## Lukijalle

Tähän julkaisuun on kerätty syyslukukaudella 2001 pitämälläni Tutkimuskurssilla tehdyt, määräaikaan mennessä valmistuneet tutkielmat.

Toimittaja

## Sisällysluettelo



.

# Neuroverkoista ja nimeämishäiriöistä

## Antti Järvelin

**Tiivistelmä.**

Tutkielmassa tarkastellaan afaattisten nimeämishäiriöiden mallintamista kahden erityyppisen lokaalistisen neuroverkon avulla. Ensimmäinen neuroverkko pohjautuu interaktiivisen aktivaation teoriaan nimeämisestä ja toinen erilliseen kaksivaiheiseen teoriaan nimeämisestä.

Avainsanat ja -sanonnat: Neuroverkot, kaksivaiheinen teoria nimeämisestä, nimeämishäiriö.
CR-luokat: I.6.3

## 1. Johdanto

Afasia on aivoperäinen puheen tuottamis- ja ymmärtämishäiriö. Yleisimmät syyt afasiaan johtaneisiin aivovaurioihin ovat aivoinfarktit sekä aivoverenvuodot [Laine ja Marttila, 1992]. Kuvannimeämistehtävän avulla on mahdollista tutkia afaattisia nimeämisongelmia [Dell et al., 1996]. Tehtävässä potilaalle esitetään kuvia esineistä yksi kerrallaan, jotka potilaan tulee nimetä, eli sanoa ääneen kuvaa vastaava sana [ibid.].

Jos visuaalinen vaihe sekä puheen artikulaatio suljetaan tarkastelusta pois, sananhakua simuloivissa malleissa erotetaan yleensä kaksi vaihetta: *lemmanhaku* (lemma access) ja *fonologinen haku* (fonological access). Ensimmäisessä vaiheessa, eli lemmanhakuvaiheessa, kuvataan nimettävän sanan käsitteellinen esitys sanan abstarktiksi ei-fonologiseksi esitykseksi, *lemmaksi* [Dell et al., 1997]. Toisessa vaiheessa eli fonologisen haun aikana lemma muutetaan sanan fonologiseksi esitykseksi.

Neuroverkot ovat luonnollinen tapa mallintaa nimeämishäriöitä, sillä niiden rakenne vastaa karkealla tasolla aivojen rakennetta. Rakenteensa johdosta neuroverkot kestävät hyvin vaurioita ja vaurioiden tuottaminen neuroverkkoihin on yksinkertaista. Vaurionkestävyydellä tarkoitetaan sitä, että vaurioitettunakin neuroverkot antavat tuloksia, jotka muistuttavat jossain määrin oikeaa tulosta.

Tässä tutkielmassa tarkastellaan kahta erilaista neuroverkkomallia kaksivaiheisen nimeämisen simuloimiseksi. Ensimmäinen, interaktiivinen aktivaation teoriaan perustuva malli, olettaa, että lemuan saantivaiheessa myös nimettävän sanan fonologia aktivoituu ja vahvistaa nimettävän sanan leimaa takaisinkytkentöjen avulla.

Interaktiivinen aktivaatioon perustuvaa mallia tarkastellaan Dellin ja muiden (1996) ja (1997) esittämän mallin pohjalta.

Toisen, erilliseen kaksivaiheteoriaan perustuvan mallin, mukaan nimeämisen vaiheet ovat toisistaan täysin erillisiä. Ensin valitaan lemma lemmojen joukosta ja valituksi tulleelle lemmalle haetaan foneemitason esitys foneemien joukosta erillään lemmatasosta. Erilliseen kaksivaiheteoriaan perustuvan suomenkielisen SLIPNET-neuroverkkon tarkastelu pohjautuu pääasiassa Juholan ja Laineen (1998) SLIPNET-verkosta esittämän version pohjalle.

Kummatkin mallit simuloivat nimeämistä kierrospohjaisesti. Tällä tarkoitetaan sitä, että nimeämiseen käytetty aika on jaettu erillisiksi aikayksiköiksi eli kierroksiksi. Esitettävät neuroverkot laskevat tuloksensa jokaisella kierroksella käyttäen hyväksi edellisen kierroksen tulosta. Kun ennalta määrätty määrä kierroksia simulaation alusta on kulunut, valitaan tämän kierroksen tulos neuroverkon tulokseksi annetulla syötteellä. Ajan pilkkominen erillisiin aikayksikköihin on yksinkertaisin tapa mallintaa neuroverkoissa tapahtuvaa rinnakkaisprosessointia.

Luvussa kaksi luodaan suppea yleiskatsaus neuroverkkoihin. Lisäksi pohditaan konnektionistisen mallintamisen soveltumista puheentuotonhäiriöiden mallintamiseen. Luvuissa kolme ja neljä esitellään kaksi erityyppistä neurolaskentamallia, jotka simuloivat kuvannimeämistä perustuen kaksivaiheeseen teoriaan nimeämisestä. Luvussa viisi vertaillaan edellisissä luvuissa esitettyjen mallien antamia tuloksia.

## 2. Neuroverkoista

### Neuroverkkojen rakenne ja laskentaperiaate

Neuroverkot koostuvat *solmuista*, eli *neuroneista*, ja niiden välisistä suunnatuista *yhteyksistä*, eli *kaarista*, jotka kytkevät kaksi solmua toisiinsa. Kaaret voidaan tulkita hermosolujen välisten yhteyksien ja solmut hermosolujen vastineeksi. Kahden solmun välisellä kaarella on *paino* (jokin reaaliluku), jolla ilmaistaan solmujen välisen yhteyden voimakkuus. Jokaiseen solmuun liittyy *kynnysarvo*, joka on solmuun saapuva ylimääräinen vakiosyöte. Verkolle on lisäksi määrätty *kynnysfunktio*, joka säätelee yhdessä kynnysarvon kanssa solmun tuloksia.

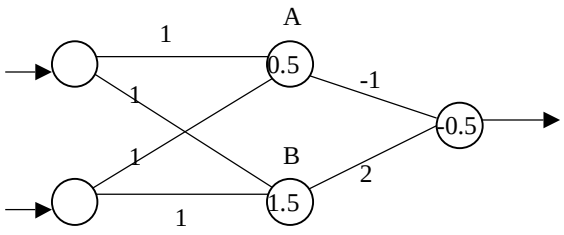
Yhden neuronin toimintaa voidaan kuvata seuraavasti. Jokainen neuroniin saapuva syöte kerrotaan sitä vastaavan kaaren painolla. Näin saadut painotetut syötteet lasketaan yhteen. Saadusta summasta vähennetään solmun kynnysarvo ja tämä luku annetaan kynnysfunktion syötteeksi. Kynnysfunktion tulos on samalla myös neuronin tulos. Neuroni lähettää tuloksensa eteenpäin sellaisille neuroneille,

joihin se on yhdistetty. Nämä laskevat tuloksensa samalla periaatteella. Täten siis neuroni voidaan pelkistää kynnysfunktioksi ja kynnysarvoksi.

Neuroverkkojen tärkeimmät käsitteet ovat siis:

1. Solmujen määrä
2. Solmujen väliset kytkennät ja niiden painokertoimet (eli verkon rakenne) sekä solmujen kynnysarvot (jotka voidaan mieltää myös ylimääräiseksi painoksi)
3. Kynnysfunktio, joka määrää kunkin solmun tuloksen.

Esimerkkinä tarkastellaan kuvan 1 neuroverkkoa. Kuvassa ympyröillä merkitään solmuja ja viivoilla kaaria. Verkon rakenne jaetaan kerroksiin siten, että vasemmanpuoleisimpia solmuja kutsutaan syötekerrokseksi, keskimmäisiä (kuvassa solmut A ja B) piilokerrokseksi, ja oikeanpuoleisinta kutsutaan tuloskerrokseksi. Kuvan 1 verkossa on siis kaksi syötekerroksen solmua, kaksi piilokerroksen solmua ja yksi tuloskerroksen solmu. Solmujen väliset kaaret ovat suunnattuja vasemmalta oikealle. Kaarien yhteydessä esiintyvät numerot ovat niiden painoja ja solmuissa esiintyvät luvut taas ovat solmuun liittyviä kynnysarvoja. Syötekerroksen solmuilla ei ole kynnysarvoja, sillä ne välittävät syötteensä eteenpäin sellaisenaan.




Kuva 1. Ekvivalenssiongelman ratkaiseva neuroverkko.

Esimerkkiverkko ratkaisee loogisen ekvivalenssin. Verkko saa syötteekseen kaksiulotteisia bittivektoreita (esim. (0,1)) ja antaa tulosteena joko bitin 1 tai 0, loogisen ekvivalenssin totuustaulun mukaisesti. Kynnysfunktiona toimii nk. Heavisiden funktio [Juhola, 1998]:

$$y = \mathrm{f}\left( \sum_{i=1}^{n} w_i x_i - \Theta \right),$$

missä

$$\mathrm{f}(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}.$$

Kynnysfunktio siis laskee yhteen neuronin jokaisen syötteen ja kertoo sen saapuvan kaaren painoarvolla. Jos näin saatu summa on suurempi kuin solmun kynnysarvo $\Theta$, ts. jos siihen tulevien syötteiden summan ja kynnysarvon $\Theta$ erotus on suurempi kuin nolla, antaa solmu tuloksen 1, muutoin solmun tulos on nolla.

Tarkastellaan verkon toimintaa syötteellä (0,0). Tällöin verkon tuloksen siis pitäisi olla 1. Syötekerroksen solmut (kuvassa 1 vasemmanpuoleisimmat solmut) saavat siis kummatkin syötteekseen 0. Syötekerroksen solmuina ne välittävät syötteensä suoraan seuraavalle kerrokselle. Nyt toisen kerroksen solmuille saadaan kynnysfunktiolla tulokset

f(1*0 + 1*0 - 0.5) = f(-0.5)=0 (solmun A tulos) ja

f(1*0 + 1*0 - 1.5) = f(-1.5)=0 (solmun B tulos).

Siis toisen kerroksen solmujen tulokset ovat myös nollia. Nyt tuloskerroksen (oikeanpuoleisin solmu) tulos saadaan laskemalla

f(-1*0 + 2*0 -(-0.5)) = f(0.5) = 1,

mikä on myös verkon tulos syötteellä (0,0). Kaikkia syötteitä vastaavat tulokset ovat taulukossa 1.

| Syöte | Piilokerros | | Tuloskerros |
|---|---|---|---|
| | A | B | |
| (0,0) | 0 | 0 | 1 |
| (0,1) | 1 | 0 | 0 |
| (1,0) | 1 | 0 | 0 |
| (1,1) | 1 | 1 | 1 |

Taulukko 1. Kuvan 1 neuroverkon solmujen tulokset.

Kun taulukon 1 tuloksia ja verkon rakennetta vertaa toisiinsa huomaa, että piilokerroksen solmu A tunnistaa tapaukset, joissa syötteenä on (1,0), (0,1) tai (1,1). Solmu B taas tunnistaa tapaukset, joissa syötteessä kumpikin bitti on 1. Solmu A tunnistaa siis loogisen tai-operaation ja solmu B loogisen ja-operaation. Näiden solmujen tulokset yhdistämällä tuloskerroksen solmu voi ratkaista ekvivalessiongelman antamalla tuloksen 1, jos kumpikin piilokerroksen solmu antaa saman tuloksen, ja tuloksen 0 muutoin.

Neuroverkkoarkkitehtuureita on useita, mutta verkkojen laskentaperiaate on aina samankaltainen. Vaikka kynnysfunktiot ja verkkojen rakenne vaihtelevatkin, auttaa yhden tyyppisen neurolaskentamallin ymmärtäminen myös muita, erilaisia neurolaskenta malleja. Bechtelin ja Abrahamsenin (1991) mukaan neurolaskenta-arkkitehtuureita voidaan luokitella seuraavilla tavoilla:

1. Solmujen kytkennät (verkon topologia)
2. Solmun aktivaation laskenta (kynnysfunktio)
3. Solmujen välisiä yhteyksiä muuttavan oppimisproseduurin luonne
4. Neuroverkkojen semanttinen tulkinta.

Tämän tutkielman kannalta erityisen tärkeää on neuroverkkojen rakenteen semanttinen tulkinta, sillä mallin rakenne täytyy saada vastaamaan

kuvannimeämisprossesissa mukana olevia käsitteitä. Oppimisproseduurilla ei ole merkitystä tämän tutkielman kannalta, sillä kumpikaan verkko ei ole oppiva. Erot tässä tutkielmassa esitettävien verkkojen välillä ovatkin topologiassa sekä verkkojen semanttisessa tulkinnassa, sillä kummankin verkon solmujen aktivaatiot lasketaan olennaisesti samalla tavalla.

Kummatkin luvuissa kolme ja neljä käsiteltävistä neuroverkoista ovat lokalistisia. Lokalistissa verkoissa yhtä käsitettä vastaa yksi verkon solmu. Jos verkko nimeää esimerkiksi sanoja, yksi verkon solmu voi esittää yhtä sanaa. Toinen esimerkki saman verkon alemmalta tasolta voisi olla solmu, joka vastaa tiettyä foneemia (esim. /k/). Tämä solmu aktivoituu aina kun foneemi /k/ esiintyy sanassa.

Hajautettua esitystä käyttävän verkon tapauksessa yhtä sanaa vastaa usea solmu, ja verkkoon tallennetut sanat voivat olla osittain päällekkäin. Tällöin yksi solmu vastaa piirrettä, joka voi olla yhteinen usealle sanalle. Tässä tapauksessa foneemi /k/ esitettäisiin piirteidensä avulla, jotka jakaantuisivat usean solmun osalle.

Yleensä lokalistisia malleja käytetään, kun verkon oppimiskyky ei ole kovin tärkeä simulaation ominaisuus. Tällöin järjestelmällä on sisäänrakennetut parametrit (mm. solmujen väliset yhteydet on asetettu jo valmiiksi), joita käytetään koko simulaation ajan. Jos oppiminen on tärkeä osa mallintamisprosessia, on hajautetun esityksen käyttäminen luonnollista. Tämä johtuu siitä, että hajautetuille malleille on olemassa tehokkaista oppimisalgoritmeja, joilla opetus voidaan hoitaa. On myös olemassa mallintamistehtäviä, joissa voidaan käyttää sekä hajautettuja että lokalistisia malleja [Martin et al., painossa].

**Miksi neuroverkot soveltuvat kielenhäiriöiden mallintamiseen?**

Yksi konnektionististen arkkitehtuurien suurista eduista on niiden vaurionkestävyys [Bechtel and Abrahamsen, 1991]. Tällä tarkoitetaan sitä, että vaikka osaa mallin solmujen välisistä yhteyksistä vaurioitettaisiin tai poistettaisiin kokonaan, niin mallilla voi silti tämän jälkeen laskea. Sama koskee solmujen poistoa. Vaikka mallin tulokset eivät ole vaurion jälkeen enää välttämättä oikeita, ovat ne kuitenkin oikeansuuntaisia. Täten vaurioitunutkin neuroverkko antaa paremman tuloksen kuin neuroverkko, jonka painot on asetettu täysin satunnaisesti (olettaen tietenkin, että vaurio ei ole täysin satunnaistanut verkon toimintaa). Esimerkiksi loogisen ekvivalenssin ratkaisevasta neuroverkosta voitaisiin poistaa joko solmu A tai B, ja verkko antaisi silti vastauksia syötteillä. Jos solmu A poistettaisiin, tulossolmuun tulisi syötteitä vain solmulta B. Tällöin verkon tulos olisi 1 kaikilla syötteillä (ks. kuva 1).

Edellä kuvattua ominaisuutta, jossa neuroverkkojen suoritus heikkenee, mutta ei lopu, kutsutaan sulavaksi taantumaksi (graceful degradation) [Bechtel and Abrahamsen, 1991]. Koska sulava taantuma on verkon rakenteesta periytyvä

ominaisuus, on jokaisella neuroverkolla on tämä ominaisuus. Myös aivot taantuvat sulavasti. Esimerkiksi aivovaurion yhteydessä ihmisen puheentuottokyky saattaa kärsiä, mutta se ei useinkaan katoa kokonaan. Aivovauriopotilaan artikulaatio voi olla häiriintynyttä, tai potilaalla voi olla vaikeuksia löytää haluamiansa sanoja, mutta potilas pystyy silti puhumaan jollakin tavalla.

Sulavaa taantumaa on vaikea toteuttaa perinteisesti ohjelmoimalla, sillä mukaan tulevien rajoitusten ja poikkeusten määrä kasvaa nopeasti hyvin suureksi. Neuroverkoissa nämä poikkeukset ja rajoitukset sisältyvät verkkojen painoarvoihin ja topologiaan, joten neuroverkoilla asiaan ei tarvitse kiinnittää erityistä huomiota [Bechtel and Abrahamsen, 1991].

## 3. Interaktiiviseen aktivaatioon perustuva neuroverkko

Interaktiivisen aktivaation teoria nimeämisestä (IA) perustuu oletukseen, että nimeäminen tapahtuu kahdessa eri vaiheessa. Ensimmäisessä vaiheessa haetaan sanan lemma ja toisessa sanan foneemit. Teorian mukaan kumpaankin vaiheeseen osallistuu koko puheentuottojärjestelmä. Haettaessa lemmaa siihen liittyvät foneemit alkavat myös aktivoitua ja vaikuttavat lemman valintaan takaisinsyöttävien yhteyksien avulla. Fonologisen haun aikana taas foneemit lähettävät aktivaatiota takaisin lemmalle, josta on yhteyksiä takaisin foneemeihin. Täten sanan lemmanhakuvaiheessa haettava lemma saa lisäaktivaatiota siihen liittyviltä foneemeilta, ja fonologisen haun yhteydessä foneemit saavat lisäaktivaatiota lemmalta.

**Verkon arkkitehtuuri**

IA-malli koostuu kolmesta kerroksesta, joiden välillä on kaksisuuntaisia yhteyksiä. Kerrosten sisäisiä yhteyksiä ei sen sijaan ole. Syötekerroksena toimii semanttinen kerros. Semanttinen kerros koostuu nimettävän kohteen semanttisista piirteistä, semeemeistä. Jokaiselle verkkoon tallennetulle sanalle on asetettu 10 semeemisolmua. Semeemikerroksesta on kaksisuuntaisia yhteyksiä sanakerrokseen. Sanakerros edustaa nimettävän sanan lemmaa. Tuloskerroksena toimii foneemikerros, joka on jaettu kolmeen osaan: alkukonsonanttiin, vokaaliin ja loppukonsonanttiin. Tulos- ja sanakerroksen välillä on myös kaksisuuntaisia yhteyksiä. Kuvassa 2 on esitetty IA-verkon rakennetta.

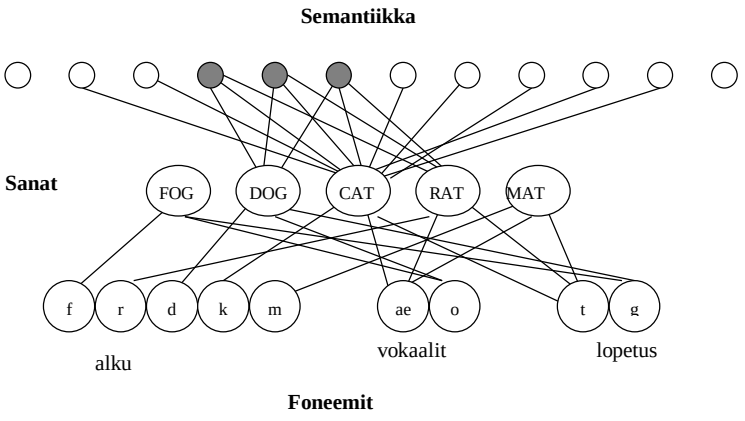
Solmun *j* tulos kierroksella *t* lasketaan kaavalla

$$\mathrm{A}_j(t) = \mathrm{A}_j(t-1)\cdot(1-q) + \sum_i^n p_{i,j}\,\mathrm{A}_i(t-1) + noise, \qquad (1)$$

missä $\mathrm{A}_i(t\text{-}1)$ on solmun *i* aktivaatio kierroksella *t*-1, *q* on vaimenemisaste (decay rate), $p_{i,j}$ on solmujen *i* ja *j* välinen paino ja *noise* satunaiskohina [Dell et al., 1996]. Satunaiskohina koostuu kahdesta osasta: verkon luontaisesta satunaiskohinasta sekä

aktivaatioon liittyvästä satunaiskohinasta. Verkon luontaisen satunaiskohinan keskiarvo on nolla ja keskihajonta on jokin muuttuja *SD1*. Aktivaatioon liittyvän satunaiskohinan keskiarvo on myöskin nolla ja sen keskihajonta lasketaan kaavalla *SD2**$A_j(t)$, missä *SD2* on jokin reaaliluku. Simulaation aikana *SD1* ja *SD2* olivat vakioita.

Verkko on toteutettu kahtena kuuden sanan naapurustona [Dell et al., 1997]. Kummassakin naapurustossa on kohdesanana "cat". Ensimmäisessä naapurustossa on kohdesanan lisäksi sille semanttisesti läheinen sana, kaksi muodoltaan samankaltaista sanaa, sekä kaksi kohdesanaan liittymätöntä sanaa. Toisessa naapurustossa on kohdesanan lisäksi kohdesanalle semanttisesti läheinen sana, yksi muodollisesti läheinen sana, yksi sana, joka on sekä semanttisesti että muodollisesti läheinen sana, sekä kaksi kohdesanaan liittymätöntä sanaa. Sanojen pituus on rajoitettu yhteen tavuun, jotta ne voitaisiin koodata kolmella foneemilla (alkukonsonantti, vokaali, loppukonsonantti).




Kuva 2. IA-neuroverkon rakennetta [Dell et al., 1997].

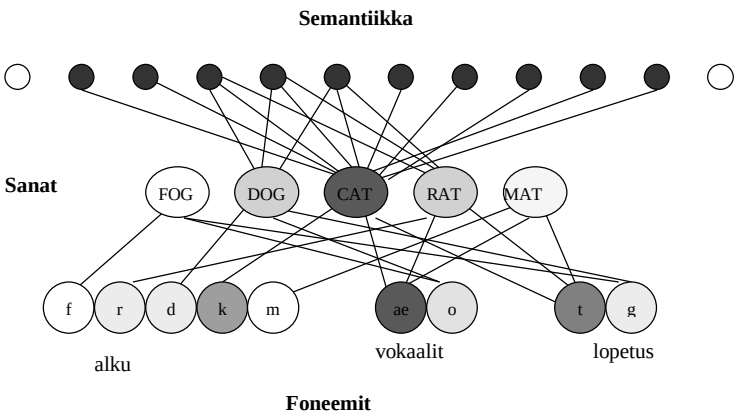
Verkon syötekerroksena siis toimii semanttinen kerros. Ensimmäisessä naapurustossa kohdesanalla ja tämän semanttisella naapurilla on kolme yhteistä semanttista piirrettä. Toisessa naapurustossa kohdesanalla on edellisen lisäksi kolme yhteistä semanttista piirrettä siihen liittyvän semanttisen sekä muodollisen naapurin kanssa [ibid.]. Kuvassa 2 on esitetty sanan "cat" semanttiset piirteet. Harmaalla merkityt semanttiset piirteet kuuluvat sanan "cat" lisäksi sanalle "dog", joka on sanan "cat" semanttinen naapuri, ja sanalle "rat", joka on sanan "cat" sekä semanttinen että muodollinen naapuri. Muut sanaan "cat" liittyvät semanttiset piirteet taas erottavat sen sanoista "dog" ja "rat".

Foneemikerroksen naapurustoihin on sisällytetty kaikki mahdolliset foneemit, joita verkkoon tallennetuilla sanoilla voi olla. Sanoissa esiintyvät foneemit on jaettu kolmeen luokkaan: alkukonsonanttiin, vokaaliin sekä loppukonsonanttiin. Jokaisesta

sanakerroksen solmusta on yhteys sen alkukonsonanttiin, vokaaliin sekä loppukonsonanttiin [Dell et al., 1996]. Täten esimerkiksi sanalla "cat" on yhteys foneemeihin /k/ (alkukonsonantti), /ae/ (vokaali) sekä /t/ (loppukonsonantti), jolloin sanan "cat" fonologinen muoto on /kaet/. Kohdesanalla ja sen muodollisella naapurilla on yhteisiä foneemeja. Esimerkiksi sanat "mat" ja "cat" eroavat vain yhdellä foneemilla, nimittäin alkukonsonantilla /m/ - /k/. Niiden semantiikalla ei kuitenkaan ole mitään tekemistä keskenään. Tämä mahdollistaa muodollisten virheiden esiintymisen verkossa (esim. "cat"→"mat") [Dell et al., 1996].

Simulaatio käynnistyy, kun nimettävän sanan (kohdesanan) semanttiset piirteet aktivoidaan semanttisesta kerroksesta. Aktivaatiota levitetään kaavan (1) mukaisesti *n* kierrosta, jonka jälkeen valitaan sanakerroksesta korkeimmin aktivoitunut solmu. Valittu solmu ei välttämättä ole solmu, jonka semanttiset piirteet on aktivoitu simulaation alussa, sillä semanttisen tason solmut ovat yhteydessä myös muihin solmuihin kuin kohdesanaan. Tällöin myös muut kuin kohdesanat aktivoituvat sanakerroksessa. Lisäksi kaikki aktivoituneet sanakerroksen solmut levittävät aktivaatiota takaisin niitä vastaaviin semanttisen kerroksen solmuihin sekä eteenpäin fonologisen kerroksen solmuihin, jotka vahvistavat niihin liitettyjä sanakerroksen solmuja.

Kun sanakerroksen voimakkaimmin aktivoitunut solmu on valittu, nollataan kaikkien solmujen aktivaatiot. Tämän jälkeen valittu solmu aktivoidaan uudestaan, ja simulaatio jatkuu samalla tavalla aktivaatiota levittäen *m* kierrosta, jonka jälkeen valitaan foneemikerroksesta korkeimmin aktivoitunut solmu kustakin foneemipaikasta (tavun alku- ja loppukonsonantti sekä vokaali). Näin saatu sanan fonologinen esitys on verkon tulos. Aktivaation leviämistä on havainnollistettu kuvassa 3. Alussa on aktivoitu sanan "cat" semanttiset piirteet. Mitä tummempi solmun väri on, sitä suurempi on sen aktivaatiotaso. Kuvassa ei esitetä semanttisia piirteitä, jotka sanojen "dog" ja "rat" aktivoituminen on aiheuttanut, eikä näiden ei-kuvattujen semanttisten piirteiden edelleen aiheuttamia aktivoitumisia.



Kuva 3. Aktivaation leviäminen.

Kuvan 3 verkossa on aktivoitu sanan "cat" semantiikkaa vastaavat solmut. Kun aktivaatiota on levitetty muutamia kierroksia, on sanakerroksen solmuista sanan "cat" lisäksi aktivoituneet sanat "dog" ja "rat", koska niillä on yhteisiä semanttisia piirteitä sanan "cat" kanssa. Niiden aktivaatiotaso ei ole kuitenkaan yhtä suuri kuin sanan "cat", sillä tämä saa lisäaktivaatiota myös seitsemältä muulta semanttiselta piirteeltä, jotka siihen liittyvät. Sanakerroksen aktivoituneet sanat ovat vuorostaan alkaneet levittää aktivaatiota niitä vastaaviin foneemikerroksen solmuihin. Tavun alkuosaa vastaavista solmuista foneemia /k/ vastaava solmu on aktivoitunut voimakkaimmin saadessaan aktivaatiota sanakerroksen voimakkaimmin aktivoituneelta solmulta "cat". Foneemeja /r/ ja /d/ vastaavat solmut ovat alkaneet myös aktivoitua saadessaan aktivaatiota vähemmän aktivoituneilta sanakerroksen solmuilta "dog" (semanttinen naapuri) ja "rat" (semanttinen ja muodollinen naapuri). Samalla tavalla voidaan tulkita myös tavun vokaali- ja lopetusosaa vastaavien solmujen aktivaatiota. Vokaali /ae/ on kuitenkin aktivoitunut muita voimakkaammin saadessaan syötteitä sekä sanalta "cat" että "rat". Mielenkiintoista on kuitenkin huomata, että alkuperäiseen syötteeseen semanttisesti täysin liittymätön sana "mat" (muodollinen naapuri) on alkanut aktivoitua. Tämä johtuu siitä, että sitä vastaavista foneemikerroksen solmuista ovat aktivoituneet solmut /ae/ ja /t/, jotka lähettävät aktivaatiota sanakerroksen solmulle "mat".

Esimerkistä ilmenee, miten verkko voi tuottaa nimeämisvirheitä. Jos nimittäin lemman valinnan yhteydessä jokin muu kuin kohdesana tulee valituksi aiheutuu joko semanttinen, muodollinen, sekoittunut (semanttinen ja muodollinen) tai liittymätön virhe riippuen siitä, mikä on valitun sanan suhde kohteeseen. Fonologisen valinnan aikana voi syntyä epäsanoja (sanoja, joita verkossa ei ole) tai muodollisia virheitä samalla periaatteella kuin lemman valinnan yhteydessä. Kun verkko on parametrisoitu terveiden nimeämisten mukaan, virheiden todennäköisyys on tietenkin pieni. Aiheutettaessa verkolle vaurio, ts. muuttamalla verkon painoarvoja sekä vaimenemiskerrointa, virheiden todennäköisyys kasvaa ja verkko alkaa tuottaa oikeiden sanojen ohella edellä kuvattuja virheitä.

**Tuloksia**

Mallin evaluointia varten sen suoritus parametrisoitiin ensin mahdollisimman lähelle terveiden ihmisten nimeämistä. Tämän jälkeen mallia parametrisoitiin mahdollisimman lähelle 23 afasiapotilaan nimeämistä [Dell et al., 1996]. Parametrisointi suoritettiin vaihtelemalla vaimenemiskerrointa ja solmujenvälisten yhteyksien painoarvoja. Ensimmäinen naapurusto oli nimeämisen kohteena 90 % kerroista ja toinen 10 %. Tällä tavalla saatiin aikaan virheiden oikeantyyppinen

jakautuminen. Jotta simulaatiosta ei olisi tullut liian monimutkainen, oletettiin afasiapotilaiden aivovaurioiden olevan kokonaisvaltaisia, jolloin vaimenemiskertoimen ja painoarvojen vaihtelut kohdistettiin koko verkkoon samalla tavalla. Mallin sovitus potilasaineistoon onnistui hyvin, eli malli saatiin tuottamaan hyvin samankaltaisia virhejakaumia kuin sovitettava potilas [ibid.]. Lisäksi potilaat oli mahdollista luokitella mallin painoarvojen perusteella suuri- ja pienipainoisiksi. Suuripainokertoimisten potilaiden ennustettiin tekevän enemmän sekoittuneita virheitä, koska aktivaation leviäminen on näillä potilailla voimakkaampaa. Tämä ennustus vahvistettiin kokeellisesti.

IA-mallin kykyä mallintaa toipuvia potilaita testattiin yrittämällä parametrisoida se mahdollisimman lähelle toipilaiden suoritusta. Jos potilaan vaurio oli luokiteltu painoarvoihin kohdistuneeksi, pitäisi mallin pystyä simuloimaan toipilaan nimeämistä palauttamalla painoarvoja lähemmäksi normaaleille parametrisoituja arvoja. Sama koskee myös muita vauriotyyppejä. Lisäksi potilaan luokituksen ei pitäisi toipumisen yhteydessä muuttua vauriotyypistä toiseen, vaan toipumisen pitäisi tapahtua verkossa aidosti. Esimerkiksi jos potilaalla on ennen toipumista mallin ennustuksen mukaan normaalit painoarvot, mutta korkea vaimenemiskerroin, toipuneeseen potilaaseen sovitetun mallin ei pitäisi ennustaa vauriota, jossa painoarvot ovat vaurioituneet ja vaimenemiskerroin palautunut normaaliksi [Dell et al., 1997]. IA-mallilla tehtyjen simulaatioiden tulokset olivat yhteneviä näiden ennusteiden kanssa.

