Lukijalle

Tähän julkaisuun on kerätty syyslukukaudella 2002 pitämälläni Tutkimuskurssilla tehdyt, määräaikaan mennessä valmistuneet tutkielmat.

Toimittaja

Sisällysluettelo

Tekoälymenetelmiä robottien jalkapallossa

Lassi Autio

Tiivistelmä

Tutkielmassa tarkastellaan haun ja oppimisen käyttöä robottien jalkapallossa tekoälyn näkökulmasta. Hakua tutkitaan robottien liikkumisessa erityisesti agenttikeskeisen hakumenetelmän kannalta. Oppimista tutkitaan muistin ja neuroverkkojen avulla oppimista ja opitun tiedon käyttöä.

Avainsanat ja -sanonnat: RoboCup, tekoäly, oppiminen, haku, neuroverkko.
CR-luokat: I.2.9

1. Johdanto

RoboCup-kilpailu on robottien jalkapallon maailmanmestaruuskilpailut, jotka järjestettiin ensimmäisen kerran 1997 Nagoyassa Japanissa ja on sen jälkeen järjestetty joka vuosi, viimeksi 2002 Fukuokassa Japanissa. RoboCup-kilpailun ovat panneet alulle kansainvälinen tutkimus ja opetus. RoboCupin tarkoituksena on edistää tekoälyn ja älykkäiden robottien tutkimista antamalla näille aloille oivat tutkimusmahdollisuudet jalkapallon muodossa.

Monet teknologiat tulevat esiin RoboCup-kilpailussa ja sitä kautta robottien jalkapallossa: itsenäisten agenttien suunnittelu, agenttien välinen yhteistyö, strategian kehittäminen, reaaliaikainen päättely, robotiikka ja ympäristön havainnointi. Suurin osa näistä vaatii robotilta tekoälyä, jotta se osaa eri tilanteissa toimia oikein. Näiden lisäksi robotteihin tulevat sovellukset antavat myös tutkimuksia sovellusalalle. RoboCup on tehtävä joukkueelle, jossa on nopeasti liikkuvia robotteja dynaamisessa ympäristössä.

RoboCup-kilpailu koostuu erilaisista sarjoista: simulaatio-liiga, pienten robottien liiga, keskikokoisten robottien liiga, Sonyn jalallisten robottien liiga ja humanoidirobottien liiga. Tässä tutkielmassa käsiteltävät asiat pätevät kaikkiin näihin sarjoihin, mutta etenkin simulaatio-liigaan sekä pienten ja keskikokoisten robottien liigaan.

Simulaatio-liigassa robotit pelaavat virtuaalisesti toisiaan vastaan käyttäen Soccer Serveriä. Se on simulaattori, jossa kaksi 11 pelaajan joukkuetta pelaavat vastakkain.

Pienten robottien liigassa molemmilla joukkueilla on viisi robottia, joiden koolla on tiettyjä rajoituksia: robotin tulee mahtua halkaisijaltaan 18 cm:n ympyrään ja olla korkeintaan 15 cm:ä korkea. Robotit pelaavat 2,8 m pitkällä ja 2,3 m leveällä kentällä oranssilla golf-pallolla.

Keskikokoisten robottien liigassa robotin tulee mahtua 50 cm x 50 cm neliöön, olla 30-80 cm korkea ja painaa korkeintaan 80 kg. Pelikentän ja pallon koko ovat FIFA:n virallisten sääntöjen mukaiset: kentän leveys 64-75 m ja pituus 100-110 m ja pallo on kokoa 5. Robottien pallonkäsittelyä on rajoitettu säännöissä (esim. robotti ei saa kantaa palloa) ja lisäksi on rajoituksia koskien muiden pelaajien ja kentän vahingoittamista.

Sonyn jalallisten robottien liigassa joukkueet koostuvat neljästä Sonyn Aibo-robottikoirasta. Nimestäkin päätellen tämän sarjan sponsorina toimii Sony.

Humanoidirobottien liigassa pelataan humanoidiroboteilla. Näiden robottien tulee täyttää pari vaatimusta: niiden tulee pystyä kävelemään kahdella jalalla ilman renkaiden tai telaketjujen apua ja niiden tulee koostua kahdesta jalasta, kahdesta kädestä, vartalosta ja päästä, kuten ihminenkin. Humanoidiliigassa on monta eri sarjaa, joista moni ei edes liity jalkapalloon: yhdellä jalalla seisominen, käveleminen määrättyä reittiä pitkin, pallon potkaiseminen tyhjään maaliin, rangaistuspotkukilpailu ja jalkapallopeli, jossa on 1-3 pelaajaa. RoboCup-kilpailun eräs tavoite on, että vuonna 2050 humanoidirobotit voittaisivat ihmisten jalkapallon maailmanmestarit.

Tässä tutkielmassa tarkastelen liikkumista, hakua ja oppimista robottien jalkapallossa tekoälyn näkökulmasta. Luvussa 2 tutkin robottien liikkumista erilaisissa tilanteissa. Luvussa 3 tarkastelen robottien liikkumista agenttikeskeisen haun näkökulmasta. Luvuissa 4 ja 5 käsittelen kahta erilaista tapaa joilla robotit voivat oppia: muistin ja neuroverkkojen avulla oppimista.

2. Robottien liikkuminen

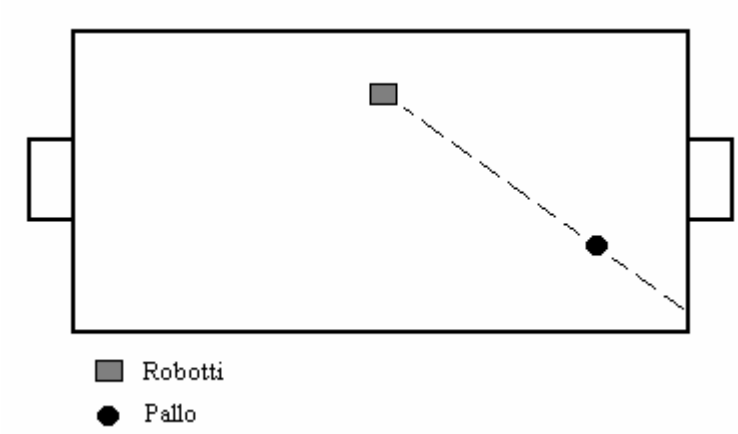
Jotta robotit voisivat pelata jalkapalloa älykkäästi ilman ihmisen apua, on niiden osattava liikkua itsenäisesti kentällä riippuen tilanteesta. Pienten ja keskikokoisten robottien liigassa robotit liikkuvat yleensä renkaiden tai telaketjujen avulla. Humanoidi- ja Sonyn jalallisten robottien liigassa robotit liikkuvat kahden ja neljän jalan avulla.

Liikkuminen on ehkä tärkein osa robottien jalkapallossa, johon liittyy myös tekoäly. Ilman liikkumista robotit eivät pysty tekemään kentällä mitään. Erityisesti jalallisilla roboteilla (humanoidirobotit ja Aibot) liikkumiseen

liittyy myös paljon teknistä osaamista. Keskityn tässä kuitenkin liikkumiseen tekoälyn näkökulmasta.

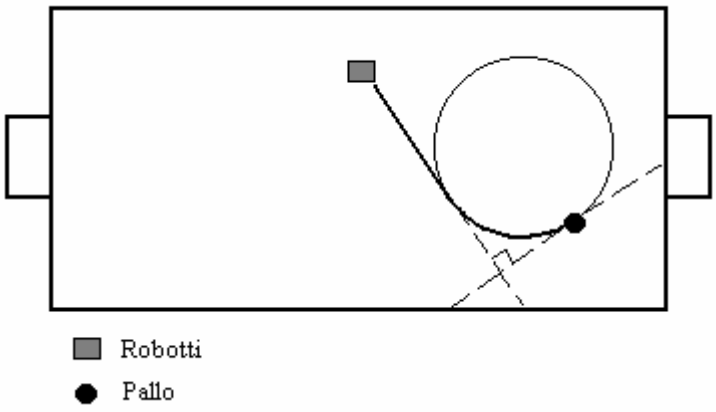
2.1. Liikkuminen palloa kohti ja potkaiseminen maaliin

Ennen kuin robotti voi liikkua minnekään, sen on suunniteltava liikkeensä. Tämä tarkoittaa sitä, että robotti esimerkiksi katsoo ympärilleen ja päättää pisteen, jota kohti mennä. Aina kuitenkaan robotti ei voi mennä vain suoraan valitsemaansa kohdetta kohti. Esimerkiksi kun robotti aikoo potkaista pallon maaliin, niin se ei voi läheskään aina valita lyhintä mahdollista reittiä palloon ja potkaista sitä suoraan. Tällöin pallo menee harvoin edes kohti maalia, kuten näemme kuvasta 1.



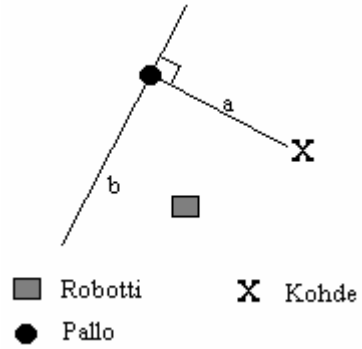
Kuva 1. Robotin liikkuminen suoraan palloa kohti.

Robotin tulisi osata mennä palloa kohti oikeassa kulmassa, jotta se voisi myös potkaista sen kohti maalia tai syöttää sen toiselle pelaajalle. Yksi esimerkki oikean kulman löytämiseksi on kuvan 2 tapa, jossa robotti pääsee nopeinta mahdollista vauhtia pallon luokse oikeassa kulmassa. Pallon tulisi olla robotista kohteeseen vedetyllä suoralla, jotta pallo menisi potkaistessa haluttuun kohteeseen. Pallon täytyy lisäksi olla tällöin paikallaan.



Kuva 2. Robotin liikkuminen palloa kohti oikeassa kulmassa [Tu, 2002].

Aina robotin ja pallon paikka ei ole näin hyvä potkaisukohteeseen nähden. Robotti laskee suoran pallon ja kohteen välille (suora a kuvassa 3) ja tähän suoraan nähden normaalin pallon kohdalla (suora b kuvassa 3). Jotta robotti pystyisi potkaisemaan tai liikuttamaan pallon kohteeseen, on robotin oltava eri puolella normaalia (suora b) kuin kohde. Mikäli robotti on samalla puolella kuin kohde, on sen ensiksi liikuttava eri puolelle ja vasta sen jälkeen se voi yrittää liikkua palloa kohti esimerkiksi em. tavalla. [Veloso *et al.*, 1997]



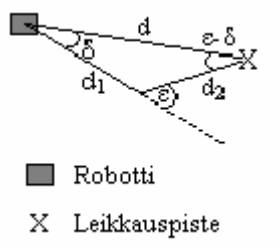
Kuva 3. Robotin ja kohteen sijainti palloon nähden [Veloso *et al.*, 1997].

2.2. Liikkuminen kohdetta kohti

Jos robotin tavoitteena on mennä johonkin valitsemaansa kohteeseen, niin sen on ensiksi käännyttävä kohdetta kohti ja tämän jälkeen suunnattava suoraan kohti tätä kohdetta. Kohde voi olla esimerkiksi paikallaan oleva pallo. Mikäli kohteen sijainti pystytään laskemaan tarkasti, niin robotin on vain käännyt- tävä • astetta ja kuljettava tämän jälkeen suoraan kohteeseen. Yleisesti robotti voisi noudattaa ohjetta: jos kohde on suoraan edessäsi, niin jatka matkaasi, muuten korjaa suuntasi. [Stolzenburg *et al.*, 2002]

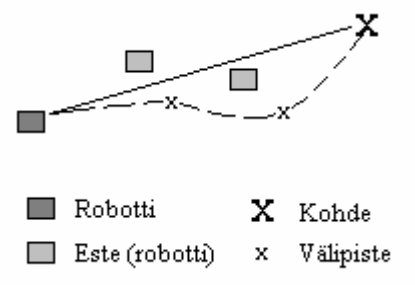
Mikäli robotin katsomassa suunnassa ja robotin suorittamissa toimenpiteissä ei ole ollenkaan epätäsmällisyyttä, niin tämä sääntö antaa parhaan mahdollisen strategian. Todellisuudessa suuntaa ei voida määritellä tarkasti, sillä robotin sensori ja arviointi ovat yleensä vähän virheellisiä. Kyseinen sääntö johtaisi siihen, että robotti muuttaisi vain koko ajan suuntaa liikkumatta ollenkaan eteenpäin.

Muutamme sääntöä siten, että robotti muuttaa suuntaansa vain mikäli robotin ja leikkauspisteen välinen kulma ylittää tietyn ennalta määrätyn kynnyksen • astetta (kuva 5). Tällöin robotin kulkema matka d1 + d2 on selvästi pitempi kuin d, joka olisi matka jos korjaus tehtäisiin arvolla • = 0˚. Vaikka matka on pitempi, robotti saavuttaa leikkauspisteen nopeammin, sillä se ei käänny liian usein. [Stolzenburg *et al.*, 2002]



Kuva 4. Suunnan korjaaminen [Stolzenburg *et al.*, 2002].

Robotin liikkuminen kentällä ei yleensä onnistu näin helposti, sillä robotti ei ole kentällä yksin, vaan kentällä on usein muita robotteja tiellä. Robotti ei siis aina pysty liikkumaan lyhintä mahdollista reittiä kohdetta kohti. Robotin on osattava väistää eteensä tulevia esteitä sujuvasti, jotka ovat usein vastustajia tai jopa omia pelaajia. Robotti ei pysty välttämättä tietämään etukäteen kuinka monta estettä sitä tulee vastaan ja missä, joten robotin tulee koko ajan tarkkailla ympäristöään. Kun robotti liikkuessaan huomaa edessään esteen, se laskee välipisteen, jonka kautta se kulkee kohdetta kohti. Robotti toistaa tämän toimenpiteen nähdessään esteen suunnittelemallaan reitillä [Veloso *et al.*, 1997]. Välipisteitä kohti robotti voi mennä esimerkiksi suunnan korjaamisella, joka on aiemmin osoittautunut käyttökelpoiseksi menetelmäksi kohteeseen liikkumiseen.

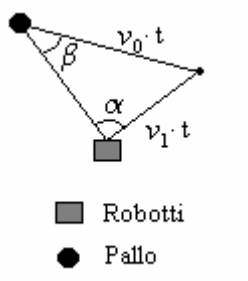


Kuva 5. Esteiden väistäminen.

2.3. Liikkuvan pallon kaappaaminen

Yleensä robotille ei tule tilannetta, jossa se pääsisi potkaisemaan palloa, joka on paikallaan. Siksipä sen täytyy osata potkaista myös palloa, joka on liikkeessä. Tällöin robotti laskee robotin ja kohteen välisen suoran ja pallon kulkuradan leikkauspisteen ja menee tästä leikkauspisteestä etäisyyden d päähän. Robotti arvioi ajan t, joka kestää, kun se liikkuu etäisyyden d. Se laskee pallon nopeuden perusteella pisteen, missä pallo on ajan t etäisyydellä leikkauspisteestä. Kun pallo on tässä pisteessä, niin robotti liikkuu kohti leikkauspistettä parasta mahdollista vauhtia, mikäli aikomuksena on potkaista pallo. Mikäli tarkoituksena on ottaa pallo hallintaan, niin liikutaan koko ajan nopeudella, jolla pallo ja robotti leikkaavat yhtä aikaa leikkauspisteen. Mikäli robotin laskut menevät oikein, niin robotti osuu oikeaan aikaan palloon ja saa sen menemään kohdetta kohti, joka voisi olla esimerkiksi maali tai toinen pelaaja. Toisena vaihtoehtona on, että robotti ottaa tällä tavalla pallon haltuunsa. [Stolzenburg *et al.*, 2002]

Yksinkertaisin tapa siepata pallo on mennä suoraan kohti palloa. Yleensä pallo on kuitenkin liikkeessä, jolloin tätä menetelmää ei voida käyttää. On parempi mennä suoraan pallon arvioidun kulkureitin ja oman kulkureitin leikkauspistettä kohti. Robotin on osattava mennä leikkauspistettä kohti oikealla nopeudella, jotta se saa siepattua pallon, kun se menee leikkauspisteen ohi.



Kuva 6. Oikean nopeuden arvioiminen. [Stolzenburg *et al.*, 2002].

Määritellään kaava jonka avulla robotti osaa leikata leikkauspisteen oikeaan aikaan ja onnistuu täten ottamaan pallon hallintaan. Pallo liikkuu nopeudella v_0 leikkauspistettä kohti, johon robotti pääsee nopeudella v_1 ja kulma • on pallon suunnan ja pallon ja robotin välisten suorien välinen kulma. Sini-säännöstä pystymme johtamaan kaavan, jolla pystymme laskemaan kulman • (kuva 7). Nyt robotin tarvitsee vain valita oma nopeutensa v_1 ja arvioitava nopeus v_0 ja kulma •, niin se pääsee leikkauspisteeseen oikeaan aikaan käyttämällä kulmaa •.

$$\sin\alpha = \frac{v_0 \cdot \sin\beta}{v_1}$$

Kuva 7. •:n ratkaiseminen sini-säännöllä.

3. Agenttikeskeiset hakumenetelmät liikkumisessa

Virheellisesti voisi luulla, että robottien jalkapallossa ei tarvita ollenkaan hakumenetelmiä ja että kyse on vain siitä, kuinka robotteihin asennetut sensorit pystyvät havaitsemaan pelikenttää. Todellisuudessa robottien jalkapallossa robottien täytyy tehdä erilaisia hakuja. Esimerkiksi pallon luokse meneminen on eräänlainen haku.

Robottien jalkapallossa robotti tuntee ympäristönsä, eli kentän. Robotti pääsee paikkaan kuin paikkaan, mikäli toista robottia ei ole edessä. Robotti pystyy aina palaamaan edelliseen paikkaansa, joten se pystyy liikkumaan vapaasti kentällä, eikä joudu tutkimaan ympäristöään kuten sokkelossa kulkeva robotti, joka ei tiedä ympäristönsä muotoa. Robottien jalkapallossa ympäristö on kuitenkin staattinen vain kentän suhteen. Kentällä on myös pallo ja muitakin robotteja jotka liikkuvat kentällä mikä tekee robottien jalkapallon ympäristöstä dynaamisen. Robottien jalkapallossa olevat haut suuntautuvat usein palloon, joka on lähes koko ajan liikkeessä. Ympäristön dynaamisuus tekee robottien jalkapallossa tehtävät haut erilaisiksi verrattuna tavallisiin hakuihin.

3.1. Agenttikeskeinen hakumenetelmä

Perinteiset hakumenetelmät suunnittelevat aluksi kokonaan hakunsa ja toteuttavan sen saman tien. Agenttikeskeiset hakumenetelmät rajoittavat suunnittelun tiettyyn osaan ympäristöstä ja tekevät suunnitelman hakunsa alulle. Agentti siirtyy suunnitellun matkan ja aloittaa taas alusta. Näitä

toimenpiteitä (suunnittelu, toteutus) toistamalla aina uudella paikalla agentti pääsee lopulta maaliin. [Koenig, 2001]

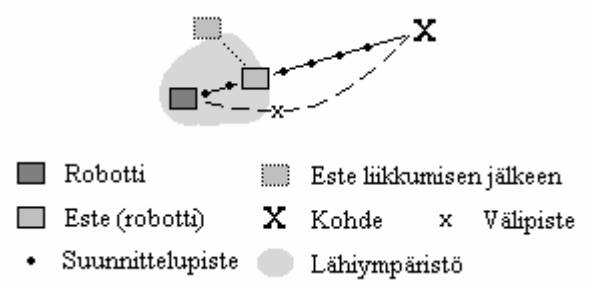
Jakamalla suunnittelun ja toteutuksen pieniin osiin agenttikeskeiset haut pystyvät supistamaan suunnittelun ja toteutuksen kuluja ja ovat yleensä nopeampia kuin tavalliset hakumenetelmät [Koenig, 2001]. Agenttikeskeiset hakumenetelmät ovat hyödyllisiä tapauksissa, joissa hakuavaruus on erittäin suuri, että koko hakuavaruuden läpi käyminen olisi tehotonta tai jopa mahdotonta. Esimerkiksi shakissa ensimmäisen siirron täydellinen tutkiminen olisi kertaluokkaa $O(a^n)$, missä $a > 1$ ja n on siirtovuorojen määrä. Tämä veisi käytännössä aivan liikaa aikaa. Mutta jos tutkitaan esimerkiksi vain viittä ensimmäistä siirtoa, niin haku olisi kertaluokkaa $O(a^{10})$, mikä veisi paljon vähemmän aikaa.

3.2. Agenttikeskeinen haku robottien jalkapallossa

Agenttikeskeiset hakumenetelmät sopivat hyvin robottien jalkapalloon, sillä agenttikeskeisillä hakumenetelmillä pystytään toimimaan hyvin dynaamisessa ympäristössä. Dynaamisessa ympäristössä toimiminen onkin agenttikeskeisten hakumenetelmien ominaisuus, joka tekee siitä käyttökelpoisen robottien jalkapalloon.

Robottien jalkapallossa kentällä liikkuu yhtä aikaa monta robottia, joista yleensä ainakin yksi liikkuu palloa kohti. Monen robotin liikkuminen kentällä muuttaa robottien ympäristöä koko ajan. Erityisesti pallon lähellä tapahtuu paljon. Kun robotti liikkuu johonkin kohteeseen, on hyvin todennäköistä, että sen reitillä on yksi tai useampi robotti tai reitille tulee yksi tai useampi robotti. Agenttikeskeisellä hakumenetelmällä robotti pystyy liikkumaan sujuvasti väistäen muita robotteja.

Kuvassa 8 oleva robotti käyttää liikkumiseensa agenttikeskeistä hakumenetelmää. Siinä robotti suunnittelee pienen matkan kohdettaan kohti kerralla. Osasuunnittelupisteet ja lähiympäristö voisivat olla esimerkiksi kuvan 8 mukaisia. Mikäli robotti huomaa suunnitellessaan seuraavaa matkaansa robotin edessään, niin se suunnittelee reittinsä kulkevan välipisteen kautta. Robotti osaa jatkaa matkaansa, vaikka sen eteen tulisi kuinka monta estettä tahansa, ja lopulta robotti saavuttaa kohteensa. Samantapaista menetelmää käytimme jo aiemmin kuvassa 5.



Kuva 8. Esteen liikkeen arviointi.

Kun robotille lisätään ominaisuus laskea matkallaan näkemiensä robottien kulkusuunta ja nopeus, niin se pystyy suunnittelemaan hakunsa vielä paremmin. Esimerkiksi mikäli robotti ei kuvan 8 tapauksessa huomioisi edessään olevan robotin liikkumista, niin se alkaisi mennä välipistettä x kohti. Tämä vastaisi aiemman kuvan 5 tapaa väistää esteitä. Mutta kun robotti osaa huomioida toisen robotin liikkumisen, niin robotti saavuttaa nopeammin kohteensa, sillä se ei tee turhaa mutkaa matkalla. Arvioimalla toisten robottien kulkusuunnan ja nopeuden robotti pystyy arvioimaan seuraavia asioita: ehtiikö edessä oleva robotti pois tieltä, tuleeko jossakin oleva robotti tielle ja tarvitseeko jotakin robottia huomioida ollenkaan. Näiden asioiden huomioiminen auttaa robottia hauissa. Agenttikeskeisillä hakumenetelmillä robotit pystyvät näkemään itselleen oleelliset asiat lähiympäristöstä ja tietävät täten mitä heidän lähiympäristössään tapahtuu.

4. Oppiminen muistin avulla

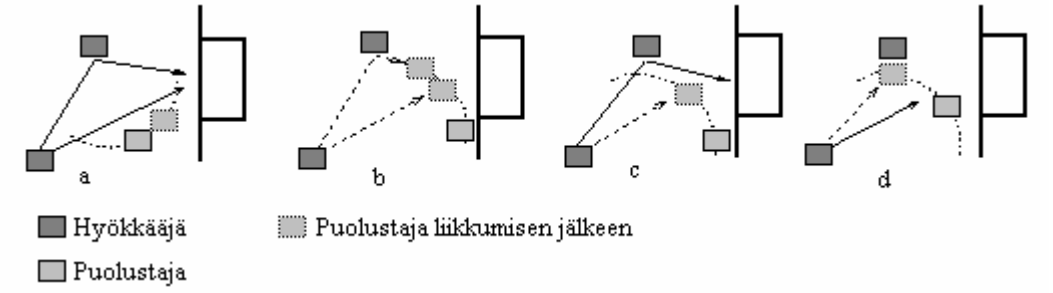
Yksi tärkeimmistä ja mielenkiintoisimmista tekoälyn alueista robottien jalkapallossa on oppiminen. Robottien tulee pystyä oppia virheistä ja onnistumisista, jotta ne osaisivat seuraavalla kerralla toimia oikein. Jatkuva oppiminen mahdollistaa myös sen, ettei robottien taso jää polkemaan paikalleen, vaan ne kehittyvät koko ajan. Robotit voivat oppia peleissä sekä harjoitella tilanteita, kuten ihmisetkin. Mitä enemmän robotti on pelannut, sitä enemmän se on yleensä oppinut pelistä. Tässä luvussa selitän erään tilanteen miten robotit oppivat pelaamaan tallentamalla harjoiteltuja asioita muistiin ja käyttämällä opittuja asioita.

4.1. Syöttää vai laukaista?

Otetaan esimerkkinä Stonen ja Veloson [1995] tilanne, jossa on kaksi hyökkääjää ja yksi puolustaja, joka on maalivahti. Puolustaja liikkuu pientä ympyrää maalin edessä jollakin nopeudella. Molemmat hyökkääjät ovat koko ajan samoilla paikoilla. Kyseessä on siis urheilutermein "kaksi vastaan nolla" -

tilanne. Pallo on toisella hyökkääjällä, jonka on tehtävä tärkeä päätös: syöttääkö joukkuetoverilleen vai laukaistako suoraan maaliin?

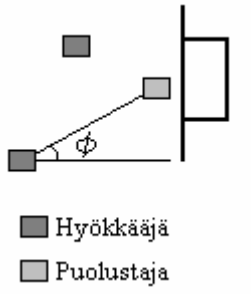
Ensimmäisellä kerralla hyökkääjä ei tiedä mitä tehdä, sillä sillä ei ole kokemusta ko. tilanteesta. Ensimmäisellä kerralla hyökkääjän on siis valittava satunnaisesti syötön ja laukaisemisen väliltä [Stone and Veloso, 1995]. Toinen vaihtoehto on keksiä jokin arviointifunktio, jonka perusteella hyökkääjä tekisi päätöksensä. Arviointifunktio voisi olla esimerkiksi etäisyys maaliin. Muutaman kerran jälkeen robotti kuitenkin oppii toimimaan älykkäästi samassa tilanteessa. Tilanne on sama myös ihmisillä.



Kuva 9. Voi syöttää (a, c), voi laukaista (a, d) ja ei voi tehdä kumpaakaan (b) [Stone and Veloso, 1995].

4.2. Muisti

Hyökkääjän on opittava kaksi funktiota tehdäkseen maali maalivahdin kulman (•) perusteella (kuva 10). Hyökkääjä opettelee funktion, joka antaa todennäköisyyden, jolla hyökkääjän laukaistessa ($P_s^*(\bullet)$) tai syöttäessä ($P_p^*(\bullet)$) pallo menee maaliin (1) tai ei mene (-1). Yleensä robotilla ei kuitenkaan ole kokemusta kaikista mahdollisista kulmista •. Sitä varten on määriteltävä funktiot, jotka palauttavat arviot onnistumisesta ($P_s(\bullet)$, $P_p(\bullet)$). Näiden arvojen oppimista varten robotilla on oltava muisti, johon se tallentaa tilanteet ja niiden tulokset. [Stone and Veloso, 1995]



Kuva 10. Puolustajan (maalivahti) kulma.

Kaikkien mahdollisten kulmien • oppiminen vaatii paljon muistilta ja tehokkuudelta. Täten 360˚ kannattaa jakaa M:ään yhtä suureen osaan ja tallennetaan kukin arvo muistiin taulukkoon Mem[n], missä on sekä arvot $P_s(\theta)$ ja $P_p(\theta)$, missä $\theta = \{360n / M \mid 0 \bullet n < M\}$. [Stone and Veloso, 1995]

Veloso ja Stone [1995] merkitsevät opetukset muodossa $E_{\bullet,a,r}$, missä • on kulma, a on toiminta (s = laukaisu, p = syöttö) ja r tulos (-1 = ei maalia, 1 = maali). Esim. $E_{45,p,1}$ merkitsee 45˚:n kulmasta laukaisua, joka meni maaliin.

Yksinkertaisin tapa laittaa arvot muistiin on pyöristää • lähimpänä olevaan θ:aan taulukossa Mem[θ] ja tallentaa se muistiin, esim. $P_a(\theta) = r$. Äkkiä katsottuna tämä vaikuttaisi hyvältä tallennusmenetelmältä, mutta sillä on huono puolensa. Olkoon M = 18, jolloin 360˚ olisi jaettu 18:aan 20˚:een väliin ja oltaisiin saatu tulokset $E_{19,p,1}$ ja $E_{22,p,-1}$. Tällöin hyökkääjä ei ole saanut maalia kun maalivahti on ollut 22˚:n kulmassa, mutta on saanut maalin kun kulma on ollut 19˚. Molemmat kulmat pyöristyisivät 20˚:een, mutta toisesta on tehty maali ja toisesta ei. Kuvassa 11 on määritelty kaava, jonka mukaan tallennettuna kulma •, joka on lähempänä kulmaa θ vaikuttaa enemmän muistipaikkaan Mem[θ]. Tällä kaavalla saadaan $E_{19,p,1}$: r' = 0.95 ja $E_{22,p,-1}$: r' = -0,90, jolloin Mem[20] = 0,95, eli kun maalivahti on 19˚ tai 22˚:een kulmassa, niin robotti laukaisee.

$$r' = r \cdot \left(1 - \frac{|\phi - \vartheta|}{360/M}\right)$$

$$Jos\ |r'| \geq |P_a(\vartheta)|,\ niin\ P_a(\vartheta) = r'.$$

Kuva 11. Tallentaminen [Stone and Veloso, 1995].

Muistin käytön tehokkuus riippuu muistin määrästä ja laadusta. Mitä useampiin lohkoihin muisti on jaettu (mitä suurempi M on), sitä useammasta tilanteesta saadaan tietoa. Tämä kuitenkin johtaa myös siihen, että tällöin monet muistipaikat jäävät käyttämättä ja tieto on puutteellista. Esimerkiksi jos M=360, niin robotti ottaa muistin jokaisen kulman. Yksi aste on kuitenkin niin pieni väli, että robotilla jää usein moniin paikkoihin muistissa aukkoja. Jotta aukkoja olisi mahdollisimman vähän on robotin käytävä läpi mahdollisimman monta eri tilannetta. Tällöin kuitenkaan yhtä muistipaikkaa kohti käytetyt opetuskerrat ovat vain kolmasosa verrattuna tilanteeseen jossa M = 120, jolloin jokaisen muistipaikan laatu on parempi. On löydettävä siis kultainen keskitie muistipaikkojen määrälle M. Tämän saa selville vain kokeilemalla.

4.3. Päätöksenteko

Pelkkä oppiminen (harjoittelu) ei riitä. On myös osattava käyttää opittuja taitoja. Robottien on osattava ko. tilanteessa päättää laukaisun ja syötön välillä. Robotti päättää toiminnastaan kuvan 12 kaavan mukaan. Robotti valitsee syötöstä ja laukaisusta aiemmin oppimansa perusteella paremman vaihtoehdon riippuen maalivahdin kulmasta (•).

Mikäli robotti ei ole vielä kyseisestä maalivahdin kulmasta syöttänyt tai laukaissut ($P_p(\bullet) = 0$ tai $P_s(\bullet) = 0$), niin robotti opettelee kyseisen tilanteen, jotta seuraavalla kerralla se osaisi toimia oikein. Jos robotti on aiemmin epäonnistunut sekä laukaistuaan että syötettyään, niin robotti valitsee näiden välillä satunnaisesti. Tällä tavalla robotti voi saada tarkempaa tietoa kyseisestä kulmasta.

$$
\begin{aligned}
&\textit{Jos } P_s(\phi) = P_p(\phi), \textit{ niin laukaise tai syötä satunnaisesti,}\\
&\textit{muuten jos } P_p(\phi) > 0 \textit{ ja } P_p(\phi) > P_s(\phi), \textit{ niin syötä,}\\
&\textit{muuten jos } P_s(\phi) > 0 \textit{ ja } P_s(\phi) > P_p(\phi), \textit{ niin laukaise,}\\
&\textit{muuten jos } P_p(\phi) = 0, \textit{ niin syötä,}\\
&\textit{muuten jos } P_s(\phi) = 0, \textit{ niin laukaise,}\\
&\textit{muuten } (P_s(\phi), P_p(\phi) < 0) \textit{ laukaise tai syötä satunnaisesti.}
\end{aligned}
$$

Kuva 12. Laukaisemisen ja syöttämisen valitseminen [Stone and Veloso, 1995].

5. Oppiminen neuroverkkojen avulla

Robottien jalkapallossa oppimisen tekee erilaiseksi muihin tekoälyn oppimistilanteisiin verrattuna se, että siinä robottien tulee osata pelata joukkueena. Perinteisissä tilanteissa robotin tarvitsee opetella vain miten toimia yksin, mutta robottien jalkapallossa joukkueessa saattaa olla jopa kymmenen muuta robottia, joiden kanssa tulee osata pelata mahdollisimman hyvin yhteen. Monen robotin yhteistyön oppiminen hyökkäyksessä ja puolustuksessa on mahdollista vain jos robotit oppivat pelaamaan yhdessä. Edellisessä luvussa käsittelin oppimista muistin avulla yksinkertaisessa tilanteessa. Tässä luvussa käsittelen oppimista neuroverkoilla vähän monimutkaisemmissa ja vaihtuvammissa tilanteissa.

5.1. Matalan tason oppiminen

Matalan tason oppimisena voidaan pitää liikkuvan pallon potkaisemista. Käsittelimme jo aikaisemmin tätä yksittäisen tilanteen kannalta, jossa robotin täytyi valita syötön ja laukaisun välillä. Yleensä pallo liikkuu, koska oma

pelaaja on syöttänyt sen, jolloin kyseessä on monen robotin yhteistoiminta [Stone and Veloso, 1996]. Robotti voi pystyä oppimaan helpolla toiminnan tietyssä tilanteessa, kuten aiemmin esitettiin. Tämä on tehotonta, sillä harvoin tulee juuri samanlaista tilannetta. Oppimisen täytyy olla tilanteesta riippumaton ja joustava, jotta robotit pystyisivät käyttämään taitojaan monissa eri tilanteissa. Tähän ei päästä kovin helpolla tallentamalla eri tilanteita muistiin.

Esimerkissämme on syöttäjä ja laukaisija. Syöttäjä liikkuu palloa kohti ja syöttää sen, johon laukaisija yrittää osata ajoittaa oikeaan aikaan tehdäkseen maalin. Laukaisijan on tehtävä päätöksensä pallon ja syöttäjän sijaintien perusteella. Jotta laukaisija pystyisi potkaisemaan pallon maaliin, on laukaisijan tähdättävä maalin ohi, jotta pallo menisi maaliin (kuva 13), sillä pallo potkaistaan suoraan liikkeestä. Laukaisijan tähtäyspisteen valitsemista kutsutaan tähtäystaktiikaksi. [Stone and Veloso, 1996]

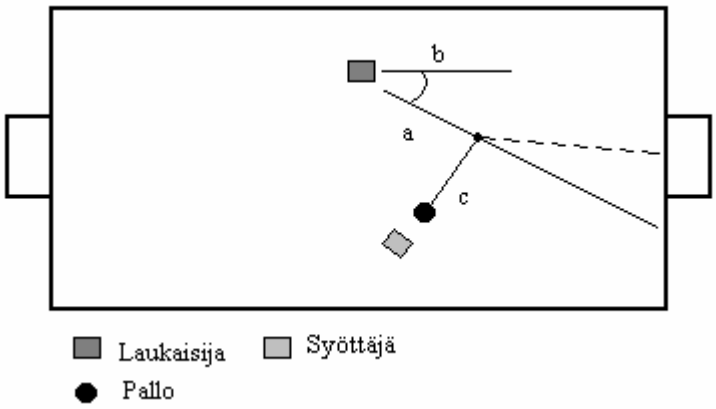
Laukaisemiseen liittyy neljä parametriä, jotka vaikuttavat vaikeusasteeseen: pallon nopeus, pallon reitti, maalin paikka ja toiminnan samanlaisuus [Stone and Veloso, 1996]. Pallo voi liikkua jokaisessa harjoitusesimerkissä eri nopeuksilla tai samoilla nopeuksilla. Pallo voi tulla samoja tai eri reittejä pitkin. Maali voi olla aina eri paikoissa. Tämä voidaan myös ajatella, että tähdätään maalin eri paikkoihin. Kaikki tilanteet voidaan tehdä samassa kentän neljänneksessä tai eri neljänneksissä. Robotit voivat yrittää käyttää hyväksi tilanteiden mahdollista symmetrisyyttä. Aiemmassa tilanteessa pallon nopeus ja reitti sekä maalin paikka olivat jokaisessa tilanteessa samat, mukaanlukien hyökkääjien paikat. Ainoastaan maalivahti liikkui. Nyt pyrimme opettamaan robotit toimimaan monissa eri tilanteissa.

5.2. Neuroverkoilla oppiminen

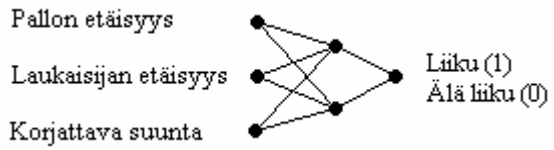
Stone ja Veloso [1996] käyttivät omien robottien opettamiseen neuroverkkoja. Neuroverkot ovatkin hyvä menetelmä opettaa robotteja toimimaan halutulla tavalla tietyissä tilanteissa. Oikein muodostetut neuroverkot pystyvät toimimaan erilaisissa tilanteissa ja ovat joustavia. On kuitenkin osattava päättää neuroverkon syötteet ja sen antamat vasteet. Kyseisen neuroverkon vaste kertoo, tuleeko laukaisijan lähteä liikkeelle (1) vai pysyä vielä paikallaan (0) riippuen pallon paikasta syötön jälkeen. Kun laukaisija lähtee liikkeelle oikeaan, aikaan se osuu palloon. Neuroverkon saamat syötteet kertovat laukaisijan nykyisestä ympäristöstä, joiden perusteella neuroverkko antaa vasteensa. Tähtäystaktiikkaa käyttämällä laukaisija pystyy valitsemaan oman reittinsä (kuva 13, suora a) ja pallon reittikin pystytään arvioimaan (kuva 13, suora c). Näiden kahden suoran avulla laukaisija pystyy arvioimaan oman

laukaisupisteensä, eli tähtäystaktiikan. Tämän pisteen avulla laukaisija pystyy laskemaan neuroverkon kolme syötettä: pallon etäisyys leikkauspisteeseen, laukaisijan etäisyys leikkauspisteeseen ja robotin suunnan korjaaminen (kuva 13, suorien a ja b välinen kulma). [Stone and Veloso, 1996]

Stonen ja Veloson [1996] opettamalla neuroverkolla (kuva 14) toimiva robotti onnistui tekemään maalin peräti 96,5%:n varmuudella. Jokaisessa harjoitus- ja testaustilanteessa pallon ja robottien nopeus oli kuitenkin aina sama. Kyseessä oli siis vielä suhteellisen yksinkertainen tilanne.



Kuva 13. Pallon laukaiseminen suoraan syötöstä.



Kuva 14. Laukaisemisen oppimiseen käytetty neuroverkko [Stone and Veloso, 1996].

Kun pallon nopeutta vaihdettiin, niin kolmen syötteen neuroverkko pystyi toimimaan enää alle 50%:n varmuudella. Todellisessa tilanteessa pallo ei aina liiku samalla nopeudella, vaan nopeus vaihtelee. Jotta laukaisija pystyisi tekemään maalin riippumatta pallon nopeudesta, on sen otettava huomioon myös pallon nopeus. Tätä varten lisäämme edelliseen neuroverkkoon neljännen syötteen: pallon nopeuden. Kun robotteja opetettiin tällä uudella neuroverkolla eri pallon nopeuksilla, niin onnistumisprosentti oli 91,5 [Stone and Veloso, 1996].

Kun muutettiin laukaisemiseen liittyvää toista parametriä, pallon reittiä, niin robotti ei kyennyt tekemään ollenkaan maalia. Tämä johtui siitä, että robotin tähtäystaktiikka ei toiminut ja robotti tähtäsi väärään kohtaan.

Robotin tulisi osata vaihtaa tähtäystaktiikkaa tarvittaessa. Tämän ongelman ratkaisemiseksi Stone ja Veloso [1996] tekivät toisen neuroverkon, jonka avulla pystytään laskemaan aiemman neuroverkon syötteitä. Uuden verkon syötteiksi valittiin pallon reitin ja laukaisijan suunnan välinen kulma ja pallon nopeus. Neuroverkko opetettiin antamaan vasteena kulma, jonka verran laukaisijan on käännyttävä, jotta sen tähtäystaktiikka olisi oikea. Tämä kulma olisi aiemmin käytetyn neljän syötteen neuroverkon yksi syöte. Stone ja Veloso [1996] onnistuivat näiden kahden neuroverkon yhdistelmällä 95,4%:n varmuudella.

Viimeisen laukaisuparametrin, maalin siirtämisen, tarkoituksena on ajatella robotin yrittävän potkaista palloa maalin eri kohtiin. Edellä käytetyillä kahden neuroverkon yhdistelmällä laukaiseminen onnistui hyvin myös kun maalia oli siirretty. Kun maalia siirrettiin yhden maalin verran alaspäin, oli onnistumisprosentti 84,1 ja vastaava luku ylös oli 74,9. [Stone and Veloso, 1996]

Neuroverkko on yleensä sitä tarkempi, mitä enemmän se saa syötteitä. Tämä asia piti paikkansa ainakin tässä tehtyihin neuroverkkoihin. Tässä käytetty menetelmä, jossa toisesta neuroverkosta saatuja vasteita käytetään toisen neuroverkon syötteinä, on epätavallista. Sillä on myös omat riskinsä. Ensimmäisen neuroverkon on toimittava hyvin, jotta sitä käyttävä neuroverkkokin toimisi hyvin. Tässä esimerkissä kyseisen tyyppinen neuroverkkoyhdistelmä toimi kuitenkin mainiosti.

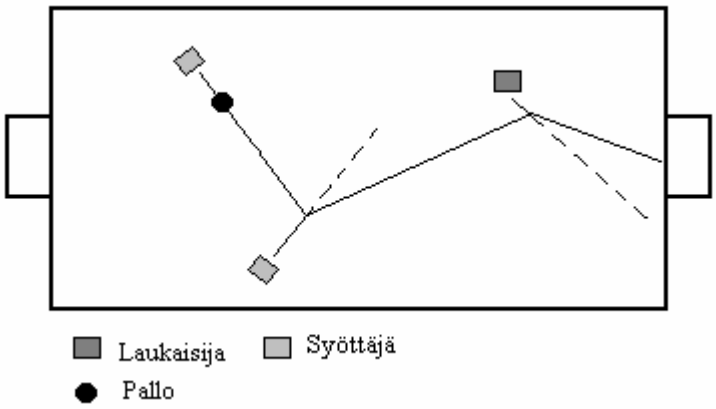
5.3. Korkean tason oppiminen

Korkean tason osaaminen perustuu yleensä perusasioiden osaamiselle. Sama asia pätee niin tekoälyssä kuin robottien jalkapallossakin. Vaativammat tekniikat rakennetaan yksinkertaisempien tekniikoiden päälle. Edellä esitelty laukaisumenetelmä on kaiken korkeamman tason toiminnan pohjana robottien jalkapallojoukkueessa. Samalla menetelmällä pystytään niin syöttelemään kuin rakentamaan hyökkäystä. Edellä käytetyt neuroverkot ovat erittäin käyttökelpoisia eri tilanteissa, koska ne ovat riippumattomia tilanteesta ja ne huomioivat eri muuttujat, jotka vaikuttavat pallon potkaisemiseen ja ne ovat vielä joustavia.

Aiemmassa tilanteessa oli vain kaksi robottia. Todellisuudessa pelissä on kuitenkin useampi robotti molemmilla puolilla. Tämä mahdollistaa useamman kuin kahden pelaajan välisen yhteistyön. Oikeassakin jalkapallossa monta pelaajaa rakentavat tilanteita yhteistyöllä. Robottien jalkapallossakin useammalla robotilla on parempi mahdollisuus tehdä monimutkaisempia

pelikuvioita. Tämä kuitenkin vaatii korkean tason oppimista ja taitoa: joukkuepeliä.

Aiemmin käytettyä maaliin laukaisemista voidaan käyttää myös syöttämiseen. Tällöin kohteena ei olisi maali vaan toinen robotti tai jokin muu kohde. Aiemmassa tilanteessa vain laukaisijaa opetettiin. Kun opettaa usealle robotilla saman tekniikan, mutta niin että voidaan myös syöttää, niin on mahdollista saada aikaiseksi kuvan 15 mukainen syöttöketju. Siinä jokainen robotti käyttää oppimiaan neuroverkkoja kukin hieman eri tarkoitukseen.



Kuva 15. Syöttöketjun muodostaminen.

Tällaisessakin tilanteessa roboteille on hyötyä agenttikeskeisistä hakumenetelmistä ja sitä kautta ennakoinnista. Niiden avulla robotit pystyvät reagoimaan ympäristön muutoksiin. Esimerkiksi yllättäen syöttö- tai laukaisureitille tullut toinen robotti sotkee suunnitelmat täysin. Mikäli robotti ei huomioi sen hetkistä ja hetken päästä tulevaa tilannetta, saattaa pallon menetys olla erittäin lähellä. Tällainen tilanne voi syntyä esimerkiksi kun vastustajan robotti on liikkumassa syöttöreitin eteen. Agenttikeskeisellä haulla robotti kuitenkin huomaa korjata suunnitelmiaan. Robotti voi esimerkiksi yrittää väistää vastustajaa ja syöttää sen jälkeen tai lähteä kuljettamaan palloa.

6. Yhteenveto

Tekoäly on oleellinen osa robottien jalkapallossa. RoboCup-kilpailun ja robottien jalkapallon avulla tutkijat pystyvät kehittämään tekoälymenetelmiä eteenpäin. Tätä kautta saadaan paljon käytännön sovelluksissa käyttökelpoisia menetelmiä tutkittua ja kehitettyä.

Olen pyrkinyt tässä tutkimuksessa keskittymään hakuun ja oppimiseen, mutta se on vain pintaraapaisu muihin tekoälymenetelmiin, joita robottien jalkapallossa tarvitaan. Luultavasti vielä menee useita vuosia ennen kuin

robotit pystyvät pelaamaan ihmisiä vastaan jalkapalloa. Tämä on kuitenkin joka päivä enemmän mahdollista tekoälyn, tekniikan ja muiden alojen kehittymisten myötä. Ehkä vielä vuonna 2050 robotit voittavat ihmiset jalkapallossa, voitettiinhan Kasparovkin shakissa.

Viiteluettelo

[Candea Hu, Iocchi, Nardi and Piaggio, 2001] Ciprian Candea, Huosheng Hu, Luca Iocchi, Daniele Nardi and Maurizio Piaggio, Coordination in multi-agent RoboCup teams. *Robotics and Autonomous Systems* **36** (2001), 67-86.

[Koenig, 2001] Agent-Centered Search. *AI Magazine*, Winter 2001, 109-131.

[robocup.org] RoboCup Official Site. http://www.robocup.org/.

[Stolzenburg, Obst and Murray 2002] Frieder Stolzenburg, Oliver Obst and Jan Murray, Qualitative velocity and ball interception. *LNAI* **2479** (2002), 283-298.

[Stone and Veloso, 1995] Peter Stone and Manuela Veloso, Beating a defender in robotic soccer: memory-based learning of a continuous function. Draft, December 11, 1995. Available as http://www-2.cs.cmu.edu/afs/cs/usr/pstone/public/papers/Membased-html/Membased-tech.html

[Stone and Veloso, 1996] Peter Stone and Manuela Veloso, Towards collaborative and adversarial learning: a case study in robotic soccer. Draft, August 22, 1996. Available as http://www-2.cs.cmu.edu/afs/cs/usr/pstone/public/papers/96ijhcs-html/article.html

[Takahashi, Tamura and Asada 2002] Yasutake Takahashi, Takashi Tamura and Minoru Asada, Strategy learning for a team in adversary environments. *LNAI* **2377** (2002), 224-233.

[Tu 2002] Kyo-Yang Tu, Design and implementation of a soccer robot with modularized control circuits. *LNAI* **2377** (2002), 459-464.

[Veloso, Stone and Han, 1997] Manuela Veloso, Peter Stone and Kwun Han, The CMUnited-97 robotic soccer team: perception and multiagent control. Draft, October, 1997. Available as http://www-2.cs.cmu.edu/afs/cs/usr/pstone/public/papers/97robot-paper/robot-paper.html

Rinnakkaistiedonsiirto

Antti Huokko

Tiivistelmä

Tutkimuksessa kuvaan tekniikan, jolla voidaan tehostaa verkosta tapahtuvaa tiedonsiirtoa. Lisäksi tekniikka tuo mukanaan monia muita etuja, kuten esimerkiksi tiedonsiirron virhealttiuden vähenemisen.

1. Johdanto

Perinteisesti verkossa tapahtuva tiedonsiirto on ollut yhdeltä yhdelle -tiedonsiirtoa. Yhdeltä yhdelle -tiedonsiirrossa asiakas avaa yksittäisen yhteyden palvelimelle ja pyytää tarvitsemansa tiedon. Tämän jälkeen asiakas jää odottamaan, että palvelin lähettää pyydetyn tiedon joko yhdessä tai useammassa osassa.

Yhdeltä yhdelle -tiedonsiirrossa tiedonsiirtonopeuteen vaikuttavat palvelin- ja asiakaskoneiden tehokkuus sekä niitä yhdistävän verkon nopeus. Käytännössä kuitenkin verkko on useimmiten tiedonsiirron tehokkuuden määräävä hitain komponentti. Verkonkaan nopeus ei ole vakio, vaan siihen vaikuttavat monet seikat, kuten esimerkiksi se kuinka monta muuta tiedonsiirtoa on parhaillaan käynnissä. On myös mahdollista, että palvelinkone hidastaa tiedonsiirtoa. Jos palvelimella on useita asiakkaita, joita se joutuu palvelemaan samanaikaisesti, niin palvelimen tehokkuus laskee ja asiakkaat joutuvat odottamaan hidasta palvelinta.

Rodriguez et al. [2000] esittivät tekniikan nimeltään rinnakkaistiedonsiirto (parallel downloading), jonka avulla voidaan tehostaa tiedonsiirtoa. Rinnakkaistiedonsiirron perusajatuksena on, että tietoa ei enää perinteiseen tapaan siirretäkään yhdeltä palvelimelta, vaan useammalta samanaikaisesti. Käytännössä tämä tarkoittaa, että asiakkaalla on tiedossaan joukko palvelimia, jotka tarjoavat saman tiedon. Niinpä asiakas ottaa samanaikaisesti yhteyden kaikkiin palvelimiin ja pyytää kultakin palvelimelta eri osan tiedosta. Kun asiakas saa kerättyä kaikki tiedon osat, niin se pystyy kokoamaan niistä pyydetyn tietokokonaisuuden.

Eräs keskeinen rinnakkaistiedonsiirrolla saavutettava etu on, että tiedonsiirto nopeutuu. Tämä johtuu ensinnäkin siitä, että rinnakkaistiedonsiirrossa verkon nopeus ei enää vaikutakaan niin ratkaisevasti tiedonsiirtonopeuteen,

kuin yhdeltä yhdelle tiedonsiirrossa. Tämä on seurausta siitä, että rinnakkaistiedonsiirrossa tietoa siirretään useasta eri lähteestä. Tämä tarkoittaa sitä, että käytetään samanaikaisesti useita eri reittejä tiedon siirtämiseen.

Toinen tiedonsiirtonopeutta parantava seikka on, että tietoa ei siirretäkään enää yhdeltä palvelimelta, vaan useammalta samanaikaisesti. Tällöin ei haittaa, vaikka osa palvelimista, onkin hitaita. Tilanne tasoittuu siten, että hitaammilta palvelimilta pyydetään suhteessa pienempi osa siirrettävästä tiedosta kuin nopeammilta. Näin saadaan samalla tasattua palvelimiin kohdistuvaa kuormaa.

2. Nykytilanne

Tiedonsiirtoa varten on kehitetty protokollia, jotka määräävät asiakkaan ja palvelimen välisen kommunikointi- ja tiedonsiirtotavan. Protokolla on siis tapa järjestää tiedonsiirto. Kuvaan seuraavaksi yleisellä tasolla kaksi yleisintä protokollaa, joita käytetään verkkoympäristössä järjestämään tiedonsiirto eri koneiden välillä. Lisäksi kerron tiedon peilauksesta (mirroring), joka on yleisesti käytetty tapa tehostaa tiedonsiirtoa.

2.1. FTP tiedonsiirtoprotokolla

FTP [RFC 959] eli File Transfer Protocol on tiedonsiirtoprotokolla, joka on suunniteltu eritoten tiedostojen siirtämistä varten. FTP on niin sanottu pyyntö-vastaus protokolla (request-response protocol). Pyyntö-vastaus -protokolla on protokollatyyppi, jossa palvelin vain odottelee passiivisesti, että asiakas pyytää siltä jotakin. Kun asiakkaan pyyntö saadaan hoidettua, niin palvelin jää taas odottamaan seuraavaa pyyntöä.

FTP-tiedonsiirrossa asiakasohjelma avaan yhteyden palvelimelle, jota pitkin asiakas lähettää pyyntöjä ja palvelin vastauksia. Asiakas lähettää tätä yhteyttä pitkin esimerkiksi pyynnön kirjautua palvelimelle sisään, johon palvelin lähettää vastauksen sisäänkirjautumisoperaation onnistumisesta tai epäonnistumisesta. Kun halutaan suorittaa varsinainen tiedonsiirto, asiakas avataan toisen yhteyden palvelimelle, jota pitkin varsinainen tieto sitten siirretään joko asiakkaalta palvelimelle tai palvelimelta asiakkaalle.

FTP on yksinkertainen ja tehokas tapa toteuttaa tiedostojen jakaminen. FTP:ssä on kuitenkin puutteensa. Suurin näistä on varmaankin salauksen puuttuminen. Kaikki liikenne asiakkaan ja palvelimen välillä kulkee salaamattomana, joka ei ole hyvä asia, sillä esimerkiksi käyttäjätunnukset ja salasanat siirretään selväkielisenä tekstinä, mikä on tietoturvan kannalta melko ongelmallista.

2.2. HTTP tiedonsiirtoprotokolla

HTTP [RFC 2616] eli Hypertext Transfer Protocol on tiedonsiirtoprotokolla, joka on tarkoitettu hypermedian siirtämiseen verkon kautta. Myös HTTP on pyyntö-vastaus -protokolla. HTTP-protokollan välityksellä voidaan siirtää monen tyyppistä tietoa. Siirrettävä sisältö voi olla esimerkiksi HTML sivu, kuva tai animaatio. HTTP-protokollaa käyttäen on myös mahdollista tehdä tiedostonsiirtoa.

HTTP-tiedonsiirrossa asiakas ottaa yhteyden palvelimeen ja pyytää esimerkiksi tiettyä HTML-sivua. Pyyntö alkaa pyyntörivillä, joka määrittelee operaation joka palvelimen halutaan suorittavan. Pyyntö voi olla vaikkapa HTML-sivun siirto. Seuraavana pyynnössä on pyyntöön liittyvät otsikot, jotka kertovat tarkemmin, kuinka pyyntö tulisi suorittaa. Otsikot kertovat esimerkiksi, minkä tyyppistä tietoa asiakas voi ottaa vastaan. Lopuksi pyynnössä voi olla jokin palvelimelle lähetettävä tietoalkio.

Saatuaan pyynnön palvelin lähettää vastauksen, joka on tässä tapauksessa pyydetty HTML-sivu. Vastaus alkaa tilarivillä, joka kertoo, onnistuiko pyyntö vai tuliko jokin virhe. Seuraavana tulevat vastaukseen liittyvät otsikot, jotka kertovat esimerkiksi, kuinka pitkä palautettava sisältö on. Lopuksi tulee itse tietosisältö, joka voi olla joko tekstimuotoista tai binäärimuotoista dataa.

HTTP on huomattavasti monipuolisempi protokolla kuin FTP, ja se mahdollistaa hyvin monentyyppisten tietosisältöjen siirtämisen. Lisäksi HTTP:n avulla asiakkaan on mahdollista määritellä hyvinkin tarkkaan, kuinka tiedonsiirto käytännössä suoritetaan esimerkiksi määrittelemällä, kuinka suurissa osissa tieto siirretään tai mikä osa tiedosta halutaan siirtää.

2.3. Tiedon peilaus

Tiedon peilauksessa sama tietosisältö annetaan usealle palvelimelle jaettavaksi. Kun käyttäjä haluaa siirtää tarvitsemansa tiedon, annetaan hänen valita palvelin jolta hän sitten suorittaa tiedonsiirron.

Tiedon peilauksella saavutetaan monia etuja verrattuna siihen, että tietosisältöä jaettaisiin ainoastaan yhdeltä palvelimelta. Koska palvelimia on useita, niin ei haittaa vaikka osa niistä on väliaikaisesti poissa toiminasta häiriön tai ylläpidon vuoksi. Jos palvelin ei vastaa, voi käyttäjä siirtää tiedon joltakin toiselta palvelimelta. Useamman palvelimen tapauksessa palvelimiin kohdistuva rasitus saadaan tasattua. Tämä johtuu siitä, että käyttäjät eivät tee pyyntöjä ainoastaan yhdelle palvelinkoneelle, jonka olisi selvittävä niistä kaikista, vaan pyynnöt jakautuvatkin useamman palvelimen kesken. Tämä parantaa tiedonsiirron tehokkuutta myös käyttäjän näkökulmasta, sillä jos

jokin palvelin on kovin hidas, voidaan vaihtaa palvelinta ja siirtää tieto palvelimelta, joka ei ole niin pahasti kuormitettu. Lisäksi peilauspalvelimet pyritään usein sijoittelemaan siten, että ne sijaitsevat verkon eri osissa. Tällöin jos jokin osa verkkoa ruuhkautuu pahasti, niin se ei vaikuta kuin yhteen palvelimeen. Lisäksi palvelimien sijoittamisella verkon eri osiin pyritään siihen, että onpa käyttäjä missä tahansa verkon osassa, niin löytyy palvelin, joka on häntä lähellä ja siten mahdollistaa nopean tiedonsiirron.

Tiedon peilaus on yleinen tekniikka tehostaa suositun tiedon jakelua eritoten Internetissä. Tiedon peilaukseen liittyy kuitenkin myös ongelmia. Yksi keskeisimmistä on, että toimiakseen tehokkaasti tiedon peilaus olettaa, että käyttäjät osaisivat tehdä järkevän valinnan käyttämänsä palvelimen suhteen. Tämä valinta ei kuitenkaan ole helppo tehdä, sillä esimerkiksi pelkän maantieteellisen läheisyyden perusteella tehty valinta ei välttämättä tuota optimaalista tulosta. Lisäksi tiedon peilaus voi tekniikkana olla riittämätön, jos kysyntä jaettavaa tietoa kohtaan kasvaa suureksi. Tällöin tehokkaampi tapa jakaa tietosisältö usealle käyttäjälle on käyttää jotakin multicasting-tyyppistä tiedonjakelutekniikkaa, joka on paljon paremmin skaalautuva tekniikka [Byers et al., 1999].

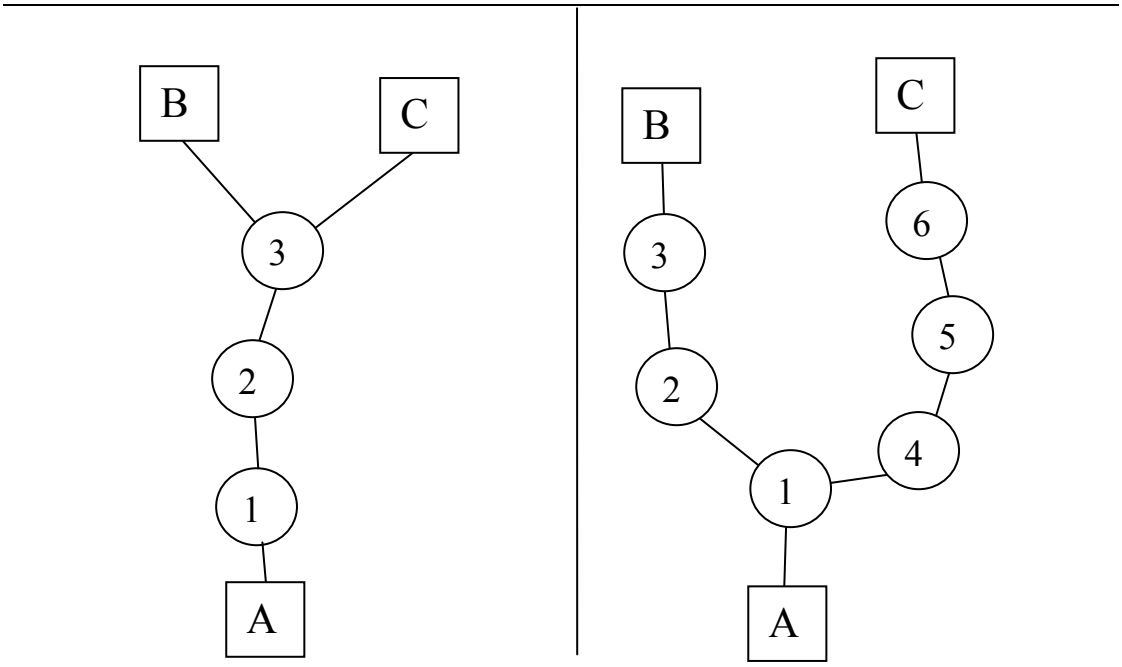
3. Tekniikan esittely

Internetistä siirretään yhä suurempia tietomääriä, kuten esimerkiksi musiikkia, elokuvia, ohjelmia ja jopa käyttöjärjestelmiä. Verkon kapasiteetti ei kuitenkaan kasva samaan tahtiin verkolle asetettavien vaatimusten kanssa. Niinpä on yleistä, että hieman suurempia tietomääriä siirrettäessä käyttäjä joutuu odottamaan pitkiä aikoja. Tämä johtuu siitä, että vaikka käyttäjällä olisikin nopea yhteys verkkoon, niin verkko on monesti niin raskaasti kuormitettu, että yksittäisen käyttäjän kannalta jäädään kauas optimaalisesta siirtonopeudesta.

Toisaalta voidaan nähdä, että on olemassa mahdollisuus tehostaa siirtonopeuksia käyttämällä hyväksi olemassa olevia tekniikoita sekä infrastruktuuria. On hyvin tavallista, että suosittuja tietosisältöjä peilataan useille palvelimille. Lisäksi kaikki palvelimet eivät yleensä ole täydessä käytössä, joten potentiaalista palvelintehoa jää käyttämättä. Niinpä olisi hyödyllistä päästä käyttämään hyödykseen käyttämättä jäävä palvelintehoa kasvattamaan tiedonsiirtonopeuksia.

Toisaalta suuret verkot ja eritoten Internet koostuvat useista vaihtoehtoisista reiteistä eri koneiden välillä. Niinpä esimerkiksi yhteys kahdelle eri palvelimelle todennäköisesti käyttää hyvin suurelta osin eri reittiä. Tämä on

oleellista, jos halutaan kasvattaa tiedonsiirtonopeutta uudistamatta verkkoa fyysisesti. Yhteyden nopeuttaminen ei onnistu, jos kaikki lähetettävät ja vastaanottavat paketit kulkevat samaa reittiä. Tällöin yhteydestä tulee helposti pullonkaula, sillä tietty yhteys pystyy viemään paketteja lävitse ainoastaan rajoitetulla nopeudella. Jos pakettien täytyy kulkea samaa pullonkaulaksi muodostuvaa yhteyttä pitkin, ei tiedonsiirtonopeutta voida kasvattaa, vaikka tietoa pyydettäisiinkin usealta eri palvelimelta.



Kuva 1. Kaksi erilaista reititystilannetta asiakaskoneen (A) ja kahden palvelinkoneen (B ja C) välillä.

Kuvassa 1 on kaksi erityyppistä reititystilannetta asiakkaan ja kahden palvelimen välillä. A on asiakas ja B ja C ovat palvelimia. Numeroidut ympyrät ovat reitittimiä, joiden kautta muodostetaan yhteys asiakkaan ja palvelimien välille.

On helppo havaita, että vasemmanpuoleinen tilanne on melko ongelmallinen tiedonsiirron tehostamisen kannalta. Jos reitillä 3 – 2 – 1 – A on hidas yhteys, niin tiedonsiirron tehostaminen ei onnistu muuten kuin päivittämällä verkko nopeampaan.

Oikeanpuoleinen kuva taas esittää melko optimaalisen tilanteen. Yhteys asiakkaan ja palvelimien välillä kulkee suurimman osan matkaa eri reittiä. Tällöin toisen reitin kuormittaminen ei vaikuta toisen reitin toimintaan. Ainoa reitityksen näkökulmasta potentiaalinen pullonkaulakohta on yhteys reitittimen 1 ja asiakkaan välillä.

Tiedonsiirto on perinteisesti ollut yhdeltä monelle -tiedonsiirtoa. Asiakas ottaa yhteyden palvelimeen, jolta se pyytää haluamansa tiedon, jonka palvelin sitten lähettää asiakkaalle. Palvelin puolestaan pystyy palvelemaan useita asiakkaita samanaikaisesti.

Rodriguezin ja muiden [2000] esittämä tekniikka nimeltään rinnakkaistiedonsiirto poikkeaa perinteisistä tiedonsiirtotavoista siten, että nyt asiakas ottaakin yhteyden useaan palvelimeen samanaikaisesti ja siirtää niiltä haluamansa tiedon. Tekniikka hyödyntää edellä mainittuja seikkoja, eli saman tiedon peilausta useille palvelimille sekä verkon sisällä olevia useita vaihtoehtoisia reittejä.

Rinnakkaistiedonsiirto voidaan jakaa vaiheisiin ja esittää algoritmina [Miu and Shih, 1999a]. Algoritmin toiminta asiakasohjelman kannalta esitettynä on seuraava:

1. Selvitä, mitkä palvelimet tarjoavat halutun tietosisällön.
2. Selvitä siirrettävän tietosisällön koko.
3. Jaa koon perusteella tietosisältö lohkoihin.
4. Pyydä yhtä lohkoa kultakin palvelimelta.
5. Odota vastausta.
6. Pyydä uutta lohkoa palvelimelta heti, kun vanha on saatu siirrettyä.
7. Toista kohtia 5 - 7 kunnes kaikki lohkot on siirretty.

3.1. Palvelimien selvittäminen

Ensiksi asiakkaan on saatava selville, mitkä palvelimet tarjoavat halutun tiedon. Koska rinnakkaistiedonsiirto ei ole yleistynyt käytäntöön, ei vielä ole olemassa tätä tehtävää hoitavaa verkkopalvelua. Kuitenkin on esitetty muutamia tekniikoita, joilla palvelimien selvitys voitaisiin toteuttaa.

Eräs tapa tarjota asiakkaalle tieto palvelimista, joilta haluttu tietosisältö on siirrettävissä, on laajentaa nimipalvelimia siten, että ne pystyvät pyydettäessä palauttamaan asiakkaalle kaikkien palvelimien osoitteet, jotka jakavat haluttua tietoa [Kangasharju et al., 1999]. Toinen tapa on perustaa palvelin, joka on varta vasten tarkoitettu listaamaan palvelimia jotka jakavat samaa sisältöä.

Kun asiakas on saanut tiedon halutun sisällöntarjoavista palvelimista, on asiakkaan vielä päätettävä, miltä palvelimilta tiedonsiirto halutaan tehdä. Mahdollisimman monelta palvelimelta tiedon siirtäminen ei välttämättä tuota optimaalista tulosta, sillä jokainen luotu yhteys aiheuttaa asiakkaalle ylimääräistä kuormaa. Niinpä monesti olisi tiedonsiirron kannalta optimaalista

valita ainoastaan muutama tehokas palvelin ja siirtää tieto niiltä [Philopoulos and Maheswaran, 2001].

3.2. Tietosisällön jakaminen lohkoihin

Seuraavassa vaiheessa asiakas selvittää siirrettävän tietosisällön koon. Koon selvittäminen on tärkeää, jotta asiakas voi ennen varsinaisen tiedonsiirron alkamista jakaa tiedon erikseen pyydettäviin lohkoihin. Asiakas voi selvittää tietosisällön koon nimipalvelimelta, jos se tukee kyseistä ominaisuutta. Vaihtoehtoinen tapa selvittää tietosisällön koko on kysyä se palvelimelta, jolta itse tiedonsiirtokin tehdään.

Kun asiakas on selvittänyt siirrettävän tiedon koon, se jakaa tietokokonaisuuden lohkoihin. Tämä on oleellista työnjaon toteuttamisen kannalta. Kukin lohko nimittäin pyydetään eri palvelimelta samanaikaisesti, jolloin saavutetaan tiedonsiirron rinnakkaisuus.

Lohkon koko on eräs kriittinen tekijä rinnakkaistiedonsiirron optimoinnissa [Rodriguez and Biersack, 2002]. Ensinnäkin lohkoja on oltava määrällisesti huomattavasti enemmän kuin palvelimia, joilta tietoa siirretään. Muuten koko rinnakkaistiedonsiirron idea jää toteutumatta. Tällöin tulee nimittäin helposti tilanne, jossa osalta palvelimista ei siirretä tietoa ollenkaan, kun taas toisilta odotetaan tietoa. Tämä ei tietenkään ole optimaalinen tilanne.

Toisekseen lohkojen koon on oltava tarpeeksi pieni. Tällöin saavutetaan se etu, että nopeammilta yhteyksiltä pyydetään suurempi osa lohkoista kuin hitailta yhteyksiltä. Tämä tehostaa tiedonsiirtoa huomattavasti, sillä on tehokasta siirtää mahdollisimman suuri osa tiedosta nopeilta yhteyksiltä.

Lisäksi pieni lohkokoko tärkeää, jotta päästäisiin tilanteeseen, jossa kukin palvelin saa tiedonsiirtonsa tehtyä samaan aikaan. Tämä on taas tärkeää tiedonsiirron tehokkuuden saavuttamiseksi. Jos asiakas joutuu odottelemaan viimeistä lohkoa hitaalta palvelimelta, niin tiedonsiirtoajat kasvavat tarpeettomasti, vaikka olisi mahdollista siirtää tieto nopeamminkin käyttämällä nopeampaa palvelinta. Pieni lohkokoko takaa, että turhilta odotteluilta vältytään: jos siirrettävä lohko on pieni, niin myös odotusaika on lyhyt. Niinpä mitä pienempi on siirrettävä lohko, sitä suurempi on todennäköisyys, että palvelimet lopettavat tiedonsiirron samaan aikaan, jolloin turhalta odottelulta vältytään.

Eräs optimointitekniikka on, että kun kaikki paitsi viimeinen lohko on saatu siirrettyä, niin viimeistä lohkoa pyydetään kaikilta palvelimilta. Vastaanotettu lohko hyväksytään nopeimmalta palvelimelta. Tällöin saadaan helpotusta ongelmalliseen tilanteeseen, jossa tiedonsiirtoajat kasvavat vain,

koska viimeistä lohkoa joudutaan odottelemaan, kun tiedonsiirto on muuten saatu päätökseen [Zeitoun et al., 2002].

Kolmas lohkon kokoon vaikuttava seikka on jossakin määrin ristiriidassa edellisen kanssa. Lohkon koko nimittäin pitäisi olla tarpeeksi suuri, jotta vältytään ylimääräiseltä verkkoliikenteeltä. Jokainen uusi lohko on nimittäin pyydettävä erikseen palvelimelta, mikä tarkoittaa aina uutta viestiä asiakkaalta palvelimelle.

Toinen suurella lohkokoolla saavutettava etu on, että odotteluaikoja lohkojen välillä tulee vähemmän. Aina kun uutta lohkoa pyydetään, joudutaan odottelemaan, että pyyntö saadaan vastaanotettua ja käsiteltyä palvelimella ennen kuin varsinaista tiedonsiirtoa päästään jatkamaan. Niinpä mitä vähemmän lohkoja joudutaan pyytämään, eli mitä suurempi on lohkokoko, sitä tehokkaampaa tiedonsiirto on. Oikea lohkokoko määräytyy näiden vaatimusten kompromissina.

On myös esitetty tekniikkaa, jossa asiakas dynaamisesti säätää lohkokokoa tiedonsiirron aikana riippuen yhteyden nopeudesta. Tämä tarkoittaa sitä, että eri yhteyksille on käytössä erisuuruiset lohkokoot. Tämä on eräs keino optimoida tiedonsiirtoa, sillä enää ei ole ennalta määrätty tiedonsiirron ominaisuuksia, jotka eivät välttämättä ole optimaaliset kaikkiin eri tilanteisiin [Miu and Shih, 1999b].

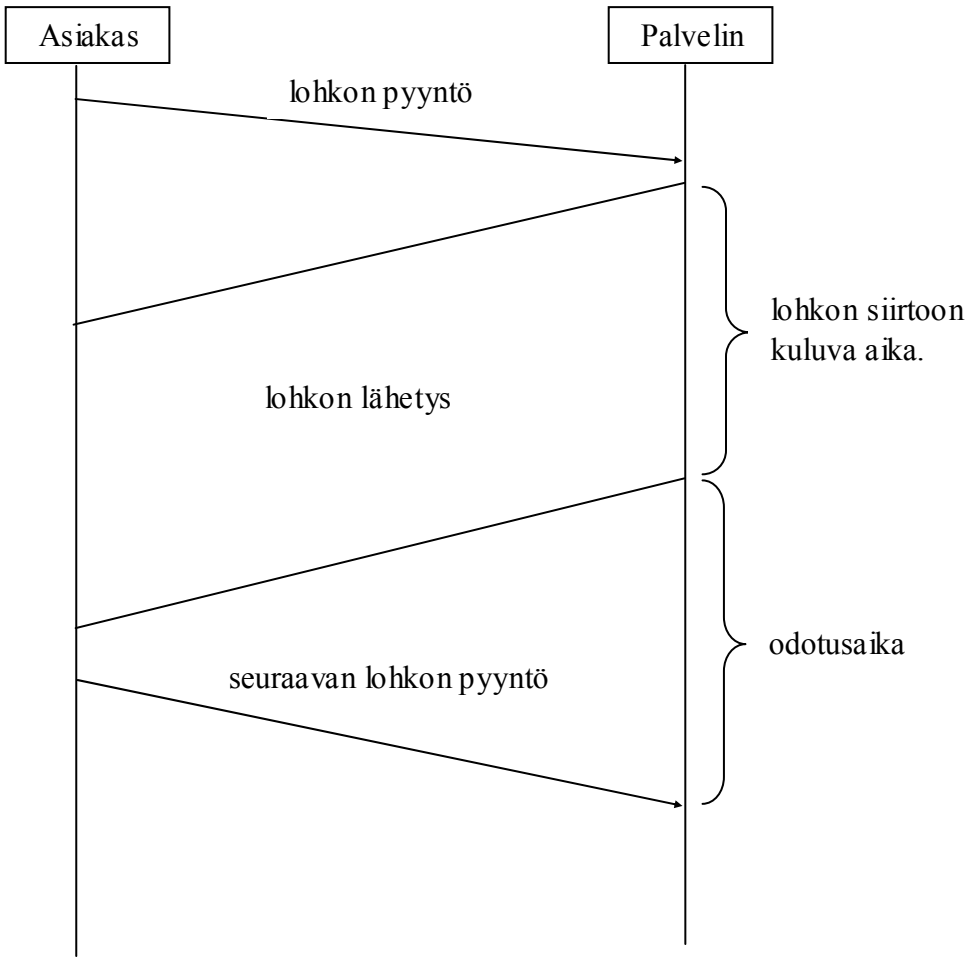
3.3. Tiedonsiirto

Kun siirrettävä tieto on saatu jaettu lohkoihin, niin yhtä lohkoa pyydetään kultakin palvelimelta ja jäädään odottamaan vastausta eli pyydettyä lohkoa. Tämän jälkeen palvelin lähettää lohkon ja asiakas ottaa sen vastaan. Lohko saatetaan joutua lähettämään useassa osassa, jos lohkon koko on suuri.

Kun jokin palvelin saa lohkonsa siirrettyä, niin tältä palvelimelta pyydetään välittömästi uutta lohkoa. Tätä jatketaan, kunnes kaikki lohkot on saatu siirrettyä. Koska yhteydet ja palvelimet toimivat eri nopeuksilla, niin nopeat palvelimet pystyvät siirtämään lohkonsa nopeammin kuin hitaat. Niinpä nopeilta palvelimilta pyydetään enemmän lohkoja kuin hitailta palvelimilta, jolloin tiedonsiirto tehostuu, sillä nopeilta yhteyksiltä siirretään suurempi osa tiedosta kuin hitailta. Käytännössä tämä tarkoittaa siis sitä, että hyödynnetään mahdollisimman tehokkaasti kunkin yhteyden siirtokapasiteetti.

Kuvassa 2 on graafisesti esitetty asiakkaan ja yhden palvelimen välinen kommunikaatio. Alkutilanteessa palvelin odottelee seuraavaa pyyntöä ja asiakas taas lähettää pyynnön palvelimelle. Tämän jälkeen asiakas jää odottamaan palvelimen vastausta. Kun palvelin vastaanottaa asiakkaan pyynnön,

niin se alkaa lähettää asiakkaalle tämän pyytämää lohkoa. Itse lohkon lähetys saattaa tapahtua useassa osassa. Kun asiakas on vastaanottanut lohkon, niin se pyytää palvelinta lähettämään seuraavan puuttuvan lohkon. Tätä jatketaan, kunnes kaikki lohkot on saatu siirrettyä.



Kuva 2. Lohkon pyytäminen palvelimelta ja lohkon lähetys.

Kuvasta 2 on havaittavissa, että kommunikointi ei tapahdu tehokkaimmalla mahdollisella tavalla. Kun palvelin on saanut siirrettyä yhden lohkon, niin se joutuu odottelemaan seuraavan lohkon pyyntöä. Tämä odotusaika johtuu siitä, että viestiltä kestää oman aikansa kulkea verkon läpi koneesta toiseen. Samanlainen odotusaika on myös havaittavissa asiakkaan päässä. Pyydettyään palvelimelta uuden lohkon asiakas joutuu odottelemaan, että palvelin vastaanottaa pyynnön, ja että palvelimen lähettämää tietoa voidaan alkaa vastaanottamaan.

Kaikki tämä odottelu lisää tiedonsiirtoon kuluvaa aikaa. Niinpä olisi hyvä päästä siitä eroon, jotta tiedonsiirto saataisiin optimoitua mahdollisimman tehokkaaksi. Ongelmaan esitetty ratkaisu on nimeltään liukuhihnoitus (pipelining) [Rodriguez and Biersack, 2002]. Liukuhihnoituksessa asiakas pyytää yhden tai useamman uuden lohkon palvelimelta ennen kuin se on saanut edellisen täysin siirrettyä. Tällöin palvelimella on puskurissa pyyntöjä odottamassa vuoroaan. Kun palvelin saa yhden lohkon lähetettyä, niin sen ei tarvitsekaan jäädä odottelemaan seuraavaa pyyntöä, sillä se on jo valmiina odottamassa palvelimen puskurissa. Tällöin palvelin tekee töitä koko tiedonsiirron ajan, eikä turhia odotteluaikoja synny. Näin saadaan taas hiukan optimoitua tiedonsiirtoa.

4. Käytännön toteutus

Rinnakkaistiedonsiirtoa hyödyntäviä sovelluksia on melko vähän. Ainoat kirjoittajan tuntemat sovellukset, jotka hyödyntävät rinnakkaistiedonsiirtoa ovat niin sanotut vertaisverkkoa hyväksikäyttävät sovellukset (peer-to-peer applications). Nämä sovellukset toimivat omina itsenäisinä ohjelminaan siten, että yksittäinen sovellus toimii samalla sekä asiakkaana että palvelimena. Kun sovellus käynnistyy, niin se samalla jakaa käyttäjän valitseman hakemiston siten, että muut käyttäjät voivat sieltä halutessaan siirtää tiedostoja. Ohjelma tarjoaa myös mahdollisuuden etsiä ja siirtää muiden käyttäjien jakamia tiedostoja.

