TAMPEREEN
AMMATTIKORKEAKOULU

OPINNÄYTETYÖRAPORTTI

WEB-STANDARDIT JA NIIDEN SOVELTAMINEN

Timo Sulanne
Kimmo Tapala

Tietojenkäsittelyn koulutusohjelma
Joulukuu 2005
Työn ohjaaja: Rami Lehtinen

TAMPERE 2005

TAMPEREEN
AMMATTIKORKEAKOULU

Tekijät Timo Sulanne ja Kimmo Tapala

Koulutusohjelma Tietojenkäsittely

Tutkintotyön nimi Web-standardit ja niiden soveltaminen

Työn valmistumis-kuukausi ja -vuosi Joulukuu 2005

Työn ohjaaja Rami Lehtinen **Sivumäärä:** 74

TIIVISTELMÄ

Web-standardit ovat nousseet web-kehittäjien keskeiseksi puheenaiheeksi. Niinpä myös Tampereen ammattikorkeakoulu on sisällyttänyt opetussuunnitelmaansa hyvin paljon web-standardeihin liittyviä opintojaksoja. Koska kuitenkin web-standardien tuntemus on keskimäärin melko heikkoa, päätimme tehdä opinnäytetyönämme tutkielman web-standardeista, jota voitaisiin käyttää myös opetusmateriaalina.

Molemmilla tekijöillä on jo vuosien kokemus web-standardien mukaan toteutetuista verkkosivustoista. Pyrimme yhdistämään oman käytännön kokemuksemme web-standardeista kirjoitettuihin normatiivisiin spesifikaatioihin ja näin luomaan kattavan oppimateriaalin siitä, mitä web-standardeilla tarkoitetaan ja miten niitä voidaan hyödyntää.

Opinnäytetyö toimii jo pelkästään varsin syväluotaavana oppimateriaalina web-standardeista. Työssä käsitellään perusteellisesti XHTML (Extensible Hypertext Markup Language) -dokumenttien ja CSS (Cascading Style Sheets) -tyylitiedostojen rakenne, käyttö ja yhdistäminen. Opinnäytetyöstä on varmasti hyötyä web-standardeihin liittyvien opintojaksojen opettajille, vaikka työtä ei opiskelijoiden oppimateriaalina käytettäisikään.

Avainsanat web-standardit, XHTML, CSS

TAMPERE
POLYTECHNIC

Authors	Timo Sulanne and Kimmo Tapala
Degree Programme	Business Information Systems
Title	Web Standards and Their Application
Month and Year	December 2005
Supervisor	Rami Lehtinen

Pages: 74

ABSTRACT

Web standards have lately become something of a hot topic among web developers. Tampere Polytechnic has also decided to add some web standards related courses to its curriculum. However, web developers' level of knowledge of web standards seems to be relatively low. This is why we decided to make this thesis about web standards to serve as a part of a learning and teaching material.

Both of the authors have a few years of experience of working with web sites based on standards. Our goal was to combine our practical experience with the normative specifications of the web standards, thus creating a comprehensive learning and teaching material of what are and what can be accomplished through the web standards.

This thesis may not be suitable for teaching purposes as such. However, it covers the structure and usage of XHTML (Extensible Hypertext Markup Language) documents and CSS (Cascading Style Sheets) style sheets so that it may be used as a learning material for the teachers who are teaching web standards related courses.

Keywords web standards, XHTML, CSS

Sisällysluettelo

1 Johdanto

Alkuperäisenä tavoitteenamme oli tuottaa opinnäytetyönämme sekä tutkielma web-standardeista, tarkemmin ottaen XHTML (Extensible Hypertext Markup Language) -dokumenteista ja CSS (Cascading Style Sheets) -tyylitiedostoista, että sähköinen oppimateriaali, jolla voitaisiin opettaa web-standardeihin liittyviä asioita. Mitä syvemmälle kuitenkin uppouduimme web-standardien tutkimiseen ja hyödyntämiseen, sitä selvemmäksi meille tuli, että joudumme jättämään sähköisen oppimateriaalin pois opinnäytetyöstämme.

Koska jouduimme uhraamaan sähköisen oppimateriaalin kattavan tutkielman edessä, oli selvää, että tutkielman tulisi ainakin suurimmaksi osaksi korvata oppimateriaalin rooli. Päätimmekin kirjoittaa opinnäytetyömme mahdollisimman käytännönläheiseksi siten, että samalla pystymme säilyttämään standardien vaatimat tarkat ilmaisut. Tätä käytännönläheisyyttä osoittaa parhaiten ehkä se, että työ sisältää verrattain paljon pieniä koodiesimerkkejä, jotka havainnollistavat kulloinkin käsiteltävää aihetta.

Web-standardit eivät ole varsinaisesti mikään virallinen standardiryhmä, vaan kontekstista riippuen web-standardeihin voidaan liittää lähes mitä tahansa World Wide Web Consortiumin, eli W3C:n, standardoimia teknologioita. Tässä opinnäytetyössä keskitymme pääasiallisesti XHTML-dokumenttehin sekä CSS-tyylitiedostoihin.

Web-standardit on sikäli varsin ajankohtainen aihe, että selainkehityksen ja kasvaneen saavutettavuuskiinnostuksen vuoksi web-standardien osaamista edellytetään tällä hetkellä lähes kaikessa nykyaikaisessa verkkojulkaisemisessa. Onkin hienoa, että koulumme on tämän vuoden aikana ottanut web-standardien opetuksen tosissaan mukaan opetussuunnitelmaansa ja näin ollen on myös ollut varsin kannustava opinnäytetyömme synnytysprosessin suhteen.

2 Lyhenteiden ja termien selitykset

ASCII American Standard Code for Information Interchange. Standardoitu koodikokoelma merkkien esittämiseen tietokoneilla.

CSS Cascading Style Sheets. W3C:n standardoima teknologia verkkosivujen esitysasun määrittelemiseen.

DOM Document Object Model. Puurakenteisen dokumentin rakennetta kuvaava malli.

DTD Document Type Definition. DTD määrittelee dokumenttityypin.

ECMA European Computer Manufacturers Association. ECMA on yksi webstandardeja kehittävistä organisaatioista.

ECMAScript ECMA:n luoma standardiversio JavaScriptistä.

HTML Hypertext Markup Language. HTML on merkkauskieli Internetissä julkaistavien dokumenttien luomiseen.

ISO-8859-1 International Standards Organizationin standardoima merkistökoodaus. ISO-8859-1:stä käytetään usein myös nimeä Latin-1.

JavaScript Skriptauskieli selainohjelmointiin. JavaScriptillä voidaan luoda toiminnallisuutta verkkosivuille.

MathML Mathematical Markup Language. MathML on matemaattisten kaavojen esittämiseen luotu XML-sovellus.

MIME-tyyppi Multipurpose Internet Mail Extensions. MIME-tyyppi kertoo, millaista tietoa jokin tiedosto sisältää.

SVG Scalable Vector Graphics. SVG on XML-sovellus vektorigrafiikan luomiseen ja esittämiseen.

Tagi (X)HTML-merkkauskielen elementin merkattu muoto. Esimerkiksi <p> -tagi on p-elementin aloitustagi.

URI Uniform Resource Identifier. Standardisoitu tapa viitata Internetissä olevaan lähteeseen.

URL Uniform Resource Locator. URL on dokumentin Internet-osoite.

UTF-8 Unicode Transformation Format (8 bit) on 8-bittinen merkistökoodaus, jolla pystytään esittämään kaikki Unicode-standardin sisältämät merkit.

W3C World Wide Web Consortium. W3C on vuonna 1994 perustettu, Tim Berners-Leen johtama, organisaatio, joka kehittää ja standardoi Internetissä käytettäviä teknologioita.

WaSP The Web Standards Project. WaSP on web-standardien promootioon keskittynyt organisaatio.

XHTML Extensible Hypertext Markup Language. XHTML on XML-pohjainen sovellus HTML:stä.

XML Extensible Markup Language. XML on puumaiseen rakenteeseen perustuva merkkauskieli.

XSL Extensible Stylesheet Language. XSL-määrittelyllä lisätään tyylejä XML-tiedostoon.

XSLT Extensible Stylesheet Language Transformation. XSLT-muunnoksessa XML-tiedosto muutetaan XSL-määrittelyjen mukaiseksi.

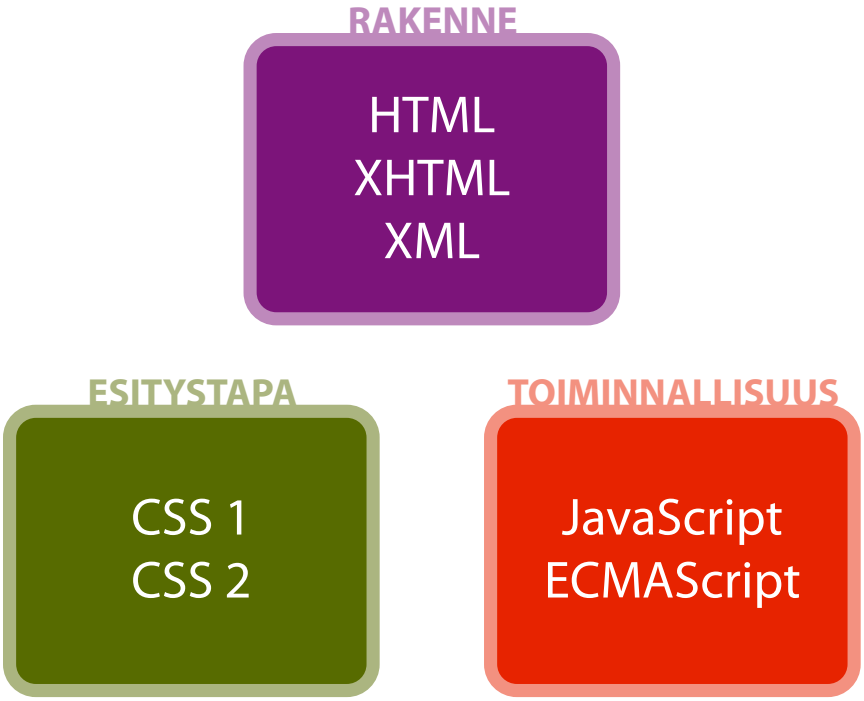
3 Mitä ovat web-standardit?

World Wide Web Consortium (W3C) sekä sen yhteistyökumppanit ovat kehittäneet teknologioita, ohjeita sekä ehdotuksia web-pohjaisen sisällön luomiseen sekä tulkitsemiseen. Näistä teknologioista käytetään yhteistä käsitettä web-standardit. Web-standardit on suunniteltu takaamaan webissä julkaistaville dokumenteille pitkäikäinen elinkaari ja saavutettavuus teknologioiden sekä selainten kehityksestä huolimatta. Web-standardeilla tarkoitetaan eritoten webin rakenteellisia kuvauskieliä, esitysasun kuvauskieliä, objektimalleja sekä skriptikieliä. (Zeldman 2003: 16)

Yleisimmin käytetyt web-standardit:

- Rakenteelliset kuvauskielet
  - HTML (Hypertext Markup Language) 4.01
  - XHTML (Extensible Hypertext Markup Language) 1.0
  - XHTML 1.1
  - XML (Extensible Markup Language) 1.0
- Esitysasun kuvauskielet
  - CSS (Cascading Style Sheets) Level 1
  - CSS Level 2 revision 1
  - CSS Level 3 (kehitteillä)
  - MathML (Mathematical Markup Language)
  - SVG (Scalable Vector Graphics)
- Objektimallit
  - DOM (Document Object Model) Level 1
  - DOM Level 2
  - DOM Level 3 Core
- Skriptikielet
  - ECMAScript 262 (JavaScriptin standardoitu versio)

Perusajatuksena web-standardien käytössä on rakenteen, esitystavan ja toiminnallisuuden erottaminen toisistaan (kuva 1).



Kuva 1. *Standardeilla rakennetun web-sivuston päärakennusmateriaalit.*

Rakenne

Merkkauskieli, kuten XHTML, sisältää tekstimuotoista dataa, joka on kirjoitettu rakenteellisesti ensimmäisen tason otsikoista, toisen tason otsikosta, kappaleesta, listoista jne. Vaikkakin XML tarjoaa huomattavasti enemmän vaihtoehtoja XHTML:ään verrattuna, niin tässä opinnäytetyössä keskitytään pääasiassa kuitenkin XHTML-merkkauskieleen, mikä on tällä hetkellä myös W3C:n suositus. Oikein laadittuna XHTML-merkkaus toimii HTML:n tapaan melkein jokaisessa web-selaimessa, ruudunlukijassa, tekstipohjaisessa selaimessa tai missä tahansa muussa Internet-päätelaitteessa. (Zeldman 2003: 54.)

Esitystapa

Esitysasun kuvauskielillä (CSS1 ja CSS2) määritellään web-sivuston typografia, elementtien sijoittelu, värit jne. eli kaikki mitä ulkoasun määrittelyyn kuuluu. CSS:n käytöllä pystytään korvaamaan turhat ulkoasumäärittelyt itse varsinaisen merkkauksen yhteydessä. Ulkoasun ollessa erillään rakenteesta mahdollistetaan myös sekä rakenteen että esitysasun nopeat muutokset ilman vaaraa siitä, että toinen vahingoittuisi päivityksen yhteydessä. Jotkut ulkoasuun tehtävät muutokset vaativat kuitenkin muutoksien tekemisen myös merkkaukseen.

Toiminnallisuus

Standardien mukaisella objektimallin (DOM) sekä ECMAScriptin käytöllä saadaan aikaiseksi kehittyneitä käyttäytymismalleja sekä tehosteita, jotka toimivat eri alustoilla ja selaimilla.

3.1 Miksi web-standardit?

Suurimmat hyödyt web-standardien käytössä ovat sivustojen eteenpäinyhteensopivuuden (forward compatilbility) takaaminen sekä saavutettavuuden (accessibility) parantuminen.

Web-standardeista puhuttaessa eteenpäinyhteensopivuudella tarkoitetaan sitä, että webissä julkaistavat dokumentit, jotka toimivat jo nyt erilaisilla selaimilla, alustoilla sekä Internet-päätelaitteilla, toimivat myös tulevaisuudessa. Eteenpäinyhteensopivuus on tietenkin vaikeampi saavuttaa kuin taaksepäinyhteensopivuus, koska tulevaa ei pystytä ennustamaan. Nykyiset web-standardit ovat kuitenkin suunniteltuja oikein käytettynä takaamaan toiminnan. Eteenpäinyhteensopivuuden voi kuvitella myös tilana, mihin kaikkien web-suunnittelijoiden tulisi pyrkiä. Mitä enemmän suunnittelijat käyttävät web-standardeja suunnittelemissaan projekteissa, sitä enemmän ohjelmistotuottajat kehittävät standardeja tukevia kehitystyökaluja ja selaimet ovat pakotettuja toimimaan yhä enemmän standardien mukaisesti.

Sen lisäksi, että nykyiset web-standardit ovat suunniteltuja toimimaan tulevaisuudessakin, ne on kirjoitettu niin, että myös vanhemmat selaimet ymmärtävät dokumenttien perusrakenteen. Sivut eivät välttämättä näytä samalta, miltä ne näyttävät uudemmilla selaimilla, mutta tärkein osa, eli sisältö, on aina käyttäjien saavutettavissa. Sama asia pätee myös hakukoneiden hakurobotteihin, jotka keräävät tietoa sivustosta ja sen sisällöstä. Sisällön ollessa erillään esitystavasta dokumentit ovat rakenteellisempia ja näin ollen niiden sisältö on helpommin käsiteltävissä ja indeksoitavissa. Tämä takaa myös paremman näkyvyyden hakukoneiden hakutuloksissa. (Web Standards Project 2002.)

Saavutettavuudella kuvataan pääasiassa Internetissä sijaitsevan materiaalin esteettömyyttä sekä käytettävyyttä. Saavutettavuutta voidaan mitata sillä, kuinka hyvin web-sivusto ja sen sisältö on kaikkien käyttäjien ulottuvilla huolimatta yksilöllisistä rajoituksista. Tämä ei kuitenkaan tarkoita sitä, että web-sivustojen tulisi olla saavutettavia pelkästään rajoittuneille ihmisille, vaan myös koneille ja ohjelmistoille. Semanttinen ja rakenteellinen merkkaus on tärkeä osa saavutettavuutta.

Edelleenkin moni web-suunnittelija rakentaa sivustot taulukkotaittoa hyväksikäyttäen. Tämä on väärin, koska alkuaan HTML:n taulukkoelementti on suunniteltu pelkästään tabulaarisen eli taulukkomuotoisen tiedon esittämiseen. Taulukkoelementin border-attribuutti mahdollisti kuitenkin taulukon käytön eräänlaisena kehikkona, johon pystyi sijoittamaan tekstiä sekä kuvia. Taulukkoa näin käyttämällä saatiin aikaiseksi lähes samanlaista jälkeä kuin painotuotteissa ja tämä tuntui aivan valtavalta kehitysaskeleelta. Näin ainakin vuonna 1997 ja sitä se itse asiassa olikin verrattuna sitä edeltäneeseen webiin.

Webin ja selainteknologioiden kehitys on kuitenkin mennyt valtavasti eteenpäin sittemmin. Nykyään taulukkotaitolla tehdyt sivustot syövät suuren osan kaistanleveydestä, niiden ollessa merkkaukseltaan tarpeettoman suuria. Lisäksi taulukkotaitolla toteutetun sivuston päivitykset ovat erittäin työläitä puhumattakaan uudelleensuunnittelusta – ja mitä enemmän on tehtävää, sitä suuremmat ovat kustannukset. Oma lukunsa on kuitenkin saavutettavuus, jota ei ole käytettäessä taulukoita esitystavan muotoilussa. (Merikallio, Bill & Pratt, Adam 2003.)

Suunnittelijan näkökulmasta web-standardien käyttö helpottaa sivujen suunnittelua sekä luomisprosessia. Samalla web-sivuston kehitysprosessi yksinkertaistuu ja valmistuskustannukset pienenevät. Standardeilla suunnitellut sivut ovat myös helpommin ylläpidettäviä muidenkin kuin pelkästään alkuperäisen suunnittelijan toimesta.

Web-standardien hyödyt pähkinänkuoressa:

- Nopeampi web-sivujen lataus ja käsittely: Web-sivujen merkkauksen ollessa vähäisempää tiedostot ovat pienempiä ja sivut näin ollen latautuvat nopeammin. Tämä merkitsee myös sitä, että palvelinpuolella saadaan säästöjä aikaiseksi palvelintilan sekä kaistanleveyden suhteen.
- Yksinkertaisempi kehitys ja ylläpito: Sisällön ja esitystavan erottaminen eli CSS-tyylitiedostojen käyttö yhdessä semanttisen merkkauksen kanssa yksinkertaistaa sivujen suunnittelua ja päivittämistä. Sisällön ja esitystavan pitäminen erillään mahdollistaa esimerkiksi koko sivuston kattavat päivitykset nopeasti päivittämällä vain haluttua tyylitiedostoa. Näin ollen myös sivustojen uudelleensuunnitteleminen helpottuu huomattavasti.
- Parempi saavutettavuus: Web-standardien käyttö mahdollistaa paremman saavutettavuuden niin ohjelmistoille, koneille kuin ihmisillekin. Toisin sanoen sisältö on helposti luettavissa eri selaimilla, näytönlukijoilla, matkapuhelimilla ja käsitietokoneilla valmistajista riippumatta. Web-standardien oikea käyttö takaa myös paremmat sijoitukset webin hakukoneissa.