## 4. SLIPNET - DTS-mallin eräs toteutus

Toisen mallin, erillisen kaksivaihemallin (discrete two-stage model, DTS), oletuksien mukaan nimeämisprosessin kaksi vaihetta ovat täysin erillisiä. Sanan lemma siis aktivoidaan ilman fonologisen tason vaikutusta. Lisäksi aktivoituneista lemmoista valitaan suurimman aktivaation omaava lemma, jonka fonologinen esitys aktivoidaan fonologisella tasolla täysin erillään edellisestä tasosta [Laine et al., 1998]. DTS-mallissa kerrosten väliset yhteydet ovat täten vain eteenpäin syöttäviä, eli ne syöttävät ylemmältä prosessointikerrokselta alemmalle. Kerrosten sisäisillä yhteyksillä on DTS-mallissa keskeinen rooli, sillä aktivaation levitys tapahtuu DTS-malleissa niitä pitkin. Tässä tarkasteltava DTS-malli SLIPNET on toteutettu suomenkielisten afasiapotilaiden nimeämishäiriöiden mallintamiseen.

### Arkkitehtuuri

Toisin kuin IA-mallissa, kerrosten sisäisillä yhteyksillä on siis DTS-mallissa keskeinen rooli. Prosessointi etenee vaiheittain alkaen lemman hausta päätyen sanan fonologisen esityksen aktivoimiseen [Laine et al., 1998]. Kummassakin vaiheessa aktivaatiota levitetään kerroksen sisällä, jonka jälkeen valitaan korkeimmin

aktivoitunut solmu kerroksen tulokseksi ja välitetään tämä eteenpäin seuraavalle kerrokselle.

SLIPNETin solmujen välisten yhteyksien painot on rajoitettu neljään eri tyyppiin: vahvoihin (paino $p$=0.4), kohtuullisiin ($p$=0.2) ja heikkoihin ($p$=0.15) yhteyksiin. Neljännellä painotyypillä merkitään puuttuvia yhteyksiä ($p$=0) [Juhola and Laine, 1992]. Painoarvot määrättiin leksikaalis-semattisessa verkossa terveiden koehenkilöiden arvioiden perusteella. Foneemiverkon painoarvot määrättiin niiden lingvististen ominaisuuksien perusteella [Tikkala and Juhola, 1995].

Aktivaation levityksen yhteydessä ei käytetä kynnysfunktiota, vaan solmun tulos lasketaan suoraan lisäämällä solmun vanhaan tulokseen sen saamien uusien syötteiden summa. Summa siis lasketaan lisäämällä solmun vanhaan tulokseen siihen yhteydessä olevien solmujen tulos yhteyden painoarvolla kerrottuna. Saatu summa kerrotaan vielä vaimenemiskertoimella (decay rate), jolloin saadaan solmun tulos. Täten siis solmun $j$ tulos ajanhetkellä $t$ voidaan laskea kaavalla

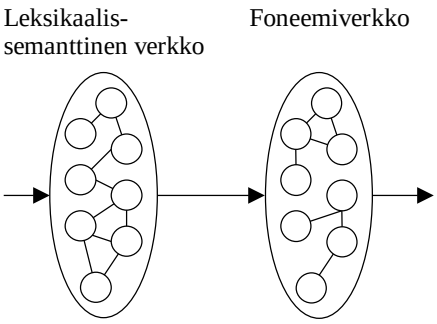
$$\mathrm{A}_j(t) = (\mathrm{A}_j(t-1) + \sum_{i=1, i\neq j}^{n} p_{ij}\,\mathrm{A}_i(t-1))(1-q),$$

missä $q$ on vaimenemisaste (decay rate) ja $\mathrm{A}_i(t\text{-}1)$ on solmun i aktivaatio ajanhetkellä $t$-1. Vaurioitunutta nimeämistä simuloidaan lisäämällä solmujen aktivaatioarvoihin normaalijakautunutta satunaiskohinaa. Vaurioituneen solmun $\mathrm{A'}_j$ aktivaatio ajanhetkellä $t$ voidaan laskea kaavalla

$$A'_j(t) = (1 + noise_{jt})A_j(t),$$

missä $noise_{jt}$ on solmun $j$ satunnaiskohina ajanhetkellä $t$ [Juhola and Laine, 1992].

SLIPNET koostuu kahdesta aliverkosta: leksikaalis-semattisesta verkosta ja foneemiverkosta. Ensimmäinen verkko simuloi lemman hakua ja toinen sanan fonologian hakua. Kuvassa 4 on esitetty SLIPNETin rakenne.




Kuva 4 SLIPNET-neuroverkon rakenne.

Leksikaalis-semanttinen verkko simuloi nimettävän sanan semantiikan hakemista. Verkossa on 27 sanaa, joista 10 on simulaatiossa käytettäviä perussanoja ja 17 näiden semanttisia sukulaisia. Esimerkiksi verkon perussanastoon kuuluvalla sanalla "aasi"

on semanttinen sukulainen "muuli". Leksikaalis-semanttisessa aliverkossa solmujen välillä on yhteyksiä semanttisin perustein. Jos sanat ovat semanttisesti läheisiä, on niiden välillä voimakas yhteys. Jos taas ovat semanttisesti kaukaisia, ei niiden välillä ole yhteyttä ollenkaan.

Foneemiverkon tehtävä on simuloida yksittäisten foneemien tuottamista. Foneemiverkko jakautuu kahteen aliverkkoon, vokaaliverkkoon sekä konsonanttiverkkoon, sillä kokeellisesti on huomattu, että konsonantin muuttuminen vokaaliksi (tai toisinpäin) on erittäin harvinaista [Tikkala and Juhola, 1995]. Foneemiverkkoon on sisällytetty kaikki foneemit, jotka esiintyvät leksikaalis-semanttisen verkon sanoissa.

Simulaation kulku on seuraavanlainen [Laine et al., 1998]:

1) Simulaation alussa aktivoidaan leksikaalis-semanttisen aliverkon solmu, joka vastaa nimettävänä olevan sanan semantiikkaa. Tämän jälkeen solmun annetaan levittää aktivaatiota leksikaalis-semanttisessa aliverkossa muutamia kierroksia.
2) Leksikaalis-semanttisesta aliverkosta valitaan se solmu, jonka aktivaatio on korkein. Tämä arvo kynnystetään. Jos valitun solmun aktivaatio $A_{max}$ ylittää kynnysarvon *h*, annetaan tulokseksi $A_{max}$, muutoin 0. Siis

$$A = \begin{cases} A_{max}, \text{ jos } A_{max} \geq h \geq 0 \\ 0, \text{ muutoin} \end{cases}.$$

3) Foneemialiverkosta aktivoidaan lemmaa vastaavat foneemit yksitellen. Aktivaation suuruuden määrää leksikaalis-semanttisen tason kynnystetty maksimiarvo. Foneemiverkko simuloi jokaisen foneemin hakua erikseen aktivaation levityksen avulla. Jos syötteen arvo on 0, verkko ei tuota mitään vastausta. Foneemiverkon tuottamia tuloksia ei kynnystetä.

Nimeämisvirheitä SLIPNET-neuroverkolla tuotettiin lisäämällä satunaiskohinaa verkon leksikaalis-semanttiseen ja fonologiseen kerrokseen sekä muuntelemalla kerrostenvälistä kynnysarvoa. Satunaiskohinan lisääminen leksikaalis-semanttiseen verkkoon lisää todennäköisyyttä, että väärän sanan lemma tulee valituksi aiheuttaen semanttisen virheen. Vastaavasti satunaiskohinan lisääminen foneemiverkkoon aiheuttaa fonologisia virheitä. Foneemien aktivoinnin yhteydessä sattuva virhe voi johtaa neologismiin eli, kieleen kuulumattomaan sanaan. Kynnysarvoon liittyvä virhetyyppi on "ei-vastausta"-vastaukset. Näitä sattuu, kun leksikaalis-semanttisen kerroksen tulokseksi valitun solmun aktivaatio ei ylitä kynnysarvoa. "Ei-vastausta"-vastauksilla voidaan mallintaa "kielen päällä"-ilmiötä (tilanne, jossa henkilö osoittaa tietävänsä sanan merkityksen, mutta ei saa sitä sanotuksi) [Laine et al., 1998]. Lisäksi

SLIPNET tuottaa muita virheitä, jotka eivät ole luokiteltavissa oikeiksi vastauksiksi tai joksikin edellä mainituista virhetyypeistä. Tärkeimpänä ryhmän virheistä voidaan pitää semanttis-fonologisia virheitä, eli virheitä, joita sattuu kun, satunaiskohinaa lisätään kumpaankin SLIPNET-neuroverkon aliverkkoon.

**Tuloksia**

SLIPNET-neuroverkon kykyä simuloida potilaiden nimeämisvirheitä testattiin vertaamalla sen suoritusta kymmeneen afasiapotilaaseen. Potilaat kärsivät neljästä erityyppisestä afasiaoireyhtymästä, ja heidän nimeämistulokset vaihtelivat [Laine et al., 1998]. Esimerkiksi joillakin potilailla neologistiset virheet muodostivat suurimman osan kaikista virheistä, kun taas toisilla suurin osa virheistä muodostui "ei vastausta"-virheistä. Potilaiden nimeämistestien tulokset oli luokiteltu edellisessä luvussa kuvattujen SLIPNETin virheluokituksen mukaisesti. SLIPNET parametrisoitiin simuloimaan jokaista potilasta erikseen vaihtelemalla satunnaiskohinan määrää kummassakin aliverkossa, sekä kynnysarvoa aliverkkojen välillä [Laine et al., 1998].

SLIPNETin parametrisointi eri potilaiden suoritusta vastaavaksi onnistui hyvin ja erot potilaan ja verkon tuottamien nimeämistulosten välillä eivät olleet merkitseviä [ibid.].

## 5. Tuloksista

Vaikka verkkojen vaurioittamistavat ja topologiat olivatkin erilaisia kummankin, verkkoarkkitehtuurin avulla oli mahdollista simuloida tarkasti eri tyyppisten afasiapotilaiden nimeämisvirheitä. IA-verkon avulla pystyttiin mallintamaan myös toipuneiden potilaiden nimeämistä. IA-malliin vauriot aiheutettiin muuntelemalla vaimenemiskerrointa ja yhteyksien välisiä painoarvoja, kun taas DTS-malli SLIPNETiä vaurioitettiin lisäämällä yhteyksiin normaalijakautunutta satunaiskohinaa. SLIPNET-neuroverkon yhteyksien painoarvoja voidaan lisäksi pitää tarkempina kuin IA-mallin, jossa painoarvot olivat samoja koko verkossa.

Kuitenkin SLIPNETin rakenteesta johtuen sillä ei voitu tuottaa sekoittuneita virheitä, joita terveillä koehenkilöillä on havaittu. Kyvyttömyys sekoittuneiden virheiden tuottamiseen SLIPNET-verkkolla johtuu siitä, että foneemikerroksen ja leksikaalis-semanttisen kerroksen välillä ei ole takaisinsyöttäviä yhteyksiä. Toisaalta Laine ja muut (1998), eivät löytäneet sekoittuneita virheitä käyttämästään kymmenen potilaan aineistosta. IA-mallihan tuotti sekoittuneita virheitä ilman lisäasetuksia.

Toinen virhetyyppi, jota SLIPNET-neuroverkolla ei voida simuloida, on fonologisen haun yhteydessä sattuvat semanttiset virheet. SLIPNET ei kykene tuottamaan näitä virheitä, sillä virheen tuottaminen vaatisi fonologisen kerroksen

vuorovaikutusta leksikaalis-semanttisen kerroksen kanssa foneemien haun aikana. IA-mallin avulla taas tällaisia virheitä on mahdollista tuottaa, sillä vaikka sanan oikea lemma olisi lemmanhakuvaiheessa aktivoitunut, vuorovaikutus kerrosten välillä mahdollistaa semanttisten virheiden synnyn vielä fonologisen haun aikana. Edellä mainitut SLIPNET-verkon puutteet ja IA-mallin vahvuudet tukevat selvästi interaktiivisen aktivaation teoriaa nimeämisestä ja heikentävät erillistä kaksivaiheteoriaa [Laine et al., 1998].

Kumpikin esitellyistä malleista kärsii simulaatiossa mukana olevien sanojen vähyydestä. On kyseenalaista, voiko muutamilla sanoilla suoritetuista onnistuneista simulaatiosta tehdä yleisiä päätelmiä nimeämisprosessin kulusta. IA-mallin heikkoutena voidaan myös pitää sitä, että simulaatiossa mukana olevat sanat olivat yksitavuisia. Tämä yksipuolistaa entisestään simulaatiossa käytettävää sanajoukkoa. SLIPNETissä sanojen pituutta ei tällä tavalla rajattu ja simulaatiossa esiintyi myös pitkiä suomenkielisiä yhdyssanoja, kuten “pyykkipoika”.

Lukujen kolme ja neljä esimerkkiverkot ovat kuitenkin tarkoitettu nimenomaan malleiksi afasiapotilaiden nimeämishäiriöistä. Täten niiden tarkoitus ei ole olla täydellisesti nimeämistä simuloivia. Yleensäkin nimeämistä täydellisesti simuloivan mallin toteuttaminen olisi mahdotonta, sillä jo sadan sanan neuroverkkosimulaation tekeminen on hyvin monimutkaista.

## 6. Yhteenveto

Tutkielmassa kartoitettiin nimeämishäiriöiden mallintamista neuroverkoilla. Esitellyt mallit simuloivat kuvannimeämistä semantiikasta fonologiaan jättäen mallien ulkopuolelle esineen havaitsemisen ja artikulaation. Mallit perustuvat eri teorioihin kaksivaiheisesta nimeämisestä, ja siksi niiden rakenne on erilainen. Kumpikin malli perustuu kuitenkin samanlaiselle aktivaationlevitysperiaatteelle. Itse asiassa IA-malli on johdettu Dellin (1986) esittelemästä lauseiden tuottamista simuloivasta mallista ja SLIPNET käyttää samaa aktivaationlevityskaavaa kuin Dellin (1986) malli. IA-mallissa aktivaationlevitys on vertikaalista, eli aktivaatio leviää kerrosten välillä, kun taas DTS-mallin toteuttavassa SLIPNET-neuroverkossa aktivaationlevitys on horisontaalista.

Kumpikin varsinaista nimeämistä käsittelevä malli kärsii sanojen puutteesta. Tämä voi olla ongelmallista, sillä sanojen vähyyden vuoksi aineiston pienetkin vaihtelut voivat vaikuttaa verkkojen tuloksiin. Lisäksi SLIPNET-neuroverkko ei kyennyt mallintamaan kaikkia virhetyyppejä, joita afaatikoilla on havaittu. Tämä tekee IA-mallista uskottavamman suhteessa DTS-mallin (SLIPNET) ja vahvistaa samalla interaktiivisen aktivaation teoriaa nimeämisestä.

Toistaiseksi nimeämishäiriöitä on mallinnettu vain lukujen kolme ja neljä tyyppisillä lokalistisilla malleilla. Mielenkiintoinen tutkimusaihe olisikin mallintaa nimeämishäiriötä oppivalla neuroverkkomallilla. Oppiva malli tarjoaisi mallista itsestään mahdollisesti nousevien uusien ominaisuuksien lisäksi mahdollisuuden mallintaa potilaiden toipumista vaurion jälkeen uudella tavalla. Lokalistisilla malleilla voidaan kyllä mallintaa potilaiden toipumista sovittamalla verkon parametrit sopivasti vastaamaan toipuneiden potilaiden nimeämistä, kuten tehtiin IA-mallin kohdalla. Oppivalla neuroverkolla olisi tämän lisäksi mahdollista tutkia potilaan kuntouttamisessa käytettävien sanojen vaikutusta kuntoutumiseen. Tällainen malli on tehty jo dysleksiasta (ääneenlukemisen aivoperäinen häiriö) kärsivien potilaiden osalta [Plaut, 1996]. Mallille opetettiin ensin perussanasto, jonka jälkeen sitä vaurioitettiin. Vaurion jälkeen mallia "kuntoutettiin" eli uudelleenoperettiin samalla oppimisalgoritmilla, jolla se oli alunperin opetettu. Uudelleenopettamisen yhteydessä ilmeni, että kuntouttaminen oli tehokkainta käytettäessä semanttisesti keskeisiä sanoja uudelleenopetusjoukkona. Semanttisesti keskeisten sanojen käyttö opetusjoukossa sai myös siinä esiintymättömien sanojen lukemisen parantumaan. Mielenkiintoista olisikin tutkia, esiintyisikö nimeämishäiriötä simuloivan mallin uudelleenopettamisen yhteydessä saman tyyppisiä ilmiöitä. Tällaisella tiedolla olisi merkitystä myös käytännössä potilaiden kuntouttamisen kannalta.

## Viiteluettelo


[Bechtel and Abrahamsen 1991] William Bechtel and Adele Abrahamsen, *Connectionism and Mind. An Introduction to Parallel Processing in Networks.* Basil Blackwell, 1991.

[Dell] Gary S. Dell, A spreading-activation theory of retrieval in sentence production. *Psychol. Rev.* **93**, 3 (1986), 283-321.

[Dell et al., 1996] Gary S. Dell, Myrna F. Schwartz, Nadine Martin, Eleanor M. Saffran and Deborah A. Gagnon, A connectionist model of naming errors in aphasia. In: James A. Reggia, Eytan Ruppin and Rita Sloan Berndt (eds.), *Neural Modelling of Brain and Cognitive Disorders.* World Scientific Publishing, 1996, 135-156.

[Dell et al., 1997] Gary S. Dell, Myrna F. Schwartz, Nadine Martin, Eleanor M. Saffran and Deborah A. Gagnon, Lexical access in aphasic and nonaphasic speakers. *Psychol. Rev.* **104**, 4 (1997), 801-838.

[Juhola, 1998] Martti Juhola, *Neurolaskenta*. Kurssimateriaali, Tietojenkäsittelyopin laitos, Tampereen yliopisto, 1998.

[Juhola and Laine, 1992] Martti Juhola and Matti Laine, Preliminary simulation of aphasic naming errors with a network model. In: Eero Hyvönen, Jouko

Seppänen and Markku Syrjänen (eds.), *New Directions of Artificial Inteligence* **2**, 224-230. *Proc. of the Finnish Artificial Inteligence Conference.*

[Laine et al., 1992] Matti Laine, Päivi Kujala, Jussi Niemi and Esa Uusipakka, On the nature of naming dificulties in aphasia. *Cortex* **28** (1992), 537-554.

[Laine ja Marttila, 1992] Matti Laine ja Reijo Marttila, Aikuisen afasia. *Duodecim* **108** (1992), 1039-1047.

[Laine et al., 1998] Matti Laine, Anneli Tikkala ja Martti Juhola, Modelling anomia by the discrete two-stage word production architecture. *J. Neurolinguistics* **11**, 3 (1998), 275-294.

[Martin et al., in press] Nadine Martin, Matti Laine and Trevor Harvey, How can cognitive models of language inform models of language rehabilitation? In: Hillis A. (ed.), *Handbook on Adult Language Disorders*, Psychology Press.

[Plaut, 1996] David C. Plaut, Relearning after damage in connectionist networks: toward a theory of rehabilitation. *Brain and Language* **52** (1996), 25-82.

[Tikkala and Juhola, 1995] Anneli Tikkala and Martti Juhola, A neural network simulation method of aphasic naming errors: properties and behavior. *Neural Computing & Applications* **3** (1995), 191-201.

# Laadunvarmistus ohjelmistoprojektissa

## Hanna Leinonen

**Tiivistelmä.**

Tässä tutkielmassa tarkastellaan ohjelmistoprojektiin liittyviä toimintoja laadunvarmistuksen kannalta. Painopiste on keskeisimpien laadunvarmistusmenetelmien esittelyssä. Tarkastelun kohteena ovat ohjelmistoprojektin eri vaiheissa suoritettavat laadunvarmistustoiminnot ja projektin taustalla tehtävä laadunvalvonta. Tutkielman tarkoituksena on tuoda esille laadunvarmistuksen rooli yhtenä ohjelmistoprojektin tärkeimmistä tekijöistä. Laatuvaatimukset koskevat niin määrittely- ja suunnitteludokumentteja kuin valmista ohjelmaakin. Laatuasioihin tulee tämän vuoksi kiinnittää huomiota projektin jokaisessa vaiheessa.

Avainsanat: laatu, laadunvarmistus, laatutyö.
CR-luokat: D.2.9.

## 1. Johdanto

Laatu on käsitteenä moniselitteinen ja sen täsmällinen määrittely on käytännössä mahdotonta. Tämä koskee myös ohjelmistotuotteita ja niiden arviointia. Tästä huolimatta – tai ehkäpä juuri tämän vuoksi – ohjelmistoalalla on alettu korostaa tuotteen laadun takaamiseen pyrkivien menetelmien tärkeyttä. Lopputuotteen laadun lisäksi on kiinnitettävä huomiota toiminnan laatuun. On suoritettava toimenpiteitä, joilla voidaan varmistua siitä, että tuote täyttää sille asetetut laatuvaatimukset. Näitä toimenpiteitä kutsutaan laadunvarmistukseksi.

Tässä tutkielmassa tarkastelen ohjelmistoprojektiin liittyviä toimintoja laadunvarmistuksen kannalta. Aluksi esittelen näkökulmia ohjelmistotuotteen laadun määrittelyyn ja luon katsauksen vallitseviin laatustandardeihin. Pääosin tutkielmassa kuitenkin keskitytään keskeisimpien laadunvarmistusmenetelmien läpikäymiseen. Tarkastelun kohteena ovat ohjelmistoprojektin eri vaiheissa suoritettavat laatutoiminnot ja projektin taustalla tehtävä laadunvalvonta. Lopuksi esittelen joitakin menetelmiä tuotteen ja samalla koko projektin laadun arvioimiseksi.

# 2. Laadun määritteleminen

## 2.1. Näkökulmia laadun määrittelyyn

Termillä laatu on monia eri tulkintoja. Puhekielessä laatukäsitteeseen kuuluu tavallisesti positiivinen ennakkoasenne: laadusta puhuttaessa puhuja käyttää sitä usein synonyymina käsitteelle hyvä tai korkeatasoinen. Laatuna pidämme mm. virheettömiä ohjelmia ja ajan tasalla olevia dokumentteja sekä ohjelman helppokäyttöisyyttä, suoritustehokkuutta ja luotettavuutta. Tuotteen sisältämät ominaisuudet ovat perusosa laatumielikuvaa, koska yleensä tuote on hankittu tyydyttämään jotain tarvetta.

Eräs tapa määritellä laatu on jakaa se objektiivisesti arvioitavissa olevaan komponenttiin, subjektiivisesti arvioitavissa olevaan komponenttiin ja kokonaan arvioimattomissa olevaan komponenttiin [Wesselius ja Ververs, 1990]. Jos tuote toteuttaa kaikki vaatimusmäärittelyssä kuvatut vaatimukset, objektiivisen laadun näkökulmasta tuote on laadukas. Muita objektiivisesti arvioitavia asioita ovat mm. projektin pysyminen sille asetetussa aikataulussa ja budjetissa tai tuotteessa ilmenevien vikojen määrä.

Subjektiivisesti arvioitava laadun osakomponentti kuvaa, kuinka hyvin tuote täyttää asiakkaan odotukset. Tämä on tärkeää, koska käytännössä on mahdotonta arvioida objektiivisesti, miltä tuote käyttäjästä tuntuu ja kuinka hyvin se täyttää asiakkaan odotukset Tällainen arviointi voidaan tehdä esimerkiksi kyselytutkimuksena, jossa kukin haastateltu arvioi omaa näkemystään tuotteesta. Muita subjektiivisesti arvioitavia asioita ovat mm. asiakkaan käsitys tuotteen hyödyllisyydestä ja helppokäyttöisyydestä.

Kolmas laatukomponentti kattaa arvioimattomissa olevat seikat. Tällä tarkoitetaan sitä, miten hyvin tuotetta kyetään muuttamaan asiakkaiden tulevien, vielä tuntemattomien tarpeiden mukaiseksi, sekä käyttäytyykö tuote asiakkaan haluamalla tavalla aavistamattomissa virhetilanteissa. Lähtökohtana tässä osakomponentissa on se, että laadukkuus on arvioitavissa ajan myötä. Ennakolta siihen liittyviä tekijöitä on hankala arvioida, koska laatuun vaikuttavia asioita ei tuotteen valmistuessa ole vielä ilmennyt. Tuotteen kehittäjät voivat varautua tämän laatukomponentin vaatimuksiin hyvien suunnitteluratkaisujen kautta.

Kansainvälisen standardisointiliiton, International Organization for Standardization (ISO), kehittämät standardit sisältävät joitakin yleisiä määritelmiä tuotteen tai palvelun laadusta. ISO-standardien mukaisessa määrittelyssä laatu merkitsee tuotteen tai palvelun ominaisuuksia, joilla se täyttää sille asetetut tai oletettavat piirteet [Haikala ja Märijärvi, 1997]. Sen mukaan laatua voidaan arvioida kahdesta eri näkökulmasta. Subjektiivinen laatu on elämys tuotteen tai palvelun

erinomaisuudesta. Arvoperusteinen laatu on puolestaan sitä, että tuote tai palvelu on siitä maksetun hinnan arvoinen.

ISO-standardeissa annetaan erikseen tarkemmat kuvaukset määrittelyjen ja toteutustyön laadusta [SFS, 1997]. Määrittelyjen laatu on toisaalta määrittelydokumenttien osuvuus tarpeisiin siten, että niiden mukaan toteutettu ratkaisu täyttää asetetut tavoitteet ja toisaalta dokumenttien yhdenmukaisuus voimassaoleviin standardeihin ja käytäntöihin verrattuna. Toteutustyön laatu on lopputuloksen yhdenmukaisuutta määrittely- ja tuotekuvausten kanssa sekä lopputuloksen yhdenmukaisuutta voimassaoleviin standardeihin ja käytäntöihin verrattuna.

### 2.2. Ohjelmistotuotteen laatu

Hyvälaatuisen ohjelmistotuotteen ominaisuuksista on esitetty useita erilaisia näkemyksiä. Tunnetuimpia ovat ISO:n luomat standardit. ISO on kehittänyt standardeja, jotka määrittelevät ominaisuuksia ohjelmistotuotteen laadun arvioimiseksi. Nämä laatumääritykset kuuluvat ISO 9000–standardisarjaan [ISO, 2001]. ISO 9000 –standardien rakenteesta, sisällöstä ja sovellusalueista kerrotaan kohdassa 3.2.

ISO 9126 –standardin kehittämisen tarkoituksena oli löytää keskeisimmät laatuominaisuudet ohjelmistotuotteille [Pressman, 2000]. Standardi määrittelee kuusi laadukkaan ohjelmistotuotteen perusominaisuutta:

- toiminnallisuus (functionality)
- luotettavuus (reliability)
- käytettävyys (usability)
- tehokkuus (efficiency)
- ylläpidettävyys (maintainability)
- siirrettävyys (portability).

Ongelmana on käsitteiden yleisyys ja moniselitteisyys. Joitakin suuntaviivoja voidaan kuitenkin hahmotella. Toiminnallisuutta voidaan arvioida sen perusteella, kuinka sopiva tuote on tarkoitukseensa ja miten hyvät turvallisuusominaisuudet se tarjoaa. Luotettavuudella tarkoitetaan virheidensietokykyä ja virheistätoipumiskykyä. Käytettävyys taas liittyy tuotteen ymmärrettävyyteen ja opittavuuteen. Tehokkuutta voidaan puolestaan mitata suorituskyvyn perusteella. Ylläpidettävyys käsittää yleisesti tuotteen vakauden ja testattavuuden. Siirrettävyyttä voidaan parhaiten arvioida sen kannalta, kuinka helposti tuote on siirrettävissä käyttöympäristöstä toiseen.

# 3. Laatustandardeista

## 3.1. Laatuhankkeiden taustaa

Teollisuustuotteiden laatuun alettiin kiinnittää huomiota toisen maailmansodan jälkeen valtavan aseiden kysynnän ja sotilaalliseen toimintaan liittyvän kurinalaisuuden myötä [Salminen, 1994]. Tavaran toimittajilta alettiin vaatia ehdotonta luotettavuutta ja uusien teknologioiden toimivuutta kaikissa olosuhteissa. Toimittajilta edellytettiin myös tuotteiden tarkastustoimintaa. Kehitys johti kokonaan uuden työkokonaisuuden, laadunvarmistuksen, syntymiseen. Itsenäisiä laadunvarmistusosastoja alettiin muodostaa valmistusosastojen rinnalle. Laadun kehityksessä tapahtui merkittävä ajattelutavan muutos: laadun tuli perustua mitattuihin tosiasioihin.

1950-luvun lopussa kehittyi uusi, laadunohjausta korostava johtamistapa, Total Quality Control (TQC) [Salminen, 1994]. TQC-ajattelun myötä laadunkehityksen painopiste siirtyi Japaniin. Siellä yritysjohto oivalsi, ettei laadunohjausta voi toteuttaa pelkästään esimiesten ja johdon toimesta, vaan laadun varmistamiseksi tarvitaan koko henkilöstön aktiivista osallistumista. Tämän seurauksena USA joutui 1970-luvulla laatukriisiin, kun japanilaiset tuotteet valtasivat markkinoita parempitasoisen laadun ja kohtuullisempien hintojen turvin. Käynnistettiin massiivisia laadunkehitysohjelmia ja laatu alettiin ymmärtää koko yritystä koskevana laajana kulttuurimuutoksena. Laatua korostavaa johtamistapaa alettiin kutsua nimellä kokonaisvaltainen laatujohtaminen, Total Quality Management (TQM) [Salminen, 1994].

Euroopassa laadunkehitystä ohjaavat pääosin ISO 9000 –laatustandardit. International Standardization Organization (ISO) on kansallisten standardointielinten maailmanlaajuinen yhteenliittymä. Sen kehittämät standardit perustuvat 1970-luvun laatuajatteluun, jonka mukaan yrityksessä on luotava tietyt menettelytavat laadun varmistamiseksi [Virtanen, 1990]. Näitä noudattamalla voidaan saavuttaa tavoiteltu laatutaso, joka samalla täyttää asiakkaan tarpeet.

Ohjelmistoalan standardeja tuottavat ISO:n lisäksi myös IEEE (Institute of Electrical and Electronics Engineers), ANSI (American National Standards Institute), NBS (National Bureau of Standards), NASA (National Aeronautics and Space Administration), DoD (Department of Defence) ja ESA (European Space Agency). Suomessa standardointiin liittyviä organisaatioita ovat KOTEL (Komponenttiteollisuuden yhteistyöjärjestö) ja SFS (Suomen standardisoimisliitto).

## 3.2. ISO 9000 –laatustandardit

Laadun ja laadunvarmistuksen kehittäminen on edistynyt huomattavasti kansainvälisen ISO 9000 –standardisarjan julkaisemisen myötä [SFS, 1997]. Standardisarja julkaistiin ensimmäisen kerran vuonna 1987. Siihen kuuluvat

standardit tunnetaan kaikissa teollisuusmaissa ja toimialasta riippumatta useat yritykset ja julkiset organisaatiot ovat ottaneet ne kehitystyönsä lähtökohdaksi. Suomessa sarja ilmestyi SFS-ISO –standardina vuonna 1988. Uudistettu ISO 9000 –sarja vahvistettiin joulukuussa 2000 ja suomeksi se julkaistiin maaliskuussa 2001 [SFS, 2001].

