuttaissuunnassa.
Parserointi	Merkkijonon sisältämien muuttujien ja erikoissymbolien käsittely.

PNG	Portable Network Graphics, on häviötön bittikarttagrafiikan tallennusformaatti.
PostGreSQL	Avoimen lähdekoodin SQL-tietokannan hallintajärjestelmä.
RIA	Rich Internet Application, työpöytäsovelluksen kaltainen websovellus.
SVG	Scalable Vector Graphichs, kaksiulotteisten vektorikuvien kuvauskieli.
UA	User Agent, yleistermi kaikille ohjelmille, joita käytetään Websivuston avaamiseen.
W3C	World Wide Web Consortium, yhteenliittymä, joka ylläpitää ja kehittää WWW:n standardeja.
Widgetti	(eng.) Widget, pieni ohjelma jolla voi olla jokin oma käyttötarkoitus. Esimerkiksi tietokoneen työpöydällä oleva muistilappusovellus on widgetti.
XML	Extesible Markup Language, W3C:n standardisoima tekniikka tekstimuotoisten dokumenttien merkkaamiseen ja käsittelyyn.
YUI	Yahoo User Interface Library, avoimen lähdekoodin JavaScriptkirjasto interaktiivisten websovellusten rakentamiseen.

1 JOHDANTO

World Wide Webissä on siirrytty uuteen aikakauteen, sen toiseen vaiheeseen. Ennen tätä vaihetta Web oli staattisempi, mutta nyt uuteen vaiheeseen siirryttyä kaikesta on tullut dynaamisempaa. Web-pohjaisista sovelluksista on kehitetty toiminnallisempia, haluttavampia ja yhteisöllisempiä ja ihmiset haluavat olla enemmän vuorovaikutuksessa Webin sisältöön eivätkä halua pelkästään olla vain ulkopuolisia tarkkailijoita. Tämä ilmiö tunnetaan paremmin termillä Web 2.0. Ilmiöön liittyvien periaatteiden mukaan Web-sivut eivät ole pelkkiä tietovarastoja vaan vuorovaikutteisia sovelluksia, joiden kautta tietoa saadaan turvallisesti sisään sekä ulos. Dynaamisuus on pääasiassa koettu positiivisena asiana. Halutaan mielellään päästä vaikuttamaan asioihin ja halutaan, että myös oma yksilöllinen kädenjälki näkyy suuressa internetissä. Uusien tekniikoiden avulla on mahdollista kehittää vuorovaikutteisia selainpohjaisia sovelluksia, jotka alkavat enemmän ja enemmän muistuttaa työpöytäsovelluksia.

CSS on olennainen osa modernia websuunnittelua. Yhdessä JavaScriptin ja muiden tunnettujen web-tekniikoiden kanssa CSS:n avulla voidaan rakentaa moderneja ratkaisuja uuden aikakauden Webiin. CSS-tyyleillä tehtyjen Web-sivujen taitot ovat antaneet tunnusomaisen ulkoasun Web 2.0-ilmiön mukaisille sivustoille. Lisäksi modernin web-suunnittelun keskeisenä tekniikkana Ajax on varmistanut innovaatisuudellaan uuden aikakauden alkaneeksi webmaailmassa.

Tässä opinnäytetyössä käsitellään erilaisia yleisempiä web-ohjelmointikieliä ja -tekniikoita, kuten Ajax, JavaScript ja CSS, joita käytetään modernien websovellusten luomiseen. Lisäksi perehdytään käytettävyyteen käyttöliittymän suunnittelussa sekä siihen, mitä pitäisi ottaa huomioon, kun suunnitellaan graafista käyttöliittymää websovellukselle. Yksityiskohtaisempaa tarkastelua saa osakseen CSS-kielen kehitys: mitä se on nykyään ja mihin se on menossa.

Case-osuudessa kehitellään Rosendahl Digital Networks Oy:n tuotannossa olevaan selainpohjaiseen portaalijärjestelmään dynaamista CSS-editoria, jonka avulla hallitaan portaalijärjestelmässä käytettäviä teemoja. Lähtökohtaisesti siitä on tarkoitus kehittää toimiva ja mahdollisesti myös ainoa ratkaisu hallita portaalijärjestelmän ulkoasuteemoja.

2 CSS -MÄÄRITELMÄT

2.1 Johdatus tyylikieliin ja CSS:n historiaan

Korpela (2008, 53) kertoo kuinka ennen CSS:ää ja osittain myös sen kehityksen rinnalla ovat kulkeneet tekstinkäsittelyohjelmien tyylimäärittelyt, esimerkiksi MS Word. Tekstinkäsittelyohjelmien tyylimäärittelyjen ideana on, että niissä voi asettaa esimerkiksi eritasoisten otsikoiden fonttilajin, fonttikoon sekä reunukset ja valita ohjelman valikosta halutun muotoilun esimerkiksi valinnalla ”Otsikko 2”, joka olkoon tässä tapauksessa 2. tason otsikko. Ajatuksellisen yhteyden lisäksi on myös tekninen yhteys. Jos Word ohjelmassa tiedosto tallennetaan nimellä ja tallennusmuodoksi valitaan Web-sivu, niin Wordissa tehdyt tyylimäärittelyt muuntuvat osittain CSS-määrittelyiksi.

Tyylikielet on luotu määrittelemään rakenteellisten dokumenttien esitysmuoto. Rakenteellinen dokumentti voi olla esimerkiksi HTML-dokumentti, minkä osiot on selkeästi määritelty sekä kategorisoitu. Kuitenkaan tyylikielet eivät rajoitu pelkästään HTML-dokumenttien käsittelyyn tai tyylikielistä puhuttaessa ei välttämättä tarkoiteta pelkästään CSS:ää.

Tyylimäärittely on joukko jollain tyylikielellä, kuten CSS, DSSL tai XSL, kirjoitettuja lauseita, jonka avulla määritellään dokumentin esitys- tai asettelumalli ja ulkoasu. Web-kehityksessä käytettävien tyylikielten kohdalla valitaan yleensä CSS ja XSL -muunnosten välillä. CSS -kieltä voidaan käyttää HTML-, XHTML- sekä XML-dokumenttien ulkoasun määrittämiseen, sen sijaan XSL-muunnoksia käytetään vain XML-dokumenttien ulkoasun määrittämiseen. (2kmediat.com 2000-2008c)

CSS:n avulla voidaan määritellä suhteellisen yksinkertaista syntaksia käyttäen muun muassa HTML-, XHTML- ja XML -dokumenttien ulkoasuja. Toisin sanoen selaimelle voidaan esittää dokumenttien ulkoasua koskevia ehdotuksia. CSS mahdollistaa melko joustavasti WWW-sivujen ulkoasun muokkaamisen. Vaikka ulkoasuun voidaan vaikuttaa joillakin HTML-elementeillä sekä rakenteilla, on CSS:n käyttö paljon tehokkaampaa. Lisäksi CSS tarjoaa paljon enemmän muotoilumahdollisuuksia kuin esimerkiksi HTML. (2kmediat.com 2000-2008c; Korpela, J.K, 2008, XII ja 2)

W3C loi CSS-standardit, joiden tarkoitus oli lisätä tyylien käyttömahdollisuuksia. Standardoinnin vuoksi tyylimallien käyttö on mahdollista yhä useammassa selaimessa ja ne toimivat keskenään samalla tavalla.

CSS:n kehitystyö alkoi 1990-luvun puolivälissä, kun kävi ilmeiseksi, että HTML -kieli ei enää vastannut sille asetettuja vaatimuksia. Selainsodan Netscapen ja IE:n välillä riehuessa huipussaan molemmat selainvalmistajat lisäsivät HTML–kieleen selainkohtaisia piirteitä, joiden avulla voitiin määritellä muun muassa tekstin ulkoasu ja erilaisia korostuskeinoja. Lopputuloksena oli sivustojen toimimattomuus eri selaimilla sekä HTML-kielen jakaantuminen selainkohtaisiin versioihin. (2kmediat.com 2000-2008c.)

Ratkaisuksi tähän kaaostilanteeseen kehitettiin CSS, jonka tavoitteena oli HTML-kielen rakenteellisuuden palauttaminen ja selainkohtaisten HTML-murteiden poistaminen. Tämä tavoite on saavutettu hyvin pitkälti XHTML-kielen myötä, sen sijaan HTML-kielen viimeisin versio 4.0 ei tähän pystynyt. Lisäksi CSS:n tavoitteena oli luoda yhtenäinen selainriippumaton sekä käyttöjärjestelmästä ja mediasta riippumaton tapa määritellä Web-dokumenttien ulkoasu. Tämä tavoite on nykyisellään saavutettu hyvin pitkälti, tosin selaimissa on edelleen hyvin paljon eriasteisia toteutuksia tai virheitä. (2kmediat.com 2000-2008c.)

2.2 Miksi CSS:ää tulisi käyttää

Alla olevan koodiesimerkin ylemmällä rivillä oleva HTML-koodi sisältää muotoiluun erikoistuneita komentoja tuottaakseen 12 pt lihavoitua tekstiä punaisella Verdana-fontilla. Alapuolella oleva XHMTL-koodi ainoastaan viittaa CSS-tyylimäärittelyn luokkaan nimeltä huomio. CSS:n käytön avulla on mahdollista vähentää sivun esittämisessä tarvittavan HTML/XHTML-koodin määrää jopa 60 %. (2kmediat.com 2000-2008c.)

```
<p><font face="Verdana" color="red" size="12pt"><b> Tärkeä kom-
mentti! </b></font></p>



<p class="huomio">Tärkeä kommentti!</p>
```

Perinteisesti HTML-kieli on tarkoitettu esittämään dokumentin rakenne sekä sisältö, ja muuten dokumentin esitystapa jätetään selaimen hoidettavaksi. CSS:n avulla dokumentin rakenteen ja sisällön voi erottaa esitystavasta.

CSS-kieli antaa mahdollisuuden suunnittelijoiden muuttaa mieltä ja toimia luovasti. Mikäli sivuston elementtien ulkoasu on määritelty tyylisivujen avulla, silloin sivuston ulkoasun muuttaminen tapahtuu tyylisivujen määrityksiä muuttamalla. Muutokset astuvat voimaan automaattisesti sivuston kaikille sivuille. CSS-kielen ansiosta verkkosuunnittelijat voivat viettää enemmän aikaa suunnittelutyössä sen sijaan, että käyttäisivät aikaa HTM-kielen rajoitteiden kanssa työskennellessä. (Smith & Negrino 2007, 263.)

Ulkoisten CSS–tyylisivujen paras puoli on se, että sivujen ulkoasua voidaan vaihtaa tyylisivun tyylimäärityksiä muuttamalla. Näin varsinainen HTML–koodi voidaan jättää ennalleen. (Smith & Negrino 2007, 265.)

2.3 Määrittelyt ja keskeiset periaatteet

W3C on muodollisesti määritellyt kahdet eri spesifikaatiot kaskadisille tyylitiedostoille: CSS1 ja CSS2. CSS1 julkaistiin vuonna 1996, ja se määritteli yksinkertaisen muotoilumallin pääosin näyttöpohjaisille esityksille. CSS1:llä on noin 50 eri ominaisuutta, esimerkiksi color ja font-size. CSS2 viimeisteltiin toukokuussa 1998 perustuen CSS1:een. CSS2 sisältää kaikki CSS1:n ominaisuudet ja lisää vielä noin. 70 uutta ominaisuutta, kuten esimerkiksi ominaisuudet kuvata ääneen perustuvia esityksiä ja tehdä sivunvaihtoja. (Wium Lie, Boss 1999, 32.)

CSS-tyylimäärittely voi sijaita joko ulkoisessa CSS-tyylitiedostossa tai olla upotettu osaksi HTML/XHTML tai XML-dokumenttia. Mikäli CSS-määrittelyt on upotettu osaksi dokumenttia, ne voidaan tällöin tehdä joko dokumentin alussa erillisen style -elementin sisään, tai määritykset voidaan tehdä elementtikohtaisesti (esimerkiksi <p style=”font-family: Arial”>). Toki joitakin ulkoasuseikkoja voidaan määrittää jopa HTML:ssä itsessään, mutta se ei ole välttämättä nykystandardien mukaista tai muutenkaan kannattavaa. Aina, kun uusia www-sivuja luodaan, kannattaisi ulkoasua koskevat seikat esittää erillisessä tiedostossa useastakin eri

syystä. Ulkoisen CSS-tyylitiedoston tiedostopääte on ”.css”. On joukko eri syitä miksi kannattaa käyttää erillistä tiedostoa:

- Ulkoasun suunnittelu ja etenkin sen viimeistely voidaan jättää myöhempään vaiheeseen. Sivu saadaan tällöin nopeammin käyttöön.
- Ulkoasun tekeminen voidaan jättää eri ihmisten tehtäväksi kuin rakenteen ja sisällön. Tästä on etua varsinkin silloin, kun ulkoasulle asetetaan suuria vaatimuksia, jolloin sen tekeminen vaatii oman ammattitaitonsa.
- Voidaan kokeilla erilaisia ulkoasuja koskematta itse sivuihin, eli lähdekoodiin.
- Samaa tyylitiedostoa voidaan käyttää eri HTML-tiedostoille sen sijaan, että kopioitaisiin lähdekoodia.
- Ulkoasua voidaan muuttaa keskitetysti, vain tyylitiedostoa muuttamalla.
- Käyttäjien valittaviksi voidaan jopa antaa vaihtoehtoisia ulkoasuja, joko eri tarkoituksiin, esimerkiksi ruudulle ja tulostukseen, tai vain maun mukaan.
- Tässä lähestymistavassa HTML-tiedostosta tulee sellainen, että se toimii tyylisäännöstöstä riippumattakin ja eri tyylisäännöstöjen kanssa. Tällöin se toimii myös sellaisissa käyttötilanteissa, jotka poikkeavat tavanomaisesta, tai jopa sellaisissa, joita ei vielä ole edes keksitty.

(Korpela & Linjama 2005, 40)

Useimmat web-kehittäjät ovat alkaneet käyttää CSS-kieltä vasta 2000 -luvun alkupuolella. CSS:n tehokkaan hyödyntämisen kannalta on olennaista ymmärtää, että tarjolla on useita eri versioita, ja että selainten tuki CSS:n eri versioilla ja niiden sisältämille ominaisuuksille on hyvinkin vaihteleva. (2kmediat.com 2000-2008c; 2kmediat.com 2000-2008d.)

TAULUKKO 1. CSS–kielen eri versioita. (2kmediat.com 2000 – 2008d.)

CSS 1	Määrittelee CSS:n keskeisen syntaksin ja ominaisuudet: yksiköt, fontit, ja laatikkomallin. Sai W3C:n suosituksen standardiksi v. 1996 (uudistettu). Nykyselainten tuki CSS 1 määrittelylle on erittäin laaja, useissa selaimissa täydellinen tuki.
CSS 2 / 2.1	Sisältää CSS1:n esittelemän syntaksin ja ominaisuudet, ja laajentaa näitä mm. Lisäämällä sijoittelumalleille, dynaamisesti generoidulle sisällölle, mediatyypeillä ja kansainvälisyyteen liittyvillä piirteillä. W3C:n suositus standardiksi v. 1998. Vuonna 2004 CSS2 avattiin korjauksia/tarkennuksia varten ja se odottaa hyväksyntää standardoimiselle. Nykyselainten tuki CSS 2 määrittelylle on vaihteleva, vain muutamissa selaimissa on täydellinen tai lähes täydellinen tuki.
CSS 3	CSS 3 sisältää aiempien määrittelyjen syntaksin ja ominaisuudet, ja laajentaa näitä muun muassa uusilla valitsinmalleilla, käyttäytymismalleilla ynnä muilla kehittyneillä piirteillä. Kehitystyö alkoi 1998 ja on edelleen luonnosasteella. Nykyselainten tuki CSS3:lle on hyvin vaihteleva.
CSS -MP	CSS Mobile Profile on mobiililaitteille suunnattu CSS:n alamurre, jossa on otettu huomioon mm. Mobiililaitteiden rajoitettu suorituskyky. Vaikka CSS-MP on edelleen niin kutsutulla luonnos-asteella, monet mobiiliselaimet tukevat sitä.

Uusien CSS-standardien myötä CSS:n käyttötarkoitus on laajentunut myös muihin medioihin kuin tietokoneen ruutuun, esimerkkeinä tulostettavien dokumenttien asetuksiin sekä mobiililaitteisiin. CSS-määritysten lähtökohtana on kuitenkin aina webympäristö, jota muunnetaan eri käyttötarpeisiin.

Kaikkien muiden ohjelmointimenetelmien tavoin myös CSS perustuu joukolle sääntöjä, jotka määrittävät, miksi asiat toimivat, kuten ne toimivat. Toisin sanoen miksi joitain asioita tulisi tehdä tai joitain ei tai miksi jotkin asiat toimivat. Seuraavissa kappaleissa käydään läpi näitä CSS:n määrittäviä sääntöjä hieman yksityiskohtaisemmin. (2kmediat.com 2000-2008e.)

Tärkein sääntö CSS:ssä on se, että se kuvaa dokumentin esitystavan. CSS -tyylimäärittelyt sisältävät informaatiota rakenteellista kuvauskieltä (kuten HTML, XHTML tai XML) hyödyntävän dokumentin esitystavasta ja sitä kautta täydentävät asiakirjaa. Tämä tarkoitta sitä, ettei CSS-määrittelyjen avulla tulisi pyrkiä esittämään muuta kuin dokumentin ja sen sisällön esitystapa.

Toinen sääntö on se, että CSS on eteen- ja taaksepäin yhteensopiva. Se on sääntö, joka löytyy mistä tahansa tietokoneella suoritettavassa kielessä. Dokumenttien, jotka hyödyntävät CSS-määrityksiä, tulee toimia sekä erittäin vanhoissa ja uusinta teknologiaa edustavissa UA-laitteissa (esimerkiksi Web-selain) ilman virhetilanteita. Säännön henkeen kuuluu myös alusta- ja mediariippumattomuus. Tämän mukaan siis CSS-tyylimäärittelyjen avulla sivun ulkoasun voi määritellä esimerkiksi näyttöä, tulostusta sekä useita muita medioita varten. (2kmediat.com 2000-2008e.)

Käytännössä asian toteutumisesta vastaa UA-laite, esimerkiksi Web-selain. CSS-määrittelyn mukaisesti UA-laitteen tulisi hyväksyä vain ja ainoastaan ymmärtämänsä, validit tyylisäännöt, kaikki muut säännöt tulisi ohittaa. Jo yksikin looginen tai merkkivirhe HTML/XHTML/XML-dokumentissa tai CSS-tyylimäärittelyssä voi aiheuttaa merkittäviä muutoksia dokumentin ulkoasuun ja toimivuuteen eri selaimissa. Tämän vuoksi CSS-tyylimäärittelyt pitäisi aina validoida ennen käyttöönottamista. Kaikissa selaimissa on kuitenkin piirteitä, joiden vuoksi ne poikkeavat tästä säännöstä ja pyrkivät toteuttamaan virheellisiä sääntöjä. Tämä on johtanut muun muassa niin kutsuttujen selainkohtaisten CSS-hackien syntymiseen erilaisten ulkoasutehosteiden aikaansaamiseen. Näiden käyttö ei kuitenkaan ole suotavaa. (2kmediat.com 2000-2008e.)

Kolmannen säännön mukaan CSS on yksinkertainen ja tehokas. Tämä yksinkertaisuus erottaa CSS:n muista tyylikielistä. Se on erittäin minimalistinen, joustava ja piirteiltään monipuolinen tapa kuvata rakenteellisen dokumentin ulkoasu. Koko dokumentin ulkoasuun liittyvien määrittelyjen tuottaminen onnistuu jo parilla yksinkertaisella koodirivillä. Kuitenkin laajamuotoiset kokonaisuudet hyvin tarkkaan määritetyllä ulkoasulla saattavat vaatia jopa tuhansia rivejä CSS-koodia. Silti se on yksinkertaisempi tapa hallita dokumentin ulkoasua, kuin jos sen yrittäisi toteuttaa merkintäkielellä. (2kmediat.com 2000-2008e.)

CSS:llä on tarkkaan määritetty syntaksi, joka koostuu kolmesta osasta: selektori, ominaisuus ja arvo. Seuraavassa koodiesimerkissä on esimerkki CSS:n syntaksista. Syntaksi on pysynyt aina samana, eikä se ole muuttunut uusien standardien myötä. (2kmediat.com 2000-2008e.)

```
p

{

text-align: center;

color: black;

font-family: arial;

}
```

Yllä olevassa koodiesimerkissä on määritelty kappaleen <p> HTML-elementtiin liittyvät määritykset. <p> -elementti on selektori, jonka ulkoasuun liittyvät määritykset halutaan tehdä. Sitä seuraa lohkosulkeiden sisällä ensin ominaisuus, attribuutti, jota halutaan muuttaa. Tässä tapauksessa ne ovat tekstin asemointi, väri ja fonttiperhe. Ominaisuudet määritellään arvojen avulla, ja ne merkitään kaksoispisteen jälkeen. Yllä oleva koodiesimerkki on luettavuuden vuoksi kirjoitettu usealle eri riville, mikä on usein varsin hyvä ratkaisu.

CSS:ä on noin 120 erilaista ominaisuutta, muun muassa näytöllä käytettävät ominaisuudet, joita ovat esimerkiksi `background` sekä `font`, ja CSS:n aural-ominaisuudet, joita ovat muun muassa `volume` sekä `speak`. CSS sisältää myös 13 eri mittayksikköä, jotka voivat olla joko suhteellisia tai absoluuttisia. Suhteelliset mittayksiköt määrittävät arvon verrattuna johonkin toiseen mittayksikköön tai arvoon, kun taas absoluuttiset mittayksiköt ovat kiinteitä arvoja. Normaalisti suositaan enemmän suhteellisia arvoja niiden skaalautuvuuden vuoksi. Niitä voidaan käyttää joustavammin erilaisissa medioissa, joista kerrotaan tarkemmin työn myöhemmässä vaiheessa, sekä erilaisissa resoluutioissa. Absoluuttisia mittoja kannattaisi välttää, elleivät esitysväline ja sen ominaisuudet ole tarkoin tiedossa kuten ne ovat esimerkiksi tulostusmediassa. (2kmediat.com 2000-2008h.)

CSS sisältää myös noin 30 erilaista valitsinta, joista osalla voidaan kohdistaa CSS-määrittelyt HTML-dokumentissa oleviin elementteihin. Jos CSS-kieltä halutaan käyttää tehokkaammin ja monipuolisemmin, täytyy opetella hyödyntämään valitsimia paremmin. Tällä hetkellä kehitteillä oleva CSS3-luonnos melkein kaksinkertaistaa valitsimien lukumäärän. Monet modernit selaimet, esimerkiksi Opera 10 Alpha-versio, sisältävät kokeellisen tuen osalle CSS3-valitsimia. Seuraavaksi

pari esimerkkiä CSS:n valitsimista, joita on sovellettuna suoraan HTML-elementtiin. (2kmediat.com 2000-2008f; Peltomäki ja Nykänen 2006, 273.)

Universaali * vaikuttaa kaikkiin elementteihin:

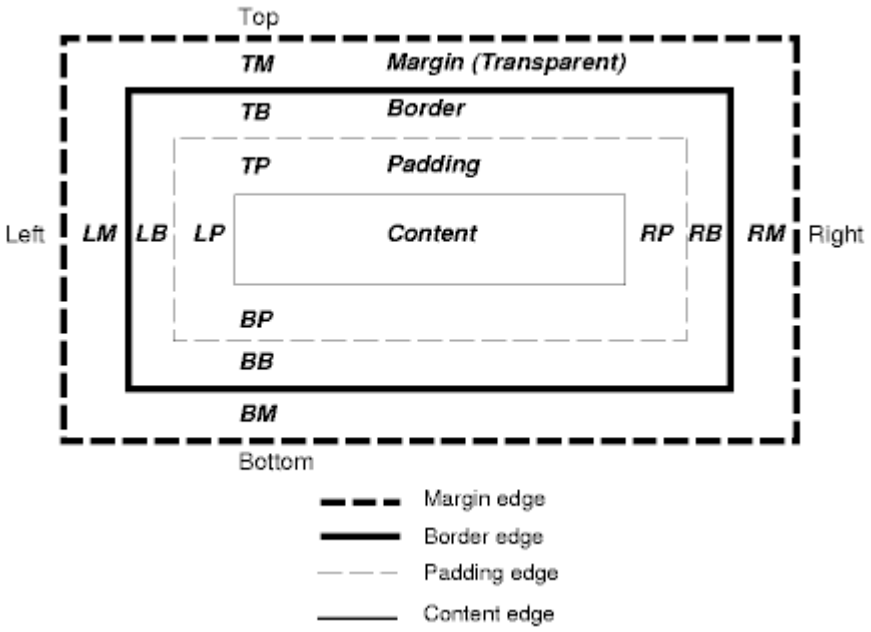
```
*{color:yellow;}
```

Ylemmällä rivillä tyyppivalitsin elementti määrittää yksittäisen elementin muotoilun. Alemmalla rivillä käytetään ryhmittelyvalitsinta, mikä on tyyppivalitsimen kaltainen määrittely, sillä poikkeuksella, että se koskee määrättyjä elementtejä.

```
h1{color:yellow;}

h2,h3,h4{color: black;}
```

CSS:n määrittelemän laatikkomalli määrittää jokaisen elementin esitystavan. Tämä tarkoittaa käytännössä sitä, että CSS:n laatikkomalli kuvaa nelikulmaiset laatikot, jotka luodaan elementeille dokumenttipuussa ja järjestetään visuaalisen järjestysmallin mukaan. Kuvassa 1 näytetään, miten ja missä järjestyksessä eri täytöt ja rajaviivat asettuvat sisällön ympärille, joka on keskimmäisenä. Jokaisella laatikolla on sisältöosa, *content area* (esimerkiksi kuva tai teksti) ja vaihtoehtoisesti ympäröivät *padding*-, *border*- ja *margin*-alueet. (W3C 2008a.)