Vertaisverkkoa käyttävät sovellukset eivät hyödynnä Internetin olemassa olevia mahdollisuuksia, vaan toteuttavat kokonaan oman protokollansa. Tämä ei kuitenkaan ollut alun perin rinnakkaistiedonsiirron kehittäjien idea. Ideana oli kehittää Internetin olemassa olevia palveluita hyväksikäyttäen tehokkaampi tiedonsiirtotapa. Tässä luvussa esittelen, millainen sovellus voisi olla.

4.1. Asiakas

Rinnakkaistiedonsiirtoa on viisasta käyttää lähinnä suurien tietomäärien siirtämiseen. Jos ollaan siirtämässä pienehköä määrää tietoa, niin riittää hyvin avata yksi yhteys ja siirtää tieto perinteisellä tavalla.

Asiakas käyttää hyväkseen mahdollisimman pitkälle olemassa olevia tekniikoita. Tiedonvälitykseen käytetään HTTP 1.1 -protokollaa, sillä se mahdollistaa pysyvän yhteyden asiakkaan ja palvelimen välille, jolloin yhteyden avaukseen liittyvää kättelyä ei tarvitse tehdä jokaiselle lohkolle uudestaan. HTTP 1.1 -protokolla mahdollistaa myös tietyn osan pyytämisen

siirrettävästä tiedostosta. Tämä mahdollistaa sen, että asiakas voi pyytää tiettyä lohkoa palvelimelta. FTP-protokolla ei tue näitä seikkoja, joten sen käyttäminen ei tässä tapauksessa tule kysymykseen.

Rinnakkaistiedonsiirto ei välttämättä tarvitse olla käyttäjälle näkyvää. Esimerkiksi selaimiin voitaisiin rakentaa tuki rinnakkaistiedonsiirrolle, joka käyttää hyväkseen peilauspalvelimia. Aina kun siirretään jotain hieman suurempaa tietomäärää, selain voisi automaattisesti tarkistaa onko sama tieto olemassa peilattuna muilla palvelimilla. Jos näin on, niin selain voisi automaattisesti siirtää tiedon useammalta palvelimelta. Käyttäjälle tämä ei tarvitsisi näkyä muuten kuin parantuneena tiedonsiirtonopeutena.

4.2. Palvelin

Koska käytetään olemassa olevia tekniikoita rinnakkaistiedonsiirron toteuttamisen, niin palvelinpuolelle ei itse asiassa tarvitse tehdä mitään muutoksia. Nykyään HTTP 1.1 on standardi tiedonsiirtoprotokolla, jota lähes kaikki palvelimet tukevat. Palvelimet pystyvät perinteisesti palvelemaan useita samanaikaisia yhteyksiä. Itse jaettavaa tietoa ei tarvitse koodata mitenkään kuten käytettäessä multicasting-tyyppistä tiedonjakotapaa. Lisäksi jo nykyään on tavallista, että käytetään tiedon peilausta, joten hyödynnettäviä palvelimia on jo olemassa.

Yhteenvetona voisi todeta, että mahdollisuus palvelinpuolelta katsottuna rinnakkaistiedonsiirron laajamittaisellekin käyttämiselle on jo olemassa. Tarvitaan vain asiakas, joka käyttää hyväkseen tätä mahdollisuutta.

5. Yhteenveto

Tutkimuksessani olen kuvannut rinnakkaistiedonsiirtoa, jonka avulla on mahdollista tehostaa verkosta tehtävää tiedonsiirtoa. Rinnakkaistiedonsiirto eroaa perinteisestä yhdeltä yhdelle -tiedonsiirrosta siten, että rinnakkaistiedonsiirron tapauksessa asiakas siirtääkin haluamansa tiedon usealta palvelimelta samanaikaisesti. Kultakin palvelimelta pyydetään tietty osa, eli lohko, halutusta tiedosta. Kaikki lohkot siirretään sitten rinnakkain, jolloin tiedonsiirtoon kuluva aika pienenee.

Lähdeluettelo

[Byers et al., 1999] John W. Byers, Michael Luby and Michael Mitzenmacher, Accessing multiple mirror sites in parallel: using tornado codes to speed up downloads. In: *Proc. of IEEE INFOCOM 1999*, 275-283.

[Kangasharju et al., 1999] Jussi Kangasharju, Keith W. Ross and James W. Roberts, Locating copies of objects using the domain name system. In: *Proc. of 4th Web Caching Workshop*.

[Miu and Shih, 1999b] Allen Miu and Eugene Shih, Performance analysis of a dynamic parallel downloading scheme from mirror sites throughout the internet. MIT, Laboratory for Computer Science. Term Paper, Dec. 1999. Also available as *http://nms.lcs.mit.edu/~aklmiu/comet/paraload.pdf* [11.10.2002].

[Philopoulos and Maheswaran, 2001] Spiro Philopoulus and Muthucumaru Maheswaran, Experimental study of parallel downloading schemes for internet mirror sites. In: *Proc. of 13th IASTED International Conference*, 44-48.

[RFC 959] IETF: FILE TRANSFER PROTOCOL (FTP). Also available as *http://www.ietf.org/rfc/rfc959.txt* [20.12.2002].

[RFC 2616] IETF: Hypertext Transfer Protocol -- HTTP / 1.1. Also available as *http://www.ietf.org/rfc/rfc2616.txt* [20.12.2002].

[Rodriguez and Biersack, 2002] Pablo Rodriguez and Ernst W. Biersack, Dynamic parallel-access to replicated content in the internet. In: *Proc. of IEEE Transactions on Networking 2002*, 441-455.

[Rodriguez et al., 2000] Pablo Rodriguez, Andreas Kirpal and Ernst W. Biersack, Parallel-access for mirror sites in the internet. In: *Proc. of IEEE INFOCOM 2000*, 864-873.

[Zeitoun et al., 2002] Amgad Zeitoun, Hani Jamjoom and Mohamed El-Gendy, Scalable parallel-access for mirrored servers. In: *Proc. of Twentieth IASTED International Conference*, 351-436.

XML ja kieliopillinen päättely

Jouni Ilomäki

Tiivistelmä

Tässä tutkielmassa käsittelen XML-kieleen ja kieliopilliseen päättelyyn liittyviä asioita. Esittelen XML-kielen ja -kielioppien ominaisuuksia ja kieliopillisen päättelyn hyödyntämistä niiden käsittelyssä. Lisäksi käsittelen hiukan koneoppimista kielioppien päättelemisen muodossa.

Avainsanat ja -sanonnat: XML, kieliopit, kieliopillinen päätely.
CR-luokat: F.4.2

1. Johdanto

XML:stä on lyhyessä ajassa tullut lähes standardi rakenteisen datan tallentamisessa käytettäväksi formaatiksi. XML:n käsittelemiseen on olemassa useita ohjelmia, mutta kieliopillisen päättelyn avulla näihin ohjelmiin saataneen lähitulevaisuudessa uusia ominaisuuksia. Nämä uudet ominaisuudet kuitenkin vaativat edelleen kielioppien tutkimista ja kehittämistä, ennen kuin niitä päästään toteuttamaan kaupallisiin ohjelmiin.

Tässä tutkielmassa esittelen ensiksi joitakin määritelmiä, joita käytän myöhemmässä vaiheessa. Määritelmien jälkeen, kolmannessa luvussa, kerron XML-kielen perusteista ja sen kieliopeista: DTD:stä ja XML Schemasta. Neljännessä luvussa kerron kieliopeista ja kieliopillisesta päättelystä, erityisesti XML-kielioppien kannalta. Kielioppien koneoppimisesta kerron luvussa kuusi.

Tämän tutkimuksen tarkoituksena on selkiyttää tilannetta XML:n ja kieliopillisen päättelyn osalta: mitä on tutkittu, mitä tutkitaan ja mitä kannattaisi tutkia?

2. Määritelmiä

Englanninkielinen termi “tag” tarkoittaa muiden muassa XML:ssä käytettäviä datan rakennetta kuvaavia merkkejä, joilla erotetaan datan eri osat toisistaan. Esimerkiksi pari ”<sandwich>” ja ”</sandwich>” rajoittavat voileivän (sandwich) määrittävän datan. Koska ”tag” sanalle ei suomenkielessä ole

vakiintunutta termiä, käytän tässä tutkielmassa puhekieleen vakiintunutta sanaa ”tagi”.

Kontekstiton kielioppi G määritellään järjestelmänä (N, Σ, P, I), jossa N on *nonterminaaliaakkosto*, Σ on *terminaaliaakkosto*, $P \subseteq N \times (N \cup \Sigma)^*$ on äärellinen *sääntöjen (productions)* joukko ja I on erillinen alkio, *alkumerkki (initial symbol)*. Sääntöjen vasemmat puolet muodostuvat siis aina täsmälleen yhdestä nonterminaalista ja oikean puolen sisältöä ei ole rajoitettu.

Laajennettu kontekstiton kielioppi on kuten ”tavallinen” kontekstiton kielioppi, mutta jokainen sääntö on muotoa $A \rightarrow E_a$, jossa *A* on nonterminaali ja E_a on aakkoston $V=N \cup \Sigma$ merkeistä muodostettu säännöllinen ilmaus, joka ei sisällä tyhjä joukko -symbolia. XML-kieliopit ovat laajennettuja konteksittomia kielioppeja.

Laajennetun kontekstittoman kieliopin kieli L(G) on sellaisten terminaalimerkkijonojen joukko, jotka ovat *saavutettavissa (derivable)* kieliopin G:n aloitusmerkistä *I*. Siis formaalisti $L(G)=\{w \in \Sigma^* \mid I \Rightarrow^+ w\}$, jossa $\Rightarrow^+$ tarkoittaa johtamisrelaation transitiivista sulkeumaa.

3. XML, DTD ja XML Schema

XML on World Wide Web Consortiumin [Bray *et al.*, 2000] suosittama formaatti rakenteisen datan tallentamiseen. XML-kielen syntaktinen (syntactic) osa kuvaa tagien suhteeliset sijainnit dokumentissa. Tämä osa erotetaan yleensä omaksi tiedostoksi: DTD:ksi [Bray *et al.*, 1998] tai XML Schemaksi [Fallside, 2001]. Myös muita formaatteja kieliopin esittämiseen on olemassa, mutta niiden käyttö on minimaalista.

3.1 DTD

DTD muodostetaan tyyppimäärittelyistä (element type definition), joissa oikeat puolet ovat laajennettuja säännöllisiä ilmauksia (extended regular expression) joita kutsutaan sisältömalleiksi (content model). Kuvan 1 esimerkki määrittelee mallin (template) voileivälle, joka koostuu leivän viipaleesta, mahdollisesta levitteestä sen päällä, täytteistä ja vielä mahdollisesti toisesta leivän viipaleesta täytteiden päällä. Elementti `loaf` on joko `sour_bread` tai `white_bread`, ja `spread` on vastaavasti `butter` tai `margarine`. Täyte leivän välissä sisältää luetellussa järjestyksessä minkä tahansa määrän juustoa (`cheese`), mahdollisesti yhden palan makkaraa (`sausage`) ja vähintään yhden viipaleen tomaattia (`tomato`). Viimeinen tyyppimäärittely kertoo `#PCDATA`-sanalla elementit, jotka eivät ole rakenteisia (structured). `#PCDATA` on DTD:n ainoa terminaalisymboli normaalin

kieliopillisen ajattelun mukaan, mutta todellisuudessa se merkitsee kaikkia merkkijonoja, jotka kuuluvat määriteltyyn terminaaliaakkostoon Σ.

```
<!DOCTYPE sandwich[
   <!ELEMENT sandwich (loaf, spread?, filling, loaf?)>
   <!ELEMENT loaf          (sour_bread | white_bread)>
   <!ELEMENT spread   (butter | margarine)>
   <!ELEMENT filling  (cheese*, sausage?, tomato+)>
   <!ELEMENT (sour bread, white bread, butter, margarine,
      cheese, sausage, tomato) (#PCDATA)>
]>
```

Kuva 1. DTD-malli voileivälle.

3.2. XML Schema

Edellistä DTD:tä vastaava XML Schema on huomattavasti pidempi, minkä vuoksi se saataa näyttää monimutkaisemmalta. XML:ää läheisesti muistuttava ulkoasu kuitenkin helpottaa rakenteen ymmärtämistä. DTD:hen verrattuna XML Schema on uudempi ja voimakkaampi formaatti, ja se tekee jo olemassa-olevien rakenteiden uudelleen käyttämisestä kohtuullisen helppoa. Lisäksi sillä on paljon yhteistä olioperustaisten ohjelmointikielten kanssa: se mahdollistaa polymorfisen sisällön, XML Schemat voivat periä toisia schemoja, elementit ja tyypit voidaan määritellä abstrakteiksi ja niin kutsutut *vastaavuusluokat (equivalence classes)* mahdollistavat elementtien korvaamisen toisilla elementeillä. Tyyppimäärittelyt ja elementtien esittelemiset voidaan kapseloida käyttäen nimiavaruuksia (namespaces), joka helpottaa periyttämisen käyttämistä. Kuvassa 2 on esitetty kuvan 1 DTD-mallia vastaava XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
   <xs:element name="butter" type="xs:string"/>
   <xs:element name="cheese" type="xs:string"/>
   <xs:element name="filling">
      <xs:complexType>
      <xs:sequence>
        <xs:element ref="cheese" minOccurs="0"
      maxOccurs="unbounded"/>
        <xs:element ref="sausage" minOccurs="0"/>
        <xs:element ref="tomato" maxOccurs="unbounded"/>
      </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="loaf">
      <xs:complexType>
      <xs:choice>
        <xs:element ref="sour_bread"/>
        <xs:element ref="white_bread"/>
      </xs:choice>
      </xs:complexType>
```

```
   </xs:element>
   <xs:element name="margarine" type="xs:string"/>
   <xs:element name="sandwich">
      <xs:complexType>
       <xs:sequence>
         <xs:element ref="loaf"/>
         <xs:element ref="spread" minOccurs="0"/>
         <xs:element ref="filling"/>
         <xs:element ref="loaf" minOccurs="0"/>
       </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="sausage" type="xs:string"/>
   <xs:element name="sour_bread" type="xs:string"/>
   <xs:element name="spread">
      <xs:complexType>
       <xs:choice>
         <xs:element ref="butter"/>
         <xs:element ref="margarine"/>
       </xs:choice>
      </xs:complexType>
   </xs:element>
   <xs:element name="tomato" type="xs:string"/>
   <xs:element name="white_bread" type="xs:string"/>
</xs:schema>
```

Kuva 2. XML Schema voileivälle.

3.4. XML

Varsinaisen XML-tiedoston ensimmäinen rivi määrittää käytettävän XML:n version ja koodauksen tyypin. Toisella rivillä kerrotaan juurielementti (root element) XML:n rakenteessa, ja annetaan käytettävän DTD:n tai XML Scheman nimi ja mahdollinen osoite. Tiedoston loppuosa pitää sisällään rakenteisen datan, joka on tallettettu kyseiseen XML:ään. XML:ssä sekä alku- että lopputagi pitää olla olemassa, toisin kuin esimerkiksi HTML:ssä. Tästä ainoan poikkeuksen tekee niin sanottu ”tyhjä tagi”, jolla voidaan korvata peräkkäin olevat aloitus- ja lopetustagi. Tyhjä tagi on kuin normaali lopputagi, paitsi että loppua ilmaiseva kauttaviiva on tagin nimen perässä. Siis esimerkiksi `<tomato></tomato>` on sama kuin `<tomato/>`. Koko XML-dokumentti on esitetty kuvassa 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sandwich SYSTEM “sandwich.dtd”>
<sandwich>
   <loaf><sour_bread>Reissumies</sour_bread></loaf>
   <spread><margarine>Flora</margarine></spread>
      <filling>
         <cheese>Olympia</cheese>
         <cheese>Oltermanni</cheese>
         <tomato/>
      </filling>
   <loaf><sour_bread>Reissumies</sour_bread></loaf>
</sandwich>
```

Kuva 3. XML-dokumentti voileivälle.

XML-dokumentit voidaan validoida vain syntaktisessa mielessä, mikä voi aiheuttaa ongelmia, sillä XML:ää käytetään usein semanttisen datan tallentamiseen. Tälläisiä ongelmia voisi tulla esimerkiksi, jos haluttaisiin, että edellisessä esimerkissä käsitelty voileipä saisi sisältää enintään viisi täyte-yksikköä. Joidenkin täytteiden määrä voidaan kyllä rajoittaa, mutta XML Schemassa tai DTD:ssä ei ole keinoa, jolla täytteiden määrää voitaisiin laskea yhteen. Sama ongelma semanttisuuden kanssa kohdataan, jos halutaan että voileivän pohja- ja kansileipä ovat samankaltaisia.

Attribuuttikieliopit (attribute grammars) ovat yksi mahdollinen tapa korjata semantiikan puuttumisesta aiheutuvia ongelmia XML:ssä. Attributtikielioppeja voidaan käyttää määrittämällä joukko attributteja kullekin XML:n rakenteelle (substructure) ja joukko sääntöjä niille kentille, jotka riippuvat attribuuttien arvoista. Attribuuttikielioppien hyödyntäminen XML-dokumenttien yhteydessä on vasta tutkimuksen alla, joten niiden hyödyntäminen käytännössä ei vielä onnistu.

4. Kieliopit ja kieliopillinen päättely

4.1. XML-kieliopit

XML-kielioppiin kuuluu alkutagit sisältävä joukko A ja niitä vastaavat lopputagit sisältävä joukko $\overline{A}$. Koska kieliopillisessa päättelyssä keskitytään pääasiassa syntaktiseen rakenteeseen, jätetään XML:n sisällä oleva teksti yleensä huomiotta. Siis XML-dokumentti ilman tekstiä on aakkoston $T = A \cup \overline{A}$ merkeistä muodostettu sana. Berstelin ja Boassonin [Berstel and Boasson, 2000] mukaan XML-kielioppi muodostuu:

1. joukosta *muuttujia* (*variables*) $V=\{X_a \mid \mathrm{a} \in \mathrm{A}\}$,
2. *terminaaliaakkostosta* $T=A \cup \overline{A}$, jossa $\overline{A} = \{\overline{a} \mid a \in A\}$,
3. merkkeihin $\mathrm{a} \in \mathrm{A}$ liittyvistä säännöllistä joukoista $R_a \subseteq V^*$, jotka määrittelevät (mahdollisesti äärettömän) joukon sääntöjä $X_a \to am\overline{a}$, jossa $\mathrm{m} \in R_a$ ja $\mathrm{a} \in A$. Voidaan kirjoittaa myös lyhyesti $X_a \to aR_a\overline{a}$, sekä
4. erityisestä alkumerkki-muuttujasta jota kutsutaan *aksioomaksi*.

Kontekstittomissa kieliopeissa olevilla nonterminaaleilla voi olla äärettömiä säännöllisiä sääntöjä. Siltikin on yleisesti tiedossa, että generoitunut kieli on kontekstiton ja deterministinen. Esimerkiksi aikaisemmin esittetty, voileipää määrittävä DTD, voitaisiin merkitä taulukossa 1 esitetyllä tavalla.

muuttuja	muuttujaa vastaava tagi-pari
X_s	<sandwich></sandwich>
X_l	<loaf></loaf>
X_d	<spread></spread>
X_f	<filling></filling>
X_r	<sour_bread></sour_bread>
X_w	<white_bread></white_bread>
X_b	<butter></butter>
X_m	<margarine></margarine>
X_c	<cheese></cheese>
X_a	<sausage></saudage>
X_t	<tomato></tomato>

Taulukko 1. Kuvan 1 DTD:tä vastaavan XML-kieliopin muuttujat.

Taulukkoon 1 liittyvä XML-kielioppi on kuvassa 3.

$X_s \to sR_s\bar{s}$, jossa
$\mathrm{R_s} = \left\{X_l X_d^i X_f X_l^j \mid 0 \le i \le 1, 0 \le j \le 1\right\}$,
$X_l \to lR_l\bar{l}$, jossa
$\mathrm{R_l} = \left\{X_r^i X_w^j \mid i = 0, j = 1 \, tai \, i = 1, j = 0\right\}$,
$X_r \to r\tau\bar{r}$,
$X_w \to w\tau\bar{w}$,
$X_d \to dR_d\bar{d}$, jossa
$\mathrm{R_d} = \left\{X_b^i X_m^j \mid i = 0, j = 1 \, tai \, i = 1, j = 0\right\}$,
$X_b \to b\tau\bar{b}$,
$X_m \to m\tau\bar{m}$,
$X_f \to fR_f\bar{f}$, jossa
$\mathrm{R_f} = \left\{X_c^i X_a^j X_t^k \mid j \le 1, k \ge 1\right\}$,
$X_c \to c\tau\bar{c}$,
$X_a \to a\tau\bar{a}$,
$X_t \to t\tau\bar{t}$,

Kuva 3. Kuvan 1 DTD:tä vastaava XML-kielioppi.

Kuvassa 3 X_S on aksiooma ja τ kuvaa mielivaltaista dataa. DTD:ssä kieliopin aksiooma esitetään sanan DOCTYPE perässä ja kieliopin säännöt ELEMENT-sanojen jälkeen. Jokainen elementti muodostuu *tyypistä (type)* ja *sisältömallista (content model)*. Tässä tapauksessa siis tyypit ovat sandwich, loaf, spread, filling, sour_bread, white_bread, butter, margarine, cheese, sausage ja tomato. Vastaavastu sisältömallit ovat säännölliset ilmaukset (loaf, spread?, filling, loaf?), (sour_bread | white_bread), (butter | margarine) sekä (cheese*, sausage?, tomato+). Lisäksi DTD:en kuuluu #PCDATA-sanalla merkitty data, joka tulee XML:ssä tagien väliin, mutta niiden käsitteleminen ei ole yleensä olennaista XML-kielioppien kannalta.

4.2. Kieliopillisen päättelyn merkitys XML:lle

Kieliopillisella päättelyllä voi tulevaisuudessa olla suurikin merkitys XML:n käytön kannalta. Päätteleminen on melko helppo sekoittaa *jäsentämiseen (parsing)*. Jäsentämisalgoritmissa syötemerkkijonon kielioppi on tiedossa, kun taas päättelyalgoritmi yrittää selvittää kyseisen kieliopin. Päättelemisellä voidaan esimerkiksi yhdistää useampi XML Schema niin, että tuloksena saatua kielioppia voidaan käyttää alkuperäisiä kielioppeja noudattaville XML-dokumenteille.

Näkymien luominen XML-dokumenteista voi olla hyödyllistä, jos yhdestä laajasta DTD:stä käytetään vain pientä osaa. Yleistä DTD:tä pystytään yksinkertaistamaan niin, että se muutetaan määrittämään vain tiettyjä dokumentteja. Käsin operaatio on varsin työläs, mutta päättelyn avulla voidaan luoda näkymiä ja alidokumentteja (subdocuments) automaattisesti.

XML dokumenttien automaattinen muokkaaminen olisi myös hyödyllinen ominaisuus, joka kieliopillisen päättelemisen uskotaan tuovan XML-työkaluihin. Tälläisen ohjelma tarpeellisuus huomataan, jos muutetaan XML Schemaa, jonka pohjalta on tehty useita XML-dokumentteja. Muokkausohjelma voisi ottaa syötteenä alkuperäisen ja muokatun XML Scheman, joista se generoisi säännöt, joiden mukaan XML-dokumentteja muokataan. Sen jälkeen näitä sääntöjä sovelletaan kaikkiin vanhan XML Scheman mukaisiin XML-dokumentteihin, ja ohjelma generoi uudet, vastaavat XML-dokumentit.

XML-dokumentin rakenteen ja sisällön erottamista on myös tutkittu. Tarkoituksena on selvittää, kuinka useilla datapopulaatioilla on samanlainen rakenne ja missä tilanteissa useita rakenteita voidaan voidaan käyttää samalla datajoukolle. Dokumenttien konvertoiminen alustalta (platform) toiselle voitaisiin automatisoida vastaavaa tekniikkaa hyväksikäyttäen.

Kontekstien suunnitteleminen päättelemällä on myös mahdollista. Esimerkiksi niin kutsuttu *caterpillar-ilmaus (*ja *-automaatti)* [Brüggemann-Klein *et al.*, 1998] voi muodostaa konteksteja XML-kielioppeihin. Catepillar-automaatti on äärellinen automaatti, joka ottaa syötteenä sanoja ja tulkitsee niitä komennoiksi, joiden mukaan käy läpi oletettua syntaksipuuta.

4.3. XML-kielten kaksi luonnehdintaa

Aakkoston $\{a, \bar{a}\}$ merkeistä muodostettu *Dyckin kieli (Dyck Prime)* on kieliopista $X \to aX^*\bar{a}$ generoitu XML-kieli, jota merkitään D_a [Berstel and Boasson, 2000]. *Tekijöiden* (factor) eli osasanojen joukkoa kielessä L merkitään *F(L),* ja $F_a(L) = D_a \cap F(L)$, jokaiselle merkille $a \in A$. Joukko $F_a(L)$ on *hyvin muodostettujen (well formed)* sanojen joukko. Nämä sanat kuuluvat Dyckin kieleen ja alkavat merkillä a. Esimerkiksi kielelle $L = \{a(b\bar{b})^n(c\bar{c})^n\bar{a} \mid n \geq 1\}$ on voimassa $F_a(L) = L$, $F_b(L) = \{b\bar{b}\}$ ja $F_c(L) = \{c\bar{c}\}$. Siis $L_G(X_a) = F_a(L)$.

Jokaiselle XML-kielelle $L \subset D_a$ on voimassa $F_a(L) = L$. Kuulukoon sana *w* Dyckin kieleen D_a, jolloin sana *w* voidaan jakaa tekijöihin siten että $w = au_{aq}u_{a2}\ldots u_{an}\bar{a}$, jossa $u_{ai} \in D_a$, kun *i=1,…,n*. Sanan *w jäljeksi (trace)* kutsutaan sanaa $a_1a_2\ldots a_n \in A^*$.

Jos *L* on *D*:n osajoukko ja $w \in L$, niin sanat u_{ai} kuuluvat joukkoon $F_{ai}(L)$. Kaikkien jälkien joukkoa sanoille $F_a(L)$:ssä kutsutaan *pinnaksi* $S_a(L)$.

XML-kieli on jonkun XML-kieliopin generoima kieli. Berstel ja Boasson ovat esittäneet kaksi eri luonnehdintaa XML-kielille [Berstel and Boasson, 2000]. Näitä luonnehdintoja voidaan käyttää hyväksi koneoppimisessa. Toinen vaihtoehto perustuu pintoihin, ja toinen on syntaktinen ja perustuu kontekstin käsitteeseen.

- Normaali (standard) kieli, joka on on yhdistetty pintaan *S*, on maksimaalinen elementti perheessä *L(S)*. Tämä kieli on ainoa XML-kieli perheessä *L(S)*.
- Joukossa $A \cup \bar{A}$ määritelty kieli L on XML-kieli jos ja vain jos
  1. $L \subset D_\alpha$, jollekin $\alpha \in A$,
  2. kaikille $a \in A$ ja $w, w' \in F_a(L)$ on voimassa $C_L(w) = C_L(w')$ sekä
  3. joukko $S_a(L)$ on säännöllinen kaikille $a \in A$.

5. Kieliopin oppiminen

XML-dokumentti voi olla käyttökelpoinen vaikka sen XML Schema (tai DTD) puuttuisi. XML:n kieliopin tuntemisesta on kuitenkin hyötyä esimerkiksi [Min *et al., 2002*] 1) XML-dokumenttien rakenteen kuvaamisessa ja muokkaa-

misessa, 2) XML-dokumenttien muokkaamisessa ja 3) XML-kyselyiden suorittamisessa.

XML-kieliopin generointiohjelmia on kehitetty muutamia, kuten XTRACT [Garofalakis *et al.*, 2000], DdbE [Berman, 2001] sekä DTD-Miner [Moh *et al.*, 2000]. Min, Ahn ja Chung vertailivat näitä ohjelmia [Min *et al.*, 2002] samalla kun esittelivät oman metodinsa oppimiseen. He rajoittavat XML Scheman muotoa ja käyttävät heuristisia sääntöjä, ja sillä tavoin nopeuttavat oppimista moninkertaisesti muihin metodeihin verrattuna.

Jos scheman oppimista yleistetään kaikkien kontekstittomien kielioppien oppimiseen, saadaan käyttökohteita esimerkiksi mallin (pattern) tunnistamisesta ja puheentunnistamisesta [Lee, 1996]. Yksi lähestysmistapa mallintunnistamiseen on sellaisen kontekstittoman kieliopin päätteleminen, joka tuottaa halutun joukon malleja. Jos luonnolliselle kielelle pystyttäisiin päättelemään kontekstiton kielippi, Horningin mukaan voitaisiin tunnistajan sisäistä kielioppia muuttaa kullekin puhujalle sopivaksi [Horning, 1969].

Kielen oppiminen voidaan myös ymmärtää useammalla tavalla. Suurin osa kieliopillisen päättelyn tutkijoista on keskittynyt *identification in the limit* -tyyliseen oppimiseen [Gold , 1967]. Tälläisessä oppimisessa oppimisalgoritmi saa kullakin ajan hetkellä *t* yhden tiedon i_t opittavasta kielestä, ja palauttaa hypoteesin $H(i_1,...,i_t)$. Algortitmi on toimiva, jos se palauttaa jostakin äärellisestä ajasta alkaen pelkästään yhtä, oikeaa hypoteesia. Toinen yleinen oppimiskriteeri on *exact identification using queries in polynomial time* [Angluin, 1988]. Tässä mallissa ideana on, että oppimisalgoritmilla on niin sanottu oraakkeli käytössä. Oraakkeli vastaa algoritmin kysymyksiin kuten: ”Kuuluuko tämä joukko kielioppiin G?”. Käytännön sovelluksissa oraakkeli on yleensä ihminen. Myös tässä oppimistavassa tarkoituksena on tuottaa oikea kielioppi polynomisessa ajassa.

5.1. Erilaisia oraakkelimenetelmiä

Gold on osoittanut että säännöllisiä (eikä siis myöskään kontekstittomia) kieliä ei voi oppia pelkästään positiivisista näytteistä [Gold, 1967]. Erilaisia oraakkeleita käyttäviä menetelmiä onkin esitetty ruunsaasti. Solomonoff [Solomonoff, 1959] käytti metodia, jossa oppimisalgoritmille annetaan syötteenä kieleen *L* kuuluva positiivinen esimerkki S^+, ja jossa algoritmilla on käytössään oraakkeli, joka kertoo, kuuluuko kuuluuko annettu merkkijono kieleen *L*. Kustakin merkkijonosta $w \in S^+$ poistetaan osamerkkijonoja yksi kerrallaan, ja siten luodaan uusia merkkijonoja *w’*. Tämän jälkeen oraakkelilta kysytään, kuuluuko *w’* kieleen *L*. Jos *w’* kuuluu kieleen, lisätää *w’*:uun useita

kertoja aikaisemmin poistettu merkkijono, ja kysytään oraakkelita muodostuneen merkkijonon kuulumisesta kieleen. Jos sekin kuuluu *L*:ään, voidaan päätellä että kieliopissa on rekursiivinen sääntö, kuten $A \rightarrow aAb$.

On olemassa muitakin tekstistä oppimisesta tehtäviä tutkimuksia, esimerkiksi Tanatsugu on esittänyt algoritmin, joka ottaa syötteekseen sekä positiivisia että negatiivisia esimerkkejä [Tanatsugu, 1987]. Tässä algoritmissa ajatuksena on poistaa *itseensäuppoutuneet (self-embending)* rakenteet äärellisestä esimerkistä, päätellä lineaarinen kielioppi esimerkistä, ja lopulta laatia saaduista lineaarisista kieliopeista kontekstiton kielioppi. Tämä algoritmi pystyy päättelemään kaikki kontekstittomat kieliopit, mutta aikavaatimus on erittäin suuri.

5.2. Rakenteesta oppiminen

Sulkukieliopilla (parenthesis grammar) voidaan generoida merkkijonoja, jotka kuvaavat rakenteellista dataa. Kullekin kontekstittomalle kieliopille *G* voidaan luoda vastaava sulkukielioppi *(G)* korvaamalla jokainen sääntö $A \rightarrow \alpha$ säännöllä $A \rightarrow (\alpha)$. Crespi-Reghizzi on esittänyt [Crespi-Reghizzi, 1971] algoritmin, joka tunnistaa *operaattoripresedenssikieliopit* (*operator precedence grammars) in the limit* positiivisesta *sulkeistetusta (parenthesised)* kieliopista.

Rakenteellinen data voidaan esitää *rakenteina (skeletons)*, jotka ovat derivaatiopuita, joista on poistettu nonterminaalien *tunnukset (labels)* [Levy and Joshi, 1978]. Näiden rakenteiden erityisominaisuus on, että ne ovat täsmälleen joukkoja, jotka *rakennepuuautomaatti* (*skeletal tree automata*) hyväksyy. Rakennepuuautomaatti on äärellisen automaatin variaatio.

Koska derivaatiopuiden rakenteet voidaan kuvata automaattina, voidaan kontekstittomien kielioppien oppiminen voidaan redusoida rakennepuuautomaattien oppimiseen. Sakakibara laajentaa Angluinin metodin äärellisten automaattien oppimisesta rakennepuuautomaattien oppimiseen polynomisessa ajassa [Sakakibara, 1992]. Tämä metodi tosin vaatii, että on mahdollista kysyä rakenteellisia jäsenyyskysymyksiä (structural membership queries) ja rakenteellisia vastaavuuskysymyksiä (structural equivalence queries). Sakakibara on myös esittänyt metodin ns. kääntyvien kontekstittomien kielioppien päättelemiseen *in the limit* pelkästään positiivisesta datasta.

5.3. Muut oppimistavat

Oraakkeleita hyödyntävän- ja datan rakenteesta oppimisen lisäksi kontekstittomien kielioppien oppimista on tutkittu tavoilla, joissa niiden esitystapa ei perustu kielioppeihin. Yokomori on esittänyt algoritmin, joka oppii konteks-

tittomia ilmauksia [Yokomori, 1988] ja Arikawa, Shinohara ja Yamamoto [Arikawa et al., 1992] tavan *alkeellisten säännöllisten formaalien järjestelmien (regular elementary formal systems)* päättelemiseen. Kumpikaan näistä algoritmeista ei kuitenkaan toimi polynomisessa ajassa.

5.4. Osajoukkojen oppiminen

Koska kaikkia kontekstittomia kielioppeja ei kyetä oppimaan, on pyritty löytämään sellaisia kontekstittomien kielioppien aliryhmiä, joiden päättely olisi mahdollista. Tälläisiä aliryhmiä ovat ainakin k-rajoitetut-, yksinkertaiset deterministiset-, säännölliset lineaariset- (even linear-), rakenteiset kääntyvät-, one-counter-, kääntyvät- (pivot)- ja erittäin yksinkertaiset kielet.

5.4.1. k-rajoitetut kontekstittomat kieliopit

k-rajoitetut (k-bounded) kontekstittomat kieliopit ovat tunnistettavissa (identifiable) polynomisessa ajassa, kun käytettävissä on yhtäsuuruuskyselyt (eqivalence) ja nontermiaalin kuuluvuus -kyselyt (nonterminal membership). Nontelminaalin kuuluvuus -kyselyssä oraakkelille annetaan merkkijono *w* ja nonteerminaali *A*. Oraakkeli palauttaa arvon tosi, jos *w* on tuottevissa *A*:sta ja muuten arvon epätosi.

5.4.2. Yksinkertaiset deterministinstiset kielet

Ishizaka käyttää Angluinin algoritmia yksinkertaisille deterministille kielille (SDL) siten, että nonterminaalin kuuluvuus –kyselyitä ei tarvitse tehdä [Ishizaki, 1990]. Sensijaan algoritmi tekee *laajennettuja yhtäsuuruuskyselyitä* (*extended equvalence)*, joissa verrattavan kieliopin G ei tarvitse olla yksinkertainen deterministinen kieli. Laajennetut yhtäsuuruuskyselyt eivät ole yhtä voimakkaita kuin nonterminaalin kuuluvuus -kyselyt, sillä ne eivät suoranaisesti tuota rakenteellista informaatiota kieliopista. Yokomori on esittänyt polynomisessa ajassa ratkeavan algoritmin SDL:en oppimiseen [Yokomori, 1988]. Kyseinen algoritmi vaatii *etuliitekuuluvuusoraakkelin (prefix memebership oracle)* käyttöä, jolta kysytään on merkkijono minkään merkkijonon etuliite opittavassa kielessä L_*.

5.5. Esimerkki kieliopin oppimisesta

Fernau esitti yhden tavan kieliopin päättelemiseen [Fernau 2001]. Seuraavassa esimerkki Fernaun oppimisskenaariosta.

$\mathcal{L}$ on *kieliluokka (language class)*, joka määritellään *laitteilla (devices)* $\mathcal{D}$, kuten kieliopeilla tai automaateilla. $\mathcal{L}$ on pääteltävissä, jos on olemassa *päättelykone (inference machine)* IM, jolla voidaan luetella kaikki syötteenä annettavat kielet

$L \in \mathcal{L}$ mielivaltaisessa järjestyksessä. Yleensä käsitellään äärellistä esimerkkijoukkoa $I_+ = \{w_1,..., w_M\}$ ja siitä muodostettua hypoteesia $\mathcal{D}_M$. Tarkoituksena on löytää sellaisen hypoteesin tuottava algoritmi, että hypoteesi kuvaa kielen $L_M \supseteq I_+$, siten että L_M on pienin I_+:n sisältävä kieli kieliluokassa $\mathcal{L}$.

Fernau on aikaisemmassa artikkelissaan [Fernau, 2000] esittänyt, että on olemassa algoritmi jolle annetaan syötteenä äärellinen esimerkkijoukko I_+, ja joka palauttaa hypoteesin $\mathcal{A}$. Kieli, jonka $\mathcal{A}$ tunnistaa, on pienin f–erityinen kieli, joka sisältää joukon I_+. Algortimin periaate on seuraava:

1. Ensin luodaan erityinen (distinguish) funktio f, joka muodostaa *harhan* (*bias*) oppimisalgoritmille.
2. XML-dokumentti muunnetaan joukoksi esimerkkejä, siten että on yksi esimerkkijoukko I^a_+ kutakin opeteltavaa *pintaa (surface)* kohti.
3. Jokainen I^a_+ annetaan syötteenä f–erityiset kielet tunnistavalle algoritmille, ja tulokseksi saadaan säännölliset f–erityiset kielet sisältävä *perhe (family)* $\mathcal{S} = \{S_a \mid a \in A\}$.
4. Palautetaan saatu XML kielioppi.

Soveltetaan esitettyä metodia ottamalla esimerkiksi kirjakauppa, joka haluaa tehdä hinnaston XML-formaatissa. Yhden kirjan tiedot voisivat olla kuvassa 4 esitettyä muotoa.

```
<book>
   <author><last-name>Abiteboul</last-name></author>
   <author><last-name>Vercoustre</last-name></author>
   <title>Research and Advanced Technology for Digital
      Libraries. Third European Conference</title>
   <price>56.24 Euros</price>
</book>
```

Kuva 4. Esimerkki kirjatietojen esittämisestä.

Oletetaan että $f : \Sigma \rightarrow F$, $|F| = 1$, eli *erityinen (distinguishing)* f funktio vastaa Angluinin [Angluin, 1982] mukaan 0-kääntyviä kieliä. Merkitään kieliopin nonterminaaleja taulukon 2 mukaisesti.

muuttuja	muuttujaa vastaava tagi-pari
X_a	<author></author>
X_b	<book></book>
X_n	<last-name></last-name>
X_p	<price></price>
X_t	<title><title>

Taulukko 2. Kirjakauppaesimerkin kieliopin muuttujat.

Lisäksi merkitään X_y:n kuuluvaa tagi-paria y:llä ja $\bar{y}$:llä, joten ylläoleva esimerkki voidaan kirjoittaa muodossa: $w = b\overline{anna}\,\overline{anna}\,\overline{tt}\,\overline{pp}\,\bar{b}$. Tässä ohitamme mielivaltaisen datan tagien välillä. Lause w kuuluu kieleen D_b. Siis voidaan sanoa että $w = bu_a u_a u_t u_p \bar{b}$, jossa $u_a = \overline{anna} \in D_a$, $u_t = \overline{tt} \in D_t$ ja $u_p = \overline{pp} \in D_p$. Sanaan w kuuluva *jälki* on siis aatp. Määritelmän mukaan aatp kuuluu pintaan S_b, joka siis pitäisi oppia. Kuvassa 5 on toinen esimerkkitapaus kirjan tiedot sisältävästä XML:stä.

```
<book>
   <author><last-name>Thalheim </last-name></author>
   <title>Entity-Relationship Medeling. Foundations of Database
      Technology</title>
   <price>50.10 Euros</price>
</book>
```

Kuva 5. Toinen esimerkki kirjatietojen esittämisestä.

Tästä esimerkistä voidaan samaan tapaan päätellä että myös atp kuuluu pintaan S_b. Päättelyalgoritmi 0-kääntyville kielille tuottaa nyt hypoteesin $S_b = a^+tp$, joka osoittautuu järkeväksi yleistykseksi kirjakaupan tapaukseen; kaikille kirjoille halutaan antaa nimi, hinta sekä vähintään yksi kirjoittaja. Liittämällä mielivaltaista dataa (#PCDATA) kuvavan merkin τ tagien väliin, saadaan pääteltyä kuvan 6 XML-kielioppi.

$X_b \to bR_b\bar{b}$, jossa
$R_b = \{X_a^j X_t X_p \mid j > 0\}$,
$X_a \to aR_a\bar{a}$, jossa
$R_a = \{X_n\}$,
$X_n \to n\tau\bar{n}$,
$X_t \to t\tau\bar{t}$,
$X_p \to p\tau\bar{p}$.

Kuva 6. Kirjakauppaesimerkin XML-kielioppi.

6. Yhteenveto

DTD ja XML Schema ovat XML:n kielioppeja, joista jälkimmäinen on uudempi, ja näyttäisi että se tulee syrjäyttämään DTD:n lähes kokonaan. XML Schema on vähän DTD:tä kehittyneempi, ja se mahdollistaa tehokkaamman uudelleenkäytettävyyden.

Kieliopillisella päättelyllä voidaan muokata XML-dokumentteja sekä sen kielioppeja koneellisesti. Muokkaaminen voi pitää sisällään esimerkiksi

kahden kieliopin yhdistämisen tai XML-dokumenttien muuttamisen muutetun kieliopin mukaiseksi. Lisäksi dokumenttien rakenteen ja sisällön erottaminen tulee luultavasti mahdolliseksi jossain XML:n elämän vaiheessa.

Usein hävinneet, salaiset tai muuten tuntemattomaksi jääneet XML-dokumenttien kieliopit haittaavat dokumenttien käsittelyä. Tiedossa oleva XML Schema voi esimerkiksi nopeuttaa hakuja XML:ään perustuvissa tietokannoissa tai helpottaa XML:n muokkaamista. Näitä puutteellisia tapauksia varten on kehitetty erilaisia keinoja päätellä kielioppi pelkän XML:n perusteella. Vaikka kaikkien kontekstittomien kielten oppiminen ei ole mahdollista, on kohtuullisen hyviä oppimismetodeita XML-kieliopeille jo olemassa.

Viiteluettelo

[Angluin, 1982] Dana Angluin, Inference of reversible languages. *Journal of the ACM* **29,** 3 (1982), 741-764.

[Angluin, 1988] Dana Angluin, Queries and concept learning, *Mach. Learning* **2,** 4 (1988), 319-342 .

[Arikawa et al., 1992] Setsuo Arikawa, Takeshi Shinohara, and Akihiro Yamamoto, Learning elementary formal systems, *Theoret. Comput. Sci.* **2,** 1 (1992), 91-113.

[Berman, 2001] Leonard Berman and Angel Diaz, Data Descriptors by Example (DDbE), IBM alphaworks, http://www.alphaworks.ibm.com/tech/DDbE, 2001.

[Berstel and Boasson, 2000] Jean Berstel and Luc Boasson, XML grammars. In N. Nielsen and B. Rovan (eds.), *Mathematical Foundations of Computer Scince (MFCS'2000)* **1893** (2000) Springer, LNCS, 182-191. Long Version as Technical Report IGM 2000-06, *www-igm.univ-mlv.fr/~berstel/Recherche.html.*

[Bray *et al.,* 1998] Tim Bray, Charles Frankston and Ashok Malhatro, Document Content Description for XML, W3C submission, http://www.w3.org/TR/ NOTE-dcd, 1998.

[Bray *et al.*, 2000] Tim Bray, Jean Paoli and C. M. Sperberg-McQueen, Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, http://www.w3.org/TR/REC-xml, 2000.

[Brüggemann-Klein *et al.,* 1998] Anne Brüggemann-Klein, Stefan Herrmann and Derick Wood. Context and catepillars and structured documents. In E. V. Munson, C. Nicholas, and D. Wood (eds.), *Principles of Digital*

Document Processing; 4th International Workshop (PoDDP'98) **1481** (1998) Springer, LNCS, 1-9.

[Crespi-Reghizzi, 1971] Stefano Crespi-Reghizzi, An effective model for grammar inference, *Inf. Proc.* **71** (1971), 524-529.

[Fallside, 2001] David C. Fallside, XML Schema. Part 0, W3C Recommendation, *http://www.w3.org/TR/xmlschema-0*, 2001

[Fernau, 2000] Henning Fernau. Identification of function distinguishable languages. In H. Arimura, S. Jain, and A. Sharma (eds.), *Proceedings of the 11th International Conference Algorihmic Learning Theory ALT, Lecture Notes in Artificial Intelligence* **1968** (2000) Springer, 116-130.

[Fernau, 2001] Henning Fernau. Learning XML grammars, *in Proc. of the 2nd Machine Learning and Data Mining in Pattern Recognition (MLDM'01), Lecture Notes in Artificial Intelligence* **2123** (2001), 73-87.

[Garofalakis *et al.*, 2000] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, Kyuseok Shim, XTRACT: A system for extracting document type descriptions from XML documents, in: *Proceeding of ACM SIGMOD*, 2000.

[Gold, 1967] E. Mark Gold. Language identification in the limit. *Inf. Contr.* **10** (1967), 447-474.

[Horning, 1969] James Jay Horning, A study of grammatical inference. University of Stanford, Computer Science Department, Report **CS 139,** 1969.

[Ishizaki, 1990] Hiroki Ishizaka, Polynomial time learnability of sample deterministic languages, *Mach. Learning* **2,** 2 (1990), 151-164.

[Lee, 1996] Lillian Lee, Learning of context-free languages: A survey to the literature. Harvard University, Center for Research in Computing Technology, Report **TR-12-96**, 1996.

[Levy and Joshi, 1978] Leon S. Levy and Aravind K. Joshi, Skeletal structural discipions, *Inf. Contr.* **2**, 2 (1978), 192-211.

[Min *et al.*, 2002] Jun-Ki Min, Jae-Yong Ahn and Chin-Wan Chung, Efficient extraction of schemas for XML documents, *Inf. Proc. Lett.* **85** (2002), 7-12.

[Moh *et al.*, 2000] Chuang-Hue Moh, Ee-Peng Lim, W. Wee Keong Ng, DTD-miner: A tool for mining DTD from XML documents, in: *Proceeding of International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS)*, 2000.

[Sakakibara, 1992] Yasubumi Sakakibara, Learning context-free grammars from structural data in polynomial time, *Theoret. Comp. Sci.* **76** (1992), 223-242.

[Solomoff, 1964] Ray J. Solomonoff, A formal theory of inductive inference, *Inf. Contr.* **7** (1964), 1-22 and 224-254.

[Tanatsugu, 1987] Keisuke Tanatsugu, A grammatical inference for context-free languages based on self-embending, *Bull. Informatics and Cybernetics* **2**, 3-4 (1987), 149-163.

[Yokomori, 1988] Takashi Yokomori, Learning simple languages in polynomial time, Japanese Society for AI, Report **SIG-FAI**, 1988.

Human-Computer Interaction Challenges of Nomadic Computing

Natalie Jhaveri

Abstract

As our dependency on computer devices mounts, so does the number of nomadic users. People now expect to have constant access to information, regardless of their current location and position. This expectation of nomadic users brings about many human-computer interaction issues. The objective of this research is to determine the human-computer interaction challenges associated with nomadic computing. By conducting a literature survey, it was found that in addition to the typical human-computer interaction concerns of computing technology, nomadic computing introduces a new set of challenges. For nomadic users, additional issues such as, consistency and uniformity of the interaction, interactivity under conditions of distraction, and appropriateness of the interaction method for the circumstance, must also be accounted for.

Keywords: nomadic computing, human-computer interaction, usability, new interaction techniques.
CR-classification: H.1.2

1. Introduction

Nomadic computing aims to support the information technology user's desire to relocate effortlessly. This new computing paradigm, which was introduced only less than a decade ago, has attracted significant consideration. Much effort has gone into defining, researching, designing and technically achieving the conditions required for a true nomadic computing environment. However, the nomadic computing environment also brings about a new set of human-computer interaction issues which must be addressed.

The conventional computing environment has been that of a stationary desktop computer which is physically connected through a network to a server that is located somewhere in the same building [Kleinrock, 1995]. In this environment users are rather familiar with the common software programs, input devices, and ergonomic positions. In contrast, the nomadic computing environment brings about a very different type of interaction

experience for the user. A nomadic user may be in motion while interacting with the device. A nomadic user may only be able to give the computational task part of his or her attention during circumstances such as driving. The nomadic user may have only one or even no hands available for interacting with the device.

It is the advancement of computing devices such as laptops, PDA's, and mobile phones, which has enabled nomadic computing. These advancements have helped to break the initial barriers which included: high cost, large size, high power consumption, and limited interfaces. As the use of such devices becomes more and more widespread, so does the need for the support of nomadic computing. From a system support point of view, the desirable conditions for nomadic computing include the independence of location, of motion, of platform and with widespread presence of access to remote files, systems and services [Kleinrock, 1995].

Considering the needs of a nomadic user, computational devices are more limited in the way of size, weight, power, network access, information management, and data entry. In such a limiting computing environment human-computer interaction factors such as usability become all the more important.

Up to date much research has been done with the goal of improving the interaction between humans and computers. For this next stage of interaction between humans and computers, whereby we are trying to achieve a true nomadic, ubiquitous, and pervasive computing environment, the systems will be practically invisible and the user interfaces will be more intuitive [Ark et al., 1999]. In these computing environments it is believed that interactions with the computer will be less like the current mouse, keyboard, and display paradigm. Instead interactions will be those which are more natural to humans, such as speech, gestures, writing, and the alteration of physical objects. [Abowd et al., 2002]

The aim of this research is to reveal the key human-computer interaction challenges associated with nomadic computing. By means of an in-depth literature survey, these challenges have been identified and presented in this paper. As well, the support needed by the nomadic user and the related new interaction techniques are also discussed.

2. Nomadic Computing

In this chapter, the concept of nomadic computing is explained in further detail. First the history of this paradigm is discussed and then its current state

is presented. As well, details of nomadic users, the relationship of nomadic computing to ubiquitous and pervasive computing, and the design limitations imposed by nomadic computing, are all presented.

Nomadic computing is a concept which was first introduced in 1995 as a vision of Leonard Kleinrock, the chairman of the UCLA computer science department. He established that "nomadicity refers the to the system support needed to provide a rich set of capabilities and services to the nomad as he moves from place to place in a transparent and convenient form" [Kleinrock, 1995].

It must be noted that in the nomadic computing paradigm, the user is not bound to continuous computing and information access. Instead, the importance is that it is readily available whenever needed. Therefore, although mobile devices are useful for nomadic users, they are not essential. In fact, a move from the desktop computer in one's office to another computer in a conference room even constitutes a nomadic move. In this case, the computing platforms and communications capability may considerably differ between the two locations but this should not obstruct the user's need to access his or her work. [Kleinrock, 1995]

Achieving nomadic computing means meeting the conditions whereby there is independence of location, motion, and platform, all while maintaining the widespread presence of access to remote files, systems and services. Therefore, a nomadic user should be able to leave his or her office for a new location, and still easily connect to the Internet and corporate network [Kleinrock, 2000].

At the time when the concept of nomadicity was first envisioned, the system support needed to fulfil its conditions did not fully exist. Since then significant efforts have been made to create the computing environment and to provide such communications that facilitate nomadic computing, and thereby actualize this vision of nomadicity. Now, as we can access the company intranet from an external location, as we can use our mobile phone to connect to the internet, and as we can perform such tasks with little technical or administrative ability, nomadicity has in many ways been finally achieved.

2.1. Attending to the Nomadic User

The main idea of nomadicity from the user's perspective is that he or she should be supported by contextualised information presentation and interaction. Users of nomadic computing systems either have connected

devices with them all the time or otherwise use existing stationary devices to access a nomadic system personalised to their needs [Fraunhofer FIT, 2002].

Therefore, according to the paradigm of nomadicity, nomadic users shouldn't have to adjust to their computing system. The computing system should rather be well adapted to the current location, position, circumstance, and perhaps even the mood of the nomadic user. The nomadic user cannot afford to spend a long time setting up his computing system, attending to the ergonomics of his position, and sacrificing 100% of his attention in, order to send a simple e-mail from an airport. It is these circumstances which illustrate some of the key human-computer interaction differences between a nomadic user and a desktop user. Clearly nomadic computing constitutes a very different set of requirements.

These requirements, which are driven by human-computer interaction principles, exemplify the main challenges which are to be addressed in this paper. Furthermore, it is these same requirements which distinguish nomadic computing from traditional desktop computing.

2.2. Ubiquitous and Pervasive Computing

The vision for a ubiquitous computing environment was first presented quite a few years before that of nomadic computing. It was Mark Weiser who initiated the ubiquitous computing work at PARC with the vision of better integrating computers with our environment and in effect make them invisible [Weiser, 1994]. Therefore, in a truly ubiquitous computing environment, people and the environment are augmented with computational resources that provide information and services whenever and wherever desired [Abowd et al., 2002].

The aim is to more successfully embed computing systems into our environment so that computers become more invisible as tools. A classic example of such an invisible, unobtrusive tool is eyeglasses. Eyeglasses allow us to perform our task without requiring us to perform any additional interactions or configurations. The tool itself is not actually invisible but the context of use or the interaction, whereby we simply look at the world rather than at the eyeglasses themselves, makes eyeglasses a good tool which enhance invisibility [Weiser, 1994].

The goals of pervasive computing are so similar to those of ubiquitous computing that the terms are used interchangeably in most cases. However, in some cases, because ubiquitous computing is a challenging ultimate goal that still remains far out of our reach, the term pervasive computing is used to

describe the trend towards increasingly ubiquitous systems. This trend includes the rise of unobtrusively embedded, completely connected, constantly available, highly intuitive and easily portable computing systems. The main goals of these two paradigms certainly differ from those of nomadic computing which are more oriented towards the mobility of the user. Still, there is one common vision shared by ubiquitous, pervasive and nomadic computing alike, and that is the achievement of unobtrusive interactions.

2.3. Limitations Imposed by Nomadic Computing

As the nomadic user must be mobile, the technology he or she uses faces many limitations. Perhaps this is not true in the case where the system already exists at his or her point of arrival, but in the cases where he or she needs to compute while in transit. In this respect, the following four aspects of nomadic computing are of primary concern: size, weight, integration, and durability.

Size has an effect on many aspects of nomadic computing. Firstly, the device itself must be of a portable size. Secondly, the size restrictions of the display bring up interesting design issues. Finally, the size of the items in the user interface itself such as controls, icons, and other information items, is a key factor in the usability of a nomadic device.

Weight is a critical factor when considering portable computing devices. This is best demonstrated by the supply and demand of laptop computers, a common piece of nomadic equipment. The price of the a laptop with the exact same specification but only 1 or 2 kilograms lighter is higher than that which is heavier. The nomadic user is likely to carry other baggage in addition to his or her laptop while relocating. The weight of all this can add up easily and surely make transportation all the more difficult. When considering wearable computing, it is all the more critical for the system to be lightweight.

Designers are constantly struggling with the attempt to decrease the weight of computing devices. Power supply is main cause for these weight problems. Complicated systems are usually weighed down by batteries and the user is forced to also carry some sort of recharger for the system.

The nomadic user will not be able to carry devices A, B & C in order to support his or her tasks X, Y & Z. Consequently, the *integration* of devices is essential for the user. Furthermore, integration must be simple and fast to carry out.

The equipment of nomadic users cannot afford to be fragile in any way. Nomadic users are constantly on the move and their equipment must have

the *durability* to keep up with them. Unlike the desktop computer, portable computing devices are constantly picked-up and put down. In this respect, the computing devices must be solid and their attachments must be fast to set-up but still firm enough to remain attached even while in motion.

3. Human-Computer Interaction

In this section, human-computer interaction, its history and its relevant principles are discussed.

Human-computer interaction is a relatively new concept which has only entered mainstream computer science terminology about a decade ago. It was during the Second World War when interaction between humans and machines was first studied with the objective to produce more effective weapon systems. Over time, as computers became more widely used, researchers began to study the interaction of humans and computers, rather than humans and machines [Dix et al., 1998].

Presently, ten years after its conception, the term human-computer interaction still hasn't emerged with a unanimously accepted definition which specifies the range of topics included in human-computer interaction. However, ACM SIGCHI offers the following as a working definition: "Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them." [Hewett et al., 2002]

Human-computer interaction is fundamental to the design of computing systems. It focuses on the principal fact that the system must support the user's task [Dix et al., 1998]. In addition to supporting the user's task, the objective is to support the task in the most simple, clear, and effective manner.

When considering the design of interactive systems, one can consider usability from a perspective of desirable conditions. These desirable conditions, which are generally accepted as principles, can be grouped into the three following categories: learnability, flexibility, and robustness. [Dix et al., 1998]

Learnability describes the "ease with which new users can begin effective interaction with the system and achieve maximal performance." Learnability is of utmost importance as it influences the user's first impression of the system. It can strongly affect whether the user continues using the system or replaces it with another tool. The key principles which support the

learnability of a system include predictability, synthesizability, familiarity, generalizability, and consistency. [Dix et al., 1998]

A predictable interactive system is one in which the user can, based on past interaction history, determine the effects of performing a certain action. Users should feel in control and should not encounter any surprises during their interaction with the system. [Dix et al., 1998]

The use of a good metaphor in the design of a system often improves the learnability of the system. The classic example of the successful use of a metaphor is that of the desktop. This metaphor has in fact opened the door to the widespread use graphical user interfaces. [Rosenfeld et al., 2001] Many graphical user interface controls, or interactive widgets, such as “windows”, “buttons”, and “menus” have enjoyed the success of being named with clever metaphors.

However, the use of metaphors in interactive paradigms can sometimes be more harmful than helpful for supporting the usability of the system. This can happen when a more experienced user extends the metaphor too far and wrongly expects certain functionalities of the system. In this case the predictability of the system can be hampered.

Flexibility represents the “multiplicity of ways in which the user and the system exchange information.” Some of the key principles of flexibility include the system’s customizability and the system’s ability to support multi-tasking. As adaptability describes the user’s ability to adjust the system’s form of input and output, it is an important part of customizability. Likewise, adaptivity, which describes the automatic customization of the user interface by the system, is also a key part of customizability. With adaptability, the user’s role is more explicit, whereas with adaptivity, the user’s role is far more implicit. [Dix et al., 1998]

When considering the nomadic user’s requirement for contextualized, personalized information, it is obvious that both adaptability and adaptivity are highly valuable characteristics to include in a nomadic information system.

Robustness depicts the “level of support provided to the user in determining the successful achievement and assessment of his or her goals.” The robustness of a system is measured in terms of its observability, its recoverability, its responsiveness, and its task conformance. [Dix et al., 1998]

User assistance has always been a common occurrence in interactive systems in order to support its usability and especially its learnability.

However, the desired conditions of nomadicity indirectly dictate a decreased dependency on user assistance.

Consider the typical user of a software program on a desktop computer just five years back. The software program was typically difficult to use but provided the user with a user's manual as well as some form of online help. Now consider a nomadic user who may carry all of his or her own computing equipment. The user is most certainly not willing to carry a few users' manuals or even just one in addition to his current equipment. As well, many of the nomadic computing devices have very limited screen space. Moreover, a nomadic user must often switch between existing workstations or computing devices. He or she cannot afford to complete a tutorial or read the help chapter of each new feature of each new computing system. For these reasons, nomadic systems must be less reliant on online help and user's manual as a method of assisting the user learning and understanding the system. Instead designers must aim towards more intuitive and consistent user interfaces which actually decrease the associated learning curve of the system.

4. Interactions of Nomadic Users

Rather than having nomadic users adapt their tasks and techniques to the available technologies, the technologies must evolve to meet the needs of the nomadic user.

However, nomadic computing presents a large challenge when it comes to suitable interaction methods. For instance, a user may use voice input when in a private location but when in public, that user may wish to perform the same task, with the same device but this time with some alternate input method which can maintain his or her privacy.

The driving force behind the rising popularity of the use of new interaction techniques is the growing trend away from explicit input methods towards implicit input methods [Abowd et al., 2002].

Common new interaction techniques, which are driven by this trend towards implicit input methods, include gesture recognition, natural language and speech interfaces, wearable computing and augmented reality, and information visualization. Each of these techniques has its own merits under different circumstances and for different users with different tasks and goals.

4.1. Consistency and Uniformity

Consistency in interaction behavior is critical for computing systems. In fact, consistency is likely the most widely mentioned principle in the literature on user interface design [Dix et al., 1998].

Uniform input behavior is especially important in the case of nomadic computing because in many cases, input behavior is not explicit and intuitively known. In fact in the case of systems which employ speech interfaces, all interaction is invisible and thus input commands must be remembered. [Rosenfeld et al., 2001] Without uniform structure, users won't be able to successfully switch between various computing devices and workstations.

Uniformity in the interaction methods augments the transference of learning from one computing device to another and therefore improves the overall usability of the devices [Rosenfeld et al., 2001]. This transference becomes largely essential in the nomadic computing environment where the user must be able to perform a certain task with the current computing system, whichever it may be.

As mentioned earlier, metaphors are a popular paradigm for interaction. However, metaphors can actually be damaging when they bring about contradictions in the interactions of various systems. For example, if a backwards gesture represents 'undo last action' on one system, while it represents 'go to previous' in another system, the user of these two systems is likely to get confused. Therefore, metaphors must be used especially carefully in the design of systems intended for nomadic users. Although the metaphors used for different devices don't need to be identical, they should at least be employed in a non-contradictory manner. This ensures that interactions with different devices are in any case consistent. [Oppermann, 2001]

4.2. Conditions of Limited Attention

In such nomadic computing environments which might involve portable or wearable computers, wireless communication, and smart spaces, a notably scarce resource is human attention. The typical nomadic user, who may be preoccupied with walking, driving, conversing with other individuals in person or on the phone, will not give the computing task, his or her full attention. [Garlan et al., 2002]

This phenomenon is not specific to nomadic computing environments. We sometimes call on our own ability to "multi-task" when we attend to other real-world interactions while we work on a computing task on our

desktop computers. We are unfortunately not very competent in this ability. We perceive to know the sequences of the system and consequently click the "OK" button before we read the "common" notice of a dialog. Only once it is too late, do we realize that by accepting the last dialog without reading it, we allowed the system to close an unsaved document or apply a new format style to an entire document. This guess is sometimes due to simple laziness or time pressure, but often it is just a case of a weakness in our attentiveness.

4.3. Suitability of Interaction

A typical nomadic user will require system support in a variety of different locations and in a variety of different conditions. The nomadic user may have to perform a computing task while walking, or among a noisy crowd, or without the use of hands. This reality poses a significant human-interaction challenge, especially in the ways of user input and user adaptivity.

User input has traditionally been carried out by means of a keyboard and mouse. While these input devices may be appropriate for the desktop user who is permanently established at a stationary workstation, it is not always appropriate for the nomadic user.

Typical devices of nomadic users include mobile computers such as PDA's, and mobile phones. These devices present many design challenges because both the available screen space and amount of the input controls are very limited compared to the desktop computing system. For this reason new interaction techniques such as speech and gestural interfaces are gaining popularity as input alternatives. Not only do these techniques offer alternatives to traditional input methods but they are also considered to be more natural methods for humans, and furthermore they also encourage mobility and require less physical resources of the user.

This shift from an explicit means of human input to more implicit forms of input, contribute to the trend towards pervasive computing. The ultimate goal being that our natural interactions with the physical environment would provide sufficient input for the system services. For instance, walking into a certain space would be enough to announce your presence and identity in that location. [Pirhonen et al., 2002] This can easily be contrasted with the traditional method of typing a user id and password into a computer connected to the network in order to announce your presence to the system.

Obviously nomadic computing offers designers the chance to explore new input methods. However, as not all input methods are appropriate in all

circumstances, nomadic computing introduces a great challenge to designers of interactive systems, especially those intended for nomadicity.

As mentioned earlier, adaptivity, one of the key elements of customizability and flexibility, is an important usability characteristic. In the past, adaptive computing systems typically adapted the “information selection and presentation to the user's goals, preferences, knowledge, and interests.” However, the enablers of nomadic computing, such as mobile and wearable computing and wireless multimedia communications, give way to new forms of user adaptive applications. In particular, nomadic information systems may utilize “roaming in the physical space with localization technologies (i.e. GPS, DGPS, Infrared, and electronic compass) to adapt to a richer context model of the user's previous and current position over the whole process of his or her activities.” [Fraunhofer FIT, 2002]

Therefore, the appropriateness of the interaction method for the user’s current circumstance is largely dependent upon the system’s methods of user input and user adaptivity.

5. Summary

The interactions of nomadic users give rise to a new set of human-computer interaction challenges. These challenges have been identified and presented in this research paper.

When designing any computing system, including nomadic computing systems, designers should uphold the desirable conditions of usability, which include learnability, flexibility, and robustness. When designing a nomadic computing system, the designer must also consider the limitations relating to size, weight, integration, and durability. In addition to these conditions and limitations, designers must also account for the special interaction concerns of nomadic users. Firstly, the nomadic computing system should maintain consistency and uniformity. Secondly, it should be designed to perform under conditions of limited attention. Lastly, the system should provide the user with a suitable interaction option for each particular circumstance of the user.

Obviously designers of nomadic computing systems are pressed by many challenges from a human-computer interaction point-of-view. They must aim to preserve both good usability and the basic conventions of nomadic computing devices, all while meeting the unique interaction needs of the nomadic user. Fortunately, these very specific requirements of nomadic computing have created vast opportunities for many new interaction techniques and will continue to do so in the future.

The research conducted for this paper provides a base for a variety of related future research subjects. Foremost, the set of human-computer interaction challenges of nomadic computing presented in this paper can certainly be expanded on. As well, user studies can be conducted on actual nomadic computing systems to determine the relative importance of each of the identified challenges as a means of enhancing the nomadic user experience.

References

[Abowd et al., 2002] Gregory D. Abowd, Elizabeth D. Mynatt, and Tom Rodden, The human experience, *IEEE - Pervasive Computing* **1**, 1 (January-March 2002), 48-57.

[Ark et al., 1999] Wendy S. Ark and Ted Selker, A look at human interaction with pervasive computers, *IBM Systems Journal* **38**, 4 (1999), 504-507.

[Dix et al., 1998] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale, *Human-Computer Interaction*, Second Edition, Prentice Hall Europe, 1998.

[Fraunhofer FIT, 2002] Fraunhofer FIT, Information Contextualization, 2002. Available as http://www.fit.fraunhofer.de/gebiete/nomad-is/index_en.xml.

[Garlan et al., 2002] David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste, Toward Distraction-Free Pervasive Computing, IEEE – Pervasive Computing, April-June, 2002, 22-31.

[Hewett et al., 2002] Hewett, Baecker, Card, Carey, Gasen, Mantei, Perlman, Strong and Verplank, ACM SIGCHI Curricula for Human-Computer Interaction, ACM SIGCHI, 1996. Available as http://sigchi.org/cdg/index.html.

[Kleinrock, 1995] Leonard Kleinrock, Nomadic Computing - An Opportunity, ACM SIGCOMM, Computer Communication Review 25 (1), 1995, 36-40.

[Kleinrock, 2000] Leonard Kleinrock, Nomadic Computing and Smart Spaces, IEEE Internet Computing, January-February 2000, 52-53.

[Oppermann, 2001] Reinhard Oppermann, Human-Computer Interaction Challenges in Nomadic Computing. ERCIM News No. 46, July 2001. Available as http://www.ercim.org/publication/Ercim_News/enw46/oppermann.html.

[Rosenfeld et al., 2001] Ronald Rosenfeld, Dan Olsen and Alex Rudnicky, Universal speech interfaces. *ACM Interactions* **8**, 6 (November-December 2001), 34–44.

[Pirhonen et al., 2002] Antti Pirhonen, Stephen Brewster and Christopher Holguin, Gestural and Audio Metaphors as a Means of Control for Mobile Devices, ACM CHI 2002, Vol. 4, Issue No. 1, April 2002.

[Weiser, 1994] Mark Weiser, The world is not a desktop, *ACM Interactions* **1**, 1 (January 1994), 7-8.

XML-pohjaisen tiedonhaun välineistä

Teemu Kumpulainen

Tiivistelmä

Tässä tutkielmassa on tarkoitus pohtia XML-dokumenttien kyselykielten ominaisuuksia. Erityisesti tarkastellaan W3-konsortiumin työstämää XQuery-kieltä ja verrataan sitä jo yleisesti käytössä olevaan XSLT-kieleen. Tutkielmassa todetaan, että on olemassa tarve deklaratiivisemmille kyselykielille kuin mitä XQueryn kaltainen kieli kykenee tarjoamaan.

Avainsanat ja -sanonnat: XML, XSLT, Xpath, XQuery.
CR-luokat: D.3, H.3.3

1. Johdanto

Tutkielmassa esitellään rakenteisten dokumenttien kuvauskieltä, XML:ää ja siihen pohjautuvaan tiedonhakuun tarkoitettuja kyselykieliä. Erityisesti tarkastellaan W3-konsortiumin työstämää XQuery-kieltä. Siihen liittyvien esimerkkien yhteydessä pyritään osoittamaan muutamia ongelmallisia osa-alueita liittyen XQueryn perusluonteeseen. Luku 2 käsittelee yleisesti XML-kielen piirteitä, siinä määrin mitä tämän tutkielman yhteydessä tarvitaan. Kolmas luku esittelee XPath- ja XSLT-kieltä, neljännessä perehdytään XQueryyn, ja katsotaan muutaman esimerkin avulla, kuinka XQuery ja XSLT eroavat ilmaisuvoimaltaan. Neljännen luvun lopussa tarkastellaan ongelmia, joita XQueryn ominaispiirteisiin liittyy. Lopuksi todetaan, että on olemassa tarve deklaratiivisemmalle kyselykielelle kuin mitä XQueryn kaltainen kieli kykenee tarjoamaan.

2. Yleisiä XML-kielen käsitteitä

Tässä luvussa on tarkoitus antaa yleiskuva XML-kielen käsitteistä ja tutustua hieman XML-muotoisen tiedon käsittelyssä käytettäviin välineisiin.

2.1. XML-kielen perusteet

XML(eXtended Markup Language) on rakenteisen tekstin merkintäkieli, joka on tarkoitettu ennenkaikkea tietoverkoissa käytettäväksi välineeksi, ja joka tarjoaa HTML-kieltä monipuolisemmat mahdollisuudet tiedon prosessointiin.

W3-konsortiumi julkaisi vuonna 1998 XML-kielen virallisen suosituksen. Nyt on käytössä vuonna 2000 revisoitu versio. [XML]

XML-muotoisen tiedon perusrakenteena voidaan pitää elementtiä, joka muistuttaa pienin eroin muiden merkintäkielten vastaavaa rakennetta. Elementti koostuu alkumerkistä (start tag), sisällöstä ja loppumerkistä (end tag). Sisältönä voi olla merkitsemätöntä tekstiä tai toisia elementtejä. Alkumerkit koostuvat merkin aloittavasta pienempi kuin -merkistä (<), elementin nimestä, mahdollisista välilyönnein erotetuista attribuuteista ja merkin lopettavasta suurempi kuin -merkistä (>). Attribuutit ovat eräänlaisia arvopareja, joissa annetaan attribuutin nimi, yhtäsuuruusmerkki (=) ja attribuutin arvo lainausmerkkien (") sisään kirjoitettuna. Loppumerkin nimi on sama kuin alkumerkinkin. Loppumerkissä on aloittavan pienempi kuin -merkin jälkeen kauttaviiva (/) eikä se voi sisältää attribuutteja. Esimerkiksi

```
<entry day="Monday">The morning was chilly and dull.</entry>
```

on XML-muotoinen elementti. (XML-koodi on luettavuuden vuoksi kirjoitettu tässä ja muissa esimerkeissä tasalevyisellä kirjasintyylillä.) Elementin, joka alkaa jonkun toisen elementin sisällä, on myös päätyttävä siellä. Erikoismerkintänä tyhjä elementti voidaan lyhentää jättämällä loppumerkki pois ja merkitsemällä kauttaviiva ennen alkumerkin päättävää suurempi kuin -merkkiä. Tällöin elementit

```
<entry day="Monday"></entry>
```

ja

```
<entry day="Monday"/>
```

tarkoittavat samaa asiaa.

2.2. Dokumentin kieliopin määrittely

Hyvin muodostettu XML-dokumentti sisältää mahdollisten kommenttien lisäksi johdanto-osan (prolog) ja yhden juurielementin (root), jonka sisälle kaikki muut elementit sijoittuvat. Johdanto-osassa voidaan antaa muunmuassa kyseisen dokumentin rakenteeseen liittyvää informaatiota.

Erona perinteisesti verkkojulkaisuun käytettyyn HTML-kieleen nähden XML sellaisenaan ei tarjoa kiinteää tietyllä tavalla nimettyä elementtijoukkoa, vaan elementit voidaan nimetä periaatteessa mielivaltaisesti. XML-muotoinen, tiettyyn sovellusalueeseen liittyvä tieto ilmaistaan yleensä tämän sovellusalueen puitteissa yhtenäistetyssä muodossa, joka määrätään tarkoitusta varten tehdyllä DTD-määrittelyllä (Document Type Definition). DTD-määrittelyn avulla voidaan siis antaa kielioppi tietyn XML-sovelluksen dokumenteille, mutta se on melko suppea ja joissakin yhteyksissä on tarve

tarkemmalle tasolle yltävästä määrittelystä. XML-sovelluksena toteutettu XML Schema tarjoaa DTD-notaatiota monipuolisemmat mahdollisuudet kieliopin määrittämiseen. [XMLSchema] Tässä sitä ei kuitenkaan käsitellä mainintaa enempää.

XML-kielen vahvuus on tiedon kuvaamisessa. Sen avulla voidaan esittää hyvin monenlaisia asioita, joihin perinteisesti verkossa käytetty HTML ei kykene. Koska sen merkkaus perustuu nimenomaan tietoon eikä ulkoasuun, on XML sovelluksineen jäsentyneessä tiedonhaussa erittäin mielenkiintoinen tutkimuksen kohde.

3. XPath ja XSLT tiedonhaussa

Tässä luvussa käsitellään XML-tiedon käsittelemiseen tarvittavia välineitä, joita ovat XML-elementteihin viittaamiseen tarkoitettu XPath, dokumenttien uudelleenmuotoiluun kykenevä yleisesti käytössä oleva XSLT-kieli ja nimenomaan kyselykieleksi suunniteltu XQuery-kieli. Ensin esitellään XSLT- ja sen jälkeen XQuery-kieltä, ja verrataan niiden välisiä eroja.

3.1. XPath

Xpath-kielellä voidaan osoittaa johonkiin osaan XML-dokumentissa [XPath, 1999]. XPath näkee XML-dokumentin puurakenteena, jonka solmuina on XML-elementtejä. Elementteihin voidaan viitata erityisillä polkumäärityksillä, joihin saatetaan liittää identifioivia predikaatteja. Esimerkiksi dokumentissa

```
<diary>
    <entry day="Monday">The morning was chilly and
    dull.</entry>
    <entry day="Tuesday">The rain was falling all the
    day.</entry>
    <entry day="Wednesday">The fog was quite thick.</entry>
    <entry day="Thursday">The sunset was beautiful.</entry>
    <entry day="Friday">The night was clear.</entry>
    <entry day="Saturday">The headache was devastating.</entry>
    <entry day="Sunday">The sun was shining.</entry>
</diary>
```
(Dokumentti 1)

voidaan viitata mihin tahansa entry-nimiseen elementtiin XPath-ilmaisulla `/diary/entry` tai pelkästään `entry`. Haluttaessa viitata maanantaihin (`day`-attribuutti arvolla `monday`) liittyvään entry-elementtiin, voidaan antaa ilmaisu

`//entry[1]` (XPath-lauseke 1a),

koska kyseinen elementti on ensimmäisenä dokumentissa 1, tai vaikkapa

`/diary/entry[@day='monday']` (XPath-lauseke 2a).

Viittaus voi olla absoluuttinen, jolloin määritellään koko polku juurielementistä lähtien, tai suhteellinen. Hakasulkeiden välissä annettava lauseke voi olla myös jokin XPathin funktioista, joilla voidaan käsitellä mm. elementin tekstisisältöä ja laskea aritmeettisia lausekkeita. XPathin syntaksissa käytetään erinäköisiä lyhenteitä joista muutamia itse asiassa käytettiin jo, kun annettiin lauseke 1a, jonka hakasulkeiden välinen osa kokonaisuudessaan kirjoitettuna olisi `position() = 1`. Mikäli XPathin kauttaviivamerkinnän (/) jälkeen ei ole kerrottu erikseen askeltyyppiä, on kyse `child`-tyyppisestä polkuaskeleesta. Kaksi viivaa (//) on puolestaan lyhenne `descendant-or-self`-tyyppisestä polkuaskeleesta. Kokonaisuudessaan lyhentämätön ilmaisu lausekkeelle 1 olisi

```
/descendant-or-self::node()/child::entry[position()=1]
```

(XPath-lauseke 1b)

ja lausekkeen 2a lyhentämätön versio olisi

```
/child::diary/child::entry[attribute::day=1]
```

(XPath-lauseke 2b).

Lausekkeessa 2b käytetään askeltyyppiä `attribute`, joka voidaan lyhentää @-merkillä. Lisäksi yleinen askeltyyppi on `parent` (vanhempisolmu), joka voidaan lyhentää kahden pisteen merkinnällä (..). Esimerkiksi diary-elementtiin voitaisiin viitata `entry`-elementistä merkinnällä

`entry/..` (lyhentämättömänä `entry/parent::node()`).

XPathin ilmaisut ovat olennainen osa XSLT- ja XQuery-kieliä. Tällä hetkellä W3-konsortiumin suosittelema versio on 1.0, mutta kehitteillä on uusi versio 2.0, jota XQueryn on tarkoitus käyttää [Xpath, 2002].

3.2. XSLT

XSLT (Extensible Stylesheet Language Transformation) -kieli on XML-dokumenttien muuntamiseen tarkoitettu kieli [XSLT, 1999]. Sillä voidaan muuntaa XML-dokumentti esimerkiksi XHTML-dokumentiksi (HTML-kieli ilmaistuna XML-notaatiolla), mutta sillä voidaan myös rakentaa täysin uudenlaisia XML-elementtejä, tai valita vain tietty joukko esitettävään muotoon vaatimalla esitettävien elementtien sisällöltä tiettyjä ehtoja. Tällä hetkellä XSLT:n versio 1.0 on ainoa yleisesti käytössä oleva XML-kyselykieli.

XSLT on XML-sovellus, eli XSLT-dokumentti koostuu XML-elementeistä. Periaatteena on, että muokkauksen lähdedokumentin tai -dokumenttien

elementtejä valitaan XPath-lausekkeilla ja tuotetaan tulokseksi XSLT-dokumentin määrittämää tekstiä. XSLT:stä on suunnitteilla myös versio 2.0, joka XQueryn tapaan tukisi XPath versiota 2.0 [XSLT, 2002].

3.3. Yksinkertainen XSLT-esimerkki

Esimerkit on tuotettu käyttäen Microsoft Windows-käyttöjärjestelmille tehtyä MSXSL-ohjelmaa, joka tuottaa XML- ja XSL-dokumenttien perusteella uuden dokumentin [MSXSL].