ISO 9000 –sarja on tarkoitettu sekä organisaation oman toiminnan kehittämiseen että tilanteisiin, joissa asiakas ja toimittaja haluavat sopimustilanteessa sopia, kuinka laatu kyseisessä toimituksessa varmistetaan [ISO, 2001]. Sarjan keskeisimmät standardit ovat:

- ISO 9000:2000 Quality management systems – Fundamentals and vocabulary,
- ISO 9001:2000 Quality management systems – Requirements,
- ISO 9004:2000 Quality management systems – Guidelines for performance improvements.

Uudistuksen myötä laatustandardien rakenne muuttui ja terminologiaa yhtenäistettiin. Sisällöltään ne kuitenkin vastaavat pääosin vuonna 1994 julkaistuja standardeja [Ryynänen, 2001]. Tämän vuoksi tutkielmassa on käytetty lähteenä myös vanhempaa standardisarjaa.

Ohjelmistoalaa varten on laadittu omat lisämääritykset ISO 9000-3 standardissa [SFS, 2001]. Se toimii lähinnä suuntaa antavana ohjeena ohjelmistoalan laatujärjestelmien rakentamiseen. Siinä ei määritellä yksityiskohtaisesti ohjelmistoja tekevän organisaation tuotantoprosessia tai lopputuotteelta vaadittavia ominaisuuksia, vaan annetaan yleisiä ohjeita tavoiteltavan laatutason saavuttamiseksi organisaation soveltamien laatutoimenpiteiden avulla. Standardi jakautuu kolmeen osaan, joissa määritellään laatujärjestelmän puitteet, ohjelmistoprojektin elinkaaren aikaiset toimenpiteet ja laadunvarmistuksen tukitoiminnot.

# 4. Laadunvarmistuksen menetelmiä

## 4.1. Peruskäsitteitä

Laadunvarmistukseen liittyvät läheisesti seuraavat käsitteet:

- Laatu (quality) tarkoittaa tuotteen tai palvelun kaikkia piirteitä ja ominaisuuksia, joiden ansiosta nimetyt tai oletettavat tarpeet tulevat tyydytetyiksi [Salminen, 1994].
- Laadunvarmistuksella (quality assurance) tarkoitetaan kaikkia niitä suunniteltuja ja järjestelmällisiä toimenpiteitä, jotka ovat tarpeen riittävän

varmuuden saamiseksi siitä, että tuote täyttää sille asetetut laatuvaatimukset [Yli-Olli, 1991].

- Laatusuunnitelma (quality plan) on dokumentti, joka määrittelee tiettyyn tuotteeseen tai projektiin liittyvät laadunvarmistustoiminnot ja muut laatukäytännöt [Salminen, 1994].
- Laatujärjestelmällä (quality organisation) tarkoitetaan yrityksen laadunvarmistuksen toteuttamista varten varattuja prosesseja, resursseja, menettelytapaohjeita ja toimintasuhteita [Yli-Olli, 1991].

## 4.2. Laadunvarmistuksen edellytykset

Ohjelmistoprojektin onnistuminen edellyttää laadunvarmistuksen huomioonottamista projektin jokaisessa vaiheessa. Hyvän laadun saavuttamiseksi vaaditaan sekä huolellista suunnittelua että kurinalaista toteutustyötä. Laadunvarmistustoiminnot on suunniteltava riittävän ajoissa, koska toimenpiteet on aloitettava jo ohjelmiston määrittelyvaiheessa. Jotta laatusuunnitelma palvelisi tarkoitustaan, on sen oltava kirjallinen ja riittävän tarkka sisällöltään.

Laatusuunnitelman noudattamisesta on suurin vastuu projektipäälliköllä, joka on samalla vastuussa koko projektin läpiviennistä. Varsinkin laajoissa projekteissa laatua valvomaan voidaan kuitenkin erikseen nimetä joku projektiryhmän jäsenistä tai perustaa erillinen laadunvarmistusryhmä [Sytyke, 1993]. Laadunvarmistukseen tulee varata riittävästi resursseja, jotta laatua voidaan tehokkaasti valvoa. Myös koulutukseen on panostettava, koska laadunvarmistustoiminnot liittyvät tavalla tai toisella jokaisen projektiin osallistuvan henkilön työhön.

Johtohenkilöiden suhtautumisella laadunvarmistukseen on ensiarvoisen tärkeä vaikutus sen onnistumiseen ohjelmistoprojekteissa [Yli-Olli, 1991]. Näkyvä laatujohtaminen vakuuttaa sekä henkilöstön että asiakkaat siitä, että laatu on aidosti tärkeä tavoite. Johdon sitoutuminen laatutavoitteisiin on käytännössä edellytys niille resursseja koskeville päätöksille, joita laadunvarmistustoiminnot edellyttävät. Vain kokonaisvaltainen laatuun sitoutuminen voi johtaa tarvittaviin, usein suuriinkin toimintatapojen muutoksiin.

## 4.3. Tarkastukset ja katselmoinnit

### *4.3.1. Yleistä*

Tarkastusten ja katselmointien kehittäjä on Fagan, joka esitteli menetelmän jo 1970-luvulla [Haikala ja Märijärvi, 1997]. Menetelmä liittyy ohjelmistokehityksen tukitoimintoihin, joilla pyritään avustamaan ohjelmistoprojektin etenemistä siten, että eteneminen olisi hallittua ja samalla näkyvää. Sen avulla voidaan todeta tietty

välivaihe saavutetuksi, kun vaiheen aikaansaannos on hyväksytty tarkastuksessa. Menetelmän avulla pyritään poistamaan virheet tuotteesta mahdollisimman aikaisessa vaiheessa.

Katselmoinneilla (review) tarkoitetaan kaikkia niitä toimia, joilla virheitä ja puutteita etsitään kohteena olevasta dokumentista [Yli-Olli, 1991]. Ne voivat kohdistua mihin tahansa projektin kuluessa syntyvään dokumenttiin. Mikäli katselmointi on määrämuotoinen, esimerkiksi Faganin menetelmään perustuva, sitä sanotaan tarkastukseksi (inspection). Faganin menetelmän mukaisessa muodollisessa tarkastuksessa noudatetaan mm. seuraavia periaatteita:

- tarkastus kohdistuu aina dokumenttiin, ei sen tekijään,
- tavoitteena on etsiä dokumentista virheitä ja puutteita,
- tarkastusmenettely sisältää vaiheinaan tarkastukseen valmistautumisen, tarkastustilaisuuden ja dokumentoinnin sekä korjausten tekemisen ja niiden valvonnan,
- tarkastustilaisuuden kesto rajataan etukäteen,
- tarkastuksen tulokset dokumentoidaan,
- tulosten kehittymistä seurataan projektista toiseen. [Yli-Olli, 1991]

### *4.3.2. Tarkastuksen kohteet*

Tarkastuksille ja katselmoinneille on ominaista kohteen läpikäynti eri menetelmillä ja osallistujien henkilökohtaiset arviot kohteen tilasta. Tällaisia katselmointeja, jotka voidaan järjestää myös tarkastuksina, ovat esimerkiksi

- sopimuskatselmointi,
- projekti- ja laatusuunnitelman katselmointi,
- projektin tilanteen katselmointi,
- määrittelyjen katselmointi,
- teknisten suunnitelmien katselmointi,
- ohjelmakoodin katselmointi,
- testaussuunnitelman katselmointi.

Käytännössä kohteet vaihtelevat projektin laajuudesta ja kestosta riippuen. Tarkastusten ja katselmointien piiriin kannattaa ottaa ainakin kaikki määrittelydokumentit, kuten asiakkaan vaatimusmäärittely ja ohjelmiston toiminnallinen määrittely, suunnitteludokumentit, kuten tekninen suunnitelma ja testaussuunnitelma sekä testiraportit. Tarkastustilaisuudet vaativat kuitenkin sen

verran aikaa ja resursseja, että esimerkiksi ohjelmakoodin tarkastamista voidaan harkita tapauskohtaisesti.

### *4.3.3. Tarkastustilaisuus*

Tarkastustilaisuuden osallistujat valitsee yleensä joko projektipäällikkö tai tarkastuksen kohteena olevan dokumentin tekijä. Tarkastuksiin tulisi kutsua projektin kaikki ne osapuolet, jotka tulevat saamaan tarkastettavan materiaalin käyttöönsä sen valmistuttua. Hyvä periaate voisi olla Haikalan ja Märijärven [1997] esittämä näkemys, jonka mukaan kaikilla tilaisuuteen osallistuvilla tulee olla siitä jotain saatavaa tai sille jotain annettavaa. Muutoin tarkastustilaisuudesta voi tulla tehoton, jolloin se menettää merkityksensä. Tällaiset tarkastukset eivät ole mielekkäitä, koska ne vievät turhaan ihmisten työaikaa ja estävät samalla jonkin muun työn etenemistä.

Tarkastukseen osallistuvien tulisi valmistautua tilaisuuteen tutustumalla etukäteen tarkastettavaan materiaaliin ja yrittämällä löytää siitä mahdollisimman paljon puutteita, epäselvyyksiä, ristiriitoja ja muita ongelmakohtia. Tällöin osallistujilla on valmiina kommentteja tarkastuksen kohteesta ja koko tilaisuudesta saadaan irti suurin mahdollinen hyöty. Itse tarkastustilaisuudessa dokumentti käydään läpi kohta kohdalta osallistujien esittäessä kommenttejaan. Tilaisuuden päätteeksi tarkastuksen kohde joko hyväksytään tai päätetään uudesta tarkastuksesta.

Tarkastukseen osallistuvasta ryhmästä voidaan tunnistaa erilaisia rooleja, jotka edesauttavat tilaisuuden etenemistä [Haikala ja Märijärvi, 1997]. Tärkeimmät ovat puheenjohtaja, sihteeri ja esittelijä. Puheenjohtajan tehtävänä on huolehtia siitä, että pysytään asiassa ja kaikki tarpeellinen kirjataan ylös. Sihteeri pitää tarkastustilaisuudesta pöytäkirjaa, johon havaitut puutteet ja ongelmat kirjataan. Tällainen menettely takaa sen, että virheet eivät jää korjaamatta. Tarkastuksen kohteena olevan dokumentin esittelijänä toimii yleensä sen tekijä. Hänen tehtävänään on esitellä materiaalia ja vastailla sitä koskeviin kysymyksiin. Tekijän vastuulla on myös pöytäkirjaan merkittyjen korjausten tekeminen.

### *4.3.4. Vaikutukset laatuun*

Ohjelmistoprojekteissa tarkastusten ja katselmointien tavoitteena on varmistaa, että projektin vaiheet ja välitulokset täyttävät niille asetetut vaatimukset. Vaatimuksina voivat olla asiakkaan tarpeet, välitulosta edeltävät tai sitä seuraavat vaiheet, sovitut menettelytavat tai noudatettavat standardit. Laadunvarmistuksen kannalta ensisijaisena tavoitteena on löytää virheet ja muut laatupoikkeamat mahdollisimman aikaisessa vaiheessa. Tarkoituksena on myös luoda yhdenmukainen laatutaso tarkastettaville dokumenteille sekä kehittää yhtenäiset menetelmät ja työskentelytavat projektin eri vaiheissa. Näitä ratkaisuja voidaan hyödyntää tulevissa projekteissa.

Ohjelmistoprojekteista tehtyjen tutkimusten mukaan tarkastukset ovat testausta tehokkaampi virheiden etsintäkeino [Haikala ja Märijärvi, 1997]. Tämä perustuu siihen, että virheiden korjauskustannukset kasvavat nopeasti, jos virheen löytyminen siirtyy projektin myöhempään vaiheeseen. Määrittelyvirheen löytäminen vasta suunnitteluvaiheessa voi johtaa jopa nelinkertaiseen kustannusten kasvuun verrattuna tapaukseen, jossa virhe löydetään jo määrittelyvaiheessa. Eräässä suuria ohjelmistoja koskeneessa tutkimuksessa todettiin, että tarkastukset olivat jopa 20 kertaa tehokkaampia kuin testaaminen [McConnell, 1998].

# 5. Laadunvarmistustoiminnot

## 5.1. Ohjelmistoprojektin vaihejakomalli

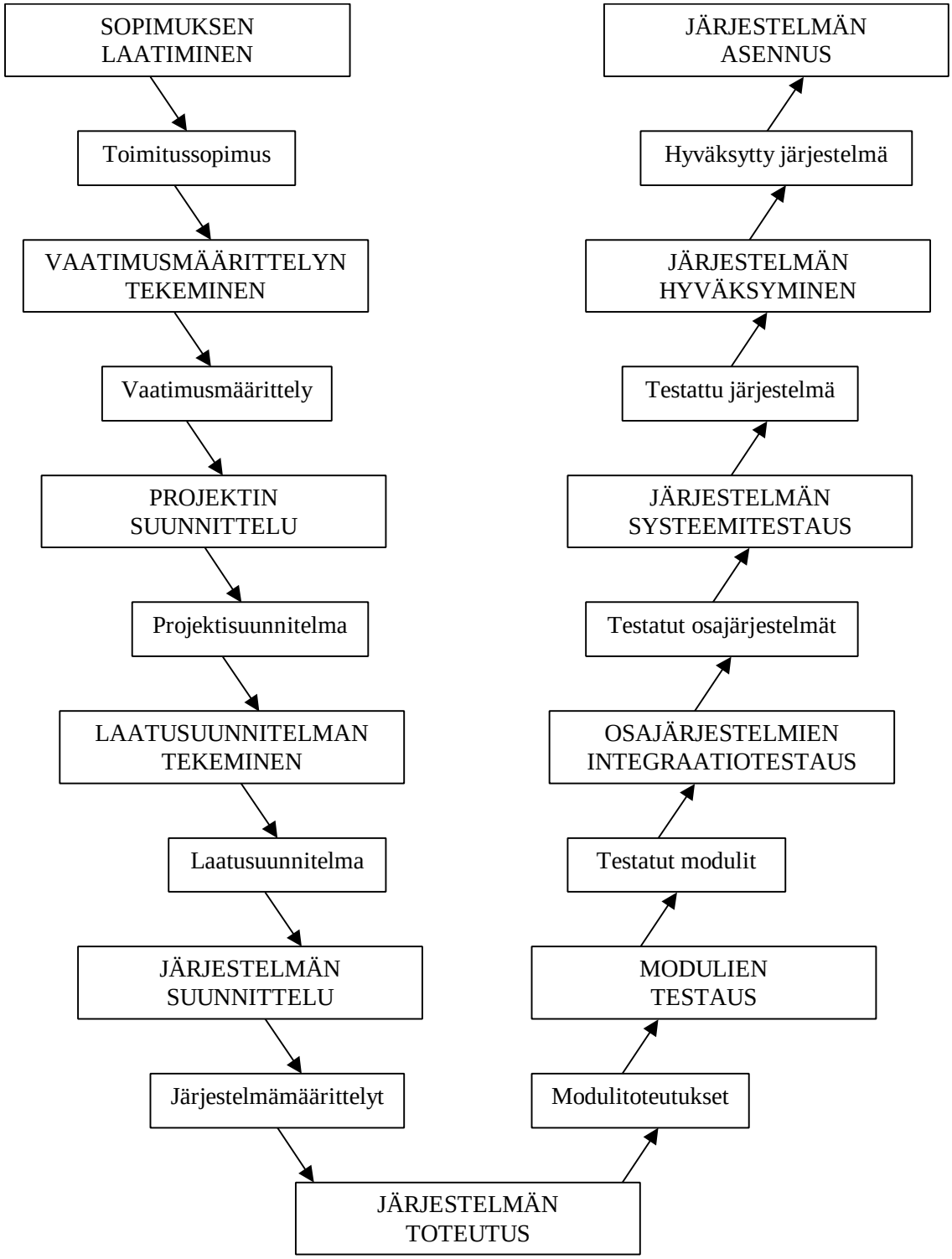
Ohjelmistoprojektin suunnittelu, seuranta ja ohjaus perustuvat tavallisesti projektin vaiheistamiselle. Tämä merkitsee käytännössä sitä, että projekti organisoidaan jonkin vaihejakomallin mukaisesti. Laatuun liittyvät toiminnot tulisi suunnitella ja toteuttaa käytetyn mallin edellyttämällä tavalla. Vaihejako on tärkeää tehdä harkitusti, koska siihen tehdyt muutokset muuttavat myös laadunvarmistustoimintoja.

ISO-standardien mukaan vaihejako tulisi määritellä ja dokumentoida yhtenä ensimmäisistä laadunvarmistustoiminnoista, koska useimmat muut toimenpiteet nojaavat siihen [Yli-Olli, 1991]. Vaihejakomallin tulisi sisältää seuraavat määrittelyt:

- toteutettavat vaiheet: nimeäminen ja työsisällön kuvaaminen,
- kunkin vaiheen alkutilanne: edellytykset vaiheen käynnistämiseksi,
- kunkin vaiheen lopputilanne: mitä vaiheen lopussa on oltava valmiina,
- kussakin vaiheessa suoritettavat todentamismenettelyt, joilla varmistetaan aikaansaannosten virheettömyys ja asianmukaisuus.

Ohjelmistoprojektin vaihejako voidaan esittää V-kaaviona, joka erittelee projektin keskeisimmät vaiheet ja niiden lopputulokset [Yli-Olli, 1991]. Eräs perusvaihejako on esitetty V-kaaviona kuvassa 1. Kaavion vasen laita edustaa päätösprosessia siitä, mitä halutaan saada aikaan ja oikea laita toteutusprosessia, jolla tulos synnytetään. Käytännössä eri tilanteisiin sopivat V-kaaviot poikkeavat toisistaan vaiheiden valinnassa, järjestyksessä ja sisällössä.

Vaihemäärittelyjen tulisi olla niin selkeitä, että kunkin vaiheen lopputulos on konkreettinen ja näin ollen sen laatu on arvioitavissa. Lopputuloksen arviointi antaa välitöntä palautetta vaiheen onnistumisesta, koska sen täytyy vastata seuraavan vaiheen lähtökohtaa. Laadunvarmistuksen tavoitteena on taata, että vaiheittaisen kehityksen lopputulos on kussakin projektin vaiheessa aiemmin tehtyjen määrittelyjen, sovittujen sääntöjen ja menettelytapojen mukainen.




Kuva 1. Ohjelmistoprojektin vaihejakomalli V-kaaviona. Mukailtu Yli-Ollin teoksesta [1991].

## 5.2. Laadunvarmistus projektin eri vaiheissa

### *5.2.1. Sopimusvaihe*

Ohjelmistoprojektin perustuessa asiakkaan ja toimittajan väliseen sopimukseen tulee toimittajan huolehtia ennen varsinaisen projektin käynnistämistä siitä, että sopimus on asianmukaisesti laadittu. Toimenpiteinä ovat tarkastukset ja katselmoinnit, joista kerrottiin kohdassa 4.3. Sopimuksesta tulisi tarkistaa, että

- vaatimukset ja molemminpuoliset velvollisuudet on dokumentoitu ja määritelty riittävän tarkasti,
- projektiin mahdollisesti liittyvät riskit on yksilöity,
- vaatimukset ja vastuut ovat molempien osapuolten hyväksymät,
- sopimuksessa käytetyt termit ovat yksiselitteisiä,
- molemmilla osapuolilla on kyky saavuttaa heille asetetut vaatimukset.

ISO-standardien mukaan sopimuksissa tulisi aina mainita tuotteen hyväksymiskriteerit [SFS, 1997]. Asiakkaan kehitysvaiheessa esittämien muutosvaatimusten käsittely tulee määritellä, samoin kuin hyväksymisen jälkeen havaittujen ongelmien käsittely. Tällaisia ongelmia ovat asiakkaan tyytymättömyys tuotteen laatuun ja siitä aiheutuvat valitukset. Muutosvaatimusten ja valitusten käsittely on otettava huomioon jo sopimusvaiheessa mahdollisten väärinkäsitysten välttämiseksi.

### *5.2.2. Vaatimusmäärittelyt*

Ohjelmistoprojektissa määrittelyillä (specifications) on kaksi roolia [Yli-Olli, 1991]. Laadunvarmistuksen tavoitteena on saavuttaa yhtäpitävyys toimitettavan tuotteen ja määrittelyjen välillä. Viimeistään tuotteen hyväksymisvaiheessa määrittelyä ja lopputuotetta verrataan toisiinsa. Toisaalta määrittelyt toimivat ensisijaisena lähtökohtana toimitettavan tuotteen suunnittelulle ja toteutukselle. Molemmat näkökulmat tulisi ottaa huomioon määrittelyjä tehtäessä.

Jotta ohjelmiston suunnittelu voidaan aloittaa, sen perustana on oltava selkeät ja riittävän tarkat vaatimukset. Nämä ominaisuudet esitetään vaatimusmäärittelyssä (requirements specification) [McConnell, 1998]. Toiminnallisten vaatimuksen lisäksi määrittelyihin tulee sisältyä kaikki muutkin tekijät, jotka ovat tarpeen asiakkaan esittämien tarpeiden tyydyttämiseksi. Tällaisia vaatimuksia voivat olla esimerkiksi suorituskyky, turvallisuus, luotettavuus ja käytettävyys. Vaatimukset on jo määrittely-vaiheessa ilmaistava täsmällisesti, jotta tuotteen hyväksymisprosessin aikana voidaan tarkistaa jokaisen ominaisuuden tulleen toteutetuksi oikein.

Toisinaan asiakas haluaa toimittaa vaatimusmäärittelyt sisältävän dokumentin. Usein kuitenkin ohjelmiston toimittajan tehtävänä on kehittää määrittelydokumentti yhteistyössä asiakkaan kanssa. Tällöin dokumentille on aina saatava myös asiakkaan hyväksyntä. Erityistä huomiota kannattaa kiinnittää siihen, että molemmat osapuolet ovat nimenneet vaatimusmäärittelystä vastuussa olevat henkilöt ja on sovittu menetelmistä määrittelyihin projektin aikana tehtävien muutosten hyväksymiseksi. Määrittelyissä käytettyjen termien on oltava riittävän yksiselitteisiä, jotta vältytään väärinkäsityksiltä niiden tulkinnassa.

Vaatimusmäärittelyjen pohjalta laaditaan yleensä vielä erikseen ohjelmiston toiminnallinen määrittely (functional specification), jossa sen toiminnalliset ja ei-toiminnalliset ominaisuudet kuvataan tarkasti ja mahdollisimman yksityiskohtaisesti [McConnell, 1998]. Toiminnallisen määrittelyn tulee aina vastata vaatimusmäärittelyssä esitettyjä vaatimuksia ja sen tulee olla asiakkaan hyväksymä. Toiminnallinen määrittely toimii perustana testaussuunnitelman tekemiselle.

### *5.2.3. Projektin suunnittelu*

Projektin suunnittelu on yksi koko ohjelmistoprojektin keskeisimmistä osa-alueista. Siitä on vastuussa yleensä projektipäällikkö. Projektisuunnnitelman varaan rakentuvat sekä projektin toteutus että sen seuranta. ISO-standardien mukaan projektisuunnitelman tulee kattaa seuraavat asiat:

- projektin yleiskuvaus sisältäen tavoitteet ja viittaukset asiayhteydessä oleviin määrittelydokumentteihin,
- projektin organisaatio sisältäen henkilöresurssit, henkilöiden vastuut ja muiden resurssien käytön,
- projektin vaiheet,
- aikataulu, josta käy ilmi kaikki tehtävät ja kuhunkin tehtävään tarvittavat resurssit sekä tehtävien väliset riippuvuudet,
- projektiin liittyvien suunnitelmien yksilöinti, kuten laatusuunnitelma, ohjelmiston toiminnallinen ja tekninen suunnitelma, versionhallinnan suunnitelma ja testaussuunnitelma. [SFS, 1997]

Projektin suunnittelu perustuu siihen, että kaikki projektiin liittyvä työ jaetaan vaiheisiin. Tämän vuoksi projektisuunnitelmassa on riittävän tarkasti määriteltävä kunkin vaiheen alku- ja lopputilanne sekä vaiheen lopuksi suoritettavat hyväksymismenetelmät ennen seuraavaan vaiheeseen siirtymistä. Lisäksi tulisi jo etukäteen tunnistaa vaiheiden läpivientiin liittyvät riskit ja laatia suunnitelma niiden ehkäisemiseksi. Projektisuunnitelmaa voidaan täsmentää tai muuttaa projektin

edetessä. Muutoksiin liittyvät menettelytavat ja valtuudet on kuitenkin määriteltävä jo projektin suunnitteluvaiheessa.

### *5.2.4. Laatusuunnittelu*

Laadunvarmistuksen toteuttamiseksi on syytä laatia joko erillinen tai projektisuunnitelmaan sisältyvä laatusuunnitelma, jossa määritellään projektin aikana toteutettavat laadunvarmistustoiminnot [McConnell, 1998]. Toiminnoilla varmistetaan, että ohjelmiston osa tai ohjelmistotuote on määriteltyjen vaatimusten mukainen ja vastaa asiakkaan tarpeita. Laatusuunnitelman tarkoitus on koota tietyssä projektissa sovellettavat laadunvarmistuksen menettelyt yhtenäiseksi suunnitelmaksi.

Laatusuunnitelman voi käytännössä rakentaa yleiseksi tai projektikohtaiseksi. Eräs tapa on muodostaa yleinen malli, jota sovelletaan kussakin projektissa erikseen [Sytyke, 1993]. Mikäli projektille on nimetty laatuvastaava tai laadunvarmistusryhmä, voivat nämä osallistua laatusuunnitelman tekoon. Muussa tapauksessa sen tekeminen on pääosin projektipäällikön vastuulla.

Määrittelyille, suunnitelmille ja muulle materiaalille tulee laatia vaatimukset niiden hyväksymistoimenpiteistä. Tämä tarkoittaa käytännössä tarkastusten, katselmointien ja testausten läpivientiä. Laadunvarmistustoiminnoille tulee laatia aikataulu ja määritellä, kenellä on valtuudet kunkin välivaiheen hyväksymiseen. Kaikki laatuun liittyvät vastuut, kuten testaus, virheiden korjaus, muutostenhallinta ja versionhallinta, on yksilöitävä.

Laatusuunnitteluun liittyy myös lopputuotteelle asetettavien laatutavoitteiden määrittely. Nämä tavoitteet tulisi ilmaista mitattavassa muodossa aina sen ollessa mahdollista, koska tällöin voidaan parhaiten arvioida niiden toteutumista. Projektin välivaiheiden tuloksena syntyvien aikaansaannosten hyväksymiseen on määriteltävä laatukriteerit, joiden on toteuduttava ennen seuraavan vaiheen aloittamista. Laatusuunnitelman hyväksymiseen tulisi soveltaa samoja periaatteita kuin muidenkin projektissa tuotettavien dokumenttien.

### *5.2.5. Suunnittelu- ja toteutusvaiheet*

Ohjelmiston suunnitteluvaiheessa määritellään ohjelman arkkitehtuuri ja rakenteelliset ratkaisut. Tuloksena syntyy ohjelmiston tekninen määrittely (technical specification) [McConnell, 1998]. Kuten muutkin suunnitteludokumentit, on tekninen määrittely hyväksyttävä tarkastusten tai katselmointien kautta, jotta varmistetaan, että suunnittelun tulokset vastaavat vaatimuksia. On myös kiinnitettävä huomiota siihen, että suunnitelman pohjalta voidaan aloittaa ohjelmiston toteutusvaihe.

Ohjelmiston toteutuksella (implementation) tarkoitetaan kaikkia niitä toimintoja, joilla määrittelyt ja suunnitelmat muunnetaan suorituskelpoiseksi ohjelmaksi [Salminen, 1994]. Käytännössä tämä vaihe sisältää ohjelmakoodin kirjoittamisen.

Toteutusvaiheessa on noudatettava aiemmin määriteltyjä sääntöjä, kuten sovittuja ohjelmointitapoja, nimeämiskäytäntöä, ohjelmakoodin yhdenmukaisuutta ja asianmukaista kommentointia. Tähän vaiheeseen liittyy yleensä myös jonkin verran ohjelman testausta.

### *5.2.6. Testaus*

Ohjelmistojen testauksen yhteydessä testaus (testing) määritellään suunnitelmalliseksi virheiden etsimiseksi ohjelmaa tai sen osaa suorittamalla [Haikala ja Märijärvi, 1997]. Testauksen yhteydessä virhe (error) on poikkeama ohjelmiston määrittelyistä. Tästä virheen määritelmästä seuraa se, että testausta ei voida suorittaa ilman määrittelyjä, joihin testauksen tuloksia verrataan. Lopputulosten oikeellisuutta ei voida todeta ilman täsmällisiä määrittelyjä. Tavallisesti testaus perustuu tuotteen toiminnalliseen ja tekniseen määrittelyyn.

Testausta tarvitaan useilla eri tasoilla yksittäisestä ohjelmamodulista koko järjestelmän testaukseen asti. Testaussuunnitelma tulee aina laatia kirjallisesti ja sen pitää sisältää tarkat kuvaukset testitapauksista ja odotetuista tuloksista. Näin huolehditaan siitä, että testin suoritusvaiheessa sen onnistuminen tai epäonnistuminen voidaan todeta vertaamalla saatua tulosta odotettuun tulokseen. Testiympäristö, testausvälineet ja tarvittavat ohjelmat tulee määritellä etukäteen, samoin kuin testauksen suorittavat henkilöt.

Testaus alkaa usein jo toteutusvaiheessa, kun ohjelmakoodia kirjoitetaan. Modulitestauksessa (module testing) ohjelmamoduleja testataan yksitellen, teknisiä määrittelyjä vasten [NASA, 1989]. Testattavana on yksittäinen ohjelmamoduli, joka voi olla tapauksesta riippuen luokka, komponentti tai laajempi ohjelman osa. Integrointitestauksessa (integration testing) erikseen toteutetut ja testatut ohjelmamodulit liitetään systemaattisesti yhteen yhä laajemmaksi kokonaisuudeksi ja testataan vaiheittain [NASA, 1989]. Modulitestauksen suorittaa tavallisesti modulin toteuttaja, integrointitestauksessa voidaan käyttää lisäksi erillistä testiryhmää.

Järjestelmätestauksessa (system testing) tarkastelun kohteena on koko järjestelmä ja tuloksia verrataan määrittelydokumentaatioon, yleensä toiminnalliseen määrittelyyn [NASA, 1989]. Toiminnan oikeellisuus tulee tarkastaa asiakkaan kanssa tehdyissä sopimuksissa määriteltyä käyttöympäristöä vastaavissa olosuhteissa. Testausvaiheet pitävät sisällään sekä toiminnallisten että muiden tuotteelle asetettujen vaatimusten testauksen, kuten suorituskyvyn, luotettavuuden ja käytettävyystestauksen. Testitapaukset on suunniteltava siten, että niiden avulla voidaan tarkistaa jokaisen vaatimuksen toteutuminen. Järjestelmätestauksen suorittajana tulisi olla toteutustyöstä mahdollisimman riippumaton testiryhmä.

### *5.2.7. Hyväksyminen*

Ennen tuotteen tarjoamista asiakkaan hyväksyttäväksi on varmistettava, että kaikki suunnitellut tarkastukset ja testaukset on suoritettu ja niiden tulokset täyttävät määritellyt vaatimukset. ISO-standardien mukaan mitään tuotetta ei tule toimittaa ennen kuin kaikki laatusuunnitelmassa tai muussa dokumentoidussa menettelyohjeessa määritellyt toiminnot on tyydyttävällä tavalla suoritettu [SFS, 1997]. Tarkastuksiin ja testauksiin liittyvien dokumenttien on oltava saatavilla ja niiden on oltava asianmukaisilla tarkastusmenettelyillä hyväksyttyjä.