KUVA 1. Viivapiirros CSS-määritysten käyttämästä laatikkomallista (Kuva: W3C.)

CSS tukee myös kymmentä eri mediatyyppiä, mikä on tyyliohjeiden yksi tärkeimmistä piirteistä. CSS:ssä voidaan määritellä, millä tavoin dokumentti esitetään eri medioissa: esimerkiksi ruudulla, paperilla, puhesyntetisaattorilla tai braille -laitteella. Jotkin CSS:n ominaisuudet on suunniteltu vain tiettyjä medioita silmällä pitäen, esimerkiksi *volume*-ominaisuus aural UA-laitteille. Joissain tapauksissa tyyliohjeet voivat jakaa samat ominaisuudet eri medioissa, mutta eri mediat voivat tarvita erilaisia arvoja. Esimerkiksi *font-size*-ominaisuus on kätevä sekä näyttö-, että tulostusmedialle. Kuitenkin nämä kaksi mediaa ovat tarpeeksi erilaisia, jotta tarvitsisivat eri arvot yhteiselle ominaisuudelle. Tässä tapauksessa tietokoneen ruudulla fontin koon on oltava yleensä suurempi kuin paperilla. Käytännössä myös päätteettömät fontit on helpompi lukea näytöltä, kun taas päätteelliset fontit ovat luettavampia paperilta. Näiden syiden johdosta on tärkeää ilmaista, että tyyliohje, tai osio siitä, pätee tiettyyn mediatyyppiin. (W3C 2008b.)

CSS:lle on myös ominaista ominaisuuksien periytyminen. Sillä tarkoitetaan tilannetta, missä lapsielementti perii ulkoasuun liittyvät ominaisuudet vanhemmaltaan. Esimerkiksi jos HTML-dokumentin *body*-elementille määritellään tekstin väriksi blue *color*-ominaisuudella, niin kaikki elementit *body*-elementin sisällä perivät saman tekstin värin. Periytymisen hyödyt ovat siinä, että sen avulla on mahdollista vähentää CSS-tiedostojen kokoa ja monimutkaisuutta. Sen kautta myös sivut latautuvat nopeammin ja koodi on helpommin ylläpidettävää. Periytymiseen liittyy myös joitakin rajoituksia, jotka pitää ottaa huomioon. Esimerkiksi taulukot ja

taulukko-elementit eivät peri kuin toisilta taulukkoelementeiltä, ja vain rajoitettu osa CSS:n noin 130 ominaisuudesta periytyvät. Tarkempaa tietoa periytyvistä ominaisuuksista löytyy W3C:n websivuilta. Lisäksi vielä, mikäli ominaisuuden periytyminen halutaan erikseen vahvistaa, voidaan se tehdä asettamalla ominaisuuden arvoksi ”inherit”. (2kmediat.com 2000-2008i.)

CSS sallii useista eri tyyliohjeista peräisin olevien sääntöjen yhdistämisen yhdeksi kokonaisuudeksi dokumentin esittämisvaiheessa. Tätä vaihetta kutsutaan nimellä cascade-prosessi. Dokumenttia varten sovellettavat tyylisäännöt voivat tulla useista eri lähteistä, joiden paino-arvo on erilainen. Taulukossa 2 on eritelty, millaisia eri tyylimäärittelyjä on, joiden kesken kaskadinen prosessi suoritetaan.

TAULUKKO 2. Tässä on eriteltynä tyyliohjeiden eri alkuperät (2kmediat.com 2000-2008f)

Laatijan tyylimäärittely	Tämä on joko linkitettynä ulkoisena tiedostona, HTML -/ XHTML -koodiin upotettuna, tai molemmin tavoin edustettuna
Merkintäkielessä olevat muotoilumääreet	Esimerkiksi HTML-dokumentin body-elementissä on määritelty dokumentin taustaväri. cascade-prosessissa selain muuntaa tämän vastaavaksi CSS-säännöksi.
Käyttäjän määrittämä tyylimäärittely	Käyttäjä voi määritellä UA-laitteeseen niin sanotun yksityisen tyylimäärittelyn (CSS-tiedosto), jonka avulla voi ylikirjoittaa dokumentissa käytettävät normaalit tyylit. Piirteen avulla voidaan muun muassa torjua dokumentissa käytettäviä mainoksia tai muotoilla sisältö erikoiskäyttäjille (esimerkiksi näkövammaiset) sopivammaksi.
UA-laitteen oletusasetukset ja perityt arvot	Oletusarvoisesti jokainen UA-laite sisältää joukon oletusarvoja, jotka määrittävät eri elementtien ulkoasun. Esimerkiksi oletusarvoisen fontin, se voi olla joko serif tai sans-serif.
Käyttäjän määrittämät UA-kohtaiset asetukset	Kuten yllä oleva kohta, mutta käyttäjän mukauttamana. Esimerkiksi moni edistynyt käyttäjä vaihtaa web-selaimen oletusarvoisen fontin niin kutsutuksi Sans-serif-kirjasimeksi.

Säännöt asetetaan useissa eri vaiheissa keskinäiseen arvojärjestykseen sen perusteella, mistä lähteestä sääntö tulee. Tähän myös vaikuttavat säännön tarkkuus, spesifisyys, ja se, onko sääntö asetettu tärkeäksi. Jos kaikki osatekijät ovat samanarvoisia, käyttöön otetaan viimeksi esiintynyt sääntö. Kokonaisvaltaisena nyrkkisääntönä voidaan pitää, että käyttäjää lähinnä olevat ominaisuudet määrittelevät hyvin pitkälti sivun esittämistä. (2kmediat.com 2000-2008f.)

Tyylisääntö voidaan merkitä tärkeäksi ja näin se vaikuttaa eri asioihin. Merkitsemällä tyylisääntö tärkeäksi, toisin sanoen se saa suuremman painoarvon kuin normaalisti cascade–prosessissa, käytetään siihen avainsanaa ”!important”, joka tulee välittömästi halutun ominaisuuden jälkeen. Seuraavassa koodiesimerkissä text-decoration ja line-height ovat merkitty tärkeiksi säännöiksi, sen sijaan color ja font-family ominaisuudet saavat normaalin painoarvon cascade-prosessissa. (2kmediat.com 2000 -2008.)

```
strong
{
  color: red;
  text-decoration: underline !important;
  line-height:1.2em !important;
  font-family:Verdana,Arial,Helvetica,sans-serif;
}
```

Ensimmäinen lajittelu suoritetaan sen perusteella, onko sääntö tärkeä vai ei. Seuraavassa luettelossa näkyy lajittelujärjestys tärkeysjärjestyksessä:

1. tärkeät säännöt käyttäjän tyylimäärittelyssä
2. tärkeät säännöt sivun omassa tyylimäärittelyssä
3. tavalliset säännöt sivun tyylimäärittelyssä
4. tavalliset säännöt käyttäjän tyylimäärittelyssä.

Jokaisessa edellä mainitussa lajitteluryhmässä, niiden sisällä, tyylisäännöt lajitellaan lisäksi sen perusteella, kuinka yksityiskohtaisia ne ovat. Mitä yksityiskohtaisempi sääntö on, sitä tärkeämpi. Yksityiskohtaisuus määräytyy kolmiportaisen asteikon mukaan, jonka tulos lajitellaan nousevaan järjestykseen. Asteikko määräytyy seuraavista tekijöistä:

1. valitsimissa olevaan id-atribuuttiin tehtyjen viitteiden lukumäärän mukaan, ja suoraan elementissä olevien tyylisääntöjen mukaan style-atribuutin kautta
2. muiden valitsimessa olevien atribuuttien ja pseudoluokkien lukumäärä
3. valitsimessa olevat elementtinimet.

(2kmediat.com 2000-2008.)

Seuraavassa koodiesimerkissä selvennetään edellä kerrottua asiaa. Alla lyhyt HTML-koodi:

```
<p class="eka">
Kappaleteksti&auml;, jossa on

<a href="#" id="linkki">linkki</a>.

</p>
```

Alla edelliseen HTML-koodiin liittyvät CSS-määritykset:

```
/* vähiten tärkeä sääntö, tavallinen elementtivalitsin */
          a {color: red; }

/* tarkempi sääntö, sisältää mm.luokkavalitsimen */
          p.eka a { color:#99CC00;}


/* tärkein sääntö, suora id-viittaus ja useita
valitsinviitteitä */

          body * a#linkki {color:#CC9900;}

/* myös id-viittaus, mutta yo. sääntö on tarkempi.
Siksi tämä sääntö ei ylikirjoita yo. sääntöä */


          a#linkki {color:#00FF00;}
```

(2kmediat.com 2000-2008.)

Tässä kappaleessa käsitellyt asiat ovat oleellinen osa CSS:n toimintaa ja luonnetta. Sen vuoksi ne on kerrottu tässä pääpiirteittäin. Joitakin edellä mainittuja asioita käsitellään hieman tarkemmin myös seuraavissa kappaleissa.

2.4 Selaintuki

Selaintuki on tärkeä asia, mikä pitää ottaa huomioon eri CSS-versioiden yhteydessä. Se on lähes ensisijainen asia, joka pitää tarkistaa, kun aletaan luoda uutta CSS-kielellä toteutettua ulkoasua. Pitkään on yritetty koota erilaisia kuvauksia siitä, mitä CSS:n piirteitä eri selaimet tukevat, mitä erilaisia ongelmia CSS-toteutuksissa on ja miten niitä voitaisiin ehkä ratkaista (Korpela 2008, 561). CSS on nykyään erittäin laajasti tuettu moderneissa Web-selaimissa. Suunniteltaessa uusia Web-selaimessa näytettäviä ulkoasuja kannattaa kulkea niin sanotusti vuosi kehityksen jäljessä, mikäli haluaa varmistaa ulkoasun laajan toimivuuden ja näkyvyyden. Tietenkin voi kehittää asiaan erikoistuneita sivuja, jotka keskittyvät esimerkiksi pelkästään CSS3-luonnoksen esittämien määrityksen esittelyyn. Nämä sivut ovat kuitenkin oma lukunsa.

CSS -tuki ilmeni web-selaimiin 4.x -sarjan myötä, tosin Opera ja Internet Explorer ovat tukeneet yksinkertaisia CSS-määrittelyjä versiosta 3.x alkaen. Varhaisten selainversioiden (versionumero alle 6.x) osalta CSS-tuen voi sanoa olevan erittäin puutteellinen. Uudemmissa selaimissa (versionumero 6.x tai isompi) CSS-tuki on kuitenkin erittäin monipuolinen ja paranee versio versiolta. (2kmediat.com 2000-2008d.)

Lähtökohtaisesti kaikki modernit selaimet tukevat kuitenkin CSS2:sta ja suurimmaksi osaksi myös CSS2.1:sta. Suosittuimmiksi moderneiksi selaimiksi voidaan luokitella Internet Explorer 6 ja sitä uudemmat, Mozilla Firefox 2 ja sitä uudemmat, Opera 9 ja sitä uudemmat sekä Safari 3 (Machintosh) ja sitä uudemmat versiot. Eniten ongelmia CSS2.1-tuen kanssa on ollut Internet Explorer 6:n kanssa. Sen tuki CSS2.1:lle on ehkä heikoin moderneista selaimista. Paljon parannuksia kuitenkin tuli Internet Explorer 7-selaimen myötä, muun muassa IE:stä aiemmin puuttunut tuki PNG-läpinäkyvyydelle saatiin uuden selaimen myötä korjattua. Sekään ei kuitenkaan saanut korjattua kaikkia IE:n kanssa esiintyviä CSS:hän liittyviä ongelmia. Kaikesta huolimatta vasta julkaistu Internet Explorer 8-selain, on huomattavasti kuronut välimatkaa kiinni muihin selaimiin nähden. Sen kehityksessä on tähdätty erityisesti CSS2.1-standardin lähes täydelliseen tukemiseen (Wikipedia 2009f.)

CSS-tukeen liittyviä virheitä on hankala koota miksikään yleispäteväksi ohjeeksi. Tämä johtuu siitä yksinkertaisesta syystä, että asiat muuttuvat nopeasti. Selaimiin tehdään jatkuvasti pieniä korjauksia, jotka muuttavat asioita. Ne voivat poistaa

virheen tai tuoda joukon uusia. Usein myös selainten virheet ilmenevät tilanteissa, joita olisi hankalaa yksiselitteisesti kuvata. Laajasti vaikuttavia virheitä on kuitenkin uusimmissa selaimissa nykyään sen verran vähän, että yleiset kuvaukset eivät ole enää kovin hyödyllisiä. (Korpela 2008, 562.)

2.5 Mediasääntö ja -tyypit

CSS2 toi mukanaan mahdollisuuden muotoilla sivustojen ulkoasua eri esityslaitteita varten. Näitä erilaisia esityslaitteita kutsutaan CSS:n yhteydessä medioiksi. Yleisimmin tuettuja medioita ovat normaali näyttöesitys, screen, ja tulostin, print. Näiden lisäksi on myös muita UA-laitteita, muun muassa näkövammaisille tarkoitetut braille-laitteet tai PDA-laite. Niille, kuten myös muillekin medioille, voidaan määritellä tyyliohjeiden avulla poikkeava esitysasu mediakohtaisesti. Tyylitiedostojen luomisen tarkoituksena on varmistaa, että sisältö säilyy muodossa, joka voidaan näyttää kaikentyyppisissä laitteissa. Esimerkiksi puhesyntetisaattorit voivat lukea dokumentteja ääneen ja matkapuhelimen näytöltä voidaan myös katsoa websivuja. (Wium Lie & Bos 1999, 259; CSS Opas 2004.)

On kaksi tapaa ilmaista mille medialle tietty tyyliohje kohdennetaan. Jos käytetään ulkoisia CSS-tyyliohjeita tai upotettuja CSS-tyyliohjeita, mediatyypin määrittely tapahtuu lähdedokumentissa. HTML-dokumentissa siis elementtien <link> tai <style> media-atribuutin avulla. Toinen tapa on hyödyntää CSS:n at-sääntöä, @media. Alla olevissa koodiesimerkeissä selvennetään, millä tavoin nämä kaksi tapaa toimivat. (2kmediat.com 2000-2008j.)

Ulkoisen tyylimäärittelyn liittäminen dokumenttiin:

```
...
<head>
<link href="/tyylit/dokumentti.css" rel="stylesheet" me-
dia="screen" type="text/css"/>
<link href="/tyylit/dokumentti-print.css" rel="stylesheet" me-
dia="print" type="text/css"/>
<title>CSS dokumentti</title>
...
```

Upotetun tyylimäärittelyn liittäminen dokumenttiin:

```
...
<head>
 <style type="text/css"  media="screen">
   <!--
   h1 {color:#fafafa;}
   -->
  </style>
  <title>CSS esimerkki</title>
</head>
...
```

Tässä esimerkki missä mediamäärittely sijoitetaan CSS -tyyliohjeen sisään:

```
@media screen {
  body {
    font:Verdana, Arial,sans-serif x-large;
   }
}

@media print {
  body {
    font:Times, "Times New Roman", serif 12pt;
  }
}
H1{font-size: 2em;}
```

@media-avainsanaa seuraa mediatyypin nimi (CSS2 määrittelee yhdeksän eri mediatyyppiä). Aaltosulkeiden sisällä ovat aivan tavalliset tyylimääritykset. Koska ne vaikuttavat vain tiettyihin mediatyyppeihin, niitä sanotaan mediasäännöllisiksi tyylitiedostoiksi. Huomattavaa on, että mediasäännölliset tyylitiedostot täydennetään valitsimilla, joilla on omat aaltosulkeet. Kuten myös huomataan yllä-olevassa koodiesimerkissä, yhdessä tyylitiedostossa voi olla monta @media kappaletta. Lisäksi koska h1 sääntö on yhteistä molemmille mediatyypeille, se on otettu pois mediasäännöllisistä osista ja liitetty koskemaan kaikkia mediatyyppejä. (Wium Lie & Bos 1999, 260.)

Seuraavassa taulukossa on kerrottu lyhyesti kaikista CSS2:ssa ilmoitetuista yhdeksästä mediatyypistä:

TAULUKKO 3. Mediatyypit. (Wium Lie & Bos 1999, 261)

Mediatyyppi	Käyttötarkoitus
Screen	Normaalit tietokoneen näytöt
Print	Tulostimet, kun tulostetaan paperille. Tätä mediatyyppiä käyettään myös kun selain näyttää ”tulostuksen esikatselu” ominaisuutta.
Aural	Puhesyntetisaattoreille, jotka lukevat websivuja ääneen.
Braille	Elektroniset pistekirjoituksen lukijat, joita käyttävät näkövammaiset.
Embossed	Braille-tulostimille, mitkä tekevät pistekirjoituksen kohokuvioina paperille.
Handheld	Kämmentietokoneet (PDA) ja matkapuhelimet, millä on yleensä ominaista pienet ruudut ja rajoitettu kaistanleveys.
Projection	Projektoreilla näyettävät esitykset, kuten kalvot tai tietokoneeseen yhdistetyt videoprojektorit. Tavallisesti näissä esityksissä on vain muutama rivi sivulla ja tarvitsevat suuren fonttikoon.
Tty	Tarkoitettu mediatyypeille, jotka käyttävät kiinteävälistä kirjasinkehikkoa ja jotka eivät tue grafiikkaa. Koska laitteet perustuvat merkkeihin eivätkä pikseleihin, niin ne eivät voi näyttää rich style tyyppisiä fontteja. Suurin etu tyylitiedostoista näille laitteille on se, että teksti pysyy tekstinä, eikä kuvina ja on yleensäkkin selkeästi katsottavaa.
Tv	Televisiossa näytettävät esitykset. Tyypillisesti näytöille joilla on rajoitettu väriskaala, pieni resoluutio, pitkä katseluetäisyys ja rajoitetut tekstin vieritys mahdollisuudet.

All	Tämä mediatyyppi sopii kaikille laitteille. Mediatyypit eivät kuuluneet CSS1:een. Sen vuoksi tätä mediatyyppiä voidaan käyttää salaamaan tyylitiedostot vanhemmilta selaimilta.

CSS:n yhteydessä on myös puhuttu erilaisista profiileista, joissa on otettu huomioon eri mediatyyppejä. CSS:ä kuvattuja eri profiileita ovat CSS Mobile-profiili, CSS Print-profiili ja CSS TV-profiili. Profiilit kokoavat joukon määrittelyjä, jotka on tarkoitettu erilaisissa tarkoituksissa käytettäviin esityslaitteisiin.

2.6 CSS3 -luonnos

2.6.1 Kehitys

CSS3 tekee jo vahvaa tuloa web-suunnittelijoiden keskuudessa. Nykyään voidaan nähdä jo kokonaisia sivustoja, jotka on jollain tapaa toteutettu CSS3-luonnoksen avulla. Varsinaisessa CSS3:n määrittelyssä ei ole päästy vielä suositus-tasolle. Lähiaikoina tämä tulee tuskin tapahtumaan, sillä se vaatisi muun muassa Internet Explorer- ja Mozilla Firefox-selainten uusia CSS3:sta paremmin tukevien päivitysten julkistamista. Nykyiset modernit nettiselaimet tukevat kuitenkin jo jossain määrin CSS3:sta. CSS3:n kehittäminen aloitettiin jo vuonna 1998, ja sen kehityksen hitaus johtuu siinä määriteltyjen ominaisuuksien valtavasta määrästä.

Syntaksi CSS3:ssa ei oikeastaan muutu lainkaan, joitakin pieniä parannuksia voi kuitenkin olla. Siinä käytetty syntaksi on samanlainen kuin jo aiemmin tunnetuissa CSS-versiossa. Tämä johtuu sisäänrakennetusta eteen- ja taaksepäisestä yhteensopivuudesta (W3C 2001.) Selaintuki CSS3:lle on vielä varsin vaihtelevaa. Suurinta osaa CSS3:n ominaisuuksista ei ole tuettu vielä missään modernissa selaimessa, mutta esimerkiksi CSS3:n selektorit alkavat olla jo suhteellisen tuettuja yleisimmin käytettyjen modernien Web-selainten versioissa.

CSS3 mahdollistaa luovemman ilmaisun websuunnittelussa. Asiat, jotka olivat aiemmin tai ovat vielä nykyäänkin jokseenkin hankalia toteuttaa, tulevat olemaan huomattavasti yksinkertaisempia toteuttaa uuden määrityksen avulla. Nykyään CSS3-luonnosta ei kannata vielä käyttää päivittäisissä asiakasprojekteissa, mutta henkilökohtaisilla sivuilla ja design-blogeissa näitä menetelmiä voi ja kannattaa

käyttää, koska niiden avulla on hyvä venyttää modernin websuunnittelun rajoja. (Smashing Magazine 2009.)

Kiinnostavimpia uutuuksia CSS3:ssa ovat muun muassa ominaisuuksien jakaminen moduleihin, pyöreät kulmat, varjostus ja kuvista muodostetut rajat sekä esitystasot. Esitystasoilla voidaan määrätä näytetäänkö enemmän vai vähemmän tietoja riippuen halutusta esitystasosta. SVG yhdistyy myös tiukemmin CSS:ään ja matemaattisten merkintöjä voidaan muotoilla erikseen. CSS 3:n tarjoaa lisäksi suuren joukon uusia valitsintyyppejä, joiden avulla voidaan muun muassa valita elementti sen sisältämän tekstin perusteella. Uusina tai parannettuina ominaisuuksina tulevat myös lisäominaisuudet tekstin muotoilemiseen, tapahtumankäsittely, nimiavaruuslaajennukset sekä sarakelayout. Lisäksi kirjasinten ulkoasua voidaan muotoilla uusien ominaisuuksien avulla, ja ääniefektit sekä uusi speech-mediatyyppi tulevat olemaan CSS3:n erityisiä ominaisuuksia. (Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2005.)

Vaikka CSS3:n tuloa on paljon odotettu ja sitä on hehkutettu Web-suunnittelijoiden keskuudessa, sen käyttäminen Web-suunnittelussa ei ole vielä mikään itsestäänselvyys. CSS3:n yhteydessä on vielä tällä hetkellä otettava huomioon muutama huomion arvoinen asia: Internet Explorer on yksi suuri tekijä, jonka vuoksi CSS3:lla tekemiä tyylimäärityksiä ei noin 46 % internetin käyttäjistä näe. Tämän vuoksi ei kannata tehdä pelkästään CSS3-määrityksiin pohjautuvaa sivustoa. Jos uusia ominaisuuksia haluaa joka tapauksessa käyttää, niin kannattaa lisätä vaihtoehtoiset määritykset CSS3-määrityksiä tukemattomille selaimille. (Smashing Magazine 2009.)

CSS3-määritykset saattavat myös invalidoida koko tyylitiedoston, koska se ei ole vielä saavuttanut yleisen standardin tasoa. Tällä hetkellä eri selaimille kohdistetaan omilla merkinnöillään halutut CSS3-määritykset. Seuraavasta koodiesimerkistä voidaan havaita merkintäerot eri selaimille tarkoitetuissa tyylisäännöissä. Koodiesimerkissä määritellään laatikkomaiselle HTML-elementille pyöristetyt reunat. Ensimmäisellä rivillä tyyliohje suunnataan Firefox-selaimelle, toisella rivillä Safari-selaimelle sekä Chrome-selaimelle ja kolmannella rivillä näkyy perusyntaksi border-radius-ominaisuudelle, joka on tarkoitettu muille ominaisuutta tukeville selaimille. (Smashing Magazine 2009.)

```
-moz-border-radius: 20px;

-webkit-border-radius: 20px;

border-radius: 20px;
```

Jotta saataisiin sama efekti aikaan, ylimääräisten merkintöjen lisääminen jokaiselle selaimelle erikseen tuo kuitenkin paljon lisärivejä CSS-koodiin, joka heikentää CSS-kielen tehokkuutta. CSS3-kieltä tullaan käyttämään todennäköisesti myös huonossa Web-designissa vain sen vuoksi, että sillä voidaan tehdä jotain uutta ja hienoa. Tätä mahdollista CSS3:n väärinkäyttöä voidaan verrata huonosti käytettyihin Adope Photohopin suotimiin, joilla voi saada aikaan erittäin mauttomia efektejä. Esimerkiksi CSS3:n drop-shadows-ominaisuudella voidaan saada amatöörimäisesti käytettynä aikaan käyttökelvottomia ulkoasuja. (Smashing Magazine 2009.)