Jos lähdedokumentti on aiemmin esitetty dokumentti 1, ja XSLT-dokumentti on

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml"/>
<xsl:template match="diary">
<content>
Diary of <xsl:value-of select="@author"/>.
----------------------------------------
<xsl:apply-templates/>
----------------------------------------
</content>
  </xsl:template>
   <xsl:template match="entry"><xsl:value-of select="@day"/>:
     <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
```

(XSLT-esimerkki 1),

niin tulokseksi saadaan dokumentti

```
<content>
Diary of Wally Week.
----------------------------------------
Monday: The morning was chilly and dull.
Tuesday: The rain was falling all the day
Wednesday: The fog was quite thick.
Thursday: The sunset was beautiful.
Friday: The night was clear.
Saturday: The headache was devastating.
Sunday: The sun was shining.
----------------------------------------
</content>.
```

XSLT-esimerkissä 1 määrätään `xsl:output`-elementin `method`-attribuutilla tulosdokumentin olevan XML:ää. Esimerkissä on tehty lähdedokumentin elementtejä varten valmiita malleja (template), jotka toimivat siten, että `xsl:template`–elementtien attribuutti `match` saa arvokseen XPath-ilmaisuja, joiden osoittamat elementit käsitellään `xsl::template`-elementin sisällä. XPath-lausekkeet, joita `match`-attribuutti saa sisältää, koostuvat vain `attribute` (/@)- tai `child` (/) -tyyppisistä polkuaskelista. Mahdollisten alielementtien sijainti tulosdokumentissa määrätään `xsl::apply-templates`-

elementillä. Mikäli jollekin osaselle ei löydy mallia, tulostuu se muokkaamattomana tulosdokumenttiin, olettaen, että XSLT-dokumentissa on käytetty `xsl:apply-templates`-elementtiä.

Toinen tapa siirtää tietoa lähdedokumentista tulosdokumenttiin on käyttää `xsl:value-of` -elementtiä, jonka select-attribuutin arvo on jälleen XPath-polkumääritys, jolloin polun osoittama elementti sijoittuu elementin osoittamaan paikkaan. Esimerkissä tätä käytettiin viikonpäivien tulostamiseen.

XSLT tarjoaa lisäksi muita ohjausrakenteita, joiden avulla XSLT-kieltä voidaan itse asiassa käyttää vaikkapa rekursiiviseen ohjelmointiin, kuten esimerkiksi Nykänen [2001] kirjassaan esittää. Tällainen soveltaminen on kuitenkin työlästä, ja yleensä on löydettävissä mielekkäämpiä menetelmiä. Yleisestikin XSLT-kielen ongelma on sen työläs kirjoittaminen. XSL-dokumentista tulee helposti monimutkainen ja sen suunnitteluun ja tekemiseen voi kulua kohtuuttoman pitkiä aikoja. XSLT tarjoaa mahdollisuudet XML-tiedon monipuoliseen prosessointiin mutta myös vaatii käyttäjältään paljon.

4. XQuery

XQuery on kehitteillä, koska on katsottu tarvittavan jatkuvasti kasvavan XML-muotoisen tiedon älykkääseen kyselyyn kykenevää kieltä [XQuery, 2002]. Vaikka XSLT on ilmaisuvoimaltaan XQueryn tasoa [Lechner et al., 2002], on silti katsottu, että on olemassa tarve helppokäyttöisemmälle kielelle.

Selkein ero XSLT-kieleen on se, että XQuery ei ole toteutettu XML-muotoisesti. XQuery 1.0 on suunnitteilla oleva laajennos suunnitteilla olevaan XPathin 2.0-versioon, joka laajentaa XPath 1.0-version funktio- ja operaatiojoukkoa ja lisää mm. tietotyyppejä päivämääriä ja muita vastaavia varten. XQueryn 1.0- ja XPathin 2.0-version operaatioita koskevassa, 15.10.2002 julkistetussa luonnoksessa esitellään yhteensä 251 eri funktiota tai operaattoria, jotka aiotaan sisällyttää kieleen [XQuery op, 2002]. Voitaneen todeta, että tässä mielessä käyttäjältä vaadittavat edellytykset XQueryn hallintaan voivat olla sitä luokkaa mitä yleensä rajapintojen avulla ohjelmoitaessa tarvitaan. Tämä tekee väitteen XQueryn helppokäyttöisyydestä hieman kyseenalaiseksi.

4.1. FLWOR-lauseke

XPath 2.0-lausekkeiden lisäksi XQuery tarjoaa niinsanotun FLWOR-lauserakenteen (For-Let-Where-Order-Return), jonka avulla se tukee muuttujien iterointia ja sitomista väliaikaisiin tuloksiin. FLWOR-lause koostuu vähintään

for- tai let- ja return-lauseesta. For- ja let-lauseet määrittävät muuttujat XPath-lausekkeilla ja return-lause tuottaa FLWOR-lauserakenteen tulosteen. Where- ja order-lauseet testaavat ja rajoittavat muuttujien sisältöä ja uudelleenjärjestävät tulosta.

For- ja let- lauseet eroavat siinä, että let-lauseen muuttuja arvotetaan uudelleen jokaista for-lauseen iteraatiokierrosta kohti. Return-lauseen antama merkkijono tulostuu samaten kerran jokaista for-lausetta kohti. Jos for-lausetta ei ole, suoritetaan let-lauseen arvotus kuten tuloksessa olisi yksi (tyhjä) for-lause mukana. Seuraavat esimerkit XQueryn spesifikaatioluonnoksesta [XQuery, 2002] valaisevat asiaa. Kysely

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

tuottaa tulokseksi out-elementin

```
<out>
   <one/>
   <two/>
   <three/>
</out> ,
```

kun taas for-lausetta käytettäessä kysely

```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

tuottaa tulokseksi elementit

```
<out>
   <one/>
</out>
<out>
   <two/>
</out>
<out>
   <three/>
</out> .
```

Huomataan, että käytettäessä let-lausetta saadaan yksi out-elementti, jonka sisään kaikki muuttujan arvotukset tulevat, ja for-lauseessa jokaista arvotusta kohti saadaan oma out-elementti. Usein FLWOR-lauseesta käytettään myös nimitystä FLWR. Lause lisää XPathin deklaratiiviseen tapaan esittää asiat proseduraalisten ohjelmointikielten silmukkarakenteen. XQueryn kyselyt rakennetaan tämän rakenteen luomaan kehikkoon.

4.2. XQuery ja XSLT -esimerkkejä ja vertailuja

Yleiskuvan saamiseksi tässä esitetään yksinkertainen esimerkki kielestä. Se on otettu XQueryn käyttötapakuvauksesta [XQuery use, 2002]. Kyselyssä käyte-

tään tiedonhaun kohteena XML-dokumenttia, jonka URL-osoitteeksi oletetaan http://www.bn.com/bib.xml:

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price>
    </book>
    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
    </book>
    <book year="1999">
        <title>The Economics of Technology and Content for Digital
         TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>
</bib>
```

(Dokumentti bib.xml).

Dokumentissa on siis kuvattu book-elementtien avulla joitakin kirjoja, joiden nimi-, tekijä- ja muita vastaavia tietoja kuvaamaan on käytetty elementtejä `title`, `author` ja niin edelleen. Nyt halutaan tietää niiden bib.xml-dokumentissa kerrottujen kirjojen julkaisuvuosi ja nimeke, jotka on julkaissut Addison-Wesley vuoden 1991 jälkeen. Kysely kirjoitetaan seuraavasti:

```
<bib>
        {
        for $b in document("http://www.bn.com/bib.xml")/bib/book
        where $b/publisher = "Addison-Wesley" and $b/@year > 1991
        return
        <book year="{ $b/@year }">
                { $b/title }
        </book>
        }
</bib>
```

(XQuery-esimerkki 1).

Tulokseksi saadaan

```
<bib>
 <book year="1994">
   <title>TCP/IP Illustrated</title>
 </book>
 <book year="1992">
   <title>Advanced Programming in the Unix environment</title>
 </book>
</bib>
```
(XQuery-esimerkin 1 tulos).

Kyselyssä määritellään for-sanalla alkavalla rivillä Xpath-lausekkeen (`document("http//..")/bib/book`) avulla muuttuja, joka osoittaa siis dokumentin `book`-elementteihin. Aaltosuluilla ({ , }) merkitty ulompi lohko suoritetaan järjestyksessä rivi kerrallaan niin monta kertaa kuin `book`-elementti löytyy. Jokaisella iteraatiokierroksella where-osassa tarkistetaan, josko XPath-lausekkeiden osoittamat elementit toteuttavat annetut ehdot. Tulosteen tuottavassa return-osassa annetaan XML-muotoista tekstiä, johon on upotettu aaltosulkujen merkitsemiin lohkoihin XPath-lausekkeita, jotka osoittavat johonkin elementtiin.

XQuery-esimerkkikysely 1 on melko suoraviivaisesti käännettävissä XSLT:ksi. Vastaavan tuloksen tuottava XSLT-dokumentti olisi seuraava:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <bib>
    <xsl:for-each
        select="bib/book[publisher='Addison-Wesley' and @year>1991]">
    <xsl:variable name="year" select="@year"/>
         <book year= "{$year}" ><xsl:copy-of
select="./title"/></book>
   </xsl:for-each>
</bib>
</xsl:template>
</xsl:stylesheet>
```
(XSLT-esimerkki 2).

XSLT-esitys vaatii hieman enemmän lukijaltaan, mutta voidaan huomata, että varsinainen kyselyssä käytetty semantiikka saadaan miltei identtisellä XPath-lausekkeella `publisher='Addison-Wesley' and @year>1991` , erona on vain se, että XQueryssa lauseke on annettu where-osassa muuttujan avulla viittaamalla ja XSLT-versiossa viitattavan elementtisolmun `bib/book` predikaattina.

Seuraavassa, hieman monimutkaisemmassa kyselyssä, XQueryssä joudutaan käyttämään sisäkkäisiä kyselyitä. Jokaista yksittäistä kirjoittajaa (`author`-elementtiä) kohti haetaan tämän kirjoittamien teosten nimet. XQuery ei tarjoa mainitun kaltaisen ongelman ratkaisevaa yhteen kyselylausekkeeseen tarpeellisen tiedon kokoavaa ilmaisua, vaan ratkaisu on suunniteltava proseduraalisten ohjelmointikielten tapaan kahden sisäkkäisen silmukkarakenteen eli FLWR-lauseen avulla:

```
<results>
  { for $a in distinct-values(
                  document("http://www.bn.com/bib.xml")//author)
    return
      <result>
        { $a }
        { for $b in document("http://www.bn.com/bib.xml")/bib/book
          where some $ba in $b/author satisfies deep-equal($ba,$a)
          return $b/title
        }
      </result>
  }
</results>
```
(XQuery-esimerkki 2).

Koska dokumentissa bib.xml voi esiintyä sama `author`-elementti useammassa `book`-elementissä, käytetään XPath 2.0 `distinct-values`-funktiota eliminoimaan samansisältöiset `author`-elementit. Ulompi kysely tulostaa haetut elementit ja sisemmän kyselyn, joka suoritetaan uudelleen jokaiselle iteraatiokierrokselle. Where-osan avainsana some vaatii, että vain jonkun muuttujan `$ba` esittämistä elementeistä on täytettävä vaadittu ehto. Funktio `deep-equal` tutkii, että sen saamien parametrien sisältö ja elementtisolmujen nimet ovat samat. Kyselyn tulisi tuottaa tulokseksi seuraavanlainen XML-elementti:

```
<results>
  <result>
    <author><last>Stevens</last><first>W.</first></author>
    <title>TCP/IP Illustrated</title>
    <title>Advanced Programming in the Unix environment</title>
  </result>
  <result>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <title>Data on the Web</title>
  </result>
  <result>
    <author><last>Buneman</last><first>Peter</first></author>
    <title>Data on the Web</title>
  </result>
  <result>
    <author><last>Suciu</last><first>Dan</first></author>
    <title>Data on the Web</title>
  </result>
</results> .
```

Tuloksen result-elementtien sisältönä on yksi `author`-elementti, ja siihen liittyvät `title`-elementit.

Seuraavaksi toteutamme kyselyn XSLT-version. Koska XSLT-kielen nykyisin suositeltu versio 1.0 ei tunne XPath-versiota 2.0, kierretään `disctinct-values`-funktion puute käyttämällä XPathin askelpolkutyyppiä `preceding` vertaamaan kulloisella `xsl:for-each`-elementin iterointikierroksella haettua arvoa dokumentissa edeltäneisiin arvoihin. XSLT-dokumentti

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
<results>
    <xsl:for-each select="//author[not(. = preceding::*//author)]">
  <result>
      <xsl:variable name="a" select="."/>
      <xsl:copy-of select="$a"/>
      <xsl:for-each select="//book[author=$a]">
        <xsl:copy-of select="./title"/>
      </xsl:for-each>
  </result>
     </xsl:for-each>
</results>
    </xsl:template>
  </xsl:stylesheet>
```
(XSLT-esimerkki 3)

tuottaa tulokseksi saman kuin XQuery-esimerkki 2. XQuery-kyselyn kahta sisäkkäistä FLWR-lausetta vastaavat myös sisäkkäiset `xsl:for-each`-elementit, joista sisemmässä on hyödynnetty muuttujaa, joka on sidottu ulomman `xsl:for-each`-elementin kontekstisolmuun. Tässäkin kyselyssä erona nousee esiin lähinnä XSLT-kielen työläs notaatio.

Rakenteensa puolesta `xsl:for-each`-elementti tarjoaa samat edellytykset kuin XQueryn FLWR-lauserakenne, ilmaisuvoimassa ero syntyykin lähinnä XPathin versioeroista.

4.3. XSLT:n ja XQueryn erot ja ongelmat

XSLT ja XQuery pystyvät ilmaisemaan samat asiat, mutta koska XQuery on suunniteltu XPath 2.0-versiolle, se tarjoaa laajemmat funktiokirjastot ja tyyppimäärittelyt kuin mihin XSLT kykenee. Kuitenkaan XSLT:n versio 2.0 ei ole enää tässäkään suhteessa heikompi, sillä se on suunniteltu samaiselle XPathin versiolle kuin XQuerykin.

Ero syntyy lähinnä käytetyn notaation tuomista piirteistä. Koska XSLT tukeutuu XML-notaatioon, on sitä tietyissä tapauksissa hankalaa tuottaa, kuitenkin se myös antaa XSLT-syntaksille selkeän rakenteen, joka on loogisesti seurattavissa. Lisäksi XSLT on sen laajalti käytössä ja tuettu hyvin eri ympäristöissä. Se ei ole suunniteltu eksplisiittesti tiedonhakua varten, mutta käy myös siihen tarkoitukseen.

XQuery tarjoaa oman notaationsa, joka ei kuitenkaan ole täysin ongelmaton sekään. XQuery on käännettävissä merkityksellisiltä osiltaan lähes suoraan XSLT:ksi [Lechner et al., 2002; Lenz, 2002]. XQuery-kielestä on pyritty luomaan helppokäyttöinen ja luettava, mutta se ei aina ole sitä. Osittain helppokäyttöisyyttä nakertaa XPath 2.0-version funktioiden ja operaatioiden suuri määrä; voi olla vaikea löytää kohtuullisissa aikarajoissa juuri johonkin

tiettyyn kyselyyn sopivaa funktiota satojen joukosta. Lisäksi XQueryn tehokas käyttö vaatii käyttäjäänsä ymmärtämään suhteellisen syvällisesti, kuinka kielen ilmaisun prosessointi tapahtuu proseduraalisesti, mikä ei välttämättä tarjoa kyselyn tekijän näkökulmasta tehokkainta lähestymistapaa, kun kyselykielen käyttäjän tulisi keskittyä kyselyn toteutuksen sijasta sen merkitykseen.

XQueryn suurimpia ongelmia ei ole se, ettei se tarjoa juuri mitään mihin XSLT ei kykenisi, vaan se, että XQueryn kysely on toteutettava samoin periaattein tai samanlaisen ajatuksellisen prosessin tuloksena kuin XSLT-kielessäkin, eli perinteisen proseduraalisen ohjelmointikielen periaattein. Kyselyssä on otettava huomioon, miten ja missä järjestyksessä kyselyn muuttujien arvotukset tapahtuvat, sekä ymmärrettävä kyselyn suorituksen proseduraalinen kulku, mikä vie huomion pois olennaisesta eli tiedonhausta. Seuraava esimerkki, joka on lyhennetty versio XQueryn käyttötapakuvauksesta [XQuery use, 2002] poimitusta tekstihakua esittelevästä kyselystä, pyrkii valaisemaan asiaa.

Oletetaan että on olemassa XML-dokumentti "news.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<news>
<news_item>
   <title> Gorilla Corporation acquires YouNameItWeIntegrateIt.com </title>
   <content>
      <par> Today, Gorilla Corporation announced that it will purchase
          YouNameItWeIntegrateIt.com. The shares of
          YouNameItWeIntegrateIt.com dropped $3.00 as a result of this
          announcement.
      </par>
      <par> As a result of this acquisition, the CEO of
          YouNameItWeIntegrateIt.com Bill Smarts resigned. He did not
          announce what he will do next.  Sources close to
          YouNameItWeIntegrateIt.com hint that Bill Smarts might be
          taking a position in Foobar Corporation.
      </par>

      <par>YouNameItWeIntegrateIt.com is a leading systems integrator
          that enables <quote>brick and mortar</quote> companies to
          have a presence on the web.
      </par>
   </content>
   <date>1-20-2000</date>
   <author>Mark Davis</author>
   <news_agent>News Online</news_agent>
</news_item>
<news_item>
   <title>Foobar Corporation releases its new line of Foo products
   today</title>
   <content>
      <par> Foobar Corporation releases the 20.9 version of its Foo
            products.  The new version of Foo products solve known
            performance problems which existed in 20.8 line and
            increases the speed of Foo based products tenfold. It also
            allows wireless clients to be connected to the Foobar
```

```
            servers.
        </par>
        <par> The President of Foobar Corporation announced that they
              were proud to release 20.9 version of Foo products and
              they will upgrade existing customers <footnote>where
              service agreements exist</footnote>
              promptly. TheAppCompany Inc. immediately announced that it
              will release the new version of its products to utilize
              the 20.9 architecture within the next three months.
        </par>
        <figure>
            <title>Presidents of Foobar Corporation and TheAppCompany
            Inc. Shake Hands</title> <image source="handshake.jpg"/>
        </figure>
    </content>
    <date>1-20-2000</date>
    <news_agent>Foobar Corporation</news_agent>
</news_item>
</news>
```
(Dokumentti news.xml)

ja lisäksi vielä toinen dokumentti "company-data.xml"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<company>
  <name>Foobar Corporation</name>
  <partners>
    <partner>YouNameItWeIntegrateIt.com</partner>
    <partner>TheAppCompany Inc.</partner>
  </partners>
  <competitors>
    <competitor>Gorilla Corporation</competitor>
  </competitors>
</company>
```
(Dokumentti company-data.xml),

ja halutaan saada kyseisistä dokumenteista selville, missä uutisjutuissa on mainittu yritys nimeltä Foobar Corporation ja joku sen yhteistyökumppaneista (`partner`) samassa jutussa (`news_item`), ja että juttu ei ole yrityksen itsensä julkaisema (`news_agent`). Tiedetään, että yhteistyökumppaneiden nimet on annettu company-data.xml-dokumentin `partner`-elementeissä. Tehdään kysely