Kun toimittaja on valmis toimittamaan tuotteen asiakkaalle, tulee asiakkaan arvioida, onko se hyväksyttävissä aiemmin sovittujen kriteerien ja sopimusten määrittelemällä tavalla. Hyväksymisen yhteydessä todetaan, että kaikki projekti-, laatu- ja testaussuunnitelmissa määritellyt tarkastukset ja testaukset on suoritettu hyväksyttävin tuloksin. Tarkastuksiin ja testauksiin liittyvät dokumentit tulee toimittaa asiakkaalle, jotta voidaan todistaa tuotteen läpäisseen ne hyväksymiskriteerien mukaisesti. Hyväksymismenettelyn aikana havaittujen ongelmien, kuten asiakkaan tyytymättömyyden, käsittelytavat on sovittava etukäteen.

Tuotteen toimitus ja asennus on tehtävä sopimuksessa määriteltyjä toimenpiteitä noudattaen. Sopimusvaiheessa voidaan määrätä, että toimitetun ohjelmiston ylläpidosta vastaa toimituksen ja asennuksen jälkeen asiakas. Muussa tapauksessa toimittajan tulee ottaa käyttöön ylläpitomenetelmät, joilla taataan, että ylläpito vastaa sopimuksessa määriteltyjä vaatimuksia. Ylläpitotoimet luokitellaan tyypillisesti virheiden korjaamiseen, toimintojen lisäämiseen ja suorituskyvyn parantamiseen [Salminen, 1994]. Myös neuvontapalvelu voi sisältyä ylläpidon piiriin. Kaikista neuvonta- ja huoltotoimista tulisi kuitenkin sopia asiakkaan kanssa etukäteen.

## 5.3. Laatua valvovat tukitoiminnot

### *5.3.1. Versionhallinta*

Versionhallinta (version control) tarkoittaa niitä menettelytapoja, joilla voidaan yksilöidä ja jäljittää kunkin ohjelmistotuotteen komponentin virallinen versio [Salminen, 1994]. Vanhoja edelleen käytössä olevia versioita joudutaan lisäksi usein ylläpitämään ja valvomaan. Tämän vuoksi versionhallinnan tueksi tulee kehittää ja toteuttaa suunnitelma, jossa määritellään versionhallinnan toiminnot, työvälineet ja työtavat. Eri osapuolten vastuut ja valtuudet on myös selvitettävä. Lisäksi on kontrolloitava järjestelmän tietyn osan samanaikaista muuttamista useamman henkilön toimesta. Apuna voidaan käyttää esimerkiksi automatisoitua tiedostonhallintaohjelmaa.

ISO-standardit määrittelevät versionhallinnalle kolme vaatimusta: tunnistus ja jäljitettävyys, muutosten hallinta ja versioiden tilan raportointi [SFS, 1997]. Jokaisella

ohjelman rakenneosalla tulee olla yksikäsitteinen tunniste, joka takaa sen tunnistamisen ja jäljitettävyyden. Rakenneosat kattavat ohjelmistokehityksen kaikki vaiheet määrittelyistä tuotteen toimitukseen asti. Versionhallinnan alaisten rakenneosien muutokset pitää yksilöidä, dokumentoida, tarkastaa ja hyväksyä. Ennen muutoksen virallistamista tulee sen oikeellisuus vahvistaa ja vaikutukset muihin rakenneosiin tutkia. Versioiden tila tulee ilmoittaa siten, että jokaisen rakenneosan nykyinen virallinen versio voidaan tunnistaa.

#### *5.3.2. Dokumenttien valvonta*

Dokumentointi on eräs ohjelmistotyön perustehtävistä. Tiedot päätöksistä, suunnitelmista ja määrittelyistä on kirjattava ylös ja tallennettava. Tietojen hallitsemiseksi on ylläpidettävä menetelmiä, joilla projektin kuluessa syntyviä dokumentteja valvotaan. On määriteltävä, mitä dokumentteja kussakin projektin vaiheessa tuotetaan ja hahmoteltava niiden sisältörakenne. Tämän lisäksi on luotava tarpeelliset valvontamenettelyt, joilla varmistetaan dokumenttien asianmukainen hyväksyminen ja muuttaminen. Tehtävään valtuutettujen henkilöiden tulee tarkastaa ja hyväksyä kaikki dokumentit ennen niiden julkaisemista. On varmistettava, että kaikissa työvaiheissa on saatavilla tarvittavien dokumenttien ajan tasalla olevat versiot.

Laatuun liittyviä dokumentteja kutsutaan ISO-standardeissa laatutiedostoiksi (quality record) [SFS, 1997]. Standardeissa korostetaan laatutiedostojen merkitystä ja tarpeellisuutta projektin jokaisessa vaiheessa. Aina kun täytyy osoittaa, että jotain on tehty tai tietty vaatimus on täytetty, tieto tästä on tallennettava. Laatutiedostoja ovat esimerkiksi projektin eri vaiheissa syntyneet tarkastus-, katselmointi- ja testauspöytäkirjat. Tiedostojen käytön valvomiseksi on yksilöitävä, kuinka kauan kutakin laatutiedostotyyppiä on tarpeen säilyttää ja missä sitä säilytetään. Tärkeää on myös päättää, kenen saatavilla tiedostojen tulee olla ja kenellä on oikeus tehdä niihin muutoksia.

# 6. Laadun arviointi

## 6.1. Laadun mittaaminen

Laadun mittaamisella (quality measurement) tarkoitetaan ohjelmistoprojektissa oman toiminnan ja sen lopputulosten mittaamista [Yli-Olli, 1991]. Mittaaminen koskee kaikkia laatujärjestelmän rakenneosia sisältäen lopputuotteen laadulliseen arviointiin ja projektin seurantaan sekä tuotteen hyväksymiseen ja ylläpitoon liittyvien tekijöiden tarkastelua. Sen tarkoituksena on palvella sekä toimittajan että asiakkaan tarpeita projektin kuluessa ja sen jälkeen.

Tärkein mittauskohde on lopputuote ja sen täyttämät laatuvaatimukset. Kohdassa 2.2 määriteltyjen laatuominaisuuksien mittaamiseen voidaan käyttää esimerkiksi seuraavia mittareita:

- luotettavuus: vikojen määrä aikayksikössä, toimintojen epäonnistumisaste,
- käytettävyys: käyttöoperaatioihin kuluva työmäärä, käytön opetteluun kuluva aika,
- tehokkuus: vasteaika, prosessorin kuormitus,
- ylläpidettävyys: vikailmoitusten määrä, vikojen korjauskustannukset.

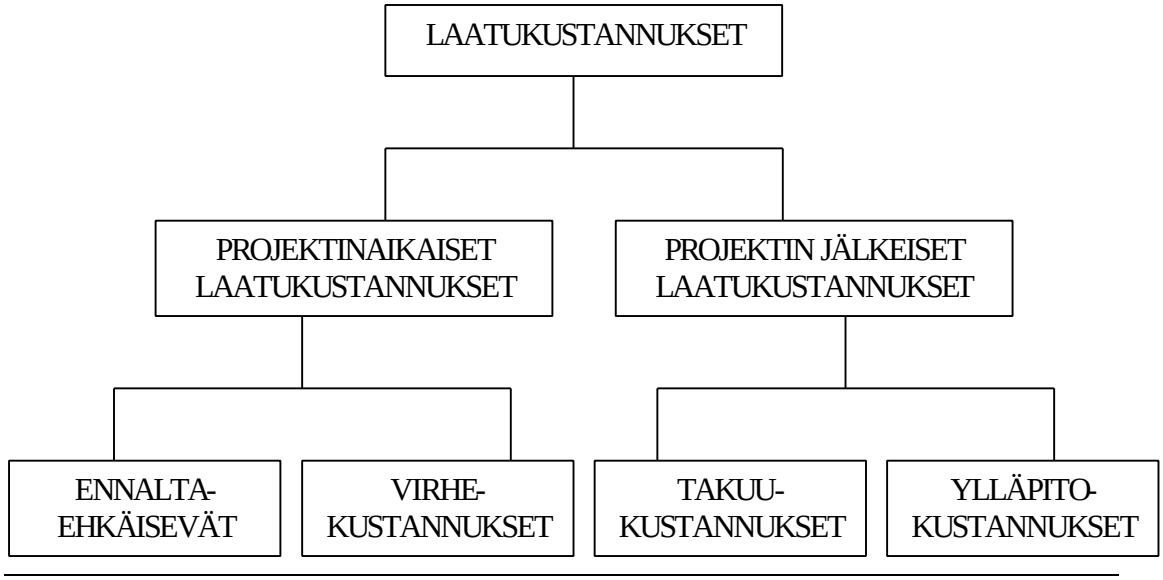
Monia laatuun liittyviä tekijöitä ei voida kuitenkaan mitata määrällisesti. Tuotteen toiminnallisuutta voidaan arvioida parhaiten sen perusteella, kuinka sopiva tuote on tarkoitukseensa ja kuinka hyvin se täyttää asiakkaan odotukset. Tämän vuoksi onkin tärkeää saada palautetta siitä, miltä tuote käyttäjästä tuntuu. Tällainen arviointi voidaan tehdä esimerkiksi kyselytutkimuksena, jossa kukin haastateltu arvioi omaa subjektiivista näkemystään tuotteesta.

Projektin seurannassa mittauskohteina ovat projektin eteneminen ja sen kuluessa havaitut poikkeamat. Etenemisen seurannassa otetaan huomioon projektin aikataulu, sen laajuus sekä siihen käytetty työmäärä ja kustannukset. Poikkeamien osalta kirjataan tarkastuksissa tai testauksissa löydettyjen virheiden määrä ja laatu sekä niiden vakavuus. Seurantaan sisältyy myös muutosten lukumäärän ja muutostyöhön kuluneen työmäärän mittaaminen.

Lopputuotteen hyväksymisen jälkeen mittauskohteena on toimituksen jälkeisten virheiden määrä ja laatu. Ylläpidon osalta on otettava huomioon järjestelmän suorituskyvyssä ilmenevät häiriöt ja niiden korjaukseen kuluva aika. Tämän lisäksi ylläpitoon kuuluvia tekijöitä ovat mahdolliset muutokset ja laajennukset, joita joudutaan tekemään sen jälkeen kun tuote on toimitettu asiakkaalle.

### 6.2. Laatukustannukset

Ohjelmistoprojektin synnyttämät laatuun liittyvät kustannukset voidaan jakaa kahteen eri ryhmään: projektinaikaisiin laatukustannuksiin ja projektin jälkeisiin laatukustannuksiin. Jaottelun rakennekaavio on esitetty kuvassa 2.




Kuva 2. Laatukustannukset ohjelmistoprojektissa. [Salminen 1994]

Salminen [1994] jakaa projektinaikaiset laatukustannukset ennaltaehkäiseviin laatukustannuksiin ja virhekustannuksiin. Ennaltaehkäisevät kustannukset koostuvat laatusuunnittelun, raportoinnin, tarkastusten ja katselmointien sekä laatuun liittyvän koulutuksen aiheuttamista kustannuksista. Virhekustannuksiin sisältyvät kaikki laatupoikkeamista johtuvat uusinta- ja korjaustoimenpiteet. Projektin jälkeisiä laatukustannuksia ovat toisaalta takuuaikana ilmenevien ongelmien aiheuttamat takuukustannukset ja toisaalta takuuajan jälkeisistä korjauksista johtuvat ylläpitokustannukset.

Kuvassa 3 tarkastellaan Juranin [1993] tekemää jaottelua laatukustannuksista. Sisäiset virhekustannukset muodostuvat kustannuksista, jotka aiheutuvat tuotteen vioista ja niiden korjaamisesta ennen sen toimittamista asiakkaalle. Ulkoisia virhekustannuksia ovat kustannukset, jotka aiheutuvat tuotteen vioista ja niiden korjaamisesta sen jälkeen kun tuote on toimitettu asiakkaalle. Lähtökohtana on se, että kaikki virheet aiheuttavat ylimääräisiä kustannuksia, mutta on edullisempaa löytää ne ennen tuotteen toimittamista asiakkaalle.

| Laatukustannusten laji | Kustannustyypin sisältö |
|---|---|
| Sisäiset virhekustannukset: | Virhekustannukset, jotka muodostuvat tuotteen vioista ja niiden korjauksesta ennen sen toimittamista asiakkaalle. |
| Ulkoiset virhekustannukset: | Virhekustannukset, jotka muodostuvat tuotteen vioista ja niiden korjauksesta sen jälkeen kun se on toimitettu asiakkaalle. |
| Arviointikustannukset: | Tarkastuksista, katselmuksista ja testauksesta yms. muodostuvat tuotteen tilan tarkistamisen kustannukset. |
| Estämiskustannukset: | Kustannukset, jotka muodostuvat virheiden syntymisen estämisestä, kuten laatusuunnittelu, valvonta ja raportointi. |

Kuva 3. Laatukustannusten jaottelu ohjelmistoprojektissa. Mukailtu Juranin teoksesta [1993].

Arviointikustannuksia ovat tarkastuksista, katselmoinneista ja testauksesta muodostuvat kustannukset. Tämäntyyppiset kustannukset ovat tavallaan vain lisäkustannuksia, koska toiminta ei varsinaisesti tuota tuotteelle lisäarvoa. Estämiskustannukset puolestaan ovat kustannuksia, jotka muodostuvat virheiden syntymisen estämisestä. Tällaisia ovat mm. laatusuunnittelun, valvonnan ja raportoinnin aiheuttamat kustannukset. Juranin [1993] estämiskustannukset voidaan rinnastaa Salmisen [1994] määrittelemiin ennaltaehkäiseviin kustannuksiin.

## 7. Yhteenveto

Laadunvarmistus on yksi ohjelmistoprojektin keskeisimmistä tekijöistä ja sen vuoksi laatuasioihin tulee kiinnittää huomiota projektin jokaisessa vaiheessa. Olen pyrkinyt esittelemään tärkeimpiä laadunvarmistustoimintoja ja tarjoamaan suuntaviivoja niiden käyttöönoton onnistumiseksi. Ohjeet koskevat niin määrittely- ja suunnitteludokumenttien kuin valmiin ohjelmankin laatua. Laadunvarmistus on kuitenkin tutkimusalueena niin laaja, että eri osa-alueita on tämän tutkielman puitteissa tarkasteltu vain yleisellä tasolla. Syvällisemmin aiheeseen voi perehtyä tutkimalla laadunvarmistukseen liittyvää kirjallisuutta ja siitä tehtyjä tutkimuksia.

Lopuksi haluan korostaa laadunvarmistuksen merkitystä kaikkia osapuolia tyydyttävän lopputuloksen aikaansaamisessa. Suurin vastuu on esimiehillä, mutta

tämän lisäksi tarvitaan jokaisen projektiryhmän jäsenen omaa panosta. Laatuun tähtäävät menetelmät edellyttävät suunnitelmallista kehitystä ja kurinalaista toteutusta. Laatu tulee asettaa tavoitteeksi ja tämän tavoitteen saavuttamiseksi vaaditaan määrätietoista ja järjestelmällistä toimintaa. Palkintona on paitsi onnistunut projekti ja laadukas tuote myös tyytyväiset asiakkaat ja työntekijät.

## Viiteluettelo


[Haikala ja Märijärvi, 1997] I. Haikala ja J. Märijärvi, *Ohjelmistotuotanto.* Suomen ATK-kustannus, Espoo, 1997.

[ISO, 2001] International Organization for Standardization, *ISO TC/176/SC2*. http://isotc176sc2.elysium-ltd.net. (1.10.2001)

[Juran, 1993] J.M. Juran, *Quality Planning and Analysis*. McGraw-Hill Publishing Company, 1993.

[McConnell, 1998] S. McConnell, *Ohjelmistoprojektit: selviytymisopas.* Suomen ATK-kustannus, Espoo, 1998.

[NASA, 1989] NASA, *Software Assurance Guidebook*. September 1989. http://satc.gsfc.nasa.gov/assure/assurepage.html. (1.10.2001)

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

[Ryynänen, 2001] S. Ryynänen, *Uudet ISO 9000 –standardit valmistuivat, mikä muuttui?* SFS-tiedotus 1/2001.

[Salminen, 1994] H. Salminen, *Laadulla tulosta: asiantuntijayrityksen uudet toimintamallit ja organisaatiorakenteet.* Hansa-Dialog, Jyväskylä, 1994.

[SFS, 1997] Suomen standardisoimisliitto, *ISO 9000 pk-yrityksille : standardien ISO 9001, ISO 9002 ja ISO 9003 soveltamisohjeita.* Suomen standardisoimisliitto, Helsinki, 1997.

[SFS, 2001] Suomen standardisoimisliitto, *SFS standardisointi*. http://www.sfs.fi/standard. (1.10.2001)

[Sytyke, 1993] Systeemityöyhdistys Sytyke ry., *Laatujärjestelmän käyttöönotto.* Suomen ATK-kustannus, Espoo, 1993.

[Virtanen, 1990] V. Virtanen, *ISO 9000, perusta toiminnan kehittämiselle.* Metalliteollisuuden kustannus, Helsinki, 1990.

[Wesselius ja Ververs, 1990] J. Wesselius and F. Ververs, Some elementary questions on software quality control. *Software Engineering Journal* **5**, 6 (March 1990).

[Yli-Olli, 1991] P. Yli-Olli, *Softa 9000: ohjelmistotuotannon laadun kehittäminen.* Mecrastor, Espoo, 1991.

# Puhekäyttöliittymät ja niiden suunnittelu

## Esa-Pekka Salonen

**Tiivistelmä.**

Puhetta pidetään yleisesti luonnollisimpana ja tehokkaimpana tapana ihmisten välisessä kommunikoinnissa. Osin tästä johtuen puhetta on pitkään yritetty tuoda ihmisen ja tietokoneen väliseksi vuorovaikutuskanavaksi. Pitkään puheella ohjattavat tietokoneet olivat vain sci-fi-elokuvien unelmia. Nykyään puheentunnistusmenetelmät ja varsin luonnollista kuulostavat puhesyntetisaattorit mahdollistavat erilaisten sovellusten kehittämisen puheella ohjattaviksi. Vaikka puheteknologia on kehittynyt nopeasti se ei kuitenkaan ole virheetöntä. Tästä johtuen hyvällä suunnittelulla joudutaan paikkaamaan teknologian puutteita. Tässä tutkielmassa selvitetään minkälaisia suunnittelumenetelmiä on sovellettu ja minkälaisiin tuloksiin niitä käyttämällä on päästy.

Avainsanat ja -sanonnat: Puhekäyttöliittymät, suunnittelu, arviointi
CR-luokat: H.5.2

## 1. Johdanto

Ottamalla puhe käyttöliittymäelementiksi voidaan saavuttaa merkittäviä etuja perinteisiin käyttöliittymätyyppeihin verrattuna. Puhetta käyttämällä otetaan käyttöön vuorovaikutusmodaliteetti, jota voidaan käyttää silloinkin, kun muita tapoja ei voida käyttää [Kamm, 1994]. Tällaisia tilanteita voivat olla sovellusten käyttäminen uusissa ympäristöissä ja tilanteissa, esimerkiksi puhelimen välityksellä tai jos käyttäjällä on jokin fyysinen rajoite, joka tekee perinteiset tietokoneen käyttötavat mahdottomiksi. Toisaalta puheen käyttö mahdollistaa huomattavasti laajemman käyttäjäkunnan, kun sovelluksia voidaan siirtää puhelimella käytettäviksi [Zue et al. 2000].

Puhekäyttöliittymät voidaan luokitella kolmeen erilaiseen alaluokkaan, kun luokitteluperusteena käytetään puheen osuutta käyttöliittymässä. Laajimmin puhetta hyväksikäyttävää sovellusta kutsutaan yleisesti *vain puheeseen perustuvaksi* sovellukseksi (speech-only). Kun puhe on vain yksi modaliteetti muiden rinnalla, puhutaan yleisesti *puheen mahdollistavasta* (speech-enabled) sovelluksesta. Kolmas alaluokka ovat sellaiset sovellukset, joissa käytetään äänikäyttöliittymää. Esimerkkinä mainittakoon pankkien puhelinpalvelut, joissa nauhoitettua puhetta käytetään tulosteina ja puhelimen näppäimistöä tiedonsyöttömekanismina. Tässä tutkielmassa keskitytään kahteen ensimmäiseen alaluokkaan.

Toiseksi tässä tutkielmassa selvitetään vuorovaikutustapahtuman määrittelyä. Vuorovaikutustapahtuman määrittelyyn kuuluu niin sopivan sanaston löytäminen kuin dialogin kulun selvittäminen ja määrittäminen erilaisia menetelmiä käyttäen. Kolmanneksi tässä tutkielmassa keskitytään siihen kuinka tietoa tulisi käyttäjälle esittää siten, että se on ymmärrettävää, auttaa käyttäjää käyttämään järjestelmän ymmärtämää kieltä ja että tiedon esitysmuoto olisi mahdollisimman miellyttävä ja luonnollinen. Viimeisenä kohtana tutkielma käsittelee puhesovelluksien käytettävyyttä ja pyrkii selvittämään millaisia keinoja puhesovelluksien käytettävyyden tutkimisessa käytetään, sekä millaisiin tuloksiin näillä menetelmillä on päästy.

## 2. Dialogimallin valinta

Dialogisuunnittelu on yksi puhesovelluksen suunnittelun tärkeimmistä osista, koska dialogimalli lopulta määrää millainen puhesovelluksen käyttöliittymä on. Dialogimallia on toisaalta myös sovelluksen tietojenkeräämisstrategia eli se tapa, jolla sovelluksen tarvitsemat tiedot käyttäjältä kerätään. Huonosti suunniteltu dialogimalli voi tehdä koko sovelluksen käyttökelvottomaksi, vaikka käytössä olisi täydellinen puheentunnistin ja täysin ihmisenkaltainen puhesyntetisaattori. Toisaalta hyvällä dialogimallilla voidaan paikata tekniikan heikkouksia, esimerkiksi antamalla käyttäjälle vihjeitä minkälaista kieltä ja sanoja järjestelmä ymmärtää.

Puhesovelluksen suunnitteluun vaikuttavia tekijöitä ovat Kammin [1994] mukaan sovelluksen avulla suoritettava tehtävä, tekniikan tarjoamat mahdollisuudet ja käyttäjäkunnan erityispiirteet. Kaikki mainitut elementit vaikuttavat siihen, millainen dialogimalli sovellukseen on tarkoituksenmukaista valita, jotta vuorovaikutus olisi tehokasta ja miellyttävää. Onnistunut dialogi koneen kanssa on Kammin [1994] mukaan sellainen, joka toteuttaa käsillä olevan tehtävän tehokkaasti ja helposti, ihmisen näkökulmasta katsoen. Tähän pyritään muun muassa tehtävään sopivan dialogimallin valinnalla.

Dialogimallilla voidaan tarkoittaa paitsi ihmisen ja koneen mallidialogia niin myös sitä, kuinka järjestelmä kerää tarvitsemiaan tietoja, toisin sanoen sitä, millainen tiedonkeräämisstrategia järjestelmällä on. Järjestelmä voi kerätä tietoja esimerkiksi lomakepohjaisesti, jolloin järjestelmä kysyy tietoja käyttäjältä kunnes kaikki tarvittavat tiedot on täytetty se sisäiseen lomake-esitykseen. Tämän jälkeen järjestelmä voi tuottaa vastauksen keräämiensä tietojen avulla siihen ongelmaan, jota varten se on suunniteltu. Tätä dialogimallia kutsutaan *dialogistrategiaksi* (Dialogue Strategy). Tämä dialogimalli osa järjestelmän sisäistä toimintaa ja sinällään on näkymätön käyttäjälle, mutta tietysti se miten järjestelmä kerää tietoja vaikuttaa siihen kuinka dialogi keskustelun tasolla hoidetaan.

Karkeasti dialogin kulkua kuvaavat dialogimallit voidaan jakaa kolmeen ryhmään sen perusteella, millä taholla aloite dialogissa on. Kun kuvataan sitä, mikä taho kontrolloi dialogin kulkua, on luonnollista puhua *dialogikontrollimallista* (dialogue control model). Kontrolli tai aloite dialogissa voi olla käyttäjällä (user-initiative), järjestelmällä (system-initiative) tai se voi vaihdella näiden välillä (mixed-initiative) tilanteen mukaan. Uusimmat järjestelmät ovat lähes pääsääntöisesti sellaisia, joissa aloite vaihtelee järjestelmän ja käyttäjän välillä. Näissä järjestelmissä pyritään keskustelevaan järjestelmään, jossa dialogi järjestelmän ja käyttäjän välillä on yhteistoiminnallista siinä mielessä, että kumpi taho tahansa voi ottaa aloitteen, jos tehtävän suorittaminen sitä vaatii.

**Aloite järjestelmällä**

Järjestelmän aloitteellisuus on perinteinen dialogimalli, jota on käytetty osin koska tekniikka ei ole mahdollistanut kuin yksinkertaisten kysymysten esittämisen käyttäjälle. Perinteisissä sovelluksissa dialogi etenee ennalta määrätyssä järjestyksessä, jonka eri vaiheissa järjestelmä kysyy tarvittavia tietoja käyttäjältä, kunnes se tuottaa keräämiensä tietojen perusteella vastauksen annettuun ongelmaan. Toki nykyisissä järjestelmissä käytetään järjestelmäaloitteellisuutta huomattavasti hienovaraisemmin, järjestelmä voi esimerkiksi hyväksyä käyttäjältä laajemman vastauksen kuin mitä häneltä kysyttiin. Antamalla aloite kokonaan järjestelmälle sovelluksien toiminta on hyvin ennakoitavissa ja niiden toiminta on myös kohtuullisen varmaa. Toisaalta järjestelmät, jotka käyttävät puhetta ja sen ominaisuuksia näin köyhästi, ovat myös kovin jäykkiä. Niiden heikkoutena on sanavaraston suppeus, joka juuri tuo niille niiden toimintavarmuuden.

Ei ole kuitenkaan mitään syytä aliarvioida järjestelmäaloitteista dialogistrategiaa, sillä se on varsin toimiva ja tehokas joissakin tapauksissa. Jos esimerkiksi halutaan varmistaa, että käyttäjä varmasti antaa tietoja joihinkin tiettyihin asioihin voi järjestelmän aloittellisuus olla hyvä ratkaisu. Tällaisia tapauksia ovat sellaiset, joissa täytyy olla varma siitä että, käyttäjä antaa juuri oikeita tietoja niihin kysymyksiin joita järjestelmä esittää. Esimerkki tällaisesta sovelluksesta on sovellus, jolla hoidetaan pankkiasioita.

**Aloite vaihtelee**

Nykyiset järjestelmät pyrkivät huomattavasti joustavampaan ja keskustelun kaltaiseen dialogimalliin. Tarkoituksena on, että käyttäjä ja voi antaa järjestelmälle useita tietoja samanaikaisesti ja vapaammassa järjestyksessä. Toisaalta järjestelmä voi ottaa aloitteen jos se huomaa, että jokin sen tarvitsema tieto puutuu. Tällä tavalla pyritään luomaan järjestelmiä, joissa ihmiset voisivat käyttää keskustelussa samoja keinoja

kuin ihmisen ja ihmisen välisissä keskustelutilanteissa. Tällöin vuorovaikutuksesta tulee luonnollisempaa ja sen lisäksi myös tehokkaampaa [Kamm, 1994].

Jotta järjestelmä voisi toimia varmasti myös vaihtelevalla aloitteella, on sen ymmärrettävä laajempaa sanavarastoa kuin tiukasti systeemialoitteellisessa lähestymistavassa, koska kun aloite on käyttäjällä syötteet ovat väistämättä monimuotoisempia kuin systeemilähtöisessä dialogissa. Käytännössä tämä tarkoittaa sitä, että järjestelmän *aihealue* (domain) ja siinä käytettävät ilmaisut on kartoitettava tarkasti, jotta käyttäjä voisi käyttää tämän tietyn ympäristön sanontoja suhteellisen vapaasti [Hickey ja St John Brittan, 2001].

## 3. Vuorovaikutustapahtuman suunnittelu

Koska nykyaikaisinkaan puheentunnistin ei kykene reaaliaikaiseen tunnistukseen vapaasta puhevirrasta täytyy, puhesovelluksille käytännössä määritellä sanasto, kielioppi ja dialogimalli. On tärkeää, että valitaan sellainen sanasto, jota ihmiset yleisesti käyttävät, puhuessaan sovelluksen kohteena olevista asioista. On myös havaittu, ettei sanastoa voi suoraan kirjoittaa esimerkiksi vastaavasta graafisesta sovelluksesta, koska graafisten käyttöliittymäelementtien komennot ja komentojen nimet saattavat poiketa merkitsevästi siitä, miksi niitä puhekielessä kutsutaan [Yankelovich, 1997]. Kirjallisuudessa on esitetty useita menetelmiä, joilla voidaan tutkia, kuinka käyttäjät toimivat vuorovaikutustilanteessa ja tällä tavoin selvittää, minkälaista kieltä käyttäjät missäkin ympäristössä käyttävät. Seuraavassa esitetään kolme erilaista lähestymistapaa vuorovaikutustilanteen arvioimiseen ja dialogikontrollin suunnitteluun: ihmisten välisen kommunikaation tutkiminen, järjestelmien simulointi ja vuorovaikutuksen tutkiminen toimivilla järjestelmillä. Näillä menetelmillä kartoitetaan vuorovaikutustilannetta ennen sovelluksen lopullista toteuttamista, jotta vältettäisiin ainakin ilmeisimmät sudenkuopat. Toisaalta nämä menetelmät voivat kertoa myös sovelluksen käytettävyydestä ja siitä, kuinka sovellusta tulisi muuttaa paremman käytettävyyden saavuttamiseksi.

### Ihmisten välisen kommunikoinnin tutkiminen

Dialogin ja sanaston selvittämiseksi voidaan tutkia, millaista dialogi on ihmisten välisessä dialogissa vastaavan tehtävän parissa. Yksi menetelmä puhesovellusten dialogin suunnitteluun on *esitutukimus* (Pre-Design Study), jossa tutkitaan ihmisten välistä kommunikointia vuorovaikutustilanteessa [Yankelovich, 1997]. Esitutkimuksella on tarkoitus selvittää sanaston ohella myös muita seikkoja. Yankelovichin [Yankelovich, 1997] mukaan esitutkimusta on hyvä käyttää seuraaviin kohteisiin:

1. sovelluksen vaatimusmäärittelyn ja toiminnallisuuden jalostamiseen

2. soveliaan sanaston keräämiseen
3. tavallisesti käytettyjen kieliopillisten rakenteiden määrittämiseen
4. tehokkaiden vuorovaikutuskuvioiden löytämiseen
5. järjestelmäpuheenvuoro- ja palautesuunnittelun helpottamiseen
6. keskustelun yleissävyn löytämiseen

Esitutkimus suoritetaan havainnoimalla mahdollisen jo olemassa olevan vastaavan palvelun dialogia, esimerkiksi käyttäjän ja puhelinkeskuksen hoitajan välillä. Esitutkimus voidaan myös suorittaa luomalla tilanne, jossa ihminen toimii tulevan järjestelmän tehtävissä, jos vastaavaa perinteistä palvelua ei ole olemassa. Esimerkiksi Yankelovich esittää tehneensä Office Monitor järjestelmälle esitutkimuksen siten, että hän oli asettanut avustajan hänen huoneeseensa istumaan ja avustaja oli vastaillut Yakelovichiä etsimään tulleiden ihmisten kysymyksiin. Nämä dialogit oli videoitu ja tutkittu, ja havaintojen perusteella sitten määriteltiin, millaisiin kysymyksiin ja millä tavoin Office Monitorin tulisi vastata. Office Monitor on automatisoitu vastaanottoapulainen, joka on sijoitettu Yankelovichin työhuoneeseen [Yankelovich, 1997].