- Eteenpäinyhteensopivuus: Web-standardien käyttäminen vähentää riskiä, että tulevaisuudessa kehitettävät selaimet eivät ymmärtäisi käytettyä merkkausta. Tällä taataan suunniteltujen dokumenttien toimivuus myös tulevaisuudessa.

(Cederholm 2004: 22.)

3.2 Ongelmat web-standardien käytössä

Useat websivustot ovat taidokkaasti ohjelmoituja, erittäin hienoja visuaalisesti sekä sisältävät hyvin kirjoitettua ja mielenkiintoista sisältöä. Kuitenkin suurimmassa osassa on unohdettu täysin rakenteellinen merkkaus, tyylitiedostot sekä standardipohjainen toiminnallisuus: osa sivuista on tehty täysin Flashillä, osa toimii vain Internet Explorer (IE) -selaimella (ja näin ollen pelkästään Microsoft Windows -alustalla) tai ne eivät ole taaksepäinyhteensopivia vanhojen selainversioiden kanssa. Web-kehittäjät haluavat samoja asioita kuin heidän asiakkaansakin, joten tuotelähtöistä ajattelua tulisi soveltaa aina toisinaan myös websivustoihin. Mitä useamman ihmisen saavutettavissa palvelu tai sisältö on, sitä enemmän on mahdollisia asiakkaita.

Monet suunnittelijat luulevat, että web-standardeja käyttäen on vaikeampaa suunnitella toimivia sekä samalla näyttäviä sivustoja. Tähän ehkä yhtenä syynä on se, että ne jotka suunnittelevat standardeja eivät markkinoi niitä oikein: esimerkiksi W3C:n sivusto (http://www.w3.org) ei välttämättä anna täyttä kuvaa web-standardien mahdollisuuksista. Ainoastaan näyttävät web-standardien mukaiset sivustot voivat muuttaa näitä vääriä ennakkokäsityksiä. Onneksi standardien puolestapuhujat ovat huomanneet hiljalleen saman asian ja ovat myös tehneet sille jotain konkreettista. Tästä hyvänä esimerkkinä voi mainita css Zen Gardenin (kuva 2), joka toimii esittelysivustona CSS:n mahdollisuuksista. (Zeldman 2003: 76.)

Kuva 2. *css Zen Garden (http://www.csszengarden.com) toimii esimerkkinä web-kehittäjille CSS:n tarjoamista mahdollisuuksista.*

Suurimmat ongelmat ovat vielä kuitenkin web-standardien hyväksymisessä. Edelleen löytyy web-suunnittelijoita, jotka eivät yksinkertaisesti viitsi tai halua opetella uusia käytäntöjä. Nämä suunnittelijat kuitenkin unohtavat sen tosiasian, että sisältö on tärkein asia, mitä käyttäjät sivuilta haluavat. Niin kauan kuin kehittäjät eivät suostu omaksumaan web-standardeja, ohjelmistokehittäjien on pakko tukea standardien vastaista web-suunnittelua.

4 XHTML (Extensible Hypertext Markup Language)

4.1 XHTML:n kehitys

Loppuvuodesta 1999 W3C julkaisi 4.01 version HTML-standardista, joka jäi myös sen viimeiseksi versioksi, sillä W3C:llä ei ole aikomustakaan kehittää sitä enää eteenpäin. Versioon sisällytettiin paljon Netscape Navigatorissa sekä IE:ssä käytettyjä innovaatioita, mikä teki siitä selkeämmän ja siistimmän aikaisempiin versioihin verrattuna. Kehittyneen CSS-tuen lisäksi HTML 4.01 esitteli toimivia toteutusmalleja takaamaan sen toiminnan selaimesta tai alustasta riippumatta. (Musciano & Kennedy 2002: xvii.)

HTML tarjoaa kuitenkin rajoitetun määrän dokumentin luontiin käytettäviä elementtejä ja näin ollen on kykenemätön käsittelemään esimerkiksi kemiallisia kaavoja tai erilaisia matemaattisia ilmaisuja. HTML:stä puuttuvat myös tuet vaihtoehtoisille mediolle, kuten tulostus-, projektio- ja mobiililaitteille. Korjatakseen nämä ongelmat W3C kehitti XML-standardin, mikä tarjosi uuden tavan luoda standardipohjaisia merkkauskieliä. XML-yhteensopivat kielet välittävät tietoa, mitä voidaan tulkita, prosessoida, näyttää tai pilkkoa monilla eri teknologioilla, joita on kehitetty sittemmin. (Musciano & Kennedy 2002: 501.)

XHTML 1.0 on W3C:n ensimmäinen suositus XHTML-standardiksi ja se yhdistää HTML 4.01:n ja XML:n parhaimmat puolet. XHTML 1.0 lainaa elementtejä ja attribuutteja HTML 4.01:stä, mutta se on uudelleenmuotoiltu XML-sovellukseksi. Näin ollen XHTML vastaa täysin XML-standardin vaatimuksia, mutta myös vanemmat selaimet osaavat tulkita sitä. XHTML mahdollistaa XML:n ansiosta laajemman näkyvyyden Internetissä julkaistaville dokumenteille, sillä yhä useampi päätelaite tarjoaa tuen XHTML-standardille. XHTML 1.0 on perusta tulevaisuuden webille.

4.1.1 XHTML:n tulevaisuus

Tämän opinnäytetyön tekovaiheessa XHTML-standardista on julkaistu jo toinen suositus eli XHTML 1.1. XHTML 1.1 esittelee uuden moduulirakenteisen XHTML-dokumenttityypin. Moduulimainen rakenne mahdollistaa sen integroimisen muihin rakenteellisiin kieliin, jolloin dokumentteihin saadaan uusia ominaisuuksia. Huolimatta siitä, että XHTML 1.1 vaikuttaa hyvin samantyyppiseltä kuin XHTML 1.0, se on suunniteltu toimimaan perustana tulevaisuuden XHTML-perheen dokumenttityypeille.

XHTML-standardin kolmas versio eli XHTML 2.0 on luonnosasteella ja se on esitelty kehittäjille kommentointia varten. Vaikkakin XHTML 2.0 perustuu löyhästi HTML 4-, XHTML 1.0- ja XHTML 1.1-standardeihin, se ei tule olemaan taaksepäinyhteensopiva aikaisempien versioiden kanssa. Tässä vaiheessa ei ole kuitenkaan tarkempaa tietoa siitä, koska XHTML 2.0 tulee käyttöön. (World Wide Web Consortium 2005a.)

Tässä opinnäytetyössä keskitytään ainoastaan XHTML 1.0-standardiin, sen ominaisuuksiin ja piirteisiin.

4.2 HTML vastaan XHTML: edut ja rajoitukset

XHTML on XML-pohjainen merkkauskieli, joka toimii ja näyttää HTML:ltä XHTML:n rakenteellisista muutoksista huolimatta. Selainten kannalta XHTML toimii tismalleen samaan tapaan kuin HTML. Kehittäjien kannalta XHTML-merkkauksen tekeminen on myös melkein sama asia, kuin HTML:n, mutta XHTML:n merkkaussäännöt ovat vain hiukan tiukemmat. Lisäksi XHTML sisältää myös joitain uusia elementtejä HTML:ään verrattuna.

HTML-merkkauksessa jotkin elementit on suljettava, kun taas toisia ei saa sulkea missään tapauksessa. Tämä HTML:n ominaisuus saattaa aiheuttaa ongelmia tietyissä tapauksissa. Esimerkiksi jotkin selaimet saattavat jättää HTML-sivun näyttämättä, jos taulukon solun td-elementti on jäänyt sulkematta. Toisin kuin HTML:ssä, XHTML:n kaikki elementit tulee sulkea XML-syntaksin mukaisesti. Web-kehittäjien kannalta tämä XHTML:n ominaisuus ennaltaehkäisee vastaavien ongelmien ilmenemisen. Toinen varsinainen XML:stä periytynyt ominaisuus on se, että kaikki XHTML:n elementit ja attribuutit tulee kirjoittaa pienillä kirjaimilla (attribuuttien arvot pitää myös kirjoittaa lainausmerkkien sisään).

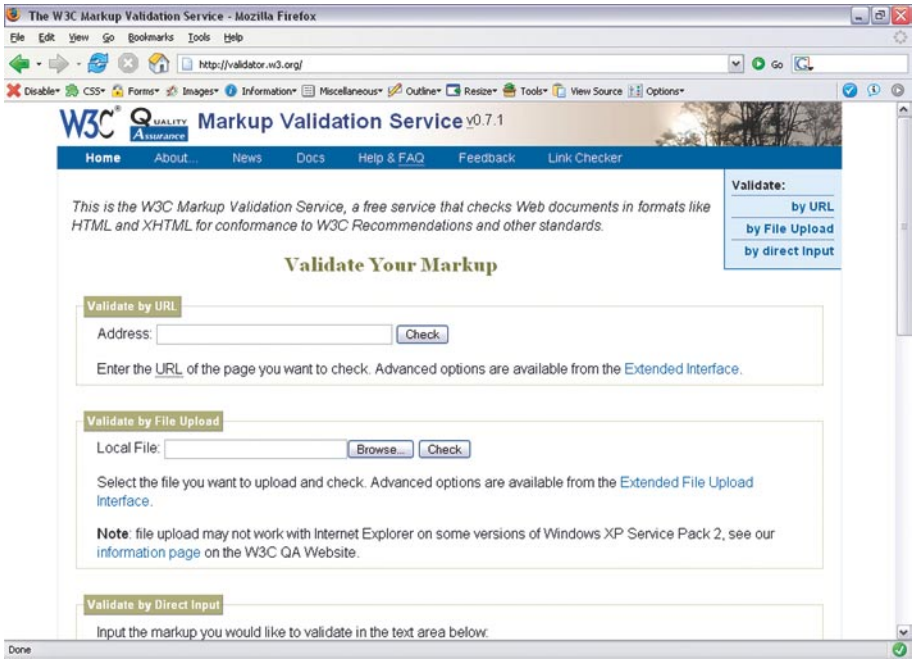
XHTML-merkkauksessa pyritään myös vähentämään ”vanhentuneiden” ominaisuuksien käyttöä. Vanhentuneilla ominaisuuksilla tarkoitetaan tässä yhteydessä tiettyjä elementtejä sekä attribuutteja, joita on käytetty aikaisemmin HTML-sivun ulkoasun määrittelyyn. Näiden vanhentuneiden ominaisuuksien sijaan W3C suosittelee käytettäväksi CSS-tyylimäärittelyjä, joilla saadaan vastaavat ulkoasumuutokset tehtyä nopeammin ja helpommin koko web-sivustolle. Tästä huolimatta kaikki vanhentuneet ominaisuudet toimivat edelleen XHTML:n Transitional -dokumenttityypillä, jotta XHTML:ään siirtyville kehittäjille siirtymisprosessi olisi helpompi. Strict -dokumenttityypillä vanhentuneiden elementtien käyttö aiheuttaa sen, että merkkaus ei ole enää oikeaoppista.

XHTML:n oikeaoppinen käyttö siis rohkaisee kehittäjää tekemään merkkauksesta rakenteellisempaa sekä yhdessä CSS:n kanssa se vähentää edelleen tarvittavan merkkauksen määrää ja dokumenttien käsittely nopeutuu. Ikävä kyllä, kaikista XHTML:n tarjoamista eduista huolimatta, se ei ole vallannut webiä niin nopeasti kuin uskottiin. XHTML-standardia tuetaan kuitenkin selainten lisäksi yhä enenevissä määrin myös kehittäjäympäristöissä, mikä tulee lisäämään varmasti XHTML-kehittäjien määrää.

Viisi syytä vaihtaa HTML:stä XHTML:ään:

1. XHTML on tämänhetkinen standardi merkkaukselle ja se korvaa HTML 4.01:n, jota ei enää kehitetä eteenpäin. XHTML on lisäksi suunniteltu toimimaan yhdessä XML-pohjaisten merkkauskielten, sovellusten sekä protokollien kanssa.
2. XHTML on eteenpäinyhteensopiva merkkauskieli toisin kuin HTML, mutta se toimii yhtälailla myös vanhemmilla selaimilla. XHTML toimii nykyisten selainten lisäksi myös muilla päätelaitteilla, kuten matkapuhelimilla ja ruudunlukijoilla, jolloin sivustot ovat useamman ihmisen saavutettavissa. Oikein käytettynä CSS-tyylitiedostojen kanssa XHTML-merkkauksesta ei tarvitse myöskään tehdä erillisiä versioita eri päätelaitteita varten.
3. XHTML:n käyttäminen johtaa todennäköisesti saavutettavamman merkkauksen tekemiseen sekä validointipalvelujen (kuva 3) käyttöön, mikä säästää aikaa testauksesta ja virheiden etsimisestä.
4. XHTML on osa web-standardien perhettä, jotka auttavat kehittäjää kontrolloimaan www-sivujen käyttäytymistä ja ulkoasua eri käyttöympäristöissä, selaimissa ja päätelaitteissa.
5. XHTML 1.0 on linkki tulevaisuuden XHTML-versioihin. Jos kehittäjä osaa soveltaa XHTML 1.0 -standardia, niin siirtyminen esimerkiksi XHTML 2.0 -standardiin tulee olemaan paljon helpompaa.

   (Zeldman 2003: 150.)



Kuva 3. *HTML/XHTML-dokumentin merkkauksen voi tarkistaa esimerkiksi W3C:n ”The W3C Markup Validation Service” -palvelun avulla. (http://validator.w3.org/)*

4.3 Semantiikka

Semantiikka tutkii merkkien esitettävää tehtävää, niiden viittaussuhteita merkkijärjestelmän ulkopuolelle sekä merkkien tulkintaa ja merkityksiä. Semantiikka pätee yhtä lailla Internetissä julkaistaviin dokumentteihin ja merkkauksen yhteydessä sillä tarkoitetaan sitä, että kuvataan itse sisältöä ja sen merkitystä eikä niinkään sisällön ulkoasua.

Tieto ei ole sama asia kuin informaatio, vaan tieto on tulkittua informaatiota. Informaatiosta tulee tietoa vasta, kun tiedon tasoa vastaava merkkaus on liitetty tiedon yhteyteen. Eli jos jokin sisällön osa on tarkoitettu otsikoksi, niin se merkataan otsikkotyypille kuuluvalla elementillä, kappale merkataan kappaleelle kuuluvalla elementillä ja niin edelleen. Tässä kaikessa ei oteta kantaa siihen, miltä merkatut elementit näyttävät. Ulkoasuun vaikuttavat jäsentelyt tulee tehdä aina varsinaisen merkkauksen ulkopuolella. HTML:n ja XHTML:n tapauksissa ulkoasumäärittelyt tehdään CSS-tyylitiedostoihin.

(X)HTML-dokumentin semantiikan kannalta on yksi ongelma ylitse muiden: HTML:n table -eli taulukkoelementti. Oikeastaan vika ei ole kuitenkaan itse elementissä vaan siinä, miten sitä on totuttu käyttämään. Kun table-elementin border-attribuutille annettiin arvo 0,

huomattiin, että taulukkoa voitiin käyttää sivuston ulkoasun määrittelevänä kehikkona. Taulukkoja ei pitäisi kuitenkaan koskaan käyttää ulkoasun määrittelemiseen, koska se tuhoaa dokumentin semantiikan täysin. Taulukkoa tulisi käyttää ainoastaan, kun jokin osa dokumentista on selvästi tabulaarista eli taulukkomuotoista tietoa.

Oikeiden elementtien käyttäminen ei pelkästään kuitenkaan riitä, että merkkaus olisi semanttista. Käytettäessä CSS-tyylitiedostoa (X)HTML-dokumentin muotoiluun, tarvitsee harkita tarkkaan, mitä nimiä tai luokkia antaa millekin elementeille (elementin nimi määritellään yleisimmin id-attribuutissa). Elementtien nimien tulisi kuvata mahdollisimman tarkkaan, mitä ne sisältävät. Useimmiten sivuston navigaatio koostuu useammasta linkistä, jolloin se on syytä merkata listaksi. Tällöin listaelementin id-attribuutiksi merkattaisiin esimerkiksi ”navigation”, jolloin se ilmaisee listan todellisen käyttötarkoituksen. Navigaatio on kuitenkin vain yksi sivuston nimettävistä kohteista, joten sivuston rakenteen ja semantiikan kannalta on hyvä nimetä myös kaikki muut osat, joita ovat esimerkiksi ylätunniste, sisältö ja alatunniste.

Standardit itsessään eivät ole semanttisia, vaan standardien oikeantyyppinen käyttö edistää semantiikkaa ja samalla julkaistavien dokumenttien saavutettavuutta.

4.4 Dokumenttityypit

XHTML-dokumentti alkaa elementillä, joka määrittelee sen, kuinka eri ohjelmien tulisi käsitellä kyseessä olevaa dokumenttia. Tämä DOCTYPE -elementti (lyhenne sanoista ”document type”) vaikuttaa siis suoraan siihen, miten dokumentin tulisi käyttäytyä loppukäyttäjän selaimessa. DOCTYPE kertoo myös validointipalveluille oikean XHTML-version, jotta ne osaisivat validoida sivuston sekä mahdollisesti sen yhteydessä käytettävän CSS-tyylitiedoston oikein. Näin ollen dokumenttityypin määrittely on erittäin tärkeä osa XHTML-dokumenttia, ja sen käyttöön kannattaa kiinnittää erityistä huomiota.

XHTML:n dokumenttityypit on määritelty XHTML-spesifikaation yhteydessä kolmessa eri dokumenttityyppiesittelyssä eli DTD:ssä (lyhenne sanoista ”document type definition”).

4.4.1 XHTML 1.0 Strict

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Strict-dokumenttityyppiä tulisi käyttää, jos kehittäjä haluaa tehdä merkkauksestaan rakenteellisempaa. HTML 4.01:n vanhentuneet ominaisuudet eivät toimi enää Strict-dokumenttityypillä, joten ulkoasumuotoilut tehdään merkkauksen sijasta sivustoon liitettävään CSS-tyylitiedostoon. CSS:n käyttö on suositeltavaa myös muiden dokumenttityyppien kanssa.

4.4.2 XHTML 1.0 Transitional

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
```

Transitional-dokumenttityyppiä on hyvä käyttää silloin, kun ollaan siirtymässä HTML:stä XHTML:n käyttöön, sillä se on lähinnä HTML 4.01-standardia ja se mahdollistaa vanhentuneiden elementtien sekä attribuuttien käytön. Esimerkiksi, jos on tottunut käyttämään `target="_blank"` -attribuuttia a-elementin kanssa (avaa linkin uuteen selainikkunaan), niin transitional-dokumenttityyppi on silloin ainoa mahdollinen vaihtoehto. Transitional-dokumenttityyppi ei ole muutenkaan niin tarkka merkkauksen suhteen kuin Strict-dokumenttityyppi.

4.4.3 XHTML 1.0 Frameset

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Frameset-dokumenttityyppiä tulisi käyttää vain, jos on aivan pakko käyttää kehyksiä sivuston eri osien jakamiseen selaimessa. Itse kehyksissä tulee kuitenkin käyttää jompaakumpaa edellä mainituista DTD:istä.

4.5 Nimiavaruus

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fi"
lang="fi">
```

XML-nimiavaruus on yksikäsitteinen URI (Uniform Resource Identifier) -merkkijono. XHTML-standardin nimiavaruus esitellään osoit-

tamalla se osoitteeseen http://www.w3.org/1999/xhtml. XHTML:n nimiavaruus sisältää tiedon elementeistä ja attribuuteista, jotka ovat käytettävissä XHTML-dokumentissa.