```
define function partners($company as xs:string) as element* {
  let $c := document("company-data.xml")//company[name = $company]
  return $c//partner }
for $item in document("news.xml")//news_item,
    $c in document("company-data.xml")//company
let $partners := partners($c/name)
where contains(string($item), $c/name)
        and some $p in $partners satisfies
        contains(string($item), $p)
        and $item/news_agent != $c/name
return $item .
```

Kyselyssä määritellään funktio `partners`, joka tekee toisen XQuery-kyselyn, jonka tuloksena tuottamien XML-elementtien joukon avulla haetaan ne jutut, joissa esiintyy jokin tuon joukon elementeistä. Tulokseksi saadaan

dokumentin news.xml ensimmäinen `news_item`-elementti (toinen on yrityksen itsensä julkaisema). Kysely on itseasiassa funktiota käyttävä proseduraalinen ohjelma ja vaatii tekijältään vähintäänkin yhtä paljon aikaa ja vaivaa kuin XSLT-kyselyn tekeminen.

5. Yhteenveto

XQueryn ja XSLT:n perustana on XPath-lausekkeet, jotka ovat yksinkertaisimmillaan selkeät, mutta saattavat pahimmillaan nekin "levitä käsiin". XSLT:n elementit ja dokumentin tapaan rakennettava muoto helpottavat hieman jäsentämistä XSLT:n kanssa työskennessä. XQuery käyttää FLWR-ohjausrakennetta kontrolloimaan XPath-lausekkeiden tuottamia tuloksia, mutta kielen määrittelyn heikko lenkki on sen liian toteutuslähtöinen näkökulma, joka johtaa siihen, että käyttäjän on tiedettävä tarkasti ja etukäteen hakemansa tiedon luonne ja rakenne, mikä on myös XSLT:n heikkous kyselykielenä. Ero on vain siinä, että kun XSLT ei ole alunperin tarkoitettu tiedonhakuun, XQuery on. Siksi sen lähtökohtana olisi pitänyt olla hieman deklaratiivisempi rakenne, XQueryn tämänhetkinen versio perustuu prosessointiin, joka vastaa lähinnä XSLT-kielessä käytettävän `xsl:for-each`-elementin silmukkaohjausta. Tämä ei ole hyvä tapa, vaan tarvitaan lähtökohtaisesti erilainen, deklaratiivisempi kieli.

Viiteluettelo

[Lechner et al, 2002] Stephan Lechner , Günter Preuner and Michael Schrefl. Translating XQuery into XSLT. *Lecture Notes in Computer Science* ***2465***, pp. 239-252.

[Lenz, 2002] Evan Lenz. XQuery: Reinventing the Wheel? Available as *http://www.xmlportfolio.com/xquery.html*.

[MSXSL] MSXSL-program and documentation. Available as http://www.msdn.microsoft.com/library/en-us/dnxslgen/html/msxsl.asp.

[Nykänen, 2001] Ossi Nykänen. *XML*. Docendo Finland Oy, Jyväskylä 2001.

[XML] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000. Available as http://www.w3.org/TR/2000/REC-xml-20001006/.

[XMLSchema] David C. Fallside. XML Schema Part 0: Primer. W3C Recommendation, 2 May 2001. Available as http://www.w3.org/TR/2001/ REC-xmlschema-0-20010502/.

[XPath, 1999] James Clark, Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. Available as http://www.w3.org/TR/xpath/.

[XPath, 2002] Anders Berglund et al. XML Path Language (XPath) 2.0. W3C Working Draft 15 November 2002. Available as http://www.w3.org/TR/xpath20/.

[XQuery, 2002] XQuery 1.0: An XML Query Language. W3C Working Draft 15 November 2002. Available as http://www.w3.org/TR/xquery/.

[XQuery op, 2002] XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Working Draft 15 November 2002. Available as http://www.w3.org/TR/ xquery-operators/.

[XQuery use, 2002] XQuery 1.0 Use Cases. W3C Working Draft 15 November 2002. Available as http://www.w3.org/TR/xmlquery-use-cases/.

[XSLT, 1999] James Clark. XSL Transformations (XSLT) Version 1.0 W3C Recommendation, 16 November 1999. Available as http://www.w3.org/ TR/xslt/.

[XSLT, 2002] Michael Kay. XSL Transformations (XSLT) Version 2.0 W3C Working Draft 15 November 2002. Available as http://www.w3.org/TR/xslt20/.

Visualisointien suunnittelu WWW-ympäristöön

Ilkka Lehtinen

Tiivistelmä

Tutkimuksessa tarkastellaan visualisontien suunnittelua WWW-ympäristöön. Tarkastelun alla on erityisesti sellaiset visualisoinnit, joiden avulla pyritään helpottamaan navigointia WWW-ympäristössä. Työn tarkoituksena on selvittää niitä seikkoja, joiden takia navigointiongelmia esiintyy ja esittää näille mahdollisia ratkaisuvaihtoehtoja. Työssä pohditaan minkä perustella visualisointi tulisi tehdä, miten se tulisi näyttää käyttäjälle sekä minkälainen visualisoinnin pitäisi olla käyttäjän suhteen? Tarkastelu keskittyy sellaisten ratkaisujen arvioimiseen, jotka ovat suhteellisen helposti toteutettavissa nykyiseen WWW-ympäristöön.

Avainsanat ja -sanonnat: Informaation visualisointi, navigointi, WWW-ympäristö, sivustokartta, käytettävyys.

CR-luokat: H.5.4

1. Johdanto

1.1. Taustaa

Internet on laajentunut sen syntyajoista alkaen koko ajan kasvavalla vauhdilla. Alun perin pienessä piirissä yliopistojen välillä käytetty järjestelmä on levinnyt aktiiviseen käyttöön ympäri maailmaa. Nykyään melkein jokaisella yrityksellä ja yhteisöllä on olemassa jonkinlainen edustus Internetissä enemmän tai vähemmän laajan WWW-sivuston muodossa.

Internetin syntyaikoina tekniikka oli vielä paljon kehittymättömämpää kuin nykypäivänä, eikä Internetiä suunniteltu sellaista käyttöä silmälläpitäen, mihin nykyään on totuttu. Intenetin taustalla oleva tekniikkaa ja käytäntöjä on kuitenkin hiottu siihen suuntaan, että uusien ominaisuuksien lisääminen ja joustava vastaaminen jälkeenpäin tulleisiin haasteisiin on ollut mahdollista.

Internetin suosituin osa on hypertekstisivujen verkon muodostava *WWW* (World Wide Web). Nykyisin standardit määrittelevät pitkälti, miten WWW:hen tulee tuottaa sisältöä. WWW:n ongelmaksi on kuitenkin muodostumassa se idea, jolle järjestelmä alun perin rakennettiin. Alkuperäinen ajatus

järjestelmästä, jossa dokumentit ovat sidottuina toisiinsa erilaisten viittausten avulla, oli mainio, mutta ei täysin ongelmaton. Dokumenttien muodostama verkko muodostuu ongelmaksi, koska siinä ei ole mitään järkevästi tukittavissa olevaa rakennetta. WWW-sivustoilla seikkaileva käyttäjä ei välttämättä pysty hahmottamaan sivujen suhdetta toisiin sivuihin ja siksi liikkuminen sivujen välillä saattaa tuottaa ongelmia.

Puuttuvan rakenteen korvaamiseksi on kehittynyt erilaisia menetelmiä hallita järjestymätöntä joukkoa erilaisia sivuja. Näistä ehkä käytetyimpiä ovat erilaiset WWW:n yhteyteen rakennetut hakupalvelut, joiden avulla tietomassasta voidaan etsiä tietoa tarkasti haluttujen hakutermien avulla. Hakukoneiden tarjomat palvelut eivät kuitenkaan auta WWW:n käyttäjää hahmottamaan asiakokonaisuuksia siten, että se helpottaisi käyttäjän liikkumista sivuilla. WWW:n yksi pahimmista ongelmista onkin todettu olevan navigoinnin vaikeus. On arvioitu, että navigoinnin hankaluus voi johtaa tilanteeseen, jossa WWW:n arvo hyödyllisenä tiedon välityksen kanavana laskee [Nielsen, 1999]. Osasyy tähän on, että suurin osa WWW-ympäristöön sivustoja tuottavista tahoista ei ole ottanut tarpeeksi huomioon tuottamiensa sivustojen käytettävyyttä.

1.2. Tutkimusongelma ja tutkimuksen rakenne

Ratkaisuksi navigointiongelmaan on esitetty visualisointien luomista WWW-ympäristöstä käyttäjän avuksi. Lähtökohtana pidetään sitä, että WWW-ympäristöstä tulisi tehdä kartta, jonka avulla käyttäjä voisi helpommin navigoida WWW-sivustolla. Tässä tutkimuksessa tarkastelen sitä, miten visualisointi WWW-sivustosta tulisi tehdä ja miten se tulisi näyttää käyttäjälle. Navigointia helpottavia tekijöitä on useita, ja tässä tutkimuksessa tarkasteltava visualisoinnin luominen on vain yksi mahdollinen ratkaisu navigoinnin helpottamiseksi. Tutkimuksessa sivutaan muita ratkaisumahdollisuuksia, mutta niitä ei tarkastella sen tarkemmin.

Visualisoinnin lähtökohdat ja ongelmat ovat hyvin samanlaiset kuin tavallisen kartan luomisessa. Voidaan hahmottaa kolme eri osa-aluetta, jotka ovat kiinteästi sidoksissa kartan suunnittelussa: ympäristö, kartta ja käyttäjä. Kartan avulla pyritään välittämään käyttäjälle tietoa ympäristöstä siten, että hän pystyy siinä sujuvasti kulkemaan. Karttaa voidaan tarkastella kommunikaatiokanavana, jossa kartan tekijä välittää informaatiota kartan käyttäjälle [Keates, 1982]. Samankaltaiset periaatteet ovat voimassa, kun suunnitellaan karttaa WWW-sivustosta.

Tutkimus rakentuu siten, että ensin tutkitaan WWW-sivuston erityislaatuisuutta visualisoitavana ympäristönä. Tämän jälkeen pohditaan, miten visualisointi ympäristöstä tulisi tehdä ja näyttää käyttäjälle. Kolmannessa osassa tarkastellaan käyttäjän roolia visualisoinnin käyttäjänä ja liikkujana visualisoitavassa ympäristössä. Lukujen jako vastaa siten osaltaan niitä vaiheita, joita visualisoinnin suunnittelussa ja toteutuksessa joudutaan käymään läpi.

2. Lähtökohdat visualisoinnille

Kolme tärkeintä syytä navigoinnin vaikeuteen ovat selainten puutteelliset käyttöliittymät, WWW-sivuston huono suunnittelu sekä WWW-sivujen luontiin käytetyn HTML-kielen huono sopivuus tehtäväänsä [Cockburn and Jones, 1997].

Käyttöliittymän huono käytettävyys johtuu pääosin siitä, että selain mahdollistaa vain eteen- ja taaksepäin liikkumisen, vaikka liikkuminen todellisuudessa tapahtuu usein epälineaarisesti. Ongelmaksi muodostuu navigointivälineen antama kuva lineaarisesta navigoinnista. Useat käyttäjät eivät olekaan tietoisia selainten eteen- ja taaksepäin vievien toimintojen todellisesta toimintalogiikasta [Cockburn and Jones, 1997].

WWW-sivut on usein suunniteltu siten, ettei käyttäjälle anneta mahdollisuutta hahmottaa WWW-sivustoa tarpeeksi hyvin. Hyvään suunnitteluun kuuluu myös muita osa-alueita kuin pelkästään sivuston rakenteen selkeä havainnollistaminen käyttäjälle. Kuvien käyttö epätarkoituksenmukaisesti ja sivujen sekava jäsentäminen aiheuttavat myös ongelmia [Cockburn and Jones, 1997].

WWW-sivujen kuvailuun käytetyn kielen sopimattomuus tehtäväänsä on myös yksi syy navigointiongelmiin. Kieli on suunniteltu siten, ettei se mahdollista havainnollisten linkkien tekemistä eri sivujen välille. Rajoitteita asettaa myös se, että eri selaimet tulkitsevat HTML-kieltä eri tavoin, mikä vaikeuttaa entisestään suunnittelijan työtä. [Cockburn and Jones, 1997]

Näistä kolmesta ongelma-alueesta WWW-sivujen visualisointi kuuluu sivustojen suunnittelun piiriin. Visualisoinnin tarkoituksena ei välttämättä ole suunnitella koko sivustoa uudelleen vaan auttaa nykyisiä toteutuksia parempaan käytettävyyteen jonkinlaisen visualisoinnin avulla. Käyttäjälle pyritään näyttämään selkeä esitys sivukokonaisuudesta. On kuitenkin myös mahdollista lähteä rakentamaan visualisointia jo sivun suunnittelun aikana, jolloin sivuston rakenne auttaa visualisoinnin muodostamisessa [Hahsler and Bernd, 2000]. Usein visualisointien tehtävänä on kuitenkin nykyisen toteutuksen

parantaminen, ja siten koko sivuston suunnittelu ei aina ole aiheellista tai tarpeellista.

Useissa tutkimuksissa on lähtökohtana visualisoinnin tarpeellisuudelle pidetty sitä, että visualisoinnilla pystytään helpottamaan käyttäjän navigoimista. Oletuksena on, että jos WWW-sivuston rakenne pystytään näyttämään käyttäjälle selkeästi, niin on mahdollista muodostaa sivustosta esityksen pohjalta *kognitiivinen kartta* (cognitive map) [Spence and Robert, 2001]. Kognitiivisen kartan varassa käyttäjän on helpompi liikkua sivuilla, koska hän pystyy paremmin hahmottamaan oman sijaintinsa suhteessa ympäristöön, jossa liikkuu.

3. WWW-sivusto visualisoitavana ympäristönä

3.1. Visualisoitava kohde

Onnistuneen visualisoinnin tuottamiseksi on löydettävä ne kohteet, jotka halutaan ja koetaan hyödyllisiksi visualisoida. WWW:n kohdalla vaihtoehtoja on useita, ja ongelmaksi muodostuukin visualisointiperusteiden valinta. Tässä luvussa tarkastellaan WWW-sivustoa visualisoitavana ympäristönä kiinnittäen huomiota siihen, minkälaisia visualisoitavia kohteita sieltä löytyy. Yksinkertaisimmillaan visualisointi tehdään sivuston muodostavien yksittäisten WWW-sivujen pohjalta, mutta vaihtoehtoja on muitakin.

Laajojen sivustojen visualisoinnissa yksi sivu on useasti liian pieni yksikkö siinä mielessä, että visualisointiin ei tällöin mahdu riittävän väljästi aseteltuna kaikkia kohteita. Tällöin visualisoitavien kohteiden määrää on rajoitettava luomalla erilaisia abstraktioita. Abstraktioiden luomiseksi on yksittäisiä kohteita ryhmiteltävä yhteen joidenkin sääntöjen mukaan. Ryhmittely voi tapahtua esimerkiksi rakenteen mukaan, jolloin WWW-sivuston linkkirakennetta analysoimalla löydetään tarvittavia ryhmiä. Toinen keino on tehdä ryhmittely yksittäisten sivujen sisällön perusteella. [Mukherjea and Foley, 1995a]

Ryhmittelyn lisäksi visualisoinnista voidaan jättää kokonaan pois sivut, joilla ei koeta olevan merkitystä visualisoinnin kannalta. Pois jätettävien sivujen valinta voidaan tehdä eri tavoin. Sivuja voidaan jättää pois sisällön mukaan, linkkien mukaan tai rakenteen mukaan [Mukherjea and Foley, 1995a]. Esimerkiksi jos yrityksen WWW-sivuilla on jokaisen työntekijän esittelevä WWW-sivu, ei kaikkia sivuja välttämättä tarvitse ottaa mukaan visualisointiin, vaan riittää, että visualisoinnista selviää mistä työntekijöiden WWW-sivut löytyvät.

Samankaltaisia toimenpiteitä ylimääräisen aineksen poisjättämiseksi joudutaan tekemään myös tavallisten karttojen luonnin yhteydessä. Keatesin [1982] mukaan maapallon kartoittamisen yhteydessä puhutaan kolmannesta *muunnoksesta* (transformation) eli *yleistämisestä* (generalization). Ensimmäiset kaksi muunnosta eivät ole oleellisia WWW-sivuston visualisoinnin kannalta, koska ne liittyvät kolmiulotteisen pinnan näyttämiseen kaksiulotteisella tasolla. Samat yleistämisen periaatteet voidaan kuitekin huomata myös WWW-sivustosta tehtävässä visualisoinnissa.

Abstraktioiden lisäksi visualisoitavien kohteiden määrää voidaan pienentää laskemalla kunkin sivun arvo tietyn kaavan mukaan ja vain näin löydetyt merkityksellisiksi havaitut sivut otetaan mukaan visualisointiin. Löydettyjä merkityksellisiä sivuja kutsutaan usein *maamerkeiksi* (landmark), koska ne ovat sivuilla kulkemisen kannalta erittäin oleellisia, aivan kuten reaalimaailmassa kohtaamamme maamerkit. Sivua voidaan esimerkiksi pitää maamerkkinä, jos se on linkitetty useisiin muihin sivuihin, vierailumäärältään suuri sekä korkealla koko sivuston hierarkkiassa [Mukherjea and Hara]. Kun edellä olevat kriteerit täyttyvät, voidaan WWW-sivu näyttää keskeisellä paikalla visualisoinnissa. Käyttäjää autetaan näin huomaamaan oleelliset sivut ja jättämään vähemmälle huomiolle muut sivut.

Hieman samankaltainen tekniikka on *yleistetty samankaltaisuuden analyysi* (Generalized Similarity Analysis) tai lyhemmin GSA [Chen, 1997]. Analysoimalla tällä menetelmällä WWW-sivuston sivut saadaan sivustosta eroteltua hyödyllisiä rakenteita. Menetelmää sovellettaessa WWW-sivuista etsitään linkityssuhteita sekä sisällön samankaltaisuuksia. Lisäksi otetaan huomioon käyttäjien selailutapoja WWW-sivuilla. Heikkoutena tällä tekniikalla on se, että se ei erityisesti pyri etsimään hierarkkisia rakenteita sekä se, että löydettyjen rakenteiden visualisoiminen selkeästi on hankalaa.

3.2. Visualisoitavien kohteiden yhdistäminen rakenteeksi

Samoin kuin normaalia karttaa tehtäessä ei WWW-sivustolta löydy mitään sellaista rakennetta, joka olisi valmiiksi annettu. Tavallisen kartan tekijä pystyy luomaan kartan yksiselitteisesti eri maamerkkien olemassa olon ja sijainnin perusteella. WWW-sivustolla tällaista yksiselitteistä lähtökohtaa ei ole, jolloin visualisoinnin tekijä joutuu itse keksimään järkevän tavan muodostaa rakenteita yhdistämään eri kohteita.

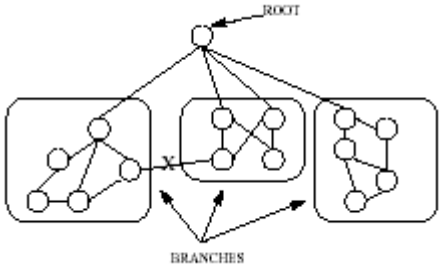
WWW:n luonteen takia voidaan visualisoinnin pohjaksi valita monta eri rakennetta. Yksinkertaisin ratkaisu on muodostaa rakenne WWW-sivujen linkkirakenteen pohjalta. Pelkän linkkirakenteen näyttäminen käyttäjälle ei

ole kuitenkaan kovin havainnollista. Yleensä linkkirakenne on näytetty graafin muodossa. Graafin piirtäminen on monimutkainen operaatio, ja graafin avulla esitetty rakenne on usein liian epäselvä ja suttuinen [Herman et al. 2000]. Graafeja käytetään kuitenkin siksi, että ne mahdollistavat myös sellaisen informaation näyttämisen, joka ei ole hierarkkisesti järjestetty. Myöhempää visualisointia ajatellen on oleellista löytää sellainen rakenne, joka on helppo visualisoida ja näyttää käyttäjälle.

Navigoinnin ja esittämisen kannalta selkeämpi vaihtoehto on esittää WWW-sivujen väliset suhteet rakenteisesti. Mahdollisia vaihtoehtoja rakenteiksi ovat perinteinen hierarkkinen rakenne, asyklinen rakenne, avaruudellinen rakenne sekä naapurustorakenne [Durand and Kahn, 1998]. Käyttäjän kannalta näistä rakenteista selkein on hierarkkinen rakenne jo senkin takia, että se on tuttu monesta muusta yhteydestä. Durand ja Kahn [1998] ovat päätyneet käyttämään MAPA-visualisoinnissaan hierarkkista rakennetta, samoin WebTOC-ohjelman kehittäjät [Nation et al. 1997]. Näistä toteutuksista kerrotaan lisää kohdissa 3.3 ja 4.4.

Edellä mainittu tapa etsiä WWW-sivuston sivujen välillä olevia yhtäläisyyksiä antaa pohjan myös rakenteen löytämiselle. Esimerkiksi linkkirakenteen tutkiminen auttaa löytämään oleelliset sivut, ja tämän avulla hierarkian pää- ja alasivut voidaan hahmottaa. Hierarkia voidaan myös rakentaa maamerkkien varaan, jolloin ylinnä WWW-sivustossa olevat maamerkkisivut ovat ylempänä hierarkiassa ja vähempiarvoiset näiden alakohtina.

Hierarkkisia rakenteita on myös mahdollista löytää WWW-sivustosta linkkirakenteen perusteella luodusta graafista. Mukherjea et al. [1995] esittävät keinon hierarkioiden löytämiseksi graafin pohjalta. Menetelmän avulla graafista pyritään etsimään aligraafeja, joista pystytään erottamaan juuri, jonka lapset muodostavat jonkin mielivaltaisen graafin (Kuva 1). Ehtona on, että aligraafien välillä ei saa olla yhteyksiä toisiinsa. Mukherjea et al. [1995] käyttävät tällaisesta aligraafista nimitystä *esipuu* (pre-tree). Mahdollisten esipuiden löytämiseksi käytetään sisältö- sekä rakenneanalyysiä. Graafin solmut ryhmitellään attribuuttien mukaan sopiviksi ryhmiksi ja rakenneanalyysin avulla etsitään esipuiden juurielementeiksi sellaisia solmuja, jotka ovat yhteydessä kaikkiin muihin solmuihin [Mukherjea et al. 1995]. Näin saadaan hahmoteltua graafista hierarkian pohjaksi sopivat rakenteet.



Kuva 1. Esipuita. [Mukherjea et al. 1995].

Joissakin toteutuksissa on päädytty ratkaisuun, jossa käyttäjä pääsee visualisointia luotaessa vaikuttamaan siihen, miten WWW-sivujen rakenne järjestetään. Esimerkiksi MAPA-ohjelmistoon kuuluva *organisoija* (organizer) tukeutuu pitkälti käyttäjän apuun sivuja luokitellessaan. Organisoija kysyy karttaa tehdessään käyttäjältä kysymyksiä sivuista. Toinen tapa muodostaa sivuille luokitteluja on sijoittaa meta-tunnisteita WWW-sivuille, joiden avulla sivuja tutkiva ohjelma tietää, miten sivun kohdalla menetellään.

WWW-sivujen merkitseminen erikseen sekä käyttäjän antamaan apuun tukeutuminen tekee sivukarttojen muodostamisesta erittäin työlästä. Parempi ratkaisu olisi, jos karttojen luominen pystyttäisiin tekemään mahdollisimman automatisoidusti. Tällöin ei sivukartan luominen olisi niin paljon kiinni WWW-sivustoilla tapahtuvista muutoksista ja visualisointi pystyttäisiin tekemään mielivaltaisesti valitusta WWW-sivustosta. Tulevaisuudessa semanttinen verkko (semantic web) auttaa visualisointien tekemistä WWW-sivustoista, kun WWW-sivuilla informaatio erotetaan selkeämmin sen esittämisestä.

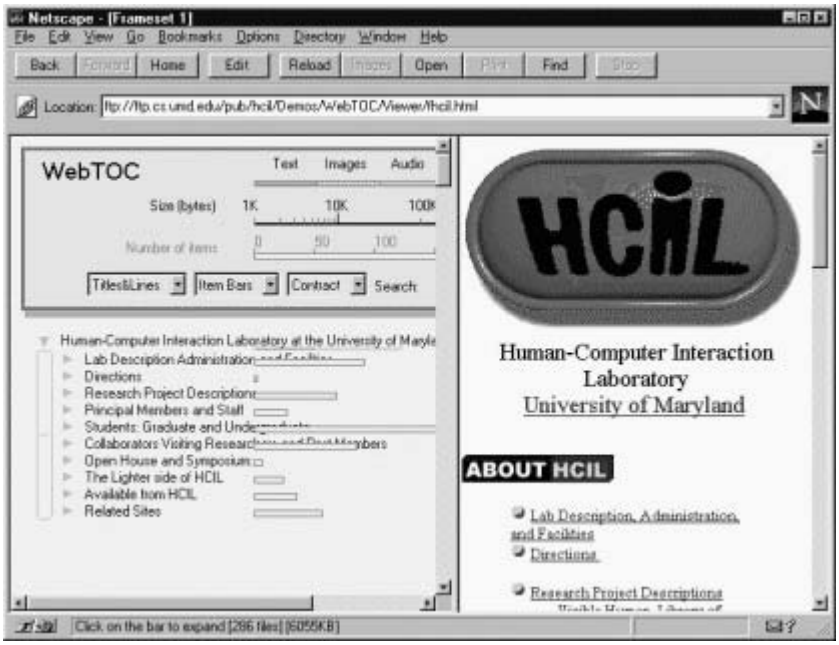
3.3. WebTOC yhtenä toteutusesimerkkinä

WebTOC [Nation et al. 1997] kehitettiin alun perin helpottamaan dokumenttien löytymistä kirjastosta, jossa dokumenttien määrä oli erittäin suuri. Kirjastossa oleva materiaali on järjestetty hierarkkiseen hakemistorakenteeseen. WebTOC:in tarkoituksena on helpottaa dokumenttien löytymistä luomalla kokoelmasta havainnollisia visualisointeja.

Ohjelma tuottaa visualisointeja joko hakemisto- tai linkkirakenteen mukaan. Ohjelmassa ei siis ole pyritty käyttämään sellaisia tekniikoita, joilla visuali-soitavan ympäristön kohteiden määrää olisi pyritty pienentämään edellä mainituilla keinoilla. Hakemistorakenteen visualisoinnin avulla voidaan hahmottaa sellaisen materiaalin muodostama kokonaisuus, jota ei ole

vielä käsitelty ja liitetty olemassa olevaan WWW-sivustoon. Linkkirakenteen visualisoinnilla pystytään hahmottamaan olemassa olevien WWW-sivujen järjestyneisyys toisiinsa nähden.

Hakemistorakenne näytetään käyttäjälle tavallisen sisällysluettelon muodossa (Kuva 2). Visualisointi on helppo tehdä, koska sisällysluettelo voidaan tuottaa suoraan hakemistorakenteen mukaan. Linkkien pohjalta tehtävä sisällysluettelo tuotetaan linkkirakenteen pohjalta siten, että usean kerran esiintyvistä linkeistä otetaan huomioon vain se, joka esiintyy korkeimmalla linkkihierarkkiassa. Myös tällä tavalla saadaan muodostettua sisällysluettelon kaltainen esitys. WebTOC:in käyttämä menetelmä linkkirakenteen järjestämiseksi hierarkkiaksi ei kuitenkaan tuota järkevää rakennetta siinä tapauksessa, että sivut on alunperin järjestetty huonosti.



Kuva 2. WebTOC [Nation et al. 1997].

Käytettävyystestien mukaan WebTOC ei nopeuttanut tiedon hakemista WWW-sivustolta, mutta käyttäjät käyttivät mieluummin WebTOC:ia apuvälineenä kuin pelkkää selainta [Nation et al. 1997]. Testihenkilöiden ja tehtävien pienen määrän vuoksi tulosten kattavuus ei kuitenkaan ole paras mahdollinen. WebTOC antaa kuitenkin kuvan siitä, miten yksinkertaisen rakenteen esittäminen WWW-sivustosta voi auttaa käyttäjää.

Jos WebTOC:in toteutuksessa olisi hyödynnetty abstraktioiden tekemistä WWW-sivuista, olisi tulos saattanut olla parempi. Visualisoinnin näyttäminen sisällysluettelon muodossa hierakkisena rakenteena mahdollistaa kuitenkin

suuren informaatiomäärän näyttämisen käyttäjälle tiiviissä muodossa ilman abstraktioiden tekemistä.

4. Visualisointi karttana WWW-sivustolle

4.1. Visualisoitavien kohteiden esittäminen

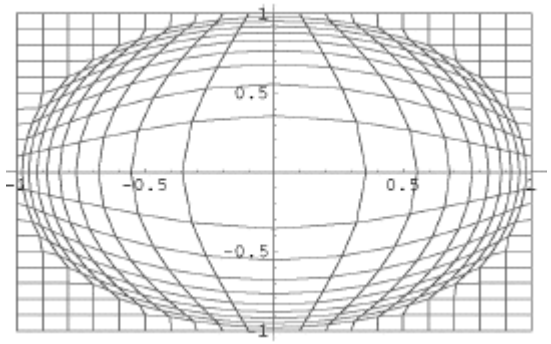
Sivukartan tarkoituksena on tarjota käyttäjälle näkymä WWW-sivuston muodostamaan kokonaisuuteen. Tämän luvun tarkoituksena on muodostaa kuva niistä mahdollisuuksista, joita on olemassa kartan näyttämiseksi käyttäjälle. Internet tarjoaa ympäristönä mahdollisuuksia visualisoinnin näyttämiseen tekstinä, grafiikkana tai kolmiulotteisena esityksenä. Visualisoinnin säilyminen helppokäyttöisenä asettaa kuitenkin rajoitteita potentiaalisille vaihtoehdoille. Kolmiulotteinen malli voi hyvinkin olla havainnollisin, mutta voi käytännössä osoittautua hankalaksi ja mahdottomaksi vaihtoehdoksi sekä käyttäjän, että toteuttajan kannalta.

Perinteisissä kartoissa visualisointi on pitkälti sidottu visualisoitavan ympäristön asettamiin vaatimuksiin. WWW:hen toteutettu visualisointi on lähtökohdiltaan paljon vapaampi. Visualisoinnin tulisi kuitenkin jossain määrin vastata sellaisia esitystapoja, joihin ihmiset ovat tottuneet. Voidaankin olettaa, että sellaisten visualisointien omaksuminen ja käyttäminen on helpompaa, jotka liittyvät vertauskuvallisesti johonkin jo aiemmin tuntemaamme. Dieberger ja Frank [1998] esittävät idean, että visualisoinnin perustaminen johonkin tunnettuun metaforaan voi olla hyödyllinen lähtökohta. Visualisointi voidaan esimerkiksi perustaa vertauskuvalle kaupungista, jolloin WWW-sivustosta visualisointia tehtäessä eri sivut ja niiden abstraktiot voitaisiin esittää kaupungista löytyviä kohteita hyödyntäen. Yksi talo esittäisi yhtä WWW-sivua, talon ikkunat linkkejä tältä sivulta muille sivuille ja niin edelleen. Ongelmana vertauskuvallisissa esityksissä on kuitenkin niiden vaatima suhteellisen suuri esitystila, koska visualisointavien kohteiden suhteet toisiinsa joudutaan esittämään tilallisesti [Dieberger and Frank, 1998].

Riippumatta siitä, käytetäänkö visualisoinnissa hyödyksi jotain vertauskuvaa, täytyy visualisoinnissa käytettyjen symbolien olla käyttäjän ymmärrettävissä. Kartoissa käytetyt symbolit voivat olla joko tavanomaisia kuvioita, joiden merkityksestä on sovittu käyttäjän kanssa, tai sitten valmiiksi symboliarvoa omaavia kuvioita [Keates, 1982]. WWW-sivu voidaan esimerkiksi esittää mustana ympyränä visualisoinnissa, jolloin sen merkityksestä on erikseen sovittu käyttäjän kanssa. Vaihtoehtoisesti sivu voidaan myös näyttää

dokumenttia esittävänä ikonina, jolloin sen merkitys on käyttäjälle jo etukäteen selvä. WWW-sivustoa visualisoitaessa erona tavallisiin karttoihin on se, että WWW-sivun näyttäminen symbolina ei riitä, vaan yleensä sivun merkitys selviää käyttäjälle vasta, kun se esitetään tekstuaalisessa muodossa. Tekstin näyttäminen visualisoinnissa on yksi vaikeimpia ongelmia visualisointia tehtäessä ja vaikuttaa suuresti siihen, miten kohteet voidaan visualisointiin asetella.

Samalla tavoin kuin perinteisissä kartoissa, määräytyy kartan sisältö ja ulkoasu sen mukaan, mihin tarkoitukseen kartan käyttäjä aikoo sitä käyttää. Perinteisen kartan sijasta visualisointia WWW-sivustosta ei yleensä käytetä reitin etsimiseen, vaan jonkun tietyn tiedon löytämiseen. Tämän takia sivujen välisien suhteiden näyttäminen ei aja samaa asiaa visualisoinnissa WWW-sivustosta kuin tavallisessa kartassa, jonka avulla pyritään hahmottamaan reitti kahden pisteen välillä. WWW-ympäristössä reitillä ei ole oleellista merkitystä. Tärkeämpää on se, missä jokin WWW-sivu sijaitsee.



Kuva 3. Kalansilmä-vääristymä. [Herman et al. 2000].

4.2. Tiedon mahtuminen visualisointiin

Visualisoitava ympäristö on usein niin laaja, ettei sitä pystytä näyttämään yhdessä staattisessa kuvassa ilman, että kuvasta tehdään luonnottoman suuri. Tämä ongelma on yleensä hoidettu siten, että visualisoinnista on kerrallaan näkyvissä vain tietty osa. On eri keinoja rajoittaa visualisoinnin näyttämää kuvaa ympäristöstä. Yleensä kuvaan mahdutetaan enemmän tietoa vääristämällä sitä jollakin keinolla.

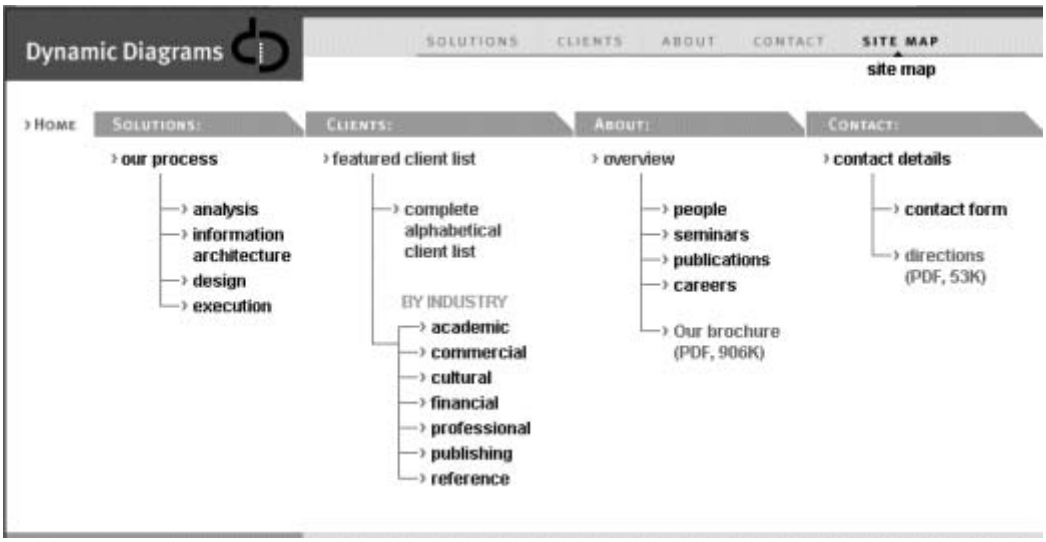
Esimerkkejä vääristämisen mahdollisuuksista on useita. Esimerkiksi Furnasin kehittämä kalansilmä-tekniikka (Kuva 3), perspektiivinen seinä (perspective wall) sekä erilaiset suurennuslasit. Käyttäjän kannalta oleelliset kohteet voidaan näyttää suuremmalla tarkkuudella kuin vähemmän tärkeät kohteet. Edellä mainitut vääristämisen keinot kuuluvat Focus+Context näkymiin. Tekniikoiden käyttämisessä on se hyöty, että käyttäjä pystyy

säilyttämään kokonaiskuvan visualisoitavasta ympäristöstä, vaikka ei näekään tarkasti koko visualisoitavaa ympäristöä. Kun epäoleelliset kohteet näytetään visualisoinnin laidoilla pienemmällä tarkkuudella, pystyy käyttäjä hahmot-tamaan suhteensa kohteisiin, ilman että ne veisivät turhan suuren tilan näytöltä. Tekniikoiden käyttöä WWW-sivustojen visualisoinnissa rajoittaa kuitenkin niiden vaatima suuri laskentateho sekä toteuttamisen hankaluus WWW-ympäristöön.

Sen lisäksi että esitettävä kuva rajataan näyttämällä vain käyttäjän kannalta oleelliset asiat, voidaan käyttöliittymään toteuttaa interaktiivisuutta käyttäjän kanssa ja antaa näin käyttäjälle mahdollisuus myös itse osallistua näkymän tuottamiseen. Interaktiivisuus ja näkymän rajoittaminen kulkevat yleensä käsi kädessä siten, että käyttäjälle annetaan mahdollisuus vaikuttaa rajaukseen. On havaittu, että ihminen sisäistää esityksen paremmin, jos hän pääsee itse vaikuttamaan siihen. Tätä asiaa käsitellään tarkemmin seuraavassa luvussa.

4.3. Visualisointien ulottuvuudet

Yksinkertaisimpia visualisointeja ovat tekstipohjaiset visualisoinnit. Tekstipohjaisessa visualisoinnissa välineenä käytetään tekstiä, joka on esitetty käyttäjälle yleensä hierarkkisessa muodossa (Kuva 4). Tekstipohjaisessa visualisoinnissa kohteen täytyy olla sellainen, että se pystytään esittämään järkevästi käyttäjälle mahdollisimman lyhenä merkkijonona; yleensä merkkijono on joko yksittäinen sana tai lause.

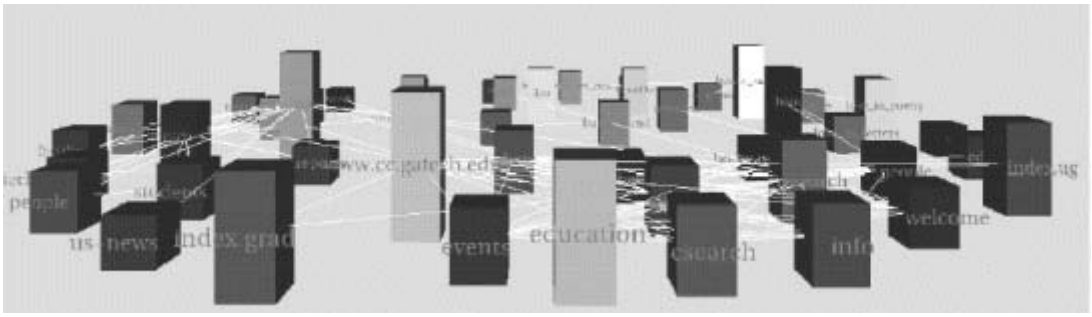


Kuva 4. Tekstipohjainen sivukartta [Dynamic Diagrams].

Tekstipohjainen visualisointi on nykyään WWW-sivustoilla yleisemmin käytetty visualisointi. Tekstipohjaisen visualisoinnin suosio johtunee ehkä juuri siitä, että se on suhteellisen helppo muodostaa. Heikkoutena on kuitenkin se, ettei pelkällä tekstin asettelulla pystytä esittämään kuin yksinkertaisia rakenteita. Yleisimmät tavat esittää visualisointi tekstin avulla ovat sisällysluettelon kaltainen rakenne WWW-sivujen hierarkkisen rakenteen pohjalta ja taulukon muotoinen esitys, jossa WWW-sivut ovat jaettu eri osioihin.

Visualisointiin voidaan mahduttaa enemmän tietoa esitettävän kohteen rakenteesta, jos visualisointi esitetään käyttäjälle kuvan muodossa. Samoin kuin tekstipohjaisessa esityksessä voidaan osa informaatiosta esittää tekstin avulla. Tämän lisäksi voidaan esityksestä tehdä havainnollisempi lisäämällä visualisointiin kohteiden välisiä suhteita sekä muuta informaatiota. Rakenteiden näyttäminen kuvassa on paljon vapaampaa, eikä visualisoitavan rakenteen tarvitse välttämättä olla rajoitettu tietyntyyppiseen hierarkkiaan, vaan kohteiden suhteet voivat olla monimuotoisempia.

Kolmiulotteisuuden käyttäminen tuo esitykseen vielä yhden ulottuvuuden lisää ja antaa mahdollisuuden esittää jonkin verran enemmän tietoa kuin kaksiulotteisessa visualisoinnissa. Monet visualisoinnit eivät ole aidosti kolmiulotteisia, vaan kolmiulotteinen maailma projisoidaan kaksiulotteiselle pinnalle, joka näytetään käyttäjälle. Jotta tällainen visualisointi olisi tarpeeksi havainnollinen, on käyttäjälle yleensä annettu mahdollisuus tarkastella visualisointia eri kulmista tekemällä esityksestä interaktiivinen (ks. kohta 5.2).



Kuva 5. Kolmiulotteinen visualisointi [Mukherjea and Hara].

Aidosti kolmiulotteisissa visualisoinneissa käyttäjä pääsee liikkumaan WWW-sivustosta tehdyssä kolmiulotteisessa mallissa (Kuva 5). Perusteena kolmiulotteisen maailman käytölle on se, että se on ihmiselle intuitiivisempi kuin tavalliset kaksiulotteiset käyttöliittymät [Chen, 1998]. Kolmiulotteisen käyttöliittymän käytön oppimisen on ajateltu myös olevan helpompaa ja

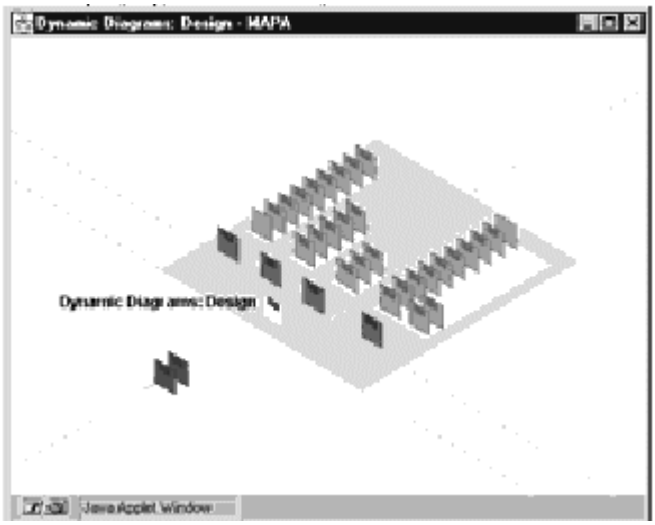
havainnollisempaa käyttäjälle, koska maailmassa täytyy itse liikkua ja sen kanssa pystyy olemaan vuorovaikutuksessa [Chen, 1998].

4.4. MAPA

MAPA [Durand and Kahn, 1998] on järjestelmä, jonka avulla voidaan luoda visualisointeja WWW-sivustoista. Suunnittelussa on otettu huomioon eri asioita, joiden oletetaan vaikeuttavan navigointia, ja tältä pohjalta on kehitetty navigointia helpottava ratkaisu. Järjestelmä koostuu eri osista, joiden avulla luodaan käyttäjälle interaktiivinen näkymä WWW-sivuston rakenteesta.

MAPA selvittää WWW-sivuston rakenteen tutkimalla sen linkkirakennetta pääosin samalla periaattella kuin WebTOC. Linkkirakenteen selvittämisen jälkeen linkit järjestetään hierarkiaksi erillisellä ohjelmalla. Hierarkian luomisessa käyttäjä on oleellisessa osassa määrittelemässä hierarkian rakennetta.

MAPA:n tuottama visualisointi näytetään käyttäjälle sovelteena. Visualisointi on osaksi kolmiulotteinen siinä suhteessa, että käyttäjä näkee viistosti ylhäältäpäin tason, johon eri sivut ovat aseteltu (Kuva 6). Näkymässä keskeisellä sijalla on se tarkasteltava WWW-sivu, jonka suhteen näytetään ne sivut, joilla käyttäjä on ollut ja joille käyttäjä voi tästä edetä. Yksittäiset WWW-sivut on esitetty pieninä ikoneina, joiden kohdalla näytetään sivun otsikko käyttäjän osoittaessa ikonia hiirellä.



Kuva 6. MAPA [Durand and Kahn, 1998].

MAPA:n tuottamaan visualisointiin saadaan mahtumaan suuri määrä WWW-sivuja ilman, että näkymää tarvitsee vääristää mitenkään. Osittain

kolmiulotteista näyttöä käyttämällä sivujen ikonit pystytään näyttämään lomittain ja saadaan näin mahtumaan pieneen tilaan. MAPA tarjoaa myös tekstipohjaisen linkkilistan, jonka avulla käyttäjä näkee kullakin hetkellä oleelliset linkit navigoinnin kannalta: missä hän on vieraillut, missä hän on tällä hetkellä ja mihin hän voi seuraavaksi edetä.

5. Käyttäjä ja WWW-sivuston kartta

5.1. Kartta käyttäjän näkökulmasta

Edellisissä luvuissa on käsitelty niitä taustatekijöitä, joita pitää ottaa huomioon tehtäessä visualisointia WWW-sivustosta. Lisäksi on käsitelty WWW-sivustoa visualisoitavana ympäristönä sekä visualisoinnin näyttämistä. Tässä luvussa tarkastellaan visualisointia käyttäjän näkökulmasta. Visualisoinnin tulee olla käyttäjän kannalta hyödyllinen ja helppokäyttöinen. Kartan tekeminen käyttäjän kannalta mahdollisimman hyödylliseksi vaatii, että navigointi-ongelmien aiheuttajat ovat tiedossa. Pitäisi tietää, mistä navigoinnin ongelmat alun perin johtuvat, jotta pystyttäisiin tekemään käyttäjän ongelmiin vastaava toteutus. Navigointiongelmien aiheuttajia on kuitenkin tutkittu melko vähän [Cockburn and Jones, 1997].

Visualisointi voidaan käyttäjän kannalta toteuttaa kahdella eri tavalla. Joko se on staattinen näkymä sivuston rakenteeseen tai käyttäjän huomioon ottava dynaaminen kartta [Cockburn and Jones, 1997]. Staattinen kartta esittää yleensä koko WWW-sivuston rakenteen yhdessä näkymässä. Käyttäjä pystyy hyödyn-tämään tätä kartaa rakentaessaan mielikuvaa WWW-sivuston rakenteesta. Käyttäjän ajatellaankin rakentavan kognitiivista kartaa WWW-sivustosta tar-kastellessaan staattista visualisointia. Dynaaminen kartta ei välttämättä esitä kokonaisuutta, vaan on riippuvainen käyttäjän tilanteesta ja sijainnista WWW-sivustolla. Käyttäjälle esitetään vain sellaista tietoa, jonka ajatellaan olevan kussakin tilanteessa oleellista. Voidaan esimerkiksi esittää tietoja siitä, mihin käyttäjän on mahdollista seuraavaksi edetä, mistä hän on tulossa ja niin edelleen. Staattiseen karttaan saadaan helposti dynaamisuutta esimerkiksi näyttämällä käyttäjälle hänen sijaintinsa kartalla.

Dynaamisuuden lisääminen navigointiympäristöön aiheuttaa yleensä sen, että vaadittavan laskentatehon vaatimus kasvaa ja joudutaan käyttämään monimutkaisempia ratkaisuja selainten suhteen kuin normaalisti. Staattisen sivukartan näyttämiseen riittää yleensä tavallinen HTML-sivu, kun dynaamisesti tuotettu esitys joudutaan rakentamaan ohjelmallisesti joko palvelimella tai käyttäjän puolella [Cockburn and Jones, 1997].

5.2. Kartan interaktiivisuus

Jotta WWW-sivuston kartta olisi hyödyllinen, olisi siinä käyttäjän kannalta interaktiivisia ominaisuuksia. Interaktiivisuuden luominen visualisointiin riippuu pitkälti siitä, minkälainen visualisointi on. Jos WWW-sivuston rakenne näytetään tekstin avulla sisällysluettelona, on yleisin tapa lisätä interaktiivisuutta antamalla käyttäjälle mahdollisuus näyttää tai piilottaa eri kappaleiden alaosiota. Näin käyttäjä pystyy aktviivisesti käyttämään visualisointia ja hahmottamaan samalla rakennetta. Jos taas WWW-sivuston rakenne näytetään graafin muodossa, voidaan interaktiivisuus toteuttaa esimerkiksi siten, että käyttäjä voi itse kohdentaa näyttöä sille alueelle, josta on kiinnostunut.

Kuten kolmannessa luvussa todettiin, pitää visualisointia luotaessa rajoittaa visualisointiin mukaan otettavien kohteiden määrä jollakin tavoin. Interaktiivisuuden tuominen visualisointiin voidaan myös nähdä yhdeksi keinoksi ratkaista kohteiden suuren määrän aiheuttamia ongelmia. Käyttäjälle annetaan mahdollisuus itse määritellä, mitä hän haluaa visualisoinnissa nähdä. WebTOC:n [Nation et al. 1997] kohdalla interaktiivisuutta on hyödynnetty siten, että käyttäjä pystyy avaamaan ja sulkemaan sisällysluettelon haaroja sen mukaan, mitkä ovat hänen kannaltaan oleellisia.

WWW-ympäristössä on moinia eri tekijöitä, jotka rajoittavat interaktiivisuuden lisäämistä visualisointeihin. Interaktiivisen visualisoinnin luominen vaatii, että otetaan käyttöön kehittyneempiä toteutusratkaisuja kuin perinteinen asetelma, jossa selaimen avulla käydään läpi WWW-sivuja. Oikeastaan ainoa interaktiivisuuden muoto WWW-ympäristön alkuperäisessä toteutuksessa oli mahdollisuus siirtyä dokumentista toiseen linkkien avulla. Nykyään interaktiivisten ratkaisuiden tekeminen on mahdollista monella eri tekniikalla, joista yleisimpänä voidaan mainita Javalla tehdyt sovelmat.

Uudempien tekniikoiden ongelmana on, että ne ovat osaltaan erillisiä perinteisistä WWW-ympäristössä käytetyistä tekniikoista ja käyttäjien vieroksumia. Koska nopeus on tärkein tekijä WWW-ympäristön käytet-tävyydessä [Nielsen, 1999] on ymmärrettävää, etteivät esimerkiksi Java- sovelteet ole kaikin puolin sopivia tähän ympäristöön. Interaktiivisuuden luominen on kuitenkin melkein yksinomaan näiden tekniikoiden hyödyn-tämisen varassa.

5.3. Kartan suhde selaimeen

Käyttäjän kannalta navigoinnin hankaluuden aiheuttaa usein käyttäjän puutteellinen tieto selainten toimintalogiikasta [Cockburn and Jones, 1996].

Tämän lähtökohdan mukaan ei riitä, että muodostetaan käyttäjän avuksi sivukartta, vaan tämän lisäksi sivukartasta on tehtävä sellainen, että se auttaa käyttäjää myös ymmärtämään selaimen toimintaa ja helpottaa navigointiratkaisuiden tekemistä. Tällä periaatteella toimii muun muassa WebNet-ohjelma [Cockburn and Jones, 1996]. Ohjelman avulla pyritään ratkaisemaan navigoinnissa löytyneitä ongelmakohtia, joita on kolme: käyttäjän vääränlainen mielikuva selaimen navigointitoiminnoista, käyttäjän paikkatietoisuuden puutteellisuus sekä käyttäjän muistin ylikuormittaminen [Cockburn and Jones, 1996]. WebNet toimii selaimen rinnalla käyttäjän apuna yrittäen paikata perinteisten selainten puutteellisuuksia. Käyttäjän liikkuessa WWW-sivustolla WebNet piirtää omaan ikkunaansa graafisen esityksen käyttäjän liikkeistä. Esityksen avulla näytetään käyttäjälle graafisesti navigointitapahtuma ja tällä tavoin helpotetaan käyttäjän toimintaa.

Toinen mahdollisuus tehdä kartasta käyttäjän kannalta dynaaminen on sijoittaa kartta WWW-sivustolle siten, että käyttäjän siirtyyessä karttasivulle karttasivu luodaan dynaamisesti sen mukaan, mistä käyttäjä on kartalle tulossa. Tällä tavalla tehtynä kartta ottaa huomioon käyttäjän, mutta ei vaadi mitään ylimääräistä ohjelmaa selaimen rinnalle. Tämänkaltaisia ratkaisuja on pohdittu vähemmän, vaikka ne ovat lähempänä tavallisen kartan käyttötapaa, jolloin karttaa käytetään silloin, kun ei tiedetä missä ollaan.

6. Päätelmät

Tässä tutkimuksessa on hahmoteltu lyhyesti niitä asioita, joita joudutaan ottamaan huomioon suunniteltaessa visualisointia WWW-sivustosta. Alueelta ei ole vielä tehty laajempaa tutkimusta osittain senkin vuoksi, että alue on vielä suhteellisen uusi. On kuitenkin monia aloja, joiden aiempia tutkimustuloksia voidaan soveltaa suunniteltaessa toimivaa visualisointia WWW-ympäristöstä. Näitä alueita ovat käytettävyystutkimus, algoritmitutkimus, informaation visualisointi sekä tutkimus karttojen tekemisestä. Tutkimusta on tehty vähemmän siitä, miten ihmiset ymmärtävät ja ennen kaikkea navigoivat hierakkisissa rakenteissa. Niin ikään on vielä avoinna kysymys siitä, onko hierarkkinen rakenne paras mahdollinen tapa toteuttaa visualisointi WWW-ympäristöstä.

Kolmannessa luvussa käytiin läpi erilaisia lähtökohtia, joita WWW-ympäristö antaa visualisoinnille. Tämä alue on eniten tutkittu osa-alue tällä visualisoinnin saralla. Erilaisten rakenteiden ja hierarkioiden löytämiseksi WWW-ympäristöstä on tehty monenlaisia tutkimuksia. Tutkimuksissa on selvinnyt niitä periaatteita, joiden pohjalle visualisointi rakennetaan. Sen

sijaan tutkimusten perusteella vaikeampi tehtävä on löytää hyödyllisin keino näyttää visualisointi käyttäjälle. Erilaisia tapoja on monia ja oikean valitseminen on vaikeaa. Käyttäjä hyötyy eniten ratkaisusta, jossa yhdistetään kolme- ja kaksiulotteinen visualisointi.

Viidennessä luvussa tarkasteltiin visualisointia käyttäjän näkökulmasta. Tarkasteltavana oli käyttäjän rooli visualisoinnin suhteen. Luvussa käytiin myös läpi käyttäjän roolia WWW-ympäristössä liikkujana ja sitä, minkälaisia tapoja kulkea WWW-sivuilla ihmisillä on nykyään. Havaittiin, että WWW-sivustojen latautumisen nopeus on tärkein tekijä käytettävyyden kannalta. Tältä pohjalta olisi aiheellista pyrkiä kehittämään sellaisia visualisointeja, jotka olisivat nopeita ja yksinkertaisia käyttää. Kuten Nielsen [1999] totesi, eivät käyttäjät yleensä tarkastele WWW-sivuja sen tarkemmin vaan kulkevat nopeasti yrittäessään löytää etsimäänsä. Siksi olisi oleellista saada visualisointi upotettua huomaamattomasti osaksi WWW-sivustoa, jolloin käyttäjällä olisi paremmat mahdollisuudet hyötyä siitä.

Nykyisellään WWW-sivuilla olevat visualisoinnit ovat hyvin yksinkertaisia tai ne eivät ole aktiivisessa käytössä. Tulevaisuudessa navigointia on mahdollista helpottaa erilaisilla visualisointimenetelmillä. WWW-ympäristö asettaa kuitenkin rajoituksia toteutusmahdollisuuksille. Visualisoinnit ovat vain yksi keino ratkaista navigoinnin ongelma, ja jääkin nähtäväksi, millainen ratkaisu menestyy tulevaisuudessa parhaiten.

Viiteluettelo

[Chen, 1997] Chen C., Structuring and visualising the WWW by generalised similarity analysis. In: *Proc of the 8th ACM Conference on Hypertext*, 177 - 186, 1997.

[Chen, 1998] Chen C., Bridging the gap: the use of pathfinder networks in visual navigation. *Journal of Visual Languages and Computing* **9** (1998), 267 - 286.

[Cockburn and Jones, 1996] Cockburn A. and Jones S., Which way now? analysing and easing inadequacies in WWW navigation, *International Journal of Human-Computer Studies* **45** (1996), 105 – 129.

[Cockburn and Jones, 1997] Cockburn A. and Jones S., Design issues for World Wide Web navigation visualisation tools. In: *Proc of RIAO'97: The Fifth Conference on Computer-Assisted Research of Information*. 55 - 74, 1997.

[Dieberger and Frank, 1998] Dieberger A. and Frank U., A city metaphor to support navigation in complex information spaces. *Journal of Visual Languages and Computing* **9** (1998), 597 – 622.

[Durand and Kahn, 1998] Durand D. and Kahn P., MAPA: A system for inducing and visualizing hierarchy in websites. In: *Proc. of the International Conference on HyperText '98, 1998.*

[Dynamic Diagrams] Dynamic Diagrams Homepage. http://www.dynamicdiagrams.com/ *(*Referenced 20.11.2002).

[Hahsler and Bernd, 2000] Hahsler M. and Bernd S., User-centered navigation re-design for web-based information systems. In: *Proc of the 6th Americas Conference on Information Systems (AMCIS 2000)*, 192 - 198.

[Herman et al. 2000] Herman I., Melanon G., Marshall M., Graph visualization and navigation in information visualization: a survey, *IEEE Transactions on Visualization and Computer Graphics* **6** (2000), 24 – 43.

[Keates, 1982] Keates J. S., *Understanding Maps.* Longman, London. 1982.

[Mukherjea and Foley, 1995a] Mukherjea S. and Foley J.D., Visualizing the world-wide web with the navigational view builder, *Computer Networks and ISDN System* **27**, 6 (Apr.1995), 1075 – 1087.

[Mukherjea and Foley, 1995b] Mukherjea S. and Foley J.D., Showing the context of nodes in the World-Wide Web, In: *Proc of ACM CHI'95 Conference on Human Factors in Computing Systems,* volume 2 of Short Papers: Web Browsing, pages 326 - 327, 1995.

[Mukherjea et al., 1995] Mukherjea S., Foley J.D. and Hudson S., Visualizing complex hypermedia networks through multiple hierarchical views. In: *Proc of CHI '95 conference on Human Factors in Computing Systems*, 331 - 337, 1995.

[Mukherjea and Hara, 1997] Mukherjea S. and Hara Y., Focus+context views of World-Wide Web nodes. In: *Proc of the Eighth ACM Conference on Hypertext*, 187 – 196, 1997.

[Nation et al. 1997] D. Nation, C. Plaisant, G. Marchionini and A. Komlodi. Visualizing websites using a hierarchical table of contents browser: WebTOC. In *Proc. of the 3rd Conference on Human Factors and the Web*, 1997.

[Nielsen, 1999] Nielsen J., User interface directions for the web. *Communications of the ACM* **42** (1999), 65 - 72.

[Spence and Robert, 2001] Spence R., *Information Visualization*. Harlow Addison-Wesley, 2001.

Extreme Programming

Harri Lindberg

Tiivistelmä

Tutkimusten mukaan ohjelmistoprojektien onnistumistodennäköisyys ei ole kovin hyvä. Projektit myöhästelevät ja ylittävät aikataulunsa. Ohjelmistoprojektien hallintaan on käytetty perinteisiä menetelmiä kuten vesiputousmallia, mutta niiden noudattaminen hyvinkään dokumentoidun laatujärjestelmän puitteissa ei useinkaan tuo riittävää apua ongelmien parissa kamppaileville projektipäälliköille.

Muutaman viime vuoden aikana on keskusteltu uusista projektinhallintametodologioista, joilla näitä ongelmia voitaisiin välttää tai vähentää. Näistä eniten esillä on ollut Extreme Programming, josta käytetään myös yleisesti lyhennystä XP. Tämän tutkimuksen tarkoituksena on tutustua kyseiseen metodologiaan sekä tehdä vertailevaa tutkimusta sen ja perinteisten metodologioiden välillä. Lopussa on esitetty projektimalleja, joiden toteuttamisessa XP:stä saattaisi olla apua. Toisaalta tarkoituksena on ollut myös suhtautua XP:hen terveen kriittisesti ja etsiä siitä mahdollisimman luotettavaa tietoa, jotta tiedettäisiin, milloin sen käyttö ei ole järkevää. Myös tällaisia projektimalleja ja reunaehtoja on esitetty tutkimuksen lopussa.

Avainsanat ja -sanonnat: Extreme Programming, projektinhallinta, ohjelmistokehitys.
CR-luokat: K.6.3, K.6.1, D.2.9.

1. Ohjelmistojen rakentaminen projekteissa

Ymmärtääksemme ohjelmistoprojekteja on syytä ensin syventyä yleiseen projektin käsitteeseen. Suomen Standardoimisliiton projektitoimintasanasto [SFS 1981] määrittelee projektin seuraavasti:

> “Projekti on varta vasten muodostetun organisaation määrätarkoitusta varten toteuttama ainutkertainen hanke, jonka laajuus- ja laatutavoitteet sekä aika- ja kustannuspanokset on ennalta määritetty.”

Ohjelmistoprojekteissa tämä määritelmä täyttyy hyvin. Sillä on tarkoitusta varten muodostettu projektiryhmä, jossa eri asiantuntijoilla on erilaisia

rooleja. Ohjelmistoprojekti on ainutkertainen hanke, sillä sen valmistuttua projektin lopputuloksia voidaan helposti monistaa. Projektilla on myös määritellyt tavoitteet sekä niiden tavoittamiseen tarvittavat aika- ja kustannustavoitteet. Kaikki tavoitteet on kirjattu projektin alussa syntyviin dokumentteihin.

Määritelmä vastaa myös siihen, miksi ohjelmistojen rakentaminen on perinteisesti ollut nimenomaan projektimuotoista toimintaa. Ohjelmiston rakentamisprosessilla nähdään olevan selkeä tavoite, selkeät alku- ja loppupisteet, ne tarvitsevat aina juuri tietynlaista asiantuntemusta onnistuakseen ja projektien laajuus- ja laatutavoitteet ovat projektia valmisteltaessa tiedossa. Ohjelmistojen helppo monistettavuus takaa ainutkertaisuuden.

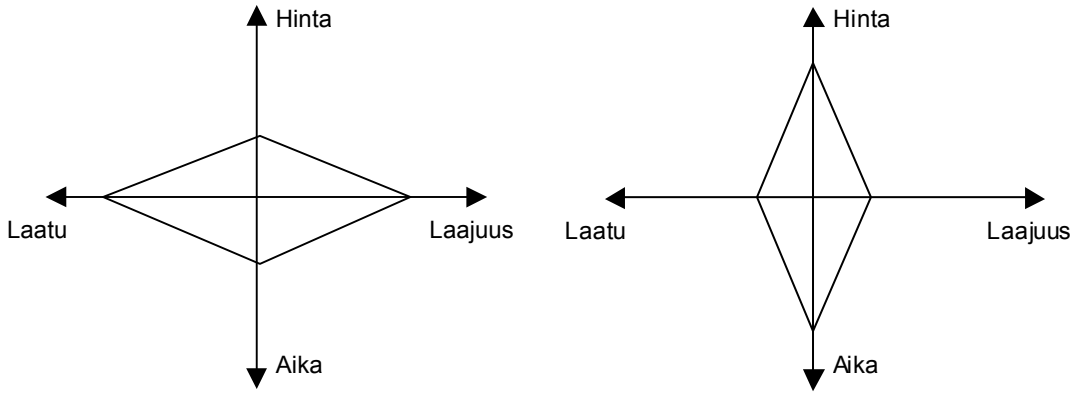
2. Neljä muuttujaa

Käytetystä määritelmästä saadaan erotettua myös neljä projektiin vaikuttavaa tekijää, jotka ovat laajuus, laatu, tarvittava aika ja kustannus:

- Ohjelmiston optimaalisella laajuudella tarkoitetaan, että ohjelmistossa on kaikki ne ominaisuudet, jotka asiakas on siihen toivonut, eikä mitään ylimääräistä.
- Ohjelmiston laadun ominaisuuksia määrittelemään tehty ISO 9126-standardi [Pressman, 2000] antaa laadukkaalle ohjelmistolle kuusi perusominaisuutta: toiminnallisuus, luotettavuus, käytettävyys, tehokkuus, ylläpidettävyys ja siirrettävyys. Nämä käsitteet ovat kuitenkin varsin moniselitteisiä.
- Tarvittavalla ajalla tarkoitetaan ohjelmiston rakentamiseen käytettyä aikaa esitutkimusprojektin aloittamisesta käyttöönoton tai mahdollisen koulutuksen päättymiseen.
- Kustannuksella tarkoitetaan ohjelmiston kokonaishintaa. Toimittajan projektityön lisäksi kustannuksiin tulee sisällyttää myös lisenssimaksut, asiakkaan kustannukset määrittely- ja käyttöönottotyöstä sekä mahdollisesta koulutukseen osallistumisesta.

Erilaiset organisaatiot asettavat projektiensa tavoitteita sen mukaan, kuinka paljon heillä on käytettävissään rahaa ja aikaa projektin läpivientiin. Ohjelmiston täydellinen laajuus ja huippulaatu ovat aluksi jokaisen organisaation tavoitteita, mutta usein joudutaan huomaamaan, että niihin tarvittavat aika- ja kustannusresurssit voivat olla organisaatiolle kestämättömiä. Esimerkiksi sellaiset virheettömyystavoitteet, jotka asetetaan sädehoitolaitteille, ovat erittäin kalliita ja aikaavieviä niiden vaatiman huolellisen testauksen takia.

Näin ollen projektin tavoitteisiin liittyy valinta toisaalta tarvittavan ajan ja kustannusten, toisaalta ohjelmiston laadun ja laajuuden välillä. Tätä valintaa voidaan havainnollistaa kuvan 1 nelikentällä.



Kuva 1: Erilaisia projektitavoitteiden nelikenttämalleja.

Nuolen suunnan kasvu kuvaa asiakkaalle edullisempaa tilannetta, toisin sanoen esimerkiksi tarvittavan rahasumman kasvu näkyy alempana olevana pisteenä hintaa kuvaavalla akselilla. Kuvassa 1 vasemmalla puolella kuvattu projekti tavoittelee korkeaa laatua ja toimintojen laajuutta. Projektista ollaan myös valmiita maksamaan enemmän ja hyväksytään se, että ohjelmiston rakentaminen kestää kauemmin. Tällainen malli on tyypillinen silloin, kun ohjelmistoa käytetään kriittisissä toiminnoissa kuten ydinvoimaloissa tai henkeä ylläpitävissä sairaalalaitteissa. Malli on käytössä myös, kun ohjelmiston päivittäminen on kallista, vaikeaa tai jopa mahdotonta. Tällaisia tapauksia ovat esimerkiksi integroidut ohjelmistot matkapuhelimiin tai avaruusluotaimiin.

Kuvassa yksi oikealla puolella kuvattu projekti tavoittelee aikataulu- ja kustannussäästöjä tinkimällä ohjelmiston laajuudesta ja osin myös laadusta. Projektimalli on yleinen pienissä organisaatioissa, joilla ei ole riittävän suurta budjettia rakentaa ohjelmistoa kerralla kuntoon. Projektimallia käytetään myös, jos ohjelmiston valmistumisella on kiire ja sen kriittiset osat halutaan saada käyttöön mahdollisimman nopeasti. Kummassakin tapauksessa puuttuvat ominaisuudet voidaan lisätä ohjelmistoon jatkoprojekteissa.

3. Perinteisiä projektimalleja

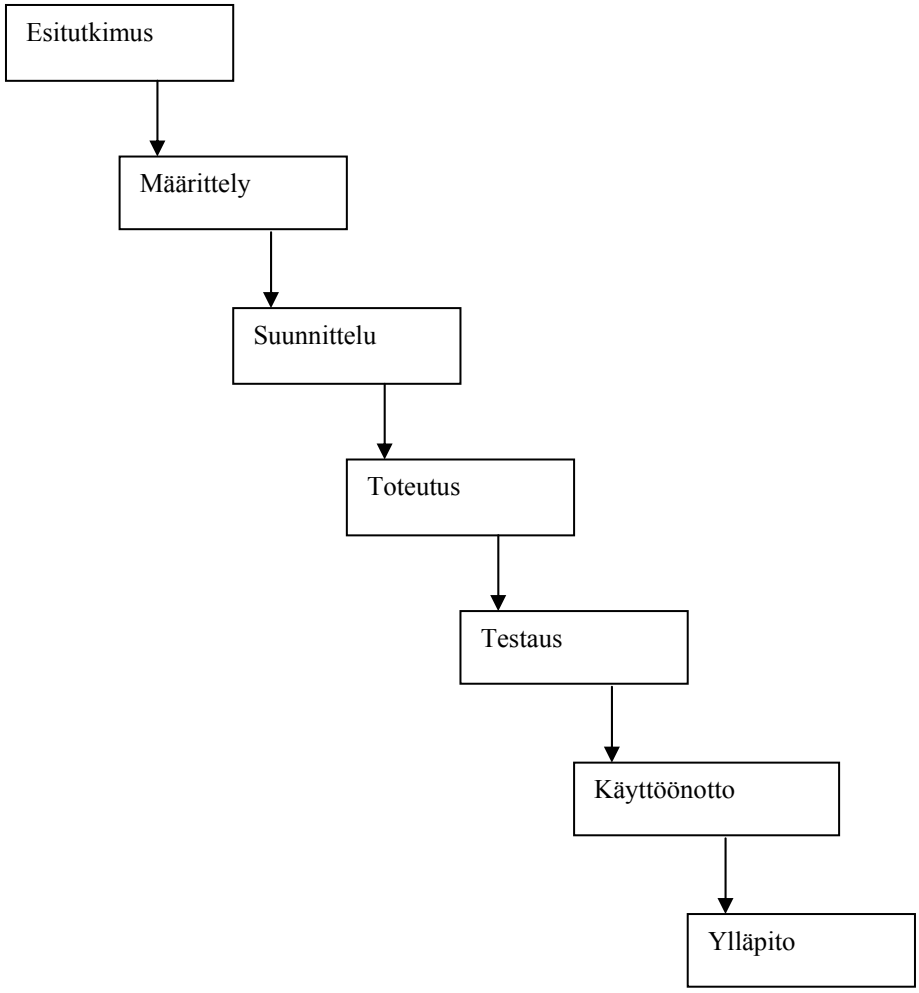
3.1. Vesiputousmalli

Vesiputousmalli on perinteinen ohjelmistoprojektin malli, jota on käytetty jo pitkään erilaisissa insinööriprojekteissa rakennettaessa siltoja, rakennuksia ja oikeastaan mitä tahansa tekniikkaa. Ilmeisesti tätä kautta vesiputousmalli on tullut myös ohjelmistojen rakentamisen välineeksi.

Vesiputousmallissa projekti jakautuu selvästi erillisiin vaiheisiin, jotka suoritetaan peräkkäin järjestyksessä ja kukin vaihe vain kerran. Eri vaiheet ovat:

- Esitutkimus, jonka tarkoituksena on tehdä kustannus-hyöty -analyysi projektista eli tutkia, onko ohjelmistoprojekti oikea ratkaisu käsillä olevaan ongelmaan. Tarkoituksena on myös saada arvio projektiin tarvittavasta ajasta, henkilöstöstä ja kustannuksesta. Esitutkimusvaiheesta syntyy yleensä raportti ja usein myös alustava projektisuunnitelma.
- Määrittelyvaihe, jossa pyritään etsimään vastaus kysymykseen "Mitä ohjelmiston tulisi tehdä?". Määrittelyvaiheessa kerätään ohjelmistolle asetetetut toiminnalliset, tekniset ja muut vaatimukset dokumenttiin nimeltä vaatimusmäärittely. Määrittelyvaiheessa tarkennetaan myös esitutkimusvaiheen projektisuunnitelma saadun uuden tiedon perusteella.
- Suunnitteluvaiheessa pyritään löytämään vastaus kysymykseen "Miten ohjelmisto tulisi rakentaa?". Tässä vaiheessa suunnitellaan ohjelma teknisesti, mutta ei varsinaisesti ohjelmoida vielä mitään. Vaiheen lopputuloksena on yksi tai useampia teknisiä dokumentteja, joissa määritellään ohjelmiston tekninen arkkitehtuuri, pysyvät datat ja niiden talletusratkaisut, käyttöliittymät, säikeistyksen periaatteet, viestinvälitysmekanismit ja niin edelleen.
- Toteutusvaiheessa ohjelmoidaan varsinainen ohjelmakoodi. Vaiheen tuloksena ovat valmiit ohjelmarakenteet sekä niiden koodikommentit.
- Testausvaiheessa toteutusvaiheessa syntynyt koodi testataan. Erilaisia testauskritereitä on useita: laajuus, toimintojen toimiminen oikeilla syötteillä, toimintojen toimiminen väärillä syötteillä, käytettävyys, tietoturva ja niin edelleen. Testausta voidaan tehdä useita kierroksia; näin tapahtuu kun löydetyt virheet korjataan ja niiden toiminta testataan uudelleen. Testausvaiheessa syntyvät yleensä dokumentit testaussuunnitelma ja testausraportti, testausta varten voidaan suuremmissa projeteissa laatia myös oma projektisuunnitelma.

- Ylläpitovaiheessa ohjelmisto on siirtynyt organisaation käyttöön ja sitä jatkokehitetään sekä käytössä huomattuja uusia toiveita toteutetaan. Myös käytössä havaittuja virheitä tai puutteita korjataan.
  Vesiputousmallin vaiheet on koottu kuvaan 2.

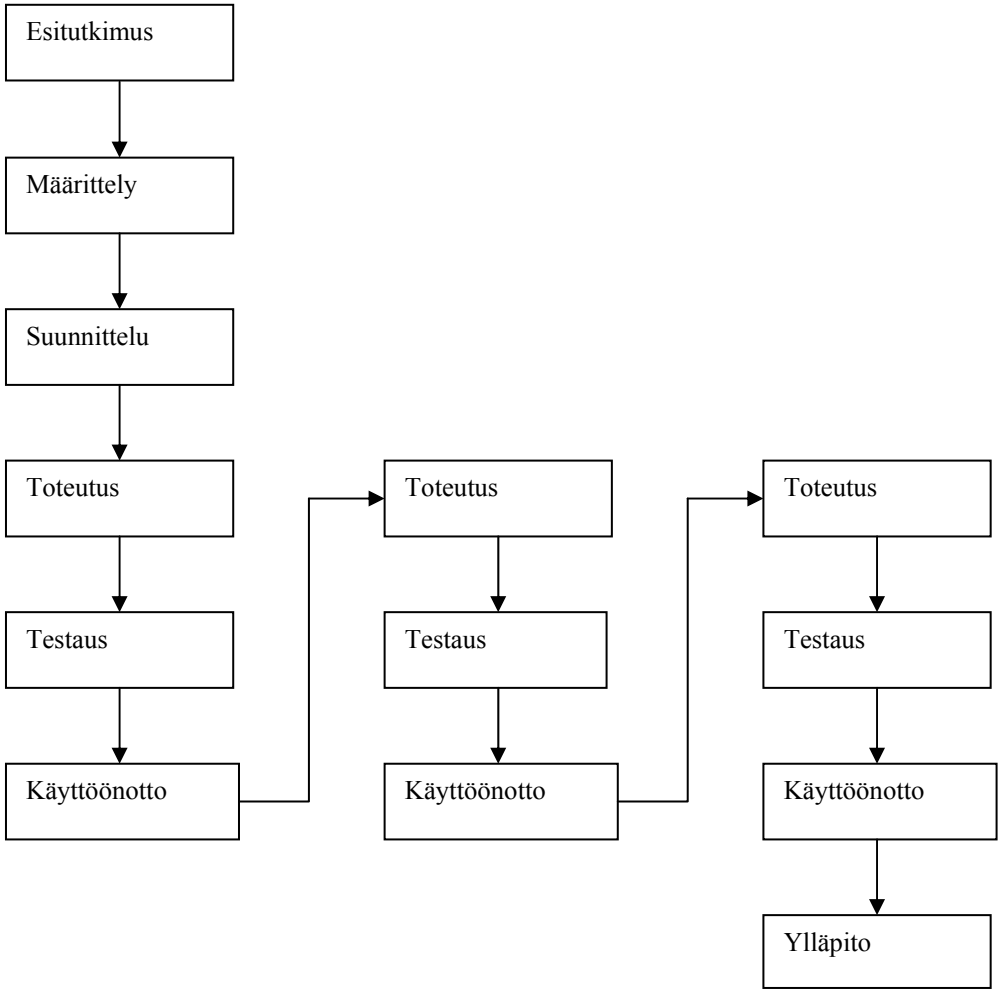


Kuva 2: Vesiputousmallin vaiheet.

3.2. Asteittainen kehittäminen

Asteittaisen kehittämisen malli on muuten samanlainen kuin vesiputousmalli, mutta suunnittelun jälkeiset vaiheet suoritetaan useampaan kertaan. Ohjelmasta toimitetaan useita versioita, joista jokainen sisältää edellisten versioiden toiminnallisuuden lisäksi uusia piirteitä ja ominaisuuksia. Huomattavaa on, että kaikki versiot pohjautuvat samoihin vaatimuksiin, määrittelyyn ja suunnitteluun.

Tämä malli sopii erityisesti melko laajoihin projekteihin sekä tuotekehitykseen. Vesiputousmalliin verrattuna hyötyjä ovat esimerkiksi aikaisempi palaute asiakkaalta sekä pienempi yhtäaikainen resurssien tarve. Asiakas näkee osia ohjelmistosta valmiina ja voi puuttua mahdollisiin ongelmakohtiin aikaisemmin. Samoin asiakas saa osan kehityskustannuksista takaisin jo kehityksen aikana. Huonona puolena mainittakoon, että myös aikaisemmin kehitetyt osat on testattava jokaisen uuden version ilmestyessä. Tämä kasvattaa ohjelmistokehityksen kokonaiskustannuksia, mikäli testausta ei ole mahdollista automatisoida. Asteittaisen kehityksen malli on esitetty kuvassa 3.



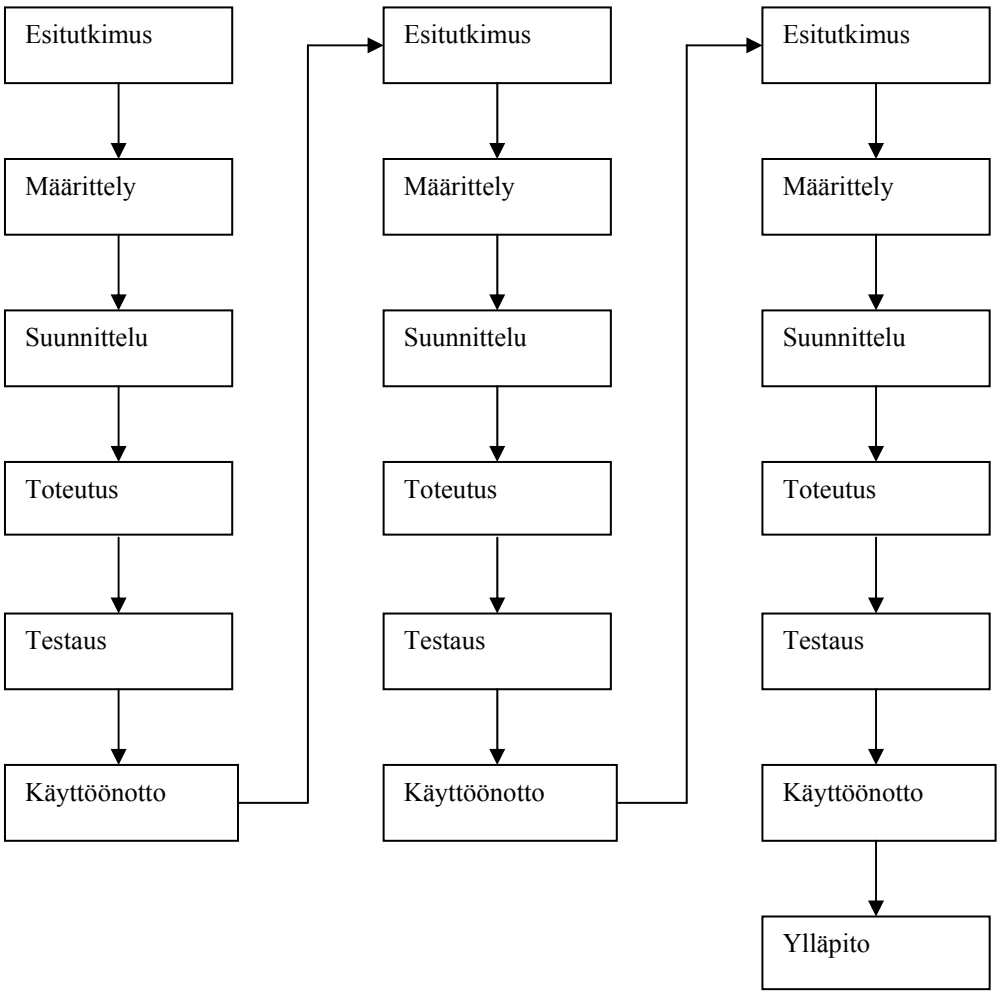
Kuva 3: Asteittaisen kehittämisen vaiheet.

3.3. Evolutionaarinen kehitysmalli

Evolutionaarisessa mallissa toistetaan suunnitelmallisesti koko vesiputousmallia useita kertoja, kunnes ohjelmisto on kokonaan toteutettu. Jokaisella evoluutiokierroksella otetaan mukaan uusia vaatimuksia ja näin ohjelmistosta syntyy uusi versio joka kerta, kun kaikki vaiheet on saatu suoritettua. Ero

asteittaisen kehittämisen malliin on siis siinä, että evolutionaarisessa mallissa toistetaan kaikkia ohjelmiston rakentamisprosessin vaiheita.

Tämä malli sopii projekteihin, joissa vaatimukset on heikosti ymmärretty tai asiakas on vielä itsekin epävarma siitä, mitkä ovat ohjelmiston lopulliset vaatimukset. Esimerkkinä voivat olla vaatimukset, jotka perustuvat uudistuvaan lainsäädäntöön, joka ei kuitenkaan ole vielä valmis. Malli on myös joustavampi kuin vesiputousmalli, kun osa vaatimuksista on selvästi muita vaatimuksia hankalampia toteuttaa tai teknologia ei ole riittävän kypsää koko järjestelmän toteuttamiseksi kerralla. Tällöin edes osa järjestelmästä saadaan käyttöön ja siten osa kustannuksista katettua nopeasti, jolloin saadaan uusia kehityspanoksia tulevia versioita varten. Evolutionaarisen mallin vaiheet on esitetty kuvassa 4.



Kuva 4: Evolutionaarisen mallin vaiheet.

4. Ohjelmistoprojektien ongelmia

4.1. Ohjelmistoprojektien onnistumisesta

Tutkimusyhtiö Standish Groupin [Standish Group, 1995] mukaan ohjelmistojen rakentaminen projekteissa ei suju kovinkaan hyvin. Raportissa jaettiin projektit kolmeen eri luokkaan niiden onnistumisen perusteella:

1. Ryhmään yksi kuuluivat projektit, jotka valmistuivat ajallaan sekä budjetin puitteissa.
2. Ryhmään kaksi kuuluivat projektit, jotka valmistuivat, mutta ylittivät aikataulunsa, budjettinsa tai molemmat.
3. Ryhmään kolme kuuluivat projektit, jotka lopetettiin ennen niiden valmistumista ja katsottiin epäonnistuneiksi.

Raportin mukaan vain 16 % projekteista oli onnistuneita eli kuului ryhmään yksi. Ryhmään kaksi, eli aikataulunsa tai budjettinsa ylittäneisiin, kuului 53 %. Kuitenkin peräti 31 % projekteista kuului ryhmään kolme, epäonnistuneet projektit. Luvut olivat keskimääräistä suurempia suuremmissa projekteissa ja keskimääräistä pienempiä pienemmissä. Jatkotutkimuksessa [Standish Group, 1998] huomattiin, että vuonna 1998 tulokset olivat hieman parantuneet, mutta epäonnistumisia oli edelleen runsaasti. Tällöin ryhmään yksi kuului 26 %, ryhmään kaksi 46 % ja ryhmään kolme 28 % ohjelmistoprojekteista.

4.2. Epäonnistumisten yleisimpiä syitä

Samassa tutkimuksessa käytiin IT-johdolle suunnatun kyselytutkimuksen avulla läpi myös epäonnistumisten yleisimpiä syitä. Syitä on luonnollisesti useita, mutta kolme yleisintä vastausta erottuvat yleisyydellään muista:

1. Käyttäjien osallistumisen puute (12,8 %).
2. Epätäydelliset määrittelyt projektin alkuvaiheessa (12,3 %).
3. Määrittelyjen muuttuminen projektin edetessä (11,8 %).

Kohdat kaksi ja kolme liittyvät läheisesti toisiinsa, ja myös ensimmäinen kohta vaikuttaa määrittelyjen tasoon. Samoin kirjassa Software Engineering Economics [Boehm, 1981] esitetään, että hyvinkin sujuneissa projekteissa noin 25 % vaatimuksista muuttuu määrittelyvaiheen jälkeen.

Ohjelmistojen vaatimukset muuttuvat siis hyvin usein. Tämä aiheuttaa väistämättä paluuta edellisiin vaiheisiin ja vaiheiden suorittamista useita kertoja. Perinteinen vesiputousmalli kuitenkin perustuu jokaisen vaiheen, myös määrittelyn, tekemiseen vain kerran. Näin ollen kaikki tätä seuraavat-

kin vaiheet tehdään ideaalitapauksessa vain kerran. Käytännössä tämä ei kuitenkaan tunnu toimivan, nimenomaan määrittelyjen muuttumisen vuoksi. Esitetyissä iteratiivisissa malleissa tämä on otettu paremmin huomioon, mutta näissäkään ei oteta huomioon käyttäjien osallistumiseen vaikuttamista.

5. Extreme programming -metodologian perusteet

Kirjassaan Extreme Programming Explained – Embrace Change [Beck, 1999] Kent Beck esittelee aluksi Extreme programming -metodologian perustana olevat neljä arvoa, jotta sen käytäntöjen ymmärtäminen olisi helpompaa. Tässä noudatetaan samaa periaatetta.

Sanonnan mukaan 80 % maailman ongelmista johtuu huonosta tai riittämättömästä kommunikaatiosta. XP:ssä noudatetaan työtapoja, joita on mahdotonta tehdä ilman kunnollista kommunikaatiota ihmisten kesken. Silti kommunikaation puute on vaarana myös XP-projekteissa, koska dokumentointia harrastetaan vähemmän kuin muissa menetelmissä. XP luottaakin enemmän keskusteluun kuin dokumenttien kirjoittamiseen. Kommunikaatiota toki arvostetaan sekä perinteisemmissä ohjelmistonkehitysmalleissa että XP:ssä.

Yksinkertaisuudella viitataan siihen, että järjestelmä toteutetaan yksinkertaisimmalla mahdollisella toimivalla tavalla. Toisin sanoen XP-suunnittelussa ei saisi koskaan varautua sellaisiin vaatimuksiin, joita ei suunnitteluhetkellä ole tiedossa, eli järjestelmää ei tietoisesti suunnitella laajennettavaksi eri suuntiin. Tämä on eräänlainen vedonlyönti sen puolesta, että muutosten aiheuttama lisätyö myöhemmin on pienempi kuin laajennettavuuden lisääminen heti suunnitteluvaiheessa. Tätä vedonlyöntisuhdetta parantaa XP:n käyttämä refactoring-menetelmä eli ohjelmiston arkkitehtuurin muokkaamiseen tarkkoja sääntöjä käyttäen.

Tämä arvo on tietoisessa ristiriidassa perinteisten ohjelmistonkehityksen mallien kanssa, jotka kannustavat rakentamaan helposti laajennettavia järjestelmiä. Tulevaisuudessa kokemukset XP-metodologian käytöstä osoittavat, onko aiemmin esitetty vedonlyönti kannattavaa. Edellisen perusteella voidaan todeta, että jos järjestelmän perusvaatimuksena on rakentaa helposti laajennettava järjestelmä tai kehityksen kohteena on yleiskäyttöinen luokkakirjasto, XP ei ole oikea projektinhallintatapa.

Kolmantena arvona on palaute, jota XP:ssä kerätään usein ja heti tehdyn työn jälkeen. Palautetta kerätään toimittajalta, asiakkaalta sekä itse rakennettavalta ohjelmistolta. Rakennettavan järjestelmän tilasta kerätään päivittäin

palautetta päivittäisten integrointien ja testauksien muodossa. Myös toimittaja ja asiakas saavat nopeaa palautetta toisiltaan.

Rohkeus on neljäs XP-metodologian arvo. Koska XP on monilta periaatteiltaan varsin radikaali, jo pelkkä menetelmän käyttäminen vaatii rohkeutta. Rohkeutta XP vaatii myös asiakkaalta, joka ei voi XP:n luonteen vuoksi saada heti projektin aluksi selkeää lupausta koko ohjelmiston kustannuksista ja työmääristä, vaan on itse mukana vaikuttamassa niihin projektin aikana.

6. Ohjelmiston rakentamisprosessi XP-metodologian mukaisesti

XP-metodologian mukainen rakentamisprosessi muistuttaa asteittaisen kehityksen ja evoluutiomallin vastaavia. Prosessi ei ota kantaa siihen, miten esitutkimusvaihe tehdään, vaan kustannus-hyöty -analyysi voidaan tehdä perinteisin menetelmin. XP:n erot muihin malleihin alkavat määrittelyvaiheesta.

Määrittelyvaihe XP:ssä on samantapainen kuin evoluutiomallin määrittelyvaihe. Koko ohjelmistoa ei määritellä kerralla, vaan projektin alussa määritellään vain ensimmäiseen versioon tarvittavat toiminnot. Koska uusia versioita annetaan asiakkaalle hyvin usein, kyse on ensimmäisen version kohdalla korkeintaan puolen vuoden projektista. Seuraavat versiot pyritään saamaan asiakkaalle aikavälillä yhdestä kolmeen kuukautta.

Määrittelyvaihetta kutsutaan nimellä suunnittelupeli (planning game). Tässä vaiheessa asiakas esittää tarvittavat toiminnot ja priorisoi ne. Tämän jälkeen ohjelmiston toimittaja antaa jokaisesta toiminnosta työmäärä-, aikataulu- ja kustannusarvion sekä kertoo asiakkaalle niiden muut tekniset vaikutukset, kuten esimerkiksi tarvittavat lisenssimaksut. Näiden tietojen avulla asiakas päättää sen, mitä toimintoja seuraavaan versioon tulee. Asiakkaalle annetaan aikatauluarvioiden perusteella mahdollisuus myös päättää siitä, koska tuleva versio on käytettävissä. Ne toiminnot, jotka eivät tule vielä seuraavaan versioon, kirjataan ylös ja säästetään seuraavaa suunnittelupeliä varten. Näin jaetaan vastuu toiminnoista ja käytetyistä teknologioista selkeästi: asiakas määrittelee toiminnot ja niiden toteuttamisjärjestyksen, toimittaja määrittelee tarvittavat teknologiat.

Määrittelyssä käytetään käyttötapaustekniikkaa (use cases). Käyttötapaus on määritelty “sarjaksi järjestelmän toimintoja, joiden tehtävänä on tuottaa mitattavaa hyötyä järjestelmän käyttäjälle” [Jacobson et al., 1995]. Tarkemmin sanottuna käyttötapauksessa kirjataan selväkielisenä, asiakkaan ymmärtämänä ei-teknisenä tekstinä ne asiat, jotka yhden tehtäväkokonaisuuden

aikana käyttäjän näkökulmasta halutaan tehdä ja millaiset tulokset hän haluaa tehtävästään nähdä.

XP:ssä käyttötapauksia kutsutaan nimellä tarina (story). XP:n ero perinteisiin käyttötapauksiin on kuitenkin siinä, että tässä tapauksessa tarinat kirjoittaa asiakas. Käyttötapauksiin ei tällöin kirjata ylös teknisiä yksityiskohtia kuten erilaisia poikkeuksia tai virhetilanteita. Kun asiakas on kirjoittanut käyttötapaukset, hän antaa ne toimittajalle, joka arvioi ne ja antaa asiakkaalle niistä palautetta, mikä on yksi XP:n arvoista. Tällä tavalla käyttötapauksia voidaan iteroida ja niiden laatua parantaa. Kun käyttötapaukset ovat riittävän yksityiskohtaiset toteutusvaihetta varten, voidaan ohjelmointi aloittaa.

Toteutusvaiheessa tulevat esiin XP:n radikaaleimmat erot muihin metodologioihin verrattuna. Ohjelmointivaiheen aluksi, ennenkuin riviäkään lopullista ohjelmakoodia on kirjoitettu, ohjelmoija kirjoittaa käyttötapaukselle vastaavan testitapauksen (test case). Vasta tämän jälkeen ohjelmoidaan varsinainen ohjelmakoodi.

Ohjelmakoodi kirjoitetaan periaatteella “yksinkertaisin mahdollinen toimiva ratkaisu”. Ohjelmoinnissa ei siis esimerkiksi koskaan käytetä olio-ohjelmoinnille tyypillistä uudelleenkäytön periaatetta, mikäli uudelleenkäytön kohdetta ei ole jo valmiiksi tiedossa. Arkkitehtuuri suunnitellaan vain niiden toimintojen perusteella, jotka ovat tulossa seuraavaan ohjelmiston versioon. Yksinkertaisuuden arvo toteutuu selvimmin juuri tässä. Lisäksi ohjelmointi vain tätä hetkeä varten, varautumatta myöhemmin tarvittaviin arkkitehtuurimuutoksiin, vaatii rohkeutta.

Kun käyttötapaukseen tarvittava koodi on valmistunut, se integroidaan välittömästi muuhun ohjelmakoodiin. Kirjoitettu testitapaus puolestaan integroidaan automatisoituun testausjärjestelmään, jollaisia ovat esimerkiksi Java-ohjelmille suunnitelleet JTest [JTest] ja JUnit [JUnit]. Kun testitapaukset on integroitu, ajetaan välittömästi kaikki testausjärjestelmän sisältämät testit. Mikäli kaikki testit menevät läpi, toteutus on onnistunut ja ohjelmointivaihetta voidaan jatkaa seuraavalla käyttötapauksella. Mikäli jokin testeistä palautti virheen, virhe korjataan välittömästi ja testaamista jatketaan kunnes virheitä ei enää tule. Tämä tukeekin hyvin edellä esitettyä palautteen arvoa.

Edellisen kaltaisesta, jopa useita kertoja päivässä tapahtuvasta integroinnista ja testaamisesta, saavutetaan etuina koodin jatkuva virheettömyys ja jatkuva tieto järjestelmän tilasta, jolloin eri ohjelmoijien koodinosat ovat aina synkronissa keskenään ja mahdolliset arkkitehtuurivirheet päästään korjaa-

maan nopeasti. Tämä tietenkin sillä edellytyksellä, että laaditut automaattiset testit saavuttavat riittävän testikattavuuden.

Ohjelmoidessa kaikki lopulliseen ohjelmaan menevä ohjelmakoodi tuotetaan pariohjelmointina, eli kahden ohjelmoijan työskennellessä yhdessä, yhdellä tietokoneella, saman ohjelmakoodin parissa. Kun toinen parista ohjelmoi, toisen tehtävänä on samalla lukea kaikki toisen kirjoittama koodi ja korjata mahdollisia virheitä saman tien. Ohjelmoijat myös keskustelevat keskenään parhaista toteutusvaihtoehdoista.

Pariohjelmointi johtaa osaltaan myös siihen, että yksittäisillä ohjelmiston osilla ei ole varsinaista sitä omistavaa ohjelmoijaa, vaan kaikki koodi on kaikkien ohjelmoijien yhteisomistuksessa. Tällöin myös kuka tahansa projektiryhmän jäsen voi muokata toisen jäsenen tekemään koodia paremmaksi. Koodin yhteisomistus, samoin kuin pariohjelmointi, vaatii yhteisen ohjelmointistandardin käyttöä määrittelemään, miltä koodin tulisi näyttää.

Koodia ei kuitenkaan saa muokata mielensä mukaan vaan ainoastaan refactoring-menetelmän mukaisesti. Refactoring on Martin Fowlerin kehittämä ja samannimisessä kirjassaan [Fowler, 1999] esittelemä ohjelmakoodin paremmaksi muokkaamiseen tarkoitettu apuväline, jossa ohjelmakoodia saa muuttaa vain sääntöjen mukaan. Kirja esittelee kymmeniä erilaisia muokkaussääntöjä tähän. Tyypillinen esimerkki säännöstä on ohjeet siitä, miten yksi oliokielen luokka jaetaan pääluokkaan ja sen perivään luokkaan.

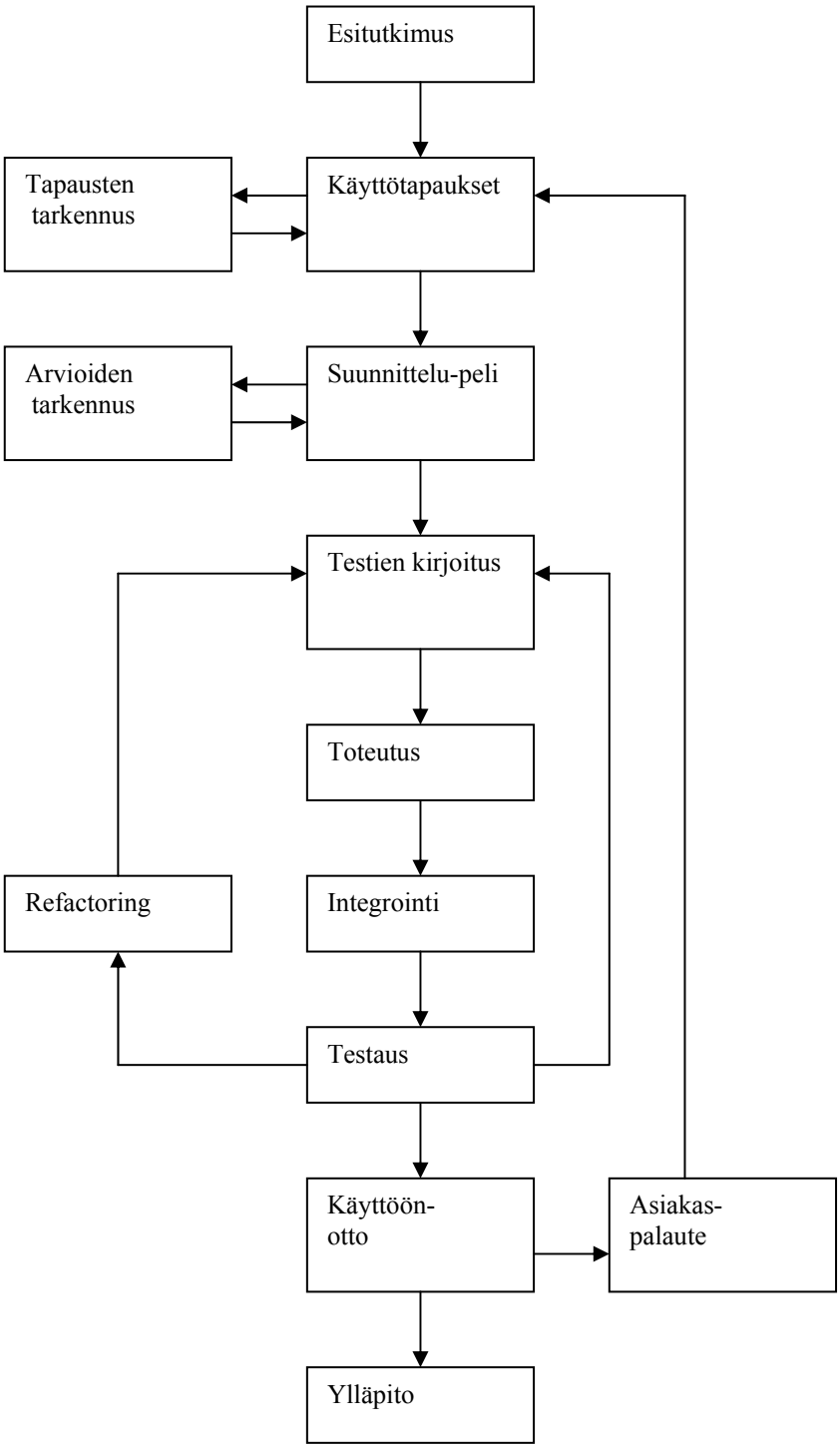
Extreme Programming kannustaa voimakkaasti ohjelmoinnissa käytettävien apuvälineiden käyttöön, jotka automatisoivat ohjelmistokehityksen rutiineja. Automaattiset testaustyökalut mainittiin aiemmin ja versionhallintatyökalujen käyttö lienee jo perusedellytys kaikissa projekteissa. Lisäksi usein tapahtuvan integroinnin takia myös testattavan versioehdokkaan, ns. “buildin”, rakentamiseen suositellaan käytettäväksi sopivaa automaattityökalua, esimerkiksi Ant-skriptiä [Ant].

XP suosittelee koodin optimoinnin tapahtuvan vasta, kun kaikki toiminnallisuus on toteutettu ja toimivaksi testattu. Tämä on jo pitkään hyväksi tiedetty tapa, jonka XP on ottanut metodikokoelmaansa mukaan. Ohjelmointi- ja testausvaihe päättyy, kun kaikki versioon aiotut käyttötapaukset on ohjelmoitu, optimoitu, niiden testit kirjoitettu ja kaikki testit ovat menneet läpi. Tällöin ohjelmisto voidaan asentaa asiakkaalle ja aloittaa seuraava kehityskierros uudella suunnittelupelillä.

On huomattava, että XP ei rajoitu ainoastaan ohjelmistojen rakentamiseen, vaan laajentaa huomiota myös projekteissa työskentelevien ihmisten hyvinvointiin. Eräs XP:n periaatteista onkin ”40 tunnin työviikko”, joka

tarkoittaa että projekteissa ei saa tehdä ylitöitä, ei ainakaan kahta viikkoa peräkkäin. Kent Beck onkin kuvannut XP:tä lauseella ”humanistinen tapa tehdä ohjelmia” [Beck, 1998].

Extreme Programming-metodologian vaihejakomalli on esitetty kuvassa 5.



Kuva 5: Extreme Programming -metodologian vaiheet.

7. Vertailua perinteisiin projektinhallinnan menetelmiin

Taulukossa yksi on koottu ne eri projektimallien piirteet, jotka poikkeavat eniten toisistaan. Taulukon perusteella huomataan, että projektimallit voidaan asettaa järjestysasteikolle sen mukaan, kuinka joustavasti muutostenhallinta sujuu, kuinka usein ohjelmistoa testataan ja kuinka usein se toimitetaan asiakkaalle. Tämä järjestys kulkee vesiputousmallista asteittaisen kehityksen ja evolutionaarisen kehityksen kautta XP:hen. Hintana joustavuudesta tulee käytettyjen resurssipanostusten ennustettavuuden heikkeneminen pitkällä tähtäimellä.

Taulukko 1: Eri projektimallien toisistaan poikkeavia piirteitä

Piirre	Vesiputous-malli	Asteittainen kehitys	Evolutionaa-rinen kehitys	Extreme Programming
Panostusten ennustettavuus	Pitkän tähtäimen ennusteet	Keskipitkän tähtäimen ennusteet	Lyhyen tähtäimen ennusteet	Lyhyen tähtäimen ennusteet
Vaihejako	Kukin vaihe vain kerran	Osittain iteratiivinen	Kokonaan iteratiivinen	Kokonaan iteratiivinen
Varautuminen arkkitehtuurin muutoksiin	Ei ohjeita, usein pyrkii varautumaan	Ei ohjeita, usein pyrkii varautumaan	Ei varautumista	Ei varautumista
Muutosten-hallinta	Kankea prosessi	Edellistä joustavampi prosessi	Edellistä joustavampi prosessi	Joustavin prosessi
Testaus	Vain kerran	Iteraation lopussa	Iteraation lopussa	Päivittäin
Ohjelmiston toimitus	Koko ohjelmisto kerralla	Muutamissa osissa	Muutamissa osissa	Usein ja pienissä osissa
Palaute asiakkaalta	Projektin valmistuttua	Edellistä useammin	Edellistä useammin	Useimmin

8. Extreme Programming-käytännöt tutkimusten valossa

Metodologian nimellä ”Extreme Programming” on Beckin mukaan [Beck, 1999] nimen molemmista sanoista muodostuva merkitys. Sanalla extreme (äärimmäisyys) hän tarkoittaa, että XP:hen kootut hyväksi havaitut käytännöt

viedään äärimmäisyyksiin. Esimerkiksi pariohjelmointi on katselmoinnin vientiä äärimmäisyyksiin, jatkuva testaus on testauksen vientiä äärimmäisyyksiin ja nopea evoluutiosykli on palautteen vientiä äärimmäisyyksiin. Sanalla programming (ohjelmointi) hän tarkoittaa, että XP on nimenomaan ohjelmointipainotteinen tapa toteuttaa projekteja. Siinä ei tuoteta paksuja teknisiä suunnittelu- tai testausdokumentteja, vaan pyritään tuottamaan mahdollisimman yksinkertaisia ohjelmia vähemmällä suunnittelulla. Mikäli tämä suunnittelun puute tuottaa huonoja arkkitehtuuriratkaisuja, ne korjataan refactoring-menetelmää hyväksikäyttäen, eli ohjelmoimalla.

Iteratiivinen ohjelmistoprojektin malli on todettu tehokkaaksi tavaksi kohdata puutteellisiin tai muuttuviin määrittelyihin liittyviä ongelmia [Brownsword ja McUmber, 1991]. Brownsword ja McUmber kuitenkin toteavat, että siirtyminen vesiputousmallista iteratiivisen mallin käyttöön vaatii organisaatiossa huolellista suunnittelua.

XP:n tapa rakentaa mahdollisimman yksinkertainen ohjelmisto ja optimoida se vasta lopuksi on tutkimuksen [Stevens, 1981] mukaan osoittautunut paremmaksi vaihtoehdoksi kuin optimoida ohjelmistoa jatkuvasti. Yksinkertaisuuden arvon noudattaminen tuottaa luonnollisesti tehokkaampia ohjelmia. Tämä on mahdollista kaikissa projektinhallintametodologioissa, mutta vain XP määrittelee sen eksplisiittisesti.

XP:n suosima pariohjelmointi on osoittautunut [McDowell *et al.*, 2002], [Williams ja Upchurch, 2001] tehokkaaksi yliopistojen kursseilla järjestetyissä kokeissa. Ohjelmien rakentamiseen tarvittu aika lyhentyi, niissä oli valmistuttuaan vähemmän virheitä, pariohjelmointiin osallistuneet opiskelijat olivat tyytyväisiä ja he halusivat jatkaa pariohjelmointia myös tulevaisuudessa. Näiden tutkimusten harjoitustehtävät olivat kuitenkin pieniä, muutaman sadan rivin ohjelmia. Näin ollen pariohjelmoinnin soveltuvuudesta huomattavasti suurempien reaalimaailman sovellusten kehittämiseen ei voida esittää kuin suuntaa antavia kommentteja.

Vaikka Extreme programming -metodologiasta ei vielä ole paljoa tutkimustuloksia tarjolla, on XP:n käytäntöjä tutkittu vuosien varrella runsaasti. Kaikkien käytäntöjen tai eri käytännöistä tehtyjen tutkimusten esittäminen tässä tutkimuksessa on mahdotonta, vaan sitä varten tarvitaan laajempi tutkimus.

9. Loppupäätelmät

Edellä mainittujen seikkojen ja lähteiden perusteella on lopuksi koottu yhteen projektimalleja, joihin Extreme programming sopii. Tämän jälkeen on listattu

malleja, joihin Extreme programming sopii huonosti tai ei ollenkaan. Listat eivät suinkaan ole täydellisiä, ainoastaan suuntaa antavia.

Projektimalleja, joihin Extreme Programming sopii hyvin, ovat:

- Erityisesti projektit, joissa vaatimukset ovat epäselviä tai niiden uskotaan muuttuvan projektin aikana. XP tuntuu keskittyvän nimenomaan määrittelyissä tapahtuvien muutosten negatiivisten vaikutusten eliminoimiseen. Luvussa 4.2. esitettiin, että suurimmat ongelmat projekteissa liittyvät nimenomaan joko määrittelyjen puutteellisuuteen tai niiden muuttumiseen. Beck totesi kirjassaan [Beck, 1999]: ”(Projektien) ongelmana eivät ole vaatimusten muuttuminen vaan se, etteivät käytetyt projektimallit varaudu vaatimusten muutoksiin”. Useat iteraatiot, vastuunjako toiminnallisten ja teknisten vaatimusten välillä sekä asiakkaan vaikutus projektin ohjaukseen tukevat tätä varautumista.
- Projektit, jotka tehdään uudelle tai vaikealle sovellusalueelle, jota projektihenkilöstö ei täysin ymmärrä. Tiivis yhteistyö asiakkaan kanssa lisää ymmärrystä alueesta ja virheet huomataan iteraatioilla nopeasti.
- Projektit, joissa perustoiminnallisuus tulee saada käyttöön hyvin nopeasti. Nopea iteraatioiden sykli tukee tätä erinomaisesti. Tämä tosin edellyttää, että ohjelmiston toiminnallisuus on jaettavissa erikseen toimitettavissa oleviin osiin.

Projektimalleja, joihin Extreme Programming sopii huonosti sekä muita XP:n käyttöä rajoittavia tekijöitä ovat:

- Sellaisten ohjelmistojen rakentaminen, joiden päivittäminen käyttöönoton jälkeen on vaikeaa tai mahdotonta. Toisaalta teknisesti helposti päivitettäviä ohjelmistoja tuottavat projektit, kuten esimerkiksi sovelluspalvelinpohjaiset selainkäyttöliittymällä toimivat ohjelmistot, voivat olla erinomaisia XP:n käytön kohteita.
- Projektit, joissa asiakkaan on tarkasti tiedettävä siihen kuluvat aika- ja kustannusresurssit. XP:ssä lopullinen kustannus tiedetään vasta ohjelmiston valmistuttua.
- Projektit, jotka toteutetaan jollain muulla ohjelmointikielellä kuin oliokielellä. Refactoring-metodi on erittäin oliokeskeinen, eikä sitä käytännössä voi soveltaa muihin ohjelmointiparadigmoihin. XP:n muut menetelmät ovat tosin käytettävissä kaikkien ohjelmointikielten kera.

- Projektit, joita aloitettaessa on syytä epäillä, ettei asiakkaalla ole riittävästi aikaa tai halua osallistua projektiin. Tämä vaikeuttaa yhteistyötä tietenkin kaikissa projektimalleissa.
- Suuret, yli kahdenkymmenen hengen projektit. Näissä Beckin [Beck, 1999] mukaan työn tuottavuus laskee liiaksi XP:tä käytettäessä tarvittavan kommunikaation takia. Ongelma koskee kaikkia projektimalleja, mutta XP:ssä on sen lisäksi saatava päivittäiseen integraatioon mukaan kaikki tuotettu koodi, mikä muodostaa prosessiin pullonkaulan. Tämän kokoluokan projektit jaetaan usein aliprojekteihin, joten aliprojektin läpivienti XP:n menetelmillä ei ole poissuljettua.
- Toisaalta myös pienet projektit, joissa on alle neljä henkilöä ovat huonoja XP:n käyttökohteita. Yhden hengen projekteissa pariohjelmointi ei ole mahdollista, kahden hengen projekteissa siitä tuskin saadaan vastaavaa hyötyä ja kolmen hengen projekteissa voidaan muodostaa vain yksi pari kerrallaan.
- Beckin mukaan myös projektit, joissa henkilöstö on jakautunut maantieteellisesti, on parempi viedä läpi jotain toista projektinhallintamenetelmää käyttäen. Tällöin henkilökohtainen kommunikointi, pariohjelmointi ja koodin yhteisomistus vaikeutuvat. Tämä usein sulkee pois myös projektit, joissa käytetään alihankintaa.
- Ehkä tärkeimpänä esteenä Beck kuitenkin mainitsee organisaatiot, joilta puuttuu tarve tai rohkeus kokeilla uusia menetelmiä. Sananlaskun mukaan ehjää ei ole tarpeen korjata, joten XP:hen tulee siirtyä vain, jos prosessien laatu ei organisaatiossa tyydytä. Toisaalta vaikka parannettavaa prosesseissa olisikin, organisaatiokulttuurit saattavat olla kykenemättömiä XP:n vaatimiin muutoksiin.

Monet XP-metodologian menetelmistä näyttävät toimivilta myös tutkimusten valossa. Toisaalta XP:n käyttöä kokonaisuutena suurissa ohjelmistoprojekteissa on tutkittu vielä varsin vähän, eikä luotettavia näyttöjä sen toimivuudesta ole vielä saatu. Toisaalta mikään ei myöskään viittaa siihen, että XP olisi huono menetelmä, vaan tähän asti esitettyjen tulosten voidaan sanoa olevan rohkaisevia.

Tietojenkäsittelytieteessä kuitenkin tiedetään, ettei mikään yksittäinen teknologia tai menetelmä tule ratkaisemaan kaikkia ongelmia, vaikka niiden tullessa laajempaan tietoisuuteen niihin kohdistuu suurta huomiota. Näin

asia on myös XP:n kanssa: siitä ei tule hopealuotia kaikkien ohjelmistoprojektien ongelmien ratkaisuun, mutta se saattaa tarjota hyvän työkalun projektipäälliköille niihin projekteihin, joihin se parhaiten sopii.

Viiteluettelo

[Ant] Ant 1.5.1. The Apache Jakarta Project. http://jakarta.apache.org/ant/.

[Beck, 1999] Kent Beck, *Extreme Programming Explained – Embrace Change*. Addison-Wesley, 1999.

[Beck, 1998] Kent Beck, Extreme Programming: A Humanistic Discipline of Software Development. *Lecture Notes in Computer Science*. 1382, 1-16.

[Boehm, 1981] Barry W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[Fowler, 1999] Martin Fowler, *Refactoring*. Addison-Wesley, 1999.

[Brownsword ja McUmber, 1991] Lisa Brownsword and Rick McUmber, *Proceedings of the conference on TRI-Ada '91: today's accomplishments; tomorrow's expectation*. 378 - 386

[Jacobson et al., 1995]. I. Jacobson, M. Ericsson and A. Jacobson, *The Object Advantage: Business Process Reengineering With Object Technology*, Addison-Wesley, 1995.

[JTest] JTest. Parasoft. http://www.parasoft.com/jsp/products/ home.jsp?product=Jtest

[JUnit] JUnit.org Homepage. http://www.junit.org

[McDowell *et al.*, 2002] Charlie McDowell, Linda Werner, Heather Bullock and Julian Fernald, *The effects of pair-programming on performance in an introductory programming course*. ACM SIGCSE Bulletin 34, 1 (2002), 38-42.

[Pressman, 2000] R.S. Pressman, *Software Engineering: A Practitioner' s Approach*. McGraw-Hill, 2000.

[SFS, 1981] Suomen Standardoimisliitto, *Projektitoiminta, toimintaverkkosanasto ja piirrosmerkit*. Suomen Standardoimisliitto, 1981.

[Standish Group, 1995] Standish Group Ltd, *Chaos Report*. Standish Group, 1995. Available also as http://www.standishgroup.com/sample_research/index.php.

[Standish Group, 1998] Standish Group Ltd, *Chaos: A Recipe for Success*. Standish Group, 1998. Available also as http://www.standishgroup.com/ sample_research/chaos1998.pdf.

[Stewens, 1981] Wayne Stevens, *Using Structured Design*. John Wiley, 1981.

[Williams ja Upchurch, 2001] Laurie Williams and Richard L. Upchurch, *In support of student pair-programming.* ACM SIGCSE Bulletin 33, 1 (2001), 327 - 331.

Suoraviestintä SIP:llä

Petri Lintula

Tiivistelmä

Session Initiation Protocol eli SIP on signalointiprotokolla ip-verkkoihin. SIP:n MESSAGE-laajennus mahdollistaa suoraviestinnän toteuttamisen SIP-protokollalla. SIP ei ota kantaa viestien sisältöön, joten viestien sisältö voi olla tekstiä tai MIME-koodattua sisältöä.

Avainsanat ja -sanonnat: SIP, suoraviestintä, SIP MESSAGE
CR-luokat: C.2.2, C.2.6

1. Johdanto

SIP eli Session Initiation Protocol on signalointiprotokolla, jonka avulla luodaan, muokataan ja päätetään istuntontoja ip-verkoissa. Istunto voi olla yksinkertainen Internet-puhelu kahden käyttäjän välillä tai usean käyttäjän multimediakonferenssi. Tämänkaltaisten istuntojen luominen mahdollistaa uusia palveluita, esimerkiksi äänellä varustetun elektronisen kaupankäynnin tai multimedia sisältöisten suoraviestin lähettämisen.

Suoraviestinnän suosio on kasvamassa työpaikoilla sekä vapaa-aikana. Tähän saakka suurin suoraviestintää rajoittava tekijä on ollut ohjelmistojen keskinäinen toimimattomuus. SIP yrittää ratkaista tämän ongelman laajennuksillaan. Suuret ohjelmistotalot ovat jo lisänneet tai ovat lisäämässä SIP-tuen omiin suoraviestiohjelmistoihinsa. Tässä tutkielmassa tarkastelen SIP:n MESSAGE-viestiä, sen mahdollisuuksia ja rajoituksia.

Matkapuhelimen kannalta suoraviestintä ei tuo muuta lisäarvoa kuin multimediasuoraviestin mahdollistamisen. Läsnäolotiedon yhdistäminen suoraviestintään avaa uusia mahdollisuuksia. Läsnäolotieto kertoo käyttäjän sen hetkisen tilan, eli onko hän tavoitettavissa. Näin saadaan varmuus vastaanottajan tavoitettavuudesta.

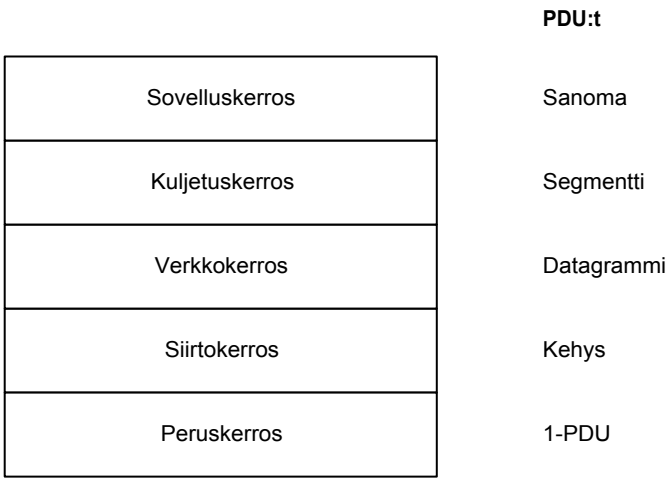
2. Internet

Internetin tarkoitus on tietoliikenneyhteyksien tarjoaminen. Muilla verkoilla on muita tarkoituksia, esimerkiksi televerkon tarkoituksena on telepalve-

luiden tarjoaminen ja tv-verkon tarkoituksena on tv-ohjelmien lähettäminen. Internet koostuukin useista pienistä verkoista, jotka ovat yhteydessä toisiinsa.

Internetin vahvuuksia ovat sen vikasietoisuus sekä soveltuvuus palveluiden luontialustaksi. Vikasietoisuutta lisää, ettei verkko tarvitse tilatietoa lähettääkseen ip-paketteja vastaanottajalle. Tällöin verkon solmun eli reitittimen lakatessa toimimasta, liikenne ohjataan perille toisen solmun kautta. Vikasietoisuutta lisää myös, että verkko koostuu älykkäistä päätelaitteista. Itse reitittimet ovat passiivia ja lähettävät ip-paketteja saamiensa ohjeiden mukaisesti. Internetin rakenne, eli sen koostuminen useista eri kerroksista helpottaa sovellusten rakentamista. Sovelluksen kehittäjä valitsee tarvittavat protokollat, jotka huolehtivat datan siirrosta.

Internet koostuukin useista kerroksista, joita ovat (Kuva 1) peruskerros, siirtokerros, verkkokerros, kuljetuskerros ja sovelluskerros. Kuvassa 1 on myös esitetty jokaisen kerroksen PDU (protocol data unit) eli siirrettävän datan nimitys. Peruskerros koostuu fyysisistä laitteista. Siirtokerroksen tehtävä on kehyksen siirto linkin yli. Verkkokerroksen, eli IP-kerroksen, tehtävänä on reitittää lähetettävät datagrammit lähettävältä isäntäkoneelta vastaanottavalle isäntäkoneelle. Kuljetuskerroksen tehtävänä on siirtää sovelluskerroksen sanomat asiakkaalta palvelimelle ja päinvastoin. Sovelluskerros on tarkoitettu sovelluksen eri komponenttien väliseen viestintään.



Kuva 1. Internetin kerrokset.

2.1. Verkkokerros

Verkkokerroksen tehtävänä on reitittää datagrammit lähettävältä isäntäkoneelta vastaanottavalle isäntäkoneelle. Internet käyttää tähän Internet-protokollaa (IP) [RFC 791]. IP mahdollistaa verkon liittämisen muihin verkkoihin riippumatta niiden teknologisista ratkaisuista ja pakettien lähettämisen näiden verkkojen välillä.

IP:tä käyttävät verkkokerroksessa reitittimet sekä palvelimet, siirtäen näin älykkyyttä päätelaitteille. Päätelaitteet ovat vastuussa ip-pakettien liikenteen kontrolloinnista, protokolla ei varmista pakettien perillemenoa eikä pakettien järjestystä.

2.2. Kuljetuskerros

Kuljetusprotokollien perustehtävänä on tarjota ylemmille sovelluksille kuljetuspalvelua kahden mahdollisesti eri koneissa tai jopa eri verkoissa olevan prosessin välillä. Tämän seurauksena kuljetusprotokollan vähimmäisvaatimuksena on kyky osoittaa lopullinen kohde eli se sovellus tai prosessi, jonka kanssa halutaan kommunikoida. TCP/IP-arkkitehtuurissa on kaksi vaihtoehtoa kuljetusprotokollaksi: TCP ja UDP.

2.2.1. TCP

Transmission Control Protocol eli TCP [RFC 793] tarjoaa luotettavan yhteydellisen kuljetuspalvelun, joka takaa pakettien järjestyksen säilymisen sekä pakettien pääsyn vastaanottajalle. Tämän ansiosta sovellusohjelmoijan ei tarvitse kiinnittää huomiota tiedonsiirrossa esiintyvien ongelmien hoitamiseen vaan TCP hoitaa ne hänen puolestaan.

TCP-protokolla perustuu lähetysikkunan käyttöön, jossa sovellus lähettää useampia paketteja kerrallaan ennen kuin jää odottamaan kuittausta. Vastaanottajan puolestaan ei tarvitse lähettää kuittausta kaikista vastaanottamista paketeista. Vastaanottaja lähettää kuittauksen, jossa se kertoo, kuinka pitkälle se on saanut dataa oikein vastaanotettua. TCP-protokollassa ei kuitenkaan kuitata paketteja niiden järjestysnumeron perusteella vaan alkuperäisestä datasta vastaanotettujen tavujen perusteella. Käytännössä tämä tarkoittaa sitä, että kuittauksessa kerrotaan, mitä tavua vastaanottaja seuraavaksi odottaa lähettäjältä. Mikäli vastaanottaja ei saa jotain pakettia tai sen sisältö on muuttunut virheelliseksi siirron aikana, vastaanottaja ei lähetä kuittausta. Tämän seurauksena tietyn odotusajan jälkeen lähettäjän kuittauksen odotusaikalaskuri nollautuu ja kaikki data viimeisessä kuittaussanomassa olleesta tavusta alkaen lähetetään uudelleen.

Kaikkien pakettien vastaanottaja on sama eli koneen ip-osoite. Paketti kohdistetaan oikealle sovellukselle TCP-porttinumeroinnin avulla, esimerkiksi porttiin 80 tulevat paketit lähetetään HTTP-sovellukselle.

2.2.2. UDP

User Datagram Protocol eli UDP [RFC 768] tarjoaa yksinkertaisen yhteydettömän kuljetuspalvelun sovelluksille. Käytännössä se ei lisää alemman protokollan eli IP:n päälle muuta kuin mekanismin osoittaa lopullinen kohde. UDP ei sisällä pakettien numerointia, eikä siten anna varmuutta pakettien järjestyksen säilymisestä. Koska lähetettäviä paketteja ei numeroida, vastaanottaja ei myöskään voi kuitata niitä vastaanotetuiksi. Tämän vuoksi UDP:tä voidaan pitää epäluotettavana kuljetuspalveluna.

UDP:ssä virheiden havaitsemiseksi lasketaan tarkistussumma koko UDP-paketista eli myös datakentästä. UDP-tarkistussumman käyttö on ainoa tapa taata datan oikeellisuus siirron jälkeen.

Kuten TCP:ssäkin kaikkien pakettien vastaanottaja on sama eli koneen ip-osoite. Paketti kohdistetaan oikealle sovellukselle porttinumeroinnin avulla.

2.3. Sovelluskerros

Internet-arkkitehtuurin ylimmällä tasolla ovat varsinaiset sovellukset. Sovelluskerroksen tarkoituksena on välittää eri sovellusten viestejä toisille sovelluksille. Sovelluskerros tarjoaa lukuisia eri protokollia, joista sovellus valitsee sen tarvitsemat protokollat. Esimerkkejä näistä protokollista ovat Hypertext Transfer Protocol eli HTTP, File Transfer Protocol eli FTP, Simple Mail Transfer Protocol eli SMTP sekä Session Initiation Protocol eli SIP.

3. SIP:n historia

3.1. SIPv1

Ensimmäinen versio SIP:stä (SIPv1) ilmestyi helmikuussa 1996 ja oli nimeltään Session Invitation Protocol. Sen kehittivät Mark Handley ja Eve Schooler ja sen julkaisi IETF-statuksella (Internet Engineering Task Force) [IETF] ”Internet draft”. SIPv1 käytti Session Description protokollaa (SDP) [RCF 2327] istuntojen kuvaamiseen ja UDP:tä tiedon siirtämiseen. SIPv1 oli tekstipohjainen ja se käsitteli pelkästään istuntojen perustamista. Merkinanto loppui käyttäjän liittyessä istuntoon. Istunnon aikaisia kontrolleja ei SIPv1:ssä myöskään ollut [Camarillo].

3.2. SCIP

Simple Conference Invitation protokollan (SCIP) julkaisi Henning Schulzrinne myös helmikuussa 1996 IETF-statuksella ”Internet draft”. Myös SCIP oli mekanismi käyttäjien kutsumiseksi kaksipisteyhteys (point-to-point) istuntoihin. Se perustui HTTP:hen ja käytti siten TCP:tä kuljetusprotokollanaan. SIPv1:n lailla se oli tekstipohjainen. SCIP käytti sähköpostiosoitteita käyttäjien tunnuksina, joiden tarkoituksena oli tarjota universaali käytäjätunnus [Camarillo].

3.3. SIPv2

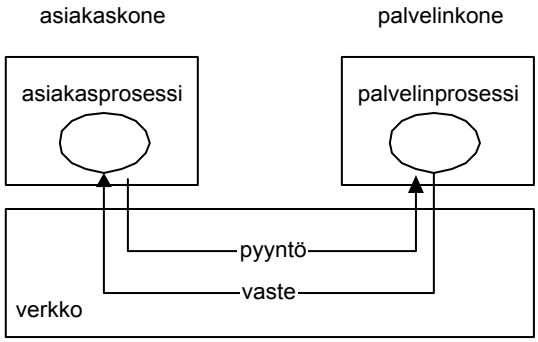
Session Invitation Protocol (SIPv2) syntyi SIPv1:n ja SCIP:n yhdistämisen tuloksena. Lopputulos piti lyhenteen SIP, mutta vaihtoi lyhenteen merkitystä. SIPv2:n Internet draftin kirjoittivat Mark Hanley, Schulzrinne ja Schooler ja se julkaistiin joulukuussa 1996. Uusi SIP perustui HTTP:hen, mutta pystyi käyttämään sekä TCP:tä että UDP:tä kuljetusprotokollanaan. SDP:tä käytettiin kuvaamaan istuntoja; SIPv2 oli tekstipohjainen.

IEFT perusti SIP työryhmän syyskuussa 1999. Maaliskuussa 2001 SIP työryhmä jaettiin kahtia. Ensimmäinen työryhmä keskittyy itse SIP:n määrittelyyn ja sen lisäosiin, vastaavasti jälkimmäinen, SIPPING-työryhmä, keskittyy sovelluksiin, jotka käyttävät SIP:tä [Camarillo].

4. SIP:n toiminnallisuus

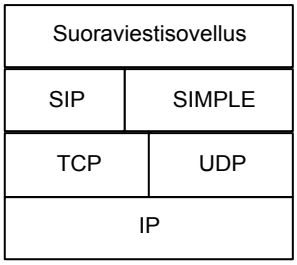
SIP [RFC 3261] luo, muokkaa ja päättää istuntoja. Sitä voidaan käyttää kutsumaan uusia käyttäjiä olemassa olevaan istuntoon tai luomaan uusi istunto. SIP ei ota kantaa itse istuntoon, vaan istunnon sisältö voi olla mitä tahansa, esimerkiksi videokonferenssi, puhelu, jaettu työpöytä tai peli-istunto. Istunnot yleensä kuvataan SDP:llä, mutta kuvaamiseen voidaan käyttää myös muita kuvausprotokollia.

SIP perustuu HTTP:hen [RFC 2616] ja kuten HTTP:kin se noudattaa asiakas-palvelin -mallia (Kuva 2). Asiakaskoneella oleva prosessi lähettää ip-paketteja suoraan palvelinkoneella olevalle prosessille, joka lähettää asiakkaalle vasteen.



Kuva 2. Asiakas-palvelin -malli.

SIP on sovelluskerroksen protokolla. Kuljetuskerroksen protokollista SIP voi käyttää molempia TCP:tä sekä UDP:tä. Verkkokerroksessa SIP kuten muutkin Internetin sovellukset käyttävät IP:tä (Kuva 3). Kuvassa 3 on esitetty SIP:tä ja SIMPLE:ä käyttävän suoraviestisovelluksen protokollapino.



Kuva 3. Suoraviestisovelluksen protokollapino.

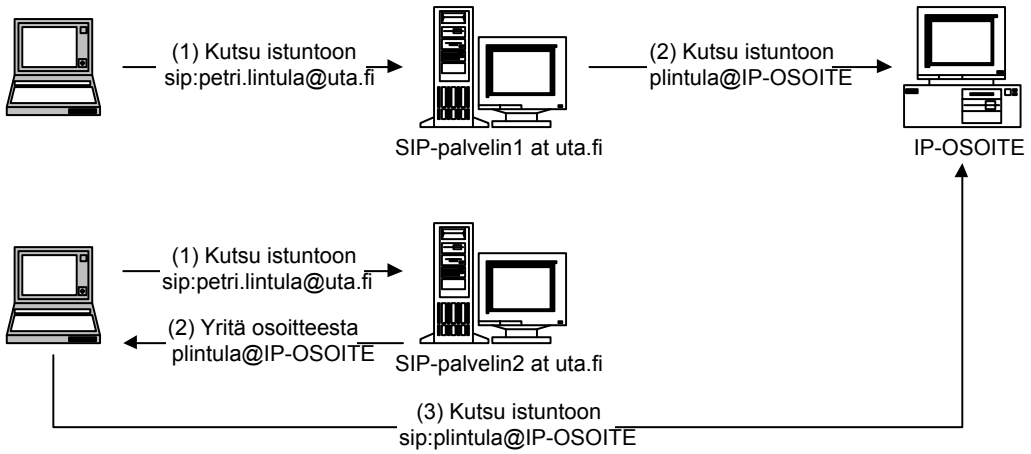
4.1. Käyttäjät

SIP ei voi kutsua käyttäjää istuntoon, ellei se tiedä mistä hänet tavoittaa. Käyttäjää ei ole sidottu tiettyyn paikkaan, vaan hän on saavutettavissa eri ip-osoitteista. Käyttäjä voi ohjata yhteyspyynnöt myös eri päätelaitteille, esimerkiksi aamulla hän ohjaa saamansa yhteyspyynnöt tietokoneelleen, iltapäivällä pda-laitteelleen ja illalla matkapuhelimeensa.

Käyttäjä tunnistetaan SIP-ympäristössä SIP Uniform Resource Locatorin (URL) avulla. SIP URL:n muoto on samanlainen kuin sähköpostiosoitteenkin, eli se koostuu käyttäjänimestä ja verkkotunnuksesta, esimerkiksi SIP:petri.lintula@uta.fi.

Jotta käyttäjä löytyisi, hänen pitää rekisteröidä sijaintinsa, ip-osoitteensa, SIP-palvelimelle. Käyttäjän rekisteröidyttyä SIP-palvelin osaa lähettää kaikki

käyttäjälle tulevat pyynnöt käyttäjän päätelaitteelle. SIP-palvelin voi toimia kahdella eri tavalla, joko välipalvelimena (proxy) tai uudelleenohjaavana (redirect) palvelimena. Välipalvelin lähettää viestin suoraan oikeaan osoitteeseen, kun taas uudelleenohjaava palvelin lähettää takaisin tiedon mistä käyttäjä löytyy (Kuva 4). Kuvassa 4 SIP-palvelin1 toimii välipalvelin periaatteella, kun taas SIP-palvelin2 toimii uudelleenohjaus periaatteella.



Kuva 4. SIP-palvelinten toimintatilat.

5. SIP-viestit

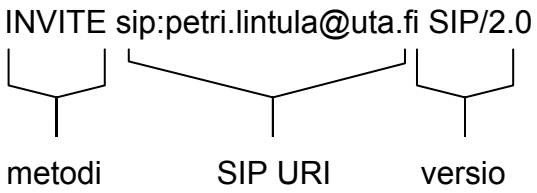
SIP-viestit koostuvat neljästä osasta: aloitusrivistä, yhdestä tai useammasta otsikkorivistä, tyhjästä rivistä ja viestin rungosta. Viestin jokaisen rivin tulee päättyä rivinvaihtoon. Aloitusrivi sisältää joko pyyntörivin tai vasterivin. Viestillä ei tarvitse olla runkoa, mutta tyhjä rivi otsikkorivin tai otsikkorivien jälkeen on pakollinen. Viestien muoto noudattaa Internetin standardoitua viestiformaattia [RFC 2822].

SIP tapahtumaksi kutsutaan tiettyyn viestiin liittyvää viestinvaihtoa asiakkaan ja palvelimen välillä. Tapahtuma sisältää aina pyynnön, yhden tai useampia tilapäisiä vasteita sekä yhden lopullisen vasteen.

5.1. Pyyntö

SIP-pyyntö tunnistetaan pyyntörivistä (Koodiesimerkki 1). Pyyntörivi sisältää metodin nimen, pyynnön vastaanottajan URI-osoitteen sekä protokollan version. SIP-määrittely määrittelee kuusi eri SIP-metodia: INVITE, ACK,

OPTIONS, BYE, CANCEL ja REGISTER (Taulukko 1). Lisäksi SIP:n lisäosat voivat määritellä lisää metodeja.



Koodiesimerkki 1. SIP pyyntörivi.

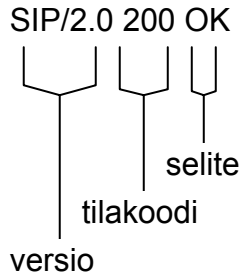
Taulukko 1. SIP-metodit

SIP-metodi	Selite
INVITE	INVITE-pyyntö kutsuu käyttäjän istuntoon, viestin runko sisältää istunnon kuvauksen.
ACK	ACK-viesti lähetetään lopullisen viestin vastaanottamisen varmistamiseksi. Asiakas lähettää INVITE-pyynnön ja saatuaan vasteen pyyntöön hän lähettää ACK-viestin. Näin saavutetaan kolmiosainen kättely: INVITE – final response – ACK.
CANCEL	CANCEL-pyyntö peruuttaa toteutumattoman tapahtuman. Mikäli INVITE-pyyntö on lähetetty, mutta lopullista vastausta ei ole vielä saatu, INVITE-pyynnön käsittely lopetetaan.
BYE	BYE-pyynnöllä käyttäjä poistuu istunnosta.
REGISTER	REGISTER-pyynnöllä käyttäjä ilmoittaa palvelimella nykyisen sijaintinsa.
OPTIONS	OPTIONS-pyynnöllä käyttäjä kysyy palvelimelta sen ominaisuuksista, kuten mitä metodeja ja mitä istunnon kuvauskieliä se tuntee.

5.2. Vaste

Jokaisen pyynnön kohdalla palvelin luo vasteen. Jokaisella vasteella on tila-koodi, joka ilmaiseen sen tilan. SIP-vaste tunnistetaan vasteen tilarivistä

(Koodiesimerkki 2). Vasteen tilarivi sisältää protokollan version, tilakoodin sekä tekstimuotoisen seliteosan.



Koodiesimerkki 2. SIP-vasteen tilarivi.

Koodit ovat kokonaislukuja väliltä [100…699] ja ne on luokiteltu (Taulukko 2) käyttötarkoituksensa mukaisesti. Vasteet väliltä [100…199] ovat tilapäisiä, vastaavasti vasteet väliltä [200…699] ovat lopullisia.

Taulukko 2. SIP-vasteiden koodit.

Koodialue	Vasteen tyyppi	Merkitys
100-199	Informational	Pyyntö vastaanotettu, jatketaan pyynnön prosessointia.
200-299	Success	Pyyntö on suoritetu onnistuneesti.
300-399	Redirection	Pyynnön toteuttamiseksi tarvitaan lisätoimenpiteitä.
400-499	Client error	Pyyntö on virheellinen ja sitä ei voida suorittaa.
500-599	Server error	Palvelin epäonnistui oikeanmuotoisen pyynnön suorittamisessa.
600-699	Global failure	Pyyntöä ei voida suorittaa millään palvelimella.

6. Suoraviesti

Suoraviestintä (Instant messaging – IM) tarkoittaa käyttäjien välisten viestien lähetystä lähes reaaliaikaisesti. Viestit ovat yleensä lyhyitä tekstiviestejä, mutta se ei ole välttämätöntä. Suoraviestintä on usein nopeatempoista, jolloin

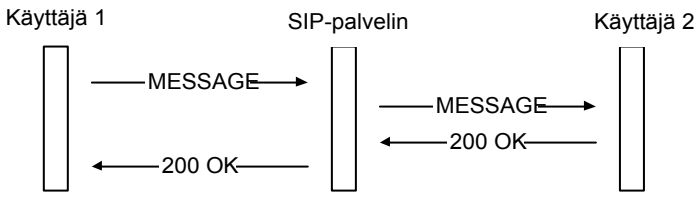
se muistuttaa paljon keskustelua. Viestejä ei yleensä tallenneta, mutta se on mahdollista. Suoraviestintä eroaa sähköpostista käyttötavaltaan, joka aiheuttaa useiden lyhyiden viestien lähettämisen edestakaisin muodostaen näin keskustelun.

Suoraviestintä itsessään on ollut olemassa jo jonkin aikaa. Aikaisempia suoraviestintä toteutuksia ovat muunmuassa zephyr ja IRC. Viime aikoina suoraviestinnän yhteyteen on liitetty läsnäolotieto (presence) ja kaverilistat (buddy list). Tällöin listalla olevan henkilön tullessa paikalle, käyttäjä saa ilmoituksen tapahtuneesta. Tähän asti toteutustavat ovat olleet sovelluskohtaisia, joten eri sovellukset eivät ole voineet toimia keskenään. SIP yrittää tuoda tähän muutosta. IETF on määritellyt yleiset vaatimukset ja mallin läsnäolo ja suoraviesti protokollille, jotka on kuvattu julkaisuissa RFC 2778 ja RFC 2779.

SIP:hen on tehty laajennus [SIP message], joka mahdollistaa suoraviestien käsittelyn. Koska kyseessä on laajennos, se perii automaattisesti SIP-määrittelyn ominaisuudet. Suoraviestin lähetykseen käytetään MESSAGE-viestiä. Itse viesti kirjoitetaan MESSAGE-osan runkoon, josta vastaanottaja osaa sen tulkita. Viestin sisältö voi olla mitä tahansa MIME-koodattua sisältöä, mutta yleensä sisältö on tekstiä.

Jokaista suoraviestiä käsitellään yksittäisenä viestinä, eikä peräkkäisistä viesteistä muodostu keskustelua. Suoraviestinnälle on olemassa myös istunto malli, jossa yksittäinen viesti kuuluu johonkin istuntoon [Session messaging]. Istunnot aloitetaan tällöin INVITE-viestillä ja lopetetaan BYE-viestillä. Tässä kuitenkin keskitytään käsittelemään suoraviestintää yksittäisten viestien kannalta.

Käyttäjä lähettää MESSAGE-tyyppisen viestin toiselle käyttäjälle. Viesti voi mennä suoraan toiselle käyttäjälle, yleensä se kuitenkin kulkee yhden tai useamman SIP-palvelimen kautta. MESSAGE-viestin saatuaan käyttäjä lähettää OK-viestin viestin lähettäjälle (Kuva 5). Näin saadaan varmistus viestin perillemenosta.



Kuva 5. MESSAGE-viestin välitys käyttäjien välillä.

Kuvassa 5 esitetyn MESSAGE-viestin muoto on esitetty koodiesimerkissä 3. Vastaavasti OK-viestin muoto on esitetty koodiesimerkissä 4. Viestin vastaanottajan osoite on määritelty Instant Message URI:lla muodossa im:user@domain. IM URI:t ovat abstrakteja ja ne on yleensä muunnettu oikealle osoitteelle CPIM -määritelmän mukaisesti [CPIM].