Esitutkimus eroaa olennaisesti seuraavaksi esiteltävästä Wizard of Oz -menetelmästä. Esitutkimuksessa on tarkoitus kartoittaa ihminen-ihminen dialogin ulottuvuudet tulevan sovelluksen alueella, kun taas Wizard Of Oz paljastaa dialogin luonteen ihmisen ja koneen välillä. Yankelovich on saanut esitutkimus menetelmää hyväksi käyttäen paljon hyviä tuloksia ja ohjenuoria etenkin edellä olleessa listassa määritellyistä kohdista, mutta myös siitä, mihin tai millaisiin tehtäviin puheteknologiaa ei tulisi soveltaa [Yankelovich, 1997].

**Järjestelmän simulointi Wizard of Oz menetelmää hyväksikäyttäen**

*Wizard of Oz* (WoZ) on menetelmä, jossa koko sovelluksen toimintaa tai osaa siitä ohjaa ihminen. On tärkeää, että käyttäjät luulevat olevansa tekemisissä oikean järjestelmän kanssa, jotta vuorovaikutustapahtuma olisi mahdollisimman aito. Esitutkimuksessa tämä ei niinkään ole tärkeää, vaan siinä merkitsee ainoastaan ihmisten keskenään käyttämä kieli ja konventiot. WoZ soveltuu hyvin puhekäyttöliittymien suunnittelu- ja/tai iterointivaiheeseen, jolloin voidaan korvata esimerkiksi puheentunnistin. Näin käyttäjä voi käyttää vuorovaikutuksessa vapaampia ilmaisuja ja tällä tavoin saadaan hyödyllistä tietoa siitä, minkälaista kieltä käyttäjät käyttävät puhuessaan järjestelmille [Mäkelä et al., 2001]. WoZ-menetelmää varten moniin puheohjelmistoarkkitehtuureihin on kehitetty välineitä ja työkaluja, joilla on helppo suunnitella ja toteuttaa WoZ-tutkimus. Eräs tällainen arkkitehtuuri on Tampereen yliopistossa kehitettävä Jaspis [Turunen and Hakulinen, 2000].

WoZ-tutkimuksia tehdään monilla eri tavoilla. Niitä voidaan tehdä jo valmiille tai lähestulkoon valmiille järjestelmille, mutta niitä voidaan myös tehdä siten, että koko järjestelmän toimintaa simuloidaan. Kun koko järjestelmän toimintaa simuloidaan, on tyypillistä, että koko järjestelmä on itse asiassa ihminen, jonka puhetta käsitellään jollakin efektilaitteella siten, että se kuulostaa koneelta. Jos taas tehdään WoZ-tutkimus valmiimmalle järjestelmälle ihminen hoitaa vain jotakin osaa järjestelmän toiminnasta esimerkiksi siten, että puheentunnistin korvataan ihmisellä [Mäkelä et al., 2001].

Kummallakin esitetyllä tavalla tehdä WoZ-tutkimus on omat vahvuutensa ja heikkoutensa. Valmiin järjestelmän tulosteet on jo lukkoon lyötyjä, jolloin ne saattavat ohjata dialogia ja menetelmän käyttö ei tuo tutkimukseen haluttua lisäarvoa. Toisaalta testaaja saattaa tuoda omaa tietouttaan dialogiin, jos koko järjestelmää simuloidaan ihmisen toimesta. Tämä taas saattaa johtaa jopa täysin virheellisiin tuloksiin [Dahlbäck et al. , 1993]. Myös järjestelmän uskottavuus saattaa vinouttaa tuloksia etenkin, jos käyttäjät eivät lainkaan usko kommunikoivansa järjestelmän kanssa. Keskeneräisiä tai puoliksi toteutettuja järjestelmiä ei suositella simuloitavaksi, koska silloin WoZ-menetelmä helposti paljastaa vain järjestelmän virheet, eikä vuorovaikutustapahtuman ongelmia, niin kuin on tarkoitus [Dahlbäck et al. , 1993]. Suorittipa WoZ-tutkimuksen kuinka tahansa hyvien tulosten saavuttamiseksi on tutkimus- ja testiolosuhteet valmisteltava ja suunniteltava huolellisesti.

**Vuorovaikutuksen määrittäminen toimivilla järjestelmillä**

Varmasti yleisin tapa sovellusten kehittämisessä on kehittää sovelluksia iteroimalla. Vaikka soveltaisi kumpaakin edellä mainituta menetelmää, joutuu luultavasti korjaamaan ja kehittämään järjestelmää kun se otetaan käyttöön oikeassa ympäristössä oikeilla käyttäjillä. Iterointimenetelmää voi myös soveltaa ilman edellä mainittuja kokeellisia menetelmiä. Hickeyn ja St Johnin mukaan yksi tapa kerätä tarvittavaa tietoa vuorovaikutuksesta on, että aluksi kehitetään sovellus eksperttikäyttäjiltä kerätyllä aineistolla ja vähitellen laajennetaan käyttäjäkuntaa iteroiden sovellusta jokaisen laajennuksen yhteydessä [Hickey ja St John Brittan, 2001].

Nopea tapa saada tuntuma siihen, millainen puhesovelluksen tulisi olla, on käyttää pikasovelluskehittimiä (Rapid Application Developer). CSLU Toolkit on yksi tällainen ympäristö, jolla voi tehdä puhesovelluksia nopeastikin [McTear, 1999]. Toinen samankaltainen ympäristö on tšekinkielinen LOTOS, joka tosin toistaiseksi toimii vain yksinkertaisilla yhden sanan syötteillä [Nouza and Nouza, 2001]. Nämä ympäristöt ovat sellaisia, joissa graafisessa ympäristössä suunnitellaan puu tai polku, joka kuvaa kuinka sovelluksen dialogi kulkee. Polun tai puun eri kohtiin voidaan

sijoittaa erilaisia toimintoja syötteiden pyytämistä, tulosteita tai vaikkapa tietokantahakuja. Tällaisen ympäristön käyttö mahdollistaa myös puhevuorovaikutuksen tutkimisen ja sovellusten tekemisen sellaisillekin ihmisille, joilla ei ole tietojenkäsittelyn tietoja. Näiden ympäristöjen haittana on se, että ne tukevat usein vain sitä kieltä jolla ne on kehitetty eikä monikielisyyttä ole otettu huomioon. Eli jos aikoo toteuttaa suomenkielisiä sovelluksia nämä järjestelmät eivät tarjoa apua.

Toinen tapa nopeiden prototyyppien kehittämiseen on VoiceXML merkitsemiskieli, joka mahdollistaa puhesovellusten tekemisen verkkoon samaan tapaan kuin HTML mahdollistaa graafisten sovellusten tekemisen. [Larson, 2001] VoiceXML on kielien suhteen yhtä ongelmallinen kuin edellä esitetyt pikakehittimet.

## 4. Tiedon esittäminen

Tiedon esittäminen vain puheella siten, että käyttäjä saa kaiken tiedon on haasteellinen tehtävä johtuen jo pelkästään puheen ominaisuuksista. Tämän lisäksi puhejärjestelmien tulosteet tulisi suunnitella siten, että tiedon esitysmuoto auttaa käyttäjää muodostamaan järjestelmän ymmärtämää kieltä tai ohjaamaan käyttäjää hienovaraisesti käyttämään oikeita sanontoja. Lisäksi järjestelmäpuheenvuorot tulisi suunnitella siten, ettei käyttäjä turhaudu dialogiin, vaan käyttää järjestelmää ensikokeiluidenkin jälkeen.

Puhe on luonteeltaan täysin erilainen vuorovaikutuskanava kuin esimerkiksi teksti tai graafinen ympäristö. Graafisessa sovelluksessa piilossa olevaa tietoa ja toiminnallisuutta voidaan tuoda näkyviin ruudulle käyttäjän ulottuville, kun taas puhesovelluksissa, etenkin vain puheeseen perustuvissa järjestelmissä, esimerkiksi menujen, nappien tai muiden käyttöliittymäelementtien näyttäminen ei ole mahdollista [Yankelovich, 1996]. Koska ääni on luonteeltaan lineaarista ajassa tapahtuvaa informaation esittämistä, niin tiedon esitys puheella tai äänellä on ainutkertainen tapahtuma. Puhe ei jää käyttäjän ulottuville. Tämän lisäksi on havaittu, että käyttäjät usein yliarvioivat puhejärjestelmien kyvyn ymmärtää luonnollista kieltä [Yankelovich, 1996]. Kuinka siis suunnitella järjestelmäpuheenvuorot siten, että ne ovat luonnollisia, ohjaavat käyttäjät käyttämään järjestelmän ymmärtämää kieltä ja että kaikki oleellinen tieto tulee käyttäjälle tietoon?

Tiedon esitystapa riippuu oleellisesti siitä, minkälaista esitettävä tieto on ja siitä kenelle tietoa esitetään. Erilaisille kohderyhmille on hyvä suunnitella erilaisia tulosteita. Jo pelkästään se, kuinka kokenut käyttäjä on vaikuttaa suuresti siihen kuinka hänelle on hyvä esittää tietoa. Toiseksi on hyvä ottaa huomioon sovelluksen kohderyhmä tulosteita suunnitellessa. On intuitiivisesti selvää, että jos esimerkiksi

suunnitellaan sovellusta nuorisolle niin tulosteissa pitää käyttää erilaista kieltä kuin jos kyseessä on sovellus eläkeläisille. [Kotelly, 2001]

Puheentunnistimien ominaisuudet vaihtelevat niiden ymmärtämien sanojen määrän, *keskeyttämisen* (barge-in) ja muiden ominaisuuksien suhteen. Tämä ei voi olla vaikuttamatta siihen, minkälainen tiedon esittämistapa valitaan. Yankelovichin [1996] mukaan puhetulosteiden ohjaavuus voi olla eksplisiittistä, implisiittistä tai jotakin tältä väliltä. Eksplisiittiset tulosteet sopivat parhaiten pienen sanavaraston puheentunnistimille. Jos halutaan käyttää implisiittisiä tulosteita, tarvitaan laajan sanavaraston ja monimutkaisia kieliopillisia rakenteita ymmärtävä puheentunnistin. [Yankelovich, 1996]

**Eksplisiittiset tulosteet**

Esimerkissä 1 [Yankelovich, 1996] puhetulosteet ovat varsin suoria ja kysyvät eksakteja kysymyksiä, joka todennäköisesti ohjaa käyttäjän vastaamaan eksaktisti kysyttyyn kysymykseen. Esimerkissä hyödynnetään kuitenkin järjestelmäpuheenvuoron keskeytystä (barge-in), joka on hyvä keino kun kyseessä on pitkä lista, josta käyttäjä valitsee haluamansa kohteen. Ilman tätä keskeytysominaisuutta käyttäjällä saattaisi olla vaikeuksia muistaa, mitä vaihtoehtoja hänelle esitettiin.

Comp: Please say the name of the artist you want.

User: The Beatles.

Comp: I have a lot of albums by the Beatles; please say the name of the album you want. For example, you could say “Magical Mystery Tour” or “Abbey Road.” For a full list of albums I have by The Beatles, say “list.”

User: List.

Comp: Here are the albums I have by the Beatles. Say the one you want as soon as you hear it. “With the Beatles.” “Beatles For Sale.” “Revolver.” “The White Album.” “Please Please Me....

User (interrupting): The White Album.

Comp: I think you said, “The White Album.” Is that correct?

User: Yes.

Comp: Okay. To sample “The White Album”...

Esimerkki 1. Eksplisiittisesti ohjattu esimerkkikeskustelu

**Implisiittiset tulosteet**

Asia voidaan esittää käyttäjälle myös implisiittisesti siten, että hänen antamistaan syötteistä annetaan varmistus muun tiedon mukana. Tällä tavoin annetaan käyttäjälle mahdollisuus korjata antamiaan tietoja, kun hän huomaa järjestelmän tehneen virheen, aivan kuten luonnollisessa keskustelussa. Seuraavassa esimerkissä on huomattavaa, että käyttäjä aloittaa dialogin ja vasta viimeisessä tulosteessa annetaan palaute käyttäjän antamista tiedoista. Toinen huomionarvoinen seikka on se, että järjestelmä osaa kysyä puuttuvan tiedon käyttäjältä. Tämä on luonteenomaista juuri järjestelmille, joissa aloite vaihtelee käyttäjän ja järjestelmän välillä. [Yankelovich, 1996]

User: I'd like to fly from Boston to London.

Comp: What date will you be travelling on?

User: I want to leave next Friday.

Comp: Here are the flights from Boston to London on Friday, May 31...

Esimerkki 2. Implisiittisesti ohjattu esimerkkikeskustelu

**Tulosteiden ymmärrettävyyttä parantavat menetelmät.**

Ei pelkästään riitä, että käyttäjät tietävät, mitä heidän tulisi sanoa minkäkin toiminnon aikaansaamiseksi. Jotta sovelluksesta voisi olla ylipäänsä mitään hyötyä, sen täytyy suorittaa jokin tehtävä tai toimia työkaluna sen suorittamisessa. Tämä tarkoittaa sitä, että käyttäjälle tulee esittää tietoa siten, että se on ymmärrettävää ja helposti muistettavaa. Tietoa on siis pilkottava ihmiselle sopiviksi paloiksi, jotka ovat sopivan mittaisia, jotta kaikki tieto tulee ymmärretyksi.[Kamm, 1994]

Järjestelmäpuheenvuorojen pituus muodostuu ongelmaksi, jos esitettävää asiaa on paljon. Liian pitkät tulosteet eivät pelkästään vaikeuta asioiden muistamista ja aiheuta asioiden sekaantumista, vaan ne myös saattavat heikentää järjestelmän käytön miellyttävyyttä. Etenkin kokeneet käyttäjät kyllästyvät järjestelmän ohjaaviin tulosteisiin, jotka ovat taas ensikäyttäjälle välttämättömiä. Tähän ongelmaan on kehitetty monia erilaisia ratkaisuja.

Järjestelmiin voidaan rakentaa käyttäjäprofiili, josta on helppo määritellä millaisia tulosteita millekin käyttäjälle tarjotaan [Marx and Schmandt, 1996]. Käyttäjäprofiili voi olla käyttäjän itsensä määriteltävissä tai sitä voidaan muokata dynaamisesti keskustelun aikana sen perusteella miten, ihminen sovellusta käyttää. Käyttäjäprofiiliin voidaan esimerkiksi merkitä toiminnot, joita käyttäjä käyttää useimmin, ja niin voidaan päätellä, ettei hän tarvitse ohjaavia tulosteita juuri näihin komentoihin tai toimintoihin. Toisaalta jos järjestelmä tarjoaa mahdollisuuden

järjestelmäpuheenvuoron keskeyttämiseen, voi käyttäjä koska tahansa keskeyttää järjestelmän, jolloin hänen ei tarvitse kuunnella pitkiä ohjaavia tulosteita.

Toinen tapa lyhentää ja näin selkeyttää järjestelmän tulosteita on mukauttaa niitä keskustelun edetessä siten, että samanlaisena toistuvia asioita jätetään sanomatta, kun käyttäjä tekee onnistuneesti samaa toimintoa useamman kerran (tapering) [Yankelovich, 1996]. Tällainen lähestymistapa on muun muassa MailCall järjestelmässä, josta seuraava esimerkki on otettu [Marx and Scmandt, 1996]. Esimerkissä yliviivattuja osia ei enää toisella kertaa kerrota käyttäjälle, kun voidaan olettaa hänen nämä jo tietävän.

User: Send a message to Jordan Slott.

MailCall: Record your message for Jordan Slott. Pause for several seconds when finished.

User: *records a voice message*

MailCall: Got it. Review the message, send it, or cancel?

User: Send.

MailCall: Sending your message. Please wait.

User: Send a message to Steve Lee.

MailCall: Record your message for Steve Lee. ~~Pause for several seconds when finished.~~

User: *records a voice message*

MailCall: Got it. Review ~~the message~~, send it, or cancel?

User: Send.

MailCall: Sending your message. Please wait.

Esimerkki 3. Esimerkkikeskustelu MailCall järjestelmästä

Toinen tapa, jota MailCall järjestelmässä käytetään selkeyttämään ja parantamaan dialogia, on järjestelmäpuheenvuorojen nopeuden muuttaminen sellaisissa kohdissa, jotka toistuvat puheenvuorosta toiseen samanlaisina [Marx and Scmandt, 1996]. Käytännössä tämä siis tarkoittaa, että puhetuloste on nopeampaa sellaisissa kohdissa, jotka ovat kiinteitä, ja puhetta hidastetaan sellaisissa kohdissa, joissa on esimerkiksi dynaamisesti muotoiltua tietoa tai jos voidaan olettaa tiedon olevan käyttäjän kannalta tärkeää.

Toisaalta järjestelmäpuheenvuoroja voidaan pidentää jos havaitaan, että käyttäjä ei tiedä, mitä järjestelmälle voi sanoa tai mitä järjestelmä ymmärtää. Tätä metodia käyttäen käyttäjälle tarjotaan ensin implisiittinen ja näin avoimempi tuloste, ja jos käyttäjä on hämmentynyt ja ei sano mitään tai käyttää sellaisia syötteitä, joita järjestelmä ei ymmärrä, muutetaan tulostetta suoremmaksi.[Yankelovich, 1996]

Comp: Welcome to ABC Bank. What would you like to do?

User: (silence)

Comp: You can check an account balance, transfer funds or pay a bill.
What would you like to do?

User: (silence)

Comp: Say one of the following choices: check balance, transfer funds or pay bills.

Esimerkki 4. Keskustelun muuttaminen suoremmaksi

## 5. Puhesovelluksien arvioinnista

Puhesovelluksien arvioinnissa käytetään kahta erilaista lähestymistapaa. Puhesovelluksia evaluoidaan mittaamalla, kuinka hyvin järjestelmät ymmärtävät käyttäjien puhetta [Polifroni et al., 1998]. Tyypillisesti kerätään todella suuri määrä dataa, jonka perusteella tehdään arviointi siitä, kuinka hyvin järjestelmä ymmärtää käyttäjien puhetta. Tämä tapa antaa hyvin eksakteja ja tilastoitavia tuloksia, mutta kertovatko ne kuitenkaan lopullista totuutta puhesovelluksen käytettävyydestä, puhumattakaan hyödyllisyydestä?

Toisen lähestymistavan kannattajat luottavat käytettävyystutkimuksissa hyviksi havaittuihin menetelmiin, haastatteluihin ja tarkkailuun. Näilläkin menetelmillä on kiistämättömät heikkoutensa, kun joudutaan luottamaan käyttäjien subjektiivisiin arviointeihin. Toisaalta on tärkeää tietää, kuinka ihmiset kokevat järjestelmän palvelevan tarkoitustaan, eikä se aina selviä pelkistä tilastollisista arvioinneista.

Perinteisesti havainnoinnissa myös mitataan kuinka käyttäjät onnistuvat tehtävissä, joita sovelluksella on tarkoitus suorittaa. Toisaalta tehtävän suorittamiseen kuluva aika kertoo kuinka tehokas järjestelmä on. Toisaalta käyttäjän vasteajalla järjestelmän kysymyksiin voidaan arvioida järjestelmän ymmärrettävyyttä.

## 6. Päätelmät

Puhesovellusten suunnittelu, kehittäminen ja arviointi käyttäjäkeskeisestä näkökulmasta on monitahoinen ongelma. Yhtäältä tarvitaan tehokasta ja luotettavaa tekniikkaa ja toisaalta tekniikka ja laitteet menevät hukkaan, jos sovelluksia ei osata suunnitella ihmisille sopiviksi. Tarvitaan taitoa ja tietoa monilta tieteiden eri aloilta ja tätä tietoa hyväksikäyttäen muodostetaan yhtenäinen ja toimiva kokonaisuus. Tässä tutkielmassa on selvitetty miten voidaan suunnitella toimivia puhesovelluksia. Pitää huomioida käytössä olevat tekniset resurssit, sovelluksen käyttökohde, sovelluksen käyttäjäkunta ja lopulta keinot joilla voidaan helpottaa ihmisen kognitiivista kuormaa, että hän muistaa ja tietää kuinka järjestelmä toimii.

## Viiteluettelo


[Dahlbäck et al., 1993] N. Dahlcbäck, A. Jönsson and L. Ahrenberg, Wizard of Oz Studies – Why and How, *Proceedings of the 1993 International Workshop on Intelligent User Interfaces (IUI'93)*, ACM Press, 1993, 193-200.

[Hickey, Brittan, 2001] Marianne Hickey and Paul St John Brittan, Lessons from the Development of a Conversational Interface. In: *Proc. 7th European Conference On Speech Communication and Technology: Eurospeech 2001*, 1295-1298.

[Kamm, 1994] Candace Kamm, User Interfaces for Voice Applications, In Roe, D., Wilpon, J. (eds), Voice Communication Between Humans and Machines, National Academy Press, Washington D.C., 1994: 422-442.

[Kotelly, 2001] Christopher Kotelly, Brancusi, Neo-plasticism, and the Art of Designing Speech-recognition Applications, *Proceedings of Workshop on Automatic Speech Recognition and Understanding (ASRU 2001)*, Keynote III, Madonna di CampiglioDecember 2001.

[Larson, 2001] Jim A. Larson, VoiceXML and Natural Language Processing, *Proceedings of Workshop on Automatic Speech Recognition and Understanding (ASRU 2001)*, Keynote II, Madonna di Campiglio, December 2001.

[Marx and Schmandt, 1996] Matthew Marx and Chris Schmandt, MailCall:

Message Presentation and Navigation in a Non-visual Environment, Proceedings of CHI '96, April 1996, 165-172.

[Mäkelä et al., 2001] Kaj Mäkelä, Esa-Pekka Salonen, Markku Turunen, Jaakko Hakulinen and Roope Raisamo, Conducting a Wizard of Oz Experiment on a Ubiquitous Computing System Doorman, *Proc. International Workshop on Information Presentation and Natural Multimodal Dialogue IPNMD2001*, Verona, Italy, December 2001.

[McTear, 1999] M. McTear, Software to Support Research and Development of Spoken Dialogue Systems, *Proc. Eurospeech 1999*, Budapest, Romania, September 1999.

[Nouza and Nouza, 2001] Tomás Nouza and Jan Nouza, Graphic platform for designing and developing practical voice systems, *Proc. 7th European Conference On Speech Communication and Technology: Eurospeech 2001*.

[Polifroni et al., 1998] Joseph Polifroni, Stephanie Seneff, James Glass and Timothy J. Hazen, Evaluation methodology for a telephone-based conversational system, *Proceedings of the First International Conference on Language Resources and Evaluation*, Granada, Spain, May 1998, 42-50.

[Turunen and Hakulinen, 2000] Markku Turunen, and Jaakko Hakulinen, Jaspis - A Framework for Multilingual Adaptive Speech Applications. *Proceedings of 6th International Conference of Spoken Language Processing (ICSLP 2000)*, Beijing, China, 2000.

[Yankelovich, 1997] Nicole Yankelovich, Using Natural Dialogs as the Basis for Speech Interface Design, Submitted to MIT Press as a chapter for the upcoming book "Automated Spoken Dialog Systems," edited by Susann Luperfoy. Available as *http://research.sun.com/research/speech/publications/mit-1998/MITPressChapter.v3.html*

[Yankelovich, 1996] Nicole Yankelovich, How Do Users Know What to Say? , *ACM Interactions* **3**, 6 (Nov/Dec 1996) 32-43.

[Zue et al., 2000] Victor Zue, Stephanie Seneff, James Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen and Lee Hetherington. JUPITER: A telephone-based conversational interface for weather information, *IEEE Transactions on Speech and Audio Processing* **8**, 1 (Jan. 2000).

# Helppokäyttöisen kyselykielen suunnittelu osa-kokonaisuussuhteille

## Samu Viita

**Tiivistelmä.**

Osa-kokonaisuushierarkioita omaavia rakenteita sisältäville tietokannoille on olemassa kyselykieliä, joilla tiedon haku ja päivitys onnistuu. Ongelmana olemassa olevissa systeemeissä on kuitenkin heikko käyttäjäystävällisyys. Lisäksi joissakin lähestymistavoissa osa-kokonaisuussuhteen tuomia piirteitä, kuten periytymisen toteuttamista osien ja kokonaisuuksien välillä, ei tueta lainkaan. Tämän takia on tarpeellista kehittää kyselykieli, joka tukee osa-kokonaisuussuhteen käsittelyä, mutta on kuitenkin niin helppokäyttöinen, että ohjelmointitaidotonkin henkilö pystyy kyselykieltä käyttämään ilmaisuvoiman siitä kärsimättä.

Avainsanat ja -sanonnat: Tiedon esittäminen, osa-kokonaisuussuhde, kyselykielet, deduktiiviset oliotietokannat, indeksointimekanismi, kompleksiset objektit, komposiitit

CR-luokat: *H 3.3*

## 1. Johdanto

Osa-kokonaisuussuhde (part-of relationship) on tärkeä käsitteellinen mallintamisrakenne. Se on keskeisessä asemassa monilla aloilla ja sovellutuksissa, kuten tietokoneavusteisessa suunnittelussa (CAD) [Halper *et al.* 1998], lääketieteellisissä sovelluksissa [Halper *et al*. 1998; Paton *et al*. 1996], maantieteellisissä informaatiosysteemeissä (GIS) [Junkkari 2001; Paton *et al*, 1996], dokumenttien hallinnassa [Junkkari 2001], ja tekoälyn alueella [Motschnig-Pitrik and Kaasboll 1999; Junkkari 2001b].

Suhteen keskeisimmät käsitteet ovat komposiitti ja komponentti. Komposiitti tarkoittaa kokonaisuutta, joka muodostuu osista. Komponentti on puolestaan komposiitin sisältämä osa. Komponentin ja komposiitin suhde voi olla kaksisuuntainen. Komposiitilla on usein ominaisuuksia, jotka johdetaan sen jonkun tai joidenkin osien ominaisuuksista. Toisaalta komponentilla saattaa olla ominaisuuksia, jotka juontuvat sen komposiitista. Lisäksi suhteet eivät aina rajoitu välittömiin komponentteihin tai komposiitteihin, sillä kompleksisen rakenteen omaavilla objekteilla on monesti transitiivisia suhteita. Tämä tarkoittaa sitä, että jos osalla A on osa B ja B:llä osa C, myös A:lla on osa C. Tämä periaate pätee myös kompositioiden

kohdalla, eli jos A on B:n kompositio ja B on C:n kompositio, myös A on C:n kompositio. Tällaisissa tilanteissa tarvitaan transitiivisten suhteiden mallintamista ja käsittelyä.

Osa-kokonaisuussuhteessa on kaksi keskeistä tasoa, intensionaalinen- ja ekstensionaalinen taso. Intensionaalisuudella tässä työssä tarkoitetaan kaaviotasoa, joka määrää, miten ekstensionaalisen eli ilmentymätason tieto on organisoitu, toisin sanoen ilmaisee ilmentymätason yhteiset piirteet. Piirteisiin kuuluvat attribuutit ja mahdollisesti metodit. Intensionaalinen taso on siis tunnistettujen ja määriteltyjen käsitteiden joukko. Niiden ilmentymät tietokannassa tai reaalimaailmassa muodostavat ekstensionaalisen tason. Ekstensionaalinen taso on siis arvojen taso ja intensionaalinen taso käsitteiden taso.

Monesti olemassa olevissa systeemeissä osa-kokonaisuussuhteen ilmaiseminen ja analysointi on vaikeaa loppukäyttäjälle sen transitiivisesta luonteesta sekä monimutkaisesta rakenteesta johtuen. Analysointi saattaa edellyttää loppukäyttäjältä rekursiivisen tai iteratiivisen ajattelutavan sisäistämistä sekä tietoa esitystavan yksityiskohdista [Järvelin and Niemi, 1999]. Esiteltävän mallintamistavan pohjalta on mahdollista kehittää kieli, joka antaa tehokkaat navigointivälineet, jolloin käyttäjän ei tarvitse huolehtia transitiivisista suhteista. Myöskään kaaviotason ei tarvitse olla käyttäjän tiedossa eksplisiittisesti. Tällöin käytettävyyden pahimmat pullonkaulat osa-kokonaisuussuhteen analysoinnissa voidaan välttää.

Osa-kokonaisuussuhteessa periytyminen on useimmiten arvo-orientoitunutta periytymistä, eli arvot periytyvät ilmentymätasolla, kun taas normaalisti tietojenkäsittelyssä periytymissuhteella (is-a -suhteella) tarkoitetaan joidenkin intensionaalisten ominaisuuksien periytymistä. Is-a –hierarkiassa periytyvät siis *ominaisuudet*, kun taas osa-kokonaisuussuhteessa periytyvät usein ominaisuuksien *arvot*. Komposition jonkin ominaisuuden arvo voi määräytyä sen joidenkin komponenttien arvosta tai komponentin arvo voi puolestaan määräytyä sen komposition arvosta. Jos siis komponentin arvo muuttuu, se saattaa muuttaa myös komposiitin arvoa. Tämän takia tarvitaan mekanismi, jolla pystytään toteuttamaan monisuuntaiset periytymissuhteet käytettävyyttä heikentämättä. Tutkielmassa esitettävä esitystapa mahdollistaa tämänkaltaisen periytymisen toteuttamisen.

Tiedon esitystavan ideana on mahdollistaa intensionaalisen- ja ekstensionaalisen tason eksplisiittinen esittäminen. Tällöin esimerkiksi puhtaasti intensionaalisten kyselyjen tekeminen mahdollistuu. Esitystapa sisältää näiden kahden tason integroinnin Tällöin siihen perustuvalla kyselykielellä pystytään tekemään molempiin tasoihin ilmaisuvoimaisia ja käyttäjälle helposti ilmaistavissa olevia kyselyjä.

Kielen toteutuksen pohjalla on olioperustaisuus ja deduktiivisuus. Olio-pohjaisuuden kautta saadaan tehokas väline tiedon esitystavalle ja olioiden metodit

mahdollistavat luontevan tavan toteuttaa osa-kokonaisuussuhteelle tyypillinen periytyminen. Deduktiivisuudella saavutetaan korkea päättelykyky.

Tutkielman aluksi sevitetään osa-kokonaisuusuhteen luonnetta. Tämän jälkeen esitellään keskeisimmät olemassaolevat tavat osa- kokonaisuussuhteen käsittelyyn. Luvussa 4 pohditaan kyselykielelle vaadittavia ominaisuuksia, jotta suhteen käsittely onnistuisi käyttäjäystävällisellä tavalla. Viidennessä luvussa kerrotaan kyselykielen edellyttämän esitystavan vaatimuksista, joiden täyttyminen mahdollistaa edellisessä luvussa olevien vaatimuksien toteuttamisen. Kuudes luku lisää esitystavasta. Seitsemännessä luvussa kerrotaan kyselykielen primitiiveistä, joita uudella esitystavalla on mahdollista toteuttaa. Luvussa 8 on yhteenveto tutkielman tuloksista.

## 2. Osa-kokonaisuussuhde käsitteenä

Osa-kokonaisuussuhteen tarkoitus on tarkasteltavan kohdealueen kokonaisuuden ja sen osien välisen suhteen mallintaminen ja esittäminen. Usein puhutaan myös kompleksisista objekteista [Järvelin and Niemi 1999; Savnik *et al, 1999;* Carey *et al*, 1988] tai komposiittiobjekteista [Halper *et al* 1998*;* Motschnig-Pitrik and Kaasboll 1999]. Kompleksinen objekti ja aggregaatti ovat väljempiä käsitteitä. Ne voivat tarkoittaa myös muunlaista, kuin osa-kokonaisuussuhteesta muodostuvaa rakennetta. Motschnig-Pitrik ja muut [1999] tekevät erottelun komposition ja aggregaatin välille. Erottelu sopii yhtä hyvin myös kompleksiseen objektiin. Aggregaatin ja komposition semantiikka eroavat toisistaan heidän mukaansa siten, että aggregaatti voi olla jokin kompleksinen rakenne, jonka jäsenillä ei ole komponentti- tai komposiittisuhdetta keskenään. Esimerkkinä Motschnig-Pitrik kollegoineen ottaa huoneen, joka voidaan tulkita joko aggregaatiksi tai komposiitiksi. Jos huoneen tulkitaan koostuvan lattiasta, seinistä, ikkunoista ja ovista, puhutaan kompositiosta. Jos taas huoneen attribuutteina ovat omistaja, koko ja sijainti, tulkitaan huone aggregaatiksi. Aggregaatti on siis erottelussa käytännöl-lisemmässä merkityksessä. Tällöin huoneeseen yhdistetään attribuutteja, jotka liittyvät jollakin tapaa huoneeseen, mutta tässä tapauksessa attribuuttien tarkoituksena ei ole esittää osia, vaan mallintaa muita siihen liittyviä asioita. Sen sijaan edellinen kompositioesimerkki oli selvästi tarkoitettu mallintamaan osa-kokonaisuussuhdetta. Tässä työssä aggregaatilla tarkoitetaan samaa kuin edellä, jotta väärinkäsityksiltä vältyttäisiin tältä osin. Kompleksisista objekteista puhuttaessa tarkoitetaan osa-kokonaisuussuhteen omaavia kompositioita.

