Erkki Mäkinen (toim.)

Pieniä tietojenkäsittelytieteellisiä tutkimuksia
Syksy 2008

TIETOJENKÄSITTELYTIETEIDEN LAITOS
TAMPEREEN YLIOPISTO

D-2008-12

TAMPERE 2008

TAMPEREEN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
JULKAISUSARJA D – VERKKOJULKAISUT
D-2008-12, JOULUKUU 2008

Erkki Mäkinen (toim.)

Pieniä tietojenkäsittelytieteellisiä tutkimuksia Syksy 2008

TIETOJENKÄSITTELYTIETEIDEN LAITOS
33014 TAMPEREEN YLIOPISTO

ISBN 978-951-44-7592-4
ISSN 1795-4274

Sisällysluettelo

Geneettiset algoritmit ja koneoppiminen

Mikko Arminen

Tiivistelmä.
Geneettiset algoritmit ovat hakualgoritmeja, joiden toiminta perustuu yksinkertaistettuun käsitykseen evoluution toiminnasta. Ne pyrkivät löytämään määriteltyyn ongelmaan ratkaisun käyttäen evoluutioteoriasta tuttujen käsitteiden, kuten valinta, risteytys ja mutaatio, mukaisia menetelmiä. Koneoppimisen tutkimuksessa on tavoitteena tuottaa järjestelmiä, jotka pystyvät mukautumaan itsenäisesti ympäristön muutoksiin. Geneettiset algoritmit ovat yksi koneoppimisessa paljon käytetty menetelmä, jolla on monia sovelluskohteita. Niillä, kuten muillakin koneoppimisen menetelmillä, on kuitenkin myös omat rajoituksensa ja vahvuutensa, joiden tunteminen on tärkeää tekniikan menestyksellisen soveltamisen kannalta.

Avainsanat ja -sanonnat: Geneettiset algoritmit, koneoppiminen.
CR-luokat: I.1.2, I.2.0, I.2.6

1. Johdanto

Koneoppimista on käytetty paljon järjestelmissä, joiden toimintaan kuuluu suurien tietomäärien analysointia. Sillä on lukuisia sovelluskohteita monilla eri aloilla, kuten esimerkiksi hakukoneissa, lääketieteellisissä diagnooseissa, luottokorttipetosten havaitsemisessa, DNA-sekvenssien luokittelussa, puheentunnistuksessa, tietokonepeleissä ja robottien liikuttamisessa. Järjestelmällä, jossa sovelletaan koneoppimista, on kyky käsitellä myös syötteitä, joita ei ole etukäteen määritelty. Tämä voi tarkoittaa käytännössä sitä, että järjestelmä pystyy suoriutumaan tehtävästä täysin itsenäisesti tai vuorovaikutuksessa ihmisen kanssa. Nykyisin saavutettu järjestelmien autonomisuuden taso, eli kyky mukautua ennustamattomiin tilanteisiin ilman ihmisen osallistumista, on kuitenkin vielä hyvin rajallinen johtuen niiden tuottamiseen käytettyjen tekniikoiden rajoituksista [Abbott, 2005]. Tietojärjestelmien käsittelemien tietomäärien jatkuva kasvu ja tiedon monimutkaistuminen synnyttävät kuitenkin lisääntyvää tarvetta koneoppimisen tutkimukselle uusien sovellutusten mahdollistamiseksi.

Tämän työn tarkoituksena on keskittyä tutkimaan yhtä koneoppimisessa paljon käytettyä menetelmää – geneettisiä algoritmeja. Geneettisten algoritmien teoreettisen tutkimuksen määrä on ollut suhteellisen vähäistä käytön runsauteen nähden [Alander, 2006]. Tämä on kuvaavaa sekä yleisesti geneettisille algoritmeille että erityisesti geneettisten algoritmien soveltamiselle koneoppimisessa. Tämän voi olettaa johtuvan osaltaan geneettisten algoritmien satunnai-

suuteen perustuvasta toiminnasta; geneettisiä algoritmeja käytetään usein menestyksekkäästi ilman, että varsinaisesti ymmärrettäisiin, mitä algoritmin suoritusaikana yksityiskohtaisella tasolla tapahtuu. Geneettisten algoritmien käytöstä koneoppimisessa tehdyssä tutkimuksessa onkin parhaiten edustettuna yksittäisten sovellutusten tarkastelu ja tekniikan yleisten mahdollisuuksien raportointi on vähäisempää.

Tämä työ keskittyy kokoamaan tekniikan teoreettisia mahdollisuuksia ja rajoituksia. Pyrkimyksenä on tämän kautta selventää, millaisten ongelmien ratkaisemiseen tekniikka soveltuu ja vastaavasti millaisten ongelmien ratkaisemiseen se soveltuu huonosti. Tässä tarkastelussa keskitytään geneettisten algoritmien perusrakenteeseen ja tarkastelun pääpaino pyritään suuntaamaan koneoppimisen kannalta tärkeisiin ominaisuuksiin. Lähtökohtana on keskittyä geneettisten algoritmien käytön teoreettisiin rajoituksiin, jotka ovat riippuvaisia niiden rakenteesta. Käytännön sovellutuksissa on luonnollisesti myös käytettävissä olevan toteutustekniikan aiheuttamia rajoituksia, mutta nämä eivät ole samalla tavalla ohittamattomia. Tavoitteena onkin hahmotella sitä, mihin geneettiset algoritmit optimaalisimmillaan pystyvät tilanteessa, jossa ulkoisia rajoitteita ei ole. Tämä tarkoittaa käytännössä sitä, että työssä ei esimerkiksi tarkastella yksityiskohtaisesti algoritmien laskennallista tehokkuutta.

Aiheen käsittely aloitetaan käsittelemällä yleisellä tasolla koneoppimisen ongelmakenttää luomalla lyhyt katsaus siihen, mistä koneoppimisessa on itse asiassa kysymys. Tämän jälkeen siirrytään geneettisiin algoritmeihin liittyvien käsitteiden esittelyyn ja yleisten toimintaperiaatteiden hahmotteluun. Perusasioiden käsittelyn jälkeen siirrytään tarkastelemaan, mikä on geneettisten algoritmien rooli koneoppimisessa. Tämä sisältää ensinnäkin sen selvittämisen, millaisissa sovellutuksissa geneettisiä algoritmeja on koneoppimisessa käytetty. Tässä yhteydessä geneettisiä algoritmeja tarkastellaan myös joidenkin, niiden toiminnan kannalta mielenkiintoisten, ominaisuuksien kautta.

2. Koneoppiminen

Koneoppimisessa on pyrkimyksenä luoda järjestelmä, joka kykenee muodostamaan ratkaisun johonkin määriteltyyn ongelmaan. Tähän on olemassa karkeasti jaoteltuna kaksi erilaista tapaa. Joko muodostetaan ratkaisu olemassa olevan tiedon perusteella tai käytetään jotakin hakualgoritmia mahdollisten ratkaisuvaihtoehtojen läpikäymiseen [Shapiro, 2001]. Yksittäisen algoritmin voidaan tavallisesti määritellä käyttävän ensisijaisesti toista näistä tiedon muodostuksen menetelmistä. Koneoppimismenetelmän luokittelu näiden perusteella ei kuitenkaan ole kaikissa tilanteissa itsestään selvää, koska diskreetin luokittelun sijaan on kyseessä jatkumo, johon eri koneoppimisen menetelmät sijoittuvat.

Menetelmän toiminta voi siis painottua toiseen näistä periaatteista, ja näin yleensä onkin, mutta aina ei ole selvää, kumpi tiedonmuodostuksentapa on ensisijainen.

Koneoppiminen on yksi tekoälyn tutkimuksen osa-alue. Koneoppimisen ongelmakentän selkiytymisen kannalta on hyödyllistä pyrkiä määrittelemään, mitkä tekoälyn osa-alueet kuuluvat koneoppimisen piiriin ja mitkä eivät. Geneettisiä algoritmeja käytetään tekoälyn alueella esimerkiksi erilaisten aikataulujen laatimiseen ja muihin tämänkaltaisiin hakutehtäviin. On kuitenkin tärkeää huomioida tällaisten tehtävien ero varsinaiseen koneoppimiseen nähden. Koneoppimista ei ole tekoälyn sovellukset, jotka vain käyttävät olemassa olevia sääntöjä. Koneoppimiselle on ennen kaikkea ominaista sen pyrkimys muodostaa uusia sääntöjä. Tämä pätee myös deduktiiviseen oppimiseen, jossa yleisiä sääntöjä sovelletaan yksittäistapauksiin eli tuodaan esiin sääntöjä, jotka sisältyvät implisiittisesti yleisempiin sääntöihin. Deduktiivinen menetelmä siis muodostaa sääntöjä, jotka pätevät rajoitetuissa tilanteissa. Tämä tieto sinänsä kuitenkin sisältyy jo alkuperäisiin yleisempiin sääntöihin.

2.1. Erilaiset koneoppimismenetelmät

Koneoppimisen vaiheet ovat oppimisvaihe, jossa järjestelmää opetetaan esimerkkien kautta, ja testausvaihe, jossa muodostetun tiedon pätevyyttä testataan ja tarvittaessa järjestelmää opetetaan lisää uusien esimerkkien kautta [De-Castro and Camargo, 2004]. Näiden vaiheiden järjestyminen ja painotuksellinen suhde sisältää vaihtelua eri koneoppimisalgoritmityyppien välillä. Esimerkiksi vahvistusoppimisessa voi de Castron ja Camargon [2004] mukaan olla tavoitteena tilanne, jossa oppiminen tapahtuu jatkuvassa vuorovaikutuksessa ympäristön kanssa. Saadun palautteen mukaan tehdään sääntöihin muutoksia, jotta algoritmin haluttu ulostulo saavutettaisiin. Tämä siis tarkoittaa käytännössä sitä, että opetus- ja testausvaiheiden vuorottelu on jatkuvaa; tavoitteena on päättymätön oppiminen. Ohjatussa oppimisessa pyritään vastaavasti muodostamaan esimerkeistä koostuvan opetusaineiston avulla funktio, joka osaa käsitellä myös opetusaineistoon kuulumattomat arvot [DeJong et al., 1993]. Eli järjestelmän opetusvaiheessa määritellään, millainen vaste esimerkille tulisi saada. Geneettiset algoritmit ovat esimerkkien käyttöä painottava tekniikka, jolla toteutettu oppiminen on ennen kaikkea vahvistusoppimista [Shapiro, 2001].

Järjestelmän opetusvaiheessa käsitellään kahdenlaisia syötteitä: esimerkkiaineistoa ja olemassa olevia sääntöjä [Mitchell, 1997]. Pääosassa koneoppimisen menetelmistä on painotus esimerkkiaineistossa. Täydellisimmin esimerkkiaineistoon perustuva menetelmä ovat neuroverkot. Vastaavasti deduktiivisessa koneoppimisessa ei käytetä lainkaan esimerkkiaineistoa vaan järjestelmän toiminta perustuu puhtaasti määriteltyihin sääntöihin. Deduktiivinen oppiminen

perustuu erityistilanteissa toimivien sääntöjen johtamiseen yleisemmin pätevistä säännöistä. Oppimismenetelmien jakoon induktiivisiin eli menetelmiin, joissa oppiminen tapahtuu esimerkeistä yleistämällä, ja deduktiivisiin, joissa sovelletaan olemassa olevia sääntöjä, sisältyy eräs huomionarvoinen ominaisuus, joka liittyy yleisemminkin tiedonmuodostukseen. Deduktiivisessa päättelyssä on käytetty sääntökokoelma usein muodostettu induktiivisen menetelmän avulla. Poikkeuksena tästä ovat aksiomaattiset järjestelmät, joiden käyttö ei kuitenkin ole yleensä mahdollista ympäröivän todellisuuden ongelmien ratkaisussa.

2.2. Induktiivinen koneoppiminen

Yleisimmin käytetty koneoppimisen muoto on induktiivinen koneoppiminen, jossa on kyse sääntöjen muodostamisesta suurista tietojoukoista [Mitchell, 1997]. Käytännössä tämä tarkoittaa kaksivaiheista toimintaa, jossa ensin pyritään abstrahoimaan tietoalkioista yleisiä säännönmukaisuuksia. Kun säännöt on saatu näin muodostettua, käytetään määriteltyjä sääntöjä uusien alkioiden luokitteluun. Uusien alkioiden luokitteluun liittyy usein eriasteista epävarmuutta, koska muodostetut säännöt eivät tavallisesti ole riittävän tarkkoja toteuttaakseen absoluuttista luokittelua. Tavoitteena onkin pyrkiä luomaan sääntökokonaisuus, joka on mahdollisimman suppea, mutta onnistuu luokittelemaan mahdollisimman laajan alkiojoukon halutulla tavalla [Mitchell, 1997].

Korb [2004] on esittänyt, että induktiivisella koneoppisella on yhteinen perusta tieteenfilosofiassa tutkitun tiedonmuodostuksessa käytetyn induktion kanssa. Hänen mukaansa ainoa induktiiviselle koneoppimiselle ominainen tieteenfilosofiasta poikkeava piirre on se, että tiedonmuodostusta tarkastellaan algoritmeina. Nämä eri tieteenalat lähestyvät tutkimuskohdettaan hyvin erilaisin metodein, mutta tutkimustyön tavoitteet ovat suurelta osin yhtenevät. Thagard [1988] esittääkin, että kaikki tieteenfilosofinen induktion tutkimus, jolla on jonkinlaisia ansioita, on tarkasteltavissa algoritmisesti. On kuitenkin myös esitetty näkemyksiä, että tieteellisessä tiedonmuodostuksessa olisi jotakin epäalgoritmista, joka jää algoritmisen esitysmuodon ulkopuolelle. Jos hyväksytään tämä voimakas yhteys kahden tutkimusalan välillä, voidaan olettaa koneoppimisen tutkimisen tuottavan hyödyllistä tietoa myös tiedon muodostuksen luonteesta yleisesti.

3. Geneettiset algoritmit

Geneettisten algoritmien toiminta perustuu darwinistiseen luonnonvalintaan ja genetiikkaan. Keskeinen ajatus on luoda joukko ratkaisuja, jotka pyrkivät kehittymään evoluution periaatteiden mukaan kohti yhä parempia ratkaisuja [Holland, 1992]. Geneettisissä algoritmeissa käytetyt evoluution periaatteet ovat va-

linta, risteytys ja mutaatio, joista on perustoiminnan lisäksi kehitetty monenlaisia muunnoksia.

Geneettisille algoritmeille on ominaista, että niiden menestyksellinen käyttö on useissa tapauksissa helppoa, mutta sen sijaan on huomattavan paljon haasteellisempaa selittää yksityiskohtaisesti, mihin tämä perustuu [Alander, 2006]. Tällaiseen toteuttamistapaan saattaa kuitenkin joissakin tapauksissa liittyä ongelmia, jos algoritmin suoritus johtaa tilanteeseen, johon ei ole riittämättömän suunnittelun takia osattu varautua. Tästä syystä on geneettisten algoritmien yleisten toimintaperiaatteiden ja niihin liittyvien ongelmien tunteminen erityisen tärkeää.

Evoluutioon perustuvia tietojenkäsittelytieteiden menetelmiä on esiintynyt jo 1950-luvulta lähtien. Varsinaisena geneettisen algoritmin -käsitteen isänä pidetään yleensä John Hollandia, jonka teoksella *Adaptation in Natural and Artificial Systems* (1975) on ollut tärkeä merkitys käsitteen tunnetuksi tulemisessa [Shapiro, 2001]. Toinen geneettisten algoritmien merkittävimmistä tutkijoista on David E. Goldberg, jonka teos *Genetic Algorithms for Search, Optimization, and Machine Learning* (1989) on yksi tietojenkäsittelytieteen viitatuimmista teoksista.

3.1. Sovellusalueet

Geneettiset algoritmit ovat hakualgoritmeja, ja tästä syystä niiden tyypillisin käyttökohde on erilaiset haku- ja optimointitehtävät, joiden yksinkertaisimpana esimerkkinä on pyrkimys löytää määritellyn funktion optimaalinen minimi tai maksimi [Alander, 2006]. Optimointi kokonaisuutena etsii kuitenkin ratkaisuja hyvin monenlaisiin ongelmiin ja käyttää tähän laajaa valikoimaa erilaisia menetelmiä. Ongelmana perinteisissä optimointimenetelmissä on, että ne toimivat paikallisesti tietyillä arvoalueilla, joten niiden avulla on joissakin tilanteissa mahdollista löytää vain paikallisia ääriarvoja. Ympäröivään maailmaan liittyvät ongelmat eivät myöskään usein muodosta jatkuvaa funktiota tai derivaattoja ei ole. Nämä kuitenkin ovat edellytyksiä useiden optimointimenetelmien käytölle. Geneettiset algoritmit soveltuvat erityisesti näihin tilanteisiin, joissa perinteiset menetelmät epäonnistuvat. Tämä on Shapiron [2001] mukaan seurausta siitä, että geneettisten algoritmien toiminta perustuu yksittäiselle hyvyysfunktion arvolle, jolloin ratkaistavana olevan ongelman muodostama funktio ei aseta toiminnalle rajoituksia.

Geneettisiä algoritmeja on sovellettu hyvin monenlaisten ongelmien ratkaisemiseen. Moniin käyttötarkoituksiin, joihin geneettisiä algoritmeja on käytetty, on kuitenkin tarjolla myös tehokkaampia ratkaisutapoja. Kuitenkin geneettisistä algoritmeista löytyy usein apua juuri ongelmiin, joihin on vaikea löytää muuta ratkaisua [Shapiro, 2001]. Esimerkkinä tällaisista ongelmista ovat erilai-

set aikataulutus- ja joukkojen järjestämistehtävät, jotka useissa tilanteissa voidaan ratkaista tehokkaasti geneettisten algoritmien avulla.

3.2. Toiminta

Geneettisten algoritmien toiminta perustuu yksinkertaistettuun käsitykseen evoluution toiminnasta luonnossa: hyvin ympäristöön sopeutuvat yksilöt pääsevät suuremmalla todennäköisyydellä siirtämään ominaisuutensa seuraavalle sukupolvelle. Tämä perustuu siihen, että jälkeläiset ovat aina suurelta osin samankaltaisia kuin vanhempansa. Variaatiota syntyy, koska (suvullisessa) lisääntymisessä risteytyksen kautta jälkeläinen saa ominaisuuksia kahdelta vanhemmaltaan. Geneettisissä algoritmeissa on kuitenkin Abbottin [2005] mukaan perustavanlaatuinen ero verrattuna esikuvaansa evoluutioon. Luonnossa yksilön hyvyyttä evoluution kannalta mitataan ainoastaan sillä, miten hyvin se sopeutuu ympäristöönsä ja miten hyvät mahdollisuudet sillä näin ollen on välittää ominaisuuksiaan seuraavalle sukupolvelle. Geneettisissä algoritmeissa sen sijaan valitaan risteyttämiseen käytettävät yksilöt määritellyn hyvyysfunktion mukaan. Luonnossa siis yksilön sopivuus ympäristöön mitataan sen kykynä lisääntyä, kun vastaavasti geneettisissä algoritmeissa lisääntymään valitaan yksilöt jonkin ominaisuuden hyvyyden perusteella.

Geneettisten algoritmien perustoiminta etenee seuraavasti:

1. Luodaan satunnainen populaatio ratkaisuvaihtoehtoja.
2. Arvioidaan kunkin ratkaisun hyvyys jonkin funktion avulla.
3. Valitaan populaation parhaimmat ratkaisut ja luodaan uusi populaatio niitä risteyttämällä.
4. Lisätään satunnaisuutta mutaatioiden avulla.
5. Palataan takaisin kohtaan 2 ja suoritetaan uusi kierros.

Yksinkertaisimpaan geneettisen algoritmin toimintamalliin liittyy monia ongelmia ja rajoituksia, joita on pyritty korjaamaan eritavoin. Esimerkiksi yksinkertaisimmat mutaation ja risteyttämisen muodot eivät ole monien ongelmien käsittelyssä toteutettavissa sellaisenaan. Geneettisten algoritmien yleisestä toimintaperiaatteesta onkin olemassa monia muunnoksia, joilla pyritään parantamaan algoritmien kykyä ratkaista tietyntyyppisiä ongelmia.

3.2.1. Alkioiden rakenne

Yksinkertaisin alkioiden esitysmuoto on binäärinen yksittäisistä biteistä koostuva jono [Shapiro, 2001]. Tämä siis tarjoaa yksittäiselle geenille kaksi vaihtoehtoista tilaa – 1 ja 0. Käytössä on kuitenkin myös suurempia lukujärjestelmiä, jotka siis tarjoavat yksittäiselle geenille useampia vaihtoehtoisia tiloja. Alkioiden

kuvaamiseen käytetään joissakin tilanteissa jopa reaalilukuja. Tavallisesti alkiot ovat vakiomittaisia. On kuitenkin toteutettu myös algoritmeja, jotka käyttävät vaihtelevan mittaisia alkioita. Tämä lisää algoritmin toiminnan mahdollisuuksia, mutta myös vaikeuttaa algoritmin toiminnan määrittelyä.

Goldbergin [1989] mukaan ongelma tulisi muotoilla siten, että eri kromosomien geenejä yhdistelemällä päästään lähemmäksi lopullista ratkaisua. Tähän perustuu *rakennuspalikkahypoteesi* (building block hypothesis), jonka mukaan geneettiset algoritmit toimivat, koska geenien määrittämillä hyvillä ominaisuuksilla on taipumus kerääntyä populaation hyviin alkioihin. Alkion rakennetta määriteltäessä ovat yksittäisten geenien riippuvuussuhteet tärkeä tekijä koko algoritmin toimintaa suunniteltaessa. Helpoin tilanne on, että arvot ovat toisistaan täysin riippumattomia, jolloin kaikki muodostetut yhdistelmät ovat kelvollisia alkioita. Todellisuudessa tätä tavoitetta ei normaalisti saavuteta, mikä aiheuttaa vaatimuksia risteytys- ja mutaatio-operaatioiden toiminnan määrittelylle.

3.2.2. Populaatio

Geneettisten algoritmien toiminta perustuu alkioista koostuvan joukon käsittelyyn. Populaation muokkaamiseen käytettävät perusoperaatiot ovat valinta, risteytys ja mutaatio. Tavoitteena on rakentaa algoritmi siten, että ratkaisut, joista tämä populaatio koostuu, muuttuisivat algoritmin ajon kuluessa hyvyysfunktion määrittelemällä tavalla paremmiksi. Populaation käsittelyä jatketaan, kunnes jokin lopetusehto täyttyy [Shapiro, 2001]. Tavallisesti tämä tarkoittaa sitä, että ongelmaan löytyy riittävän hyvä ratkaisu, joka täyttää määritellyn ongelman vaatimukset. Algoritmin suorituksen syklien määrä voidaan myös määrittää vakioksi siten, että suoritusta jatketaan määritelty aika riippumatta saavutettujen ratkaisujen hyvyydestä.

Tavallisesti populaation koko on määritelty kiinteäksi, joten se säilyy algoritmin suorituksen aikana vakiona [Mitchell, 1997]. Näin ei kuitenkaan tarvitse olla kaikissa tapauksissa, vaan ratkaistavana olevasta ongelmasta riippuen saattaa joissakin tilanteissa olla asianmukaista rakentaa algoritmi tavalla, joka saa aikaan populaation koon ajonaikaista vaihtelua.

Algoritmin tehokkaan toiminnan kannalta on tärkeää, että populaatiossa säilyy koko suorituksen ajan riittävä alkioiden diversiteetti [Holland, 1992]. Jos tämä vaatimus ei täyty, jää uusien ratkaisujen löytyminen mutaatioiden varaan, mikä on hidasta. Tämän voi pyrkiä ratkaisemaan mutaatioita lisäämällä, mutta runsaassa mutaatioiden käytössä on myös omat ongelmansa. Liiallinen mutaatioiden käyttö saattaa hävittää jo saavutettuja hyviä ominaisuuksia [Holland, 1992]. Tämä vaara ei kuitenkaan ole merkittävä tilanteessa, jossa tietty ratkaisu on vallannut koko populaation, koska tästä ratkaisusta on monia esiintymiä. Eli

tilanne, jossa ne kaikki häviäisivät yhden suorituskierroksen aikana, vaatisi mutaation tapahtumista jokaiseen populaation alkioon. Näin runsas mutaatioiden käyttö kuitenkin tekee algoritmin toiminnasta liian satunnaista, jolloin se tuskin on enää tehokas minkään ongelman ratkaisussa.

Alkupopulaatio tulee aina pyrkiä määrittelemään siten, että se sisältää paljon alkioiden välistä erilaisuutta [Shapiro, 2001]. Tämä siksi, että algoritmin toiminta jatkuisi tehokkaana mahdollisimman pitkään ja ratkaisuissa tapahtuva variointi olisi voimakasta, kunnes päästään lähelle optimaalista ratkaisua. Usein aloituspopulaatio luodaan käyttäen satunnaisesti luotuja alkioita [Shapiro, 2001]. Jos aloituspopulaation luontivaiheessa on tiedossa alkioiden ominaisuuksia, jotka saattavat olla ongelman ratkaisun kannalta edullisia, saattaa näiden lisääminen populaatioon auttaa ratkaisun löytymistä. Mutta pyrkimys ohjata algoritmin toimintaa johonkin suuntaan saattaa toisaalta rajoittaa löydettävissä olevien ratkaisujen määrää.

3.2.3. Hyvyysfunktio ja valinta

Alkion saama hyvyysfunktion arvo ei geneettisten algoritmien perustoiminnassa suoraan määritä, tuleeko alkio valituksi mukaan seuraavan populaation muodostamiseen vai ei. Hyvyysfunktio määrittää vain todennäköisyyden sille, että paremmat ratkaisut tulevat valituksi suuremmalla todennäköisyydellä ja huonojen ominaisuuksien määrä populaatiossa näin vähenee [Holland, 1992]. Tällä pyritään säilyttämään populaatiossa riittävää diversiteettiä, joka on tärkeää geneettisen algoritmin tehokkaalle toiminnalle. Huonossakin ratkaisussa, joka ei siis sellaisenaan sovellu ongelman ratkaisuksi, saattaa olla jokin ominaisuus, jota kehittämällä on mahdollisuus luoda ongelmaan sopiva ratkaisu. Tähän sisältyy kuitenkin myös se mahdollisuus, että suorituksessa menetetään myös hyviä ratkaisuja. Tämän takia joissakin tilanteissa on mahdollista määritellä algoritmin toiminta siten, että parhaimmat ratkaisut pääsevät aina mukaan uuden sukupolven muodostamiseen. Tätä perustoiminnan muunnosta kutsutaan elitismiksi [Shapiro, 2001].

Hyvyysfunktion arvon tyyppi voidaan valita monella tavalla [Holland, 1992]. On mahdollista käyttää esimerkiksi täysin sovelluskohderiippuvaista arvoa, jolloin pelkän arvon perusteella on mahdoton päätellä alkion hyvyyttä. Yleisempää on kuitenkin käyttää jotakin tekniikkaa, jolla muodostetaan yleisemmin säännönmukainen mitta-asteikko. Arvo voidaan esimerkiksi standardisoida siten, että pienempi arvo ilmaisee parempaa ratkaisua. Algoritmin toiminnan seurauksena saattaa syntyä tilanne, jolloin ratkaisujen hyvyysfunktion arvot ovat hyvin samankaltaisia. Tällöin valinta ei enää toimi optimaalisella tavalla, jos käytetään absoluuttisia arvoja. Tämä tilanne voidaan korjata käyttä-

mällä skaalattuja hyvyysfunktion arvoja, jolloin alkioiden väliset erot korostuvat [Shapiro, 2001].

Populaation ennenaikainen yhdenmukaistuminen on ongelma, joka rajoittaa löydettyjen ratkaisujen määrää [Shapiro, 2001]. Tämä ehkäisee vaihtoehtojen riittävän kattavan läpikäymisen, jonka seurauksena hyvän ratkaisun löytymisestä tulee epävarmaa. Populaation ennenaikaista yhdenmukaistumista voidaan ehkäistä esimerkiksi laskemalla hyvyysfunktion lisäksi arvo, joka ilmaisee ratkaisun ominaisuuksien yleisyyttä populaatiossa [Goldberg, 1989]. Tämän arvon perusteella lisätään sellaisten alkioiden valituksi tulemisen todennäköisyyttä, joiden ominaisuudet ovat populaatiossa vähemmän yleisiä.

Yleisin käytetty valintamekanismi on niin kutsuttu *rulettivalinta* (roulette wheel selection, proportional fitness selection), joka perustuu alkion suhteelliseen hyvyyteen [Mitchell, 1997]. Rulettivalinnassa rulettipyörän yksittäinen sektori edustaa yhtä populaatiossa olevaa alkiota. Sektorin koko määräytyy alkion saaman hyvyysfunktion arvon mukaan.