```
MESSAGE im:user2@domain.com SIP / 2.0
Via: SIP / 2.0 / TCP 172.16.200.86:5060
From: sip:user1@domain.com
To: sip:user2@domain.com
Contact: sip:smsClient@172.16.200.86
Call-ID: smsClient481948@172.16.200.86
CSeq: 1 MESSAGE
Content-Type: text / plain
Content-Length: 11
<tyhjä rivi>

Hello world
```

Koodiesimerkki 3. MESSAGE viesti.

```
SIP / 2.0 200 OK

Via: SIP / 2.0 / TCP 172.16.200.86:5060

From: sip:user2@domain.com

To: sip:user1@domain.com

Call-ID: smsClient481948@172.16.200.86

CSeq: 1 MESSAGE

Content-Length: 0
```

Koodiesimerkki 4. OK-viesti.

Käyttäjä ei välttämättä tiedä viestin vastaanottajan olinpaikkaa ja näin ollen hänen päätelaitettaan. Erot päätelaitteiden ominaisuuksissa saattavat johtaa siihen, että vastaanottaja ei pystykään lukemaan viestiä tai jotain sen osaa viestistä. OPTIONS-viesti mahdollistaa vastaanottajan päätelaitteen ominaisuuksien kyselyn, mutta käyttäjää ei voi vaatia kysymään jokaisen vastaanottajan päätelaitteen ominaisuuksia.

Kaikkien päätelaitteiden tulee pystyä käsittelemään tekstimuotoisia viestejä. Viestien sisältöjen ollessa suuria ja muodostuessa eri mediatyypeistä, kuten kuvista ja videoleikkeistä, ei voida olettaa kaikkien päätelaitteiden osaavan käsitellä niitä. Erityisen haastavaa tämä on mobiilipäätelaitteiden kohdalla, jossa mukaan tulee muita rajoituksia kuten näytön koko ja resoluutio, muistin määrä sekä tuetut viestiformaatit. Viesti siis voi olla liian suuri mahtuakseen päätelaitteen muistiin.

Päätelaiteriippumattomuuden saavuttamiseksi on ehdotettu, että viestit muutetaan vastaanottajan päätelaitteelle sopivaksi. Tämä muunnos voi tapahtua lähettäjän päätelaitteessa tai todennäköisemmin SIP-palvelimella. Näin lähettäjän ei tarvitse huolehtia viestin vastaanottajien päätelaitteiden ominaisuuksista. Vastaanottaja taasen saa viestin, joka on muokattu hänen päätelaitteelleen, eikä kaikille vastaanottajille sopivaan formaattiin [Message adaptation].

7. Tietoturva

Tietoturva on oleellinen asia, koska tietoa lähetetään Internetissä, jota pidetään vihamielisenä ympäristönä. Tietoturva SIP:ssä rajoittuu istunnon luontiin ja sen muokkaamiseen, kuten itse SIP:kin. Tämän vuoksi istunto voidaan perustaa turvallisesti, mutta istunnossa lähetettävä data voi olla salaamatonta. Tällöin data voidaan helposti kaapata ja tulkita.

7.1. Käyttäjien autentikointi

SIP perustuu HTTP:hen, joten se voi käyttää HTTP:n koostetunnistetta (digest) [RFC 2617] autentikointimekanisminään. HTTP:ssä on myös perustunnistusmekanismi (basic), jota ei suositella käytettäväksi heikon tietoturvansa takia. Palvelin tai käyttäjän päätelaite voi milloin tahansa haastaa toisen osapuolen todistamaan henkilöllisyytensä.

Perustunnistuksessa asiakas lähettää käyttäjätunnuksen ja salasanan valtuustietonaan. Käyttäjätunnus ja salasana lähetetään tekstimuodossa, jonka takia tunnistusta ei suositella käytettäväksi. Koostetunnistuksessa näin ei tapahdu, vaikkakin sekin perustuu käyttäjätunnus ja salasana -mekanismiin. Palvelin ja asiakas lähettävät toisilleen käyttäjätunnuksesta ja salasanasta lasketun tarkastussumman, jonka perusteella palvelin tietää asiakkaan tuntevan oikean salasanan. Tällöin käyttäjätunnusta ja salasanaa ei lähetetä ollenkaan asiakkaalta palvelimelle.

7.2. Viestin eheys ja luottamuksellisuus

Viestin eheys (integrity) tarkoittaa, varmuutta ettei viestiä ole muutettu matkalla palvelimelta asiakkaalle. Viestin luottamuksellisuus (confidentiality) vastaavasti tarkoittaa, että viestin pystyy lukemaan vain vastaanottaja, jolle viesti on tarkoitettu.

Yleinen mekanismi viestin eheyden ja luottamuksellisuuden varmistamiseksi on Secure/Multipurpose Internet Mail Extension (S/MIME) [RFC 2633]. S/MIME:n avulla viestin sisältö allekirjoitetaan käyttäen salaisen ja julkisen avaimen tekniikkaa. Käyttäjä/sovellus laskee digitaalisen allekirjoituksen käyttäen salaista avainta ja viestin sisältöä. Muut käyttäjät pystyvät avaamaan viestin julkisella avaimella ja varmistamaan, että viesti on alkuperäisessä muodossa. Vastaavasti muut käyttäjät voivat salata takaisin lähetettävän viestin julkisella avaimella, jonka vain salaisen avaimen omaava käyttäjä pystyy avaamaan.

8. Läsnäolo

Liittämällä suoraviestisovellukseen läsnäolotieto saavutetaan huikea etu tavallisiin läsnäolosovelluksiin verrattuna. Esimerkiksi SMS-viestit ovat aika- ja paikkariippumattomia. Lähettäjä ei kuitenkaan voi olla varma, saavuttiko viesti vastaanottajan vai vain hänen puhelimensa. Läsnäolotiedolla pyritään poistamaan tämä ongelma. Käyttäjä voi itse asettaa oman läsnäolotilansa, esimerkiksi ”lounaalla”, ”työpaikalla”, ”kokouksessa” tai ”tavoittamattomissa”. Lähettäessään tiedon palvelimelle muut tietävät onko käyttäjä tavoitettavissa.

Kaverilistoihin voi kerätä ystäviä, jolloin käyttäjän sovellus tarkkailee listaan merkittyjen käyttäjien läsnäolotietoja. Tähän saakka protokollat tämän toteuttamiseksi ovat olleet sovelluskohtaisia, jolloin eri sovellusten keskinäinen kommunikointi on mahdotonta.

9. Yhteenveto

SIP soveltuu hyvin suoraviestintä sovellusten viestinvälitysprotokollaksi. Se ei ota kantaa välitettävään sisältöön mahdollistaen MIME-tyyppisten viestin välityksen. SIP avoin standardi, joten ohjelmistojen kynnys alkaa käyttämään sitä on pienempi kuin lisensoitavien protokollien käyttö.

Läsnäolotieden liittäminen suoraviestintään voidaan ajatella teksti- ja mms-viestien seuraavaksi askeleeksi. SIP on laitteistoriippumaton, joten sitä voidaan käyttää yhtä hyvin tietokoneissa, kämmenmikroissa sekä matkapuhelimissa. Se mahdollistaa uusia palveluita, joissa vastaanottaja tavoitetaan

aina, mikäli hän niin haluaa. Tietoturva SIP:ssä on riittävällä tasolla, jotta sitä voidaan pitää luotettavana. Voidaan toki miettiä onko mikään Internetissä toimiva palvelu loppuen lopuksi riittävän turvallinen.

Viiteluettelo

[Camarillo, 2001] Gonzalo Camarillo, *SIP Demystified*. McGraw-Hill Professional Book Group, 2001.

[CPIM] Crocker, D., Diacakis, A., Mazzoldi, F., Huitema, C., Klyne, G., Rosenberg, J., Sparks, R., Sugano, H., Peterson, J., A Common Profile for Instant Messaging (CPIM), IETF Internet draft, http://www.ietf.org/internet-drafts/draft-ietf-impp-cpim-03.txt. [25.11.2002]

[Message adaptation] Coulombe, S., Pessi, P., Costa-Requena, J., Requirements for message adaptation in the context of SIP Instant Messaging and Presence applications, IETF Internet draft, http://www.ietf.org/internet-drafts/draft-coulombe-message-adaptation-requirements-00.txt. [25.11.2002]

[RFC 768] Postel, J., User Datagram Protocol, RFC 768, *http://www.ietf.org/rfc/rfc768.txt*. [25.11.2002]

[RFC 791] Postel, J. (ed.), Internet Protocol - DARPA Internet Program Protocol Specification, RFC 791, http://www.ietf.org/rfc/rfc791.txt. [25.11.2002]

[RFC 793] Postel, J. (ed.), Transmission Control Protocol - DARPA Internet Program Protocol Specification, RFC 793, http://www.ietf.org/rfc/rfc793.txt. [25.11.2002]

[RCF 2327] Handley, M., Jacobson, V., SDP: session description protocol, RFC 2327, http://www.ietf.org/rfc/rfc2327.txt. [25.11.2002]

[RFC 2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T., Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, *http://www.ietf.org/rfc/rfc2616.txt*. [25.11.2002]

[RFC 2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L., HTTP Authentication: Basic and Digest Access Authentication, RFC 2617, http://www.ietf.org/rfc/rfc2617.txt. [25.11.2002]

[RFC 2633] Ramsdell. B. (ed.), H., S/MIME Version 3 Message Specification, RFC 2633, http://www.ietf.org/rfc/rfc2633.txt. [25.11.2002]

[RFC 2778] Day, M., Rosenberg, J., Sugano, H., A Model for Presence and Instant Messaging, RFC 2778, http://www.ietf.org/rfc/rfc2778.txt. [25.11.2002]

[RFC 2779] Day, M., Aggarwal, S., Vincent, J., Instant Messaging / Presence Protocol Requirements, RFC 2779, http://www.ietf.org/rfc/rfc2779.txt. [25.11.2002]

[RFC 2822] Resnick, P. (ed.), Internet Message Format, RFC 2822, http://www.ietf.org/rfc/rfc2822.txt. [25.11.2002]

[RFC 3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Jonston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E., SIP: Session Initiation Protocol, RFC 3261, http://www.ietf.ogr/rfc/rfc3261.txt. [25.11.2002]

[Session messaging] Rosenberg, J., Using MESSAGE for IM Sessions, IETF Internet draft http://www.ietf.org/internet-drafts/draft-rosenberg-simple-message-session-00.txt. [25.11.2002]

[SIP message] Campbell, B., Rosenberg, J., Session Initiation Protocol Extension for Instant Messaging, IETF Proposed Standard, http://www.ietf.org/ internet-drafts/draft-ietf-sip-message-07.txt. [25.11.2002]

[SIP presence] Rosenberg, J., Willis, D., Schulzrinne, H., Huitema, C., Aboba, B., Gurle, D., Oran, D., SIP Extensions for Presence, IETF Internet draft, http://www.ietf.org/internet-drafts/draft-ietf-simple-presence-07.txt. [25.11.2002]

Äänitiedon satunnaisuus

Aki Loponen

Tiivistelmä

Satunnaisten lukujen luomiseksi käytettävät menetelmät tarvitsevat satunnaisen alkuarvon, ns. siemenen. Tämän siemenen on oltava aidosti satunnainen, eli siinä ei saa olla minkäänlaista tilastollista riippuvuutta. Mikäli tilastollista riippuvuutta esiintyy on luku löydettävissä tilastollisella analyysilla. Mm. kryptologiassa on oleellista, että satunnaisluvut ovat aidosti satunnaisia, eikä ennalta arvattavia näennäissatunnaisuuksia. Tässä tutkielmassa tarkastellaan erilaisia keinoja satunnaisen siemenen luomiseksi, sekä tutkitaan äänitiedon soveltuvuutta satunnaisen siemenen lähteeksi.

Avainsanat ja -sanonnat: satunnaisuus, satunnaisluvut, ääni
CR-luokat: G.3

1. Johdantoa

Luku, joka esiintyy satunnaisesti, on satunnaisluku. Täydellistä satunnaislukua ei voi ennalta tietää, sillä se ei sisällä minkäänlaista tilastollista riippuvuutta. Käytännössä luvun satunnaisuutta on hyvin vaikea todistaa, sillä aukotonta todistusmenetelmää on hyvin vaikea kehittää, mistä luonnollisesti seuraa, että aidosti satunnaisten lukujen ja lukusarjojen luonti on myös hyvin vaikeaa ja ongelmallista.

Monissa sovelluksissa käytetään ns. *näennäisiä satunnaislukuja* (pseudo-random numbers), jotka sisältävät tilastollista riippuvuutta ja jotka siten eivät ole aidosti satunnaisia lukuja. Nämä voivat olla kuitenkin käteviä tehtävissä, joissa luotujen lukujen aito satunnaisuus ei ole oleellista ja joissa nämä luvut tulee voida luoda nopeasti ja laskennallisesti kevyesti. Näennäiset satunnaisluvut eivät siis sovellu esim. kryptografisiin tarkoituksiin, sillä ne voidaan kryptoanalyysilla löytää.

Satunnaislukugeneraattori (random number generator, RNG) on menetelmä, joka muuttaa sille syötetyn satunnaisen siemenen pidemmäksi, yksilölliseksi ja *näennäissatunnaiseksi* (pseudo-random) tulostesarjaksi. Täydellinen, eli turvallinen, satunnaislukugeneraattori on silloin, kun sen tulostesarjoja ei voi erottaa kaikista mahdollisista samanpituisista sarjoista [Han and

Hemaspaandra, 1996], ts. tulostesarjat jakautuvat tasaisesti. Satunnaislukugeneraattorin täydellisyys on kuitenkin hankalasti todistettava ominaisuus.

Siemenen satunnaisuus on kryptografisessa mielessä hyvin tärkeää. Mikäli siemen on näennäisesti satunnainen ja ulkopuolinen taho saa haltuunsa siemenen löytämiseen käytetyn näennäisesti satunnaisen menetelmän sekä siemenestä luodun tulostesarjan, voi hän periaatteessa luoda siemenen uudelleen. Turvallisen satunnaislukugeneraattorin tiettyjä tulostesarjoja ei voi tuottaa uudelleen [Schneier, 1996]. On huomioitava, että kaikkein satunnaisinkin tulostesarja on mahdollista arvata; satunnaisuuden vahvuus perustuu tilastollisen riippumattomuuden lisäksi suureen kombinaatioiden määrään, mikä vähentää arvauksen onnistumisen todennäköisyyttä.

Tämän tutkielman toisessa luvussa tutustutaan erilaisiin menetelmiin satunnaisen siemenen hankkimiseksi. Kolmannessa luvussa perehdytään tarkemmin mahdollisuuteen käyttää ääntä ja digitaalista äänitietoa satunnaisuuden lähteenä sekä tutkaillaan tähän mahdollisesti liittyviä sudenkuoppia ja heikkouksia. Neljännessä luvussa tutkitaan konkreettisesti äänitiedostoa, sekä tehdään käsittelemättömille ja käsitellyille ääninäytteille erinäisiä kokeita, jotka ilmaisevat näytteiden satunnaisia ominaisuuksia. Samalla esitellään kaksi algoritmia, jotka pyrkivät suodattamaan näytteitä siten, että syntyisi satunnaisia sarjoja. Viidennessä luvussa luodaan vielä katsaus tehtyihin kokeisiin sekä pohditaan parannuskeinoja esitellyille menetelmille.

2. Keinoja satunnaisen siemenen saantiin

Pelkästään tietokoneella ei voi tuottaa satunnaisuutta, sillä tietokone toimii ennalta ohjelmoitujen periaatteiden mukaisesti ja täten täysin ennalta arvattavasti. Tietokoneella voidaankin tarkkailla laitteiden ja ohjelmien poimimia satunnaisuuksia ympäristöstä, sekä muuttaa tätä ”luonnollista” satunnaisuutta digitaaliseen muotoon. Tietokone ei kuitenkaan ole mikään ehdoton edellytys satunnaisuuden kirjaamiselle.

Schneier [1996] ehdottaa muutaman yksinkertaisen tavan hankkia satunnaisia bittijonoja ympäristöstä. Mitataan kahden epätasaisesti toistuvan, luonnollisen tapahtuman välinen aika ja verrataan sitä seuraavien kahden tapahtuman väliseen aikaan. Mikäli ensimmäinen aikaväli on suurempi, kirjataan muistiin bittiarvo 1, ja vastaavasti, jos toinen aikaväli on pidempi, kirjataan muistiin arvo 0. Tätä toistetaan seuraavien tapahtumien kohdalla. Näin saadaan halutun mittainen bittijono. Tällainen tapahtuma voi olla vaikkapa kävelemään opettelevan lapsen kaatuminen, mittauspisteen ohi kulkeva

ihminen tai puolison toistuva, omituinen sätky (muusta kuin mittaajasta johtuva).

Satunnaisia lukuja voi saada myös erilaisista taulukoista eri menetelmillä. Tikalla voi heittää riittävän etäisyyden päästä talouslehden pörssisivuja ja mikäli tikan yläpuolinen pörssiarvo on parempi kuin alapuolisen, saadaan bitin arvoksi yksi. Tätä kahdeksan kertaa toistamalla saa satunnaisen merkin aikaiseksi. Satunnaislukutaulukoitakin on olemassa. Vuonna 1955 Rand julkaisi miljoona satunnaista numeroa sisältävän kirjan [RAND, 1955], jossa on tosiaankin miljoona lukua väliltä 0-9. Luvut on lueteltu viiden luvun pätkissä, viisikymmentä lukua rivillä ja viisikymmentä riviä sivulla. Sivuja on nelisen sataa. Tästä kirjasta voi valita satunnaiset lukunsa vaikkapa siten, että selaa mielivaltaisesti jonkin aukeaman auki ja mielivaltaisesti valitsee jonkin lukupätkän aukeamalta, josta mielivaltaiseen suuntaan etenemällä valitsee lukuja.

Tietokoneella voidaan mitata käyttäjän aiheuttamien satunnaisten tapahtumien välisiä eroja. Hiiren osoittimen sijaintia voidaan kirjata tietyin väliajoin ja näistä koordinaattieroista laskea bitin arvo. Myös näppäimistön näppäinten painamisen välinen aika on mitattavissa, ja se soveltuu satunnaisuuden lähteeksi, mikäli kirjoittaja ei ole todella kouliintunut konekirjoittaja. Tällöin nimittäin näppäinten painamisen väli on hyvin tasainen.

Hiiren osoittimen sijainnin mittaamisessa olisi kuitenkin hyvä laskea vain yksi bitti käyttäjää tai käyttökertaa kohden, sillä mikäli käyttäjä esimerkiksi kirjoittaa tekstiä, ei hän välttämättä koske hiiren juuri ollenkaan. Näppäimistön lukemisen hankaluutena on se, että jotkin tietokonejärjestelmät keräävät näppäimistöltä tulevat viestit ensin puskuriin, josta viestit lähetetään samanaikaisesti. Tämä luonnollisesti pilaa mahdollisuuden mitata näppäimen painamisen aikaa [Eastlake *et al.*, 1994]. Heikkoudeksi on luettava myös, että mikäli satunnaisia bittejä tarvitaan hyvin pitkään salausavaimeen, täytyy näppäimistöllä kirjoittaa pitkän tutkielman verran tekstiä [Schneier, 1996]. Tällaisia pitkiä avaimia tarvitaan mm. *one-time pad* –salauksessa, jossa jokaista salattavan tekstin merkkiä vastaa merkki salausavaimessa, eli avain on yhtä pitkä kuin viestikin.

Eräs kehitetty menetelmä satunnaisen bittivirran luomiseksi perustuu vapaasti värähtelevän oskillaattorin taajuuden epätasaisuuksiin [Fairfield *et al.*, 1985]. Tämä menetelmä tuottaa vahvasti satunnaisia lukuja suhteellisen edullisesti ja kykenee havaitsemaan laitteistoviat yksinkertaisen rakenteensa vuoksi.

Myös tietokoneen tietyn kiintolevylohkon lukemiseen tarvittavia aikoja on tutkittu, ja näiden hakuaikojen vaihteluita on käytetty satunnaisten lukujen lähteenä [Fenstermacher *et al.*, 1994]. Tutkimuksen mukaan nämä vaihtelut perustuvat kiintolevyn kiinteän kuoren sisällä olevan ilmanpaineen vaihtelun aiheuttaman ilmanvastuksen muutoksiin. Kokein on todettu tämän menetelmän kykenevän tuottamaan sata satunnaista bittiä minuutissa [Fenstermacher *et al.*, 1994]. Monissa muissa laitteistopohjaisissa satunnaislukujen haravointimenetelmissä on haittana hankalasti havaittavat laitteistoviat, jotka voivat poistaa satunnaisuuden tulossarjoista. Kiintolevyihin pohjautuvat menetelmät ovat tässä suhteessa useita muita laitteistoihin perustuvia menetelmiä paremmassa asemassa, sillä kiintolevyä käyttävä käyttöjärjestelmä kykenee usein hyvinkin nopeasti havaitsemaan mahdolliset laitehäiriöt [Eastlake *et al.* 1994].

3. Ääni satunnaisuuden lähteenä

Tiivistämätön, puhdas äänitieto kotitietokoneilla tallennetaan useimmiten *WAV*–tiedostomuodossa. Tässä tiedostomuodossa tiedoston ensimmäiset 44 tavua sisältävät ohjelmille suunnattua ohjaustietoa tiedoston sisältämän äänitiedon muodosta ja suuruudesta. Loput tavut sisältävät itse äänitiedon. Tiedoston kokoon vaikuttaa äänitiedon kesto ja tallennuksessa käytetty taajuus, ei niinkään äänen ”sisältö”.

Mikäli tallennetussa äänitiedossa on täydellistä hiljaisuutta, on seurauksena sarja tavuja, joiden arvo on 0. Vastaavasti, mikäli äänitieto on hyvin tasaista, tallentuu toistuvia tavusarjoja. Tämän vuoksi täytyy satunnaiseksi siemeneksi kaavaillun äänitiedon olla ennalta arvaamatonta ja vaihtelevaa.

Vaikka äänitieto olisikin hyvin monipuolista ja vaihtelevaa, on ongelmana äänen aaltomaisuus. Aaltomaisuus on ongelma nimenomaan silloin, kun äänitietoa halutaan käyttää satunnaisuuden lähteenä, koska aaltomaisuus tuo mukanaan ennalta arvattavuutta. Äänitiedosta olisikin hyvä löytää satunnaisuutta jollakin muulla keinolla.

Äänitiedonkin suhteen voi käyttää saman kaltaisia metodeja, kuin luvussa 2 on mainittu. Menetelmä voisi laskea tietyn äänentason ylittävien piikkien etäisyyttä ja näitä vertaamalla määrätä bitin arvon. Mikrofoni voisi olla jatkuvasti päällä jossakin tilassa, jossa on tarpeeksi vaihtelevaa hälyä ja hälinää, ja jossa millään muulla taholla ei ole äänityslaitteita. Tällöin tietokoneelle tulisi jatkuvasti äänitietoa, josta eri menetelmin voitaisiin sitten satunnaisuutta turvallisesti muodostaa. On huomioitava, että hälyn ei tietenkään auta olla tasaista, eikä tasaisesti toistuvaa.

Tietoturvallisuuden kannalta on tärkeää, että käytettävä äänivirta saadaan vasta kyseisellä hetkellä suoraan satunnaislukugeneraattoriin. Näin ulkopuolinen taho ei saa tietoa käytetystä äänestä. Mikäli äänitieto otetaan julkiselta tallenteelta, ulkopuolisen tahon tarvitsee vain käydä kaikki julkinen materiaali läpi. Vaikka valikoiman luulisi olevan suunnattoman suuri, on valikoima oikeilla menetelmillä käyty suhteellisen nopeasti läpi [Eastlake *et al.*, 1994]. Yksityinen ja ainutkertainenkaan tallenne ei ole turvallinen, ellei sitä huolellisesti hävitä heti käytön jälkeen, sillä esimerkiksi varkauden jälkeen tallenne menettää turvallisuutensa.

Hyvänä äänitiedon lähteenä on yleisesti pidetty lämpökohinaa [Eastlake et al., 1994], sekä radion kohinaa taajuuksilla, joilla ei ole lähetyksiä [Haahr, 1999]. Yleiskäyttöisimpien tietokoneiden äänikorttien mikrofoniportit ovat hyvä lämpökohinan lähde silloin, kun niihin ei ole liitetty mitään laitetta. Laadukkaampien, eritoten hifi-musiikin tekoon tarkoitettujen äänikorttien portit ovat puolestaan siinä määrin tasokkaita, että kohina on miltei olematonta.

Lyhyt näyte käsittelemätöntä äänitietoa on melko helposti arvattavissa, joten peräkkäisten bittien käyttäminen tulossarjana on arveluttavaa. Parempia tapoja voisivat olla joko bittien valinta laajemmasta tietonäytteestä tai jonkin äänitietoa muokkaavan menetelmän soveltaminen.

4. Kokeet äänitiedostoilla

Tähän tutkielmaan liittyen on tehty kokeita, jotka tutkivat äänitiedostojen satunnaisuutta. Kokeiden lähtökohtana oli selvittää, kuinka heikosti satunnaisia käsittelemättömät äänitiedostot ovat, sekä pystyykö niistä suodattamaan yksinkertaisilla algoritmeilla satunnaisuutta esiin. Kahta algoritmia sovellettiin alkuperäisiin näytteisiin ja hypoteesi oli, että kummankin algoritmin tulossarjat olisivat huomattavasti vahvempia kuin alkuperäiset näytteet. Myös arveltiin algoritmeista toisen suoriutuvan satunnaisuuskokeista toista paremmin. Luvussa 3 esitettyjen äänitiedon ominaisuuksien johdosta oli arveltavissa, että käsittelemättömistä äänitiedostoista ei olisi edes välttäväksi satunnaisuuden lähteeksi. Hieman arveltiin jo ennalta, että algoritmitkaan eivät olisi riittävän tehokkaita.

4.1. Näytteistä

Kokeissa oli näytteinä kuusi äänitiedostoa, jotka kaikki olivat samanmuotoisia ja kokoisia. Näytteet olivat 10 sekunnin pituisia, taajuudeltaan CD–tasoista (44100Hz) ja kooltaan 3528044 tavua. Tiedostot tallennettiin WAV–muodossa,

jonka ehdottomia etuja tämän tutkimuksen kannalta on äänen tallentaminen käsittelemättömänä ja pakkaamattomana. Tiedostoista karsittiin pois ensimmäiset 44 tavua, sillä nämä sisältävät tiedostoon ja tiedostomuotoon liittyviä seikkoja, joita tiedostoa lukevat sovellukset tarvitsevat tunnistaakseen tiedoston. Karsinnasta huolimatta tiedostoja pystyi toistamaan ohjelmalla, joka salli karsittujen tietojen syöttämisen manuaalisesti ohjelmalle, jolloin ohjelma syötteiden pohjalta tulkitsi tiedostoa.

Näytteistä viisi luotiin tietokoneella ja yksi äänitettiin äänikortin mikrofoniliittimestä, johon ei oltu kytketty mikrofonia. Tämä näyte otettiin, jotta saataisiin vertailukohta muille, keinotekoisemmille, näytteille. Etukäteen odotettiinkin tämän näytteen satunnaisuuden yltävän muita korkeammalle. Mikrofoniportista äänitetylle näytteelle käytetään nimeä *micnoise*. Näytteistä kaksi, *brownoiz* ja *pinknoise*, luotiin äänenmuokkausohjelman kohinanluontifunktioilla käyttäen eri alkuarvoja. Näiden satunnaisuuden suhteen ei ennalta osannut arvella mitään, mutta varmaa oli ainakin se, että tietoturvallisuuden kannalta nämä ovat heikkoja, sillä ne ovat samoilla asetuksilla luotavissa täysin samanlaisina uudelleen. Näyte *beamray* luotiin samalla ohjelmalla kuin brownoiz ja pinknoise, mutta hieman erilaisella funktiolla; ääni muistuttaa kohinan sijasta 1960-luvun avaruussarjojen sädeaseita. Näytteet *oschell* ja *elephant* ovat puolestaan oskillaattorilla ja samplerilla luotua avaruusromua. Oschel*l* sisältää yksinkertaisen melodian, jota soitetaan oskillaattorilla. Elephant sisältää saman teeman, mutta noin seitsemän sekunnin kohdalle on lisätty oikealle kanavalle norsun töräys. Näin voi vertailla pienen eron vaikutusta satunnaisuutta mittaavissa kokeissa. Taulukossa 1 näytteet on selkeämmin eroteltuina.

4.2. Satunnaisuuden analysointi

Tiedon satunnaisuudesta voi päätellä jotain tarkastelemalla tiedon tiiviyttä, eli entropiaa. Mitä suurempi entropia vallitsee tiedossa, sitä hankalampaa on ennustaa seuraavaa merkkiä edellisten perusteella. Pelkästään hyvä entropia ei kuitenkaan takaa tiedon satunnaisuutta, joten myös muita ominaisuuksia on tarkasteltava [Haahr, 1999].

Internetistä löytynyt ilmainen testiohjelma, Ent [Walker, 1998], laskee syötetiedoston tavusarjoista entropian lisäksi neljä muuta satunnaisuutta mittaavaa arvoa, joten kyseistä ohjelmaa käytettiin tutkimuksessa näytteiden analysointiin. Kuvassa 1 on esitettynä ohjelman antama tulostus.

```
Entropy = 7.980627 bits per character.

Optimum compression would reduce the size
of this 51768 character file by 0 percent.

Chi square distribution for 51768 samples is 1542.26, and randomly
would exceed this value 0.01 percent of the times.