Nimiavaruus esitellään html-elementin xmlns-attribuutilla. Nimiavaruuden esittely on pakollinen, mutta W3C:n validaattori ei anna kuitenkaan virhettä sen puuttumisesta, sillä `xmlns="http://www.w3.org/1999/xhtml"` on kiinteä arvo ja se lisätään automaattisesti html-elementtiin sen puuttuessa. Esimerkin kaksi muuta attribuuttia, `xml:lang="fi"` ja `lang="fi"`, määrittelevät dokumentin sisällön kielen. Sellaisissa selaimissa, jotka eivät osaa tulkita XHTML:ää XML-dokumenttina sisällön kieli määräytyy lang-attribuutin mukaan, muissa xml:lang-attribuutin.

4.6 Merkistökoodaus

Jotta selaimet ja muut päätelaitteet osaisivat tulkita XHTML-dokumentin ja sen sisällön oikein, dokumentissa tarvitsee määritellä dokumentin käyttämä merkistökoodaus. Yleisimmin käytettyjä merkistökoodauksia ovat UTF-8 (Unicode) ja ISO-8859-1 (Latin-1). UTF-8-merkistökoodausta on hyvä käyttää kansainvälisissä sivustoissa ja ISO-8859-1:tä esimerkiksi suomenkielisissä sivustoissa. Merkistökoodauksen voi esitellä XHTML-dokumentissa kahdella eri tavalla: XML-julistuksella (esimerkki 1) tai META Content -elementillä (esimerkki 2).

Esimerkki 1. XML-julistus.

```
<?xml version="1.0" encoding="ISO-8859-1">
```

XML-julistus määrittelee XHTML-dokumentissa käytettävän XML:n version ja merkistökoodauksen. XML-julistus ei kuitenkaan toimi kaikilla selaimilla (esimerkiksi Internet Explorerilla) tai se saattaa aiheuttaa joitain ongelmia sivuston tulkitsemisessa, joten se on hyvä jättää tietyissä tapauksissa kokonaan pois (kuten se on jätetty pois myös esimerkki 3:ssa). Jos kuitenkin haluaa käyttää XML-julistusta, niin se tulee sijoittaa XHTML-dokumentissa ensimmäiselle riville ennen dokumenttityypin määrittelyä.

Esimerkki 2. META Content -elementti.

```
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1" />
```

XML-julistuksen ohessa on suositeltavaa käyttää META Content -elementtiä. META Content -elementti sijoitetaan dokumentissa head-elementin sisään, yleensä heti title-elementin jälkeen esimerkki 3:n mukaisesti.

4.7 XHTML-dokumentin rakenne

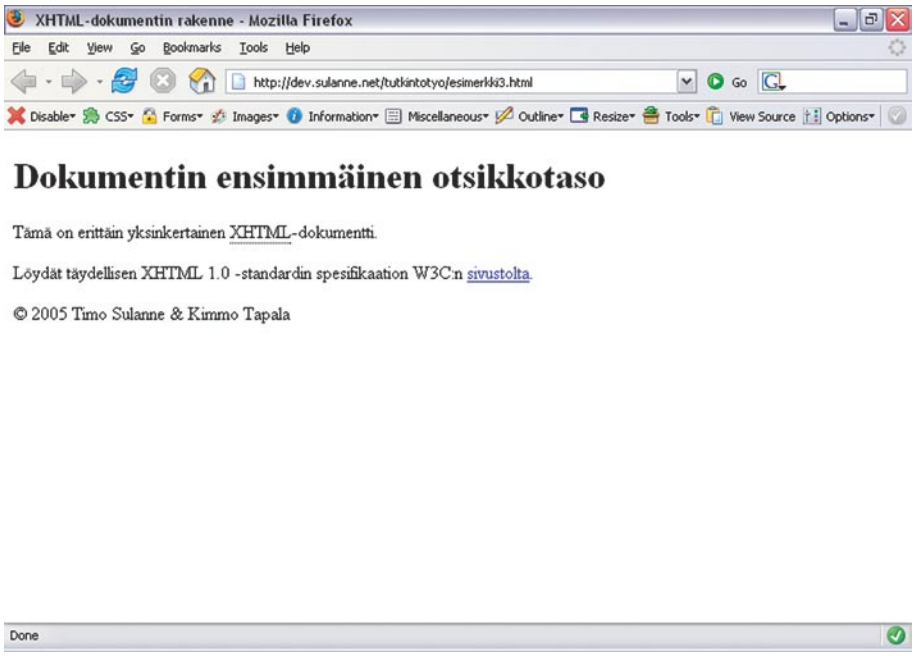
HTML-ja XHTML-dokumenttien suunnittelua helpottamaan on tehty monia eri kehitysympäristöjä. XHTML:n kannalta on kuitenkin ikävää, että useat näistä kehitysympäristöistä ovat ominaisuuksiltaan vielä vajavaisia tai muuten hankalia käyttää. XHTML-dokumenttien teossa alkuun pääseekin parhaiten modernilla selaimella (esimerkiksi Mozilla Firefox tai Opera 8) ja tekstieditorilla, joka pystyy tallentamaan ASCII-muotoista tekstiä. Nämä yksinkertaiset kehitysvälineet ovatkin suositeltavia varsinkin alkuvaiheessa, jotta suunnittelija oppisi käyttämään kaikkia XHTML:n elementtejä oikein ja tekemään samalla semanttisempaa sekä rakenteellisempaa merkkausta.

XHTML-dokumentti kostuu kolmesta osasta (esmerkki 3). Ensimmäisessä osassa määritellään dokumentin käyttämä dokumenttityyppi sekä nimiavaruus. Toisessa osassa eli head-elementissä määritetään dokumentin otsikkotiedot (title-elementti) sekä muut dokumentin ominaisuudet. Varsinainen dokumentin sisältö ja sen rakenne määritellään kolmannessa osassa eli body-elementissä.

Esimerkki 3. XHTML-dokumentin rakenne (kuva 4).

```
<!DOCTYPE html PUBLIC ”-//W3C//DTD XHTML 1.0 Strict//EN”
”http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml” xml:lang=”fi”
lang=”fi”>
<head>
  <title>XHTML-dokumentin rakenne</title>
  <meta http-equiv=”Content-Type” content=”text/html;
  charset=ISO-8859-1” />
</head>
<body>
  <!-- Tästä alkaa dokumentin varsinainen sisältö -->
  <h1>Dokumentin ensimmäinen otsikkotaso</h1>
  <p>
    Tämä on erittäin yksinkertainen
    <acronym title=”Extensible Hypertext Markup Language”>
    XHTML</acronym>-dokumentti.
  </p>
```

```
    <p>
      Löydät täydellisen XHTML 1.0 -standardin
      spesifikaation W3C:n <a href="http://www.w3.org"
      title="World Wide Web Consortium">sivustolta</a>.
    </p>
    <p>&copy; 2005 Timo Sulanne &amp; Kimmo Tapala</p>
</body>
</html>
```



Kuva 4. Yksinkertainen XHTML-dokumentti selaimessa.

4.7.1 Elementtien esitteleminen sekä sulkeminen

XHTML on HTML:n tapaan sisäkkäinen merkkauskieli, jossa sisältö sekä sen tulkitsemiseen tarkoitetut ohjeet kulkevat rinnakkain samassa dokumentissa. XHTML:ssä sisältö jäsennetään elementtien ja niitä ohjaavien attribuuttien avulla. Ensimmäinen kirjain tai sana < ja > -merkkien välissä ilmoittaa yleensä elementin tarkoituksen. Esimerkiksi `<p>` -tagin p tulee sanasta paragraph. Sillä sekä sen `</p>` -lopetustagilla merkatään tekstikappale. Muut sanat elementin sisällä ovat elementin attribuutteja ja niiden arvoja. Elementtien nimet tulee kirjoittaa aina pienillä kirjaimilla XML-syntaksin mukaisesti.

Osa elementeistä vaikuttaa vain tiettyyn osa-alueeseen dokumentista. Osa-alue alkaa kohdasta, missä elementti ja sen attribuutit mainitaan ensimmäistä kertaa, ja jatkuu kunnes elementtiä vastaava lopetustagi sulkee osa-alueen (elementtiä tarkoittavan sanan eteen lisätään kaut-

taviiva eli / -merkki). Lopetustagit eivät koskaan sisällä attribuutteja. Toisin kuin HTML:ssä, XHTML:ssä jokainen elementti tulee sulkea, mukaan lukien myös ns. tyhjät elementit, joita ovat HTML:ssä esimerkiksi `<br>` ja `<img>` -tagit. XHTML:ssä kyseiset elementit suljetaan lisäämällä välilyönti sekä kauttaviiva ennen elementin määrittelyn sulkevaa > -merkkiä. Edellä mainitut elementit siis suljetaan seuraavasti: `<br />` ja `<img />`. Elementit voidaan sulkea myös vaihtoehtoisesti: `<br></br>` ja `<img></img>`. Jälkimmäinen tapa on kuitenkin harvemmin käytetty.

XHTML:n tyhjät elementit:

- area
- base
- basefont (vanhentunut)
- br
- col
- frame
- hr
- img
- input
- isindex (vanhentunut)
- link
- meta
- param

4.7.2 Sisäkkäiset elementit

Elementtien sulkemisen lisäksi on erittäin tärkeää, että ne sijoitetaan oikein myös sisäkkäisesti. Perusajatuksena voidaan pitää sitä, että elementit tulee sulkea siinä järjestyksessä, missä ne on esitelty. Jos jokin elementti on toisen elementin sisällä, sisemmän elementin lopetustagi kirjoitetaan aina ennen ulomman elementin lopetustagia. XHTML-standardissa sisäkkäisille elementeille määriteltyjä rajoituksia ovat:

- a-elementti ei voi sisältää toista a-elementtiä;
- pre-elementti ei voi sisältää seuraavia elementtejä: img, object, big, small, sub tai sup;
- button-elementti ei voi sisältää seuraavia elementtejä: input, select, textarea, label, button, form, fieldset, iframe tai isindex;
- label-elementti ei voi sisältää toista label-elementtiä ja
- form-elementti ei voi sisältää toista form-elementtiä.

(Musciano & Kennedy 2002: 509)

4.7.3 Attribuuttien käsittely

XHTML-elementtien ominaisuudet määritellään attribuuteilla ja XHTML:n merkkikohtaisuus pätee myös niihin, joten niiden nimet tulee kirjoittaa elementtien tapaan pienellä. Toinen suuri muutos HTML:ään verrattuna on se, että attribuuttien arvot kirjoitetaan aina lainausmerkkien sisään. Esimerkki 3:ssa käytetyn acronym-elementin title-attribuutti antaa oikean merkityksen XHTML-lyhenteelle.

Jos elementille annetaan attribuutti, pitää attribuutille antaa myös aina jokin arvo. HTML:ssä saatettiin käyttää attribuuttimerkkauksessa myös niin kutsuttuja minimoituja attribuutteja (taulukko 1). Esimerkiksi radiobutton- tai checkbox-elementti voitiin HTML:ssä merkata valituksi pelkästään kirjoittamalla checked-attribuutti ilman mitään arvoa. XHTML:ssä minimoitujen attribuuttien arvot ovat samat kuin niiden nimet. Esimerkiksi radiobutton merkataan valituksi antamalla checked-attribuutille arvo ”checked”.

Taulukko 1. Minimoitujen attribuuttien merkkaaminen HTML 4.01:ssä sekä XHTML 1.0:ssa. (W3 Schools 2005.)

HTML 4.01	**XHTML 1.0**
compact	compact=”compact”
checked	checked=”checked”
declare	declare=”declare”
readonly	readonly=”readonly”
disabled	disabled=”disabled”
selected	selected=”selected”
defer	defer=”defer”
ismap	ismap=”ismap”
nohref	nohref=”nohref”
noshade	noshade=”noshade”
nowrap	nowrap=”nowrap”
multiple	multiple=”multiple”
noresize	noresize=”noresize”

4.7.4 Erikoismerkkien käyttäminen merkkauksessa

XHTML on erittäin tarkka erikoismerkkien käytön suhteen varsinkin jos JavaScriptiä ja CSS:ää sijoitetaan normaalin merkkauksen sekaan. Suositeltavaa onkin, että kaikki skriptit ja tyylimäärittelyt olisivat erillisissä linkitetyissä tiedostoissa. Erikoismerkit ovat ongelmallisia myös itse XHTML-merkkauksessa, eritoten attribuuttien arvoissa. Jotta väl-

tettäisiin erikoismerkkien sekoittuminen merkkaukseen, ne on syytä korvata käyttäen ASCII-entiteettejä. Tällöin esimerkiksi hyvinkin yleinen & -merkki korvataan käyttäen `&` -entiteettiä. Myös merkkien < ja > käytössä on hyvä kiinnittää erityistä huomiota ja niiden korvaavat entiteetit ovat `<` ja `>`. Tarkempi entiteettilistaus löytyy esimerikiksi osoitteesta http://www.w3schools.com/tags/ref_entities.asp.

4.7.5 Kommentointi

Kommentointi on aina vapaaehtoista ja on täysin kehittäjän päätettävissä kommentoiko hän tekemäänsä merkkausta. Kommentointi on kuitenkin suositeltavaa ja se helpottaakin useassa tapauksessa dokumentin jäsentelyä. Hyvin kommentoitu lähdekoodi auttaa myös muita kehittäjiä ymmärtämään tehtyä merkkausta ja näin koko kehittäjäyhteisö voi hyötyä mahdollisista uusista innovaatioista. XHTML-merkkauksessa kommentointi kirjoitetaan tagien `<!--` ja `-->` väliin. Tagien väliin voi kirjoittaa määrättömästi tekstiä kunhan vain muistaa, että kommentti ei saa sisältää kahta peräkkäistä viivaa. Kommentoinnin kanssa on hyvä olla muutenkin varuillaan, sillä se näkyy kaikille, jotka haluavat katsoa sivuston lähdekoodia selaimesta.

4.7.6 Sisältö

Kaikki muu tieto, mikä ei kuulu osaksi elementtitagia, on dokumentin sisältöä eli tekstiä. Osa elementeistä määrittelee tekstille sen rakenteen, kuten esimerkiksi otsikot ja kappaleet. Loput elementit määrittelevät, kuinka sisältöä tulisi käsitellä. Kuvat, animaatiot, videoleikkeet ja muu multimedia ovat kyllä myös sisältöä, mutta ne eivät kuulu itse XHTML-dokumenttiin, vaan ne sijaitsevat aina erillisissä tiedostoissa. Multimediatiedostot sisällytetään osaksi XHTML-dokumenttia käyttäen niille erikseen määrättyjä elementtejä. Esimerkiksi kuva liitetään dokumenttiin img-elementillä seuraavasti:

```
<img src=”kuvatiedosto.jpg” alt=”Kuvan selitys” />
```

Tärkeimmät XHTML-dokumentin kirjoittamista koskevat säännöt:

- XHTML-merkkaus on merkkikohtaista, joten kaikki elementit ja niiden attribuutit (koskee myös JavaScriptiä käsitteleviä attribuutteja) tulee kirjoittaa pienillä kirjaimilla. Attribuuttien, lukuun ottamatta class ja id, arvoissa voi käyttää kuitenkin sekä pieniä että isoja kirjaimia.
- Kaikki attribuuttien arvot tulee kirjoittaa lainausmerkkien sisään. Lisäksi jokaisella käytetyllä attribuutilla tarvitsee olla myös jokin arvo.

- Kaikki elementit tulee sulkea. Niin sanotut ”tyhjät” elementit, kuten `<br />` ja `<img />`, tulee myös sulkea käyttäen välilyöntiä ja kauttamerkkiä.
- ASCII-entiteettejä tulee käyttää korvaamaan erikoismerkkejä, jotta vältetään niiden sekoittuminen itse XHTML-merkkaukseen. Esimerkiksi `&` on &-merkin entiteetti, `<` on < -merkin (pienempikuin) ja `>` on > -merkin (suurempikuin).
- Kehittäjän tulisi kirjoittaa mahdollisimman johdonmukaista eli semanttista merkkausta. Kirjoitettu merkkaus on hyvä sisentää ja kommentoida tarpeen mukaan.

4.8 Elementit

XHTML-standardin mukaan muodostetussa dokumentissa on rakenteen kannalta varsinaisesti ainoastaan neljä pakollista elementtiä: html, head, body ja title. Muuten kehittäjä päättää itse siitä, kuinka dokumentin jäsentelee ja mitä elementtejä siinä käyttää. XHTML-dokumentin alussa määritelty dokumenttityyppi antaa kylläkin tiettyjä reunaehtoja elementtien käytölle ja seuraavissa kappaleissa esiteltävien elementtien joukosta onkin jätetty pois ns. vanhentuneet elementit, joita ei enää tueta täysin kaikissa XHTML-standardin dokumenttityypeissä. Täydellinen elementtilistaus löytyy esimerkiksi osoitteesta http://www.w3schools.com/tags/default.asp.

XHTML 1.1:ssä elementit on jaoteltu eri moduuleihin. Vaikka XHTML 1.0 ei perustukaan vielä moduuleihin, niin sen elementtien kohdalla voidaan kuitenkin käyttää samaa jaottelua.

4.8.1 Rakenne

Body-, head-, html- ja title-elementit

Rakenteelliset elementit määrittelevät XHTML-dokumentin perusrakenteen.

4.8.2 Teksti

Otsikkoelementit: h1, h2, h3, h4, h5, h6
Lohkotason (block) elementit: address, blockquote, div, p, pre
Rivitason (inline) elementit: abbr, acronym, br, cite, code, dfn, em, kbd, q, samp, span, strong, var

Tekstielementit määrittelevät tekstin rakenteen, mutta elementtejä ei käytetä tekstin ulkoasun muotoiluun. Rakenteellisesti oikein merkatussa XHTML-dokumentissa nämä tekstielementit kirjoitetaan seuraavassa järjestyksessä: otsikko, lohkotaso, rivitaso. Lohkotason elementit voivat sisältää mitä tahansa elementtejä, mutta rivitason elementit voivat sisältää vain toisia rivitason elementtejä.

4.8.3 Hyperteksti

A-elementti

Hypertekstielementtiä käytetään lohkoelementtien sisällä rivielementtinä. A-elementti määrittelee hypertekstilinkin ja linkin kohde määritellään sen href-attribuutissa.

4.8.4 Listat

Dd-, dl-, dt-, li-, ol- ja ul-elementit

XHTML:ssä on kolme eri listatyyppiä: määrittelylista, järjestetty lista sekä järjestämätön lista. Kaikki listaelementit ovat lohkotason elementtejä.