Yksi perusvalinnan muunnoksista on *turnausvalinta* (tournament selection) [Mitchell, 1997]. Tästä menetelmästä on olemassa monia muunnoksia, mutta perustoiminnassa valitaan populaatiosta joukko alkioita, joita kilpailutetaan keskenään ja tämän tuloksen perusteella valitaan parhaat. Tämä valintamekanismi on lähimpänä luonnon mallia, jossa tietyt yksilöt usein kilpailevat mahdollisuudestaan päästä lisääntymään. Yksinkertainen turnausvalinnan muoto on valita populaatiosta kaksi alkiota, joita kilpailutetaan ja parempi korvaa populaatiossa huonomman [Shapiro, 2001]. Turnausvalinnalle on ominaista, että sitä käytettäessä ei tarvitse laskea hyvyysfunktion arvoa kaikille alkioille, vaan pelkästään valituille.

Eräs valinnan muoto on *sijalukuvalinta* (rank selection), jossa muodostetaan ratkaisujen hyvyyden järjestyksen määrittävät arvot [Mitchell, 1997]. Alkioiden valinta suoritetaan sitten tämän arvon mukaan sen sijaan, että käytettäisiin suoraan hyvyysfunktion arvoa. Tämä on hyödyllistä erityisesti tilanteissa, joissa hyvyysfunktion arvot ovat lähestyneet voimakkaasti toisiaan.

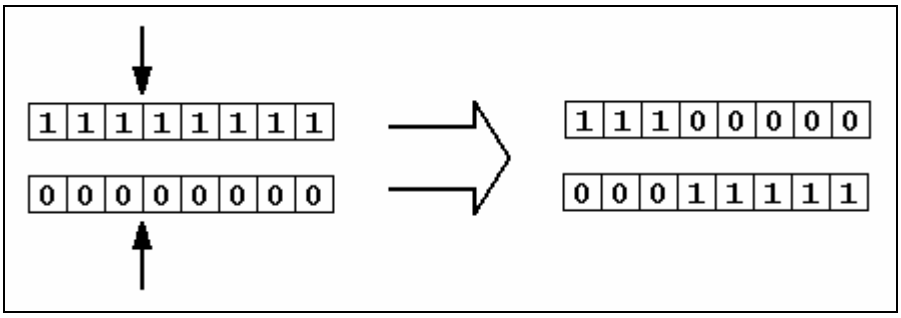
Vakaan tilan valinnassa (steady-state selection), johon liittyy myös *mating pool* -käsite, ei muuteta koko populaatiota kerralla, vaan muutosoperaatiot suoritetaan vain muutamalle alkiolle kierroksen aikana [Mitchell, 1997]. Näin saadut uudet alkiot palautetaan takaisin vanhaan populaatioon. Tämänkaltainen algoritmi on hitaasti kehittyvä, mutta hyvien ratkaisujen häviämisen riski on pieni. Algoritmi on mahdollista toteuttaa myös siten, että populaatio koostuu useammasta osapopulaatiosta, joita kehitetään itsenäisesti [Shapiro, 2001]. Näin toteutetut populaation osajoukot kehittyvät suurella todennäköisyydellä kohti

eri optimeita. Kehityksen hidastuessa siirretään osapopulaatioista alkioita toisiinsa, minkä tavoitteena on saada kehitys jatkumaan.

Geneettisen algoritmin perusmuodossa jokainen alkio säilyy täsmälleen yhden sukupolven ajan. Tämä ei kuitenkaan päde kaikkiin edellä esiteltyihin valinnan muotoihin. Populaation uusiutuminen voidaankin ilmaista arvolla, joka ilmaisee populaation uusiutumisen nopeutta [Mitchell, 1997]. Normaalilla geneettisellä algoritmilla tämä arvo on 1, koska koko populaatio uudistuu jokaisella kierroksella. Vastaavasti sellaiselle vakaan tilan algoritmille, jossa yhdellä kierroksella korvataan vain yksi alkio, tämä arvo on: 1/(populaation koko).

3.2.4. Risteyttäminen

Yksinkertaisin risteytystapa on risteyttää kaksi alkiota yhden pisteen mukaan. Risteytys operaatio alkaa valitsemalla piste, jonka mukaan geenijonot katkaistaan. Molemmat risteytettävistä geenijonoista katkaistaan tästä samasta määritellystä kohdasta. Tämän jälkeen vaihdetaan osia keskenään siten, että muodostetuissa alkioissa on osa kummastakin vanhemmasta. Risteytyksen toiminta on havainnollistettu kuvassa 1. Yhden pisteen risteytys on yksinkertaisin risteytystapa, mutta siihen liittyy kuitenkin joitakin ongelmia. Sen toimivuus on voimakkaasti sidoksissa geenien järjestykseen. Esimerkiksi vierekkäiset geenit säilyvät suurella todennäköisyydellä samoina ja vastakkaisissa päissä sijaitsevat geenit taas eivät koskaan säily yhdessä [Mitchell, 1997]. Tämän ongelman vaikutuksia voidaan kuitenkin kiertää, jos ongelman rakenne tunnetaan riittävän hyvin.



Kuva 1 Yhden pisteen risteytys

Piste tai pisteet, joiden mukaan risteyttäminen tapahtuu, voidaan määritellä joko kiinteästi tai satunnaisuuteen perustuen. Satunnaisuuden käyttö aiheuttaa lisähaasteita alkioiden esitystavan valinnalle. Risteytysoperaatioiden tulisi siis tuottaa uusia alkioita, jotka ovat muodollisesti kelvollisia ongelman ratkaisuja. Risteytyksen toteuttamiselle voidaan myös määritellä todennäköisyys [Shapiro, 2001]. Tällöin on mahdollista, että valitut alkiot lisätään seuraavaan populaatioon sellaisenaan risteyttämättä. Tämänkaltaisen todennäköisyyden määrittely

hidastaa populaation kehittymistä, mutta ehkäisee hyvien ratkaisujen häviämistä.

Perusristeytystavasta on olemassa monia muunnoksia. Esimerkiksi risteytyksessä voidaan risteyttää samalla kertaa useampia alkioita tai leikkauspisteitä voi olla useita. On jopa mahdollista valita geenit yksittäin risteytettävänä olevista alkioista. Valittavan risteytystavan määrää ensisijaisesti alkioiden rakenne. Esimerkiksi risteyttämistä yksittäisten geenien mukaan voi menestyksellä käyttää vain tilanteissa, joissa geenien välillä ei ole riippuvuutta tai riippuvuus on heikko. Risteytysoperaatiota määriteltäessä on kahden vanhemman käyttö luonnollisten rajoitusten puuttuessa täysin mielivaltaista. Yhtä hyvin voidaan käyttää myös jotakin muuta risteytettävien alkioiden määrää sen mukaan, kuinka tehokas se on ongelman ratkaisemisen kannalta.

3.2.5. Mutaatio

Mutaation avulla lisätään algoritmin toimintaan satunnaisuutta, joka tekee mahdolliseksi uusien ratkaisujen syntymisen. Kohdistettaessa alkioon mutaatio-operaatio muutetaan alkion satunnaisesti valitun geenin arvoa. Mutaatioiden vaikutuksen runsaus ilmaistaan arvolla, joka kertoo, millä todennäköisyydellä alkion yksittäinen geeni tulee altistetuksi mutaatiolle [Shapiro, 2001].

Mutaatioiden käytössä on tärkeää mutaatioiden sopiva määrä hyvien ratkaisujen löytämisen kannalta. Jos mutaatioita on liian vähän, saattavat ratkaisuvaihtoehdot vakiintua ennenaikaisesti johonkin paikalliseen arvoon. Vastaavasti jos mutaatioiden käyttö on liian voimakasta, on vaarana, että menetetään jo saavutettu hyvä ratkaisu. [Holland, 1992]

Mutaatioiden käytön hallintaan on olemassa monia mahdollisuuksia. On esimerkiksi mahdollista, että kaikkia alkioita ei altisteta mutaatioille vaan parhaimpiin ratkaisuihin kohdistetaan vain risteytysoperaatioita. Mutaatioiden määrän ei myöskään tarvitse olla vakio läpi koko algoritmin suorituksen, vaan niiden vaikutusta voidaan vahvistaa tilanteissa, joissa ratkaisujen kehittyminen näyttää pysähtyneen johonkin paikalliseen arvoon.

3.3. Tehokkuus

Mitchell [1997] esittää, että geneettiset algoritmit eivät ole erityisen nopeita varsinkaan yksinkertaisten ongelmien ratkaisussa. Hänen mukaansa geneettisille algoritmeille on kuitenkin ominaista, että ne löytävät monissa sovelluksissa suhteellisen nopeasti hyviä ratkaisuja, jos hakuavaruus on erityisen laaja ja monimutkainen. Ne sopivatkin siis usein sellaisten ongelmien ratkaisuun, joihin on vaikea löytää ratkaisua suoraviivaisemmilla menetelmillä. Toisaalta kaikkein parhaimman ratkaisun löytämiseen, jopa todella pienestä alkiojoukosta, ovat geneettiset algoritmit usein tehottomia [Grefenstette, 1993]. Tämän takia

geneettisiä algoritmeja Grefenstetten [1993] mukaan käytetäänkin usein yhdessä muiden hakualgoritmien kanssa. Esimerkiksi käytetään geneettistä algoritmia hyvän likiarvon löytämiseen, minkä jälkeen sovelletaan jotakin muuta hakumenetelmää, jolla pyritään optimoimaan geneettisen algoritmin löytämä ratkaisu.

4. Geneettisten algoritmien käyttö koneoppimisessa

Vaikka monet geneettisten algoritmien sovellukset löytyvät funktion optimoinnista, parametrien säädöstä, jaksottamisesta ja muista kombinatorisista ongelmista, on geneettisiä algoritmeja sovellettu myös perinteisiin koneoppimisen ongelmiin kuten alkioiden luokitteluun esimerkkien perusteella (concept learning), neuroverkkojen kytkentöjen painoarvojen oppimiseen ja peräkkäisyys sääntöjen muodostamiseen [Grefenstette, 1993].

No free lunch -teoreeman mukaan mitkä tahansa hakualgoritmit ovat tehokkuudeltaan yhtä hyviä, jos tehokkuus ilmaistaan kaikkien mahdollisten ongelmien ratkaisemisen kokonaistehokkuutena [Wolpert and Macready, 1995]. Käytännössä tämä tarkoittaa, että mikään algoritmi ei sovellu kaikenlaisten oppimisongelmien ratkaisemiseen, ja tämä pätee luonnollisesti myös geneettisiin algoritmeihin. Parhaan mahdollisen tehokkuuden saavuttamiseksi on siis tärkeää valita algoritmi, joka sopii hyvin juuri käsiteltävänä olevan ongelman ratkaisemiseen. Seuraavassa pyritään selvittämään, mitkä ovat nämä geneettisten algoritmien vahvat alueet koneoppimisessa.

4.1. Ominaispiirteitä

Shapiron[2001] mukaan on löydettävissä kolme keskeistä syytä, jotka tekevät geneettisistä algoritmeista tärkeän koneoppimisen menetelmän. Ensinnäkin niitä voidaan käyttää epäjatkuvien funktioiden ratkaisemiseen. Toiseksi geneettisiä algoritmeja on rakenteensa puolesta suoraviivaista hyödyntää vahvistusoppimismenetelmänä. Geneettiset algoritmit soveltuvatkin erityisen hyvin käytettäviksi tilanteissa, jossa ratkaisujen hyvyys on riittävä palaute. Viimeiseksi geneettiset algoritmit muodostavat ratkaisujen joukon, mikä saattaa joissakin tilanteissa olla hyödyllisempää kuin yksittäisen hyvän ratkaisun käyttö. Esimerkki tällaisesta on erilaiset agenttijärjestelmät.

Geneettisiä algoritmeja voidaan käyttää koneoppimisessa kahdella erilaisella tavalla. Ensimmäinen käyttötapa on pyrkiä tuottamaan geneettisellä algoritmilla joukko hypoteeseja, joista jokainen on itsessään kokonainen ratkaisu [DeJong et al., 1993]. Toinen vaihtoehto on muodostaa joukko yksittäisiä sääntöjä, jotka yhdessä muodostavat ratkaisun [Holland, 1992].

Geneettiset algoritmit toteuttavat koneoppimisen samoin kuin hakualgoritmit yleisemminkin – etsimällä hakuavaruudesta ratkaisuvaihtoehtoja määriteltyyn ongelmaan [Shapiro, 2001]. Koneoppimisessa käytetyt menetelmät voidaan jakaa sen mukaan, kuinka hallittua tai vastaavasti hallitsematonta niiden toiminta on. Tässä jaottelussa geneettiset algoritmit kuuluvat algoritmien hallitsemattomaan ja eräällä tavalla tyhmään päähän, jonka toiminta perustuu vähiten tietoon ja eniten puhtaaseen hakuun, joka siis etenee yritysten ja erehdysten kautta [Shapiro, 2001]. Käytännön sovellutuksissa geneettisiä algoritmeja yhdistellään usein muihin hakualgoritmeihin, jolloin saadaan luotua tehokkaita hybridihakualgoritmeja [Grefenstette, 1993]. Näin saadaan käyttöön useamman tekniikan vahvuudet ja järjestelmästä on mahdollista toteuttaa tehokkaampi.

4.2. Sovelluskohteet

Seuraavassa käsitellään joitakin koneoppimisen sovelluksia, joissa geneettisiä algoritmeja on menestyksellisesti käytetty.

4.2.1. Päättelysääntöjoukkojen muodostaminen

Geneettisiä algoritmeja on mahdollista käyttää koneoppimisessa käytettävien sääntöjoukkojen muodostamiseen. Sääntöjen muodostaminen on mahdollista toteuttaa kolmen eri periaatteen mukaan, jotka ovat Michigan-periaate, Pittsburgh-periaate ja iteratiivinen periaate. Michigan-periaatteen mukaisessa tilanteessa ovat populaation alkiot yksittäisiä sääntöjä, Pittsburgh-periaatteen mukaan jokainen alkio on kokonainen sääntöjoukko ja iteratiivista periaatetta noudattavassa tilanteessa populaatio koostuu yksittäisistä säännöistä kuten Michigan-periaatteessa, mutta näistä säännöistä valitaan vain yksi. [DeCastro and Camargo, 2004]

De Castro ja Camargo [2004] ovat tutkineet geneettisten algoritmien käyttöä sumean logiikan mukaisten päättelysääntöjen muodostamisessa. Heidän järjestelmässään geneettisiä algoritmeja käytettiin kahteen erilliseen tehtävään. Ensin muodostettiin sääntöjen joukko, jota tämän jälkeen pyrittiin optimoimaan poistamalla siitä samankaltaisuudet.

Esimerkkinä päättelysääntöjoukkojen muodostamiseen käytetystä geneettisestä algoritmista toimii GABIL [DeJong et al., 1993]. GABIL-algoritmin käsittelemät alkiot ovat päättelysääntöjä, joita voidaan käyttää esimerkkien luokitteluun. GABIL-algoritmissa alkioiden rakenne on toteutettu siten, että yksittäinen geeni edustaa yhden muuttujan yhtä mahdollista arvoa. Ykkönen edustaa geenin tilana tapausta, jossa muuttuja voi saada kyseisen arvon ja nolla vastaavasti tilannetta jossa muuttuja ei voi kyseistä arvoa saavuttaa. Tämän lisäksi yksittäiseen päättelysääntöön sisältyy geeni, joka määrittää, kuuluuko luokiteltava esimerkki säännön toteutuessa luokkaan vai suljetaanko se luokan ulkopuolel-

le. Algoritmille ominaista on risteytysoperaatio, joka pystyy tuottamaan alkioita, jotka muodostuvat vaihtelevan mittaisista bittijonoista. Tämä saa aikaan sen, että kokonainen sääntö voi koostua vaihtelevasta määrästä osasääntöjä.

4.2.2. Agenttijärjestelmät

Agenttijärjestelmä (agent system) on kokonaisuus, joka sisältää agentin tai agentteja, jotka toimivat vuorovaikutuksessa ympäristönsä kanssa. Agentin määritelmään kuuluu, että se aistii ympäristöään ja reagoi sen muutoksiin. Lisäksi agentin toiminnan tulee olla joidenkin määriteltyjen tavoitteiden ohjaamaa [Russell and Norvig, 1995]. Tämä sulkee siis pois puhtaasti satunnaisuuteen perustuvat järjestelmät.

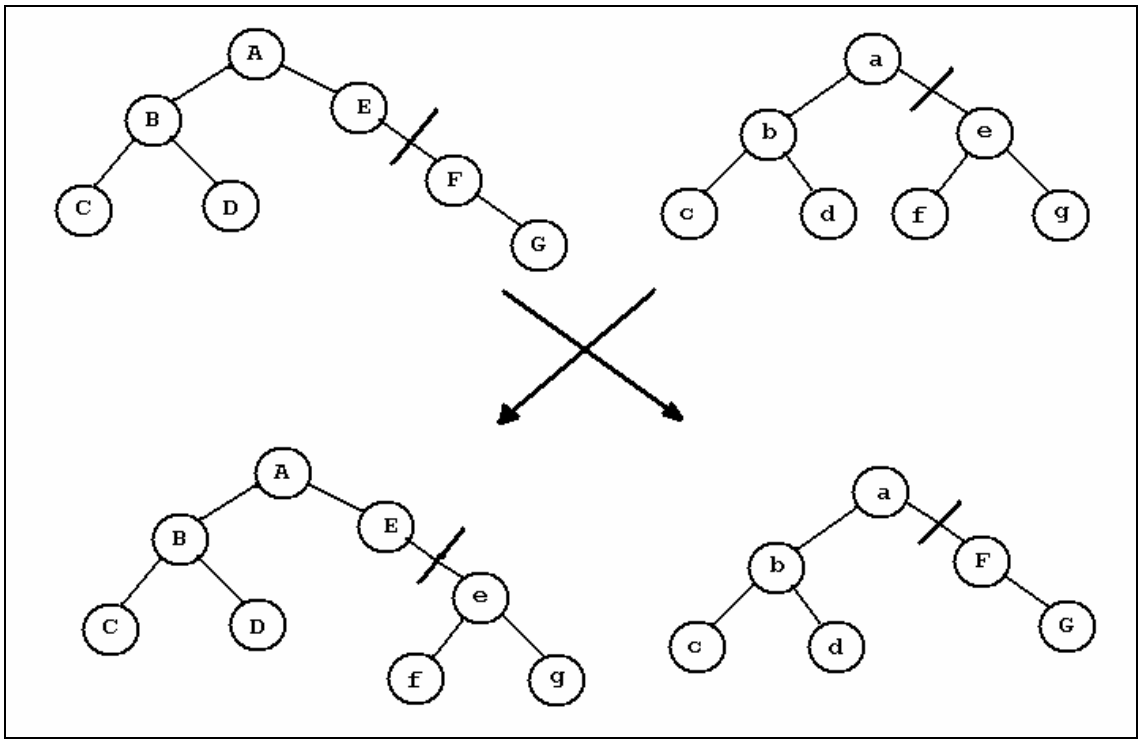
Geneettisiä algoritmeja voi agenttien toteuttamisessa hyödyntää kahdella tavalla. Ensinnäkin voidaan agenteista muodostaa populaatio, jonka kehittymistä ohjataan geneettisellä algoritmilla. Tätä periaatetta käyttää esimerkiksi AGA (Agent-Based Genetic Algorithm) [Wang et al., 2005]. Toinen käyttömahdollisuus on luoda toimintasääntöjen populaatio, joka määrittelee yksittäisen agentin toiminnan erilaisissa tilanteissa. Tästä on eräänä esimerkkinä SAMUEL-oppimisjärjestelmä, jonka yksi sovellus on itsenäisesti liikkuvat ajoneuvot [Grefenstette et al., 1990]. Määritelty algoritmi tuottaa siis joko valmiita agentteja tai toimintasääntöjä, joiden perusteella agentin toiminta määritellään. Näiden eri tekniikoiden välillä on se ero, että populaation koostuessa agenteista eivät nämä agentit ole tavallisesti oppivia agentteja, vaikka järjestelmä kokonaisuutena sitä onkin. Vastaavasti muokattaessa agentin toimintasääntöjä on kyseessä oppiva agentti. Geneettisten algoritmien on havaittu olevan tehokas tapa hallita strategioiden oppimista, jotka ovat toimivia muiden agenttien toiminnan säännönmukaisuuksien hyödyntämisessä [Grefenstette, 1993].

4.2.3. Geneettinen ohjelmointi

Geneettinen ohjelmointi on geneettisten algoritmien käytön erikoistapaus. Geneettisessä ohjelmoinnissa käsiteltävä populaatio muodostuu tietokoneohjelmista, joiden tehokkuutta pyritään maksimoimaan. Yksittäinen alkion esitysmuoto geneettisessä ohjelmoinnissa on tavallisesti puurakenne. Tämä alkioiden normaalista jonomuodosta poikkeava rakenne aiheuttaa luonnollisesti omat vaatimuksensa mutaatio- ja risteytysoperaatioiden määrittelylle.

Käytetty puurakenne on rakenteeltaan sellainen, että puun lehtisolmut eivät käsittele syötteitä. Muut solmut vastaavasti käsittelevät oman alipuunsa solmuja (sekä lehtisolmuja että sisäsolmuja) [Alander, 2006]. Tällä saavutetaan tilanne, jossa solmujen väliset riippuvuussuhteet ovat puussa suuntautuneet aina juuresta kohti lehtisolmuja. Tämän perusteella on risteytysoperaatio toteutettu siten, että risteytettävistä puista valitaan satunnaiset toisistaan riippumattomat

leikkauskohdat, joiden mukaan alipuut vaihdetaan keskenään. Tämä on havainnollistettu kuvassa 2.



Kuva 2 Puumuotoisten alkioiden risteytys

Schmidhuber [1987] on esittänyt tekniikan käyttöä, jota voi kutsua geneettiseksi metaohjelmoinniksi. Ajatuksena on, että sopivan ohjelman muodostamista ohjaava geneettinen algoritmi olisi myös itse altistettu geneettisen algoritmin ohjaamalle valinnalle. Tämän tekniikan mukanaan tuomat hyödyt ja mahdollisuudet ovat vielä toistaiseksi kiistanalaisia.

4.2.4. Geneettiset algoritmit ja neuroverkot

Neuroverkot on suosittu koneoppimismenetelmä, jonka toiminta yksinkertaistettuna perustuu siihen, että järjestelmän toimintaa ohjaavan verkon rakenne pyritään muodostamaan sellaiseksi, että tietty syöte antaa halutunlaisen vasteen. Tämä verkko rakentuu neuroneista ja niitä yhdistävistä synapseista. Neuroverkon toiminta ei perustu määriteltyihin sääntöihin vaan synapseille määriteltyihin painoarvoihin. Neuroverkkoja käytetään tavallisesti ohjatussa oppimisessa, jossa verkonrakenne muodostetaan halutunlaiseksi esimerkkisyötteiden avulla.

Geneettisiä algoritmeja käytetään neuroverkkojen yhteydessä kahteen tarkoitukseen [Shapiro, 2001]. Niiden avulla on ensinnäkin mahdollista ohjata varsinaista verkon muodostamista. Käytännössä tämä tarkoittaa sitä, että neuroverkon opetusvaihe toteutetaan geneettisellä algoritmilla. Tässä tehtävässä geneettinen algoritmi on Shapiron [2001] mukaan tavallisesti tehottomampi kuin

perinteisemmät optimointi menetelmät. Tässäkin geneettisten algoritmien etuna on kuitenkin se, että algoritmin toiminta ei ole rajoittunut paikallisiin ääriarvoihin.

Toisena käyttökohteena on olemassa olevan verkon optimointi. Tämä voi perustua pelkästään mutaatioihin, mutta myös risteytysten käyttö on mahdollista. Käytettäessä risteytyksiä joudutaan kuitenkin suunnittelemaan yksittäisen verkon esitysmuoto sellaiseksi, että operaation käyttö on mahdollista. Yleisesti risteytysoperaatio on vaikeaa suunnitella sellaiseksi, että se todella tuottaa toimivia ratkaisuja [Shapiro, 2001].

4.2.5. Tekoelämä ja emergenssi

Tekoelämän (artificial life) tutkimus on poikkitieteellinen tutkimusala, jossa pyritään luomaan elollisen kaltaisia keinotekoisia järjestelmiä. Tekoelämän tutkimuksen tavoitteena on tuottaa tietoa elävien organismien toiminnasta ja niiden vuorovaikutuksen rakenteesta sekä vastavuoroisesti hyödyntää näitä periaatteita tekniikassa [Dictionary, 2008]. Tietotekniikan avulla toteutetut elämää simuloivat järjestelmät ovat yleisin esimerkki tekoelämän alaan kuuluvista järjestelmistä. Näiden lisäksi tekoelämää on toteutettu robottijärjestelmillä ja biokemiallisilla järjestelmillä. Kuitenkin myös robottijärjestelmissä on tärkeimmässä roolissa järjestelmää ohjaavat algoritmit. Tekoelämäjärjestelmät eivät välttämättä ole oppivia järjestelmiä, ja usein ne toimivatkin vain valmiiden sääntöjen mukaan, mutta näiden alueiden välillä on kuitenkin selvä yhteys. Jotta pystyttäisiin luomaan yksiköitä, jotka pyrkivät mukautumaan ja reagoivat ympäristöön muutenkin kuin valmiiden ärsyke-vaste -parien mukaan, täytyy järjestelmällä olla kyky opiskella ympäristöään ja muodostaa sen toiminnasta ja rakenteesta yleistäviä sääntöjä. Geneettisten algoritmien toimintaa epädeterministisenä tekniikkana on vaikea ennustaa. Tämän seurauksena geneettisten algoritmien voi ajatella toimivan itsenäisesti niille määriteltyjen sääntöjen ja rakenteen puitteissa, mikä tekee niistä paljon käytetyn tekoelämän toteuttamisen tekniikan.

Emergenssi (emergence) on kiinteästi tekoelämään liittyvä käsite, jonka yleisenä ajatuksena on jonkin kokonaisuuden muodostuminen yksinkertaisemmista osista. Jotta kyseessä olisi emergenssi, täytyy näin muodostuneella kokonaisuudella olla, emergenssin määritelmän mukaisesti, jokin ominaisuus tai ominaisuuksia, joita sen osilla ei ole. Tekoelämään käsite liittyy siten, että se ilmaisee alkioiden vuorovaikutuksen tuloksena syntyvän jotakin, joka ei jo alun perin sisälly käytetyn algoritmin ja syötteiden määrittelyyn. Abbottin [2005] mukaan asian tarkasteluun liittyviä ongelmia on esimerkiksi, miten algoritmi tulisi määritellä, jotta sen kykyä emergenssiin olisi rajoitettu mahdollisimman vähän. Tietojenkäsittelytieteissä on toteutettu monia järjestelmiä, jotka kykenevät emergenssiin. Emergenssin hyödyntäminen on kuitenkin hyvin rajoittunutta,

koska esimerkiksi ennustamattoman emergenssin seurauksena syntyneiden kokonaisuuksien välistä vuorovaikutusta ei ole onnistuttu toteuttamaan [Abbott, 2005].

4.3. Genotyyppi ja fenotyyppi

Biologiassa erotetaan toisistaan yksilön *genotyyppi* (genotype) ja *fenotyyppi* (phenotype). Genotyyppi ilmaisee yksilön piilevän potentiaalin siinä missä fenotyyppi on todella aktualisoitunut yksilö [Shapiro, 2001]. Tämä merkitsee sitä, että luonnossa yksilöllä voi olla geneettinen taipumus toteuttaa itsessään jokin ominaisuus, mutta ympäristö rajoittaa yksilöä siten, että tämä ominaisuus ei ikinä täyty. Vaikka tässä on ennen kaikkea kyse rajoittavasta tekijästä, on tämä myös eräänlainen mukautumisstrategia; yksilöllä on enemmän potentiaalia kuin mitä yhdessä ympäristön tilassa tarvitaan tai on mahdollista toteuttaa.

Biologisilla organismeilla genotyypin ja fenotyypin välinen ero on usein huomattava, mutta Shapiron [2001] mukaan geneettisillä algoritmeilla ero ei tavallisesti kuitenkaan ole yhtä merkittävä. Tämä pätee erityisesti silloin, kun rajoitutaan tarkastelemaan yksittäisiä algoritmin muodostamia alkioita ja määritellään yksittäisen alkion muoto sen genotyypiksi. Asiaa voidaan lähestyä kuitenkin myös toisin, jolloin lähimpänä genotyypin määritelmää voidaan ajatella olevan algoritmin hyvyysfunktio, joka implisiittisesti määrittelee, millaisia ratkaisujen tulisi olla [Shapiro, 2001]. Tällöin vastaavasti syntyneet ratkaisut, jotka pyrkivät täyttämään tämän määritelmän, ovat fenotyyppejä. Kaikilla populaation alkioilla on siis yhteinen genotyyppi, jonka ne onnistuvat eriasteisesti toteuttamaan.