Arithmetic mean value of data bytes is 125.93 (127.5 = random).
Monte Carlo value for Pi is 3.169834647 (error 0.90 percent).
Serial correlation coefficient is 0.004249 (totally uncorrelated = 0.0).
```

Kuva 1. Ent–ohjelman esimerkkituloste.

Entropia (entropy) kuvaa tiedon tiiviyttä ja se ilmaistaan tulosteessa joko bittiä / tavu tai bittiä / bitti riippuen siitä, tarkastellaanko ohjelmalla tavu- vai bittisarjoja. Mitä lähempänä arvo on arvojen ylärajaa (ylärajat ovat kahdeksan bittiä / tavulla ja yksi bittiä / bitillä), sitä tiiviimpää tieto on ja sitä korkeampi on entropia. Tulosteen toisella rivillä ilmoitetaan, kuinka paljon optimaalinen pakkaaminen vaikuttaisi tiedoston kokoon.

Khiin neliö (Chi square) on yleisimmin käytettyjä satunnaisuuden mittareita. Se havaitsee jaksollisuuden todella herkästi ja antaa hyviä arvoja vain hyvin satunnaisille sarjoille. Tulosteessa ilmaistaan tutkittujen näytteiden (tavut tai bitit) määrä, khiin neliön jakauma, sekä prosentuaalinen arvo, joka kertoo, millä todennäköisyydellä täysin satunnainen sarja ylittää näytteestä lasketun jakauman. Mikäli prosentuaalinen arvo on alle 1% tai päälle 99%, ei näytesarja ole juurikaan satunnainen. Mikäli arvo on välillä 1%-10% tai 90%-99%, ei näytesarja todennäköisesti ole satunnainen. Mitä lähempänä arvo on 50%:a, sitä satunnaisempi se on.

Aritmeettinen keskiarvo (arithmetic mean) testissä on yksinkertaisesti laskettu yhteen kaikkien tavujen (tai bittien) arvot ja jaettu saatu summa tavujen (tai bittien) lukumäärällä. Mitä satunnaisempi näytesarja on, sitä lähempänä tämä arvo on arvoa 127,5 (bittien tapauksessa lähestytään arvoa 0,5).

Piin arvio käsittelee näytettä kuuden tavun jaksoissa laskien jokaiselle jakson kohdalle, täyttääkö se tiettyjä vaatimuksia. Jaksojen, jotka täyttävät vaatimukset, osuudesta jaksojen kokonaismäärään voidaan laskea arvio piille. Tulosteessa näkyy tämä piin arvio, sekä prosenttiluku, joka ilmaisee, kuinka pahasti arvio meni pieleen. Mikäli prosentuaalinen virhe on enemmän kuin 1%, ei tämän testin valossa näytesarjaa voida pitää satunnaisena. On huomioitava, että tämän mittarin antamien piin arvioiden yläraja on 4, joten pahasti epäonnistuneiden näytteiden virheellisyyttä ei voida tarkasti vertailla,

mikäli virhe ylittää 27,32%:a. On myös huomioitava, että tähän tulokseen ei vaikuta, tutkitaanko bittijonoa tavu- vai bittikohtaisesti.

Sarjakorrelaatio (serial correlation) kertoo kuinka vahvasti jokainen tavu näytteessä riippuu edellisestä tavusta. Satunnaisella näytesarjalla tämä arvo lähenee nollaa, kun taas hyvin ennalta arvattavalla tiedolla (pakkaamattomat bittikartat yms.) tämän testin tulos lähenee ykköstä.

Mikään edellä mainituista kokeista ei anna yksistään muuta kuin viitteitä näytesarjan satunnaisuudesta, mutta mikäli kaikki antavat optimitulosta lähellä olevan tuloksen samalle näytesarjalle, voidaan tällöin melko varmasti todeta näytesarjan olevan satunnainen. On kuitenkin muistettava, että vaikka jokin näytesarja antaisikin optimaaliset arvot jokaisesta testistä, ei se tarkoita, että näytesarja olisi turvallisesti satunnainen. Mikäli sama näytesarja voidaan tuottaa uudelleen samoin menetelmin, ei se ole satunnainen.

4.3. Ääninäytteiden käsittely

Random.org [Haahr, 1999] käyttää satunnaislukujen tuottamiseen metodia, joka oli pohjana tämän tutkimuksen ensimmäiselle algoritmille, joskin paljon yksinkertaisempana. Toiseen algoritmiin idea tuli Schneieriltä [1996]. Idea oli nimenomaan, että ensimmäinen algoritmi toimisi toisena verrokkina, kun tarkastellaan toisen algoritmin tuloksia. Ensimmäisen algoritmin ei odotettu pärjäävän toiselle algoritmille, mutta ei sen odotettu tehtävässään aivan epäonnistuvankaan.

Ensimmäinen algoritmi lukee lähdetiedostosta merkin (tavun) kerrallaan muistiin, käsittelee sitä ja lukee sitten seuraavan. Käsiteltävästä merkistä luetaan järjestyksessä kaksi bittiä, joiden arvoja verrataan. Mikäli bitit ovat samat, siirrytään suoraan seuraaviin kahteen bittiin. Mikäli bittien arvot eroavat, kirjataan muistiin ensimmäisen bitin arvo ja siirrytään lukemaan seuraavaa paria. Kun merkin kaikki bitit on käyty läpi, luetaan uusi merkki. Kun muistiin on kirjattu kahdeksan bittiarvoa, muodostetaan näistä merkki, joka kirjoitetaan tulostetiedostoon, ja tyhjennetään muisti uusia bittiarvoja varten.

Toinen algoritmi lukee lähdetiedostosta kaksi merkkiä kerrallaan. Merkkejä verrataan toisiinsa numeerisesti (merkeillähän on omat numeroarvoihin perustuvat koodinsa) ja mikäli merkkien arvot ovat samat, ts. merkit ovat samat, ei niitä käsitellä sen enempää, vaan luetaan kaksi seuraavaa merkkiä. Mikäli ensimmäinen merkki on arvoltaan suurempi, talletetaan muistiin bittiarvo nolla. Mikäli jälkimmäinen merkki on suurempi, talletetaan muistiin bittiarvo yksi. Kun bittiarvoja on muistissa kahdeksan kappaletta, muodos-

tetaan niistä ensimmäisen algoritmin tapaan merkki, joka kirjoitetaan kohdetiedostoon.

Kumpikaan algoritmi ei varsinaisesti pakkaa lähdetiedostoa, vaan siitä muodostetaan eri menetelmillä uusi tiedosto, jonka rakenteeseen lähdetiedosto vaikuttaa. Koska tietoa hukattiin lähdetiedostoa käsiteltäessä, ei ole mahdollista palauttaa lähdetiedostoa algoritmilla muodostetusta kohdetiedostosta.

On myös huomattava, että kumpikin algoritmi käsitteli alkuperäisiä näytetiedostoja, eivätkä ne käytä lähdetiedostoina toistensa tulostiedostoa. Tulostetiedostot nimettiin lähdetiedoston mukaisiksi paitsi, että tiedostopäätteet vaihtuivat. Ensimmäisen algoritmin tulostetiedostojen päätteeksi tuli *.op1* ja toisen *.op2*. Alkuperäiset tiedostot (eli algoritmien lähdetiedostot) säilyttivät oman *.wav* –päätteensä.

4.4. Ääninäytteiden analysointi

Kun näytteet oli käsitelty, oli niiden lukumäärä kasvanut kuudesta kahdeksaantoista. Jokainen tiedosto syötettiin kahteen kertaan Ent–ohjelmalle: ensin tiedostoa analysoitiin tavuittain ja sitten biteittäin. Ensin analysoitiin alkuperäiset, käsittelemättömät tiedostot, sitten seurasi ensimmäisellä algoritmilla käsitellyt ja lopuksi toisella algoritmilla käsitellyt. Mikäli ainakin toinen algoritmeista voitaisiin katsoa toimivaksi, olisi ainakin yhden tiedoston kaikkien tulosten oltava lähellä optimia.

4.4.1. Tulokset

Taulukkoon 1 on koottu käsittelemättömien näytetiedostojen tulokset Ent–testeistä. Parhaiten sekä tavu- että bittikohtaisista kokeista selvisivät *beamray.wav* ja *brownoiz.wav*. Kummankin entropia-arvot ovat hyviä: tiedostojen optimaalinen pakkaus olisi bittitasolla vain parin prosentin luokkaa. Aritmeettiset keskiarvot ovat suhteellisen lähellä optimia, eikä sarjakorrelaatiokaan kaukana optimista ole. Piin arvioinnissa beamray.wav on omaa luokkaansa alle prosentin virheellä, vaikkei 0,71% aivan optimi vielä olekaan. Valitettavasti tärkein koe, khiin neliö, antaa todella heikon tuloksen: satunnaisuutta ei juuri näytä olevan.

Näytteen nimi *.wav	Entropia (bittiä/tavu)	Optimaali tiivistys	Khiin Neliön tod.näk.	Aritm. Keskiarvo	Piin Arvion virhe	Sarja Korrelaatio
Micnoise	3,720001	53 %	0,01	58,5612	27,32 %	0,010955
Pinknoise	4,797667	40 %	0,01	63,8364	27,32 %	0,002267
Beamray	7,465352	6 %	0,01	121,2267	0,71 %	-0,005633
Brownoiz	7,432706	7 %	0,01	119,2917	3,09 %	-0,012236
Oschell	3,993004	50 %	0,01	58,6273	27,32 %	0,050448
Elephant	4,013542	49 %	0,01	58,911	27,32 %	0,048748

Näytteen nimi *.wav	Entropia (bittiä/bitti)	Optimaali tiivistys	Khiin Neliön tod.näk.	Aritm. Keskiarvo	Piin Arvion virhe	Sarja Korrelaatio
Micnoise	0,779735	22 %	0,01	0,231	27,32 %	0,449705
Pinknoise	0,811519	18 %	0,01	0,2502	27,32 %	0,363719
Beamray	0,980795	1 %	0,01	0,4186	0,71 %	0,131384
Brownoiz	0,978288	2 %	0,01	0,4135	3,09 %	0,123933
Oschell	0,782138	21 %	0,01	0,2324	27,32 %	0,4865
Elephant	0,783866	21 %	0,01	0,2334	27,32 %	0,485212

Taulukko 1. Käsittelemättömien tiedostojen tulokset.

Selvästi heikommin käsittelemättömistä tiedostoista pärjäävät, ehkä odotetustikin, *oschell.wav* ja *elephant.wav*. Muut näytteet olivat enemmänkin pelkkää kovaa kohinaa, kun nämä kaksi puolestaan sisälsivät taukoja ja vähemmän vaihtelua äänen taajuuksissa. *Micnoise.wav*:n heikot tulokset hämmästyttävät, sillä päälisin puolin se on aivan samanlaista kohinaa, kuin esimerkiksi brownoiz.wav, mutta ilmeisesti mikrofoniportti poimi hyvin vähän häiriöitä ja täten tuotti tasaista kohinaa.

Taulukossa 2 puolestaan on ensimmäisellä algoritmilla käsiteltyjen näytetiedostojen antamat tulokset, edelleen tavu- ja bittikohtaisesti. Tavukohtaisesti tutkittuna tulokset näyttävät kaikkien kokeiden osalta tasaisilta. Entropia-arvojen haarukka on käsittelemättömiin näytteisiin verrattuna pieni, mutta arvot sinänsä ovat heikkoja. Aritmeettiset keskiarvot ovat heikkoja, kuten myös piin arviointi. Sarjakorrelaatiot ovat kuitenkin kohtuullisen hyviä. Herkin, khiin neliö, antaa huonot tulokset näissäkin. Bittikohtaiset arvotkin ovat tasaisia ja heikkoja. Mikään yksittäinen näyte ei erotu muista, eli kaikki ovat tasaisen huonoja.

Näytteen nimi *.op1	Entropia (bittiä/tavu)	optimaali tiivistys	Khiin Neliön tod.näk.	Aritm. keskiarvo	Pii Arvion virhe	Sarja Korrelaatio
Micnoise	6,572959	17 %	0,01	176,785	49,71 %	0,056361
Pinknoise	7,023518	12 %	0,01	161,6031	33,16 %	0,013988
Beamray	6,678239	16 %	0,01	174,6312	47,28 %	0,017039
Brownoiz	6,548975	18 %	0,01	178,8166	51,21 %	0,069153
Oschell	6,908481	13 %	0,01	166,9019	38,48 %	0,049136
Elephant	6,90507	13 %	0,01	167,173	39,00 %	0,043919

Näytteen nimi *.op1	Entropia (bittiä/bitti)	optimaali tiivistys	Khiin Neliön tod.näk.	Aritm. keskiarvo	Piin Arvion virhe	Sarja Korrelaatio
Micnoise	0,847214	15 %	0,01	0,726	49,71 %	0,02908
Pinknoise	0,910524	8 %	0,01	0,6743	33,16 %	-0,019988
Beamray	0,859389	14 %	0,01	0,7171	47,28 %	0,014945
Brownoiz	0,840631	15 %	0,01	0,7306	51,21 %	0,014261
Oschell	0,892234	10 %	0,01	0,6908	38,48 %	-0,018674
Elephant	0,891615	10 %	0,01	0,6913	39,00 %	-0,018827

Taulukko 2. Ensimmäisellä menetelmällä käsiteltyjen tiedostojen tulokset.

Toisella algoritmilla käsiteltyinä (taulukko 3) näytteet antoivat jo aavistuksen verran parempia arvoja. Tavukohtaisessa käsittelyssä entropia ei eroa ensimmäisen käsittelyalgoritmin vastaavista tuloksista muuten kuin *oschell.op2*:n ja *elephant.op2*:n heikompina suoriutumisina. Aritmeettiset keskiarvot ovat vain hitusen ensimmäisen algoritmin tuloksia parempia. Sarjakorrelaatiot ovat keskimäärin ensimmäistä algoritmia huonompia, mutta piin arvioissa tämä toinen algoritmi tuotti huomattavasti paremmat tulokset. Hieman yllättäen muuten huonosti pärjänneet oschell.op2 ja elephant.op2 saivat piin arviosta todella hyvät tulokset: elephant.op2:n virhe on vain 0,09%. Khiin neliöt ovat edelleen tasaisen huonoja.

Näytteen nimi *.op2	Entropia (bittiä/tavu)	Optimaali Tiivistys	Khiin Neliön tod.näk.	Aritm. Keskiarvo	Piin Arvion virhe	Sarja Korrelaatio
Micnoise	6,658712	16 %	0,01	98,3761	1,59 %	0,313301
Pinknoise	6,982438	12 %	0,01	104,3437	5,56 %	0,261116
Beamray	7,269728	9 %	0,01	113,7088	4,65 %	0,133473
Brownoiz	6,914044	13 %	0,01	111,0516	9,74 %	0,070845
Oschell	5,376694	32 %	0,01	92,9296	0,31 %	0,370835
Elephant	5,505094	31 %	0,01	93,6671	0,09 %	0,368742

Näytteen nimi *.op2	Entropia (bittiä/bitti)	Optimaali Tiivistys	Khiin Neliön tod.näk.	Aritm. keskiarvo	Piin Arvion virhe	Sarja Korrelaatio
Micnoise	0,994067	0 %	0,01	0,4547	1,59 %	0,305955
Pinknoise	0,998373	0 %	0,01	0,4763	5,56 %	0,075944
Beamray	0,997957	0 %	0,01	0,5266	4,65 %	0,066809
Brownoiz	0,997126	0 %	0,01	0,5316	9,74 %	0,07952
Oschell	0,971765	2 %	0,01	0,4014	0,31 %	0,524689
Elephant	0,973577	2 %	0,01	0,4046	0,09 %	0,495325

Taulukko 3. Toisella menetelmällä käsiteltyjen tiedostojen tulokset.

Bittikohtaisesti toisen algoritmin tulostesarjat saivat todella hyviä tuloksia. Entropia-arvot olivat lähes täydellisiä: neljällä näytteellä tiivistyssuhde oli 0% ja kahdella muullakin vain 2%. Aritmeettiset keskiarvot olivat hyvin lähellä optimia. Sarjakorrelaatiot olivat suhteellisen hyviä, paitsi kahdella: oschell.op2 ja elephant.op2 suoriutuivat tälläkin kertaa muita heikommin tässä kokeessa. Khiin neliöt olivat surkeita näilläkin näytteillä.

4.4.2. Tulosten arviointia

Oletuksen mukaisesti käsittelemättömät näytetiedostot saivat heikkoja arvoja satunnaisuuskokeista. Mikäli äänitieto oli nopeasti ja epätasaisesti vaihtelevaa, olivat tuloksetkin hieman muita parempia. Käsitellyistä näytetiedostoista jotkut saattoivat yltää neljässäkin kokeessa hyvään tulokseen, mutta jäljelle jääneet kokeet antoivat heikkoja arvoja ja täten kyseenalaistivat satunnaisuuden esiintymisen.

Tiedostojen käsittely esitellyillä menetelmillä ei siis tuottanut toivottua tulosta. Menetelmät olivat hyvin yksinkertaisia, joten voisi olettaa, että paremmilla menetelmillä voisi saada parempia arvoja. Toinen algoritmi osoittautui alttiiksi äänen aaltomaisuudelle, sillä algoritmin suorittaman vertailun tulos oli arvioitavissa verrattavista tavuista: jos ensimmäinen on pienempi kuin toinen, on todennäköisempää, että seuraavista kahdesta tavustakin ensimmäinen on pienempi. Algoritmia voisi parantaa mahdollisesti siten, että vertailujen tulosta ja tavujen arvoja verrataan taulukkoon, jonka pohjalta lopullinen tulos määräytyy. Toinen keino satunnaisemman

tuloksen saamiseen voisi olla kahden tai useamman tulosnäytteen ajaminen sekoitusfunktion läpi, joka voisi käyttää yksinkertaistakin *XOR*–funktiota (exclusive or) [Eastlake *et al.*, 1994]. Tämä luultavasti poistaisi aaltomaisen jaksollisuuden.

Erityinen pettymys oli mikrofoniportista äänitetyn lämpökohinan heikko menestyminen suhteessa muihin näytteisiin, nimenomaan käsittelemättömänä. Kun kyseisen tiedoston aaltomuotoa tarkasteltiin graafisesti, havaittiin, että äänitieto ei ollut jakautunut kohinalle ominaisesti. Tästä pääteltiin, että käytössä ollut mikrofoniportti oli viallinen.

Ongelmana on kuitenkin edelleen se, että jos ulkopuolinen taho saa selville, mitä tiedostoja on käytetty ja millä menetelmillä niitä on käsitelty, voi hän tuottaa saman satunnaisen tulossarjan. Mikrofoniportista saadun näytteen oletettiin olevan pätevä tätä vastaan, sillä periaatteessa jokainen nauhoitus tuottaa yksilöllistä kohinaa. Äänitetty näyte täytyisi vain käsitellä tässä esitettyjä algoritmeja tehokkaammin. Tätä äänitettyä tietoa ei tietenkään saa pysyvästi tallentaa mihinkään, vaan sitä pitää käyttää heti ja poistaa käytön jälkeen muisteista.

5. Johtopäätöksiä

Satunnaisen bittisarjan tuottaminen ei loppujen lopuksi ole hankalaa, sillä lantin heitolla voi määrätä bitin arvon satunnaisesti. Yli kaksisataa lantin heittoa kahteenkymmeneen kertaan päivässä on kuitenkin aikaa ja hermoja vievää puuhaa, joten satunnaisten bittien luonti on järkevää automatisoida tietokoneelle. Tietokone ei kuitenkaan peruslaitteistollaan kykene tuottamaan aitoa satunnaisuutta, joten erikoistuneet laitteet tai menetelmät ovat tarpeen.

Satunnaisuuden hankkimista varten luoduissa laitteissa on kuitenkin yksittäisen käyttäjän kannalta haittana heikko saatavuus ja suuret kustannukset. Käytännöllisempiä ovat menetelmät, jotka hyödyntävät yleiskäyttöisiä tietokoneen lisälaitteita, kuten kiintolevyä, mikrofonia tai tietokoneen sisäisiä komponentteja, jolloin suuria lisäkustannuksia ei tule. Laitteita hyödyntävän menetelmän tulee kuitenkin olla vahva.

Käytettävästä satunnaisuuden lähteestä riippumatta on tärkeää, että menetelmä sisältää myös vahvan sekoitusfunktion, jolla lähteen antamasta tietovirrasta muovataan aidosti satunnainen tulostesarja. Sekoitusfunktion tulisi perustua toiseen satunnaiseen lähteeseen, ts. funktio sekoittaisi kahdesta eri lähteestä tulevat tulossarjat keskenään, eikä lähteiden välillä saa olla minkäänlaista riippuvuutta. Muutoin tulokset ovat yhtä epäonnistuneita

satunnaisuuden kannalta, kuin tämän tutkielman yhteydessä tehdyissä kokeissa.

Äänitieto varmastikin käy hyvin satunnaisuuden lähteeksi, kunhan jatkokäsittely- ja sekoitusfunktiot laaditaan huolellisesti. Äänen lähteenkin tulee olla huolellisesti valittu siten, ettei se tuota samalla tavalla käyttäytyvää ääntä toistuvasti. Lämpökohina osoittautui aiempien tutkimusten valossa tehokkaaksi ja helposti äänitettäväksi lähteeksi.

Kokeiden pohjalta oli todettavissa, että heikot menetelmät äänitiedon satunnaistamiseksi eivät ole riittäviä. Menetelmien heikkous johtui niiden yksinkertaisuudesta, sekä varsinaisen sekoitusfunktion puutteesta. Parempia tuloksia voisi syntyä jo pelkästään sekoittamalla kaksi eri menetelmillä käsiteltyä äänitiedostoa. Paremmat menetelmät olisi kuitenkin syytä kehittää.

Viiteluettelo

[Eastlake *et al.*, 1994] D. Eastlake, S. Crocker, J. Schiller, Randomness Recommendations for Security. Request For Comment, **RFC1750**, December 1994.

[Fairfield *et al.*, 1985] R.C. Fairfield, R.L. Mortenson, K.B. Coulthart, An LSI random number generator (RNG). *LNCS* **196** (1985), 203-230.

[Fenstermacher *et al.*, 1994] P. Fenstermacher, R. Ihaka, D. Davis, Cryptographic randomness from air turbulence in disk drives. *LNCS* **839** (1994), 114-120.

[Haahr, 1999] M. Haahr, Introduction to Randomness and Random Numbers (June 1999), http://www.random.org/essay.html [12.12.2002].

[Han and Hemaspaandra, 1996] Y. Han, L.A. Hemaspaandra, Pseudorandom Generators and the Frequency of Simplicity. *J. Cryptology* **9** (1996), 251-261.

[RAND, 1955] RAND Corporation, *A Million Random Digits with 100,000 Normal Deviates*. Free Press Publishers, 1955.

[Schneier, 1996] B. Schneier, *Applied Cryptography, 2nd Edition*. J. Wiley & Sons, Inc., 1996.

[Walker, 1998] J. Walker, ENT – A Pseudorandom Number Sequence Test Program http://www.fourmilab.ch/random/ [12.12.2002].

Etäisyysfunktioista tiedonlouhinnassa

Janne Lumijärvi

Tiivistelmä.
Tutkielmassa tarkastellaan etäisyysfunktioita, jotka ovat keskeinen komponentti useissa tiedonlouhintasovelluksissa. Tarkemmin perehdytään euklidiseen, HEOM- ja HVDM-funktioon. Tutkielman tarkoituksena on erityisesti pohtia etäisyysfunktioiden soveltuvuutta heterogeenisen datan käsittelyyn. Heterogeeninen data on kuvattu sekä nominaalisilla että kvantitatiivisilla attribuuteilla. Ongelmana perinteisissä homogeenisissä funktioissa on nominaalisen datan sopimaton käsittely. Heterogeenisissä funktioissa on pyritty siihen, että nominaalinen ja kvantitatiivinen data käsitellään erikseen kummallekin sopivalla tavalla. Funktioita vertailtiin käyttäen lähimmän naapurin menetelmää ja viittä aineistoa. Tulokset antavat viitteitä siitä, että heterogeeniset funktiot ovat joissakin ongelmissa homogeenisiä funktioita parempia, mutta myös aineiston ominaispiirteet vaikuttavat funktioiden tuloksiin.

Avainsanat ja -sanonnat: etäisyysfunktiot, tiedonlouhinta, koneoppiminen, luokittelu.
CR-luokat: H.2.8, I.5.3.

1. Johdanto

Tiedonlouhinta on tietojenkäsittelytieteiden osa-alue, jossa on tavoitteena etsiä implisiittisiä hahmoja ja rakenteita jostain reaalimaailman kohteesta. Näiden hahmojen ja mallien avulla pyritään saamaan hyödyllistä tietämystä kyseisestä kohteesta. Jotta hahmojen ja rakenteiden löytäminen voisi onnistua, täytyy reaalimaailman oliot pystyä erottelemaan jollain tavalla toisistaan. Voidaan sanoa, että jotkut kaksi oliota ovat samankaltaisia tai erilaisia keskenään, tai että olioiden välinen etäisyys on pieni tai suuri. Sen määrittäminen, kuinka etäällä kaksi reaalimaailman oliota ovat toisistaan, ei ole yksiselitteistä. Kahden kokonaisluvun välisen etäisyyden määrittelemiseen on olemassa hyvin vakiintunut tapa, mutta miten laskea, kuinka paljon esimerkiksi kaksi potilasdiagnoosia eroaa toisistaan? Yksi tiedonlouhinnan keskeisistä ongelmista onkin, miten määrittää olioiden välinen erilaisuus, samanlaisuus tai etäisyys. Jotta voidaan määritellä objektiivisesti reaalimaailman olioiden välisiä etäisyyksiä, täytyy aluksi sopia jonkinlainen tapa, jolla voidaan kuvata

reaalimaailman data. Paloitellaan olio aluksi ominaisuuksiin. Eri mittaustekniikoiden avulla saadaan kullekin ominaisuudelle jonkinlainen symbolinen esitys. Näin olio voidaan esittää vektorin avulla, jonka alkioina ovat mittaustulokset olion ominaisuuksista. Tämän jälkeen tarvitaan vielä funktio tai algoritmi, joka saaduista kahdesta oliovektorista määrittää kyseisiä vektoreita vastaavien olioiden välisen etäisyyden. Luonnollisesti tavoitteena on, että lasketut etäisyysarvot kertoisivat mahdollisimman hyvin olioiden todellisista suhteista. Reaalimaailman data voi olla luonteeltaan hyvinkin heterogeenistä, mikä tarkoittaa, että olioiden ominaisuudet vaihtelevat mitta-asteikoiltaan ja arvoalueiden laajuudeltaan. Tämä asettaa haasteita samanlaisuuden, erilaisuuden ja etäisyyden laskemiselle.

Tässä tutkimuksessa on tarkoituksena perehtyä funktioihin, jotka laskevat kahden olion välisen etäisyyden. Lähemmin tarkastellaan kolmea erilaista etäisyysfunktiota. Ideana on ennen kaikkea pohtia funktioiden soveltuvuutta sekamuotoisen datan etäisyyksien määrittelemiseen. Funktioiden hyvyyttä testataan käytännössä luokittelutehtävän avulla. Testissä on mukana viisi valmiiksi luokiteltua aineistoa, joista kolme on reaalimaailman aineistoa ja kaksi keinotekoista aineistoa. Aineistot luokitellaan uudelleen käyttäen lähimmän naapurin menetelmäksi kutsuttua koneoppimistekniikkaa, jonka komponenttina toimii etäisyysfunktio. Luokittelun tarkkuuden arvioinnin apuna käytetään ristiinvalidointia.

Luvussa 2 tutustutaan menetelmiin, joilla reaalimaailman data saadaan symboliseen muotoon. Luvussa 3 pohditaan tiedonlouhintaa ja koneoppimista. Luvussa 4 tarkastellaan etäisyyden käsitettä yleisellä tasolla. Luvussa 5 määritellään joitakin etäisyysfunktioita pohtien niiden soveltuvuutta heterogeeniseen aineistoon. Luku 6 on tutkielman empiirinen osuus, jossa luokitellaan erityyppistä heterogeenistä aineistoa käyttäen kuvattuja menetelmiä. Lopuksi saadut tulokset analysoidaan. Tulosten pohjalta mietitään, voidaanko kenties antaa ohjeita, missä tilanteissa funktioita tulisi käyttää.

2. Datan esitystapa

Tiedonlouhintaprosessin ensimmäinen vaihe on kuvata reaalimaailman data symbolisesti. Ensiksi tulee määrätä ne olion *ominaisuudet*, joita halutaan tutkia. Tämän jälkeen tulee eri mittaustekniikoiden avulla mitata ja esittää ominaisuudet symbolissa muodossa. Tässä luvussa määritellään datan esitystapa, jota käytetään jatkossa. Lisäksi tarkastellaan etäisyyslaskennan kannalta olennaista asiaa, olion ominaisuuksien luonnetta.

2.1. Havaintomatriisi

Yleensä tiedonlouhinta-algoritmien lähtökohta on, että tutkittava aineisto esitetään *havaintomatriisina*. Havaintomatriisi on tietyn sopimuksen mukainen havaintoaineiston esitys [Puntanen, 1998], joka esitetään tavallisesti kuvassa 2.1 esitetyllä tavalla.

$$E = \begin{array}{c} \\ r_1 \\ r_2 \\ \dots \\ r_n \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & \dots & v_m \end{array} \\ \begin{pmatrix} e_{11} & e_{12} & \dots & e_{1m} \\ e_{21} & e_{22} & \dots & e_{2m} \\ \dots & \dots & \dots & \dots \\ e_{n1} & e_{n2} & \dots & e_{nm} \end{pmatrix} \end{array}$$

Kuva 2.1. Havaintomatriisi.

Kuvassa 2.1. havaintomatriisin E sarakkeet (pystyvektorit) v_1, $v_2, \dots, v_m$ kuvaavat *muuttujia* eli *attribuutteja*. E:n rivit r_1, r_2, …, r_n (vaakavektorit) kuvaavat puolestaan *havaintoja* eli *esimerkkejä*. Voidaan ajatella, että matriisi E virittää *havaintoavaruuden*. Havaintomatriisi E on esimerkki *kaksitilaisesta matriisista* (two-mode matrix) [Puntanen, 1998]. Toisin sanoen siinä riveillä ja sarakkeilla on eri merkitys, koska rivit kuvaavat esimerkkejä ja sarakkeet attribuutteja. Toinen mahdollinen esitystapa olisi sellainen, jossa rivit vastaisivat attribuutteja ja sarakkeet havaintoja. Matriiseja merkitään jatkossa isolla kursivoidulla kirjaimella. Merkintää $E_{n\times m}$ käytetään kuvattaessa matriisin rivi- ja sarakemäärää eli matriisin *dimensiota*.

2.2. Mitta-asteikoista

Attribuutin *mitta-asteikko* ilmaisee, kuinka paljon attribuutti kertoo olioiden välisistä suhteista. Reaalimaailman data on monidimensioista ja harvoin mitta-asteikon suhteen homogeenistä. Huomioimalla mitta-asteikkojen ominaispiirteet etäisyysfunktiossa vältetään tilanne, jossa etäisyyksien laskennassa käytettäisiin mitta-asteikolle sopimattomia operaatioita. Myöhemmin tarkastellaan etäisyysfunktioita, jotka käsittelevät attribuutteja mitta-asteikkoinformaatiota hyödyntäen.

Attribuutit jaetaan tavallisesti mitta-asteikkonsa perusteella *kvalitatiivisiin* sekä *kvantitatiivisiin*. Tämän yleisemmän tason jaottelun jälkeen attribuutit jaotellaan vielä *nominaalisiin*, *ordinaalisiin*, *välimatka-asteikollisiin* ja *suhdeasteikollisiin*. Jaottelu on esitetty kuvassa 2.2. Lähteistä riippuen kvalitatiivisiin attribuutteihin luetaan nominaaliset sekä ordinaaliset attribuutit tai pelkästään nominaaliset. Kvantitatiivisiin luetaan välimatka-asteikolliset ja suhdeasteikolliset sekä joissain lähteissä näiden lisäksi ordinaaliset. Usein käytännön

sovelluksissa ei ole tarpeen erotella välimatka-asteikollisia ja suhdeasteikollisia attribuutteja, koska ne ovat luonteeltaan hyvin lähellä toisiaan. Tässä tutkielmassa kvalitatiivisiksi attribuuteiksi ymmärretään ainoastaan nominaaliset attribuutit. Usein joudutaan pohtimaan myös ovatko kvantitatiiviset attribuutit luonteeltaan *diskreettejä* tai *jatkuvia*. Jatkossa tässä työssä tarkoitetaan diskreeteillä sellaisia attribuutteja, joiden arvot on koodattu kokonaisluvuiksi tiedonkeruuvaiheessa ja jatkuvilla sellaisia, joiden arvot on tallennettu reaalilukuina. Mitta-asteikot lueteltiin edellä heikoimmasta vahvimpaan; vahvempi sisältää aina myös heikomman asteikon ominaisuudet. Esimerkiksi suhdeasteikolliset attribuutit sisältävät kaikkien muiden mitta-asteikoiden ominaisuudet.

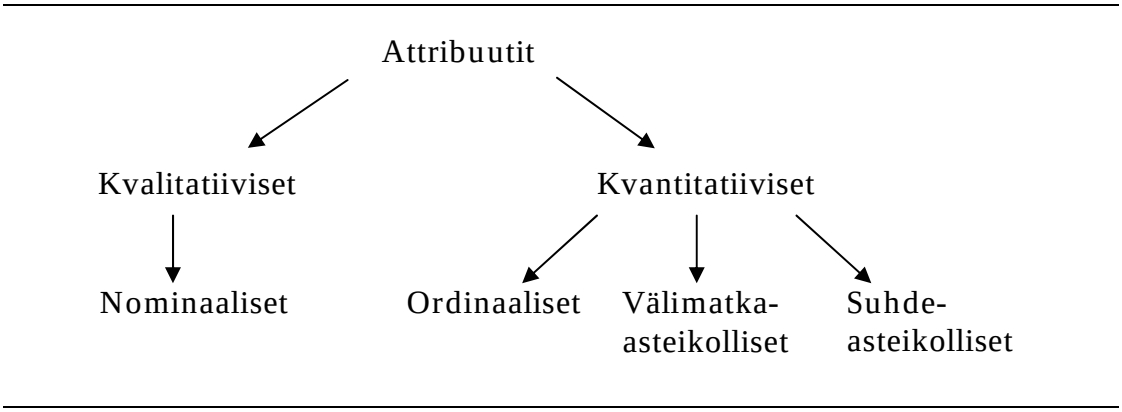
Nominaaliset attribuutit ovat attribuutteja, joiden arvoille ei voida määrittää järjestystä. *Väri* on esimerkki tällaisesta attribuutista. Eri värejä ei voi selvästikään laittaa yksiselitteiseen, yleisesti hyväksyttyyn järjestykseen. Etäisyyslaskentaa varten attribuutin arvot tulee *koodata* jollain tavalla. Nominaalisen attribuutin arvot voidaan koodata periaatteessa millä tavalla tahansa, koska järjestystä niiden välillä ei ole. Helpoin ja yleisin tapa on esittää ne kokonaislukuina. Väriesimerkin tapauksessa voitaisiin määrätä esimerkiksi *punainen* = 0, *valkoinen* = 1, *musta* = 2 jne. Toisaalta yhtä hyvin, vaikkakin turhan hankalasti, voitaisiin määrätä *punainen* = 0,5, *valkoinen* = 239, *musta* = 67,1. Juuri nominaaliset attribuutit ovat ongelmallisimpia etäisyyslaskennassa, koska ne kertovat ainoastaan, eroaako kaksi oliota toisistaan. Ne eivät kerro sitä, millä tavalla tai kuinka paljon oliot eroavat. Näin ollen esimerkiksi kahden nominaalisen arvon erotus ei anna mielekästä kuvaa olioiden etäisyydestä. Nominaalisen attribuutin erikoistapaus on *binäärinen* attribuutti, josta esimerkkinä *sukupuoli*. Binäärinen attribuutti voi saada kaksi eri arvoa. Binäärisille attribuuteille on runsaasti samankaltaisuusfunktioita, joista keskeisimmät esittelee muun muassa Gower [1985].

Ordinaaliset eli *järjestysasteikolliset* attribuutit ovat attribuutteja, joiden arvot voidaan asettaa mielekkääseen järjestykseen, mutta joiden kahden peräkkäisen arvon välimatka ei välttämättä ole yhtä suuri. Esimerkkinä ordinaalisesta attribuutista voi ajatella vaikkapa taudin *vakavuusastetta*. Voidaan sanoa, että joku tila on vakavampi kuin joku toinen. Sen sijaan sitä, kuinka suuri ero taudin vakavuudessa on, ei voida objektiivisesti tämän attribuutin perusteella määrittää. Koodaustapa on ordinaalisilla attribuuteilla vapaa, kunhan arvojen järjestys säilyy. Kahden ordinaalisen arvojen erotus kertoo enemmän kuin kahden nominaalisen, mutta sekään ei kerro kuin arvojen järjestyksen.

Välimatka- eli *intervalliasteikolliset* attribuutit poikkeavat ordinaalisista siinä, että niiden arvojen välit ovat yhtä suuret. Esimerkki välimatka-asteikollisesta attribuutista on lämpötila muilla kuin Kelvin-asteikoilla mitattuna. Tämä sisältää sekä nominaalisen, että ordinaalisen ominaisuuden. Lisäksi tiedetään, että esimerkiksi lämpötilojen 99 ja 100 *Celsiusta* ero on yksi Celsius-aste. Toisaalta mitään tietoa ei menetetä, vaikka käytettäisiin Celsiuksen asemesta mitta-yksikkönä *Fahrenheit*-asteikkoa. Näin ollen esimerkiksi 100 Celsius-astetta ei tarkoita kaksi kertaa niin lämmintä kuin 50 Celsius-astetta.

Suhdeasteikolliset attribuutit kertovat olioiden suhteesta kaikkein eniten. Paitsi, että sisältävät muiden asteikoiden ominaisuudet, suhdeasteikollisilla attribuuteilla on lisäksi *absoluuttinen nollapiste*, jossa ominaisuus ikään kuin häviää. Tämä tarkoittaa, että ne ilmaisevat myös olion suhteellisen eron toiseen olioon. Esimerkki suhdeasteikollisesta attribuutista on *paino* tai lämpötila mitattuna *Kelvin*-asteikolla.

Jotkut etäisyysfunktiot hyödyntävät kaikkien neljän mitta-asteikon ominaispiirteitä. Tässä tutkielmassa käsiteltävien etäisyysfunktioiden kannalta voidaan kuitenkin tyytyä suhteellisen yksinkertaiseen kaksijakoon. Attribuutit jaotellaan asteikoiltaan *nominaalisiin* ja *kvantitatiivisiin*. Tässä jaottelussa kvantitatiivisiin kuuluu siis ordinaaliset, välimatka-asteikolliset ja suhdeasteikolliset attribuutit.



Kuva 2.2. Attribuuttien jaottelu.

2.3. Muunnokset

Tiedonlouhinnassa datalle tehdään usein erilaisia *muunnoksia*, jotta valittu louhintamenetelmä sopisi paremmin ongelmanratkaisuun. Reaalimaailman dataa käsiteltäessä ongelmia aiheuttavat esimerkiksi erilaiset mittayksiköt ja attribuuttien erisuuret vaihteluvälit. Lisäksi jokaisen tutkitun olion jokaiselle attribuutille ei välttämättä pystytä mittaamaan arvoa.

Normalisointi eli *standardointi* on menetelmä, jolla havaintomatriisin attribuutit normalisoidaan eli niiden painoarvot yhdenmukaistetaan. Tällöin muunnetaan attribuuttien arvoja niin, että kaikkien attribuuttien arvot saadaan jakautumaan jollekin samanpituiselle välille. Oletetaan, että kaikki havaintomatriisin arvot ovat positiivisia reaalilukuja. Yksinkertaisin tapa normalisoida tällainen matriisi on jakaa attribuutin arvo kyseisen attribuutin arvojen *vaihteluvälillä*. Tällöin jokaisen attribuutin arvoalueeksi tulee [0,1]. Ongelma tässä tavassa on se, että arvot eivät välttämättä jakaudu tasaisesti. Niinpä onkin esitetty normalisointi käyttäen attribuutin arvojen *keskihajontaa* [Wilson and Martinez, 1997]. Normalisointiin palataan myöhemmin luvussa 5 heterogeenisiä etäisyysfunktioita tarkasteltaessa.

Monet tiedonlouhintatekniikat vaativat, että havaintomatriisi on täydellinen. Tästä syystä datan puuttuvat arvot on usein imputoitava. *Imputoinnilla* tarkoitetaan puuttuvien arvojen paikkausta uusilla estimoiduilla arvoilla. Oletetaan havaintomatriisin alkioiden olevan reaalilukuja. Yksinkertaista on käyttää esimerkiksi *keskiarvoa*, *mediaania* tai *moodia*. Suosittu, mutta edellisiä monimutkaisempi menetelmä on *maksimiuskottavuusestimointi* (expectation maximization), jonka kuvaavat esimerkiksi Hand et al. [2001].

3. Tiedonlouhinnasta ja luokittelemisesta

Tässä tutkimuksessa käsiteltävien etäisyysfunktioiden keskeisen sovelluskentän muodostavat *tiedonlouhinnan* eri ongelmat. Tiedonlouhinnan tavoitteena on hyödyllisen tiedon seulominen suurista datajoukoista [Hand et al., 2001]. Sen tutkimus on lisääntynyt viime aikoina johtuen digitaalisesti talletetun datan määrän kasvusta. Suuri osa talletetusta datasta on joko hyödytöntä tai sen valtava määrä estää minkään hyödyllisen informaation saamisen esiin siitä. Tiedonlouhinnan menetelmien avulla datajoukoista etsitään tietojen välisiä suhteita ja pyritään koostamaan hyödyllinen informaatio ymmärrettävään muotoon. Tiedonlouhinnan menetelmiä käytetään hyödyksi esimerkiksi lääketieteellisissä asiantuntijajärjestelmissä [Auramo, 1993; Laurikkala, 2001].

3.1. Koneoppiminen

Erilaiset tiedonlouhintatehtävät ovat usein palautettavissa *luokitteluongelmiksi*, joissa annetut esimerkit on sijoitettava äärelliseen määrään toisensa poissulkevia luokkia [Quinlan, 1986]. Luokitteluongelmat ratkaistaan tyypillisesti *koneoppimismenetelmillä*. Koneoppimistehtävän ensimmäisenä vaiheena on tavallisesti hahmojen ja luokkien etsintä datajoukosta eli *opetusjoukosta*, ja toisena vaiheena uusien tapausten eli *testijoukon* luokittelu kerätyn tietämyksen

avulla. Luokittelu (koneoppiminen) voi olla joko *ohjattua* tai *ohjaamatonta*. Ohjatussa luokittelussa luokat tiedetään etukäteen ja uudet tapaukset tulee leimata etäisyysinformaation perusteella johonkin luokkaan. Ohjaamattomassa luokittelussa ei luokkia tiedetä etukäteen ja tehtävänä onkin etäisyyksien perusteella ryhmitellä tapaukset.

Usein luokittelussa olennainen on etäisyyden käsite, josta lisää kappaleessa 4. Tulee määrätä jokin etäisyysfunktio, joka virittää avaruuden, jossa oliolla on aina jokin etäisyys toisesta oliosta. Luokittelu suoritetaan näin ollen etäisyysinformaation perusteella. Etäisyyteen perustuvia luokittelumenetelmiä ovat esimerkiksi *k-lähimmän naapurin menetelmä* (*k-nearest neighbor method*, KNN), joka on esimerkki ohjatusta menetelmästä, sekä *klusterointi*, joka on esimerkki ohjaamattomasta menetelmästä. Tässä tutkimuksessa (luku 6) käytetään lähimmän naapurin menetelmää, johon tutustutaan seuraavassa luvussa. Klusterianalyysiä ei käsitellä tässä tutkielmassa. Sitä ovat käsitelleet runsaasti esimerkiksi Jain ja Dubes [1988], Jain et al. [1999] ja Everitt et al. [2001].

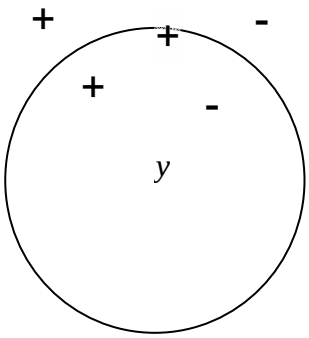
3.2. K-lähimmän naapurin menetelmästä

Koneoppimisessa KNN-menetelmällä lähtökohta on, että opetusjoukko T on esitetty kuvan 2.1 mukaisena havaintomatriisina. Lisäksi oletetaan esimerkki y, joka on tarkoitus luokitella opetusjoukosta saatavan tietämyksen perusteella. KNN-menetelmää sanotaan *havaintoperustaiseksi* oppimismenetelmäksi [Mitchell, 1997]. Olennaista seikka on luonnollisesti opetusjoukon esimerkkien ja esimerkin y väliset etäisyydet ja käytettävä etäisyysfunktio. KNN-menetelmässä etsitään opetusjoukosta T esimerkin y k ($1 \le k \le |T|$) lähintä naapuria. Olkoon näin saatu joukko S. Lasketaan joukosta S luokkafrekvenssit. Olkoon yleisimpien luokkien (eli luokkien, joiden frekvenssi on maksimaalinen joukossa S) esimerkeistä muodostettava joukko S'. Jos S' koostuu yhden luokan esimerkeistä, valitaan tämä luokka. Jos joukossa S' on usean luokan edustajia, voidaan käyttää yhden lähimmän naapurin menetelmää ainoastaan joukossa S'. Jos tämäkään ei määrää yksikäsitteistä luokkaa, voidaan käyttää joko satunnaisvalintaa tai jotain determinististä valitsemistapaa (esimerkiksi valitaan käsittelyjärjestyksessä ensimmäisen esimerkin luokka).

Erikoistapaus on tilanne, jolloin k lähimmän naapurin etäisyys on sama kuin k+1 lähimmän naapurin etäisyys. Tilanne voidaan jättää huomioimatta, mikäli se on harvinainen. Voidaan kuitenkin myös valita joukkoon S kaikki sellaiset k+n lähimmät naapurit ($n \ge 1$), siten, että etäisyys k lähimmästä naa-

purista on sama kuin *k*+*n* lähimmästä naapurista. Tämän jälkeen toimitaan kuten edellä.

Tarkastellaan kuvaa 3.1. Siinä kuvataan tilannetta, jossa olio *y* luokitellaan 3-lähimmän naapurin menetelmällä joko positiiviseksi tai negatiiviseksi. Hieman samankaltaisen esimerkin esittää Mitchell [1997]. Havaitaan, että kolmen lähimmän naapurin joukossa on kaksi positiivista ja yksi negatiivinen. Näin *y* luokittuu positiiviseksi.



Kuva 3.1. 3-lähimmän naapurin luokittelutilanne.

3.3. Koneoppimisen tulosten arviointi

Koneoppimisen tuloksia voidaan arvioida esimerkiksi luokittelemalla opetusaineisto uudelleen. Tällöin on kuitenkin vaarana, että *ylioppimisen* seurauksena saadaan harhaisia tuloksia. Kun luokittelutietämys sisältää liikaa opetusjoukolle tyypillisiä, mutta perusjoukossa harvinaisia piirteitä, ovat pelkästään opetusjoukosta hankitut tulokset liian hyviä. Tästä syystä arviointiin käytetään erillistä testijoukkoa eli opetusjoukossa käyttämätöntä dataa. *Ristiinvalidointi* on hyödyllinen menetelmä tähän tarkoitukseen. Siinä datajoukko on jaettu erillisiin osiin ja vuorotellen yksi näistä toimii testijoukkona ja loput opetusjoukkona. Termi *n-kertainen ristiinvalidointi* tarkoittaa, että data jaetaan *n* erilliseen osaan ristiinvalidointia varten.

Tunnusluku *tarkkuus* (accuracy) ilmaisee, kuinka hyvin luokittelu onnistui. Olkoon TP_i onnistuneiden luokitteluiden lukumäärä luokassa c_i (i = 1,..,*C*, missä *C* on luokkien lukumäärä) ja *N* kaikkien esimerkkien lukumäärä. Tarkkuus määritellään

$$accuracy = \frac{\sum_{i=1}^{C} TP_i}{N} \cdot 100\% . \qquad (3.1)$$

4. Etäisyys, samankaltaisuus, metrisyys ja ultrametrisyys

Tässä luvussa määritellään etäisyyden, metrisyyden ja metriikan käsitteet. Pääosa luvusta, mukaan lukien määritelmät perustuu Bobergin väitöskirjaan [1999].

Vertailtaessa olioita käytetään yleensä termejä *lähekkäisyys* (proximity), *samankaltaisuus* (similarity), *erilaisuus* (dissimilarity) ja *etäisyys* (distance). Lähekkäisyys on yleistermi, joka voi tarkoittaa sekä samankaltaisuutta että erilaisuutta. Erilaisuus ja etäisyys tarkoittavat intuitiivisesti samaa. Käytettäköön vastedes termiä etäisyys erilaisuuden asemesta. Käsitteellä samankaltaisuus on päinvastainen merkitys etäisyyteen verrattuna. Mitä enemmän oliot muistuttavat toisiaan sitä suurempi niiden välinen samankaltaisuus on ja sitä pienempi on niiden välinen etäisyys.

Olkoon *x* ja *y* olioita avaruudessa *E*.

Määritelmä 4.1. Oletetaan, että d_0 on reaaliluku. Funktio $d : E \times E \to \boldsymbol{R}$ on etäisyysfunktio, jos

1. $d(x, y) \geq d_0 \qquad \forall x, y \in E$,
2. $d(x, x) = d_0 \qquad \forall x \in E$, ja
3. $d(x, y) = d(y, x) \quad \forall x, y \in E$.

Määritelmä 4.2. Oletetaan, että s_0 on reaaliluku. Funktio $s : E \times E \to \boldsymbol{R}$ on samankaltaisuusfunktio, jos

1. $s(x, y) \leq s_0 \qquad \forall x, y \in E$,
2. $s(x, x) = s_0 \qquad \forall x \in E$, ja
3. $s(x, y) = s(y, x) \quad \forall x, y \in E$.

Näin ollen etäisyysfunktiolla on alaraja d_0 (usein 0) ja samankaltaisuusfunktiolla yläraja s_0 (usein 1), jotka kuvaavat olioiden samantasoista vastaavuutta. Ehdot 2 ja 3 tunnetaan refleksisyys- ja symmetrisyysehtoina.

Määritelmä 4.3. Jos etäisyysfunktio $d : E \times E \to \boldsymbol{R}$ täyttää kolmioepäyhtälön ehdon

$$d(x, y) + d(y, z) \geq d(x, z) \qquad \forall x, y, z \in E,$$

niin *d*:n sanotaan olevan *pseudometriikka E*:ssä, ja (E, d):n sanotaan olevan *pseudometrinen avaruus.*

Edelleen, jos

$$d(x, y) = d_0 \Leftrightarrow x = y \tag{4.1},$$

niin *d*:tä sanotaan *metriikaksi E*:ssä. Jos etäisyysfunktio ei toteuta kolmioepäyhtälöä, sanotaan etäisyysfunktion olevan ei-metrinen.

Jos kolmioepäyhtälö ei päde, on *E*:ssä sellaiset oliot *x*, *y* ja *z*, että $d(x, y) > d(x, z) + d(z, y)$. Tämä tarkoittaa, että etäisyysarvo $d(x, y)$ ei ole lyhin

etäisyys olioiden *x* ja *y* välillä, vaan on olemassa ”väliolio” *z*, joka ”lyhentää” etäisyyttä olioiden *x* ja *y* välillä. Tason geometriaa terminologiaa käyttäen, lyhin polku pisteiden *x* ja *y* välillä olisi lyhyempi kuin viivan pituus pisteestä *x* pisteeseen *y*, mikä on mahdotonta euklidisessa avaruudessa.

Määritelmä 4.4. Jos etäisyysfunktio $d : E \times E \to \boldsymbol{R}$ täyttää ehdon (4.1) ja *ultrametrisen epäyhtälön*

$$\max\{d(x,y), d(y,z)\} \geq d(x,z) \qquad \forall x, y, z \in E,$$

niin *d*:n sanotaan olevan *ultrametriikka E*:ssä, ja (E,d):n sanotaan olevan ultrametrinen avaruus. Huomioitava on, että ultrametrisessä epäyhtälössä etäisyysarvoista kahdella tulee olla sama arvo. Helposti on nähtävissä, että ultrametrinen epäyhtälö toteuttaa myös kolmioepäyhtälön ehdot, joten ultrametrinen avaruus on aina myös metrinen avaruus. Olioavaruus on harvoin ultrametrinen. Esimerkiksi taso $\boldsymbol{R}^2$ varustettuna tavallisella euklidisella etäisyydellä on metrinen, mutta ei ultrametrinen.

Aiemmin mainittiin, että voidaan käyttää samankaltaisuusfunktiota etäisyysfunktion asemesta. Kolmioepäyhtälö samankaltaisuusfunktiolle *s* on

$$|s(x,y) + s(y,z)| \cdot s(x,z) \geq s(x,y)s(y,z) \quad \forall x, y, z \in E.$$

Samankaltaisuusfunktio, joka toteuttaa kolmioepäyhtälön on metriikka. Ultrametrinen epäyhtälö samankaltaisuusfunktiolle *s* on

$$\min\{s(x,y,), s(y,z)\} \leq s(x,z) \qquad \forall x, y, z \in E.$$

Etäisyyden ja samankaltaisuuden välillä on monia hyödyllisiä suhteita. On helppo vaihtaa samankaltaisuudesta etäisyyteen ja päinvastoin, minkä seuraava lause osoittaa. Käytetään notaatiota, jossa esimerkiksi $s_0 - s$ tarkoittaa samaa kuin $s_0 - s(x,y)$.

Lause 4.1. (*Samankaltaisuus vs. etäisyys*)

1. Funktio *s* on metrinen samankaltaisuusfunktio, jos ja vain jos funktio $s_0 - s$ on metrinen etäisyysfunktio.
2. Jos *d* on metrinen etäisyysfunktio, jolla on äärellinen yläraja $D = \max\{d(x,y) | x, y \in E\}$, niin *D-d* on metrinen samankaltaisuusfunktio.
3. Oletetaan, että $d(x,y) > 0$ kaikille esimerkeille *x* ja *y*. Tällöin *d* on metrinen etäisyysfunktio, jos ja vain jos $1/d$ on metrinen samankaltaisuusfunktio.

Lauseen 4.1 perusteella samankaltaisuuden ja etäisyyden käsitteet ovat yhtä eksakteja. Tässä tutkielmassa käytetään poikkeuksetta käsitettä etäisyys samankaltaisuuden asemesta.

5. Joitakin metrisiä etäisyysfunktioita

Tässä luvussa käsitellään metriikoita erilaisissa olioavaruuksissa. Aluksi tarkastellaan *Minkowski*-metriikkaa ja sen joitain variantteja, jotka soveltuvat lähinnä vain kvantitatiivisia attribuutteja sisältäviin aineistoihin. Sitten tarkastellaan VDM-funktiota, joka soveltuu lähinnä nominaalisiin aineistoihin. Lopuksi käsitellään eräitä heterogeenisiä funktioita, jotka yhdistelevät edellä mainittujen ominaisuuksia.

5.1. Etäisyysfunktioita homogeeniselle datalle

Homogeeninen data on sellaista dataa, jossa attribuuttien mitta-asteikot ovat tyypiltään samanlaisia. Seuraavassa esitetään funktioita, jotka olettavat datan olevan homogeenistä.

5.1.1. Minkowski-metriikka ja sen erikoistapaukset

Kuten luvussa 2 määriteltiin, olioavaruus E voidaan esittää $n \times m$-matriisin avulla $E = (e_{ij})$, missä $i = 1,2,...,n$ ja $j = 1,2,...,m$. Seuraavassa oletetaan, että $x_j, y_j \in \boldsymbol{R}$.

Oletetaan esimerkit x ja y, joiden arvoihin viitataan alaindeksillä $j = 1,2,...,m$. Kaikkien tunnetuin metriikka lienee *euklidinen* etäisyys d_2, joka määritellään seuraavasti:

$$d_2(x,y) = \sqrt{\sum_{j=1}^{m} (x_j - y_j)^2} \,. \tag{5.1}$$

Euklidinen metriikka on erikoistapaus *Minkowski-metriikasta*, joka määritellään

$$d_p(x,y) = \left(\sum_{j=1}^{m} |x_j - y_j|^p \right)^{\frac{1}{p}}. \qquad (p \geq 1) \tag{5.2}$$

Minkowski-metriikan erikoistapauksilla on mielenkiintoinen suhde: jos $p \geq q$, niin $d_p(x,y) \leq d_q(x,y)$ kaikille arvoille x ja y.

Yleisimmät Minkowski-etäisyyden erikoistapaukset ovat metriikat, jotka saadaan, kun $p = 1$, $p = 2$ ja $p = \infty$. Kun $p = 1$, saadaan

$$d_1(x,y) = \sum_{j=1}^{m} |x_j - y_j|, \tag{5.3}$$

jota tunnetaan nimillä *Manhattan-*, *City block-* ja *Taxicab*-etäisyys. Kun $p = \infty$, saadaan maksimietäisyys

$$d_\infty(x,y) = \max_{j=1,...,m} \{ |x_j - y_j| \}. \tag{5.4}$$

Minkowski-metriikat eivät ole invariantteja attribuuttien arvojen skaalaamiselle. Tämä tarkoittaa, että siirryttäessä yhdestä mittayksiköstä toiseen muuttuvat olioiden välisten etäisyyksien suhteet. Esimerkiksi, jos attribuutin *x* mittayksikkö muutetaan metristä senttimetriin, kasvavat kaikkien olioavaruuden *E* olioiden väliset etäisyydet, ja lisäksi attribuutin *x* painoarvo suhteessa muihin attribuutteihin kasvaa. Luvussa 2 tarkasteltiin datan muunnosmenetelmä, jota voidaan käyttää etäisyyslaskennan yhteydessä. Myöhemmin käsitellään heterogeenisiä etäisyysfunktioita, ja tämän ohessa tarkastellaan erilaisia normalisointimenetelmiä tarkemmin.

5.1.2. VDM

Nominaalisia attribuutteja sisältävälle datalle on esitetty *VDM-funktio* (*Value Difference Metric*). [Stanfill and Waltzin, 1986]. VDM-funktiossa otetaan Minkowski-metriikohin nähden varsin poikkeava tapa laskea etäisyys nominaalisille attribuuteille. Siinä etäisyyden laskeminen nominaalisille attribuuteille perustuu ehdollisiin todennäköisyyksiin ja aineiston luokkainformaatioon. Se poikkeaa aiemmista siinä, että se käyttää luokkainformaatiota hyväkseen etäisyyslaskennassa. Tästä seuraa, että VDM ei ole käyttökelpoinen muunlaiselle aineistolle, kuin luokkatiedot sisältävälle. Näin ollen käytännössä VDM soveltuu ainoastaan luokitteluongelmiin.

Riisuttu versio VDM-metriikasta määritellään seuraavasti:

$$vdm_a\left(x_j, y_j\right) = \sum_{c=1}^{C}\left|\frac{N_{a,x_j,c}}{N_{a,x_j}} - \frac{N_{a,y_j,c}}{N_{a,y_j}}\right|^q = \sum_{c=1}^{C}\left|P_{a,x_j,c} - P_{a,y_j,c}\right|^q, \tag{5.5}$$

missä

- N_{a,x_j} on niiden esimerkkien lukumäärä opetusjoukossa *T*, joilla on arvo x_j attribuutille *a*;
- $N_{a,x_j,c}$ on niiden esimerkkien lukumäärä opetusjoukossa *T*, joilla on arvo x_j attribuutille *a* ja, jotka kuuluvat luokkaan *c*;
- *C* on luokkien lukumäärä joukossa ongelma-alueella;
- *q* on vakio, yleensä 1 tai 2; ja
- $P_{a,x_j,c}$ on ehdollinen todennäköisyys, että luokka on *c*, olettaen, että attribuutin *a* arvo on x_j. Kuten jo nähtiin kaavasta 5.5, $P_{a,x_j,c}$ määritellään seuraavasti:

$$P_{a,x_j,c} = \frac{N_{a,x_j,c}}{N_{a,x_j}}, \tag{5.6}$$

missä N_{a,x_j} on summa $N_{a,x_j,c}$ yli kaikkien luokkien, eli:

$$N_{a,x_j} = \sum_{c=1}^{C} N_{a,x_j,c} \tag{5.7}$$

ja summa $P_{a,x_j,c}$ yli kaikkien luokkien on 1 määrätyille arvoille *a* ja x_j.

Kun käytetään etäisyysfunktiota $vdm_a(x_j,y_j)$, kahden arvon oletetaan olevan sitä lähempänä (ts. sitä pienemmällä etäisyydellä) toisistaan mitä enemmän samanlaisia luokitteluja kyseisillä arvoilla on, riippumatta siitä missä järjestyksessä arvot on annettu. Näin ollen attribuutin arvot voidaan koodata uudestaan millä tahansa tavalla, eikä etäisyysarvo silti muutu.

Tarkastellaan esimerkkinä attribuuttia *väri* ja sen arvoja *punainen*, *vihreä* ja *sininen*. Sovelluksen tehtävänä on identifioida olio omenaksi. Tällöin *punaisen* ja *vihreän* katsotaan olevan lähempänä kuin *sininen*, koska ne selvästi korreloivat luokan *omena* kanssa.

Edellä kuvattu VDM-algoritmi on riisuttu versio alkuperäisestä algoritmista [Stanfill and Waltz, 1986], jossa huomioidaan myös attribuuttien suhteellinen paino. HVDM-algoritmi, joka esitellään myöhemmin ei sisällä myöskään attribuuttipainoja. Sovelluskohtaisesti voidaan kuitenkin useimmissa tapauksissa lisätä toteutettavaan algoritmiin attribuuttipainot [Wilson and Martinez, 1997].

Ongelmaksi VDM-metriikassa muodostuvat tilanteet, jossa syöte-esimerkissä on opetusjoukossa esiintymätön arvo. Tällöin $N_{a,x_j,c}$ tulee olemaan 0 kaikille luokille *c* ja tästä seuraa, että myös N_{a,x_j} on 0, eikä arvoa $P_{a,x_j,c}$ ole määritelty. Nominaalisille attribuuteille on mahdotonta määrittää todennäköisyyttä tässä tilanteessa, joten todennäköisyydeksi tulee valita jokin kiinteä arvo. Perusteltuja vaihtoehtoja täksi arvoksi ovat joko 0 tai $1/C$ [Wilson and Martinez, 1997].

Jos VDM-metriikkaa käytetään laskemaan etäisyyttä kvantitatiivisille attribuuteille, on todennäköistä, että päädytään edellä kuvattuun tilanteeseen. Kun kvantitatiivisen attribuutin arvoalue on laaja, niin on todennäköistä, että kaikki sen arvot opetusjoukossa ovat uniikkeja. Tällöin arvon x_j etäisyys mistä tahansa muusta arvosta on 1 ja etäisyys itsestään on 0. Näistä syystä VDM-metriikka on sopimaton käsittelemään kvantitatiivisia ja varsinkin jatkuvia attribuutteja.

Eräs ratkaisu tähän ongelmaan on *diskretisoida* kvantitatiiviset attribuutit. Diskretisoinnissa määritellään kvantitatiiviselle attribuutille diskreetit arvoalueet ja näin saadaan muunnettua kvantitatiivisesta attribuutista nominaalinen. Jos esimerkiksi tiedetään, että diskreetti kvantitatiivinen attribuutti *a* voi saada arvoja alueelta [0,999], voidaan muodostaa neljä

ryhmää A([0,249]), B([250,499]), C([500,749]) ja D([750,999]). Ryhmän nimen jälkeen suluissa on se arvoalue, johon kyseiseen ryhmään kuuluva arvo sijoittuu (esimerkiksi kuuluakseen ryhmään A tulee arvon x_j täyttää ehto $0 \le x_j \le 249$). Tämän pohjalta voidaan määritellä uusi nominaalinen attribuutti *a*', jonka arvot saadaan koodaamalla ne edellä mainitulla tavalla attribuutin *a* arvoista. Diskretisoinnissa menetetään kuitenkin arvokasta tietoa, joka on kvantitatiiviselle attribuutille ominaista. Jotta jokaisen attribuuttityypin ominaisuudet tulisivat hyödynnetyksi, Wilson ja Martinez [1997] ovat kehittäneet joukon metriikoita heterogeeniselle datalle, joiden perustana on VDM, ja joissa kvantitatiivisten attribuuttien käsittelyyn on kiinnitetty aiempaa enemmän huomiota.

5.2. Etäisyysfunktioita heterogeeniselle datalle

Luvussa 2 esiteltiin tilastotieteissä käytettävä attribuuttien kaksijako nominaalisiin ja kvantitatiivisiin attribuutteihin. Kuten on tullut jo ilmi etäisyysfunktion valinta vaikuttaa olennaisesti luokittelutulokseen. Etäisyysfunktioita voidaan tarkastella sen mukaan, otetaanko niissä huomioon attribuuttien mitta-asteikot. Homogeeninen etäisyysfunktio siis olettaa, että mitta-asteikko on sama jokaisella attribuutilla, kun taas heterogeeninen etäisyysfunktio olettaa datan olevan mitta-asteikoiltaan vaihtelevaa. Ongelmallinen on myös tilanne, jossa jokaisella attribuutilla ei ole samaa painoarvoa ajateltaessa informaatiota, jonka se antaa olioiden erilaisuudesta. Tässä tutkielmassa kuitenkin oletetaan, että attribuuttien painoarvoissa ei ole eroja.

Seuraavassa esitellään heterogeeniselle datalle soveltuvat etäisyysfunktiot *HEOM* (Heterogeneous Euclidean-Overlap Metric) ja *HVDM* (Heterogeneous Value Difference Metric). Funktioiden nimille ei ole vakiintuneita suomennoksia, joten käytetään ainoastaan lyhenteitä. Kumpikin näistä etäisyyysfunktiosta on metriikka [Juhola and Laurikkala, 2001]. Luvussa 6 verrataan käytännön luokittelutilanteissa tässä luvussa esiteltäviä etäisyysfunktioita ja edellisessä luvussa käsiteltyä euklidista etäisyysfunktiota.

5.2.1. HEOM

Ahan HEOM-funktion [Wilson and Martinez, 1997] ideana on käyttää esimerkin kullekin attribuutille etäisyysfunktiota sen mukaan, mikä on attribuutin mitta-asteikko. Tältä pohjalta HEOM määrittelee etäisyyden attribuutin *a* arvojen x_j ja y_j välillä seuraavasti:

$$d_a(x_j, y_j) = \begin{cases} 1, & \text{jos } x \text{ tai } y \text{ tuntematon,} \\ overlap(x_j, y_j), & \text{jos } a \text{ on nominaalinen,} \\ rn\_diff_a(x_j, y_j) & \text{muutoin.} \end{cases} \quad (5.8)$$

Tuntemattomat attribuutin arvot käsitellään palauttaen etäisyysarvo 1, joka on useimmiten myös maksimietäisyys, jonka funktio voi palauttaa. Funktiot *overlap* ja *rn_diff* määritellään seuraavasti:

$$overlap(x_j, y_j) = \begin{cases} 0, \text{ jos } x_j = y_j \\ 1, \text{ muutoin} \end{cases}, \quad (5.9)$$

$$rn\_diff_a(x_j, y_j) = \frac{|x_j - y_j|}{range_a}. \quad (5.10)$$

Attribuutin vaihteluväli $range_a$ määritellään

$$range_a = \max_a - \min_a,$$

jossa min_a ja max_a ovat minimi- ja maksimiarvot attribuutille *a* opetusjoukossa. Voidaan puhua myös attribuutin *a alueen* (domain) koosta samassa merkityksessä.

Käyttäen yllä olevaa määritelmää etäisyysarvolle d_a saadaan tyypillisesti arvo väliltä [0,1], riippumatta siitä, onko attribuutti nominaalinen vai kvantitatiivinen. Tämä edellyttää, että mikäli attribuutti *a* on kvantitatiivinen, ei esimerkin arvo sijaitse lasketun arvoalueen ulkopuolelle. Luvussa 6 käytetään luokittelussa ristiinvalidointia (ks. luku 3), jota käytettäessä on edellä kuvattu tilanne mahdollinen ja näin ollen on myös mahdollista, että $d_a > 1$. Wilsonin ja Martinezin [1997] mukaan tämä ei kuitenkaan ole iso ongelma, sillä ensinnäkin kyseiset tilanteet ovat harvinaisia ja toiseksi tällaisissa tilanteissa voi olla myös hyväksyttävää, että etäisyysarvo on poikkeuksellisen suuri. Kokonaisetäisyyden esimerkeille *x* ja *y* on *HEOM*(*x*,*y*):

$$HEOM(x, y) = \sqrt{\sum_{a,j=1}^{m} d_a(x_j, y_j)^2}. \quad (5.11)$$

Tämä etäisyysfunktio poistaa nominaalisten attribuuttien arvojen mielivaltaisen järjestyksen vaikutukset, mutta toisaalta sen tapa käsitellä nominaalisia attribuutteja on vielä varsin yksinkertainen. Se jättää huomioimatta nominaalisten attribuuttien ominaisuuksia, joista saattaa olla hyötyä luokittelussa [Wilson and Martinez, 1997]. Seuraavaksi esiteltävä HVDM-funktio käsittelee nominaaliset attribuutit huomattavasti monipuolisemmin. Huomattava on, että mikäli aineistossa on pelkkiä kvantitatiivisia attribuutteja, redu-

soituu HEOM tavalliseksi euklidiseksi etäisyysfunktioksi, joka on standardoidu attribuutin arvoalueella.

5.2.2. HVDM

Edellä havaittiin, että euklidinen etäisyys on sopimaton nominaalisten attribuuttien käsittelyyn ja toisaalta VDM ei erityisen hyvin sovellu kvantitatiivisten attribuuttien käsittelyyn. Ongelman ratkaisemiseksi Wilson ja Martinez [1997] ovat kehittäneet HVDM-metriikan (Heterogeneous Value Difference Metric), joka yhdistää euklidisen metriikan ja VDM-metriikan ideat heterogeeniseksi etäisyysfunktioksi. HVDM määritellään seuraavasti:

$$HVDM(x,y) = \sqrt{\sum_{a,j=1}^{m} d_a^2\left(x_j, y_j\right)}, \tag{5.12}$$

missä *m* on attribuuttien lukumäärä. Funktio $d_a\left(x_j, y_j\right)$ palauttaa etäisyyden arvojen x_j ja y_j välillä attribuutin *a* kohdalla, ja määritellään

$$d_a\left(x_j, y_j\right) = \begin{cases} 1, & \text{jos } x_j \text{ tai } y_j \text{ on tuntematon,} \\ normalized\_vdm_a, & \text{jos } a \text{ on nominaalinen,} \\ normalized\_diff_a, & \text{jos } a \text{ on kvantitatiivinen.} \end{cases} \tag{5.13}$$

Funktio $d_a(x,y)$ käyttää seuraavaa funktiota silloin, kun kyseessä on kvantitatiivinen attribuutti:

$$normalized\_diff_a\left(x_j, y_j\right) = \frac{\left|x_j - y_j\right|}{4\sigma_a}, \tag{5.14}$$

missä σ on attribuutin *a* arvojen keskihajonta.

Kuten kohdassa 2.3 sekä HEOM-metriikkaa tarkasteltaessa huomattiin on attribuuttien normalisointi tarpeellista, jotta jokaisella attribuutilla olisi sama vaikutus etäisyyttä laskettaessa. HEOM-metriikassa kvantitatiiviset attribuutit normalisoitiin attribuutin vaihteluvälillä. Ongelmia aiheutuu, kun syöteesimerkissä on arvo, joka ei esiinny opetusjoukossa. Tällöin etäisyysarvo voi karata suuremmaksi kuin 1. Toinen ongelma vaihteluvälillä normalisoitaessa on mahdollinen *poikkeava arvo* (outlier) opetusjoukossa. Esimerkiksi, jos attribuutti saa arvoja suurimmaksi osaksi alueelta [1,100], mutta joukkoon sisältyy yksi poikkeava arvo 1000, niin HEOM-etäisyydeksi tulee lähes aina arvo, joka on alle 0,1. Tästä syystä robustimpi normalisoimismetodi on käyttää keskihajontaa [Wilson and Martinez, 1997]. Normaalisti jakautuneessa datassa 95 % prosenttia arvoista sijoittuvat keskiarvon ympärille kahden keskihajonnan väliin. Näin ollen HVDM-metriikassa lineaariset attribuutit normalisoidaan neljällä keskihajonnalla.

Wilson ja Martinez [1997] pohtivat kolmea vaihtoehtoa funktioksi *normalized_vdm*$_a$. Määritellään funktiot N1, N2 ja N3 seuraavasti:

$$\text{N1: } normalized\_vdm1_a(x_j, y_j) = \sum_{c=1}^{C} \left| \frac{N_{a,x_j,c}}{N_{a,x_j}} - \frac{N_{a,y_j,c}}{N_{a,y_j}} \right|, \tag{5.15}$$

$$\text{N2: } normalized\_vdm2_a(x_j, y_j) = \sqrt{\sum_{c=1}^{C} \left| \frac{N_{a,x_j,c}}{N_{a,x_j}} - \frac{N_{a,y_j,c}}{N_{a,y_j}} \right|^2} \text{ ja} \tag{5.16}$$

$$\text{N3: } normalized\_vdm3_a(x_j, y_j) = \sqrt{C \cdot \sum_{c=1}^{C} \left| \frac{N_{a,x_j,c}}{N_{a,x_j}} - \frac{N_{a,y_j,c}}{N_{a,y_j}} \right|^2}\ . \tag{5.17}$$

Funktio N1 on kaava 5.5, jossa $q = 1$. Funktio N2 on sama kaava, jossa $q = 2$ eli yksittäisten attribuuttien eroavuudet neliöidään. Funktion N1 ja N2 ero on vastaavanlainen kuin ero, joka on Manhattan- ja euklidisen metriikan välillä. Syynä tähän on, että funktion N2 tuottamien etäisyyksien jakauma on muodoltaan sellainen, että hyvin suuria tai vastaavasti hyvin pieniä etäisyyksiä tulee vähemmän suhteessa keskisuuriin etäisyyksiin. Funktiolla N1 ei tällaista ominaisuutta ole. Funktio N3 lisää funktioon N2 vielä luokkien määrällä kertomisen. Wilson ja Martinez [1997] toteavat funktion N2 olevan hypoteettisesti robustimpi kuin N1 tai N3. Seuraavassa kappaleessa esitetyissä luokittelutehtävissä käytetään funktiota N2. Juuren ottaminen jätetään kuitenkin pois, koska juuri otetaan jo funktiossa *HVDM*(*x*,*y*).

6. Etäisyysfunktiot käytännössä

Tutkielman tarkoituksena oli, paitsi pohtia etäisyysfunktioiden ominaisuuksia, niin myös käytännössä vertailla funktioita erilaisissa koneoppimistilanteissa. Tavoitteena oli testata, miten eri etäisyysfunktiot luokittelevat erityyppisiä heterogeenisiä aineistoja. Varsinaisena luokittelijana käytettiin yhden lähimmän naapurin menetelmää ja sen komponenttina kolmea etäisyysfunktiota: euklidista, HEOM- sekä HVDM-funktiota. Testejä varten kirjoitettiin Java-kielinen ohjelma. Oppimistulosten vertailemiseen käytettiin ristiinvalidointia ja luokittelutarkkuutta. Seuraavassa on kuvaukset testeissä mukana olleista datajoukoista sekä yhteenveto tuloksista. Aluksi esitetään joitakin tarkennuksia käytetyistä menetelmistä, ristiinvalidoinnista, lähimmän naapurin menetelmästä sekä etäisyysfunktioista.

Testeissä käytettiin 10-kertaista ristiinvalidointia. Koko datajoukko jaettiin siis kymmeneen lähes yhtä suureen osajoukkoon C_1, C_2 ,..., C_{10}. Harva datajoukko oli jaollinen kymmenellä, joten käytännössä jakojäännös sijoitettiin tasaisesti osajoukkoihin alkaen joukosta C_1. Näin ollen osajoukkojen koot poikkesivat toisistaan maksimissaan yhdellä, mikä on suhteellisesti pieni hajonta, koska osajoukot olivat useamman kymmenen kokoisia.

Testeissä käytettiin yhden lähimmän naapurin menetelmää (katso luku 3). Tasapelitilanteissa käytettiin kohdassa 3.2 kuvattua heuristiikkaa, ja ratkaisemattomissa tilanteissa valittiin käsittelyjärjestyksessä ensimmäisen olion luokka.

Testeissä käytettiin euklidista, HEOM- ja HVDM-funktiota. Euklidinen etäisyys otettiin mukaan testeihin, jotta nähtäisiin miten funktio, joka ei ota huomioon attribuuttien mitta-asteikoiden luonnetta, selviytyy heterogeenisen aineiston luokittelusta. On kiinnostavaa tutkia, eroavatko tulokset heterogeenisillä funktioilla saaduista. Euklidiseen etäisyysfunktioon lisättiin standardointi, joka suoritettiin HVDM-funktion tapaan (luku 5) neljällä keskihajonnalla [Wilson and Martinez, 1997]. Euklidiseen etäisyysfunktioon sisällytettiin myös puuttuvien tietojen käsittely. Mikäli toinen tai molemmat attribuutti-arvot puuttuivat, tulkittiin etäisyyden arvoksi 1. Tosin puuttuvia tietoja ei käytetyissä datajoukoissa ollut.

HEOM-funktio toteutettiin kuten se on esitetty alakohdassa 5.2.1. HVDM-funktion toteutuksessa käytettiin kaavaa 5.16. Keskihajonta, vaihteluväli ja nominaalisten attribuuttien ehdolliset todennäköisyydet laskettiin aina kustakin opetusjoukosta.

6.1. Tuloksista

Taulukossa 6.1 on annettu testeissä käytetyistä viidestä aineistosta aineiston koko sekä attribuuttien mitta-asteikot. Huomataan, että aineistoista kolme ovat heterogeenisiä. Lasiaineistoissa on ainoastaan kvantitatiivisia ja munkit 3 -aineistossa pelkästään nominaalisia attribuutteja.

Aineisto	Tapauksia	Attribuutteja	
		Nominaalisia	Kvantitatiivisia
Huimaus	914	13	27
Inkontinenssi	529	9	7
Lippu	194	20	10
Lasi	214	0	11
Munkit 3	432	8	0

Taulukko 6.1. Yhteenveto aineistoista.

Etäisyysfunktioita vertailtiin ristiinvalidoinnista saatavaa tarkkuutta tutkimalla. Taulukossa 6.2 on yhteenveto saaduista tuloksista. Parhaat tarkkuudet on lihavoitu. Seuraavassa käydään tulokset jokaisen aineiston kohdalta läpi ja pohditaan syitä, miksi taulukon 6.2 kaltaisia tuloksia on saatu.

Aineisto	Euklidinen	HEOM	HVDM
Huimaus	71,55	75,38	**76,7**
Inkontinenssi	84,69	**86,01**	83,93
Lippu	50	48,97	**55,67**
Lasi	**72,43**	70,56	**72,43**
Munkit 3	68,29	99,07	**100**

Taulukko 6.2. Yhteenveto tuloksista.

6.1.1. Huimausaineisto

Huimausaineisto on peräisin ONE-asiantuntijajärjestelmän tietokannasta. Huimauksen kaltaisia oireita voivat aiheuttaa monet sairaudet. ONE-järjestelmä on suunniteltu tukemaan diagnoosin tekemistä huimaustapauksissa [Auramo et al., 1993]. Alkuperäinen ONE-tietokanta on kuvattu Kentalan artikkelissa [1996]. Alkuperäisessä tietokannassa oli 564 potilastapausta, joihin on vähitellen lisätty 350 tapausta. Huimaus-aineisto jakaantuu yhdeksään luokkaan eli diagnoosiin. Lisäksi on mukana luokka, joka ilmaisee, että kyseiselle tapaukselle ei löydetty diagnoosia. Luokkafrekvenssit vaihtelevat varsin paljon, mikä vaikeuttaa koneoppimismenetelmillä suoritettavaa diagnosointia. Alkuperäisestä aineistosta puuttui jonkun verran tietoja. Tätä tutkimusta varten aineisto imputoitiin. Puuttuvat tiedot täytettiin mitta-asteikkojen mukaan moodeilla (nominaaliset attribuutit) ja mediaaneilla (kvantitatiiviset attribuutit).

Taulukosta 6.1 nähdään, että aineistossa on jotakuinkin puolet attribuuteista nominaalisia ja puolet kvantitatiivisia. Käytettyjen etäisyysfunktioiden ominaisuudet tuntien varsin ilmeinen hypoteesi on, että HEOM- ja HVDM-funktioilla saadaan parempia oppimistuloksia kuin euklidisella funktiolla. Taulukko 6.2 osoittaa, että hypoteesi pitää paikkansa myös käytännössä. Ero ei ole kuitenkaan kovin selvä heterogeenisten funktioiden eduksi. Tälle voi olla monia syitä. Myös nominaalisiksi luokitellut attribuutit saattavat sisältää sittenkin jonkinlaista järjestysinformaatiota, jolloin euklidisen etäisyyden laskentametodi ei aiheuta kovinkaan merkittävää haittaa oppimistulokselle. Sekoittamalla nominaalisten attribuuttien koodauksia saattaisi ero heterogeenisten funktioiden hyväksi olla selvempi. Toinen syy euklidisen etäisyyden suhteellisen hyvälle tarkkuudelle voi olla nominaalisten attribuuttien melko pienet arvoalueet, jolloin mahdollisia etäisyysarvojakaan ei ole monta näille attribuuteille, minkä voi arvioida vähentävän nominaalisten attribuuttien merkitystä luokittelussa. HVDM- ja HEOM-funktiot antoivat käytännössä saman tuloksen.

6.1.2. Inkontinenssiaineisto

Inkontinenssiaineisto [Laurikkala et al., 2001] on peräisin IES1-asiantuntijajärjestelmästä [Laurikkala, 2001]. Järjestelmän tarkoitus on auttaa virtsainkontinenssiin eli tahattomaan virtsankarkailuun liittyvien tautien diagnosoinnissa. Huimausaineiston tapaan inkontinenssidata sisälsi jonkin verran puuttuvia arvoja, jotka on imputoitu maksimiuskottavuusmenetelmän avulla. Potilastapauksia on yhteensä 529. Attribuutit ovat jakautuneet suunnilleen tasan nominaalisten ja kvantitatiivisten kesken.

Taulukosta 6.2 havaitaan, että homogeenisen ja heterogeenisten funktioiden luokittelutarkkuus ei eroa merkittävästi. Itse asiassa Euklidinen funktio luokittelee jopa vajaan prosenttiyksikön verran tarkemmin kuin HVDM, vaikka kyseessä on varsin heterogeeninen aineisto. Parhaiten luokittelusta suoriutuu HEOM. Tuloksista paljastuu, että HVDM-metriikan monisävyinen tapa rakentaa tietämystä nominaalisista attribuuteista ei ole aina parempi tapa kuin yksinkertainen joko-tai -tapa, jota HEOM käyttää, tai edes parempi kuin euklidisen funktion tapa käsitellä nominaalisia attribuutteja kvantitatiivisten attribuuttien tapaan. Funktion valinta tulisikin tehdä aineiston perusteella.

6.1.3. Lippuaineisto

Lippuaineisto on saatu UCI-tietokannasta [Blake and Merz, 1998], joka on koneoppimisen tutkimiseen tarkoitettu tietokanta. Kaksi kolmasosaa attribuu-

teista on nominaalisia ja tapauksia on 194 kappaletta. Lippuaineisto poikkeaa paljon edellisistä huimaus- ja inkontinenssiaineistosta siinä, että luokitteluongelma on siinä varsin keinotekoinen. Liput luokitellaan niiden oikean alakulman kuvion mukaan. Diagnoosidataan verrattuna yhtä selkeää korrelaatiota lipun luokan ja sen muiden ominaisuuksien välillä ei näytä olevan, koska luokittelutulokset ovat varsin kehnoja. HVDM osoittautuu tässä tilanteessa selvästi parhaaksi luokittelijaksi, mikä johtunee nominaalisten attribuuttien suuresta lukumäärästä. Tällä aineistolla euklidinen funktio osoittautuu paremmaksi kuin HEOM. Mielenkiintoista on huomata, että homogeeninen funktio osoittautuu jo toisessa aineistossa heterogeenistä funktiota paremmaksi. Tämä vahvistaa johtopäätöstä, että etäisyysfunktio tulee valita käsiteltävän aineiston mukaan, koska heterogeeninen funktio ei ole välttämättä parempi kuin homogeeninen. On kuitenkin huomattava, että Wilson ja Martinez [1997] osoittavat laajemman aineistojoukon kanssa, että heterogeeniseen dataan soveltuu keskimäärin parhaiten heterogeeninen funktio.

6.1.4. Lasiaineisto

Lasiaineisto on lippuaineiston tapaan peräisin UCI-tietokannasta. Muista testiaineistoista poiketen lasiaineisto ei sisällä lainkaan nominaalisia attribuutteja. Muistetaan, että HVDM-funktio redusoituu euklidiseksi, kun aineistossa on pelkkiä kvantitatiivisia attribuutteja. Tulos on näin ollen sama näillä kahdella funktiolla. HEOM-funktio antaa hieman kahta edellistä huonomman tuloksen. HEOM-funktio käyttää standardointiin attribuutin vaihteluväliä, kun taas muut kokeiltavat funktiot käyttävät neljää keskihajontaa. Luvussa 5 käsiteltiin tällaisen normalisoimismenetelmän ongelmia ja nuo mainitut ongelmat näyttävät vaikuttavan käytännön luokittelutilanteessakin.

6.1.5. Munkit 3 -aineisto

Munkit 3 -aineisto on niin ikään otettu UCI-tietokannasta. Kyseistä aineistoa on käytetty testiaineistona useissa koneoppimismenetelmiä tutkineissa projekteissa. Munkit 3 on keinotekoinen aineisto, johon on lisätty viisi prosenttia ylimääräistä kohinaa eli luokittelua tahallisesti harhauttavia attribuutteja. Kaikki attribuutit ovat nominaalisia ja tapauksia on 432. Tuloksista taulukossa 6.2 havaitaan, että euklidinen funktio suoriutuu tästä aineistosta selvästi muita funktioita huonommin. Sen tarkkuus 68 prosenttia on yli 30 prosenttiyksikköä huonompi kuin sekä HEOM- että HVDM-funktioilla saadut tulokset. Tämä testi siis vahvistaa arviota euklidisen funktion sopimattomuudesta kaikkiin nominaalisiin aineistoihin. HVDM-funktiolla saatu tulos on tasan 100 prosenttia eli se luokittelee aineiston täydellisesti. Tämä tarkoittaa,

että aineistossa on selkeästi jomman kumman luokan kanssa korreloivia attribuuttien arvoja.

7. Yhteenveto

Tutkielmassa tutustuttiin etäisyysfunktioihin, jotka ovat tärkeä komponentti useissa koneoppimissovelluksissa. Ensin määriteltiin reaalimaailman datan esitystapa. Sitten tutustuttiin luokitteluun ja lähimmän naapurin menetelmään, jotka ovat tärkeimmät etäisyysfunktioiden sovellusalueet. Lisäksi perehdyttiin joihinkin etäisyysfunktioihin ja erityisesti pohdittiin niiden soveltuvuutta heterogeenisiin aineistoihin. Tämän jälkeen suoritettiin käytännön luokittelukokeita viidelle eri aineistolle tätä tehtävää varten kirjoitetun ohjelman avulla. Kokeissa käytettiin ristiinvalidointia ja k-lähimmän naapurin menetelmää. Lähimmän naapurin menetelmän komponenttina toimi vuorollaan euklidinen, HEOM- ja HVDM-funktio.

Tulokset kertoivat sen, että mitään täsmällistä ohjetta etäisyysfunktion valintaan ei voida antaa. Etäisyysfunktion valinta tuleekin riippua aina luokiteltavasta aineistosta. Aineiston ominaisuuksia tutkimalla voidaan arvioida, mitä etäisyysfunktiota tulisi käyttää. Pohdittava on ainakin, käyttääkö funktiota, joka käsittelee nominaaliset attribuutit eri tavalla kuin kvantitatiiviset. Näiden tulosten perusteella nominaalisia attribuutteja sisältävään aineistoon kannattaa valita attribuuttien nominaalisuuden huomioiva funktio, vaikkakin tähänkin on olemassa poikkeuksia. Lisäksi on pohdittava, mitä standardointimenetelmää käyttää kvantitatiivisille attribuuteille. Näiden tulosten perusteella keskihajonnan käyttö vaihteluvälin asemesta on suositeltavaa. Näiden tulosten perusteella ei voi kuitenkaan tehdä ehdottomia johtopäätöksiä, koska testien määrä oli siksi vähäinen. Jatkotutkimuksessa testien lukumäärää on tarkoitus kasvattaa ja käsitellä huomattavasti suurempia aineistoja, jotka ovat tyypillisiä tiedonlouhinnan kohteita.

Tutkimus on tarkoitus laajentaa pro gradu -tutkielman laajuiseksi. Jatkotutkimuksessa testattavaksi otetaan lisää etäisyysfunktioita ja aineistoja. Ideana on myös pohtia mahdollisuuksia tehdä joitakin muutoksia etäisyysfunktioihin luokittelutulosten parantamiseksi.

Viiteluettelo

[Auramo et al., 1993] Yrjö Auramo, Martti Juhola and Ilmari Pyykkö, An expert system for the computer-aided diagnosis of dizziness and vertigo. *Medical Informatics*, **18** (1993), 293-305.

[Blake and Merz, 1998] C.L. Blake and C.J. Merz, *UCI Repository of Machine Learning Databases*, University of California, Irvine, Department of Information and Computer Science, 1998. Available as http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Boberg, 1999] Jorma Boberg, *Cluster Analysis – A Mathematical Approach with Applications to Protein Structures*. Ph.D. Thesis, University of Turku, 1999.

[Everitt et al., 2001] Brian S. Everitt, Sabine Landau, Morven Leese, *Cluster Analysis*. Arnold, London, 2001.

[Gower, 1985] John C. Gower, Measures of similarity, dissimilarity, and distance. In: S. Kotz, N.L. Johson and C.B Read (eds.), *Encyclopedia of Statistical Sciences* **5** (1985), Wiley, New York, 397-405.

[Hand et al., 2001] David Hand, Heikki Mannila and Padhraic Smyth, *Principles of Data Mining*. MIT Press, 2001.

[Jain et al., 1999] Anil K. Jain, M.N. Murty and P.J. Flynn, Data clustering: A review. *ACM Computing Surveys* **31**, 3 (Sep. 1999), 264-323.

[Jain and Dubes, 1988] Anil K. Jain and Richard C. Dubes, *Algorithms for Clustering Data*. Prentice-Hall, New Jersey, 1988.

[Juhola and Laurikkala, 2001] Martti Juhola and Jorma Laurikkala, On metricity of heterogeneous Euclidean-overlap metric and heterogeneous value difference metric with missing values. Submitted for publication to *Pattern Recognition*.

[Kentala, 1996] Erna Kentala, Characteristics of six otologic diseases involving vertigo, *The American Journal of Otology* **17** (1996), 883-892.

[Laurikkala et al., 2001] Jorma Laurikkala, Martti Juhola, Seppo Lammi, Jorma Penttinen ja Pauliina Aukee, Analysis of the imputed female urinary incontinence data for the evaluation of expert system parameters. *Computers in Biology and Medicine* **31** (2001) 239-257.

[Laurikkala, 2001] Jorma Laurikkala, *Knowledge Discovery for Female Urinary Incontinence Expert System*. Ph.D. Thesis, Dept. of Computer – and Information Sciences, University of Tampere, 2001.

[Mitchell, 1997] Tom M. Mitchell, *Machine Learning*. McGraw-Hill, New York, 1997.

[Puntanen, 1998] Simo Puntanen, *Matriiseja tilastotieteilijälle*. Matemaattisten tieteiden laitos, Tampereen yliopisto, 1998.

[Quinlan, 1986] J. R. Quinlan, Induction of decision trees. *Machine Learning* **1** (1986), 81-106.

[Stanfill and Waltz, 1986] Craig Stanfill and David Waltz, Toward memory-based reasoning. *Communications of the ACM* **29,** 12 (Dec. 1986), 1213-1228.

[Wilson and Martinez, 1997] D. Randall Wilson, Tony R. Martinez, Improved heterogeneous distance functions. Journal of Artificial intelligence research 6, 1 (1997), 1-3

DES-algoritmin turvallisuudesta

Jussi Palola

Tiivistelmä.
Tutkielmassa tarkastellaan DES-salausalgorithmin elinkaarta. Tarkastelussa ovat raa'an voiman murtoon (brute force attack), kryptoanalyyseihin perustuvat murtomenelmät ja erilaiset algoritmin turvallisuutta lisäävät parannukset.

Avainsanat ja -sanonnat: Data Encryption Standard, DES, kryptografia, kryptoanalyysi, salakirjoitus, tietoturva.
CR-luokat: E.3

1. Johdanto

Ihminen on pyrkinyt salaamaan tietoa aina muinaisen Egyptin ajoista lähtien [Kahn, 1967]. Ensimmäiset salausalgoritmit olivat hyvin yksinkertaisia, mutta nykyiset algoritmit ovat jo matemaattisesti hyvin monimutkaisia. Kuten niin monella muullakin tieteenalalla myös kryptografiassa ensimmäiset sovellukset olivat sotilaskäytössä. Laajempaan tietoisuuteen kryptografia tuli vasta toisen maailmansodan jälkeen, kun tietoliikenteen määrä alkoi kasvaa ja tiedon salaaminen tuli tarpeelliseksi jokapäiväisessä elämässä.

Salauksessa viesti, jota kutsutaan ilmitekstiksi (plain text), salakirjoitetaan (encrypt) ensin salakirjoitukseksi (cipher text), jonka jälkeen se toimitetaan vastaanottajalle. Vastaanottaja avaa (decrypt) salakirjoituksen ja saa näin haltuunsa alkuperäisen ilmitekstin. Tavoitteena on siis salata tieto siten, että ulkopuolinen ei saa ilmitekstiä haltuunsa eli siis kykene murtamaan salausta. [Schneier, 1996]

Tutkielmassa käsiteltävä DES (Data Encryption Standard)-algoritmi [NBS, 1977] on ehkä tunnetuin salausalgoritmi maailmassa. Sitä on käytetty yleisesti 1970-luvulta lähtien ja se on myös yksi tutkituimmista salausalgoritmeista. DES:in vaiheet aina sen suunnittelusta nykypäivään ovat mielenkiintoisia ja niihin tässä tutkielmassa pyritään lukijaa tutustuttamaan.

DES-algoritmi perustuu symmetriseen lohkosalaukseen. Symmetrisessa salauksessa teksti salakirjoitetaan ja avataan samalla avaimella (private key). Lohkosalauksessa (block cipher) salattava tieto jaetaan tietyn mittaisiin lohkoihin, joista jokaiseen käytetään samaa salausalgoritmia ja avainta.

Ensimmäisessä luvussa käydään läpi DES-algoritmin toimintaperiaate eli kuinka itse algoritmi toimii, mikäli se on lukijalle entuudestaan tuttu voi lukija huoletta siirtyä toiseen lukuun. Toisessa luvussa pureudutaan DES-algoritmin syntyhistoriaan. Käsitellään miksi se luotiin ja minkälaisia vaatimuksia sille oli asetettu sekä selostetaan algoritmin ominaisuuksia. Kolmannessa kappaleessa käsitellään DES:in murtamista ja siitä tehtyjä kryptoanalyysejä. DES-algoritmia ja sen ominaisuuksia on tutkittu monesta eri näkökulmasta usean vuoden ajan. Se, mitä on saatu selville ja miten DES:iä voidaan yrittää murtaa, selviää kyseisessä kappaleessa. Samalla kun algoritmia on tutkittu ja löydetty siitä sekä vahvuuksia, että heikkouksia on tullut esille myös muutoksia joilla algoritmista voidaan tehdä turvallisempi ja hankalampi murtaa. DES:in modifikaatioita ja vaihtoehtoisia versioita käsitellään neljännessä luvussa.

2. Data Encyption Standard (DES)

Salakirjoitukseen tarvitaan DES:ssä salakirjoitettava ilmiteksti (vähintään 64 bittiä) sekä tasan 64 bittiä pitkä avain K. DES on lohkosalain: se salakirjoittaa tietoa 64 bitin lohkoissa. Algoritmille syötetään 64 bitin mittainen lohko ilmitekstiä ja se antaa tulokseksi 64 bitin mittaisen lohkon salakirjoitusta. DES on symmetrinen algoritmi. Samaa algoritmia ja avainta käytetään sekä viestin salakirjoitukseen että sen avaamiseen. DES algoritmi koostuu kierroksista. Jokainen salakirjoitettava lohko käy läpi 16 iteraatio kierrosta. Algoritmi käyttää vain tavanomaisia aritmeettisiä ja loogisia operaatioita ja enintään 64 bitin mittaisilla luvuilla.

Aluksi avain *K* (64 bittiä) permutoidaan *PC-1* -permutaatiota käyttäen. Permutaatioissa bitit vaihtavat paikkaa Taulukon 1 mukaiseen järjestykseen. *PC-1* -permutaatiossa tulokseksi saadaan 56 bittiä pitkä avain, joka jaetaan kahtia. Alkupuoliskosta tulee *C0* ja loppuosasta *D0*. Eli:

PC-1(K) = C0D0.

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Taulukko 1. PC-1 -permutaatiotaulu.

Jokaiselle DES-kierrokselle *i* (yhteensä 16 kierrosta) luodaan oma aliavain (subkey) K_i. Aliavaimen määritys tapahtuu kahdessa vaiheessa. Ensin kummallekin edellisen kierroksen K_{i-1} alkuperäiselle avainlohkolle C_{i-1} ja D_{i-1} (ensimmäisellä kierroksella siis C_0 ja D_0) suoritetaan syklinen siirto vasemmalle (*LS*, leftshift). Siirron suuruus riippuu siitä, minkä kierroksen aliavainta ollaan luomassa. Siirtojen pituudet on esitetty Taulukossa 2. Syklisessä siirrossa siirretään 28-bitin lohkon ensimmäisiä bittejä lohkon viimeiseksi. Siirto-operaatio tehdään erikseen kummallekin lohkolle C_{i-1} ja D_{i-1}. Esimerkiksi ensimmäisen kierroksen aliavaimia luotaessa C_0-lohkon alusta (vasemmasta päästä) siirretään 1 bitti C_0-lohkon loppuun (oikeaan päähän). Siirtojen tuloksena on lohkot C_i ja D_i.

kierros	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
määrä	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Taulukko 2. Siirrettävien bittien määrä eri kierroksilla.

Bittien siirtojen jälkeen lohkot Ci ja Di yhdistetään ja permutoidaan PC-2 permutaation läpi. Tästä saadaan tulokseksi kierroksen i 48-bittinen aliavain Ki. Eli:

Ci = LSi(Ci-1,), Di = LSi(Di-1) ja Ki = PC-2(CiDi).

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Taulukko 3. PC-2 -permutaatiotaulu.

Salakirjoitettava 64 bitin lohko *x* käy läpi alkupermutaation (initial permutation, *IP*). Alkupermutaatiossa jokainen 64 bitistä vaihtaa paikkaa, 58. bitistä tulee ensimmäisestä bitti, 50. bitistä tulee toinen bitti ja niin edelleen. Alkupermutaation tuloksena saadaan bittijono x_0, jonka 32 ensimmäistä bittiä muodostavat L_0:n ja 32 viimeistä bittiä R_0:n (katso Taulukko 4). Eli:

$x_0 = IP(x) = L_0R_0$.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	53	35	27	21	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Taulukko 4. Alkupermutaatiossa käytetty IP -permutaatiotaulu.

Alkupermutaation jälkeen käydään läpi 16 iterointikierrosta, joissa salakirjoitus tapahtuu. Lähtökohtana jokaiselle kierrokselle on edellisen kierroksen kaksi lohkoa (L_{i-1} ja R_{i-1}) sekä kierroksen aliavain K_i. Jokaisen kierroksen tuloksena saadaan L_i ja R_i (jotka taas siirtyvät seuraavalle kierrokselle).

Edellisen kierroksen oikea lohko R_{i-1} on suoraan käsiteltävän kierroksen vasen lohko L_i. Oikean puoleisen lohkon R_i laskemiseksi on aloitettava funktiosta *f*. Funktiolla *f* on syötteenä edellisen kierroksen oikea lohko R_{i-1} (syöte *A*) ja käsiteltävän kierroksen aliavain K_i (syöte *J*), eli *f(A, J)*. Ensimmäiseksi funktiossa *f* laajennetaan 32-bittinen syöte *A* 48-bittiseksi laajennusfunktiolla *E*. *E(A)* sisältää 32 bittiä alkuperäisestä syötteestä ja puolet (16) biteistä toistetaan permutaation *E(A)* mukaan. Käytetty permutaatiotaulu on esitelty Taulukossa 5.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Taulukko 5. E(A) –permutaatiotaulu.

Tulos *E(A)* yhdistetään loogisella XOR-operaattorilla syötteen *J* kanssa. Tulokseksi saatu 48 bitin jono jaetaan kahdeksaan kuuden bitin lohkoon. $B_1 = b_1,b_2,b_3,b_4,b_5,b_6$, $B_2 = b_7,b_8,b_9,b_{10},b_{11},b_{12}$, $B_3 = \ldots$

Seuraava vaihe on substituutio. DES:ssä on määritelty kahdeksan S-laatikkoa (S-box, substition box). Jokainen S-box koostuu 4 rivistä ja 16 sarakkeesta. Niillä on 6 syöte (input) bittiä ja 4 tuloste (output) bittiä. S-box:lla käsitellään kaikki kahdeksan kuuden bitin lohkoa joko-tai operaation tuloksesta. $B_j=b_1,b_2,b_3,b_4,b_5,b_6$ lasketaan $S_i(B_i)=C_i$. Kaksi bittiä (bitit b_1 ja b_6) yhteenlaskettuna määrittävät S-box:n tulos rivin (0-3) ja neljä bittiä (bitit

b_2,b_3,b_4 ja b_5) yhteenlaskettuna määrittävät tulos sarakkeen. Kun substituutio on käyty läpi kaikkien kahdeksan S-box:in osalta saadaan tulokseksi 32 bitin (8 x 4 bittiä) jono *C*. S-box:t on annettu Taulukossa 6.

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	6	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Taulukko 6. S-box:t 1-8.

S-box substituutiosta saatava jono *C* permutoidaan *P*-permutaation (katso Taulukko 7) mukaan. Tuloksena saatava bittijono *P(C) = f(A,J)*.

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Taulukko 7. P-permutaatio.

Funktion *f* tulos yhdistetään XOR-operaattorilla edellisen kierroksen vasempaan puoliskoon Li_{-1}. Operaation tulos on uusi oikea puolisko R_i. Tämän jälkeen voidaan siirtyä seuraavaan kierrokseen. Kun kaikki 16 kierrosta on käyty läpi, suoritetaan viimeinen permutaatio (IP^{-1}, final permutation) joka on käänteinen aluksi suoritetulle alustavalle permutaatiolle. Viimeisen permutaation taulu on esitetty Taulukossa 8.

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Taulukko 8. Viimeinen permutaatio (IP^{-1}).

Salakirjoituksen avaaminen tapahtuu samalla algoritmilla kuin itse salakirjoitus, salattavan ilmitekstin sijaan algoritmissa käsitellään jo salakirjoitettu lohko. Ainut ero algoritmissa salakirjoitusta avattaessa on, että kierrosten aliavaimia K_i käytetään käänteisessä järjestyksessä K_{16}, K_{15}, K_{14}, ..., K_1.

3. DES:in ominaisuuksista

DES:in etsintä alkoi vuonna 1973 kun Yhdysvaltojen National Bureau of Standars (NBS) julkisti pyynnön löytää salakirjoitusstandardi. Suunnittelukriteereinä standardille olivat mm. turvallinen, helposti ymmärrettävä, avaimeen perustuva salakirjoitus joka helposti siirrettävissä elektronisiin laitteisiin. Kun pyyntö vuonna 1974 uusittiin sai NBS lupaavan ehdokkaan IBM:ltä. IBM oli edelleen kehittänyt vanhaa Lucifer-salakirjoitintaan vuosina

1973-1974. Vuonna 1977 Data Encryption Standard (DES) sai nimensä ja valittiin Yhdysvalloissa viralliseksi salakirjoitusstandardiksi.

Alustava (*IP*) tai lopullinen permutaatio (IP^{-1}) eivät vaikuta DES:in turvallisuuteen mitenkään, mutta ne ovat osa DES-standardia. Luultavimmin nämä kaksi permutaatiota lisättiin algoritmiin, jotta ilmiteksti tai salakirjoitus olisi helpompi ladata erilliseen DES-siruun tietyn kokoisina lohkoina. Ohjelmistojen kannalta *IP* ja IP^{-1} ovat hankalia toteuttaa tehokkaasti ja osassa sovelluksista ne onkin jätetty pois. Tällöin ei ole kuitenkaan enää kyse varsinaisesta DES-salakirjoituksesta.

Avain syötetään algoritmille 64 bittisenä. Itse algoritmi käsittelee avaimesta vain 56 bittiä, sillä joka kahdeksatta bittiä käytetään pariteetin tarkistukseen. Käytännössä avain on siis 56-bittinen ja sillä saadaan siis 2^{56} (noin 10^{17}) erilaista mahdollista avainta.

E(A) permutaatiolla on algoritmissa kaksi tarkoitusta. Se laajentaa 32-bittisen jonon 48-bittiseksi XOR-operaatiota varten ja laajentaa syötettä, jotta se voitaisiin tiivistää uudelleen substituutio-vaiheessa. Tärkeintä *E(A)*-permutaatiossa on kuitenkin se, että se laajentaa yksittäisen bitin vaikutusta kahteen eri substituution myöhemmässä vaiheessa. Joten 16 kierroksen aika 'lumivyöry'-efekti saa aikaan sen, että jokainen salakirjoituksen bitti on riippuvainen jokaisesta selvätekstin bitistä ja jokaisesta avaimen bitistä.

S-box:it ovat tärkein osa DES:in turvallisuutta. Muut vaiheet algoritmissa ovat lineaarisia ja helposti analysoitavissa. S-box:it ovat suunniteltu tarkoituksella hankaloittamaan erilaisia hyökkäyksiä. Suunnittelukriteerit olivat alun perin salaisia, mutta yksi algoritmin suunnitelijoista paljasti ne perustellessaan DES:in turvallisuutta erilaisia hyökkäyksiä vastaan [Coppersmith, 1994]:

- Jokaisella S-box:illa on kuusi syötebittiä ja neljä tulostebittiä (suurin mahdollinen koko joka pystyttiin toteuttamaan 1974 vuoden siruissa)
- Yksikään tulostebitti ei saa olla liian lähellä mitään syötebittien lineaarista funktiota.
- Jokaisella S-box:in rivillä kukin 16 mahdollisesta tulosteesta esiintyy vain kerran.
- Jos kaksi S-box:in syötettä eriävät vain bitin verran, tulee niiden tulosteiden olla eriävät vähintään kahden bitin verran.
- Jos kaksi S-box:in syötettä eriävät vain kahdelta keskimmäiseltä bitiltään, tulee niiden tulosteiden olla eriävät vähintään kahdellä bitillä.

- Jos kaksi S-box:in syötettä eriävät kahdelta ensimmäiseltä bitiltään ja ovat yhteneviä kahdelta viimeiseltä bitiltä, tulee niiden tulosteiden olla eriävät.
- Jokaista ei-nollaa 6-bitin eroa syötteissä kohden ei voi esiintyä samaa eroa tulosteissa useamassa kuin 8:ssa 32:sta parista, joissa kyseinen ero esiintyy.
- Sama kuin edellä mutta ulottaen se kolmeen samalla kierroksella aktiiviseen S-box:iin.

Coppersmith [1994] paljasti myös P-permutaation suunnittelukriteerit:

- S-box:ien neljä tuloste bittiä kierroksella *i* vaihdetaan niin että kaksi niistä vaikuttaa S-box:ien keskimmäisiin bitteihin kierroksella *i+1* ja muut kaksi vaikuttavat pääty bitteihin (kahteen ensimmäiseen tai viimeiseen bittiin).
- Jokaisen S-box:in neljä tuloste bittiä vaikuttaa kuuteen eri S-box:iin seuraavalla kierroksella; mikään kahden bitin pari ei vaikuta samaan S-box:iin.
- Jos jonkun S-box:in tuloste bitti vaikuttaa toisen S-box:in keskimmäisiin bitteihin, jälkimmäisen S-box:in tuloste bitti ei voi vaikuttaa ensimmäisen S-box:in keskimmäisiin bitteihin.

4. Murtaminen

DES:in ollessa ensimmäinen laajasti käytetty ja turvalliseksi todettu salausalgoritmi, se on toiminut kryptologian kehittäjänä. Sitä on tutkittu laajasti ja siihen on etsitty erilaisia hyökkäyksiä, osa toimii vain teorian tasolla, mutta osa myös käytännössä.

4.1. Heikot avaimet

DES:in kaikista mahdollisista avaimista löytyy muutama, jotka aiheuttavat poikkeuksellista toimitaan algoritmissa [Davies, 1983]. Neljä avaimista on niin sanottua heikkoa avainta. Koska DES:in aliavaimet muodostetaan kahdesta lohkosta ja kumpaakin lohkoa käsitellään erilleen, aiheuttaa se ongelmia lohkoissa, joissa koko lohko muodostuu samasta arvosta (täynnä ykkösiä tai täynnä nollia). Näissä neljässä tapauksessa avain pysyy samana jokaisella salauskierroksella.

Heikkojen avaimien lisäksi DES:istä löytyy puoliheikkoja avaimia. Kysymys on avainpareista, jotka tuottavat samasta ilmitekstistä samanlaisen salakirjoituksen. Nämä kuusi avainparia tuottavat algoritmissa vain

kahdenlaisia aliavaimia. Lisäksi DES:istä on löydetty mahdollisesti heikkoja avaimia, jotka algoritmin sisällä tuottavat vain neljänlaisia aliavaimia.

DES:in mahdollisista avaimista nämä muodostavat hyvin pienen joukon ja kaiken lisäksi ne on helppo tarkistaa jo avainta valittaessa.

4.2. Raaka voima

Raa'alla voimalla (brute-force) tehdyt hyökkäykset ovat tähän mennessä olleet tehokkaimpia keinoja DES:in murtamiseen. Teoriatasolla on löydetty raakaa voimaa nopeampiakin ratkaisuja, mutta niissä on yleensä ollut jonkin muu puute, joka on estänyt käytännön toteutuksen, esimerkiksi suunnattomat muistivaatimukset. Raa'alla voimalla murtaminen tarkoittaa, että algoritmia puretaan normaalisti kokeilemalla läpi kaikki mahdolliset avaimet järjestyksessä, kunnes oikea avain löytyy.

Raa'an voiman hyökkäykset ovat yleensä tunnetun ilmitekstin hyökkäyksiä (known plaintext); ne siis edellytävät että hyökkääjä tuntee salatun tekstin tai osan siitä (yhden lohkon). Muussa tapauksessa hyökkääjä joutuisi analysoimaan avaamiaan salakirjoituksia, jotta löytäisi miljoonista teksteistä juuri sen alkuperäisen salakirjoitetun ilmitekstin. Tunnettu ilmiteksti DES:in kohdalla voi olla esimerkiksi 64 bittiä määrämuotoisen tiedoston otsikosta. Murtamalla yhden lohkon hyökkääjä voi murtaa myös kaikki muut samalla avaimella salatut muut lohkot, vaikka ei niiden ilmitekstiä tuntisikaan.

DES-algoritmin tapauksessa hyökkääjällä on siis testattavanaan 2^{56} erilaista avainta. Jo DES:in julkaisun aikoihin epäiltiin 56-bitin avaimen riittämättömyyttä raa'an voiman hyökkäyksiä vastaan. Jo vuonna 1981 esitettiin teorioita 50 miljoonaa dollaria maksavasta koneesta, joka voisi murtaa DES:in kahdessa päivässä [Diffie, 1981]. Vuonna 1998 EFF (Electronic Frontier Foundation), Cryptography Research ja AWT-mikropiirivalmistaja rakensivat yhdessä DES:in murtamiseen tarkoitetun DES Cracker-laitteen 250 000 dollarilla. DES Cracker pystyi ratkomaan 92,6 miljardia avainta sekunnissa, kaikkien mahdollisten avainten testaamiseen laitteella meni siis noin 216 tuntia [EFF, 1998]. Vuotta myöhemmin 1999 ja Distributed.Net yhdistivät DES Cracker-laitteen ja noin 100 000 PC:tä Internetissä pystyen ratkomaan 245 miljardia avainta sekunnissa eli kokeilemaan kaikki mahdolliset avaimet läpi 81 tunnissa [EFF, 1999]. Nämä kohtuullisen halvat ja julkiset ratkaisut DES:in murtamiseksi osoittavat, että 56-bitin avain voidaan murtaa raa'alla voimalla melko helposti ja ainakaan vanhentumattoman tiedon salakirjoitus on näin ollen altis kyseisille hyökkäyksille.

4.3. Differentiaalianalyysi

Israelilaiset matemaatikot Eli Biham ja Adi Shamir kehittivät uuden menetelmän kryptoanalyysille. Alkuperäinen differentiaalianalyysi vuonna soveltui vain 15 kierroksen DES:ille [Biham and Shamir, 1991], mutta paria vuotta myöhemmin he esittelivät kokonaisen differentiaalianalyysin täydelliselle 16 kierroksen DES:ille. Ensimmäistä kertaa oli löydetty menetelmä joka mahdollistaa raakaa voimaa nopeamman hyökkäyksen teoriassa [Biham and Shamir, 1993]. Menetelmän avulla oli mahdollista laatia valitun ilmitekstin hyökkäys vain 2^{37} DES-operaatiolla. Valitun ilmitekstin hyökkäyksessä (chosen plain-text attack) hyökkääjä valitsee tietyn kriteerin mukaiset ilmitekstit ja saa haltuunsa vastaavat salakirjoitukset ja näiden avulla voi purkaa aikaisemmin näkemättömän salakirjoituksen.

Differentiaalianalyysissa tarkastellaan ilmiteksti parin ja niitä vastaavien salakirjoitusten muuntumista algoritmin iteraatiokierroksilla, kun niitä salakirjoitetaan samalla avaimella. Kun pareja analysoidaan, eri avaimille annetaan todennäköisyyksiä, ja kun analysoidaan yhä enemmän ilmiteksti pareja, nousee yksi avain todennäköisimmäksi.

Hyökkäys on teoreettinen, sillä analysoitavan valitun ilmitekstin määrä on niin valtava, ettei sitä käytännössä voi analysoida. Hyökkäys voidaan muuntaa tavalliseksi tunnetun ilmitekstin hyökkäykseksi, mutta silloin se vaatii enemmän operaatioita kuin hyökkäys, joka käy läpi kaikki avaimet.

Vaikka differentiaalianalyysi tuli yleiseen tietoisuuteen vasta 1990-luvun alussa, väittävät DES:in suunnittelijat IBM:llä tienneen kyseisen tyyppisestä hyökkäyksen mahdollisuudesta jo 1970. Se näytteli suurta osaa S-boxien ja P-permutaation suunnittelussa, mutta he eivät halunneet paljastaa menelmää yleisesti koska se oli uhka monelle sen ajan salakirjoitusmenetelmälle [Coppersmith, 1994].

4.4. Lineaarianalyysi

Lineaarianalyysin kehitti Mitsuru Matsui vuonna 1993. Lineaarianalyysin menetelmä perustuu faktaan, että jotkut bitit painottuvat toisia bittejä enemmän DES-algoritmissa. Menetelmä vaatii kuitenkin suuren määrän tunnettua ilmitekstiä, jotta avain voitaisiin löytää [Matsui, 1994].

DES:in heikkous lineaarianalyysia vastaan on viidennessä S-box:ssa. Toinen syötebitti on yhtäsuuri neljän tulostebitin XOR-operaation kanssa vain 12 tapauksessa 64:stä, kun sen tulisi olla mahdollisimman lähellä 32/64:ää. Adi Shamir [1986] löysi tämän heikkouden ensimmäisenä, mutta ei keksinyt keinoa hyödyntää sitä.

Lineaarianalyysiin perustuvassa tunnetun ilmitekstin hyökkäyksessä hyödynnetään tätä heikkoutta. Vaatimuksena on 2^{47} tunnetua ilmitekstiä ja menetelmä on nopeampi kuin kaikkien avainten läpikäynti. Lineaarianalyysi on tällä hetkellä tehokkain DES:iä vastaan toteutettu hyökkäys, tosin tunnetun ilmitekstin vaatimus tekee siitä käytöntöön soveltumattoman.

5. TDEA ja uusi standardi

DES:in vanhentuessa siihen on kehitetty erilaisia modifikaatioita ja parannuksia, jotka tekisivät siitä turvallisemman. Standardin käyttöönotosta lähtien sitä on uudelleen arvioitu muutaman vuoden välein. Vuonna 1983 se läpäisi arvioinnin, mutta jo 1987 sen arveltiin käyvän vähitellen vanhaksi. Uutta standardia ei kuitenkaan ollut vielä kehitelty, joten DES:ille annettiin viisi vuotta lisäaikaa kunnes uusi standardi olisi valmis 1993. Vielä vuonna 1993 ei uutta standardia ollut näkyvissä ja DES sai jälleen viiden vuoden jatkoajan. Vuonna 1999 uusi standardi oli edelleen kateissa, nyt tosin kehitteillä, mutta silti DES sai jatkaa pienin parannuksin. Nyt mahdollistettiin TDEA (Triple Data Encryption Algorithm tai 3DES) alkuperäisen DES:in variaatio [NBS, 1999]. Viimein vuonna 2001 julkaistiin uusi salakirjoitus standardi AES (Advanced Encryption Standard) [NBS, 2001].

5.1. TDEA

TDEA kehitettiin vahvistamaan DES standardia, kun kokonaan uusi ja testattu standardi ei ollut vielä valmis. TDEA tunnetaan myös nimillä 3DES ja triple-DES. Se muodostuu moninkertaisesta DES-salauksesta, johon käytetään yhdestä kolmeen erilaista avainta. Standarin mukaan ensin ilmiteksti salakirjoitetaan avaimella K_3, tämän jälkeen salakirjoitus avataan avaimella K_2 ja lopuksi salakirjoitetaan se vielä avaimella K_1. Purkaminen tapahtuu käänteisessä järjestyksessä ja käänteisin operaatioin. Kaikki kolme avainta (K_1, K_2, K_3) voivat olla erillisiä (kolme avainta), samoja (yksi avain) tai K_1 ja K_3 voivat olla samoja (kaksi avainta).

Kaksinkertainen tai jopa kolminkertainen salaus olisi turha mikäli mille tahansa avainkolmikolle K_1, K_2, K_3 löytyisi aina K_4, joka toteuttaisi ehdon:

$$E_{K3}(E_{K2}(E_{K1}(P)))=E_{K4}(P)$$

Tämän on kuitenkin osoitettu olevan DES:in kohdalla mahdotonta [Campbell and Wiener, 1993].

Yhdellä avaimella toteutettuna TDEA vastaa täysin vanhaa DES standardia, kahdella avaimella toteutettuna salakirjoituksen purkamiseen tarvitaan 2^{112} DES-operaatiota ja kolmella 2^{168}. Moninkertaisesta salauksesta on

löydetty heikkouksia verrattuna pitemmän avaimen yksinkertaiseen salaukseen. Diffie ja Hellman esittelevät meet-in-the-middle-hyökkäyksen, jossa kahden k-bittisen avaimen kolminkertainen salakirjoitus voidaan purkaa 2^k operaatiolla ja 2^k muistitilalla. Meet-in-the-middle-hyökkäyksessä voidaan myös vaihtaa operaatioiden määrää muistiin ja päin vastoin kunhan niiden tulo pysyy samana (2^{112}) [Diffie and Hellman, 1977]. Lisäksi on todettu, että lisäämällä tunnetun ilmitekstin määrää voidaan muistin ja operaatioiden määrää vähentää entisestään murrettaessa kahden avaimen kolminkertaisia salakirjoituksia [van Oorschot and Wiener, 1991].

5.2. Advanced Encryption Standard (AES)

Vuonna 2001 päättyi viisivuotinen etsintä DES ja TDEA:n korvaajaksi [NBS, 2001]. NIST (National Institute of Standards and Technology) sai prosessin aikana kaiken kaikkiaan 15 ehdotusta uudeksi standardiksi. Uusi standardi perustuu belgialaisten Joan Daemen ja Vincent Rijmen Rijndael-algoritmiin. Algoritmissa on käytössä 128, 192 ja 256 bitin avaimenpituudet ja monimutkaisuudestaan huolimatta se vaatii toimiakseen vain 52 tavua muistia. Standardin on tarkoitus vähitellen syrjäyttää DES ja TDEA [Daemen and Rijmen, 1999].

6. Yhteenveto

Tietokone-lehden arvioon [Järvinen, 2002] ”Des ja 3des ovat erittäin turvallisia”, ei tule yhtyä, mutta lähes kolmekymmentävuotiaaksi salausalgoritmiksi DES on kestänyt yllättävän hyvin. Ilmeisiä heikkouksia siitä on löydetty vähän ja täydellinen avainten etsintä on edelleenkin tehokkain tapa murtaa DES-salakirjoitus. Kovin pitkäaikaisen tai arvokkaan tiedon salakirjoittamiseen tavallinen DES ei ole turvallinen, mutta lyhytikäisen tiedon salakirjoitukseen se yhä soveltuu ei vähiten nopeutensa ja keveytensä ansiosta. Nykyteknologialla ja riittävillä resursseillä DES voidaan kuitenkin murtaa tunneissa.

Kaksin- tai kolminkertainen DES-salakirjoitus tuntuu lähinnä tekohengitykseltä kuolevalle algoritmille. Näitäkään menetelmiä ei voi suositella arvokkaan tai pitkäaikaisen tiedon säilyttämiseen kun ottaen huomioon moninkertaisen salakirjoitusmenetelmien heikkoudet. Vaikkakin kaksin- tai kolminkertaisen salakirjoituksen murtamiseen tarvittava teho on eksponentiaalinen ei sen murtaminen silti ikuisuuksia kestä, tietokoneiden tehon ja muistikapasiteetin kasvaessa jatkuvasti.

Algoritmien turvallisuudesta puhuttaessa tulee ottaa huomioon myös muut tekijät. Suurin osa hyökkäyksistä perustuu oletukseen, että hyökkääjä tietää osan salakirjoitetusta ilmitekstistä tai että hyökkääjä on valinnut osan salakirjoitettavaa ilmitekstiä. Salaiseen avaimeen (private key) perustuvat algoritmit sisältävät myös aina inhimillisen tekijän avaimen leviämisestä. Salakirjoitus on turvassa korkeintaan niin kauan kuin sen salainen avainkin, oli algoritmi kuinka turvallinen tahansa.

Viiteluettelo

[Biham and Shamir, 1991] El Biham and Adi Shamir, Differential cryptanalysis of DES-like cryptosystems. In: *Advances in Cryptology – Crypto '90 Proceedings* (1991), Springer, 2-21.

[Biham and Shamir, 1993] El Biham and Adi Shamir, Differential cryptanalysis of the full 16-Round DES. In: *Advances in Cryptology – Crypto '92 Proceedings* (1993), Springer, 487-496.

[Campbell and Wiener, 1993] Keith W. Campbell and Michael J. Wiener, DES is not a group. In: *Advances in Cryptology – Crypto '92 Proceedings* (1993), Springer, 512-520.

[Coppersmith, 1994] Don Coppersmith, The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development* **38**, 3 (May 1994), 243-250.

[Daemen and Rijmen, 1999] Joan Daemen and Vincent Rijmen, The Rijndael Block Cipher. AES Proposal, 1999.

[Davies, 1983] Donald W. Davies, Some regular properties of the DES. In: *Advances in Cryptology: Proceedings of Crypto'82* (1983), Plenum Press, 89-96.

[Diffie, 1981] Whitfield Diffie, *Cryptographic Technology: Fifteen Year Forecast*. BNR Inc, 1981.

[Diffie and Hellman, 1977] Whitfield Diffie and Martin E. Hellman, Exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer* **10**, 6 (June 1977), 74-84.

[EFF, 1998] Electronic Frontier Foundation, *Cracking DES*. O'Reilly & Associates, 1998.

[EFF, 1999] Electronic Frontier Foundation, DES Challenge III broken in record 22 hours. RSA Data Security Conference Press Release, 1999.

[Järvinen, 2002] Petteri Järvinen, Salausohjelma turvaa kiintolevyn. *Tietokone*, 10 (Lokakuu 2002), 41-43.

[Kahn, 1967] David Kahn, *The Codebreakers*, Macmillan Publishing Company, 1967.

[Matsui, 1994] Mitsuru Matsui, Linear cryptanalysis method for DES Cipher. In: *Advances in Cryptology – Eurocrypt '93 Proceedings* (1994), Springer, 386-397.

[NBS, 1977] National Bureau of Standards, Data Encryption Standard (DES). FIPS pub. 46, U.S. Deparment of Commerce, 1977.

[NBS, 1999] National Bureau of Standards, Data Encryption Standard (DES). FIPS pub. 46-3, U.S. Deparment of Commerce, 1999.

[NBS, 2001] National Bureau of Standards, Advanced Encryption Standard (AES). FIPS pub. 197, U.S. Deparment of Commerce, 2001.

[van Oorschot and Wiener, 1991] Paul C. van Oorschot and Michael J. Wiener, A known-plaintext attack on two-key triple encryption. In: *Advances in Cryptology – Eurocrypt '90* (1991), Springer, 318-325.

[Schneier, 1996] Bruce Schneier, *Applied Cryptography Second Edition*, John Wiley & Sons Inc, 1996.

[Shamir, 1986] Adi Shamir, On the security of DES. In: *Advances in Cryptology – Crypto '85 Proceedings* (1986), Springer, 280-281.

Henkilökohtaiset agentit

Lauri Pekkala

Tiivistelmä.
Agenttitutkimus on suhteellisen uusi tietojenkäsittelytieteiden tutkimusalue. Sen vuoksi tutkimusalueella ei ole varsinaista standardin mukaista jakoa erityyppisiin agentteihin eikä varsinaista yleistä määritelmää agentin käsitteelle. Tässä tutkielmassa perehdytään henkilökohtaisiin agentteihin ja niiden sovellusalueisiin. Jako henkilökohtaisten ja muiden agenttien välillä perustuu tässä siihen, että henkilökohtaiset agentit ovat henkilökohtaistettu käyttäjänsä persoonan ja toimintatapojen mukaan. Tutkielman alussa perehdytään hieman ohjelmistoagenttien määritelmiin ja rajataan henkilökohtaisten agenttien käsitettä. Sitten esitellään erilaisia henkilökohtaisten agenttien sovelluksia, kerrotaan henkilökohtaisten agenttien toiminnasta tiimeissä, esitellään case-luontoisesti erästä agenttijärjestelmää, ja kerrotaan henkilökohtaistamisprosessista. Tutkielman tavoite on muodostaa käsitys siitä, mitä henkilökohtaiset agentit ovat ja mitä hyötyä niistä on.

Avainsanat ja -sanonnat: Agentit, älykkäät järjestelmät, henkilökohtaistaminen.
CR-luokat: I.2.11

1. Johdanto

Agenttitutkimuksen eräs tärkeä osa-alue on henkilökohtaiset agentit. Tutkimusalue on laaja, sillä se koostuu monista muista tietojenkäsittelytieteiden tutkimusalueista, joista ehkä tärkeimpänä voidaan pitää tekoälytutkimusta.

Henkilökohtaiset agentit ovat ohjelmia, jotka ovat tarkoitettu tarjoamaan henkilökohtaista apua erilaisissa tietokoneen suoritettavaksi sopivissa tehtävissä. Periaatteessa henkilökohtainen avustaja voitaisiin upottaa vaikkapa auton ajotietokoneen ohjelmistoon. Siinä tapauksessa agentti opettelisi käyttäjän ajotottumuksia ja kertoisi, paljonko polttoainetta kuluu tietyllä matkalla käyttäjän ajotyylillä. Sitten se laskisi polttoaineen hinnan kyseisellä matkalla kuljettajan yleensä käyttämän huoltoaseman hinnaston perusteella ja voisi jopa tehdä kaiken tämän jo tarvittaessa valmiiksi, jos se olisi oppinut, että kyseinen kuljettaja ajaa aina tiettynä päivänä paikasta A paikkaan B tiettyyn

kellonaikaan. Se ottaisi myös ruuhkat huomioon suunnitellessaan ajallisesti optimaalisinta ajoreittiä, varoittaisi mustasta jäästä ja hirvistä sekä tekisi vielä ehkä paljon muuta.

Edellä kuvatun kaltaisia ajoneuvoja on näihin päiviin saakka ollut lähes yksinomaan tv-sarjoissa ja elokuvissa. Mikään yllämainituista tehtävistä ei nykyään ole kuitenkaan mahdotonta toteuttaa. Oikeastaan vain mielikuvitus on rajana suunniteltaessa tietoteknisten ja erityisesti tekoälysovellusten mahdollisuuksia yhdistettynä moderniin teknologiaan. Henkilökohtaisten agenttien tutkimus on myös sidoksissa sulautettujen järjestelmien tutkimukseen, sillä käyttäjän ominaisuuksien mukaan muokattu agentti voidaan auton lisäksi upottaa puhelimeen, pesukoneeseen, kahvinkeittimeen tai vaikkapa mikroaaltouuniin.

Henkilökohtaisten agenttien suunnittelun yhteydessä esiintyy monimutkaisia kysymyksiä, kuten kuinka itsenäinen voi agentti toimissaan olla ja miten se oppii toimimaan oikein myös toimiessaan ilman käyttäjän valvontaa. Tässä onkin eräs tärkeä henkilökohtaisten agenttien suunnitteluun liittyvä tekoälyn tutkimusalue, koneoppiminen. Agentin on pystyttävä päättelemään, milloin se voi tehdä autonomisen päätöksen. Sen on ymmärrettävä viimeistään tekemistään virheistä, milloin olisi syytä konsultoida käyttäjää. Agentin on myös pystyttävä kommunikoimaan muiden agenttien kanssa. Tämä yhteistoiminnallisuus on myös haasteena kehitettäessä agenttiteknologiaan pohjautuvia ohjelmistoja.

Tämän tutkimuksen tavoitteena on luoda käsitys siitä, mitä henkilökohtaiset agentit ovat ja millaisia sovelluksia niistä on tehty.

2. Henkilökohtaiset agentit

Henkilökohtaisella agentilla tarkoitetaan yksinkertaistettuna jonkinlaista tietokoneohjelmaa, joka osaa suorittaa käyttäjänsä puolesta erilaisia tehtäviä. Niille on nimestäkin päätellen ominaista, että ne ovat henkilökohtaisia; ne osaavat toimia käyttäjäkohtaisesti.

2.1. Määritelmiä ja ominaisuuksia

Kaukonen et al. [1999] määrittelevät ohjelmistoagentin seuraavasti:

> ”Ohjelmistoagentti on tietokoneohjelmakokonaisuus, joka on toiminnassa jatkuvasti ja itsenäisesti tietyssä toimintaympäristössä. Samassa ympäristössä toimii muitakin agentteja ja prosesseja, jotka kommunikoivat keskenään ilman käyttäjän jatkuvaa ohjausta ja valvontaa.”

Tässä ohjelmistoagentin määritelmässä huomataan kaikkien agenttien ehkä olennaisin ominaisuus, itsenäisyys; käyttäjän ei tarvitse valvoa agentin tekemisiä. Samassa artikkelissa mainitaan myös muita ohjelmistoagentin ominaisuuksia. Näitä ovat agentin

1) tila

2) päämäärä

3) tietoisuus ympäristöstä ja muista läsnä olevista agenteista

4) yhteistyökyky muiden agenttien kanssa

5) oppimiskyky

6) liikkuvuus

7) kyky toimia käyttäjän edustajana erilaisissa transaktiotilanteissa.

Henkilökohtaisilla agenteilla on erityisesti oltava viimeksi mainittu ominaisuus; ne toimivat käyttäjänsä edustajina. Henkilökohtaisen agentin ominaisuuksin on kuitenkin hyvä kuulua koko edellä mainittu lista, sillä niiden on liikuttava tietoa haettaessa, opittava toimimaan käyttäjän mieltymysten mukaan, kommunikoitava muiden agenttien kanssa, sekä ohjelmointiteknisesti ajatellen niillä on oltava lista attribuuteista, joka kokonaisuutena kuvaa niiden tilaa.

Agtivityn [Krupansky, 2002] verkkosivuilta löytyvän määritelmän mukaan ohjelmistoagentti on tietokoneohjelma, joka suorittaa tehtäviä jonkun toisen osapuolen puolesta tietyssä ajassa ja ilman suoraa valvontaa tai kontrollia.

Majasen [2002] mukaan mobiili ohjelmistoagentti on ohjelma, joka toimii käyttäjän tai toisen ohjelman puolesta ja kykenee liikkumaan verkossa omin voimin. Agentti päättää itse missä ja milloin suorittaa toimenpiteitään, ja tarpeen tullen voi keskeyttää toimintansa ja jatkaa sitä toisaalla verkossa. Tutkielman neljännessä luvussa kerrotaan hieman tarkemmin mobiileista ohjelmistoagenteista.

Raisamon [2001] mukaan agenttikäsitettä on käytetty monenlaisissa yhteyksissä: agentti suorittaa tehtäviä annetussa ajassa, yhdistää useita informaatioresursseja, määrittää rajapintoja hajautetuille tekoälysovelluksille, toimii välittäjänä käyttöliittymissä, toimii älykkäänä apurina (tämän tutkielman aihe), on uskottava graafinen olio ja agentilla on kyky ymmärtää agenttien keskustelukieltä. Tältä pohjalta voidaan sanoa, että ainoa vedenpitävä määritelmä agentille on, että agentti on sitä, mitä se tekee.

2.2. Oppaana verkkopalvelussa

Käyttäjän tunteva agentti voidaan laittaa toimimaan käyttäjän ja verkkopalveluita tarjoavan yrityksen välissä opastaen käyttäjää verkkopalvelun käytössä. Näin voidaan tarjota parempaa palvelua, kun tarjolla on asiantunteva opas.

Tutkimukset ovat osoittaneet, että käyttäjät kohtelevat palvelun agenttia uutena ihmissuhteen kaltaisena suhteena, mikä antaa verkkokaupankäynnille uusia ulottuvuuksia. Eräs kukkakauppa esimerkiksi raportoi internetissä tapahtuvan myynnin nousseen kolminkertaiseksi siitä lähtien, kun heidän sivustoillaan alkoi palvella henkilökohtaisia agentteja, jotka opastivat käyttäjiä kukkien ja kukkakimppujen ostossa [Yong ja Kong, 2001].

Verkkosivuilla on ollut jo kauan erilaisia keskustelevia ohjelmia, mutta tavallisen keskusteluohjelman ja henkilökohtaisen agentin ero on nimenomaan siinä, että tavallinen keskusteluohjelma ei käytä hyväkseen tietoja käyttäjästä keskustelun aikana, ja näin ollen tällaista ihmissuhteen kaltaista suhdetta ei pääse muodostumaan.

2.3. Tukena organisaation toiminnassa

Monet organisaatioissa säännöllisesti suoritettavista tehtävistä sopisivat hyvin henkilökohtaisille agenteille, jotka voisivat vähentää huomattavasti työntekijöiden päivittäistä työmäärää. Tutkimusinstituutiolla tällaisia tehtäviä ovat muun muassa kokousten aikataulutus, aikataulujen uudelleenjärjestely, luennoitsijoiden valinta kokouksiin, dokumenttipohjien tekeminen ja apu ohjelmistokehityksessä.

Chaplusky et al. [2002] ovat kehittäneet organisaation toimintaa tukevaa agenttiteknologiaa, jonka avulla muun muassa edellä mainittuja tehtäviä voitaisiin suorittaa automaattisesti, tai ainakin osittain automaattisesti henkilökohtaisten agenttien avulla. Järjestelmän käyttäjällä on käytössään taskutietokone, joka sisältää henkilökohtaisen agenttiohjelman, joka puolestaan toimii liittymänä järjestelmän muihin palveluihin. Järjestelmän avulla saa muun muassa tietoa lähtevistä ja saapuvista lennoista. Järjestelmän käyttö edellyttää tietysti, että muillakin järjestelmän käyttäjillä on käytössään taskutietokone varustettuna henkilökohtaisella agentilla. Kolmannessa luvussa kerrotaan hieman tällaisesta agenttien tiimityöskentelystä.

2.4. Apuna matkan suunnittelussa

Henkilökohtainen agentti voi olla myös apuna matkaa varatessa, jolloin vuorovaikutus käyttäjän ja agentin välillä voisi toimia vaikkapa seuraavaan tapaan [Kaukonen et al. , 1999]:

1) Käyttäjä pyytää omaa henkilökohtaista agenttiaan etsimään ulkomaanmatkoja tarjoavan matkatoimiston.
2) Agentti toimittaa käyttäjälleen elektronisen esitteen saatavilla olevista kohteista.
3) Esite on muokattu käyttäjän profiilin ja tämän aikaisempien tilausten perusteella mahdollisimman hyvin tälle sopivaksi.
4) Kun käyttäjä on viimein saanut valittua haluamansa matkan, lähettää tämä pyynnön oman agenttinsa kautta matkatoimistolle, jotta häneen otettaisiin matkan tiimoilta yhteyttä mahdollisimman pikaisesti.
5) Matkasuunnitelma viimeistellään sitten henkilön ja matkatoimiston välillä.

Tässä esimerkissä havainnollistuu, kuinka palveluiden saatavuus paranee agenttiteknologian myötä ja kuinka hajautettua tietoa voidaan käyttää paremmin hyväksi palveluiden tarjonnassa. Käyttäjien kommunikoidessa keskenään omien agenttiensa kautta ei heidän tarvitse olla tietoisia vastapuolien fyysisestä sijainnista tai niiden käyttämistä päätelaitteista, sillä viestit voidaan muuntaa kunkin käyttäjän profiilin ja päätelaitteen mukaisesti sopivimpaan muotoon.

2.5. Tiedonhaun apuvälineenä

Suuresta määrästä tietoa on osattava hakea olennainen. Koska olennaistakin tietoa on usein liikaa, on hyvä tietää tarkalleen mitä haluaa. Usein parasta olisi tehdä intuitiivinen ja nopea päätös pohtimatta sen enempää. Vielä helpommalla pääsisi, jos tämän päätöksen tekisi joku toinen. Mitä suuremmaksi tietoavaruus kasvaa, sitä enemmän monet tiedonhakuun tarkoitetut apuvälineet menettävät merkitystään. Tiedonhakustrategioilla on sitä suurempi merkitys, mitä isommasta joukosta tietoa haetaan.

Suosittua Google-hakukonetta voitaisiin pitää eräänlaisena (tosin melko yksinkertaisena ja suoraviivaisena) henkilökohtaisen agentin ”esimuotona”, jonka ainoana tehtävänä on suorittaa hakuja, mutta jonka logiikka ei sisällä tekoälyä. Samoin on muiden internetin hakukoneiden laita. Niiden ongelmaksi on muodostunut, että ne usein palauttavat liian suuren tietomäärän. Käyttäjän tehtäväksi jää monesti kaivaa itse esille olennainen tieto. Tässä prosessissa auttavat tietysti hakijan hakutaidot, mutta peruskäyttäjä haluaisi mielellään mielekästä tietoa siedettävässä ajassa ja ennen kaikkea vähällä vaivalla. Mikä siis voisi auttaa tässä tiedonhakuprosessissa?

Sumi ja Mase [2002] kertovat artikkelissaan mielenkiinto-ohjatusta tiedonhausta. Se saattaisi olla yksi ratkaisu tiedonhakuongelmaan. Heidän

kehittämänsä semanttinen, käyttäjälle graafisesti esitettävä verkko voidaan muodostaa käyttäjän mielenkiintoalueiden mukaan. Näin ollen se ei helposti paisu liian suureksi ja siten säilyttää selkeytensä. Käyttäjän oma agentti taltioi näitä isäntänsä mielenkiinnon kohteita, ja semanttinen verkko voidaan muodostaa käyttäjälle sopivasti, kun tämä kirjautuu sisään järjestelmään. Tosin kaikesta huolimatta verkko paisui ajoittain liian suureksi. Tällaisissa tapauksissa käyttäjät turhautuivat ja lopettivat sen käytön. Sumin ja Masen järjestelmää on käytetty konferenssien yhteydessä tukemaan konferenssivieraiden yhteisiä mielenkiintoalueita ja siitä kerrotaan hieman kohdassa 2.6.

Chen ja Sycara [1998] ovat kehittäneet hakuagentin WWW:n selaamisen apuvälineeksi. Tarve tällaiselle agentille syntyi juuri siitä, että käyttäjien täytyy usein käyttää paljon aikaa löytääkseen haluamansa tiedon. Tämä tiedonhakuun erikoistunut WebMate selvittää käyttäjän kiinnostuksen kohteita ja oppii niitä automaattisesti. WebMaten eräs käytännöllinen ominaisuus on, että se osaa muodostaa elektronisen sanomalehden käyttäjän mieltymysten mukaan keräämällä uutisia internetin uutisia tarjoavilta sivustoilta. Sanomalehdestä on karsittu sellaista tietoa, jonka agentti olettaa käyttäjänsä haluavan jättää pois.

WebMate sijaitsee käyttäjän ja WWW:n välissä. Kaikki käyttäjän istunnon aikana suoritettavat tapahtumat menevät sen läpi. Siten se pystyy kirjaamaan kaiken tapahtuneen ylös ja oppimaan käyttäjänsä tapoja internetin käytössä. Käyttäjä voi lisäksi opastaa agenttiaan hakuprosessin aikana käyttöliittymänä toimivan appletin avulla.

2.6. Mukana konferensseissa ja näyttelyissä

Konferenssit ovat tapahtumia, joissa jokin tietty yhteinen kiinnostuksen kohde kerää ihmiset yhteen. Konferensseissa on tyypillisesti paljon luentoja, näyttelyitä ja kaikkea muuta kulloiseenkin aiheeseen liittyvää asiaa. Aikataulut on laadittu tarkasti, sillä konferenssien pitämiseen on varattu rajallinen määrä aikaa. Monet mielenkiintoiset luennot ja näyttelyt saattavat olla päällekkäin, jolloin on jollakin perusteella valittava, mihin tilaisuuteen aikoo osallistua.

Vaikka aihealue on konferensseissa yhteinen, vaikkapa konferenssi agenteista, jakautuvat ne usein moniin osa-alueisiin, kuten henkilökohtaisiin agentteihin, rajapinta-agentteihin ja muihin vastaaviin osiin. Silloin henkilökohtaisista agenteista kiinnostunut voi pitää yhtenä valintaperusteena luentoja ja näyttelyitä, jotka sivuavat kiinnostavia aiheita. Näistäkin tapahtumista pitää kuitenkin vielä valita. Seuraavana valintaperusteena voisi

olla se, mitä muut samoista asioista kiinnostuneet ovat pitäneet eri tapahtumista. Se edellyttää tietysti tiedustelua muilta kollegoilta.

Konferenssit ovat sopivia tapahtumia henkilökohtaisille agenteille, jotka on sijoitettu taskutietokoneeseen. Jos kullekin vieraalle jaetaan konferenssin ajaksi taskutietokone ja agenttiohjelma, helpottuu konferenssitilaisuus vieraiden kannalta huomattavasti. Taskutietokoneen agentille on syötetty konferenssiaikataulut ja lyhyet kuvaukset konferenssin tapahtumista. Agentti henkilökohtaistetaan käyttäjänsä mukaan, jolloin se tietää käyttäjänsä mielenkiinnon kohteet. Se osaa kertoa, milloin jokin kiinnostava luento on ja ehdottaa vaikkapa hyvää ravintolaa lounasaikaan.

Monesti myös konferenssien valitseminen saattaa olla aikaa vievää puuhaa, sillä nykyään monilla aloilla on useita konferensseja vuodessa. Konferensseilla on usein verkkosivut, joilla kerrotaan, mitä konferenssi käsittelee ja millainen aikataulu sillä on. Osallistujan pitää siis löytää eri tapahtumien verkkosivut, tehdä vertailuja ja tutustua aikatauluihin. Tätäkin varten on olemassa agenttijärjestelmä.

Minen ja muiden [2001] järjestelmässä käyttäjän agentti kerää käyttäjää kiinnostavien konferenssien nimiä, aikatauluja, ilmoittautumispäivämääriä, URL-osoitteita ja konferenssien pitopaikkoja käyttäjän vastaanottamien sähköpostien ja konferenssien verkkosivujen avulla. Nämä tiedot tallennetaan tietokantaan johon käyttäjä pystyy agenttikäyttöliittymänsä avulla tekemään kyselyitä luonnollisella kielellä. Tällainen järjestelmän rakenne ylimmällä tasolla on seuraava: Tiedon vastaanotto ja verkkosivujen keräys moduuli, tiedon suodatus moduuli, tietokannan hallinta moduuli ja käyttöliittymä, mikä analysoi luonnollista kieltä sekä oppii käyttäjänsä kiinnostuksen kohteita.

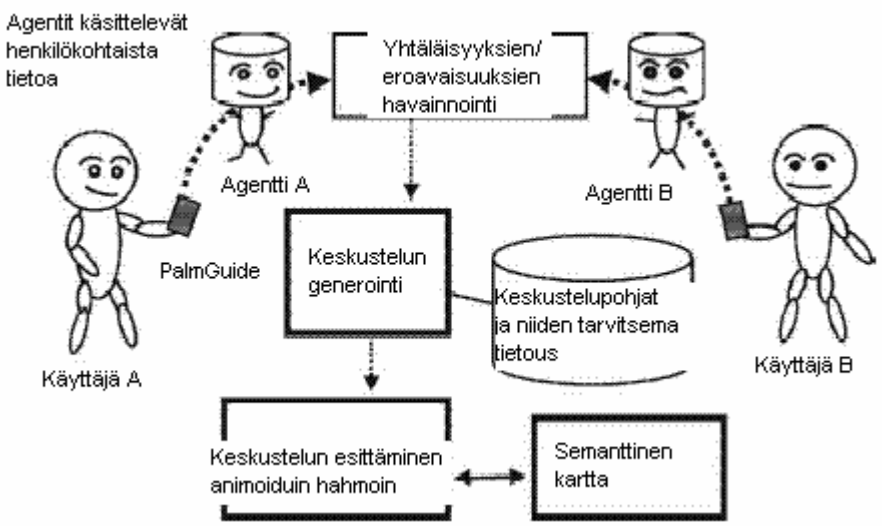
Sumi ja Mase [2002] ovat kehittäneet järjestelmää, joka tukee yhteisiä kiinnostuksen kohteita jossakin yhteisössä. Heidän järjestelmänsä käyttäjällä on mukanaan taskutietokone, jonne on asennettu henkilökohtainen tapahtumaopas, PalmGuide. Aluksi käyttäjä valitsee haluamansa animoidun opashahmon. Sitten agentti toimii käyttäjän oppaana tapahtumassa. Se osaa suositella näyttelyitä käyttäjän mielenkiinnon perusteella. Agentti pitää kirjaa niistä näyttelyissä, joissa käyttäjä on käynyt ja käyttää tätä tietoa hyväkseen etsiessään muita käyttäjiä, joita kiinnostavat samat kohteet.

Käyttäjä voi itse arvostella ne näyttelyt, joissa tämä on toistaiseksi käynyt. Näiden arvioiden perusteella agentin kyky suositella muita näyttelyitä paranee, sillä agentti poimii näyttelyihin liittyvistä tiedoista avainsanoja, joita se vertaa muiden näyttelyiden tietoihin. Jos käyttäjä on arvostellut jonkin

aiemman näyttelyn korkealle ja agentti löytää avainsanojen perusteella vastaavan näyttelyn, suosittelee agentti sitä.

AgentSalon on osa Sumin ja Masen kehittämää järjestelmää. Se on suurella kosketusnäytöllä varustettu informaatiokioski, jossa eri käyttäjien agentit voivat kommunikoida keskenään käyttäjien seuratessa keskustelun kulkua. AgentSalon-järjestelmässä voi samanaikaisesti olla kahdesta viiteen agenttia.

Oletetaan, että käyttäjä A on käynyt näyttelyissä 1, 2, 3 ja 4 ja käyttäjä B on käynyt vastaavasti näyttelyissä 2, 4, 5 ja 6. Tässä tapauksessa käyttäjien agentit havaitsevat, että molemmat ovat käyneet näyttelyissä 2 ja 4, eli heillä on yhteistä kokemusta aiheesta. Siten käyttäjän A agentti suosittelee näyttelyitä 1 ja 3 käyttäjälle B ja käyttäjän B agentti suosittelee näyttelyitä 5 ja 6 käyttäjälle A.



Kuva 1: AgentSalon-järjestelmän arkkitehtuuri [Sumi ja Mase, 2002].

Jos käyttäjien kiinnostuksen kohteet ovat eriäviä, esimerkiksi käyttäjä A on arvioinut näyttelyn 1 kiinnostavaksi, mutta käyttäjä B ei, keskustelevat käyttäjien henkilökohtaiset agentit aiheesta. Käyttäjän A agentti sanoo esimerkiksi näyttelyn 1 olleen erittäin mielenkiintoinen, kun taas käyttäjän B agentti kertoo, että he eivät pitäneet siitä. Jos taas mitään ei tapahdu pitkään aikaan, tarjoaa käyttäjän agentti puheenaiheeksi esimerkiksi kaikkein suosituinta tapahtumaa.

3. Tiimeissä toimimisesta

Kuvitellaan tehtävää, jossa tarkoituksena on evakuoida ihmisiä uhatusta paikasta. Evakuointia ei voi laajassa mittakaavassa suorittaa yhden henkilön

voimin ja jotta evakuointi voidaan yleensä suorittaa, tarvitsee huomioida useita tehtävään liittyviä parametreja. Niitä ovat evakuointipaikka, tehtävässä tarvittavien helikoptereiden määrä, laskeutumispaikat niille, sekä lukuisat muut parametrit, mitä tällainen operaatio vaatii [Pynadath ja Tambe, 2000]. On myös huomioitava, että lähes aina tällaisissa tehtävissä suunnitelmat muuttuvat jatkuvasti, sillä mukaan tulee odottamattomia muuttujia lähes väistämättä. Pitää siis voida varautua siihen, mihin ei voi varautua!

Ohjelmoitaessa monimutkaisia henkilökohtaisia agentteja ja niistä rakentuvia järjestelmiä, voidaan esimerkkiä ottaa edellä mainitun kaltaisista tilanteista. Agenttien on osattava toimia tiimeissä, jotta ne voisivat olla oikeasti avuksi tietynlaisissa tilanteissa. Yksin toimiva agentti ei voi suorittaa edellä mainittua tai sen kaltaista tehtävää. Yksinään ei siitä ainakaan ole juuri mitään apua. Useiden agenttien kommunikaatio ja yhteistyö mahdollistaa osaltaan esimerkin kaltaisen operaation organisointia.

Kriisipalveluorganisaatiossa apuvälineinä toimivat henkilökohtaiset agentit voivat jakaa henkilöstöresursseja tiimeihin sekä materiaaliresursseja kriisiryhmille niiden tarpeiden mukaan, jotta voidaan nopeuttaa toimintaa kriisin sattuessa. Tällaiseen toimintaan valjastetut agentit voivat myös valvoa toiminnan edistymistä kriisin aikana, ja jakaa resursseja uudelleen sen mukaan, missä apua tarvitaan eniten. Samalla tiimityöskentely periaatteella on henkilökohtaisten agenttien osattava toimia missä tahansa organisaatiossa, sillä kriisiorganisaatio ei poikkea olennaisesti muista organisaatioista.

Dynaaminen agenttien tiimityöskentely mahdollistaa organisaatiolle täsmällisen, luotettavan ja joustavan tavan päästä tavoitteisiin. Järjestelmä pystyy tällöin reagoimaan nopeasti odottamattomiin tilanteisiin ja mukautumaan joustavasti uusiin tilanteisiin [Pynadath ja Tambe, 2000].

Kuvitellaan seuraavaksi, että kriittinen kokous on määrätty pidettäväksi juuri viime hetkellä ja tehtävänä on päättää, ketkä kokoukseen osallistuvat, mitä välineitä kokoukseen tarvitaan ja mistä ne saadaan, suunnitella matkat sekä valvoa suunnitelman toteutumista. Oletetaan, että tätä tehtävää varten suunnitelmat on saatu alustavasti tehtyä; kokouksen osallistujat on valittu, matkasuunnitelmat on tehty, sekä varusteiden toimitusta varten on saatu laadittua suunnitelma.

Jos nyt joku osanottajista sairastuu, on pystyttävä valitsemaan uusi osallistuja kokoukseen. Jos taas lennot ovat myöhässä, on otettava huomioon tässä ongelmatilanteessa mukana olevat osapuolet (erityisesti lentoyhtiöiden osalta) ja laadittava uudet kokousaikataulut muuttuneiden tietojen pohjalta. Jos varustetoimituksissakin tulee yllättäen häiriöitä, on otettava yhteyttä

paikallisiin tietokoneita ja muita varusteita toimittaviin tahoihin, jotta tarvittavat välineet saadaan paikalle.

Tämän esimerkin perusteella voidaan tutkimusalueeseen liittyviä ongelmia listata seuraavasti [Tambe et al., 2000]:

1) Dynaaminen tiimin muodostaminen.

2) Pysyvät tiimit ympäri vuorokauden seitsemän päivää viikossa.

3) Tiimien koordinointi ja monitorointi.

4) Agenttien korvaaminen toisilla agenteilla laajassa mittakaavassa.

5) Uusien agenttien nopea mukaan ottaminen.

6) Agenttien kompleksisuuden, määrän ja heterogeenisuuden skaalautuvuus.

Järjestelmän kehityksessä pitäisi huomioida seuraavia asioita:

1) Tiimien koostamista pitää tukea myös lyhyessä ajassa suoritettavien tehtävien osalta.

2) Järjestelmä pitää sisällään sekä ihmisiä, että tietokoneita.

3) Tiimi koostuu täysin ihmisten asioita välittävistä agenteista.

4) Järjestelmän pitää tukea toisten tekemien agenttien dynaamista integrointia järjestelmään.

5) Tiimien muodostamiselle pitää olla edeltä määritellyt toimintamallit.

6) Poikkeaviin tilanteisiin tulee reagoida nopeasti.

7) Ihmisten ja tietokoneiden sijoittumista organisaatiossa tulee voida valvoa.

4. Osana laajempaa järjestelmää

Henkilökohtainen agentti on usein osa jotain laajempaa tietojärjestelmää. Se käyttää hyväkseen järjestelmän muiden osien palveluita. Palveluntarjoajatkin voivat olla agentteja, joiden kanssa henkilökohtainen agentti kommunikoi. Agenttipohjaisen lähestymistavan eräs tarkoitus on laajentaa viestiympäristön toimivuutta ja mahdollistaa uusia viestintätapoja verrattuna perinteiseen elektroniseen viestintään, kuten sähköpostiin ja tekstiviesteihin.

Kaukonen et al. [1999] ovat esitelleet agenttiteknologian hyväksikäyttöä verkkopalveluihin liittyvässä artikkelissaan, josta selviää, mitä kaikkea voi olla mukana suuressa agenttiteknologiaa hyödyntävässä järjestelmässä. Tässä ja seuraavassa luvussa tarkoituksena on selventää lähinnä edellä mainitun artikkelin pohjalta case-luontoisesti, millainen rooli henkilökohtaisella

agentilla usein on koko järjestelmässä. Tullaan myös huomaamaan, että henkilökohtainen agentti ei välttämättä tee kaikkea omin avuin, vaan käyttää hyväkseen erilaisiin tehtäviin erikoistuneita järjestelmässä toimivia muita agentteja.

4.1. Agenttiteknologia ja tietoverkot

Tutkielman toisessa luvussa annettiin esimerkki siitä, miten henkilökohtainen agentti voi auttaa matkan suunnitteluun liittyvissä toimenpiteissä. Seuraavaksi havainnollistetaan hieman tarkemmin, mitä kaikkea tapahtuu, kun käyttäjä suunnittelee matkaansa henkilökohtaisen agenttinsa avulla.

Käyttäjä voi ottaa yhteyttä agenttiinsa millä tahansa saatavilla olevalla päätelaitteella, esimerkiksi SMS-viesteillä, WWW-selaimella tai sähköpostitse. Miten on sitten mahdollista, että käyttäjän oma agentti ymmärtää, mitä siltä pyydetään? Rajapinta-agentti osaa reitittää komennot suoraan käyttäjän agentille. Sen ainoana tehtävänä onkin muuttaa agenttien välinen viestintä käyttäjän ymmärtämään muotoon ja käyttäjän viestit agenttien ymmärtämään formaattiin.

Seuraavaksi käyttäjän agentti etsii palveluntarjoaja-agentteja, jotka ovat erikoistuneet ulkomaanmatkoihin. Tavoitettu palveluntarjoaja-agentti lähettää käyttäjän mukaan profiloidun elektronisen esitteen suoraan käyttäjälle. Lopulta käyttäjä lähettää agenttinsa kautta yhteydenottopyynnön matkanjärjestäjälle. Pyynnön yhteydessä on tiedot käyttäjän päätelaitteesta, sijainnista ja asiakkaan mahdollisista aiemmista matkoista, jolloin matkanjärjestäjä voi osaltaan tarjota parempaa palvelua.

Asiakas voi vielä muokata matkaansa, mikäli niin haluaa. Matkanjärjestäjä voi ehdottaa asiakkaalle parempia hotelleja ja näyttää käyttäjälle videoesityksiä kohteesta. Videoesitys siirretään operaattorilta käyttäjälle operaattorin agentin luoman videon hakuun ja lähetykseen erikoistuneen videoagentin avulla. Luotu agentti hakee halutun videon tietokannasta ja alkaa lähettää videokuvaa asiakkaalle. Käyttäjän päätelaitteelle siirretään videoesityksen mahdollistava käyttöliittymäagentti, joka vastaanottaa videon ja näyttää sen käyttäjälle päätelaitteen tarjoamien mahdollisuuksien mukaan.

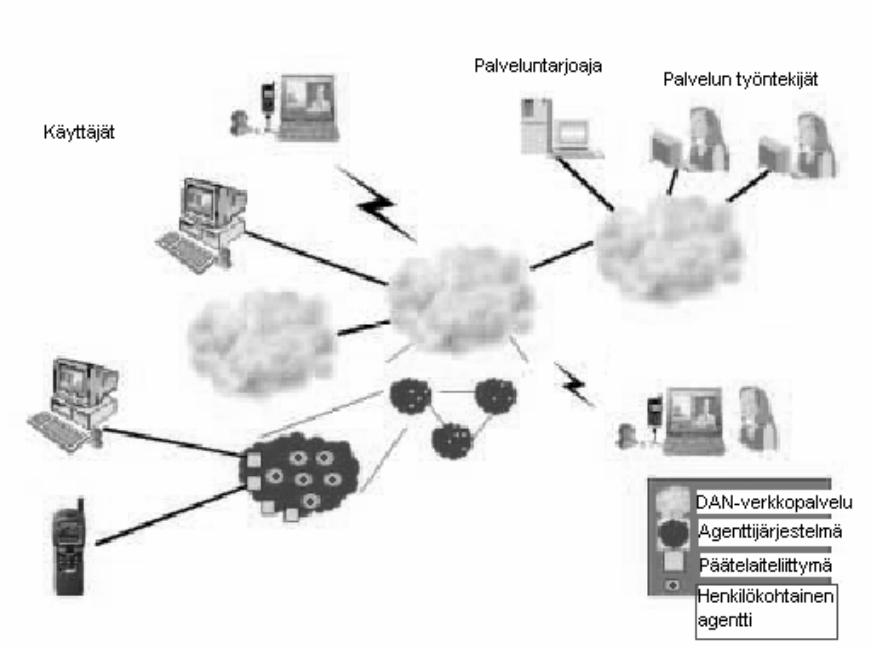
Ohjelmistoagenttiteknologia helpottaa erityyppisten verkkojen ja laitteiden yhteensulauttamista. Rajapinta-agentit hoitavat erilaisten verkko- ja päätelaitteiden sekä sovellusten sulauttamisen. Agenttiteknologia tarjoaa myös loistavat puitteet dynaamisen palvelualustan toteuttamiseen. Agentit maksimoivat suorituskyvyn käyttämällä vähiten kuormitettuja verkon osia.

Ne kykenevät tarpeen vaatiessa tasoittamaan järjestelmän kuormitusta siirtymällä suotuisampaan ajoympäristöön.

4.2. DAN - Agenttipohjainen järjestelmä

Media Team Oulun CTI-hankkeessa on kehitetty ohjelmistotekniikoita tele- ja tietoverkkojen sekä päätelaitteiden sulauttamiseksi. CTI (Computer Telephony Integration) tarkoittaa yhteenliittämisen teknologiaa, jonka avulla saadaan tietoverkkojen sisällöt matkaviestimiin ja päinvastoin [MediaTeam, 2001].

Järjestelmän ytimen muodostaa DAN-palvelualusta (Distributed Agent Network), jossa matkapuhelimen, kannettavan tai pöytätietokoneen käyttäjää palvelee verkossa liikkuva oma ohjelmistoagentti. DAN-järjestelmän tavoitteena on yhdistää erilaisia päätelaitteita ja palveluja käyttäjän kannalta yhtenäiseksi ja helppokäyttöiseksi kokonaisuudeksi. Järjestelmää voidaan käyttää myös muihin tarkoituksiin kuin henkilökohtaiseen viestintään, kuten videopuheluihin ja internet-kaupankäyntiin [Tekes, 2001].

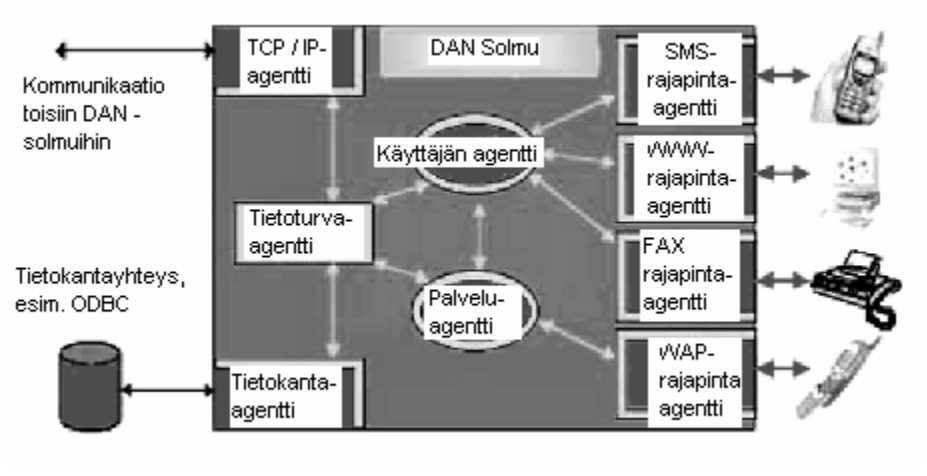


Kuva 2: DAN-järjestelmän rakenne [Kaukonen et. al, 1999].

Jokaiselle käyttäjälle varataan DAN-järjestelmässä oma henkilökohtainen agentti, jonka avulla käyttäjä käyttää verkon palveluita. Viestiliikenne rajapinta-agentilta henkilökohtaiselle agentille ja sitä kautta palveluagenteille mahdollistavat sen, että käyttäjän päätelaite ei aiheuta riippuvuuksia eikä agenttien tai palveluiden sijainnilla ole käyttäjän kannalta merkitystä. Käyttäjän agentti voi luoda uusia agentteja avukseen, jotka voivat esimerkiksi

ylläpitää käyttäjäprofiilia, kalenteria tai osoitekirjaa. Kuvista 2 ja 3 nähdään DAN- järjestelmän rakenne koko järjestelmän ja yhden solmun tasolla.

DAN-järjestelmä koostuu solmuista, joissa jokaisessa on yksi tai useampi agenttijärjestelmä. Jokaisessa solmussa on agenttien toimintaa ja viestintää valvovia ylläpitoagentteja sekä hakemistoagentteja, jotka välittävät tietoa palveluista ja niiden käyttäjistä. Tietoliikenneagentit hoitavat solmujen välisen kommunikaation.



Kuva 3: Solmu DAN-järjestelmässä [Kaukonen et. al, 1999].

Päätelaitteet on kytketty verkkoon rajapinta-agenteilla, jotka eivät yleensä voi liikkua verkossa, vaan jokaisessa agenttijärjestelmässä on niitä lisälaitteiden tai liityntämahdollisuuksien mukaan. Käyttäjän agentti voi ottaa yhteyttä käyttäjään eri tavoilla rajapinta-agenttien ansiosta. Agenttijärjestelmän tulee pitää yllä tietoa agenteista ja niiden liikkeistä, jotta rajapinta-agenttiin kytketyt käyttäjä ja hänen agenttinsa löytävät toisensa. Käyttäjän agentti taltioi käyttäjänsä liikkeitä järjestelmässä, ja näin ollen palvelun-tarjoajan ei tarvitse taltioida käyttäjähistoriaa. Käyttäjän agentilla voi olla useita tehtäviä, joita DAN-järjestelmä voi poistaa tai lisätä käyttäjän tai tämän agentin pyynnöstä. Tällainen tehtävä voi olla esimerkiksi, että agentti odottaa käyttäjän pyytämää tietoa, ja ilmoittaa, kun se on saatavilla.

Agentit voivat käyttää tietokanta-agentin tarjoamaa tietovarastoa. Järjestelmän ydin voi tarvittaessa tallentaa myös agentteja tietovarastoon, mikäli niillä ei ole tehtäviä, tai käyttäjä ei ole sillä hetkellä kytkeytyneenä verkkoon. Tietokanta-agentin tehtäviin kuuluu myös pitää solmujen väliset tietovarastot samoina. Tietoliikenne-agentit hoitavat DAN-solmujen kommunikoinnin. Siirrettävää tietoa on esimerkiksi agenttien siirrot, viestin siirto tai tietovaras-ton päivitystiedot.

DAN muodostaa siis useamman agenttiohjelmiston kokonaisuuden, jossa agentit voivat liikkua vapaasti ja jonka resurssit ovat kaikkien agenttien käytössä. Henkilökohtainen agentti toimii älykkäänä käyttöliittymänä tähän järjestelmään.

5. Henkilökohtaistaminen

Jotta henkilölle olisi apua henkilökohtaisesta agentista, pitää se henkilökohtaistaa käyttäjän tavoitteiden, tapojen ja mieltymysten mukaan. Olisi kuitenkin aikaa vievää ja hankalaa, mikäli henkilökohtaistaminen pitäisi suorittaa aina asiantuntijan toimesta. Ei olisi myöskään viisainta, että yksi ihminen organisaatiossa saisi vastuulleen koko organisaation henkilökohtaisten agenttien kouluttamisen kunkin organisaation jäsenen toiveiden ja tapojen mukaan.

Ihanteellinen agentin ominaisuus siis olisi, että se oppisi asioita seurailemalla käyttäjää ja tekemällä omia huomioita, sekä päivittäisi huomioidensa perusteella sääntökantaansa tai muuta muistia mallintavaa tietorakennettaan automaattisesti. Toinen vaihtoehto taas olisi jonkinlainen ohjelma, jonka avulla agentti voitaisiin valjastaa uuden käyttäjänsä rutiineihin sopivaksi. Tälle ohjelmalle ominaista tulisi olla helppokäyttöisyys mille tahansa käyttäjäryhmälle, joka käyttää henkilökohtaisia agentteja. Ohjelman tulisi olla helposti opittavissa vain vähäisillä tietoteknisten laitteiden käyttötaidoillakin.

Terveenin ja Murrayn [1996] mukaan on olemassa pääsääntöisesti kolme tapaa, joilla agentti voi hankkia tarvitsemaansa tietoa: loppukäyttäjän suorittama ohjelmointi (EUP, End User Programming), oppiminen ja demonstroimalla ohjelmointi (PBD, Programming By Demonstration).

EUP on näistä helpoin toteuttaa, mutta käyttäjän tarvitsee itse tehdä melko paljon töitä agentin opettamiseksi ja tietojen ylläpitämiseksi. Perinteiselläkin oppimismallilla on heikkoutensa, nimittäin agentin täytyy käyttäjän toimia seuraamalla oppia todella monia esimerkkejä ennen kuin siitä on hyötyä. Oppivaa menetelmää soveltava agentti ei myöskään voi tarjota apua aivan uusissa tilanteissa, sillä se ei ole vielä oppinut niitä. Näitä oppimismallin huonoja puolia on yritetty vähentää antamalla agenttien oppia toisiltaan. Tässäkin herää kysymys, miten käyttäjän henkilökohtainen agentti voi oppia luottamaan toisiin agentteihin.

Demonstroimalla ohjelmoinnissa laitetaan järjestelmä ”äänitysmoodiin” ja tehdään toimenpiteitä, joita järjestelmä tallentaa. Tämä menetelmä sopii hyvin tapauksiin, jotka on luontevaa esittää graafisesti. PBD yhdistää sekä EUP:ta että oppimista [Terveen ja Murray, 1996].

Minkä tyyppistä tietoa sitten henkilökohtainen agentti voi saada syötteenä? Henkilökohtaistettu järjestelmä voi saada kolmenlaista syötettä: tietoa käyttäjän luonteenpiirteisiin liittyvistä asioista, tietoa käyttäjän tietokoneenkäyttötavoista ja tietoa käyttäjän tietokoneen ohjelmistoista, tietokoneen komponenteista sekä fyysisestä ympäristöstä [Yong ja Kong, 2001].

Mitchell et al. [1994] ovat tehneet järjestelmän avustamaan kalenterin käytössä (CAP, The Calendar Apprentice). Se kykenee oppimaan käyttäjänsä tapoja automaattisesti, joten käyttäjän ei tarvitse itse kouluttaa agenttiaan, vaan se oppii sääntöjä ihmisen tavoin kokemuksen kautta. Käyttäjälle tarjotaan sopiva käyttöliittymä toiminnoille, kuten sähköpostille ja kalenterille. Kun järjestelmää käytetään, kohtelee se kaikkia vuorovaikutustilanteita harjoitusesimerkkeinä käyttäjänsä tavoista. Tästä opitusta tiedosta se tekee yleistyksiä ja käyttää opittua tietoa hyväkseen jatkossa.

Ei kuitenkaan ole itsestään selvää, että agentti, joka omaksuu sääntöjä pelkästään oppimalla, olisi paras mahdollinen kaikkiin tapauksiin. Sen vuoksi seuraavassa kohdassa esitellään hieman tarkemmin järjestelmää, jonka Terveen ja Murray [1996] ovat kehittäneet loppukäyttäjille. Järjestelmän avulla käyttäjät voivat itse säätää agenttinsa heidän tarkoituksiaan vastaavaksi. Ohjelman avulla on helppoa päivittää agentin sääntökantaa itsenäisesti. Järjestelmän etuna muihin vastaaviin lähestymistapoihin on se, että järjestelmä havaitsee automaattisesti ristiriidat sääntöjen välillä ja opastaa käyttäjää ratkaisemaan niitä. Näin ollen käyttäjä ja järjestelmä tekevät yhteistyötä kehittäessään ja ylläpitäessään sääntöjen joukkoa, jotka edesauttavat käyttäjää käsittelemään lukuisaa erilaisten tilanteiden määrää.

5.1. Agent Manager

Agent Manager [Terveen ja Murray, 1996] on järjestelmä, jolla käyttäjät voivat itse ohjelmoida sääntöjä omalle henkilökohtaiselle agentilleen graafisen käyttöliittymän kautta. Se soveltaa EUP:ta sääntöjen syöttämisessä tehden siitä kuitenkin yksinkertaista loppukäyttäjän kannalta. Vaikka sääntöjen syöttäminen on periaatteessa yksinkertaista, on ongelmana ollut tavallisesti vuorovaikutus sääntöjen välillä. Agent Managerissa on pyritty lähestymään tätä ongelmaa käyttäjän kannalta, jotta tämän ei tarvitsisi olla varsinainen ohjelmistoasiantuntija kouluttaessaan agenttiaan.

EUP:lla on selkeitä hyviä puolia. Mikäli käyttäjällä on mielessään sääntö, jonka hän haluaisi agenttinsa oppivan, on se voitava syöttää suoraan agentille sen sijaan, että odottaisi tämän oppivan säännön joidenkin toimintojen kautta. Joskus sääntö on saatava toimimaan heti, eikä ole aikaa odotella, että agentti

oppisi sen. Muun muassa edellä mainittujen perusteiden nojalla on EUP joskus parempi tapa agentin henkilökohtaistamiseen kuin puhtaasti oppiva agentti.

Agent Manager analysoi säännöt ongelmina, määrittää menetelmät ongelmien ratkaisua varten ja opastaa käyttäjää valitseman ja käyttämään korjauksia. Ajan kanssa käyttäjä ja järjestelmä ovat saaneet aikaiseksi sääntöjoukon, joka vastaa käyttäjän tavoitteita ja mieltymyksiä [Terveen ja Murray, 1996].

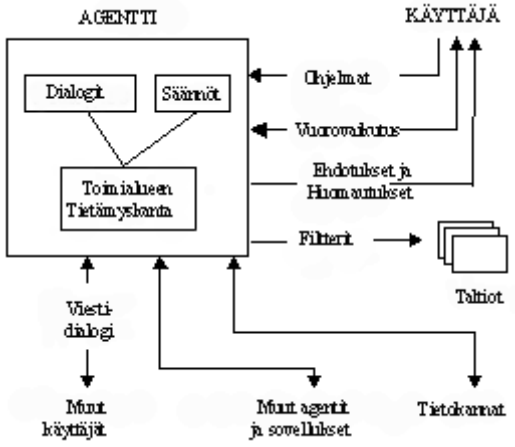
Seuraavassa luettelossa on esitelty kolme esimerkkitapausta, joissa säännöt ovat muotoa ”kun nämä ehdot täyttyvät, suorita tietyt toimet”:

1) Viestintä: Kun saan viestin pomoltani työajan jälkeen, jossa otsikkona on ”kiireellinen”, soita kotinumerooni.

2) Pankki: Kun käyttötilini saldo putoaa alle 500 € ja säästötilini saldo on yli 2000 €, siirrä 500 € säästötililtäni käyttötililleni.

3) Talous: Jos osakkeen O hinta putoaa alle 30 €, osta 100 osaketta.

Tärkeä piirre henkilökohtaisella agentilla on myös osata kommunikoida käyttäjänsä puolesta. Tämä tarkoittaa muistuttamista, kun tietyt ehdot tulevat toteen, viestien lähettämistä ja toisten käyttäjien viesteihin vastaamista.

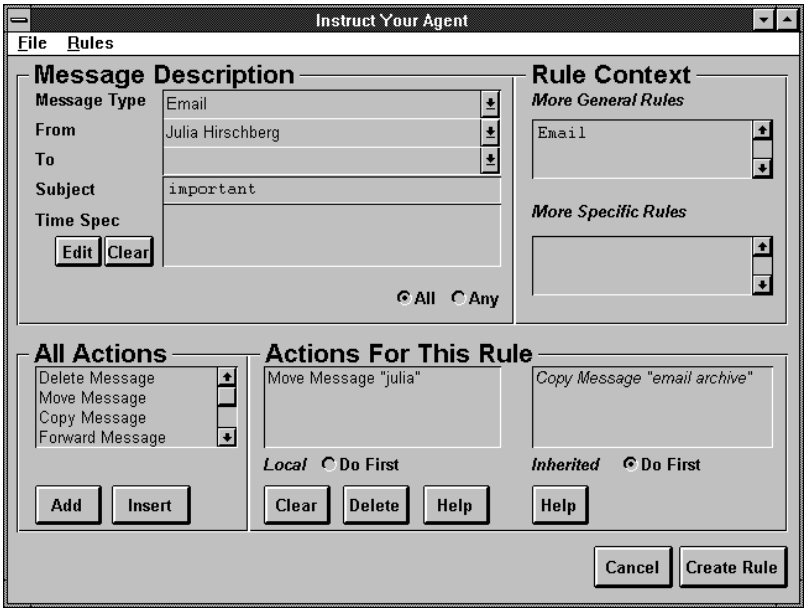


Kuva 4: Agent Managerin rakennekaavio [Terveen ja Murray, 1996].

Agent Manager toimii viestien prosessoinnissa, mutta on myös kehitettävissä muihin toiminta-alueisiin. Käyttäjä määrittelee säännöt rajoittamalla viestin tyyppiä, lähettäjää, vastaanottajia, aihetta ja päivämäärää muodostaen siten säännön tietyntyyppisille viesteille. Järjestelmä hoitaa sääntökannan päivityksen ja perii sääntöön piirteitä ylemmän tason säännöiltä.

Kuvassa 5 nähdään Agent Managerin käyttöliittymä. Kuvan tapauksessa muodostetaan sääntö, joka pätee Julia Hirchberg -nimisen henkilön lähettämiin sähköposteihin, joiden otsikossa on sana ”important”. Järjestelmä

havaitsee, että sääntö nimeltä ”Email” on yleisempi sääntö ja että yksi toiminto, Copy Message ”email archive” peritään tässä tapauksessa uudelle säännölle.



Kuva 5: AgentManagerin käyttöliittymä [Terveen ja Murray, 1996].

Toinen vastaava editori antaa käyttäjän määritellä päivämääriin liittyviä asioita, kuten ajoituksia eri tapahtumille. Agent Manager havaitsee myös ristiriidat sääntöjen välillä ja näin estää virheellisten sääntöjen muodostamisen. Se laskee korjauskaavoja, joiden avulla väärät säännöt saadaan korjattua, ja esittää nämä korjauskaavat vaihtoehtoina käyttäjälle selittäen, missä mennään vikaan, mikäli toimitaan käyttäjän haluamalla tavalla. Agent Manager toteuttaa siis näin vuorovaikutusta käyttäjän kanssa.

6. Yhteenveto

Henkilökohtaisia agentteja on suunniteltu monenlaisiin tehtäviin. Tiedonhakua helpottamassa on olemassa henkilökohtaisia agentteja, jotka matkustavat väsymättä ympäri internetiä hakien tietokokonaisuuksia omistajansa intressien mukaisesti. Internet on tullut matkapuhelimiin ja muihin kannettaviin tietokoneisiin. Näin ollen internetistä tietoa ammentava agentti on tavoitettavissa ajasta ja paikasta riippumatta.

Monet henkilökohtaisille agenteille tarkoitetuista tehtävistä ovat sellaisia, joita useat ihmiset haluavat kuitenkin myös jatkossa suorittaa ilman agentin apua. Esimerkiksi hinta-laatu -vertailujen tekeminen erilaisia kodin hankintoja tehtäessä on ehkä joidenkin mielestä mukavaa puuhaa, jota ei haluta delegoida agenteille. Toisaalta taas liike-elämässä tilausta voisi olla yrityksen

materiaalihallinnon johtajan henkilökohtaiselle agentille, joka hakisi tietoja ja muodostaisi esitteitä esimerkiksi toimistokalusteista, tietokoneista, ja ohjelmistoista erilaisten kriteerien mukaan. Liike-elämässä aika on rahaa ja kaikki aikaa säästävät toimenpiteet ovat erittäin tervetulleita.

Onko tutkimuksessa esitellyillä ohjelmilla tulevaisuutta ihmisten jokapäiväisinä työvälineinä, vai jäävätkö henkilökohtaiset agentit ainoastaan marginaalisten tutkimusryhmien apuvälineiksi ja hautautuvat hyvinä ideoina muiden tietoteknisten innovaatioiden joukkoon? Henkilökohtaisten agenttien suosio riippuu paljolti siitä, mihin tarkoitukseen ne soveltuvat ja kenelle ne ovat suunnattu. On kuitenkin vaikeaa ennustaa, tulevatko henkilökohtaiset agentit olemaan lähes kaikkien tietotekniikkaa käyttävien ihmisten apuvälineitä. Tämä riippuu monesta seikasta, mutta varmana voidaan kuitenkin pitää, että henkilökohtaisten agenttien mahdollisuudet tekoälytutkimuksen-, ja tietotekniikan kehityksen myötä tulevat laajentumaan.

Viiteluettelo

[Chaplusky et al., 2002] Hans Chaplusky, Yolanda Kim, Craig A. Knoblock, Kristina Lerman, Jean Oh, David V. Pynadath, Thomas A. Russ and Milind Tambe. Electric Elves: Agent Technology for supporting human organizations. *AI Magazine*, **Summer 2002**, 11-24.

[Chen ja Sycara, 1998] Liren Chen and Katia Sycara.WebMate: A personal agent for browsing and searching. In: *Autonomous agents* (1998). 132 – 139.

[Kaukonen et al., 1999] Saku Kaukonen, Marko Palola ja Marko Heikkinen. Henkilökohtaiset agentit – verkkopalveluiden uusi aikakausi. *Prosessori*, 50 – 52, **Marraskuu** 1999.

[Krupansky, 2002] John W. Krupansky. WWW - resource Agtivity - Foundations about software agents. Available at http://agtivity.com/. Checked 16.12.2002.

[Majanen, 2002] Mikko Majanen. Johdatus aktiiviverkkoihin. Saatavana osoitteesta http://www.student.oulu.fi/~mmajanen/essee.html. Viitattu 16.12.2002.

[MediaTeam, 2001] Media Team Oulun projektiraportti. Verkkosivu osoitteessa http://www.mediateam.oulu.fi/projects/info/cti/?lang=fi. Viitattu 16.12.2002.

[Mine et al., 2001] Tsunenori Mine, Makoto Amamiya and Teruko Mitamura. Conference information management system: Towards a personal assistant system. In: *WI* (2001). LNAI 2198, 247 – 253 (2001).

[Mitchell et al., 1994] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott and David Zabowski. Experience with a learning personal assistant. *Comm ACM* **37**, 7 (1994), 80-91.

[Pynadath ja Tambe, 2000] David Pynadath ja Milind Tambe. Teamcore: Rapid, Robust Teams from Heteregeneous, Distribute Agents. Slides available at http://www.isi.edu/agents-united/CoABS-slides/teamcore/sld001.htm. Checked 16.12.2002.

[Raisamo, 2001] Roope Raisamo. Lecture at a course Software Agents. Autumn 2001. Available as http://www.cs.uta.fi/kurssit/agents/agents-lecture2.ppt. Checked 16.12.2002.

[Sumi ja Mase, 2002] Yasuyuki Sumi and Kenji Mase. Supporting the awareness of shared interests and experiences in communities. *Int. J. Human – Computer Studies* **56** (2002), 127 – 146.

[Tambe et al., 2000] Milind Tambe, David Pynadath, Craig Knoblock, Steve Minton, Yolanda Gil and Hans Chalupsky. Dynamic team formation and crisis response in complex organizations. Slides available at http://www.isi.edu/agents-united/CoABS-slides/dynamic-team/sld001.htm. Checked 16.12.2002.

[Tekes, 2001] Tekesin verkkosivut. Älykkäitä palveluita tulevaisuuden tietoverkkoihin. Uutinen saatavilla osoitteesta http://www.tekes.fi/uutisia/uutis_tiedot.asp?id=1161. Viitattu 16.12.2002.

[Terveen ja Murray, 1996] Loren G. Terveen and La Tondra Murray. Helping users program their personal agents. In: *Conference on Human Factors and Computing Systems* (1996). 355 - 361.

[Yong ja Kong, 2001] Lim Tek Yong and Tang Enya Kong. The exploratory of personal assistants. In: *AH* (2002). LNCS 2347, 608 - 612.

Ohjelmoitavan 3D-liukuhihnan toiminta

Jyrki Rasku

Tiivistelmä.
Tässä tutkielmassa tarkastellaan nykyisten PC-koneiden grafiikoiden ohjelmoitavuutta. Aluksi luodaan lyhyt katsaus kotitietokoneiden grafiikan esittämisen historiaan ja tarkastellaan arkkitehtuuria, jonka avulla ohjelmoitavuus on tehty mahdolliseksi. Seuraavaksi käsitellään liukuhihnan ohjelmoitavia komponentteja, jotka vastaavat geometrian laskemisesta ja pinnoittamisesta. Tämän jälkeen tarkastellaan 3D-liukuhihnalle kirjoitetun ohjelman liittämistä varsinaiseen ohjelmaan. Tutkielman lopussa pohditaan 3D-grafiikan ohjelmoitavuuden tulevaisuutta sekä mahdollisia hyvä ja huonoja puolia.

CR-luokat: I 3.3, I 3.7, D 3.2, I 3.1.

1. Johdanto

PC-koneissa käytettävät sovellukset ovat nykyisin suurimmaksi osaksi graafisia, tai ainakin niiden käyttöliittymä on graafinen. Internet, erilaiset pelit ja suunnittelusovellukset asettavat suuria vaatimuksia PC-koneiden grafiikan käsittelylle.

Ennen vuotta 1995 PC-koneiden grafiikan tuottamisesta vastasi prosessori lähes kokonaan. Vähitellen prosessorin kuormaa alettiin pienentää siirtämällä osa pelkän grafiikan tuottamiseen tarvittavista toiminnoista näytönohjaimen tehtäviksi. Aluksi parannukset olivat vain marginaalisia tai tuottivat laaduk-kaan kuvan suorituskyvyn kustannuksella. Laitevalmistajien ja ohjelmointirajapintojen tarjoajien yhteistyön avulla on saatu kehitettyä arkkitehtuuri, joka siirtää suuren osan grafiikan tuottamisesta näytönohjaimelle. Tämän arkkitehtuurin sovelluksena näytönohjaimen grafiikkaprosessoriin on lisätty ns. Transform&Lightning- ja Multitexturing- yksiköt. Transform&Lightning-tekniikasta käytetään yleisesti lyhennettä T&L.

Transform&Lightning-yksikkö vastaa geometrian sekä valaistuksen laskemisesta, ja Multitexturing-yksikkö erilaisten tekstuurien sekoittamisesta ja lisäämisestä geometriaan. T&L-arkkitehtuurin mukaisen liukuhihnan ohjelmoiminen on suhteellisen helppoa Microsoftin DirectX- [4] ja Silicon

Graphicsin OpenGL-ohjelmointirajapintojen [7] avulla. T&L-arkkitehtuurin hyviä puolia ovat sen nopeus ja laitteistoriippumattomuus. Arkkitehtuurin heikkoutena on sen jäykkyys, joka johtuu siihen staattisesti liitetyistä metodeista, joita ei voi muuttaa. Tämä rajoittaa ohjelmoijan vapautta toteuttaa omia erikoisefektejään ja johtaa sovellusten visuaaliseen samankaltaisuuteen.

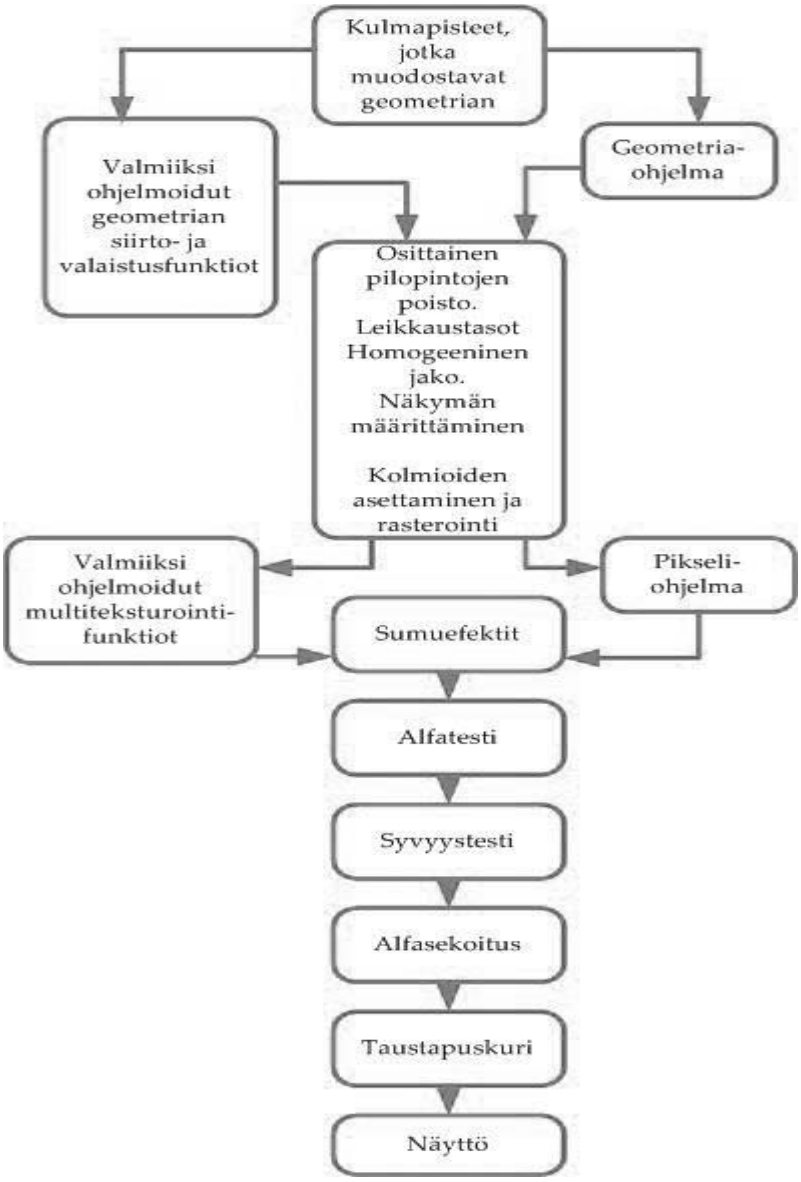
Seuraava kehitysaskel on olemassa olevan 3D-liukuhihnan tiettyjen osien muuttaminen ohjelmoitaviksi. Tämän tutkielman tarkoituksena on kuvata DirectX 8.1 -tasoista 3D-liukuhihnaa ohjelmoitavilta osin.

2. 3D-liukuhihna

Yleiskuvan saamiseksi tarkastellaan aluksi kuvan 1 mukaista yksinkertaistettua 3D-liukuhihna-arkkitehtuuria, jossa ohjelmoitavuus ja valmiiden funktioiden käyttö on vapaasti valittavissa. Kuvasta voidaan helposti huomata, että uudet ominaisuudet on lisätty olemassa olevaan arkkitehtuuriin. Tästä aiheesta enemmän lähteessä [3].

Luetaan kuvaa 1 ylhäältä alas. Ensimmäisenä liukuhihnalla on PC-koneen prosessorin näytönohjaimelle lataamat geometrian kulmapisteet. Kulmapisteiden käsittely voidaan suorittaa vaihtoehtoisesti käyttämällä valmiiksi ohjelmoituja funktioita, tai ohjelmoidun geometriaohjelman avulla. Seuraavassa vaiheessa käsitellyille kulmapisteille suoritetaan toimenpiteitä, joiden avulla geometrian virtuaalista näkymää aletaan valmistella kaksiulotteiseen tasoon piirtämistä varten. Tässä vaiheessa näkymästä poistetaan osat, jotka eivät mahdu näytölle. Kulmapisteiden projisointi kaksiulotteiseen tasoon lopettaa kulmapisteiden olemassaolon, joka jälkeen piirrettävää geometriaa aletaan käsitellä kaksiulotteisena, pikseleistä muodostuvana kuvana.

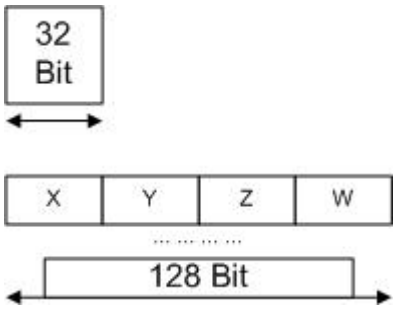
Seuraava vaihe on tasoon muutetun geometrian pinnoittaminen. Pinnoittaminen voidaan geometrian luomisen tavoin suorittaa valmiiden funktioiden tai ohjelmoidun pikseliohjelman avulla. Grafiikan käsittely multiteksturoinnin tai pikseliohjelman jälkeen ei kuulu tämän tutkielman alueeseen. Aiheesta enemmän lähteestä [2].



Kuva 1. 3D-liukuhihnan periaate.

3. Geometriayksikön arkkitehtuuri

Tarkastellaan aluksi geometriayksikön arkkitehtuuria yleisesti. Geometria-yksikkö koostuu joukosta rekistereitä, vektorilaskentayksiköstä ja skalaarilas-kentayksiköstä. Jokainen rekisteri on kooltaan 128 bittiä, johon voidaan tallentaa neljästä liukulukukomponentista muodostuva vektori. Yksittäisen komponentin koko on siis 32 bittiä. Kuva 2 esittää pienintä tallennettavaa tietoa.



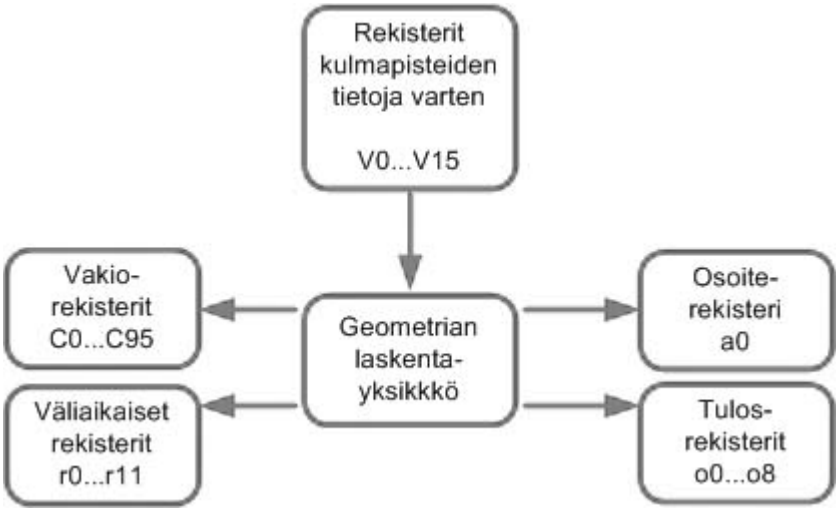
Kuva 2. Pienin tallennettava yksikkö.

Geometriaprosessorissa on viisitoista syöterekisteriä, joiden lataamisen suorittaa PC:n prosessori. Syöterekisterit sisältävät tietoa kulmapisteiden attribuuteista, joita tarkastellaan myöhemmin. Prosessori täyttää myös geometriayksikön vakiorekisterit, joita on toteutuksesta riippuen 96 tai enemmän. Tämä riippuu näytönohjaimen valmistajasta. Vakiorekisterien lukumäärä on valittu siten, että niihin mahtuu tietoa, jonka avulla voidaan suorittaa monipuolisia animaatioita. Tästä kerrotaan enemmän konferenssiartikkelissa [3]. Rekisterien sisällöstä kerrotaan myöhemmin.

Aikaisemmin mainittiin, että kaikki rekisterit sisältävät liukulukuja. Tämä ei aivan pidä paikkaansa, koska geometriayksikköön kuuluu erityinen kokonaislukutyyppinen osoiterekisteri. Osoiterekisterin avulla voidaan tehdä suhteellisia ja absoluuttisia osoituksia vakiorekistereihin. Tätä voidaan ajatella vaikkapa c++ -tyylisenä taulukon indeksointina.

Geometriayksikön suorittamia laskutoimituksia varten on kaksitoista väliaikaista rekisteriä, joita käytetään laskennassa tarvittavien välitulosten tallentamiseen. Valmiiden laskutoimitusten tulokset geometriayksikkö tallentaa tulosrekistereihinsä. Tulosrekisterien lukumäärä on valmistajakohtainen.

Geometriayksikön arkkitehtuuria esittää kuva 3.



Kuva 3. Geometriayksikön arkkitehtuuri.

3.1. Rekisterit

Geometriayksikön käytön aloittamiseksi PC:n prosessori täyttää syöterekisterit kulmapisteiden attribuuteilla. Näitä ovat tyypillisesti kulmapisteen koordinaatit x,y,z,w, diffuusio-väri (r,g,b,a), heijastunut väri (r,g,b,a), tekstuurikoordinaatit (t,u), sumu (f,*,*,*) ja pisteen koko (p,*,*,*). Nämä ovat geometriayksikön kannalta ainoastaan luettavissa olevia attribuutteja. Syöterekistereihin viitataan merkinnöillä v0,…,v15 rekisterin paikasta riippuen. Yksittäiseen rekisterin komponenttiin viitataan pistenotaatiolla. Esimerkiksi rekisterin v1 y-komponenttiin viitataan merkinnällä v1.y. Tehokkuuden takaamiseksi syöterekisterit ovat dirty-bit -tyypisiä, eli ainoastaan muuttuneet tiedot ladataan rekistereihin. Syöterekisterit ja muut luettavissa olevat rekisterit tukevat komponenttien vastaluku- ja vaihto-ominaisuuksia, joista kerrotaan myöhemmin.

Vakiorekisterit sisältävät kulmapisteiden siirtoon tarvittavat matriisit sekä näkymän valaistukseen liittyvät tiedot, esimerkiksi valonlähteen paikkavektorin. Lisäksi ohjelmoija voi vapaasti käyttää vakiorekistereitä haluamallaan tavalla. Vakiorekistereitä merkitään käyttämällä pientä c-kirjainta. Rekisterit ovat tyypillisesti välillä c0, …, c95, mutta tämä vaihtelee eri toteutusten välillä. 3D-ympäristössä tavanomainen pisteen siirto suoritetaan pisteen paikkavektorin ja 4x4-matriisin tulona. Tämän tyyppisen matriisin tallentamiseksi vakiorekistereihin käytetään peräkkäisiä paikkoja. Esimerkiksi rekisterit välillä c0,…, c3 voidaan käsittää matriisiksi, jonka sarakkeena on yksi rekisteri. Syöterekisterien tavoin vakiorekisterien lataamisesta vastaa PC:n prosessori, ja ne ovat geometriayksikölle ainoastaan luettavissa.

Osoiterekistereitä on vain yksi ja sitä käytetään vakiorekisterien osoittamiseen. Osoiterekisteriä merkitään pienellä a-kirjaimella. Riippuen a:n alkuarvosta vakiorekisterin tiettyyn kohtaan voidaan viitata merkinnällä c[a.x+n], missä n voi kasvaa väleillä 1 tai 4 riippuen siitä, viitataanko matriisiin vai vektoriin. Väli voi olla tietysti mielivaltainen, mutta silloin selaamisella pitää olla jokin muu tarkoitus.

Väliaikaisia rekistereitä on kaksitoista kappaletta ja niitä merkitään pienellä r-kirjaimella. Nämä rekisterit ovat geometriayksikön kannalta luettavissa ja kirjoitettavissa, mutta lukuja voi olla vain kolme ja kirjoituksia yksi suoritettua käskyä kohti. Väliaikaisien rekisterien tarkoituksena on toimia geometriayksikön laskutoimitusten suorittamisen apuna. Rekisterien luku- ja kirjoitusoikeuksien rajoittaminen tukee tiedon nopeaa läpivirtausta [6].

Geometriayksikkö kirjoittaa laskutoimitustensa tulokset tulos-rekistereihin, joiden lukumäärä vaihtelee toteutuskohtaisesti. Yleensä näitä on yhdeksän ja niihin viitataan pienellä o-kirjaimella. Tulosrekistereihin tallennetuille tiedoille käytetään ennalta määrättyä nimeämistapaa, koska näitä rekistereitä käytetään jatkossa pikseliohjelman syötteinä. Taulukossa 1 esitetään tulosrekisterien nimeämisen merkitys.

Nimi	**Koko**	**Kuvaus**
oDn	64 bittiä	diffuusioväri oD0 ja heijastunut väri oD1
oPos	32 bittiä	kulmapisteen paikka
oTn	256 bittiä	tekstuurikoordinaatit oT0…oT7
oPts.x	32 bittiä	pisteen koko
oFog.x	32 bittiä	sumun etäisyys pisteestä

Taulukko 1. Tulosrekisterien nimeäminen.

3.2. Vektori- ja skalaarilaskentayksiköt

Geometriayksikkö suorittaa rinnakkain vektori- ja skalaarilaskentaa käyttämällä edellä kuvattuja rekistereitä. Vektoriyksikkö suorittaa tyypilliset vektorioperaatiot, esimerkiksi ristitulon laskemisen. Skalaariyksikköä tarvitaan mm. vektorien normeeraamiseen ja vertailujen suorittamiseen.

Suorituskyvyn parantamiseksi skalaarilaskennassa tarvittava jakolasku on korvattu käänteisluvulla kertomisella. Tietovirran mahdollisimman nopean läpimenon ja kulmapisteiden keskinäisen riippumattomuuden takaamiseksi geometriayksikkö ei sisällä silmukkarakenteen eikä haarautumiskäskyjen ohjelmointimahdollisuutta. Tämä suunnitteluratkaisu kuvataan paremmin liitteessä [3].

Vektori- ja skalaariyksiköt mahdollistavat rekisterien lukemisen yhteydessä tiedon muuttamisen vastaluvukseen, ja komponenttien paikkojen vaihtamisen. Vastaluvun mahdollistamisen vuoksi käskykantaan ei tarvitse lisätä vähennyslaskua. Lisäksi rekisterien kirjoitusvaiheessa on mahdollista maskeerata tulosrekisterit. Tämä tarkoittaa sitä, että osa rekisterien komponenteista voidaan jättää kirjoittamatta. Koodikatkelma 1 kuvaa komponenttien paikkojen vaihtamista ja vastaluvun ottamista ristitulon laskemisessa.

