Nina Latostenmaa

MERKISTÖIHIN LIITTYVÄT ONGELMAT MONIKIELISEN WWW-JÄRJESTELMÄN TUOTANTOPROSESSISSA

Tietojärjestelmätieteen
pro gradu –tutkielma
16.6.2003

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

ABSTRACT

Latostenmaa, Nina Karoliina
Character set related problems in the development process of a multilingual WWW-system / Nina Latostenmaa
Jyväskylä: University of Jyväskylä, 2003.
92 p.
Master's thesis

This research describes character set related problems in multilingual systems with WWW user interface (= WWW-systems) and offers some solutions for these problems. This research also introduces what needs to be taken into account in the creation process of multilingual WWW-systems that these problems would not arise. The process model used in this study is a combination of the generic life cycle model and the waterfall model.

Character set related problems are very common in multilingual systems, but information about them cannot be found from one source. The main goal of this research is to gather this knowledge together. Therefore this thesis is primarily a literature review. Two expert interviews give practical view to this research.

Typical character set related problems are missing font and wrong character set or encoding definitions in WWW-document or WWW-browser. The problems appear as incorrect characters or unreadable text in WWW-pages. This research shows that most of these problems can be avoided, if character set requirements are taken into account in the creation process of multilingual WWW-system.

KEYWORDS: WWW-system, character set, encoding, font, Unicode, process model

TIIVISTELMÄ

Latostenmaa, Nina Karoliina
Merkistöihin liittyvät ongelmat monikielisen WWW-järjestelmän tuotantoprosessissa / Nina Latostenmaa
Jyväskylä: Jyväskylän yliopisto, 2003.
92 s.
Tutkielma

Tutkielmassa kuvataan millaisia merkistöihin liittyviä ongelmia voi ilmetä monikielisissä WWW-käyttöliittymillä varustetuissa tietojärjestelmissä (eli WWW-järjestelmissä) ja esitetään ratkaisuehdotuksia näihin ongelmiin. Tutkielmassa selvitetään mitä pitää ottaa huomioon WWW-järjestelmän tuotantoprosessissa, jotta näitä ongelmia ei ilmenisi. Tuotantoprosessin vaihejako pohjautuu geneeriseen elinkaarimalliin sekä vesiputousmalliin.

Merkistöongelmat ovat hyvin yleisiä monikielisissä järjestelmissä, mutta täydellistä tietoa niistä ei saa yhdestä lähteestä. Tutkielman tarkoitus onkin koota tätä tietämystä yhteen ja tutkielma on siksi ensisijaisesti luonteeltaan kirjallisuuskatsaus. Käytännön näkökulmaa tutkimukseen tuovat kaksi asiantuntijahaastattelua.

Tyypillisiä merkistöihin liittyviä ongelmia ovat puuttuva fontti sekä väärät merkistö- ja enkoodausmääritykset WWW-dokumentissa tai WWW-selaimessa. Ongelmat ilmenevät väärinä merkkeinä tai lukukelvottomana tekstinä WWW-sivuilla. Tutkielma osoittaa, että suurin osa näistä ongelmista voidaan välttää, jos merkistövaatimukset huomioidaan kunnolla monikielisen WWW-järjestelmän tuotantoprosessissa.

AVAINSANAT: WWW-järjestelmä, merkistö, enkoodaus, fontti, Unicode, prosessimalli

SISÄLLYSLUETTELO

1 JOHDANTO 6
1.1 Tutkimuksen tausta 7
1.2 Tutkimusongelma ja tutkimusalueen rajaus 8
1.3 Tutkielman sisältö 9
1.4 Tutkimusmetodit 9

2 MÄÄRITELMIÄ 11

3 MERKISTÖT JA ENKOODAUKSET 16
3.1 Merkistöt 16
3.1.1 ASCII 17
3.1.2 ISO/IEC 8859 -merkistöperhe 18
3.1.3 Unicode ja UCS 20
3.1.4 Paikalliset merkistöt 22
3.2 Enkoodausmuodot 23
3.2.1 Kiinteämittainen ja vaihtelevamittainen enkoodaus 24
3.2.2 Big endian ja little endian 24
3.2.3 ISO 2022 standardi 26

4 TEKNISET MERKISTÖONGELMAT JA NIIDEN MAHDOLLISET RATKAISUT 28
4.1 Fontit 29
4.2 Merkistöstä puuttuvat merkit 33
4.3 Selaimen asetukset 34
4.4 WWW-dokumentin merkistö- ja enkoodausmääritykset 37
4.5 Tiedonsiirto 42
4.6 Tekstin syöttö 44
4.7 Ohjelmointikielet 48
4.8 Käyttöjärjestelmät 52
4.9 Yhteenveto 53

5 WWW-JÄRJESTELMÄN MERKISTÖT HUOMIOIVA TUOTANTOPROSESSI 57
5.1 Prosessimallit 58
5.1.1 Vesiputousmalli 58
5.1.2 WWW-järjestelmien uudelleenrakentamisen malli 60
5.1.3 Geneerinen elinkaarimalli 63
5.2 Merkistöongelmat huomioiva WWW-järjestelmän luontiprosessi 66
5.2.1 Vaatimusanalyysi 66
5.2.2 Käyttötapausten arviointi 68
5.2.3 Käyttöliittymän suunnittelu 69
5.2.4 Skenaarioiden luonti 72
5.2.5 Järjestelmäarkkitehtuurin suunnittelu ja luonti 72

5.2.6 Sovellusarkkitehtuurin suunnittelu ja luonti 74
5.2.7 Komponenttien ja järjestelmän toteutus 75
5.2.8 Testaus 79
5.2.9 Komponenttien ja järjestelmän julkaisu 80
5.2.10 Sisällön ylläpito 80
5.3 Yhteenveto 81

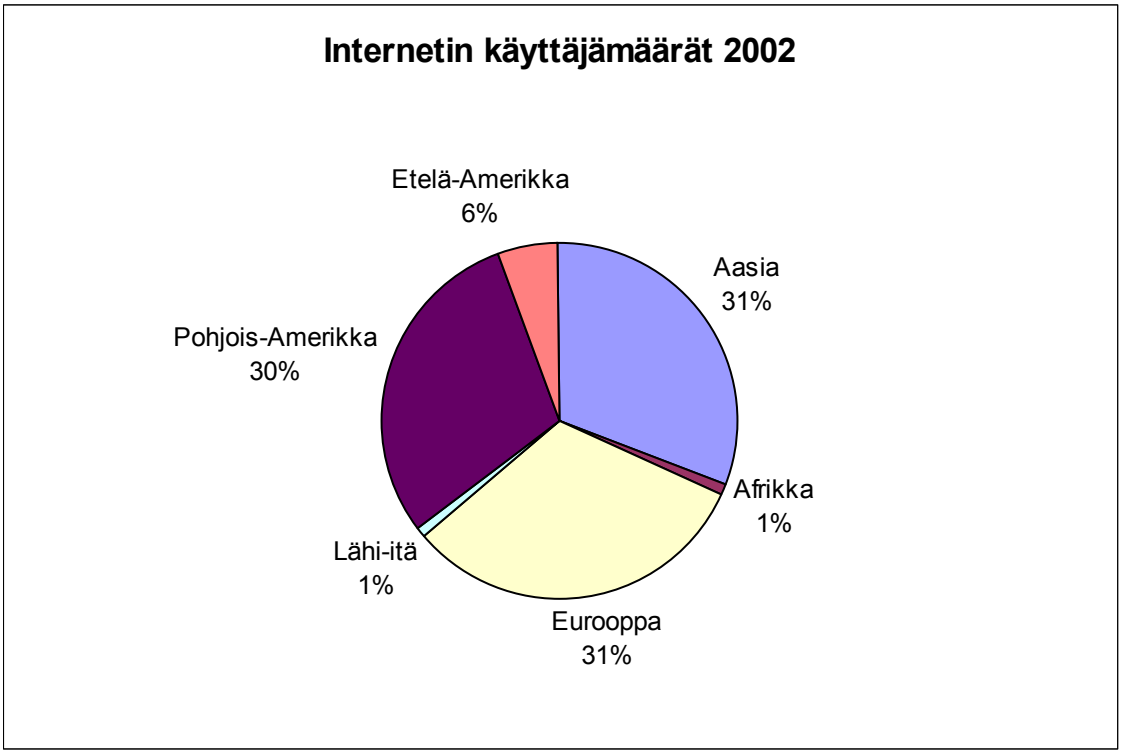
6 YHTEENVETO 85

LÄHDELUETTELO 91

LIITE 1 96

1 JOHDANTO

Internetin käyttö on lisääntynyt Euroopassa ja Aasiassa huimasti 2000-luvulla. Vielä vuonna 1999 NUA:n (2003) mukaan Internetillä oli noin 180 miljoonaa käyttäjää, joista noin 60% oli Pohjois-Amerikasta. Vuoden 2002 käyttäjämäärä oli NUA:n tutkimuksen mukaan 605,60 miljoonaa ja pohjoisamerikkalaisten osuus käyttäjistä oli enää 30%. Kaaviosta 1 näkyy vuoden 2002 käyttäjämäärien suhteet alueittain. (Aasiaan on luettu tässä tutkimuksessa myös Australia.)



KAAVIO 1. Internetin käyttäjämäärät vuonna 2002. (NUA Internet Surveys 2003)

Kuten kaaviosta 1 näkyy, Eurooppa ja Aasia ovat jo ohittaneet käyttäjämääriltään Pohjois-Amerikan. Näin ollen ne ovat varteenotettavia markkina-alueita elektronisessa kaupassa. Hen (2001, 83) mukaan monet yritykset ovat huomanneet, että monikielisen ja –kulttuurisen WWW-sivuston tekeminen on halpa tapa kasvattaa tuottoa kansainvälisessä elektronisessa kaupassa. Tästä syystä monikielisiä WWW-sivustoja tuotetaan yhä enemmän.

Monikielisyys aiheuttaa kuitenkin usein ongelmia WWW-sivustoja ja -järjestelmiä rakennettaessa. Varsinkin Itä-Euroopan ja Aasian kielet sisältävät merkkejä, jotka eivät kuulu WWW-selainten yleiseen oletusmerkistöön, ISO-8859-1:een. Kun WWW-sivustoilla halutaan käyttää näitä kieliä, on WWW-järjestelmän toteutukseen valittava merkistö, joka sisältää kyseisten kielten merkit. Kielten kuvaamiseen on olemassa monia vaihtoehtoisia merkistöjä, mutta WWW-selaimet eivät tue kaikkia niistä. Vaikka valittu merkistö siis kattaa halutun kielen, asiakas voi käytännössä nähdä selaimellaan pelkkiä tyhjiä neliöitä tai kysymysmerkkejä oikean tekstin sijaan. Toisaalta, vaikka merkistöksi valitaan esimerkiksi Unicode, joka on uusimpien WWW-selainten yleisesti tukema ja useimmat kirjoitetut kielet kattava merkistö, sekään ei välttämättä takaa, että teksti näkyisi käyttäjälle oikein. Jokin järjestelmän osista saattaa käsitellä merkistöä eri tavalla kuin pitäisi ja teksti muuttuu sen vuoksi lukukelvottomaksi. Tässä tutkielmassa käyn tarkemmin läpi millaisia ongelmia WWW-järjestelmissä voi olla merkistöjen kanssa ja esittelen mahdollisia ratkaisuja näihin ongelmiin. Selvitän myös mitä WWW-järjestelmän tuotantoprosessissa tulisi ottaa huomioon, jotta näiltä merkistöongelmilta vältyttäisiin.

1.1 Tutkimuksen tausta

Ohjelmistojen kansainvälistämisestä (*internationalization*) ja lokalisoinnista (*localization*) on jo olemassa jonkin verran kirjallisuutta, mutta varsinkin monissa artikkeleissa näkökulma on enemmän kulttuurieroja pohdiskeleva kuin teknisiin ongelmiin paneutuva. Esimerkiksi He (2001, 83-88) on tutkinut viiden kansainvälisen yrityksen kiinaksi lokalisoituja WWW-sivustoja. Hän on tullut siihen tulokseen, että pelkkä sivustojen kääntäminen toiselle kielelle ei riitä, vaan kulttuurilliset seikat pitäisi ottaa myös huomioon. Essenlink (2000) taas keskittyy teoksessaan lokalisoinnin kielellisiin ongelmiin, ei niinkään teknisiin ongelmiin, joita käyn läpi tässä tutkielmassa. Artikkelit, jotka

kuvaavat teknisiä ongelmia, keskittyvät yleensä yhteen järjestelmän osaan, kieleen tai merkistöön (esimerkiksi Graham keskittyy vain Unicodeen kirjassaan Unicode, A Primer). Kiinnostukseni aiheeseen lähti työelämän käytännön ongelmista, kun olin mukana kansainvälisen sovelluksen teossa. Kaipasin itse lähdettä, jossa merkistöjen aiheuttamista teknisistä ongelmista olisi kirjoitettu laajemmin. Tässä tutkimuksessa tarkoituksenani on etsiä tietoa eri merkistöjen käytön teknisistä vaikutuksista WWW-järjestelmien tuotantoon. Aihetta ei ole toistaiseksi vielä tutkittu paljon ja tieto on aika hajallaan eri lähteissä. Tämän tutkielman päätarkoitus onkin koota tietämys yhteen, jotta sitä on helpompi käyttää hyödyksi.

1.2 Tutkimusongelma ja tutkimusalueen rajaus

Tutkimusongelma voidaan kiteyttää seuraavasti: Millaisia ongelmia merkistöt aiheuttavat monikielisille WWW-järjestelmille ja miten näitä ongelmia voidaan ehkäistä monikielisen WWW-järjestelmän tuotantoprosessissa?

Tutkielma pyrkii vastaamaan myös seuraaviin kysymyksiin:

- Miksi kaikki merkit eivät näy WWW-käyttöliittymissä oikein ja miten tilannetta voidaan korjata?
- Mitä kansainvälisen WWW-järjestelmän suunnittelussa tulisi ottaa huomioon merkistöjen kannalta?

Tutkimuksessa rajaudutaan käsittelemään WWW-sivustojen lokalisointia ja kansainvälistämistä vain teknisestä näkökulmasta eri merkistöjen käytön kannalta. En paneudu lokalisoinnin muuhun problematiikkaan, kuten kulttuurillisiin ongelmiin tai käännöstyön vaikeuteen. Tutkimuksen ulkopuolelle jäävät myös kansalliset asetukset, kuten ajan ja päivämäärien muodot, sekä niiden vaihtamiseen liittyvät ongelmat lokalisoiduissa WWW-sivustossa.

Merkistöistä olen ottanut mukaan tutkielmaan vain muutaman, yleisesti tunnetun merkistön. Pääpaino on Unicodella. WWW-käyttöliittymien kohdalla ongelmia käydään läpi vain Microsoftin ja Netscapen selainten kannalta. Muita WWW-selaimia ei käsitellä. Muissakin kuin WWW-käyttöliittymissä saattaa olla vastaavanlaisia ongelmia merkistöjen kanssa, mutta tutkielman hallittavuuden vuoksi keskityn vain WWW-selaimilla katseltaviin WWW-käyttöliittymiin. Tutkielman ulkopuolelle jäävät siis muun muassa WAP- ja SMS-käyttöliittymät. Tiedonsiirtoprotokollista otan huomioon MIME:n, HTTP:n ja SMTP:n. Ohjelmistokielistä käsittelen vain muutamia WWW-sovelluksille yleisiä kieliä. Käyttöjärjestelmistä käyn läpi vain tunnetuimpia ja niitäkin hyvin yleisellä tasolla.

1.3 Tutkielman sisältö

Luvussa 2 määrittelen tutkielmassa käytetyt termit ja luvussa 3 esittelen tutkielmassani käyttämäni merkistöt ja enkoodaukset. Luvussa 4 selvitän merkistöjen käyttöön liittyviä yleisimpiä teknisiä ongelmia ja mahdollisia ratkaisuja ongelmiin. Viimeisessä sisältöluvussa eli luvussa 5 käyn läpi mitä WWW-järjestelmän tuotantoprosessin eri vaiheissa tulee ottaa huomioon, jotta luvussa neljä lueteltuja ongelmia olisi mahdollisimman vähän.

1.4 Tutkimusmetodit

Tutkielmani on kirjallisuuskatsaus, jota on täydennetty kahdella laadullisella haastattelulla. Haastattelut on suoritettu sähköpostihaastatteluina keväällä 2003. Haastattelun kysymykset ovat liitteenä (LIITE 1). Luvun 4 merkistöongelmien löytämiseen sain apua erityisesti Pasi Aaltosen haastattelusta. Aaltonen on ollut viimeisen kolmen vuoden ajan toteuttamassa suurta, kansainvälistä ja monikielistä WWW-järjestelmää ja hän on erikoistunut juuri merkistöongelmien ratkaisemiseen. Aaltonen on antanut paljon vinkkejä tutkielmaani myös haastattelun ulkopuolella. Ulla-Maija Kasatkin on

puolestaan ollut pitkään projektipäällikkönä projekteissa, joissa on luotu monikielisiä WWW-järjestelmiä. Nykyisin hän toimii arkkitehtinä kansainvälisessä projektissa. Kasatkinin tietämystä on hyödynnetty erityisesti luvussa 5 eli monikielisen WWW-järjestelmän tuotantoprosessin kuvaamisessa.

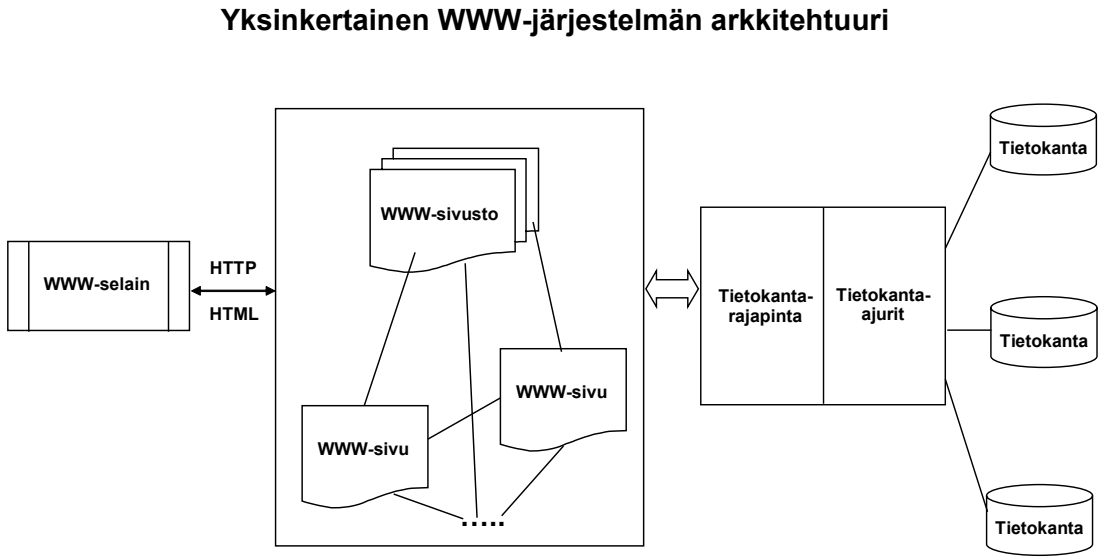
2 MÄÄRITELMIÄ

Tässä luvussa määrittelen lyhyesti termit, joita käytän tässä tutkielmassa.

WWW-sivu (*Web page*) on WWW-selaimella luettavissa oleva dokumentti. Dokumentti ei välttämättä ole staattinen HTML-sivu, vaan se voi olla myös dynaamisesti rakentuva, reaaliaikaisesti päivittyvä sivu. **WWW-sivusto** (*Website*), tai yksinkertaisimmin pelkkä **sivusto**, on taas WWW-sivuista koostuva looginen kokonaisuus tai palvelu. Tutkielmassani viittaan sanalla **käyttäjä** WWW-sivustoa WWW-selaimella lukevaan henkilöön. **WWW-järjestelmällä** (*Web system*) tai **järjestelmällä** tarkoitan tässä tutkielmassa muutakin kuin käyttäjälle näkyvää sivustoa. WWW-järjestelmään voi kuulua WWW-sivujen lisäksi esimerkiksi **http-palvelinohjelmisto** (*http-server software*), **sovelluspalvelinohjelmisto** (*Application server software*), sovelluspalvelimella ajettava sovellusohjelma eli **WWW-sovellus** (*Web application*), sovelluksen tarvitsema **sovellusalusta** sekä mahdolliset **tietovarastot** ja rajapinnat toisiin järjestelmiin. Http-palvelinohjelmisto lähettää WWW-sivut palvelimelta käyttäjälle. Sovelluspalvelinohjelmisto taas huolehtii WWW-sovelluksen ajosta sovellusalustalla ja yhteyksistä tietovarastoihin tai toisiin järjestelmiin. Joissain tuotteissa, kuten Bea:n WebLogic Serverissä, http-palvelinohjelmisto ja sovelluspalvelinohjelmisto on yhdistetty yhdeksi palvelintuotteeksi. Sovellusalusta sisältää muun muassa sovelluksen ohjelmointikielen tarvitsemat kirjastot. Esimerkiksi ohjelmointikieli Javan tämänhetkinen sovellusalusta on Java 2 Runtime Environment.

Kaaviossa 2 on määritelty yleinen, yksinkertainen WWW-järjestelmän arkkitehtuuri Lin ym. mukaan (2000). Tässä arkkitehtuurissa käyttäjä katselee WWW-selaimellaan WWW-sivuista koostuvaa sivustoa. Http-palvelinohjelmisto välittää WWW-sivut käyttäjän selaimelle HTML-muodossa HTTP-protokollan avulla. WWW-sovellus on tässä arkkitehtuurissa typistynyt vain tietokantarajapinnaksi. Sovelluspalvelinohjelmisto huolehtii WWW-

sovelluksen ajosta sovellusalustalla ja yhteyksistä tietokantoihin tietokanta-ajureita hyväksi käyttäen.

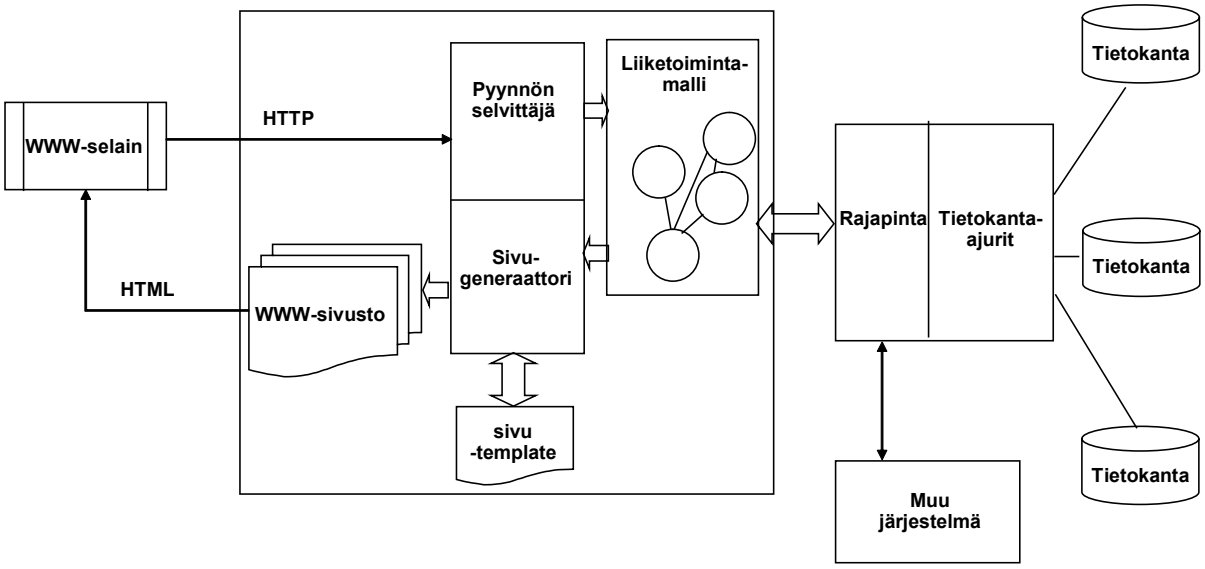


KAAVIO 2. Yksinkertainen WWW-järjestelmä (Li ym. 2000, 268)

Kaaviossa 3 on mallinnettu hieman monimutkaisempi WWW-järjestelmä. Tässä arkkitehtuurissa selain välittää sivupyynnöt palvelimelle HTTP:tä käyttäen. Palvelimella pyynnön selvittäjä (*request dispatcher*) eli http-palvelinohjelmisto tulkitsee pyynnön ja välittää sen eteenpäin WWW-sovelluksen liiketoimintayksikölle. Liiketoimintalogiikan sisältävän ohjelmakoodin ajosta huolehtii sovelluspalvelinohjelmisto. Sovelluspalvelinohjelmisto ottaa tarvittaessa yhteyden tietokantaan tietokantarajapinnan ja tietokanta-ajureiden kautta. Tietoa voidaan hakea myös muista erillisistä järjestelmistä WWW-sovellukseen määritellyn rajapinnan kautta. Kun tarvittava data on saatu, muodostetaan WWW-sivu WWW-sovelluksen sivugeneraattorin avulla, joka taas käyttää hyödyksi sivu-templatea eli sivumallia. Kun sivu on luotu, http-sovelluspalvelinohjelmisto lähettää sen HTML-muodossa käyttäjän selaimelle.

WWW-sovellukseen kuuluvat siis pyynnön selvittäjä, liiketoimintalogiikan sisältävä ohjelmakoodi, rajapinnat, tietokanta-ajurit ja sivugeneraattori. (Li ym. 2000, 267-270)

Monimutkaisempi WWW-järjestelmän arkkitehtuuri



KAAVIO 3. Monimutkaisempi WWW-järjestelmä (Li ym. 2000, 269)

Tutkielmassani WWW-järjestelmällä viittaan näihin molempiin arkkitehtuureihin. Tutkielman kannalta WWW-järjestelmän arkkitehtuuria tärkeämpää on se, että WWW-sivusto on **monikielinen** (*Multilingual*).

Monikielisen WWW-sivuston voi toteuttaa monella tavalla. Essenlinkin (2000) mukaan yksinkertaisin tapa tuottaa tällainen WWW-sivusto on käännättää suoraan kaikki sivuston tekstit halutuille kielille käännöstoimistossa tai käännöstyökaluilla. **Käännetyssä** (*Translation*) WWW-sivustossa ei tehdä sivustoon mitään paikallisia muutoksia, mutta itse tekstiä saatetaan muokata hieman paikalliselle kielelle sopivammaksi. Kun WWW-järjestelmä **lokalisoidaan** (*Localization, L10n*), käännettyä tekstiä muokataan ottamaan

huomioon kohdemaan tai -alueen kulttuuri ja tavat. Lokalisoinnissa esimerkiksi WWW-sivuston kuvat saatetaan muuttaa kohdealueelle paremmin sopiviksi ja jotkin sivuston sivut saatetaan jättää pois paikallisesta käännetystä versiosta. **Kansainvälistäminen** (*Internationalization*, *I18n*) on lähempänä pelkkää kääntämistä kuin lokalisointi. Essenlink määrittelee, että ”Kansainvälistäminen on prosessi, jossa tuotteesta tehdään mahdollisimman geneerinen siten, että se voi ilman uudelleensuunnittelua käsittää useampia kieliä ja kulttuureiden käyttötapoja. Kansainvälistäminen tapahtuu jo ohjelman suunnittelun ja dokumentin luonnin tasolla.” Hyvin kansainvälistetyssä WWW-järjestelmässä sivusto voidaan käännättää vaikka kaikille maailman kielille ja se toimii kaikkien kielten kanssa oikein. Toisaalta se ei myöskään loukkaa tai ärsytä minkään kulttuurin edustajia, vaan kaikki pystyvät käyttämään sivustoa paikallisilla tavoilla. Essenlink nostaa esille, että keskeistä kansainvälistyksessä on pystyä tukemaan tarvittavia merkistöjä ja kansallisia merkintätapoja esimerkiksi päivämäärille ja ajoille. (Essenlink 2000, 2-8) Keskityn tutkielmassani lähinnä kansainvälistämisen tuomiin ongelmiin.