Jos kiinnitetään tarkemmin huomiota siihen, miten hyvyysfunktion ymmärtäminen genotyypiksi vaikuttaa käsitykseen geneettisen algoritmin kyvystä sopeutua dynaamiseen ympäristöön, nousee esiin tärkeä huomio: Jos ongelma, johon algoritmi etsii ratkaisua, muuttuu, täytyy myös algoritmin hyvyysfunktion muuttua, jotta tuotettaisiin ratkaisuja oikeaan ongelmaan. Vastaavasti myös populaation alkioiden tulisi olla rakenteeltaan sellaisia, että niiden on mahdollista tuottaa toimivia ratkaisuja muuttuneeseen ongelmaan. Tämä tuo esille sen, että eräs geneettisten algoritmien käytön olennainen rajoitus on se, että sen rajatun osan toiminta ja rakenne on sidottu toisiin algoritmin osien toimintoihin ja rakenteisiin. Jotta algoritmi saavuttaisi optimaalisen mukautumiskyvyn, täytyisi tämän toteutua myös sen erillisissä osissa.

5. Yhteenveto

Geneettisiä algoritmeja on mahdollista soveltaa moniin koneoppimisen ongelmiin. Niillä, kuten kaikilla muillakin koneoppimisen menetelmillä, on vahvuu-

tensa ja rajoituksensa, jotka tulee ottaa huomioon tekniikkaa sovellettaessa. Ratkaisun tehokkuuden kannalta onkin tärkeää, että ratkaisu on toteutettu ongelman asettamat rajoitukset huomioiden. Olen pyrkinyt tässä työssä tuomaan esiin näitä rakenteelle asetettuja vaatimuksia sekä esitellyt esimerkkejä siitä, millaisiin tehtäviin geneettiset algoritmit soveltuvat.

Geneettiset algoritmit koneoppimisessa on aihealueena laaja ja moniulotteinen alue, johon liittyviä ongelmia tutkitaan monilla eri aloilla. Tämän seurauksena kokonaiskuva aiheesta on pirstaloitunut. Parhaimman kehityksen saavuttaminen aihealueen tutkimuksessa vaatiikin todennäköisesti poikkitieteellistä lähestymistapaa, jonka pyrkimyksenä on purkaa kehityksen esteenä olevat teoreettiset ja tekniset rajoitukset. Koneoppimisen keskiössä voikin nähdä olevan yleiset tiedonmuodostuksen teoriaan liittyvät ongelmat. Ilman kehitystä tällä osa-alueella on vaikea odottaa läpimurtoja tapahtuvan myöskään koneoppimisessa.

Viiteluettelo

[Abbott, 2005] Russ Abbott, Challenges for biologically-inspired computing. In: *Proc. of the 2005 Workshops on Genetic and Evolutionary Computation*, 12-22.

[Alander, 2006] Jarmo T. Alander, Geneettisten Algoritmien Mahdollisuudet. Vaasan yliopisto, Tietotekniikan ja tuotantotalouden laitos, Raportti **97–4**, lokakuu 2006. Saatavana myös *ftp://ftp.uwasa.fi/cs/report97-4/Finnish.pdf*.

[DeCastro and Camargo, 2004] Pablo Alberto de Castro and Heloise A. Camargo, A study of the reasoning methods impact on genetic learning and optimization of fuzzy rules. In: *Brazilian Symposium on Artificial Intelligence No17*, 414-423.

[DeJong et al., 1993] Kenneth A. de Jong, William M. Spears and Diana F. Gordon, Using genetic algorithms for concept learning. *Machine Learning* **13**, 2-3 (Nov. 1993), 161-188.

[Dictionary, 2008] Dictionary.com, Artificial Life. http://dictionary.reference.com/browse/artificial%20life. Checked 2.12.2008.

[Goldberg, 1989] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[Grefenstette, 1993] John J. Grefenstette, Genetic algorithms and machine learning. In: *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, 3-4.

[Grefenstette et al., 1990] John J. Grefenstette, Connie Loggia Ramsey and Alan C. Schultz, Learning sequential decision rules using simulation models and competition. *Machine Learning* **5**, 4 (Oct. 1990), 355-381.

[Holland, 1992] John H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[Korb, 2004] Kevin B. Korb, Introduction: machine learning as philosophy of science. *Minds and Machines* **14**, 4 (Nov. 2004), 433-440.

[Mitchell, 1997] Tom Mitchell, *Machine Learning*. McGraw Hill, 1997.

[Russell and Norvig, 1995] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[Shapiro, 2001] Jonathan Shapiro, Genetic algorithms in machine learning. In: *Machine Learning and Its Applications*. Springer, 2001, 146-168.

[Schmidhuber, 1987] Jürgen Schmidhuber, Evolutionary principles in self-referential learning. Technische Universität München, Diploma Thesis, May 1987. Available as http://www.idsia.ch/~juergen/diploma.html. Checked 14.12.2008.

[Thagard, 1988] Paul R. Thagard, *Computational Philosophy of Science*, MIT Press, 1988.

[Wang et al., 2005] Honggang Wang, Jianchao Zeng and Yubin Xu, Design of the agent-based genetic algorithm. In: *Advances in Natural Computation,* Springer, 2005, 22-27.

[Wolpert and Macready, 1997] David H. Wolpert, William G. Macready, No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* **1**, 1 (Apr. 1997), 67-82.

Tietokannan samanaikaisuuden hallinta tilannevedoksilla

Pauli Borodulin

Tiivistelmä.
Tämä tutkielma käsittelee tilannevedoksia vaihtoehtoisena menetelmänä tietokannan samanaikaisuuden hallitsemiseksi. Tutkielman lähtökohtana on ensin tarkastella perinteisesti käytettyä lukintapohjaista menetelmää ANSI SQL:n näkökulmasta ja sitten esitellä rinnalle tilannevedoseristyvyys vaihtoehtoisena menetelmänä sekä vertailla tätä suhteessa perinteiseen menetelmään.

Avainsanat ja -sanonnat: tietokannat, samanaikaisuus, eristyvyys, tilannevedokset
CR-luokat: H.2.4

1. Johdanto

Samanaikaisuuden hallinta on tärkeä osa mitä tahansa tietokannanhallintajärjestelmää, joka mahdollistaa järjestelmän hallinnoimien tietokantojen samanaikaisen käsittelyn. Samanaikaisuuden hallinnalla tarkoitetaan lyhyesti kuvattuna menettelyä, jolla varmistetaan, että tietokannassa samanaikaisesti suoritettavien tapahtumien operaatiot pysyvät loogisesti erillään muiden tapahtumien operaatioista ja niiden suoritus säilyttää tietokannan eheyden. [Elmasri and Navathe, 2007, pp. 598–599.]

ANSI:n ja ISO:n ensi kertaa 1980-luvulla standardoima Structured Query Language eli SQL-kieli on muodostunut sittemmin koko alan standardiksi relaatiotietokantojen tietojen käsittelemisessä [Elmasri and Navathe, 2007]. SQL-käskyistä rakentuvat tapahtumat olivat SQL-kielen osana jo SQL-standardin ensimmäisessä versiossa SQL-86:ssa [SQL86]. Samanaikaisten käyttäjien määrä ei kuitenkaan ollut merkittävä tekijä vielä SQL-86:n aikakaudella ja kaikki tapahtumat suoritettiinkin silloisissa SQL-pohjaisissa tietokannanhallintajärjestelmissä aina sarjallistuvasti käyttäen kaksivaiheista lukintaa.

Käyttäjämäärien kasvaessa ehdoton sarjallistuva suoritus kaksivaiheisella lukituksella osoittautui rajoittavaksi tekijäksi samanaikaiselle käytölle ja SQL-kielen seuraavaan versioon SQL-92:een lisättiin tuki eristyvyystasoille (isolation levels) [SQL92]. Eristyvyystasojen avulla tietokannan käyttäjän oli mahdollista parantaa tietokantaoperaatioiden samanaikaisuutta vähentämällä tapahtumien eristyvyyttä.

Eristyvyystasojen toteuttamiseksi käytetty perinteinen lukintapohjainen menetelmä on kuitenkin epäedullinen tilanteissa, joissa tietokantaa käsitellään lukupainotteisesti. Tällöin lukijat rajoittavat kirjoittajia usein täysin tarpeetto-

masti [Kung and Robinson, 1981]. Tämä on tavallista erityisesti WWW-sovellusten kohdalla, joiden näkymiin koostetaan tietoa kerralla useista tietokantatauluista. Sen sijaan tietoja muokataan vain hyvin pienissä osissa, useimmiten vain yhden rivin verran kerrallaan ja harvoin samaa tietoalkiota.

Perinteinen lukintapohjainen menetelmä onkin saanut kilpailijakseen toisen, varsin erilaiseen lähestymistapaan perustuvan menetelmän samanaikaisuuden hallitsemiseksi. Sen sijaan, että yritettäisiin ennaltaehkäistä samanaikaisuusongelmat lukinnoilla, tilannevedoseristyvyydessä tapahtumien operaatiot suoritetaan rajoituksitta ja mahdolliset ongelmatilanteet tarkistetaan vasta tapahtumien lopuksi.

2. Tapahtumat ja samanaikaisuus

Tietokannanhallintajärjestelmän yksi perustehtävä on huolehtia siitä, että tietokannassa suoritettavat operaatiot (luku ja kirjoitus) ovat atomisia. Operaatioista saadaan tietokannan käyttäjän toimesta koottua isompia loogisia operaatioita käyttämällä tapahtumia (transactions). Tapahtuma on joukko operaatioita, joiden suorittaminen joko onnistuu tai epäonnistuu kokonaisuutena jättäen tietokannan eheään tilaan tapahtuman suorituksen jälkeen.

Tietokannan eheyden säilyttämiseksi on tarpeen kontrolloida samanaikaisten tapahtumien suorittamista viivyttämällä pääsyä tietoalkioihin tai peruuttamalla koko suorituksessa oleva tapahtuma. Ideaalitilanne olisi, että tapahtumille voitaisiin taata täysi eristyvyys rajoittamatta suoritusta lainkaan. Käytännössä eristyvyyden ja rajoitusten välillä on tehtävä kompromissi, jossa merkittävänä tekijänä on käytettävä samanaikaisuuden hallintamenetelmä.

Hallintamenetelmän tehtävä on huolehtia siitä, että rinnakkaisesti suoritettavien tapahtumien historia (history) on sarjallistuva. Historia kuvaa osittaisena järjestyksenä samanaikaisten tapahtumien peräkkäin lomitettujen operaatioiden suoritusjärjestyksen. Sarjallistuvalla historialla tarkoitetaan historiaa, jolla on sama riippuvuusgraafi kuin jollakin sellaisella historialla, jossa samat tapahtumat suoritetaan yksi kerrallaan peräkkäin. Sarjallistuvuus takaa tapahtumien ACID-ominaisuudet, jotka taas takaavat tietokannoissa yleisesti vaaditut tiedon oikeellisuuden säilytysominaisuudet.

3. ANSI SQL ja perinteinen toteutus – lukinnat

ANSI SQL:ää standardoitaessa tietokannanhallintajärjestelmiä ei haluttu sitoa standardin kannalta pelkästään yhteen tiettyyn samanaikaisuuden hallinnan tekniseen toteutustapaan (esim. lukintapohjaiseen). Tästä syystä eristyvyystasot määriteltiin standardissa suhteessa tunnistettuihin samanaikaisuusongelmien ilmentymiin eli eristyvyysanomalioihin (isolation anomalies). Anomalioiden

määrittely pohjautuu kuitenkin läheisesti lukkovuorottajien käyttäytymiseen. [Berenson et al., 1995.]

3.1. ANSI SQL:n eristyvyystasot

ANSI SQL:n tunnistamat anomaliat ovat likainen luku (dirty read), ei-toistettava luku (nonrepeatable read) ja haamut (phantoms). Eristyvyystasot ja niiden määrittelemiseksi käytetyt samanaikaisuusongelmien ilmentymät on lueteltu taulukossa 1 [SQL92].

	Eristyvyysanomalia		
Eristyvyystaso	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED	Kyllä	Kyllä	Kyllä
READ COMMITTED	Ei	Kyllä	Kyllä
REPEATABLE READ	Ei	Ei	Kyllä
SERIALIZABLE	Ei	Ei	Ei

Taulukko 1. ANSI SQL:n eristyvyystasot suhteessa määriteltyihin anomalioihin. [Berenson et al., 1995; Elmasri and Navathe, 2007]

Edellä mainitut eristyvyysanomaliat kuvataan perinteisesti seuraavasti:

1. **Likainen luku**: Tapahtuma T_1 voi lukea tietoalkion, jonka vielä sitoutumaton tapahtuma T_2 on tallentanut. T_2:n peruuntuessa on T_1 tullut lukeneeksi alkion, jota ei todellisuudessa koskaan ollut olemassa.
2. **Ei-toistettava luku**: Tapahtuma T_1 lukee aluksi jonkin haluamansa tietoalkion. Tämän jälkeen tapahtuma T_2 päivittää alkion arvon tai poistaa sen, jonka jälkeen T_1:n lukiessa alkion uudelleen, on sen arvo muuttunut tai koko alkio poistettu.
3. **Haamut**: Tapahtuma T_1 lukee joukon rivejä taulusta käyttäen jotakin hakuehtoa. Tämän jälkeen T_2 luo tauluun uuden rivin, joka täyttää T_1:n aiemmin käyttämän hakuehdon. T_1:n suorittaessa aiempi haku uudelleen, sisältääkin tulosjoukko uuden rivin.

Myöhemmissä tutkimuksissa on tunnistettu ANSI SQL:n anomalioiden lisäksi myös muita lukintapohjaisiin toteutuksiin ja samanaikaisuuteen yleensä liittyviä anomalioita, joita ei ole erikseen huomioitu ANSI SQL:ssä [Berenson et al., 1995].

ANSI SQL:n eristyvyystasot on perinteisesti toteutettu käyttämällä lukintoja. Esittelen seuraavaksi perinteisen toteutustavan yleisellä tasolla.

3.2. Kaksivaiheinen lukitus – perinteinen toteutustapa

Edellä kuvattu samanaikaisuuden hallinta on tietokannanhallintajärjestelmissä perinteisesti toteutettu lukoilla. Lukoilla tarkoitetaan tietoalkioita suojaavia muuttujia, jotka kuvaavat alkioiden tilaa. Alkioiden tilaa voidaan kuvailla lukkotyypeillä. Yleisesti tunnettuja lukkotyyppejä ovat binäärilukot (binary locks), luku-/kirjoituslukot (read / write locks) ja varmennuslukot (certify locks). Erilaisilla lukkotyypeillä voidaan hienojakoistaa lukituksia siten, että samanaikaisuus paranee. [Elmasri and Navathe, 2007, pp. 628–630.]

Lukkoihin perustuvia menetelmiä on olemassa useita, mutta näistä menetelmistä käyttöön on varsinaisesti vakiintunut vain kaksivaiheinen lukitus (two-phase locking, 2PL), sillä se takaa tapahtumien sarjallistuvuuden. Kaksivaiheisen lukituksen perusidea on, että kaikki tarvittavat lukitukset hankitaan ennen kuin yhtäkään lukitusta puretaan. Menetelmän nimi johtaa sen kahdesta vaiheesta, joista ensimmäisessä vain tehdään lukintoja ja toisesta, jossa lukintoja puretaan. Menetelmästä on olemassa erilaisia variaatioita koskien vaiheiden ajoitusta ja kestoa suhteessa tapahtuman kulkuun. Seuraavassa on esiteltynä variaatiot Elmasrin ja Navathen [2007, pp. 634–635] mukaan.

Konservatiivisessa (conservative, C2PL) tavassa kaikki tarvittavat lukitukset hankitaan ennen kuin tapahtuma alkaa, eikä itse tapahtuman aikana ole mahdollista tehdä lukintoja. Tästä seuraa se etu, että tapahtumat eivät voi lukkiutua (deadlock). Tapa on kuitenkin varsin epäkäytännöllinen eikä sovellu kovin hyvin ANSI SQL:n näkökulmasta lukintojen toteutustavaksi.

Tiukassa (strict, S2PL) tavassa kirjoituslukinnat puretaan vasta tapahtuman päättyessä (sitoutumiseen tai peruutukseen). Lukulukintoja voidaan purkaa jo ennen tapahtuman päättymistä, kunhan se tapahtuu viimeisen lukinnan tekemisen jälkeen. Lukitusten ensimmäisen vaiheen päättymisen tunnistaminen on kuitenkin monimutkaista, minkä takia tämän tavan sijasta suositaan seuraavaa, ankaraa tapaa.

Ankarassa (rigorous, R2PL) tavassa kaikki luku- ja kirjoituslukot puretaan vasta, kun tapahtuma päättyy (sitoutumiseen tai peruutukseen). Ankara tapa tunnetaan myös nimellä strong strict two-phase locking (SS2PL). Nimi johtaa siitä, että ankara tapa vastaa tiukkaa tapaa sillä lisäyksellä, että mitään lukintoja ei saa purkaa ennen tapahtuman päättymistä.

Edellä kuvatuista kolmesta tavasta ankara kaksivaiheinen lukinta on siis vakiintunut kaksivaiheisen lukituksen tavaksi tietokannanhallintajärjestelmissä. Seuraavassa kappaleessa käsittelen joitakin ankaraan kaksivaiheiseen lukintaan liittyviä ongelmia.

3.3. Ankaraan kaksivaiheiseen lukintaan liittyvistä ongelmista

Kaksivaiheisen lukituksen ehdottomana etuna voidaan pitää sitä, että se takaa tapahtumien sarjallistuvuuden. Lukkopohjaisiin menetelmiin liittyy kuitenkin joitakin haittoja ja ongelmia, joista seuraavassa.

Tapahtumien suorittamisessa tapahtuu lukkiuma (deadlock), kun kahdesta tai useammasta tapahtumasta jokainen haluaa lukita tietoalkion, joka on jo lukittuna jollakin toisella tapahtumista. Tällöin jokainen tapahtuma jää odottamaan lukkoa toiselta tapahtumalta, joka on edelleen jäänyt odottamaan lukkoa joltakin tapahtumalta. Lukkiumien estämiseksi on kehitetty useita menetelmiä, mutta niiden aiheuttamien haittojen tai monimutkaisuuden takia on käytännöllisempää keskittyä tunnistamaan, milloin tapahtumat päätyvät lukkiumaan [Elmasri and Navathe, 2007, pp. 637]. Kun lukkiuma tunnistetaan, peruu tietokannanhallintajärjestelmä valitun joukon tapahtumista siten, että muiden tapahtumien suoritus voi jatkua. Koska lukkiumat ovat välttämätön osa kaksivaiheista lukitusta, on tietokantaa käyttävien sovellusten varauduttava käsittelemään lukkiumista seuraavat tapahtumien peruuntumisen hallitusti.

Toinen kaksivaiheisen lukinnan perusongelma on nälkiintyminen (starvation). Nälkiintymisellä tarkoitetaan tilannetta, jossa tapahtuma odottaa lukkoa tarvitsemaansa tietoalkioon koskaan sitä saamatta, koska tapahtuma häviää aina kilpailun lukitusvuorosta muille tapahtumille. Tämä kilpailutilanne on helposti ratkaistavissa käyttämällä odottajille jotakin jonotusperiaatetta kuten tulojärjestystä. Toinen mahdollinen nälkiintymistilanne voi syntyä siitä, että tapahtuma päätyy toistuvasti lukkiumaan muiden tapahtumien kanssa ja sama tapahtuma tulee perutuksi toistuvasti. Näissä tilanteissa perumisalgoritmi voisi esimerkiksi antaa jatkuvasti perutuiksi tuleville tapahtumille korkeamman prioriteetin, jotta niiden onnistunut suoritus voidaan taata.

Edellä kuvatut ongelmat ovat laajalti tunnetut ja niiden huomioimiseksi on olemassa käyttökelpoiset ratkaisut.

4. Tilannevedokset vaihtoehtona lukinnoille

Perinteisesti käytetty kaksivaiheinen lukitus on samanaikaisuuden hallintamenetelmänä pessimistinen (pessimistic). Pessimistisissä menetelmissä tehdään aina jonkinlaisia tarkistuksia ennen tapahtuman kunkin operaation suorittamista. Lukitusten osalta tämä tarkoittaa sitä, että ennen operaation suorittamista tarkistetaan, onko alkio lukittuna ja jos, niin minkä tyyppisellä lukolla ja kenelle. Mikäli alkio on eksklusiivisesti lukittuna jonkin muun tapahtuman toimesta, joutuu lukkoa tarvitseva tapahtuma aina odottamaan, kunnes alkio vapautuu sen käyttöön. Tarkistusten avulla tietokannan tila pidetään eheänä, mutta sa-

malla tarkistukset ja lukkojen odottamiset viivyttävät tapahtuman suorittamista. [Kung and Robinson, 1981]

Optimistisissa (optimistic) menetelmissä tapahtuman aikana ei tehdä minkäänlaisia tarkistuksia, vaan tarkistukset hoidetaan vasta tapahtuman lopuksi. Esittelen seuraavaksi tilannevedoseristyvyyden, joka on laadultaan optimistinen menetelmä.

Tilannevedoseristyvyys (snapshot isolation) on samanaikaisuuden hallintamenetelmä, joka perustuu moniversiointiin. Tilannevedoseristyvyys on muunnelma Bernsteinin ja kumppaneiden [1987] kuvaamasta menetelmästä Multiversion Mixed.

4.1. Toimintaperiaate

Tilannevedoseristyvyydessä tapahtumat näkevät suorituksensa ajan tietokannan sisällön sellaisena kuin sen sitoutunut tilanne oli ennen tapahtuman ensimmäistä lukuoperaatiota. Näin ollen samaan aikaan suorituksessa olevat tapahtumat eivät vaikuta muiden tapahtumien näkemään tietokannan tilannevedokseen, mutta ne näkevät kuitenkin itse omaan vedokseensa tekemät muutokset. [Berenson et al., 1995]

Käytännössä tilannevedokset toteutetaan käyttämällä tapahtumille jonkinlaisia aikaleimoja. Aikaleima voi olla joko yksinkertainen juokseva numero tai päiväyksestä ja kellonajasta muodostuva yksilöivä tieto, jonka avulla tapahtumien keskinäinen järjestys voidaan päätellä. Tapahtuma saa aina alkaessaan aloitusaikaleiman T_s (Start-Timestamp) ja lopuksi sitoutuessaan sitoutumisaikaleiman T_c (Commit-Timestart). Näiden aikaleimojen avulla hoidetaan tapahtumien välisten konfliktien tarkistaminen sekä kiinnitetään tilannevedokset johonkin tietokannan sitoutuneeseen tilanteeseen.

Koska tilannevedoseristyvyys on moniversioiva, saattaa tietokannan tietoalkioista olla aktiivisten tapahtumien toimesta samanaikaisesti olemassa useita eri versioita. Näin ollen käynnissä olevien tapahtumien on aina valittava asianmukainen versio tietoalkioista käsittelyä varten. Valinta tapahtuu siten, että tapahtuma, jonka aloitusaikaleima on T_s, lukee vain sellaisia tietoalkioiden versioita, joiden sitoutumisaikaleimat ovat juuri ennen T_s:ää. Lisäksi ennen viimeisimmän sitoutuneen version lukemista tarkistetaan vielä erikseen, ettei tapahtumalla ole olemassa omaa paikallista versiota tietoalkiosta.

Tapahtumalle annetaan sitoutuessaan sitoutumisaikaleima, jonka avulla tarkistetaan samalla, voiko tapahtuma todella sitoutua. Tavanomainen tarkistustapa on *ensimmäinen sitoutuja voittaa* (first committer wins). Tässä tarkistustavassa tapahtuma T1 voi sitoutua lopullisesti vain, jos mikään tapahtuman T1 aikana (aloitus- ja sitoutumisaikaleiman välillä) sitoutunut tapahtuma T2 ei ole kirjoittanut samaan tietoalkioon tapahtuman T1 kanssa [Berenson et al., 1995].

Ensimmäinen sitoutuja voittaa -tavasta on olemassa myös paranneltu versio *ensimmäinen päivittäjä voittaa* (first updater wins), jossa alkuperäisen tavan ehto toteutetaan tekemällä tarkistuksia jo tietoalkioiden päivitysten yhteydessä. Tässä tavassa tietoalkiota päivittävä tapahtuma varaa samalla kirjoituslukon päivitettävään tietoalkioon estääkseen tarpeettomat samanaikaiset päivitykset samaan tietoalkioon. Tarpeettomat siksi, että jälkimmäinen tapahtuma tulisi joka tapauksessa peruuntumaan suorituksensa lopuksi tehtävän ensimmäinen sitoutuja voittaa -tarkistuksen perusteella.

Ensimmäinen päivittäjä voittaa -tapa toimii seuraavasti. Kun T1 ja T2 ovat samanaikaisesti suoritettavia tapahtumia ja T1 päivittää tietoalkiota X, ottaa T1 kirjoituslukon X:ään. T2:n yrittäessä päivittää samaa tietoalkiota X (ennen T1:n sitoutumista), joutuu T2 jäämään odottamaan kirjoituslukkoa. T1:n sitoutuessa T2 perutaan aina, mutta jos T1 peruuntuukin, vapautuu kirjoituslukko ja T2:n suoritusta voidaan jatkaa. Toisaalta, jos T1 päivittää alkiota X ja sitoutuu ennen kuin T2 yrittää päivittää samaa alkiota, perutaan T2 sen yrittäessä päivittää X:ää. [Fekete et al., 2005]

4.2. Erityispiirteet

Tilannevedoseristyvyys ei aina takaa sarjallistuvuutta. Fekete ja kumppanit [2005] ovat tunnistaneet joukon anomalioita, joille tilannevedoseristyvyys on altis. Huolimatta tunnistamistaan anomalioista Fekete ja kumppanit kuitenkin toteavat, että tunnistetut anomaliat ovat tosimaailman sovelluksissa harvinaisia. Esimerkiksi tietokannanhallintajärjestelmän suorituskykyä mittaava TPC-C–testi suoriutuu kokonaisuudessaan ilman ongelmia käytettäessä tilannevedoseristyvyyttä. TPC-C:tä voidaan pitää hyvänä mallina siitä, millaisia operaatioita sovellukset yleensä tietokannassa suorittavat. Lisäksi anomaliat on mahdollista huomioida sovellusta suunniteltaessa ja näin välttää tilannevedoseristyvyyteen liittyvät ongelmat.

Tilannevedoksia käyttävissä tapahtumissa tietokannan lukuja ei viivytetä lukoilla, joten tapahtumat voivat esteettä lukea tietokannan tietoja riippumatta muista samanaikaisista tapahtumista. Poikkeuksen tästä muodostaa ensimmäinen päivittäjä voittaa -tarkistustapa, jolloin kuitenkin pyritään vain välttämään resurssien tarpeetonta käyttöä sen sijaan, että tarpeettomasti viivytettäisiin tapahtuman suoritusta. Tämän takia tilannevedoseristyvyys tarjoaa erittäin hyvän suorituskyvyn, kun tapahtuma haluaa pelkästään lukea tietokantaa [Kung and Robinson, 1981].

Tilannevedokset tarjoavat myös kätevän ratkaisun tietokannan varmuuskopioimiseen normaalia käyttöä keskeyttämättä. Perinteistä lukintaa käytettäessä on käytännössä koko tietokanta lukittava varmuuskopion muodostamisen ajaksi. Tällöin käyttäjät eivät voi tehdä tietokantaan muutoksia, koska muutoin

luotava varmuuskopio ei olisi eheä. Tilannevedoksen avulla tietokannasta voidaan ottaa käsiteltäväksi sen viimeisin sitoutunut ja eheä tilanne, joka sitten tallennetaan johonkin erilliseen varastoon.

5. Yhteenveto

Tilannevedoseristyvyys on houkutteleva vaihtoehto perinteiselle lukintapohjaiselle samanaikaisuuden hallinnalle. Huolimatta siitä, että tilannevedoseristyvyys ei aina takaa sarjallistuvaa tapahtumien suoritusta, on se silti tärkeä vaihtoehto suorituskyvyn ja eristyvyyden kannalta tarjoamansa hyvän balanssin takia. Kaikissa tilanteissa ei edes ole välttämätöntä, että voidaan taata täysin sarjallinen suoritus.

Niissä tapauksissa kun sarjallistuva suoritus on välttämätöntä, on tietokantasovellus mahdollista muokata suorittamaan tietokannan käsittely siten, että tilannevedoseristyvyydelläkin tavoitetaan käytännössä sarjallistuva suoritus.

Viiteluettelo

[Berenson et al., 1995] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil, A critique of ANSI SQL isolation levels. In: *Proc. of the 1995 ACM SIGMOD International Conference on Management of Data,* 1–10.