4.8.5 Esitys

B-, big-, hr-, i-, small-, sub-, sup- ja tt-elementit

Tekstin esitykseen tarkoitetut elementit vaikuttavat vain tekstin ulkoasuun eikä niillä ole siis mitään rakenteellista merkitystä. Näistä elementeistä tulisi käyttää vain hr-, sub- ja sup -elementtejä. B- ja i-elementit tulisi korvata rakenteellisemmilla strong- ja em-elementeillä. Muiden elementtien sijasta tulisi käyttää CSS-muotoiluja.

Tästä ryhmästä hr-elementti on ainoa lohkotason elementti

4.8.6 Lomakkeet

Form-, input-, label-, option-, select- ja textarea-elementi

Lomake-elementeillä määritellään XHTML-dokumentin yhteydessä käytettävät lomakkeet ja niiden ominaisuudet. Form-elementti on

tämän ryhmän ainoa lohkotason elementti ja samalla juurielementti muille elementeille.

4.8.7 Taulukot

Caption-, col-, colgroup-, table-, tbody-, tfoot-, td-, thead-, th- ja tr-elementit

Taulukkoelementtejä käytetään taulukkojen määrittelyyn sekä esittämiseen. Taulukkoelementtejä tulisi käyttää kuitenkin vain tabulaarisen tiedon esittämiseen eikä sivuston ulkoasun muotoiluun.

4.8.8 Kuvat

Img-elementti

Img-elementillä voidaan sijoittaa kuva XHTML-dokumentin sisään. Elementin src-attribuutti määrittelee kuvatiedoston sijainnin. Alt-attribuutti antaa kuvalle vaihtoehtoisen tekstiselityksen selaimille, jotka eivät käytä kuvia. Selainten yleisesti tukemia kuvaformaatteja ovat tällä hetkellä GIF-, JPEG- ja PNG-formaatit.

4.8.9 Objektit

Object- ja param-elementit

Objektielementeillä liitetään mediaobjekteja (esimerkiksi Flash ja Quicktime) XHTML-dokumenttiin ja määritellään niiden käyttäytyminen.

4.8.10 Kehykset

Frameset-, frame- ja noframe-elementit

Kehyselementeillä määritellään kehyksien ominaisuudet Frames-dokumenttityypillä varustetussa dokumentissa. Frameset-elementti on juuritaso frame- ja noframe-elementeille.

Iframe-elementti

Iframe-elementillä voidaan upottaa kehys normaalin XHTML-dokumentin sisään. Iframe on rivitason elementti.

4.8.11 Metatiedot, tyylit, linkit, skriptit ja perusosoite

Meta-elementti

Meta-elementeillä määritellään XHTML-dokumenttia koskeva metatieto, kuten esimerkiksi kuvaus dokumentin sisällöstä ja sitä koskevat hakusanat.

Style-elementti

Style-elementillä sisällytetään tyylimäärittelyjä XHTML-dokumenttiin.

Link-elementti

Link-elementillä määritellään kahden toisiinsa linkitetyn tiedoston suhde.

Script-elementti

Script-elementti määrittää sivuston toiminnallisuutta ohjaavan skriptin, kuten esimerkiksi JavaScriptin.

Base-elementti

Base-elementillä määritellään XHTML-dokumentissa olevien linkkien URL (Uniform Resource Locator) -osoite.

4.9 Attribuutit

XHTML-elementtien ominaisuudet määritellään attribuuteilla ja niille annettavilla arvoilla. Tässä yhteydessä mainitut attribuutit ovat yhteisiä kaikille XHTML-elementeille. Elementtikohtaiset attribuutit ovat listattuna myös osoitteessa http://www.w3schools.com/tags/default.asp.

4.9.1 Keskeiset ominaisuudet

Id-attribuutti

Id-attribuutilla määritellään elementille yksilöity nimi. Tämä tarkoittaa sitä, että samaa arvoa ei voida antaa millekään toisella elementille saman dokumentin sisässä. Attribuutin arvo voi alkaa ainoastaan joko isolla tai pienellä kirjaimella (väliltä a-z), jonka jälkeen arvo voi sisältää kirjaimia (a-z), numeroita (0-9) ja erikoismerkkejä (”-”, ”_”, ”:” ja ”.”). Id-attribuuttia käytetään yleisimmin merkkaamaan dokumentin lohkoja tai otsikoita.

Class-attribuutti

Class-attribuutilla määritellään elementin kuuluminen joko yhteen tai useampaan luokkaan ja se antaa kehittäjälle mahdollisuuden määritellä elementin tyypin. Poiketen id-attribuutista, mikä tahansa elementti voi saada toisen elementin kanssa saman class-attribuutin arvon.

Style-attribuutti

Style-attribuutilla voidaan määritellä käytettävät tyylisäännöt yhdelle tai useammalle elementille. Useimmissa tapauksissa on kuitenkin suositeltavaa käyttää joko id- tai class-attribuuttia yhdessä CSS-tyylien kanssa.

Title-attribuutti

Title-attribuutilla määritellään elementille nimike ja sitä käytetään useimmiten a-, link-, img- ja object-elementtien kanssa.

4.9.2 Kansainväliset ominaisuudet

Lang-attribuutti

Lang-attribuutilla määritellään elementin attribuuttien sekä sisällön kieli. Yleisimmin dokumentin kieli määritellään nimiavaruuden yhteydessä html-elementissä. XML-syntaksin mukaisesti dokumentin kieli on hyvä määritellä myös xml:lang-attribuutilla.

Dir-attribuutti

Dir-attribuutilla määritellään elementin tekstin suunta. Dir-attribuuttia käytetään yleisimmin html-elementin kanssa, jolloin attribuutti vaikuttaa koko XHTML-dokumenttiin. Arvolla ”ltr” tekstin suunta on vasemmalta oikealle (XHTML-dokumentin vakioasetus) ja arvolla ”rtl” oikealta vasemmalle.

4.9.3 Näppäimistöominaisuudet

Accesskey-attribuutti

Accesskey-attribuutilla määritellään elementille näppäimistöpikakomento. Attribuutin arvo voi olla joko kirjain (a-z) tai numero (0-9). Attribuutilla merkittyä elementtiä kutsutaan selaimessa yleisimmin joko alt-näppäin + attribuutin arvo (PC) tai ctrl-näppäin + attribuutin arvo (Mac).

Tabindex-attribuutti

Tabindex-attribuutilla määritellään missä järjestyksessä dokumentissa siirrytään painamalla tabulaattorinäppäintä. Attribuutin arvo on numero.

5 CSS (Cascading Style Sheets)

5.1 CSS:n historia

Aikoinaan sanomalehtitaloissa päätoimittaja kävi läpi kaiken lehteen tulevan tekstimateriaalin ja merkitsi sivujen marginaaleihin taittajalle ohjeet siitä, miten mitkäkin tekstilohkot ja otsikot tulisi lehteen taittaa. Ohjeet sisälsivät kaiken tarvittavan tiedon taiton tekemistä varten: tekstityypin, tekstin muotoilut, marginaalit ja rajaukset. Niinpä olikin yleistä, että sivujen marginaalit olivat varsin täynnä kaikenlaisia merkintöjä ennen taittoa. Tämä lisäsi virheiden määrää ja lehden sivujen ulkoasu saattoi vaihdella sivukohtaisesti.

Näiden ongelmien ratkaisemiseksi päätettiin ryhtyä käyttämään erityisiä tyyliliuskoja. Tyyliliuskoihin voitiin merkitä samat tyylimerkinnät taittoa varten, mitä aikaisemmin merkittiin sivujen marginaaleihin. Tässä tekniikassa oli kuitenkin yksi ratkaiseva ero: nuo tyylimerkinnät oli jaoteltu otsikko- ja tekstilohkotyypeittäin. Päätoimittajan piti entisten tyylimerkintöjen sijaan merkitä ainoastaan, mitä otsikko- tai tekstilohkotyyppiä kulloinenkin käsiteltävä otsikko tai tekstilohko oli. Taittaja etsi taittovaiheessa tyyliliuskoista vastaavat tyylimäärittelyt ja näin lehden ulkoasu pystyttiin pitämään yhtenäisenä eikä samoja tyylimäärittelyitä tarvinnut kirjoittaa moneen kertaan. Jos taas haluttiin tehdä muutoksia lehden ulkoasuun, tarvitsi tehdä muutokset vain tyyliliuskoihin.

> *”Tämä hieno menetelmä [CSS] pohjautuu malliin, joka otettiin käyttöön satoja vuosia sitten: toimittajien ja latojien käyttämiin muistilappuihin, joissa määriteltiin käsikirjoituksen eri osien ulkoasu. Tänään samaa menetelmää sovelletaan tekstitiedostojen ja selainohjelmistojen väliseen viestintään. Esitystapa ja rakenne ovat voimallisessa yhteydessä toisiinsa, mutta kuitenkin erillisiä.” (Veen 2002: 25.)*

1990-luvulla selainsota merkitsi HTML:lle rappiota. Selainvalmistajat – Microsoft ja Netscape olivat tuolloin lähes ainoat – pyrkivät lisäämään kilpailuetua toisiinsa nähden lisäämällä semanttisen HTML:n sekaan sivujen ulkoasuun vaikuttavia elementtejä. Näin syntyivät esimerkiksi font- ja blink-elementit sekä valtava joukko erilaisia attribuutteja, joilla voitiin hallita dokumentin esitystapaa. HTML-sivujen tekijät huomasivat näissä uusissa ominaisuuksissa mahdollisuuden erottua kilpailijoista ja niinpä nämä ominaisuudet otettiin hetkeäkään epäröimättä käyttöön.

> *”Netscape ja myös juuri markkinoille tullut kilpailija Microsoft ryhtyivät lisäämään selainohjelmistoihinsa niin paljon uutta teknologiaa ja uusia muotoilutunnisteita kuin vain ikinä kykenivät. Uskomattoman lyhyen ajan sisällä Webistä oli yhtäkkiä tullut rehevä maisema, joka oli pullollaan uusia ajatuksia, uusia kokemuksia ja kokeiluja.” (Veen 2002: 13.)*

Tim Berners-Leen johtama World Wide Web Consortium kuitenkin huomasi näissä uusissa ominaisuuksissa huolestuttavia piirteitä: HTML oli suunniteltu kuvaamaan dokumenttien rakennetta ja sisältöä, ei esitystapaa. Nyt kuitenkin esimerkiksi font-elementti ei kuvannut ollenkaan sisältöä, vaan vaikutti ainoastaan siihen, miten sisältö esitetään selaimessa. Samaan aikaan, kun selainsota rehotti, W3C kehitti menetelmiä, joiden avulla HTML-dokumenttien esitystapaa voitaisiin tehokkaasti muokata. W3C:n työn uskottiin kuitenkin kestävän pitkään, eivätkä selainvalmistajat halunneet odottaa. (Veen 2002: 13.)

Koska W3C:n HTML-jaosto huomasi, että HTML oli selvästi jakautumassa kahtia Microsoftin ja Netscapen omiksi versioiksi, ryhdyttiin W3C:n toimintamallia muuttamaan. Aikaisemmin W3C oli tehnyt normatiivista standardointityötä, jossa valmiit määräykset saneltiin ylhäältä päin, mutta nyt nähtiin tarve ryhtyä tekemään enemmän deskriptiivistä työtä, jossa voitaisiin luoda yhtenäiset pelisäännöt alan tarpeiden mukaan. (Veen 2002: 14.)

Deskriptiivinen työ tuotti tulosta ja yritysten edustajien ja alan asiantuntijoiden kanssa muodostetut pienet viralliset työryhmät (editorial review boards) tekivät varsin vilkkaasti ehdotuksia standardien kehittämiseksi. W3C:n aikaisemmin julkaisema HTML-suosituksen versio 3 jäi käytännössä luonnosasteelle, eikä milloinkaan yleistynyt käytössä. Uudella menetelmällä tuotetut versiot 3.2 ja 4.0 sen sijaan yleistyivät nopeasti ja saivat vihdoinkin yhtenäisen tuen selainvalmistajilta. HTML-kieli oli kuitenkin kehityksessä muuttunut selkeästä dokumentin rakennetta kuvaavasta merkintäkielestä kokoelmaksi erilaisia ulkoasumäärittelyjä. Tätä tilannetta korjaamaan W3C oli tuottanut CSS-standardin. (Veen 2002: 15.)

5.2 CSS-tyylit

Tässä opinnäytetyössä keskitytään pääasiallisesti CSS 2.0 -standardiin ja sen spesifikaatioihin. Mukana on myös joitakin osia CSS 2.1:stä sekä vielä keskeneräisestä CSS 3:sta. Osiin, joissa kerrotaan muista kuin CSS 2.0 -standardista, on merkitty kulloinkin käsiteltävän standardin versionumero.

CSS-teknologia luotiin, koska nähtiin tarpeelliseksi erottaa dokumentin rakenne ja esitystapa. Käytettäessä CSS:ää dokumentin visuaaliseen muotoiluun, voidaan jättää (X)HTML:lle ainoastaan dokumentin rakenteen kuvaaminen. Tämä osaltaan helpottaa myös sivustojen päivitystä, mutta oikein käytettynä myös vähentää palvelinten kuormitusta ja tietoverkon kaistankäyttöä.

CSS-lyhenne tulee sanoista Cascading Style Sheets. Näissä sanoissa pääpaino on ehdottomasti sanalla ”Cascading”. Tällä tarkoitetaan portaattaisesti etenevää muotoilua, ja juuri porrastus tekee CSS:stä hyvin monikäyttöisen muotoilutyökalun. CSS-muotoilussa käytetään pääpiirteittäin neljää porrasta, jotka määräävät lopulta (X)HTML-dokumentissa kuvatun elementin esitystavan:

1. Etsitään kaikki tyylimäärittelyt, jotka pätevät määrättyyn elementtiin ja elementin ominaisuuteen määrätyllä mediatyypillä.
2. Järjestetään tyylimäärittelyt lähteen ja painotuksen mukaan. Lähteet järjestettynä tärkeimmästä vähäisimpään ovat dokumentin kirjoittajan määrittelemät tyylit (ns. inline-tyylit), käyttäjän määrittelemät tyylit (ulkoiset tyylitiedostot) ja selaimen omat vakiotyylit. Määrittelyiden painotusta voidaan muuttaa !important-määreellä.
3. Järjestetään tyylimäärittelyt selektorin (valitsimen) tarkkuuden mukaan. Mitä tarkemmin selektori määrittelee jonkin elementin, sitä tärkeämpiä nuo määrittelyt ovat.
4. Käytetään hyödyksi tyylimäärittelyiden järjestystä tyylitiedostoissa tai dokumentissa. Mikäli kaksi tyylimäärittelyä ovat yhtä vahvat painotukseltaan ja viittaavat samaan elementtiin, myöhemmin esiintyvät määrittelyt jäävät voimaan.

   (Meyer 2001: 8.)

5.3 CSS:n käyttö

CSS:ää voidaan käyttää joko dokumentin ulkoisista tiedostoista tai sisäisesti osana dokumenttia. Ulkoisia tyylitiedostoja kutsutaan käyttäjän määrittelemiksi tyyleiksi (user styles) ja sisäisiä tyylimäärittelyjä dokumentin kirjoittajan määrittelemiksi tyyleiksi (author styles).

Sisäisiä tyylimäärittelyjä voidaan liittää dokumenttiin kahdella tavalla: lisäämällä dokumentin head-osaan style-elementti, jossa määritellään dokumentin elementtien tyylit samalla tavalla kuin se tehtäisiin ulkoisessa tyylitiedostossa tai käyttämällä (X)HTML-elementtien style-attribuuttia.

Style-elementti ei sovellu kovinkaan hyvin kokonaisen sivuston tyylimäärittelyiden tekemiseen, sillä jokaisella sivulla on tällöin oltava omat tyylimäärittelynsä. CSS:n suurin hyöty jää näin saavuttamatta. Sivuston ulkoasun suunnitteluvaiheessa style-elementti kuitenkin on hyvä apuväline, koska käytettäessä style-elementtiä dokumentin head-osassa voidaan helposti muokata samassa tiedostossa sekä dokumentin rakennetta että esitystapaa. Style-elementti kirjoitetaan dokumentin head-osaan seuraavasti:

```
<style type="text/css">
   ...
</style>
```

Edelläolevaan esimerkkiin tyylimäärittelyt tulisivat kolmen pisteen tilalle.

Mikäli halutaan tehdä tilapäisiä poikkeuksia CSS-tyylimäärittelyihin tai muokata vain jotakin tiettyä elementtiä vain yhdessä dokumentissa, voidaan käyttää style-attribuuttia. Koska style-attribuutti on järjestyksessä viimeinen luettava CSS-määrittely, jäävät usein siinä määritellyt ominaisuudet voimaan porrastussääntöjen mukaisesti. Tämä voidaan vielä varmistaa käyttämällä määritellyille ominaisuuksille !important-määrettä, jolloin määrittelyt varmasti saavat suurimman mahdollisen painoarvon porrastuksessa. Mikäli halutaan kuitenkin tehdä samanlaisia muotoiluja useammalle elementille, on järkevää käyttää jotakin muuta määrittelytapaa. Seuraavassa esimerkki, jossa p-elementin tekstin väri (piirtoväri) muutetaan vihreäksi:

```
<p style="color: green;">Tämä teksti on vihreää.</p>
```

Edelläolevassa esimerkissä useampia ominaisuuksia voitaisiin määritellä samaan attribuuttiin puolipisteillä eroteltuina.

Myös ulkoisia tyylitiedostoja voidaan lisätä dokumenttiin kahdella tavalla: käyttämällä dokumentin head-osassa joko pelkästään link-elementtiä tai style-elementtiä @import-säännön kanssa. Koska ulkoiset tyylitiedostot ovat järkevin tapa käyttää CSS:ää dokumenttien tyylien määrittelyyn, keskitytään tässä opinnäytetyössä pääasiallisesti juuri niihin.

Link-elementillä pystytään liittämään dokumentteihin paljon erilaisia ulkoisia tiedostoja. CSS-tyylit liitetään seuraavanlaisella XHTML-link-elementillä:

```
<link rel="stylesheet" type="text/css" href="css-
tiedosto.css" media="screen" />
```

Mikäli kyseessä olisi HTML-dokumentti, pitäisi W3C:n spesifikaatioiden mukaisesti jättää elementin päättävä /-merkki pois.

Käytettäessä style-elementtiä ulkoisten tyylitiedostojen liittämiseen dokumenttiin, tarvitaan CSS:n @import-sääntöä. Koska @import-sääntö on osa CSS:ää, se toimii myös ulkoisissa tyylitiedostoissa hyvänä keinona liittää useampia tiedostoja samaan dokumenttiin. Style-elementillä ulkoisen tyylitiedoston liittäminen dokumenttiin tapahtuu seuraavasti:

```
<style type="text/css">
  @import url(css-tiedosto.css) screen;
</style>
```

Style-elementti siis on aivan samanlainen kuin määriteltäessä sisäisiäkin tyylejä, mutta tyylimäärittelyjen asemesta elementin sisällä on @import-sääntö, joka viittaa ulkoiseen tyylitiedostoon. @import-säännön lopussa on määritelty, että kyseinen tyylitiedosto ladataan vain screen-mediatyyppiä käytettäessä, eli jos dokumenttia tarkastellaan tietokoneen monitorilla.