2.6.2 Modulaarisuus

CSS3–luonnoksen ominaisuudet on jaettu eri moduleihin, toisin kuin edeltävät versiot, joista ohjelmat voivat valita, mitä moduleita haluavat tukea. Moduleja on yli 30 kappaletta. (Jyväskylän yliopiston IT-tiedekunta 2008.)

CSS3 päätettiin jakaa useisiin eri moduleihin sen vuoksi, koska yhtenäinen CSS2 olisi vastaavassa laajuudessa ollut liian hankala käyttää. Lisäksi CSS:n jakaminen moduuleihin mahdollistaa myös W3C:n kehittämään modulit suositustasolle eri aikoihin. (css3.info 2009.)

Osa uusista kehitteillä olevista moduleista lisää täysin uusia toiminnallisuuksia CSS-kielen yhteyteen. Näitä ominaisuuksia ei ole aiemmin esiintynyt CSS1- ja CSS2-suositusten toiminnoissa. Täysin uusia ominaisuuksia sisältäviä moduleita ovat esimerkiksi CSS animation, CSS transition, CSS 2D transforms ja CSS 3D transforms, joiden kehitys sai alkunsa Apple WebKit-tiimin ehdotuksesta. (CSS3.info 2009.)

3 DYNAAMISET WEBTEKNIIKAT

3.1 JavaScript

Javascript on Netscape Communications Corporationin kehittämä oliopohjainen skriptauskieli, joka julkaistiin joulukuussa 1995. Alun perin sen suunnitteli ja kehitti Brendan Eichi. Prototyyppivaiheessa JavaScriptiä kutsuttiin Mochaksi, minkä jälkeen seurasivat LiveWire ja LiveScript. Näiden nimeämisvaiheiden jälkeen nimi vakiintui JavaScriptiksi. (Aselson ja Schutta 2007, 6.)

JavaScript kehitettiin käytettäväksi web-ympäristössä helpottaakseen Web -suunnittelijoiden sekä -ohjelmoijien sovelluskehitytyötä. JavaScriptin tärkeimpänä ominaisuutena on lisätä interaktiivisuutta internetsivuilla. Se on oliopohjainen skriptattava kieli, jonka pohja perustuu heikosti c-ohjelmointikieleen. Standardisoitua JavaScriptiä kutsutaan ECMAScriptiksi. JavaScript on saanut vaikutteita monista ohjelmointikielistä, ja se suunniteltiin muistuttamaan Java -ohjelmointikieltä, mutta se kehitettiin helpommin ymmärrettäväksi muun muassa internet -suunnittelijoille sekä -ohjelmoijille.(Aselson ja Schutta 2007, 6; Wikipedia 2008c; Wikipedia 2008d.)

JavaScript on tehokas tapa dynaamisten verkkosovellusten luomiseen. Alun perin se kehitettiin muokkaamaan HTML-kielen tageja dynaamisesti, jotta verkkosivujen käyttö olisi käyttäjälle miellyttävämpää. Huomattiin, että sivuja voitaisiin käsitellä olioiden tapaan, ja näin syntyi DOM -malli (Document Object Model). Ensin JavaScript ja DOM -malli sidottiin tiukasti yhteen, mutta lopulta ne kehittyivät täysin erilisiksi rakenteiksi. DOM-mallia voidaan pitää tapana käsittää verkkosivu täysin oliokeskeisessä muodossa, jota voidaan muokata skriptauskielien (kuten JavaScript ja VBScript) avulla. (Aselson ja Schutta 2007, 6.)

W3C standardoi lopulta DOM -rakenteen ja ECMA (European Computer Manufactures Association) vahvisti JavaScriptin ECMAScript -kielen määritelmässään. Edellä mainittujen standardien mukaan rakennetut sivut ja skriptit tulisi toimia yhdenmukaisesti missä tahansa selaimessa, joka toteuttaa standardin näiden määritelmien mukaisesti.(Aselson ja Schutta 2007, 6.)

Alkuaikoina monet seikat olivat JavaScript -kieltä vastaan. Selaintuki oli puutteellista (jopa nykyään skriptit saattavat toimia eri selaimissa eri tavoilla), ja näin käyttäjille on tarjottu mahdollisuutta kääntää JavaScript tuki pois päältä selaimes-

ta. JavaScript -ohjelmien kehittämisen vaikeus on aiheuttanut aliarvostuta kehittäjien keskuudessa. Useimmat kehittäjät tyytyivätkin luomaan kielellä vain yksinkertaisia lomakepohjaisia sovelluksia koettuaan JavaScript -koodin käyttämisen, testaamisen ja virheiden jäljittämisen liian monimutkaiseksi. (Aselson ja Schutta 2007, 6.)

3.2 Ajax

3.2.1 Ajaxin määrittely ja sen ongelmallisuus

Smithin ja Negrinon (2007, 357) mukaan verkko muuttuu jatkuvasti. Vuonna 2005 yleiseen käyttöön ilmaantui suositeltuja verkkosovelluksia, joista useimmat olivat Googlen käsialaa, kuten esimerkiksi Gmail, Google Maps sekä joitakin muita. Lisäksi muita maininnan arvoisia olivat Flickr ja MyYahoo!-portaali. Näiden sovellusten yhdistävä tekijä oli työpöytäsovelluksia vastaava toiminnallisuus, joka toi verkkokäyttöön nopeat ja responsiiviset käyttöliittymät. Tämä oli verkko- ja Web-kehittäjien mielestä mullistavaa. Uudet sivut toimivat nopeammin ja päivittyivät nopeammin kuin aiemmin. Yleensä käyttäjä joutui odottamaan useita sekunteja, kunnes palvelin vastasi pyyntöön ja selain päivitti sivun. Uusien sivujen myötä palveluiden interaktiivisuus ja käyttäjäkokemus paranivat huomattavasti. Näitä uusia sivuja yhdisti yksi uusi tekniikka nimeltään Ajax. Ajax-tekniikkaa käytetään tekemään sivuista vuorovaikutteisempia, nopeampia sekä käytettävämpiä.

Ajax-tekniikkassa selainohjelma vaihtaa pieniä määriä dataa kerrallaan palvelimen kanssa taustalla siten, ettei koko sivua tarvitse ladata uudelleen, silloin kun käyttäjä tekee muutoksia. Ajaxissa yhdistyvät useat eri verkkosovellustekniikat, kuten XHTML (tai HTML), CSS, DOM asiakaspuolen skriptikielellä (kuten JavaScript tai Jscript), XMLHttpRequest -objekti ja XML. Ajax ei siis ole itsenäinen teknologia vaan yhdistelmä eri tekniikoista, jotka ovat jo ennestään tuttuja Web-kehittäjille. Kaikki selaimet eivät tue Ajaxia. Esimerkiksi Internet Explorer 5.0 ja sitä vanhemmat IE:t eivät ole tuettuja, koska XMLHttpRequest -objektin tukea ei ole. Toimiakseen kunnolla Ajax-sivua on tarkasteltava modernilla Web-selaimella. Standardeihin perustuvaan esitysmuodon luomiseen käytetään XHTML:ää ja CSS:ää. Ajaxin toimintatavasta johtuen kehitystä ja testausta suoritettaessa luetavien tiedostojen on aidosti sijaittava palvelimella, eli tiedostot eivät voi sijaita paikallisella koneella. (Wikipedia 2008a; Garret 2005.)

3.2.2 Ajaxin käyttäminen

Ajaxista on olemassa monta hyvää esimerkkiä, muun muassa Google Maps ja Flickr, mutta on myös hyvin helppo luoda hämmentäviä ja vaikeasti käytettäviä Ajax-sovelluksia. Tässä kappaleessa esitellään joitakin ongelmia, joihin saattaa törmätä Ajax-sovelluksia toteuttaessa.

- Websivujen käyttäjät ovat tottuneet yleensä käyttämään selaimen paluunappia palatakseen niille websivuille, joilla he ovat juuri olleet. Valitettavasti, ellei asiaa erikseen oteta huomioon, paluunappi ei toimi odotetusti Ajax-sovelluksissa. Koska Ajax-sovelluksessa muuttuu vain osa sivun tiedoista, ei edelliseen haluttuun tilaan enää pääse. Paluunappia painamalla voi joutua ulos koko sivustolta. On olemassa kuitenkin monia Ajax-kehyksiä, jotka saavat selaimen paluunapin toimimaan tavalla, johon käyttäjät ovat tottuneet. (Thau 2006, 274.)

- Websivun URL-osoite voidaan yleensä kopioida ja lähettää ystävälle tai lisätä kirjanmerkkeihin. Tähän liittyen yhtenä ongelmana voidaan pitää sitä, ettei websivun URL-osoite muutu, koska sillä oleva Ajax-sovellus ei muuta sisältöään sivun lataamisen yhteydessä (koko sivu vaihtuu kerralla). Kannattaa siis kiinnittää huomiota URL-osoitteen luomisessa, jotta se voitaisiin kirjamerkitä tai lähettää esimerkiksi sähköpostissa. (Thau 2006, 274.)

- Tottuneet käyttäjät ovat tulleet tutuiksi tavallisen suorita-odota-uudelleen lataa –metodin kanssa webvuorovaikutuksessa. Tämän tyylisessä kommunikoinnissa koko websivu päivittyy, kun informaatio palautetaan serveriltä, minkä vuoksi käyttäjä yleensä huomaa koko sivun olevan uusi. Kun käytetään Ajaxia, voi websivu vaihtua ilman, että käyttäjä huomaa mitään. Websuunnittelijan tehtävänä on ilmoittaa tärkeistä muutoksista käyttämällä suunnitteluun liittyviä tekniikoita, kuten muuttamalla väriä tai reunuksia. (Thau 2006, 274.)

- Ajax tarjoaa myös uusia navigointitapoja websuunnittelijalle. Ennen Ajaxia suunnittelijat käyttivät kuvia ja linkkejä websivujen väliseen navigointiin. Ajax ja DHMTL parantaa joustavuutta navigoinnissa, mutta tämä joustavuus voi aiheuttaa hämmentäviä vaihtoehtoja navigoinnissa. Esimerkksi voitaisiin suunnitella websivu, jolla on interaktiivinen nuppi, jota käyttäjät voivat kääntää navigoidakseen sivuston eri sivuilla. Vaikka tämä interaktiivinen nuppi voi olla kätevä, se voi myös hämmentää niitä käyttäjiä, jotka ovat tottuneet navigoimaan websivuilla käyttämällä linkkejä. Toisin sa-

> noen uuden hienon navigointityylin lisääminen ei aina ole hyvä idea, ellei vain halua näyttää Ajax-ohjelmointi taitojaan. (Thau 2006, 275.)

Ajaxia ei pitäisi käyttää vain sen vuoksi, että sen avulla voi saada aikaan vaikuttavan näköisiä web-sovelluksia. Kannattaa pitää mielessä käyttäjien tottumukset. Sitä ei myöskään pitäisi käyttää vain sen vuoksi, että se on uusi tekniikka, sillä Ajax ei ratkaise kaikkia ongelmia vaan vastakohtaisesti tuo joukon uusia. Se ei myöskään ole suotavaa, että korvaa jonkin vanhan ja toimivan tekniikan jollain uudella ja hämmentävällä. Esimerkiksi perinteiset linkit ovat mainio keino johdattaa käyttäjät websivulta toiselle. Jos lisää uuden monimutkaisen navigaation, se pikemminkin voi karkoittaa kävijöitä. (Thau 2006, 275.)

Ajaxia on hyvä käyttää siinä tapauksessa, kun objektia manipuloitaessa muu osa sivustosta halutaan pysyvän muuttumattomana sen ajan, kun objektia manipuloidaan. Esimerkiksi Google Maps:ssa voidaan karttaa siirtä pannaamalla sitä. Loput sivusta ei muutu eikä sivu ole missään vaiheessa tyhjä. Karttasovellus, joka ei käytä Ajaxia lataa uudelleen koko websivun joka kerta, kun halutaan pannata kartaa. Toinen hyvä käyttökohde Ajaxille ovat interaktiiviset widgetit. Ne ovat pieniä websivun osia, jotka yleensä sijaitsevat websivun reunaosilla. Sellaiset sivun osat kuin uutissyötteet, pikakyselyt ja kirjautumislomakkeet lukeutuvat tällaiseen joukkoon. Jokaisessa edellä mainitussa tapauksessa ollaan vain vuorovaikutuksessa widgetin kanssa, ja siitä voi seurata tuloksia, jotka eivät vaadi koko sivun uudelleen latautumista. Esimerkiksi jos käyttäjä sattuu syöttämään virheellisen käyttäjätunnuksen tai salasanan, yrittäessään sisäänkirjautua, niin virheilmoitus voi ilmestyä sen hetkiselle sivulle. Toisin kuin siinä tapauksessa missä käyttäjä tuodaan uudelle sivulle, jossa ei ole muuta kuin ilmoitus ”sisäänkirjautuminen epäonnistunut”. (Thau 2006, 276.)

Kolmas syy, millaisessa tilanteessa Ajaxia kannattaa käyttää, on automaattinen tallennus. Se voidaan ajatella samankaltaiseksi kuin tavallisissa tekstinkäsittelyohjelmissa. Tämän tekniikan avulla voidaan Ajaxia käyttäen lähettää serverille tietoa, ilman että käyttäjää varoitetaan tai muuten häiritään sillä tiedolla. Tämän kaltainen taustalla toimiva ohjelma on täydellinen ratkaisu Ajaxille. (Thau 2006, 276.)

3.2.3 Toimintaidea

Toimintaidealla tarkoitetaan tässä tapauksessa Ajax-ohjelman perusvaiheiden läpikäymistä asiakaspuolella, eli selaimen päässä. Ensimmäisessä vaiheessa asiakaspuolella käynnistyy Ajax-tapahtuma kutsumalla jotakin tapahtumankäsittelijää, käytännössä painetaan vaikkapa nappia Web-sovelluksessa. Seuraavaksi luodaan XMLHttpRequest-olio, ja samalla avataan yhteys palvelimelle sekä asetetaan tapahtumankäsittelijä (`onreadystatechange`), minkä jälkeen lähetetään pyyntö palvelimelle. (Peltomäki, Nykänen 2006, 300-303.)

Palvelin tekee halutut toiminnot ja palauttaa vastauksen asiakkaalle. Palvelin voi esimerkiksi käydä hakemassa tai tarkistamassa tiedot tietokannasta tai palvelimella olevasta XML-tiedostosta. Palvelin palauttaa vastauksen MIME-tyyppinä `text/xml`, mutta XMLHttpRequest-objekti osaa käsitellä vain `text/html`-tyyppisiä olioita. Vastaus voi sisältää myös DOM-lausekkeita. Kun vastus on saapunut takaisin selaimelle, asiakaspäähän, XMLHttpRequest-olio kutsuu takaisin kutsuttavaa metodia, joka tarkkailee `readyState`-ominaisuuden arvoa ja käsittelee `onreadystatechange`-tapahtuman pyyntövaiheessa asetetulla tapahtumankäsittelijällä. (Peltomäki, Nykänen 2006, 300-303.)

Kun tarvittavaa metodia on kutsuttu, XMLHttpRequest-olio ottaa yhteyden palvelimeen ja palauttaa pyydettyä dataa. Tämä prosessi voi kestää ennustamattoman kauan, minkä vuoksi täytyy erikseen seurata milloin olio on saanut kaiken datan takaisin. Tähän tarvitaan tapahtumankuuntelijoita, joka seuraa XMLHttpRequest-olion `readyState`-ominaisuuden muuttumista. `ReadyState`-ominaisuuden arvo kasvaa nollasta neljään. `Onreadystatechange`-tapahtumankäsittelijää kutsutaan kaikilla arvoilla, mutta yleensä vain ollaan kiinnostuneita, milloin pyyntö on tehty kokonaan, eli `ReadyState`-ominaisuus on saanut arvon neljä, ja vastaus on vastaanotettu. Yhteyden muodostumisen jälkeen on myös tarkistettava, onko XMLHttpRequest-olio saanut onnistuneesti kaiken datan. Kun http-vastauksen `status`-ominaisuuden arvo on 200, se merkitsee, että http-pyyntö on onnistunut. (Peltomäki, Nykänen 2006, 300-303.)

Parserointi XMLHttpRequest-oliosta

Jotta haettua dataa voitaisiin käyttää selaimenpäässä, täytyy se ensin parseroida XMLHttpRequest-oliosta. Käytännössä se tapahtuu kahden XMLHttpRequest-olion ominaisuuden kautta, jotka voi sisältää erityyppistä dataa:

- ResponseXML tallentaa DOM-rakenteisia olioita mistä tahansa XML-datasta. Dataa voidaan käydä läpi käyttämällä standardeja JavaScript DOM-metodeja ja –ominaisuuksia, kuten `getElementsByTagName`(), `childNodes`[] tai `parentNode`.

- ResponseText tallentaa datan merkkijonona. Jos palvelimen palauttaman datan sisältötyyppi on `text/plain` tai `text/html`, vain tämä ominaisuus sisältää dataa. Kaikki `text/xml`-tyyppinen data muutetaan ja sijoitetaan tänne vaihtoehtoiseksi `responseXML`-olion kanssa.

Monimutkaisemmat datarakenteet muunnetaan tarvittaessa XML-formaattiin. Kun data on palautettu palvelimelta ja parsittu XMLHttpRequest-oliosta, sitä voidaan vapaasti muuttaa, poistaa ja päivittää selainpuolen sovelluksessa. (Peltomäki, Nykänen 2006, 300-303.)

3.2.4 XML

XML on W3C:n standardoima tekniikka tekstimuotoisten dokumenttien merkkaamiseen ja käsittelyyn. Keskeinen idea on sopia siitä, kuinka tietoa kannattaa tallentaa tietorakenteina niin sanottuina XML-dokumentteina siten, että tieto on helposti ja yksikäsitteisesti luettavissa. Koska tarkoituksena on automatisoida tietojenkäsittelyä nimenomaan tietokoneiden avulla, ovat XML-suositukset lähtökohdiltaan melko teknisiä. (Peltomäki, Nykänen 2006, 4.)

XML tekniikkaa voidaan soveltaa laajasti. XML tarjoaa suunnittelijan käyttöön niin sanotun metakielen, jonka avulla on mahdollista määritellä sovelluskohtaisten tietojen kuvaamiseen tarkoitettuja merkkauskieliä (esimerkiksi verkkosivuista tuttu XHMTL ja vektorikuviin soveltuva SVG). Tämä tarkoittaa sitä, että XML-tekniikkaa voidaan hyödyntää monenlaisten sovellusten suunnittelussa ja toteuttamisessa. XML tarjoaa perustan verkkosivujen toteutustekniikan ohella vaikkapa organisaation dokumenttijärjestelmän tietomallin suunnitteluun sekä teknisten

piirrosten tietojen esittämiseen. Laajemmin tulkittuna XML tarjoaa puitteet myös (meta)tiedon käsittelyyn sekä ohjelmistojen välisen viestiliikenteen toteuttamiseen. Kyse on nimenomaan teknisille suunnittelijoille tarkoitetusta tekniikasta. Loppukäyttäjille XML ei välttämättä näy suoraan lainkaan. (Peltomäki, Nykänen 2006, 4.)

Sovelluksissa XML voidaan nähdä esimerkiksi relaatiotietokannan kaltaisena yleiskäyttöisenä tekniikkana. Syy on se, että relaatiotietokantojen tekniikan tapaan myös XML ohjaa sovelluksen tietomallin suunnittelua ja osoittaa välineitä tietojenkäsittelyyn. Keskeisin ero piilee XML-tekniikoille ominaisessa protokollapinoajattelussa sekä tarkassa sopimuksessa XML-dokumenttien rakenteen tasolla. (Peltomäki, Nykänen 2006, 4.)

3.2.5 XMLHttpRequest-objekti

XMLHttpRequest sisällytettiin ensimmäisenä Internet Explorer 5 -selaimeen ActiveX -komponenttina. Sen toteutus ainoastaan Internet Explorer -selaimissa esti kehittäjiä käyttämästä sitä laajamittaisesti aina siihen asti, kunnes myös Mozilla- ja Safari -selaimet toteuttivat tekniikan hyväksyen sen aseman epävirallisena standardina. On tärkeää huomata, että XMLHttpRequest ei ole W3C -standardin mukainen tekniikka. Koska edellä mainittua tekniikka ei ole standardisoitu, sen toiminnallisuus voi vaihdella selaimesta riippuen. Kuitenkin suurin osa sen tarjoamista metodeista ja ominaisuuksista ovat laajasti tuettuja. Tällä hetkellä Firefox, Safari, Opera, Konqueror ja Internet Explorer ovat toteuttaneet XMLHttpRequest -objektin toiminnallisuuden ja ominaisuudet yhdenmukaisesti. (Aselson ja Schutta 2007, 25.)

XMLHttpRequest -objekti on luotava JavaScriptillä, jotta voidaan lähettää selaimelle pyyntöjä ja käsitellä vastauksia. Koska XMLHttpRequest ei ole virallinen standardi, olion voi luoda kahdella eri tavalla. Internet Explorer toteuttaa XMLHttpRequestin ActiveX -komponenttina muiden selainten toteuttaessa sen sisäisenä JavaScript -oliona. Tämän toteutuseron vuoksi JavaScript -koodin on tuettava molempien selainryhmien toteuttamaa logiikkaa, jotta verkkosovellus toimisi yhtäläisesti kummankin selainryhmän selaimilla. (Aselson ja Schutta 2007, 25.)

Tarvittava JavaScript -koodi selaintyypin tunnistamiseen on melko lyhyt. Riittää, että koodi tarkistaa tukeeko selain ActiveX -objekteja vai ei. Jos selain tukee ActiveX -komponentteja, luodaan koodissa IE -selainta varten XMLHttpRequest -objekti ActiveX -komponentin avulla. Jos selain ei tue ActiveX -objekteja, luodaan XMLHttpRequest -objekti JavaScript -koodissa kuten mikä tahansa olio. Seuraavassa koodiesimerkissä luodaan XMLHttpRequest-objektin ilmentymä selainriippumattomalla tavalla. (Aselson ja Schutta 2007, 26.)

```
var xmlHttp;
           function createXMLHttpRequest(){
           if (window.ActiveXObject){
                                    xmlHttp = new ActiveXOb-
ject(”Microsoft.XMLHTTP”);
                       }
                       else if (window.XMLHttpRequest){
xmlHttp = new XMLHttpRequest();
                       }
           }
```

JavaScriptin dynaamisen tyypityksen ja XMLHttpRequestin laajan ja yhdenmukaisen toteutuksen ansiosta XMLHttpRequestin ominaisuuksia voi käyttää käyttäjän selaimesta riippumatta. Ainoa selainkohtaisesti eroava tekijä on juuri koodiesimerkissä esitelty XMLHttpRequest -olion luominen. Tämä ominaisuus yksinkertaistaa ja nopeuttaa kehitysprosessia sekä tukee selainriippumattomuutta. (Aselson ja Schutta 2007, 26.)

3.2.6 DOM-malli

DOM on W3C:n alusta- ja kieliriippumaton määritelmä dokumentin sisällön ja rakenteen muokaamiseen. Toisin sanoen kyseessä on yleinen tapa sekä esittää ja editoida HTML- tai XML -dokumenttia. DOM -malli ei ole kielisidonnainen, vaikka se alun perin suunniteltiin tavaksi tehdä JavaScript -koodista siirrettävää eri selainten välillä. Sittemmin DOM-mallin käyttökohteet ovat lisääntyneet.(Aselson ja Schutta 2007, 39.)

DOM-malli on aito oliomalli oliokeskeisen suunnittelun näkökulmasta. DOM määrittelee olioita, joita tarvitaan dokumentin esittämiseen ja muokkaamiseen.

Lisäksi DOM määrittelee kyseisten olioiden käytöksen ja ominaisuudet sekä olioiden väliset suhteet. DOM-mallin voi käsittää puumaiseksi rakenteeksi esittää tietoa ja tiedon rakenteita, vaikka itse toteutus ei ole välttämättä vastaavanlainen. DOM-mallin loistava ominaisuus on sen tarjoama standardoitunut tapa käsitellä dokumentteja. Ilman DOM-mallia useimpia Ajaxin mielenkiintoisimmista ominaisuuksista ei olisi mahdollisia toteuttaa. Koska DOM tarjoaa tavan sekä tiedon muokkaamiseen ja läpikäyntiin, on sen avulla mahdollista tehdä aidosti dynaamisia verkkosovelluksia. (Aselson ja Schutta 2007, 39)

3.2.7 JavaScript ja CSS

Uusimmissa selainversioissa voi JavaScriptillä käsitellä suurinta osaa CSS-tyylimäärityksistä. Näiden kahden yhdisteleminen voi saada aikaan mielenkiintoisen ja rikkaan lopputuloksen. CSS-tyylimäärityksiä voidaan manipuloidan JavaScriptillä DOM-ohjelmointirajapinnan avulla.