Merkistö (*character set*) on kokoelma merkkejä, joka sisältää yhden tai useamman kielen kirjoittamiseen tarvittavat merkit (Turnbull 1999). Yksi tunnetuimpia merkistöjä on ASCII, joka sisältää kaikki englanninkielisen tekstin tuottamiseen tarvittavat merkit, mutta ei esimerkiksi suomenkieleen kuuluvia aakkosia å, ä ja ö. Siksi suomea kirjoitettaessa on käytettävä ASCII:n sijasta esimerkiksi ISO-8859-1 -merkistöä, joka sisältää nuo puuttuvat merkit. ISO-8859-1 merkistökään ei sisällä kaikkia maailman merkkejä. Siihen tarkoitukseen on luotu Unicode-merkistö.

Merkistöistä puhuttaessa puhutaan usein myös siitä, kuinka monta bittiä tarvitaan yhden merkin koodaamiseen. Tavun sijasta käytetään yleensä termiä **oktetti** (*octet*), koska tavu voi olla kuusi, seitsemän tai kahdeksan bittiä pitkä, mutta oktetti viittaa aina kahdeksan bitin joukkoon. Jaakohuhdan (2001) IT Ensyklopedian mukaan oktetti on ”esitystapa, jossa luvut esitetään 8 bitin

ryhmissä.” Lisäksi mainitaan, että oktetti on tiedonsiirron yksikkö pakettivälitteisessä verkossa, kuten Internetissä.

Enkoodaus (*encoding*) on tapa, jolla merkistön merkit laitetaan koneen ymmärtämään muotoon. Käytän tässä tutkielmassa merkistöjen koodaamisesta termiä enkoodaus (*encoding*), jotta se eroaa selvästi sanasta koodaus, jolla tarkoitan ohjelmakoodin tuottamista (*programming, coding*). Yhdelle merkistölle voi olla useampi kuin yksi enkoodausmuoto. Esimerkiksi Unicodea voidaan enkoodata joko niin, että jokainen merkki vie aina 16 bittiä (= kiinteämittainen enkoodaus) tai niin, että merkkiä kohden käytetään 8-32 bittiä (= vaihtelevanmittainen enkoodaus). Kun käytetään 16-bittistä kiinteämittaista enkoodausta kaikille Unicode-merkistön merkeille, niin merkit, jotka voidaan esittää yhden 8-bitin eli yhden oktetin avulla, saavat parikseen 8-bitin kokoisen ”tyhjän” oktetin. ”Tyhjää” oktettia sanotaan **vähemmän merkitseväksi tavuksi**. Enkoodaus määrittääkin käytetyn bittimäärän lisäksi myös sen, onko mahdollinen ”tyhjä” oktetti merkeissä ensimmäisenä (= little endian) vai viimeisenä (= big endian). (Graham 2000, 97-100)

Fontti (*font*) sisältää ulkoasun merkistön merkeille. Merkistöllä ja fontilla on siis kiinteä suhde toisiinsa. Merkistö määrittää, että merkin ’A’ näyttämiseksi fontista haetaan sille ulkoasu ’A’. Jos fontti määrittää kyseisen merkin ulkoasuksi ’£’, niin dokumentissa näytetään jokaisen ’A’-merkin paikalla ’£’. Tämä tietysti haittaa huomattavasti tekstin lukemista. Siksi merkistöt käyttävät yleensä tiettyjä standardoituja fontteja, kuten Arial. (Turnbull 1999)

Tiivistettynä voidaan sanoa, että merkistö on kokoelma merkkejä, joille fontti antaa ulkoasun. Enkoodaus on taas tapa, jolla merkit on laitettu koneen ymmärtämään muotoon.

3 MERKISTÖT JA ENKOODAUKSET

Tässä luvussa esittelen merkistöt, joita käsittelen tutkielman muissa osissa ja käyn läpi enkoodauksen eri muodot.

3.1 Merkistöt

Maailmassa on satoja eri merkistöjä. Merkistöt kattavat yhden tai useamman kielen kirjoittamiseen tarvittavat merkit. Ensimmäiset merkistöt, kuten ASCII, sisälsivät vain englannin kielelle tarpeelliset merkit, koska tämä oli tietotekniikan kehittäjien osaama kieli. Merkistöjen suppeuteen vaikutti myös se, että tietokoneet käsittelevät kaikkea dataa tavuina eli kuudesta kahdeksaan bittiin olevina yksiköinä. Tuolloin yhden tavun piti pystyä merkitsemään eksplisiittisesti yhtä kirjoitusmerkkiä, joten mahdollisia merkkipaikkoja oli käytettävissä enintään 256 kappaletta. Tämä tarkoitti, että yksi merkistö pystyi kerrallaan sisältämään enintään 256 eri merkkiä.

Kun tietotekniikka levisi myös muihin kuin englanninkielisiin maihin, tuli tarve kehittää uusia merkistöjä, jotka kattaisivat paikallisen väestön kielet. Tietokoneiden prosessointi oli vielä hidasta, joten uudet kielet haluttiin kuvata mahdollisimman suppealla bittimäärällä, jolloin datan käsittely oli mahdollisimman nopeaa. Tämä johti siihen, että eri merkistöissä samoille merkkipaikoille on määritelty eri merkit. Tästä on hyvänä esimerkkinä ISO-8859 -merkistöperhe. Prosessointitehon ja markkina-alueiden kasvaessa tuli tarve kehittää yhtenäinen merkistä, jolla voitaisiin kuvata kaikki maailman kielet. Tältä pohjalta syntyi Unicode.

Merkistöistä puhuttaessa on hyvä huomata, että samalla merkistöllä voi olla useampi nimi. Esimerkiksi ISO-8859-1 tunnetaan myös nimellä Latin-1. Merkistöjen nimet ovat Internet-protokollien parametriarvoja, joiden määrittämistä koordinoi keskusorganisaatio IANA (*Internet Assigned Numbers*

Authority). Päivitetty lista rekisteröidyistä nimistä on nähtävissä IANA:n WWW-sivuilla (IANA 2002).

3.1.1 ASCII

ASCII (*American Standard Code for Information Interchange*) on yksi vanhimmista ja käytetyimmistä merkistöistä. ASCII sisältää kaikki englantiin kuuluvat merkit. ASCII:sta on monta eri versiota, jotka eroavat hieman toisistaan joidenkin merkkien osalta. Järvinen (1996) listaa kaksi US-ASCII versiosta poikkeavaa ASCII–merkistöä. Joissakin UNIX ja MS-DOS koneissa yhä vielä käytetyssä ASCII:ssa ”ä” ja ”ö” merkit on sijoitettu haka- ja kaarisulkeiden tilalle. Toinen versio ASCII:sta on IBM:n PC-koneissa käyttöönotettu versio, jossa alkuperäinen 7-bittinen ASCII on laajennettu kahdeksanbittiseksi, jotta se käsittää enemmän kansallisia merkkejä. Tässä tutkielmassa viittaan ASCII-sanalla ”aitoon ja alkuperäiseen” vuonna 1968 rekisteröityyn ASCII-versioon, joka tunnetaan nimillä US-ASCII ja ANSI_X3.4-1968 (IANA 2002). Tämän ASCII:n on luonut ANSI (*American National Standards Institute*), joka laatii Yhdysvaltain kansallisia standardeja (Jaakohuhta 2001).

ASCII on 7-bittinen merkistö eli yhden merkin kuvaamiseen käytetään seitsemän bittiä tilaa. Tästä johtuen ASCII käsittää vain 128 merkkiä. Vain 94 näistä on kirjoitukseen käytettäviä merkkejä eli kirjaimia, numeroita, matemaattisia symboleja ja välimerkkejä. Loput ovat tarkoitettu tietokoneohjelmien toimintaa sääteleviksi komentomerkeiksi, joita ei voi tulostaa näytölle näkyviin. Vaikka nykyiset ohjelmat eivät näitä komentomerkkejä käytäkään, niitä ei silti ole korvattu ASCII:ssa uusilla merkeillä, vaikka siten saataisiin ASCII kattamaan useampia kieliä. Syyksi tähän Turnbull (1999) mainitsee, että maailmassa on jo niin paljon ohjelmia, jotka pitävät näitä kontrollimerkkejä varattuina merkkeinä, että kukaan ei vakavissaan harkitse näiden uudelleenallokointia joillekin uusille merkeille.

Toisaalta asiaan voi myös vaikuttaa huomatut yhteensopivuusongelmat jo olemassa olevien ASCII-versioiden kanssa.

Monissa standardoiduissa merkistöissä ASCII on otettu merkistöön mukaan sellaisenaan. Esimerkiksi Unicodessa ensimmäiset 128 merkkiä ovat täysin samat kuin ASCII:ssa. ASCII on niin laajalti käytetty merkistö, että uudemmat merkistöt on haluttu tehdä ASCII-yhteensopiviksi. ASCII:n kanssa toiminut ohjelmakoodi toimii siis myös näiden muiden merkistöjen kanssa.

3.1.2 ISO/IEC 8859 -merkistöperhe

ISO/IEC 8859 -merkistöperhe syntyi, koska ASCII:ta ei haluttu muuttaa, mutta oli tarvetta laajentaa merkistöä kattamaan useampia Euroopan kieliä. Merkistöt ovat ISO/IEC:n (*International Standards Organization / International Electrotechnical Commission*) standardoimia. Turnbull (1999) listaa kolme tavoitetta, jotka olivat tärkeimpiä ISO 8859 standardimerkistöä luotaessa: 1) merkistön tuli olla ASCII-yhteensopiva, 2) merkistön tuli kattaa mahdollisimman monta kieltä yhden oktetin avulla ja 3) merkistön piti noudattaa ISO 2022 standardia. ASCII-yhteensopivuus tarkoitti sitä, että merkistön täytyi sisältää koko ASCII-merkistö ja merkkien järjestys täytyi olla sama. Yhdessä oktetissa pitäytyminen tarkoitti taas sitä, että merkistössä oli tilaa vain 256 merkille. Tämä rajoite oli ongelmallinen, koska 256 merkkiä ei riittänyt edes kaikille eurooppalaisille kielille puhumattakaan muista kielistä. Koska yhden oktetin rajoitteesta ei kuitenkaan haluttu luopua, päätettiin luoda useampia merkistöjä, jotka kaikki sisälsivät ASCII:n merkistön merkkiavaruuden alussa. ASCII:n jälkeen tulevilla 128 merkkipaikoilla on joka merkistössä eri merkit. ISO 2022-standardi tarjosi tekniikan, jonka avulla näitä kaikkia ISO-8859 –merkistöjä pystyttiin käyttämään yhdessä kuin yhtenä monikielisenä merkistönä. ISO 2022-standardi on esitelty tarkemmin enkoodausten yhteydessä.

Ensimmäinen ISO/IEC 8859-merkistöperheeseen kehitetty merkistö oli ISO-8859-1, joka tunnetaan myös nimellä Latin-1. ISO-8859-1 -merkistö sisältää merkit suurimpaan osaan länsieurooppalaisista kielistä. ISO-8859-1 on merkistöperheen tunnetuin merkistö, siksi että se otettiin oletusmerkistöksi monissa ohjelmistotuotteissa, kuten WWW-selaimissa. Siinä asemassa se on vieläkin ainakin Microsoftin ja Netscapen WWW-selaimissa. Czyborra (1998) varoittaa, että ISO-8859-1 –merkistöä ei pidä sekoittaa Microsoftin omaan Windows Latin1 (code page 1252) merkistöön, joka eroaa ISO standardin Latin-1 merkistöstä.

Tällä hetkellä ISO/IEC 8859 –merkistöperheeseen kuuluu 15 toteutettua merkistöä, jotka on lueteltu alla olevassa taulukossa (TAULUKKO 1). Olen koonnut taulukkoon ISO:n WWW-sivuilla (ISO 2002) myynnissä olevat viimeisimmät versiot näistä merkistöstandardeista. Lisäsin taulukkoon myös esimerkkejä merkistöllä kirjoitettavista kielistä Czyborran mainitsemien esimerkkien pohjalta. Huomattavaa on, että 8859-12 merkistö puuttuu kokonaan. Czyborran (1998) mukaan se saattaa olla varattuna Intiassa käytettävälle ISCII-merkistölle.

Jokainen näistä merkistöistä toteuttaa Turnbullin (1999) listaamat tavoitteet. Ensimmäiset 128 merkkiä ovat kaikissa perheen merkistöissä täysin samat kuin ASCII:ssa. Loput 128 merkkiä on käytetty eri kielten kuvaamiseen ja ne sisältävät eri merkit jokaisessa eri merkistössä. Suomenkieleen kuuluvat aakkoset å, ä ja ö ovat ISO-8859-1:ssa. Niiden paikoilla ISO-8895-5:ssa ovat merkit χ, φ ja i.

TAULUKKO 1. ISO-8859 -merkistöt

Merkistö	Aakkoset	Kielet
ISO/IEC 8859-1:1988	Latinalaiset aakkoset nro 1	Länsieurooppalaiset kielet (mm. ranska, espanja, portugali, italia ja suomi)
ISO/IEC 8859-2:1999	Latinalaiset aakkoset nro 2	Itäeurooppalaiset kielet (mm. tšekki, unkari ja puola)
ISO/IEC 8859-3:1999	Latinalaiset aakkoset nro 3	Eteläeurooppalaiset kielet (mm. malta)
ISO/IEC 8859-4:1998	Latinalaiset aakkoset nro 4	Pohjoiseurooppalaiset kielet (mm. Baltian maiden kielet)
ISO/IEC 8859-5:1999	Latinalaiset ja kyrilliset aakkoset	Venäjä
ISO/IEC 8859-6:1999	Latinalaiset ja arabialaiset aakkoset	Arabia
ISO 8859-7:1987	Latinalaiset ja kreikkalaiset aakkoset	Kreikka
ISO/IEC 8859-8:1999	Latinalaiset ja heprealaiset aakkoset	Heprea
ISO/IEC 8859-9:1999	Latinalaiset aakkoset nro 5	Turkki
ISO/IEC 8859-10:1998	Latinalaiset aakkoset nro 6	Inuit ja saame
ISO/IEC 8859-11:2001	Latinalaiset ja thai aakkoset	Thai
ISO/IEC 8859-13:1998	Latinalaiset aakkoset nro 7	Latvia
ISO/IEC 8859-14:1998	Latinalaiset aakkoset nro 8	Kelttiläiset kielet, kuten wales ja gaeli
ISO/IEC 8859-15:1999	Latinalaiset aakkoset nro 9	ISO-8859-1:n kielet, paitsi muutamat suomen ja ranskan kielen merkit poistettuna
ISO/IEC 8859-16:2001	Latinalaiset aakkoset nro 10	ISO-8859-1:n kielet + euro-merkki (€)

3.1.3 Unicode ja UCS

Unicode Consortium (2000) luettelee kolme pääongelmaa lähtökohdiksi Unicoden luonnille. Ensinnäkin fontit olivat ongelma, kun eri merkistöissä sama merkkikoodi tarkoitti eri merkkiä. Jokaiselle merkistölle piti siis olla oma fonttinsa. Toisaalta oli olemassa paljon paikallisia merkistöjä ja niille

teollisuudessa luotuja vastineita, jotka erosivat vain hieman toisistaan, mikä aiheutti paljon sekaannusta. Esimerkkinä Unicode Consortium nostaa esiin jo Czyborrankin (1998) mainitseman ISO-8859-1 (Latin-1) ja Microsoftin Windows Latin 1 (code page 1252) merkistöjen eron. Kolmantena Unicode Consortium tuo esiin ASCII:n ja Aasian kielten kahta tai useampaa oktettia merkkiä kohden käyttävien merkistöjen erottamisen vaikeuden. Internetin suuri suosio ja nopea leviäminen vain lisäsivät tarvetta yhdelle yhtenäiselle merkistölle, jonka avulla yhdellä WWW-sivulla voidaan näyttää sekaisin minkä kielistä tekstiä tahansa. WWW-sivulla kun ei voida näyttää kuin yhden merkistön merkkejä kerrallaan.

Näistä syistä johtuen jo 1980-luvulla alettiin suunnitella yhtenäistä merkistöä, jolla voitaisiin kuvata kaikki mahdolliset kirjoitettavat merkit. Tehtävää alettiin ratkoa kahdella taholla; ISO muodosti JTC1/SC2/WG2-työryhmän vuonna 1984 ja samoihin aikoihin joukko yhdysvaltalaisia yrityksiä lyöttäytyi yhteen asian tiimoilta (Adams 1993). ISO:n työryhmän työn tuloksena syntyi aluksi Unicode Han Character Set vuonna 1989. Sen pohjalta työryhmä tuotti UCS (*Universal Multiple-Octet Coded Character Set*) merkistön. Yhdysvaltalaisten yritysten ryhmittymä, Unicode Consortium, taas nimesi oman merkistönsä Unicodeksi. Unicoden ensimmäinen versio julkaistiin vuonna 1991. Samaan aikaan ISO:n työryhmä esitti ISO 10646 standardin ensimmäisen version, joka kuvasi UCS:n. Adamsin mukaan sitä ei hyväksytty, koska enemmistö kokouksen äänestäjistä pelkäsi kahden melkein samanlaisen merkistön, Unicoden ja ISO 10646:n, aiheuttavan vain ongelmia ja yhteensopimattomuutta tuotteissa. Tämän jälkeen ISO ja Unicode Consortium alkoivat tehdä yhteistyötä ja yhdistää näitä kahta uutta merkistöä. Tuloksena syntyi ISO 10646 standardi vuonna 1992 ja vuonna 1993 Unicode versio 1.1. Unicode 1.1 oli merkeiltään ja niiden paikkamäärityksiltään täysin yhteneväinen ISO 10646 standardin kuvaaman UCS-merkistön kanssa. Yhteistyö on jatkunut ja Unicode sekä UCS ovat yhä merkeiltään yhteneväisiä. Merkistöjen ero onkin enkoodauksissa. (Graham 2000, 3-17; Adams 1993, 34-36)

ISO/IEC 10646:n määrittelemät enkoodaukset UCS:lle ovat (The Unicode Consortium 2000):

- UCS-2: Jokainen merkki ilmaistaan kahdella oktetilla. Tästä syystä UCS-2 käsittää vain 65536 merkkiä.
- UCS-4: UCS-2:n laajennus, jossa jokainen merkki ilmaistaan neljällä oktetilla. UCS-4:ssa on tilaa 1112064 merkille.

Unicode Consortium (2000) on määritellyt seuraavat Unicoden enkoodaukset:

- UTF-7: Tiedonsiirtoa varten luotu muoto. Jokainen Unicode-merkki muutetaan okteteiksi, joissa tieto on ilmaistu 7-bittisinä ASCII-merkkeinä.
- UTF-8: Jokainen merkki voidaan ilmaista käyttämällä yhdestä neljään oktettia. Sisältää ASCII:n sellaisenaan. Pystyy esittämään kaikki UCS-4:n merkit, mutta vaatii vähemmän tilaa.
- UTF-16: UCS-2:n laajennus siinä mielessä, että 65536 ensimmäistä merkkiä ilmaistaan kahdella oktetilla ja loput neljällä.
- UTF-32: Sama kuin UCS-4.

Näistä enkoodauksista kaikkein yleisimmin on käytössä UTF-8, koska sen avulla pystyy esittämään yksikäsitteisesti suurimman mahdollisen määrän merkkejä pienimmällä mahdollisella oktettimäärällä.

Merkistöissä on vielä tilaa uusille merkeille. Unicode ja UCS merkistöjä päivitetään kattamaan uusia kieliä ja merkkejä koko ajan (Graham 2000, 77). Kirjoitushetkellä viimeisin versio Unicodesta on 3.2.

3.1.4 Paikalliset merkistöt

Koska jo ennen Unicodea oli tarvetta kirjoittaa tekstiä muillakin ASCII-merkistön merkeillä, syntyi paljon paikallisia merkistöjä. Paikalliset merkistöt sisältävät yleensä yhden tai muutaman kielen tarvitsevat merkit. Muutamat näistä merkistöistä saavuttivat niin vakaan aseman, että ne ovat vieläkin laajalti käytössä. Esimerkkeinä tunnetuista ja vieläkin käytössä olevista paikallisista merkistöistä voidaan mainita KOI-8 (venäjä), Big5 (perinteinen kiina), GB2312

(yksinkertaistettu kiina) ja Shift-JIS (japani). ISO-8859 –merkistöperhe kuuluu myös paikallisiin merkistöihin.

Turnbullin (1999) mukaan monet tunnetuista paikallisista merkistöistä ovat ASCII-yhteensopivia, mutta ne eivät yleensä toteuta ISO 2022 enkoodausstandardia. Tästä syystä paikallisella merkistöllä kirjoitetun tekstin siirtäminen järjestelmien välillä on usein hankalaa. Yksi päätahoista, joka luo ja rekisteröi ahkerasti tällaisia merkistöjä, on Microsoft. Microsoftin Windowsia varten luomat merkistöt eivät toteuta ISO 2022 standardia (Turnbull 1999). Adams (1993, 33-34) näkee taas syyksi sen, että vaikka ISO 2022 standardi mahdollistaakin merkkien vaihdon eri merkistöjen välillä, se tuo huomattavia ongelmia tällaisen tekstin prosessointiin.

3.2 Enkoodausmuodot

Yhtä merkistöä voidaan enkoodata monella eri tavalla. Kuten jo aiemmin tuli ilmi, enkoodauksella voidaan määrittää käytettyjen tavujen määrä merkkiä kohden. Graham (2000, 79) määrittelee, että ”enkoodauksessa koodattu merkistö yhdistyy tiettyyn joukkoon yhtenäisen kokoisia tietoyksiköitä”. Tietoyksiköillä voidaan tarkoittaa, bittejä, tavuja tai oktetteja. Turnbull (1999) taas määrittelee enkoodauksen olevan ”abstraktin merkin yhdistämistä yhteen tai useampaan oktettiin”. Turnbullin määritelmä on muuten hyvä, mutta se ei ota seitsemänbittisiä enkoodauksia huomioon, joita käytetään aika yleisesti. Kumpikaan näistä määritelmistä ei ota huomioon, että enkoodauksella voidaan ilmaista myös merkin sisältämien oktettien tavujärjestys. Tavujärjestyksellä on merkitystä silloin, kun merkistön kaikkia merkkejä ei voida ilmaista yhdellä tavulla.

3.2.1 Kiinteämittainen ja vaihtelevamittainen enkoodaus

Kiinteämittaisessa enkoodauksessa jokainen merkistön merkki vie saman verran tilaa tavuina. Esimerkkejä kiinteämittaisista enkoodauksista ovat ASCII ja UTF-32. ASCII:ssa jokaisen merkin koodaamiseen käytetään seitsemän bittiä. UTF-32:ssa taas jokainen merkki ilmaistaan neljällä oktetilla eli 32 bitillä.

Vaihtelevamittaisessa enkoodauksessa käytetään eri määrä oktetteja eri merkkien ilmaisuun. UTF-8 ja UTF-16 ovat esimerkkejä vaihtelevamittaisista enkoodauksista. Unicode-merkistöä koodattaessa UTF-8 enkoodauksella, jokaista merkkiä kohden käytetään yhdestä neljään oktettia. Merkistön alkupäässä olevat ASCII merkit ilmaistaan yhden oktetin avulla, kun taas merkistön loppupäässä olevat mongolian merkit vievät neljä oktettia. UTF-16 enkoodauksessa yhden merkin koodaamiseen käytetään joko aina kaksi tai neljä oktettia. ASCII merkit vievät UTF-16 enkoodauksessa kaksi oktettia tilaa. UTF-8 merkistöä käyttämällä voi siis säästää paljon tilaa UTF-16 tai UTF-32 enkoodauksiin verrattuna, jos suurin osa tekstin merkeistä kuuluu Unicode-merkistön alkupäähän. Mongoliaa kirjoitettaessa taas kaikki kolme enkoodausta käyttävät saman verran oktetteja. (Graham 2000, 35-90)

3.2.2 Big endian ja little endian

Merkistöt, joissa on enemmän kuin 256 merkkiä, tarvitsevat osalle merkeistä useampaa kuin yhtä oktettia. Kun näille merkistöille käytetään kiinteämittaista enkoodausta, niin kaikki merkit halutaan ilmaista samalla määrällä oktetteja. Silloin merkit, jotka voitaisiin ilmaista yhden oktetin avulla, saavat seurakseen yhden tai useamman tyhjän oktetin. Tyhjät oktetit voidaan laittaa joko tietoa sisältävän oktetin eteen tai taakse. Tietoa sisältävää oktettia sanotaan tässä tapauksessa merkitsevämmäksi tavuksi. Big endian –enkoodauksessa merkitsevämpi tavu on ensimmäisenä ja mahdolliset tyhjät oktetit ovat sen

perässä. Vastaavasti little endian –enkoodauksessa merkitsevämpi tavu on viimeisenä.

Esimerkki 1 kuvaa kuinka heprean kirjain BET (ב, heksakoodina 0x05D1) tallennetaan big endian and little endian –muodoissa eri Unicoden enkoodauksilla. Esimerkki on otettu Grahamin Unicode, A Primer –kirjasta (Graham 2000, 99). (Heksakoodin 0x jätetään merkitsemättä.)

ESIMERKKI 1. Heprean BET-kirjaimen merkintä eri Unicoden enkoodauksilla

UTF-16 big endian:	05 D1
UTF-16 little endian:	D1 05
UTF-32 big endian:	00 00 05 D1
UTF-32 little endian:	D1 05 00 00

Esimerkistä voimme päätellä, että little endian ja big endian –enkoodaukset vaikuttavat siis kaikkiin merkistön merkkeihin, eikä vain niihin, joissa on tyhjiä oktetteja. Siksi enkoodausta voi kutsua myös tavujärjestykseksi.

ISO/IEC 10646-1:1993 ja The Unicode Standard, versio 3.0 tukevat molempia tavujärjestyksiä samanveroisina (The Unicode Consortium 2000). Tavujärjestyksen tietäminen on erityisen tärkeää, kun siirretään tietoa little endian –järjestelmästä big endian –järjestelmään tai päinvastoin. Grahamin (2000) mukaan little endian –järjestelmiä ovat Intelin prosessorin sisältävät tietokoneet. Motorolan PowerPC:n prosessorit taas käyttävät big endian –muotoa.

Tavujärjestyksen tunnistamisessa käytetään **BOM**-merkkiä (*Byte Order Mark*), jonka heksa-arvo on FEFF. BOM tunnetaan myös nimellä zero-widht non-breaking space. Big endian –muodossa BOM on tallennettu muotoon FEFF ja little endian –muodossa FFFE. FFFE ei kuulu Unicode-merkistöön ja saadessaan sen ensimmäisenä merkkinä järjestelmä tietää, että data on little endian -

muodossa. Jos datan vastaanottava järjestelmä on big endian –enkoodausta käyttävä, se tietää BOM-merkin saadessaan, että tavut täytyy kääntää toiseen järjestykseen ennen datan muuta käsittelyä. (Raggett, Le Hors & Jacobs 1999) Esimerkissä 2 näkyy BOM-merkin käyttö heprean BET-merkin kanssa (Graham 2000, 99). Esimerkissä 1 on vastaavat BET-merkit ilman BOM-merkkiä.