[Bernstein et al., 1987] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[Elmasri and Navathe, 2007] Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems*, *Fifth Edition.* Addison Wesley, 2007.

[Fekete et al., 2005] Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha, Making snapshot isolation serializable. *ACM Transactions on Database Systems* **30**, 2 (June 2005), 492–528.

[Kung and Robinson, 1981] H. T. Kung and John T. Robinson, On optimistic methods for concurrency control. *ACM Transactions on Database Systems* **6**, 2 (June 1981), 213–226.

[SQL86] ANSI X3.135-1986, American National Standard for Information Systems – Database Language – SQL, May 1986.

[SQL92] ANSI X3.135-1992, American National Standard for Information Systems – Database Language – SQL, November 1992.

Katseenseurantaa hyödyntävät agentit oppimisympäristöissä

Johanna Hakola

Tiivistelmä.

Tässä tutkielmassa käsitellään katseenseurantaa hyödyntäviä agentteja. Katseenseuranta mahdollistaa uudenlaisia vuorovaikutustapoja tietokoneen ja käyttäjän välille. Opetus, yhteydenpito, tiedon hallinta ja tiedon suodatus ovat vain muutamia sovellusalueita, jotka voivat hyötyä katseenseurannan ja katsetta hyödyntävien käyttöliittymien ja käyttöliittymien osien kehityksestä ja yleistymisestä. Agentit ovat käyttöliittymän osia, joiden tarkoitus on kommunikoida käyttäjän kanssa. Agentteja voidaan käyttää monissa erilaisissa kohteissa ja käyttötarkoituksissa, mutta keskityn erityisesti oppimisympäristöissä käytettäviin katseenseurantaa hyödyntäviin agentteihin.

Avainsanat ja -sanonnat: HCI, agentit, katseenseuranta
CR-luokat: H.5, K.3, J.4

1. Johdanto

Agentteja käytetään erilaisissa tietokoneohjelmissa ja nettisivuilla muun muassa opastamassa käyttäjää ohjelman tai sivuston käytössä tai avustamassa tiedon hallinnassa. Agentteja käytetään myös esimerkiksi etäneuvotteluissa, joissa agentti edustaa neuvottelun osallistujaa: muut neuvottelun osallistujat näkevät toisten osallistujien agentit ja kommunikoivat näille ja näiden kanssa eri menetelmin.

Katseenseurantaa hyödyntävän agentin tietäessä, mihin kohtaan näyttölaitetta käyttäjä katsoo, agentti voi antaa tarkempia ohjeita vaikkapa kohdasta, johon käyttäjä katsoo pitkään tai useasti. Katseen pitkä kesto tarkoittaa usein sitä, että käyttöliittymän kohta on ongelmallinen tai tärkeä ja oleellinen. Agentti voi myös reagoida jollain tavalla siihen, että käyttäjä katsoo agenttia tietyn ajan tai tiettyyn kohtaan. Agentteja voi olla myös useampia, jolloin niillä on erilaiset roolit.

Etäneuvotteluissa katseenseurantaa hyödyntävistä agenteista on hyötyä esimerkiksi siten, että katseen avulla tehtävä sanaton viestintä mahdollistuu paremmin kuin etäneuvotteluissa yleensä. Puhe voidaan esimerkiksi helposti kohdentaa jollekin tietylle osallistujalle, koska agentti katsoo sitä agenttia, jonka ”taustalla” olevalle ihmiselle puhe tai muu viesti on tarkoitettu. Tiedon suodatuksessa ja hallinnassa agentteja käytetään keräämään tietoa erilaisista tieto-

kannoista, uutispalveluista, internetistä, sähköposteista, käyttäjän fyysisestä toimintaympäristöstä ja käyttäjän toiminnasta.

Katseenseurannassa ja agenteissa on paljon tutkittavaa, mutta teen pienen kartoituksen liittyen juuri katseenseurannan hyödyntämisestä agenttien avulla Aluksi kerron perusasioita katseenseurannan historiasta ja nykypäivästä. Sitten kuvailen agentteja yleisesti ja niiden muutamia käyttötarkoituksia. Kerron myös oppimisympäristöistä, jonka jälkeen perehdyn agentteihin oppimisympäristöissä.

Tässä tutkielmassa agentin fyysiseksi toimintaympäristöksi ajatellaan tavallinen pc-tietokone ja käyttäjän toimintaympäristöksi normaali työpöytätyöskentely katseenseurantajärjestelmällä. Agenttien käyttö ei rajoitu vain pc-ympäristöön ja sillä työskentelyyn, mutta työn rajaamisen vuoksi mobiilien päätelaitteiden, kaikkialla läsnä olevan tietotekniikan (ubiquitous computing) ja muiden uudempien sovellusympäristöjen mahdollisesti vaatimat erityispiirteet on jätetty käsittelemättä.

2. Katseenseurannasta

Katseenseurantalaitteita on alun perin kehitetty sotateollisuuden tarpeisiin, kun on täytynyt tutkia lentäjien toimintaa ja katseen kulkua lentokoneen ohjaamossa [Laarni, 2004]. Kiinnostuksen kohteina olivat esimerkiksi hahmontunnistus, värien, liikkeen ja muutosten havaitseminen ja katseen kulku ohjaamossa. Tutkimuslaitteina käytettiin useimmiten filmikameraa ja mekaanisia silmän lihasten liikkeitä mittaavia laitteita [Selker, 2004; Fitts et al., 1950].

Tietokoneiden yleistyessä katseenseurannan tutkimus ja katseenseurantalaitteiden kehitys siirtyi tavallisten käyttäjien toiminnan tutkimukseen. Kiinnostuksen kohteina ovat olleet muun muassa katseen kohdistuminen laadunvalvonnassa, mainonnassa, lukunopeuden- ja tekniikoiden tutkimuksessa ja tiedon hallinnassa ja esityksessä. Viimeisimpiä katseenseurannan sovellus- ja tutkimusalueita ovat olleet Tampereen yliopistossakin tutkitut ja kehityksen kohteena olleet vaikeasti vammaisille tarkoitetut katseella ohjattavat käyttöliittymät [Istance et al., 2006] ja katseella kirjoittaminen [Hyrskykari et al., 2000].

Silmänliikkeiden tutkimiseen käytetään erilaisia menetelmiä. Nykyisin käytössä on yleensä työpöydällä tietokoneen läheisyydessä pidettävä katseenseurantalaite (eye tracking device), eikä käyttäjän tarvitse enää takavuosien tapaan pitää laitetta päässään. Laitteiden ongelmana on kuitenkin huono soveltuvuus testiympäristöjen ulkopuolelle. Katseen paikantaminen on epätarkkaa ja katseenseurantalaitteet täytyy kalibroida eli mitoittaa joka käyttäjälle erikseen. Voimakkaat pään liikkeet ja katseen kohdistaminen näytön ulkopuolelle saattavat aiheuttaa sen, että laite kadottaa katseen paikantumisen (focuksen) joko

hetkellisesti tai pysyvästi. Katseenseurantalaite täytyy tällöin kalibroida uudelleen [Hyrskykari, 2006]. Kehittyessään ja yleistyessään laitteet ja sovellukset kuitenkin toivottavasti tarjoavat uudenlaisia ja tehokkaita vuorovaikutustapoja käyttäjän ja tietokoneen välille.

3. Mitä agentit ovat ja missä niitä käytetään

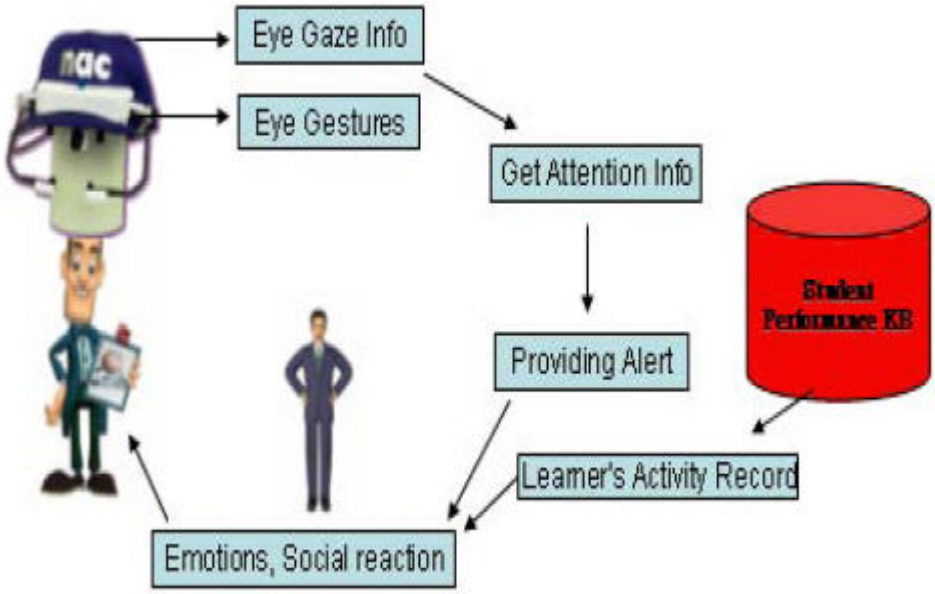
Agentit ovat verkkosivuilla ja tietokoneohjelmissa olevia käyttäjää eri tavoin avustavia hahmoja. Yleisesti ajatellaan, että agentiton selkeästi tunnistettava hahmo ja sillä on ainakin kasvot, ellei koko vartaloa. Agentti pystyy liikkumaan näytöllä ja esittämään erilaisia ilmeitä, eleitä ja toimintoja [Cassell, 1999].

Se, mitä agentti varsinaisesti tekee, riippuu sen toimintaympäristöstä ja käyttötarkoituksesta. Oppimisympäristössä agentit esittävät tietoa ja motivoivat opiskelijaa. Tiedon suodatukseen ja esittämiseen tarkoitetut agentit keräävät, arvioivat ja lopulta esittävät valikoitua tietoa käyttäjälle [Maglio et al., 2004]. Yhteydenpitoon ja tiedotukseen tarkoitetut agentit toimivat käyttäjänsä hahmona monen käyttäjän etäyhteyksissä. Kuten muutkin katseenseurantaa hyödyntävät agentit, ne reagoivat katseeseen, eleisiin ja ilmeisiin ja osaavat muun muassa kohdistaa katseensa puhujaan.

Kaikille katseenseurantaa hyödyntäville agenteille yhteistä on, että ne keräävät dataa käyttäjän toimista katseenseurannan, hiiren ja näppäimistön avulla, reagoivat käyttäjän toimintaan ja toimivat sen mukaan. Agentit ovat käyttäjän apulaisia: hyvä agentti on ystävällinen, ei häiritse käyttäjää, se on helposti lähestyttävä ja on tietoinen käyttäjän toimista [Maglio et al., 2004]. Lickliderin [1960] mukaan agenttien - ja tietokoneiden yleensäkin - tulisi toimia periaatteella, että ihminen asettaa toiminnan päämäärän ja agentti tai tietokone tekee tarvittavat toiminnot tämän päämäärän saavuttamiseksi. Agentit ja tietokoneet ovat siten toiminnan aktiivisia osallistujia mutta kuitenkin käyttäjän alaisuudessa.

Kuvassa 1 selvitetään, miten käyttäjältä saatu palaute ja tallennettu informaatio vaikuttaa agentin toimintaan ja sen antamaan palautteeseen. Käyttäjää kuvaa vasemmassa reunassa oleva isompi hahmo ja agentti on keskellä oleva pienempi hahmo. Käyttäjältä tulevaa informaatiota ovat katse (eye gaze info) ja eleet (eye gestures). Tämä palaute yhdistetään aikaisemmin tallennettuun käyttäjää koskevaan dataan (learner's activity record) ja näin saadaan perustoimintamalli, joka sopii juuri tälle käyttäjälle. Tämä yhdistetään vielä kyseiseen tilanteeseen, jolloin agentin reaktio, emootiot, toiminta ja siten käyttäjälle antama palaute määräytyvät tilanteeseen sopiviksi. Tässä kaaviossa on otettu huomioon vain katseenseurantalaitteelta tuleva ja tallennettu palaute, mutta palautet-

ta voidaan saada myös muulla tavoin, esimerkiksi näppäimistön ja hiiren käytöstä.



Kuva 1.

Käyttäjältä tuleva palaute yhdistyy tallennettuun tietoon ja käyttötilanteeseen. Agentin toiminta määräytyy näiden perusteella.

Huonojen agenttien ominaisuuksia ovat heikko käytettävyys, hyödyttömyys, toimimattomuus ja kenties kaikkein ärsyttävimpänä ominaisuutena käyttäjän toiminnan toistuva keskeyttäminen. Agentin tulee joka tilanteessa olla hyödyksi, ei häiriöksi.

Ihanteellisessa tapauksessa agentti tarjoaa selkeän ja nopean yhteyden saatavilla olevaan tietoon sellaisella tavalla, joka käyttäjän on helppo ymmärtää [Wang et al., 2006a]. Cassell [2001] esittää, että katseeseen reagoivilla agenteilla on samanlaisia käyttäytymistapoja kuin ihmisilläkin keskustellessaan keskenään: kyky ymmärtää ja tuottaa puhetta, eleitä ja ilmeitä ja taito käydä keskusteluja antamalla ja ottamalla puheenvuoroja.

4. Oppimisympäristöistä

Opetusohjelmat ja oppimisympäristöt ovat lisääntyneet opetuksessa viime vuosina paljon. Niiden avulla on helppoa ja tehokasta jakaa tietoa usealle oppijalle ja ympäristöt tarjoavat myös vuorovaikutusmahdollisuuksia opiskelijoiden kesken ja opiskelijoiden ja opettajien välille. Oppimisympäristöissä jaetaan paitsi tietoa – muun muassa artikkeleja, luentomateriaaleja, kuvia, opiskelijoiden tekemiä tekstejä – myös oppimiskokemuksia.

Oppimista ja oppimisympäristöjä tutkittaessa on todettu, että ihmiskontaktin läsnäolo oppimisympäristössä parantaa opiskelumotivaatiota [Wang et al., 2006a]. Oppimistulokset paranevat ja sitoutuminen opiskeluun kasvaa, kun opiskelijan ei tarvitse kokea olevansa yksin, vaan saatavilla on neuvoja ja vertaistukea. Oppimisessa ja opetuksessa ei ole tärkeää vain tiedon tarkoituksenmukainen järjestäminen ja esittäminen, vaan kuten muussakin ihmisen välisessä kanssakäymisessä, ilmeet, tunteet ja eleet ovat oleellisia oppimisen vuorovaikutuksessa. Palloff ja Pratt [2001] ovat jopa esittäneet, että tavalla esittää asia on enemmän merkitystä kuin sen sisällöllä ja opiskelijat saattavat menettää opiskelumotivaationsa ja mielenkiintonsa opiskeltavaan asiaan hyvin nopeasti, jos oppimisympäristö ei sovellu heidän tarpeisiinsa. Tämä asettaa suuria haasteita oppimisympäristöille, jotka monesti ovat keskittyneet vain opiskelumateriaalin välittämiseen opettajalta oppilaille. Oppimisympäristön tulisi pystyä ylläpitämään opiskelijan motivaatiota ja kiinnostusta.

Koska oppimisympäristöjen ideana ja selvänä etuna perinteiseen luokkahuoneopetukseen on vapaus opiskella silloin ja sillä aikataululla, kun opiskelijalle sopii, ei ole mahdollista luoda oikean ihmisen läsnäoloa oppimisympäristöihin, ainakaan jatkuvasti ja kovin pitkiksi ajoiksi kerrallaan. Siksi on mietittävä muita tapoja toteuttaa opettajan tai toisten opiskelijoiden läsnäolo oppimisympäristössä. Tässä agenteista voi olla suurta apua. Ne ovat väsymättömiä

opettajia, motivoijia, kertauskysymysten esittäjiä ja oppimisen tukijoita. Ne ovat aina paikalla silloin, kuin opiskelijakin.

5. Agentit opetusohjelmissa

5.1. Katseenseuranta ja agentit

Yksi ratkaisu oppimisympäristöjen vuorovaikutuksen parantamiseen itse ympäristön ja opiskelijan välillä on hyödyntää katseenseurantaa. Katseenseurannan avulla saadaan tietoa käyttäjän mielenkiinnon kohteista samalla, kun käyttäjälle voidaan katsetta hyödyntävien ja käyttäjän toimiin reagoivien animoitujen agenttien avulla luoda tunne opettajan läsnäolosta oppimisympäristössä. Tämä voi parantaa opiskelijan motivaatiota ja tehdä oppimisesta mukavampaa. Oleellista on, että agentit on suunniteltu keräämään tietoa käyttäjästä ja hyödyntämään tätä tietoa älykkäästi, jolloin niistä on käyttäjälle oikeaa hyötyä, eivätkä ne jää pelkästään hauskaksi mutta pian unohtuvaksi yksityiskohdaksi oppimisympäristössä [Wang et al., 2006a].

Katseenseurantaa hyödyntävät agentit keräävät samanaikaisesti dataa käyttäjän käyttäytymisestä ja antavat sen mukaan palautetta käyttäjälle. Agentit saavat silmän liikkeiden, katseen pituuden, pupilleissa tapahtuvien muutosten ja pään liikkeiden perustella tietoa, mihin käyttäjä katsoo ja kuinka kauan katse kestää. Yleensä pitkä katse tarkoittaa sitä, että katsomisen kohde kiinnostaa käyttäjää tai kohta on jollain tapaa ongelmallinen. Pupillit laajenevat, kun käyttäjä näkee jotain miellyttävää tai mielenkiintoista, kun taas väsymyksen tai pitkästymisen takia pupillit pienenevät.

Katseenseurannasta saatua palautetta ja käyttäjän muuta toimintaa hyödynnetään agentin ja käyttäjän välisessä vuorovaikutuksessa siten, että agentin käyttäytyminen, ilmeet ja emootiot muuttuvat tilanteeseen sopivaksi. Agentti voi antaa ystävällisesti ohjeita käyttäjälle, jos tämä epäröi tai ei ymmärrä jotain asiaa. Agentti käyttää hyödykseen myös aiemmin kerättyä dataa palautteen antamisessa ja käyttäjältä tulevan palautteen keräämisessä.

5.2. Agenttien toimintoja opetusohjelmissa

Opetusohjelmassa katseen paikasta saatavaa tietoa voidaan käyttää moniin yksinkertaisiinkin tarkoituksiin. Yksi sellainen on opiskeltavan aiheen valinta. Kun agentti havaitsee käyttäjän katsovan jotain kuvaa pidempään kuin muita kuvia, agentti voi korostaa kuvaan liittyviä muita kohteita sivulla tai näyttää lisätietoa kohteesta. Kuvassa 2 on tilanne, jossa valittuna on keskimmäinen kuva [Wang et al., 2006a].

Kuva 2. Aiheen valinta.

Katseen ja agenttien avulla voidaan tehdä myös aihevalintoja ja aiherajauksia. Eleitä voi käyttää yksinkertaisiin kyllä/ei-vastauksiin ja valintojen tekemiseen. Nyökyttely tarkoittaa kyllä ja pään pudistelu tarkoittaa ei. Valinnat tehdään katsomalla kohdetta tietyn aikaa. Myös katseen yhdistäminen hiiren ja näppäimistön käyttöön on mahdollista ja tehokasta, jolloin valinta voi tapahtua esimerkiksi siten, että katsella rajataan tai valitaan kohde ja näppäimistöltä painamalla hyväksytään valinta. Näppäimistön ja hiiren käytössä katseeseen yhdistettynä on muitakin etuja, koska katse ei voi olla ”päällä” koko aikaa. Katseenseuranta pitää voida kytkeä päälle ja pois päältä jollain muullakin tavalla kuin pelkästään katseen avulla.

Monimutkaisempia käyttötarkoituksia saadaan, kun mukaan otetaan emootiot, katseen ja silmän fyysiset muutokset ja niiden hyödyntäminen. Agentit voivat olla erittäin hyödyllisiä käyttäjän motivoinnissa. Jos agentti havaitsee käyttäjän katselevan pitkään muualle kuin tietokoneen näytölle, agentti voi pyytää käyttäjää keskittymään opiskeltavaan asiaan ja näyttää tyytymättömältä. Jos taas käyttäjä tutkii opiskeltavaa sisältöä kiinnostuneesti, agentti on iloinen ja tyytyväinen. Käyttäjän pitkästyminen tai väsyminen opiskeluun tai aiheeseen ilmenee monesti siten, että silmän ja pään liikkeet vähenevät, pupillien koko pienenee ja hiiren liikkeet ja näppäimistön käyttö vähenevät. Toisin sanoen käyttäjän aktiivisuus vähenee. Tässä tilanteessa agentti voi ehdottaa taukoa tai opiskeltavan aiheen vaihtoa.

Jos käyttävä vaikuttaa juuttuvan johonkin hankalaan kohtaan tai vastaa väärin esitettyyn kysymykseen, agentti voi antaa vihjeitä oikean vastauksen löytämisestä, olla empaattinen ja tarjota tukea ja apua [Wang et al., 2006a]. Kaikissa toiminnoissa, joissa agentin antama palaute riippuu käyttäjältä tulleesta datasta ja informaatiosta, on tärkeää, että agentti osaa käyttää aiemmin saatua tietoa hyväkseen. Tätä varten käyttäjistä saatua dataa tallennetaan myöhemmin hyödynnettäväksi. Käyttäjät ovat erilaisia persoonallisuudeltaan, emootioiltaan ja käytökseltään ja heillä on erilaisia odotuksia oppimisympäristönsä ja siltä saatavan palautteen suhteen, joten on tärkeää, että agentti oppii käyttäjästään jatkuvasti. Tämä tarkoittaa sitä, että agentin on mukauduttava käyttäjän toimin-

taan. Jos itsekseen pohtiva käyttäjä jatkuvasti hiljentää agentin sen luullessa käyttäjän lopettaneen opiskelun ja yrittäessä ”herätellä” tätä, agentin odotusajan on pidennyttävä. Joskus taas kärsimättömämpi käyttäjä haluaa agentilta vastausvinkkejä heti, kun ei keksi vastausta itse, jolloin agentti voi antaa vihjeitä nopeammin kuin muille käyttäjille keskimäärin tai keksiä jonkun tavan kannustaa opiskelijaa etsimään vastausta vielä hetki itsenäisesti. Periaatteessa mitä tahansa käyttäjäkohtaista dataa, vaikkapa opiskelurytmin ja lukunopeuden, voi tallentaa hyödynnettäväksi myöhemmin. Oppimisympäristöihin saattaa kuulua mahdollisuus seurata oppimisen etenemistä jollain tavoin, ja tämän tiedon tulee tietysti olla myös agenttien hyödynnettävissä.

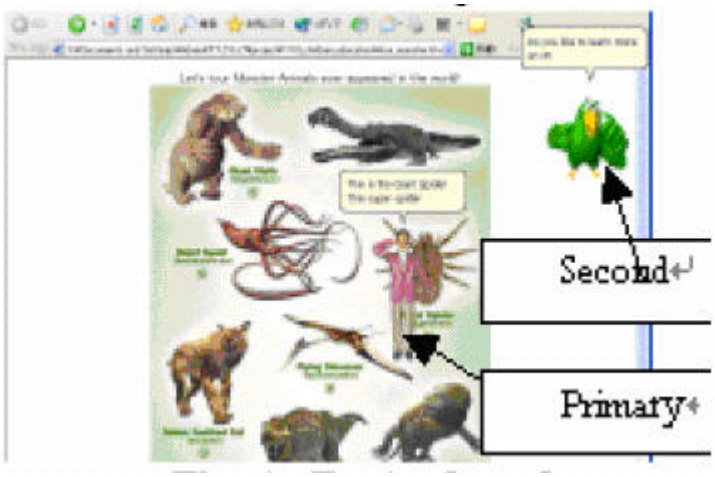
5.3. Useamman agentin käyttö samanaikaisesti

Agentteja voi olla samanaikaisesti käytössä useampi kuin yksi. Tällöin agenteilla on eri roolit ja ne ovat vuorovaikutuksessa paitsi käyttäjän kanssa, myös keskenään. Käytettäessä kahta agenttia ensisijaisen, käyttäjän kanssa enemmän kommunikoivan agentin tehtävä on esitellä opiskeltava asiasisältö, korostaa oikeita kohtia aineistosta ja esimerkiksi tehdä kysymyksiä alueesta ja käyttäjän kiinnostuksen kohteista.

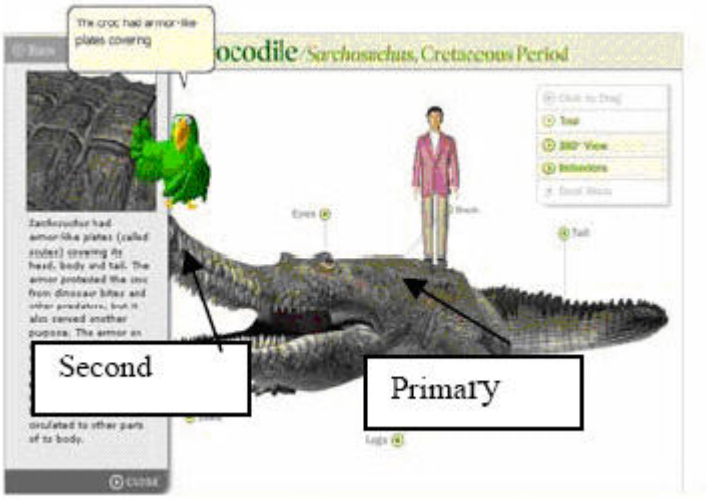
Toisen, käyttäjän kanssa vähemmän kommunikoivan ja siten enemmän sivuosassa olevan agentin rooli on toimia tiedon välittäjänä käyttäjän ja järjestelmän välillä. Tämä kakkosagentti tarkkailee käyttäjän keskittyneisyyttä aiheeseen, emootioita, motivaatiota ja vireystasoa. Se voi myös kerätä tietoa silmän liikkeistä ja katseen kohdistumisesta ja välittää tätä tietoa järjestelmään ja toiselle agentille [Wang et al., 2006b]. Mainittuja toimintoja voidaan jakaa useammallekin kuin kahdelle agentille, mutta lienee kuitenkin tarkoituksenmukaista, ettei agentteja lisätä määrättömästi.

Useampi agentti mahdollistaa agenttien välisen kommunikaation. Agentit voivat esimerkiksi käydä mielipiteen vaihtoa erilaisista näkökulmista liittyen opiskeltavaan aiheeseen. Agenttien keskustelut tuovat vaihtelua opiskeluun, pitävät opiskelijan mielenkiintoa yllä ja tarjoavat loistavan tilaisuuden asioiden kertaamiselle, mikä on tärkeää tehokkaan oppimisen kannalta.

Kuvassa 3 on tilanne, jossa on käytössä kaksi agenttia oppimisympäristön aiheenvalintasivulla. Kuva 4 on samoista agenteista kyseisen oppimisympäristön sisältösivuilta. Ihmishahmo on ensisijainen agentti (primary) ja papukaija toissijainen agentti (secondary). Näissä esimerkeissä ihmishahmoagentin tehtävänä on puhua käyttäjälle ja toissijainen papukaija-agentti tarjoaa järjestelmään liittyvää informaatiota ja yrittää pitää opiskelijan mielenkiintoa yllä [Wag et al., 2006b].



Kuva 3. Kaksi agenttia oppimisympäristön aiheenvalintasivulla.



Kuva 4. Kaksi agenttia oppimisympäristön sisältösivulla.

6. Lopuksi

Yleistyessään katseenseuranta ja sitä hyödyntävät agentit voivat tuoda mielenkiintoisen lisän käyttäjän ja tietokoneen väliseen vuorovaikutukseen. Erityisen hyödyllisiä agentit ovat tilanteissa, joissa käyttäjä kaipaa ohjausta ja joissa käyttäjän toiminta ja emootiot tulisi ottaa huomioon. Agenttien käytön ei tule olla itsetarkoitus vaan niiden pitää tuoda jotain selvää hyötyä ja etua käyttäjälle.

Tässä tutkielmassa agentteja käsiteltiin melko teoreettisella ja yleisellä tasolla ja keskittyen lähinnä vain oppimisympäristöjen agentteihin. Tutkimusta voisi laajentaa vaikkapa käytännön sovellusten hyötyjen ja haittojen tutkimukseen, käyttäjäkokemuksiin, muiden ympäristöjen agentteihin ja muiden kuin pc-laitteiden sovelluksiin.

Jokainen nykyään yleisesti käytössä oleva tekninen sovellus on joskus ollut uusi ja vasta tutkimuskohteena ja kehitysasteella tutkijoiden ja suunnittelijoiden työpöydällä ja testiympäristöissä. Mielenkiintoista onkin nähdä, milloin katseenseurannan hyödyntäminen on yhtä arkipäiväistä kuin vaikka sähkön

käyttö nykyisin ja milloin voin todeta kirjoittaneeni katseenseurannasta jo silloin, kun sen käyttö ei vielä ollut kaikille tuttua.