3.3 PHP

Heinisuon (2001, 15) mukaan PHP, Hypertext Prerpocessor, oli alun perin eräänlainen palvelinpäässä ajettavien CGI-ohjelmien tekemistä helpottava komentokokoelma. PHP on kehittynyt nopeasti, ja versioiden kolme ja neljä kautta on siirrytty jo versioon viisi. PHP 5 on täysiverinen ohjelmointikieli, joka julkaistiin heinäkuussa 2004. Sen ytimenä on Zend Engine II, joka muun muassa tukee olioohjelmointia ja sisältää sisäänrakennetun tietokantamoottorin (SQLite) (Wikipedia 2009.)

PHP-ohjelmointikielen komentoja voi kirjoittaa suoraan HTML-koodin sisään, ja niitä voidaan kirjoittaa mihin tahansa kohtaan HTML-koodia tai mitä tahansa WWW-palvelimella olevaa tekstimuotoista tiedostoa. PHP on tulkattava kieli, mikä tarkoittaa, että HTML-dokumentin sisällä oleva PHP-koodi ajetaan aina palvelimella tulkkausohjelman avulla, juuri ennen kuin sivu lähetetään selaimeen. PHP asennetaan WWW-palvelimen laajennuksena, joka mahdollistaa monimutkaistenkin sovellusten toteutuksen palvelimella. Tästä syystä sivujen lukijat ja sovellusten käyttäjät eivät näe PHP-koodia, kun he tutkivat sivun lähdekoodia selaimessa. PHP ei siis kilpaile selaimissa toimivan JavaScriptin kanssa vaan on täysin palvelin pään ohjelmointikieli. Seuraavassa koodiesimerkissä voi nähdä yksinkertaisen esimerkin PHP-kielen syntaksista. (Heinisuo 2001, 16.)

```
<p>
<?php
echo”Hello world!”;
?>
</p>
```

Kun yllä oleva koodi suoritetaan, selaimesta näkyy ainoastaan ohjelman tulostus. Esimerkistä siis tulostuisi selaimeen Hello world! . PHP-tiedostoja voidaan tutkia ainoastaan silloin, kun käsitellään palvelimella olevia tiedostoja FTP:n tai tiedostojen jaon kautta. Tästä syystä tiedostoja ei voida ajaa oman kotitietokoneen kovalevyltä, ellei kotikoneelle ole asennettu PHP-tulkkia. PHP suoritetaan WWW-palvelimella ja PHP-sivut on myös ladattava WWW-palvelimelta. PHP:n voima on siinä, että yksinkertainen tulostuskomento voitaisiin korvata monimutkaisemmilla komennoilla, jotka tekisivät tietokantahaun ja tulostaisivat haun osaksi HTML-dokumenttia. Tietokantoja käsitellään PHP:sta käsin melko yksinkertaisilla komennoilla. (Heinisuo 2001, 17.)

Heinisuo (2001, 17) toteaa, että PHP sisältää kaikki ohjelmointikielille tyypilliset rakenteet, kuten for- ja while-silmukat, if-ehtolauseet, sekä muuttujat ja funktiot. Lisäksi siinä on, PHP4 versiosta lähtien, olio-ohjelmoinnille tuki, joka mahdollistaa sovellusten toteuttamisen vaihtoehtoisesti luokkien avulla.

PHP-ohjelmointikieli mahdollistaa rakenteillaan sekä komennoillaan SQL-komentojen välittämisen tietokannalle. PHP-kielen avulla voidaan lukea tietoja tietokannan tauluista, niin että ne generoituvat web-sivuina. Tämä toimii myös toiseen suuntaan, sillä web-sivuilla olevilta lomakkeilta voidaan PHP-kielen avulla välittää tietoja tietokantaan. PHP on joustava ja suhteellisen yksinkertainen ohjelmointirajapinta tietokantapohjaisten web-sivujen sekä muiden selainpohjaisten sovellusten tekemiseen. (Heinisuo 2001, 60.)

3.4 Relaatiotietokannat

3.4.1 Yleisesti

Tässä työssä esitellään relaatiotietokannat lähinnä yleisellä tasolla ja kerrotaan mitä relaatiotietokanta on ja miten se toimii. Tietotekniikassa tietokannalla tarkoitetaan tietovarastoa. Siihen on koottu tietoja, joiden välillä on oltava looginen yhteys toisiinsa. Tietokantojen koko voi paljonkin vaihdella: pienimmillään tietokanta voi olla yhteen tiedostoon tallennettu taulukko ja suurimmillaan se voi käsittää useita miljoonia tietueita, jotka ovat tallennettu useista kiintolevyistä koostuville levypaikoille. Niihin voidaan tallentaa eri formaateissa olevaa tietoa, esimerkiksi tekstiä, ääntä tai videokuvaa. Relaatiotietokanta on suosituin tietokantaratkaisu Web-sovelluksissa. Relaatiotietokannan hallintajärjestelmiä on tarjolla lukuisia joista suosituimpia ovat ilmaiset MySQL ja PostGreSQL. (Asmala 2006; Wikipedia 2009c.)

3.4.2 Suunnittelu

Tietokannan hyvä suunnittelu on erittäin olennaista, mikäli halutaan saada aikaan toimiva ja eheä tietokantarakenne. Suunnittelun voi jakaa eri vaiheisiin, jotka koostuvat vaatimusmäärittelyistä ja analyysistä, käsitteellisestä mallintamisesta, loogisesta suunnitteluvaiheesta sekä toteutusvaiheesta. (Ekonoja, Lahtonen & Mäntylä 2004.)

Käsiteanalyysi

Käsiteanalyysilla pyritään niin sanottuun todellisuuden mallintamiseen, joka tarkoittaa sitä, että jäsennetään tietyn kohdealueen kiinnostuksen kohteet, niiden väliset riippuvuudet sekä ominaisuudet. Asiat on tarkoitus kuvata yhtenäisenä käsitteistönä eli käsitemallina, joka koostuu yksilöidyistä, luokitelluista sekä ryhmitellyistä kiinnostuksen kohteista. Tieto on tarkoitus tallentaa vain kerran. Käsitemalli on myös työryhmän yhteinen näkemys siitä, millä tavoin he näkevät kohdealueen. (Asmala 2006.)

Käsitemalli kuvataan ER-kaaviona, josta nopeasti näkee käsiteanalyysin käsitteet, niiden ominaisuudet, niiden väliset suhteet sekä toimintatavat toisiinsa nähden. Eri käsiteluokkia ovat esineet, henkilöt, tapahtumat sekä käsitteelliset yksilöt, joita ovat esimerkiksi kustannuspaikka ja työsuhde. Näistä asioista tallennetaan tietoa tietokantaan. Käsitteet kuvataan ER-kaaviossa yleensä laatikolla. (Asmala 2006.)

Ominaisuudet eli attribuutit kuvaavat käsitettä. ER-kaaviossa ne merkitään ovaalina. Jos käsite on esimerkiksi asiakas, niin käsitteen ominaisuuksia ovat asiakkaan nimi, osoite ja puhelinnumero. Jokainen asiakas tarvitsee yksilöivän tiedon eli perusavaimen, tässä tapauksessa asiakastunnuksen. Tämä avainominaisuus on siis jokaiselle käsitteelle ainutkertainen. (Asmala 2006.)

Käsitteiden välisillä suhteilla tarkoitetaan riippuvuutta, johon tarvitaan vähintään kahta kohdetta. Suhteet merkitään ER-kaaviossa timantti-kuviona. Kardinaalisuus suhteiden välillä kertoo kuinka moneen suhteeseen kohde voi samanaikaisesti osallistua ja kuinka monta kohdetta voi osallistua samaan aikaan siihen. Kardinaalisuus voi olla:

- yhden suhde yhteen (1 to 1)
- yhden suhde moneen (1 to M) tai monen suhde yhteen (M to 1)
- monen suhde moneen (M to M).

(Ekonoja, Lahtonen & Mäntylä 2004.)

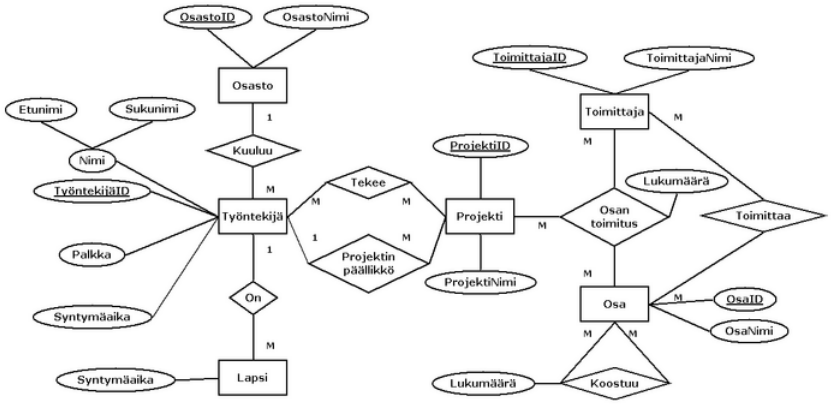
Tarveanalyysi

Tarveanalyysilla tarkennetaan käsitemallia. Mallia tarkastetaan uudelleen uusien tietotarpeiden valossa ja sitä myös testataan. Tarvittaessa siihen lisätään uusia käsitteitä, ominaisuuksia ja suhteita. Tietotarpeita voivat olla sovelluksen ikkunoita, näyttöjä, raportteja tai muita ohjelmia, jotka tulevat käsittelemään tietokantaa. Testausta varten tarveanalyysissa selvitetään myös tieto-, käyttäjä- sekä tapahtumamäärät. (Asmala 2006.)

Käsitemalli

Käsitemalli on käsiteanalyysin tulos. Yleensä se kuvataan ER-kaaviona, joka toimii tietokannan piirustuksena toteutusvaihetta varten. Käsiteanalyysistä saatua käsitemallia voidaan pitää raakaluonnoksena, jossa ei vielä pyritä kohdealueen tarkkaan kuvaamiseen. Siinä ei oteta vielä huomioon teknillisiä yksityiskohtia, suorituskykyä tai tietokannan fyysistä rakennetta. Käsitemalli on tuote- ja tietokantariippumaton, eli sen pohjalta voidaan muodostaa tietokanta useilla erityyppisillä hallintajärjestelmillä. (Asmala 2006.)

Asmalan (2006) mukaan Käsitemallissa jokainen asia voidaan merkitä vain yhdellä tavalla joko käsitteeksi, suhteeksi tai ominaisuudeksi. Vain käsitteillä voi olla ominaisuuksia ja suhteet muodostetaan vain kahden käsitteen välille. Seuraavassa kuvassa (KUVA 2.) havainnollistetaan käsitemallissa olevia käsitteitä, ominaisuuksia ja käsitteiden välisiä suhteita.



KUVA 2. ER-kaavio (Ekonoja, Lahtonen & Mäntylä 2004b)

ER-kaavion transformointi

ER-kaavio muutetaan relaatioksi, eli tauluksi, seuraavien perussääntöjen mukaan:

- Jokainen käsite muutetaan relaatioksi.
- Jokainen M to M -suhde muutetaan tavalliseksi relaatioksi.
- M to 1 -suhteet liitetään jo olemassa oleviin relaatioihin.

(Ekonoja, Lahtonen & Mäntylä 2004c.)

Normalisointi

Tietokannan normalisointi tehdään vaiheittan, minkä avulla saadaan tietokannan rakenne tukemaan tietojen ehjää tallennusta. Jokainen vaihe vähentää tiedon redudanssia, eli tiedon moneen kertaan tallentumista, ja parantaa tallennetun tiedon eheyttä. Toisaalta tämä kuitenkin hidastaa usein tietokantojen suoritusnopeutta. Denormalisointia, joka on normalisoinnin vastakkainen tapahtuma, käytetään monesti tehokkuuden parantamiseen. (Wikipedia 2009d.)

Normaalimuotoja on kuusi, ja tietokantaa pidetään normalisoituna, mikäli se täyttää ehdot neljänteen normaalimuotoon asti. Tietokannan on täytettävä tällöin myös järjestysluvultaan pienempien normaalimuotojen ehdot. Käytännössä normalisointi toteutetaan siirtämällä ominaisuuksia toiseen joko olemassa olevaan tai täysin uuteen tietokantatauluun. (Wikipedia 2009d.)

3.4.3 SQL

SQL, *Structured Query Language*, on pitkälle standardoitu relaatiotietokantojen kysely- sekä määrittelykieli. SQL on täysin merkkipohjainen kieli, eli se on täysin ASCII-pohjainen eikä pohjaudu binääreihin, näin se ei myöskään tarvitse erillistä kääntäjää. Sen rakenne muistuttaa etäisesti englannin kieltä, mistä johtuen alkuperäinen englanninkielinen nimi *Stuctured English Query Language*. Komentosanoina käytetään muun muassa `select`, `update`, `delete`, `where` ja `order by`. Tietokantataulujen luomisen ja poiston lisäksi SQL:llä voidaan käsitellä tauluihin

tallennettuja tietoja. SQL-kielen komennot jaotellaan määrittely- ja käsittelykomentoihin. (Heinisuo 2001, 60.)

Kyselyiden tekeminen tietokantaan ei ole SQL-kielen ainoa ominaisuus. Sillä voidaan myös:

- muuttaa sekä määritellä tietokantojen rakennetta
- lisätä, muuttaa sekä poistaa tietoja tietorakenteesta
- hoitaa turvallisuuttaa ja antaa valtuuksia sen käytölle
- ohjata tapahtumankäsittelyjä
- hallita upotettua SQL:ää ja kohdistimia.

(Asmala 2006.)

SQL-kieltä voidaan käyttää usealla eri tavalla ja monissa eri yhteyksissä. Se, millä tavalla kieltä käytetään, riippu käyttötarkoituksesta. Sitä voidaan käyttää vuorovaikutteisesti, upotetusti tai dynaamisesti. Vuorovaikutteisessa SQL:n käytössä käskyjä annetaan omassa ikkunassaan esimerkiksi komentoriviltä, ja vastaukset saadaan samaan ikkunaan. Upotetussa SQL:ssä käskyt upotetaan ohjelmointikielen joukkoon tai sovelluskehittimeen. Tässä vastaukset saadaan suoraan ohjelmointikielen muuttujiin. Dynaamisessa SQL:ssä käskyt luodaan dynaamisesti ohjelmakoodissa. (Asmala 2006.)

Niiden Web-sovelluksien, jotka käyttävät tietokantoja tiedon tallennuspaikkana, etuna on aina skaalautuvuus eli kasvupolku on taattu. Esimerkiksi yksinkertaiseen tauluun, johon voi lisätä linkkejä ja kuvaukset niille, voidaan lisätä niin sata kuin miljooniakin linkkejä. Tekstitiedostoihin voidaan myös tallentaa tietoja aivan kuten tietokantoihin, mutta tiedon määrän kasvaessa törmätään monenlaisiin ongelmiin. Nämä ongelmat yleensä ratkaistaan siirtymällä tietokantaan tietojen tallennukseen ja hakuun. Esimerkiksi MySQL-tietokannan hyväksi käyttäminen PHP-komentojen avulla on yksinkertaisempaa ja varmempaa kuin samaisten tietojen tallentaminen Web-palvelimella oleviin tiedostoihin. (Heinisuo 2001, 61.)

3.4.4 Toteutus

Yksi relaatiotietokannan keskeisimmistä komponenteista on taulu. Taulu koostuu kentistä, jotka ovat tallennettu tauluun sarakkeittain sekä tietueista, jotka on tallennettu riveittäin. Taulun pienin tietoyksikkö on yksittäinen kenttä, johon voidaan tallentaa esimerkiksi henkilön nimi. Kentän tietotyyppi määritellään sen mukaan, millaista tietoa siihen halutaan tallentaa. Erilaisia tietotyyppejä voivat olla merkkijonot, numerot, päivämäärät, aikaleimat ja binääritieto. Jonkin taulun kentistä on toimittava tietueen tunnistimena, joka sisältää yksilöllisen tiedon taulun jokaiselle riville. Tätä kutsutaan taulun pääavaimeksi, ja sen avulla haluttu tietue löydetään taulusta nopeasti. Relaatiotietokanta muodostuu yleensä useammasta taulusta. Näissä kahden taulun tiedot yhdistetään viiteavainkentän avulla, mikä lisätään liitettävään tauluun. Tähän viiteavainkenttään tallennetaan toisen taulun pääavainarvo. (Mureakuha 2006a.)

Relaatiotietokanta voidaan toteuttaa monella eri tietokantaratkaisulla. Yleensä SQL-kielen peruskomennot ovat kuitenkin suhteellisen samanlaisia eri tietokantaratkaisuissa, koska ne ovat pääosin ANSI-SQL yhteensopivia. Syntakseissa voi kuitenkin olla joitakin eroavaisuuksia. (Mureakuha 2006b; 2kmediat.com 2000-2008k.)

4 KÄYTETTÄVYYS OSANA OHJELMISTON SUUNNITTELUA

4.1 Käytettävyyden tarpeellisuus

Käytettävys on menetelmä- ja teoriakenttä, jonka avulla määritellään, mikä olisi paras tapa saada käyttäjän ja laitteen yhteistoiminta tehokkaammaksi ja mielyttävämmäksi käyttäjän näkökulmasta. Käytettävyydessä on siis kyse ihmisen ja koneen vuorovaikutuksesta. Käytettävyys käyttää hyväksi kognitiivisen psykologian sekä ihmisen ja koneen vuorovaikutuksen tutkimusta. (Sinkkonen, Kuoppala, Parkkinen & Vastamäki 2006.)

Tuotteen ominaisuutena käytettävyys kuvaa, kuinka sujuvasti tuotteen toimintoja käyttäjä käyttää päästäkseen haluamaansa päämäärään. Nielsenin mukaan käytettävyys on osa tuotteen käyttökelpoisuutta. Englannin kielessä käytetäänkin termin käytettävyys, usability, rinnalla usein ihminen-tietokone-vuorovaikutusta (Human-Computer-Interaction, HCI) puhuttaessa tietoteknisten sovellusten käytettävyydestä. Käytettävyys ei ole pelkästään tietoteknisten tuotteiden ominaisuus. Esimerkiksi tavallisella ovella on käyttöliittymä siitä, miten se avataan ja suljetaan. Käytettävyys voi olla hyvä tai huono. Esimerkiksi oven käyttöliittymässä ei saa selville, mistä reunasta ovi pitäisi avata, mutta lisäämällä pienen muutoksen, jonkin visuaalisen yksityiskohdan, voi käytettävyys parantua ratkaisevasti. (Kuutti 2003, 13.)

Käytettävyys koostuu osa-alueista, joita ovat opittavuus, muistettavuus, tehokkuus, pieni virhealttius ja miellyttävyys. Käytettävyyden ja käyttöliittymien yhteydessä puhutaan usein intuitiivisesta käyttöliittymästä. Intuitiivisuus on tavallaan tuttuus aikaisemman kokemusmaailman valossa. Intuitiivinen on kuitenkin hyvin yksilöllinen käsite, koska se perustuu yksilön aikaisempaan kokemusmaailmaan. (Kuutti 2003, 13.)

Perinteisesti erittäin tiukasti rajattuna käytettävyys on käsittänyt vain käyttöliittymät. Useat markkinoille tulevat tuotteet ovat kuitenkin epäonnistuneet, vaikka niissä olisi ollut hyvinkin pitkälle ajateltu ja viilattu käyttöliittymä. Myös tuotteen rakenne ja sen ominaisuuksien valinta vaikuttaa käytettävyyteen. Hyvän käyttöliittymän lisäksi tuotteessa tulisi olla juuri käyttäjän tarvitsemat ominaisuudet selkeästi esillä. (Kuutti 2003, 15.)

ISO määrittelee käytettävyyden kokonaisuudeksi, joka kuvaa, miten hyvin tietyt käyttäjät kykenevät käyttämään käytössään olevia työvälineitä tiettyjen tehtävien suorittamiseen tietyssä ympäristöissä tavoitteiden saavuttamiseen. ISO määrittelee standardissa ISO 9241 käytettävyydessä tarkasteltaviksi kohteiksi käyttäjän, hänen tehtävänsä, työvälineensä ja toimintaympäristönsä. Tämä standardi on varsin ajanmukainen, modernit käyttäjäkeskeiset suunnittelumenetelmät kun korostavat kontekstin, eli työtehtävien ja ympäristön merkitystä käytettävyyden suunnittelussa. (Kuutti 2003, 15.)

Kuutin (2003, 19) mukaan käytettävyyteen ei kiinnitetä tarpeeksi huomiota ohjelmistoprojekteissa. Heikon käytettävyyden seuraukset huomataan helposti, jos on enemmänkin käyttänyt tietoteknisiä sovelluksia. Syynä tähän on monesti puutteelliset tiedot. Monessakaan valmiissa ohjelmistotuotannon prosessimallissa ei oteta huomioon käytettävyyttä. Käyttäjäkeskeisiin suunnittelumenetelmiin ei jakseta yleensä perehtyä tarpeeksi, ja niitä pidetään myös liian suurina kustannuserinä suunnitteluprosessissa. Tämä ei välttämättä pidä paikkansa, sillä pitemmällä aikavälillä käytettävyyden huomioonottaminen on osoittautunut taloudellisesti kannattavaksi.

Kuutti (2003, 19) toteaa, että käytettävyys kannattaa sisällyttää luonnollisena osana käytettävää ohjelmistotuotantoprosessia. Tällöin siitä tulisi rutiininomaisempaa, eikä se alana tuntuisi enää vaikealta omaksua. Hänen mukaansa käytettävyyttä tulisi myös testata, kuten muitakin ohjelmiston osia. Testaamisen tulisi olla suunnitelmallista, ja se kannattaisi laatia jo ennen ohjelmiston toteuttamista tai vielä parempi, rinnakkain toteutuksen kanssa. Jos testaamisen yhteydessä löytyy jotain korjauksia, ne tehdään, jos ja kun kaikkia tavoitteita ei ole yleensä jostain syystä saavutettu.

Käytettävyystyössä kannattaa dokumentoida hyvin prosessin aikana, sillä se säästää pidemmällä tähtäimellä paljon aikaa ja vaivaa. Tästä johtuen se säästää myös rahaa. Erityisesti epäonnistuneiden ratkaisujen tarkka dokumentointi kannattaa, sillä virheistä voidaan oppia ja säästytään tekemästä samoja virheitä uudestaan. (Kuutti 2003, 19.)

4.2 Heuristinen arviointi

Tässä opinnäytetyössä perehdytään nimenomaan kevyempään noin kymmenen kohtaa sisältävään Nielsenin listaaan, mikä on erityisen käytetty heuristinen arviointi sekä testausmenetelmä käyttöliittymä suunnittelussa, varsinkin tietoteknisissä sovelluksissa. Nielsenin listassa käytettävyysopit on tiivistetty muutamaan kohtuullisen helposti opittavaan ja sovellettevaan sääntöön. Oikein käytettynä kevyemmälläkin heuristiikalla saadaan paljastettua kaikkein yleisimmät sekä kaikkein vakavimmat käytettävyysongelmat. (Kuutti 2003, 47.)

Seuraavassa listassa on Nielsenin heuristiset säännöt:

- Vuorovaikutuksen käyttäjän kanssa tulee olla yksinkertaista ja luonnolista.
- Vuorovaikutuksen tulee käyttää käyttäjän kieltä.
- Käyttäjän muistin kuormitus tulee minimoida.
- Käyttöliittymän tulee olla yhdenmukainen.
- Järjestelmän tulee antaa käyttäjälle kunnollista palautetta reaaliajassa.
- Ohjelmassa ja sen osissa tulee olla selkeät poistumistiet.
- Oikopolkuja ja tehokasta työskentelyä tulisi tukea.
- Virheilmoitusten tulee olla selkeitä ja ymmärrettäviä.
- Virhetilanteisiin joutumista tulisi välttää.
- Käyttöliittymässä tulee olla kunnolliset avustustoiminnot ja dokumentaatio.

(Kuutti 2003, 50)

Arviointi tämän listan pohjalta onnistuu täysin kokemattomaltakin arvioijalta, mutta sen täydellinen hyödyntäminen ja kaikkien kohtien ymmärtäminen vaatii tarkempaa taustatietoa asiasta. (Kuutti 2003, 50.)

4.3 Käytettävyys visuaalisen suunnittelun osana

Visuaalinen suunnittelu on oleellinen osa käytettävyyttä, sillä sovelluksen ulkonäkö on se, mitä käyttäjä sovellusta käyttäessään katselee. Toisin sanoen tuoteen ulkoasu vaikuttaa myös sen käytettävyyteen, eikä ulkoasu ole pelkästään visuaalinen koriste. Ohjelmistossa ensisilmäyksellä vain pelkältä koristeelta näyttävällä yksityiskohdalla voi olla merkittävä osa sovelluksen käytettävyydessä. (Kuutti 2003, 90.)