ESIMERKKI 2. BOM-merkin käyttö Unicoden eri enkoodausten kanssa

UTF-16 big endian:	FE FF 05 D1
UTF-16 little endian:	FF FE D1 05
UTF-32 big endian:	00 00 FE FF 00 00 05 D1
UTF-32 little endian:	FF FE 00 00 D1 05 00 00

Unicoden käyttö on lisääntynyt Internet sivustoilla huimasti viime vuosina. Koska sivuston tekijä ei voi tietää millaisella koneella sivustoa tullaan katselemaan, on W3C antanut Unicoden käyttöä koskevia suosituksia. Kun HTML sivulla käytetään UTF-16 -enkoodausta, W3C:n HTML 4.01 spesifikaation (1999) mukaan data pitäisi siirtää verkossa big endian –muodossa. Jotta vältyttäisiin lisäsekaannuksilta, spesifikaatio suosittelee käyttämään myös BOM:ia HTML dokumenttien alussa. (Raggett, Le Hors & Jacobs 1999)

3.2.3 ISO 2022 standardi

ISO 2022 on enkoodausstandardi, joka tarjoaa tavan laajentaa ISO-organisaation 7- ja 8-bittiset merkistöt käsittämään useampia kieliä. Adams (1993, 31-33) kertoo standardin syntyneen tarpeesta kuvata yhden tavun merkistöillä useampia kieliä kuin mihin tavun määrittämät merkkipaikat riittivät. Tämä oli tilanne esimerkiksi ISO-8859 -merkistöjen kanssa. Standardi tarjoaa mekanismin vaihtaa merkistön merkkejä toisen merkistön merkkeihin tekstin keskellä. ISO 2022 –standardin avulla siis käytännössä voidaan yhdellä WWW-sivulla

näyttää tekstiä, joka koostuu yhtä aikaa ISO-8859-1 ja ISO-8859-2 –merkistöjen merkeistä. (Tässä tapauksessa merkeillä ei tarkoiteta ASCII:n merkkejä, jotka ovat molemmissa ISO-8859 -merkistöissä samoilla merkkipaikoilla.)

ISO 2022:n mukaan (1994) merkit jaetaan **merkistöelementteihin** (*code element*). Käytettävä elementti voidaan vaihtaa toiseen, joka sisältää samoilla koodipaikoilla eri merkkejä. Käytännössä vaihto toteutetaan **ohjausmerkkijonojen** (*escape sequences*) avulla. ISO 2022 -standardissa määritellään ohjausmerkkijonojen olevan ”jono bitti-yhdistelmiä, jota käytetään merkistön laajentamisessa.” Ohjausmerkkijonot alkavat aina ESCAPE merkillä, saattavat sisältää muutaman tavun verran bittejä (*Intermediate Bytes*), joilla ilmoitetaan mihin merkistöelementtiin vaihdetaan ja päättyvät ohjausmerkkijonon lopettavaan tavuun (*Final Byte*). ISO 2022 -standardin toteuttavissa merkistöissä ESCAPE, SPACE ja DELETE merkkien paikat on tarkasti määritelty, koska niitä käytetään vaihdossa kontrollimerkkeinä. Paikallisista merkistöistä monet kiinalaiset ja japanilaiset merkistöt (mm. GB2312, JISX 0208) ovat ISO 2022 -standardin mukaisia (Kubota 2002).

4 TEKNISET MERKISTÖONGELMAT JA NIIDEN MAHDOLLISET RATKAISUT

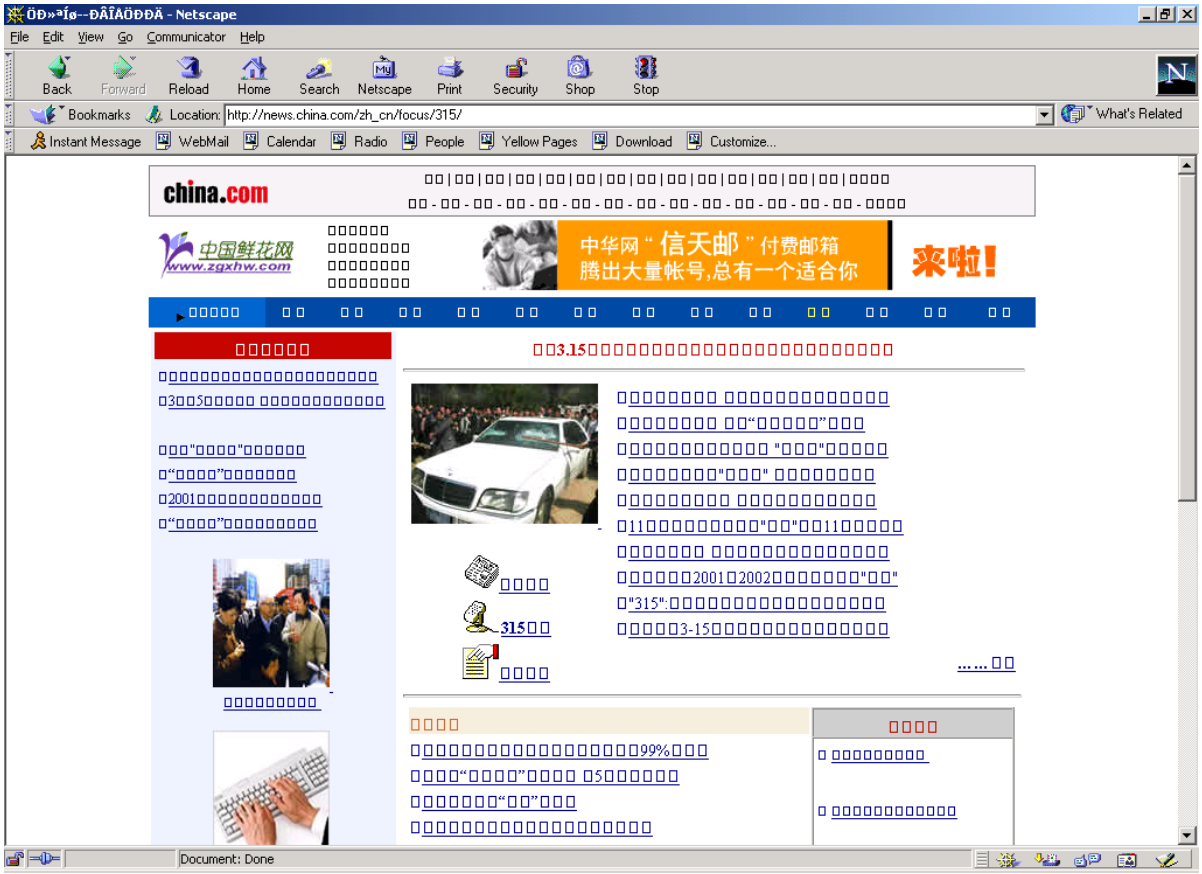
HTTP/1.1 (*Hypertext Transfer Protocol*) -protokollassa (1999) määritellään ISO-8859-1:n olevan oletusmerkistö, jos mitään muuta merkistöä ei ole määritelty. Tästä johtuen ASCII ja ISO-8859-1 -merkistöillä kirjoitetut WWW-dokumentit näkyvät selaimilla useimmiten ongelmitta. Ulla-Maija Kasatkin tuo haastattelussa esille, että merkistöongelmat koskevat yleensä merkkejä, jotka eivät kuulu ISO-8859-1 -merkistöön. Merkit voivat näkyä väärin joko kaikissa tai vain joissain tietyissä käyttöympäristöissä. Kasatkinin mukaan merkistöongelman vakavuus voi vaihdella pelkästä kosmeettisesta harmista siihen, että järjestelmän toimintoja on mahdoton käyttää. WWW-sivulla teksti voi pahimmillaan näkyä kokonaan pelkkinä tyhjinä neliöinä, kysymysmerkkeinä tai lukukelvottomana merkkisotkuna. Hieman lievemmissä tapauksissa vain osa merkeistä on korvautunut väärillä merkeillä, mutta teksti on silti luettavissa. WWW-järjestelmän käytettävyyttä haittaa myös huomattavasti se, jos WWW-sivusto sallii käyttäjän syöttää erikoismerkkejä, mutta käsittelee niitä väärin. Esimerkiksi käyttäjätunnuksen ja salasanan kohdalla tämä on vakava ongelma (esimerkiksi käyttäjä voi rekisteröityä tunnuksella, joka muuttuu järjestelmän tallentaessa sen).

Pasi Aaltonen mainitsi haastattelussa monia merkistöongelmia, joita hän on huomannut toteuttaessaan monikielisiä WWW-järjestelmiä. Tässä luvussa käsittelen kirjallisuuden pohjalta teknisiä syitä sekä hänen mainitsemiinsa että kirjallisuudesta esiin tulleisiin merkistöongelmiin. Pääajatuksenani on selvittää, miksi kaikki merkit eivät näy oikein WWW-käyttöliittymissä ja miten tilannetta voidaan korjata. Aluksi käsittelen WWW-selainten tekstin näyttämiseen liittyviä ongelmia. Näitä ovat esimerkiksi fonttiongelmat ja selaimen väärät asetukset. Näiden jälkeen esittelen tiedonsiirtoon liittyviä ongelmia, kuten puutteellisten tai virheellisten merkistöparametrien käyttöä dokumentissa. Lisäksi käsittelen WWW-käyttöliittymästä tapahtuvaan tekstinsyöttöön liittyviä

merkistö- ja enkoodausongelmia. Lopuksi siirryn WWW-sovelluksen sisäisiin ongelmiin ja WWW-järjestelmän osioiden yhteentoimivuusongelmiin.

4.1 Fontit

Suurin osa näkyvyysongelmista eri käyttöliittymissä johtuu siitä, että järjestelmässä ei ole tarvittavaa fonttia. Jos fontista puuttuu tietyn merkin vastine, selain näyttää kyseisen merkin tilalla oletusmerkin, joka on järjestelmästä riippuen joko tyhjä neliö (□), pystysuora viiva (|) tai matomerkki eli tilde (~). (Shabalin 2002) Yleensä fontin puuttuessa kaikki WWW-sivun sisältämät merkistöön kuuluvat merkit näkyvät neliöinä (ks. KUVA 1).



KUVA 1. Puuttuva fontti merkistölle GB2312, selaimena Netscape Communicator 4.78

Kuvassa 1 oikein näkyvät kiinalaiset merkit ovat kuvia, eivät varsinaista tekstiä. Muiden merkkien kohdalla asia korjaantuu, kun tarvittava fontti-tiedosto

tallennetaan käyttäjän koneeseen siihen hakemistoon, josta WWW-selain sitä etsii (ks. KUVA 2).

KUVA 2. Fontti saatavilla merkistölle GB2312, selaimena Microsoft Internet Explorer 5.5.

Pulmallisinta onkin tietää, mistä WWW-selain yrittää fonttia hakea ja mitä fonttia se etsii. Uusimmat versiot Microsoftin Internet Explorer -selaimista ovat tässä suhteessa helppokäyttöisiä. Ne ilmoittavat käyttäjälle, kun huomaavat tarvittavan fontin puuttuvan ja tarjoavat käyttäjälle mahdollisuuden ladata puuttuvan fontin Microsoftin ilmaisesta fonttivalikoimasta. Näin ladattuna fontti menee suoraan oikeaan paikkaan käyttöjärjestelmässä ja WWW-selain löytää sen. Netscapen WWW-selaimet eivät ilmoita fontin puuttumisesta. Netscapen tuen (2000) mukaan Netscapenkin selainten kohdalla tilanteen korjaa fontin lataus oikeaan paikkaan käyttöjärjestelmää. Kuitenkin vaikka fontti on ladattu käyttöjärjestelmään ja Microsoftin Internet Explorer pystyy

käyttämään sitä, Netscapen Communicator ei sitä käytä. Todisteena tilanteesta ovat kuvat 1 ja 2. Kuvat on otettu samalta sivustolta käyttäen samaa tietokonetta (käyttöjärjestelmänä Windows 98), mutta eri selainta. Tästä nähdään, että Microsoftin Internet Explorer löytää koneelta fontin merkistölle GB2312, mutta Netscapen Communicator 4.78 ei. Kokeiluni perusteella sama ilmiö on havaittavissa muillakin Microsoftin (Internet Explorer 4.7, 5.0, 5.5, 6.0) ja Netscapen (Communicator 4.7, 4.78) selainversioilla Microsoftin käyttöjärjestelmissä (Windows 95, 98, 2000, NT 4). Kuitenkin Netscape 6.1 – selain näyttää sivun oikein. Netscapen tuen sivut eivät anna suoraa selitystä tälle ilmiölle. Selityksenä voi mielestäni olla, että joko Netscapen vanhat, mutta vielä laajalti käytössä olevat selaimet ja Microsoftin selaimet hakevat fontteja eri paikasta käyttöjärjestelmää tai Netscapen vanhemmat selaimet eivät jostain syystä pysty käyttämään tarjolla olevaa fonttia.

Fontit ovat käyttöjärjestelmäkohtaisia. Esimerkiksi Microsoftilla on laaja valikoima ilmaisia fontteja sekä Windows että Mac -ympäristöihin, jotka eivät toimi esimerkiksi UNIX-ympäristössä. Fontit eivät kuitenkaan ole WWW-selainkohtaisia, joten muutkin kuin Microsoftin omat selaimet pystyvät käyttämään näitä fontteja. Netscapen tuen sivuilla kehotetaankin käyttäjiä lataamaan puuttuvia fontteja Microsoftin sivuilta, kun käyttäjällä on Windows-käyttöjärjestelmä. Toisena hyvänä latauspaikkana se suosittelee Oregonin yliopiston alaista Yamada Language Centeriä. Nämä eivät tietenkään ole ainoita fonttien latauspaikkoja, mutta näistä paikoista ladatut fontit noudattavat standardeja. Netscape varoittaakin käyttäjiä muista fonteista, jotka on saatettu toteuttaa vain osittain tai muuten väärin. Ne eivät välttämättä toimi selaimen kanssa ollenkaan, jolloin selain ei näytä tekstiä WWW-sivulla tai teksti on lukukelvotonta. (Netscape 2000)

Kun käyttöliittymässä halutaan näyttää paljon tekstiä, johon ei ole fonttia, Netscape tarjoaa ratkaisuksi dynaamisia fontteja. Dynaamisten fonttien idea on siinä, että sovelluksen tekijä voi liittää HTML-dokumentin mukaan .pfr-

tiedoston, joka sisältää fontit kyseisen dokumentin merkeille. Dynaamiset fontit toimivat Netscape Communicator 4.x selaimen Windows, Mac ja UNIX -versioissa sekä Microsoft Internet Explorer selaimen versiosta 4 ylöspäin. Netscapen Communicator 6 ei kuitenkaan tue dynaamisia fontteja. (Bitstream 2002, Netscape 2002, Netscape 2000) Tästä syystä dynaamisia fontteja ei voida pitää loistavana ratkaisuna WWW-sovelluksissa, mutta ideaa voi käyttää hyväksi sovelluksissa, joilla on jokin muu käyttöliittymä.

Tekstin lukeminen vaikeutuu tai muuttuu mahdottomaksi, jos fonttiin on toteutettu väärät ulkoasut merkeille. Tämä on ongelma myös Unicodessa muutamien aasialaisten kielten kohdalla. Itä-Aasian kielistä kanji (japani), hanzi (kiina) ja hanja (korea) sisältävät samankaltaisia merkkejä. Unicode Consortium ja ISO/IEC 10646 yhdisti nämä samankaltaiset merkit säästääkseen tilaa muille merkeille, koska oletti näiden merkkien olevan samoja. Adamsin (1993, 36-38) mukaan näiden kielten toteuttaminen erikseen Unicodeen olisi vaatinut noin 40000 - 60000 merkin verran lisää tilaa. Merkkien erot tulevat esiin eri kielten perusfontteja käytettäessä. Kun Unicodelle käytetään kiinalaista fonttia, Kubotan (2000) mukaan japanilaiset pitävät näitä merkkejä väärin kirjoitettuna, koska japanilaisella fontilla merkit olisivat hieman toisen näköisiä. Muutamat näistä samankaltaisista merkeistä ovat yleisiä esimerkiksi japanin kielessä. Ratkaisuna on tuottaa joka kielelle oma fontti, jossa näiden merkkien toteutus on paikallisen kirjoitusasun mukainen. Kubotan mielestä tämä ratkaisu toimii vain esimerkiksi kaupallisissa käyttöjärjestelmissä, joita voidaan toimittaa eri versioina eri alueille. Esimerkiksi Microsoft on toiminut näin Windowsin kanssa. Muissa järjestelmissä ongelma onkin vaikeampi ratkaista. Tästä syystä Kubota pitää Unicodea huonosti toteutettuna. Adams (1993) tunnustaa ongelman, mutta puolustaa Unicoden toteutusta. Hänen mielestään asian ratkaisu kuuluu fonttien tuottajille eikä Unicode Consortiumille tai ISO/IEC 10646:lle. Adams toteaa, että merkkien ulkoasujen pohtiminen ei kuulu näille standardointiorganisaatioille, koska niiden tehtävä on vain määrittää

yksiselitteinen koodi kullekin merkille. Adamsin mielipidettä selittää osaksi se, että kun päätös yhdistämisestä tehtiin, UCS:stä oli olemassa vasta UCS-2 –versio. 16-bitin ajateltiin riittävän kaikkien kielten kuvaamiseen ja siksi tilaa haluttiin säästää. Jos vastaava päätös jouduttaisiin tekemään nyt, kun merkistö on jo laajennettu 32-bittiseksi, niin ehkä samankaltaisia merkkejä ei enää yhdistettäisi.

4.2 Merkistöstä puuttuvat merkit

Edes Unicode ei sisällä kaikkia maailman kirjoitettuja merkkejä, vaan sitäkin täydennetään jatkuvasti. Kuten edellisessä aliluvussa tuli ilmi, aina ei osata aluksi varautua millaisia merkkejä tulevaisuudessa tullaan tarvitsemaan. Hyvänä esimerkkinä tästä on euro-merkki (€), jota ei ollut olemassakaan, kun esimerkiksi ISO-8859-1 –merkistö luotiin. Kyseinen merkki sisällytettiin ISO-8859-15 –merkistöön, jota oltiin luomassa euron tulon aikoihin. Aaltonen toteaakin, että euro-merkki aiheutti paljon muutospaineita WWW-sivustoille, koska sitä ei ole muissa ISO-8859 –perheen merkistöissä kuin ISO-8859-15 -merkistössä. Ennen kuin ISO-8859-15 voidaan ottaa sivustolla käyttöön esimerkiksi ISO-8859-1 –merkistön tilalla, Aaltosen mukaan sivut pitää konvertoida. Sekään ei välttämättä takaa merkin näkyvyyttä kaikissa tilanteissa, sillä vain osa selaimista tukee ISO-8859-15 –merkistöä, koska se on niin uusi merkistö.

Aaltosen mukaan tämän ongelman voi kiertää niin, että euro merkitään tekstin sekaan HTML-standardin mukaisena **entitynä** eli &eur; -muodossa. Entity-merkit ovat selaimen ymmärtämiä koodeja, jotka selain parsii tekstin joukosta ja näyttää ne haluttuina merkkeinä käyttäjälle WWW-sivulla. Entityjen avulla voidaan WWW-sivulla näyttää muutamia vakiintuneita merkkejä, jotka eivät yleensä kuulu merkistöihin. Näitä ovat euron lisäksi muun muassa erilaisia kaupallisia oikeuksia määrittelevät merkit kuten ©, ® ja ™. Entity-merkinnän toimiminen vaatii kuitenkin, että merkki on määritelty HTML-standardiin.

Euro-merkki on vasta uusimmassa HTML-standardissa, joten Aaltosen mukaan vanhat Netscapen selaimet eivät osaa näyttää euroa entity-merkinnälläkään oikein.

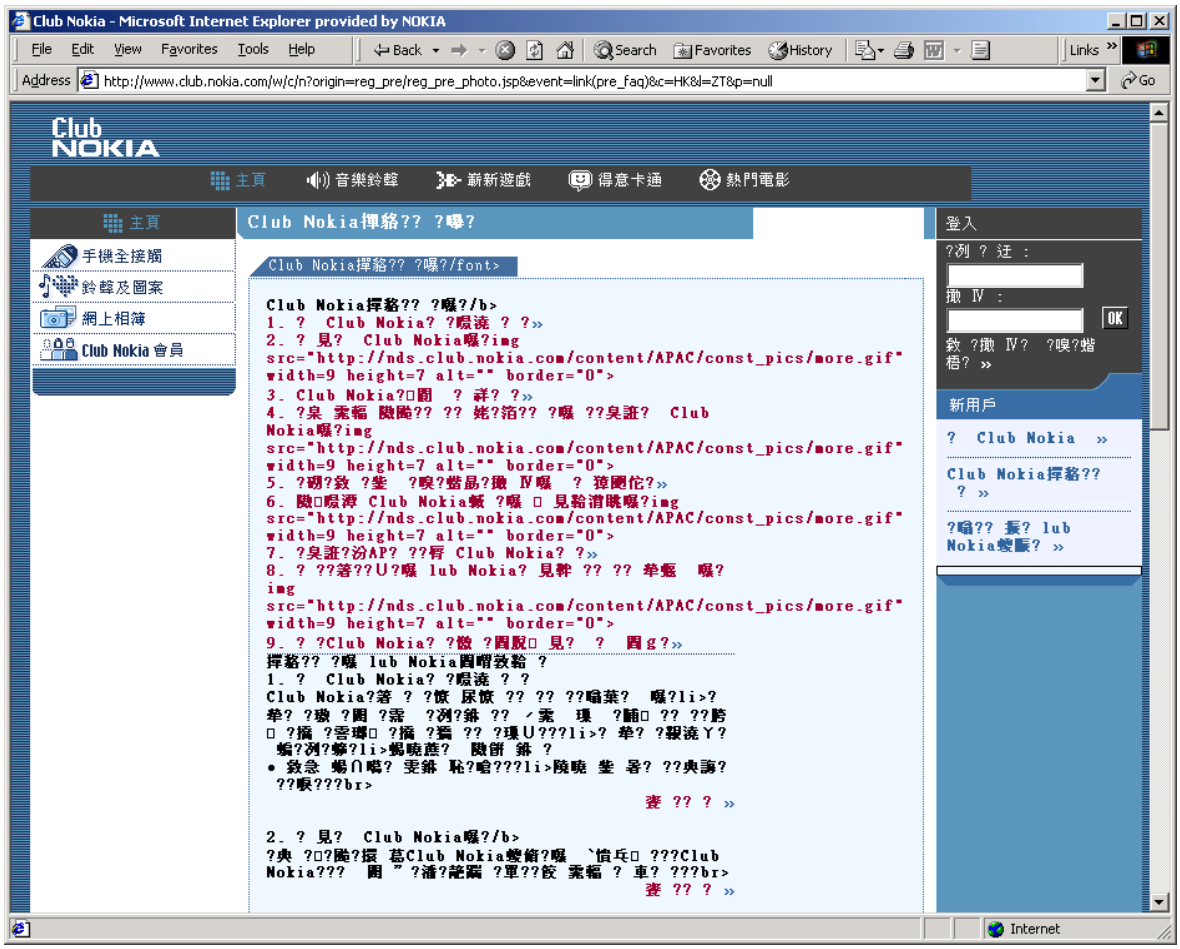
Kun järjestelmässä käytetään Unicodea ja halutaan sisällyttää tekstiin merkkejä, joita Unicode-standardi ei vielä kata, voidaan näille merkeille käyttää arvoja yksityiskäyttöön varatusta koodiavaruudesta. HTML 4.01 –spesifikaatio (1999) suositteleekin tätä tapaa sen sijaan, että merkeille määriteltäisiin arvot vielä määrittelemättömältä alueelta. Koska Unicodeen lisätään merkkejä, niin tällä tavoin ei tule konflikteja Unicoden uudempien versioiden kanssa. Tämä tapa tietysti vaatii, että järjestelmässä on käytössä fontti, jossa on oikea toteutus näille yksityiskäyttöön varatusta koodiavaruudesta käytetylle merkeille. WWW-sivustojen kanssa näin ei voida olettaa olevan, koska selaimet hakevat Unicode-merkeille fontit suoraan käyttäjän oman koneen käyttöjärjestelmästä. Tällöin siis pitäisi vaatia, että käyttäjä lataa uuden fontin, jossa on toteutus halutulle merkille.

4.3 Selaimen asetukset

Kaikki näkyvyysongelmat eivät johdu puutteellisesta tai kokonaan puuttuvasta fontista. WWW-selaimen asetuksilla on suuri merkitys merkkien näkyvyyteen. Jos selaimessa on valittuna eri merkistö tai enkoodaus kuin mitä WWW-dokumentissa on käytetty, pahimmassa tapauksessa teksti muuttuu täysin lukukelvottomaksi merkkisotkuksi (ks. KUVA 3) ja (ks. KUVA 4).

KUVA 3. GB2312-merkistöllä kirjoitettua sivua tulkitaan ISO-8859-1 merkistöllä

Kuvan 3 tilanne syntyy Shabalinin (2002) mukaan, kun kahta oktettia merkkiä kohden käyttävän GB2312-merkistön tekstiä yritetään lukea ISO-8859-1 merkistönä. Tällöin selain tulkitsee jokaisen GB2312 merkin olevan kaksi ISO-8859-1 merkkiä. Selain näyttää ISO-8859-1 merkistön mukaiset kirjaimet koodiarvoille ja tästä syystä esimerkiksi kiinalaiset merkit hajoavat lukukelvottomiksi.



KUVA 4. Selaimesta valittu merkistö Big5, kun www-sivun oikea merkistö on UTF-8.

Shabalinin mukaan merkit korvautuvat kysymysmerkeillä kuten kuvassa 4, jos Unicode-tekstiä tulkitaan jonain muuna kuin Unicode-merkistönä. Kuvassa 4 kiinalaisina merkkeinä näkyvät merkit eivät ole samoja kuin alkuperäisen tekstin kiinalaiset merkit (vertaa KUVA 5). Myös suurin osa sivun ylälaidassa olevista linkeistä hajosi, kun sivua tulkittiin kaksioktettisen Big5-merkistön avulla, kun Unicodessa näille merkeille käytetään kolme oktettia. Melkein kaikki ”Club Nokia” -tekstit säilyivät kunnossa, koska niiden merkit kuuluvat ASCII-merkistöön, joka on sekä Unicoden että Big5:n osa. Merkistön vaihtaminen UTF-8:ksi korjaa tämänkin sivuston luettavaksi, kuten kuvasta 5 voi havaita.

KUVA 5. Oikea merkistö (UTF-8) valittuna sivun katseluun

Ongelmia voi tuottaa myös se, että Netscapen kanssa käyttäjä voi valita haluamansa fontin merkistölle. Windows ja Mac -ympäristöissä ongelmana on se, että käyttäjä voi valita yhdistelmän, jossa valitulla fontilla ei ole ulkoasuja valitun merkistön merkeille. Tällöin fontittomat merkit näkyvät neliöinä. UNIX-käyttäjälle Netscape tarjoaa vain niitä fontteja merkistölle, jotka todella kattavat merkistön merkit. (Netscape 2000)

4.4 WWW-dokumentin merkistö- ja enkoodausmääritykset

Vaikka esimerkiksi Netscape antaa ohjeita miten oikea merkistö valitaan selaimesta, käyttäjän ei kuitenkaan pitäisi joutua valitsemaan tai vaihtamaan itse merkistöä tai enkoodausta. HTML 4.01 -spesifikaation (1999) mukaan selaimen tulisi käyttää WWW-dokumentissa ilmaistua merkistöä ja enkoodausta oletuksena ja ohittaa käyttäjän itse selaimesta tekemät valinnat. Tällöin voidaan olla varmempia siitä, että WWW-dokumentti näkyy käyttäjälle

niin kuin pitääkin. Ongelmia ilmenee kuitenkin, jos WWW-dokumentin merkistö- ja enkoodausmääritykset on ilmaistu väärin tai puutteellisesti. Tällöin tuloksena olla voi samankaltaisia virhetilanteita kuin kuvissa 3 ja 4 on kuvattu eli tekstistä tulee lukukelvotonta. Erona selaimen asetuksista riippuviin ongelmiin kuitenkin näissä on se, että dokumentti näkyy väärin kaikissa selaimissa eikä käyttäjä välttämättä voi saada oikeaa tekstiä näkyviin selaimen merkistö- tai enkoodausasetuksia vaihtamalla. Netscape (2000) suositteleekin, että sivuston tekijät ilmaisisivat WWW-dokumentissaan merkistön ja enkoodauksen yksikäsitteisesti, jotta selain voisi tunnistaa sen automaattisesti oikein.