Viiteluettelo

[Cassell et al., 1999] Cassell, J., Bickmore, T., Campbell, L., Vilhjsson, H., and Yan, H. Conversation as a system framework: Designing embodied conversational agents. In: Cassell, J. et al., *Embodied Conversational Agents*, MIT Press, Cambridge, MA.

[Fitts et al., 1950] Fitts, P. M., Jones, R. E., and Milton, J. L. Eye movement of aircraft pilots during instrument-landing approaches. *Aeronautical Engineering Review* **9** (1950), 24-29.

[Graham, 1984] Graham, S. Communicating sympathy and anger to black and white children: the cognitive [attributional] consequences of affective cues. *Journal of Personality and Social Psychology* **47** (1984), 40-54.

[Hyrskykari, 2006] Hyrskykari, A. Utilizing eye movements: Overcoming inaccuracy while tracking the focus of attention during reading. *Computers in Human Behavior*, **22**(4), (July 2006), 657-671.

[Hyrskykari et al., 2000] Hyrskykari, A., Majaranta, P., Aaltonen, A., and Räihä, K.-J. Design Issues of iDict: A Gaze-Assisted Translation Aid. In: *Proceedings of ETRA2000*, Eye Tracking Research and Applications Symposium, Palm Beach Gardens, FL, ACM Press, November 2000, 9-14.

[Istance et al., 2006] Istance, H., Hyrskykari, A., Koskinen, D., and Bates, R., Gaze-based Attentive User Interfaces (AUIs) to support disabled users: towards a research agenda. In: *Proceedings of COGAIN2006*. http://www.cogain.org/cogain2006/COGAIN2006_Proceedings.pdf [3.6.2008].

[Laarni, 2004] Laarni, J. Silmänliikkeiden rekisteröinti käyttöliittymien tutkimuksessa. *Psykologia*, Helmikuu 2004.

[Licklider, 1960] Licklider, J. C. R. Man-Machine symbiosis. *IRE Transactions on Human Factors in Electronics* **HFE-1** (1960), 4-11.

[Maglio et al., 2003] Maglio, P. P., and Campbell, C. S. Attentive agents. *Communications of the ACM* **46,** 3 (2003), 47-51.

[Palloff and Pratt, 2001] Palloff, R. M., and Pratt, K. *Lessons from the Cyberspace Classroom: The Realities of Online Teaching*. San Francisco: Jossey-Bass, 2001.

[Selker, 2004] Selker, T. Visual Attentive Interfaces. *BT Technology Journal* **22**, 4, (2004), 146-150.

[Wang et al., 2006a] Wang, H., Chignell, M., and Ishizuka, M. Empathic Tutoring Software Agents Using Real-time Eye Tracking. In: *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications,* San Diego, California, (2006), 73-78.

[Wang et al., 2006b] Wang, H., Chignell, M., and Ishizuka, M, Are two talking heads better than one? When should use more than one agent in e-learning? In: *IUI'06*, Sydney, Australia, (2006), 366-368.

Skriptikielien ohjelmistokehykset web-ohjelmoinnissa

Jukka Hell

Tiivistelmä.
Tämän tutkimuksen tarkoitus on selvittää kolmen eri skriptikielen ohjelmistokehyksen (framework) soveltuvuutta web-ohjelmointiin. Ohjelmistokehyksiksi valitsin PHP:n Zend Frameworkin, Rubyn Ruby on railsin sekä Groovyn Grailsin. Vertailupohjana ovat kullakin kielellä ohjelmoidut suppeat sähköpostinlähetysohjelmat, joissa on syötteiden tarkastukset mukana. Ohjelmointityöt on tehty oliopohjaisesti ja ohjelmista tarkastellaan pääosin koodin määrää, selkeyttä sekä käyttöönottoa ensikertalaisen näkökulmasta. Grails osoittautui ohjelmistokehyksistä helpoimmaksi oppia ja ominaisuuksiltaan kattavimmaksi. Vaikka PHP oli ohjelmoijalle entuudestaan tuttu, Zend Framework oli hankalin alustaa käyttöön sekä edellytti eniten koodausta. Ruby on Rails oli nopein ottaa käyttöön ja ominaisuuksiltaan Grailsin tasoa, mutta syntaksiltaan hankalampi Java-ohjelmoijalle.

Avainsanat ja -sanonnat: Skriptikieli, Web-ohjelmointi, ohjelmistokehys.
CR-luokat: D.2.6, D.1.5

1. Johdanto

Web-sovellusten määrän kasvaessa 1990-luvun puolivälissä hyvien suunnittelumallien puuttuminen tuotti paljon huonosti suunniteltua ja puutteellisesti toteutettua ohjelmakoodia. Tämä voi johtaa siihen, että ohjelman ylläpitäminen on hyvin vaikeaa tai jopa mahdotonta. Ratkaisuksi on kehitetty useita eri suunnittelumalleja, jotka auttavat ohjelmoijaa ratkaisemaan erilaisia ohjelmointiin liittyviä ongelmia. Suunnittelumallien yksi tärkeimmistä tehtivistä on helpottaa ohjelmistojen ylläpidettävyyttä.

Suunnittelumallien synty on suurilta osin ohjelmistokehysten ansiota. Ohjelmoinnissa ei useinkaan enää riitä pelkkä ohjelmointikieli itsessään, vaan yleensä rinnalle otetaan erilaisia jaettuja kirjastoja (libraries) ja ohjelmistokehyksiä nopeuttamaan projektin valmistumista sekä helpottamaan kehitystä.

Ohjelmistokehykset tekevät ohjelmoijan puolesta huomattavan paljon asioita. Ohjelmistokehys pitää usein sisällään valmiin suunnittelumallin, joka helpottaa projektin aloittamista ja nopeuttaa itse toteutusta. Ohjelmointityötä hel-

pottavat ohjelmistokehysten valmiit funktiot ja toiminnallisuudet, joita skriptikieleen ei itseensä ole toteutettu. [Poteet, 2008].

2. Skriptikielet

Ohjelmointikielien syntyvaiheessa ohjelmat kirjoitettiin monimutkaisilla symbolisilla konekielillä. Niiden jälkeen ohjelmointikielet ovat kehittyneet ymmärrettävämmiksi suoritustehokkuuden ja muistinkäytön kustannuksella. Tietokoneiden suorituskyvyn kasvaessa ei muistin käyttöllä ja suoritustehokkuudella ollut enää niin suurta merkitystä, vaan alettiin kiinnittää huomiota ohjelmoinnin tehokkuuteen ja helppoutteen. Skriptikielien helppous perustuu seuraaviin seikkoihin: niitä ei tarvitse kääntää, tyypittömyys, sisäänrakennetut funktiot ja automaattinen muistinhallinta [Prechelt, 2002]. Usein juuri nämä ominaisuudet määrittävät skriptikielen ja juuri näiden ominaisuuksien takia skriptikielet eivät ole yhtä tehokkaita kuin muut ohjelmointikielet.

Edellä mainittujen määritelmien ansiosta skriptikielet ovat syntaksiltaan varsinaisia ohjelmointikieliä yksinkertaisempia ja soveltuvat erinomaisesti dynaamisten web-sovellusten luomiseen. Skriptikielet soveltuvat hyvin web-ohjelmointiin siksi, että kehitystyö on nopeaa, eivätkä web-sovellukset useinkaan ole niin komplekseja, ettei skriptikieli olisi tarpeeksi tehokas.

2.1. PHP

PHP (Personal Home Page tools) on verrokeista yleisimmin käytetty web-ohjelmointiin suunniteltu skriptikieli [Tiobe, 2008]. PHP:n kehitys alkoi vuonna 1994 Rasmus Lerdorfin toimesta, kun hän halusi korvata kotisivujensa Perl-skriptit. PHP on kehitetty C-kielellä, ja Lerdorf julkaisi PHP:n kesäkuussa 1995 nopeuttaakseen virheiden korjausta ja parantaakseen ohjelman laatua.

PHP:n uusin versio on kirjoittamishetkellä PHP 5, jossa on parannettu huomattavasti oliopohjaista ohjelmointia ja suorituskykyä. PHP:n suosio on luultavasti sen yksinkertaisen syntaksin ansiota. Kaikkien PHP-skriptien on oltava erottimien `<?php` ja `?>` välissä. Mikään erottimien ulkopuolella oleva koodi ei mene PHP-parserin läpi, vaan ne tulostetaan näytölle normaalina tekstinä. Muuttujat ovat dynaamisia ja niiden etuliitteenä toimii dollari-merkki ($). Toisin kuin funktioiden ja luokkien nimissä, muuttujissa isot ja pienet kirjaimet katsotaan eri kirjaimiksi. PHP seuraa paljon C:n, C++:n, Javan ja Perlin syntaksia lohkojen rakenteissa, kuten If-ehto, for-, ja while-silmukka, sekä funktioiden return-määre. Koodiesimerkissä 1 on yksinkertainen PHP-luokka, jossa esimerkkejä PHP:n syntaksista.

```
<?php
class Book {
  private $title;
  private $authors = array();

  function Book($name) {
       $this->name = $name;
  }

  function getTitle() {
       return $this->title;
  }

  function addAuthor($author) {
       array_push($this->authors, $author);
  }

  function printAuthors() {
       foreach($authors as $author) {
            echo $author . ”<br />”;
       }
  }
}
?>

$book = new Book(”Harry Potter”);
$book->addAuthor(“J. K. Rowling”);
$book->printAuthors();
```

Koodiesimerkki 1: PHP-luokka, jossa rakennin ja kolme funktiota.

Esimerkistä näkee, että jokainen lauseke loppuu puolipisteeseen ja privaatti-attribuutteihin on viitattava aina this-määreellä. PHP:ssa taulut eivät ole olioita, mutta niille on useita sisäänrakennettuja funktioita helpottamassa taulujen manipulointia. PHP:ssa on myös monia muita sisäänrakennettuja funktioita, kuten merkkjonoihin ja matemaattisiin kaavoihin liittyviä apufunktioita, joita ei käytetä oliopohjaisesti.

2.2. Groovy

Skriptikielien helppo omaksuminen ja nopea koodaaminen ovat houkutteleva vaihtoehto kevyille ohjelmille. Joskus kuitenkaan ei haluta luopua varsinaisten ohjelmointikielten ominaisuuksista, mutta halutaan skirptikielien mukavuutta. Groovy on Java 2 -alustalla toimiva ketterä ja dynaaminen skriptikieli, jonka tarkoituksena on yhdistää skriptikielien mukavuus Javan toiminnallisuuksiin. Useat syntaktiset ominaisuudet, joita ei Javasta löydy, ovat hyödyllisiä muissa

kielissä. Juuri näitä ominaisuuksia hyödynnetään Groovyssa ja sallitaan ohjelmoijien silti käyttää Javan API:a.

Groovyn ominaisuuksiin kuuluu muun muassa dynaamiset muuttujat ja syntaksin lyhenteet. Staattiset muuttujat vaativat aina muuttujan tyypin esittelyn ennen arvon asettamista muuttujaan. Mikäli esittelyä ei kuitenkaan hoideta, tapahtuu useimmiten käännösvaiheessa virhe. Dynaamiset muuttujat sallivat arvon asettamisen ja sen perusteella kieli asettaa käännösvaiheessa muuttujalle tyyppinsä. Suurin osa skriptikielistä tukee dynaamisia muuttujia, mutta joissakin skriptikielissä voi käyttää myös staattisia muuttujia. Dynaamisten muuttujien etuna on lisäksi mahdollisuus muuttaa muuttujan tyyppiä ajonaikaisesti.

Syntaksin lyhenteet yksinkertaistavat Javassa käytettäviä pitkiä ilmauksia, esimerkiksi ruudulle tulostaminen. Koodiesimerkeissä 2 ja 3 verrataan Javan ja Groovyn koodeja, joissa molemmissa sama toiminnallisuus.

```
import java.util.StringTokenizer;
public class JavaTokenizer {
   public static void main(String[] args) {
      String text = "Teksti, joka pilkotaan";
      StringTokenizer tokenizer = new StringTokenizer(string);
      while(text.hasMoreTokens())
         System.out.println(text.nextToken());
      }
}
```

Koodiesimerkki 2: Java-toteutus merkkijonon pilkkomisesta ja tulostamisesta

```
text = "Teksti, joka pilkotaan"
tokenizer = new StringTokenizer(string)
while(text.hasMoreTokens())
   println text.nextToken()
```

Koodiesimerkki 3: Groovy-toteutus merkkijonon pilkkomisesta ja tulostamisesta

Groovy-esimerkistä huomaa, ettei koodin tarvitse olla luokan sisällä, muuttujat eivät tarvitse esittelyä, StringTokenizer-luokkaa ei tarvitse tuoda, puolipisteitä ei tarvitse lausekkeiden perässä eikä tulostamista tarvitse kutsua System.out-kirjaston kautta. Tämä nopeuttaa ohjelman kirjoittamista ja selkeyttää koodin lukemista etenkin isommissa ohjelmissa.

2.3. Ruby

Yukihiro Matsumoton vuonna 1995 julkaisema Ruby on verrokeista toiseksi yleisimmin käytetty skriptikieli [Tiobe, 2008]. Ruby on moderni täysin oliopohjainen, syntaksiltaan lyhyt ja helppolukuinen skriptikieli, joka on helppo omak-

sua, mikäli käyttäjällä on kokemusta ohjelmistorajapintoihin (API) pohjautuvista kielistä.

Rubyn ominaisuuksiin kuuluu muun muassa tiedon kapselointi ja luokkien perintä, moniperintä mukaan luettuna. Kaikki Rubyn datatyypit, kuten merkkijonot, numero-arvot ja jopa vakiot sekä luokat, ovat olioita. Kuten Groovy, ja useimmat muut skriptikielet, myös Ruby tukee dynaamisia muuttujia. [Østerlie, 2002].

Rubyn lohkotukset eivät ala eivätkä pääty aaltosulkuihin, kuten useimmissa muissa kielissä, vaan sen sijaan lohkot päättyvät aina ”end”-määreeseen. Luokka-attribuutit määritellään @-merkillä ja staattiset attribuutit kahdella @-merkillä (@@) (ks. koodiesimerkki 4).

```
class BankAccount
   def interest_rate
        @@interest_rate = 0.2
   end

   def accountNumber
        @accountNumber
   end

   def accountNumber=( value )
        @accountNumber = value
   end

   def accountName
        @accountName
   end

   def accountName=( value )
        @accountName = value
   end

end

account = BankAccount.new()
account.accountNumber = "1234567980"
account.accountName = "Peter Petrelli"
```

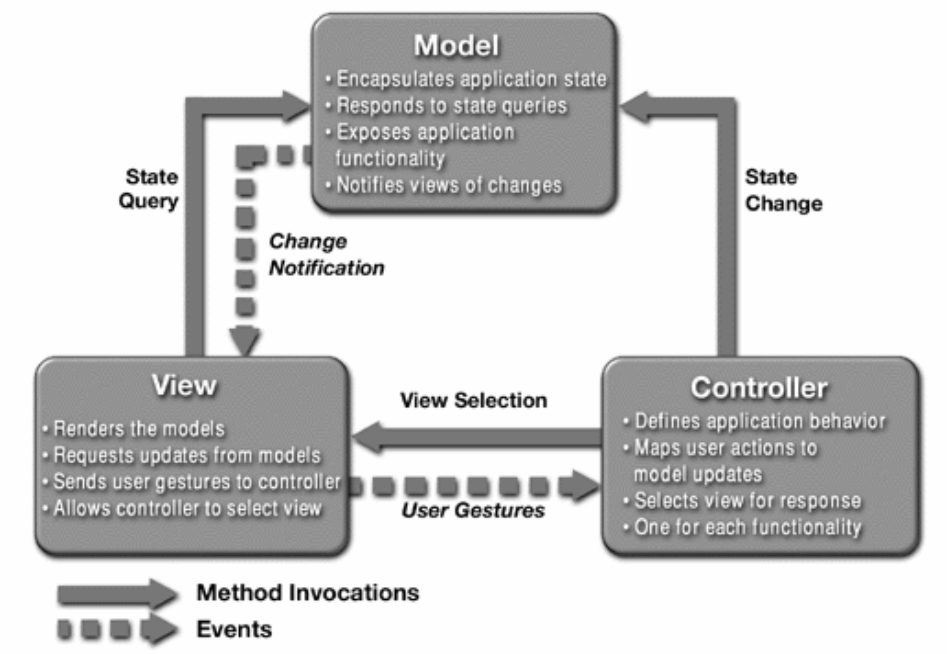
Koodiesimerkki 4: Ruby-kielellä toteutettu luokka ja instanssin luominen.

3. MVC-suunnittelumalli

Suunnittelumallit on luotu ratkaisemaan ohjelmointiin liittyviä ongelmia. Etenkin suuremmat ohjelmistot tarvitsevat hyvin suunnitellun arkkitehtuurin, jotta niitä olisi helpompi ylläpitää ja laajentaa. MVC-suunnittelumallin kehitti Trygve Reenskaug [2003] SmallTalk-ohjelmointikielelle vuonna 1979.

MVC-suunnittelumalli pilkkoo ohjelman kolmeen eri kokonaisuuteen: malliin, näkymään ja ohjaimeen (model-view-controller). Jaottelun päätehtävänä on erottaa sovelluslogiikka näkymästä. Mallin tehtävänä on ylläpitää ohjelman tai olion tilaa, mikä web-sovelluksissa yleensä tapahtuu tietokantaan tallentamalla. Mallin tehtävänä on myös välittää tietoa omasta tilastaan näkymälle. Näkymä renderöi mallin sisällön käyttäjälle ja välittää käyttäjän toimet ohjaimelle. Yhdellä mallilla voi olla useita näkymiä käyttötarkoituksesta riippuen. Web-sovelluksissa ohjain käsittelee http-pyynnöt ja välittää tarvittavan tiedon malleille. Kun malli palauttaa vastauksen, ohjain antaa vastauksen perusteella käyttäjälle uuden näkymän (ks. kuva 1). [Sun Microsystems, 2008].

MVC-suunnittelumallia on hyödynnetty Grailsissa, Ruby on Railsissa sekä Zend Frameworkissa samalla tavoin. Ainoastaan Zend Frameworkissa MVC ei rakennu automaattisesti, vaan ohjelmoijan pitää tehdä itse rakenne.



Kuva 1. MVC-arkkitehtuurimallin suhteiden kuvaus. [Sun Microsystems, 2002].

4. Kehykset

Ohjelmistokehykset tarjoavat ohjelmoijille mahdollisuuden kehittää joustavampia ja vakaampia järjestelmiä tehokkaasti. Ohjelmistokehykset auttavat nopeuttamaan kehitysprosessia tarjoamalla tarpeellisia ominaisuuksia valmiiksi

toteutettuna. Ohjelmistokehyksille on yleensä ominaista sisältää käyttäjätilien hallinnoinnin, välimuistin hallinnan (caching), tietosuojaominaisuusuuksia, arkkitehtuurin ja paljon muuta. Näin ohjelmistokehykset auttavat ohjelmoijaa keskittymään tärkeimpiin asioihin, kuten suunnitteluun ja esimerkiksi projektin johtamiseen. Yleisimmin esiintyvät rutiinit, kuten CRUD (Create, Read, Update, Delete) -operaatiot ovat ohjelmistokehyksen ansiosta nopea toteuttaa. [Poteet, 2008].

4.1. Zend Framework

Zend Framework sai alkunsa vuoden 2005 alussa, kun monet muut ohjelmistokehykset kuten Ruby on Rails ja Spring Framework kasvattivat suosiotaan web-kehittäjien yhteisöissä. Zend Framework julkistettiin ensimmäistä kertaa Zendin pitämässä konferenssissa [Zend, 2005]. Samaan aikaan tuntemattomampia ohjelmistokehyksiä annettiin ladattavaksi PHP-yhteisölle täyttämään samankaltaisia web-kehittämiseen liittyviä olennaisia tarpeita. Zend Frameworkin suunnittelijat tähtäsivät helppokäyttöisyyden ja nopean tuotekehityksen yhdistämiseen, joita uusilla ohjelmistokehyksillä jo oli. PHP-yhteisöt arvostivat yksinkertaisuutta, avoimuutta ja käytännöllisyyttä, jotka suunnittelijat ottivat myös Zend Frameworkin tavoitteiksi.

Zend Framework tarjoaa komponentteja MVC ja Table Gateway -suunnittelumalleille, jotka ovat käytössä useimmissa Web-sovelluksissa. Arkkitehtuuri perustuu hyvin paljon Paul M. Jonesin kehittämään Solar Framewrokkiin, jossa on paljon yhtäläisyyksiä Zend Frameworkin kanssa.

On olemassa kolmen tyyppisiä Web-ohjelmistokehyksiä:

1. Kiinteän infrastruktuurin tarjoavat: Symfony, Solar, Ruby on Rails, Grails ja Django
2. Komponenttikirjaston tarjoavia: ezComponents ja PEAR
3. Molemmat edellä mainitut tarjoava: Zend Framework.

Zend Framework ei siis tarjoa ainoastaan kiinteää infrastruktuuria, vaan myös kattavan komponenttikirjaston. Komponenttien rakenne Zend Frameworkissa on suunniteltu käsittämään mahdollisimman vähän riippuvuuksia muihin komponentteihin. Tämä löyhästi kytketty (loosely-coupled) arkkitehtuuri sallii ohjelmistokehittäjien käyttää Zend Frameworkin komponentteja erikseen ja irrallaan muista. Komponenttikirjasto sisältää paljon toiminnallisuuksia, joita jokainen web-ohjelmistokehittäjä tarvitsee usein. Olio-pohjaisessa ohjelmoinnissa riippuvuudet ja kytkennät määrittävät mitkä komponentit tarvitsevat muita komponentteja toimiakseen. Suurin etu löyhästi kytketystä arkkitehtuurista on, että se sallii ohjelmistokehittäjien käyttää komponentteja erikseen. Zend Framework ei kuitenkaan sisällä minkäänlaista komponenttia tai konfiguraatio-

mekanismia, millä mallien ja ohjaimien riippuvuuksia voisi hallita. [Cargnelutti, 2008].

Ohjelmistokehyksen suorituskykyyn vaikuttavat monet eri tekijät, erityisesti http-palvelimen konfiguraatio. Kuitenkin sovelluksen suunnittelu tekee suuria eroja sivuston nopeuteen. Eräät mittaustulokset [Ekerete, 2008] näyttävät, että Zend Framework olisi hitaampi kuin muut vertailussa mukana olleet ohjelmistokehykset. Vertailussa oli mukana myös Ruby on Rails.

Zend Frameworkillä ei ole aivan täysin loppuun vietyä tietojen tarkistusta, kuten Grailsilla ja Ruby on Railsilla. Ohjaimessa on kuitenkin helppo luoda objekti ja tehdä tietojen tarkastus ennen tietokantaan syöttöä, kun Zend Framework hoitaa ”likaisimmat” työt taustalla (ks. koodiesimerkki 5). Automatisoidumpaa merkkijonojen tarkistusta ja muokkaamista olisi voinut olla tarjolla, joka hoitaisi esimerkiksi mahdolliset tietokantainjektiot.

```
 // Luodaan email-objekti
 $email = new Email();
 // Asetetaan attribuutit
 $email->set($this->_request->getParams());
 $validator = new ModelValidator();
 // Kysytään menikö tarkastukset läpi
 if($validator->isValid($email)) {
     $data = array(
          'sender' => $sender,
          'receiver' => $receiver,
          'subject' => $sender,
          'text' => $receiver
     );
     // Syötetään sähköpostin tiedot tietokantaan ja lähetetään
     $email->insert($data);
     $mailer = new Mailer();
     $mailer->send($this->_request->getParams());
}
```

Koodiesimerkki 5: Email-objektin tietojen tarkistaminen, tietokantaan syöttäminen ja lähettäminen.

Edellisessä koodiesimerkissä email-objektin luonti tapahtuu malli-luokan kautta, johon piti erikseen toteuttaa set-funktio. Funktio määrittää sähköpostiosoitteille validoinnin ja asettaa jokaisen attribuutin. Attribuuttien rajoitukset määritellään erillisessä taulussa, jota Zend Framework osaa automaattisesti etsiä (ks. koodiesimerkki 6).

```php
<?php
class Email extends Zend_Db_Table
{
    // Emailin vaatimat attribuutit
    public $sender = '';
    public $receiver = '';
    public $subject = '';
    public $text = '';
    public $sentTime = '';

    // Tietojen tarkastamiseen vaadittu taulu
    public $constraints = array(
        "subject" => "Zend_Validate_NotEmpty",
        "text" => "Zend_Validate_NotEmpty",
    );

    // Asetetaan attribuutit ja sähköpostien tarkastusketju
    public function set($params)
    {
         $validatorChain = new Zend_Validate();
         $validatorChain->addValidator(new Zend_Validate_EmailAddress())
                        ->addValidator(new Zend_Validate_NotEmpty());
         $this->constraints["sender"] = $validatorChain;
         $this->constraints["receiver"] = $validatorChain;

         foreach($params as $key => $value) {
              $this->$key = $value;
         }
         $this->sentTime = date("Y-m-d H:i:s");
    }
}
?>
```

Koodiesimerkki 6: Email-luokka joka määrittää emailin tarvitsemat attribuutit ja niiden rajoitukset.

4.2. Grails

Grails on Javan skriptikielen Groovyn ohjelmistokehys, josta ilmestyi versio 1.0 helmikuussa 2008. Groovyn yksi suurimmista eduista on, että ohjelmissa voi käyttää Groovyn oman syntaksin lisäksi kaikkia Javan ominaisuuksia. Suurin haitta on ohjelmistokehyksen nuori ikä, jonka johdosta bugeja ja kehittämistä on vielä paljon. Grails-yhteisö ei myöskään ole vielä kovin suuri muihin kehyksiin verraten, minkä takia apua ja esimerkkejä ei juuri löydy muualta kuin Grailsin omilta kotisivuilta.

Grails hyödyntää tunnettuja Javan teknologioita, Spring- ja Hibernate-ohjelmistokehyksiä, ja helpottaa näiden käyttöä hoitamalla esimerkiksi kaiken XML-konfiguroinnin automaattisesti. Eniten Grailsin oppimista hepottaa automaatti-

nen projektin ja MVC-mallin luonti. Perinteisiä tietokantakyselyitä ei Hibernaten ansiosta tarvitse kirjoittaa, sillä jokaiselle mallille (model) luodaan dynaamiset funktiot, joiden kautta tieto haetaan tietokannasta. Tietokantaan tallentaminen tapahtuu myös dynaamisen funktion kautta. (ks. koodiesimerkki 7).

```
// Luodaan email-objekti, tallennetaan se tietokantaan ja lähetetään
def email = new Email(params)
if(!email.hasErrors() && email.save())
  emailerService.sendEmail(params)

// Haetaan kaikki emailit
def emails = Email.findAll()

// Haetaan emailit jonka lähettäjänä on “Jukka”
def emails = Email.findAllBySender("Jukka")
```

Koodiesimerkki 7: Grailsin luomia dynaamisia Hibernatea hyödyntäviä metodeja.

Malli-luokan kautta Grails osaa automaattisesti luoda tietokantataulut ja näiden keskinäiset relaatiot. Malli pitää sisällään ainoastaan luokan sisältämät relaatiot muihin luokkiin, mallin attribuutit sekä attribuuttien rajoitteet (ks. koodiesimerkki 8).

```
class Book {
    static hasMany = [authors: Author]
    String title
    Date publishDate = new Date()

    static constraints = {
         title (blank:false, maxLength:100)
         publishDate (blank:false)
    }
}
```

Koodiesimerkki 8: Book domain -luokka Grails-kehyksellä. Kirjoilla voi olla useita kirjoittajia ja kirja sisältää nimen sekä julkaisupäivän. Nimi ei saa olla tyhjä eikä yli 100 merkkiä pitkä.