Kuutin (2003, 90) mukaan visuaalisessa suunnittelussa suunnitelmallisuus on tärkeää, ja suunnittelussa yhdenmukaisuus on ensiarvoisen tärkeää. Samaa kerran valittua suunnittelun linjaa tulisi käyttää koko sovelluksessa, mikä tukee hyvin Nielsenin yhdenmukaisuus-sääntöä.

Vaikka visuaalinen suunnittelu on vain yksi osa käytettävyyttä, on se tärkeä osa eikä sitä pitäisi sivuutta. Hyväkään visuaalinen suunnittelu ei kuitenkaan pelasta epäonnistunutta rakennetta, sillä myös rakenne vaikuttaa käytettävyyteen. Esimerkiksi jos websivujen HTML-lähdekoodi on rakennettu huonosti ja sisältö on sen puolesta miten sattuu, on sitä CSS:n avulla mahdotonta rakentaa uudelleen. Usein samaa pohjalla käytettävää rakennetta voidaan käyttää visuaalisesti hyvin erilaisten käyttöliittymien kanssa. Käyttäjäkeskeisistä suunnittelumenetelmistä muun muassa Contextual Design opettaa, että saman rakenteen päälle voidaan tarpeen mukaan rakentaa niin graafinen kuin tekstipohjainenkin käyttöliittymä. (Kuutti 2003, 91.)

Vaikka suunnittelu olisi kuinka hyvin tehty, lopputuloksen onnistuneisuus on aina hyvä varmistaa testaamalla. Virheiden havaitseminen aikaisessa vaiheessa on helpompaa ja säästää kustannuksissa. Visuaalinen suunnittelu on siis syytä testata aikaisessa vaiheessa. Testausta varten kannattaa ensin tehdä luonnoksia paperille ja mahdollisuuksien mukaan testata se joko asiantuntija-arviona tai käytettävyystestissä. (Kuutti 2003, 91.)

4.4 Käyttäjän tunteminen

Käytettävyyden maksimoimiseksi käyttöliittymän suunnittelussa on erittäin tärkeää tuntea käyttäjä erittäin hyvin. Käyttäjän mallintamiseen on kehitetty lukuisia erilaisia tekniikoita, joiden avulla käyttäjästä saadaan kerättyä kaikki tarpeellinen tieto sekä esitettyä se mahdollisimman käyttökelpoisessa muodossa. Yksinkertai-

simmillaan se voi olla vain lista, missä on listattu vapaamuotoisia tietoja käyttäjästä. Kehittyneemmät käyttäjäkeskeiset suunnittelumenetelmät voivat laatia hyvinkin tarkkaan määritettyjä malleja, joista eri vaiheiden kautta saadaan jalostettua kehittävän järjestelmän määrittely. (Kuutti 2003, 117.)

Yksinkertaisimmillaan suunnittelija voi vierailla käyttäjän luona tuotteen luontaisella käyttöpaikalla ja tutkia, miten ja mihin käyttäjä tuotetta käyttää. Tämä ei kuitenkaan riitä, jos halutaan saada aikaan erinomaisia tuloksia, sillä käyttöliittymän suunnitteluun tarvittava informaatio on suoraan johdettavista käyttäjästä irti saadusta informaatiosta. Kun käytettävissä on erittäin tarkka määritelmä käyttäjästä ja tämän tavoitteista, voidaan tästä teoriassa johtaa suoraan käyttäjälle erittäin hyvin sopiva käyttöliittymä. Käyttäjän täydellinen mallintaminen on käytännössä mahdotonta, jonka lisäksi se on hankalaa sekä kallista. Vielä kalliimmaksi kuitenkin tulee kehittää järjestelmä, joka on suunniteltu lähtökohtaisesti väärin, koska käsitys ja malli käyttäjästä ovatkin olleet vääriä tai väärin ymmärrettyjä. Tällainen tuote on käyttökelvoton, jolloin koko tuotekehitys on ollut turhaa. (Kuutti 2003, 117.)

4.5 Käyttäjäkeskeiset suunnittelumenetelmät

Käyttäjäkeskeisessä suunnittelussa otetaan käyttäjä keskeiseksi osaksi suunnitteluprosessia. Esimerkiksi joissain kokeiluissa käyttäjä on otettu suunnitteluryhmän tasavertaiseksi ryhmän jäseniksi. Kokemukset eivät olleet kuitenkaan kovin rohkaisevia, eivätkä loppukäyttäjät ymmärtäneet suunnittelijoiden kieltä ja kaavioita. Suunnittelijat puolestaan eivät saaneet käyttäjistä paljoakaan irti. Käyttäjät eivät osanneet tai ymmärtäneet ilmaista itseään tarpeeksi selkeästi. (Kuutti 2003, 140.)

Kuutin (2003, 141) mukaan käyttäjäkeskeinen suunnittelu voi olla hyvin yksinkertaista. Minimissään määritelmä käyttäjäkeskeiselle suunnittelulle sisältää kolme vaihetta:

1. tärkeimpien projektiin osallistujien toistensa tapaaminen, jossa sovitaan projektin eri vaiheista ja varmistetaan, että kaikilla on samat tavoite
2. testaaminen paperiprototyypin avulla
3. käyttäjätestit toimivalla prototyypillä. Pyritään paljastamaan tuotteessa edelleen olevat käytettävyysongelmat.

5 GRAAFISEN KÄYTTÖLIITTYMÄN OMINAISUUDET

5.1 Määrittely, Graphic User Interface

Jokainen asia näyttää ja tuntuu joltain. Look and feel-termiä käytetään usein ohjelmistosuunnittelussa graafisen käyttöliittymän yhteydessä kuvaamaan sen suunnittelun ulkomuotoa. Käyttöliittymän ”look”-osaan sisältyy käytetyt värit, muodot, layout ja tekstin kirjasintyylit. ”Feel”-osan muodostavat käyttöliittymän dynaamisten elementtien, kuten esimerkiksi nappien ja menu-elementtien, toimintatavan. (Wikipedia 2009e.)

Käyttöliittymä voi olla mitä vain. Kaikella mitä ihminen käyttää ja on ihmisen käytettävissä on myös määriteltävissä jonkinlainen käyttöliittymä. Se ei välttämättä aina ole graafinen tai millään tavalla visuaalinen, vaan se voi olla vain tieto siitä miten jotain asiaa tai esinettä käytetään. Graafinen käyttöliittymä tarkoittaa yleensä sellaista käyttöliitymää, mikä on nähtävissä sekä kätytettävissä joltain näyttölaitteelta. Siihen yleensä tarvitaan syöttö ja tulostuslaite. Syöttölaite voi olla esimerkiksi tietokoneen hiiri ja tulostuslaitteena tietokoneen näyttö. Tässä tapauksessa käsittelemme tietokoneen ruudulta katsottavia graafisia käyttöliittymiä, jotka on toteutettu yleensä web-ympäristössä.

5.2 Kuva vai teksti

Aina siitä lähtien kun graafisia käyttöliittymiä on ollut, on pohdittu sitä, kumpi on parempi tiedon esittämistapa: kuva vai teksti. Molemmilla esitystavoilla on puolensa: kuva on yleensä nopeampi tunnistaa, teksti puolestaan yksiselitteisempi ja ymmärrettävämpi. Kuva, toisin sanoen ikoni, voi olla aikaisemmasta kokemusmaailmasta tuttu tai sen joutuu opettelemaan uudestaan. Hyvä kuva on kuitenkin hyvin intuitiivinen, jolloin sen merkitys on helposti arvattavissa, vaikkei asiaa niin hyvin tuntisikaan. Kuva voi olla kulttuurisidonnainen, tietyssä kulttuurissa itsestään selvät symbolit eivät välttämättä ole sitä jossain aivan erilaisessa kulttuurissa. Tietyt, varsinkin fysikaallisiin ilmiöihin liittyvät kuvat ovat hyvin kulttuuririippumattomia, esimerkiksi pisara kuvaa vettä ja liekki tulta. Teksti on sidottu kieleen ja toimii vain, jos osaa kyseistä kieltä. Hyvät symbolit voivat kuitenkin olla lähes kansainvälisiä. (Kuutti 2003, 98.)

Symboli eli ikoni voi kuvata suoraan jotain asiaa, tällaisia ovat esimerkiksi roskakori tai postilaatikko, ja ne ovat yleensä helpoiten ymmärrettyjä. Symboli voi kuvat myös jotain tekemistä, esimerkiksi kynä voi kuvata piirtämistä ja levyke tallentamista. Myös tällaiset symbolit ovat helposti ymmärrettyjä. Ominaisuutta kuvaavat symbolit voivat olla kuitenkin vaikeampia ymmärtää, esimerkiksi kilpikonna voi tarkoittaa hidasta ja jänis nopeaa. Tämän tyyppiset asiayhteyteen huonosti sopivat symbolit voivat olla jo vaikeampia tulkita. Hankalimmin tulkittavia symboleja ovat sanaleikkeihin tai synonyymeihin perustuvat symbolit. Lisäksi ne ovat yleensä kieleen sidottuja, jolloin menetetään osa symbolin käyttämisen eduista. Usein käytetäänkin symbolia ja tekstiä rinnakkain, jolloin saavutetaan suurin osa molempien hyvistä puolista. Kannattaa muistaa myös se, että tuotteessa käytettävät symbolit tulisi aina testata oikeilla tuotteen käyttäjillä. (Kuutti 2003, 100.)

5.3 Värit

Värien valinnassa yksi tärkeimmistä kriteereistä on värien toimivuus. Käyttöliittymän tulisi olla selkeä ja helppolukuinen. Värien kanssa yleisempiä käytettävyysongelmia on liiallinen ja epäjohdonmukainen värien käyttö. Useimpien lähteiden mukaan kerrallaan käyttöliittymässä käytettävien värien määrä tulisi rajata maksimissaan viiteen. Varsinkin jos käyttäjän on muistettava värien merkitys, kannattaa pitää mielessä ihmisen lyhytkestoisen muistin rajoitukset. Jos värillä halutaan kuvata esimerkiksi jonkin asian määrää, kannataa eri määrille valita eri värejä. Jos eri määrien kuvaamiseen halutaan käyttää saman värin eri sävyjä, esimerkiksi vaalea sävy tarkoittaa vähän ja tumma paljon. Silloin tulisi välttää sinisen värin sävyjen käyttämistä, koska ne erottuvat huonoimmin toisistaan. (Kuutti 2003, 100.)

Tekstissä kannattaa olla erityisen konservatiivinen värien käytön kanssa. Silloin tärkeintä on luettavuus ja tutkimusten mukaan luettavuudeltaan paras yhdistelmä on musta teksti valkoisella pohjalla. Voimakkaita väriyhdistelmiä tulisi välttää, esimerkiksi puhdasta sinistä ja punaista ei pitäisi käyttää vierekkäin, varsinkin tekstin ja taustan väreinä. Tällainen teksti on erittäin vaikealukuista. Kontrastiero puolestaa helpottaa lukemista. Tumma teksti vaalealla pohjalla tai toisin päin, toimivat varsin hyvin. (Kuutti 2003, 100.)

Väreihin liittyy usein myös konventioita, mikä tarkoittaa samantyyppisistä tuotteista tuttua, aikaisemmin opittua. Esimerkkinä väreihin liittyvästä yleisestä konventiosta ovat www-sivujen siniset alleviivatut linkit. Konventiota kannattaa hyödyntää, sillä niillä nopeutetaan aikaisemmin vastaavaan tuoteryhmään tutustuneiden käyttäjien toimintaa. Silloin tuotetta voidaan sanoa intuitiiviseksi. (Kuutti 2003, 101.)

Väri on hyvin voimallinen keino esimerkiksi huomion kiinnittämiseen ja tuotteen tekemisestä selkeämmäksi ja tehokkaammaksi. Kuten monet muutkin hyvin tehokkaat työkalut, myös värin käyttäminen vaatii taitoa ja epäonnistuessaan värien käytöllä voi saada aikaan suurta harmia. Sen vuoksi värien käytössä kannattaa pitäytyä totutuissa hyviksi havaituissa yhdistelmissä, ellei täsmälleen tiedä, mitä on tekemässä. (Kuutti 2003, 100.)

5.4 Käyttäjän huomion ohjaaminen

Välillä halutaan kiinnittää käyttäjän huomio johonkin tiettyyn käyttöliittymän osaan. Siellä voi olla jotain tärkeää, tai sitten siellä on esimerkiksi virheellinen korjaamista vaativa syöte. Tekstin sisällä tehokas tapa ohjata huomiota on tekstin lihavoiminen. Toinen vaihtoehto olisi kirjoittaa isoilla kirjaimilla, mutta tämä mielettään yleensä huutamiseksi, millä voi olla kielteinen psykologinen vaikutus. Siksi lihavoiminen on parempi tapa. Lisäksi isokirjaiminen teksti on hitaampaa ja vaikeampaa lukea erityisesti ikääntyville ihmisille. Kannattaa siis ottaa myös käyttäjäryhmä huomioon. (Kuutti 2003, 92.)

Väri on tehokas ja hyväksi havaittu tapa ohjata huomiota. Käyttöliitymän normaalista värityksestä poikkeava väri kiinnittää varmasti katsojan huomion. Esimerkiksi virheellinen syöte voidaan kuvata punaisella värillä muutoin väritykseltään neutraalissa ympäristössä. Siinä tapauksessa tämänlainen tehokeino on toimivin. Kirjavassa käyttöliittymässä aistit turtuvat, eikä mikään väri enää kiinnitä huomiota. Tyhjän tilan käyttäminen on myös yksi tapa ohjata käyttäjän huomiota. Huomio kiinnittyy helposti tyhjän tilan ympäröimään kohtaan käyttöliittymässä. (Kuutti 2003, 93.)

Myös erilaisilla kuvilla voidaan kiinnittää käyttäjän huomiota. Kuvan huomioarvoon vaikuttaa kuvan koko ja väritys sekä kuvan sisältö. Ihminen kiinnittää par-

haiten huomion asioihin, jotka ovat joko pelottavia tai kiinnostavia. Myös henkinen tila vaikuttaa huomion kiinnittymiseen. Esimerkiksi nälkäisenä ruoan kuva kiinnittää varmasti huomion ja mielenkiintoiset asiat kiinnittävät huomion niin kuvana kuin tekstinä. Huomion kiinnittämisessä olisi tärkeää pitää kuitenkin jonkinlainen maltti. Liika häly ja keskenään huomiosta kilpailevat elementit pilaavat helposti käyttöliittymän. Käyttäjän huomiota kannattaa ohjata, mutta hyvin harkitusti vain silloin kun siihen on oikeasti tarvetta, eikä kovin moneen asiaan samalla kertaa. (Kuutti 2003, 95.)

Kuutin (2003, 95) mukaan esitettävän informaation määrää on myös mietittävä. Käytettävyysoppien mukaan käyttäjälle tulisi näyttää kaikki käyttäjän tarvitsema informaatio, mitään ei saisi jättää pois. Kannattaa kuitenkin välttää ylimääräisen ja turhan tiedon esittämistä, koska se turruttaa käyttäjän ja täten jotain oleellista voi jäädä huomaamatta.

5.5 Sommittelu

Käyttöliittymän visuaalisen ulkoasun sommittelulla tarkoitetaan yleensä yhden kokonaisuuden, tyypillisesti näytön, sisällön sijoittelua. Normaalisti länsimainen ihminen lukee vasemmalta oikealle ja ylhäältä alas. Tämä on usein luonnollisin etenemissuunta esimerkiksi lomaketta täyttäessä tai käyttöliittymää käyttäessä. Ihmisen huomio ei kuitenkaan automaattisesti kulje tätä reittiä. Etenkin jos graafisessa käyttöliittymässä on voimakkailla visuaalisilla ärsykkeillä, huomio on ohjattu ensiksi jonnekin muualle kuin vasempaan yläkulmaan. Tämä voi kuitenkin hidastaa ihmisen havainnoimisprosessia ja hidastaa normaalia etenemistä. Siksi visuaalisilla vihjeillä tulisi normaalisti pitäytyä normaalin lukusuunnan mukaisessa järjestyksessä ja poikeata siitä vain harkiten sekä hyvästä syystä. Kansainvälisiä tuotteita suunniteltaessa tulee toki muistaa, että kaikkialla ei lueta kuten meillä, jolloin asioiden havainnointijärjestys voi olla kokonaan toinen. (Kuutti 2003, 91.)

5.6 Visuaalinen tasapaino

Kuutti (2003, 97) toteaa, että visuaalinen tasapaino on tärkeää käyttöliittymän miellyttävyyden kannalta. Jos käyttöliittymän elementit ovat kovin epätasapainossa, tulee helposti mielikuva, että käyttöliittymä kaatuu johonkin suuntaan. Jotta käyttöliittymä olisi tasapainossa, pitäisi sen eri puolilla olevien elementtien painoarvojen summa olla suunnilleen sama. Painoarvoihin vaikuttavat elementtien koko, sijainti, väri ja liikesuunta. Suuremman kappaleen painoarvo on suurempi kuin

pienen. Painoarvoon vaikuttaa myös kohteen etäisyys keskipisteestä. Tumma kappale on painoarvoltaan vaaleaa suurempi. Eri ominaisuuksilla voidaan kompensoida toisia. Tumma, pienempi esine voi olla painoarvoltaan samanlainen kuin suurempi vaalea esine.

5.7 Prototyypit

Kuutin (2003, 104) mukaan prototyypin hyödyllisyys etenkin käyttöliittymän määrittelyssä on tunnustettu. Se kuuluu keskeisimpänä osana kaikkiin yleisimpiin käyttäjäkeskeisen suunnittelun menetelmiin sekä myös muihin käytettävyyden huomioon ottaviin tuotesuunnittelumenetelmiin. Se on suunnittelun apuväline ja prototyypin käyttämisellä on pitkät perinteet. 1980-luvulla perustuivat laajassa mittakaavassa käyttöön otetut suunnittelumenetelmät käytettävyyden arviointiin prototyypistä ja prototyypin iteratiiviseen kehittämiseen lopputuotteeksi. Se on myös erottamaton osa käyttäjäkeskeistä Contextual Design -suunnittelumenetelmää.

Prototyyppejä voi olla hyvin monen tasoisia. Niiden laajuus ja toteutustapa vaihtelevat hyvin paljon riippuen siitä, mihin niitä käytetään. Ne voivat yksinkertaisimmillaan olla paperille piirrettyjä malleja tai laajoja yksityiskohtaisesti tuotteen ominaisuuksia mallintava prototyyppi. Prototyyppi voi esimerkiksi sisältää tuotekonseptin avainominaisuudet, vuorovaikutukseen tarvittavat toiminnat, alustavan visuaalisen ilmeen sekä tyylin tai viitteellisen tietorakenteen ja -sisällön. (Adage Usability 2006.)

Kuutti (2003, 104) pohtii sitä, että joidenkin lähteiden mukaan prototyyppi ei sopisi kovin hyvin perinteisesti ohjelmistotuotannossa käytettyyn vesiputousmallin. Siinä edetään yleensä määrittely- ja suunnitteluvaiheiden kautta toteutukseen ja testaukseen. Mallin ongelmana pidetään välituotteiden huonoa testatavuutta. Käytännössä tämä tarkoittaa sitä, että käytettävyyttä päästään kokeilemaan käytännössä liian myöhään, jolloin suurten muutosten tekeminen on kallista ja usein erittäin työlästä. Pelkkä hyvä dokumentaatio ei kuitenkaan riitä hyvän käytettävyyden varmistamiseksi. Tuotesuunnittelussa on kuitenkin erityisen tärkeää, varsinkin uusien konseptien kanssa, prototyypin laatiminen tarpeeksi aikaisessa vaiheessa. Prototyypit voivat olla yksinkertaisia ja nopeasti laadittuja. Niiden avulla voidaan tutkia vaikka vain tuoteen yhtä osa-aluetta. Taulukossa 4. on esitelty eri prototyyppimalleja.

Taulukko 4. Erilaisia prototyyppimalleja ja niiden selitykset lyhyesti.

Prototyyppimalli	Selitys
Toiminnallinen prototyyppi	Pisimmälle viety muoto, kyseessä on periaatteessa toimiva sovellus.
Paperiprototyyppi	Paperinen käyttöliittymän kuva, jota voidaan testata loppukäyttäjällä.
Ozin velho	Tässä ihminen tekee asian, joka on teknisesti vaikea ja aikaa vievä tai jopa mahdoton toteuttaa.
Emulaatio	Tässä matkitaan jollakin laitteella jonkin muun laitteen toimintaa.
Simulaatio	Jäljitellään jotain oikeaa laitetta muunlaisella tekniikalla. Esimerkiksi lentokonesimulaattorit.
Käsikirjoitukset	Tuotteesta tehdään käsikirjoitus, hieman kuten elokuvastakin. Siinä tehdään jokin etukäteen nimetty tehtävä. Myös tehtävän lopputulos on ennalta suunniteltu.

Kun prototyyppi on tehty ja sitä on käytetty sen vaatimaan tarkoitukseen eli virheiden etsimiseen ja testaamiseen käyttäjäryhmällä, kannattaa heittää se pois. Lopullisen tuotteen tekeminen kannattaa aloittaa alusta asti ja tehdä se huolella. Toisaalta lopullisen tuotteen voi tehdä myös prototyypin pohjalta, mutta silloin lopullisessa tuotteessa voi ilmetä ongelmia, johtuen esimerkiksi huolimattomasta ohjelmoinnista. (Kuutti 2003, 114.)

5.8 Testaaminen

Käyttäjäkeskeisessä suunnittelussa testaaminen tehdään käyttäjätestinä. Siinä kohderyhmää parhaiten edustava koehenkilö tekee joukon etukäteen määritettyjä tehtäviä joko prototyypillä tai valmiilla sovelluksella. Testaajat ovat mukana testitilanteessa ja tekevät havaintoja käyttöliittymästä, käytettävyysongelmista sekä käytettävyyspuutteista. Käyttäjätestit liittyvät olennaisesti iteratiiviseen tuotekehitykseen. Perinteisesti iteratiivisessa tuotekehitysprosessissa käyttäjätestausta tehdään jo kehityksen alkuvaiheessa melko aikaisen vaiheen prototyypillä. Kuitenkin pitäisi myös testata valmista käyttöliittymää, varsinkin silloin kun ollaan kehittämässä uutta versiota käyttöliittymästä. Silloin uudelle käyttöliittymälle on olemassa selkeä mitattavissa oleva vertailukohta, vanha versio. Tällaisessa tapauksessa uuden version on selkeästi oltava parempi kuin vanhan. (Kuutti 2003, 68.)

Käyttäjätestiä pidetään Heuristisen arvioinnin kanssa tasavertaisena tuotteen testausmenetelmänä. Ne eivät korvaa toisiaan, koska niiden avulla voidaan paljastaa erityyppisiä käytettävyysongelmia. Yleensä kannattaa käyttää eri menetelmiä rinnakkain, jotta voitaisiin päästä parempiin lopputuloksiin, kuin jos käytettäisiin vain yhtä menetelmää. (Kuutti 2003, 69.)

Käyttäjätesteihin liittyy myös ongelmia, joista suurimpana pidetään testaustilanteen luonnottomuutta. Missään tilanteessa ei voida saavuttaa täysin luonnollisia olosuhteita sovelluksen käyttämiselle. Tarkkailtava tietää aina olevansa tarkkailtu ja jollakin tapaa se aina vaikuttaa tilanteeseen. Toinen yleinen ongelma on koehenkilöiden valinta. Tieto todellisista loppukäyttäjistä voi olla varsin puutteellinen. Monesti tieto sovelluksen lopullisesta käyttäjästä voi olla täysin intuition varassa. Käyttäjätestit voidaan jakaa kolmeen suurempaan vaiheeseen, joita ovat testin valmistelu, itse testi ja lopulta testissä saadun informaation purkaminen käytännön havainnoiksi. (Kuutti 2003, 70.)

6 CSS-EDITORI -SOVELLUKSIA

CSS-editorilla tarkoitetaan pääasiassa sellaista sovellusta, jota käytetään vain CSS-tiedostojen muokkaamiseen ja luomiseen. Monesti myös HMTL-editoreiden yhteydessä on varsin toimivia ratkaisuja. CSS-editorit eivät yleensä ole kovin suurikokoisia tiedostokooltaan, vain muutamia megatavuja, joten ne ovat täten kevyitä käyttää. CSS-editorisovelluksia on nykyään käytössä monentyyppisiä. Ne voivat olla joko tekstieditorien kaltaisia tai osittain graafisia sovelluksia.

Ammattimaiset suunnitelijat ja webkehittäjät pärjäävät mainiosti ilman minkäänlaisia apueditoreita tyyliohjeita luodessaan. He käyttävät usein vain yksinkertaista Notepad++:n kaltaisia tekstieditoreita. Yleensä ammattimaiset websuunnittelijat voivat käyttää niitä lähinnä helpottamaan ja tehostamaan työn kulkua. Toisesta näkökulmasta katsottuna editoreista voi olla aloittelijoille suurikin hyöty, jos CSS-kieli on heille vielä suhteellisen tuntematon käsite. (Smashing Magazine 2008.)