Ulkoisia tyylitiedostoja liitettäessä dokumenttiin pystytään paitsi vähentämään verkon ja palvelinten kuormitusta, myös valikoimaan selaimia, joille tietyt tyylimäärittelyt näytetään. Netscape-selaimen 4-versiot lataavat CSS-tiedostot, mikäli ne on liitetty dokumenttiin link-elementillä. Ikävä kyllä selaimen CSS-moottori on niin pahasti puutteellinen, että useimmiten CSS vain sotkee dokumentin ulkoasun. Niinpä onkin järkevää käyttää style-elementtiä ja @import-sääntöä ulkoisten tyylitiedostojen liittämiseen, mikäli käytetään sellaisia CSS-määrittelyjä, joita Netscapen vanhemmat versiot eivät osaa tulkita. Tällöin tyylitiedostot jäävät näillä selaimilla lataamatta ja dokumentin käytettävyys ei kärsi. Microsoft Internet Explorer puolestaan jättää huomiotta kaikki tyylitiedostot, jotka on liitetty dokumenttiin style-elementtiä käyttäen, jos @import-säännössä on määritelty mediatyyppi. Edellisessä esimerkissä style-elementillä ja @import-säännöllä liitetty tyylitiedosto siis olisi jäänyt lataamatta Netscape-selaimen 4-versioilla sekä Microsoft Internet Explorerilla. Internet Explorerille mediatyyppi voitaisiin kuitenkin määritellä style-elementin media-attribuutilla.

Tätä tietoa sekä CSS:n porrastussääntöjä hyväksikäyttäen pystytään rakentamaan yksinkertainen valintajärjestelmä, jolla voidaan ilman epäluotettavia selaintunnistuksia määrittää, mitkä tyylitiedostot näytetään millekin selaimelle. Rakennettaessa kuitenkin tällaisia valintajärjestelmiä, tulee muistaa, että sivustolla pyritään aina antamaan käyttäjälle mahdollisimman hyvä käyttäjäkokemus. Kannattaa toki

myös ottaa huomioon, että useiden CSS-tiedostojen lataaminen ja hallinta kasvattavat niin ylläpito- ja päivitysvaatimuksia kuin palvelin- ja verkkokuormitustakin.

5.4 Tyylien periytyminen

Useimmat tyylit voivat periytyä lapsielementeille. Tämä tarkoittaa sitä, että jos esimerkiksi body-elementille on määritelty CSS-tiedostossa piirtoväriksi musta, näkyvät body-elementin sisällä olevat lapsielementtien tekstit myös mustina. Tämä periytyminen ei kuitenkaan koske esimerkiksi linkkitekstejä, jotka silti esitetään selaimen vakiotyyleillä, vaan niille pitää määritellä omat tyylinsä. Kaikille tyylimäärittelyiden ominaisuuksille (lukuun ottamatta page-ominaisuutta) voidaan antaa arvoksi inherit, jolloin lapsielementti perii ominaisuuden arvon vanhemmaltaan. (Meyer 2001: 10.)

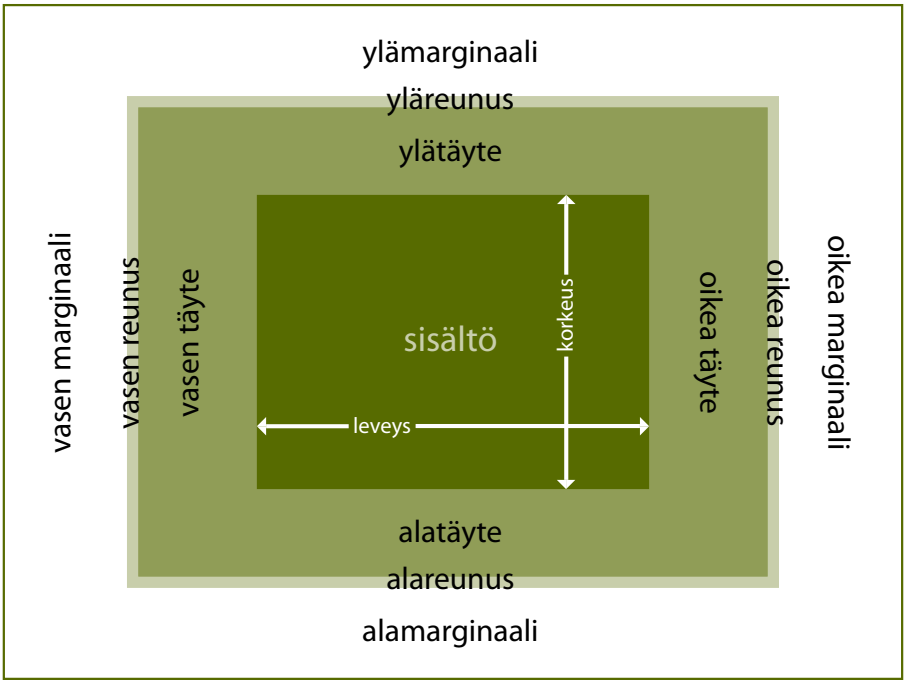
Esimerkki usein käytetystä tyylien periytymisestä:

```
body
{
  font-family: sans-serif;
}
```

Edellä annettu esimerkki asettaa kaikki body-elementin sisällä olevat elementit käyttämään sans-serif-tyyppistä kirjasinta. Tässä tapauksessa myös linkit perivät ominaisuuden ja muuttuvat. Koska periytyminen ei toimi ilman inherit-arvoa kaikkien ominaisuuksien ja elementtien kohdalla samalla tavalla, tarvitaan tyylien periytymisen kanssa tarkkuutta.

5.5 CSS:n laatikkomalli

CSS:n toimintaperiaate perustuu hyvin pitkälti laatikkomalliin (kuva 5). Laatikkomalli määrää, missä järjestyksessä lohkoelementin eri osat ovat visuaalisessa esityksessä. Elementit koostuvat useasta eri osasta, joiden visuaalisiin ominaisuuksiin CSS:llä voidaan vaikuttaa. Nämä osat ovat itse sisältö (content), täyte (padding), reunus (border) ja marginaali (margin).



Kuva 5. CSS:n laatikkomalli.

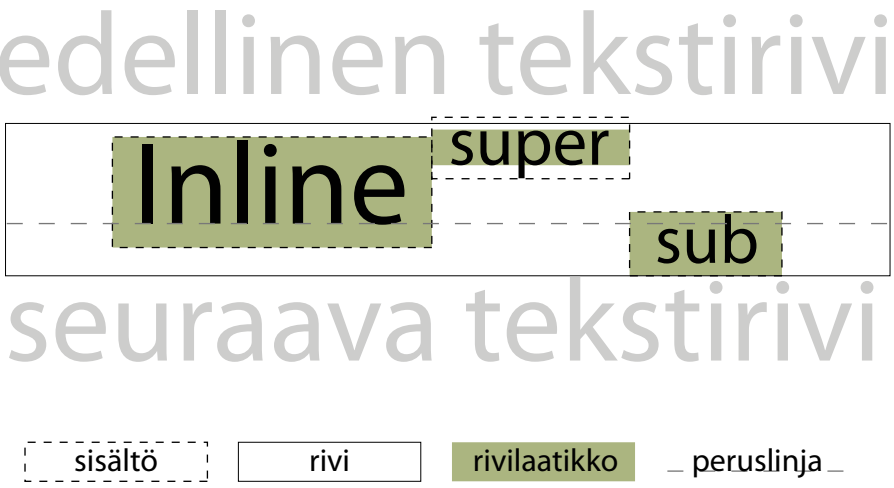
Ikävä kyllä, laatikkomalli ei tämän opinnäytetyön kirjoitushetkellä ole kaikissa selaimissa samanlainen. Etenkin Microsoft Internet Explorer käsittelee täytteen (padding) ja reunuksen (border) eri tavalla kuin valtaosa CSS:ää tukevista selaimista. Tämä on selvä osoitus CSS-standardin puuttellisesta tuesta vielä tälläkin hetkellä selainrintamalla. Tulevassa Internet Explorer 7.0:ssa on luvattu parantaa CSS 2.0:n tukea huomattavasti, mutta koska Internet Explorer on edelleen vallitseva selain markkinoilla, ei selaimen CSS-moottoriin voida tehdä liian suuria muutoksia. Suuret muutokset CSS-moottoriin saattaisivat vaarantaa yhteensopivuuden aikaisemmille selainversioille tehtyjen sivujen kanssa.

5.6 CSS:n rivimalli

Laatikkomallin ohella toinen tärkeä elementtimalli on rivimalli (kuva 6). Rivimalli toimii hieman laatikkomallin tapaan, mutta nimensä mukaisesti rivimalli koskee rivielementtejä. Rivimalli kertoo, mihin kohtaa riviä sisältö sijoitetaan ja kuinka korkea rivistä tulee. Rivimallin mukaisesti yhden rivielementin korkeus muodostuu seuraavista osista:

- Sisällön korkeus: itse tekstin korkeus, joka määräytyy kirjasimen koon mukaan.
- Puoliriviväli: laskennallisesti (kirjasimen korkeus - rivin korkeus) / 2.

- Rivilaatikko: laatikko, joka saadaan, kun vähennetään puoliriviväli sisällön ylä- ja alapuolelta. Jokaisella tekstiosasella rivilaatikon korkeus on sama kuin rivin korkeus.



Kuva 6. CSS:n rivimalli.

5.7 CSS-sääntöjen rakenne

Yksi CSS-sääntö määrittelee, miten jokin tietyt kriteerit täyttävä elementti tulee esittää. CSS-säännöt koostuvat kolmesta pääosasta: selektorista eli valitsimesta (selector), ominaisuudesta (property) ja arvosta (value). Ominaisuuden ja arvon muodostamaa kokonaisuutta sanotaan julistukseksi (declaration). Koska yhdessä säännössä voi olla monia julistuksia, käytetään CSS:ssä lohkoihin perustuvaa merkintätapaa, jossa selektorin jälkeen seuraavat julistukset kirjoitetaan aaltosulkujen sisään. CSS-sääntö, jossa on vain yksi julistus, voisi siis olla esimerkiksi seuraavanlainen:

```
p {color: #a00}
```

Tässä esimerkissä selektorina on ”p”, ominaisuutena ”color” ja arvona ”#a00”. Tämä siis tarkoittaisi sitä, että dokumentin kaikki p-elementin sisällä olevat tekstit olisivat punaisia. Koska CSS ei piittaa rivinvaihdoista eikä välilyönneistä, voidaan säännöt kirjoittaa usealle riville sisennettyinä. Tämä onkin käytännöllistä, kun kirjoitetaan sääntöjä, joissa on useampia julistuksia. Useita julistuksia voitaisiinkin kirjoittaa vaikkapa seuraavasti:

```
p
{
  color: #a00;
  font-size: 0.8em;
}
```

Tässä esimerkissä määritellään värin lisäksi dokumentin kaikkien p-elementtien tekstin koko hieman oletuskokoa pienemmäksi. On huomattava, että samassa säännössä olevat julistukset erotellaan puolipisteellä. Säännön viimeiset julistukset eivät tarvitse päättävää puolipistettä, jota kuitenkin on järkevää käyttää jokaisen julistuksen perässä, sillä tällöin voidaan lisätä uusia julistuksia suoraan edellisten jälkeen. Tämä myös mahdollistaa julistusten uudelleenjärjestelemisen suuremmiksi kokonaisuuksiksi.

5.8 Selektorit eli valitsimet

Selektoreilla määrätään, mihin elementtiin tai elementteihin julistukset vaikuttavat. Selektoreita on useita erilaisia ja juuri tuetuissa selektoreissa ovat suurimmat erot eri CSS-versioiden välillä. Selektoreiden käsittelyssä on myös eroja eri selainten välillä. Tällä hetkellä onkin turvallisinta käyttää ainoastaan CSS 2.0 -standardin mukaisia selektoreita. (Meyer 2001: 43.)

5.8.1 Tyyppiselektorit

Tyyppiselektorit valitsevat kaikki määrättyä tyyppiä olevat elementit. Esimerkkejä tyyppiselektorin käytöstä:

```
h1
{
  font-family: sans-serif;
  font-size: 1.6em;
}

p
{
  font-family: serif;
  font-size: 0.8em;
}
```

```
img
{
  border: 1px solid #000;
}
```

Esimerkeissä ensimmäinen muotoilee kaikki dokumentissa käytetyt h1-elementit, seuraava p-elementit ja viimeinen img-elementit. (Meyer 2001: 43.)

5.8.2 Jälkeläisselektori

Jälkeläisselektoria käytetään valitsemaan elementit, jotka ovat joidenkin toisten elementtien jälkeläisiä. Jälkeläisselektorit koostuvat vähintään kahdesta selektorista, jotka kirjoitetaan hierarkkisessa järjestyksessä vanhimmasta nuorimpaan välilyönneillä eroteltuina. Jälkeläisselektoria voitaisiin periaatteessa käyttää hyvinkin monimutkaisten hierarkioiden kanssa, mutta syvempien hierarkioiden kirjoittamisen asemesta on järkevämpää käyttää jotakin paremmin yksilöivää selektoria. Jälkeläisselektori yhdistettynä muihin selektoreihin mahdollistaa kuitenkin hyvin joustavien valintojen tekemisen. Jälkeläisselektoria käytetään seuraavasti:

```
p strong
{
  color: #006;
  text-decoration: underline;
}

#wrapper span.description acronym
{
  font-weight: 900;
  color: #8a0;
}
```

Ensimmäinen esimerkkisääntö valitsee kaikkien p-elementtien sisällä olevat strong-elementit ja jälkimmäinen kaikki acronym-elementit, jotka ovat luokkaa ”description” olevien span-elementtien sisällä, jotka puolestaan ovat nimetyn ”wrapper”-elementin sisällä. On huomioitava, että elementtien ei ole välttämättä oltava suoraan määrättyjen vanhempien lapsia, vaan välissä voi olla määrittelemätön määrä elementtihierarkiaa. (Meyer 2001: 44.)

5.8.3 Yleisselektori

Yleisselektori valitsee kaikki määrätyt elementit. Yleisselektorin käyttö rajoittuu lähinnä hyvin yleisluontoisiin valintoihin tai yhdistelmiin toisten selektoreiden kanssa. Yleisselektori merkitään *-merkillä, jota varsin yleisesti käytetään ertilaisissa tietojärjestelmissä ns. villinä merkkinä (wild card). Esimerkkejä yleisselektorin käytöstä:

```
*
{
  padding: 0;
  margin: 0;
}

#container *
{
  border: 1px solid #44e;
}
```

Esimerkkien ensimmäinen sääntö valitsee dokumentin kaikki elementit ja poistaa niiltä selaimen vakiotyyleissä olevan täytteen ja marginaalin. Tämä on hyvin yleinen käyttötapaus yleisselektorille, sillä monet suunnittelijat haluavat lähteä suunnittelussaan alkuun tilanteesta, jossa kaikki ”ylimääräinen” tila elementtien sisällön ympäriltä on poistettu. Tätä sääntöä kutsutaan nimellä ”global whitespace reset”. Jälkimmäisessä esimerkkisäännössä puolestaan valitaan kaikki elementit, jotka ovat nimetyn ”container”-elementin sisällä ja niille asetetaan yhden pikselin paksuinen sininen reunusviiva. Mikäli CSS-määrittelyissä jollakin säännöllä ei ole ollenkaan selektoria, käytetään oletuksena yleisselektoria. (Meyer 2001: 44.)

5.8.4 Lapsiselektori

Lapsiselektoria käytetään valitsemaan jonkin elementin lapsielementtejä. Jälkeläisselektori ja lapsiselektori eroavat toisistaan siten, että jälkeläisselektoria käytettäessä vanhemman ja jälkeläisen välissä voi olla määrittelemätön määrä elementtihierarkiaa, mutta lapsiselektoria käytettäessä lapsielementin on oltava suoraan vanhemman lapsi. Lapsiselektori merkitään kirjoittamalla > -merkki vanhemman ja lapsen väliin. Esimerkki lapsiselektorin käytöstä:

```
body > h2
{
  color: #882;
}
```

Esimerkissä valitaan kaikki h2-elementit, jotka ovat suoraan lapsia body-elementille ja asetetaan niiden piirtoväriksi hieman kellertävä väri. H2-elementit, jotka ovat esimerkiksi body-elementin lapsenlapsia jäävät siis valitsematta. (Meyer 2001: 45.)

5.8.5 ”Seuraava sisarus” -selektori

Nimensä mukaisesti ”seuraava sisarus” -selektori valitsee määrätyn elementin, joka seuraa välittömästi toista määrättyä elementtiä samalla tasolla hierarkiassa. ”Seuraava sisarus” -selektorin merkkinä toimii + -merkki, joka kirjoitetaan etummaisen ja jälkimmäisen elementin valitsevan selektorin väliin. ”Seuraava sisarus” -selektoria voitaisiin käyttää seuraavasti:

```
h1 + p.ingressi
{
  background-color: #ddd;
}
```

Esimerkissä valittaisiin kaikki luokan ”ingressi” p-elementit, jotka seuraavat välittömästi h1-elementtiä. Mikäli esimerkiksi h1-elementtiä seuraisi ensin vaikkapa h2-elementti, ei haluttu luokan ”ingressi” p-elementti enää olisi välittömästi seuraava h1-elementin sisarelementti ja jäisi näin valitsematta. Seuraava sisarus -selektorin käytössä on huomioitava, että jotkin hieman vanhemmat selaimet eivät osaa suorittaa valintaa oikein, vaan jättävät täysin huomiotta selektorin alkuosan (esimerkissä ”h1 +”) ja suorittavat valinnan pelkän loppuosan mukaan (esimerkissä p.ingressi). On myös huomattava, että selektori jättää huomiotta kaiken elementtien välissä olevan tekstin, joten esimerkissä h1 ja p-elementtien välissä voisi olla kirjoitettuna tekstiä, mutta ei ainoatakaan elementtiä. (Meyer 2001: 45.)

5.8.6 Attribuuttiselektorit

Attribuuttiselektoreilla suoritetaan elementtien valinta perustuen elementtien määrättyjen attribuuttien olemassaoloon ja arvoihin. Attribuuttiselektoreita on CSS 2.0 -standardissa neljä erilaista:

1. elementti[attribuutti]
   – Valitsee kaikki elementit, joilla on määritelty attribuutti.
2. elementti[attribuutti=”arvo”]
   – Valitsee kaikki elementit, joilla on määritellyllä attribuutilla määritelty arvo. Arvot voivat sisältää välilyöntejä ja erikoismerkkejä.

3. elementti[attribuutti~=”arvo”]
   – Valitsee kaikki elementit, joiden määritelty attribuutti koostuu listasta välilyönnillä eroteltuja arvoja ja sisältää määritellyn arvon.
4. elementti[attribuutti|=”arvo”]
   – Valitsee kaikki elementit, joilla määritelty attribuutti sisältää määritellyllä arvolla alkavan tavuviivalla erotetun arvon. Tämä attribuuttiselektori on tarkoitettu erityisesti kieliattribuuttien perusteella tehtävien valintojen tekemiseksi.