```
mul r0, r1.yzxw, r2zxyw
mad r0, -r2.yzxw, r1.zxyw, r0
```

Koodikatkelma 1. Ristitulo.

Koodikatkelman 1 rivillä 1 suoritetaan rekisterien r1 ja r2 komponenttien paikkojen vaihto pistenotaation avulla. Jos kirjoitettaisiin mul r0, r1, r2, niin tämä vastaisi samaa kuin mul r0, r1.xyzw, r2.xyzw. Koodinkatkelman 1 rivillä 2 rekisterin r2 sisältö muutetaan lukemisen yhteydessä vastaluvukseen.

Tulosrekisterien kirjoittamisen maskeeraamista kuvaa koodikatkelma 2.

```
mov r1.x , r2
```

Koodikatkelma 2. Yhden komponentin kirjoittaminen.

Koodikatkelmassa 2 rekisterin r2 sisällöstä kirjoitetaan vain x-komponentti rekisteriin r1. Vastaavalla tavalla voidaan määrätä muut kirjoitettavat tulos-rekisterin komponentit.

3.3. Käskykanta

Geometriayksikön käskykanta muodostuu seitsemästätoista erilaisesta käskystä. Nämä jaetaan käyttötarkoituksiensa perusteella vektori-, skalaari- ja sekalaisiin käskyihin. Taulukko 2 sisältää käskykannan kuvauksineen.

Käsky/Operaatiokoodi	Nimi	Kuvaus
MOV	move	vektori-käsky
MUL	multiply	vektori-käsky
ADD	add	vektori-käsky
MAD	multiply and add	vektori-käsky
DST	distance	vektori-käsky
MIN	minimum	vektori-käsky
MAX	maximum	vektori-käsky
SLT	set on less than	vektori-käsky
SGE	set on greater than	vektori-käsky
RCP	reciprocal	skalaari-käsky
RSQ	reciprocal square root	skalaari-käsky
DP3	3 term dot product	skalaari-käsky
DP4	4 term dot product	skalaari-käsky
LOG	2 base logarithm	sekalainen-käsky
EXP	2 base exponent	sekalainen-käsky
LIT	phong lightning	sekalainen-käsky
ARL	address register load	sekalainen-käsky

Taulukko 2. Geometriayksikön käskykanta.

Jako vektori- ja skalaarikäskyihin on selkeä, mutta sekalaiset käskyt vaativat hieman tarkennusta. Ohjelmoitavat 3D-liukuhihnan osat on lisätty olemassa olevaan arkkitehtuuriin, jossa valaistuksesta vastaa oma yksikkönsä. Tämän yksikön ohjelmoitavuus olisi käytännössä liian vaikeaa [3]. Lisäksi paljon valaistusominaisuuksia sisältävät itse ohjelmoidut mallit vaatisivat paljon optimointia toimiakseen hyväksyttävällä nopeudella. Tämän vuoksi valaistusyksikkö kytketään pois geometriaohjelman suorituksen ajaksi, ja valaistus toteutetaan käskykantaan lisätyn LIT-käskyn avulla. Loput sekalaisista käskyistä liittyvät pääasiassa sumuefektien toteuttamiseen.

Geometriayksikkö vastaa kolmiulotteisen mallin ääriviivojen laskemisen kaksiulotteiselle näytölle. Seuraava vaihe kuvan muodostamisessa on geometriayksikön tuottaman rautalankamallin pinnoittaminen, jonka suorittaa pikseliyksikkö.

4. Pikseliyksikkö

Pikseliyksikön arkkitehtuuri on samanlainen kuin geometriayksikön. Rekisterien tarkoitus ja käskykanta on kuitenkin toisenlainen. DirectX 8.1 -rajapinta tukee pikseliyksikön versioita 1.1, 1.2, 1.3 ja 1.4. Eri versioihin perustuvat

toteutukset ovat keskenään täysin erilaisia [2]. Tässä tutkielmassa tarkastellaan ainoastaan versiota 1.1.

Pikseliyksikön arkkitehtuuria esittää kuva 4.

Kuva 4. Pikseliyksikön arkkitehtuuri.

4.1. Rekisterit

Värirekisterit V0,…, V1 saavat syötteensä geometriayksiköltä. Näihin rekistereihin on talletettu kulmapisteeseen liitetyt diffuusio- ja heijastumisvärit. Värirekisterit ovat pikseleiden laskentayksikölle vain luettavissa olevia arvoja.

Vakiorekisterit c0,…,c7 on tarkoitettu vain luettaviksi, ja niiden lataamisen suorittaa PC:n prosessori tai pikseliohjelma Näihin voidaan tallentaa esimerkiksi valaistuksen laskemiseen liittyviä vektoreita. Vakiorekistereiden arvoalue rajoittuu välille [-1.1].

Väliaikainen rekisteri r0 on myös tulosrekisteri. Aritmeettisten operaatioiden välitulosten tallentamiseen käytetään siis rekistereitä r0,…,r1.

Rekisterit t0…t3 liittävät käytettävän tekstuurin ja sitä vastaavat tekstuurikoordinaatit yhteen.

4.2. Käskytyypit

Pikseliohjelma muodostuu neljästä vaiheesta, jotka jakautuvat suoritusvaiheessa olevien käskytyyppien mukaan. Ensimmäisenä suoritetaan käsky, joka kertoo käytettävän ohjelman versionumeron. Jos vakiorekisterien lataamiseen käytetään pikseliyksikköä PC:n prosessorin sijasta, suoritetaan lataamiseen liittyvät käskyt välittömästi versiokäskyn jälkeen. Seuraavana suoritusvuorossa ovat käskyt, joiden tarkoituksena on lukea tekstuurien tietoja rekistereistä t0,…,t3. Viimeisessä vaiheessa pikseliyksikkö suorittaa aritmeettiset käskyt, joiden avulla lopullinen geometrian pinnoittaminen saadaan toteutettua.

Pikseliyksikön version 1.1 käskykantaan kuuluu 33 erilaista käskyä. Suurin osa käskyistä käsittelee tekstuurien ja tekstuurikoordinaattien välisiä suhteita. Nämä käskyt liittyvät kuvan 4 kohtaan tekstuurien osoittaminen. Pikseliyksikön käskyjen tarkat kuvaukset kerrotaan lähteessä [2, ss. 89-120].

5. 3D-liukuhihnalle tehtyjen ohjelmien käyttäminen

Tekstimuotoon kirjoitettujen geometria- ja pikseliohjelmien lataaminen suoritettavaan pääohjelmaan voidaan suorittaa kahdella tavalla. Käytetään tästä lähtien geometria- ja pikseliohjelmista yhteisnimitystä 3D-ohjelma. Ensimmäinen vaihtoehto on käyttää assembler-kääntäjää, ja linkittää käännetty ohjelma pääohjelmaan käytetyn ohjelmointirajapinnan metodeilla. Toinen tapa on kirjoittaa 3D-ohjelma suoraan pääohjelman lähdekoodiin, ja käyttää ohjelmointirajapinnan metodia kääntämään 3D-ohjelma pääohjelman suoritusaikana. Tarkastellaan seuraavaksi yksinkertaisen geometriaohjelman käyttöönottoa Directx 8.1 -rajapinnan avulla [4]. Koodikatkelman 3 kuvaama geometriaohjelma piirtää näytölle yhden pisteen.

```
const char Point[] = \n”
    "vs.1.1\n”
   "dp4 oPos.x, v0, c4 \n”
   "dp4 oPos.y, v0, c5 \n”
   "dp4 oPos.z, v0, c6 \n”
   "dp4 oPos.w, v0, c7 \n”
    "mov oD0, c8 \n”
```

Koodikatkelma 3. Pisteen piirto näytölle.

Koodikatkelman 3 mukainen merkkijonotaulukko kirjoitetaan pääohjelman sisään, josta ohjelmointiorajapinta voi tulkita sen. Rivillä 2 kerrotaan käytettävän geometriaohjelman versionumero. Riveillä 3-6 suoritetaan jokaiselle pisteen koordinaatin komponentille pistetulo, jonka avulla piste projisoidaan kaksiulotteiselle näytölle. Pisteen projisointiin käytettävä matriisi on tallennettu vakiorekistereihin c4,…, c7. Viimeisellä rivillä rekisterissä c8 tallennettuna oleva pisteen väri-arvo sijoitetaan tulosrekisteriin o0. Vakiorekisterin lataaminen kuvataan myöhemmin. Kulmapisteiden liittämiseksi pääohjelmaan tarvitaan koodikatkelman 4 mukainen struktuuri.

```
struct VERTEX
 {
   FLOAT x, y, z;
};
```

Koodikatkelma 4. Kulmapisteen määrittely.

Koodikatkelmassa 4 kulmapisteen struktuuri muodostuu yhdestä kolmiulotteisen avaruuden vektorin määräämästä pisteestä. Kulmapisteiden attribuuttien ja niitä vastaavien syöterekistereiden vastaavuus kuvataan koodikatkelmassa 5.

```
DWORD dwDecl[] =
{
    D3DVSD_STREAM(0),
    D3DVSD_REG(0, D3DVSDT_FLOAT3)
D3DVSD_END()
};
```

Koodikatkelma 5. Kulmapisteiden attribuuttien ja syöterekisterien vastaavuus.

Koodikatkelmassa 5 jokaisen kulmapisteen paikkakoordinaattia kuvaava attribuutti liitetään syöterekisteriin V0. Rivillä 4 toisena parametrina oleva makro D3DVSDT_FLOAT3 kuvaa rekisteriin tallennettavan tiedon tyypin. Jos tallennettava tieto olisi esimerkiksi tason vektori, niin vastaava makro olisi D3DVSDT_FLOAT2 [2].

Geometriaohjelman käyttöön ottamiseksi se täytyy ensin kääntää binaariseen muotoon. Tämä voidaan suorittaa helposti DirectX 8.1 -rajapinnan metodilla D3DXAssembleShader, jonka käyttöä kuvaa koodikatkelma 6.