Ennen kuin muokattavaa web-dokumenttia päästään käsittelemään CSS-editorilla, kannattaa suunnitella dokumentin ulkoasu hyvin ja tehdä huolella siihen liittyvät valmistelut, esimerkiksi tarvittavien kuvien muokkaus kuvankäsittelyohjelmalla. Se tehostaa työn kulkua ja mahdollistaa ottamaan kaiken irti editorin hyödyistä. Ennakkoon suunnittelu on tietenkin makuasia. Toisissa editoreissa on panostettu enemmän sovelluksen graafisuuteen, mikä mahdollistaa sen, että dokumentin suunnittelun voi tehdä CSS-koodin kanssa samanaikaisesti. Toiset editorit on taas vain pyritty pitämään mahdollisimman yksinkertaisina ja kevyinä. Tämän vuoksi tekijän on jo valmiiksi tiedettävä mitä haluaa tehdä.

Yksinkertaisimmillaan CSS-editorit sisältävät vain perustekstinmuokkausominaisuudet värin ja fontin muutoksilla. Lisäksi voidaan lisätä joitain tiettyjä kuvien ominaisuuksia, kuten sijainti tai vaikkapa vain vaikutetaan sivun taustakuvaan. CSS:n avulla tämä onnistuu varsin yksinkertaisesti. Tässä kappaleessa esittelen muutamia osittain kaupallisia sekä ilmaisia CSS-editorisovelluksia. Kaikilla niillä on lähtökohtana muokata CSS-koodista validia, jotta se toimisi kaikilla selaimilla mahdollisimman virheettömästi, toisin sanoen näyttäisi samanlaiselta selaimesta riippumatta. Toisissa näistä editoreista virheenkäsittelyyn on panostettu enemmän, toisissa ei.

Tässä kappaleessa ei käsitellä lainkaan Adoben Dreamweaveria tai muita vastaavia sovelluksia, koska se on kokonaisvaltainen web-kehitystyökalu eikä rajoitu pelkästään CSS-editoriin. Kuitenkin on mainittava, että Dreamweaverista löytyy myös tehokas CSS-editori, jota moni ammattilainen käyttää. Lisäksi tässä kappaleessa pyritään esittelemään editoreita, joita on saatavilla eri käyttöjärjestelmille, kuten Windows, Machintosh ja Linux.

CSS-editorien suurin etu on niihin intergroiduista kehitysympäristöistä, joita ne tarjoavat webkehittäjille. CSS-editorin päätehtävä on yhdistää tehokkaasti CSS-tyyliohjeiden, HTML-lähdekoodin ja sivun layoutin editointiominaisuudet tiiviksi käyttöliittymäksi. Monesti CSS-editorit menevät vielä pidemmälle. Erillisenä osana koodaustoiminnoista hyvät CSS-editorit tarjoavat avustustyökaluja virheiden etsintään sekä testaukseen. Lisäksi ne antavat työkalut myös live-editointia, esikatselukuvia, edistynyttä koodin selaamista, koodin muotoilijoita sekä tiivistäjiä, validointia, sisäänrakennettuja CSS-referenssejä ja projektin hallintatyökaluja. Näitä toiminnallisuuksia yhdistelemällä saa käyttöönsä hyödyllisten työkalujen arsenaalin. (Smashing Magazine 2008.)

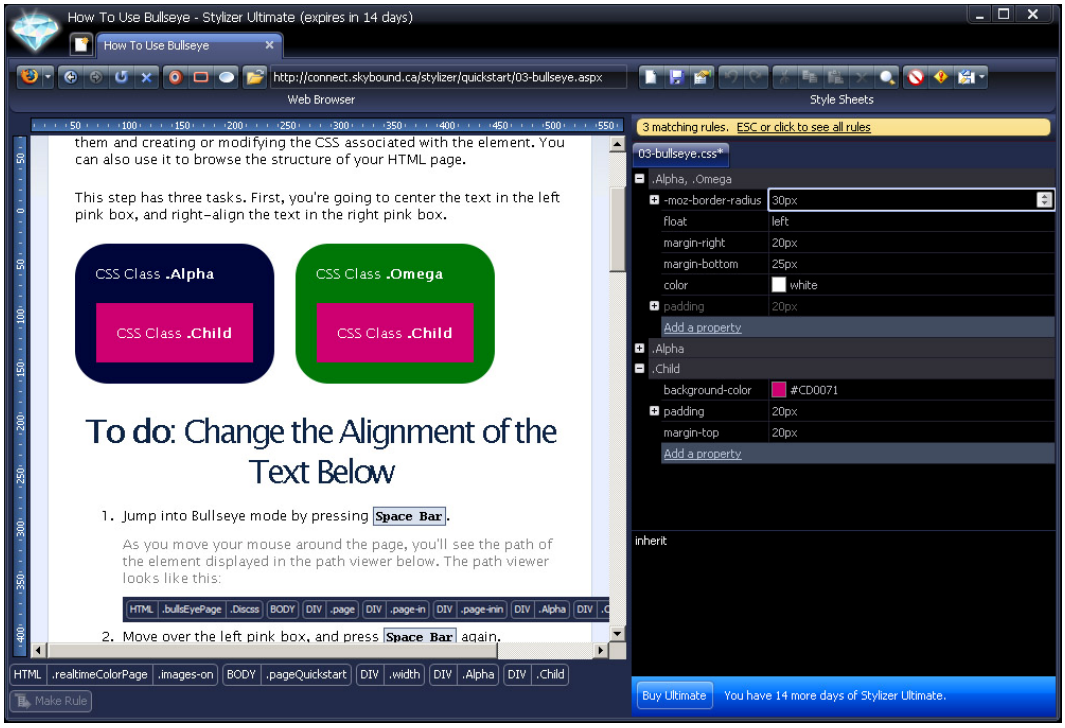
Joidenkin editorien avulla on mahdollista organisoida osia koodista kansioihin ja näin suodattaa tyyliohjeet helposti, mikä yksinkertaistaa hallintaa ja tekee lähdekoodista helpompaa lukea. Toinen käytännöllinen ominaisuus on välitön tyyliohjeen esikatselu eri selaimilla, kuten IE tai Firefox. Tätä ominaisuutta on vaikea löytää standardoiduista HTML-editoreista. Sen lisäksi CSS-editoreilla on helppo analysoida koodissa ilmeneviä ongelmia koodintarkastajalla ja hajoittaa tyyliohje käyttämällä niin sanottua ”X-ray”-ominaisuutta. (Smashing Magazine 2008.)

Lyhyesti sanottuna suurin hyöty CSS-editorista on se, että se tarjoaa integroidun kompaktin ympäristön CSS-suunnittelulle ja mahdollistaa nopean sekä tehokkaan työnkulun. Se, mikä on editorissa koottu yhteen käyttöliittymään, voisi muuten vaatia kymmeniä ylimääräisiä ohjelmia. (Smashing Magazine 2008.)

Stylizer

Stylizer, aiemmin tunnettu nimellä StyleSpread, on Skybound Softwaren kehittämä CSS-editori. Se on tällä hetkellä saatavissa ainoastaan Windows-käyttöjärjestelmälle, mutta siitä on kehitteillä Machintosh-käyttöjärjestelmän kanssa yhteensopiva versio. Stylizer voidaan käsittää puhtaasti visuaalisena editorina, joka keskittyy ainoastaan CSS:n muotoilemiseen. Se käyttää ”grid”-mallia

liittymässään, joka on erilainen kuin tekstieditoreissa. Tämän vuoksi CSS:ään on kehittäjän mukaan mahdotonta tulla virheitä. Editoriin on myös sulautettu Firefox sekä IE, joiden välillä käyttäjä voi vaihtaa sivu näkymää. Esikatselukuva on koko ajan näkyvissä, ja se toimii samalla sunnittelualustana. Lisäksi editorilla on mahdollista muokata muun muassa elementtien mittoja, värejä sekä reunuksia reaaliajassa. Stylizer on shareware-sovellus, jonka ilmaisversiossa on puolet vähemmän ominaisuuksia kuin maksullisessa versiossa. Tämä ei juuri kuitenkaan rajoita editorin tehokasta käyttöä. Stylizer on tiedostokooltaan suhteellisen pienikokoinen editori, minkä vuoksi se ei vaadi tietokoneelta suuria resursseja toimiakseen. Seuravassa kuvassa näkyy Stylizer-editorin käyttöliittymä. Muokattava sivu on editorin valmistajan tuottama demo-sivu.



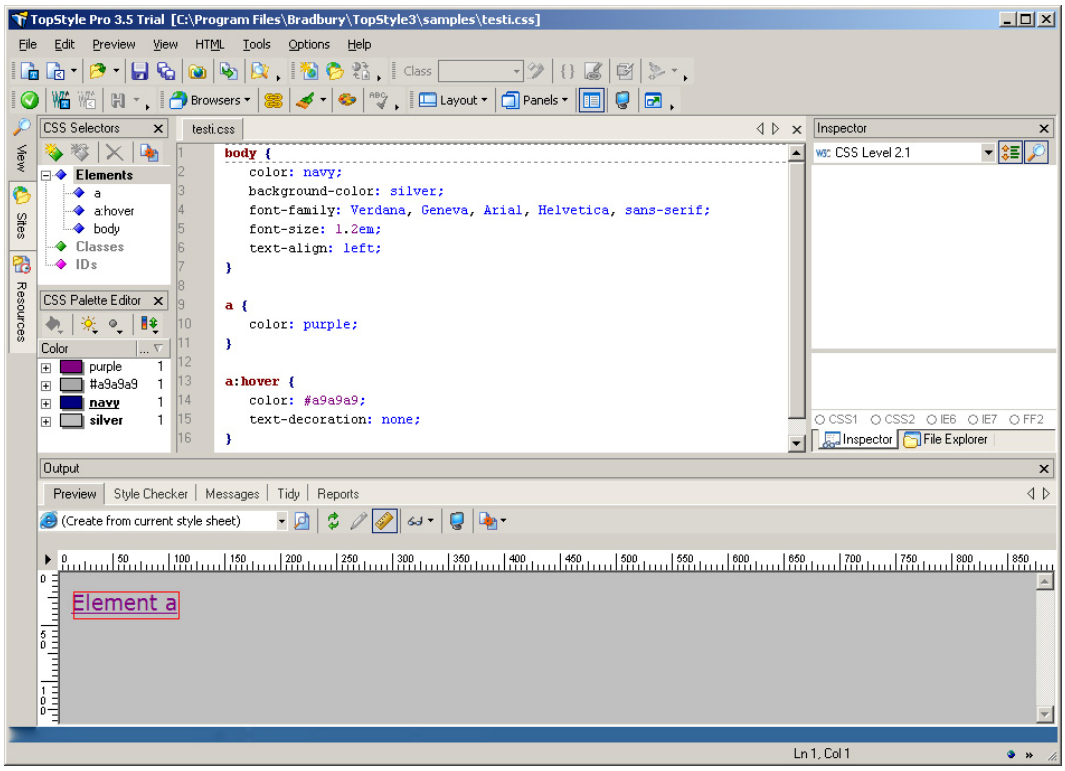
KUVA 3. Näkymä Stylizer-editorin käyttöliittymästä.

Stylizer tukee CSS2.1:n lisäksi myös joitakin CSS3:sta tuttuja ominaisuuksia, kuten esimerkiksi *border-radius*. Siihen voi tuoda valmiita CSS-tiedostoja ja sillä voi luoda täysin uuden tyyliohjeen. Stylizerin käyttöliittymä on selkeä, minkä ansiosta ohjelman tehokkaan käyttämisen luulisi oppivan nopeasti. Kuitenkin ominaisuuksia on niin paljon, että moni voi alussa turhautua niiden kaikkien opettelussa.

TopStyle

Tämä editori on varsin monipuolinen väline CSS-tiedostojen käsittelyyn. Editorista on olemassa kevytversio, joka on sopivampi aloittelijoille, ilman kehittyneempiä työkaluja. Editori on saatavissa ainoastaa Windows-alustalle. Täysversio sopii paremmin ammattimaiseen käyttöön lukuisine eri ominaisuuksineen. Näitä ominaisuuksia ei löydy muista editoreista. Esimerkiksi intergroidulla HTML-tidy:llä voi helposti konvertoida standardoimattoman HTML-koodin validiksi XHTML-koodiksi. Integroidulla ”Tyyli päivittäjällä” korvataan standardoimattomat tagit, kuten <font>-tagi korvaavalla validilla CSS-määrityksellä. Jaettu ikkuna mahdollistaa suoran selainten välisen vertailun websivusta Internet Explorerilla tai Firefoxilla. Doctype-määritystä voidaan myös vaihdella, jotta nähtäisiin, miten eri Doctype-deklaraatiot vaikuttavat sivun layouttiin eri selaimissa. (Smashing Magazine 2008.)

Ehkä tehokkain työkalu TopStyle-editorissa on niin sanottu ”tyylin tarkistin”, jonka avulla voidaan validoida tyyliohjeita ja tästä johtuen varmistaa ulkoasun oikea esitystapa eri selaimissa. Editorissa voidaan myös ennakoida virheet suosituimmissa selaimissa, jotka voivat ilmaantua validista CSS-koodista huolimatta. Lisäksi editorista voidaan edelleen lähettää W3C:n validointipalveluun, jotta ne ongelmat voitaisiin korjata, mitä editori ei ole löytänyt. Seuraavassa kuvassa TopStyle-editorin käyttöliittymä perustilassa. (Smashing Magazine 2008.)

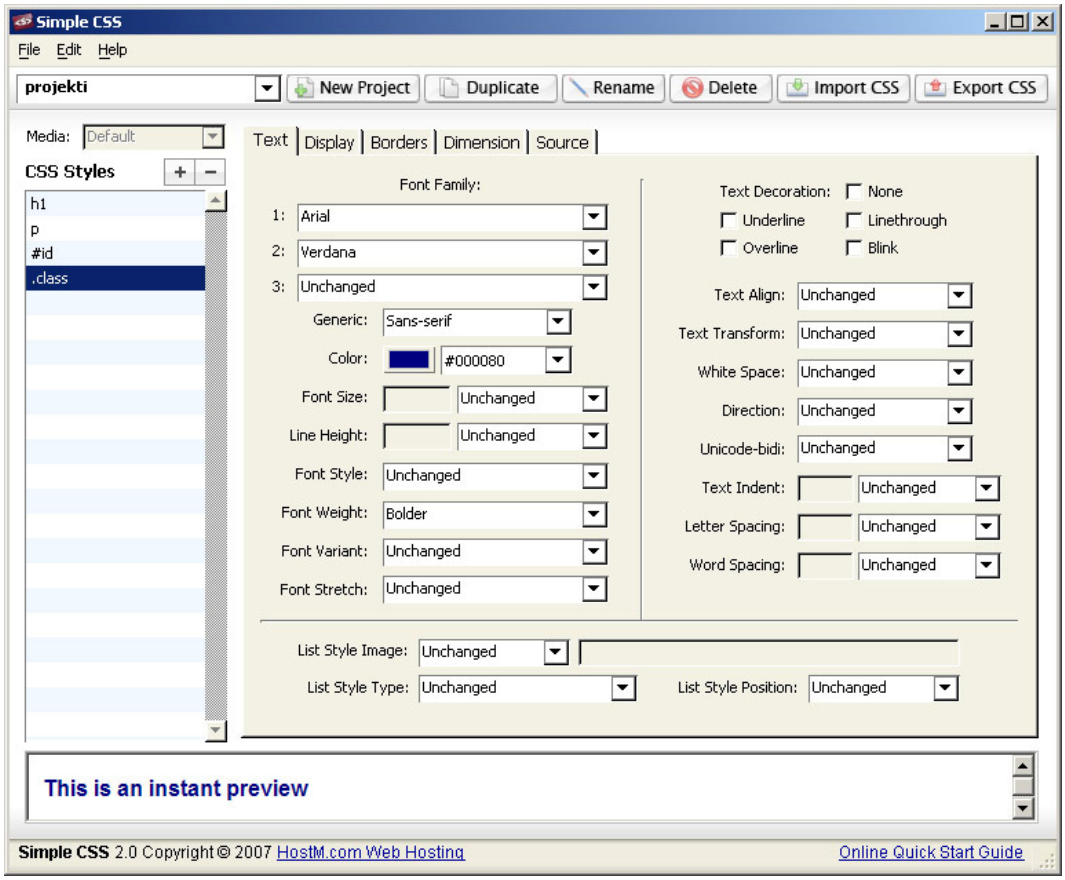


KUVA 4. TopStyle –editorin käyttöliittymä.

TopStyle-editorin käyttöliittymä on tekstieditorin kaltainen ja varsin karun näköinen. Siinä ei ole keskitytty visuaaliseen ilmeeseen, vaan käyttöliittymässä on esitetty kaikki tarpeelliset ominaisuudet. Koska ominaisuuksia on niin paljon, vaikuttaa se käytettävyyteen ja selkeyteen väistämättä. Editoria on helppo käyttää sellaisen käyttäjän, jolla on jo aiempi tuntemus vastaavista web-kehitykseen tarkoitetuista editoreista.

Simple CSS 2.0

Simple CSS 2.0 on yksinkertainen CSS2:een perustuva CSS-editori. Se on erittäin kevyt ja täysin ilmainen. Editoria voidaan käyttää niin Windows, Machintosh kuin myös Linux alustalla. Käyttöliittymä on yksinkertainen ja helppo omaksua. Se on toteutettu tuttujen perus käyttöliittymä -elementtien avulla, ilman turhia lisäominaisuuksia. Editoriin voi tuoda valmiita tyylitiedostoja tai voi luoda uuden tyylitiedoston. Seuraavassa kuvassa näkyy perusnäkymä käyttöliittymästä.



KUVA 5. Näkymä Simple CSS-editorin käyttöliittymästä.

Parhaiten Simple CSS sopii ehkä aloittelijalle tai niille, jotka haluavat vain saada nopeasti kasaan jotain yksinkertaisia tyyliohjeita CSS2:lla. Sovellus ei toiminnaltaan ole täysin virheetön. Joistakin valikoista ei valinta tule voimaan tai ne eivät ole käytettävissä lainkaan, vaikka niiden pitäisi olla. Lisäksi preview-ikkuna ei aina löytänyt tarvittavaa HTML-sivua.

Firebug ja Web Developer

Firebug ja Web Developer ovat Firefox-selaimen lisäosia, jotka asennetaan selaimeen itseensä. Ne ovat ilmaisia ja sangen tehokkaita. Molempia voi käyttää myös erikseen, mutta web-kehittäjien keskuudessa näiden yhteiskäyttöä on kehuttu kovasti. Vaikka nämä kaksi lisäosaa eivät suoraan ole pelkkiä CSS-editoreita, ovat ne silti tehokkaita sekä käytettyjä työkaluja CSS-tiedostojen käsittelyssä.

Firebugin CSS-välilehti kertoo kaiken oleellisen, mitä CSS:stä pitää tietää. Koodia muokatessa muutokset näkyvät selaimessa välittömästi. Firebugissa voi kytkeä tyylimääritykset päälle ja pois yhdellä klikkauksella. Visuaalisena apuna lisäosassa voidaan pitää mittaustyökalua, joka mittaa ja näyttää, mikseivät jotkin elementit ole sivulla linjassa keskenään.

Web Developer–työkalupalkki näkyy selaimen yläosassa. Se on saatavissa Firefoxin lisäksi myös Flock- sekä Seamonkey-selaimelle ja toimii kaikilla niillä alustoilla, joilla nämä selaimet toimivat. Eli näitä ovat Windows, Mac OS X ja Linux. Työkalupalkin lisäksi lisäosan mukana tulee valikko, jonka kautta pääsee käsiksi Web Developerin eri työkaluihin ja ominaisuuksiin. Osat niistä on samoja, mitä on nähtävissä jo heti työkalupalkissa. Laajennuksen avulla voidaan muun muassa selaimeen liittää käyttäjän oma tyyliohje, joka haluttaessa jyrää yli muut sivuille asetetut tyyliohjeet. Myös tämän lisäosan CSS-editointimahdollisuudet ovat varsin yksinkertaiset ja visuaaliset avut puuttuvat kokonaan. CSS-koodin muokkaamiseen siis vaaditaan CSS-kielen jonkinasteista tuntemusta. Laajennuksessa on kuitenkin hyvät informaation näyttämisominaisuudet, joiden avulla sivujen ulkoasua on helppo analysoida. Seuraavassa kuvassa näkyy Firebug- sekä Web Developer-lisäosan käyttöiittymät punaisella rajattuna. Firebug -lisäosa näkyy selaimen alaosassa ja Web Developer -lisäosa selaimen yläosassa.

KUVA 6. Firebug ja Web Developer -käyttöliittymät rajattu punaisella Firefox-selaimen yhteydessä

7 CASE: ROSENDAHL DIGITAL NETWORKS OY - CSS -TEEMAEDITORI

7.1 Tietoja yrityksestä

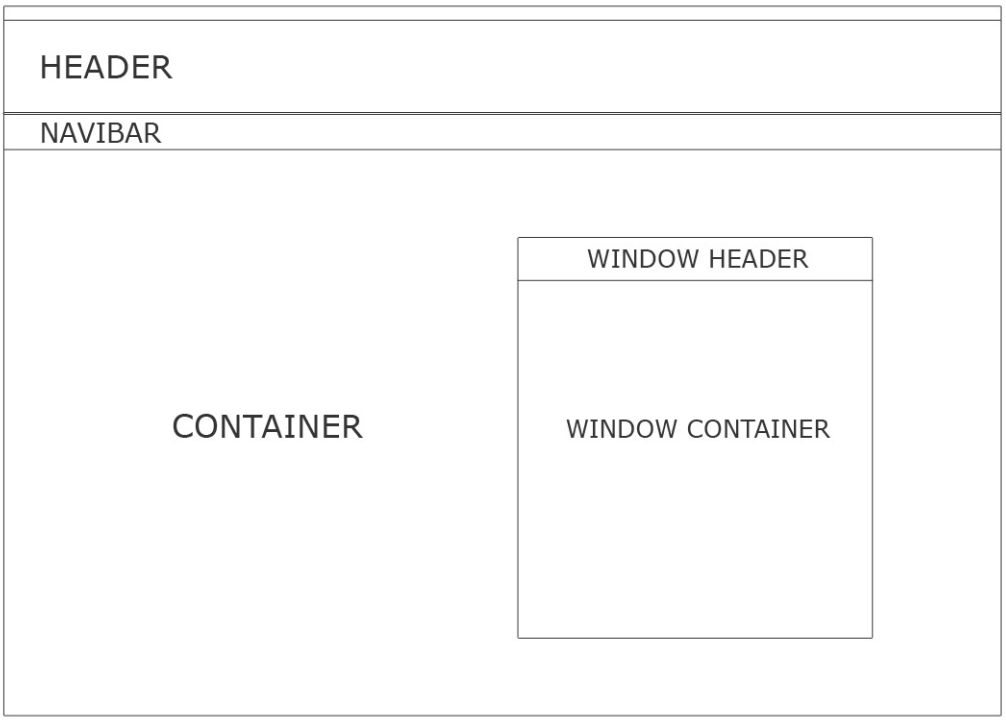
Rosendahl Digital Networks Oy on perustettu vuonna 2005. Perustajat, Markus ja Miikka Rosendahl työskentelivät aikaisemmin IT-järjestelmien kehittäjänä sekä markkinointikehittäjänä Naisten Pukutehdas Oy:ssa. RDN Oy on perustettu vuosien järjestelmäkehitystyön tuloksena kehittyneen, vahvan vaatetusalan ammattitaidon pohjalta. Yrityksen liiketoiminta on jaettu kahteen yksikköön: RDN Software ja RDN Media. Yritys on erikoistunut uuden sukupolven liiketoimintasovelluksiin sekä sähköiseen mediatuotantoon. Päätoimisesti RDN Oy kehittää ja soveltaa uusimpia teknologioita, kuten Web 2.0 ja RFID, muokattuna asiakkaidensa tarpeiden mukaan. (Rosendahl Digital Networks Oy 2008; Inkiläinen 2007, 33.)

7.2 Toimintaympäristö

7.2.1 Portaali

Teemaeditorin käyttöympäristönä toimii selainympäristössä toimiva portaaliratkaisu, joka on kehitetty Rosendahl Digital Networks Oy:n tuotantokäyttöön. Tämän portaaliympäristön on tarkoitus toimia itsenäisenä intranetsivustona sekä pohjana yrityksen Business Set- ja Customer Portal- järjestelmille. Portaaliratkaisu, jota käytetään RDN Oy:n tuotannossa ja jonka yhteyteen teemaeditoria kehitetään, muistuttaa rakenteeltaan iGoogle- ja pageflake.com-sivustoja. Käytännössä tämä tarkoittaa, että portaalien yhteyteen voidaan koota eri toimintoja sekä palveluita, joiden avulla voidaan muun muassa lukea sähköpostia, lukea yrityksen sisäisiä tiedotteita, hakea hakukoneilla, tehdä muistiinpanoja tai käyttää kalenteria. Tällainen toimintojen yhdistäminen yhden sivun alaisuuteen onnistuu nykyään helposti Ajax-tekniikan avulla. (Vallittu 2008, 39 – 40.)