Tieto WWW-dokumentin merkistöstä ja enkoodauksesta välitetään erilaisten parametrien avulla WWW-dokumentissa. **Charset**-parametrin käyttö on HTML 4.01 –spesifikaation (1999) mukaan ehkä helpoin tapa ilmaista ja tunnistaa dokumentissa käytetty merkistö ja enkoodaus. Lisäksi tiedonsiirrossa voidaan käyttää merkistön ja enkoodauksen ilmaisemiseen parametreja **Content-Transfer-Encoding, Accept-Encoding, Content-Encoding** tai **Transfer-Encoding**. Käytettävät parametrit riippuvat valitusta tiedonsiirtostandardista. Protokollat ja tiedonsiirtostandardit suosittelevat käyttämään näissä määrityksissä IANA:n standardoimia merkistöjen nimiä. (Fielding ym. 1999; Freed & Borenstein 1996) Esittelen seuraavaksi MIME (1996) ja HTTP/1.1 (1999) –protokollien mukaan edellä mainitut parametrit. Mukana on myös esimerkkejä niiden käytöstä HTML 4.01-spesifikaation toteuttavissa dokumenteissa.

Sekä MIME (*Multipurpose Internet Mail Extensions*) että HTTP (*Hypertext Transfer Protocol*) -standardit määrittelevät **Content-Type** -kentän, johon kuuluu kiinteästi charset-parametri:

Content-Type: text/html; charset=utf-8

Charset-parametrin arvo on IANA:n standardoima merkistön (ja enkoodauksen) nimi. Tämän parametrin tehtävänä on ilmoittaa käytetty merkistö selaimelle ja muille dokumenttia käsitteleville sovelluksille. Selaimet tunnistavat käytetyn merkistön ensisijaisesti tästä määrityksestä. Merkistön/enkoodauksen nimi voidaan kirjoittaa isoilla tai pienillä kirjaimilla, joten UTF-8 on yhtä lailla hyväksyttävä arvo kuin utf-8. Jos halutaan käyttää jotain muuta kuin IANA:n määrittelemää merkistöä, MIME:n (1996) mukaan merkistön nimi pitää alkaa määrityksissä: ”x-”. HTML:ssä merkistö ja enkoodaus ilmaistaan **HTTP-HEADER** –osiossa näin:

<HTTP HEADER> Content-Type: text/html; charset=utf-8 </HTTP HEADER>

Jos palvelin ei anna lähettää HTTP-HEADER tietoa WWW-dokumentin alussa, niin charset-parametri voidaan ilmaista myös HTML:n **META**-määrityksessä:

<META http-equiv=”Content-Type” content=”text/html; charset=UTF-8”>

Jos charset-parametriä ei ole annettu HTTP-HEADER- eikä META-osiossa, se voidaan liittää moniin muihin HTML-elementteihin. HTML 4.01 suosittelee kuitenkin, että ensisijaisesti tämä parametri ilmoitettaisiin aina HTTP-HEADER –osiossa. Jos se ei ole mahdollista, niin pitäisi käyttää META-määritystä. Muihin elementteihin liittäminen ei ole suositeltavaa, vaikkakin se on mahdollista. (Fielding ym. 1999; Freed & Borenstein 1996)

HTTP/1.1 –protokollassa määritellään, että jos merkistöä ei ole ilmaistu edellä mainituilla tavoilla, niin se on (tai ainakin selaimet voivat olettaa sen olevan) ISO-8859-1. MIME:n ja SMTP:n (2001) mukaan oletusmerkistö on US-ASCII. HTML 4.01 –spesifikaatiossa (1999) kritisoidaan näitä olettamuksia, koska sen mukaan kaikki palvelimet eivät anna lähettää merkistömäärittelyä WWW-dokumentin mukana. Siksi selain ei saisi olettaa mitään, jos määrittely puuttuu. Toisaalta merkistö saattaa olla myös väärin määritelty. Tämän vuoksi HTML 4.01 -spesifikaatio hyväksyy sen, että selain tarjoaa käyttäjälle mahdollisuuden vaihtaa merkistön. Jotta väärin määriteltyjä sivustoja ei syntyisi vahingossa,

spesifikaation mukaan tätä merkistön vaihtoa ei pitäisi sallia WWW-dokumentteja muokattaessa selaimen mahdollisilla muokkausohjelmilla. (Netscape 2000; Raggett, Le Hors & Jacobs 1999; Fielding ym. 1999; Freed &Borenstein 1996; Klensin 2001)

Jos ContentType ja Charset -parametrien arvoja ei tunnisteta MIME-protokollaa käytettäessä, niin MIME:n mukaan dokumenttia pitää käsitellä kuin ContentType olisi ”application/octet-stream”. Siinä muodossa tekstiä ei yritetä tulkita mitenkään, vaan bitit välitetään sellaisenaan. Tällöin käyttäjän pitää tietää millä sovelluksella tekstiä voi lukea, koska selain ei välttämättä näytä sitä automaattisesti. (Freed & Borenstein 1996) Kun merkistömäärittely puuttuu, Aaltonen toteaa haastattelussa useimpien selainten toimivan kuitenkin käytännössä niin, että ne käyttävät viimeksi vieraillun WWW-sivun merkistömääritystä uudenkin sivun oletusmerkistönä. Jos siis käyttäjä vierailee ensin kiinankielisellä WWW-sivulla, jonka merkistömäärittely on <META http-equiv="Content-Type" content="text/html; charset=gb2312">, kiinankielinen teksti näkyy sivulla aivan oikein, sillä merkistömäärittely on kunnossa. Jos käyttäjä tämän jälkeen menee suomenkieliselle WWW-sivulle, jossa merkistömääritystä ei ole asetettu, niin ”ääkköset” (eli å, ä ja ö –kirjaimet) ja sanoissa niitä seuraavat kirjaimet muuttuvat kiinalaisiksi merkeiksi. Tekstin lukemisesta tulee siis ainakin vaikeaa, ellei mahdotonta. Aaltosen mukaan tämä johtuu siitä, että ”ääkköset” eivät kuulu ASCII-merkistöön ja ne tulkitaan GB2312-merkistön vuoksi olevan kaksioktettisia merkkejä. Selain siis yhdistää ”ääkkösen” ja sitä seuraavan merkin yhdeksi merkiksi. Tilanne korjaantuu, kun käyttäjä sulkee selaimen ja avatessaan sen uudelleen menee ensimmäiseksi sille suomenkieliselle sivulle, josta merkistömäärittely puuttui. Kun selain suljetaan, viimeksi selatun WWW-sivun merkistövalinnat lakkaavat vaikuttamasta. Tällöin sivu, jolta merkistömäärittely puuttuu, näytetään selaimen oletusmerkistöllä eli ISO-8859-1 –merkistöllä, joka on kyseisessä esimerkissä myös suomenkielisen sivun tekstille oikea merkistö.

Jos WWW-järjestelmään valittu merkistö sisältää muita kuin 7-bittisiä merkkejä, on MIME-protokollan mukaan ilmaistava myös tiedonsiirrossa käytettävä enkoodaus. Tämä tehdään **Content-Transfer-Encoding** –parametrin avulla seuraavasti:

Content-Transfer-Encoding: 8bit

MIME-protokollan mukaan Content-Transfer-Encoding –parametrin arvoksi voi antaa vain ”7bit”, ”8bit” tai ”binary”. Oletusarvo on ”7bit”. HTTP/1.1 –standardissa vastaava tiedonsiirron enkoodausparametri on **Transfer-Encoding**:

Transfer-Encoding: chunked

Jos dokumentin vastaanottava palvelin ei tunnista annettua tiedonsiirtomääritystä tai parametrin arvoa, HTTP/1.1-protokollan mukaan palvelimen pitäisi antaa käyttäjälle HTTP 501 –virheilmoitus ja sulkea yhteys. Jos taas käytössä on HTTP/1.0-protokolla, niin lähettävä palvelin ei saisi sisällyttää tiedonsiirtomääritystä mukaan ollenkaan. (Fielding ym. 1999; Freed & Borenstein 1996; Klensin 2001) Koska HTTP-protokollat eroavat näin radikaalisti, eli että toisessa parametri pitää välittää ja toisessa sitä ei saa välittää, niin WWW-järjestelmää määritettäessä pitäisi aina myös määritellä tiedonsiirrossa välitettävä protokolla. Näin WWW-sivuston toteuttajat tietävät pitääkö määritys lisätä mukaan WWW-dokumentteihin vai ei.

HTTP/1.1-protokolla määrittelee myös **Accept-Encoding** tai **Content-Encoding** –parametrit. Näiden parametrien tarkoituksena on mahdollistaa tekstin tiivistäminen enkoodausta vaihtamalla ilman, että informaatiota katoaa tiivistettäessä. Tiivistystä käytetään esimerkiksi silloin, kun siirretään suuria määriä dataa hitaan verkkoyhteyden yli. Parametrin käyttöperiaate on sama kuin Transfer-Encoding –parametrin kanssa:

Content-Encoding: gzip

Accept-Encoding: compress, gzip

Accept-Encoding: compress;q=0.5, gzip;q=1.0

Content-Encoding -parametri eroaa Transfer-Encoding –parametrista siinä, että Transfer-Encoding –parametri vaikuttaa aina koko dokumenttiin, kun taas Content-Encoding voidaan määritellä myös vain tietylle dokumentin osalle. Jos jompikumpi on määritelty, se yleensä tarkoittaa, että dokumentti ei ole kokonaan luettavissa ennen kuin dokumentti on purettu perusmuotoon. Dokumentille voidaan määritellä monta peräkkäistä tiivistystä. Tällöin parametrin arvot pitää olla lueteltuna samassa järjestyksessä kuin missä niitä on käytetty dokumentille. Muutoin purkaminen tehdään väärässä järjestyksessä ja tekstistä tulee lukukelvotonta. Siis kuvien 3 ja 4 tilanteet voivat myös johtua siitä, että tiivistysparametrien arvot on lueteltu väärässä järjestyksessä.

Jos käyttäjä voi lähettää palvelimelle takaisin tekstiä WWW-sivulta, WWW-sivun Accept-Encoding –parametrilla voidaan määritellä, mitä tiivistysmuotoja käyttäjän kirjoittamaan tekstiin saa käyttää lähetettäessä sitä takaisin. HTTP/1.1-protokollan mukaan, jos Accept-Encoding on määritelty, mutta palvelin ei voi lähettää dokumenttia määrityksen mukaan, palvelimen pitäisi antaa virhe 406 (Not Acceptable) eikä välittää dokumenttia. Jos taas vastaanottava palvelin ei hyväksy määritystä, palvelimen pitäisi antaa virhe 415 (Unsupported Media Type). (Fielding ym. 1999) Ongelmana voi kuitenkin olla, että palvelin ei lähetä virheilmoitusta ja käsittelee dokumenttia väärin. Tällöin turvallisinta olisikin olla käyttämättä mitään tiivistyskeinoja.

4.5 Tiedonsiirto

Tiedonsiirtoprotokollat kuten HTTP (*Hypertext Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*) ja MIME (*Multipurpose Internet Mail Extensions*) määrittelevät miten eri merkistöjä sisältävät tekstit tulisi välittää tiedonsiirtoväylien kautta selaimille. Tieto käytetystä merkistöstä ja

enkoodauksesta välitetään aliluvussa 4.4 määriteltyjen parametrien avulla. Vaikka parametrit olisivat ilmaistu WWW-dokumentissa oikein, WWW-järjestelmällä voi olla vaikeuksia välittää dokumenttia halutussa muodossa. Tällöin tekstin lukukelvottomuus voi myös johtua siitä, että tekstiä ei ole alunperinkään tuotu WWW-käyttöliittymään oikeassa muodossa. Joskus tiedonsiirrossa tapahtuvia merkistöjen käsittelyvirheitä on vaikea erottaa aiemmin esitellyistä vääristä selainasetuksista ja WWW-dokumentin merkistömäärittelyistä johtuvista ongelmista. (Fielding ym. 1999; Freed & Borenstein 1996)

Goldsmithin ja Davisin (1997) mukaan monet MIME-tiedonsiirtoväylät eivät pysty käsittelemään edes 8-bittistä ASCII-merkistöä. MIME-protokolla (1996) ei vahvista tätä, mutta sen mukaan SMTP pystyy käyttämään vain 7-bittistä tiedonvälitystä. Kun 8-bittistä tietoa välitetään 7-bittisen tiedonsiirtoväylän (*gateway*) kautta konvertoimatta sitä 7-bittiseen muotoon, 8-bittinen tieto hajoaa lukukelvottomaksi. Jos ei ole täyttä varmuutta siitä, että 8-bittinen tiedonvälitys onnistuu, niin on turvallisinta muuttaa kaikki tekstit siirron ajaksi 7-bittisiksi, koska kaikki tiedonsiirtoväylät pystyvät käsittelemään niitä. WWW-dokumenttien kanssa tämä on erityisen tärkeää, sillä dokumentin luoja ei pysty yleensä tietämään minkä tiedonsiirtoväylän kautta dokumentti välitetään vastaanottajalle.

Tekstin konvertoinnissa enkoodauksesta toiseen on riski, että konvertointi ei onnistukaan suunnitellusti. Riskin minimoimiseksi alkuperäinen merkistö olisi hyvä olla merkittynä WWW-dokumenttiin esimerkiksi HTTP-HEADER -osioon. Kun konvertointi aloitetaan, uusi enkoodaus pitää muistaa ilmaista aliluvussa 4.4 esiteltyjen parametrien avulla, jotta vastaanottaja osaa tarvittaessa muuttaa tekstin takaisin toiseen enkoodaukseen. (Freed & Borenstein 1996) Jos enkoodaus on ilmaistu väärin, niin erittäin luultavasti konvertointi epäonnistuu eikä vastaanottaja enää saa selvää tekstistä. Pahimmassa tapauksessa palvelin ei lähetä tekstiä ollenkaan vastaanottajalle.

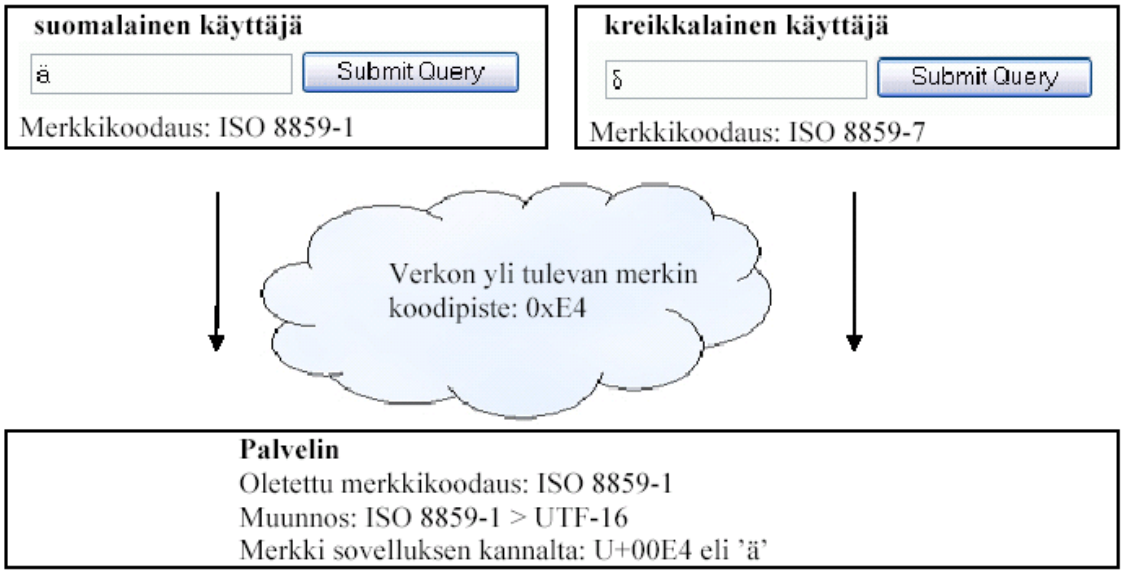
Unicode 3.0 -standardissa suositellaan, että Unicode-merkit siirrettäisiin MIME-protokollan toteuttavassa tiedonsiirtoväylässä aina UTF-7-muodossa (The Unicode Consortium 2000). UTF-7:n kuten muidenkin 7-bittisten enkoodausten etu on se, että jos tiedonsiirtoväylä on 8-bittinen, oktetin kahdeksatta bittiä voidaan käyttää tiedonsiirron ja konversion onnistumisen tarkistamiseen. Tätä bittiä kutsutaan Järvisen (1996) mukaan **pariteettibitiksi** (*parity bit*). **Pariteetti** (*parity*) voi olla pariton (*odd*), jolloin oktetin 1-bittien määrä on pariteettibitti mukaan luettuna pariton. Vastaavasti pariteetti on parillinen (*even*), kun määrä on parillinen. Jos tiedonsiirrossa tapahtuu virhe ja oktetissa menee yksikin bitti väärin, niin tämä virhe havaitaan pariteettitarkistuksen avulla. Kuitenkaan pariteettitarkastuksessa ei voida havaita kaikkia tapahtuneita virheitä. Nimittäin jos tavussa menee parillinen määrä bittejä väärin, niin virheet kumoavat toisensa ja pariteettitarkastuksessa kaikki näyttää olevan kunnossa. (Järvinen 1996) Pariteettibitti voidaan tietysti lisätä muunkin mittaisiin enkoodauksiin ja pariteettitarkistusta voidaan käyttää hyväksi muuallakin kuin tiedonsiirrossa. Adams (1993, 32) huomauttaa, että ASCII:n kanssa kahdeksatta bittiä ei yleensä kuitenkaan huomioida ohjelmassa ollenkaan, se poistetaan kokonaan tai sitä käytetään ohjelman sisäiseen prosessointiin. Prosessointiin käytettäessä pariteetti voi tarkoittaa esimerkiksi jotakin virhetilaa järjestelmässä. Turnbull (1999) antaa esimerkkejä sisäisestä prosessoinnista. Hänen mukaansa jotkin ajurit käyttävät pariteettibittiä ulkoasumuutosten tekoon, kuten alleviivaukseen tai videon takaperin näyttämiseen. Jos pariteettitarkistus halutaan tehdä, on hyvä varmistua etukäteen siitä, että pariteettibitin käytöllä ei ole yllättäviä seurauksia järjestelmässä eli pariteettia ei käytetä järjestelmässä esimerkiksi tekstin muotoiluun.

4.6 Tekstin syöttö

WWW-järjestelmissä on usein osioita, joissa käyttäjä voi syöttää tekstitietoa järjestelmään. Yksi yleisimmistä tällaisista WWW-sivuston sovelluksista on rekisteröitymislomake. Käyttäjän odotetaan usein rekisteröityvän järjestelmän

käyttäjäksi, jolloin käyttäjästä tallennetaan vähintäänkin nimi. WWW-järjestelmissä on Aaltosen kokemuksen mukaan paljon ongelmia käyttäjätietojen käsittelyssä. Haastattelussa Aaltonen toteaakin, että varsinkin ulkomaalaisiin järjestelmiin kirjautuessa käyttäjän kannattaa nimi- ja osoitetiedot ilmoittaa vain ASCII-merkkejä sisältävinä teksteinä. Yleensä jo suomalaiset ä ja ö-kirjaimet tallennetaan väärin, puhumattakaan kiinankielisistä merkeistä. WWW-sovellukset tulisi kuitenkin luoda niin, että varsinkin rekisteröitymissivulla voisi syöttää tekstin Unicodena.

Tekstinsyöttö voi helposti muodostua ongelmaksi myös WWW-palveluissa, joissa käyttäjä voi esimerkiksi hakea palvelusta jotain vapaalla hakusanalla kielestä riippumatta. Tällöin pitäisi varmistua siitä, että hakutekstin merkistö ja enkoodaus ovat samat kuin teksteissä, joihin niitä haettaessa verrataan. Jos ne eroavat, pahimmassa tapauksessa haku ei palauta mitään tai palauttaa jotain aivan väärää. Kuvassa 6 esitetään kuinka suomalainen ja kreikkalainen voivat tehdä samalta WWW-sivulta haun eri merkeillä, mutta UTF-8:aa käyttävässä WWW-sovelluksessa molemmat merkit tulkitaan olevan ’ä’, koska käyttäjän oletetaan käyttävän palvelua ISO-8859-1 –merkistöllä. Tällöin WWW-sovellus palauttaa kreikkalaiselle käyttäjälle väärät hakutulokset.



KUVA 6. Syötteen tulkitseminen palvelimessa (Mäkinen 2002, 16-17)

Tällaisia ongelmia on tyypillisesti muun muassa hakukoneissa ja kirjastosovelluksissa. Aaltonen toteaa, että hakukoneista Lycos käyttää windows-1252 –merkistöä ja HotBot ISO-8859-1 –merkistöä, joten niillä voi olla ongelmia esimerkiksi japanilaisten tekstien kanssa. Hakukoneista UTF-8 –pohjaisia ovat Altavista, Yahoo ja Google.

Ongelmia on myös WWW-palveluissa, jotka suoraan julkaisevat käyttäjän syöttämän tekstin sivustolle (mm. chat-palvelut tai sisällöntuotantotyökalut) tai lähettävät syötetyn tekstin sähköpostina esimerkiksi palvelun ylläpitohenkilölle (mm. palautesivustot). Yleensä ASCII–merkkejä sisältävät tekstit näkyvät kuitenkin oikein, jos ASCII sisältyy sekä käyttäjän että järjestelmän käyttämiin merkistöihin. Muissa tapauksissa teksti muuttuu lukukelvottomaksi merkkisotkuksi (ks. KUVA 3 ja KUVA 4). Sivun merkistön tulisi siis aina olla sama kuin käyttäjän syöttämässä tekstissä, mikä on mahdotonta toteuttaa. WWW-sivuilla voidaan yleensä kerrallaan näyttää vain yhteen merkistöön kuuluvaa tekstiä. Poikkeuksen muodostaa Aaltosen mukaan HTML:n tarjoama FRAME-osio. FRAME:lla saadaan näytettyä yhdellä WWW-sivulla vierekkäisiä

sivuja, joille jokaiselle voidaan määrittää oma merkistö ja enkoodaus. Kaikki selaimet eivät kuitenkaan tue FRAME-osioita.

Yleisesti ongelmana tekstinsyötössä on se, että käyttäjä voi halutessaan vaihtaa selaimestaan merkistön WWW-sivun määrityksestä poikkeavaksi. Tällöin tekstikenttään syötetty teksti onkin käyttäjän valitseman merkistön mukaista eikä järjestelmän käyttämän merkistön mukaista. Jos käyttäjä vaihtaa selaimen merkistöasetuksia, niin WWW-sivun muukin teksti näkyy useimmiten väärin, joten käyttäjän pitäisi tämä huomata virhe. Näitä selaimen asetuksista johtuvia näkyvyysongelmia käsiteltiin luvussa 4.3 Selaimen asetukset. Jos kuitenkin WWW-sivun teksti on ASCII-tekstiä ja syöteteksti esimerkiksi Unicodea, niin selaimen asetusten vaihto ei välttämättä vaikuta sivulla näytettävään tekstiin, vaan ainoastaan käyttäjän syötteeseen. Mäkisen (2002) mukaan käyttäjä voi myös kopioida syötetekstin jostain muualta, jolloin sen merkistö ja enkoodaus voivat olla jotain muuta kuin mitä WWW-järjestelmässä luullaan olevan. WWW-sovelluksen pitäisi siis pystyä tunnistamaan käyttäjän lähettämän tekstin merkistö ja enkoodaus. HTML-lomakkeessa (FORM-tagissa) voidaan määritellä accept-charset, jolla selaimelle voidaan ilmoittaa millä merkistöllä lomakkeen tiedot WWW-järjestelmä haluaa ottaa vastaan. Mäkisen mukaan vielä vuonna 2002 ei yksikään selain tukenut toimintoa, jossa selain olisi muuttanut itse tekstin merkistön accept-charset määrityksen mukaiseksi. Käytännössä selaimet siis lähettävät tekstin käyttäjän selaimen merkistövalinnan mukaisesti välittämättä siitä, mitä WWW-järjestelmä haluaisi ottaa vastaan. Tähän ongelmaan Mäkinen ehdottaa ratkaisuksi sitä, että HTML-lomakkeessa olevaan, käyttäjältä piilotettuun kenttään tallennetaan tietty, erikoisia merkkejä sisältävä merkkijono. Kun käyttäjä lähettää tekstin, WWW-sovellus voi verrata tätä erikoismerkkijonoa alkuperäiseen ja varmistaa onko se muuttunut. Jos erikoismerkkijono on muuttunut, on se osoitus siitä, että käyttäjä on valinnut selaimestaan jonkin muun merkistön kuin mitä WWW-järjestelmä käyttää ja käyttäjän syöttämä teksti ei ole myöskään oikeassa

muodossa. Jos erikoismerkkijono on pysynyt samana, käyttäjän kirjoittama teksti on välitetty sovellukselle asti siinä muodossa missä käyttäjä sen kirjoitti. (Mäkinen 2002, 16-18)

Tämä ratkaisu ei käsittääkseni toimi ollenkaan kuvan 6 tapauksessa. Ongelmana on se, että kaikissa ISO-8859 –merkistöissä fontilla määritellään millaisina merkit näkyvät. Kreikkalainen kirjain 'δ' ja suomalainen 'ä' ovat samalla koodiarvolla. Jos käyttäjä siis syöttää tekstiä ISO-8859-7 merkistöllä ISO-8859-1:n sijaan, tarkistusmerkkijono pysyy samana, koska tarkastuksessa fontteja ei voi käyttää mitenkään hyväksi. Lisäksi tämä ratkaisu auttaa vain tunnistamaan tapaukset, joissa merkistö on muuttunut, mutta WWW-sovellusta suunniteltaessa pitää silti päättää, mitä mahdollisesti muuttuneille syötteille tehdään. Syötteet eivät siis korjaannu itsestään, vaikka tarkistusmerkkijonoa käytettäisiin. Syötetty teksti voidaan tietysti hylätä ja käyttäjälle voidaan ilmoittaa tästä ja esimerkiksi pyytää käyttäjää syöttämään teksti uudelleen toisella tavalla. WWW-sovellus voi myös sisältää muunnoskomponentin muuttuneiden syötteiden varalle, jolloin syötetekstiä voidaan yrittää muuttaa enkoodauksesta ja merkistöstä toiseen merkkijonoa hyväksikäyttäen. Parhaimmillaan teksti saadaan muutettua järjestelmän haluamaan muotoon. Tarkistusmerkkijonon käyttö on kuitenkin joka tapauksessa parempi keino virhetilanteiden tunnistamiseen kuin että käytössä ei olisi mitään keinoa.

4.7 Ohjelmointikielet

Kuvan 4 ongelmat voivat johtua myös siitä, että itse WWW-sovellus käsittelee merkistöä väärin. Ohjelmointikielten merkistötuki eri merkistöille vaihtelee. WWW-sovelluksen ohjelmointiin käytettävän kielen olisi hyvä suoraan tukea valittua merkistöä, jotta tekstin sisäinen käsittely olisi mahdollisimman yksinkertaista.

Jos valittu merkistö on Unicode, niin esimerkiksi Java on hyvä valinta ohjelmointikieleksi, koska Javassa kaikkia merkkijonoja käsitellään aina

Unicode-muodossa. Haible (2001) kuitenkin huomauttaa, että Javan rajapintaluokat java.io.DataInput ja java.io.DataOutput eivät käytä puhdasta UTF-8:a, vaan enkoodaavat NUL-merkin kaksitavuisena merkkinä (0xC0 0x80) normaalin yksitavuisen (0x00) sijaan ja loppuun lisätään vielä tavallinen NUL-tavu (0x00). Tämä ei kuitenkaan tuo Haiblen mukaan ongelmia merkkijonojen käsittelyyn edes C++ -metodien kuten strlen tai strcpy kanssa.