```
rc = D3DXAssembleShader( Point , sizeof(Point) -1,
                0 , NULL , &pVS , &pErrors );
                if ( FAILED(rc) )
  { OutputDebugString( "Compilation failed, Errors:\n" );
      OutputDebugString( (char*)pErrors->GetBufferPointer() );
           OutputDebugString( "\n" );}
```

Koodikatkelma 6.

Koodikatkelman 6 kuvaaman metodin ensimmäinen parametri on merkkijonotaulukko, joka sisältää käännettävän geometriaohjelman tekstimuotoisen kuvauksen. Toinen parametri sisältää kyseessä olevan merkkijonotaulukon koon. Kolmas parametri on tässä esimerkissä jätetty nollaksi, mutta sen avulla voidaan käännökseen liittää virheenjäljitystietoa [2]. Viides parametri sisältää osoitteen käännettyyn geometriaohjelmaan, ja kuudes parametri on osoitin merkkijonotaulukkoon, joka sisältää tiedot mahdollisesta käännösvirheestä.

Käännetty geometriaohjelma liitetään näytönohjaimeen, joka suorittaa kuvan piirtämisen näytölle. Koodikatkelma 7 asettaa näytönohjaimen syöterekisterit käyttöön. Tässä muuttuja m_pd3dDevice kuvaa osoitinta näytönohjainlaitteeseen ja muuttuja &m_VertexShader on osoitin kyseessä olevaan geometriaohjelmaan. Metodin CreateVertexShader ensimmäinen parametri sisältää koodikatkelman 5 kuvaaman kulmapisteiden attribuuttien ja niitä vastaavien rekisterien kuvauksen.

```
rc = m_pd3dDevice->CreateVertexShader( dwDecl, (DWORD*)pVS-
>GetBufferPointer(),
                &m_dwVertexShader, 0 );
        if ( FAILED(rc) )
        {
                OutputDebugString( "Failed to create the vertex shader, errors:\n" );
                D3DXGetErrorString(rc,szBuffer,sizeof(szBuffer));
                OutputDebugString( szBuffer );
                OutputDebugString( "\n" );
        }
```

Koodikatkelma 7.

Kun näytönohjaimen syöterekisterit on sidottu, voidaan geometriaohjelma asettaa pääohjelman käytettäväksi. Pääohjelma lataa aluksi näytönohjaimen geometriayksikön vakiorekisterit, ja asettaa geometriaohjelman käytettäväksi koodikatkelman 8 mukaisesti.

```
FLOAT fMaterial[4] = {0,1,0,0};
 m_pd3dDevice->SetVertexShaderConstant(8, fMaterial, 1);
 m_pd3dDevice->SetVertexShaderConstant(4, matTemp, 4);
 m_pd3dDevice->SetVertexShader( m_dwVertexShader );
```

Koodikatkelma 8.

Koodikatkelmassa 8 rivillä 1 alustetaan vihreä väri käytettäväksi kaikille kulmapisteille. Metodi setVertexShaderConstant suorittaa attribuuttien sijoittamisen vakiorekistereihin. Tämän metodin ensimmäinen parametri kuvaa käytettävän rekisterin numeroa. Toinen parametri kertoo viitteen sijoitettavaan tietoon ja viimeinen parametri kertoo tarvittavien peräkkäisten rekisterien lukumäärän. Rivillä 2 sijoitetaan syöterekisteriin c8 värin sisältävä vektori, joka mahtuu yhteen rekisteriin. Rivillä 3 sijoitetaan 4x4 matriisi rekisteriin c4, ja sille varataan neljä peräkkäistä rekisteriä, joihin jokaiseen talletetaan yksi saraketta kuvaava vektori. Tämän jälkeen geometriaohjelma on valmis pääohjelman käytettäväksi.

Koska geometriaohjelman lataamisessa varataan muistia, on se ohjelman lopettamisen yhteydessä vapautettava. Tämä suoritetaan koodikatkelman 9. mukaisesti.

```
if ( m_dwVertexShader != 0xffffffff )
{
  m_pd3dDevice->DeleteVertexShader( m_dwVertexShader );
m_dwVertexShader = 0xffffffff;
}
```

Koodikatkelma 9.Muistin vapauttaminen.

Koodikatkelmassa 9 vapautetaan geometriaohjelman tarvitsema muistialue takaisin järjestelmän käyttöön.

6. Tulevaisuuden näkymät

PC-koneisiin tarkoitettujen näytönohjaimien kehitys on tällä hetkellä yksi nopeimmin kehittyvistä tietotekniikan osa-alueista. Tässä tutkielmassa kuvattu Directx 8.1 -tasoinen 3D-liukuhihna on ensimmäinen askel kohti

grafiikan ja varsinaisen pääohjelman erottamista toisistaan. Tässä käsitelty arkkitehtuuri on tehty riittävän yksinkertaiseksi, jotta kynnys sen ohjelmoinnin aloittamiseksi ei olisi liian korkea [3]. Nyt vaikuttaa siltä, että ohjelmoijat ovat ottaneet 3D-ohjelmoinnin haasteen hyvin vastaan ja ovat valmiita kehittämään sitä eteenpäin. Hyvä esimerkki 3D-ohjelmoinnin suosiosta on Nvidian [6] kehittämä cg-kieli, joka nostaa ohjelmointikielen abstraktiotason assembler-kielestä c-tyyppiseksi helpommin luettavaksi olevaan kieleksi. Tällä hetkellä ainoastaan kulmapisteisiin perustuva ohjelmoiminen on mahdollista, mutta suunnitelmia myös suurempien kokonaisuuksien ohjelmoinnista on tehty. Tästä lisää tietoa liitteessä [6].

3D-liukuhihnan ohjelmoitavuuden mahdollistaminen on tuonut mukanaan myös ongelmia. Eri laitevalmistajat ovat kehittäneet versioita, jotka eivät ole keskenään yhteensopivia. Tämä aiheuttaa sovellusten kehittäjille lisätyötä, koska heidän on kirjoitettava ohjelmansa siten, että ne toimisivat hyvin kaikilla mahdollisilla laitealustoilla.

7. Lopuksi

Tässä tutkielmassa on käsitelty lyhyesti yksinkertaistettu ohjelmoitavan 3D-liukuhihnan arkkitehtuuri. Lisäksi on tarkasteltu mekanismia 3D-ohjelman saattamiseksi pääohjelman käyttöön. Mikäli lukija on tämän tutkielman luettuaan saanut yleiskuvan 3D-liukuhihnan toiminnasta, voi kirjoittaja suositella lisälukemista 3D-ohjelmien kirjoittamista varten. Parhaiten asiaan voi perehtyä lukemalla ACM:n Siggraph-konferenssijulkaisuja. Lisäksi laitevalmistajien kotisivuilla on nähtävillä esimerkkikoodeja, joiden avulla ohjelmointiin pääsee nopeasti sisälle. Hyviä esimerkkejä löytyy esimerkiksi Atin [1] ja Nvidian [5] developer sivuilta.

Viitteet

[1] Ati Developer. http://mirror.ati.com/developer/index.html. [14.12.02]

[2] Wolfgang F. Engel. *Direct3D ShaderX Vertex and Pixel Shader Tips and Tricks.* Wordware Publishing, Inc. July 2002.

[3] Erik Lindholm, Mark J Kilgard and Henry Moreton. A User-programmable vertex engine. *Computer Graphics (Proceedings of SIGGRAPH 2001)*, 149-158, August 2001.

[4] Microsoft Directx SDK information. http://msdn.mirosoft.com/directx. [7.11.02]

[5] NVIDIA Developer. http:/ / developer.nvidia.com. [14.12.02]

[6] Kekoa Proudfoot, William R. Mark, Svetoslav Tzvetkov and Pat Hanrahan. A Real-time procedural shading system for programmable graphics hardware., *Computer Graphics (Proceedins of SIGGRAPH 2001)*, 159-170, August 2001.
[7] Silicon Graphics OpenGL information.
http://www.sgi.com/software/opengl. [7.11.02]

Semantic Web ja agentit käytännössä

Antto Sierla

Tiivistelmä

Semantic Webistä on povattu vallankumousta Internetin hyödyntämisessä. Aihe on kuuma tutkimuskohde useilla tietojenkäsittelytieteen osa-alueilla, ja uusia teknologioita ja standardeja kehitetään koko ajan. Semantic Webin toteutumiselle on kuitenkin välttämätöntä myös käyttökelpoisten sovellusten kehittäminen. Eräs Semantic Webin olennainen sovellusalue on agenttisovellukset. Tämän tutkielman tarkoituksena on auttaa agenttisovellusten kehittäjiä tarjoamalla katsaus teknologioiden ja työkalujen nykytilanteeseen. Motivoivana esimerkkinä tarkastellaan Semantic Web -teknologioiden soveltamista yliopistojen tarpeisiin.

Avainsanat ja -sanonnat: Semantic Web, ontologiat, OWL, DAML+OIL, agentit, RDF, Web Services.
CR-luokat: H.3.5, H.5.3, I.2.11

1. Johdanto

Internetissä olevan tiedon määrä on valtava ja se kasvaa koko ajan. Uusimpien tutkimusten mukaan julkisia staattisia WWW-sivuja on arvioitu olevan noin 3 miljardia kappaletta [Patel-Schneider and Fensel, 2002]. Lisäksi ei-julkista tietoa on olemassa sähköisessä muodossa yritysten ja muiden organisaatioiden sisäisissä verkoissa. Halutunlaisen tiedon löytämisen ja useista eri lähteistä saadun tiedon yhdistämisen automatisointi on kuitenkin usein hankalaa. Koska tieto on yleensä esitetty luonnollisella kielellä, sen merkityksen päätteleminen ohjelmallisesti on hyvin vaikeaa.

Näin suuren tietomäärän tehokas hyödyntäminen vaatisi nimenomaan tiedonhaun automatisointia. Semantic Web -hankkeen tavoite on tehdä tämä mahdolliseksi tarjoamalla välineitä tiedon esittämiseen tietokoneohjelmille ymmärrettävässä muodossa.

Jotta Semantic Web ei jäisi vain tutkijoiden utopistiseksi visioksi, on tärkeää, että kehitettyjä standardeja ja teknologioita sovelletaan myös käytännössä. Koska nämä ovat vielä aktiivisen tutkimuksen ja kehityksen kohteena, sovellusten kehittäjien voi olla vaikea päättää, mitä standardeja ja työkaluja heidän tulisi käyttää. Tämän tutkielman tarkoitus on auttaa

erityisesti agenttisovellusten kehittäjiä päätösten tekemisessä tarjoamalla kuva Semantic Webin ja siihen liittyvien työkalujen nykytilasta.

Luvussa 2 esitellään Semantic Web -hankkeen taustoja ja tavoitteita. Luku 3 käsittelee hankkeen hyötyjä erityisesti agenttiohjelmistojen toteuttamisessa. Luku 4 luo katsauksen nykyisin tarjolla oleviin työkaluihin. Luku 5 tarjoaa esimerkin siitä, miten esiteltyjä tekniikoita voisi hyödyntää yliopistomaailmassa. Lopuksi luvussa 6 on yhteenveto tutkielman tuloksista ja pohdintaa Semantic Webin tulevaisuudesta.

2. Mistä on Semantic Web tehty?

Semantic Web –hankkeen isänä voidaan pitää Tim Berners-Leetä, joka esitteli visionsa WWW-artikkkelissaan 1998 [Berners-Lee, 1998]. Tällä hetkellä pääkehitysvastuu on World Wide Web Consortiumilla (W3C). Kuten edellä on todettu, tavoitteena on nykyisen internetin laajentaminen niin, että tieto olisi sekä koneiden että ihmisten helposti käsiteltävissä.

Semantic Webin voidaan katsoa koostuvan useasta teknologiakerroksesta niin, että alemman tason teknologiat tarjoavat perustan ylemmille tasoille. Koska tavoitteena on mahdollisimman laaja ja joustava käyttö, perusteknologiaksi on valittu XML (eXtensible Markup Language). XML on kuvauskieli, jolla on mahdollista määritellä erilaisten tekstimuotoisten dokumenttien rakenne. XML valittiin, koska se on laajasti käytössä oleva standardi ja sen käsittelyyn on olemassa runsaasti tehokkaita työkaluja.

XML ei kuitenkaan ole suunniteltu dokumenttien sisällön *merkityksen* kuvaamiseen. Tätä tarkoitusta varten W3C on kehittänyt RDF:n (Resource Definition Framework), jolle on määritelty XML-esitysmuoto (RDF / XML).

Vaikka RDF on riittävä merkityksen kuvaamiskieli useissa tapauksissa, on ilmennyt tarve vielä ilmaisuvoimaisempien kuvaamismenetelmien kehittämiselle. Niin sanottujen *ontologioiden* (ontology) kuvaamiseksi on kehitetty ja kehitteillä useita kieliä. Niistä uusimmat ovat W3C:n kehittämä OWL ja DAML+OIL.

Seuraavassa tarkastellaan edellä mainittuja Semantic Webin rakennuspalikoita tarkemmin.

2.1. Metadata

Standardoitu tapa merkityksellisen tiedon esittämiseksi on välttämätön perusta Semantic Webille. Yksi hankkeen tavoitteista on ollut kehittää standardi WWW-dokumenttien sisältöä koskevan tiedon, *metadatan* esittämiseen. Metadatan hyödyllisyys Semantic Webin kannalta on sen

mahdollistama tehokas tiedonhaku. Nykyiset WWW-hakukoneet etsivät annettuja avainsanoja dokumentin koko tekstisisällöstä. Haku palauttaa kaikki dokumentit, joissa kyseisiä sanoja esiintyy, riippumatta niiden mahdollisesti erilaisista merkityksistä kussakin dokumentissa. Mikäli dokumentit olisi varustettu metadatalla, voisi hakuja rajoittaa esimerkiksi tekijän nimen perusteella. Tällöin haku ei palauttaisi sellaisia dokumentteja, joissa annettu nimi esiintyisi jossakin muussa merkityksessä.

W3C:n kehittämä RDF (Resource Description Framework) on kieli WWW-tietolähteiden kuvaamiseen. Sen avulla voidaan esittää tietoa mistä tahansa asiasta, joka on mahdollista identifioida Uniform Resource Identifierilla (URI). Tällaisia asioita ovat WWW-sivujen lisäksi vaikkapa yritykset, henkilöt tai tuotteet.

Tieto esitetään RDF:ssä ns. *kolmikkoina* (triple), jotka muodostuvat *subjektista, predikaatista* ja *objektista*. Subjekti on kohde, jota kuvataan, esimerkiksi WWW-sivu. Predikaatti on jokin kohteen ominaisuus, esimerkiksi otsikko. Objekti on predikaatin ilmaiseman ominaisuuden arvo, esimerkiksi ”Semantic Web Road Map”. RDF/XML-muodossa tämä kolmikko näyttää seuraavalta:

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

            xmlns:dc="http://purl.org/dc/elements/1.1/">

    <rdf:Description rdf:about="http://www.w3c.org/DesignIssues/Semantic">

           <dc:title>Semantic Web Road map</dc:title>

    </rdf:Description>

</rdf:RDF>
```

Predikaatti voi olla merkkijonon sijasta myös URI-viittaus johonkin vaikkapa tietyn henkilön identifioivaan resurssiin.

RDF Schema laajentaa tiedonkuvauksen mahdollisuuksia tarjoamalla keinot määritellä uusia tietotyyppejä, luokkia, luokkien ominaisuuksia ja niiden välisiä suhteita. Semantic Webin edistyneemmät sovellukset tarvitsevat vielä tarkemmin määriteltyä tietoa. Lisäksi tieto on voitava esittää sellaisessa muodossa, että siitä voidaan automaattisesti tehdä päätelmiä logiikan menetelmin. Tähän tarkoitukseen on kehitetty niin sanottuja *ontologiakieliä*.

2.2. Ontologiat ja päättely

Ontologialla tarkoitetaan Semantic Webin yhteydessä tiettyyn aihepiiriin liittyvien käsitteiden ja niiden välisten semanttisten suhteiden formaalia kuvausta. Lisäksi ontologian voidaan katsoa sisältävän käsitteitä koskevia yksinkertaisia päättelysääntöjä [Hendler, 2001].

Ontologioiden kuvaukseen on olemassa useita kieliä, joiden esitysvoima vaihtelee. W3C:n Web Ontology Working Group kehittämä OWL (Ontology Web Language) pyrkii täyttämään yleiskäyttöisen WWW-käyttöön tarkoitetun ontologiakielen vaatimukset. OWL pohjautuu DAML+OIL:iin, joka taas on kehitetty yhdistelemällä useiden aikaisempien ontologia- ja logiikkakielien ominaisuuksia [Mcguinness et al., 2002].

Uusimman spesifikaation [Smith et al., 2002] mukaan OWL laajentaa RDF:n ja RDF Scheman ilmaisuvoimaa muun muassa tarjoamalla:

- paremmat keinot määritellä ominaisuuksien piirteitä, kuten symmetrisyys, transitiivisuus tai arvojen tarkka lukumäärä;
- mahdollisuuden määritellä luokkia käyttämällä joukko-operaattoreita (yhdiste, leikkaus ja erotus);
- mahdollisuuden määrittää, että kaksi luokkaa ovat alkiovieraat (disjoint), eli yksilö ei voi olla molempien ilmentymä.

Lisäksi OWL tukee ontologioiden yhdistämistä ja uusiokäyttöä tarjoamalla:

- mahdollisuuden määrittää, että kaksi eri skeemoissa määriteltyä luokkaa tai ominaisuutta tarkoittavat samaa;
- mahdollisuuden määrittää, että kaksi erillään määriteltyä instanssia kuvaavat samaa yksilöä;
- tuen ontologioiden versioinnille;
- tuen toisaalla määriteltyjen ontologioiden sisällyttämiseen uuteen ontologiaan.

Jotta ontologioita voitaisiin käyttää tehokkaasti hyödyksi, on tärkeää, että niistä on mahdollista ohjelmallisesti päätellä sellaisia asioita, jotka eivät ole eksplisiittisesti annettuja. Esimerkiksi jos on määritelty, että luokat Mies ja Nainen ovat alkiovieraat ja on annettu luokan Mies instanssi Matti, voidaan päätellä, että väite ”Matti on Nainen” on epätosi.

Päättelyn mahdollistamiseksi ontologiakielet pyritään rakentamaan niin, että niillä esitetyille lauseille on annettu yksiselitteinen, formaali merkitys. Tavoite voidaan saavuttaa useilla tavoilla, joista esim. DAML+OILin kehittäjät ovat valinneet kuvauksen *ensimmäisen kertaluvun logiikassa* (first-order logic) [Mcguinness et al., 2002]. Tämä ratkaisu mahdollistaa olemassa

olevien päättelykoneiden käytön halutun tiedon hakemiseen ja myös ristiriitaisuuksien etsimiseen.

3. Semantic Web ja agentit

Yhtenä tärkeimmistä Semantic Webin sovelluksista pidetään agenttiohjelmia [Berners-Lee et al., 2001]. Agenteille ei ole olemassa yleisesti hyväksyttyä määritelmää, mutta tässä tutkielmassa agenteilla tarkoitetaan itsenäisesti toimivia ohjelmia, jotka toimivat ihmiskäyttäjästä riippumattomasti suorittaen niille annettuja tehtäviä. WWW:tä hyödyntäville agenteille on tärkeää, että ne voivat tulkita löytämänsä tiedon merkityksen, saada tietoa erilaisista lähteistä ja mahdollisesti myös kommunikoida toisten agenttien kanssa. Semantic Webin tavoitteet sopivat siis hyvin yhteen näiden tarpeiden kanssa. Seuraavassa käydään läpi ne sovellusalueet, joissa Semantic Webin tarjoamista välineistä on hyötyä agenttiohjelmistojen rakentamisessa.

3.1. Hakuagentit

Yksi agenttiohjelmien laajimmalle levinneistä käyttötavoista on niiden hyödyntäminen tiedonhaussa. Nykyiset hakukoneet muodostavat tietokantansa käyttämällä agentteja, jotka “ryömivät” internetissä sivulta toiselle hyperlinkkien välityksellä ja indeksoivat dokumenteista löytämänsä sanat käyttäen *kokoteksti-indeksointimenetelmää* (full-text indexing).

Semantic Web mahdollistaa tehokkaampien hakujen tekemisen. Jo pelkästään yksinkertaisen metadatan liittäminen WWW-dokumentteihin mahdollistaa tarkempien hakujen tekemisen, mikäli metadatassa esiintyville käsitteille on määritelty standardoitu merkitys. Esimerkiksi Dublin Core Metadata Initiative [DC Metadata Initiative, 2002] pyrkii tällaisen standardin kehittämiseen.

Hakuagentit voivat käyttää hyväksi metadataa löytämiensä dokumenttien luokittelussa jäsentämällä dokumenttiin liitetyt tietoelementit. Hakupalvelun käyttäjä voi tällöin rajoittaa hakua haluamiensa kriteerien perusteella. Tällaisia kriteerejä voivat olla esimerkiksi dokumentin tekijä, julkaisija tai käsiteltävä aihe.

Hakuagenttien voidaan katsoa kuuluvan ns. *reaktiivisiin agentteihin*. Reaktiiviset agentit ottavat vastaan syötteitä ja suorittavat tehtäviä niiden sisällön perusteella, mutta niillä ei ole varsinaisesti ymmärrystä ympäristöstä, jossa ne toimivat [Nwana, 1996]. Monimutkaisempien tehtävien, kuten Berners-Leen ja muiden [2001] kuvaaman lääkäriajan varauksen suorittamiseen tarvitaan agentteja, jotka ymmärtävät enemmän ympäristöstään.

3.2. Älykkäät agentit

Älykkäiksi agenteiksi voidaan kutsua sellaisia agentteja, jotka osaavat suorittaa monimutkaisia tehtäviä käyttäen hyväksi tietämyskantoja soveltaen loogisia päättelysääntöjä ja mahdollisesti myös kommunikoimalla muiden agenttien kanssa. Älykkäitä agentteja voidaan vastakohtana reaktiivisille agenteille kutsua *proaktiivisiksi* [Nwana, 1996]. Proaktiivisilla agenteilla on sisäinen malli ympäristöstään ja sen tilasta. Tällaiset agentit havainnoivat ympäristöään ja muuttavat sisäistä kuvaustaan ympäristön muuttuessa. Yleensä älykkäät agentit ovat kuitenkin hybridejä, jotka toimivat sekä reaktiivisesti että proaktiivisesti. Semantic Webin toteutuminen avaa aivan uudenlaisia näkymiä tällaisten agenttien kehittämiseen.

Sen lisäksi, että ontologioita voidaan käyttää agenttien omien tietovarastojen perustana, ontologiat ovat hyödyksi myös eri tietolähteistä saadun tiedon yhdistämisessä. Jaetut standardimuodossa esitetyt ontologiat tarjoaisivat agenteille mahdollisuuden käyttää hyväksi sellaista tietoa, jota ei olisi erityisesti suunniteltu juuri kyseisen agentin käytettäväksi. Tähän pyritään rakentamalla ontologiakieliin keinoja eri ontologioissa eri tavalla esitettyjen käsitteiden merkitysten linkittämiseen. Standardoidut ontologiakielet helpottavat tällaisten ontologioiden kehittämistä ja ylläpitoa mahdollistaen yleiskäyttöisten työkalujen kehittämisen.

Ontologioita voidaan hyödyntää myös agenttien välisessä kommunikaatiossa. Järkevä keskustelu edellyttää käytettävien käsitteiden merkityksen yhdenmukaista määrittelyä keskustelijoiden kesken. Yksittäisten käsitteiden lisäksi täytyy voida määritellä myös niiden väliset suhteet ja mahdollisten kyselyjen merkitys. Agentti voi esimerkiksi tiedustella toiselta, onko jokin väite sen tietämyskannan mukaan tosi. Kommunikaatio voi olla myös monimutkaisempaa, kuten tietyn palvelun, vaikkapa ajanvarauksen tai tavaratilauksen tekeminen.

Erityisesti ontologioiden esittäminen yleisessä muodossa ja niiden laaja saatavuus mahdollistaa uusien käsitteiden omaksumisen ja maksimaalisen hyödyn. Näiden tavoitteiden saavuttamisessa pyrkii auttamaan mm. FIPAn (Foundation for Intelligent Physical Agents) Ontology Agent Specification [FIPA, 2001]. Tässä määrityksessä ehdotetaan erityisen *ontologia-agentin* (Ontology Agent) käyttämistä. Tällainen agentti tarjoaisi muille agenteille seuraavat palvelut:

- julkisten ontologioiden paikantaminen,
- tällaisten ontologioiden ylläpitäminen,

- eri ontologioissa tai sisältökielissä annettujen ilmausten kääntäminen toiselle,
- eri ontologioissa määriteltyjen käsitteiden vastaavuuksien selvittäminen ja
- kommunikaatiossa käytettävän jaetun ontologian tunnistamisessa avustaminen.

Lisäksi määritellään protokolla ja sanasto, joiden avulla agentit voivat hyödyntää näitä palveluja.

Koska älykkäät agentit toimivat ihmiskäyttäjän edustajina, ne tarvitsevat tietoa käyttäjästään. Agentit pitävät yllä *käyttäjäprofiilia*, joka voi perustua sekä käyttäjän itsensä antamiin että käyttäjää tarkkailemalla saatuihin tietoihin. Ontologioita voidaan hyödyntää myös tässä tehtävässä. Käyttäjäontologiat voidaan rakentaa käyttäen standardeja kieliä niin, että niistä on mahdollista loogisesti päätellä erilaisia asioita käyttäjistä. Lisäksi tällaisia ontologioita voidaan jakaa internetissä, jolloin erilaiset profilointia hyödyntävät agentit ja palvelut voivat käyttää hyväkseen toisten keräämää tietoa.

Kuten Heflin et al. [2002] toteavat, eräs hyödyllinen ominaisuus yleiskäyttöisessä WWW-ontologiakielessä on tuki *digitaalisille allekirjoituksille* (digital signature). Digitaalisten allekirjoitusten avulla voidaan varmistua siitä, että WWW-lähteestä saatu tieto on jonkin luotettavan tahon julkaisemaa. Keinot tietolähteen tai palvelun luotettavuuden selvittämiseen ovat välttämättömiä monimutkaisia tehtäviä ihmisen puolesta suorittavien agenttien toteutukselle. Esimerkiksi agentit, jotka suorittavat varainsiirtoja päästäkseen käyttämään jotakin WWW-palvelua, tarvitsevat varmuuden siitä, että taho, joka palvelun tarjoaa, on luotettava [Hendler, 2001]. Tällainen varmistus voidaan hankkia erityisiltä *luottamuspalvelimilta*, jotka pitävät yllä listaa luotetuista palvelun-tarjoajista. Luottamussuhteita voitaisiin solmia myös agenttien välille, jolloin luotetulta agentilta hankittua tietoa voidaan käyttää riippumatta siitä, mikä tiedon alkuperä on.

3.3. Palvelut

Erilaisten WWW:ssä tarjottavien palvelujen, kuten lipunvaraus-, tai vaikkapa säätietopalvelujen automatisointi on Hendlerin [2002] mukaan yksi potentiaalisimmista Semantic Web -ontologioiden käyttökohteista. Tällaisia palveluita on nykyisinkin tarjolla varsin kattavasti, mutta ongelmana on niiden käyttötapojen standardoimattomuus. Tästä johtuen palvelujen käyttö

edellyttää yleensä ihmisen osallistumista tai palvelua käyttävän ohjelman rakentamista ainoastaan kyseistä palvelua varten.

Näiden palveluiden täysi hyödyntäminen vaatii standardoidut keinot seuraavien tehtävien automatisointiin [DAML-S Coalition, 2002]:

- *Palveluiden löydettävyydellä* tarkoitetaan halutun tyyppisten palvelujen paikantamista.
- *Palvelujen käytön kuvaaminen* tarkoittaa palvelun käyttöprosessin ja sen vaatimien syötteiden ja tulosteiden määrittelemistä.
- *Palvelujen yhdistäminen* mahdollistaa useiden erilaisten palvelujen käyttämisen sellaisten tehtävien suorittamiseen, joita mikään yksittäinen palvelu ei toteuta.
- *Palvelujen tilan kysely* voi olla tarpeen tilanteissa, joissa palvelun suorittaminen kestää kauan tai sisältää useita vaiheita.

Kaikkien yllämainittujen toimintojen automatisoinnissa voidaan hyödyntää tarkoitusta varten kehitettyjä ontologioita, jotka sisältävät keinot kuvata toiminnot tietokoneille ymmärrettävässä muodossa. Esimerkiksi palvelujen käytön kuvaamisessa ontologioita voidaan Hendlerin [2001] mukaan hyödyntää mallintamalla palvelun käyttöprosessia *äärellisen tilakoneen* (finite-state machine) avulla. Tässä tapauksessa perusontologia sisältäisi mm. luokat tila (State) ja linkki (Link). Tilalla olisi alaluokat alkutila (StartState) ja lopputila (EndState). Ontologia määrittelisi myös, että linkki toimii suunnattuna siirtymänä kahden tilan välillä ja jokaisella tilalla olisi identifioiva tunniste, kuten URI. Palvelun tarjoajat voisivat tätä ontologiaa hyväksikäyttäen asettaa tilojen välisille siirtymille esiehtoja ja määritellä muita palvelulle tyypillisiä ominaisuuksia. Näiden tietojen avulla palvelua käyttävä agentti voisi päätellä, mitä tietoja ja missä järjestyksessä sen tulee tarjota päästäkseen alkutilasta lopputilaan, eli käyttääkseen palvelua menestyksekkäästi halutun tehtävän suorittamiseen.

4. Työkalut

Semantic Webin tarjoamia teknologioita agenttien toiminnan tehostamiseen on hyvin vaikeaa lähteä hyödyntämään puhtaalta pöydältä. Koska näitä teknologioita kehitettäessä on otettu huomioon yhteensopivuus aiempien, laajalle levinneiden standardien kanssa, sovellusten kehittäjälle on tarjolla useita valmiita työkaluja. Seuraavassa esitellään muutamia hyödyllisiä apuvälineitä agenttiohjelmistojen kehittäjille. Valintakriteerinä on käytetty työkalujen tukea uusimmille standardeille ja niiden perustumista Java-teknologiaan sen alustariippumattomuuden vuoksi.

4.1. Metadatan käsittely

RDF-muodossa esitetyn tiedon ohjelmalliseen käsittelyyn tarkoitettuja jäsentäjiä voidaan käyttää apuna esimerkiksi metadatan erottamiseen WWW-dokumentista. Niitä voidaan hyödyntää myös RDF-pohjaisten tietovarastojen hallinnassa.

ICS-FORTH Validating RDF Parser (VRP) on tehokas ja ajantasainen työkalu RDF-dokumenttien jäsentämiseen ja validointiin [ICS-FORTH, 2002]. VRP:n jäsennin tukee uusinta RDF-syntaksimääritystä ja validaattori uusinta RDF Schema -määritystä. Työkalu sisältää sekä komentorivipohjaisen käyttöliittymän että Java-ohjelmointirajapinnan. Tuki HTML:n sekaan upotetun RDF:n lukemiseen on sisäänrakennettu. Ohjelmointirajapinnan avulla voidaan RDF-dokumentista muodostaa ns. *dokumenttimalli* (document model), jota on mahdollista helposti käsitellä (esimerkiksi lisätä väitteitä tai tutkia ominaisuuksien arvoja).

Toinen RDF:n käsittelyyn tarkoitettu työkalu on Jena [McBride et al., 2002]. Sen uusin versio sisältää uusinta määrittelyä tukevan jäsentäjän, mutta ei validaattoria. Jena tarjoaa RDF-mallien ohjelmalliseen käsittelyyn kaksi tapaa: *lausepohjaisen* ja *resurssipohjaisen*. Lausepohjaisessa tavassa käsiteltävänä on yksittäisiä subjektin, predikaatin ja objektin sisältämiä lauseita. Resurssipohjainen käsittely perustuu resursseihin ja niille annettuihin ominaisuuksiin. Jenaan sisältyy myös yksinkertainen kyselykieli RDQL, jonka avulla voidaan tehdä SQL-tyyppisiä kyselyjä RDF-mallin sisältämään tietoon. RDQL on kuitenkin tarkoitettu vain mallissa suoraan annetun tiedon hakemiseen, se ei sisällä päättelyominaisuuksia.

4.2. Ontologiat, tietämyskannat ja päättelykoneet

Varsinkin laajojen ontologioiden luominen ja ylläpitäminen ilman apuvälineitä on erittäin hankala tehtävä. Seuraavassa esitellään muutamia työkaluja ontologioiden luomiseen, editointiin, versionhallintaan ja yhdistämiseen. Lisäksi tutustutaan päättelytyökaluihin, joiden avulla on mahdollista automatisoida ontologioihin tallennetun tiedon hyödyntäminen.

Protégé-2000 tarjoaa helpon käyttöliittymän ontologioiden ja niihin perustuvien tietämyskantojen hallintaan [Noy et al., 2001]. Sen avulla on mahdollista luoda luokkia, määritellä luokkien välisiä suhteita ja luokkien ominaisuuksia. Lisäksi sillä voidaan rakentaa ontologioihin perustuvia tietämyskantoja luomalla määriteltyjen luokkien instansseja. Tällä hetkellä Protégé-2000 sisältää tuen RDF Schema -muotoisten ontologioiden lukemiseen ja tallentamiseen. DAML+OIL-tukea ollaan parhaillaan kehittämässä. Jo

nykyisessä versiossa on mahdollisuus käsitellä OIL-ontologioita erillisen lisäosan avulla. Työkalu tukee melko suurta osaa OIL:n syntaksista. Myös päättelyominaisuudet on mahdollista sisällyttää lisäosaa käyttämällä. Päättelyyn käytetään seuraavassa alaluvussa (3.3.) kuvattavaa JESS:tä.

Kaupallinen työkalu OntoEdit omaa pitkälti samat ominaisuudet kuin Protégé-2000 [Ontoprise Ltd., 2002]. Lisäksi siinä on jo nyt tuki DAML+OILin käytölle. Myös OntoEditiin on tarjolla runsaasti lisäosia, mm. päättelyn mahdollistava OntoBroker.

Vapaasti levitettävä OilEd on edellisiä yksinkertaisempi, mutta silti käyttökelpoinen ontologiaeditori [Bechhofer et al., 2001]. Se ei tarjoa täyttä tukea laajojen ontologioiden hallinnalle, kuten versioinnille, mutta sen avulla on kuitenkin mahdollista kehittää DAML+OIL-ontologioita. OilEd sisältää myös FaCT-päättelykoneen, jonka avulla voidaan etsiä ontologioiden sisältämiä ristiriitaisuuksia ja piilotettuja luokkasuhteita.

4.3. Päättelykoneet

Älykkäiden agenttiohjelmistojen toiminta perustuu usein tietämyskantaan ja joukkoon *päättelysääntöjä*, jotka määrittelevät tietämyskannan sisällön perusteella laukaistavia toimintoja. Tällaisten ohjelmien kehittäjälle ehkä hyödyllisin työkalu on Java Expert System Shell (JESS) [Friedman-Hill, 2002]. Sen käyttökelpoisuus perustuu tehokkaaseen, Java-pohjaiseen ohjelmointi-rajapintaan sekä mahdollisuuteen yhdistää se muiden hyödyllisten työkalujen, kuten luvussa 4.2. mainitun Protégé-2000:n ja seuraavassa alaluvussa esiteltävän JADE-agenttikehyksen kanssa.

JESS-kielen peruskäsitteet ovat *faktat* (facts) ja *säännöt* (rules). Tietämyskanta muodostuu faktoista, jotka ovat aihepiiriä koskevia tosia väitteitä. JESS:ssä voidaan määrittää myös luokkia vastaavia tietorakenteita, joita kutsutaan templateiksi. Säännöt ovat kuvauksia siitä, miten tiettyjen faktojen toteutumiseen reagoidaan. Tällaista päättelyä kutsutaan ns. *eteenpäin ketjutukseksi (*forward-chaining). JESS tukee myös ns. *taaksepäin ketjuttavaa* -päättelyä (backward-chaining), jossa järjestelmä pyrkii etsimään ratkaisua etsimällä sääntöjä, joiden laukeaminen aiheuttaisi jonkun toisen säännön laukeamisen ja johtaisi lopulta halutun tilan toteutumiseen. Tämän tiedon perusteella voidaan sitten muuttaa faktoja niin, että haluttu tapahtumaketju toteutuu. Lisäosan avulla JESS:in on lisättävissä myös sumean logiikan tuki. Toinen hyödyllinen lisäosa mahdollistaa Protégé-2000:n käyttämisen tietämyskantana niin, että sen avulla syötettyä tietoa tarkastellaan JESS-faktoina.

4.4. Agenttialustat ja rajapinnat

Monimutkaisten, agenttienvälistä kommunikointia tukevien agenttiohjelmistojen kehittäminen on hyvin työlästä ilman apuvälineitä. JADE (Java Agent Development Framework) on *moniagenttijärjestelmien* (multi-agent system) kehittämiseen suunniteltu ohjelmointikehys [Bellifemine et al., 2002]. Se tukee suurta osaa FIPA:n älykkäiden agenttien määrityksistä. JADE tarjoaa myös FIPA-yhteensopivan agenttialustan, joka sisältää agenttien hallinnan (AMS), hakemistopalvelun (DF) ja agenttien välisen kommunikaatiokanavan (ACC). Tarjolla on myös useita virheiden etsintää tukevia työkaluja.

Semantic Webiin liittyen JADE:ssa agenttien välisen kommunikoinnin mahdollistamiseksi on määriteltävä ontologiat, jotka kuvaavat keskustelussa käytettävät käsitteet. JADE tarjoaa välineet ontologioiden luomiseen Java-rajapinnan kautta. Ontologiat käsitteitä vastaavat Java-luokat, joiden jäsenmuuttujat edustavat luokkien ominaisuuksia. Luokkahierarkia heijastaa ontologian käsitteiden välisiä suhteita. Käsitteille ja ominaisuuksille on mahdollista määritellä *rajoitteita* (facets), joiden noudattamisesta JADE huolehtii automaattisesti. Laajojen ontologioiden luomista helpottamaan on olemassa Protégé-2000:n liitettävä lisäosa, joka mahdollistaa ontologian luomisen graafisen käyttöliittymän avulla ja JADE:n tarvitsemien luokkien automaattisen generoimisen. [Caire, 2002]

Päättelyn toteuttamiseksi on lisäosan avulla mahdollista käyttää JESS:n sääntöpohjaista päättelykonetta. Agentit voivat asettaa faktoja JESS:in tietämyskantaan ja toisaalta voidaan toteuttaa JESS-sääntöjä, jotka käynnistävät agenttien toimintoja. Myös JADEn, Protégé-2000:n ja JESS:n yhteiskäyttö on mahdollista. Käyttämällä kaikkia kolmea yhdessä on mahdollista luoda hyvin monipuolisia älykkäitä agenttijärjestelmiä. JADE:sta on olemassa myös kannettaviin päätelaitteisiin soveltuva versio, LEAP.

Toinen hyödyllinen agenttikehys on FIPA-OS, joka tarjoaa paljolti samat ominaisuudet kuin JADE [Emorphia Ltd, 2002]. JESS-tuki on siinä sisäänrakennettuna ja se tarjoaa myös rajapinnat erilaisten tietokantojen hyödyntämiseen. Uusien ohjelmaversioiden ilmestymistiheydestä ja lisäosien määrästä päätellen näyttää kuitenkin siltä, että FIPA-OS on jäämässä kehityksessä jälkeen JADE:sta. JADE:n viimeisin versio ilmestyi syyskuussa 2002, kun taas FIPA-OSia on viimeksi päivitetty vuoden 2001 puolella. FIPA-OS on kuitenkin kokeilemisen arvoinen työkalu, eikä sen kehitystyötä ole ainakaan virallisesti lopetettu, joten kannattaa tarkkailla uusien versioiden ilmestymistä.

5. Esimerkki yliopistomaailmasta

Kuten mikä tahansa uusi teknologia, myös Semantic Webiin liittyvät teknologiat, tarvitsevat toimivia ja laajan suosion saavuttavia sovelluksia tehdäkseen varsinaisen läpimurtonsa. WWW:kin oli aluksi vain pienen tutkijayhteisön käytössä. Semantic Web ei sekään tule heti leviämään koko WWW:n laajuiseksi ja aluksi sitä voidaankin soveltaa erilaisten suppeampien yhteisöjen ja verkostojen keskuudessa. Esimerkiksi yliopistot voivat hyötyä Semantic Webistä monin tavoin. Seuraavassa tarkastellaan muutamia mahdollisia sovelluskohteita.

Lukukauden alussa opiskelija joutuu yleensä käyttämään huomattavasti aikaa lukujärjestyksen laatimiseen. Kurssien tietoja täytyy etsiä monesta eri paikasta, kuten eri tiedekuntien WWW-sivuilta tai ilmoitustauluilta. Samoin kurssien valinta niin, että luentojen ja harjoitusryhmien päällekkäisyyksiä olisi mahdollisimman vähän, on vaikeaa. Näiden tehtävien automatisointi on nykyisin useimmissa tapauksissa hyvin hankalaa, ellei mahdotonta, koska tietoa ei ole esitetty standardoidussa, tietokoneille ymmärrettävässä muodossa.

Ontologioita hyödyntämällä tehtävän automatisointi on mahdollista toteuttaa. Aluksi on suunniteltava ja toteutettava tarvittavat käsitteet määrittelevä ontologia. Käsitteitä (luokkia) voisivat olla esimerkiksi Opiskelija, Opettaja, Kurssi, Luento, Opetustila, Luentosali, jne. Luokkien ominaisuuksista ja niiden välisistä yhteyksistä voidaan mainita vaikkapa seuraavat: opiskelijalla on nimi ja opiskelija numero, luento järjestetään opetustilassa ja luentosali on opetustilan alaluokka. Ontologian rakentamisessa voidaan käyttää apuna esimerkiksi Protégé-2000:a.

Seuraavassa vaiheessa kurssien järjestäjien ja muiden tahojen käyttöön annetaan työkalut (joko valmiit, kuten Protégé-2000 tai käyttötarkoitukseen räätälöidyt), joilla he voivat syöttää vastuullaan olevia tietoja perustuen luotuun ontologiaan. Nämä tiedot voidaan työkalun avulla tallentaa esimerkiksi RDF- tai DAML+OIL-muotoon ja julkaista WWW:ssä.

Tämän jälkeen on mahdollista esimerkiksi JADE:n avulla toteuttaa agenttiohjelma, joka näitä tietoja hyväksikäyttäen auttaa opiskelijaa lukujärjestyksen muodostamisessa. Agentti voisi opiskelijan aiempiin opintoihin ja opetussuunnitelmaan perustuen ehdottaa sopivia kursseja tai toimia kokonaan opiskelijan valitsemien kurssien perusteella. Se voisi sovittaa annetuista kursseista optimaalisen lukujärjestyksen ja tallentaa sen WWW-sivulle sekä RDF/DAML+OIL että HTML muotoisena. Lisäksi agentti voisi

myös tarkkailla kurssien sivuja muutoksien varalta ja päivittää lukujärjestystä niiden mukaisesti.

Semantic Webin tarjoamat menetelmät tiedonhallintaan tuovat lisäarvoa myös yliopistoille. Yliopistojen WWW-sivuilla ja intranetissä on tarjolla monenlaista tietoa opiskelemaan pyrkiville, opiskelijoille ja tutkijoille. Kehittämällä sopiva metadataontologia tämän tiedon kategorisointiin ja upottamalla metadata dokumentteihin on mahdollista tehostaa tiedon hakua. Näiden toimenpiteiden jälkeen on mahdollista kehittää hakuagentti, joka kerää ja luokittelee tiedon ja tallentaa sen tietokantaan. Tätä tietokantaa käyttävä hakupalvelu voi sitten palvella tiedon tarvitsijoita. Tällaisten ontologioiden jakaminen mahdollisimman monien yliopistojen kesken tarjoaa mahdollisuuden yhä parempien hakupalvelujen toteuttamiseen.

Tieteenalojen sisällä on yleensä käytössä vakiintunut käsitteistö. Tällaisen käsitteistön rakentaminen ontologiaksi, joka voidaan jakaa tiedeyhteisön käyttöön, olisi monella tapaa hyödyllistä. Tiedon haettavuus ja tiedonvaihto helpottuisi huomattavasti. Lisäksi tällaisiin ontologioihin perustuen voidaan rakentaa laajoja tietämyskantoja, joita voidaan käyttää asiantuntija-järjestelmien tietovarastona. Esimerkiksi lääketieteessä tällaiset järjestelmät voivat olla apuna diagnoosien tekemisessä.

Yliopistomaailman ulkopuolisille tahoille tarjottavista palveluista voidaan mainita esimerkkinä opintojen etenemisen tarkkailu. Opintotukea myönnetään vain riittävän määrän opintoja lukukaudessa suorittaneille opiskelijoille. Yliopistot voisivat tarjota WWW-palveluja, joiden avulla opintotuesta päättävät tahot voisivat automaattisesti tarkkailla opintojen etenemistä. Tällainen palvelu tulisi tietenkin toteuttaa niin, että vain luotettavat tahot voivat päästä tietoihin käsiksi.

Semantic Webin sovellusmahdollisuudet eivät rajoitu tässä tutkielmassa esitettyihin. Mielikuvituksen lisäksi rajoituksena on kuitenkin vielä teknologioiden keskeneräisyys. Seuraavassa luvussa tuodaan esille muutamia tämän päivän ongelmia.

6. Ongelmia ja tulevaisuudennäkymiä

Kuten edellä on todettu, Semantic Web tarjoaisi toteutuessaan huomattavia hyötyjä sekä kuluttajalle että erilaisille yhteisöille ja yrityksille. Tarvittavien teknologioiden kehitys on hyvässä vauhdissa ja ne mahdollistavat Semantic Web –sovellusten toteuttamisen. Kuitenkin on huomattava, että hanke on vasta lapsenkengissään ja monia ongelmia on ratkottava, ennen kuin

tutkijoiden hurjimmat visiot ovat todellisuutta. Seuraavassa käsitellään eräitä esille tulleita ongelmia.

Haustein ja Pleumann [2002] esittävät huolensa Semantic Webin toteuttamisen liiasta monimutkaisuudesta. Koska käyttökelpoisen metadatan ja erityisesti ontologioiden rakentaminen vaatii jonkin verran tietämystä ja vaivannäköä, Semantic Web tulee tuskin koskaan leviämään koko WWW:n laajuiseksi. Tämä ei kuitenkaan ole edellytys Semantic Webin hyödyntämiselle. Kun teknologiat ovat tarpeeksi kehittyneitä, niiden käyttöönoton tarjoamat hyödyt yrityksille ja yhteisöille tulevat olemaan kiistattomia. Helppokäyttöisten työkalujen vapaa saatavuus on yksi edellytys Semantic Webin laajalle leviämiselle.

Monet Semantic Webin tarvitsemista teknologioista hakevat vielä muotoaan ja useita päällekkäisiä standardeja on kehitteillä. Se estää laajamittaisten sovellusten kehittämisen, mutta on välttämätön vaihe kehityksessä. On tärkeää, että sovellusten tekijöille on tarjolla useita vaihtoehtoja, koska niiden hyvät ja huonot puolet tulevat esille juuri kokeiltaessa niitä käytännössä. Kun teknologioiden kehittäjät saavat palautetta kentältä, he voivat yhdistellä eri teknologioiden vahvoja osia ja tämän prosessin tuloksena toivottavasti syntyy teknologioita, jotka saavat riittävän tuen saavuttaakseen standardin aseman.

Eräs tähän mennessä esille tullut käytännön ongelma koskee hyvin ilmaisuvoimaisten ontologiakielten, kuten OWL:n, pääteltävyyttä. Kaksi ontologiakielten vaatimuksista, suuri ilmaisuvoima ja pääteltävyys, ovat keskenään ristiriidassa. Mitä ilmaisuvoimaisempi kieli on, sen vaikeammaksi tulee hyödyllisten päätelmien tekeminen. Horrocks [2002] huomauttaa, että tällä hetkellä ei ole olemassa menetelmää, jolla OWL-ontologioista voitaisiin automaattisesti tehdä monimutkaisia päätelmiä. Ei ole edes varmaa, onko tällaista menetelmää mahdollista kehittää. Voi siis olla, että yleiskäyttöisen ontologiakielen ilmaisuvoimaa joudutaan rajoittamaan pääteltävyyden parantamiseksi.

Toinen OWL:n toteutuksessa havaittu ongelma liittyy sen RDF-perustaisuuteen. Patel-Schneider ja Fensel [2002] huomauttavat, että OWL:n kehittäjien asettama vaatimus kielen syntaksin ja semantiikan perustumisesta RDF:ään ja RDF Schemaan aiheuttaa Russelin paradoksia muistuttavan ristiriidan luokkien määrittelyssä. W3C:n OWL-työryhmän ratkaistavaksi jää, mikä olisi paras tapa ongelman ratkaisemiseksi.

Yleiskäyttöisen ontologiakielen toteutumiseen on siis vielä matkaa. Semantic Webin visio on kuitenkin tavoittelemisen arvoinen ja kuten tässä

tutkielmassa on käynyt ilmi, työkalujen kehittäjät ovat lähteneet tukemaan kehitystä. WWW:n läpimurtokin tapahtui vasta vuosia sen mahdollistavan tekniikan kehittämisen jälkeen. Ehkä jo muutaman vuoden kuluttua ohjelmistoagentit hoitavat useita arkiaskareitamme.

Viiteluettelo

[Bechhofer et al., 2001] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens, OilEd: a reason-able ontology editor for the semantic web. In: *Proc. of the Joint German Austrian Conference on AI, Lecture Notes In Artificial Intelligence* **2174 (**2001), Springer-Verlag, 396-408.

[Bellifemine et al., 2002] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco and Giovanni Rimassa, JADE Programmer's Guide, July 2002, TILab S.p.A.

[Berners-Lee, 1998] Tim Berners-Lee, Semantic Web Road Map, http://www.w3c.org/DesignIssues/Semantic. [16.12.2002]

[Berners-Lee et al., 2001] Tim Berners-Lee, James Hendler and Ora Lassila, The Semantic Web, *Scientific American* **284**, 5 (May 2001), 34-44.

[Caire, 2002] Giovanni Caire, JADE Tutorial: Application Specific Content Languages and Ontologies, June 2002, TILab S.p.A.

[DAML-S Coalition, 2002] The DAML Services Coalition, DAML-S: Semantic Markup for Web Services, http://www.daml.org/services/daml-s/0.7/daml-s.html. [13.12.2002]

[DC Metadata Initiative, 2002] Dublin Core Metadata Initiative, An Overview of the Dublin Core Metadata Initiative, http://www.dublincore.org/about/overview/. [16.12.2002]

[Emorphia Ltd, 2002] Emorphia Ltd, FIPA-OS Feature List, http://fipa-os.sourceforge.net/features.htm. [19.12.2002]

[Friedman-Hill, 2002] Ernest J. Friedman-Hill, JESS, The Expert System Shell for the Java Platform version 6.1a4, August 2002, http://herzberg.ca.sandia.gov/jess/docs/61/. [19.12.2002]

[Haustein and Pleumann, 2002] Stefan Haustein and Jörg Pleumann, Is Participation in the Semantic Web Too Difficult?, *Lecture Notes in Computer Science* **2342** (2002), 448-453.

[Heflin et al., 2002] Jeff Heflin, Raphael Volz and Jonathan Dale (eds.), Requirements for a Web Ontology Language W3C Working Draft 8 July 2002, http://www.w3.org/TR/2002/WD-webont-req-20020708/. [13.10.2002]

[Hendler, 2001] James Hendler, Agents and the semantic web, *IEEE Intelligent Systems* **16**, 2 (2001), 30-37.

[Horrocks, 2002] Ian Horrocks, Reasoning with expressive description logics: Theory and practice. In: Andrei Voronkov, editor, *Proc. of the 18th Int. Conf. on Automated Deduction (CADE-18), Lecture Notes in Artificial Intelligence* **2392** (2002), Springer-Verlag, 1-15.

[ICS-FORTH, 2002] ICS-FORTH, VRP Overview, http://139.91.183.30:9090/RDF/VRP/index.html. [19.12.2002]

[Manola and Miller, 2002] Frank Manola and Eric Miller (editors), RDF Primer W3C Working Draft 26 April 2002, http://www.w3.org/TR/2002/WD-rdf-primer-20020426/. [13.10.2002]

[McBride et al., 2002] Brian McBride, Andy Seaborne and Jeremy Carroll, Jena Tutorial for Release 1.4.0, April 2002, http://www.hpl.hp.com/semweb/doc/tutorial/index.html. [19.12.2002]

[Mcguinness et al., 2002] Deborah L. Mcguinness, Richard Fikes, James Hendler and Lynn Andrea Stein, DAML+OIL: an ontology language for the Semantic Web, *IEEE Intelligent Systems* **17**, 5 (2002), 72-80.

[Noy et al., 2001] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson and M. A. Musen, Creating Semantic Web Contents with Protege-2000, *IEEE Intelligent Systems* **16**, 2 (2001), 60-71.

[Ontoprise Ltd., 2002] Ontoprise Ltd., OntoEdit Datasheet 2002, http://www.ontoprise.de/com/download/ontoedit_data_sheet.pdf. [20.12.2002]

[Patel-Schneider and Fensel, 2002] Peter F. Patel-Schneider and Dieter Fensel, Layering the Semantic Web: Problems and Directions. In: I. Horrocks and J. Hendler, editors, *ISWC 2002, Lecture Notes in Computer Science* **2342** (2002), 16-29.

[Smith et al., 2002] Michael K. Smith, Deborah McGuinness, Raphael Volz, Chris Welty, Web Ontology Language (OWL) Guide Version 1.0 W3C Working Draft 4 November 2002, http://www.w3.org/TR/2002/WD-owl-guide-20021104/. [19.12.2002]

Vektoriavaruusmalli tekstitiedonhaussa

Tuomas Talvensaari

Tiivistelmä
Tutkielmassa tarkastellaan vektoriavaruusmallia tekstidokumenttien tiedonhaun työkaluna. Käyn läpi vektoriavaruusmallin matemaattisia perusteita, automaattisen indeksoinnin perusmenetelmiä, termien painoarvojen laskemiseen käytettäviä menetelmiä ja vektoriavaruusmallista johdettua kontekstivektorimallia.

Avainsanat ja -sanonnat: tiedonhaku, vektoriavaruusmalli.
CR-luokat: H.3.3

1. Johdanto

Tiedonhakujärjestelmien tarkoituksena on järjestää dokumenttikokoelmia siten, että käyttäjien on mahdollista kyselyiden avulla hakea niistä haluamaansa tietoa. Aiemmin tiedonhakujärjestelmät perustuivat boolean-tyyppisiin hakuihin, joissa käyttäjä määritteli tiedon tarpeensa yhdistelemällä hakutermejä AND OR tai NOT -operaattoreilla. Näiden järjestelmien heikkous oli se, että niissä ei pystytty juurikaan järjestämään haettuja dokumentteja niiden relevanssin mukaiseen järjestykseen, ts. siten että viittaus käyttäjän tiedontarpeen kannalta oleellisimpaan dokumenttiin olisi ensimmäisenä, ja tätä seuraisivat laskevassa järjestyksessä viittaukset seuraavaksi oleellisimpiin dokumentteihin [Singhal, 2001]. Vektoriavaruusmallissa tällainen relevanssin mukainen järjestys saavutetaan laskemalla dokumenttien ja kyselyjen välisen samanlaisuuden aste mittaamalla näistä muodostettujen termivektoreiden samanlaisuutta. Vektoriavaruusmallia kokeiltiin laajalti 1960-luvun lopulta lähtien Gerard Saltonin johdolla kehitetyssä SMART-tiedonhakujärjestelmässä Cornellin yliopistossa [Harman, 1992].

2. Vektoriavaruusmallin perusperiaatteet

Vektoriavaruusmallissa dokumentteja edustavat termivektoreiden lineaarikombinaatiot. Vektorit koostuvat joko termien painoarvoista (jotka riippuvat niiden tärkeydestä ko. dokumenttikokoelmassa) tai yksinkertaisemmin

pelkästään 1:stä tai 0:sta sen mukaan, onko ko. termiä ko. dokumentissa. Vektoriavaruuden lähtökohtana voidaan ajatella seuraavanlaista matriisia:

$$D = \begin{array}{c} \\ t_1 \\ t_2 \\ \vdots \\ t_n \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & \cdots & d_m \end{array} \\ \begin{pmatrix} w_{11} & w_{21} & \cdots & w_{m1} \\ w_{12} & w_{22} & \cdots & w_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{mn} \end{pmatrix} \end{array}.$$

Tässä kokoelmassa on m dokumenttia ja n indeksointiin valittua termiä. Matriisia tulkitaan siten, että termi t_j esiintyy dokumentissa d_i w_{ij} kertaa (tai vaihtoehtoisesti w_{ij} on 1 silloin, kun termi t_j esiintyy dokumentissa d_i, ja 0 muulloin). Matriisista voidaan sitten laskea dokumentteja esittävät termivektorit

$$\vec{d}_i = (w_{i1}, w_{i2}, \ldots, w_{in}).$$

Useimmiten on tarkoituksenmukaista korvata pelkät termien esiintymislukumäärät normalisoiduilla painoarvoilla, jotta estettäisiin pitkien dokumenttien (runsastermisten dokumenttien) saama etu (muitakin syitä on). Näistä painoarvoista kerrotaan lisää myöhemmin.

Myös käyttäjän muotoilemat kyselyt voidaan ajatella dokumentteina, joista muodostetaan samaan tapaan vektorit. Näin dokumentti- ja kyselyvektorin välinen samankaltaisuus voidaan laskea summana

$$\vec{d} \cdot \vec{q} = \sum_{i,j=1}^{n} w_{di} w_{qj} t_i \cdot t_j .$$

Ongelmaksi muodostuu termien välisten korrelaatioiden $t_i \cdot t_j$ laskeminen. Nehän eivät sinällään sisälly matriisiin D. Tämä ongelma on useimmiten kierretty olettamalla termien välisen korrelaation olevan 1, kun $t_i = t_j$, ja 0 muulloin, jolloin termivektorit olisivat toisiinsa nähden ortogonaalisia. Tästä seuraa niiden keskinäinen lineaarinen riippumattomuus, mikä takaa sen että ne muodostavat matemaattisessa mielessä lineaarisen vektoriavaruuden kannan. Vaikka ratkaisu on käytännössä toiminutkin, ei se ole ongelmaton [Raghvanan and Wong, 1986], sillä onhan intuitiivisestikin selvää, että eri termit eivät esiinny dokumenteissa täysin sattumanvaraisesti toisiinsa nähden. Esimerkiksi termi *ohjelmointikieli* esiintyy varmasti useammin samassa yhteydessä termin *Java* kuin vaikkapa termin *makaronilaatikko* kanssa. Olettamalla, että termien välillä ei ole korrelaatiota, saadaan dokumenttien ja kyselyjen välisen samanlaisuuden kaavasta yksinkertaisempi muoto:

$$sim(d,q) = \sum_{i=1}^{n} w_{di} w_{qi} \ .$$

Saadun tuloksen perusteella dokumentit voidaan järjestää laskevaan samanlaisuusjärjestykseen kyselyyn nähden [Salton, 1989].

3. Automaattisen indeksoinnin menetelmiä

Ei ole tarkoituksenmukaista käyttää kaikkia dokumenttien termejä niiden kuvaamiseen. Tämä johtaisi vektoriavaruusmallissa termiavaruuden dimension suhteettomaan suuruuteen.

Salton [1975] kuvaa termin kelpoisuutta indeksointiin sen erottelukykyarvolla (*discrimination value*). Hänen mukaansa huonoimpia indeksointitermejä ovat yleisimmät termit, tarkemmin sanoen sellaiset, jotka esiintyvät yli kymmenesosassa kokoelman dokumenteista. Myöskään kaikista harvinaisimmat, alle sadasosassa kokoelman dokumenteista esiintyvät termit, eivät erottele riittävän hyvin dokumenttiavaruutta. Tällaiset laskelmat ovat kuitenkin hyvin kokoelmakohtaisia, ja ne sopivatkin ehkä parhaiten aihealueeltaan hyvin spesifien kokoelmien yhteyteen.

Yleisimmät hyvien indeksointitermien löytämiseen käytettävät menetelmät ovat ns. hukkasanalistojen (*stoplist*) käyttäminen ja stemmaus (*stemming*) eli sanarunkojen haku.

3.1 Hukkasanalistat

Yleisimmät luonnollisissa kielissä esiintyvät sanat eivät ole hyviä indeksointitermejä, ts. niiden arvo dokumenttien kuvaajina on mitätön. Tällaisia hyödyttömiä termejä ovat englannin kielessä mm. sanat *the, of, and* ja *to* ja suomen kielessä *ei, hän,* ja *nyt*. Dokumentteja indeksoitaessa ne käydään läpi ja turhat sanat poistetaan ns. hukkasanalistojen (*stoplist*) avulla. Poistamalla hukkasanat pienennetään myös indeksitiedostojen kokoa 30-50% [Rijsbergen, 1979] ja vektoriavaruusmallissa termiavaruuden kokoa.

3.2 Stemmaus

Hukkasanalistojen lisäksi toinen yleinen keino lisätä indeksoinnin tehoa ja pienentää termiavaruuden dimensiota on sanarunkojen haku eli stemmaus (*stemming*). Tämä perustuu sanojen morfologiaan eli muoto-oppiin. Yksinkertaistettuna stemmaus toimii siten, että luodaan lista kielen sanojen mahdollisista päätteistä ja tämän listan avulla poistetaan päätteet indeksoitavista sanoista. Stemmaus on kuitenkin toteutukseltaan huomattavasti ongelmallisempi kuin hukkasanojen poisto. Luonnollinen kielihän ei ole koskaan täysin säännönmukainen, minkä vuoksi pelkkä päätelista ei riitä

stemmauksessa. Esimerkiksi englannin kielessä on pääte *–ing* , joka poistamalla saadaan sanasta *interesting* sen kantamuoto. Mutta myös sanasta *king* löytyy sama pääte, ja on selvää, että sitä ei pidä mennä poistamaan. Ongelmia tuottavat myös epäsäännölliset muodot, esim. sanan *understood* perusmuoto on *understand*. Edelleen englannissa tiettyihin konsonantteihin päättyvät sanat tuplaavat viimeisen kirjaimensa kun niihin lisätään pääte –esim. *running*–sanan perusmuoto ei ole *runn* vaan *run*. Näin stemmauksen toteuttaviin järjestelmiin tulee sisällyttää myös kontekstuaalista tietoa kielestä. Tunnettuja toteutuksia englanninkielen stemmauksesta ovat Lovinsin (1968) ja Porterin (1980) toteutukset. Lovinsin menetelmä etsii pisimmän päätteen isosta päätelistasta ja poistaa sen. Porter käyttää hienostuneempaa iteratiivista algoritmia, ja sen päätelista on pienempi [Salton, 1989].

Tiedonhakututkimuksessa on keskusteltu suhteellisen paljon stemmauksen hyödyllisyydestä. Hull [1996] vertailee erilaisia stemmauksen toteutustapoja ja päätyy siihen, että stemmauksesta *on* hyötyä englanninkielisissä aineistoissa. Kyselyjen pituudella ja yksityiskohtaisuudella on kuitenkin vaikutusta stemmauksen tehokkuuteen: lyhyet kyselyt hyötyvät stemmauksesta enemmän. Etuliitteiden poisto on Hullin mukaan lähinnä haitallista; englannin kielen etuliitteethän yleensä muuttavat sanan merkitystä varsin radikaalisti.

Kokonaan toinen asia on se, miten stemmaus sopii suomenkieliseen aineistoon. Suomen kielessä on valtavasti enemmän päätteitä kuin englannissa. Pidempien pääte-listojen lisäksi erilaiset epäsäännöllisyydet tekisivät suomenkielen stemmauksesta käytännössä mahdottoman toteuttaa.

4. Termipainoarvot

Etsittäessä hyvää tapaa painottaa termejä dokumenttivektoreissa on otettava huomioon seuraavanlaisia seikkoja:

1. On edullista hajottaa dokumenttiavaruutta siten, että toisistaan paljon eroavat dokumentit ovat mahdollisimman etäällä toisistaan. Toisaalta dokumenttien, joiden sisältö on keskenään samansuuntainen, tulisi olla dokumentti-avaruudessa lähekkäin. [Salton *et al.*, 1975]
2. Jotta tällainen erottelu olisi mahdollinen, tulisi painoarvojen korostaa niitä termejä, joiden erottelukyky on suuri. Näin esimerkiksi termit, jotka esiintyvät runsaslukuisina suurimmassa osassa kokoelman dokumenteista, eivät ole erottelukyvyltään hyviä, ja niiden painoa tulisi vähentää.

3. Pitkien dokumenttien ei tulisi saada etua tiedonhaussa pelkän pituutensa ansiosta. Satoja sivuja pitkä dokumentti, jossa tietty käyttäjää kiinnostava termi esiintyy yhtä monta kertaa kuin 10-sivuisessa dokumentissa, on käyttäjän kannalta vähemmän kiinnostava.

Salton ja Buckley [1988] jaottelevat termin painoarvoon vaikuttavat tekijät kolmeen komponenttiin: termifrekvenssi-, kokoelmafrekvenssi- ja normalisointi-komponenttiin.

Termifrekvenssikomponentti saadaan yleensä suoraan dokumentti-termi -matriisista D; se on siis termien esiintymisten lukumäärä (*term frequency, tf*) dokumentissa (tai yksinkertaisemmissa binäärivektoriratkaisuissa 1 tai 0 sen mukaan, esiintyykö termi dokumentissa).

Pelkän termifrekvenssikomponentin käyttö ei sinällään useimmiten riitä. Kokoelma-frekvenssikomponentin tarkoituksena on antaa etua sellaisille termeille, jotka hajottavat dokumenttiavaruutta, ja toisaalta rangaista sellaisia termejä, jotka esiintyvät suhteellisen tasaisesti kokoelman dokumenteissa. Tähän käytetään käänteistä dokumenttifrekvenssiä (*inverse document frequency, idf),* joka useimmiten esiintyy muodossa

$$\log \frac{N}{n},$$

missä N on dokumenttien määrä kokoelmassa ja n on niiden dokumenttien lukumäärä, joissa termi esiintyy.

Normalisointikomponentin tarkoituksena on estää pitkien dokumenttien saama etu tiedonhaussa. Pitkät dokumentit hyötyvät Singhalin *et al.* [1996] mukaan kahdestakin syystä:

1. Yksittäisten termit esiintyvät useammin pitkissä dokumenteissa; tällaisen dokumentin termifrekvenssit ovat siis keskimäärin suuremmat kuin lyhyillä dokumenteilla.
2. Pitkissä dokumenteissa on myös enemmän eri termejä, mikä suurentaa tällaisen dokumentin mahdollisuuksia tulla löydetyksi käyttäjän muotoileman kyselyn myötä.

Yleisimmin käytetty normalisointitapa on kosininormalisointi, jossa termin painoarvo kerrotaan arvolla

$$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_t^2}},$$

jossa w_i on termin normalisoimaton painoarvo $tf \times idf$. Tämä normalisointimenetelmä korjaa molemmat edellä esitetyt pitkien dokumenttien saamat edut. Toinen yleisesti käytetty normalisointitapa on

käyttää dokumentin suurinta termifrekvenssiä. Tällöin termin painoarvo voidaan laskea vaikkapa kaavalla

$$0{,}5 + 0{,}5 \times \frac{tf}{\max tf},$$

jossa max *tf* on dokumentin suurin termifrekvenssi. Tämä tapa normalisoi termin painoarvon välille [0,5, 1]. Heikkoutena tässä on se, että se puuttuu vain ensimmäiseen edellä mainituista pitkien dokumenttien saamista eduista.

Salton ja Buckley [1988] esittävät, että dokumentti- ja kyselyvektoreissa tulisi käyttää erilaisia termipainotuksia. Tämä vaikuttaakin perustellulta, ovathan kyselyt luonteeltaan täysin erilaisia kuin dokumentit. Huomattavin ero on tietysti pituus. Lisäksi voisi ajatella, että (ainakin lyhyissä) kyselyissä jokainen termi on yhtä arvokas, ja että termifrekvenssejä ei tarvitse välttämättä ottaa huomioon – voidaan siis käyttää binäärivektoreita. Dokumenttivektoreissa Salton ja Buckley [1988] ehdottavat käytettävän painoarvoja, joissa käytetään termifrekvenssiä, käännettyä dokumentti-frekvenssiä ja kosininormalisointia. Termin paino voidaan laskea kaavalla

$$\frac{tf \cdot \log \frac{N}{n}}{\sqrt{\sum_{vektori} \left(tf_i \cdot \log \frac{N}{n_i} \right)^2}}.$$

On kuitenkin havaittu että kosininormalisointi rankaisee pitkiä dokumentteja liikaakin. Singhal *et al.* [1996] ehdottavat tämän korjaamiseksi ”kierrettyä” normalisointimenetelmää (*pivoted normalization*). Menetelmä perustuu havainnolle, että on olemassa pituus, jota pidempien dokumenttien todennäköisyys löytyä haussa on pienempi kuin niiden relevanssin todennäköisyys. Toisaalta tätä rajapistettä lyhyemmät dokumentit löytyvät haussa todennäköisemmin kuin on niiden todennäköinen relevanssi. Rajapistettä käytetään kohtana, jonka ympäri normalisointifunktiota ”kierretään” loivemmaksi, jolloin relevanssin ja löytymisen todennäköisyydet lähenevät toisiaan dokumenttien pituusjakauman molemmissa päissä. Rajapisteeksi voidaan valita vanhan normaalisointifunktion keskiarvo, jolloin normalisointikomponentiksi muodostuu

$$\frac{1}{(1{,}0 - s) + s \times \frac{old\_norm}{av\_old\_norm}},$$

jossa *s* on jokin vakio, joka saadaan opetusaineistosta (Singhal *et al.* päätyvät arvoon 0,7) ja *old_norm* on vanha, ”kääntämätön” normalisointifunktio

(vaikkapa kosininormalisointi) ja *av_old_norm* sen määräämien arvojen keskiarvo.

5. Tiedonhakumenetelmien arviointi

Tiedonhakumenetelmien arvioinnin perusyksiköiksi ovat vakiintuneet saanti (*recall)*, eli haettujen relevanttien dokumenttien osuus kaikista kokoelman relevanteista dokumenteista, ja tarkkuus (*precision)*, relevanttien dokumenttien osuus haetuista dokumenteista. Tutkimusaineistona käytetään yleensä dokumenttikokoelmaa ja joukkoa tähän kokoelmaan muotoiltuja kyselyjä. Dokumenttien relevanssi kyselyihin nähden on myös arvioitu etukäteen. Tyypillisin tapa arvioida käsillä olevaa tiedon-hakumenetelmää on mitata haun tarkkuutta eri saantipisteissä, ts. relevanttien dokumenttien osuutta haetuista dokumenteista kun relevanteista dokumenteista on löytynyt tietty osuus. Yleensä saantipisteet ovat 10%:n välein, ja näiden saantipisteiden tarkkuusarvoista otetaan keskiarvo. Lisäksi voidaan mitata saantia ja tarkkuutta tietyn dokumenttimäärän jälkeen. Hull [1996] kritisoi tiedonhakututkimuksia siitä, että niissä luotetaan liikaa tällaisten tunnuslukujen voimaan, eikä lähemmin analysoida tutkimustuloksia. Ei esimerkiksi tutustuta tarkemmin tutkimusaineiston luonteeseen, kuten dokumenttien tai kyselyjen pituuteen. Esimerkiksi Hullin käyttämän TREC-aineiston kyselyt ovat pitkiä ja hienostuneita kuvauksia eri aihealueista, toisin kuin useampien tavallisten käyttäjien tiedonhakukyselyt, jotka yleensä koostuvat muutamasta termistä. TREC-aineistossa (eikä varmasti ainoastaan siinä) ongelmana on Hullin mukaan myös se, että kyselyihin liitettyjen relevanttien dokumenttien määrä voi vaihdella suuresti. Johonkin kyselyyn on löydetty yli sata relevanttia dokumenttia, toiseen vain muutama. Tällaisessa muutaman relevantin dokumentin kyselyssä voi yhden dokumentin sijoittuminen hakutuloksessa vaikuttaa suuresti tarkkuus- ja saantilukuihin, mikä tekee tuloksista epäluotettavia ja alttiita sattumalle.

Hull [1996] esittää käytettäväksi kolmea eri tunnuslukua: 11 saantipisteen tarkkuuskeskiarvoa, keskiarvoa ensimmäisen 5,6,…,15 haetun dokumentin tarkkuudesta ja keskiarvoa ensimmäisen 50, 60, …, 150 haetun dokumentin saannista. Ensimmäinen tunnusluvuista on vakiintunut tiedonhakututkimuksien perusluvuksi ja se antaa yleisarvion tiedonhakualgoritmin toimivuudesta. Toinen mittaa suppeiden hakujen tehokkuutta ja edustaa ehkä parhaiten tavallisen ”maallikkokäyttäjän” tarpeita, kun taas kolmas tunnusluku arvioi laajojen hakujen tehokkuutta.

6. Vektoriavaruusmallin laajennuksia

Vektoriavaruusmallin perusversion suurin heikkous piilee siinä, ettei siinä määritellä termien välisiä suhteita, vaan niiden korrelaatio oletetaan nollaksi. Tästä seuraa se, että laskettaessa dokumentin ja kyselyn samanlaisuutta otetaan huomioon vain täysin yhtenevät termit. Kuitenkin on niin, että samasta aihepiiristä kertovat dokumentit voivat käyttää eri termejä (synonyymit). Samanlaiset termit voivat myös eri yhteyksissä viitata eri käsitteisiin, ja tällaisten termien yhteneväisyys samanlaisuuslaskussa toisi virheellisen tuloksen.

Eräs keino parantaa tilannetta ovat erilaiset kyselynlaajennustekniikat (*query expansion*). Niissä käyttäjän aluksi muotoilemaa kyselyä laajennetaan niin, että se lähenee dokumenttiavaruudessa oletettuja relevantteja dokumentteja. Tämä voidaan toteuttaa vaikkapa ns. relevanssipalautteen (*relevance feedback*) avulla, jossa käyttäjä valitsee ensimmäisestä hakutuloksesta dokumentit, jotka ovat hänen mielestään eniten relevantteja. Näitä dokumentteja analysoimalla tiedonhakujärjestelmä sitten muokkaa kyselyä lisäämällä siihen termejä. Ns. sokeassa palautteessa (*blind feedback*) käyttäjän ei tarvitse osallistua kyselynlaajennukseen, vaan järjestelmä tutkii ensimmäisessä haussa parhaiten sijoittuneita dokumentteja ja lisää niistä termejä kyselyyn [Billhardt *et al.*, 2002].

Vektoriavaruusmallista on myös kehitetty versioita, joissa termien välinen korrelaatio otetaan paremmin huomioon. Billhardt *et al.* [2002] esittelevät kontekstivektorimallin (*context vector model*), jossa pyritään esittämään dokumentit semanttisina kokonaisuuksina, eikä pelkkinä yksittäisten termien lineaari-kombinaatioina.

Myös kontekstivektorimallin lähtökohtana on luvussa 2 esitetty matriisi D, johon lasketaan indeksoitavien termien esiintymisten lukumäärät eri dokumenteissa:

$$D = \begin{array}{c} \\ t_1 \\ t_2 \\ \vdots \\ t_n \end{array} \begin{array}{cccc} d_1 & d_2 & \cdots & d_m \\ w_{11} & w_{21} & \cdots & w_{m1} \\ w_{12} & w_{22} & \cdots & w_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{mn} \end{array} .$$

Seuraava askel on laskea termikontekstivektorit, jotka saadaan $n \times n$–matriisista (n on termien lukumäärä) T:

$$T = \begin{pmatrix} c_{11} & c_{21} & \cdots & c_{n1} \\ c_{12} & c_{22} & \cdots & c_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1n} & c_{2n} & \cdots & c_{nn} \end{pmatrix}.$$

Matriisin T alkio c_{ij} kuvaa termin t_j vaikutusta termiin t_i, ja se lasketaan seuraavalla tavalla:

$$c_{ij} = \begin{cases} 1 & , jos\ i = j \\ \dfrac{\sum_{\substack{k=1, \\ missä\ w_{kj} \neq 0}}^{m} w_{ki}}{\sum_{k=1}^{m} w_{ki}} & , muuten. \end{cases}$$

Näin c_{ij} on termin t_i niiden painoarvojen, jotka esiintyvät samassa dokumentissa termin t_j kanssa, summan suhde termin t_i kaikkien painoarvojen summaan. Termin t_i kontekstivektori on

$$\vec{t}_i = (c_{i1}, c_{i2}, \ldots, c_{in}).$$

Dokumenttien kontekstivektorit saadaan kaavalla

$$\vec{d} = \frac{\sum_{j=1}^{n} w_{ij} \frac{\vec{t}_j}{|\vec{t}_j|}}{\sum_{j=1}^{n} w_{ij}},$$

missä $\vec{t}_j$ on termin t_j kontekstivektori ja $|\vec{t}_j|$ sen pituus, jolla termikontekstivektorin elementit normalisoidaan. Kyselyjen ja dokumenttien samanlaisuus mitataan tavallisella kosinisamanlaisuusfunktiolla. Sen sijaan termipainoarvot poikkeavat jonkin verran tavallisista, tässäkin artikkelissa esitetyistä, mutta niitä ei lähdetä erittelemään tässä.

Kontekstivektorimallin ja perinteisen vektoriavaruusmallin olennaisin ero on siis se, että kun perinteisessä mallissa lineaariavaruuden kantana ovat yksittäiset termit, perustuu kontekstivektorimalli enemmänkin semanttisiin käsitteisiin. Tämän seurauksena on mm. se, että kontekstivektorimallin dokumenttivektorit ovat paljon ”tiheämpiä” kuin perusversiossa. Perusversiossa suurin osa dokumenttivektoreiden termipainoarvoista on 0. Kontekstivektorimallissa termin painoarvo ei välttämättä ole 0, vaikka termiä ei kyseisessä dokumentissa esiintyisikään. Tämä asettaa erilaiset vaatimukset säilytystilalle ja järjestelmän suorituskyvylle.

7. Loppupäätelmiä

Vektoriavaruusmalli on erilaisine laajennuksineen osoittautunut tehokkaaksi tekstitiedonhaun menetelmäksi. Jo 1960-luvulta lähtien suoritetun tutkimuksen avulla on kehitetty varsin vakiintuneita tapoja kuvailla tekstidokumenttien sisältöä ja vertailla sitä käyttäjän muotoilemiin kyselyihin. Tässä tutkielmassa käyttämäni kirjallisuus on kuitenkin pohjautunut lähinnä englanninkieliselle tutkimusaineistolla tehtyihin tutkimuksiin. Itse aion jatkossa tutkia myös muunkielisiä aineistoja, ja samalla myös muita keinoja mitata dokumenttien välistä samanlaisuutta.

Viiteluettelo

[Billhardt et al., 2002] Holger Billhardt, Daniel Borrajo and Victor Maojo, A context vector model for information retrieval. *Journal of the American Society for Information Science and Technology* **53**, 3 (2002), 236-249

[Harman, 1992] Donna Harman, Ranking algorithms. In: William B. Frakes and Ricardo Baeza-Yates (eds.), *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, 1992, 363-392.

[Hull, 1996] David A. Hull, Stemming algorithms: a case study for detailed evaluation. *Journal of the American Society for Information Science* **47**, 1 (1996), 70-84.

[Raghvanan and Wong, 1986] Vijay V. Raghvanan and S. K. M. Wong, A critical analysis of vector space model for information retrieval. *Journal of the American Society for Information Science* **37**, 5 (1986), 279-287.

[Rijsbergen, 1979] C. J. van Rijsbergen, *Information Retrieval*. Butterworths, 1979.

[Salton, 1989] Gerald Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[Salton and Buckley, 1988] Gerard Salton and Christopher Buckley, Term-weighting approaches in automatic text retrieval. *Information Processing & Management* **24**, 5, (1988), 513-523.

[Salton *et al.*, 1975] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM* **18**, 11 (1975), 613-620.

[Singhal, 2001] Amit Singhal, Modern information retrieval: a brief overview. *IEEE Data Engineering Bulletin* **24**(4), (2001), 35-43.

[Singhal *et al.*, 1996] Amit Singhal, Chris Buckley and Mandar Mitra, Pivoted document length normalization. In: *Proceedings of the 19th ACM Conference on Research and Development in Information Retrieval (SIGIR'96)*, (1996), 21-29.