Attribuuttiselektorit mahdollistavat hyvin monipuolisten elementtivalintojen tekemisen. Yksinkertaiset esimerkit CSS 2.0 -standardin attribuuttiselektoreiden käytöstä:

```
img[alt]
{
  border: 1px dashed #000;
}

img[alt=”Piirretty kuvio sillan rakenteesta”]
{
  border: 4px solid #f00;
}

img[alt~=”kuvio”]
{
  border: 2px dotted #0ff;
}

*[lang|=”en”]
{
  font-style: italic;
}
```

Ensimmäinen sääntö valitsee kaikki img-elementit, joilla on alt-attribuutti ja tekee niille yhden pikselin paksuisen, mustan katkoviivareunuksen. Toinen sääntö valitsee kaikki img-elementit, joilla on alt-attribuutin arvona teksti ”Piirretty kuvio sillan rakenteesta” ja tekee niille neljän pikselin paksuisen, punaisen reunusviivan. Kolmas esimerkkisääntö valitsee dokumentin kaikki img-elementit, joiden alt-attribuutin arvossa esiintyy sana ”kuvio” ja tekee niille kahden pikselin paksuisen, sinivihreän pistereunuksen. Viimeinen esimerkkisääntö valitsee kaikki elementit, joilla on lang-attribuutilla ”en”-alkuinen arvo siten, että attribuutin arvo voi olla joko pelkästään ”en” tai se voi sisältää alun ”en” erotettuna arvon loppuosasta tavuviivalla, ja kursivoi elementtien sisältämän tekstin. Viimeisessä esimerkissä siis valittaisiin elementit,

joilla olisi lang-attribuutilla arvona esimerkiksi ”en”, ”en-uk” tai ”en-us”, mutta ei valittaisi elementtejä, joilla olisi lang-attribuutilla arvona esimerkiksi ”eng” tai ”english”. (Meyer 2001: 46.)

Näiden CSS 2.0 -standardin mukaisten attribuuttiselektoreiden lisäksi on joissakin selaimissa myös tuettuna tulevan CSS 3.0 -standardin mukaisia attribuuttiselektoreita. CSS 3.0 tuo mukanaan ainakin kolme uutta attribuuttiselektoria:

1. elementti[attribuutti^=”arvo”]
   - Valitsee kaikki elementit, joiden määritelty attribuutti alkaa määritellyllä arvolla.
2. elementti[attribuutti$=”arvo”]
   - Valitsee kaikki elementit, joiden määritelty attribuutti päättyy määritellyllä arvolla.
3. elementti[attribuutti*=”arvo”]
   - Valitsee kaikki elementit, joiden määritelty attribuutti sisältää määritellyn arvon.

Koska CSS 3.0 -standardi ei vielä tätä opinnäytetyötä kirjoitettaessa ole valmis, ei ole varmuutta, muuttuvatko nämä attribuuttiselektorit uuden standardiversion kehityksessä jotenkin.

5.8.7 Luokkaselektorit

Luokkaselektoreilla valitaan elementtejä jostakin tietystä luokasta. Elementin luokka määrätään merkkauksessa class-attribuutilla. Luokkaselektoreita voidaan käyttää kolmella tavalla:

1. elementti.luokka
   - Valitsee kaikki määritellyt elementit, joilla on class-attribuutilla arvona määritelty luokka.
2. elementti.luokka1.luokka2...
   - Valitsee kaikki määritellyt elementit, joiden class-attribuutilla on arvona välilyönneillä eroteltu lista arvoista, joihin sisältyvät määritellyt arvot missä tahansa järjestyksessä.
3. .luokka
   - Valitsee kaikki elementit, joilla on class-attribuutilla arvona määritelty luokka.

Luokkaselektoria käytetään, kun halutaan muotoilla elementtejä, joilla on useita ilmentymiä dokumentissa. Seuraavassa esimerkkejä luokkaselektorin käytöstä:

```
p.ingressi
{
  text-transform: uppercase;
}

p.huomautus.sisalto
{
  background-color: #ffa;
}

.sivumerkinta
{
  background-color: #777;
  color: #fff;
}
```

Ensimmäinen esimerkki valitsee kaikki p-elementit, joilla on class-attribuutilla arvona ”ingressi” ja muuttaa ne kokonaan isoilla kirjaimilla kirjoitetuksi. Toinen esimerkki valitsee kaikki p-elementit, joiden class-attribuutti sisältää arvot ”huomautus” ja ”sisalto” missä tahansa järjestyksessä. Toinen esimerkki siis valitsisi elementit, jotka olisi merkattu dokumenttiin vaikkapa seuraavasti:

```
<p class=”huomautus sisalto”></p>
<p class=”sisalto alaosa huomautus”></p>
```

Valitsematta kuitenkin jätettäisiin elementit, joissa jokin selektoriin määritellyistä class-attribuutin arvoista puuttuisi. Kolmas esimerkki valitsee kaikki elementit, joilla on class-attribuutilla arvona ”sivumerkinta”. (Meyer 2001: 48.)

5.8.8 Id-selektorit

Id-selektorit toimivat hyvin samalla tavalla kuin luokkaselektorit. Elementin id määrätään merkkauksessa id-attribuutilla. Koska elementin id-attribuutti on yksilöivä, voi yksi id olla ainoastaan yhdellä elementillä. Id-selektoria voidaan käyttää kahdella tavalla:

1. elementti#id
   - Valitsee kaikki määritellyt elementit, joilla on id-attribuutilla arvona määritelty arvo.
2. #id
   - Valitsee kaikki elementit, joilla on id-attribuutilla arvona määritelty arvo.

Id-selektoria siis käytetään, kun halutaan valita vain yksi elementti. Useimmat selaimet osaavat kyllä muotoilla useampiakin samalla id:llä dokumentissa olevia elementtejä, mutta tämä on vastoin HTML- ja XHTML-spesifikaatioita. Seuraavassa esimerkkejä id-selektorin käytöstä:

```
div#header
{
  border: 1px solid #000;
  background-color: #666;
}

#sidebar
{
  background-color: #069;
}
```

Ensimmäinen esimerkki valitsee div-elementin, jolla on id-attribuutilla arvo ”header” ja tekee sille yhden pikselin paksuisen, mustan reunaviivan sekä vaihtaa taustaväriksi tummahkon harmaan. Toinen esimerkki valitsee elementin, jolla on id-attribuutilla arvo ”sidebar” ja vaihtaa sen taustavärin sinivihreäksi. (Meyer 2001: 49.)

5.8.9 Pseudoluokat

Pseudoluokat ovat luokkaselektoreita, jotka selain itse määrää dokumentin käytön aikana. Pseudoluokilla pystytään muuttamaan sellaisia luokkia, jotka eivät esiinny varsinaisessa dokumentin merkkauksessa ollenkaan. Tällaisia luokkia ovat esimerkiksi linkkien erilaiset tilat. Pseudoluokat toimivat hyvin samalla tavalla kuin normaalit luokkaselektoritkin. (Meyer 2001: 49.)

:first-child

:first-child-selektori valitsee elementin, joka on vanhempansa ensimmäinen lapsi. Myöskään :first-child-selektori ei välitä ennen lapsielementtiä olevasta tekstisisällöstä. Esimerkki selektorin toiminnasta:

(X)HTML-merkkaus:

```
<p>
  Tässä on tekstiä, jossa on <em>painotettua
  tekstiä</em> sekä hieman lisää <em>painotettua
  tekstiä</em>.
</p>
```

CSS-sääntö:

```
em:first-child
{
   color: #f00;
}
```

Edeltävästä (X)HTML-merkkauksesta valittaisiin annetulla CSS-säännöllä ensimmäinen em-elementti ja vaihdettaisiin sen väri punaiseksi, mutta toinen em-elementti jätettäisiin muotoilematta. (Meyer 2001: 50.)

:link

:link-selektori valitsee linkkielementin, joka viittaa vierailemattomaan osoitteeseen. Ainoastaan a-elementti, jolla on href-attribuutti voi saada :link-pseudoluokan. :link-pseudoluokka on :visited-pseudoluokan kanssa toisensa poissulkevia. Esimerkki :link-pseudoluokan käytöstä:

```
a:link
{
   color: #ff0;
   text-decoration: underline;
}
```

Esimerkki valitsee kaikki a-elementit, joilla on href-attribuutti, joka osoittaa vierailemattomaan osoitteeseen, ja muuttaa niiden väriksi keltaisen sekä alleviivaa ne. (Meyer 2001: 51.)

:visited

:visited-pseudoluokka sisältää linkkielementit, jotka viittaavat vierailtuihin osoitteisiin. Käytöltään :visited vastaa täysin :link-pseudoluokkaa. (Meyer 2001: 51.)

:hover

:hover-pseudoluokkaan kuuluvat elementit, joita käyttäjä osoittaa. Vaikka :hover-pseudoluokkaan voi kuulua mikä tahansa elementti, tekevät useat selaimet :hover-pseudoluokan ainoastaan linkkielementeille. :hover-pseudoluokkaa ei myöskään esiinny ollenkaan vanhemmissa selaimissa. Käytöltään :hover vastaa :link- ja :visited-pseudoluokkia. (Meyer 2001: 52.)

:active

:active-pseudoluokkaan kuuluvat elementit, joita käyttäjä osoittaa ja jotka ovat sillä hetkellä aktiivisia. Kuten :hover-pseudoluokkaakin, eivät useat selaimet anna :active-pseudoluokkaa muille kuin linkkielementeille. Myös :active-pseudoluokka puuttuu kokonaan vanhemmista selaimista. Käytöltään :active ei eroa :link-, :visited- ja :hover-pseudoluokista. (Meyer 2001: 53.)

:link-, :visited-, :hover- ja :active-pseudoluokkia käytetään pääsääntöisesti linkkielementtien muotoiluun. Vaikka periaatteessa elementtien pseudoluokkien määrittelyllä ei pitäisi olla mitään merkitystä CSS:ssä, joillakin selaimilla linkkielementtien pseudoluokat pitää määritellä järjestyksessä :link, :visited, :hover, :active.

:focus

:focus-pseudoluokkaan kuuluu elementti, joka sillä hetkellä on valittuna. Tämä siis tarkoittaa sitä, että esimerkiksi valittuna oleva tekstikenttä lomakkeessa kuuluisi :focus-pseudoluokkaan. Tässä käytössä :focus-pseudoluokkaa yleensä nähdäänkin, sillä näin pystytään selvästi osoittamaan käyttäjälle, mihin tekstikenttään ollaan kirjoittamassa. Jotkin selaimet soveltavat :focus-pseudoluokkaa ainoastaan lomakeelementteihin sekä toisinaan linkkielementteihin. Käytöltään :focus-pseudoelementti ei eroa :link-, :visited-, :hover- tai :active-pseudoluokista. (Meyer 2001: 53.)

:lang(RFC 1766 -kielikoodi)

:lang-pseudoluokkaan kuuluvat kaikki elementit, joiden sisältö on kirjoitettu määritellyllä kielellä. :lang-pseudoluokka toimii lähes samalla tavalla kuin |=-attribuuttiselektori, mutta :lang-pseudoluokalla voidaan tehdä valinta myös perustuen dokumentin head-osassa olevan meta-elementin tai jopa dokumentin HTTP-otsakkeiden kielimäärityksiin. :lang-pseudoluokkaa käytetään seuraavasti:

```
*:lang(fr)
{
  font-family: serif;
}
```

Esimerkki valitsee kaikki elementit, joiden kieleksi on määritelty ranska ja muuttaa ne käyttämään serif-tyyppistä kirjasinta. (Meyer 2001: 54.)

:left, :right ja :first

:left, :right ja :first ovat kaikki ainoastaan sivutetussa mediassa käytettyjä pseudoluokkia. :left-pseudoluokkaan kuuluvat kaikki dokumentin vasemmanpuoleiset sivut, kun taas :right-pseudoluokkaan puolestaan kuuluvat kaikki dokumentin oikeanpuoleiset sivut. :first-pseudoluokka sisältää ainoastaan dokumentin ensimmäisen sivun. Näitä pseudoluokkia käytetään @page-kokonaisuudessa, joka määrittelee halutun sivun ja sen ominaisuudet. (Meyer 2001: 216.)

5.8.10 Pseudoelementit

Siinä missä selain voi antaa tietylle elementille pseudoluokan, voi se myös määritellä dokumentin tekstisisältöön kokonaan uusia pseudoelementtejä. Pseudoelementit toimivat kuin normaalit elementitkin, mutta niitä vain ei ole varsinaisessa dokumentin merkkauksessa.

Käytettäessä pseudoelementtiselektoria, on huomioitava, että pseudoelementtiselektorin on oltava viimeinen osa säännön selektorimäärittelyä. Niinpä pseudoelementtiselektorin on tultava myös kaikkien pseudoluokkaselektoreiden jälkeen.

:first-letter

:first-letter-pseudoelementti valitsee elementin tekstisisällön ensimmäisen kirjaimen. Seuraavassa esimerkkejä :first-letter-pseudoelementin käytöstä:

```
p:first-letter
{
  font-size: 1.5em;
  color: #f00;
}

h2.sidebar:first-letter
{
  font-size: 2em;
  color: #0ad;
}

p#ingressi:first-letter
{
  color: #060;
}
```

Ensimmäinen esimerkki valitsee kaikkien p-elementtien ensimmäiset kirjaimet, vaihtaa niiden koon isommaksi ja värjää ne punaiseksi. Toinen esimerkki valitsee kaikkien h2-elementtien, joilla on class-attribuutilla arvo ”sidebar” ensimmäiset kirjaimet, vaihtaa niiden koon isommaksi ja värin sinivihreäksi. Kolmas esimerkki valitsee p-elementin, jolla on id-attribuutilla arvo ”ingressi” ensimmäisen kirjaimen ja vaihtaa sen väriksi tummanvihreän. On huomattava, että :first-letter-pseudoelementille ei voida määritellä kaikkia ominaisuuksia. (Meyer 2001: 56.)

:first-line

:first-line-pseudoelementti vastaa :first-letter-pseudoelementtiä, mutta ensimmäisen kirjaimen asemesta valitaan ensimmäinen rivi elementin tekstisisällöstä. Käytöltään :first-line ei eroa mitenkään :first-letter-pseudoelementistä. Myöskään :first-line-pseudoelementin kanssa ei voida käyttää kaikkia ominaisuuksia. (Meyer 2001: 57.)

:before

:before-pseudoelementti lisää sisältöä dokumenttiin ennen määriteltyä elementtiä. Lisättävä sisältö määritellään content-ominaisuudella. Koska :before-pseudoelementillä dokumenttiin tuotettu sisältö ei näy varsinaisessa merkkauksessa ollenkaan, sitä ei näy selaimissa, jotka eivät ymmärrä :before-pseudoelementtiä eivätkä esimerkiksi hakukoneet indeksoi sitä ollenkaan. Niinpä onkin ensiarvoisen tärkeää, että :before-pseudoelementillä ei tuoteta informaatiosisällöltään tärkeää sisältöä, vaan sitä käytetään ainoastaan sisällön visuaaliseen muotoiluun. Esimerkkejä :before-pseudoelementin käytöstä:

```
h2.valiotsikko:before
{
  content: url(images/valiotsikkomerkki.gif);
  padding: 0.2em;
}

blockquote:before
{
  content: open-quote;
}
```

Ensimmäinen esimerkki valitsee kaikki h2-elementit, joilla on class-attribuutilla arvo ”valiotsikko” ja lisää niiden eteen kuvan, jolle määrätään lopuksi pieni täyte. Jälkimmäinen esimerkki valitsee kaikki blockquote-elementit ja lisää niiden eteen aloittavan lainausmerkin. (Meyer 2001: 57.)

:after

:after-pseudoelementti toimii muuten täsmälleen samalla tavalla kuin :before-pseudoelementtikin, mutta :after lisää sisältöä määritellyn elementin jälkeen. Käytöltään :after-, ja :before-pseudoelementit eivät eroa mitenkään toisistaan. Edellisessä esimerkissä :before-pseudoelementillä lisättiin blockquote-elementtien eteen aloittavat lainausmerkit. Käyttämällä :after-pseudoelementtiä, voitaisiin luoda noille samoille blockquote-elementeille myös lopettavat lainausmerkit:

```
blockquote:after
{
  content: close-quote;
}
```

Samoista syistä kuin :before-pseudoelementilläkin, ei pitäisi myöskään :after-pseudoelementillä tuottaa informaatiosisällöltään tärkeää sisältöä, vaan sitä pitäisi myös käyttää ainoastaan sisällön visuaaliseen muotoiluun. (Meyer 2001: 58.)

5.8.11 @-säännöt

@-säännöillä pystytään vaikuttamaan CSS-tyylien tulkintaan selaimissa. Nämä säännöt merkitään alkavaksi aina @-merkillä ja säännöt sisältävät määritykset omana lohkonaan. Lohko voi olla kirjoitettuna aaltosulkujen sisään tai se voidaan kirjoittaa ilman aaltosulkuja, jolloin lohkon loppua merkkaa puolipiste.

@import

@import-säännöllä voidaan liittää CSS-määrittelyyn tyylitiedostoja. @import-säännön pitää olla joko merkkauksessa style-elementin sisällä tai ulkoisessa tyylitiedostossa. @import-säännön syntaksista on esimerkki kappaleessa ”CSS:n käyttö”.

@media

@media-säännöllä voidaan määritellä tietyt CSS-tyylit pätemään tietyssä mediassa. Niinpä esimerkiksi voitaisiin määritellä niin ruudulta luettavalle sisällölle kuin tulostetulle ja projektorin kautta katsottavalle sisällöllekin omat tyylinsä samaan tiedostoon. @media-sääntöä voitaisiin käyttää esimerkiksi seuraavasti:

```
@media screen
{
  body
  {
    background-color: #000;
    color: #fff;
  }
}

@media print, projection
{
  body
  {
    background-color: #fff;
    color: #000;
  }
}
```

Ensimmäinen sääntö vaihtaisi normaalisti ruudulta luettavan sisällön body-elementille taustaväriksi mustan ja asettaisi tekstin väriksi valkoisen. Jälkimmäinen sääntö puolestaan vaihtaisi tulostetulle ja projektorin kautta katsotulle sisällölle värit juurikin päinvastoin.

Vaikka se kovin erikoiselta tuntuukin, CSS 2.0-spesifikaation mukaan @media-säännölle ei ole pakko määritellä mediatyyppiä.

@charset

@charset-säännöllä voidaan merkitä CSS-tiedosto käyttämään jotakin tiettyä merkistökoodausta. CSS-tiedostossa voi olla vain yksi @charset-sääntö ja ennen sitä ei saa olla tiedostossa mitään. @charset-sääntöä ei voida käyttää muissa tyylimäärittelyissä kuin ulkoisissa tyylitiedostoissa. Esimerkki @charset-säännön käytöstä:

```
@charset "ISO-8859-1";
```

Esimerkissä CSS-tiedoston merkistökoodaukseksi asetettaisiin ISO-8859-1, joka siis on toiselta nimeltään Latin-1.

@font-face

@font-face-sääntö pakottaa jonkin tietyn kirjasimen käytettäväksi dokumentin kirjasimena. Koska @font-face-sääntö on toiminnaltaan hyvin monimutkainen, ei toimi kaikissa selaimissa eikä sitä edes suositella käytettäväksi, ei sitä käsitellä tässä yhteydessä tämän enempää.