Greenin (1998) mukaan taas Javan 1.0 alusta tukee vain ISO-8859-1 –merkistöä. Hänen mukaansa Unicodea tuetaan kyllä versiosta 1.1 ylöspäin. Java Appletit kirjoitetaan Javalla, jolla siis on Unicode-tuki. JSP:ssä (*Java Server Pages*) yhdistetään HTML ja Java-koodi. Koska HTML 4.01 ja Java tukevat molemmat Unicodea, tukee JSP:kin sitä.

Java saattaa kuitenkin tuottaa yllätyksen paikallisten merkistöjen suosijalle. Sun Microsystems (1999) on jakanut merkistöt kahteen ryhmään. Perusryhmään kuuluvat ASCII, ISO-8859-1 ja Unicode. Unicoden enkoodauksista tuettuina ovat UTF-8 sekä UTF-16 ja niiden little endian ja big endian -muodot. Toiseen ryhmään kuuluvat paikalliset merkistöt kuten Big5 ja ISO-8859 –merkistöperhe. Javan ilmoitetaan käsittelevän tekstiä sisäisesti Unicoden UTF-16 muodossa. Sun Microsystemsin WWW-sivuilta selviää, että Java 2 Runtime Environment (JRE) Standard Edition versiosta 1.3.1 Windowsille on olemassa kaksi versiota: Amerikan markkinoille suunnattu US-only –versio ja kansainvälinen versio. US-only –versio tukee vain merkistöjen perusryhmää, kun taas kansainvälinen tukee myös paikallisia merkistöjä. Kaikkia Sunin listaamia merkistöjä tukevat myös Sunin Java 2 Software Development Kit (SDK), Standard Edition, versio 1.3.1 kaikille käyttöjärjestelmille ja Java 2 Runtime Environment (JRE), Standard Edition, versio 1.3.1 Solarikselle ja Linuxille. (Sun Microsystems 2002) US-only –versiota pitäisi siis välttää kansainvälisten järjestelmien luonnissa, jos haluaa säästyä ikäviltä yllätyksiltä merkistötuen kanssa.

Kaikki kielet eivät kuitenkaan käytä Unicodea sisäisesti, vaan Unicode-tekstin käsittelyyn voi olla käytössä eri metodit kuin muiden 8-bittisten merkistöjen käsittelyyn. Näin on esimerkiksi C:n ja C++:n tapauksessa. C-kielen char-tyyppi on 8-bittinen ja siksi se ei sovi Unicoden käsittelyyn. Haiblen mukaan ISO/ANSI C:n 1995 versioon lisättiin wchar_t -tyyppi, joka suunniteltiin myös muun mittaisten merkkijonojen käsittelyyn ja jota siis voidaan käyttää Unicoden-kanssa. C/C++:n voidaan siis katsoa tukeneen Unicodea vuodesta 1995 lähtien. Tämä Unicode-tuki ei kuitenkaan ole automaattinen. Jos ISO-8859-1 -merkistöä käyttävä WWW-sovellus halutaan saada tukemaan Unicodea, niin C/C++:n tapauksessa kaikki koodin tekstiä käsittelevät metodikutsut joudutaan vaihtamaan toisiksi. Tämä vie aikaa ja vaatii huolellista testausta. (Haible 2001)

Perl toimii hieman samalla periaatteella kuin C/C++ siinä mielessä, että Perl 5.6 tallentaa merkkijonot UTF-8 muodossa, jos ohjelmakoodiin kirjoittaa ”use utf8;”. Muiden yli 8-bittisten merkkijonojen käsittelyyn tarvitaan konvertointikomponentti. Perlin 5.6 versiota edeltävissä versioissa ei ole tukea Unicodelle. (Haible 2001)

Konvertointi on mahdollinen ratkaisu myös muiden kielten kohdalla, jos valittu ohjelmointikieli ei tue valittua merkistöä. Tällöin teksti konvertoidaan tuettuun merkistömuotoon sovelluksen sisäisen käsittelyn ajaksi. Tämä voi kuitenkin tuoda mukanaan uusia merkistöongelmia, sillä usein konvertointikomponentti koodataan itse, mikä puolestaan aiheuttaa lisävaatimuksia järjestelmän laadun varmistamiseksi. Markkinoilla on myös valmiita konvertointikomponentteja. Haible (2001) listaa kolme mahdollista konvertointikomponenttia UTF-8:n ja useiden muiden 8-bittisten merkistöjen väliseen konvertointiin (Iconv, Librecode ja ICU). Näistä vain yhdellä eli Iconv:llä on standardoitu rajapinta. Haittapuolena on myös, etteivät komponentit toimi kaikissa järjestelmissä. Valmiiden komponenttienkin käyttämisessä on siis omat riskinsä.

WWW-sovellusten tuottamisessa yhden ongelma-alueen muodostavat ohjelmointikielet, jotka ovat kiinteästi liitoksissa selaimeen. Esimerkiksi JavaScript-kielessä on Unicode tuki versiosta 1.3 alkaen, joka julkaistiin vuonna 1999. Eri Netscapen selainversiot tukevat eri versiota JavaScriptistä (Sun-Netscape Alliance 1999). Tuki on kuvattu taulukossa 2.

TAULUKKO 2. Netscapen selainten Javascript-tuki

JavaScript	**Netscapen Navigator selain**
JavaScript 1.0	Navigator 2.0
JavaScript 1.1	Navigator 3.0
JavaScript 1.2	Navigator 4.0 - 4.05
JavaScript 1.3	Navigator 4.06 – 4.5

Netscapen eri selainversioiden erilaisen JavaScript-tuen vuoksi JavaScriptiä ei kannata suosia Unicodea käyttävissä WWW-sovelluksissa, jos tarkoituksena on tukea laajasti myös vanhoja selainversioita. Aaltonen tuo haastattelussa ilmi, että JavaScript -koodin kanssa tulee helposti myös ongelmia, jos tekstin seassa on joitain merkkejä, jotka eivät kuulu WWW-sivulla ilmoitettuun merkistöön. Tällöin JavaScript-koodi ei yleensä toimi, vaan antaa virheilmoituksen. Virheilmoituksessa kerrotaan, että sivulla on ylimääräinen ”hipsu” (').

Microsoftin luoma JavaScriptin vastine, VBScript (eli Visual Basic Script), toimii vain Microsoftin selaimissa. Se on tukenut Unicodea vuodesta 1997 alkaen. (Graham 2000, 250 -251)

4.8 Käyttöjärjestelmät

Käyttöjärjestelmätkään eivät tue kaikkia merkistöjä. Siksi on tärkeää valita käyttöjärjestelmä, joka tukee WWW-palvelimiin valittua merkistöä ja enkoodausta. Jos käyttöjärjestelmä ei tue valittua merkistöä, voi seurauksena olla samanlaisia ongelmia kuin edellisessä aliluvussa ohjelmointikielten kohdalla. Perusperiaatteena on, että yleisimmät merkistöt ovat tuettuina melkein joka käyttöjärjestelmissä. Esimerkiksi nykyiset käyttöjärjestelmät toimivat ainakin ASCII:n ja ISO-8859-1:n kanssa. Unicode on kuitenkin niin uusi merkistö, että käyttöjärjestelmissä on eroja sen tuen kohdalla. Lisäksi jos käyttöjärjestelmä tukee Unicodea, se yleensä tukee vain UTF-8 enkoodausta. Kuhnin (2003) mukaan tämä perustuu siihen, että monet käyttöjärjestelmät (esimerkiksi UNIX) on luotu toimimaan alunperin ASCII:n kanssa ja UTF-8 on Unicoden enkoodauksista se, joka on suoraan yhteensopiva ASCII.n kanssa. Näin Unicode-tuki on saavutettu suhteellisen pienellä vaivalla, kun ASCII:ta on edelleen voitu käyttää ainakin joissain käyttöjärjestelmän osissa.

Kuhnin (2003) mukaan UNIX ja Linux –käyttöjärjestelmät tukevat Unicodea ja yleisesti vain UTF-8 enkoodausta. Myös Sun Microsystemsin mukaan (2003) sen Solaris 7-käyttöjärjestelmä käsittelee Unicodea UTF-8 muodossa. Solaris 7:llä on hyvä tuki myös paikallisille merkistöille, kuten Big5:lle ja ISO-8859-merkistöperheelle. Solariksen uudemmatkin versiot tukevat Unicodesta vain UTF-8 -muotoa.

Applen Macintosh käyttöjärjestelmät ovat tukeneet Unicodea jo versiosta 8.5 alkaen. Apple on ollut Unicode Consortiumin jäsen jo alusta alkaen ja se osaltaan selittää Grahamin (2000, 236-237) mukaan Macintoshin hyvää Unicode-tukea. Toisaalta Apple on panostanut muidenkin merkistöjen tukeen ja sen käyttöjärjestelmät tukevat pääsääntöisesti hyvin kaikkia tunnetuimpia merkistöjä.

Microsoft on tullut mukaan Unicode Consortiumin jäseneksi hieman myöhemmin ja vain osa sen tuotteista on täysin Unicode-yhteensopivia. Grahamin (2000) mukaan Windows 95 ja 98 –käyttöjärjestelmät ovat parhaimmillaankin vain puoliksi Unicode-yhteensopivia, kun taas Windows NT jo suunniteltiin toimimaan Unicoden kanssa. Kaplan ja Wissink (2002) selventävät, että Windows NT, 2000 ja XP käsittelevät sisäisesti dataa Unicode muodossa, kun taas Windows 95, 98 ja Me perustuvat ANSI-pohjaisten koodisivujen (*code pages*) käyttöön. Koska Windows 95, 98 ja Me ovat vieläkin yleisiä käyttöjärjestelmiä, on Microsoft pyrkinyt parantamaan niiden Unicode-tukea. Microsoft julkaisi 2002 vuoden loppupuolella MSLU-komponentin (*The Microsoft Layer for Unicode on Windows 95/98/Me Systems*). Kun MSLU-komponentti on asennettu Windows 95, 98 tai Me –käyttöjärjestelmään, on Unicode-sovellusten luominen sille Kaplanin ja Wissinkin mukaan yhtä helppoa kuin NT:lle. Hankaluutena tietysti on, että komponentti pitää asentaa jokaiseen Windows 95, 98 tai Me -käyttöjärjestelmällä varustettuun koneeseen, jossa Unicode-sovellusta on tarkoitus ajaa.

Myös IBM tukee käyttöjärjestelmissään Unicodea ja yleensäkin lokalisointia ohjelmistokomponenttikirjaston avulla. Kirjasto on nimeltään ICU (*International Components for Unicode*). ICU perustuu avoimeen lähdekoodiin, jota kehittää pääasiallisesti ryhmä IBM:n työntekijöitä. ICU:sta on C/C++ ja Java -versiot. ICU-kirjastoa voidaan käyttää kaikkien IBM:n tarjoamien käyttöjärjestelmien (kuten AIX, OS/2 ja OS/390 –käyttöjärjestelmien) kanssa. (IBM 2003)

4.9 Yhteenveto

Tässä luvussa olen käsitellyt merkistöongelmia WWW-järjestelmässä ja joitakin ratkaisuja niihin. Merkistöongelmat näkyvät käyttäjälle niin, että WWW-sivulla oleva teksti ei näy käyttäjälle oikein. Yleisin merkistöongelma WWW-sivustoilla on, että koko teksti tai jokin osa siitä näkyy pelkkinä tyhjinä neliöinä (□), pystysuorina viivoina (|) tai matomerkkeinä eli tildeinä (~). Tällöin selain ei

löydä sivuston merkistölle fonttia, joka sisältäisi ulkoasun kyseisille merkeille. Koska selain tunnistaa kuitenkin merkit, se näyttää niiden paikalla oletusmerkin, joka on jokin kolmesta yllämainitusta merkistä. Tilanne korjaantuu, kun käyttäjän koneelle ladataan puuttuva fontti. Osa selaimista on niin kehittyneitä, että ne ilmoittavat käyttäjälle fontin puuttumisesta ja tarjoavat jotain fonttia ladattavaksi käyttäjän koneelle ennen kuin näyttävät sivun.

Hieman lievimmissä tapauksissa vain jotkin tietyt merkit tekstissä ovat korvautuneet väärillä merkeillä, mutta teksti on silti vielä pääsääntöisesti luettavissa. Syynä voi olla, että selain käyttää merkistölle fonttia, johon on toteutettu väärät, standardista poikkeavat ulkoasut joillekin merkeille. Tämä on kuitenkin aika harvinaista. Yleensä syynä on, että käyttäjä on valinnut väärän merkistön tai enkoodauksen WWW-selaimeensa ja selain tulkitsee tiettyjen merkkien koodiarvojen tarkoittavan muita merkkejä kuin mitä tekstin kirjoittaja on tarkoittanut. Yleensä merkistöt sisältävät ASCII-merkistön merkit niiden alkuperäisillä koodiarvoilla, joten englanninkieliset tekstit näkyvät oikein väärästä selaimen merkistövalinnasta huolimatta. Jos käyttäjä ymmärtää, että vika on selaimen asetuksissa, niin hän voi korjata sen vaihtamalla selaimensa merkistö- ja enkoodausasetuksia. Käyttäjien ei voi kuitenkaan olettaa olevan teknisesti niin osaavia, että he osaisivat itse ratkaista tilanteen. WWW-sivuston tarjoaja voi ehkäistä tämän tilanteen syntymistä määrittelemällä WWW-dokumentteihin merkistön ja enkoodauksen esimerkiksi HTTP-HEADER tai META -osioissa Charset-parametrin avulla. Selaimen pitäisi aina ensisijaisesti näyttää dokumentti määritellyn merkistön ja enkoodauksen mukaisesti. WWW-sivuston tarjoajan pitää olla kuitenkin huolellinen, että määritys on oikein. Jos selain ei tunnista määriteltyä merkistöä esimerkiksi määrityksessä olevan kirjoitusvirheen vuoksi, se yleensä näyttää tekstin oletusmerkistöllä, joka on yleisesti selaimissa ISO-8859-1 tai ASCII. Tällöin esimerkiksi kiinankielinen teksti näytetään kysymysmerkkeinä tai teksti hajoaa täysin lukukelvottomaksi merkkisotkuksi.

Jos käytetty merkistö ei ole 7-bittinen tai 8-bittinen, niin yleensä tiedonsiirron ajaksi teksti joudutaan muuttamaan 7- tai 8-bittiseen muotoon, koska tiedonsiirtoväylät tukevat ainakin niitä muotoja. Jos muutosta ei tehdä, teksti ei välity käyttäjälle oikein vaan näkyy luultavimmin kysymysmerkkeinä tai merkkisotkuna. Tiedonsiirron onnistumista voidaan tarkkailla 8-bittisissä tiedonsiirtoväylissä pariteetin avulla, jos siirrettävä teksti on 7-bittisessä muodossa. Yleensä WWW-dokumenttiin voidaan myös määritellä tiedonsiirrossa käytetty merkistö ja enkoodaus. Tällöin voidaan kertoa vastaanottavalle koneelle, että dokumentti pitää muuttaa takaisin toiseen muotoon ennen sen näyttämistä käyttäjälle. Tiedonsiirrossa käytettyjä parametrejä ovat Content-Transfer-Encoding, Accept-Encoding, Content-Encoding ja Transfer-Encoding. Jos näiden parametrien arvoksi laitetaan jokin muu kuin mitä tiedonsiirrossa on oikeasti käytetty, niin WWW-dokumentti puretaan vastaanottavalla koneella väärin ja teksti näkyy käyttäjälle lukukelvottomana merkkisotkuna. Jos taas WWW-dokumentin vastaanottava palvelin ei tunnista ollenkaan annettua tiedonsiirtomääritystä tai parametrin arvoa, HTTP/1.1-protokollan mukaan palvelimen pitäisi antaa käyttäjälle HTTP 501 –virheilmoitus ja sulkea yhteys. Oikea parametrin arvo korjaa yleensä tämän virhetilanteen. Protokollissa on kuitenkin eroja tiedonsiirtomääritysten suhteen. Esimerkiksi HTTP/1.0 -protokollan kanssa tiedonsiirtomäärityksiä ei pidä käyttää ollenkaan.

WWW-järjestelmissä, joissa käyttäjä voi tehdä esimerkiksi tekstihakuja tai lähettää tekstiä WWW-järjestelmän kautta, ongelmia voi olla myös käyttäjän syötteen tulkitsemisessa. Käyttäjä voi muuttaa selaimensa merkistö- ja enkoodausasetuksia, jolloin käyttäjän antama teksti lähetetään WWW-sovellukselle käyttäjän selaimen uusien merkistövalintojen mukaisena. WWW-sovellus yleensä olettaa tekstin tulevan saman merkistön mukaisena kuin mitä sivulla on ennen vaihdosta käytetty. WWW-sovellus voi yrittää tarkistaa WWW-lomakkeelle piilotettujen tarkistusmerkkien avulla onko käyttäjä

vaihtanut selaimensa merkistöasetusta ennen tekstin lähettämistä vai ei. Vaikka WWW-sovellus havaitsisikin tapahtuneen vaihdon, tekstin muuttaminen takaisin oikeaan muotoon voi olla mahdotonta.

Kun teksti näkyy käyttäjälle lukukelvottomana merkkisotkuna, syy voi olla muuallakin kuin WWW-dokumentin väärissä parametriarvoissa. Esimerkiksi WWW-sovellus saattaa käsitellä tekstiä väärin jo ennen kuin WWW-dokumentti lähetetään tiedonsiirtoväylän kautta käyttäjälle. Ohjelmointikielet, ohjelmointialustat eivätkä käyttöjärjestelmät tue kaikkia merkistöjä ja enkoodauksia. Lisäksi joissain kielissä, kuten C++:ssa, on eri metodit yli ja alle 8-bittisten merkkien käsittelyyn. Jos ohjelmointikieli, ohjelmointialusta tai käyttöjärjestelmä ei tue valittua merkistöä tai enkoodausta, on yhtenä ratkaisuna muuttaa teksti ohjelmallisesti toiseen, tuettuun muotoon tekstin käsittelyn ajaksi. Tekstin muuttaminen merkistöstä tai enkoodauksesta toiseen ohjelmallisesti (eli konvertoimalla) on kuitenkin aina riski, vaikka käytössä olisi ostettu konvertointiohjelma. Tärkeintä WWW-järjestelmän merkistökäsittelyn kannalta onkin, että järjestelmään on valittu vain osia, jotka tukevat valittua merkistöä ja enkoodausta.

5 WWW-JÄRJESTELMÄN MERKISTÖT HUOMIOIVA TUOTANTOPROSESSI

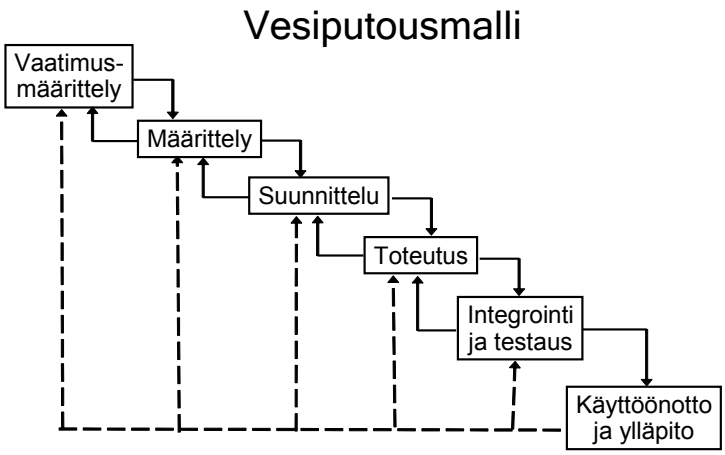
Jotta luvun 4 merkistöongelmia voitaisiin välttää jo etukäteen, ne on otettava huomioon WWW-järjestelmän tuotantoprosessissa. Tässä luvussa esittelen ensin yleisesti WWW-järjestelmän tuottamiseen soveltuvia prosessimalleja sekä niiden yhtäläisyyksiä ja eroja. WWW-keskeisistä prosessimalleista esittelen Goton ja Cotlerin (2001) WWW-järjestelmien uudelleenrakentamisen mallin sekä Noackin ja Schienmannin (1998) geneerisen elinkaarimallin. Sekä Goton ja Cotlerin mallilla että Noackin ja Schienmannin malleilla on paljon yhtymäkohtia vesiputousmalliin. Siksi aloitan esittelemällä vesiputousmallin, jota käytetään yleisesti ohjelmistokehityksessä prosessimallina. Jätän tutkielmastani pois code-and-fix –mallin sekä protoilun, vaikka ne saattavat todellisuudessa olla suosituimpia WWW-järjestelmän rakentamiseen käytettäviä malleja. Ne eivät siitä huolimatta sovellu parhaiten kansainvälisten WWW-järjestelmien rakentamiseen, koska esimerkiksi code-and-fix -menetelmässä ei suunnitella mitään etukäteen, jolloin monikielisyyden vaatimuksia on vaikea hallita. (Haikala & Märijärvi 1997, 29) Toisaalta myös WWW-järjestelmiä on tuotettu jo niin paljon, että mielestäni niitä voidaan tuottaa jo suunnitelmallisemmin kuin protoilemalla.

Prosessimallien esittelyn jälkeen käyn tarkemmin läpi monikielisen WWW-järjestelmän tekemisessä huomioitavia seikkoja vaihe vaiheelta Noackin ja Schienmannin (1998) geneerisen elinkaarimallin vaihejaon mukaisesti. Myös vaihejaosta puuttuvat kohdat testaus ja ylläpito käydään läpi, sillä ne osoittautuivat haastatteluissa olennaisiksi kohdiksi. Tämän luvun asiat perustuvat luvussa 4 esiin tulleisiin asioihin sekä Ulla-Maija Kasatkinin haastatteluun. Ulla-Maija Kasatkin on toiminut vuosia kansainvälisten monikielisten WWW-järjestelmien projektipäällikkönä ja arkkitehtinä. Hänen haastattelunsa tuo siis käytännön näkökulmia aiheeseen.

5.1 Prosessimallit

5.1.1 Vesiputousmalli

Vesiputousmalli on saavuttanut vakiintuneen aseman yhtenä ohjelmistotuotannon prosessimalleista. Siitä on olemassa paljon variaatioita ja tutkielmassani käytän Haikalan ja Märijärven (1997) esittelemää vesiputousmallia. Myöhemmin esiteltävät Goton ja Cotlerin (2001) WWW-järjestelmien uudelleenrakentamisen prosessimalli ja Noackin ja Schienmannin (1998) geneerinen elinkaarimalli pohjautuvat Vesiputousmalliin. Vesiputousmalli on kuvattu kaaviossa 4.



KAAVIO 4. Vesiputousmalli (Haikala & Märijärvi 1997, 25)

Haikalan ja Märijärven (1997) vesiputousmallissa ohjelmistotuotanto on jaettu seitsemään vaiheeseen, jotka ovat: vaatimusmäärittely, määrittely, suunnittelu, toteutus, integrointi ja testaus sekä käyttöönotto ja ylläpito. Jokaisen vaiheen päätteeksi suoritetaan hyväksymistestaus, jonka tulee mennä hyväksytysti läpi, jotta voidaan siirtyä seuraavaan vaiheeseen. Jokaisesta vaiheesta voidaan palata edelliseen vaiheeseen, jos se katsotaan tarpeelliseksi. Jos esimerkiksi

vaatimukset muuttuvat käyttöönottovaiheessa, niin prosessissa palataan takaisin määrittelyvaiheeseen asti.

Vesiputousmalli alkaa vaatimusmäärittelystä, jossa asiakkaalta kerätään vaatimukset toteutettavalle järjestelmälle (tai ohjelmistolle). Tässä vaiheessa määritellään WWW-järjestelmän käyttötarkoitus ja päätetään kohdeyleisö. Lisäksi päätetään missä selaimissa ja käyttöjärjestelmissä WWW-sivuston tulisi toimia. Kansainvälisessä WWW-järjestelmässä valitaan myös kielet, joita järjestelmän tulisi tukea. Vaatimusmäärittelyvaiheen tuotoksena ovat käyttötapaukset (*use cases*). Tuotos verifioidaan asiakkaan kanssa katselmoimalla käyttötapaukset. Katselmointi siis on vaatimusmäärittelyvaiheen hyväksymistestaus.

Määrittelyvaiheessa määritellään järjestelmän arkkitehtuuri ja valitaan käytettävät tekniikat. Tässä vaiheessa asiakkaan yleiset vaatimukset järjestelmälle muutetaan teknisiksi ohjelmistovaatimuksiksi. Haikalan ja Märijärven mukaan ohjelmistovaatimuksia ovat esimerkiksi vasteaika, suoritusteho ja käytettävyys. Vesiputousmallin suunnitteluvaiheessa taas suunnitellaan WWW-sovelluksen sisäinen toiminta. Tähän vaiheeseen kuuluu myös käyttöliittymäsuunnittelu, jolla on suuri merkitys erityisesti WWW-järjestelmissä

WWW-sovelluksen ohjelmointi ja fyysisen laiteympäristön pystytys kuuluvat toteutusvaiheeseen. Monikielisissä WWW-järjestelmissä tämän vaiheen testaus vaatii enemmän huomiota ja aikaa kuin samanlaisen yksikielisen järjestelmän testaus. Testaus olisi hyvä toteuttaa kaikilla vaatimusmäärittelyssä valituilla kielillä, jotta voidaan todeta järjestelmän vastaavan vaatimuksia. Jos järjestelmässä on käytetty vain yhtä merkistöä, testaus voidaan suorittaa helpommin ja nopeammin kuin jos järjestelmässä käytetään useampia merkistöjä. Mitä enemmän merkistöjä on testattavana, sitä enemmän testaukseen kuluu resursseja. WWW-järjestelmissä integrointivaihe on

tarpeellinen vain, jos järjestelmä koostuu monesta itsenäisestä osasta. Staattisten WWW-sivujen kohdalla tätä vaihetta ei tarvita. Kun kyseessä on inkrementaalinen kehitys, niin integrointivaiheessa uudet osat integroidaan jo olemassa olevaan järjestelmään.

Käyttöönotossa WWW-järjestelmästä tulee julkinen - ainakin valitulle kohderyhmälle. Ylläpito on Haikalan ja Märijärven (1997, 26) mukaan ”asiakkaan ongelmien ratkomista, virheiden korjaamista, ohjelman muuttamista vaatimusten muuttuessa sekä uusien piirteiden lisäämistä.” (Haikala & Märijärvi 1997, 24-27)

5.1.2 WWW-järjestelmien uudelleenrakentamisen malli

Goton ja Cotlerin (2001) mukaan WWW-järjestelmiä on pitkään suunniteltu ja rakennettu ad hoc –menetelmällä ja samoja virheitä on toistettu vielä silloinkin, kun järjestelmästä on suunniteltu täysin uusi versio. Jotta virheistä päästäisiin, Goto ja Cotler määrittelevät WWW-järjestelmien uudelleenrakentamiseen prosessin, joka on kuvattu kaaviossa 5. Goton ja Cotlerin mukaan prosessi sopii yhtä hyvin myös ensimmäistä kertaa luotaviin WWW-järjestelmiin kuin uudelleen rakennettaviinkin. Heidän mukaansa malli on niin yleinen, että sen avulla voidaan suunnitella ja toteuttaa mitä tahansa WWW-järjestelmiä niiden koosta ja käyttötarkoituksesta riippumatta.