Osa-kokonaisuussuhteisiin liittyy muutamia peruskäsitteitä, joita tässäkin työssä käytetään. Perusolio (basic object) tarkoittaa kompositiossa olevaa oliota, jolla ei ole komponentteja. Ylimmän tason olio (super object) puolestaan tarkoittaa oliota, jolla ei ole kompositiota. Polku jostakin oliosta toiseen osa-kokonaisuussuhteessa tarkoittaa sitä transitiivista reittiä, jota pitkin päästään oliosta A olioon B.

Osa-kokonaisuussuhteita on useita eri tyyppejä ja suhteen tarkka rajaaminen on vaikeaa. Kielitieteen ja kognitiotieteen saralla on tehty ontologinen erotteluehdotus [Winston *et al.* 1987], joka pohjautuu erilaisiin osa-kokonaisuussuhteiden tyyppeihin. Artale ja muut [1996] ovat tehneet yhteenvedon tästä erotteluehdotuksesta:

**Komponentti/kokonaisuuden muodostava olio:** Kokonaisuuden muodostavilla olioilla on rakenne, jonka muodostavat erilliset komponentit, joilla on tietty oma merkitys tai toiminta kokonaisuuden kannalta. Esimerkiksi renkaat ovat osa autoa ja fonologia on osa kielitiedettä.

**Jäsen/kokoelma:** Muodostaa käsityksen jäsenyydestä kokoelmassa. Jäsenillä ei ole toiminnallista roolia kokonaisuudessa, mutta ne voidaan tunnistaa kokonaisuudesta. Esimerkiksi puu (jäsen) on osa metsää (kokoelma).

**Osa, osuus massasta:** Kokonaisuutta tarkastellaan homogeenisenä koosteena ja sen osuudet tai osat ovat samankaltaisia kuin kokonaisuus, mutta kuitenkin erotettavissa siitä. Esimerkiksi siivu on osa piirasta.

**Aine/olio:** Ilmaisee aineksena olevan olion: ”… on osittain puuta” tai ”…on tehty jostakin”. Ainetta, josta olio on tehty, ei voida erottaa oliosta, eikä sillä ole funktionaalista roolia.

**Piirre/toiminta:** Ilmaisee aktiviteetin vaiheen. Vaihe on kuin komponentti, jolla on toiminnallinen rooli, mutta se ei ole kuitenkaan erotettavissa. Esimerkiksi halkoon tarttuminen on osa halkojen pinoamista.

**Paikka/alue:** Avaruudellinen relaatio alueiden kesken, joilla eri oliot sijaitsevat. Jokainen osa on samanlainen kuin koko alue, mutta niitä ei voi erottaa toisistaan erilleen. Esimerkiksi lähde on osa aavikkoa.

Yllä olevaa erottelua on myös kritisoitu liiasta kielitieteellisestä näkökulmasta [Artale *et al.* 1996], mutta se antaa kuitenkin hyvän kuvan osa-kokonaisuussuhteen moniselitteisyydestä.

Kuten aikaisemmin kerrottiin, osa-kokonaisuussuhteessa on keskeisessä asemassa transitiivisuusominaisuus. Transitiivisuus ei ole kuitenkaan mielekäs joka tilanteessa. Ajatellaan esimerkiksi jalkapalloilijan kättä. Käsi on jalkapalloilijan osa ja jalkapalloilija on jalkapallojoukkueen osa, mutta käden ei voida sanoa olevan joukkueen osa. Transitiivisuusominaisuus menettää mielekkyytensä edellisessä esimerkissä siksi, että käden ja ihmisen suhteen luonne on erilainen, kuin ihmisen ja joukkueen suhde. Edellä olevan suhteiden erottelun mukaan jalkapalloilijan ja käden suhde on komponentti/ kokonaisuuden muodostava objekti –tyyppiä, kun taas jalkapalloilijan ja joukkueen suhde on jäsen/kokoelma –tyyppinen. Onkin pantu merkille, että transitiivisuusominaisuuden mielekkyys katoaa, jos erilaisia

suhdetyyppejä sekoitetaan keskenään [Artale *et al*, 1996]. Kehiteltävä kyselykieli on tarkoitettu tilanteisiin, joissa eri suhdetyyppejä ei sekoiteta keskenään.

**Ominaisuuksien periytyminen osa-kokonaisuussuhteessa**

Is-a ja part-of suhteet eroavat toisistaan [Halper *et al*, 1998;], vaikka joskus ne sekoitetaankin keskenään. Molemmat kuvaavat hierarkioita, mutta ne ovat luonteeltaan hyvin erilaisia.

Osa-kokonaisuussuhteeseen liittyy täysin oma semanttinen luonne, ja sitä ei voida is-a –suhteella mallintaa. Osat eivät juuri koskaan ole kokonaisuuk-siensa erikoistapauksia ja ominaisuuksien mahdollinen periytyminen on eri luonteista kuin is-a -suhteessa. Kuten aikaisemmin sanottiin, periytyminen part-of -suhteessa tapahtuu sekä intensionaalisella että ekstensionaalisella tasolla, kun taas is-a -suhteessa periytyminen on pelkästään intensionaalista.

Periytyminen voi tapahtua kahteen suuntaan, jolloin osat voivat periä joidenkin ominaisuuksien arvoja kokonaisuudeltaan tai kokonaisuus voi puo-lestaan periä ominaisuuksien arvoja yhdeltä tai useammalta osaltaan. Jos periytyminen tapahtuu komponentin tai komposiitin yhden ominaisuuden perusteella, periytyminen on suoraa. Jos periytyminen lasketaan useammasta osasta tai kokonaisuudesta, kyseessä ovat johdetut attribuutit [Halper *et al.* 1998]. Edellistä tapausta edustaa seuraava esimerkki: kokonaisuuden sijainti on sama kuin sen osien sijainti. Jälkimmäisestä on esimerkkinä polkupyörän paino, joka on sen kaikkien osien yhteenlaskettu paino.

Part-of –suhteessa sovellutuksen suunnittelijalla on vastuu siitä, miten periytyminen tapahtuu. Koska periytyminen voi olla johdettujen attribuuttien tapaista, luontevaa on käyttää metodia, joka laskee osista tai kokonaisuudesta tarvittavat laskutoimitukset. Tämän mekanismin tulisi olla läpinäkyvä loppukäyttäjälle.

**Osa-kokonaisuussuhteessa huomioitavia rajoituksia**

Käsitteellisessä mallintamisessa ja tietokantojen toteutuksessa tulee ottaa huomioon monia osa-kokonaisuussuhteen luonteesta johtuvia rajoituksia, kuten riippuvuus (dependency), poissulkevuus (exclusiveness) ja monikertaisuus (multiplicity) [Halper et al, 1998].

Riippuvuus osien kesken tarkoittaa, että jos jokin osa A on riippuvainen B:stä, A ei voi olla olemassa, jos B:tä ei ole olemassa. B voi tässä tapauksessa olla joko osa, kokonaisuus tai molempia. Riippuvuuden suunta voi olla kokonaisuudesta osiin tai osista kokonaisuuteen [Junkkari, 2001b]. Myös molemminsuuntainen riipuvuus on mahdollista. Ensimmäisestä esimerkkinä tilanne, jossa kirja poistetaan tietokannasta. Tällöin on luonnollista, että myös sen kappaleet ja sisällysluettelo poistetaan, joten tässä tapauksessa osat ovat riippuvaisia kokonaisuudesta. Jos taas polkupyörän runko poistetaan, sitä ei enää kokonaisuutena ole olemassa. Tässä tapauksessa kokonaisuus

on riippuvainen osastaan. Suunnittelussa täytyy myös ottaa huomioon, onko tarkoituksenmukaista tuhottaessa kokonaisuutta poistaa kaikki sen osat, vai voivatko esimerkiksi tuhotun pyörän osat jäädä olemaan, vaikka pyörä tuhottaisiinkin.

Poissulkeutuvuusominaisuus tarkoittaa sitä, voiko osa olla useamman kuin yhden kokonaisuuden osana vai ei. Kun on kyse konkreettisista esineistä, osalla on useimmiten ominaisuutena poissulkeutuvuus. Poissulkeutuvuudesta esimerkkinä on moottori, joka voi olla osana vain yhdessä autossa kerrallaan. Toisesta on esimerkkinä artikkeli, joka voi esiintyä useammassa eri lähteessä, kuten konferenssissa tai laitosraportissa yhtä aikaa.

Kolmas eli monikertaisuusrajoitus on tyypillinen kaikenlaisille suhteille käsitteellisessä mallintamisessa. Suunniteltaessa osakokonaisuussuhteita tulee ottaa huomioon, kuinka monta osaa objektilla voi vähintään ja enintään olla. Esimerkiksi tietokoneen emolevyssä voi olla yksi tai kaksi prosessoria kiinni, mutta muut määrät ovat kiellettyjä.

# 3. Olemassaolevia esitys– ja käsittelytapoja osa-kokonaisuus-suhteelle

Kun tietoa haetaan kompleksisesta rakenteesta, on yleistä, että käyttäjän tarvitsee ilmaista navigointi rakenteessa jonkinlaisena polkuesitystapana Tämä on tyypillistä esimerkiksi oliotietokannoille, kuten $O_2$:lle [Lecluse *et al.* 1988; Deux *et al.* 1990] tai ODMG-standardille [Cattell *et al. 2000*]. Polkuesitystapa tarkoittaa sitä, että käyttäjä määrittelee eksplisiittisesti reitin, jonka kautta haluttu tieto löytyy. Kuvitellaan seuraava tilanne. Käyttäjän tarvitsee selvittää, mikä on moottoripyörän takapyörän hinta. Silloin hän tarvitsee esimerkiksi seuraavanlaista navigointia:

moottoripyörä.takaosa.rengas.hinta.

Esimerkistä huomataan, että käyttäjän täytyy tuntea osa-kokonaisuussuhteen rakenne osien nimineen ja attribuutteineen täsmällisesti kyselyn formuloinnissa. Tämän tyyppinen navigointi on tyypillistä oliotietokannoille.

Oliomallissa on lisäksi ongelmana suhteiden väliset viittaukset. Jos suhteet on mallinnettu siten, että attribuutit ovat osoittimia olioihin, osoittava olio tietää osoitettavan, mutta osoitettava ei tiedä osoittavaa oliota [Junkkari, 2001]. Siitä seuraa, että komponentti ei tiedä, minkä kokonaisuuden osana se on. Monille olio-orientoituneille esitystavoille on siis mahdollista tietää osan komposiitti vain käymällä kaikki kompleksisia objekteja sisältävät rakenteet läpi komposiitista komponenttiin kuten edellä olevassa navigointiesimer-kissäkin tehtiin. Navigointi on siis yksisuuntaista. Tämä ei ole kuitenkaan ehdotonta olio-orientoituneissa ratkaisuissa,

sillä ODMG standardissa [Cattell *et al. 2000*] mallinnetaan suhteet joka oliossa kaksisuuntaisiksi sellaisissa olioissa, jotka osallistuvat johonkin suhteeseen. Jos siis kaksi oliota ottavat osaa suhteeseen, suhteen tieto pitää mallintaa molempiin olioihin, eli yhteensä neljä kertaa. Tämä on epätaloudellinen tapa mallintaa suhteet.

Monesti oliomallin yhteydessä käytetään luonteeltaan funktionaalisia kieliä, joissa käyttäjä voi muodostaa sisäkkäisiä metodirakenteita Tällöin siis jokin metodi palauttaa arvoksi olion, jolta voidaan kutsua edelleen palautetussa oliossa määriteltyjä metodeja. Esimerkkinä tällaisesta lähestymistavasta on QAL -kyselykieli [Savnik *et al*. 1999]. Tällöin voidaan saavuttaa korkea ilmaisuvoima. Deklaratiivisuuden aste on kuitenkin tällaisessa tapauksessa olematon. Lisäksi kaaviotason tietämys täytyy olla käyttäjän hallussa todella hyvin, jotta hän tietää kaikki tyyppien, attribuuttien ja metodien nimet sekä metodien parametrien määrät ja palautusarvon tyypit ja edelleen niiden mahdolliset metodit. Käyttäjältä vaaditaan myös ohjelmointitaitoja, joten ohjelmointitaidoton ihminen ei kieltä pysty käyttämään.

Aiemmin esitetyn polkuesitystavan sijasta toinen yleinen lähestymistapa on poistaa itse kompleksisuutta purkamalla komponenttien rakenne tasaiseksi, kompleksisuutta sisältämättömäksi rakenteeksi. Operaatio on tällöin *unnest*, ja päinvastainen hierarkiatason lisäävä operaatio on *nest*. Nest muodostaa relaatiosta attribuutin johonkin toiseen relaatioon, kun taas unnest korvaa relaatiomuotoisen attribuutin joukon alkiolla samalle tasolle, missä attribuuttikin sijaitsi tehden rakenteesta jälleen normaalilla relaatioalgebralla lähestyttävän. Tätä tapaa käytetään erityisesti $nf^2$ –relaatiomallissa (non-first normal form) [Roth et al. 1988], josta operaatiot ovat lähtöisin, mutta esimerkiksi oliomalliin pohjautuvassa kyselykielessä QAL:ssä [Savnik et al. 1999], voi polkuesityksen lisäksi käyttää myös nest ja unnest –operaatioita. Nest ja unnest -operaatioiden käyttö edellyttää käyttäjältä intensionaalisen tason eksplisiittistä tuntemusta, aivan kuten oliomallin navigoinninkin tapauksessa. Kummassakaan tavassa pelkästään intesionaaliseen tasoon kohdistuvia kyselyjä ei tueta lainkaan. Lisäksi $nf^2$ –relaatiomallissa on se heikkous, että periytyminen on part-of suhteen kohdalla mahdoton toteuttaa, kun käytössä ei ole oliomallista tuttua käyttäytymispiirrettä.

Deduktiivisilla tietokannoilla on suuri ilmaisuvoima. Niiden heikkoutena on kuitenkin käytön vaikeus rekursion ja mallinsovituksen johdosta. Lisäksi rakenteellisen tiedon esittäminen on vaikeaa [Järvelin and Niemi, 1999]. Ratkaisuna tiedon esittämisen ongelmaan ovat deduktiiviset oliotietokannat, joissa on yhdistetty olio-ominaisuuden esitysvoima ja metodien tuoma käyttäytymisaspekti ja deduktiivisten tietokantojen ilmaisuvoima. Mallin ongelmana on kuitenkin se, miten olioesitystavan ja deduktiivisuuden heikkoudet voidaan kitkeä pois.

# 4. Helppokäyttöisen ja ilmaisuvoimaisen kyselykielen edellytyksiä

Nykyiset kyselykielet ovat pääosin keskittyneet antamaan ekstensionaalisia vastauksia. Sen sijaan intensionaalinen taso näissä kyselykielissä täytyy olla eksplisiittisesti käyttäjän mielessä. Monissa tilanteissa olisi kuitenkin tärkeää, että myös intensionaaliseen tasoon liittyvää rakenteellisuutta voitaisiin analysoida. Intensionaalisella tasolla voidaan olla kiinnostuneita saamaan tietoa jonkin olion ominaisuuksien nimistä. Kompleksisen objektin luonteesta saa melko hyvän kuvan kysymällä, mitkä ovat sen komponenttien nimet ja niitä vastaavat ominaisuudet. Näiden kahden tason lisäksi erityistä ilmaisuvoimaa sisältyy kyselyihin, joissa käsitellään molempia tasoja samassa kyselyssä. Tällainen ominaisuus lisää kielen käytettävyyttä, kun tasoja voidaan yhdistää sekä kyselyissä että kyselyjen vastauksissa. Transitiivisia suhteita sisältäviä kyselyitä helpottaa, jos polkuja ei tarvitse määritellä, vaan voidaan kysyä komposiittiin/komponenttiin liittyvät sekä välittömät että välillisetkin komponentit/komposiitit. Helppokäyttöisyys siis kasvaa, jos voidaan kulkea kuvailevalla tavalla suhteiden välillä jolloin systeemin tehtävänä on transitiivisten suhteiden huomiointi ja laskenta [Järvelin and Niemi, 1999]. Lisäksi tärkeää on, että systeemin huoleksi voidaan jättää kaikkien polkujen läpikäynti kyselyissä, jossa tämänkaltaista ominaisuutta tarvitaan. Esimerkiksi kaikkien materiaaliltaan rautaa olevien komponenttien täytyisi löytyä ilman, että käyttäjä käy kaikkien olioiden kohdalla katsomassa löytyisikö rautaosia, jolloin käyttäjä joutuisi ilmaisemaan monia polkuesityksiä.

Jotta navigointi voisi tapahtua molempiin suuntiin, eli komposiitista komponenttiin ja päinvastoin, suhteita ei tulisi joutua mallintamaan edellä esitetyllä ODMG-standardin mukaisella tavalla redundanssin välttämiseksi.

Navigointiongelmien lisäksi ongelmia aiheuttaa attribuuttien nimien tarkka muistaminen. Monesti kyselyjä tehtäessä attribuuttien nimien on oltava kyselyn suorittajan mielessä. Esimerkiksi jos polkupyöräkaupassa halutaan tietää, minkä värisiä runkoja polkupyörille on saatavilla, on luonnollista, että attribuutti, josta ollaan kiinnostuneita, on nimeltään väri. Joskus on kuitenkin hyödyllistä viitata olion ominaisuuteen vain kertomalla kyselykielellä, että ollaan kiinnostuneita olion ominaisuudesta, jonka arvo on punainen. Toisaalta voimme olla kiinnostuneita saaman selville kaikki sellaiset oliot, joiden omien tai osien ominaisuuksien arvona on asbesti. Tällöin ei siis tarvitse tietää, onko attribuutti mallinnettu nimellä materiaali, aine vai jokin muu. Tämä seikka tuo uudenlaisen tavan ominaisuuksien tarkasteluun.

Käyttäjäystävällisyyttä saadaan lisättyä myös sillä, että kyselyitä tekevän loppukäyttäjän ei tarvitse tietää, onko ominaisuus johdettu vai ei. Tämä onnistuu naamioimalla metodit käyttäjältä attribuuttien kaltaisiksi. Käyttäjä voi siis käsitellä

kaikkia olion ominaisuuksia attribuutteina välittämättä siitä, muodostuvatko ne oikeasti suoraa vai johdetusti.

Edellä esitetyt ilmaisuvoimaiset näkökohdat tulee voida siirtää käyttäjälle helppokäyttöisellä kyselykielellä. Kyselykielen perusvaatimus on se, että kielen käyttäjän ei tarvitse osata ohjelmoida. Käyttäjän tulee siis voida suorittaa kyselyjä hallitsematta iteratiivista tai rekursiivista ajattelutapaa. Häneltä ei voida myöskään vaatia tietämystä deduktiivisille tietokannoille luonteenomaisesta mallinsovituksesta. Edelleenkään hänen ei tarvitse osata käyttää funktioita, eikä ajatella funktionaalisen ohjelmointikielen tavoin. Yleisesti ottaen käyttäjän täytyy kyetä suorittamaan kyselyt kuvailevalla, eli deklaratiivisella tavalla, eli kertomalla kyselykielen avulla *mitä* halutaan, eikä *miten* lopputulos saadaan aikaiseksi.

Tämän tutkielman ehdotus helpon, mutta ilmaisuvoimaisen kyselykielen syntaksille on seuraava: Käyttäjän tarvitsee ymmärtää ainoastaan jaetun muuttujan idea, kielen perusprimitiivit sekä konjunktio ja disjunktio. Jaetun muuttujan ideaa voidaan demonstroida seuraavalla esimerkkillä: ”X is_composite_type_of wheel, Y is_property_of X”. Isolla kirjoitetut kirjaimet merkitsevät muuttujia. Täten X viittaa kyselyn molemmissa, pilkulla erotetussa kohdassa samaan asiaan, joka voi olla olio, tyyppi tai tyypin ominaisuus. Perusprimitiiveinä kieleen on määritelty luonnollista kieltä mukailevia ilmaisuja, jotka ilmaisevat primitiivin käyttöön liittyvän intuition. Tällaisia perusprimitiivejä ovat esimerkiksi edellä esitetty is_composite_type_of -primitiivi, joka tarkoittaa, että operaattorin vasemmalle puolelle laitetaan se tyyppi, joka on operaattorin oikealle puolelle laitettavan tyypin kompositiotyyppinä. Primitiivit voivat olla prefix-, postfix- tai infix-operaattoreita. Operandit voivat olla muuttujia tai atomisia. Edellinen esitettään siis isolla kirjaimella alkavalla merkkijonolla ja jälkimmäinen pienellä kirjaimella alkavalla merkkijonolla tai numerolla. Jos kummallekin puolelle tulee muuttuja, vastaukseksi tulisi saada kaikki primitiivin toteuttavat operandien yhdistelmät. Konjunktio ja disjunktio ovat niin intuitiivisia loogisia käsitteitä, että käyttäjän ei tarvitse opetella logiikkaa osatakseen niitä. Konjunktiota merkitään pilkulla ja disjunktion tapauksessa käytetään puolipistettä.

## 5. Esitystavasta

Jotta käyttäjäystävällisen ja ilmaisuvoimaisen kyselykielen edellytykset voitaisiin toteuttaa, pitää kehittää uudenlainen tiedon esitystapa, joka poikkeaa olemasta olevista keinoista. Intensionaalisten ja ekstensionaalisten tietojen käsittelyn yhdistäminen edellyttää, että molempien tasojen tieto on eksplisiittisesti esitettynä. Tämän lisäksi tarvitaan mekanismi, joka liittää eksplisiittisesti esitetyt tasot toisiinsa. Tasojen liittämisen myötä saavutetaan ilmaisuvoimaisten, analyyttistä voimaa

sisältävien tarkastelujen mahdollisuus. Jotta edellä esitetty korkea päättelyvoima saadaan aikaiseksi esimerkiksi tilanteessa, joissa operandeiksi voi primitiivin molemmille puolille tulla muuttujat, kielen toteutuksen pohjalla on hyvä olla deduktiivinen systeemi. Lisäksi pohalle tarvitaan olio-orientoituneisuutta tukeva systeemi. Logiikka-ohjelmointiparadigma tarjoaa deduktiivisen lähestymistavan. Tämän takia kielen toteutusvälineeksi on valittu prolog++, joka yhdistää logiikkaohjelmointiparadigman ja oliolähestymistavan.

**Prolog++ lyhyesti**

Prolog++ on hybridikieli, joka yhdistää Prologin päättelykyvyn ja oliomallin tiedon mallintamisvoiman. Seuraavaksi esitetään lyhyesti LPA:n kehittämän Prolog++:n yleisiä piirteitä [Moss, 1994]. Prolog oletetaan tunnetuksi.

Kieli on rakennettu normaalin Prologin päälle lisäämällä siihen olio-ohjelmoinnin piirteitä. Luokassa määritellään attribuutit ja metodit. Attribuutit ovat luonteeltaan samanlaisia muuttujia kuin Prologissa yleensä, eli niihin voidaan soveltaa samaistusta. Metodit ovat puolestaan normaaleja Prolog–predikaatteja. Jos luokassa on komponentteina muita olioita, ne määritellään sanalla parts osa1, osa2,…,osaN, jossa osat 1-N ovat ohjelmassa määriteltyjä luokkia. Kun luokan ilmentymä luodaan, sille luodaan automaattisesti myös sen osat. Tämän ominaisuuden johdosta kielellä voidaan tehdä vain poissulkevia olioita, vaikka myöhemmin esitettävä intensionaalisen ja ekstensionaalisen tiedon esitystapa ei tällaista rajoitusta astekaan. Esimerkkinä luokan määritelmä kolmipyörän tilanteessa:

```
class tricycle.
  public instance attribute price.
  parts frame,saddle,steering,rear.
      weigth(X) :- frame#1<-weigth(X1),saddle#1<-weigth(X2),steering#1<-
  weigth(X3),rear#1<-weigth(X4),X is X1+X2+X3+X4.
end tricycle.
```

Luokan määritys aloitetaan class –sanalla, jota seuraa luokan nimi. Määrittely päätetään end -sanaan, jota seuraa uudelleen luokan nimi. Luokalla on osina frame, saddle, steering ja rear. Attribuuttina on price. Weight on luokan metodi, joka kysyy osiensa painoa ja laskee ne yhteen. Olio esitetään kielessä tyyliin (tricycle|543251), missä tricycle on luokan nimi ja 543251 on luokan identifioiva tunnus. Oliolle voidaan lähettää viesti, jos halutaan tietää sen jonkin attribuutin tai metodin arvo. Hinta kysytään yllä olevan esimerkin tapauksessa tyyliin (tricycle|543251) <-price(X), jolloin X arvottuu olion tilan mukaisella arvolla. Metodin antamaa arvoa kysytään samalla tavalla kuin attribuutin tilanteessa, eli muodossa (tricycle|543251)

<- price(X). Metodi poikkeaa siis useimmista olio-orientoituneista kielistä siinä, että operaatiolla ei ole tarvetta palauttaa arvoa. Suluissa olevaa muuttujaa käytetään sekä lähettämään että vastaanottamaan metodilta arvo samaistuksen avulla [Moss, 1994].

**Intesionaalisen ja ekstensionaalisen tason esitys**

Intensionaalisen ja ekstensionaalisen tiedon esittämiseen sovelletaan Timo Niemen [1983] kehittämää arvo-orientoitunutta seven-tuple –esitystä. Se integroi intensionaalisen ja ekstensionaalisen tason toisiinsa tähän tarkoitukseen kehitetyllä indeksointimenetelmällä. Esitystapaa on edelleen kehitetty Niemen, Kalervo Järvelinin ja Marko Junkkarin [2001] toimesta. Junkkari ehdottaa, että mikä tahansa osa-kokonaisuussuhde esitetään part-of -rakenne-elementtinä (part-of structure element, lyhyesti PSE) [Junkkari 2001], joka sisältää kompleksisen objektin ekstensionaalisen ja intensionaalisen tason tiedot. Ekstensionaalinen taso koostuu attribuuttien arvoista ja olioiden identiteeteistä. PSE esitetään parina (Ext,N), jossa Ext on ekstensionaalinen taso ja N on intensionaalinen taso. Intensionaalinen taso esitetään binäärirelaationa, joka sisältää järjestettyjä pareja. Parin ensimmäisenä elementtinä on joko olion tyypin tai attribuutin nimi ja toisena elementtinä indeksi, joka liittyy kyseiseen nimeen. Ekstensionaalinen taso on järjestetty siten, että jokaiselle intensionaalisella tasolla kuvattuun indeksiin liittyy yksikäsitteinen arvo ekstensionaalisella tasolla. Voimme siis ekstensionaalisella tasolla etsiä tiettyyn arvoon liittyvän indeksin rakenteen perusteella ja katsoa, mikä tyypin tai attribuutin nimi sitä vastaa intensionaalisella tasolla. Indeksointimekanismi toimii samalla tavalla myös toiseen suuntaan, eli voimme katsoa, mihin arvoon/arvoihin tai olion identiteettiin/identiteetteihin tietty intensionaalisella tasolla oleva tyyppinimi liittyy. Lisäksi indeksejä analysoimalla pystymme selvittämään monenlaista informaatiota olioiden/oliotyyppien keskinäisistä suhteista [Junkkari 2001].

Esitystapa voidaan siirtää formalismin tasolta käytäntöön edellä esitetyllä Prolog++ -kielellä. Jokaista PSE:tä vastaa yksi intensionaalisen tason Prolog-termi. Ekstensionaalisen tason Prolog-termejä on niin monta, kuin PSE:n ilmentymiä on luotu PSE-tietokantaan. Jokainen PSE:n ilmentymä on yksikäsitteinen.

**Intensionaalisen tason Prolog–esitys**

Yhtä PSE:n intensionaalisen tason kuvausta vastaa binäärirelaatio, joka koostuu sekä osa-kokonaisuussuhteessa olevista oliotyypeistä ja niiden attribuuteista, että niihin liittyvistä indekseistä. Alla esimerkkinä intensionaalisen tason PSE-esitys kolmipyörästä Prolog-terminä.

```
pse([map(tricycle,t(1)),map(price,t(1,1)),map(weigth,t(1,2)),map(frame,t(1,3)),map(saddle,t(1,4)),map(steering,t(1
,5)),map(rear,t(1,6)),map(frame_no,t(1,3,1)),map(material,t(1,3,2)),map(weigth,t(1,3,3)),map(pad,t(1,4,1)),map(w
eigth,t(1,4,2)),map(h,t(1,5,1)),map(weigth,t(1,5,2)),map(front_axle,t(1,5,3)),map(handlebars,t(1,5,4)),map(pedals,t
```

(1,5,5)),map(wheel,t(1,5,6)),map(b,t(1,6,1)),map(weigth,t(1,6,2)),map(rear_axle,t(1,6,3)),map(wheel,t(1,6,4)),map(l,t(1,5,3,1)),map(weigth,t(1,5,3,2)),map(b,t(1,5,4,1)),map(weigth,t(1,5,4,2)),map(diam,t(1,5,5,1)),map(weigth,t(1,5,5,2)),map(diam,t(1,5,6,1)),map(r_type,t(1,5,6,2)),map(weigth,t(1,5,6,3)),map(b,t(1,6,1)),map(weigth,t(1,6,2)),map(diam,t(1,6,3,1)),map(l,t(1,6,3,2)),map(weigth,t(1,6,3,3)),map(wheel,t(1,6,4)),map(diam,t(1,6,4,1)),map(r_type,t(1,6,4,2)),map(weigth,t(1,6,4,3))]).

Termin nimi on pse. Kaikki termin jäsenet ovat map(X,Y) –muotoisia, joissa X on attribuutin tai tyypin nimi ja Y on indeksin kertova predikaatti. Tällaista rakennetta kutsutaan map-konstruktoriksi [Niemi and Järvelin 1991] Yllä oleva termi kuvaa kolmipyörän, joka on hierarkiassa ylin kompositio, ja liittää siihen indeksin t(1). Kaikki komposition sisällä olevat attribuutit ja oliot kuvataan indeksille, joka on muotoa t(1,Z), joka puolestaan esitetään tuple-konstruktorina. Z tarkoittaa yllä attribuutin tai olion järjestysnumeroa. Koska hinta on kolmipyörän attribuutti, se on kuvattu indeksille t(1,2). Steering on puolestaan kolmipyörään sisältyvä oliotyyppi, joka on mallinnettu indeksille t(1,5). Steering:in kaikki attribuutit ja välittömät oliot kuvataan indekseille muotoa t(1,5,R), missä R tarkoittaa vähintään yhden numeron pituista merkkijonoa. Kuten esimerkistä huomaa, binaarirelaation tyyppien tai attribuuttien nimien ei tarvitse olla yksikäsitteisiä. Sama nimi voi siis olla useammalla indeksillä. Indeksit sitä vastoin ovat yksikäsitteisiä.