@page

@page-sääntö määrittelee halutun sivun ja sen ominaisuudet. @page kuuluu sivutetun median kanssa käytettäviin määrittelyihin, joita ei myöskään käsitellä tässä yhteydessä.
(Meyer 2001: 59.)

5.9 Ominaisuudet

CSS:llä voidaan määritellä elementeille hyvin monia ominaisuuksia. Osa ominaisuuksista on periytyviä ja osa puolestaan ei – lähes kaikille ominaisuuksille kyllä voidaan antaa arvoksi inherit, jolloin tuo ominaisuus peritään vanhemmalta. Lista CSS 2.0:n ominaisuuksista on nähtävissä esimerkiksi osoitteessa http://www.w3schools.com/css/css_reference.asp.

5.10 Yksiköt ja arvot CSS:ssä

CSS-määrittelyt sisältävät usein erilaisia avaruudellisia määreitä. Avaruudellisia määreitä ovat niin pituus- ja leveysmääreet kuin esimerkiksi väritkin. Nämä määreet tarvitsevat yleensä lisätietoa siitä, millaisina yksikköinä arvot annetaan.

Yksiköiden valinta saattaa olla hyvinkin ratkaisevassa osassa siinä, miten käyttäjä pystyy hyödyntämään tarjottua sisältöä. Esimerkiksi kirjasinkoon määritteleminen absoluuttisena yksikkönä, vaikkapa pikseleinä, pakottaa kirjasimen aina tietyn kokoiseksi käyttäjän omista selainasetuksista ja ruuturesoluutiosta riippumatta. Jos taas kirjasinkoko olisi määritelty jonakin relatiivisena yksikkönä, vaikkapa em-yksikkönä, voisi käyttäjä määrätä kirjasinkoon oman mielensä mukaiseksi. Nykyään tosin ainakin Mozilla-selaimet antavat käyttäjälle mahdollisuuden isontaa ja pienentää myös absoluuttisina yksikköinä määriteltyjä kirjasimia.

5.10.1 Väriarvot

Väriarvojen esittämiseen CSS:ssä voidaan käyttää useata erilaista tapaa: arvot voidaan esittää punaisesta, vihreästä ja sinisestä komponentista koostuvana RGB-arvona (red, green, blue) tai niiden esittämiseen voidaan käyttää tietyille väreille annettuja nimiä.

#RRGGBB

#RRGGBB-määrittelyssä jokaista komponenttia vastaa pari heksadesimaaliarvoja. Ensimmäinen pari esittää punaisen komponentin arvon välillä 0 - 255, toinen vihreän ja kolmas sinisen. Näin saadaan käyttöön koko 24-bittinen väriavaruus.

#RGB

#RGB-määrittely on lyhennetty muoto edellisestä. Kun väri annetaan muodossa #RGB, voidaan käyttää ainoastaan ns. replikoituja heksadesimaalipareja. Esimerkiksi CSS:ssä asetettu väriarvo #6F8 tulkitaan selaimessa muotoa #RRGGBB vastaavaksi arvoksi #66FF88.

Rgb(rrr%, ggg%, bbb%)

Käytettäessä määrittelyä, joka on muotoa rgb(rrr%, ggg%, bbb%) annetaan komponenttien määrät prosentteina. Tämä siis tarkoittaa, että määrittely rgb(100 %, 100 %, 100 %) vastaa #RRGGBB-määrittelynä arvoa #FFFFFF, eli valkoista.

Rgb(rrr, ggg, bbb)

Edellisestä poiketen määrittely, joka on muotoa rgb(rrr, ggg, bbb) ei aseta komponenttien määriä prosentteina, vaan desimaalilukuina väliltä 0 - 255. Tällä tavalla valkoinen siis esitettäisiin muodossa rgb(255, 255, 255).

Värin nimi

Värejä voidaan myös määritellä käyttämällä joillekin tietyille väreille annettuja nimiä. Nimiä on 16 kappaletta ja ne periytyvät alkuperäisestä Microsoft Windowsin VGA-paletista. Jotkin selaimet voivat tunnistaa useampiakin värinimiä, mutta ainoastaan seuraavat ovat CSS 2.0 -standardin mukaisia: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white ja yellow.

(Meyer 2001: 32.)

5.10.2 Pituusyksiköt

Pituusyksiköitä käytetään elementtien pituuksien ja leveyksien sekä muiden ulottuvuudellisten arvojen yhteydessä. Kuten edellä mainittiin, pituusyksiköitä on kahta päätyyppiä: absoluuttisia ja relatiivisia. Absoluuttisiin pituusyksiköihin kuuluvat kaikki ne, joille voidaan löytää

reaalimaailmasta jokin vastine. Relatiiviset pituusyksiköt puolestaan määrittelevät ulottuvuudet suhteessa johonkin toiseen mittaan.

In (tuuma)

Kun jokin mitta määritellään tuumina, käyttää selain silloin hyväkseen tietoa siitä, millainen resoluutio senhetkisessä näyttölaitteessa on käytössä. Tämä resoluutio voi vaihdella hyvinkin paljon riippuen näyttölaitteesta, mikä tekeekin tuumasta järkevän yksikkövaihtoehdon ainoastaan, kun käytössä on printtimedia, eli esimerkiksi tuloste verkkosivusta.

Cm (senttimetri) ja mm (millimetri)

Senttimetri ja millimetri vastaavat käytöltään täysin tuumaa. Niinpä myös senttimetriä ja millimetriä kannattaa käyttää ainoastaan printtimediassa.

Pt (point eli piste)

Pisteitä käytetään varsin yleisesti etenkin tekstinkäsittelyohjelmissa kirjasimen koon määrittelemiseen. Tämän vuoksi myös monilla sivustoilla kaikki kirjasinmäärittelyt on tehty käyttäen pisteitä. Tämä ei kuitenkaan ole hyvä käyttökohde pisteille, sillä kuten muutkin absoluuttiset pituusyksiköt, pisteet sopivat huonosti ruutu- ja projektiomedioihin. Pisteen määritelmän mukaan yksi tuuma sisältää 72 pistettä, mikä tekee myös pisteen yksikkönä riippuvaiseksi näyttölaitteen resoluutiosta. Printtimediassa pisteen käyttäminen yksikkönä on jopa suositeltavaa, sillä etenkin kirjasinkoon määrittelemisessä pisteiden käyttöön useimmat ihmiset ovat tottuneet tekstinkäsittelyohjelmien myötä.

Pc (pica)

Pican määritelmän mukaan yksi pica on 12 pistettä. Käytöltään pica ei eroa pisteestä eikä myöskään picaa tulisi käyttää ruutu- tai projektiomediassa.

Px (pikseli)

CSS:ssä yksi pikseli olisi määritelmän mukaan 1/90 tuumaa näytöllä. Useimmat selaimet kuitenkin käyttävät näytön fyysistä pikseliä yhden pikselin kokona. Koska pikseleiden käyttäminen mm. kirjasinkoon määrittelyyn on varsin yleistä, useimmiten tulostettaessa selaimet skaalaavat kirjasimet vastaamaan nykytulostinten korkeata resoluutiota. Pikseleitä ei tulisikaan käyttää kirjasinkoon määrittelyyn, vaan ennemminkin tilanteissa, joissa ollaan tekemisissä esimerkiksi bittikarttagrafiikan kanssa.

Em (em-korkeus)

Em on käytössä olevan kirjasimen kirjainlaatikon korkeus. Em on siis relatiivinen pituusyksikkö, jolla voidaan esimerkiksi asettaa kirjasimen koko suhteessa kulloinkin käytössä olevaan kirjasinkokoon. Jos vaikkapa kirjasimen kooksi asetettaisiin 1.2 em, tarkoittaisi se samaa kuin se, että kirjasimen kooksi asetettaisiin 120 %. Em on hyvin käyttökelpoinen yksikkö ruutu- ja projektiomedioissa, sillä käytettäessä em-yksikköä esimerkiksi kirjasinkoon määrittelemiseen, voi käyttäjä muuttaa oman mielensä mukaan kirjasinkokoa siten, että mahdolliset kokoerot eri otsikkotyypeillä ja leipätekstillä pysyvät suhteessa samana.

Ex (ex-korkeus)

Siinä missä em oli kirjasimen kirjainlaatikon korkeus, on ex kirjasimen pienen x-kirjaimen korkeus. Useimmat selaimet kuitenkin käyttävät yhden ex-yksikön korkeutena automaattisesti puolta em-yksikköä. Käytöltään ex ei eroa em-yksiköstä.

% (prosentti)

Prosenttia voidaan käyttää myös pituusyksikkönä. Määriteltäessä elementtien pituuksia tai leveyksiä prosentteina, lasketaan ulottuvuudet osuutena käytössä olevasta tilasta. Jotkin selaimet tosin tulkitsevat prosentit osuutena koko ikkunanäkymästä (viewport). Toiminnaltaan prosentit vastaavat em- ja ex-yksikköjä.

Väri- ja pituusyksiköiden lisäksi CSS 2.0 sisältää paljon muihin käyttötarkoituksiin soveltuvia yksiköitä. Näitä ovat muun muassa sekunti, hertzi ja aste. Näitä ei voida kuitenkaan käyttää sivuston visuaaliseen ulkoasuun vaikuttavien ominaisuuksien kanssa, joten niitä ei käsitellä tässä opinnäytetyössä. (Meyer 2001: 31.)

5.11 Yleisesti käytettyjä CSS-tekniikoita

Perinteiset taulukkorakenteella tehdyt sivustot nojautuivat hyvin vahvasti pikselintarkasti määriteltyyn ulkoasuun, jolloin esimerkiksi käyttäjän tekemät muutokset selaimen kirjasinasetuksiin saivat sivustot toimimaan väärin. Taulukkorakenteella saatiin kuitenkin tehtyä sivustoja, jotka näyttivät lähes identtisiltä eri selaimilla katsottuna. Tämä onkin asia, joka usein saa taulukkorakenteisiin tottuneen web-suunnittelijan tuskastumaan CSS:n kanssa.

Suunniteltaessa sivustoa, joka käyttää ulkoasun muotoiluun CSS:ää, pitää asiaa katsoa erilaisesta näkökulmasta. CSS:n kanssa koko ul-

koasua ei pyritä muotoilemaan pikselintarkasti, vaan päämääränä on luoda sisällölle säännöt, jotka kertovat, miten sisällön tulee dokumentissa käyttäytyä. CSS:n käytössä on muutamia perustekniikoita, joiden varaan useimmat CSS:ää käyttävät sivustot rakentuvat.

5.11.1 Elementtien kellutus

(X)HTML-dokumentin tulkinta ja esittäminen etenee selaimessa dokumenttivuon mukaisesti. Dokumenttivuo siis määrää sen, mihin kohtaan dokumentin esityksessä selain asettaa dokumentissa olevat elementit. Kellutuksella tarkoitetaan sitä, että elementit poistetaan dokumenttivuosta, joilloin ne nousevat ”kellumaan” niin ylös kuin mahdollista. Elementit pystyvät kuitenkin kellumaan ainoastaan niitä ympäröivän lohkotason elementin (block-level element) sisällä, jolloin esimerkiksi p-elementin sisällä oleva img-elementti pystyisi kellumaan vain p-elementin rajaamalla alueella.

Kellutus tuo mukanaan yhden erittäin kiusallisen ongelman: dokumenttivuosta poistetut elementit eivät vaikuta dokumenttivuon etenemiseen. Tämä tarkoittaa käytännössä sitä, että esimerkiksi edellisenkaltaisessa tilanteessa, jossa p-elementin sisällä oleva img-elementti kellutettiin, ei img-elementtiä laskettaisi enää osaksi p-elementin sisältöä, vaan img-elementti jatkuisi p-elementin alareunan yli. Kellutettuja elementtejä voidaan kuitenkin ohittaa toisilla elementeillä. Tätä ominaisuutta hyväksikäyttämällä dokumenttivuota saadaan ohjailtua näyttämään kellutetut elementit dokumentissa oikein. Kellutusta ja ohitusta voitaisiin käyttää esimerkiksi näin:

```
p img
{
  float: right;
}

p
{
  clear: both;
}
```

Esimerkissä p-elementin sisällä oleva img-elementti määrätään kellumaan sisällön oikealla puolella. Koska tämän jälkeen p-elementti määrätään ohittamaan molemmilla puolilla kelluvat elementit, alkaisi img-elementin sisältämän p-elementin jälkeen tuleva p-elementti vasta kellutetun img-elementin jälkeen. Näin siis varmistettaisiin se, että kellutetut kuvat kelluisivat ainoastaan niiden kappaleiden sisällä, mihin ne on HTML-merkkauksessa laitettu.

5.11.2 Tekstin korvaaminen kuvalla (image replacement)

CSS:ää on käytetty hyvin paljon tekstin korvaamiseen kuvalla. Tämä tehdään yleensä käyttämällä tekstielementillä taustakuvaa siten, että varsinainen teksti on tavalla tai toisella piilotettuna. Tämä tietysti edellyttää sitä, että asiakasselain tukee CSS:ää sekä kykenee näyttämään kuvat. Yleisimmin käytetty tekniikka tähän tarkoitukseen on käyttää tekstielementin ominaisuutta text-indent. Teksti voitaisiin korvata kuvalla esimerkiksi seuraavanlaisella CSS-säännöllä:

```
h1
{
  width: 700px;
  height: 150px;
  background: url(taustakuva.gif) no-repeat #fff;
  text-indent: -9999px;
}
```

Esimerkissä h1-elementille annettaisiin leveydeksi 700 pikseliä ja korkeudeksi 150 pikseliä. Tämän jälkeen h1-elementin taustakuvaksi asetettaisiin taustakuva.gif. Mikäli taustakuvalle ei annettaisi arvoa no-repeat, monistuisi taustakuva täyttämään koko h1-elementin, vaikka taustakuva itse olisi elementtiä pienempi. Koska taustalle on myös määritelty taustaväriksi valkoinen, täytetään valkoisella taustavärillä nyt ne alueet, joihin taustakuva ei mahdollisesti yllä h1-elementin sisällä. Viimeiseksi h1-elementin teksti siirretään 9999 pikseliä elementin vasemman reunan yli. Näin teksti ei tule näkyviin, vaan tekstin sijaan näkyvissä on elementin taustakuva.

Tilanteessa, jossa selain tukee CSS:ää, mutta käyttäjä on esimerkiksi asettanut selaimesta kuvien näyttämisen pois päältä, ei käyttäjä näe tekstiä eikä elementin taustalle asetettua kuvaa.

6 MIME-tyypit (Multipurpose Internet Mail Extensions)

Tiedoston MIME-tyyppi kertoo, millaista tiedoston sisältö on. Esimerkiksi HTML-dokumenttien MIME-tyyppi on text/html ja GIF-kuvien image/gif. Koska HTML eroaa XHTML:stä syntaksin osalta varsin paljon, käytetään selaimissa myös niiden tulkitsemiseen erilaisia tapoja. Päätöksen siitä, miten kulloinenkin dokumentti tulkitaan, selain tekee MIME-tyypin perusteella.

HTML:n oikea MIME-tyyppi on siis text/html, mutta XHTML-dokumentti pitäisi tarjoilla palvelimelta MIME-tyypillä application/xhtml+xml. Varsin yleisesti XHTML-dokumentteja kuitenkin tarjoillaan text/html-MIME-tyypillä, mutta tällöin selain ei tulkitse dokumenttia XHTML:nä, vaan käyttää tulkitsemiseen HTML-tulkkia. Tästä ei usein ole mitään haittaa, mutta toisaalta on myös turha tehdä merkkauksesta tiukemman XHTML:n mukaista, jos se lopulta kuitenkin tulkitaan HTML:nä.

Jos kuitenkin XHTML-dokumentti tarjoillaan oikealla MIME-tyypillä, saattaa ilmentyä odottamattomia ongelmia tietyissä tilanteissa. Selainten HTML-tulkki esittää automaattisesti HTML-dokumentin body-elementin siten, että se täyttää koko ikkunanäkymän. XHTML-tulkki puolestaan täyttää koko ikkunanäkymän ainoastaan html-elementillä ja käyttää body-elementtiä normaalina osana dokumenttivuossa. Tämä tulee ottaa huomioon etenkin käytettäessä kellutettuja elementtejä.

Toinen, paljon suurempi ongelma XHTML:n tarjoilemisessa oikealla MIME-tyypillä on selainyhteensopimattomuus. Esimerkiksi Microsoft Internet Explorer (IE) ei osaa tehdä mitään sellaisille dokumenteille, jotka tarjoillaan application/xhtml+xml -MIME-tyypillä. Tämä on varsin suuri ongelma siksi, että ainakin opinnäytetyön kirjoitushetkellä IE hallitsee selainmarkkinoita.

W3C:n mukaan XHTML:ää ei välttämättä tarvitse tarjoilla application/xhtml+xml -MIME-tyypillä, vaan MIME-tyyppi voi olla periaatteessa mikä tahansa XML-MIME-tyyppi. XML-MIME-tyypillä tarjoiltuna dokumenttiin pitää liittää XSL-tiedosto (Extensible Stylesheet Language), jolloin selaimet käyttävät XSLT:tä (XSL Transformation) muuntaessaan lukemansa XML-dokumentin esitettävään muotoon. Tarvittava XSL-tiedosto on varsin yksinkertainen, sillä muunnettava sisältö on jo valmiiksi oikeassa muodossa. Niinpä XSLT:tä tarvitaankin vain kopioimaan XML-tiedoston sisältö esitettävälle sivulle. XHTML-dokumentin tarjoileminen näin korjaa tilanteen IE:n kanssa, mutta selainten XSLT-tuki on toistaiseksi niin heikkoa, että monen muun selaimen koh-

dalla tämä on paljon huonompi tapa kuin tarjoilla dokumentti oikealla application/xhtml+xml -MIME-tyypillä.

Paras tapa tarjoilla XHTML-dokumentteja on suorittaa sisältöneuvottelu ennen dokumentin tarjoamista asiakasselaimelle. Sisältöneuvottelussa palvelin pyytää asiakasselaimelta listaa siitä, mitä MIME-tyyppejä selain tukee. Nämä MIME-tyypit tulevat selaimelta prioriteettijärjestyksessä, jolloin palvelin pystyy MIME-tyyppien järjestyksestä päättelemään, suosiiko asiakasselain XHTML- vai HTML-dokumentteja. Mikäli selain kertoo suosivansa XHTML-dokumentteja HTML-dokumenttien asemesta, voidaan XHTML-dokumentti tarjoilla palvelimelta oikealla MIME-tyypillä, mutta mikäli tilanne on päinvastainen, vaihdetaan dokumentin MIME-tyyppi text/html:ksi. Tällöin rikotaan W3C:n suositusta siitä, että XHTML-dokumentti tulisi tarjoilla XML-MIME-tyypillä varustettuna, mutta nykyisten selainten puutteellisen XHTML- ja XML-tuen vuoksi XHTML-dokumenttien tarjoileminen text/html -MIME-tyypillä on tietyissä tapauksissa hyväksyttävää.