Rosendahl Digital Networks Oy:llä (RDN) käytettävää portaaliratkaisua on tarkoitettu käytettäväksi usean eri käyttäjän kesken siten, että jokainen käyttäjä voi muokata oman portaalinäkymänsä haluamakseen. Käyttäjän asetukset tallennetaan ja järjetelmä muistaa ne seuraavalla kerralla, vaikka käyttäjä kirjautuisi järjestelmään eri päätteeltä. Portaaliratkaisuissa käytettäviä ikkunoita voidaan liikuttaa ja niiden kokoa voidaan muuttaa vapaasti. Portaaliratkaisu (KUVA 7.) muistuttaakin näin hyvin paljon työpöytäsovelluksista tuttua ikkunoitua käyttöliittymää. (Vallittu 2008, 39 – 40.)



KUVA 7. Rautalankakuva alustana toimivasta portaalista.

RDN Business Set- sekä RDN Customer Portal-järjestelmät koostuvat eri moduuleista, joita käytetään portaalin ikkunoiden kautta. Ikkunat voidaan asetella halutulla tavalla niille varatulle, niin sanotulle työpöytä-alueelle. Portaalien ulkoasua voidaan muokata muun muassa taustakuvan ja ikkunakomponenttien osalta. Portaalien hallintaa on tehostettu Ajax-tekniikan avulla siten, että konfigurointi tapahtuu samalla sivulla sovelluksen kanssa. Käyttäjän ei tarvitse navigoida erilliselle uudelle konfigurointisivulle, vaan se voi tehdä haluamansa asetukset samalla sivulla. (Vallittu 2008, 40.)

Alun perin portaalin yhteydessä oli mahdollista vaihtaa eri ulkoasuteemojen välillä. Ne oli valmiiksi suunniteltu ja lisätty portaalin lähdekoodin yhteyteen. Vaihdettava ulkoasuteema vaikuttaa muun muassa portaalin värimaailmaan, taustakuviin sekä ikkunakomponentin ulkoasuun.

Portaalin konfigurointi on ratkaistu erilaisten editorien avulla, ja näin erillinen teemojen vaihtamiseen ja niiden muokkaamiseen keskittyvä editori oli luonteva ratkaisu. Sitä voidaan pitää kehittyneempänä versiona alkuperäisestä portaalin yhteyteen kehitetystä ratkaisusta, jossa halutut tyylit jouduttiin tuottamaan erikseen kolmansien osapuolten sovelluksilla ja lisäämään portaalin yhteyteen manuaalisesti.

Business Set- ja Customer Portal- järjestelmiä suositellaan käytettävän Windows Xp-käyttöjärjestelmässä ja käytettäväksi selaimeksi suositellaan Mozilla Firefox 1.5 tai uudempaa. Näiden vuoksi voitaisiin myös teemaeditorissa ottaa keskitetysti huomioon vain Firefox-selaimen tukemat ominaisuudet. Pääasiassa pitäydytään kuitenkin mahdollisimman valideissa määrityksissä niin ulkoasuteemojen CSS-tiedostojen suhteen kuin myös itse editorin lähdekoodin suhteen. Tällä tavalla otetaan huomioon myös ne käyttäjät, jotka jostain syystä käyttävät jotain muita selaimia.

7.2.2 Lähtökohdat CSS-teemoille ja kansiorakenteelle

Jokaisella teemalla on oma kansionsa, mihin portaalista linkitettään. Teemakansio sisältää portaalin teemaan liittyvän CSS-tiedoston ja kuvat omassa alakansiossaan. Ikkunakomponenttiin liittyvät CSS-tiedostot, ja kuvat ovat erikseen omissa kansioissaan portaalin GUI-kansion alaisuudessa. CSS-luokat, joita teemojen yhteydessä käytetään, on nimetty niiden tarkoituksen mukaan eikä sen mukaan, miltä ne näyttävät. Kaikkia teemoihin liittyviä CSS-tiedostoja saa muuttaa, poistaa, ja niihin voi tehdä uusia luokkia siinä määrin mitä tarve vaatii. Ainoa vaatimus teemojen CSS-tiedostoille on, että jos yhteen CSS-tiedostoon lisätään uusi luokka tai luokan nimeä muutetaan, on sama muutos tehtävä jokaiseen vastaavaan CSS-tiedostoon. Syy tähän on teemojen suunnittelun rakenne: CSS-tiedostot on suunniteltu siten, että tiedostoon, joka määrittelee portaalin sisältöalueen, ikkunan tai käytettävän modulin lähdekoodiin, ei tarvitse tehdä muutoksia. Vain ne rivit, mitkä määrittelevät käytettävän CSS-tiedoston, tarvitsee vain muuttua. Kaikki CSS-tiedostot sisältävät samalla tavalla nimetyt luokat.

Teemat, joita käytetään portaalin, ikkunakomponentin ja moduulisovellusten yhteydessä, perustuvat CSS-luokkien käyttöön. Jokainen teema ja sen CSS-tiedosto sisältävät kaikkien kolmen osan muotoilumääritykset. Jokaisella teemalla on vain yksi CSS-tiedosto. Nämä tiedostot on nimetty teeman mukaan, ja CSS-luokat on nimetty niiden käyttötarkoituksen mukaan. Esimerkiksi luokka `name.content_tbl` sisältää modulisovelluksessa käytettävän taulukon ulkoasumäärityksiä.

Alun perin teemoissa käytettyjä nappien HTML-elementtien ulkoasuun ei ollut tarkoitus vaikuttaa, vaan niiden oli tarkoitus pysyä samanlaisina teeman vaihtuessa. Selainkohtaiset erot pyrittiin myös ottamaan huomioon teemojen testausvaiheessa. Testattavia selaimia olivat Mozilla Firefox 1.5 ja uudemmat sekä Internet Explorer 6 ja sitä uudemmat. Teemoissa käytettävät kuvat ovat PNG-tiedostoja. Koska IE 6 ei tue PNG:n läpinäkyvyysominaisuutta, käytetään tätä varten toista CSS-tiedostoa, jota käytetään vain ainoastaan IE:n selainokohtaista *alphaImageLoader* suodinta, joka lataa alfa-kanavan. Jotta IE 6:n käyttäjillä toimisi PNG-läpinäkyvyys, käyttäjän on sallittava selaimsessaan aktiivisen sisällön näyttämisen.

Alun perin luokat oli jaettu portaalissa käytettävien luokkien, ikkunakomponentissä käytettävien luokkien kesken sekä ikkunakomponentin sisältöosan kesken. Portaalin luokat määrittelivät portaalin työpöytäalueen, header-osiossa olevan logon, header osion taustan ja navigointipalkin ulkoasut. Erikseen olevat ikkunakomponentin luokissa määriteltiin sen kehysten, header-palkin ja header-palkissa olevien nappien ulkoasu. Lisäksi oli omat luokat ikkunakomponentin sisältöalueelle, joilla päästiin määrittelemään muun muassa sisältöalueella olevien taulukoiden ulkoasua.

7.2.3 Teemat

Portaalin teemat määritellään pääsääntöisesti CSS-tiedostojen avulla. Teeman määrittelyyn tarvitaan kolmea eri CSS-tiedostoa. Portaalin teema vaatii oman tiedoston, kuten myös ikkunakomponentti, ja lisäksi ikkunoiden sisälle avattavalle sisällölle tulee tehdä omat CSS-määrityksensä. Noiden kolmen tyylitiedoston avulla määritellään portaaliratkaisulle yhtenäinen ulkoasu eli teema. Koko portaaliratkaisun teeman, johon sisältyy myös ikkunakomponentti ja sen sisältö, voidaan vaihtaa JavaScriptin avulla. Tämä vaatii kuitenkin tarvittavien CSS-tiedostojen

olemassa olon. Portaalin ja ikkunakomponentin vaihtamiseen käytetään omia erillisiä funktioita. Käytännössä funktio vaihtaa CSS-tiedostot ja -luokat toisiksi. Tämä tehdään dynaamisesti siten, että annetaan linkitetylle CSS-tiedostolle oma uniikki ID, jolla siihen voidaan viitata JavaScriptin kautta. Tämän jälkeen käydään muuttamassa *link*-elementin *href*-atribuuttia uuden teeman mukaisesti. (Vallittu 2008, 48.)

Ikkunakomponentin tyylin määrittelemiselle on oma mekanisminsa. Niille voi luoda uuden teeman tekemällä ikkunan muodostukseen tarvittavat yhdeksän pientä kuvaa asettamalla kuvat omaan kansioonsa ja tekemällä ikkunalle oman tyylitiedostonsa. Periaatteessa jokaisella ikkunalla voi olla oma tyylinsä. (Vallittu 2008, 48.)

Ikkunoiden sisältöä voidaan ladata kahdella eri tavalla, ja tämä vaikuttaa siihen, miten sisältöä käsitellään. Sisällön voi ladata joko Ajax-tekniikalla tai dynaamisesti JavaScriptin avulla. Yhteenvetona, teeman määrittelemiseen tarvitaan portaalin tyylit esimerkiksi theme_test.css, manage_organizations_test.css (ikkunoiden sisällön tyylit) ja test.css (ikkunan tyylit). Lisäksi jokaiselle ikkunalle on oma kuvakansionsa *test*, missä ikkunassa käytettävät kuvat ovat. (Vallittu 2008, 48.)

7.3 Vaatimusmäärittely

Teemaeditorin tulisi olla toimivat ja mahdollisesti ainoa vaihtoehto teemojen luomiseen. Editorilla rajoitettaisiin virheellisten määritysten tekeminen teemoihin. Esimerkiksi teemaeditori ei anna tehdä sellaisia CSS-määrityksiä tai mitään muutakaan määritystä, mitä ei tulkita ”teema” käsitteen alle. Ikonien käytöstä teemoissa päätettiin pääasiassa luopua, sillä niiden merkitys ei välttämättä selittyisi kaikille samalla tavalla. Ellei ikoneiden yhteydessä olisi myös tekstiä, ne voisivat toimia paremmin, mutta se veisi jo liikaa tilaa. Lisäideana päätettiin, että nappien ulkoasu laajennettaisiin osaksi teemaa, eli portaalin käyttöliittymän painikkeet muuttuvat myös teeman mukana.

Alun perin suunniteltu työjärjestys olisi pääpiirteittäin seuraavanlainen:

- Järjestää alkuperäisten teematiedostojen uudelleen.
- Ajaa nykyiset teemat tai CSS-tiedostot tietokantaan.
- Kehittää dynaaminen teemaeditori.

Teemaeditori kehitetään aluksi vain Business Setin ja Customer Portalin yhteyteen, mutta jatkokehitysideana olisi, että sillä voitaisiin vaikuttaa muihinkin tuotteisiin. Kaikkien RDN Oy:n tuotteiden hallinnointi tapahtuu kuitenkin pääasiassa Business Setin kautta, joten olisi järkevää, jos sitä kautta teemoihin tehdyt muutokset voitaisiin muuttaa muun muassa RDN Vendor -sovellukseen. Toisin sanoen kaikkien ohjelmistojen ilmettä olisi helpompi yhtenäistää.

Teemaeditorin yhtenä vaatimuksena oli se, että sitä voivat käyttää lähes kaiken tasoiset tietokoneen käyttäjät. Teemaeditori lisäosaa ei ole pelkästään tarkoitettu sellaisille käyttäjille, jotka hallitsevat HTML-kielen, JavaScriptin tai CSS:n. Riittäisi, että käyttäjä osaisi käyttää tekstinkäsittelyohjelmia. Vaikka teemaeditori sovelluksen kautta lisätään myös kuvia, on niiden lisäämisestä pyritty tekemään mahdollisimman yksinkertaista.

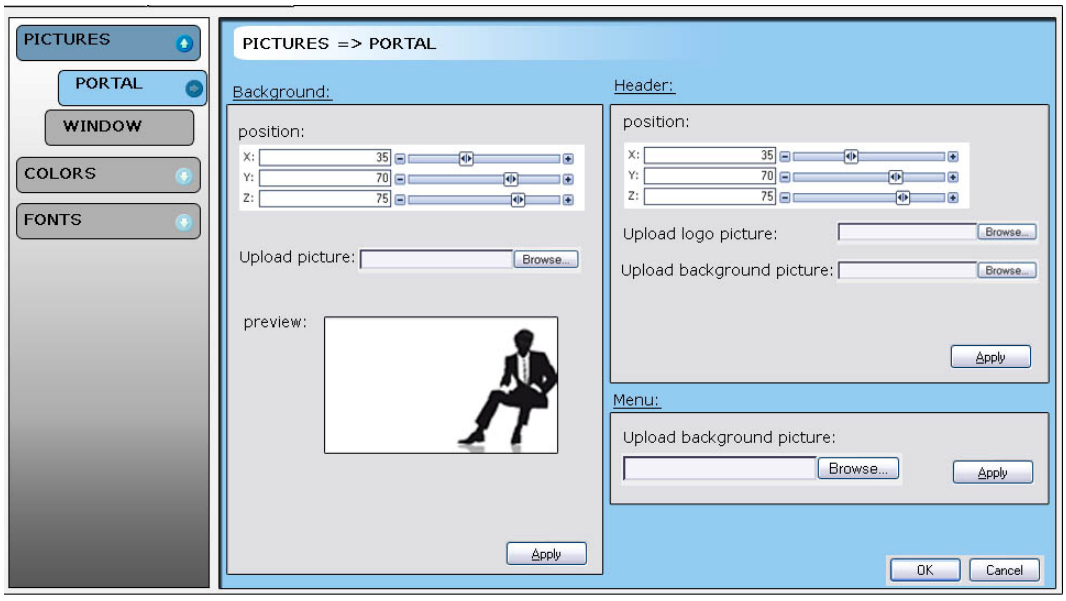
Teemaeditorilla tuotettava CSS-määritykset perustuvat CSS2.1-suositukseen. Tarkoitus on pysyä noin vuosi kehityksen jäljessä, jotta selaintuki olisi CSS-määrityksille mahdollisimman laaja. Tarkoitus on pyrkiä hyödyntämään monipuolisesti CSS2.1:n eri ominaisuuksia. Yritetään myös selvittää, millä tavoin CSS3:n uusia ominaisuuksia voitaisiin hyödyntää teemaeditorissa, ja millä tavoin ne voitaisiin lisätä teemaeditorin kautta teematiedostojen yhteyteen.

Tarkoitus olisi myös hyödyntää eri mediatyypeille määriteltäviä ominaisuuksia teemoissa, esimerkiksi miten screen:llä näkyvä layout tulostuu paperille tai miltä se näyttää PDA-laitteessa ja miten mediatyypit otetaan huomioon vai otetaanko lainkaan.

Käyttöliittymä

Lopullinen editori on enemmän graafisen käyttöliittymän kuin tekstieditorin kaltainen. Ensimmäinen prototyyppi käyttöliittymästä on mahdollisesti tekstipohjainen ja siinä voidaan säätää CSS-arvoja.

Käyttäjäryhmän tarpeita pidettiin lähtökohtana kun alettiin miettiä käyttöliittymän ominaisuuksia. Aluksi määriteltiin ketkä editoria tulisivat käyttämään, olisivatko niitä media-alan ihmiset, grfaafikot vai kuka tahansa? Millainen tietämys pitäisi olla CSS-tyyliohjeista vai riittäisikö yleinen tietokoneen tuntemus? Millä tavoin eri säätömahdollisuudet tulisi esittää? Onko käyttöliittymässä valmiita valikoita, valitsimia ja säätimiä? Käyttöliittymän välttämättömiä ominaisuuksia miettiessä pitäisi pohtia, millä tavoin editori tulisi parhaiten palvelemaan käyttäjäryhmää. Suunnitteluvaiheessa tehtiin luonnoksia mahdollisesta käyttöliittymän rakenteesta (KUVA 8.).



KUVA 8 Luonnos teemaeditorin käyttöliittymästä

Oli pohdittava myös sitä, mitä valmiita Ajax-tekniikkaa käyttäviä kirjastoja on jo olemassa, joita voitaisiin käyttää käyttöliittymän rakentamiseen. Ajax mahdollistaa sivujen osittain lataamisen siten, ettei koko sivua tarvi ladata uudelleen, eli saman tien voi nähdä jo itse sovelluksessa tekemänsä muutokset, eikä tarvita erillistä esikatselukuvaa. Valmis kirjasto nopeuttaisi työn kulkua huomattavasti, kos-

ka käyttöliittymään käytettäviä komponentteja ei tarvitsisi alkaa kehitellä ja testata aivan alusta.

Editoria tultaisiin käyttämään varsinaisissa yrityksen tuotteissa. Editori kehitettäsiin, jotta päästäisiin manuaalisesti luotavista CSS-tiedostoista ja teemojen suunnittelu voitaisiin osittain tehdä itse editorissa. Editorin tuottamista CSS-tiedostoista pitäisi saada myös monipuolisesti kustomoitavia ja asiakkailla pitäisi olla paremmat mahdollisuudet tehdä omannäköisiään teemoja. Tällä hetkellä teemaeditori tehdään portaalin yhteyteen, minkä kautta hallinnoidaan erilaisia moduleita. Editorin myöhempää kehitystä ajatellen pohdittiin myös sitä, että teemaeditorista voisi kehittää itsenäinen ohjelmisto.

Tarkoitus oli myös tehdä lisää valmiita ulkoasuteemoja, joiden on toimivat tietokannnasta käsin, ja teemoja pitäisi myös pystyä vaihtamaan editorissa lennosta. Tämän lisäksi olisi mahdollisuus kustomoida jo olemassa olevia teemoja tai jopa luoda alusta asti oma teema. Pitäisi olla myös mahdollisuus lisätä custom-scriptiä.

Tekniseen toteutukseen kuuluvaa

Editorin on lähtökohtaisesti toimittava webympäristössä. Valmiit CSS-tiedostot oli saatava siirrettyä tietokantaan, josta ne voidaan hakea editorin avulla. Seuraavassa vaiheessa tehdään graafinen käyttöliittymä, joka hyödyntää Ajax-tekniikkaa. Käytännössä editori toimii siten, että editorista lähtetetään PHP-kyselyt tietokantaan, jonne CSS-määritykset on ensin talletettu. PHP-skripti palauttaa kyselyt tietokannasta, joiden avulla muodostetaan tarvittava CSS-tiedosto. Lopputuloksena on ihan tavallinen CSS-tiedosto. Editorin kautta on myös mahdollisuus lisätä kustomoitua skriptiä, joka mahdollistaa esim. myöhemmin CSS3:n ominaisuuksien tuomisen selainten tukiessa sitä tarpeeksi.

Käytettävyys ja käyttäjät

Pääasiallisena käyttäjäryhmänä on yritysten graafisesta ulkoasusta vastaavat henkilöt. Näiden henkilöiden tulee pääsääntöisesti ymmärtää jotain webgrafiikasta ja sen suunnittelusta. Näitä henkilöitä voivat olla muun muassa graafiset suunnittelijat, websuunnittelijat, media-assistentit tai jollain tavalla aiheeseen enemmän perehtyneet henkilöt. Teemoja ei pääse luomaan aivan kuka tahansa, mutta teemaeditoria kehitetään kuitenkin siitä lähtökohdasta, että sen peruskäyttämiseen ei tarvita aiempaa CSS-kielen tuntemusta.

Teemaeditorin testaus

Alun perin teemaeditoria oli tarkoitus testata vain RDN Oy:n media-osastolla. Testausta päätettiin laajentaa koko yritykseen, jotta saataisiin selville useita eri käyttäjäkokemuksia muun muassa eri tasoisilta sekä taustaisilta henkilöiltä.

7.4 Työn kulku

Syksy/Talvi 2007

Ensimmäiset varsinaiset palaverit, jotka koskivat teemaeditoria, käytiin kesän 2007 loppupuolella. Lähtökohdat teemaeditorille olivat silloisen toimintamallin tehottomuus. Portaaliympäristö oli jo osittain käytössä asiakkailla ja niissä oli yksi perusulkoasu, joka oli erittäin yksinkertainen. Asiakkailta oli ilmeisesti tullut toivomuksena, että he voisivat saada oman yrityksen muuhun ulkonäköön yhteensopivan teeman omaan portaalisovellukseen. Kesän 2007 aikana erilaisia teemoja portaaliympäristöön alettiin ideoida ja myös toteuttamaan. Osittain teemojen tarkoitus oli olla visuaalisesti miellyttävämmän näköisiä, mutta toisaalta myös niiden oli tarkoitus vaikutta myönteisesti myös sovelluksen käytettävyyteen. Teemojen kehityksessä oli toki myös rajoituksia. Käyttöliittymän layout oli jo valmiiksi päätetty, joten teemoilla voitiin vaikuttaa ainoastaan fonttien asetuksiin sekä taustakuvien asetuksiin.

Aluksi oli tarkoitus järjestää vanhalla tavalla toteutettu CSS-teemajärjestelmä uudella tavalla, jotta uuden dynaamisen teemaeditorin kehittäminen saataisiin selkeämälle pohjalle. Entisten CSS-tyylien kansiorakennetta selkeytettiin ja tyylitiedostoja yhtenäistettiin tuomalla muun muassa eri tiedostoissa olleet tyylimäärittelyt samaan tyylitiedostoon. Käyttöliittymän koodia ei pääosin muutettu millään tavalla lukuun ottamatta muutamaa pientä korjausta, jonka tarkoitus oli parantaa CSS-määritysten yhteensopivuutta.

Kevät 2008

Teemaeditori-projektin eteneminen oli keväällä 2008 hidasta, koska projektille ei ollut tehty vielä varsinaisia virallisia alkumäärittelyjä. Tässä vaiheessa projektille ei ollut varmistunut teknistä toteuttajaa ohjelmoijien muiden työkiireiden vuoksi. Loppukeväästä saimme aikaan sopimuksen teemaeditori-projektin käynnistämisestä.

Kesä 2008

Kesän 2008 aikana projektin varsinainen toteutus lähti käyntiin. Kesäkuun alussa pidettiin aika tiiviseen tahtiin muutamia palavereita, joissa päätettiin pääosin teemaeditorin alkumäärittelyistä, projektin aikataulusta sekä projektin osallistujista. Kävimme myös läpi teemojen sen hetkisen tilanteen, eli millä tavalla ne oli toteutettu ja mitä muutoksia niihin oli saatava. Aikaa teemaeditorin ensimmäisen version toteuttamiselle oli kolme kuukautta. Minun lisäkseni projektissa oli mukana sovelluskehittäjä sekä projektin valvoja. Kesän lopussa projektin tuloksena piti olla ensimmäinen toimiva prototyyppi teemaeditorista.

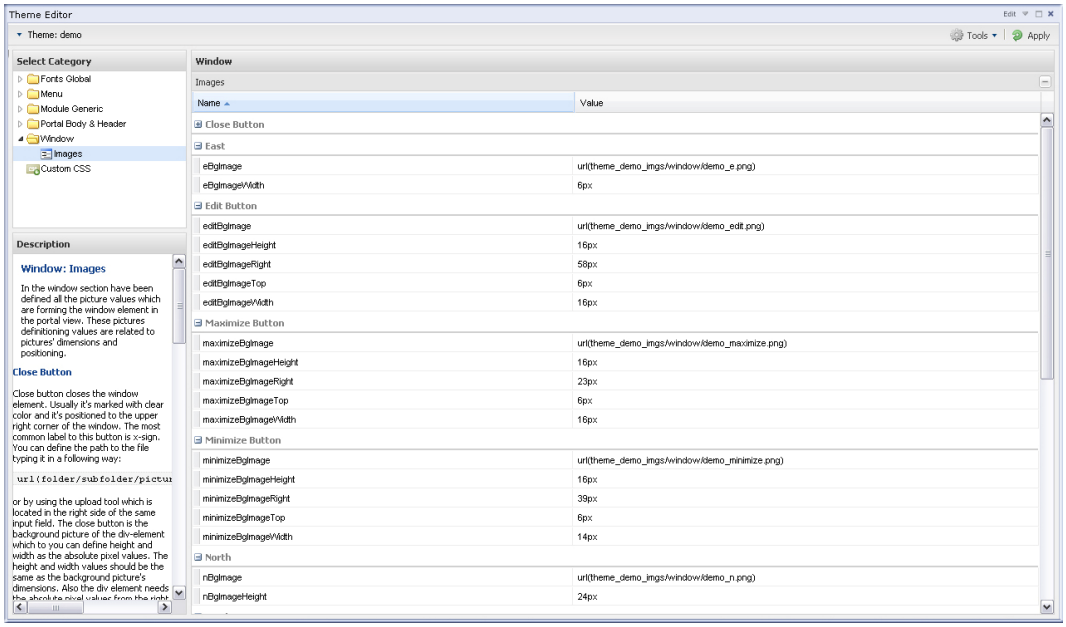
Projektin osalta sain hieman lisävastuuta. Sain osittain vastuulleni projektin koordinoinnin, ja tämän vuoksi jouduin perehtymään ohjelmistoprojektin suunnitteluun sekä tarkemmin sen osa-alueisiin. Projektin etenemiseen vaikutti osaltaan myös lomakausi, jonka vuoksi sovelluskehittäjä oli kolmasosan projektin ajasta lomalla. Resurssipulasta johtuen editorista ei saatu kehitettyä niin monipuolista kuin alussa oli määritelty. Kesän aikana tehtäviini kuului vaatimusmäärittelyjen tekeminen, ohjeistuksen tekeminen editorin käytöstä sekä editorin graafisesta käyttöliittymästä. Loppukesästä oli editoria tarkoitus myös testata, minkä suunnitelu sekä koordinointi oli myös vastuullani. Loppujen lopuksi testausta ei ehditty toteuttaa aiotussa laajuudessa.