**Ekstensionaalisen tason Prolog–esitys**

Ekstensionaalinen taso muodostuu olioista ja niitä vastaavista PSE-esityksen ekstensionaalista tasoa vastaavista Prolog-termeistä. Termi muodostetaan vain ylimmän tason kompositiosta, jolloin termi pitää sisällään kaikki sen komponenttiolioidenkin tiedot. Komponentteina olevat oliot esitetään Prolog-termeissä samalla tavalla kuin ylimmän tason oliot, joten olion esitystapa on aina samanlainen riippumatta siitä, puhutaanko ylimmän tason esityksestä vai komponentin esityksestä. Yksi termi kuvaa yhden PSE:n ilmentymän. Termin nimi on yksikäsitteinen, mutta sisältää aina aluksi merkin ’t’, joka merkitsee tuple-konstruktoria [Niemi and Järvelin, 1991]. Seuraavaksi tulee merkkijono, joka kertoo, minkä luokan ilmentymä pse:n olio on, sillä kaikki pset ovat olioita. Aina, kun luodaan uusi PSE –olio, se tulostaa itsestään termin, joka kuvaa olion tilan. Tulostus liitetään eli assertoidaan prolog–koodiin, jolloin se on seuraavalla ajokerralla terminä muiden termien joukossa. Tulostus tapahtuu seuraavalla tavalla: Olio lähettää tulostuspyynnön rekursiivisesti kaikille komponenteilleen, jotka siis ovat myös olioita. Kaikki oliot tulostavat Prolog-termin, jonka nimi muodostuu ’t’–merkistä ja olion luokan nimestä sekä olion identifioivasta numerosta. Numero on Prolog++:n generoima uniikki numeroarvo. Nimen jälkeen tulee termin sisältö suluissa, joka koostuu olion attribuuttien arvoista. Sitä seuraa mahdolliset komponentit, jotka on

esitetty hakasulkujen sisällä. Hakasulut tarkoittavat joukkokonstruktoria [Niemi and Järvelin, 1991], joka muodostuu komponenteista. Aivan kuten joukko-opissa yleensäkin, joukon alkioiden, eli tässä tapauksessa komponenttien, järjestyksellä ei ole väliä. Yksikäsitteinen indeksi osoittaa yhteen joukkokonstruktoriin, eli samalle indeksille voi kuvautua useampia olioita. Jos useampia olioita on hakasulkurakenteessa, tämä tarkoittaa tilannetta, jossa kompleksisella objektilla on rakenne, jossa on useampia erillisiä, mutta semantiikaltaan samanlaisen funktionaalisuuden tai sijainnin omaavaa osaa. Esimerkkinä tällaisesta tilanteesta on kolmipyörä, jolla on kaksi takarengasta. Tällöin voi olla paikallaan mallintaa ne samanarvoisiksi hierarkiassa, jos niiden järjestyksellä ei ole väliä. Mikään ei estä keksimästä muitakin semantiikkoja, jossa tätä ominaisuutta voidaan hyödyntää. Jos olio on perusolio, se tulostaa vain itsensä, ja attribuuttinsa, eikä sillä tällöin ole yhtään hakasuluissa olevaa komponenttia.

Yksittäisen ilmentymän esitys Prolog -terminä voi olla esimerkiksi seuraavanlainen kolmipyörän tapauksessa:

```
ttricycle53607(104,25,[tframe53617(124353,wood,4)],[tsaddle53627(leather,1)],[tsteering53637(20,17,[tfront_a
xle53647(5,14)],[thandlebars53657(7,1)],[tpedals53667(5,1)],[twheel53677(8,united,1)])],[trear53687(13,3,[tre
ar_axle53697(1,12,1)],[twheel53707(6,united,1),twheel53717(6,united,1)])])
```

Ilmentymän esitys on tässä tapauksessa kuusipaikkainen Prolog–termi, jonka nimi on ttricycle53607. Vasemmalta oikealle mentäessä kaksi ensimmäistä predikaatin, eli olion jäsentä, ’104’ ja ’25’, ovat attribuuttien arvoja ja loput neljä listoja. Listat ovat:

```
[tframe53617(124353,wood,4)]

[tsaddle53627(leather,1)]

[tsteering53637(20,17,[tfront_axle53647(5,14)],[thandlebars53657(7,1)],[tpedals53667(5,1)],[twheel53677(8,un
ited,1)])]

[trear53687(13,3,[trear_axle53697(1,12,1)],[twheel53707(6,united,1),twheel53717(6,united,1)])]
```

Prolog-listat ovat komponentteja. Listat sisältävät alkioinaan olioita, joilla on samat indeksit keskenään. Pse:n ilmentymän indeksiä t(1) vastaa koko rakenne. Indeksiä t(1,1) vastaa arvo 104 ja indeksiä t(1,3) vastaa tuple-rakenne tframe53617(124353,wood,4), joka on siis olio, joka kuvaa pyörän rungon. Kuten huomataan, rakenne on samankaltainen kuin ylimmälläkin tasolla. Tällä kertaa kyseessä on vain perusolio, jolla ei siis ole hakasulkuja komponentteja kuvaamaan. Rakenteessa oleva arvo ’wood’ on nyt koko pse:n ilmentymän kannalta indeksillä

t(1,3,2). Toiseksi alin ja alin lista sisältävät olion, joka ei ole perusolio, sillä siinä on listoja eli komponentteja.

Termiesityksestä voidaan konstruoida jälleen olio. Esimerkiksi tframe53617(124353,wood,4) kuvaa frame–oliota. Prolog-predikaatilla se voidaan ”herättää henkiin” siten, että poistetaan ensimmäinen t-kirjain predikaatin edestä ja erotellaan kirjainosuus numero-osuudesta eri muuttujiin, esimerkiksi X ja Y. Tämän jälkeen olion attribuutteihin ja metodeihin pääsee käsiksi esitystavalla (X|Y), joka on siis olion viite.

Miksi ekstensionaalisen esityksen termeissä on esitettynä myös attribuuttien arvot, kun niihin pääsee käsiksi edellä esitetyllä tavalla? Tämä ominaisuus on ekstensionaalis-ekstensionaalisia kyselyjä varten. Kuten esimerkiksi etsimään vastausta kysymykseen, minkä olion/olioiden tai tyypin/tyyppien attribuutin arvona on vaikkapa 53. Kysymykseen voidaan vastata, jos tieto esitetään tällä tavoin.

## 6. Indekseihin perustuva analysointi

Edellä on esitelty ekstensionaalinen ja intensionaalinen taso tavalla, joka mahdollistaa niiden välisen integroinnin. Nyt pystytään hyppäämään tasolta toiselle, koska indeksit pitävät tasot tiiviisti yhdessä. Indeksit ovat voimakas väline suhteiden analysointiin [Niemi, 1983; Junkkari, 2001]. Junkkari [2001] on esittänyt formaalisti monia indekseihin perustuvia analysointimahdollisuuksia, joista seuraavaksi esitetään muutama esimerkki Prologin näkökulmasta. Indeksiesitys voidaan muuttaa Prologissa listamuotoon, joita Prologilla on helppo analysoida monien listojen analysointiin tarkoitettujen predikaattien ansiosta. Esimerkiksi predikaatilla prefix(X,Y), voidaan tarkistaa, onko ensimmäinen lista X listan Y alussa [Sterling and Shapido, 1986]. Esimerkiksi kaikki tyyppiin liittyvät indeksit ovat sellaisia pse:ssä olevia indeksejä, joille löytyy indeksi, jonka alku on sama, mutta löydetty indeksi on yhden numeron pidempi. Tämä johtuu siitä, että tyypillä tai oliolla on aina vähintään yksi attribuutti [Junkkari, 2001], jolloin attribuuttiin liittyvä indeksi on alusta sama, mutta pidempi. Attribuutteihin liittyvät indeksit ovat puolestaan sellaisia, joihin liittyy indeksi, jolle ei löydy indeksiä, jolla olisi samanlaista alkua, mutta olisi pidempi. Tämä johtuu siitä, että attribuuteilla ei voi olla attribuutteja tai komponentteja [Junkkari, 2001]. Komposition indeksi on puolestaan muodoltaan sama alusta, mutta lyhempi. Esimerkiksi indeksin t(1,3,4,7) kompositioiden indeksit ovat t(1,3,4), t(1,3), t(1). Tämä on siis täsmälleen sama lopputulos, johon päädymme jos ajamme prefix – predikaatin seuraavalla tavalla prefix(X, [1,3,4,7]). Tyypin tai olion komponenttien indeksejä ovat puolestaan sellaiset indeksit, jotka ovat tyyppi-indeksejä ja joiden alku on sama kuin kyseisen tyypin tai olion indeksi ja ne ovat lisäksi pidempiä.

Tässä oli vain muutama esimerkki siitä, miten indeksejä analysoimalla voi saada osa-kokonaisuussuhteista monipuolista tietoa. Muita indekseistä selville saatavia tietoja ovat esimerkiksi se, kuvaako indeksi perusoliota, välitöntä tai välillistä komponenttia/komposiittia. Joka kohdassa pystytään analysointi tekemään kumpaan tasoon tahansa. Voidaan esimerkiksi kysyä, mikä on renkaan komposiittioliot, sillä kun saamme renkaaseen liittyvät indeksit, voimme kysyä ekstensionaaliselta tasolta tällä indeksillä olevia arvoja. Tässä tapauksessa arvot kuvaisivat olioita, jotka pystytään muuttamaan edellä kerrotulla tavalla sellaiseen muotoon, että niiden avulla pystytään edelleen kutsumaan olion metodeja.

## 7. Kielen primitiivit

Edellä olevan esitystavan toteutuksen pohjalta on mahdollista muodostaa kyselykieli, joka täyttää tutkielmassa esitetyt vaatimukset. Lopuksi esitetään primitiivejä, joiden toteuttaminen on nyt mahdollista. Primitiivejä voi siis yhdistää toisiinsa konjunktiolla ja disjunktiolla jaettujen muuttujien avulla. Seuraavassa luetellaan kyselykielen 14 primitiiviä:

**1. is_composite_type_of**

**2. is_top_type_of**

**3. is_subcomponent_type_of**

Primitiivit 1-3 ovat infix-operaattoreita. Ne siis saavat molemmille puolilleen operandin. Operandi voi olla muuttuja tai vakio. Ensimmäisen primitiivin vasemmalle puolelle tulee kompositio ja oikealle puolelle komponentti. Toinen primitiivi kertoo, mikä on oikealle puolelle asetettavan tyypin ylin tyyppi, joka ei siis ole minkään komponenttina. Kolmas primitiivi on tarkoitettu samaan selville alikomponentit halutulta tyypiltä.

**4. is_basic_type**

Primitiivi on prefix-operaattori, joka antaa kaikki perustyypit.

**5. is_property_of**

Antaa vastaukseksi tyypin ominaisuudet.

**6. is_instance_of**

**7. is_type_of**

Kuudes primitiivi on intensionaalisen ja ekstensionaalisen tason yhdistäviä kyselyjä varten. Siinä vasemmalle puolelle tulee oikean puolista tyyppiä oleva olio. Seitsemäs on kuudennen käänteisprimitiivi.

**8. is_composite_object_of**

**9. is_subcomponent_object_of**

Kahdeksas primitiivi yhdistää myös intensionaalisen ja ekstensionaalisen tason. Vasemmalle puolelle tulee olio ja oikealle tyyppi. Se siis antaa tyyppiä vastaavat kompositio-oliot. Yhdeksäs operaatio on käänteinen kahdeksannelle.

**10. apply_to**

Kymmenes operaatio määrää, mille pse:ille kyselyjä suoritetaan. Tämä on tärkeää, sillä tulokset ovat erilaisia, jos kysytään esimerkiksi, mikä on jonkin tyypin kompositiona. Jos samoja tyyppejä on eri pse:issä, vastaus voi olla erilainen, jos kyselyä sovelletaan kaikkiin pse:ihin, verrattuna tilanteeseen, jossa kyselyä sovelletaan vain yhteen pse:hen. Oletuksena kyselyitä sovelletaan kaikkiin pse:ihin.

**11. common_component**

**12. common_components**

Yhdestoista primitiivi antaa sillä hetkellä tarkastelun kohteena olevien pse:iden ne komponenttityypit, jotka ovat yhteisiä niille kaikille. Kahdestoista primitiivi on samanlainen kuin edeltäjänsä, paitsi että se antaa vastauksen listaesityksenä.

**13. is_path_to**

13. primitiiville annetaan vasemmalle puolelle parametriksi muuttuja, joka arvottuu tuple-muotoisella polkuesityksellä sitä, mikä polku johtaa primitiivin oikealle puolelle annettavaan tyyppiin. Polkuja voi olla useita, joten kaikki vaihtoehdot esitetään. Kuten muissakin kyselyissä, apply_to –primitiivi määrää, mihin pse:ihin kyselyä sovelletaan. Kuvitellaan tilanne, jossa polkupyörätehtaalla halutaan analysoida, mitkä polut johtavat polkupyörien renkaisiin kolmipyörissä ja normaaleissa polkupyörissä. Tällöin voidaan soveltaa kyselyä ”apply_to [tricycle,bicycle], X is_path_to wheel” Tulokseksi voitaisiin saada esimerkiksi seuraavat polut t(tricycle,steering,wheel), t(tricycle,rear,wheel), t(bicycle,wheel) ja t(bicycle,steering,wheel).

**14. :**

Tällä primitiivillä voidaan viitata olion ominaisuuteen. Esimerkiksi kysely Y:diam(X) asettaa X:n arvoksi Y-muuttujassa olevan olion diam–nimisen ominaisuuden arvon. Jotta kysely onnistuisi, täytyy Y:n olla olio, jolla on diam–niminen attribuutti tai yksipaikkainen metodi. Edellinen esitystapa, Y:diam(X), toimii

varsinaisessa kyselyssä. Prolog++:n ansiosta metodi voidaan siis kapseloida tällä tavalla käyttäjältä normaaliksi attribuutiksi.

## 8. Keskustelua

Tutkielmassa on todettu osa-kokonaisuussuhteella olevan semanttinen luonne, joka eroaa is-a –mallintamishierarkiasta. Todettiin, että tarvetta helppokäyttöiselle, mutta ilmaisuvoimaiselle ja osa-kokonaisuussuhteen erityispiirteet huomioivalle kyselykielelle. Tutkielmassa esitetyn esittämistavan pohjalta pystytään luomaan kyselykieli, joka poistaa nämä epäkohdat. Esitystavan ideana on intensionaalisen ja ekstensionaalisen tiedon eksplisiittiseen esittämiseen ja niiden sitomiseen toisiinsa indeksien avulla sekä indeksointimenetelmä, jonka avulla voidaan analysoida osa-kokonaisuushierarkioita kummalla tasolla tahansa. Esitystavan ansiosta kaaviotason rakenteesta ja rakenteen komponenttien ominaisuuksista voidaan saada tietoa. Pystytään toteuttamaan siis kieli, jossa voidaan myös yhdistää intensionaalisen ja ekstensionaalisen tason kyselyjä. Tämän esitystavan käyttöönotto onnistuu esimerkiksi Prolog++ -ohjelmointikielellä, jossa yhdistettään logiikkaohjelmoinnin päättelykyky ja olio-ohjelmoinnin esitysvoima.

## Viiteluettelo


[Cattell *et al, 2000*] R.G.G Cattell, Douglas Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda and Fernando Velez. *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann, 2000.

[Deux *et al*, 1990] O. Deux and workgroup, The story of $O_2$. *IEEE Transactions on Knowledge and Data Engineering*, **2,** 1, (1990).

[Halper *et al*, 1998] Michael Halper, James Geller, Yehosua Perl. An OODB Part-Whole model: Semantics, notation and implementation. *Data & Knowledge Engineering*. **27**, (1998), 59-95.

[Junkkari, 2001] Marko Junkkari, The systematic object-oriented representation for managing intensional and extensional aspects in modeling of part-of relationships. University of Tampere, Dept. of Computer and Information Sciences, Report **A-2001-5,** June 2001.

[Junkkari, 2001b] Marko Junkkari, Set theoretical specification for presenting relationships among components. Draft, November, 2001.

[Järvelin and Niemi, 1999] Kalervo Järvelin and Timo Niemi, Integration of complex objects and transitive relationships for information retrieval. *Information processing and management*. **35,** 5, (1999), 655-678.

[Lecluse *et al*, 1988] Chritophe Lecluse, Philippe Richard and Fernando Velez, O2, an object-oriented data model. *ACM SIGMOD International Conference on Managing of Data* **17**, 5,*(Sep. 1988),* 424-433.

[Niemi, 1983] Timo Niemi, A seven-tuple representation for hierarchical data structures. In: *Information Systems*. **8**, 3, (1983) 151-157.

[Niemi and Järvelin, 1991] Timo Niemi and Kalervo Järvelin, Prolog-based meta-rules for relational database representation and manipulation. *IEEE Transactions on Software Engineering*. **17**,8 (1991), 762-788.

[Moss, 1994] Chris Moss, *Prolog++ The Power of Object-Oriented and Logic Programming*. Addison-Wesley, 1994.

[Motschnig-Pitrik and Kaasboll 1999] Renate Motschnig-Pitrik and Jens Kaasboll, Part-whole relationship categories and their application in object-orinented analysis. In: *IEEE Transactions on Knowledge and Data Engineering*, **11,** 5, (1999) 779-796.

[Paton *et al]* Norman Paton, Richard Cooper, Howard Williams and Philip Trinder, *Database Programming Languages.* Prentice Hall, *1996.*

[Roth *et al,* 1988] Mark A. Roth, Henry F. Korth, and Abraham Silberschafz, Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*. **13**, 4 (1988) 389-417.

[Savnik *et al*, 1999] Iztok Savnik, Zahir Tari, Tomaz Mohoric. QAL: A query algebra of complex objects. *Data & Knowledge Engineering*, **30**, (1999) 57-94.

[Sterling and Shapido, 1986] Leon Sterling and Ehud Shapido, *The Art of Prolog*, The MIT Press, 1986.

# WWW-julkaisu XSLT-tyylisivujen avulla

## Johanna Virtanen

**Tiivistelmä.**

Tutkimuksessani tarkastellen XSLT-tyylisivuja ja erityisesti niiden soveltuvuutta www-julkaisuun. Luon lyhyen katsauksen XSLT:n kielioppiin ja esittelen piirteitä, jotka ovat olennaisia erityisesti html-muunnosta tehtäessä. Tarkastelukriteereinä on käytetty XSLT:n kieliopin mukautuvuutta, käytön helppoutta ja XSLT-tyylisivujen yhteensopivuutta eri selainten kanssa. Esille otetaan myös muutamia sovellusalueita ja käyttömahdollisuuksia. Tutkimuksen aikana käy ilmi, että XSLT on monipuolinen, monikäyttöinen ja mukautuva kieli ja se soveltuu erityisesti suurten tietokantojen julkaisuun WWW:ssä. Sen tehokas käyttö vaatii kuitenkin laatijaltaan paljon, kun vielä siihen sopivat työkalut ovat keskeneräisiä. Näinpä se käyttö pienimmissä sovelluksissa ei ehkä toistaiseksi ole kovin tarkoituksen mukaista.

Avainsanat ja -sanonnat: HTML, XML, XSLT.
CR-luokat: D3.2

## 1. Johdanto

WWW:n jatkuvasti kasvava tietomäärä on luonut tarpeen erialisille työkaluille, joilla tätä tietomäärää ja sen esitystapaa voidaan tehokkaasti hallita. Vanhat Web-kielet, kuten HTML ja tyylisivujen kuvauskieli CSS, ovat osoittanut ilmaisuvoimaltaan liian heikoksi, ja niinpä uusia ratkaisuja on vähitellen alettu kehittämään niiden tilalle. Erityisesti XML on viime aikoina kasvattanut vahvasti suosiotaan dokumentin hallinta- ja kyselykielien joukossa. XML:ään kuuluu myös sellainen tyylisivujen rakennuskieli, kuin XSLT, johon tämä tutkimus keskittyy. Tarkoituksena on tarkastella XSLT:n sopivuutta nimenomaan www-julkaisuun. Koska XSLT pohjautuu XML:ään, myös sen tiettyjä piirteitä nousee väistämättä esiin. Tutkimukseni perustuukin sille olettamukselle, että lukija on tutustunut XML:n perusteisiin Samuel Holznerin teoksen *Inside XML* [Holzner2000] kattamien asioiden osalta. XML:stä löytyy laajalti tietoa myöskin internetistä. [xml2001]

Aluksi luodaan lyhyehkö katsaus XSLT:n kehitykseen, sen alkuperään sekä sitä luonnehtiviin piirteisiin. Seuraavaksi annetaan lyhyt esitys XSLT:n peruskielioppia ja sitä kautta tarkastellaan myös sen syntaksin sopivuutta verkkojulkaisuun. Tämän lisäksi on luonnollista vertailla kielen yhteensopivuutta eri selainten kanssa

On syytä muistaa, että XSLT on vielä pitkälti tulevaisuuden työkalu, ja sen kehitys on vasta alkuvaiheessa. Tällä hetkellä kielen käyttäminen vaatii vielä käyttäjältään kohtalaisesti taitoa ja viitseliäisyyttä. Työkaluja kielen muotoilemiseen on tarjolla vähän, ja niiden suorituskyky on suhteellisen alhainen. Kaikkiin pieniin sovelluksiin XSLT:n käyttö siksi tuskin vielä on perusteltua, mutta on selvää, että suurempia kokonaisuuksia, varsinkin tietokantasovelluksia, hallittaessa sen käyttöä kannattaa todella harkita. Lisäksi XSLT:n käyttö tulee helpottumaan tulevaisuudessa. Kieltä kehitetään jatkuvasti ja sen generointiin tarkoitettuja prosessoreita ja niiden ominaisuuksia parannellaan.

Tutkielmassa käytetään seuraavia lyhenteitä:

**CSS** Cascading StyleSheets

**DTD** Document Type Definition

Tarkoitettu dokumenttien rakenteiden kuvaamiseen, määritellään elementit sekä attribuuttien nimet sekä elementtien keskinäinen tärkeysjärjestys.

**HTML** Hyper Text Markup Language

**SGML** Standard Generalized Markup Language

**XML** eXtensive Markup Language

**XPATH** Työkalu, jonka avulla voidaan osoittaa XML-dokumentin eri osia.

**XSL** eXtensible Stylesheet Language

Standardi, jonka mukaan XML-dokumentit voidaan muuttaa helppolukuiseksi tai tulostuskelpoiseksi. Se jakaantuu kahteen osaan, XSL Transformationsiin (XSLT) ja XSL Formatting Objectsiin (XSL-FO). Standardin molemmat osat pohjautuvat syntaksiltaan XML:ään.

**XSLT** eXtensible Stylesheet Language Transformations

Kieli, jonka avulla XML-dokumentit voidaan automaattisesti muokata toiseksi rakenteiseksi dokumentiksi, esimerkiksi HTML:ksi, tekstiksi tai XML-dokumentiksi, jolla on erilainen kielioppi kuin alkuperäisellä.

**XSL-FO** XSL Formatting Objects

XSL-FO:n avulla XML-dokumentista laaditaan esitys- tai tulostuskelpoinen dokumentti, esimerkiksi PDF-tiedosto.

## 2. XSLT:n esittelyä

XML on sisällönkuvauskieli, jonka avulla tieto voidaan kuvata laite- ja ohjelmariippumattomasti. XML:n merkityksestä ja käytön helppoudesta on useasti korostettu. Peruskäyttäjälle ei kuitenkaan ole iloa pelkistä XML-dokumenteista, joita ei voi katsella helposti ja vaivattomasti selaimella. Tässä XSL astuu mukaan kuvaan. XSL (Extensible Style Language) on World Wide Web Consortiumin suosittelema standardi, jonka mukaan XML-dokumentit voidaan muuttaa helppolukuiseksi tai

tulostuskelpoiseksi. XSL on XML-pohjainen kieli. Se muodostuu kahdesta osasta: XSL Formatting Object -määrittelystä ja XSLT:stä. XSL:n avulla ilmaistaan varsinaiset muotoilut ja XSLT:n avulla on tarkoitus tehdä XML:n muunnos rakennepuusta julkaisupuuksi.

XSLT standardi valmistui loppuvuodesta 1999, ja sen ensimmäinen julkinen luonnos oli julkaistu noin vuotta aiemmin. Standardin takana on pitkälti ollut James Clark. [w3c2000] XSLT käyttää XML-puun haarojen valitsemiseen XPath-kieltä, josta tuli W3C:n standardi yhtä aikaa XSLT:n kanssa. XSL-työryhmässä, joka määritteli myös XSLT:n on jäseniä mm. sellaisista yrityksistä kuin IBM, Adobe, Sun ja Xerox.

XSLT:llä voidaan tehdä käytännössä millainen tahansa muunnos XML-rakenteesta toiseen. XSLTn avulla voidaan muuttaa sekä automaattisesti tuottaa dokumenttien rakenteita. Esimerkiksi jollakin omalla DTD:llä määritelty teksti voidaan muuntaa XSLT:n avulla HTML-muotoon. Huomattavaa on, ettei tuotettavan dokumentin tarvitse muistuttaa lainkaan alkuperäistä dokumenttia, elementtien esiintymisjärjestys, nimeäminen ja attribuutit ovat kaikki mahdollista määritellä uusiksi. Vanhoja arvoja voidaan tarvittaessa käyttää, mutta uusilla nimillä. Kaiken kaikkiaan XSLT on hyvin tehokas tapa käsitellä dokumentteja. Kaiken kaikkiaan se on varsin monipuolinen, mutta samalla myös monimutkainen kokonaisuus.

XSLT-dokumentti, jota kutsutaan yleisesti tyylisivuksi, on syntaktisesti XML:ää ja koostuu joukosta sääntöjä, jotka ohjaavat XML-dokumentin muunnosta. Itse muunnoksen suorittaa erityinen XSLT-prosessori, joka lukee sisään sekä muunnettavan dokumentin (lähdedokumentti) että tyylisivun ja tuottaa uuden dokumentin (loppudokumentti) näiden pohjalta.

Dokumentin muunnoksessa prosessori etsii XML-dokumentin jokaista elementtiä vastaavan XSLT-säännön ja suorittaa sen. Sääntö koostuu esimerkiksi loppudokumenttiin tuotettavista elementeistä, attribuuteista ja leipätekstistä sekä XSLT:n muuttujien asetuksista, ehtolauseista ja silmukoista. Sääntö voi kohdistua useaan elementtiin kerralla ja sen laukeamiseen voi vaikuttaa elementtien järjestys, sisäkkäisyys tai attribuuttien arvot. Säännössä voidaan myös kieltää tai erityisesti määrätä elementin lapsielementtien käsittely.

XSL on tarkoitettu ensisijaisesti käytettäväksi asiakaslaitteissa (client) XML-datan muotoiluun ja esittämiseen. Esimerkkikohteita XSL:n käytöstä voivat olla esimerkiksi XML-muotoon kapseloidun ostoskorin tai lomakkeen sisältämän datan esittäminen ja muotoilu.

Mielenkiintoinen piirre on myös HTML:n tuottaminen, sillä HTML:hän ei ole XML:ää vaan SGML:ää. Vaikka erot XML:n ja SGML:n välillä ovat usein pieniä, ovat ne kuitenkin olemassa, joten HTML on erikoistapaus XSLT:n näkökulmasta.

Ulkoasultaan standardien web-sivustojen tuottamisessa on jo pidemmän aikaa käytetty CSS:ää. Tuki eri CSS-standardin versioille on kuitenkin rajoittunutta ja osittaisia XML dokumenttien visualisointeja voidaan tehdä vain MS IE 5.0, Opera ja Mozilla selaimilla. Myöskään interaktiivisten osien luominen HTML:n forms-tyyppisellä toiminnallisuudella ei ole mahdollista CSS:n avulla. Lisäksi CSS mahdollistaa vain esityksen määrittelyn, ei rakenteen muunnosta, transformaatiota, toisen mallin mukaiseksi. Edellä mainituista seikoista johtuen XSLT onkin vähitellen alkanut korvata CSS:ää sivujen tyylin määrittäjänä. CSS tarjoaa keinot vain datan esittämiseen, XSL/XSLT tarjoaa saman sekä tavan muokata dataa. Vaikka CSS onkin huomattavasti helpompi lähestymistapa, se ei ole läheskään yhtä tehokas kuin XSLT. [Stark&Hjelm2001]

Sekä XSL:ää että CSS:ää voidaan käyttää XML-dokumenttien ulkoasun määritykseen. Jotkin XSL:n formatointiobjektit ja monet niiden ominaisuudet on kopioitu suoraan CSS2:sta. Ominaisuudet voidaan jakaa neljään tyyppiin sen perusteella, missä suhteessa ne ovat CSS:ään:

- CSS-ominaisuudet, jotka on kopioitu suoraan,
- CSS-ominaisuudet, joiden arvojoukkoa on laajennettu,
- CSS-ominaisuudet, joita on tarkennettu jakamalla ne osiin,
- Vain XSL:lle tyypilliset ominaisuudet.

CSS:n ja XSL:n formatointiprosessi on samantapainen, mutta XSL käyttää XML-notaatiota ja CSS omaansa. CSS:ssä dokumentin lähdepuu ja formatointiobjekteja sisältävä tulospuu ovat lähes samanlaiset, mutta XSL:ssä ne voivat erota toisistaan huomattavasti. XSL on tarkoitettu monimutkaiseen formatointiin, jossa dokumentin sisältö saatetaan esittää monessa eri paikassa. Esimerkiksi tekstin otsikko voi esiintyä myös automaattisesti generoidussa sisällysluettelossa. CSS soveltuu parhaiten yksinkertaisempien dokumenttien kuvaamiseen näytöllä, kun taas XSL soveltuu hyvin myös paperijulkaisun taittoon.

## 3. Miten XSLT toimii?

Tyylimäärittelyjen avulla tietoa voidaan muuntaa sellaiseen muotoon, joka on kohdeyleisölle sopiva. Koska tämä yleisö vaihtelee tietyn tiedon osan mukaan, tulevat erialiset tyylimäärittelyt tarpeeseen. Tällä tavalla voidaan tieto saattaa jokaisen yleisön vaatimuksien mukaiseksi. Tällä tarkoitetaan tiedon osien järjestämistä ja esim. tiettyjen osien esilletuomista ja korostamista.

Ensimmäinen XSLT:n perussääntö on se, että kaikkein tyylisivujen tulee olla hyvin muodostettuja. Kaikkien hakasulkeiden tarvitsee olla joko osa HTML:stä tuttua aloitus / lopetussukuparia (<> ja </>) tai tyhjä elementti ’sulje’, joka sisältää lopetusta osoittavan kauttaviivan. (</>) [w3c2000]

Käännösprosessissa XSLT käyttää Xpath:ia määritelläkseen sellaisia lähdedokumentin osia, jotka vastaavat yhtä tai useampaa ennalta määrättyä mallia (engl. t*emplate*) eli ns. sääntökokoelmaa. Kun vastaavuus löytyy, XSLT kääntää tämän vastaavaan osan lähdedokumenttia lopputulosdokumentiksi. Ne lähdedokumentin osat, joille ei löydy vastaavuutta mallista, päätyvät muokkaamattomina lopputulosdokumenttiin.

Säännön perusmalli perusmalli on esitelty ohjelmakatkelmassa 1.

```
<xsl:template match="paikannuspolku">
... säännön sisältö ...
</xsl:template>
```

Ohjelmakatkelma 1. XSLT säännön perusmalli.

YhteenvetoMuiden sääntöjen kutsuminen säännön sisältä tapahtuu ohjelmakatkelmassa 2 esitetyllä tavalla.

```
<xsl:apply-templates />

<xsl:choose>
<xsl:when test="ehto 1"> ... </xsl:when>
<xsl:when test="ehto 2"> ... </xsl:when>
<xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>
```

Ohjelmakatkelma 2. Muiden sääntöjen kutsuminen säännön sisältä.