7 Yhteenveto

Vaikka yleinen tuntemus web-standardien kohdalla on lisääntynyt viime aikoina web-kehittäjien keskuudessa, tuntuu tuntemus silti olevan varsin pinnallista. Web-standardeista ja erityisesti saavutettavuudesta onkin muodostunut jonkinlaisia muotisanoja, joilla voidaan myydä niin selaimia ja mainostoimistojen palveluita kuin koulujen opintojaksojakin.

Tämä opinnäytetyö rakentuu kokonaisuudessaan sen olettamuksen varaan, että web-standardien avulla voidaan pitkällä tähtäimellä paitsi nopeuttaa ja helpottaa verkkosivustojen rakentamista sekä ylläpitoa, myös rakentaa paremmin toimivia ja käyttäjäystävällisempiä sivustoja. Sivustojen toimivuus on tietysti riippuvainen siitä käyttöympäristöstä, missä niitä käytetään, joten selainkehityksen tulisi kulkea standardikehityksen kanssa hyvin lähekkäin. Ikävä kyllä, opinnäytetyön kirjoittamishetkellä joidenkin valtavirtaselaintenkin kehitys kulkee hieman standardikehityksen jäljessä.

Verkkosivustojen toimivuus ja standardienmukaisuus eivät ole pelkästään kiinni itse dokumenteista, vaan vaatimuksia asetetaan myös palvelimelle. Palvelimen tulee pystyä toimittamaan halutut dokumentit halutussa muodossa ja haluttuun aikaan. Tämä luo kriittisen osan verkkosivujen saavutettavuutta.

Myöskään se, että verkkosivut pystytään toimittamaan mainittujen kriteerien mukaisesti, ei vielä tarkoita sitä, että sivujen käyttäjä pystyisi käyttämään sivuilla olevaa tietoa. Sivujen tulee olla tyypiltään sellaisia, että käyttäjän selain pystyy ne näyttämään ja sivuston pitää olla ulkoasultaan käyttäjän luettavissa. Tiettyjen tapausten kohdalla tämä on mahdoton yhtälö, mutta web-standardien oikealla käytöllä ja järkevällä sivustosuunnittelulla voidaan päästä varsin lähelle täydellisyyttä.

Koska web-standardien kehitys jatkuu, ei tämän opinnäytetyön voida katsoa olevan kaikilta osin kovinkaan pitkäikäinen. Web-standardien kehitystyössä kuitenkin pyritään siihen, että uudet standardiversiot ovat yhteensopivia vanhojen versioiden kanssa, eivätkä esimerkiksi käytettävyyden ja saavutettavuuden perusperiaatteet muutu tulevaisuudessakaan.

Lähteet

Painettu kirjallisuus

Cederholm, Dan 2004. Web Standards Solutions. The Markup and Style Handbook. USA: Apress L.P.

Meyer, Eric A. 2001. Cascading Style Sheets 2.0. Programmer's Reference. USA: The McGraw-Hill Companies.

Musciano, Chuck & Kennedy, Bill 2002. HTML & XHTML. The Definitive Guide. USA: O'Reilly Media, Inc.

Nykänen, Ossi 2003. XML. Jyväskylä: Dodenco Finland Oy.

Veen, Jeffrey 2002. Webdesign. Helsinki: IT-Press, Edita Publishing Oy.

Zeldman, Jeffrey 2003. Designing with Web Standards. USA: New Riders.

Internet-lähteet

Merikallio, Bill & Pratt, Adam 2003. Why tables for layout is stupid: problems defined, solutions offered. [online] [viitattu 7.6.2005] http://www.hotdesign.com/seybold/

W3 Schools 2005. XHTML Syntax. [online] [viitattu 10.11.2005] http://www.w3schools.com/xhtml/xhtml_syntax.asp

Web Standards Project 2002. FAQ [online] [viitattu 6.6.2005] http://webstandards.org/learn/faq/

World Wide Web Consortium 2002a. XHTML 1.0: The Extensible Hypertext Markup Language (Second Edition). [online] http://www.w3.org/TR/xhtml1/

World Wide Web Consortium 2002b. XHTML Media Types. [online] http://www.w3.org/TR/xhtml-media-types/

World Wide Web Consortium 2005a. HyperText Markup Language (HTML). [online] [viitattu 3.11.2005] http://www.w3.org/MarkUp/

World Wide Web Consortium 2005b. Cascading Style Sheets. [online] http://www.w3.org/Style/CSS/

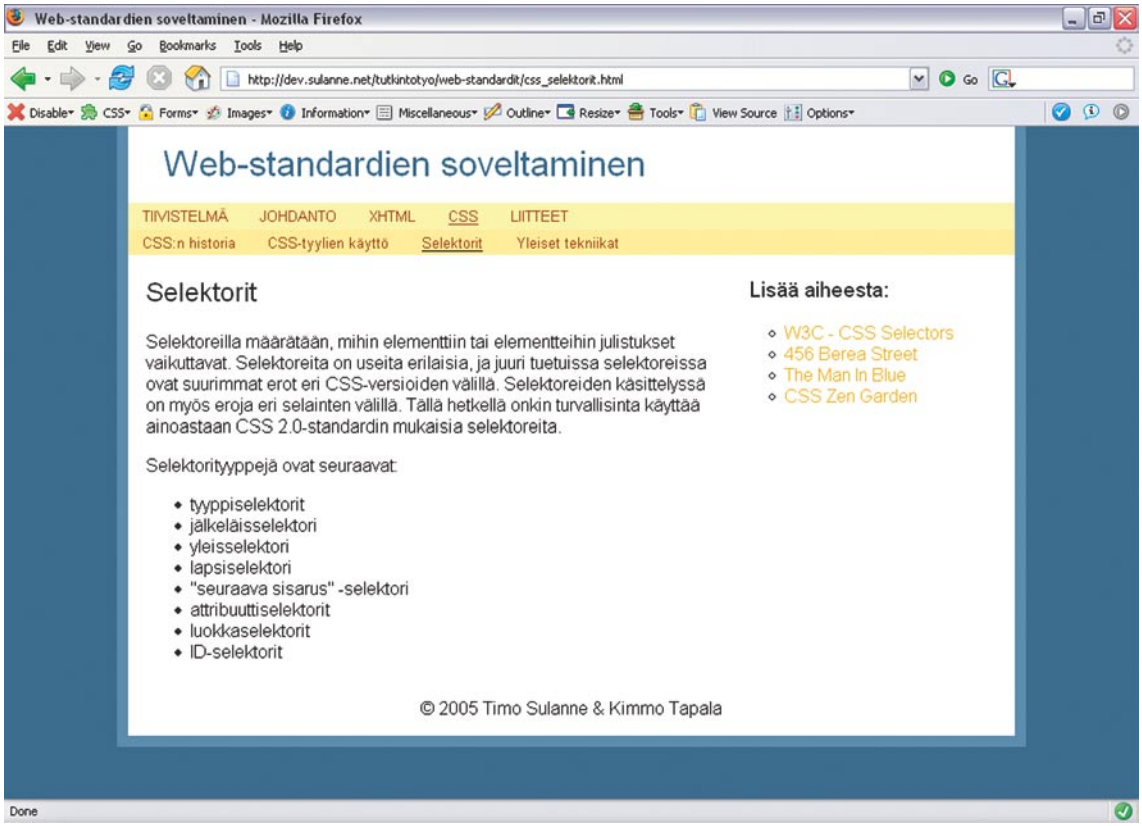
Liite 1: HTML- ja XML-piirteiden vertailua

Lähde: Nykänen 2003: 7

Piirre	**HTML 4.01**	**XML**
Dokumentin tarkoitus	WWW-hyperteksti	Rakenteinen tekstidokumentti
Käyttökohde	WWW-selain	Jokin XML-sovellus (esim. XML-selain, XML-muunnin tms. XML-ohjelma)
Hyödyntää käytännössä	WWW- ja Internet-ohjelmia	Edellisten lisäksi alun perin HTML:n käyttöön rakennettua WWW-infrastruktuuria (esim. DOM, CSS, URIt, palvelinverkostot jne.) ja muita ohjelmia (esim. Java-, PDF- ja SGML-ohjelmat)
Tekstin merkkaus	SGML-tyyppinen merkkaus, käytössä erilaisia lyhennemerkintöjä	Tarkkaan rajattu, supistettu SGML-merkkaus ilman mitään lyhenne merkintöjä
Elementit ja attribuutit	Esim. HTML-kappaleen voi merkata lyhyesti `<P ALIGN=left>moi`	Aina täydellinen muotoilu, esim. `<p align=left”>moi</p>` (perus-XML ei määrittele kappale-tyyppistä elementtiä eli p tarkoittaa vain XML-elementtiä)
Elementtien tulkinta	Satoja metatieto-, rakenne- ja formatointi-elementtejä	Perus-XML 1.0 määrittelee tasan kaksi attribuuttia (xml:lang ja xml:space)
Tyhjän elementin merkkaus	Esim. ` ` tai ` </BR>`, elementin SGML-semantiikan mukaisesti	Vastaava merkintä ` </br>` tai ` ` (taas ilman rivinvaihdon semantiikkaa)
Syntaksivirhe esim. paljas ”<”	HTML-selain arvaa asiayhteydestä, että tarkoitettiin merkkiä ”<”	XML-prosessori keskeyttää suorituksensa peruuttamattomaan virhetilanteeseen
Hypertekstilinkki	`<A HREF=”kohde.html”>klikkaa </A>`	Ei löydy perus-XML 1.0:sta (monipuolisten hyperlinkkien määrittely löytyy esim. XLink-spesifikaatioista)

Kuva	Esim. `<IMG HREF=”logo.gif”>`	Ei löydy perus-XML 1.0:sta (toteutetaan XML:ssä aina tyylien avulla)
Uusien elementtien määrittely	Ei mahdollista	Perustoiminto, esim. `<moi>maailma</moi>`
Kommentit	`<!-- kommentti -->`	Sama kuin HTML:ssä
Merkkiviittaukset	Viittaus merkin Unicode-indeksin perusteella	Sama kuin HTML:ssä
Entiteettiviittaukset	Runsaasti valmiita määrittelyjä, esim. © (©-merkki)	Määriteltynä vain viisi merkkauksen erikoismerkkiä, mutta uusien erityyppisten entiteettien määrittely on mahdollista
Dokumentin tyyppimäärittely	Aina kiinteä HTML 4.01 DTD (kolme vaihtoehtoa)	Voidaan määritellä vapaasti XML DTD:n tai XML-skeemojen avulla
Dokumenttien jakaminen osiin	Koko käsitettä ei ole	Tekstientiteettien avulla
Nimiavaruus	Koko käsitettä ei ole (koska vain yksi sanasto)	Peruskäsite eri XML-sanastojen yhdistämiseen samassa dokumentissa

Liite 2: Esimerkkitoteutus



Kuva 1. Ruutukaappaus esimerkkisivusta Mozilla Firefox 1.5 -selaimessa.

XHTML-lähdekoodi

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fi" lang="fi">
<head>
   <title>Web-standardien soveltaminen</title>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
   <style type="text/css" title="Vakiotyylit" media="screen">
      @import url(tyylit.css);
   </style>
</head>
```

```
<body>

  <!-- Dokumentin ylätunniste, joka sisältää sivun otsikon -->
  <div id="header">
    <h1>Web-standardien soveltaminen</h1>
  </div>

  <!-- Päänavigaatio, joka sisältää navigaatioon käytetyn linkkilistan -->
  <div id="navigation">
    <ul>
      <li><a href="index.html">Tiivistelmä</a></li>
      <li><a href="johdanto.html">Johdanto</a></li>
      <li><a href="xhtml.html">XHTML</a></li>
      <li class="active"><a href="css.html">CSS</a></li>
      <li><a href="liitteet.html">Liitteet</a></li>
    </ul>
  </div>

  <!-- Varsinaisen sisällön ympäröivä elementti -->
  <div id="content_wrapper">

    <!-- Alanavigaatio, joka on muodoltaan päänavigaation kaltainen -->
    <div id="subnavigation">
      <ul>
        <li><a href="css_historia.html">CSS:n historia</a></li>
        <li><a href="css_kaytto.html">CSS-tyylien käyttö</a></li>
        <li class="active"><a href="css_selektorit.html">Selektorit</a></li>
        <li><a href="css_tekniikat.hmtl">Yleiset tekniikat</a></li>
      </ul>
    </div>

    <!-- Sisältöpalstat -->
    <div id="content">

      <!-- Päätekstisisältö, eli vasemmanpuoleinen palsta -->
      <div id="main_content">
        <h2>Selektorit</h2>
        <p>
          Selektoreilla määrätään, mihin elementtiin tai elementteihin
          julistukset vaikuttavat. Selektoreita on useita erilaisia, ja
          juuri tuetuissa selektoreissa ovat suurimmat erot eri
          CSS-versioiden välillä. Selektoreiden käsittelyssä on myös eroja
          eri selainten välillä. Tällä hetkellä onkin turvallisinta käyttää
          ainoastaan CSS 2.0-standardin mukaisia selektoreita.
        </p>
```

```
            <p>
               Selektorityyppejä ovat seuraavat:
            </p>
            <ul>
               <li>tyyppiselektorit</li>
               <li>jälkeläisselektori</li>
               <li>yleisselektori</li>
               <li>lapsiselektori</li>
               <li>&quot;seuraava sisarus&quot; -selektori</li>
               <li>attribuuttiselektorit</li>
               <li>luokkaselektorit</li>
               <li>ID-selektorit</li>
            </ul>
         </div>

         <!-- Sivupalkki, eli oikeanpuoleinen palsta -->
         <div id="sidebar">
            <h3>Lisää aiheesta:</h3>
            <ul>
               <li><a href="http://www.w3.org/Style/CSS/">W3C - CSS Selectors
               </a></li>
               <li><a href="http://www.456bereastreet.com/">456 Berea Street
               </a></li>
               <li><a href="http://www.themaninblue.com/">The Man In Blue</a>
               </li>
               <li><a href="http://www.csszengarden.com/">CSS Zen Garden</a>
               </li>
            </ul>
         </div>

         <!-- Sivun alatunniste -->
         <div id="footer">
            <p>
               &#169; 2005 Timo Sulanne &amp; Kimmo Tapala
            </p>
         </div>

      </div>
   </div>
</body>
</html>
```

CSS-tyylimäärittelyt

```
/*
   ”Web-standardien soveltaminen”-esimerkkisivun CSS-tyylitiedosto
   (c)2005 Timo Sulanne ja Kimmo Tapala

   Esimerkkisivu on toteutettu ns. ”elastisella” CSS-taitolla, mikä tarkoittaa,
   että koko sivun koko muuttuu suhteessa kirjasinkokoon.
*/

/*
   Asetetaan yleiset asetukset sivun juurielementeille ja otsikoille.
   Tässä tapauksessa html- ja body-elementit saavat samat tyylimäärittelyt,
   koska ”oikeana” XHTML:nä tiedoston html- ja body-elementit täyttävät eri
   osan ikkunanäkymästä.
*/

html, body
{
   background-color: #468;
   color: #000;
   font-family: arial, sans-serif;
   text-align: center;
   padding: 0;
   margin: 0;
}

h1, h2, h3, h4, h5, h6
{
   font-family: ”gill sans”, sans-serif;
   font-weight: normal;
}

/*
   Tehdään yleiset tyylimäärittelyt dokumentin kolmelle runkoelementille:
   ylätunnisteelle, navigaatiolle ja sisältää ympäröivälle elementille.
*/

#header, #navigation, #content_wrapper
{
   background-color: #fff;
   width: 50em;
   text-align: left;
   border-left: 10px solid #68a;
   border-right: 10px solid #68a;
   margin: 0 auto;
}
```

```
/*
   Tehdään ylätunnisteen tyylimäärittelyt
*/

#header
{
   padding: 1em 0;
}

#header h1
{
   padding: 0 1em;
   color: #468;
   margin: 0;
}

/*
   Tehdään navigaation tyylimäärittelyt. Asettamalla li-elementin
   display-ominaisuuden arvoksi ”inline”, saadaan lista pysymään yhdellä
   rivillä. Periaatteessa sama lopputulos voitaisiin saada kelluttamalla
   li-elementit vierekkäin, mutta silloin tarvittaisiin dokumenttivuohon
   jokin lisäelementti, joka ohittaisi kellutetut elementit.
*/

#navigation
{
   background-color: #ffa;
   color: #830;
   text-transform: uppercase;
}

#navigation ul
{
   font-size: 0.8em;
   list-style: none;
   margin: 0;
   padding: 0.3em 0;
}

#navigation li
{
   padding: 0 1em;
   display: inline;
}
```

```
#navigation li.active
{
   text-decoration: underline;
}

/* Navigaation linkkien tyylit (pseudoluokilla) */

#navigation a:link, #navigation a:visited
{
   color: #830;
   text-decoration: none;
}

#navigation a:hover, #navigation a:active
{
   color: #ca0;
   text-decoration: underline;
}

/*
   Alinavigaatiolle tehdään hyvin samanlaiset tyylimäärittelyt kuin
   varsinaiselle navigaatiollekin.
*/

#subnavigation
{
   background-color: #fe9;
}

#subnavigation ul
{
   list-style: none;
   margin: 0;
   padding: 0.3em 0;
   font-size: 0.8em;
}

#subnavigation li
{
   padding: 0 1em;
   display: inline;
}
```

```
#subnavigation li.active
{
   text-decoration: underline;
}

#subnavigation a:link, #subnavigation a:visited
{
   color: #830;
   text-decoration: none;
}

#subnavigation a:hover, #subnavigation a:active
{
   color: #ca0;
   text-decoration: underline;
}

/*
   Tehdään tyylimäärittelyt itse sisällölle. Pääsisällön sisältävä
   main_content-lohko asetetaan kellumaan vasemmalla puolella, jolloin siitä
   tulee vasemmanpuoleinen palsta.
*/

#content_wrapper
{
   border-bottom: 10px solid #68a;
}

#content
{
   margin: 0;
}

#main_content
{
   float: left;
   width: 32em;
   margin: 0 0 1em 1em;
}

/*
   Tehdään sivupalkin tyylimäärittelyt. Sivupalkki asetetaan myös kellumaan
   vasemmalle, jolloin se siirtyy välittömästi pääsisällön oikealle puolelle.
   Mikäli haluttaisiin vielä kolmas palsta dokumenttiin, voitaisiin sekin
   kelluttaa vielä vasemmalle puolelle, jolloin se siirtyisi välittömästi
   sivupalkin oikealle puolelle.
*/
```

```
#sidebar
{
   float: left;
   width: 13em;
   margin: 0;
   padding: 0 1em 0 2em;
}

#sidebar ul
{
   list-style-type: circle;
   margin: 0;
   padding-left: 2em;
}


#sidebar a:link, #sidebar a:visited
{
   color: #da0;
   text-decoration: none;
}

#sidebar a:hover, #sidebar a:active
{
   color: #830;
   text-decoration: underline;
}

/*
   Tehdään alatunnisteen tyylimäärittelyt. Alatunniste toimii kellutukset
   ohittavana elementtinä, joka saa sisällön ympäröivän elementin jatkumaan
   sisällön alareunaan asti.
*/

#footer
{
   padding: 0 1em;
   clear: both;
   text-align: center;
}
```