Alkukesästä teimme sovelluskehittäjän kanssa alkumäärittelyjä editorille. Määrittelyt oli oltava valmiina kesäkuun puoleenväliin mennessä. Näiden määrittelyjen pohjalta sovelluskehittäjä alkoi suunnitella varsinaista editorin ohjelmistorakennetta, ja minä keskityin enemmän graafisen ulkoasun suunnitteluun.

7.5 Tekninen toteutus

Teknisestä toteutuksesta vastasi yrityksen ohjelmistokehittäjä, joka oli vastannut käyttöympäristönä toimivan selainpohjaisen käyttöliittymän kehittämisestä. Teknisessä toteutuksessa jouduttiin turvautumaan valmiiseen Ext JS 2.0 JavaScript -kirjastoon, jotta olisimme voineet saada aikaan tarvittavanlainen käyttöliittymä määrätyssä ajassa. Lisäksi valmiin kirjaston käyttäminen säästi eri selainten yhteensopivuuksien testaamiselta ja muidenkin ominaisuuksien aikaa vievältä testaamiselta.

Ext JS 2.0 JavaScript-kirjasto pohjautuu Yahoon YUI JavaScript-kirjastoon, jonka komponentteja on käytetty myös käyttöympäristönä toimivassa portaaliratkaisussa. Ext JS 2.0-kirjaston avulla voidaan rakentaa interaktiivisia websovelluksia, käyttäen sellaisia tekniikoita, kuten Ajax, DHTML ja DOM.



KUVA 9. Yksi näkymä editorin käyttöliittymästä.

7.6 Casen arviointi

Case-osuudessa haastavinta oli käytettävä aikataulu ja siinä pysyminen. Aikaa teemaeditorin toteuttamiselle järjestäytyi kesällä 2008 lopulta kokonaiset kolme kuukautta. Mikäli ohjelmistoprojekti saataisiin täydellisesti onnistumaan tässä ajassa, vaatii se tietyn määrän resursseja. Aikaresurssi oli jo rajattu, siinä ei voitu juurikaan joustaa. Seuraavaksi tulivat ihmiset.

Minun lisäkseni projektiin mukaan oli varsinaisesti varattu yksi ohjelmoija, joka vastasi teknisestä toteutuksesta, ja hänen työaikansa. Kuitenkin todettiin, jotta saataisiin määritelty ohjelmistopalikka määrätyssä ajassa valmiiksi, pitäisi jossain vaiheissa oikoa.

Ratkaisut, joihin päädyttiin projektin aikana, johtuivat pääosin aikataulun tiukkuudesta ja muista lisänä tulleista työtehtävistä. Muita työtehtäviä ei ollut otettu tarpeeksi huomioon projektin aikataulua laatiessa. Näin ollen jouduttiin joustamaan jonkin verran teemaeditorin ominaisuuksista sekä testauksesta. Projektin etenemiseen vaikutti myös uudet aihealueet projektisuunnittelun puolella, joiden tutustumiseen meni oma aikansa.

Kesän jälkeen lopputuloksena saatu teemaeditorin prototyyppi, saatiin suhteellisen hyvään vaiheeseen, kun otetaan huomioon käytössä ollut aika sekä muut resurssit. Editorin kanssa ei päästy vielä niin pitkälle, mitä alun perin oli suunniteltu, mutta se tarjoaa hyvän pohjan editorin tarvittavien ominaisuuksien testaamiseen mahdollisia uusia versioita kehitettäessä. Aluksi teemaeditorin toiminta portaaliympäristössä ei ollut vakaata, mutta loppujen lopuksi se saatiin osittain toimimaan halutulla tavalla.

Case-osuudessa toteutetun teemeditori-projektin aikana tuli arvioitua ajankäyttöä väärin, mikä olisi ensiarvoisen tärkeää ottaa huomioon, kun tehdään kehitystyötä tiiviillä aikataululla. Silloin pitää miettiä hyvin tarkkaan eri resusrssien suhteet projektissa. Kriittiseksi tekijäksi opinnäytetyöhön liittyvän projektin osalta tulivat myös muut projektiin liittymättömät tehtävät.

8 YHTEENVETO

Verkkotekniikoiden kehitys menee vauhdikkaasti eteenpäin. Webympäristöstä lähennytään jatkuvasti entistä enemmän työpöytäsovelluksien kaltaisia ratkaisuja, ja internet onkin nykypäivänä näkyvä osa meidän kaikkien elämää. Websivujen staattisuuden hävitessä dynaamisuus ja vuorovaikutteisuus korostuvat entisestään. Web-sivut ovat tarkkaan suunniteltuja interaktiivisia kokonaisuuksia, jotka kokoavat tekstiä, kuvaa, ääntä sekä videokuvaa yhdeksi visuaaliseksi esitykseksi. Kaiken näiden erilaisten visuaalisten ärsykkeiden keskellä voi tuntua hankalalta saada se kaikki miellyttävään esitysmuotoon ja vieläpä sellaiseen, että sitä voitaisiin seurata useasta eri mediasta. Tämä visuaalisuuden esittäminen on onneksi saatu ratkaistua CSS-kielen avulla, joka on kehitetty vastaamaan juuri erilaisten esitystapojen tarpeisiin esittää websivuja tai -sovelluksia.

Työssä pyrittiin selvittämään mitä ominaisuuksia ja tekniikoita websovelluksen toteuttamiseen tarvitaan. Näitä tietoja pyrittiin soveltamaan työn case-osuudessa. Tässä tavoitteessa onnistuttiin osittain, mutta huomattiin myös, ettei yksi ihminen pysty hallitsemaan suvereenisti kaikkia alueita. Sovelluksen kehitykseen vaaditaan kokonainen työryhmä, joka koostuu eri alojen osaajista. On erittäin tärkeää, että työryhmän eri jäsenten kesken kommunikointi toimii mutkattomasti sekä myös se, että kaikilla jäsenillä on selkeä, yhtenäinen, mielikuva lopputuotteesta.

CSS:n avulla voidaan, ja kannattaa erottaa, ulkoasu dokumentin rakenteesta. Se on nykyselaimissa laajasti tuettu, ja se saadaan näkymään monipuolisemmin kaikentyyppisille käyttäjäryhmille eri mediatyyppimääritysten avulla. CSS3 tulee viemään CSS:n kehitystä entistä enemmän eteenpäin ja mahdollistaa omalta osaltaan työpöytäsovellusten ja websovellusten fuusioitumisen.

Käytettävyyteen kannattaa kiinnittää huomiota suunnitteluprosessin alusta loppuun saakka. Ohjelmiston tavoitteena on pääasiassa saavuttaa jokin tavoite käyttäjän avulla. Kannattavinta se olisi tehdä käyttäjälle mahdollisimman vaivattomaksi, toteutusvaiheen hyvällä ja perinpohjaisella suunnittelulla. Tämän avulla säästetään käyttäjän hermoja sekä ohjelmiston tuottajan kustannuksia. Vaikka ohjelmiston suunnittelussa tai totetuksessa epäonnistutaan, kannattaa se dokumentoida perinpohjaisesti. Tällä tavoin voidaan välttää jo aiemmin tehdyt virheet siinä vaiheessa, kun ollaan kehittämässä uutta versiota ohjelmistosta.

Graafiset käyttöliittymät ovat näkyvä osa nykyaikaista webkulttuuria. Graafisten käyttöliittymien täytyy olla erittäin pitkälle suunniteltuja, jotta pelkällä graafisella ulkoasulla pärjää. On eri asia, onko käyttöliittymä hyvännäköinen vai onko se hyvä käytettävyydeltään. Mikäli nämä molemmat ominaisuudet pätisivät samassa käyttöliittymässä, on se yleensä hyvin pitkälle viedyn sovelluskehitysprosessin tulos. Graafisen käyttöliittymän suunnittelussa ei kannata mennä liiallisuuksiin, koska tässäkin yhteydessä yksinkertaisuus on edellytys hyvään lopputulokseen. Käyttäjän toimintaan pyritään vaikuttamaan monilla tavoin, mikä on kuitenkin pyrittävä tekemään mahdollisimman yksiselitteisesti. Käyttöliittymää suunniteltaessa ei kannata rynnätä toteuttamaan lopullista sovellusta heti ensimmäisenä, vaan se tapahtuu kehitysprosessin kautta rakentamalla aluksi pelkkä prototyyppi. Prototyyppiä voidaan käyttää apuna selvittämään kehitettävän sovelluksen käytettävyysongelmat.

Kehitysprosessiin kuuluu myös olennaisena osana tuotteen testaaminen. Testaamisen avulla havaitaan jo varhaisessa vaiheessa olennaisimmat ongelmat, ja nämä voidaan korjata ennen kuin sovelluksesta tehdään lopullista versiota. Testaamisella minimoidaan mahdolliset suuret ongelmat, joita voi ilmaantua varsinaisen tuotteen käytössä. Nykyajan käytäntö tuntuu olevan kuitenkin se, että testaaminen suoritetaan vasta varsinaisilla loppukäyttäjillä, ohjelmiston julkistamisen jälkeen. Tämän tarkoitus on ilmeisesti vain saada tuote mahdollisimman pian markkinoille ja säästää tuotantokustannuksissa.

Webkehitystyössä täytyy tuntea useita eri tekniikoita, jotta pystyisi olemaan mukana kehittämässä monipuolisia ja rikkaita websovelluksia. Ajax on tämän hetken kuumimpia termejä webkehityksessä. Yhdessä uusien ja vanhojen webtekniikoiden kanssa sen avulla voidaan rakentaa moninaisia ja käytännöllisiä sovelluksia. Täytyy kuitenkin muistaa, ettei uusia tekniikoita käytetä läheskään aina toimiviin tai hyviin ratkaisuihin. Toimimattomia ja huonoja ratkaisuja on siinä missä hyviäkin. Webkehitys ei rajoitu pelkästään selaimenpäässä tapahtuvaan ohjelmointiin, vaan vähintään yhtä tärkeää, ellei tärkeämpääkin, on palvelinpään ohjelmointi. Näiden kahden saumaton yhteentoimivuus on ensiarvoisen tärkeää.

Työn alkuosassa käsitellyt asiat kulminoituvat case-osassa käsitellyn CSS-teemaeditorin toteutukseen. Lähtökohtaisesti kannattaa olla selkeä päämäärä mihin CSS-editorin kanssa halutaan päästä. Vertailukohtina editorille voidaan pitää jo toteutettuja CSS-editoreita, jotta voidaan esimerkiksi välttää niissä ilmeneviä ongelmia. Projektin tavoite saavutettiin ainakin osittain. Saatiin toteutettua en-

simmäinen prototyypin versio editorista, jolla voi jo testata teemaeditorin pääominaisuuksia. Täytyy kuitenkin myöntää, että vielä on kehittämisen varaa. Siinä mielessä ei aivan tavoitteeseen päästy, että olisi ehditty testata prototyyppiä kaikilla halutuilla tavoilla. Heuristinen arviointi prototyypille tehtiin, mutta loppukäyttäjätestiä ei ehditty projektin aikana toteuttaa. Case-osuudessa tehdyt virheet kannattaa dokumentoida hyvin, jotta mahdolliselle uudelle editori versiolle saataisiin mahdollisimman hyvät lähtökohdat.

Seuraava vaihe teemaeditorin kehitysprosessissa olisi prototyypin käyttäjäryhmällä testaaminen ja testitilanteen dokumentointi. Saatujen tulosten perusteella teemaeditorin kehityssuuntaa voitaisiin tarkentaa ja tehdä siitä parhaiten käyttäjäryhmän tarpeita vastaava.

Selainpohjaisten sovellusten kehittyessä entistä enemmän työpöytäsovellusten kaltaisiksi tulee niiden kanssa vastaan täysin uusia haasteita, jotka voivat olla enemmän ominaisia työpöytäsovelluksille. Työpöytäsovellusten kaltaisia RIA-sovelluksia voidaan toteuttaa selainpohjaisina versioina esimerkiksi Flashin ja JavaScriptin avulla.

LÄHTEET

Kirjalliset lähteet:

Asleson, R., Schutta, N.T. 2007. Ajax: Tehokas hallinta. Gummerus.

Heinisuo, R. 2001. PHP ja MySQL, tietokantapohjaiset verkkopalvelut. Jyväskylä: Gummerus Kirjapaino Oy.

Korpela, J.K., Linjama, T. 2005. Web-suunnittelu. Porvoo: Docendo.

Korpela, J.K. 2008. CSS verkkosivujen muotoilussa. Porvoo: WS Bookwell (Docendo).

Kuutti, W. & Talentum Media Oy. 2003. Käytettävyys, suunnittelu ja arviointi. Saarijärvi: Gummerus Kirjapaino Oy.

Peltomäki, J., Nykänen, O. 2006. Web selainohjelmointi. Porvoo: Docendo.

Sinkkonen, I., Kuoppala, H., Parkkinen, J. & Vastamäki, R. 2006. Käytettävyyden psykologia. 3. Uudistettu painos. Helsinki: Edita Prima Oy.

Smith, D., Negrino, T. 2007. JavaScript: Tehokas hallinta. Readme.fi.

Thau, D. 2006. Book of JavaScript: A Practical Guide to Interactive Web Pages. 2. painos. San Fransisco, CA, USA: No Strarch Press, Incorporated.

Wium Lie, H., Boss, B. 1999. Cascading Style Sheets: Designing for the web. 2. painos. Addison-Wesley.

Julkaisemattomat lähteet:

Vallittu, M. 2008. Kehysohjelmisto Ajax –portaalin käyttöliittymän toteutukseen. Ohjelmistotekniikan opinnäytetyö. Lahti: Lahden ammattikorkeakoulu, Tekniikan laitos.

Inkiläinen, M 2007. Tietokantasovelluksen käyttöohjepohja. Mediatekniikan opinnäytetyö. Lahti: Lahden ammattikorkeakoulu, Tekniikan laitos.

Verkkolähteet:

2kmediat.com 2000-2008a. CSS opas: Internet Explorer-selainten CSS-laajennukset. Koulutus- ja konsultointipalvelut KK Mediat [viitattu: 21.2.2008]. Saatavissa: http://www.2kmediat.com/css/ie55scrollbars.asp

2kmediat.com 2000-2008b.CSS opas: Mozilla/ Firefox –selainten CSS-laajennukset. Koulutus- ja konsultointipalvelut KK Mediat [viitattu: 21.2.2008]. Saatavissa: http://www.2kmediat.com/css/mozilla-laajennukset.asp

2kmediat.com 2000-2008c. CSS opas: Johdatus tyylikieliin ja CSS:n historiaan. Koulutus- ja konsultointipalvelut KK Mediat [viitattu: 21.2.2008]. Saatavissa: http://www.2kmediat.com/css/johdanto.asp

2kmediat.com 2000-2008d. CSS opas: CSSN versiot. Koulutus- ja konsultointipalvelut KK Mediat [viitattu: 22.2.2008]. Saatavissa: http://www.2kmediat.com/css/johdanto2.asp

2kmediat.com 2000-2008e. CSS opas: Keskeiset säännöt ja periaatteet. Koulutus- ja konsultointipalvelut KK Mediat [viitattu: 22.2.2008]. Saatavissa: http://www.2kmediat.com/css/periaate.asp

2kmediat.com 2000-2008f. CSS opas: Cascade prosessi. Koulutus- ja konsultointipalvelut KK Mediat [viitattu: 27.3.2008]. Saatavissa: http://www.2kmediat.com/css/cascade.asp

2kmediat.com 2000-2008g. CSS opas: Syntaksit ja arvot – valitsin esittelylohkot ja !important. Koulutus- ja konsultointipalvelut KK Mediat [viitattu: 27.3.2008]. Saatavissa: http://www.2kmediat.com/css/syntaksi-valitsin.asp

2kmediat.com 2000-2008h. CSS opas: Syntaksi ja arvot – arvot ja mittayksiköt. Koulutus– ja konsultointipalvelut KK Mediat [viitattu 28.3.2009]. Saatavissa: http://www.2kmediat.com/css/syntaksi3.asp

2kmediat.com 2000-2008i. CSS opas: Periytyminen. Koulutus- ja konsultointipalvelut KK Mediat [viitattu 29.3.2009]. Saatavissa: http://www.2kmediat.com/css/periytyminen.asp

2kmediat.com 2000-2008j. CSS opas: Tuetut mediatyypit ja CSS-tyylin kohdistaminen määrätylle medialle. Koulutus- ja konsultointipalvelut KK Mediat. [viitattu 29.3.2009]. Saatavissa: http://www.2kmediat.com/css/mediatyypit.asp

2kmediat.com 2000-2008k. SQL-opas: Johdatus SQL:n maailmaan. Koulutus- ja konsultointipalvelut KK Mediat [viitattu 8.4.2009]. Saatavissa: http://www.2kmediat.com/sql/alkeet.asp

Adage Usability 2006. Prototyyppi. Adage Usability [viitattu: 6.4.2009]. Saatavissa: http://www.adage.fi/palvelut/suunnittelu/prototyyppi.html

Asmala, H. 2006. Relaatiotietokannat – SQL – Yleistä. Hannu Asmala (SAMK) [viitattu 7.4.2009]. Saatavissa: http://www.tp.spt.fi/~salabra/ha/Relaatiotietokannat/SQL/SQL%201%20Yleista.html

CSS3.info 2009. CSS3.info [viitattu 30.3.2009]. Saatavissa: http://www.css3.info

Garret, J.J. 2005. Ajax: A new Approach to Web Applications. Adaptive Path Inc. [viitattu: 20.2.2008]. Saatavissa: http://www.adaptivepath.com/ideas/essays/archives/000385.php

Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2004a. Suunnittelun vaiheet. JYT, Jyväskylän yliopiston IT-tiedekunta [viitattu 7.4.2009] Saatavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/suunnittelu/index1.html

Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2004b. Suunnittelun vaiheet. JYT, Jyväskylän yliopiston IT-tiedekunta [viitattu 7.4.2009]. Saatavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/suunnittelu/index3.html

Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2004c. Suunnittelun vaiheet. JYT, Jyväskylän yliopiston IT-tiedekunta [viitattu 7.4.2009]. Saatavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/suunnittelu/index4.html

Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2005. CSS-kikat, CSS3, sivuston testaaminen ja kertaus –Luento 12- WWW-julkaiseminen ITK024. JYT, Jyväskylän yliopiston IT-tiedekunta [viitattu 15.2.2008]. Saatavissa: http://appro.mit.jyu.fi/2004/syksy/www/luennot/luento12/#TOC7

Mureakuha 2006a. Yleistä tietokannoista. Mureakuha [viitattu 7.4.2009]. Saatavissa: http://wiki.mureakuha.com/wiki/Yleist%C3%A4_tietokannoista

Mureakuha 2006b. SQL aloittelijoille. Mureakuha [viitattu 7.4.2009]. Saatavissa: http://wiki.mureakuha.com/wiki/SQL_aloittelijoille

Smashing Magazine 2008. CSS Editors Reviewed. Smashing Media GmbH i.Gr [viitattu 27.3.2009]. Saatavissa: http://www.smashingmagazine.com/2008/06/19/css-editors-reviewed/

Smashing Magazine 2009. You're your Web Design Into the Future with CSS3. Media GmbH i.Gr [viitattu 8.4.2009] Saatavissa: http://www.smashingmagazine.com/2009/01/08/push-your-web-design-into-the-future-with-css3/

Wikipedia 2008a, Ajax (ohjelmointi). Wikimedia –säätiö [viitattu 21.1.2008]. Saatavissa: http://fi.wikipedia.org/wiki/Ajax_%28ohjelmointi%29

Wikipedia 2008b. Ajax (programming). Wikimedia –säätiö[viitattu 15.2.2008]. Saatavissa: http://en.wikipedia.org/wiki/Ajax_%28programming%29

Wikipedia 2008c. Cascading Style Sheets. Wikimedia -säätiö [viitattu 13.2.2008]. Saatavissa: http://en.wikipedia.org/wiki/Cascading_Style_Sheets

Wikipedia 2008d. JavaScript. Wikimedia -säätiö [viitattu 14.2.2008]. Saatavissa: http://fi.wikipedia.org/wiki/JavaScript

Wikipedia 2008e. JavaScript (eng.). Wikimedia -säätiö [viitattu 14.2.2008]. Saatavissa: http://en.wikipedia.org/wiki/Javascript

Wikipedia 2009a. Style sheet language. Wikimedia -säätiö [viitattu: 17.3.2009]. Saatavissa: http://en.wikipedia.org/wiki/Stylesheet_language

Wikipedia 2009b. PHP. Wikimedia –säätiö[viitattu 30.3.2009]. Saatavissa: http://fi.wikipedia.org/wiki/PHP

Wikipedia 2009c. Tietokanta. Wikimedia-säätiö [viitattu 7.4.2009]. Saatavissa: http://fi.wikipedia.org/wiki/Relaatiotietokanta

Wikipedia 2009d. Tietokannan normalisointi. Wikimedia-säätiö [viitattu 7.4.2009]. Saatavissa: http://fi.wikipedia.org/wiki/Tietokannan_normalisointi

Wikipedia 2009e. Look and Feel. Wikimedia-säätiö [viitattu 8.4.2009]. Saatavissa: http://en.wikipedia.org/wiki/Look_and_feel

Wikipedia 2009f. Internet Explorer 8. Wikimedia-säätiö [viitattu 14.4.2009] Saatavissa: http://en.wikipedia.org/wiki/Internet_Explorer_8

W3C 2001. Introducion to CSS3. World Wide Web Consortium [viitattu 7.4.2008]. Saatavissa: http://www.w3.org/TR/css3-roadmap

W3C 2008a. Box model. World Wide Web Consortium [viitattu 29.3.2009]. Saatavissa: http://www.w3.org/TR/CSS2/box.html#img-boxdim

W3C 2008b. Media types. World Wide Web Consortium [viitattu 29.8.2009]. Saatavissa: http://www.w3.org/TR/CSS2/media.html

MUUT LÄHTEET:

Rosendahl Digital Networks Oy 2008, Power-point esitys yrityksestä.

KUVALÄHTEET:

Kuva 1. W3C 2008. Box model. World Wide Web Consortium [viitattu: 29.8.2009]. Saatavissa: http://www.w3.org/TR/CSS2/images/boxdim.gif

Kuva 2. Ekonoja, A., Lahtonen, T. & Mäntylä, J. 2004. [viitattu: 7.4.2009] Saatavissa: http://appro.mit.jyu.fi/doc/tiedonhallinta/suunnittelu/index3.html

Kuva 3. Törrö, A. 2009 Kuvakaappaus Stylizer –editorista.

Kuva 4. Törrö, A. 2009. Kuvakaappaus TopStyle-editorista.

Kuva 5. Törrö, A. 2009. Kuvakaappaus SimpleCSS-editorista.

Kuva 6. Törrö, A. 2009 Kuvakaappaus Mozilla Firefox selaimesta ja laajennuksista.

Kuva 7. Törrö, A. 2009. Rautalankamalli portaaliratkaisusta.

Kuva 8. Törrö, A. 2009. Luonnoskuva teemaeditorin käyttöliittymästä.

Kuva 9. Törrö, A. 2009. Kuvakaappaus editorisovelluksesta.

TAULUKKOLÄHTEET:

TAULUKKO 1. 2kmediat.com 2000 – 2008d. CSS opas: CSS:n eri versiot [viitattu 22.2.2008]. Saatavissa: http://www.2kmediat.com/css/johdanto2.asp

TAULUKKO 2. 2kmediat.com 2000-2008f. CSS opas: Cascade prosessi [viitattu: 27.3.2008]. Saatavissa: http://www.2kmediat.com/css/cascade.asp

TAULUKKO 3. Wium Lie, H., Boss, B. 1999. Cascading Style Sheets: Designing for the web. 2. painos. Addison-Wesley.

LIITTEET

Liite 1: CD-ROM
Sisältö: Opinnäytetyö pdf-muodossa, tiivistelmä ja englanninkielinen abstrakti rtf-muodossa sekä sähköiset lähteet.