KAAVIO 5. Goton ja Cotlerin WWW-järjestelmien uudelleenrakentamisen prosessimalli (Goto & Cotler 2001, 11)

Goton ja Cotlerin prosessimalli muistuttaa perusajatukseltaan aika paljon vesiputousmallia, joka on kuvattu kaaviossa 4. Goton ja Cotlerin projektin määrittelyvaihe vastaa vesiputousmallin vaatimusmäärittelyä. Siinä määritellään WWW-järjestelmän tarkoitus ja tehdään projektisuunnitelma.

WWW-sivuston rakenteen määrittely taas vastaa vesiputousmallin määrittelyvaihetta. Goton ja Cotlerin mallissa määrittelyvaiheessa keskitytään WWW-sivuston rakenteen ja sisällön määrittelyyn. He jakavat määrittelyn kolmeen osaan: sisällön yleiseen määrittelyyn, sivuston rakenteen määrittelyyn ja yksittäisten WWW-sivujen rakenteen määrittelyyn. Tässä vaiheessa suunnitellaan kansainvälistetäänkö sivusto vai lokalisoidaanko se. Jos sivusto päätetään lokalisoida, saattaa sillä olla vaikutusta kaikkiin kolmeen edellä mainittuun määrittelyosa-alueeseen. Sisällön määrittelyssä voidaan esimerkiksi päättää, että tiettyjä sivuja ei julkaista Aasian maihin lokalisoiduissa

sivustoversioissa, koska sivujen sisällöt eivät ole kulttuurillisesti niihin maihin sopivia. Sivun rakennetta suunniteltaessa taas pitää ottaa huomioon se, että joissain maissa lukusuunta on oikealta vasemmalle, jolloin koko valikkorakenne pitäisi pystyä kääntämään toiseen laitaan.

Goton ja Cotlerin mallissa sivuston ja sivujen visuaalinen suunnittelu on erotettu näistä rakenteen ja sisällön määrittelyistä omaksi osakokonaisuudekseen. Tämä osakokonaisuus vastaa vesiputousmallin suunnitteluvaihetta. Goto ja Cotler ovat sisällyttäneet tähän vaiheeseen myös testauksen, jolla varmistetaan, että sivusto näyttää siltä miltä pitääkin. Tässä vaiheessa lokalisoinnissa muokataan kuvista paikalliseen makuun sopivat versiot kullekin maalle. Kansainvälistettäessä kuvat ja muu ulkoasu suunnitellaan niin, että se ei loukkaa minkään kulttuurin edustajia. Lokalisoitaessa sivustoa usealle kielelle pitää muistaa varmistaa, että tekstit on luettavissa kaikilla kielillä. Jos sivustolle valitaan yksi yhteinen tyylitiedosto, jossa esimerkiksi fonttikoko on määritelty, pitää fontin olla niin iso, että myös kiinan, japanin ja korean kielisistä merkeistä saa selvää (Morgan ym. 2001, 231).

Toteutusvaihetta vastaa tuottaminen ja laadunvarmistus –vaihe. Laadunvarmistus on tässä mallissa sama kuin testaus. Vesiputousmallin integrointivaiheelle ei Gotolla ja Cotlerilla ole vastinetta, mutta käyttöönottoa ja ylläpitoa vastaa julkaisu ja ylläpito –vaihe.

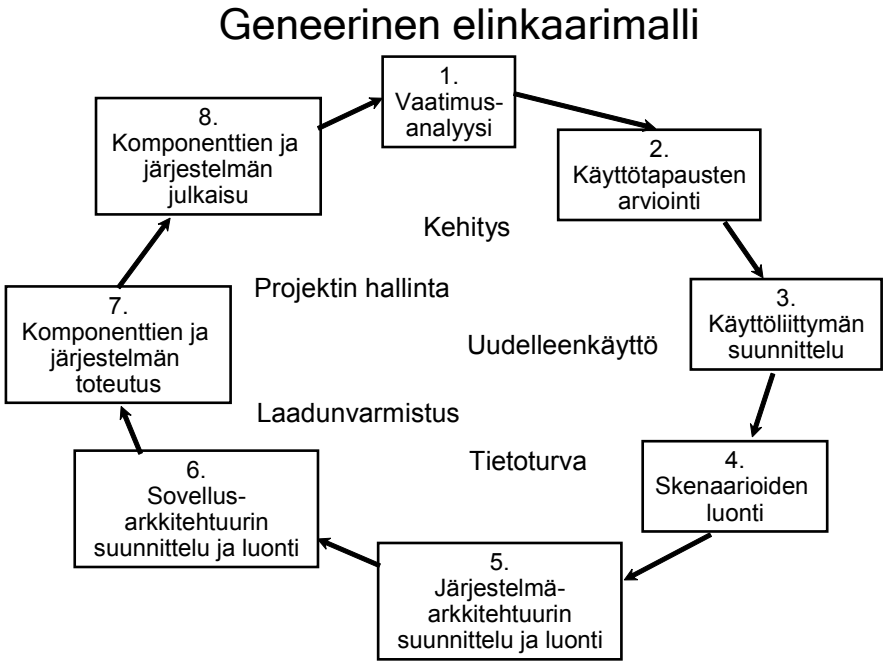
Goton ja Cotlerin mallista puuttuu järjestelmäarkkitehtuurin suunnittelu. Mallissa ei suunnitella millaisella palvelimella tai palvelimilla WWW-sovellusta ajetaan, millaisia tietokantoja tai muita tietovarastoja järjestelmä tarvitsee ja onko järjestelmällä yhteyksiä muihin järjestelmiin. WWW-sovelluksen sisäistä arkkitehtuuria ei myöskään suunnitella missään vaiheessa. Vesiputousmallissa nämä asiat kuuluvat suunnitteluvaiheeseen.

Mielestäni Goton ja Cotlerin malli sopii mielestäni parhaiten staattisten WWW-sivustojen suunnitteluun, joilla ei ole yhteyksiä muihin järjestelmiin ja jotka

eivät sisällä monimutkaista liiketoimintalogiikkaa. Malli on siis parhaimmillaan yksinkertaisen WWW-järjestelmän suunnittelussa. Yksinkertainen WWW-järjestelmä on kuvattu luvussa 2 olevassa kaaviossa 2. Tällaisen WWW-järjestelmän kohdalla Goton ja Cotlerin malli sisältää kaikki oleelliset vaiheet. (Goto & Cotler 2001, 1-11)

5.1.3 Geneerinen elinkaarimalli

Hieman monimutkaisempien WWW-järjestelmien suunnitteluun (ks. KAAVIO 3 luvussa 2) sopii Goton ja Cotlerin mallia paremmin Noackin ja Schienmannin (1998, 495 - 498) geneerinen elinkaarimalli, joka kattaa myös järjestelmäarkkitehtuurin ja sovellusarkkitehtuurin määrittelyn. Tämä malli on kuvattu kaaviossa 6.



KAAVIO 6. Noackin ja Schienmannin geneerinen elinkaarimalli (Noack & Schienmann 1998, 496)

Noack ja Schienmann kehittivät tämän geneerisen elinkaarimallinsa luodessaan WWW-järjestelmää pankin tarpeisiin. Mallin taustalla on muun muassa olio-

ohjelmoinnin vaatimuksia. Mallissa korostuu arkkitehtuurien suunnittelun tarpeellisuus sekä toisaalta iteratiivisuus ja inkrementaalisuus. Malli on luotu erityisesti tapauksiin, joissa järjestelmää kehitetään pienissä osissa. Inkrementtejä eli uusia sivuston osia tai toiminnallisuuksia lisätään, kun edelliset on julkaistu. Iteratiivisuus näkyy Noackin ja Schienmannin mukaan myös niin, että kehittäjä ja asiakas ovat jatkuvassa yhteistyössä koko prosessin ajan. Arkkitehtuurikeskeisyys taas sopii heidän mukaansa hyvin hajautettuihin järjestelmiin. Noack ja Schienmann toteavat mallinsa olevan toisaalta myös hyvin asiakaslähtöinen, koska skenaarioiden luonti on nostettu omaksi osa-alueekseen.

Noackin ja Schienmannin geneerisen elinkaarimallin vaiheissa on yhtäläisyyksiä vesiputousmalliin. Vesiputousmallin vaatimusmäärittely on tässä mallissa jaettu vaatimusanalyysiin ja käyttötapausten arviointiin. Vesiputousmallissahan vaatimusmäärittelyn tuotoksena oli juuri käyttötapaukset, jotka verifioidaan vaiheen päätyttyä. Geneerisessä elinkaarimallissa jaon avulla korostetaan vaatimusten tarkastamista ennen prosessissa eteenpäin siirtymistä. Mallissa on jaettu muitakin vesiputousmallin vaiheita tarkempiin vaiheisiin. Esimerkiksi vesiputousmallin suunnitteluvaihe on jaettu käyttöliittymän, järjestelmäarkkitehtuurin ja sovellusarkkitehtuurin suunnitteluihin. Geneerisessä elinkaarimallissa on siis tarkennettu vesiputousmallin vaiheita.

Noack ja Schienmann etenevät käyttötapausten määrittelystä käyttöliittymän suunnitteluun. WWW-sivuston rakenteen määrittelyä ei ole kuitenkaan nostettu omaksi kohdakseen kuten Goton ja Cotlerin mallissa. Mielenkiintoista geneerisessä elinkaarimallissa on se, että käyttöliittymän suunnittelu tulee ennen vesiputousmallin määrittelyvaihetta vastaavia vaiheita eli skenaarioiden luontia. Geneerisessä elinkaarimallissa siis suunnitellaan käyttöliittymä jo ennen kuin järjestelmän toiminta on määritelty. Koska järjestelmän toiminta

asettaa aina vaatimuksia käyttöliittymälle, niin mielestäni vaiheiden pitäisi olla joko samanaikaisia tai toisessa järjestyksessä.

Eroa muihin malleihin on myös siinä, että vesiputousmallin toteutusvaihe on jaettu useampaan osioon eli sovellus- ja järjestelmäarkkitehtuurien toteutukseen sekä komponenttien ja järjestelmän toteutukseen. Vesiputousmallin integrointivaihe puuttuu myös tästä mallista. Komponenttien ja järjestelmän julkaisu vastaa suoraan vesiputousmallin käyttöönottoa. Mallin iteratiivisuudesta johtuen ylläpitoa ei ole nostettu omaksi kohdakseen. Järjestelmän ylläpitoon kuuluvat tehtävät hoidetaan lisäämällä uusi iteraatiokierros, joka alkaa vaatimusanalyysistä ja päättyy julkaisuun.

Noackin ja Schienmannin geneerisessä elinkaarimallissa on lisäksi viisi osa-aluetta, jotka vaikuttavat prosessin aikana, mutta eivät ole varsinaisesti sidoksissa mihinkään tiettyyn prosessin vaiheeseen. Nämä osa-alueet ovat projektin hallinta, kehitys, uudelleenkäyttö, laadunvarmistus ja tietoturva. Projektipäällikön projektinhallinnalliset tehtävät kuuluvat luonnollisena osana jokaisen WWW-järjestelmän luontiin. Projektipäällikkö pitää huolta muun muassa aikatauluista ja resursseista. Geneerisessä elinkaarimallissa jokaisen julkaisun jälkeen projektipäällikkö arvioi projektin siihen mennessä saavutetut tulokset ja tekee uuden projektisuunnitelman seuraavalle iteraatiokierrokselle.

Koska geneerinen elinkaarimalli on iteratiivinen, järjestelmän kehitysideoita kirjataan ylös koko prosessin aikana ja niistä sopivimmat otetaan vaatimuksiksi tulevissa iteraatiokierroksissa. Kehityksellä voidaan tarkoittaa myös parempien toimintatapojen kehittämistä projektin kuluessa, kuten että dokumentointia pyritään parantamaan tai ohjelmakoodia luodessa käytetään tiettyjä nimeämiskäytäntöjä.

Uudelleenkäyttö pitää ottaa huomioon jo suunnitteluvaiheissa. Noack ja Schienmann toteavat, että uudelleenkäyttö parantaa ohjelmiston laatua ja nopeuttaa tuotantoa, mutta vaatii systemaattista lähestymistapaa. Jotain

valmiita komponentteja voidaan uudelleenkäyttää suunniteltavassa ohjelmistossa. Toisaalta, uusi järjestelmä voidaan luoda niin yleiskäyttöiseksi, että se voidaan konfiguroinnin avulla ottaa käyttöön jossain muussakin projektissa. Jos koko järjestelmää ei haluta luoda uudelleenkäytettäväksi, niin ainakin joitakin komponentteja voidaan luoda sellaisiksi. Tähän Noack ja Schienmann suosittelevat olio-ohjelmointia.

Laadunvarmistus näkyy vähintäänkin testauksena vaiheiden loputtua, mutta laadunvarmistukseen voi kuulua myös prosessien kehittäminen. Laatujärjestelmät määrittelevät yleensä laadunvarmistustehtäviä prosessin vaiheille vaatimusmäärittelystä ylläpitoon asti. Tietoturva pitää taas ottaa huomioon erityisesti suunnittelussa ja toteutuksessa.

5.2 Merkistöongelmat huomioiva WWW-järjestelmän luontiprosessi

Tämän luvun vaiheet perustuvat Noackin ja Schienmannin geneerisen elinkaarimallin vaiheisiin. Niiden lisäksi tähän lukuun on otettu vaiheiksi myös testaus ja ylläpito.

5.2.1 Vaatimusanalyysi

Vaatimusanalyysissä kerätään asiakkaalta vaatimukset WWW-järjestelmälle. Tässä vaiheessa selvitetään yleisellä tasolla mitä WWW-sivuston pitäisi sisältää ja millainen on järjestelmän fyysinen arkkitehtuuri. Myös sivuston kohdeyleisö päätetään tässä vaiheessa. Kun kyseessä on monikielinen WWW-järjestelmä, määritellään erityisesti millä kielillä käyttäjien pitäisi pystyä sivustoa käyttämään. Adamsin (1993, 31) mukaan tietojärjestelmien käyttäjien ei voida vaatia opettelemaan joitain yleisiä kieliä, kuten englantia, voidakseen käyttää tietojärjestelmiä. Adams kannattaa sitä, että käyttäjien pitäisi aina pystyä käyttämään tietojärjestelmiä omalla kielellään. Koska WWW-sivustot ovat yleensä Intranet ja Extranet -sivustoja lukuun ottamatta kaikkien saatavilla,

asiakas saattaa valita jotkin kohdemaat tai –alueet, joiden käyttäjille sivusto halutaan erityisesti suunnata. Kasatkinin mukaan vaatimusanalyysissä tuleekin aluksi määritellä onko sivusto globaali vai maakohtainen. Valituissa kohdemaissa puhutut kielet ovat myös yleensä sivuston toteutukseen valitut kielet. Globaalissa sivustossa valitaan toteutukseen usein joukko asiakkaan kannalta tärkeimpiä kieliä. Adamsin mukaan kohdemaat valikoituvat usein niiden markkinavoiman ja Internetin käyttöasteen perusteella. Adams toteaakin, että markkinavoimien vuoksi tietojärjestelmät julkaistaan yleensä vain joko länsieurooppalaisilla tai aasialaisilla kielillä. (Länsieurooppalaiset kielet kattavat myös Amerikoissa puhutut kielet.)

Kasatkin tuo haastattelussa esille, että vaatimusanalyysivaiheessa WWW-sivuston kansainvälistämisen ja lokalisoinnin tarpeen selvittäminen on erittäin tärkeää. Vaatimusanalyysissä Kasatkinin mukaan tulee selvittää mitkä järjestelmän osat ovat monikielisiä ja mitkä mahdollisesti yksikielisiä. Lisäksi asiakas voi haluta, että vain jotkin osat sivustosta on lokalisoitu ja jotkin eivät. Nämä tulee kirjata vaatimuksiksi järjestelmälle, koska näillä on merkitystä muun muassa sovellusarkkitehtuurin suunnittelulle ja toteutukselle. Koska lokalisointi kattaa muutakin kuin tekstien kääntämisen muille kielille, lokalisointivaatimuksilla on myös merkitystä sisällöntuottajille. Sisällöntuottajat vastaavat käyttäjälle näkyvien sivuston sisältöjen muokkaamisesta maakohtaisiksi. Sisällön lokalisointi on usein kallista ja aikaa vievää, joten Kasatkinin mukaan on hyvä arvioida mistä maista minkin kielistä sisältöä tullaan luultavimmin käyttämään. Näin lokalisointi osataan tehdä oikeille sivuston osille. Toisaalta jo vaatimusanalyysivaiheessa tulee määrittää kuka tulee hoitamaan sisällön ylläpidon, miten ja mistä. Sisällöntuotanto on hidasta, joten se tulee aloittaa hyvissä ajoin. Lisäksi WWW-sivuston sisältöä tulee päivittää usein sivuston julkaisunkin jälkeen, joten sisällöntuotannon prosessi ja vastuu olisi hyvä olla selvillä mahdollisimman varhaisessa vaiheessa.

Tekniseltä kannalta vaatimusanalyysivaiheessa tulee määrittää millaisissa käyttöympäristöissä järjestelmän tai sen eri osien tulee vähintään toimia. Kasatkinin mukaan olisi hyvä myös todeta, millaiset merkistöongelmat voidaan hyväksyä. Kaikki WWW-selaimet eivät tue kaikkia merkistöjä, joten valitut selaimet asettavat rajoituksia myös osaltaan WWW-järjestelmän merkistön ja enkoodauksen valinnalle. Kasatkin tuo esille, että merkistön valintaan vaikuttavat myös muut järjestelmälle asetettavat vaatimukset, kuten esimerkiksi kustannuksiin, toteutusvälineisiin ja olemassa olevien järjestelmien hyväksikäyttöön liittyvät vaatimukset. Jos merkistövalinta joudutaan suorittamaan useamman sopivan merkistön välillä, kannattaa valita merkistö, joka on standardoitu ja muutenkin jo laajassa käytössä. Näin taataan, että vaikka järjestelmän osia jouduttaisiin vaihtamaan, uudetkin osat todennäköisesti tukevat merkistöä. Standardien merkistöjen valintaa puoltaa myös se, että niiden fontit ovat yleensä aina valmiina käyttäjien koneiden käyttöjärjestelmissä. Oikeiden fonttien asentaminen järjestelmään ja asetusten oikein laittaminen on monille käyttäjille liian tekninen tehtävä.

Tärkein vaatimus merkistövalinnassa on Kasatkinin mukaan, että valittu merkistö kattaa kaikki kielet, jolla järjestelmää tullaan käyttämään. Merkistön vaihto on aina suuri työ, joten vaatimusanalyysivaiheessa kannattaa valita jo valmiiksi merkistö, joka tukee mahdollisimman monia kieliä. Toisaalta, kun tehdään kansainvälistä järjestelmää, kannattaa katsoa, että valittu merkistö on laajalti käytössä myös kohdemaassa. Varsinkin Aasian maissa on hyvin erilaiset merkistöt käytössä kuin Euroopassa.

5.2.2 Käyttötapausten arviointi

Vaatimusanalyysin yhtenä tuotoksena ovat käyttötapaukset. Käyttötapauksissa selitetään yksinkertaisesti mitä käyttäjä voi sivustolla tehdä ja miten. Noack ja Schienmann (1998) ovat geneerisessä elinkaarimallissa nostaneet käyttötapausten arvioinnin omaksi vaiheekseen. Vesiputousmallissa tämä vaihe

vastasi vaatimusmäärittelyn testausta. Nimensä mukaisesti tässä vaiheessa arvioidaan käyttötapaukset. Vaiheen aikana käyttötapauksista johdetaan tekniset vaatimukset WWW-järjestelmälle sekä tehdään aika- ja kustannusarviot projektille. Kun käyttötapausten arviointi päättyy, on järjestelmän yleiset vaatimukset päätetty ja hyväksytty.

Koska kyseessä on iteratiivinen prosessimalli, vaatimukset priorisoidaan ja tärkeimmät vaatimukset toteutetaan ensin. Muut vaatimukset joko hylätään tai päätetään toteuttaa seuraavien iteraatiokierrosten aikana. Tulevaisuuden vaatimukset järjestelmälle on kuitenkin hyvä pitää mielessä näitä päätöksiä tehtäessä. Kasatkin tuo esiin, että esimerkiksi erikoismerkkien käytön sallimista käyttäjätunnuksissa ja salasanoissa kannattaa harkita perusteellisesti. Vaikka ne toimisivat mainiosti alkuperäisessä tilanteessa, ongelmia voi olla odotettavissa järjestelmää uusittaessa tai uusia esimerkiksi mobiilikäyttöliittymiä lisättäessä, sillä mobiilipäätelaitteet eivät välttämättä tue kyseisiä merkistöjä. Toisaalta taas merkistö pitäisi valita mahdollisimman monia kieliä kattavaksi, sillä merkistön vaihto vaatii pahimmillaan ohjelmakoodin uudelleen kirjoittamista (Haible 2001). Vaikka siis monikielistä WWW-järjestelmää päätettäisiin alkaa toteuttaa kustannussyistä ensin vain yksikielisenä, toteutukseen on kuitenkin syytä valita merkistö, joka kattaa kaikki mahdolliset tulevatkin kielet.

5.2.3 Käyttöliittymän suunnittelu

Käyttöliittymäsuunnittelu pohjautuu hyväksyttyihin käyttötapauksiin ja vaatimuksiin. Käyttötapausten perusteella voidaan määritellä yleisesti millaisia WWW-sivuja sivuston tulee sisältää ja millaista toiminnallisuutta niissä tulee olla. Noackin ja Schienmannin (1998) käyttöliittymän suunnittelu –vaiheen voisi jakaa tarkempiin osiin Goton ja Cotlerin (2001) mallin mukaisesti. Goton ja Cotlerin mallissa määritellään ensin sivuston yleinen sisältö. Tämän jälkeen keskitytään koko WWW-sivuston ja yksittäisten WWW-sivujen rakenteiden

määrittelyyn. Lopuksi suunnitellaan sivuston ja yksittäisten sivujen visuaalinen ulkonäkö. (Goto ja Cotler 2001, 1-11)

Monikielisen WWW-järjestelmän käyttöliittymäsuunnittelussa erityisen tärkeitä vaatimuksia ovat lokalisoinnin ja kansainvälistämisen tarpeet, jotka vaikuttavat sekä sisällön, sivuston rakenteen että sivujen ulkonäön suunnitteluun. Kansainvälistettäessä Essenlinkin (2000) mukaan sivustosta tulee tehdä mahdollisimman geneerinen niin, että se on helppo käännättää uusille kielille ja julkaista uusiin maihin ilman suuria muutoksia ohjelmakoodiin tai muihin järjestelmän osiin. Hyvin kansainvälistetty sivusto ei sisällä mitään maakohtaista tietoa, vaan tekstit on vain käännetty uusille kielille. Tällöin sisällön kannalta pitää ottaa huomioon, että teksti on helppo kääntää uusille kielille eikä se sisällä maakohtaisia viittauksia. Essenlinkin mukaa lokalisoinnissa taas teksti sisältää paljon maakohtaista informaatiota ja kuvat saattavat olla erilaiset joka maalle. Sivusto on tällöin enemmänkin maakohtainen kuin kielikohtainen. Lokalisoinnilla on siis vaikutusta sivujen ulkonäönkin määrittelyyn, sillä lokalisoitaessa jokaisen sivuston kuvat on luotava erikseen. (Essenlink 2000, 2-5)

Kansainvälistetyssä sivustossa jokainen sivu on käännetty kaikille valituille kielille. Lokalisoidussa versiossa taas jotkin sivut on saatettu kääntää vain joillekin kielille. Lokalisoinnissa sivuston rakenne määritellään siis maakohtaisesti. Lisäksi lokalisoitaessa samalla kielellä on saatettu julkaista useampi eri versio sivustosta, koska eri maissa voi olla samoja kieliä. Esimerkiksi sekä Suomelle että Ruotsille sivustosta voi olla julkaistuna ruotsinkieliset versiot, jotka poikkeavat toisistaan. Jos järjestelmässä on joitain osia, jotka eivät toimi tiettyjen kielten kanssa (esimerkiksi sähköpostin lähettäminen tai tiedon haku), voidaan lokalisoidessa päättää, että kyseisiä toimintoja ei julkaista maihin, joissa kielet ovat käytössä.

Morganin, Luttrellin ja Liun (2001) mukaan merkistöjen kannalta on otettava huomioon, että joidenkin kielten kanssa fonttikoko ei voi olla niin pieni kuin toisten kielten kanssa voi. Kansainvälistettäessä on siis suunniteltava tekstien fonttikoko sen verran suureksi, että kiinalaisistakin merkeistä saa selvää. Sivujen ulkonäön suunnittelussa on myös otettava huomioon, että joissain kielissä sanat ovat huomattavasti pidempiä kuin toisissa. Varsinkin valikkorakennetta suunniteltaessa tämä tulee ottaa huomioon. Lisäksi sivuston ja sivujen rakennetta määriteltäessä kannattaa muistaa, että joissain maissa lukusuunta on oikealta vasemmalle (heprea) tai pystysuoraan ylhäältä alas (kiina), jolloin esimerkiksi valikkorakenne joudutaan muuttamaan. (Morgan ym. 2001, 230-231)

Sisällön yleisessä määrittelyssä pitää myös määritellä näytetäänkö sivulla joitain tekstejä kuvina. Monissa WWW-sovellusten suunnitteluoppaissa neuvotaankin tekijöitä laittamaan ISO-8859-1 merkeistä eroavat tekstit kuvina sivuille, jos kyseisiä tekstejä ei ole paljon. Silloin käyttäjä näkee ne ilman erityisiä toimenpiteitä. Yksi näistä oppaista on Morganin ja muiden (2001) Designing Multiligual Web Sites. Kuvat kasvattavat kuitenkin sivun tavukokoa, mikä tekee sivujen lataamisesta hitaampaa. Siksi niitä ei pitäisi käyttää tilanteissa, joissa suurin osa tekstistä vaatii jonkin muun merkistön kuin ISO-8859-1. Morgan ym. varoittavat vielä siitä tosiseikasta, että tekstiä on vaikeampi ylläpitää, jos osa teksteistä on upotettuna kuviin. (Morgan ym. 2001, 231) Toisaalta kaikki selaimet eivät myöskään tue kuvia. Aaltosen mukaan kuviin ei pitäisi myöskään laittaa tärkeitä termejä, joilla dokumenttia saatetaan hakea. Hakukoneet löytävät vain tekstissä olevat sanat, eivät kuvissa olevia.

Aaltonen muistuttaa, että alasvetovalikoihin (*drop down list*) ei saisi suunnitella tulevan tekstejä jotka kuuluvat eri merkistöihin. Esimerkiksi monikielisillä, paikallisia merkistöjä käyttävillä WWW-sivustoilla saatetaan haluta olevan aloitussivun, jossa käyttäjä saa valita kielen, jolla sivuston muut sivut näytetään. Kielet halutaan yleensä listata paikallisilla nimillä (English, Suomi,

Deutch…), jolloin esimerkiksi kiinan kielen nimi muodostuu ongelmaksi. Kiinalaisia merkkejä ei ole esimerkiksi ISO-8859-1 –merkistössä, joten jos aloitussivu näytetään sillä merkistöllä, niin ”kiina” näkyy selaimen oletusmerkkeinä (□, | tai ~). Aaltosen mukaan tällainen lista saadaan näkymään oikein vain jos kaikki tekstit ovat Unicodena ja sivun merkistöksi on myös valittu Unicode.

5.2.4 Skenaarioiden luonti

Skenaarioiden luonti pohjautuu sekin käyttötapauksiin. Kasatkinin mukaan tässä vaiheessa karkean tason käyttötapauksia syvennetään niin, että tiedetään mitä toimintoja järjestelmässä tarkalleen ottaen on, mitä merkkejä voi esiintyä missäkin käyttöliittymässä ja ketkä sivustoa käyttävät ja millä kielillä. Lisäksi tässä vaiheessa otetaan huomioon käyttöliittymäsuunnittelun asettamat rajoitteet sivuston toiminnalle. Suunnittelijan tulee huomioida, että nämä tarkennetut suunnitelmat eivät ole ristiriidassa sen tavan kanssa, jolla on käyttötapausten arviointi -vaiheessa päätetty toteuttaa järjestelmän monikielisyys ja erilaisten merkkien tuki.