4.3. Ruby on Rails

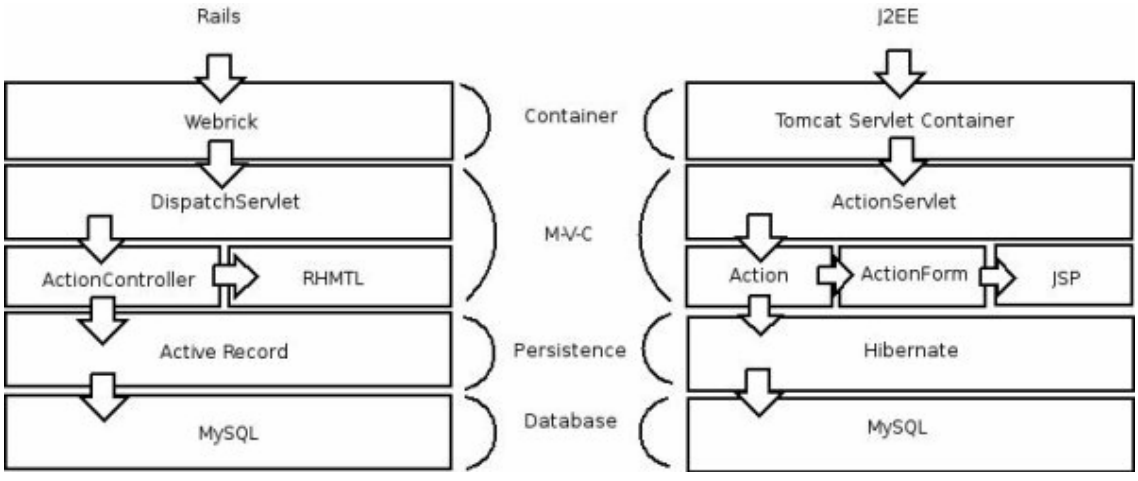
Ruby on Rails (jatkossa Rails) on avoimen lähdekoodin ohjelmistokehys Ruby-kielelle. Railsin kehitti alun perin David Heinemeier Hansson [2006] 37signals-yrityksen työkaluksi. Rails julkaistiin ensimmäistä kertaa avoimen lähdekoodin alla kesäkuussa 2004, mutta kehitysoikeudet projektille jaettiin vasta helmikuussa 2005 [Ruby on Rails, 2008].

Rails toimii kaikilla yleisimmillä käyttöjärjestelmillä, kuten Linux, OS X ja Windows, sekä tukee yleisimpiä vapaan lähdekoodin tietokantoja, kuten MySQL ja PostgreSQL. Web-ohjelmistokehittäjille luultavasti yksi yleisimmistä tehtävistä on tehdä kyselyitä tietokannasta. Rails on pyrkinyt tekemään tietokantayhteyksistä tehokkaampia ja yksinkertaisempia käyttäen Railsin Active records -kirjastoja. [Mommaerts, 2006].

Railsilla on kaks sääntöä, joita kehitystyössä on pyritty pitämään mielessä. Ensimmäinen sääntö, DRY (Don't Repeat Yourself), ohjeistaa, ettei ohjelmoijan tarvitse kirjoittaa samaa toiminnallisuutta kuin ainoastaan yhden kerran. Tämä mahdollistaa helpomman ylläpidon ja kehityttämisen, koska tällöin ei ole tarvetta pitää järjestelmän useita eri osia synkronoituna keskenään. [Mommaerts, 2006].

Ensimmäisen säännön kanssa käsi kädessä kulkee 'Convention over Configuration' -sääntö. Railsilla on sarja koodi- ja nimeämiskäytäntöjä, joiden tarkoituksena on eliminoida tarve konfiguroida jokaista Rails-sovelluksen eri toiminnallisuusaluetta. Ainoastaan erityisimmät tapaukset vaativat konfigurointia, kuten ulkoiset resurssit, joita ohjelmoija ei itse kontrolloi. Käyttämällä näitä kahta filosofiaa, Rails antaa ohjelmoijan luoda enemmän toiminnallisuuksia vähemmällä koodimäärällä samassa ajassa, kuin esimerkiksi Javalla tai .NET:llä, silti pitäen ylläpidon helppona jälkikäteenkin. [Mommaerts, 2006]

Myös Rails käyttää MVC-arkkitehtuuria, ja se sisältää useita erillisiä kirjastoja. Kaksi tärkeintä MVC:n toteuttavaa kirjastoa ovat ActionPack ja ActiveRecord. Ensiksi mainittu hoitaa näkymä- ja ohjainkerroksia ja kaikki mallit perivät aina jälkimmäisen. Kuvassa 2 on vertailtu J2EE- (Struts) ja Rails-ohjelmistokehysten rakenteita. Kuvasta huomaa, ettei eroa ole juuri muualla, kuin ohjain-kerroksessa, mutta muuten samat osat ovat toteutettuna eri tekniikoilla.



Kuva 2. Rails ja J2EE rakennevertailu [Rustad, 2005]

Railsissa ohjaimen CRUD-operaatiot generoidaan automaattisesti, aivan kuten Grailsissa. Tämä helpottaa ja nopeuttaa ohjelmoijan työtä suuresti. Tarvittavat muutokset on helppo kirjoittaa valmiin koodin sekaan, sillä koodia on toiminnallisuuteen nähden huomattavan vähän. Koodiesimerkissä 9 on sähköpostien luontifunktio, joka on muuten Railsin automaattisesti generoimaa (hieman karsittuna), lukuun ottamatta Mailer-luokan sähköpostinlähetys-kutsua.

```
# Funktio uuden sähköpostin luontiin
def create
     # Luodaan uusi sähköposti-objekti välitettyjen parametrien perusteella
     @email = Email.new(params[:email])

     respond_to do |format|
     # Jos tallennus onnistuu
     if @email.save
         # Lähetä sähköposti
         Mailer.deliver_mail(params[:email])
     end
end
```

Koodiesimerkki 9: Sähköposti-objekti luonti ja lähetys Railsilla.

5. Testijärjestelyn kuvaus

Jokaisen kehyksen arviointiin käytännössä kehitettiin kullakin ohjelmistokehyksellä yksinkertainen sähköpostinlähetys-ohjelma. Jokainen toteutus pitää sisällään syötteiden tarkastuksen ja lähetettyjen sähköpostien tallentamisen tietokantaan.

Käytössä oli kunkin skriptikielen ja kehyksen uusimmat saatavilla olevat versiot Windows XP -käyttöjärjestelmälle. Tietokantana käytössä oli MySQL-tietokanta ja lähtevän postin palvelimena Gmailin tarjoama palvelin. Grails ja Rails käyttivät omia sisäänrakennettuja http-palvelimia ja Zend Frameworkilla käytössä oli Apachen http-palvelin. Testijärjestelyissä kaikilla testattavilla kehyksillä oli käytössä Googlen smtp-palvelin.

Testin toteuttajalla (ohjelmoijalla) on aiempaa kokemusta PHP:sta, Javasta ja Grailsista. Syntaksin tuttuus ja tuttuihin kieliin mieltyminen oli varmasti suuri vaikutustekijä testiä tehtäessä, joten Ruby jäi tässä suhteessa luultavasti hieman muiden varjoon.

6. Testin tulokset

Sähköpostiohjelmien rakentaminen on suhteellisen yksinkertaista riippumatta ohjelmointikielestä. Usein jo pelkkä ohjelmointikieli pitää sisällään kirjastot

sähköpostien lähetykseen, mutta viimeistään ohjelmistokehyksissä on tuki tähän. Ennen sähköpostien lähetystä on oleellista tarkistaa lähetyksessä tarvittavat kentät, jotta käyttäjälle voidaan antaa asianmukainen palaute mahdollisista virheistä. Tietojen tarkastus tulee usein vastaan kaikissa web-pohjaisissa sovelluksissa, joissa on interaktiota yhden tai useamman käyttäjän kanssa. Kaikilla kehyksillä oli valmiiksi automatisoitu järjestelmä tarkastuksia varten. Grails oli kehyksistä ainut, jossa oli valmis sähköpostiosoitteen tarkistus sisäänrakennettuna.

Jokainen onnistuneesti lähetetty sähköposti tallennettiin tietokantaan. Ohjelmistokehyksistä kaikki tarjosivat tietokanta-rajapinnan, jonka ansiosta käytetyllä tietokannalla ei ollut merkitystä. Tietokantojen käsittely oli jokaisella ohjelmistokehyksellä erittäin yksinkertaista ORM-tekniikan (Object-relational mapping) hyödyntämisen ansiosta.

Huomattavasti helpotusta käyttöliittymäohjelmointiin ja koodin ymmärrettävyyteen tuo ohjelmistokehyksien mahdollisesti mukana tulevat mallinejärjestelmät (template engine). Ainoastaan Zend Frameworkissa ei ollut sisäänrakennettua mallinejärjestelmää, mutta PHP:lle on olemassa Zend Frameworkiin kytkettäviä irrallisia mallinejärjestelmiä, kuten Smarty [2008].

Taulukossa 1 on vertailua ohjelmistokehysten eri ominaisuuksista ja testiohjelmien koodimääristä. Taulukossa listatut ominaisuudet ovat useimmissa projekteissa hyvin oleellisia ja tukevat usein esille tulevia ohjelmoinnillisia ongelmia. Testiä tehdessä oppimista tukivat parhaiten CRUD-operaatioiden automaattinen luonti. Koodaustyötä tehostavia tekijöitä olivat automatisoitu attribuuttien tarkistus ja CRUD-operaatioiden luonti, mallinejärjestelmä sekä ulkoiset liitännäiset.

	Zend Framework	Grails	Ruby on Rails
Koodin sanamäärä	314	212	251
Koodin rivimäärä	125	58	78
Tietokantariippumattomuus	X	X	X
URL-mappays	X	X	X
Automaattinen CRUD		X	X
Automatisoitu attribuuttien tarkistus	X	X	X
Automatisoitu projektin luonti		X	X
Mallinejärjestelmä		X	X
http-palvelin		X	X
Ulkoiset liitännäiset (plugins)	X	X	X

Taulukko 1. Ohjelmistokehysten ominaisuusvertailu

7. Yhteenveto

Tässä tutkielmassa käsiteltiin kolmen ohjelmistokehysten käyttöä ja käyttöönottoa web-ohjelmointiympäristöissä. Ohjelmistokehyksistä vertailtiin Groovyn Grails, Rubyn Ruby on Rails ja PHP:n Zend Framework. Vertailukohdaksi valittiin kullakin ohjelmistokehyksellä kehitettävät yksinkertaiset sähköpostilähetysohjelmat. Lähetettävän sähköpostin kaikki kentät tarkistetaan ennen lähetystä ja lähetetyt viestit tallennetaan tietokantaan. Kaikki ohjelmistokehykset nopeuttavat ja helpottavat huomattavasti kehitystyötä valmiiksi toteutetuilla ominaisuuksillaan. Ominaisuuksiltaan parhaimmaksi osoittautui Grails, jolla muun muassa selviytyy vähimmällä koodimäärällä ja oppiminen oli helpointa pitkällisen automatisoinnin ansiosta. Tässä kohtaa tarvitsee muistuttaa kuitenkin ohjelmoijan Java-taustaisuudesta, joka helpottaa huomattavasti Groovy-kielen kirjoittamista ja ymmärtämistä. Zend Frameworkilla oli työläintä ja hankalinta aloittaa siitäkin huolimatta, että ohjelmoija on tehnyt hyvin paljon PHP:lla töitä. Aloituksen jälkeen työ helpottui, kun MVC-mallin sai rakennettua ja Apachen konfiguroitua. Zend Framework oli myös ominaisuuksiltaan muita (Grails ja Rails) vajavaisempi, mutta löyhä kytkentä mahdollistaa ohjelmistokehyksen irrallaan käyttämisen. Rails-ohjelma oli nopein toteuttaa. Railsilla oli paljon yhtäläisyyksiä Grailsin kanssa ja hyvä vaihtoehto, mikäli Javan syntaksi ei ole tuttu tai mielekäs.

Viiteluettelo

[Cargnelutti, 2008] Federico Cargnelutti, Zend Framework Architecture, PHP::Impact. http://phpimpact.wordpress.com/2008/07/28/zend-framework-architecture/. Checked 2.12.2008.

[Ekerete, 2008] Ekerete, PHP framework comparison benchmarks. http://www.avnetlabs.com/php/php-framework-comparison-benchmarks. Checked 5.12.2008.

[Grails, 2008] Grails, http://grails.org/, Checked 29.9.2008.

[Heinemeier Hansson, 2006] David Heinemeier Hansson, haastattelu helmikuu 2007.

[Henry, 2006] Kevin Henry, A crash overview of groovy. *Crossroads* **12** (2006).

[Mommaerts, 2006] Nico Mommaerts, An introduction to web development using the Ruby on Rails Framework, *Methods & Tools* (2006). http:// www.methodsandtools.com/archive/archive.php?id=47.

[Østerlie, 2002] Thomas Østerlie, Ruby: Yet another scripting language? *Linux Journal* **2002** (2002).

[Ousterhout, 1998] John K. Ousterhout, Scripting: higher level programming for the 21st century, *IEEE Computer* **31** (1988), 23-30.

[PHP, 2008] PHP: Hypertext preprocessor. http://www.php.net. Checked 29.9.2008

[Poteet, 2008] Chris Poteet, Frameworks Round-Up: When To Use, How To Choose? http://www.smashingmagazine.com/2008/01/04/frameworks-round-up-when-to-use-how-to-choose/. Checked 29.9.2008.

[Prechelt, 2002] Lutz Prechelt, Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java In: M. Zelkowitz (ed.) *Advances in Computers* **57**, Academic Press, 2003. pp. 205-270.

[Reenskaug, 2003] Trygve Reenskaugh, MVC Xerox Parc 1978-79. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. Checked 29.9.2008.

[Ruby, 2008] Ruby on Rails, Ruby/Ruby on Rails programming tutorials. http://www.meshplex.org/wiki/Ruby/Ruby_on_Rails_programming_tutorials. Checked 29.9.2008.

[Ruby on Rails, 2008] Ruby on Rails. http://www.rubyonrails.org. Checked 8.12.2008.

[Rustad, 2005] Aaron Rustad, Ruby on Rails and J2EE: Is there room for both?, IBM. http://www.ibm.com/developerworks/web/library/wa-rubyonrails. Checked 8.12.2008.

[Smarty, 2008] Smarty template engine. http://www.smarty.net/. Checked 28.11.2008.

[Sun Microsystems, 2002] Sun Microsystems, Java BluePrints Model-View-Controller. http://java.sun.com/blueprints/patterns/MVC-detailed.html. Checked 29.9.2008.

[Sun Microsystems, 2008] Sun Microsystems, J2EE Architecture Approaches. http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html. Checked 29.9.2008.

[Tiobe, 2008] Tiobe Sofware, TIOBE Programming Community Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Checked 29.9.2008.

[Zend, 2005] Zend, Zend Announces Industry-Wide PHP Collaboration Project at its Inaugural PHP Conference.

[Zend Framework, 2008] Zend Framework. http://framework.zend.com/. Checked 29.9.2008.

Ketterä Scrum-metodi ohjelmistoprojektin hallintaan

Hannu Korhonen

Tiivistelmä.
Tässä tutkielmassa käsittelen yhtä yritysten suosituinta ketterää ohjelmistoprojektin hallinnan metodia, Scrumia. Esittelen Scrum-metodin prosessimallin sekä siihen liittyvät menetelmät ja organisaatioon olennaisesti kuuluvat roolit. Lisäksi vertaan Scrum-metodin vahvuuksia aikaisempiin metodeihin verrattuna.

Avainsanat ja -sanonnat: Scrum, ketterä ohjelmistokehitys, agile, ohjelmistoprojekti.
CR-luokat: D.2.1

1. Johdatus ketteriin menetelmiin

Ohjelmistokehitysmetodeja on ollut käytössä yrityksissä jo 1960-luvulta lähtien. Vuosien saatossa metodeista on syntynyt erilaisia variaatioita yritysten omiin tarpeisiin ja tutkimuskäyttöön. Jotkut näistä metodeista ovat säilyneet tähän päivään asti ja sopeutuneet uuteen ympäristöönsä. Ennen 1980-lukua ohjelmistokehitys yrityksissä oli muutamien ammattilaisten käsialaa [Royce, 1998]. Kehitysmetodit ja -työkalut olivat yrityskohtaisia. Sen ajan yleinen ohjelmistokehitysmetodi oli vesiputousmalli (waterfall model). Seuraavat pari vuosikymmentä olivat tekniikan kypsymisen ja standardoimisen aikaa ohjelmistojen kehityksessä. Kehitysmetodeista käytettiin sitä metodia, jolla yleensä onnistuttiin tekemään tuote jotakuinkin aikataulussa ja budjetissa. Työkalujen standardointi ja niiden käyttö yleistyi kuitenkin nopeasti. Moderni aika ohjelmistojen kehityksessä katsotaan alkaneeksi 2000-luvun alusta. Nykyään ohjelmistojen kehitys tapahtuu standardoiduilla työkaluilla ja kehitysmetodit ovat testattuja ja työn tulokset ovat mitattavia. Iteratiivinen kehitysmetodi on yleisesti käytössä yrityksissä. Monet ohjelmat rakentuvat valmiista komponenteista ja keskimäärin vain muutama kymmenen prosenttia koodista on yrityksien itsensä tekemää.

Miksi sitten niin moni ohjelmistoprojekti myöhästyy, ylittää budjetin tai pahimmassa tapauksessa epäonnistuu totaalisesti? Yleinen syy epäonnistumisiin on projektiorganisaation kyvyttömyys sopeutua muutoksiin, jotka johtuvat vaihtuvista tuotteen vaatimuksista asiakkailta sekä kohdeympäristön teknologisista muutoksista. Scrum-metodi pyrkii vastaamaan muutosten aiheuttamiin ongelmiin ohjelmistoprojekteissa olemalla joustava vaatimusten suhteen ja tuomalla kehittäjien taidot paremmin yrityksen sekä tiimin käyttöön.

2000-luvun alussa ohjelmistokehityksen ammattilaiset keskustelivat uusista ketteristä menetelmistä ohjelmistojen kehityksessä ja niiden tuomista eduista vanhoihin metodeihin nähden. Ketterien menetelmien kattotermiksi luotiin ”Agile methods”, eli ketterät menetelmät [Agile Manifesto, 2008]. Ketterien metodien sateenvarjon alle mahtuu Scrumin lisäksi monia muitakin ketteriä kehitysmetodeja, kuten Extreme Programming (XP), Crystal, Agile modeling, Adaptive software development, Feature driven development, TTD Test Driven Development tai DSDM [Wikipedia, 2008].

Agile-manifesti julistettiin vuonna 2001 Wasatchissa, Utahissa. Siinä todetaan seuraavat asiat, jotka ovat keskeisiä ketterille menetelmille [Agile, 2008; Manifesto, 2008]:

Me etsimme parempia keinoja ohjelmistojen kehittämiseen tekemällä sitä itse ja auttamalla siinä muita. Tässä työssämme olemme päätyneet arvostamaan

Yksilöitä ja interaktioita enemmän kuin prosesseja ja työkaluja
Toimivaa sovellusta enemmän kuin kokonaisvaltaista dokumentaatiota
Asiakasyhteistyötä enemmän kuin sopimusneuvotteluita
Muutokseen reagoimista enemmän kuin suunnitelman noudattamista.

Vaikka oikeallakin puolella on arvoa, me arvostamme vasemmalla olevia asioita enemmän.

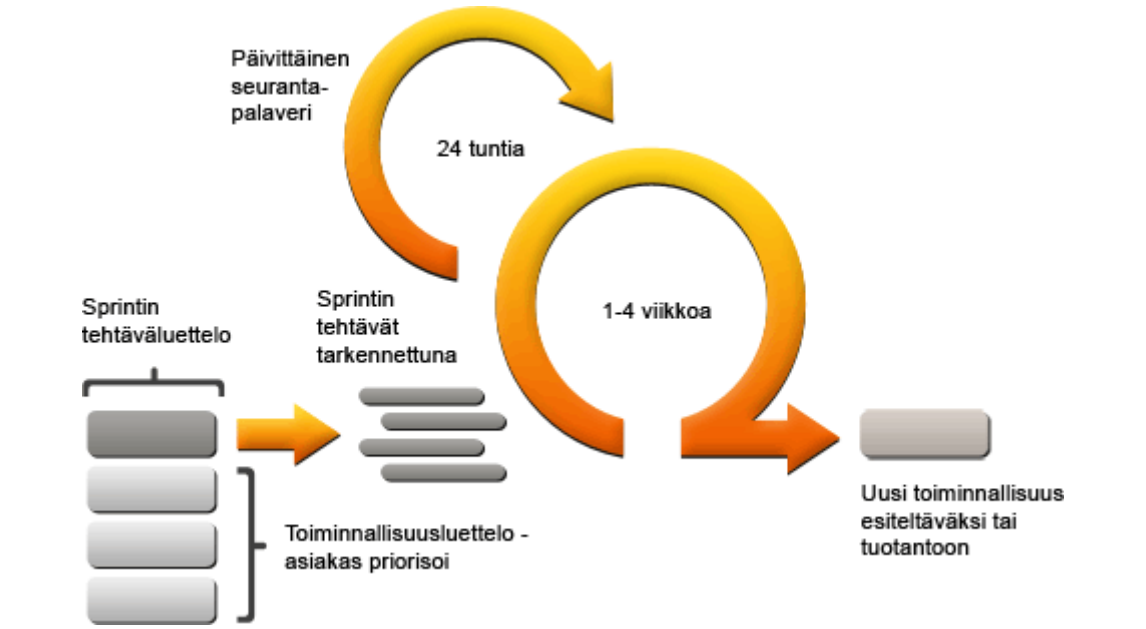
Scrum-metodi noudattaa manifestissa yksilöityjä teemoja käytännössä keinoin, jotka on havaittu toimiviksi. Interaktiota voidaan lisätä tarjoamalla tiimille työtilat, jotka tukevat vuorovaikutuksen syntymistä ja ajatusten vaihtoa. Dokumentointiin ei kiinnitetä niin paljon huomiota kuin vanhoissa metodeissa. Ohjelmiston iteraatioista pyritään esittelemään demoja jo varhaisessa vaiheessa, jolloin sovelluksen kehitystä voidaan esitellä kaikille osapuolille havainnollisesti. Lyhyet iteraatiot kehityksessä tarkoittavat hyvää reagointikykyä muutoksiin. Scrum-metodiin on pyritty hakemaan parhaat käytännöt eri kehitysmetodeista, jotka on havaittu käytännössä hyviksi ja tehokkuutta parantaviksi. Seuraavissa luvuissa tutustutaan tarkemmin Scrum-metodiin käytännössä sekä Scrumin vaatimiin rooleihin ja käytäntöihin. Luvusssa 2 käydään läpi Scrum-metodin viitekehys. Luvussa 3 tutustutaan Scrum-metodin käytäntöihin. Neljännessä luvussa käydään läpi Scrumin vahvuuksia verrattuna aikaisempiin kehitysmetodeihin. Lopuksi on yhteenveto Scrumista.

2. Scrum-metodi

Scrum-metodista on tullut viime vuosina erittäin suosittua ketterien projektien hallinnassa. Scrum on onnistunut lunastamaan paikkansa yrityksissä uutena ketteränä metodina, joka luo kilpailukykyä ja lisäarvoa tuotekehitykseen [Koskela, 2008]. Metodia voidaan soveltaa myös muilla tuotannonaloilla kuin vain ohjelmistotekniikassa. Scrum-metodi sopii kaikkiin projekteihin, joissa tarvitaan muutoksensietokykyä sekä joustavuutta ulkopuolelta tuleviin vaatimuksiin ja muuttuneisiin suunnitelmiin. Ohjelmistoprojektin ympärillä oleva maailma muuttuu jatkuvasti teknologian sekä kaupallisten vaatimusten mukaan, joten riskien hallintaan on hyvä varautua. Nämä muutokset voivat aiheuttaa vakavia haittoja ohjelmiston kehitykselle ja sen valmistumiselle aikataulussa tai budjetissa.

Scrum-tiimin ei tarvitse ympärillä olevista muutoksista välittää, vaan sen täytyy keskittyä olemassa olevaan tehtävälistaan ja sen sisältämien tehtävien implementoimiseen. Tehtävälista on priorisoitu, josta sprintteihin (sprint) valitaan aina kriittisimmät tai tärkeimmät tehtävät ensin. Muutokset ympäristössä tai vaatimuksissa tulevat näkyviin tehtävälistan kautta, josta ne sitten voidaan ottaa sprintteihin mukaan. Sprintti sisältää tietyn määrän tehtäviä, jotka tiimi arvioi saavansa valmiiksi sen aikana. Sprintti kestää noin 1-4 viikkoa riippuen sprintin luonteesta ja tiimistä itsestään. Päivittäisillä palavereilla (daily scrum) seurataan sprintin edistymistä sekä tuodaan esille ongelmia ja esteitä, jotta ne voidaan selvittää ja jatkaa työntekoa häiriöttä. Sprintin päätteeksi valmistuneet tehtävät esitellään ja arvioidaan niiden onnistumista ja merkitystä tuotteelle ja sen laadulle.

Scrum-metodin vahvuuksina voidaankin pitää sitä, että suunnittelijat pystyvät keskittymään täysillä vaatimusten toteuttamiseen samalla kun Scrum-mestari raivaa tieltä pois mahdollisia ongelmia [Schwaber and Beedle, 2001]. Scrum-metodia käyttävällä projektilla on selkeä tavoite sekä vaatimukset, jotka sen tulee toteuttaa. Scrum-tiimi, tuotteen omistaja sekä asiakas ovat koko ajan selvillä siitä, mitä tehtäviä sprintissä toteutetaan ja mikä on sprintin tarkoitettu lopputulos. Lopputuotteen kaikkia ominaisuuksia ei tarvitse tietää heti projektin alussa, vaan asiakas voi lisätä ominaisuuksia tuotteelle kehityksen edetessä. Käytettävä teknologia vaikuttaa myös tehtävälistan sisältöön. Vesiputousmalliin verrattuna tämä Scrum-metodin ominaisuus on ylivertainen. Vesiputousmallissa kaikki tuotteen ominaisuudet täytyy kerätä ja spesifioida jo projektin alussa. Vesiputousmalli on joustamaton nykyisissä tilanteissa, joissa tuotteen ominaisuuksia täytyy muuttaa tai uudelleenfokusoida jatkuvasti. Kuvassa 1 esitetään Scrum-metodin kehitysmalli.



Kuva 1. Scrum-metodin kehityskaari [Koskela, 2008].

3. Scrum käytännössä

Tässä luvussa tarkastellaan Scrum-metodin käytäntöjä ja siihen kuuluvia tapaamisia ja organisaatioon kuuluvia rooleja.

3.1. Scrum-mestari

Scrum tuo organisaatioon uuden roolin, Scrum-mestarin, joka toimii ryhmässä eräänlaisena projektipäällikkönä. Scrum-mestarin keskeisimpiin työtehtäviin kuuluu kaikkien Scrumin käytäntöjen toteuttaminen ja vahtiminen niin, että ne myös toteutuvat käytännössä. Scrum-mestari toimii tiedonvälittäjänä Scrum-tiimin ja toteuttajayhtiön johdon välillä.

Scrum-mestarin päivittäisiin tehtäviin kuuluu kuunnella tarkasti Scrum-tiimiin kuuluvien kehittäjien ja testaajien edistystä meneillään olevissa työtehtävissä. Hänen tulee poistaa kaikki esteet, jotka haittaavat tehtävien valmistumisen ajallaan. Jos esimerkiksi kehittäjä tarvitsee uuden prototyyppilaitteen funktion toiminnallisuuden testausta varten, tulee Scrum-mestarin hankkia hänelle prototyyppilaite niin pian kuin mahdollista. Scrum-mestarin tulee myös edistää tiimin vuorovaikutusta ja sitouttaa heitä sekä etsiä ratkaisuja kehitysongelmiin hankkimalla teknistä tukea muilta tiimeiltä, jos tiimin omat taidot eivät riitä ratkaisemaan vaikeita teknisiä ongelmia [Schwaber and Beedle, 2001]. Koskela [2008] näkee seuraavat viisi tehtävää Scrum-mestarin tärkeimpinä työtehtävinä:

- Esteiden poistaminen IT:n ja asiakkaan välillä siten, että asiakas ohjaa kehitystä suoraan.
- Asiakkaan opettaminen Scrumin käytössä tuottojen maksimointiin ja tavoitteiden saavuttamiseen.
- Kehitystiimin elinolojen parantaminen tukemalla ja kannustamalla luovuutta ja itseohjautuvuutta.
- Kehitystiimin tuottavuuden kasvattaminen.
- Menetelmien ja käytäntöjen parantaminen siten, että jokaisen sprintin lopputulos on teknisesti tuotantokelpoinen.

Scrum-mestari, yhtiön johdon edustaja, eli tuotteen omistaja (product owner), sekä Scrum-tiimi muodostavat Scrum-organisaation. Tuotteen omistaja, asiakas, Scrum-mestari sekä tiimi yhdessä sopivat seuraavan sprintin tehtävälistan. Katselmoinneissa sekä muissa palavereissa Scrum-mestari johtaa puhetta, mikä selkeyttää katselmoinnin edistymistä.