Esimerkki kuvaa ehtolausetta, joka vastaa monien ohjelmointikielten switch- ja case-lauseita. XSLT-muunnin suorittaa ensimmäisen `<when>`-elementin sisällön, jonka ehto on tosi. Jos mikään ehto ei ole tosi, suoritetaan `<otherwise>`-elementin sisältö. Pidempien XSLT-säännöstöjen kirjoittaminen on kuitenkin usein hidasta ja työlästä, sillä pienenkin toiminnon tekemiseen tarvitsee yleensä kirjoittaa suhteellisen paljon XSLT-koodia. Esimerkiksi normaalin `if-then-else`-rakenteen kirjoittaminen `xsl:choose`-elementin avulla käy helposti aikaavieväksi. Valitettavasti kirjoittamista nopeuttavia työkaluja on vähän, kuten myöhemmin tässä tutkimuksessa tulemme näkemään. [Kay2001]

Kun XSLT:tä käytetään www-julkaisuun, voi koodi sijoittua HTML:ään nähden kahdella tapaa.

Ensinnäkin XSLT-säännöt voivat täydentää HTML-dokumenttia. Tätä kutsutaan *täydennettäväksi malliksi*. (katso ohjelmakatkelma 3)

```
<html xsl:version="1.0"
xmlns:xsl="http://www.w3.org/2000/XSL/Transform">
         //onko polku oikein
<head>
<title><xsl:value-of select="//otsikko" /></title>
```

```
</head>
```

Ohjelmakatkelma 3. Täydentävä malli.

Toinen tapa on muuntaa jokin XML-dokumentti elementti elementiltä HTML-dokumentiksi. Eli otetaan XML-elementti ja tehdään siitä suoraan jokin HTML-elementti ilman ylimääräisiä muunteluita. Kun jokaiselle XML-elementille on tehty oma sääntönsä, voidaan vähitellen siirtyä monimutkaisempiin asioihin, kuten elementtien siirtelyyn toiseen paikkaan, sisällön kopioimiseen, sisällysluettelon tekemiseen, jne. Tätä kutsutaan *sääntöpohjaiseksi malliksi*. Ohjelmakatkelma 4 havainnollistaa asiaa.

```
<xsl:template match="otsikko">
<title>
<xsl:apply-templates />
</title>
</xsl:template>
```

Ohjelmakatkelma 4. Sääntöpohjainen malli.

**Askeleet muodostavat Xpath-polun**

Sääntöjen laukeamissäännöissä käytetään Xpath-standardia. XPath on olennainen työväline XSLT- tyylisivuja luotaessa. XPath:n avulla XML-dokumentista voidaan etsiä osajoukkoja, joka toteuttaa annetun XPath-polun. Osajoukko voi sisältää yksittäisiä elementtejä, elementtiryhmiä tai suurempia dokumentin osia. Mikäli polku ei johda minnekään, tulokseksi saadaan tyhjä joukko.

XPath-polku koostuu yhdestä tai useammasta askeleesta, jotka koostuvat kolmesta osasta: askeltyyppi, määrite ja predikaatti. Askeltyyppi kertoo käytettävän askeleen, esimerkiksi suunnan XML-dokumentissa, määrite asettaa ehtoja askeleen ottamiselle ja predikaatti tarkentaa määritettä, yleensä lisäehdoilla. Askel kirjoitetaan seuraavasti

```
askeltyyppi:: määrite[predikaatti].
```

Sekä askeltyyppi että predikaatti ovat valinnaisia. Askeleet erotetaan toisistaan kauttaviivoilla, jolloin niistä muodostuu polku. Polku suoritetaan askel kerrallaan ja tulokseksi saadaan joukko elementtejä tai tyhjä joukko.

Askeltyypillä määritetään askeleen toiminto. Muutamia askeltyyppejä on koottu tämän luvun lopusta löytyvään taulukkoon. Mikäli askeltyyppiä ei erikseen määritetä, käytetään lapsiin siirtyvää askeltyyppiä

```
(child::).
```

Polun suoritus alkaa nykyisestä elementistä ja etenee askeltyyppien avulla tiettyyn joukkoon elementtejä. Nykyisen elementin lapsielementit, joiden nimi on , valittaisiin säännöllä:

```
child::p
```

Jos halutaan valita vain ensimmäinen noista elementeistä, olisi sääntö

```
child::p[position()=1].
```

Vastaavasti voidaan valita elementtiä ulommista elementeistä, käyttäen seuraavanlaista sääntöä:

```
ancestor-or-self::header[@level='1']/child::p/child::text().
```

Sääntö valitsee nykyistä elementtiä ulommista elementeistä -elementit, joiden attribuutin "level" arvo on "1". Näiden elementtien lapsista valittaisiin kaikki -elementit, joiden lapsista valittaisiin kaikki leipätekstisirpaleet.

Koska säännöistä tulee helposti pitkiä, on yleisimmille askeltyypeille määritelty lyhenteitä. Lyhennettynä edellä esitetty sääntö olisi

```
//header[@level='1']/p/text().
```

Useita erinimisiä elementtejä voidaan valita yhdessä askeleessa luettelemalla elementtien nimet pystyviivalla erotettuna, esimerkiksi

```
header/p|image.
```

XPath-säännöillä voidaan valita hyvinkin monimutkaisia ja mielivaltaisia alueita dokumentista, jolloin säännöistäkin tulee helposti pitkiä ja hankalia ymmärtää. Peruskäyttöä varten ei kuitenkaan yleensä tarvita yhtä tai kahta askelta pitempiä polkuja.

**Päällekkäiset ja sisäänrakennetut säännöt**

Yhteen lähdedokumentin elementtiin voi vaikuttaa useita XSLT-sääntöjä, joista XSLT-prosessori valitsee elementin sijainnin tai attribuuttien mukaan sopivimman. XPath-polkujen avulla voidaan määrittää sääntö koskemaan esimerkiksi kaikkia elementtejä, jotka ovat elementin sisällä, siis "`header/p`". Tämän säännön lisäksi voi olla sääntö, joka koskee kaikkia elementtejä , eli "`p`". XSLT-prosessori valitsee säännöistä aina sen, joka tarkimmin kuvaa elementin tilannetta. Vain yksi sääntö suoritetaan. Tässä tapauksessa siis jälkimmäinen sääntö valitaan sellaisissa tapauksissa, joissa elementin ulompana elementtinä ei ole elementtiä . [xslt2001]

Aina ei ole tarpeen tai käytännöllistä käydä kaikkia elementin sisältämiä elementtejä läpi. Osa elementeistä voi olla turhia kyseisessä muunnoksessa tai ne

käsitellään osana toista sääntöä. Yksinkertaisimmillaan elementtien valinta tapahtuu lisäämällä elementtiin select-attribuutti, joka saa arvoksen XPath-polun, esimerkiksi

```
<xsl:value-of select="TITLE"/>.
```

Nyt XSLT-prosessori suorittaa säännöt ainoastaan -TITLE-elementeille ja muut elementit jätetään huomiotta. -elementeille valitaan sääntö käyttäen normaalia säännönvalintamekanismia.

XSLT:ssa on muutama sääntö sisäänrakennettuna. Niiden avulla käsitellään sellaiset elementit, joita varten ei ole määritelty omia sääntöjä. Sisäänrakennetut säännöt käyvät kaikki elementit läpi, kopioivat attribuutit ja leipätekstisirpaleet ja jättävät kommentit huomioitta. XSLT sisältää myös toiminnallisuudet elementtien ja attribuuttien kopiointiin lähdedokumentista loppudokumenttiin, ehdollisten alueiden ja toistojen rakentamisen sääntöihin, ja useiden tyylisivujen yhdistämiseen.

Seuraavaan taulukkoon on tämän osan lopuksi koottu yhteenvetona XSLT:n erilaisia askeltyyppejä, lyhenteineen ja selityksineen.

| Askeltyyppi | Lyhenne | Selitys |
|---|---|---|
| root | / | Dokumentin juurielementti (polun alussa) |
| self | . | Nykyinen elementti |
| parent | .. | Nykyistä elementtiä ulompi elementti |
| ancestor | | Nykyistä elementtiä ulommat elementit |
| ancestor-or-self | | Nykyinen elementti ja sitä ulommat elementit |
| child | Tyhjä | Elementin lapsielementit |
| Descendant | // | Elementin lapsielementit ja näiden lapset |
| Descendant-or-self | | Nykyinen elementti kaikkine lapsineen |
| preceding | | Nykyistä elementtiä edeltävät elementit |
| following | | Nykyistä elementtiä seuraavat elementit |
| attribute | @ | Attribuutti |
| text() | | Leipätekstisirpale |

Taulukko 1 Erilaisia askeltyyppejä

# 4. XSLT ja selaimet

Lähdettäessä tarkastelemaan XSLT:n yhteensopivuutta eri Internet selainten kanssa, on tarkastelu parasta aloittaa XML:stä käsin, koska XSLT on kiinteä osa sitä. Nykyisin käytössä olevista yleisimmistä selaimista Internet Explorer 4.01 sekä sitä uudemmat versiot tukevat XML-formaattia. Niissä on myös käytössä Microsoftin kehittämät JavaScript ja VisualBasicScript XML-dokumenttien käsittelyyn tarkoitetut luokat (XMLDOM). [msfaq2001] Käytännössä tämä tarkoittaa, että puhtaita XML-dokumentteja voi käyttää vai Internet Explorerin tai jonkun muun XML:n editoimiseen tarkoitetun työkalun avulla. Mikäli XML-aineistoa halutaan käyttää selainriippumattomasti on käyttöön otettava scriptejä, jotka muokkaavat dokumentin selaimelle sopivaksi (yleensä HTML-muotoon). Tällöin esimerkiksi palvelimelle asennettu scripti muuttaa tiedoston HTML-muotoon ennen selaimelle lähetystä. Tämä voi tapahtua joko käyttämällä hyväksi Microsoftin XMLDOM-luokkaa, saatavilla olevia muunnostyökaluja tai normaaleja tekstinkäsittelyoperaatioita, jolloin tehtävästä tulee huomattavasti hankalampi. Mikäli konversio tapahtuu selaimen tasolla, on ensin selain tunnistettava ja Netscapen kyseessä ollessa käytettävä pelkkiä tekstinkäsittelyoperaatioita. Internet Explorer osaa käyttää myös XSL-tiedostoja, jolloin XML-dokumentin esitystapaa on suhteellisen helppo muokata tarpeen mukaan. Netscapen mukaan sen seuraava versio (Netscape 6.0) sisältää täyden XML-tuen, mutta ainakaan hiljattain julkaistussa beta-versiossa ei XML-tukea ollut juurikaan tarjolla. Tämä johtuu mitä ilmeisemmin siitä, ettei selain ymmärrä XSL-määrittelyjä. Avattaessa XML-tiedosto Netscape 6.0 selaimella, tulostuu dokumentin sisältö (eivät hakasulkeet) yhtenä merkkijonona näytölle.

XSLT-tyylimäärittelyllä voidaan siis määrittää dokumentille tietty esitystapa selaimella HTML-muodossa. Kuten jo aiemmin nähtiin, XSLT:n avulla dokumentista voidaan suodattaa, järjestää ja hakea tietoa eri parametreillä. Se ei sinällään

mahdollista uuden tiedon lisäämistä dokumenttiin, vaan määrittää miten XML-dokumentti muutetaan toiseen muotoon tai esitetään.

WWW-julkaisun ollessa kyseessä halutaan yleensä varmistaa sivuston näkyminen mahdollisimman monella selaimella. Tätä varten on olemassa `xsl:output`-elementti. Kun sen metodin attribuutti saa arvon "html", `xsl:output`-elementti, joka on esitelty ohjelmakatkelmassa, käskee XSLT prosessoria näyttämään vain tyhjät HTML-elementit (`area`, `base`, `basefont`, `br`, `col`, `frame`, `hr`, `img`, `input`, `isindex`, `link`, `meta` ja `param`, XSLT spesifikaation mukaan) yhtenä hakasulkeena ilman lopetus viivaa. Ohessa lyhennetty esimerkki output-elementistä.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>     //output-elementti

<xsl:template match="esimerkki">
  <html><body>
    <xsl:apply-templates/>
  </body></html>
</xsl:template>

<xsl:template match="title">
  <h1><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="tekstirivi">
  <xsl:apply-templates/><br/>
</xsl:template>

</xsl:stylesheet>
```

Ohjelmakatkelma 5. XSL-output-elementti.

XSLT-prosessori muokkaa ohjelmakatkelmassa 5 kuvattua tyylisivua hyväksi käyttäen ohjelmakatkelman 6 XML-dokumentin ja tulokseksi saadaan ohjelmakatkelmassa 7 esitetty HTML-esitys.

```
<esimerkki><title>Esimerkki</title>
<tekstirivi>rivi tekstiä.</tekstirivi>
<tekstirivi>Toinen rivi tekstiä.</tekstirivi>
</esimerkki>
```

Ohjelmakatkelma 6. Esimerkkidokumentti.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<body>
<h1>Esimerkki</h1>
<hr>
<p>
Rivi tekstiä.<br>
Toinen rivi tekstiä.<br>
</p>
<hr>
</body>
</html>
```

Ohjelmakatkelma 7. Ohjelmakatkelmasta 6 saatu HTML-esitys.

Otsikon jälkeen tulostetaan jokainen esimerkki-elementti ja sen jälkeen tulostetaan HTML:n `br` eli rivinvaihto elementti. Koska kaikkien tyylisivujen tarvitsee olla hyvinmuodostettuja, `br` elementti sisältävää aloitus- ja lopetushakasulkeet. `Br`-elementissä on myös lopetusviiva osoittamassa sitä, että se edustaa tyhjää elementtiä. Koska `xsl:output`-elementin metodin arvosta johtuen, br elementti on yksittäinen hakasulku ilman lopetusviivaa, ei tämän tyylisivun pitäisi aiheuttaa ongelmia millekään selaimelle.

# 5. XSLT käytännössä

XSL-transformaatio voidaan suorittaa pääsääntöisesti kolmella eri tavalla:

- XML-dokumentti ja siihen liittyvä XLS-tiedosto lähetetään asiakkaan selaimelle, jonka prosessori suorittaa transformaation.
- Transformaatio suoritetaan palvelimessa esimerkiksi ASP- tai JSP-koodilla ja valmis tulosdokumentti lähetetään asiakkaan selaimelle.
- Dokumentti transformoidaan ennen kuin se sijoitetaan palvelimelle, jolloin sekä palvelin että asiakas käsittelevät vain tulosdokumenttia.

Tällä hetkellä toimivimmalta ratkaisulta vaikuttaisi palvelimessa suoritettava transformaatio. Yksi XSL:n kehittämisen päämääristä on ollut mahdollistaa tiedon muuntaminen muodosta toiseen palvelimessa, jolloin voidaan tarjota sopiva esitysmuoto kaikille mahdollisia selaimille.

**XSLT-prosessoreita**

Käyttääkseen XML:ää täytyy opetella joukko uusia työkaluja sekä kieliä. XSLT:llä on hyvin tärkeä rooli tällaisessa sovellusmallissa. Vaikka XSLT onkin suhteellisen yksinkertainen, on se myös hyvin ilmaisuvoimainen kieli, jonka täydellinen osaaminen vaatii työtä. XSLT:n tuottamista varten on olemassa erilaisia prosessoreita, mutta näiden prosessorienkaan käyttö ei ole ongelmatonta. Tällaisia prosessoreita ovat mm. seuraavat:

**IXSLT** Infoterian kaupallinen C ++ :lla toteutettu prosessori,

**Xalan-C,** Apache-projektin vapaa C ++ :lla toteutettu prosessori,

**XT,** James Clarkin vapaa Java-pohjainen prosessori,

**Saxon,** Michael Kayn vapaa Java-pohjainen prosessori.

Näiden prosessorien käyttö ei kuitenkaan ole ongelmatonta, kuten Mika Raento [Raento2000] pro gradu- tutkielmassaan osoittaa. Kyseisessä tutkielmassa prosessoreita on vertailtu niiden muistikulutuksen ja aikavaativuuden perusteella. Kaikki esitellyt prosessorit rakentavat koko kohdepuun valmiiksi muistiin ennen sen tulostamista, minkä voi päätellä siitä, että melkein koko ajoaika kului ennen kuin prosessori tulosti mitään. Prosessorit säilyttävät koko lähdepuun ja koko kohdepuun muistissa yhtäaikaa, ja siten käyttävät melko paljon muistia. Jos XSLT:tä halutaan käyttää reaaliaikaisessa prosessoinnissa, täytyy prosessorin kyetä käsittelemään muunnos suhteellisen lyhyessä ajassa.

**Tietokantasovellukset, esimerkkinä Oracle XSQL –servlet**

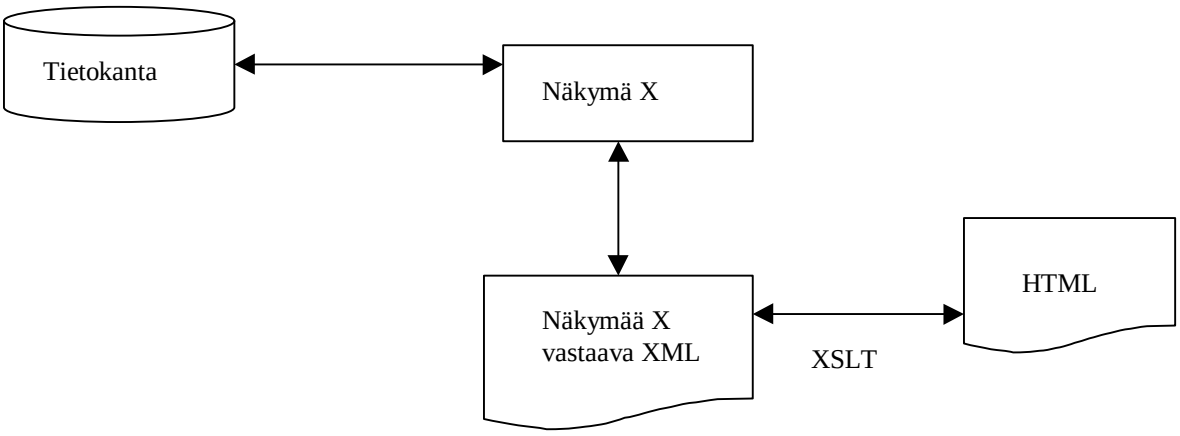
WWW:ssä olevan tietomäärän jatkuva kasvu on samalla lisännyt tietokanta-sovellusten käyttöä. Tästä johtuen HTML:n ilmaisuvoima ei enää ole ollut riittävä. HTML-dokumentit eivät anna tiedonhakuun muuta mahdollisuutta kuin sisällön läpikäymisen sana sanalta. XML puolestaan mahdollistaa dokumenttien erittäin joustavan määrittelyn, kuten olemme jo aiemmin tässä tutkimuksessa havainneet. XML-kyselykieli tuo myös merkittäviä parannuksia moniin www:ssä ilmeneviin ongelmiin. Koska XML mahdollistaa dokumenttien rakenteen ja sisällön määrittämisen, on selvää, että esimerkiksi www-hakukoneiden osumatarkkuus paranee merkittävästi. XML-kyselykielen avulla voidaan hakuja tehdä perinteisistä www-kyselykielistä poiketen suoraan XML-dokumenttikokoelmiin. Tämä johtuu siitä, että jos dokumenttien rakenne tai DTD on tunnettu, voidaan kyselyt tehdä suoraan dokumenttikokoelmiin, ilman, että niistä tehdään ensin erillinen tietokanta, johon kyselyt kohdistetaan.

Yksi merkittävimpiä XML-kyselykielen mukanaan tuomia parannuksia, on eri lähteistä löytyvien tietojen yhteensulauttaminen automaattisesti. Esimerkiksi monet laitevalmistajat ja jälleenmyyjät päivittävät tällä hetkellä käsin tarjoamiensa

kokoonpanojen eri komponenttien yksityiskohtaiset tiedot. Kun tietojen tallentamisessa käytetään riittävän hyvin määriteltyjä ja tunnettuja XML-rakennemäärittelyjä, voi esimerkiksi jälleenmyyjä hakea sivuilleen automaattisesti käyttämiensä komponenttien viimeisimmät tiedot suoraan valmistajan www-sivuilta. Samaan tapaan www-sivustojen hallinta helpottuu merkittävästi, koska sivustot voidaan luoda automaattisesti olemassa olevista dokumenteista. Tällöin www-sivujen ylläpitäjien ei tarvitse tehdä päivityksiä sivuilleen aina dokumenttien sisällön muuttuessa, vaan riittää, että sivujen luomiseen käytetyt hakuehdot ovat ajan tasalla.

XML:n avulla voidaan helposti luoda yksikäsitteinen esitystapa tietokanta-näkymälle ja muut muodot voidaan muuntaa omaa näkymää vastaavaan muotoon XSLT:n avulla. Vaikka itse XML on kypsä käytettäväksi, ei XML Schema ole vielä valmis ja XSLT:n käytössä tähän tarkoitukseen on suorituskykyongelmia. XSLT on valmis käytettäväksi, mutta nykyiset prosessorit ovat vielä muistinkulutukseltaan kehnoja. XSLT:n käyttöä sellaisissa sovelluksissa, joissa käsitellään suuria määriä tietoa, tulee varoa, mutta luultavasti tulevaisuudessa saadaan muistinkäytöltään optimoituja XSLT-prosessoreita.

Kuva 1 havainnollistaa XSLT:n hyödyntämistä tietokantapohjaisissa www-sovelluksissa.




Kuva 1. Esimerkki XSLT:n käytöstä tietokantasovelluksessa.

Tietokantaohjelmistoihin keskittyneen Oraclen XSQL servletti mahdollistaa ulkoisten XSLT-tyylisivujen määrittelyn. Nämä tyylisivut voivat sijaita fyysisesti missä vain. Servletti sisältää XSL-muunnokset itsessään. Lisäksi se sisältää tavan tehdä SQL-kyselyjä relaatiotietokantoihin ja mahdollistaa läpinäkyvän jakelun loppukäyttäjille. Ongelmana tässä on kuitenkin se, että XSLT tyylisivuissa on mahdollista suorittaa Java Web-palvelimella. Javan suorittaminen Web-selaimella saattaa kuitenkin tehdä sen toiminnasta epävakaata.

Oraclen kehittämä XSQL Pages - julkaisutyökalu tarjoaa laajan alustan XML-pohjaisen tiedon julkaisulle missä tahansa formaatissa. Se yhdistää SQL:n, XML:n ja XSLT:n, ja mahdollistaa näin tietokantapohjaisen informaation dynaamisen web-julkaisun.

Ohjelmakatkelmassa 8 on esimerkki XSQL-dokumentista, jonka avulla voitaisiin hakea esim. kirjaston tietokannasta tekijä sukunimen perusteella.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="author.xsl"?>
<!-- Haetaan julkaisun tekijän tiedot -->
<xsql:query pub_id="%"
               rowset-element="TEKIJAGROUP"
               row-element="TEKIJA"
               xmlns:xsql="urn:oracle-xsql"
               connection="sukunimi">
SELECT
        p.h_enimi AS ETUNIMI,
        p.h_snimi AS SUKUNIMI,
        a.edi
FROM    pub,
        a,
        p

WHERE   pub.p_id = '{@pub_id}'
        and a.p_id = pub.p_id
        and a.h_id = p.h_id
ORDER BY indeksi
</xsql:query>
```

Ohjelmakatkelma 8. Esimerkki XSQL-dokumentista.

## 6. XSLT:n käyttömahdollisuudet nyt ja tulevaisuudessa

Useat XSLT-työkalut ovat edelleen keskeneräisiä, kuten aiemmin jo tuli ilmi. Erityisesti näin on muiden kuin Java-työkalujen kohdalla. Tämä ei tarkoita, että kaikki työkalut olisivat käyttökelvottomia tai huonoja. Täyttä XSLT-tukea etsivät joutuvat pettymään monen sovelluksen kohdalla, mutta muille useat keskeneräiset sovelluksetkin toteuttavat varsin käyttökelpoisen osan XSLT:stä. Tärkeää on määritellä omat käyttötarpeensa ja tutustua sovelluksiin ennen lopullisen käyttöpäätöksen tekemistä. Yleensä ongelmia aiheuttavat seuraavat asiat

- nimiavaruudet (erityisesti DTD:hen yhdistettynä)
- merkistöt (erityisesti muut kuin UTF-8 ja ASCII).

Koska XSLT on nuori teknologia, esiintyy sen käytön yhteydessä usein tehottomuuteen ja muistivaatimuksiin liittyviä ongelmia. Kehittyneimmätkin XSLT-muuntimet ovat tällä hetkellä vasta alkuvaiheen sovelluksia. Niissä on tehty joitain optimointeja muunnosten tehostamiseksi, mutta paljon on vielä toteuttamatta. Itse asiassa tällä alueella on vielä paljon tutkimustyötäkin tekemättä. Sovellusten lisäksi

myös itse XSLT-määrittely on vielä suhteellisen nuori. Vasta pidemmän ajan käyttökokemukset paljastavat määrittelyn suurimmat puutteet ja kehitystarpeet. Joitakin tällaisia kohteita - kuten tuotettavan lopputuloksen pilkkominen useampaan tiedostoon - on jo löytynyt, ja moni XSLT-muunnin toteuttaa ne määrittelyn ulkopuolisena laajennuksena.

Koska XSLT on XML:ää, tarkastelen seuraavassa lyhyesti myös XML:än käyttömahdollisuuksia www-ympäristössä. XSLT yhdistettynä johonkin ohjelmointikieleen tarjoaa mahdollisuuden esittää XML-muotoinen tieto halutun muotoisena ilman, että asiakaslaite sitä havaitsee. Käytännössä sama tieto voidaan siis esittää ja siirtää niin html2- ,3- ja 4- standardeihin kuin WML-ympäristöön, joko vaihtamalla XSLT-asiakirja tai luomalla se dynaamisesti päätelaitteen mukaisesti. Ainut rajoitus tällöin on palvelimella käytössä oleva XML-selain.

Vaikka teknologinen tuki XML-sovelluksille onkin pitkälti jo olemassa, on matkalla monia esteitä. Ennen kuin XML:n voi siirtää puhtaasti asiakasympäristöön, on enemmistön päätelaitteista tuettava sitä. Tätä nykyä esimerkiksi Suomessa selaimista noin 80% on XML-kelvollisia; maailmanlaajuisesti luku on hieman alle 70%.

Yksi suuri ongelmakohta on XML:n ja XSLT:n osaamisen vaikeus. Vaikka molemmat kielet ovatkin perusteiltaan suhteellisen helppoja ja hyvin muotoiltuja dokumentteja on niiden avulla suhteellisen helppo rakentaa, on syvempien rakenteiden ymmärtäminen vaikeaa. Monipuolisen ja laajan XML-dokumentin tai XML-sovelluksen rakentaminen alusta alkaen on vaativa tehtävä, ja sen lisäksi on usein luotava vielä DTD, XML Schema ja tarvittavat XSL-tiedostot.

Ongelmana on toistaiseksi ollut myös se, että XSL-tuki ohjelmistopuolella on heikkoa. XSLT–transformaatiokieli alkaa olla jo käyttökelpoinen, koska ensimmäinen standardi on julkistettu, mutta XSL-FO on vielä kokeiluasteella. Esimerkiksi CSS:n toteutus on jo huomattavasti valmiimpi ja tuki selaimille jo melko vankalla pohjalla. Paras tapa kuvata XML-dokumentin ulkoasua onkin tällä hetkellä CSS. XSLT:stä näyttäisi tulevan merkittävä kieli jo lähitulevaisuudessa, sillä XML-dokumenttien julkaisu tietokannoista eri formaateissa tulee yleistymään. Myös uusiin selaimiin on kiitettävästi lisätty uusimman XSLT-määrityksen tuki. Sen sijaan XSL-FO:n tilanne riippuu siitä, kuinka nopeasti standardi saadaan valmiiksi, sillä sen myötä saadaan tukea ohjelmistopuolelle.

XSL:ää pidetään yleisesti varsin hankalana tapana määrittää dokumentin ulkoasu ainakin CSS:ään verrattuna. Mutta kun tyylisivu on kerran saatu valmiiksi, sitä voidaan käyttää moniin tarkoituksiin. Jos XSL:stä halutaan muidenkin kuin erikoisosaajien työkalu, tarvitaan sovellusohjelmia, joilla XSL-dokumenttien luontia voidaan helpottaa.

Yllä esitellyistä ongelmista ja vaikeuksista huolimatta tarjoaa XSLT nyt ja tulevaisuudessa monia etuja. Ensinnäkin XSLT mahdollistaa dokumenttien muuntamisen esitysmuodosta toiseen. Lisäksi sen avulla on mahdollista muuttaa dokumentissa olevien elementtien suoritusjärjestystä sekä lisätä ja poistaa elementtejä, ja samalla siis luoda erilaisia näkymiä ja versioita erilaisiin tarkoituksiin. Tyyliin ja dokumentin muotoiluun liittyviä hyötyjä ovat vakiotekstien, automaattisen otsikoinnin ja sisällysluetteloiden luonti. Tyylielementit mahdollistavat tarkan ja monipuolisen ulkoasun määrittelyn. Ja koska XSLT pohjautuu XML:ään, on se laajennettavissa eri käyttäjäkohtaisiin tarkoituksiin. Pienimuotoisiin www-sovelluksiin XSLT:tä en vielä suosittelesi käytettäväksi, koska silloin on mahdollista, että saatava hyöty jää käytettyjä voimavaroja pienemmäksi. Lähdettäessä toteuttamaan suuremman luokan kokonaisuuksia www:hen, on XSLT kuitenkin erittäin varteen otettava vaihtoehto.

## Viiteluettelo


[Arciniegas, 2001] Fabio Arciniegas, *XSLT Applications Patterns and Best Practices*, Howard W Sams & Company, 2001.

[Stark&Hjelm2001] P. Stark & J. Hjelm, *XSLT: The Ultimate Guide to Transforming Web Data*, Wiley, 2001.

[Kay, 2001] Michael Kay, *XSLT Programmer's Reference*, Wrox Press Ltd, 2001.

[Holzner2000] Samuel Holzner, *Inside XML***,** New Riders Publishing, 2000.

[Skonnard, Gudgin, 2001] Aaron Skonnard, Martin Gudgin, *Essential XML Quick Reference: A Programmer's Reference to XML, XPath, XSLT, XML Schema, SOAP, and More*, Addison-Wesley, 2001.

[Grandi, Mandreoli,2000] Fabio Grandi, Federica Mandreoli, The Valid Web: An XML/**XSL** infrastructure for temporal management of web documents, *Lecture Notes in Computer Science 1909* (2000) 294-303.

[Kerer, Kirda,2001] Clemens Kerer, Engin Kirda, Layout, content and logic separation in web engineering, *Lecture Notes in Computer Science* 2016 (2000) 10-21.

[Raento2000] Mika Raento, XML:n käyttäminen tiedon rakenteen esittämiseen ohjelmien ja ohjelmistokomponenttien välisessä tiedonsiirrossa. Tietotekniikan pro gradu-tutkielma, Jyväskylän yliopisto Tietotekniikan laitos 23.5.2000. *http://www.co.jyu.fi/~mikie/raportti.pdf*

[w3c2000] World Wide Web Consortium, XSL Specification *http://www.w3.org/Style/XSL/* viitattu 1.12.2001.

[xslt2001] A Resource Site for XSLT *http://www.xslt.com* viitattu 1.12.2001.

[msfaq2001] Unofficial MSXML XSLT FAQ
*http://www.netcrucible.com/xslt/msxml-faq.htm* viitattu 1.12.2001.
[xml2001] XML:n kotisivut. *www.xml.org* viitattu 1.12.2001.