5.2.5 Järjestelmäarkkitehtuurin suunnittelu ja luonti

Järjestelmäarkkitehtuuria suunnitellessa päätetään mitä fyysisiä laitteita järjestelmään otetaan ja mitä ohjelmistoja tai sovelluksia niillä tulee olla WWW-sovelluksen lisäksi. Suunnitellessa kannattaa varmistaa, että laitteilla ja ohjelmistoilla on tuki valitulle merkistölle. Jostain tuotteesta voi olla tarjolla myös kaksi versiota; kansainvälinen ja Amerikan markkinoille suunnattu. Näin on esimerkiksi Java 2 Runtime Environment (JRE) Standard Edition versiosta 1.3.1 Windowsille (Sun Microsystems 2002). Jos sovelluskehitystä aiotaan tehdä Javalla, monikielisen WWW-järjestelmään kannattaa valita kansainvälinen versio, koska sillä on laajempi tuki merkistöille kuin US-only –versiolla.

Suunnittelussa määritellään käytettävät tiedonsiirtoväylät ja –standardit. WWW-järjestelmissä yleisin protokolla on HTTP–protokolla. Siitä kannattaa käyttää uusinta eli HTTP/1.1 –versiota, sillä HTTP/1.0 –protokollassa esimerkiksi merkistön määrittelyä ei tueta yhtä hyvin kuin HTTP/1.1 –protokollassa (Fielding ym. 1999). Käytettävä protokolla vaikuttaa siihen kuinka merkistö ja enkoodaus tulee ilmoittaa WWW-dokumentissa.

Järjestelmäarkkitehtuuria suunniteltaessa on otettava huomioon myös mahdolliset olemassa olevat järjestelmät, joilla tulee olemaan yhteyksiä luotavaan järjestelmään. Haastattelussa Kasatkin ehdottaa, että jos näiden järjestelmän osien kyvystä käsitellä merkistöjä suunnitellulla tavalla on vain epävarmaa tai toisen käden tietoa, tulisi niiden merkistökäsittely testata jo tässä vaiheessa. Jos tuki ei ole sellainen kuin pitäisi ja se huomataan vasta lopputestauksessa, toiminnallisuudesta voidaan joutua tinkimään. Kasatkinin mukaan tällaiset ongelmat ovat yleensä isoja ja vaikeasti korjattavia. Arkkitehtuurisuunnittelun aikana voidaan vielä päättää, että osan tilalla käytetäänkin jotain muuta osaa, jolla on parempi tuki merkistölle. Joskus on mahdollista lisätä tuki järjestelmään lisäkomponenttien avulla, kuten Unicode-tuki on lisätty Microstoftin 95:een ja 98:iin MSLU:n avulla. Toinen vaihtoehto on konvertoida teksti toiseen merkistö- tai enkoodausmuotoon siksi aikaa, kun kyseinen järjestelmän osa käsittelee sitä.

Merkistöjen käsittelyssä on tärkeää, että WWW-järjestelmän osat tukevat valittua merkistöä ja enkoodausta. Jos yksikin tekstiä käsittelevä järjestelmän osa (esimerkiksi WWW-sovellus, selain tai tietonsiirtoväylä) ei tue merkistöä tai enkoodausta, niin virhe näkyy aina käyttäjälle lukukelvottomana tekstinä sivustolla. Samanlaisia ongelmia tulee, jos jokin järjestelmän osa tukee eri merkistön enkoodausta (little endian, big endian). Myös Aaltonen tuo haastattelussa ilmi, että kaikki järjestelmät ja järjestelmän osat eivät tue kaikkia merkistöjä. Valittu merkistö voi olla niin uusi, että järjestelmän osa on julkaistu ennen merkistöä, eikä siksi ei tue sitä. Kannattaakin tarkistaa onko järjestelmän

osasta julkaistuna uudempaa versiota, joka tukee uutta merkistöä. Vaikka Unicode-tuki yleistyykin, niin esimerkiksi vanhemmat selaimet eivät tue sitä. Joissain WWW-sovelluksissa halutaan tukea laajasti myös vanhoja selaimia ja siksi niissä käytetään mieluummin vanhoja paikallisia merkistöjä kuin Unicodea.

5.2.6 Sovellusarkkitehtuurin suunnittelu ja luonti

Sovellusarkkitehtuurin suunnittelussa määritellään millaisia sovelluksen osia järjestelmässä tulee olla. Tässä vaiheessa viimeistään valitaan toteutukseen käytettävä ohjelmointikieli, jollei sitä ole jo aiemmin valittu. Ohjelmointikieltä valitessa kannattaa pitää mielessä, että eri kielillä on erilainen tuki merkistöille. Esimerkiksi Java tukee suoraan Unicodea, kun taas C++:ssa Unicodelle joudutaan käyttämään eri metodeja kuin 7– tai 8-bittisille merkistöille (Haible 2001). Aaltonen toteaa haastattelussa, että konversio-ohjelmilla teksti voidaan muuttaa toiseen merkistö- ja enkoodausmuotoon käsittelyn ajaksi, mutta konvertoinnissa on aina se riski, että konversio ei onnistu täydellisesti. Siksi kannattaa toteutukseen valita suoraan ohjelmointikieli, joka tukee suoraan valittua merkistöä ja enkoodausta.

Sovellusarkkitehtuurin suunnittelussa määritellään myös sovelluksen luontiin ja ylläpitoon käytettävät työkalut, valmiskomponentit ja valmisohjelmistot. Monikielisissä WWW-järjestelmissä kannattaa kiinnittää huomiota etenkin sisällönhallinnan välineisiin. Näiden merkistötuki ei välttämättä ole riittävä. Tämä usein tarkoittaa sitä, että koodaajilla pitäisi olla tarvittavaa osaamista luoda konvertterit merkistömuodosta toiseen itse, niin että tieto pystytään pitämään samana koko käsittelyprosessin ajan. Kasatkin suosittelee tässä tapauksessa osien vaihtamista sellaisiin, jotka tukevat merkistöä, jos se vain on mahdollista.

WWW-ohjelmistoa suunniteltaessa kannattaa pitää mielessä, että ohjelmisto saatetaan ehkä haluta siirtää toiseen laiteympäristöön, kuin mitä alun perin on suunniteltu. Merkistöjen kannalta ohjelmisto on helposti siirrettävissä järjestelmästä toiseen, jos merkistö ja enkoodaus ovat tuettuna uudessakin järjestelmässä. Tunnetuimmat merkistöt kuten Unicode ja ISO-8859-1 ovat yleensä varmoja valintoja. Grahamin (2000, 229) mukaan monikielisten ohjelmistojen siirrettävyyttä helpottaa, jos merkistövalintana on Unicode, sillä Unicodea tuetaan jo laajalti eri tuotteissa. Paikallisten merkistöjen kanssa tilanne on paljon huonompi. Valittua merkistöä ei välttämättä tueta toisessa järjestelmässä tai järjestelmän osassa ollenkaan. Ongelmia saattaa tulla myös Unicodea käytettäessä. Graham toteaa, että Unicoden kanssa ongelmia tuottavat erityisesti big endian ja little endian –enkoodaukset, jotka vaihtelevat eri valmistajien laitteissa. Myös vaihtelevamittainen ja kiinteämittainen enkoodaus saattavat tuottaa päänvaivaa. Nämä ovat kuitenkin aika pieniä ja helposti korjattavissa olevia ongelmia, sillä Unicodessa on määritelty kuinka eri enkoodausmuodosta voidaan vaihtaa toiseen. Järjestelmän osien pitäisi siis hallita enkoodausmuotojen vaihdokset tarvittaessa. BOM-merkin tutkinta on helppo tunnistustapa big endian ja little endian –muodoille. (Graham 2000, 67-68)

Sovellusarkkitehtuurin suunnitteluun kuuluu myös nimeämiskäytäntöjen luominen. WWW-järjestelmissä pitää kiinnittää Aaltosen mukaan erityistä huomiota siihen, että url-osoitteissa ei käytetä mitään muita kuin ASCII-merkkejä. Jos WWW-sivun nimessä on esimerkiksi ä-kirjain, useimmat sovellus- tai http-palvelinohjelmistot (*application server, http-sever*) eivät pysty välittämään sivua käyttäjälle.

5.2.7 Komponenttien ja järjestelmän toteutus

Kasatkin toteaa, että kun prosessin mukaiset suunnitelmat on tehty kunnolla ja merkistöongelmat on huomioitu tarpeeksi ajoissa, järjestelmän toteuttaminen

suunnitelmien mukaan pitäisi olla suhteellisen suoraviivaista. Monikielisen WWW-järjestelmän toteutusvaiheessa saattaa kuitenkin ilmetä ongelmia työkalujen, valmiskomponenttien tai valmisohjelmistojen merkistöjen tuessa, jolloin arkkitehtuuria voidaan joutua joiltain osin muuttamaan ja osia vaihtamaan. Kasatkin suositteleekin, että järjestelmän osat ja käytettävät komponentit testattaisiin eri kielillä jo ennen tuotantovaihetta, jotta ikäviltä yllätyksiltä säästyttäisiin. Kannattaa myös varmistaa, että kaikki järjestelmän osat tukevat samaa enkoodausmuotoa merkistölle. Jos kaikki osat eivät tue valittua merkistöä tai enkoodausta, voidaan järjestelmään lisätä konverttereita, jotka muuttavat merkistön ja enkoodauksen sellaiseen muotoon, että tukematonkin osa voi sitä käsitellä. Tarjolla on muutamia valmiskonverttereita, mutta usein konvertterit koodataan itse, jolloin niiden toiminta pitää testata tarkasti. Konversion epäonnistuminen näkyy käyttäjälle lukukelvottomana tekstinä WWW-sivulla. (Freed & Borenstein 1996)

Valittu merkistö saattaa vaatia jotain erityistä WWW-sovelluksen ohjelmakoodiin. Esimerkiksi Unicoden kohdalla Perl 5.6 tallentaa merkkijonot UTF-8 muodossa vain jos ohjelmakoodiin kirjoittaa ”use utf8;”. Muiden yli 8-bittisten merkkijonojen käsittelyyn Perlissä tarvitaan konvertointikomponentti ja aiemmat kuin 5.6 –versiot eivät tue Unicoden käsittelyä lainkaan. C++:ssa taas Unicodelle joudutaan käyttämään eri tietotyyppiä (wchar_t) kuin 7– tai 8-bittisille merkistöille (char). (Haible 2001)

Monikielisen WWW-sivuston toteutuksessa on tärkeää, että käytetty merkistö ja enkoodaus merkitään oikein jokaiseen WWW-dokumenttiin, jotta käyttäjän selain osaisi näyttää dokumentin automaattisesti oikein. HTML 4.01 –spesifikaation (1999) mukaan helpoin tapa ilmoittaa dokumentin sisällön merkistö ja enkoodaus on käyttää ContentType –kentän Charset-parametria. Merkintätavat on kuvattu luvussa 4.4. Lisäksi Unicodea käytettäessä enkoodaus tulee ilmoittaa HTML-dokumentin tekstiosuuden alussa BOM-merkkillä (Byte Order Mark), joka määrittää onko Unicodesta käytössä big endian vai little

endian –muoto. HTML 4.01 -spesifikaation (1999) mukaan selaimen tulisi käyttää dokumentissa ilmaistua merkistöä ja enkoodausta oletuksena ja ohittaa käyttäjän itse selaimesta tekemät valinnat. Kaikki selaimet evät kuitenkaan noudata tätä. Koska käyttäjää ei voi estää valitsemasta selaimestaan toista merkistöä sivujen katseluun, on merkistön ja enkoodauksen ilmoittaminen WWW-dokumentissa on ainoa tapa, jolla WWW-järjestelmän luoja voi yrittää vaikuttaa siihen, että selain näyttäisi sivut oikein. (Raggett, Le Hors & Jacobs 1999)

Käytettävästä tiedonsiirtoprotokollasta riippuen tulisi tiedonsiirron aikana käytettävä merkistö ja enkoodaus ilmoittaa WWW-dokumentissa. Tämä voidaan ilmoittaa Content-Transfer-Encoding tai Transfer-Encoding –parametrin avulla. Kuitenkaan HTTP/1.0 –protokollan kanssa ei tiedonsiirron merkistöä ja enkoodausta merkitä. (Fielding ym. 1999)

Tiedonsiirrossa olisi turvallisinta käyttää 7-bittistä enkoodausta merkistöille, sillä sitä tukevat kaikki tiedonsiirtoväylät. Esimerkiksi Unicodelle suositellaan tiedonsiirtoon UTF-7 –muotoa (The Unicode Consortium 2000). Turnbullin (1999) mukaan 7-bittisessä muodossa on lisäksi se hyvä puoli, että 8-bittisessä tiedonsiirtoväylässä kahdeksatta bittiä voidaan käyttää pariteettibittinä tiedonsiirron onnistumisen tarkistamiseen. Tekstin tiivistämistä tiedonsiirron ajaksi ei kannata tehdä, ellei se tiedonsiirtoväylän hitauden vuoksi ole aivan välttämätöntä, sillä tekstin muuttaminen muodosta toiseen sisältää aina riskin, että muutos ei onnistu. Tiivistäminen ilmoitetaan Accept-Encoding ja Content-Encoding –parametrien avulla. (Fielding ym. 1999)

Luvussa 4 todettiin, että suurin osa käyttäjien havaitsemista merkistöongelmista johtuu käyttäjän koneen puuttuvasta fontista. Useimmissa WWW-järjestelmissä tätä ongelmaa ei edes yritetä korjata, vaan luotetaan siihen, että käyttäjän selain ilmoittaa puuttuvasta fontista ja käyttäjä osaa itse korjata asian. Korjausvaihtoehtoina ovat tekstien näyttäminen kuvina tai

dynaamisten fonttien käyttö. Jos tekstit näytetään kuvina, sivustosta tulee hidas ladata sekä vaikea ylläpitää (Morgan ym. 2001, 231). Lisäksi Aaltonen tuo esiin, että hakukoneet eivät löydä tekstejä kuvista. Dynaamiset fontit taas toimivat niin, että WWW-dokumentin mukana käyttäjän koneelle latautuu myös sivun merkeille fontit. Dynaamiset fontit eivät kuitenkaan toimi kaikissa selaimissa. Lisäksi sivujen latausaika pitenee, kun fontit ladataan jokaiselle käyttäjälle aina sivua ladattaessa. (Bitstream 2002, Netscape 2002, Netscape 2000)

Sivustolla ei yleensä voida näyttää kerrallaan kuin yhteen merkistöön kuuluvia merkkejä. Poikkeuksena on HTML:n FRAME-tagi, jonka avulla yksi WWW-sivu voidaan jakaa useampiin ”alasivuihin”, joista jokaiselle voidaan määritellä oma merkistö ja enkoodaus. Aaltonen kuitenkin varoittaa, että kaikki selaimet eivät tue FRAME-määritystä. HTML mahdollistaa myös, että joitakin tiettyjä merkkejä voidaan ilmaista HTML-standardin mukaisina entityinä (eli esimerkiksi euro-merkki muodossa &eur;), vaikka ne eivät kuulukaan merkistöön. Aaltonen tuo kuitenkin haastattelussa esiin, että esimerkiksi vanhat Netscapen selaimet eivät osaa näyttää euroa entity-merkinnälläkään oikein.

Jos käyttäjällä on mahdollisuus syöttää järjestelmään tekstiä WWW-sivustolta, niin sisään tuleva syöte ei välttämättä kuuluu samaan merkistöön tai käytä samaa enkoodausta kuin WWW-järjestelmä. Käyttäjän syöte olisi hyvä siis tarkistaa, varsinkin jos syöte esimerkiksi julkaistaan sivustolla sellaisenaan tai tallennetaan myöhempää käyttöä varten. Mäkisen (2002, 16-17) mukaan HTML-lomakkeelle kannattaa tallentaa erikoisia merkkejä sisältävä merkkijono, jonka muuttuessa voidaan todeta, että käyttäjän syötekin on jossain muussa enkoodaus- tai merkistömuodossa, kuin mitä järjestelmä odottaa sen olevan. Erityisen tärkeää Kasatkinin haastattelun mukaan on syötteen tarkastaminen tapauksissa, joissa sisällöntuotannossa on käytössä WWW-käyttöliittymällä varustettu työkalu.

5.2.8 Testaus

Testaus puuttuu erillisenä vaiheena kokonaan Noackin ja Schienmannin (1998) geneerisestä elinkaarimallista, mutta mallissa voidaan katsoa laadunvarmistuksen vastaavan muiden mallien testausvaihetta. Päätin lisätä testauksen kuitenkin erilliseksi kohdakseen. Kasatkin painottaa, että monikielisten WWW-järjestelmien kohdalla testaus on yksi kolmesta kaikkein keskeisimmästä vaiheesta järjestelmän luontiprosessissa. Kasatkin listaa kahdeksi muuksi tärkeäksi vaiheeksi vaatimusanalyysin ja arkkitehtuurin suunnittelun. Hänen mukaansa testaus ohitetaan usein liian kevyesti. Mitä enemmän WWW-järjestelmässä on merkistöjä ja osia, sitä enemmän testaukseen tulee panostaa. On hyvä muistaa, että myös sisällönsyöttötyökalut ovat osa järjestelmää, ja siksi nekin tulee testata mieluiten jo ennen sovellustestauksen alkua. Kasatkinin mukaan testauksen päätarkoituksena on varmistaa, että järjestelmän kaikki osat on testattu tuotantoympäristöä vastaavassa ympäristössä kaikkia keskeisiä käyttötilanteita silmälläpitäen.

Testaukseen varatut resurssit ovat yleensä aika rajalliset. Kasatkin toteaakin, että monikielisessä WWW-järjestelmässä testaus tulee tehdä riittävän monella kielellä riittävän monessa käyttöympäristössä. Testausta ei välttämättä tarvitse suorittaa kaikilla kielillä, joilla järjestelmä tullaan julkaisemaan. Joskus pari kieltä voi olla tarpeeksi, jos järjestelmässä on vain yksi merkistö. Jos WWW-järjestelmässä on käytössä useampia merkistöjä, ne kaikki tulee kuitenkin testata vähintään kahden kielen avulla. Paikallisia merkistöjä käytettäessä testaajilta odotetaan, että he tietävät mitkä merkit kuuluvat mihinkin merkistöön. Muuten he eivät voi testata kattavasti. Monikielisen WWW-järjestelmän testausvaihetta nopeuttaa ja helpottaa siis se, että toteutukseen on valittu vain yksi merkistö, joka kattaa kaikki kielet. Kasatkin kuitenkin varoittaa, että edes Unicoden kohdalla ei tulisi tyytyä testaamaan vain yhdellä kielellä. Hän tuo esiin, että esimerkiksi mahdolliset kiinteästi ohjelmakoodiin upotetut tekstit (ns. kovakoodaukset) tulevat ilmi vasta useammilla kielillä

testattaessa. Lisäksi Unicodenkaan kanssa ei pitäisi kuitenkaan tyytyä englanninkielellä testaamiseen, sillä mahdolliset järjestelmän osien yhteensopivuusongelmat tulevat parhaiten ilmi Unicode-merkistön yläpäähän sijoittuvilla merkeillä. Tästä syystä Unicodea käytettäessä olisi erittäin tärkeää testata muillakin kuin ASCII-merkeillä, esimerkiksi japanilla tai kiinalla. Shabalin (2002) ehdottaakin, että kansainvälinen ohjelmisto kannattaisi testata vähintään saksalla, japanilla ja tekstillä, joka on yhdistelmä niistä molemmista. Hänen mukaansa ainakin Microsoftin Windows 2000 -käyttöjärjestelmää käytettäessä näillä yhdistelmillä saadaan ilmi suurimmat virheet.

5.2.9 Komponenttien ja järjestelmän julkaisu

Jos järjestelmä on testattu huolella tuotantoympäristöä vastaavassa ympäristössä ja se on testauksen aikana toiminut oikein, Kasatkinin mukaan julkaisun ei pitäisi aiheuttaa ongelmia. Hän toteaa, että tällöin ainoaksi ongelmaksi saattaa tulla sisällöntuotanto. Sisällöntuotanto on usein hidasta, joten julkaisuaikataulua saattaa venyttää se, että kaikkea sisältöä ei ehditä saada ajoissa valmiiksi, sekä tarkastettua ja hyväksyttyä.

5.2.10 Sisällön ylläpito

Ylläpito on toinen kohta, joka puuttuu Noackin ja Schienmannin (1998) geneerisestä elinkaarimallista. Syynä tähän on mallin iteratiivisuus. Noackin ja Schienmannin mukaan kaikki julkaisun jälkeen tehtävä kehitys tulee tuottaa käymällä kaikki geneerisen elinkaarimallin vaiheet läpi. Nostin sisällön ylläpidon kuitenkin omaksi kohdakseen, sillä monikielisessä WWW-järjestelmässä mielestäni sisällön päivittämiseen ei tarvita kaikkia edellisiä prosessivaiheita. Sisällön päivittäminen on kuitenkin yksi tärkeitä useimmin toistuvia WWW-järjestelmän vaiheita.

Kasatkinin mukaan sisällön ylläpidossa on tärkeää, että sisällön ylläpitäjille on annettu tarkat ohjeet siitä, miten, millä välineillä ja missä ympäristössä sisältö tulee tuottaa. Erityisen tärkeää on, että kaikki tuntevat mahdolliset tiedossa olevat rajoitukset merkistöjen käsittelyssä. Jos jossain järjestelmän osassa ei esimerkiksi teknisten ongelmien vuoksi saa käyttää kansallisia erikoismerkkejä, niin sisällöntuottajille on painotettava, että niitä ei tule käyttää teksteissä.

5.3 Yhteenveto

Tässä luvussa esittelin kaksi WWW-järjestelmien luomiseen sopivaa prosessia; Goton ja Cotlerin WWW-järjestelmien uudelleenrakentamisen mallin sekä Noackin ja Schienmannin geneerisen elinkaarimallin. Molemmilla malleilla on yhtäläisyyksiä vesiputousmalliin, joten esittelin sen luvun alussa. Goton ja Cotlerin malli sopii parhaiten yksinkertaisten WWW-järjestelmien luomiseen, lähinnä staattisten WWW-sivujen julkaisuun. Goton ja Cotlerin mallissa painotetaan erityisesti käyttöliittymäsuunnittelua ja sen eri osa-alueita kuten sisällön määrittelyä ja sivuston rakenteen määrittelyä. Noackin ja Schienmannin geneerisessä elinkaarimallissa painottuu taas enemmän arkkitehtuurisuunnittelu, joka Goton ja Cotlerin mallista puuttuu kokonaan. Geneerinen elinkaarimalli soveltuukin parhaiten monimutkaisten WWW-järjestelmien luomiseen. Noackin ja Schienmannin geneerinen elinkaarimalli jakautuu kahdeksaan vaiheeseen, jotka ovat: vaatimusanalyysi, käyttötapausten arviointi, käyttöliittymän suunnittelu, skenaarioiden luonti, järjestelmäarkkitehtuurin suunnittelu ja luonti, sovellusarkkitehtuurin suunnittelu ja luonti, komponenttien ja järjestelmän toteutus sekä komponenttien ja järjestelmän julkaisu. Lisäksi mallissa on viisi osa-aluetta, jotka vaikuttavat useampaan geneerisen elinkaarimallin vaiheeseen. Näitä ovat projektin hallinta, kehitys, uudelleenkäyttö, laadunvarmistus ja tietoturva.

Luvussa kävin läpi kuinka luvussa 4 esiin tulleita merkistöongelmia voidaan välttää WWW-järjestelmän tuotantoprosessissa. Käsittelin myös haastattelujen

pohjalta mitä muuta pitää ottaa huomioon prosessin eri vaiheissa kun luodaan monikielistä WWW-järjestelmää. WWW-järjestelmän tuotantoprosessiksi otin Noackin ja Schienmannin geneerisen elinkaarimallin ja lisäsin siihen Goton ja Cotlerin mallista vielä kaksi vaihetta; testauksen ja sisällön ylläpidon.

Vaatimusanalyysivaiheessa määritellään yleiset vaatimukset luotavalle järjestelmälle, kuten mitä toimintoja järjestelmä sisältää, ketkä ovat sen käyttäjiä ja millaisessa laiteympäristössä järjestelmä toimii. Monikielisen WWW-järjestelmän vaatimusanalyysissä pitää lisäksi päättää onko järjestelmä globaali, vai käytetäänkö sitä vain joistain maista ja mitkä tulevat olemaan kielet, joilla järjestelmää voi käyttää. Kansainvälistämisen ja lokalisoinnin tarve pitää myös määritellä. Jo vaatimusanalyysivaiheessa olisi hyvä valita alustavasti järjestelmässä käytettävä merkistö ja enkoodaus.

Vaatimusanalyysin tuotoksena ovat käyttötapaukset, joissa järjestelmän karkean tason vaatimukset tarkennetaan. Käyttötapausten arvioinnissa vaatimukset priorisoidaan ja päätetään mitkä niistä toteutetaan ensin. Vaikka WWW-järjestelmä toteutettaisiin ensin yksikielisenä, kannattaa se silti suunnitella toimimaan myös useamman kielen kanssa.

Käyttöliittymäsuunnittelun voi jakaa Goton ja Cotlerin mallin mukaan sivuston yleisen sisällön määrittelyyn, koko sivuston sekä yksittäisten WWW-sivujen rakenteiden määrittelyyn ja sivuston sekä yksittäisten sivujen visuaaliseen suunnitteluun. Käyttöliittymän suunnittelussa on tärkeää ottaa huomioon, että WWW-sivuilla voidaan yleensä näyttää vain yhden merkistön merkkejä kerrallaan. Jos valittu merkistö ei kata kaikkia haluttuja merkkejä, pitää päättää näytetäänkö puuttuvat merkit/sanat kuvina, dynaamisten fonttien avulla vai vaihdetaanko sivun merkistö sellaiseksi joka kattaa kaikki tarvittavat merkit. Käyttöliittymän suunnitteluun vaikuttavat myös järjestelmään valitut kielet, sillä toisissa kielissä sanat ovat pidempiä kuin toiset ja siksi vaativat enemmän tilaa esimerkiksi valikoissa. Joissain kielissä lukusuunta on oikealta

vasemmalle, jolloin esimerkiksi valikkorakenne joudutaan kääntämään sivun toiseen laitaan. Käyttötapauksia tarkennetaan vielä lisää skenaarioiden luontivaiheessa.

Järjestelmäarkkitehtuurin suunnittelussa määritellään tarkemmin mitä laitteita järjestelmään kuuluu ja mitä protokollia tai standardeja järjestelmässä käytetään. Lisäksi määritellään järjestelmän yhteydet muihin olemassa oleviin järjestelmiin. Tässä vaiheessa on tärkeää selvittää, että valitut laitteet ja järjestelmät tukevat valittua merkistöä ja enkoodausta. Jos ne eivät tue, pitää päättää, voidaanko järjestelmän osa vaihtaa toiseen tai lisätä järjestelmään konversio-ohjelma, jolla merkistö saadaan muutettua toiseen muotoon tätä osaa varten. Sovellusarkkitehtuurin suunnittelussa määritellään samat asiat ohjelmistokomponenteille. Lisäksi luodaan nimeämiskäytännöt ja määritellään sovelluksen ohjelmointikieli.