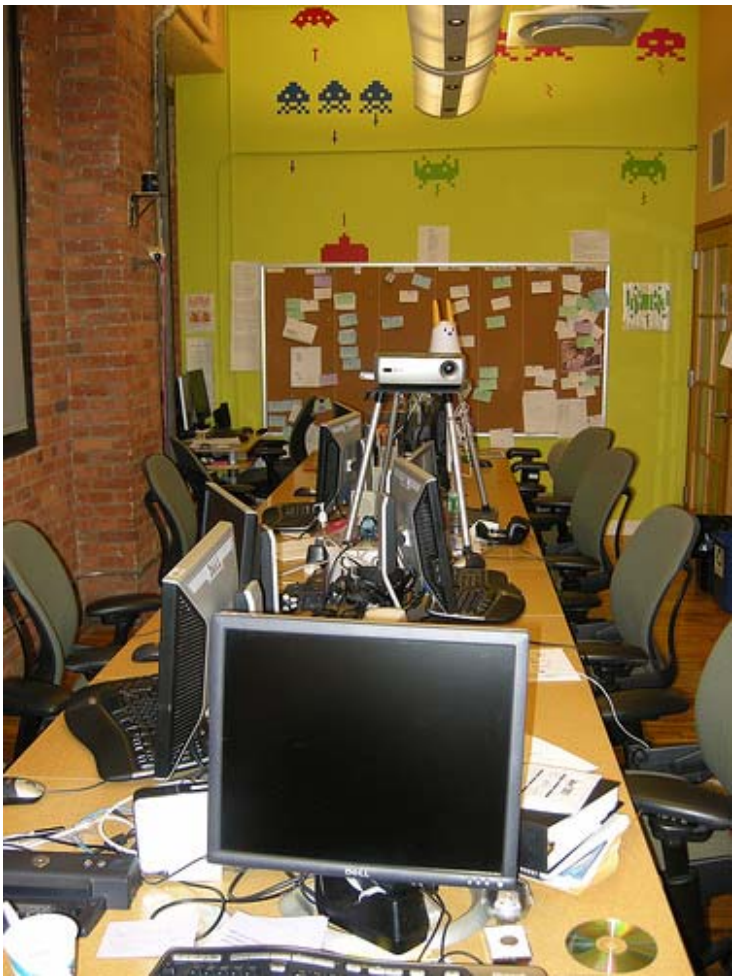
3.2. Scrum-tiimi

Scrum-metodissa kehityksestä ja laadunvalvonnasta vastaa Scrum-tiimi. Alkuperäisen ajatuksen mukaan Scrum-tiimi pyrkii valvomaan laatua itsenäisesti ilman erillistä testausorganisaatiota. Scrumissa laadunvalvonnan tärkeitä työkaluja ovat moduulitestaus (unit testing) sekä jatkuva integrointi (continuous integration, CI). Kehittäjät pyrkivät varmistamaan tuotteen laadun ajamalla moduulitestejä päivittäin löytääkseen mahdolliset ohjelmistovirheet. Jatkuva integrointi voidaan toteuttaa automaattisesti niin, että yrityksen palvelin kääntää ja ajaa yhteiset koodit päivittäin. Näillä työkaluilla ohjelmistovirheiden määrää voidaan vähentää oleellisesti integrointivaiheessa. Pelkkä kehittäjien laadunvalvonta ei kuitenkaan riitä suurissa ja monimutkaisissa ohjelmistoprojekteissa. Testausorganisaatio on siten myös laadun varmistamisen kannalta hyvä olla prosessissa mukana ja jopa testausinsinöörin saaminen itse Scrum-tiimiin on hyvä ratkaisu [Pöri, 2008]. Tiimi sopii Scrum-mestarin kanssa tehtävälistan, joka pyritään toteuttamaan sprintin aikana. Tiimi sitoutuu toteuttamaan tehtävälistan tehtävät; toteutustavan sekä työkalut tiimi voi päättää itse. Scrum-tiimin pitää olla itseorganisoituva, jolloin jokainen kehittäjä voi valita tehtävälistasta itseään kiinnostavan tehtävän tai tehtävän, joka vaatii kehittäjän erikoisosaamista. Itseorganisoituva tiimi on usein myös motivoituneempi kuin projektitiimi, jossa tehtävät määrätään kehittäjille [Schwaber and Beedle, 2001].

Tiimi ei saa olla liian suuri, koska silloin kommunikaatio etääntyy ja vähenee tiimin jäsenten välillä. Organisaatiosta tulee sen koon kasvaessa kömpelömpi eikä se pysty vastaamaan muutoksiin nopeasti, mikä on tärkeää Scrumin

kehitysmallissa. Kommunikaatiota on tärkeätä pitää yllä kehittäjien välillä koko ajan, koska näin tietämys teknisistä yksityiskohdista ja taidoista leviää Scrum-tiimin jäsenten välillä. Hyvä kommunikointi helpottaa esimerkiksi uusien kehittäjien integroimista osaksi tiimiä ja oppiminenkin nopeutuu. Schwaber ja Beedle [2001] ovat havainneet yleisesti Scrum-tiimien hyväksi kooksi seitsemän plus miinus kaksi työntekijää. Scrum-mestarin tulee myös tarjota parhaat työkalut tiimin käyttöön, sillä se lisää tiimin tuottavuutta huomattavasti, kun kaikenlainen ylimääräinen ympäristöjen tai työkalujen säätäminen jää pois kehittäjien vastuulta.

Scrum-metodissa on myös tärkeää, että työtilat tukevat ajatusten vaihtoa sekä ongelmien ratkaisua. Yleensä Scrum-tiimi työskentelee sitä varten suunnitellussa tilassa, jossa tiimin jäsenet ovat lähekkäin, eikä huoneessa ole seiniä estämässä kommunikointia. Seinillä voi olla tauluja, joissa jokainen tehtävä on merkitty muistilapuille ja sen kehitysvaihe on merkitty muistilappuun (ks. Kuva 2). Tämä käytäntö lisää läpinäkyvyyttä tehtävien edistymisestä. Scrum-huoneessa pitäisi myös olla laitteet suunnittelupalaverien pitoon, ettei tiimin tarvitsisi käyttää aikaa kuljeskelemalla ympäri toimistorakennusta tai -aluetta.



Kuva 2. Kuva Scrum-huoneesta [Judy, 2008].

3.3. Scrum-tapaamiset

Scrum-metodi sisältää prosessissa määrättyjä tapaamisia, joita jotkut kutsuvat leikkimielisesti jopa rituaaleiksi. Onnistuneen sprintin suorittamiseksi tapaamiset tulee aina suorittaa niin kuin on sovittu ja määrätty. Tapaamiset parantavat työn edistymisen seurantaa sekä ongelmien nopeaa ratkaisua. Prosessin läpinäkyvyys asiakkaalle sekä yrityksen johdolle myös parantuu, kun tapaamisia pidetään säännöllisesti. Scrum-metodiin kuuluvat seuraavat tapaamiset: sprintin suunnittelu (sprint planning), päivittäiset Scrum-tapaamiset, sprintin katselmointi (review) sekä retrospektiivi (retrospective). Seuraavissa alakohdissa tutustutaan tarkemmin jokaiseen näistä.

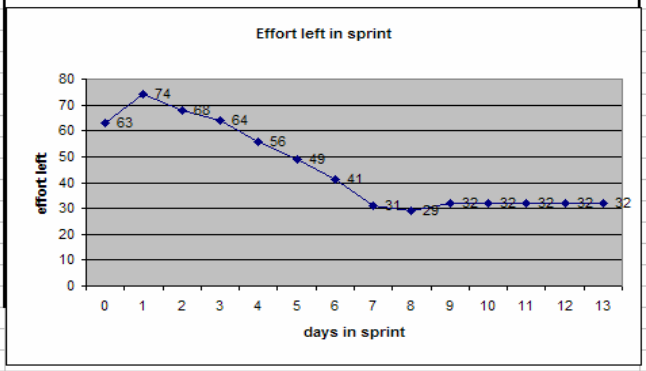
3.3.1. Sprintin suunnittelu

Jokainen sprintti alkaa aina tapaamisella, jossa sprintti ja sen sisältö suunnitellaan. Paikalla täytyy olla Scrum-tiimi, Scrum-mestari, asiakkaat sekä tuotteen omistaja. Sprintin suunnittelussa käytetään pohjana tuotteen tehtävälistaa (product backlog).

Tehtävälista sisältää siihen mennessä identifioidut tehtävät, joista sitten valitaan sprinttien sisältämät tehtävät. Sprinttiin valitaan niin paljon priorisoituja tehtäviä kuin kehittäjät arvioivat pystyvänsä toteuttamaan sprintin aikana. Asiakkaan ja tiimin toiveet täytyy ottaa huomioon, jos joitain tehtäviä joudutaan karsimaan kiireellisempien tehtävien tai ajan puutteen vuoksi. Tehtäväkuvausten on oltava tarpeeksi yksiselitteisiä, jotta niihin tarvittava työmäärä voidaan arvioida. Tarkkojen tehtävien kirjoittaminen voi olla vaikeaa kokemattomalle asiakkaalle, ja monesti niitä täytyy purkaa ja tarkentaa projektitiimissä asiantuntijavoimin, ts. uudelleenarvioida. Jokaiseen tehtävään käytettävä aika arvioidaan käyttämällä Scrum-tiimissä sovittuja mittareita (työpäivä, työviikko). Tuntimäärien onnistunut arvioiminen riippuu tiimistä ja heidän asiantuntemuksestaan alalla. Onnistunut tuntimäärien arviointi helpottaa sprinttiin sisältyvien tehtävien mukaan ottamista tai hylkäämistä.

Lopuksi osapuolet sopivat ja hyväksyvät seuraavan sprintin sisällön sekä sprintin keston. Sprintille on myös yleistä asettaa jokin sanallinen tavoite, joka pitää saavuttaa, jotta sprintti voidaan hyväksyä katselmointitapaamisessa. Esimerkiksi yhden sprintin tavoitteena voi olla, että palvelimen etäyhteys toimii langattoman verkon yli. Sprintin ideaalikesto Schwaberin ja Beedlen [2001] mukaan on 30 päivää eli noin neljä viikkoa. Sprintti voi kestää lyhyemmänkin ajan riippuen tiimistä tai sprinttiin valituista tehtävistä. Kuvassa 3 on esitelty kuvitteellisen tuotteen tehtävälista.

Story ID	Story/task	days in sprint / effort left													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
		63	74	68	64	56	49	41	31	29	32	32	32	32	32
10	**Fetch one day temperature data from the weather provider system**														
	Make our server connect and authenticate to the provider system	4	16	12	8	3	3	3	3	3	3	3	3	3	3
	Read provider's data directory	8	7	7	7	4	0	0	0	0	0	0	0	0	0
	Parse the current temperature out of the data	6	6	4	4	4	1	1	1	1	1	1	1	1	1
	Push the temperature data to the client	16	16	16	16	16	16	8	2	0	0	0	0	0	0
11	**Fetch rain, snow, etc details from the provider**														
	Parse snow/rain data from the provider's data	4	4	4	4	4	4	4	0	0	0	0	0	0	0
	Push the snow/rain data to the client	4	4	4	4	4	4	4	4	4	0	0	0	0	0
	Redesign client screen a bit										3	3	3	3	3
	Refactor the server code										4	4	4	4	4
12	**Fetch several days data from the provider**														
	Parse the weather data in day packs	10	10	10	10	10	10	10	10	10	10	10	10	10	10
	Push several days data to the client	3	3	3	3	3	3	3	3	3	3	3	3	3	3
13	**Auto-refresh feature**														
	Make the client ping server once per 4 hours	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	Make the server update the client	2	2	2	2	2	2	2	2	2	2	2	2	2	2



Backlog state taken after day 9

Kuva 3. Esimerkki tehtävälistasta [Laurila, 2008].

3.3.2. Sprintti

Sprintti on ennalta määrätty kiinteä aika, joka käytetään tehtävälistalta sprinttiin valittujen tehtävien tekemiseen valmiiksi. Tiimi on suunnitellut sprintin sisällön ja ajan, jossa se toteutetaan. Sprintin aikana tärkeimpiä tapaamisia ovat päivittäiset Scrum-tapaamiset. Ne kestävät noin 5 - 15 minuuttia, jonka aikana käydään läpi kolme yksinkertaista kysymystä: mitä kukin on tehnyt edellisen tapaamisen jälkeen, onko ollut ongelmia ja mitä aikoo tehdä seuraavaksi. Tässä tapaamisessa tulee tuoda esille kaikki tehtävien valmistumisen estävät ongelmat Scrum-mestarille, jotta hän voi yrittää poistaa ilmitulleet ongelmat. Sprinttien kesto voi vaihdella 1 – 4 viikon välillä tiimistä tai sprintin luonteesta riippuen. Yksittäinen sprintti voi olla myös pidempi, mutta se ei ole Scrum-mallin mukaan suotavaa. Voi olla, että ensimmäinen sprintti sisältää vasta esitutkimusta järjestelmän ominaisuuksista ja kehitysympäristön ylösajoa. Tällöin yhden viikon mittainen sprintti voi olla hyvä lähtökohta. Neljän viikon sprintti on nähty optimaaliseksi, jolloin kehitys tapahtuu sopivan mittaisissa iteraatioissa. Tarpeeksi pitkässä sprintissä on myös aikaa suunnitella rauhassa arkkitehtuurisia ongelmia, jotka vaikuttavat pitkälle tuotteen suunnittelussa. Scrum-tiimissä tätä suunnittelua voidaan tehdä helposti tiimin sisällä ja etsiä parhaat ratkaisut

arkkitehtuurisiin ongelmiin. Sprinteissä toimiminen vaatii tietysti tiimiltä harjoittelua, mutta jo kahden tai kolmen sprintin jälkeen työnteon tehokkuus alkaa nousta huippuunsa. Jokaisen sprintin lopputuloksena on tuotteesta syntynyt uusi versio, eli iteraatio, joka voidaan haluttaessa viedä tuotantoon tai esimerkiksi asiakkaalle koekäyttöön.

3.3.3. Sprintin katselmointi

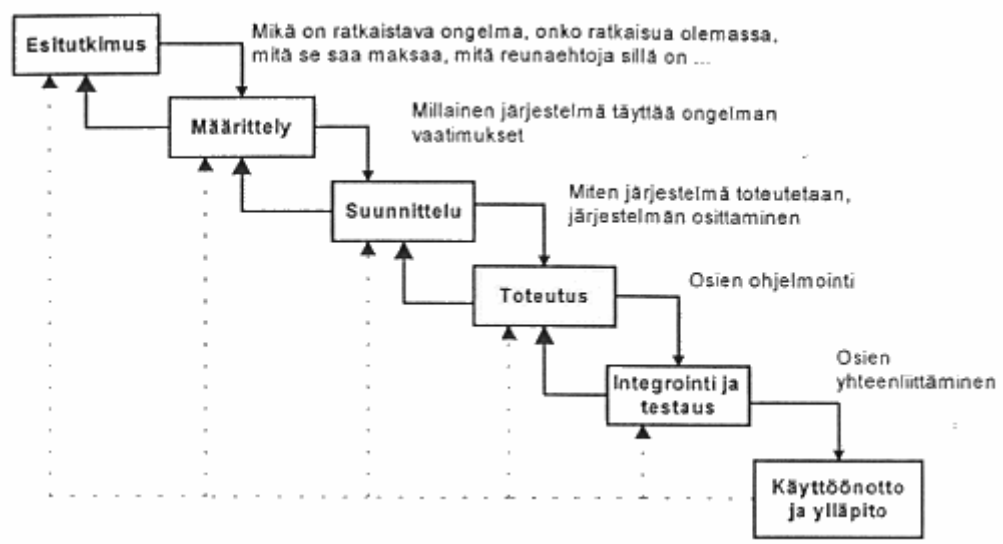
Sprintin katselmointi tapahtuu sprintin lopussa, ja siihen osallistuu Scrum-mestari, tiimi, tuotteen omistaja sekä asiakkaat. Katselmoinnissa käydään läpi ne tehtävät, jotka sprintin tehtävälistaan oli valittu suunniteltaessa sprinttiä. Puhetta johtaa yleensä Scrum-mestari. Scrum-mestari aloittaa tapaamisen kertomalla yleisesti, kuinka sprintti on sujunut ja onko kohdattu ongelmia sen aikana. Yleisen esittelyn jälkeen tehtävät esitellään yksi kerrallaan ja arvioidaan, kuinka niiden toteuttaminen on onnistunut. Scrumissa demojen esittäminen on tärkeää, koska näin asiakas ja muut osapuolet näkevät työn tulokset konkreettisemmin ja voivat tutustua tarkemmin toiminnallisuuteen. Asiakas voi halutessaan kommentoida toiminnallisuutta ja pyytää lisäämään tai poistamaan jotain siitä seuraavaan sprinttiin. Tehtävät, joita ei mahdollisesti ole saatu valmiiksi, tulee käydä tarkasti läpi asiakkaan sekä tiimin kanssa ja pyrkiä selvittämään ongelmat, jotka johtivat tehtävän epäonnistumiseen. Tämä on tärkeää siksi, että asiakas tietää, onko jokin tietty tehtävä mahdoton toteuttaa vai arvioitiinko esimerkiksi tehtävään kuluva aika väärin. Keskeneräinen tehtävä voidaan siirtää seuraavaan sprinttiin, jossa se pyritään tekemään loppuun. Lopuksi on tavallista päättää katselmointi siihen, onko sprintti onnistunut tavoitteissaan vai ei. Jos on epäonnistuttu, on hyvä keskustella syistä, jotka ovat johtaneet tähän tilanteeseen.

3.3.4. Retrospektiivi

Katselmoinnin jälkeen on tapana pitää lyhyt retrospektiivi, tapaaminen, jossa käydään läpi sprintissä onnistuneita sekä epäonnistuneita asioita [Derby and Larsen, 2006]. Retrospektiiviin osallistuu Scrum-mestari sekä tiimi. Tiimi pyrkii reflektoimaan kokemuksia menneestä sprintistä tuomalla esiin ongelmakohtia, jotta niihin voidaan puuttua seuraavassa sprintissä. Lisäksi voidaan antaa myös positiivista palautetta siitä, mikä onnistui ja niistä käytännöistä, jotka koettiin hyviksi. Scrum-mestarin tehtävänä on varmistaa, että huonoksi havaittuja toimintatapoja muutetaan ennen seuraavaa sprinttiä. Käytäntöjen parantaminen sitouttaa työntekijöitä parantamaan prosessia omatoimisesti, mikä parantaa myös heidän motivaatiotaan tehdä parhaansa.

4. Scrumin vahvuudet verrattuna muihin metodeihin

Haikala ja Märijärvi [2001] esittelevät ohjelmistoprojektien kehitysmalleiksi tuttuja metodeja, kuten vesiputousmalli (kts. kuva 4.), EVO-malli sekä erilaiset protoilumallit. Vesiputousmallia on käytetty jo 1960-luvulta lähtien, ja sitä soveltamalla on kehitetty uudempia malleja, kuten EVO-malli. Metodien kehitys tapahtuukin ajan kuluessa, jopa vuosikymmenten aikana, ja aina seuraavaan malliin pyritään ottamaan hyvät asiat vanhoista malleista ja lisäämään jotain uutta. Uudet oivallukset ohjelmistokehityksessä voivat tulla esimerkiksi uusista työkaluista tai tekniikoista.



Kuva 4. Esimerkki vesiputousmallista [Haikala ja Märijärvi, 2001].

Scrum-metodin viitekehys on johdettu aikaisemmista metodeista. Metodiin on yritetty etsiä niiden parhaat puolet ja käytännöt. Jotkut sanovatkin, että Scrum on eräänlainen ketterä ohjelmistokehityksen filosofia.

Vesiputousmallin perimmäisenä ongelmana on vaatimusten lukkoon lyöminen asiakkaan kanssa heti projektin esitutkimus- ja määrittelyvaiheessa, kuten kuvasta 4. voidaan todeta. Tämä johtaa siihen, että projekti on kykenemätön reagoimaan nopeasti vaatimusten muuttumiseen, koska esimerkiksi isot projektit voivat kestää vuosia toteuttaa, jonka aikana myös ympäristö ja vaatimukset muuttuvat. Scrumissa nämä vastaavat kehitysvaiheet käydään läpi aina yhden sprintin aikana, joten vaatimusten muutoksiin voidaan reagoida nopeasti. Vesiputousmallissa ajaudutaan siihen, että vaatimuksia joudutaan muuttamaan projektin loppuvaiheessa, mikä johtaa usein budjetin ylittämiseen ja mahdollisesti projektin epäonnistumiseen. Scrum-metodissa riskienhallinta on tärkeässä roolissa. Riskien hallintaan kuuluu olennaisesti aina ajan tasalla oleva tehtävälista sekä aktiivinen vuorovaikutus eri osapuolien välillä. Näin ollen asiakas saa

perusteellisemman kuvan projektin etenemisestä. Aikaisemmissa metodeissa on koettu tärkeäksi panostaa työaikaa hyvään ja yksityiskohtaiseen tuotteen tai järjestelmän dokumentointiin. Scrumissa on katsottu parhaaksi, että dokumentointiin ei panosteta työaikaa niin paljon, vaan ylijäävä työaika käytetään mieluummin tehtävien tekemiseen. Tämä ei kuitenkaan tarkoita sitä, että Scrum-metodissa dokumentointia ei tarvitse tehdä, se tehdään vain vähän kevyemmin. Järjestelmän dokumentointi voidaan hoitaa esimerkiksi dokumentoimalla funktiot lähdetiedostoissa ja tekemällä hyviä esimerkkiohjelmia funktioista, joista muut kehittäjät näkevät helposti, miten järjestelmä toimii.

Scrum-tiimi toimii itsenäisesti ja itseohjautuvasti, jolloin se tuo esille ihmisten parhaan osaamisen ja tiimi on motivoituneempi. Tehtävät myös vaihtuvat sprinteittäin, joten kehittäjien mielenkiinto säilyy korkealla tasolla läpi järjestelmän kehitysajan, koska kehittäjän ei tarvitse tehdä samanlaista työtä päivästä toiseen. Scrum-metodin hyviksi puoliksi voisi myös lukea sen, että tuoteiteraatiot ovat selkeitä kokonaisuuksia ja ne pystytään verifioimaan helposti. Tuotteen prototyyppien tai demojen esittely on tärkeää, jotta tuotteen toiminnallisuus voidaan todentaa myös asiakkaalle. Iteratiivinen kehitys sprinteissä antaakin hyvät seuranta- ja valvontatyökalut Scrum-mestarille, asiakkaalle sekä yrityksen johdolle. Tästä seuraa, että Scrum-prosessi on selkeä ja läpinäkyvä kaikille osapuolille, mikä vähentää projektiin liittyviä riskejä merkittävästi.

5. Yhteenveto

Scrum-metodista on tullut merkittävä kehitysmalli yrityksissä sen joustavuuden ja tehokkuuden takia. Metodi antaa tiimille vapaat kädet toteuttaa tehtäviä olemalla samalla motivoiva ja itseohjautuva. Scrum-metodi luotiin korvaamaan aiemmat kehitysmetodit, jotka oli todettu joustamattomiksi käytännön ohjelmistoprojekteissa. Ketterät kehitysmallit ovat korvaamassa aiemmat metodit kehitystyössä. Scrum-metodi ei kuitenkaan ole täydellinen kehitysmalli, joka automaattisesti takaa projektien onnistumisen. Scrum vaatii sitoutumista prosessin eri vaiheisiin. Lisää tietoa tarvitaan siitä, kuinka Scrum-metodi skaalautuu suuriin ohjelmistoprojekteihin, jotka toimivat eri maissa. Scrum toisaalta sopii monenlaisiin projekteihin eri aloilla. Se on havaittu toimivaksi vaativissakin ohjelmistoprojekteissa, joissa on esimerkiksi paljon riskitekijöitä teknologian tai vaatimusten suhteen. Yksinkertaisempien projektien voi olla parempi käyttää perinteisiä kehitysmalleja, kuten vesiputous- tai iteraatiomalleja.

Scrum-metodin implementointi yrityksen käyttöön vaatii aina muutaman sprintin läpikäymisen, että päästään parhaaseen työtehoon ja Scrumin tehokkuus löytyy. Scrum ei ole mikään pelastava metodi, jos perusasiat organisaatiossa eivät ole kunnossa, kuten luottamus tiimin jäsenten välillä tai yrityksen

johdossa on ongelmia. Toisekseen Scrum ei välttämättä sovi kaikille organisaatioille tai ihmisille. Toiset kehittäjät haluavat tehdä työnsä itsenäisesti ja pyrkivät vastustamaan avointa työympäristöä. Scrum-metodin tarvitsee avointa interaktiota kehittäjien välillä päivittäin, jotta projekti voisi olla menestyksekäs. Yhtenä ongelmana Scrumissa on se, kuinka Scrum-tiimi saadaan noudattamaan metodin sääntöjä ja osallistumaan päivittäisiin tapaamisiin sekä suunnitteluun että katselmointeihin. Tiimin hallinta vaatii hyviä ihmissuhdetaitoja ja sovittelevaa ilmapiiriä Scrum-mestarilta sekä tiimin jäseniltä. Scrum-metodista ei ole vielä käytännössä tarpeeksi kokemusta, jotta voitaisiin todistaa sen parantaneen merkittävästi työnlaatua tai ohjelmistokehityksen tehokkuutta. Tällä hetkellä näyttää kuitenkin siltä, että Scrum tuo lisäarvoa yritykselle, joka pystyy sen implementoimaan yrityksen omiin prosesseihinsa.

Viiteluettelo

[Agile, 2008] http://fi.wikipedia.org/wiki/Agile, Tarkistettu 9.11.2008.

[Derby and Larsen, 2006] Esther Derby and Diana Larsen, *Agile Retrospectives Making Good Teams Great*. The Pragmatic Programmers, 2006.

[Haikala ja Märijärvi, 2001] Ilkka Haikala ja Jukka Märijärvi, *Ohjelmistotuotanto*. Talentum, 2001.

[Koskela, 2008] Lasse Koskela, Scrum: Ketterien menetelmien markkinajohtaja. Reaktor Innovations Oy. Saatavilla https://ttlry-fi.directo.fi/haku/?E*Q=scrum, Tarkistettu 9.11.2008.

[Manifesto, 2008] http://www.agilemanifesto.org/, Checked 9.11.2008.

[Pöri, 2008] Mikko Pöri, Testaus Scrum-prosessimallissa. Pro gradu -tutkielma, Helsingin yliopisto, 2008. Saatavilla http://www.cs.helsinki.fi/u/mpori/gradu/gradu_pori.pdf. Tarkistettu 9.11.2008.

[Royce, 1998] Walker Royce, *Software Project Management*. Addison-Wesley, 1998.

[Schwaber and Beedle, 2001] Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum*. Prentice-Hall, 2001.

[Scrum Alliance, 2008] http://www.scrumalliance.org/, Checked 9.11.2008.

[Judy, 2008] Ken H. Judy, Picture of a Scrum team room. http://judykat.com/ken/2007/11/24/our-team-room/, Checked 27.11.2008.

[Laurila, 2008] Jukka Laurila, Example product backlog, http://agilesoftwaredevelopment.com/tags/backlog, Checked 27.11.2008.

Reunatäsmäyspeleistä

Tuomas Kujala

Tiivistelmä.

Käsittelen tässä tutkielmassa reunatäsmäyspelejä, jotka ovat mielenkiintoisia ja laskennallisesti todella vaativia lautapelejä. Kerron aluksi hieman reunatäsmäyspelien rakenteesta ja aikavaativuudesta, jonka jälkeen esittelen kehittelemäni ratkaisualgoritmin reunatäsmäyspeleille. Testaan ratkaisualgoritmiani käytännössä ja vertaan saamiani tuloksia eräiden tutkijoiden saamiin tuloksiin.

Avainsanat ja -sanonnat: Reunatäsmäyspelit, NP-täydellisyys

CR-luokat: F.1.3, F.2.2, F.4.3, I.2.8

1. Johdanto

Ihmisellä vaikuttaa olevan luontainen halu ja kiinnostus kehittää ja ratkaista erilaisia ongelmia vapaaehtoisesti. Mahdollisesti tämän seikan takia syntyivät aikoinaan mm. erilaiset ongelmankaltaiset "pähkinät" (engl. *puzzle*) sekä lautapelit.

Lautapelien historia ulottuu hyvin pitkälle menneisyyteen. Peli nimeltä *Senet* on maailman vanhin dokumentoitu lautapeli, jonka keksiminen ajoittuu vuosille 3500–3100 eaa. [Piccione, 1980] Jo siis yli 5000 vuotta sitten ihmiset pelasivat nykyisenkaltaisten lautapelien esi-isiä. Senet ei liene kovinkaan tunnettu peli maailmalla, mutta esimerkiksi shakki ja go (kiinaksi *wéiqí*) ovat tunnettuja historiallisia lautapelejä, joita pelataan hyvin paljon nykyisinkin. Nämä kaksi peliä ovat nykyään mahdollisesti suositumpia kuin koskaan aikaisemmin, sillä maailman globalisoituminen on vienyt monia ennen vain paikallisesti pelattuja pelejä muuallekin maailmaan.