Jos on todettu ja testattu, että kaikki järjestelmään valitut osat tukevat valittua merkistöä ja enkoodausta, komponenttien ja järjestelmän toteutus on suhteellisen helppoa. Tärkeintä on määritellä WWW-dokumentteihin käytetty merkistö ja enkoodaus. Tiedonsiirrossa määrittely tulee tehdä käytetyn tiedonsiirtoprotokollan vaatimusten mukaisesti. Ohjelmointikieli saattaa vaatia erilaisia metodeja tai tietotyyppejä eri merkistöille, joten tämä kannattaa ottaa huomioon ohjelmoinnissa. Pitää myös muistaa, että WWW-järjestelmissä ei voida vaikuttaa siihen, vaihtaako käyttäjä selaimestaan merkistöä sivun lataamisen jälkeen. Tämän vuoksi käyttäjien syötteiden merkistö kannattaa tarkistaa varsinkin WWW-järjestelmissä, joissa käyttäjien syöttämä teksti julkaistaan suoraan sivustolle tai lähetetään sähköpostina. Syötetty teksti voi sisältää merkkejä muista merkistöistä, jotka suoraan julkaistuna muuttuvat lukukelvottomiksi.

Monikielisen WWW-järjestelmän testauksessa tulee kiinnittää huomiota siihen, että jokainen järjestelmän merkistö tulee testattua ainakin kahden kielen avulla.

Edes Unicoden kanssa ei kannata tukeutua vain yhteen kieleen, sillä järjestelmän merkistöongelmat paljastuvat parhaiten, kun testauksessa käytetään merkkejä merkistön ylä- ja alapäästä. WWW-järjestelmä pitäisi aina testata oikeaa käyttöympäristöä vastaavassa ympäristössä niillä selaimilla, joita on määritelty tuettavan.

Kun testaus on suoritettu, järjestelmä on valmis julkaistavaksi. Julkaisua saattaa myöhästyttää käännöstyön hitaus, jos sivusto julkaistaan monella kielellä. Julkaisun jälkeen alkaa sisällön ylläpito. Erityisen tärkeää on muistaa sisällön merkistörajoitukset WWW-järjestelmän osissa. Jos jokin osa ei tue joitain merkkejä, niin sinne ei pidä niitä yrittää laittaa.

Lopputuloksena on, että melkein kaikki luvun 4 merkistöongelmat voidaan välttää huolellisella järjestelmän suunnittelulla ja toteutuksella. Ainoastaan käyttäjän syötteen merkistön oikeellisuutta ei voida aina varmuudella tarkistaa eikä käyttäjän tekemiin selaimen merkistövaihdoksiin voi vaikuttaa.

6 YHTEENVETO

Tämän tutkielman tavoitteena oli koota yhteen eri lähteissä olevaa tietoa merkistöongelmista ja tavoista välttää niitä WWW-järjestelmiä luotaessa. WWW-järjestelmällä tarkoitan tässä tutkielmassa järjestelmää, joka koostuu palvelimesta, palvelimella olevista http- ja sovelluspalvelinohjelmistoista, WWW-sovelluksesta, WWW-sivuista, tiedonsiirtoväylästä, asiakkaan WWW-selaimesta sekä mahdollisista yhteyksistä ja rajapinnoista tietokantoihin ja muihin tietojärjestelmiin. WWW-sivusto taas koostuu useista WWW-sivuista, jotka voivat olla yhtä hyvin staattisia kuin ajon aikana dynaamisesti rakentuvia.

Tärkeimmät merkistöt tutkielmani kannalta ovat Unicode, ISO-8859-1 ja ASCII. ASCII on yksi vanhimpia standardoituja merkistöjä ja se on sisällytetty osaksi useimpia muita merkistöjä. ISO-8859-1 (eli Latin-1) on ASCII:n laajennus ja osa ISO-8859 –merkistöperhettä. ISO-8859 –merkistöperhe muodostuu 8-bittisistä merkistöistä, jotka kaikki sisältävät 7-bittisen ASCII:n samoilla merkkipaikoilla, mutta ASCII:n jälkeen tulevilla 128 merkkipaikoilla on joka merkistössä eri merkit. Suurin osa nykyisistä tietojärjestelmistä, standardeista ja ohjelmista tukee ASCII:ta tai ISO-8859-1 –merkistöä oletusmerkistöinään. ASCII ja ISO-8859-1 sisältävät kirjoitusmerkit vain länsieurooppalaiset kieliin. Paikalliset merkistöt on luotu kattamaan muiden kielten tarpeet. Melkein joka kielelle on syntynyt oma merkistö, jolla ei voida kirjoittaa muita kieliä. Unicode luotiin, jotta yhdellä merkistöllä pystyttäisiin kuvaamaan kaikki maailman kirjoitetut merkit. Unicodesta on olemassa erilaisia enkoodauksia, jotka määrittelevät muun muassa kuinka monta oktettia (eli 8-bittistä tavua) yhden Unicode-merkin koodaamiseen käytetään. Koska ASCII ja ISO-8859-1 ovat standardien ja ohjelmistojen oletusmerkistöjä, WWW-järjestelmissä suurimmat merkistöongelmat tulevat muiden merkistöjen kanssa.

Tutkielman yksi pääkysymyksistä on: millaisia merkistöongelmia voi olla monikielisissä WWW-järjestelmissä? Tähän kysymykseen vastasin luvussa

neljä. Merkistöongelmat näkyvät WWW-järjestelmän käyttäjälle pahimmillaan niin, että koko WWW-sivun teksti on muuttunut lukukelvottomaksi merkkisotkuksi tai tekstin tilalla näkyy pelkkiä tyhjiä neliöitä. Hieman lievemmissä tapauksissa vain osa merkeistä on korvautunut väärillä merkeillä, mutta teksti on silti luettavissa. Merkistöongelmien ratkaisemista haittaa se, että merkit voivat näkyä väärin pelkästään vain joissain tietyissä käyttöympäristöissä. Merkistöongelmat voivat johtua esimerkiksi puuttuvasta tai virheellisestä fontista, vääristä WWW-selaimen merkistö- ja enkoodausasetuksista, virheellisistä tai puuttuvista WWW-sivun merkistö- tai enkoodausmäärityksistä, tiedonsiirron tai tekstinsyötön aikana tekstin käsittelystä väärässä muodossa tai siitä, että merkistöön ei kuulu tekstissä tarvittavat merkit. Ongelmia tuottaa myös se, jos jokin järjestelmän osa (kuten palvelimen käyttöjärjestelmä) tai vaikka toteutukseen käytetty ohjelmointikieli ei tue valittua merkistöä. Tutkielma pyrki myös vastaamaan millaisia ratkaisuja näihin ongelmiin on olemassa ja kuinka näitä ongelmia voidaan ehkäistä monikielisen WWW-järjestelmän tuotantoprosessissa. Ongelmien ratkaisuja on esitelty luvussa 4 ja tuotantoprosessiin liittyviä asioita luvussa 5. Tutkielmassani olen valinnut monikielisen WWW-järjestelmän tuotantoprosessiksi Noackin ja Schienmannin geneeriseen elinkaarimalliin perustuvan prosessimallin, jossa on seuraavat vaiheet: vaatimusanalyysi, käyttötapausten arviointi, käyttöliittymän suunnittelu, skenaarioiden luonti, järjestelmäarkkitehtuurin suunnittelu ja luonti, sovellusarkkitehtuurin suunnittelu ja luonti, komponenttien ja järjestelmän toteutus, testaus, komponenttien ja järjestelmän julkaisu sekä sisällön ylläpito. Luettelen seuraavaksi lyhyesti tutkielmassani esiin tuodut merkistöongelmat, niiden ratkaisut ja miten ne tulee huomioida monikielisen WWW-järjestelmän tuotantoprosessissa.

Kun käyttäjän koneelta puuttuu fontti, WWW-sivuilla näkyy tyhjiä neliöitä oikeiden merkkien sijaan. Ongelma ratkeaa, kun oikea fontti asennetaan

käyttäjän koneelle. Jotkin selaimet (esimerkiksi Microsoftin Internet Explorer) ilmoittavat tällöin käyttäjälle fontin puuttumisesta ja pyytävät käyttäjää asentamaan fontin koneelleen ennen sivun näyttämistä. Koska kaikki selaimet eivät tätä tee eivätkä käyttäjät välttämättä tiedä, kuinka ongelman voi korjata, WWW-järjestelmää suunniteltaessa kannattaisikin valita mahdollisimman yleisesti käytössä oleva merkistö, jolle todennäköisesti löytyy fontti jokaisen käyttäjän koneelta. Merkistön tulisi kuitenkin sisältää merkit kaikille niille kielille, joilla järjestelmän halutaan toimivan. Kielet valitaan vaatimusanalyysivaiheessa, jossa määritellään myös sivuston lokalisoinnin ja kansainvälistämisen tarve. Vaatimusanalyysivaiheessa merkistön ja enkoodauksen valintaan tuleekin kiinnittää erityistä huomiota. WWW-sivuilla voidaan yleensä näyttää vain yhteen merkistöön kuuluvia merkkejä. Jos järjestelmään valitaan merkistö, joka ei sisälläkään kaikkia tarvittavia merkkejä, merkistö joudutaan usein vaihtamaan, sillä muuten merkkejä ei voida näyttää WWW-sivuilla. Poikkeuksia kuitenkin on. Muutamia vakiintuneita merkkejä voidaan toteutusvaiheessa lisätä WWW-sivuille entity-merkinnällä. HTML-standardin FRAME-tagilla voidaan yhdellä WWW-sivulla näyttää monta eri WWW-dokumenttia, joilla kaikilla voi olla määriteltynä eri merkistö ja enkoodaus. Kaikki selaimet eivät kuitenkaan tue näitä ratkaisuja, joten WWW-järjestelmään kannattaa aina valita mahdollisimman laaja ja yleinen merkistö kuten Unicode. Jos WWW-järjestelmän vaatimusanalyysissä päädytään johonkin harvinaisempaan merkistöön, voidaan näkyvyysongelmia ehkäistä käyttämällä dynaamisia fontteja tai sisällyttämällä teksti kuviin. Päätökset dynaamisten fonttien tai kuvien käytöstä tehdään käyttöliittymän suunnittelun aikana. Kaikki selaimet eivät kuitenkaan tue dynaamisten fonttien käyttöä eivätkä edes kuvia, joten nämä eivät ratkaise ongelmaa täydellisesti. Lisäksi jos teksti on sisällytetty kuviin, tekstien päivittäminen hankaloituu eivätkä hakukoneet löydä sanoja. Tällä ratkaisulla on siis vaikutusta myös sisällön ylläpitovaiheeseen.

Jos vain osa merkeistä on korvautunut väärillä merkeillä, WWW-sivua tulkitaan selaimessa yleensä väärällä merkistöllä. Ongelma korjaantuu, jos käyttäjä valitsee selaimeen oikean merkistön. Tämä saattaa kuitenkin olla monille käyttäjille liian tekninen tehtävä, joten WWW-sivuja luotaessa eli tuotantovaiheessa onkin erityisen tärkeää, että tekstin merkistö ja enkoodaus on merkitty oikein WWW-dokumentteihin. Tällöin HTTP-protokollan mukaan selainten tulisi näyttää WWW-sivut merkityn merkistön ja enkoodauksen mukaisesti ilman, että käyttäjä joutuu valitsemaan mitään. Jos merkintä puuttuu, WWW-sivu näytetään yleensä edellisen selatun sivun merkistöasetuksilla tai selaimen oletusmerkistöllä. Tällöin tekstin merkistöä saatetaan tulkita väärin. Useimmat selaimet antavat käyttäjälle mahdollisuuden vaihtaa merkistöä ja enkoodausta, jolla sivua näytetään. Selain ei kuitenkaan estä käyttäjää vaihtamasta oikeaa merkistöä vääräksi.

Käyttäjän mahdollisuus vaihtaa selaimen merkistöasetuksia saattaa muodostua ongelmaksi myös tekstin syötössä. Syöteteksti saattaa olla kirjoitettu eri merkistöllä kuin millä WWW-sovellus olettaa sen olevan. Tällöin WWW-sovellus saattaa esimerkiksi tallentaa tai lähettää tekstin väärässä muodossa, jolloin teksti korruptoituu lukukelvottomaksi. Tämän vuoksi jo vaatimusanalyysivaiheessa tulee määritellä voiko käyttäjä syöttää tekstiä järjestelmään ja viimeistään sovellusarkkitehtuurin luonnissa pitää päättää miten syötetekstin oikeellisuus tarkistetaan vai tarkistetaanko sitä lainkaan. Varsinkin järjestelmissä, joissa syöteteksti julkaistaan sellaisenaan sivustolle, syötetekstin merkistö kannattaa varmistaa piilotettua tarkistusmerkkijonoa hyväksi käyttäen.

Jos koko teksti (tai osa siitä) näkyy WWW-sivulla lukukelvottomana merkkisotkuna, niin ongelma voi olla siinä, että monioktettista merkistöä (kuten Unicodea) tulkitaan yksioktettisena merkistönä (kuten ISO-8859-1-merkistönä) tai päinvastoin. Jos tilanne ei korjaannu käyttäjän selaimen merkistöasetuksia vaihtamalla, ongelma on luultavimmin WWW-järjestelmän

sisäisessä merkistökäsittelyssä. Jokin WWW-järjestelmän osa saattaa käsitellä merkistöä esimerkiksi väärässä enkoodausmuodossa, jolloin teksti saattaa muuttua lukukelvottomaksi. Muun muassa Unicodesta on erilaisia enkoodauksia ja jotkin WWW-järjestelmän osista saattavat tukea eri enkoodausta kuin toiset. WWW-järjestelmän järjestelmä- ja sovellusarkkitehtuurisuunnittelussa tulisikin ottaa selvää, mitä merkistöjä ja enkoodauksia järjestelmän osat ja ohjelmistokomponentit tukevat. Jos kaikki osat eivät tue valittua merkistöä ja enkoodausta, ne tulisi vaihtaa sellaisiin, jotka tukevat. Jos vaihtoa ei voida tai haluta tehdä, voidaan toteutusvaiheessa järjestelmään lisätä konverttereita, jotka muuttavat tekstin toiseen merkistö- tai enkoodausmuotoon siksi aikaa, kun osa käsittelee tekstiä. Yleisin konversioiden käyttötilanne on tiedonsiirrossa. Tiedonsiirron ajaksi teksti muutetaan yleensä 7- tai 8-bittiseen muotoon. Muuttamisen aikana saattaa kuitenkin tapahtua jokin virhe ja teksti korruptoituu lukukelvottomaksi. Kun 8-bittisessä tiedonsiirtoväylässä käytetään tekstille 7-bittistä enkoodausta, tiedonsiirron onnistumista voidaan tarkistaa pariteetin avulla.

Näiden teknisten merkistöongelmien lisäksi monikielisissä WWW-järjestelmissä on usein paljon suunnitteluun liittyviä merkistöongelmia. Esimerkiksi käyttöliittymän suunnittelussa kannattaa huomioida, että eri kielissä samaa tarkoittavat sanat voivat olla hyvin eri mittaisia. Teksteille kannattaa siis varata kunnolla tilaa. Lisäksi kovin pientä fonttikokoa kannattaa välttää, sillä varsinkin kiinankieliset merkit vaativat länsimaalaisia merkkejä suuremman fonttikoon näkyäkseen kunnolla.

Viimeistään WWW-sovelluksen toteutusvaiheessa pitää huomioida valitun merkistön ja ohjelmointikielen yhteensopivuus. Joissain ohjelmointikielissä yli 8-bittisten merkkien käsittelyyn tarvitaan eri metodit kuin lyhempien merkkien käsittelyyn. Teksti muuttuu lukukelvottomaksi, jos sitä käsitellään väärillä metodeilla. Jos sovellusarkkitehtuurin suunnittelussa päätetään käyttää valmiskomponentteja tai useita eri ohjelmointikieliä WWW-sovelluksen tekoon,

niin on hyvä tarkistaa, että niillä kaikilla on tuki valitulle merkistölle ja enkoodaukselle. Samoin sovellusalustan ja käyttöjärjestelmän tuki kannattaa varmistaa.

Monikielisen WWW-järjestelmän testaaminen vaatii yleensä enemmän resursseja kuin yksikielisen. Vaikka järjestelmässä olisi käytössä vain yksi merkistö, joka kattaa kaikki kielet kuten Unicode, ei testauksessa kannata tyytyä vain yhden puhutun kielen testaamiseen. Unicodenkin kanssa merkistöongelmat tulevat parhaiten esille, kun järjestelmää testataan kielillä, joista osa kuuluu merkistö yläpäähän ja osa alapäähän. Lisäksi WWW-järjestelmää tulisi testata eri tuottajien WWW-selainten kanssa, jotta esimerkiksi fontin tai merkistön tuen puute huomattaisiin. Monikielisen WWW-järjestelmän tuotanto- ja julkaisuaikataulussa kannattaa ottaa testauksen lisäksi huomioon myös sisällöntuotannon hitaus. Tekstin kääntäminen vie jo itsessään paljon aikaa ja sisällön syöttöön tarkoitetut työkalut eivät välttämättä tuekaan kaikkia tarvittavia merkistöjä niin hyvin kuin pitäisi. Myös sisällöntuotantotyökalut tulisi siis testata mahdollisimman varhaisessa vaiheessa.

Vaikka kaikkia merkistöihin liittyviä ongelmia ei voidakaan ratkaista WWW-järjestelmää luotaessa, hyvällä suunnittelulla ja tarkalla toteutuksella voidaan kuitenkin ehkäistä suurin osa merkistöongelmista. Vastaavanlaisia merkistöihin liittyviä ongelmia on ollut havaittavissa myös muissa kuin WWW-käyttöliittymillä varustetuissa järjestelmissä. Tutkielmassani rajauduin kuitenkin käsittelemään päätelaitteista vain WWW-selaimia, sillä muiden päätelaitteiden erityisominaisuudet toivat mukanaan paljon uusia merkistöongelmia, joita WWW-selaimissa ei ole. Esimerkiksi PDA-laitteiden muistin vähyys aiheuttaa rajoituksia myös laitteiden fonttivalikoimaan ja fonttien toteutuksen kattavuuteen. Toisaalta käyttäjä ei myöskään yleensä voi tallentaa niihin uusia fontteja. Jatkotutkimusaiheena voisi siis olla muiden kuin WWW-järjestelmien tai vaikka vain eri käyttöliittymien (esimerkiksi mobiilikäyttöliittymien) merkistöihin liittyvät ongelmat.

LÄHDELUETTELO

Adams G., 1993. Internationalization and Character Set Standards. StandardView 1993(1), 31-39.

Bitstream Dynamic Fonts: Software Requirements, 2002. [online] Massachusetts: Bitstream. [Viitattu 24.4.2002] Saatavilla www-muodossa <URL: http://www.truedoc.com/webpages/intro/software.html>

Character Sets, 2002. [online] IANA. [Viitattu 31.3.2002] Saatavilla www-muodossa <URL: http://www.iana.org/assignments/character-sets>

Client-Side JavaScript Reference v1.3., 1999. [online] California: Sun-Netscape Alliance. [Viitattu 28.4.2002] Saatavilla www-muodossa <URL: http://developer.netscape.com/docs/manuals/js/client/jsref/index.htm >

Communicator 4: Basic Setup Information for International Users, 2000. [online] California: Netscape. [Viitattu 31.3.2002] Saatavilla www-muodossa <URL: http://home.netscape.com/eng/intl/basics.html>

Czyborra R., 1998. The ISO 8859 Alphabet Soup. [online] [Viitattu 9.11.2002] Saatavilla www-muodossa <URL: http://czyborra.com/charsets/iso8859.html>

Dynamic Font Support, 2002. [online] California: Netscape. [Viitattu 24.4.2002] Saatavilla www-muodossa <URL: http://home.netscape.com/communicator/v4.0/using/dynamicfonts.html>

Essenlink B., 2000. A Practical Guide to Localization (Language International World Directory). Amsterdam: John Benjamins Publishing Co.

Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P. & Berners-Lee T., 1999. Hypertext Transfer Protocol – HTTP/1.1. [online] The Internet Engineering Task Force (IETF), Network Working Group. [Viitattu 24.11.2002] Saatavilla www-muodossa <URL: http://www.ietf.org/rfc/rfc2616.txt >

Freed N. & Borenstein N., 1996. Multipurpose Internet Mail Extensions (MIME) Part Two: Mediatypes. [online] The Internet Engineering Task Force (IETF), Network Working Group. [Viitattu 24.11.2002] Saatavilla www-muodossa <URL: http://www.ietf.org/rfc/rfc2046.txt>

Goldsmith D. & Davis M., 1997. UTF-7, A Mail-Safe Transformation Format of Unicode. [online] The Internet Engineering Task Force (IETF), Network Working Group [Viitattu 27.11.2002] Saatavilla www-muodossa <URL: http://www.ietf.org/rfc/rfc2152.txt>

Graham T., 2000. Unicode: A Primer. United States of America: M&T Books

Green D., 1998. Unicode Font Support. [online] [Viitattu 28.4.2002] Saatavilla www-muodossa <URL: http://java.sun.com/products/jdk/1.2/docs/guide/internat/unicode_font.doc.html>

Goto K. & Cotler E., 2001.Web Redesign: Workflow that works. Indianapolis: New Riders Publishing.

Haible B., 2001. The Unicode HOWTO. [online] [Viitattu 22.1.2003] Saatavilla www-muodossa <URL: http://www.tldp.org/HOWTO/Unicode-HOWTO-6.html>

Haikala I. & Märijärvi J., 1997. Ohjelmistotuotanto. Espoo: Suomen ATK-kustannus Oy.

He S., 2001. Interplay of language and culture in global E-commerce: a comparison of five companies' multilingual websites. Teoksessa Proceedings of the 19th annual international conference on Computer documentation. Santa Fe, New Mexico, USA, October 2001. ACM. s. 83-88.

IBM, 2003. International Components for Unicode. [online] [Viitattu 16.4.2003] Saatavilla www-muodossa <URL: http://oss.software.ibm.com/icu/ >

ISO International Organization for Standardization, 1994. ISO/IEC 2022:1994 Information technology – Character code structure and extension techniques.

ISO International Organization for Standardization, 2002. Catalogue searched for standards. [online] [Viitattu 9.11.2002] Saatavilla www-muodossa <URL: http://www.iso.org/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=8859>

Jaakohuhta H., 2001. IT Ensyklopedia. Helsinki: IT Press.

Järvinen P., 1996. Tietotekniikan termit, PC-tietosanakirja, versio 2.0. Juva: WSOY.

Klensin J. (toim.), 2001. Simple Mail Transfer Protocol. [online] The Internet Engineering Task Force (IETF), Network Working Group [Viitattu 24.11.2002] Saatavilla www-muodossa <URL: http://www.ietf.org/rfc/rfc2821.txt >

Kubota T., 2002. Introduction to i18n. [online] [Viitattu 9.11.2002] Saatavilla www-muodossa <URL: http://www.debian.org/doc/manuals/intro-i18n/>

Kuhn M., 2003. UTF-8 and Unicode FAQ for Unix/Linux. [online] [Viitattu 16.4.2003] Saatavilla www-muodossa <URL: http://www.cl.cam.ac.uk/~mgk25/unicode.html>

Ksar M., 2002. ISO/IEC 10646 & The Unicode Standard. Teoksessa Proceedings on 21st International Unicode Conference. Dublin, Ireland, May 2002. Unicode Consortium.

Kaplan M. & Wissink C., 2002. The Microsoft Layer for Unicode on Windows 95/98/Me Systems. Teoksessa Proceedings on 22nd International Unicode Conference. San Jose, California, September 2002.Unicode Consortium. Saatavilla myös www-muodossa <URL: http://www.microsoft.com/globaldev/Presentations/unicode22/382.doc >

Li J., Chen J. & Chen P., 2000. Modeling Web Application Architecture with UML. Teoksessa Proceedings of 36th Conference on Technology of Object Oriented Languages and Systems 2000. 265-274. Xi'an, China. November 2000. IEEE.

Morgan T., Luttrell C. & Liu Y., 2001. Designing Multilingual Web Sites: Applied Authoring Techniques. Annual ACM Conference on Systems Documentation. October 21-24, 2001. Santa Fe, New Mexico, USA. ACM 230-231.

Mäkinen S., 2002. Kansainvälinen WWW-sovellus. Espoo-Vantaan teknillinen ammattikorkeakoulu, Viestintätekniikan koulutusohjelma, Insinöörityö.

Noack J. & Schienmann B., 1998. Designing an Application Development Model for a Large Banking Organization. Teoksessa Proceedings of the 20th International Conference on Software Engineering 1998. 495-498. Kyoto, Japan. ACM.

Nua Internet Surveys, 2003. How Many Online? [online] [Viitattu 15.1.2003] Saatavilla www-muodossa <URL: http://www.nua.ie/surveys/how_many_online/>

Raggett D., Le Hors A. & Jacobs I. (toim.), 1999. HTML 4.01 Specification. [online]. United States of America: W3C. [Viitattu 31.3.2002] Saatavilla www-muodossa <URL: http://www.w3.org/TR/1999/REC-html401-19991224/cover.html#minitoc>

Shabalin R., 2002. Testing software for World-Readiness. [online] [Viitattu 19.11.2002] Saatavilla www-muodossa <URL: http://www.microsoft.com/globaldev/articles/wrtesting.asp>

Sun Microsystems, 1999. Supported Encodings. [online] [Viitattu 28.4.2002] Saatavilla www-muodossa <URL: http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html>

Sun Microsystems, 2003. Whitepapers: Unicode support in the Solaris Operating Environment. [online] [Viitattu 14.4.2003] Saatavilla www-muodossa <URL: http://wwws.sun.com/software/whitepapers/wp-unicode/#1015102 >

The Unicode Consortium (toim.), 2000. The Unicode Standard, Version 3.0. Massachusetts: Addison-Wesley.

Turnbull S., 1999. Alphabet Soup. Linux Journal 59 (2).

LIITE 1

Haastattelu sähköpostitse 5.5.2003

Hei!

Pro gradu -tutkielmani aiheena on merkistöihin liittyvät ongelmat monikielisissä WWW-järjestelmissä ja niiden mahdolliset ratkaisut (Character set related problems in WWW-systems). Tutkielmassani käsittelen vain WWW-sivustoja ja niiden alla toimivia sovelluksia. Ulkopuolelle jäävät esimerkiksi SMS ja WAP. Merkistöistä mukana ovat Unicode (ja sen kaikki muodot), ASCII, ISO-8859 -merkistöperhe, GB2312, Big5 ja Shift-JIS. Muidenkin merkistöjen ongelmia voi kertoa vastauksissa. Tällä haastattelulla haen tukea viimeiseen sisältökappaleeseen, jossa käsittelen WWW-järjestelmän tuotantoprosessia ja kuinka merkistöongelmat otetaan siinä prosessissa huomioon.

Toivoisin vastauksia alla oleviin kysymyksiin 12.5. mennessä. Kysymyksiä on vain kaksi, mutta ne ovat todella laajoja. Voitte vastata useammassa erässä, jos siltä tuntuu. Lyhyistäkin vastauksista on apua, pitkistä tietysti sitäkin enemmän. Vastata voi suomeksi tai englanniksi. Tutkielmani on suomenkielinen.

Kysymykset/Questions:

1. Millaisia merkistöihin liittyviä ongelmia olette havainneet WWW-järjestelmissä? / What kind of a character set related problems have you noticed in WWW-systems?

2. Mitä pitää ottaa merkistöjen kannalta huomioon monikielisen WWW-järjestelmän tuotantoprosessin eri vaiheissa, jotta merkistöongelmia ei esiintyisi? (WWW-järjestelmän tuotantoprosessin vaiheita ovat: vaatimusanalyysi, käyttötapausten arviointi, käyttöliittymän suunnittelu,

skenaarioiden luonti, järjestelmäarkkitehtuurin luonti, sovellusarkkitehtuurin luonti, komponenttien ja järjestelmän toteutus, testaus, komponenttien ja järjestelmän julkaisu, ylläpito.) / What should be taken into account in WWW system creation process, so that there would not be character set related problems? (Phases of WWW-creation process are: Requirement Analysis, Use Case Analysis, User Interface Design, Modelling of Scenarios, System Architecture Creation, Component Architecture Creation, Implementing Components/System, Testing, Roll-out Components/System, Maintenance.)

Kiitos jo etukäteen!

- Nina