Perinteiset palapelit vastaavat rakenteeltaan hyvin paljon tutkielman aiheena olevia reunatäsmäyspelejä. Palapelit voidaan oikeastaan lukea eräänlaisiksi reunatäsmäyspeleiksi. Palapelien uskotaan syntyneen vuoden 1760 tienoilla, kun Lontoossa asuva kartanpiirtäjänä ja kaivertajana toiminut John Spilsbury kiinnitti erään karttansa puiselle alustalle ja leikkasi ohutteräisellä sahalla valtiot irti kartasta niiden rajoja pitkin. Erilaiset palapelit toimivat aluksi opetustyökaluina, mutta nykyisin palapelit ovat lähinnä viihdekäyttöön tarkoitettuja. [McAdam, 2004]

Kerron luvussa 2 yleisesti reunatäsmäyspeleistä ja niiden rakenteesta. Tämän lisäksi käyn läpi hieman reunatäsmäyspeleihin liittyvää kombinatoriikkaa.

Luvussa 3 käsittelen päätösongelmia ja aikavaativuusluokkia P, NP, NP-

kova ja NP-täydellinen. Nämä seikat liittyvät läheisesti reunatäsmäyspeleihin.

Luvussa 4 kerron reunatäsmäyspelien ratkaisemisemisesta ja siihen liittyvistä asioista. Tämän jälkeen esittelen kehittelemäni ratkaisualgoritmin ja testaan sen toimintaa. Vertaan saatuja tuloksia eräiden reunatäsmäyspelejä tutkineiden tutkijoiden tuloksiin.

Luvussa 5 tarkistelen kehittämäni algoritmin tuloksia ja pohdin tuloksiin mahdollisesti vaikuttaneita seikkoja.

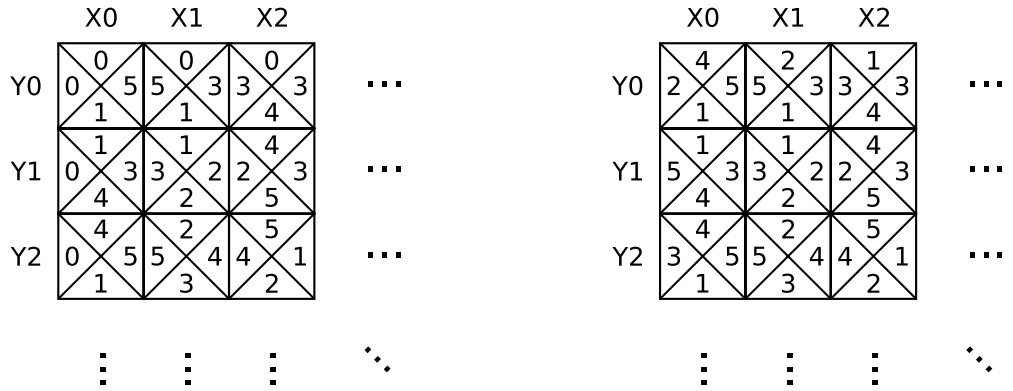
2. Yleistä reunatäsmäyspeleistä

2.1. Rakenne

Termiä "reunatäsmäyspeli" ei ole määritelty eksaktisti, joten kyseisen kategorian rajat ovat melko häilyvät ja reunatäsmäyspeleiksi voidaan lukea monia erilaisia pelejä. Yleisesti reunatäsmäyspelien tarkoituksena on latoa määrätyn muotoisia paloja pelikentälle tiettyjen sääntöjen mukaan. Tavoitteena voi olla esimerkiksi erimuotoisista monikulmioista suurempien yhtenäisten kuvioiden muodostaminen, tai pelaajan tulee latoa samanmuotoisia symmetrisiä monikulmioita pelikentälle siten, että vierekkäisten palojen vastinsivut ovat yhtenevät. Vaikka erilaisia reunatäsmäyspelejä onkin olemassa suuri määrä, keskitytään tässä tutkielmassa ainoastaan erääseen säännöiltään yksinkertaiseen pelimuotoon.

Tästä eteenpäin reunatäsmäyspeleillä tarkoitetaan pelejä, jotka sisältävät n kappaletta neliön muotoisia paloja, joiden jokaisella neljällä sivulla on jokin tunniste, kuten vaikkapa kuvio tai numero. Tässä tutkielmassa palojen sivujen tunnisteina käytetään numeroita.

Tutkielmassa käsiteltävän reunatäsmäyspelityypin tarkoituksena on latoa kaikki palat suorakaiteen muotoiselle pelilaudalle, jossa on x saraketta ja y riviä siten, että $x \times y = n$. Pelilaudalle asetettujen vierekkäisten palojen vastinsivujen tulee olla yhtenevät (sivu- ja pystysuunnassa). Jokainen pala voidaan kiertää oletusasennon lisäksi kolmeen muuhun eri asentoon kääntämällä palaa oletusasennosta 90°, 180° tai 270° myötäpäivään (tai vastapäivään). Joissain reunatäsmäyspeleissä on tavallisten palojen lisäksi myös erityisiä reunapaloja, joiden pelilaudan ulkoreunaan vastaavilla sivuilla tulee olla erityinen tunniste. Tässä tutkielmassa edellämainittuna tunnisteena käytetään numeroa nolla (0). Kuvassa 1 esitetään kaksi reunatäsmäyspelin esimerkkiä, joista toisessa on käytössä erilliset reunapalat (palat, joissa esiintyy numero 0) ja toisessa ei.



Kuva 1. Reunatäsmäyspelin esimerkki reunapaloilla (vas.) ja ilman reunapaloja.

Tietokonepeli nimeltä *Tetravex* on reunatäsmäyspeli, jossa ei käytetä erillisiä reunapaloja. Mikä tahansa pelattavista paloista voidaan siis sijoittaa pelilaudan reunaan tai nurkkaan. Tetravexissa palojen kiertymä on kiinnitetty, eli palasia ei voi siis käännellä lainkaan. Linux-käyttöjärjestelmän GNOME-työpöytä-ympäristön mukana tulee avoimen lähdekoodin versio Tetravexistä. Tässä versiossa pelaaja voi valita pelikentän kooksi 2 x 2 – 6 x 6, jolloin käytettävissä olevia paloja on siis vähimmillään $n = 4$ ja enimmillään $n = 36$. [Tetravex, 1991]

Kuva 2. Tetravexin eräs pelitilanne

Eräs esimerkki reunapaloja käyttävästä reunatäsmäyspelistä on *Eternity II*. Eternity II on julkaistu heinäkuussa 2007 ainoastaan lautapelinä, ja sen ensimmäisen täydellisen ratkaisun keksijälle on luvassa kahden miljoonan dollarin palkinto. Eternity II koostuu 16 x 16 palan kokoisesta pelilaudasta ja

näin ollen $n = 256$ erilaisesta palasta. [Eternity II, 2007] Äkkiseltään Eternity II:n ratkaiseminen kuulostaa naurettavan helpolta tehtävältä (varsinkin kun kannustimena on kaksi miljoonaa dollaria), mutta pian näemme, etteivät reunatäsmäyspelit ole niin yksinkertaisia ratkaista kuin voisi kuvitella.

2.2. Erilaisten kombinaatioiden määrä

Tässä tutkielmassa käsiteltävissä reunatäsmäyspeleissä on siis suorakaiteen muotoisella pelialustalla n kappaletta neliön muotoisia paloja. Kombinatoriikan perusteiden mukaisesti n erilaista alkiota voidaan asettaa $n!$ erilaiseen järjestykseen, joten myös n eri reunatäsmäyspelin palaa voidaan asettaa pelilaudalle yhtä monella eri tavalla. Jos jokainen pala voidaan kääntää neljään eri asentoon, voidaan n:ää eri palaa kääntämällä muodostaa 4^n erilaista kombinaatiota. Kun nämä kaksi eri tulosta yhdistetään, saadaan erilaisten siirtojen määräksi $n!$ reunatäsmäyspeleille, joiden paloja ei voi kiertää. Pelimuodoille, joissa palojen kiertäminen on madollista, erilaisia siirtoja on $n! \times 4^n$ kappaletta. Näin ollen Tetravexin (3 x 3)-kokoiselle pelikentälle (ei kääntöjä) voidaan palaset sijoittaa $9! = 362880$ erilaisella tavalla. Tämä määrä on jo liian työläs ihmiselle järjestelmällisesti kokeiltavaksi (joskin ihminen yleensä pyrkii käyttämään ratkaisemisessa hieman kehittyneempiä keinoja kuin järjestelmällistä kokeilua). Eternity II:ssa mahdollisia siirtoja on jopa $256! \times 4^{256} \approx 1{,}15 \times 10^{661}$ kappaletta! Jos Eternity II päätettäisiin ratkaista kokeilemalla järjestelmällisesti jokaista ratkaisua tietokoneella, joka pystyisi kokeilemaan sekunnissa vaikkapa jopa 10^{20} erilaista järjestystä, kestäisi tältä tietokoneelta noin $\frac{1{,}15 \times 10^{661}}{10^{20} \times 86400 \times 365{,}24} \approx 3{,}64 \times 10^{633}$ vuotta käydä läpi jokainen järjestys!

Tällaista niinsanottua väsytysmenetelmään (engl. *brute force*) perustuvaa ratkaisualgoritmia käyttävä voi siis käytännössä luopua toivosta ratkaista Eternity II ja voitaa kahden miljoonan dollarin palkinto. Reunatäsmäyspelit ovat siis todella vaativia laskennallisesti, eikä väsytysmenetelmä ole järkevä keino niiden ratkaisuun.

2.3. Suhde muihin ongelmanratkaisupeleihin

Kuten aikaisemmin jo mainittiin, reunatäsmäyspelit muistuttavat läheisesti tavallisia palapelejä. Demaine ja Demaine [2007] osoittivatkin, että jokainen reunatäsmäyspeli voidaan muuntaa ”yhtä vaikeaksi” palapeliksi. Vastaavasti jokainen palapeli voidaan muuntaa yhtä vaativaksi reunatäsmäyspeliksi.

Kuinka sitten on mahdollista, että ihminen pystyy ratkaisemaan ilman mitään apuvälineitä jopa usean tuhennen palan palapelejä? Ratkaisu tähän kysymykseen löytyy palapelin pintaan yleensä painettavasta kuvasta. Reunatäsmäyspeleissäkin palojen pinnassa on usein kuvioita, mutta tämä kuviointi ei ole palojen yli jatkuva kuvio, joka muodostaa yhtenäisen kokonaisuuden (esim.

valokuvan tai maalauksen), kun peli on valmis. Reunatäsmäyspeleissä kuviointi on palakohtainen siten, että sopiva vastinsivu tiettyyn palaan voi löytyä useammasta muusta eri palasta. Vaikka reunatäsmäyspelin ratkaisija saisi koottua usean palan kokoisen alueen, hän ei voi varmuudella tietää, kuuluvatko hänen kokoamansa palat todella yhteen jossain lopullisessa ratkaisussa. Palapelin kokoaja pystyy sen sijaan välittömästi sanomaan, onko hän liittänyt kaksi palaa oikealla tavalla toisiinsa, sillä useimmiten palapelin palojen vastinreunat ovat yksilöllisiä palapelin painatuksen takia.

Yksinkertaistetusti ajatellen tavallinen palapeli voidaan muuttaa reunatäsmäyspeliksi muuntamalla palapelin palojen erimuotoiset reunat suoriksi, jolloin paloista muodostuisi esimerkiksi neliöitä (muoto riippuu tietenkin palapelin alkuperäisten palojen muodosta). Näin muodostettu peli olisi vielä kohtalaisen helppo ratkaista, sillä palojen reunat olisivat yksilöllisiä painatuksen takia. Jos palojen painatus tämän jälkeen muutettaisiin muutamaksi erilaiseksi palojen sivuissa toistuvaksi kuvioksi, olisimme saaneet muodostettua tässä tutkielmassa käsiteltävän tyylisen reunatäsmäyspelin.

3. NP-täydellisyys

3.1. Optimointi- ja päätösongelmat

Kaikille ongelmille ei voida antaa samantyylistä vastausta. Jotkut ongelmat vaativat vastauksekseen jonkin tietyn joukon suotuisimman alkion. Toisiin ongelmiin voidaan antaa vain myönteinen tai kielteinen vastaus, ei välttämättä minkään erityisen joukon alkiota.

Seuraava ongelma on esimerkki tilanteesta, johon vastaukseksi kelpaa vain vastausjoukon suotuisin alkio: “Kuinka pitkä on lyhin mahdollinen reitti maanteitä pitkin Tampereen ja Helsingin välillä?”. Tähän kysymykseen on olemassa useita erilaisia vastausvaihtoehtoja, joista yksi ainoa[1] on ns. “paras” vastaus, eli tässä tapauksessa lyhimmän mahdollisen reitin pituus. Tämän kaltaisia ongelmia kutsutaan *optimointiongelmiksi*.

Jokaiselle optimointiongelmalle voidaan konstruoida vastaava *päätösongelma*. Päätösongelmat ovat ongelmia, joiden jokaiselle syötteelle odotetaan vastausta “Kyllä” tai “Ei” (vaihtoehtoisesti “1” tai “0”). Yllä mainitun Helsingin ja Tampereen välistä lyhintä välimatkaa koskeva optimointiongelma voidaan esittää päätösongelmana seuraavasti: “Onko reitti r lyhin mahdollinen reitti Tampereen ja Helsingin välillä?”. Kuten nähdään, tähän kysymykseen ei voida vastata muuten kuin “Kyllä” tai “Ei”.

1 Tämä kysymys ei huomioi sitä, että Tampereen ja Helsingin välillä voi olla useampi kappale lyhyimpiä reittejä, jolloin reiteistä yksi ainoa ei olisi tällöin paras. Vastauksen arvoon useamman lyhyimmän reitin olemassaolo ei kuitenkaan vaikuta, sillä näiden lyhimpien reittien pituudet ovat varmastikin samat.

Yksittäisen optimointiongelman ratkaisu antaa ongelmasta enemmän informaatiota kuin yksittäinen päätösongelma, sillä päätösongelman ratkaisu kertoo vain, toteutuuko jokin tietty ehto. Optimointiongelman ratkaisu sen sijaan antaa – kuten edellä mainittiin – parhaan ratkaisun ongelmaan. Saadulla parhaalla ratkaisulla pystytään suoraan myös ratkaisemaan vastaava päätösongelma.

3.2. Aikavaativuusluokka P

Turingin kone on Alan Turingin vuonna 1937 kehittämä teoreettinen kone, jonka avulla voidaan simuloida minkä tahansa tietokonealgoritmin logiikkaa. Turingin konetta ei ole tarkoitettu käytännön ongelmien ratkaisemiseen, vaan sen avulla voidaan tutkia laskennan rajoja. [Hopcroft et al., 2003] Kun sanotaan jonkin ongelman ratkeavan Turingin koneella polynomisessa ajassa, on se yhtäpitävää sen kanssa, että on mahdollista kehittää aikavaativuudeltaan polynominen algoritmi jollain ohjelmointikielellä saman ongelman ratkaisemiseksi.

Aikavaativuusluokkaan P (engl. *polynomial time*) kuuluvat kaikki ne päätösongelmat, jotka voidaan ratkaista Turingin koneella polynomisessa ajassa, eli niiden aikavaativuus on luokkaa $O(n^k)$, missä n on syötteen koko, ja k on jokin vakio. Polynomisessa ajassa toimivia algoritmeja pidetään yleisesti ottaen tehokkaina.[2] [Cormen et al., 2007]

Jokaiselle ongelmalle voidaan esittää jokin ratkaisuehdotus (joka voi olla myös väärä), jota voidaan kutsua myös *varmenteeksi* (engl. *certificate*). Varmenteen tarkastava algoritmi on myös päätösongelma, nimittäin: “Ratkeaako ongelma S varmenteella V?”.

Intuitiivisesti tuntuu siltä, että ongelman varmenteen tarkastaminen vie huomattavasti vähemmän aikaa kuin itse ongelman ratkaiseminen. Useasti tilanne onkin juuri tämänlainen, mutta varmenteen tarkastamiseen kuluva aika on kuitenkin ongelmakohtainen, eikä asiasta voi tehdä yleisiä johtopäätöksiä. Varmenteen tarkistamisen aikavaativuus ei voi kuitenkaan olla suurempi kuin itse ongelman ratkaisun aikavaativuus, sillä jos itse ongelma ratkeaa polynomisessa ajassa, ratkaisualgoritmi voidaan helposti muuttaa muotoon, jossa se hyväksyy ainoastaan “oikeellisia” ongelman ratkaisevia varmenteita.

3.2. Aikavaativuusluokka NP

Jos jonkin päätösongelman ehdotettu ratkaisu pystytään varmistamaan oikeaksi jollain algoritmilla polynomisessa ajassa, kuuluu tämä ongelma aikavaativuusluokkaan NP (engl. *non-deterministic polynomial time*). Edellisen kohdan perusteella huomaamme, että $P \subseteq NP$, eli luokka P sisältyy luokkaan NP. [Cormen et al., 2007]

2 Algoritmi, jonka suoritus kestää esimerkiksi n^{100} aikayksikköä kun syötteen koko on n, kuuluu määritelmän mukaan myös luokkaan P, vaikka tällaista algoritmia ei voida missään nimessä pitää tehokkaana. Käytännössä tällaisia algoritmeja on kuitenkin käytössä hyvin vähän (jos ollenkaan).

Luokan NP englanninkielisen nimen suora käännös on *epädeterministinen polynomiaikainen* (*luokka*). Tämä tarkoittaa sitä, että epädeterministisellä Turingin koneella luokan NP päätösongelmat voidaan ratkaista polynomisessa ajassa.

Normaali tietokone toimii samalla periaatteella kuin tavallinen (eli deterministinen) Turingin kone. Tämä tarkoittaa käytännössä sitä, että jonkin algoritmin saama syöte määrittelee täysin algoritmin tulosteen. Deterministisen Turingin koneen toiminta on siis ”ennustettavissa”.

Epädeterministisen Turingin koneen voi ajatella tavaksi määritellä jonkin algoritmin laskentojen joukko. Tämä joukko muodostuu kaikista laskennoista, jotka saadaan algoritmin kaikissa haarakohdissa tehtävillä valinnoilla.

Vaikka luokan NP kaikkien päätösongelmien ratkaisut pystytään varmistamaan polynomisessa ajassa, ei kaikkia luokan NP ongelmia pystytä nykyisen tietämyksen mukaan ratkaisemaan polynomisessa ajassa. Ongelman kuulumisesta luokkaan NP ei voi siis tehdä suoria johtopäätöksiä ongelman aikavaativuudesta.

Yksi tietojenkäsittelytieteen suuri ratkaisematon ongelma kuuluu: onko luokka P sama kuin luokka NP? Toisin sanoen, jos luokan NP minkä tahansa ongelman varmenne voidaan tarkistaa polynomisessa ajassa, niin voidaanko sama ongelma myös ratkaista polynomisessa ajassa? Tutkijoiden mielipiteet asiasta eroavat toisistaan, mutta Gasarch:n vuonna 2002 tekemän tutkimuksen perusteella valtaosa haastatelluista tutkijoista oli sitä mieltä, että luokka P on luokan NP aito osajoukko, eli $P \neq NP$. [Gasarch, 2002]

3.3. Aikavaativuusluokat NP-kova ja NP-täydellinen

Reunatäsmäyspelit ovat siitä mielenkiintoisia, että niille ei ole löydetty vielä ainoatakaan polynomisessa ajassa toimivaa ratkaisualgoritmia. Vaikka polynomista ratkaisualgoritmia ei ole vielä löydetty, ei se kuitenkaan tarkoita, ettei sellaista välttämättä ole olemassa. Tähän seikkaan liittyy läheisesti luokat NP-kova ja NP-täydellinen.

Ongelmasta B sanotaan, että se on NP-kova, jos mikä tahansa luokan NP ongelma A voidaan muuntaa ongelman B ilmentymäksi polynomisessa ajassa. Tämä tarkoittaa sitä, että jos ongelman A ratkaisualgoritmi palauttaa arvon a syötteellä x, palauttaa myös ongelman B ratkaisualgoritmi muunnetulle ongelmalle arvon a syöttellä x. [Cormen et al., 2007]

Ongelma C on NP-täydellinen, jos se kuuluu luokkaan NP, ja on myös NP-kova. NP-täydelliset ongelmat ovat vaikeimpia luokan NP ongelmia.

Määritelmistä seuraa, että jos jollekin NP-kovalle tai NP-täydelliselle ongelmalle löydetään polynomiaikainen ratkaisualgoritmi, voidaan jokainen luokan NP ongelma ratkaista polynomisessa ajassa. Tästä seuraisi, että $P = NP$.

Ongelman osoittaminen NP-täydelliseksi ei ole mitenkään triviaali tehtävä. Ensimmäinen, yksinkertaisempi vaihe on todistaa, että ongelma kuuluu luok-

kaan NP. Tämä todetaan osoittamalla, että ongelman yksittäisen varmenteen oikeellisuus voidaan tarkistaa polynomisessa ajassa. NP-kovuuden toteaminen onkin jo huomattavasti vaikeampi tehtävä. Erittäin harvoin yritetään suoraan todistaa, että ongelma on NP-kova. Eräs hyvin käyttökelpoinen tapa on yrittää muuntaa käsiteltävä ongelma joksikin muuksi ongelmaksi, josta jo tiedetään, että se on NP-kova. Jos muunto onnistuu, on myös käsiteltävä ongelma NP-kova. Tämän menetelmän oikeutus perustuu suoraan luokan NP-kova määritelmään.

Yleensä muunnettavaksi kohdeongelmaksi valitaan jokin tunnettu, mahdollisimman alkeellinen ongelma, johon muunnos on helppo tehdä. Eräs tunnettu tällainen ongelma on propositiologiikan toteutuvuusongelma (engl. *Boolean Satisfiability Problem*) eli SAT-ongelma tästä eteenpäin. SAT-ongelman täytyy luonnollisesti olla NP-kova, jotta muunnoksessa olisi jotain järkeä. Sen lisäksi, että SAT-ongelma on NP-kova, on se myös NP-täydellinen [Cormen et al., 2007]. Tässä ongelmassa pyritään selvittämään, evaluoituuko tietty propositiologiikan lauseke todeksi joillain muuttujien arvoilla. Ongelma esitetään usein 3-CNF–muodossa (*3-Conjunctive Normal Form*) seuraavanlaisesti:

$$(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge (x_{31} \vee x_{32} \vee x_{33}) \wedge \ldots.$$

Lauseke siis muodostuu TAI-konnektiivilla ($\vee$) toisiinsa liitetyistä kolmen literaalin (eli muuttujan tai sen negaation) ryhmistä, jotka ovat liitetty toisiinsa JA-konnektiiveilla ($\wedge$). Mikä tahansa propositiologiikan lauseke voidaan muuntaa 3-CNF–muotoon [CNF, 2008]. Muunnos onnistuu käyttämällä kaksoisnegaation lakia, De Morganin sääntöjä sekä osittelulakeja. 3-CNF-muodossa esitetystä SAT-ongelmasta käytetään nimityistä 3-SAT.

3.4. Reunatäsmäyspelit ja NP-täydellisyys

Koska reunatäsmäyspelien ratkaisemiseksi ei ole ainakaan toistaiseksi löydetty ainoatakaan polynomiaikaista algoritmia, oletamme, etteivät reunatäsmäyspelit kuulu luokkaan P. On kuitenkin helppo tarkistaa, ratkaiseeko jokin tietty palojen järjestys (eli varmenne) reunatäsmäyspelin. Reunatäsmäyspelien sertifikaatti voidaan varmentaa nopeasti, sillä pelilaudalta täytyy tarkistaa ainoastaan jokaisten vierekkäisten palojen muodostamien parien väliset reunat. Jos reunatäsmäyspeli käyttää erillisiä reunapaloja, täytyy tarkistaa myös näiden reunapalojen oikeellinen sijainti. Jos pelilaudan leveyttä merkitään x:llä ja korkeutta y:llä, palojen kokonaismäärä on tällöin xy. Nyt pelilaudan yhdellä rivillä sijaitsevista paloista voidaan muodostaa $x-1$ sivuttaissuuntaista vierekkäistä paria. Koska rivien määrä on y, sivuttaissuunnassa tarkastettavien reunojen kokonaismääräksi saadaan $y \times (x-1)$. Vastaavasti pystysuunnassa vierekkäisiä (eli päällekkäisiä) pareja on $x \times (y-1)$ kappaletta. Oletetaan vielä, että reunatäs-

mäyspeli käyttää reunapaloja, jolloin pelilaudan ulkoreunalla sijaitsevista paloista tarvitsee tarkistaa lisäksi $2 \times x + 2 \times y$ reunaa. Kun kaikki edellä muodostetut termit yhdistetään, saadaan tarvittavien tarkistusoperaatioiden määräksi

$$y \times (x-1) + x \times (y-1) + 2 \times x + 2 \times y = 2xy + x + y .$$

Koska palojen kokonaismääräksi saatiin $n = xy$, on reunatäsmäyspelin sertifikaatin tarkastuksen aikavaativuus $O(n)$. Näin ollen reunatäsmäyspelit kuuluvat luokkaan NP.

Reunatäsmäyspelien todistaminen NP-koviksi sivuutetaan tässä, mutta todetaan, että Takenaga ja Walsh [2006] ovat todistaneet Tetravexin NP-täydelliseksi käyttäen NP-kovuuden todistamiseksi 3-SAT–ongelman muunnelmaa 1-in-3-SAT. Demaine ja Demaine [2007] ovat todistaneet myös reunatäsmäyspelit NP-täydellisiksi.

4. Reunatäsmäyspelien ratkaiseminen

4.1. Yleistä

Kuten aikaisemmin kerrottiin, heinäkuussa 2007 julkaistun Eternity II -pelin ensimmäisen täydellisen ratkaisun keksijälle maksetaan palkintorahaa kaksi miljoonaa dollaria. Vielä ei tiedetä, onko kukaan ratkaissut tätä peliä, sillä ensimmäinen ratkaisujen tarkistuspäivä on 31.12.2008. [Eternity II, 2007]

Reunatäsmäyspelit ovat hyvää viihdettä ongelmapelien ystäville, mutta moni on varmasti miettinyt näiden pelien ratkaisemista myös tietokonetta apuna käyttäen. Tämä ajatus vaikuttaa varsin mielekkäältä, sillä reunatäsmäyspelien ratkaiseminen sisältää paljon yksitoikkoista laskentaa, joka on helppo suorittaa tietokoneen avulla.

Tietokone ei silti tuo niin reunatäsmäyspelien kuin monien muidenkaan ongelmien ratkaisijalle välitöntä ja helppoa ratkaisua, sillä kone itsessään on laskutehostaan riippumatta täysin turha ilman pätevää algoritmia. Toisaalta, vaikka käytössä olisi tehokas ratkaisualgoritmi, ei sillä todennäköisesti pystyisi ratkaisemaan kovinkaan suuria reunatäsmäyspelejä. Nykyisillä algoritmeilla reunatäsmäyspelien ratkaisemiseen tarvittava aika kasvaa eksponentiaalisesti reunatäsmäyspelin palojen määrän lisääntyessä, joten polynomisessa ajassa toimiva ratkaisualgoritmi mahdollistaisi hieman suurempienkin reunatäsmäyspelien ratkaisemisen.

Rakenteeltaan mahdollisesti yksinkertaisin algoritmi reunatäsmäyspelien ratkaisemiseen on jo aikaisemmin mainittu väsytysmenetelmä. Tämä algoritmi on täysin hyödytön vähääkään isommilla pelilaudoilla. Jo kuudentoista palan kokoisen pelilaudan ratkaisemiseen voi väsytysmenetelmällä kulua useampi

viikko.

Koska reunatäsmäyspelit ja SAT-ongelmat ovat NP-täydellisiä, voidaan ratkaistava reunatäsmäyspeli muuntaa SAT-ongelman ilmentymäksi, ja tämän jälkeen riittää ratkaista kyseinen SAT-ongelma. Tällainen menettely voi olla jossain mielessä perusteltu, sillä SAT-ongelma on varsin yleisesti tunnettu ja sitä on tutkittu paljon. Näin ollen erilaisia SAT-ongelmien ratkaisualgoritmeja on olemassa useampia. Erilaisia SAT-ratkaisijoita on käytetty myös reunatäsmäyspelien ratkaisemisessa [Ansótegui et al., 2008].

4.2. Käsiteltävän ratkaisualgoritmin kuvaus

Periaatteeltaan erilaisia ratkaisualgoritmeja reunatäsmäyspeleihin on varmasti useita, mutta tässä tutkimuksessa keskitytään kehittämäni ratkaisualgoritmin tutkimiseen, jonka toiminta muistuttaa läheisesti tapaa, jolla monet ihmisetkin voisivat ratkaista reunatäsmäyspelin.

Käsiteltävä ratkaisualgoritmi on suunniteltu ainoastaan reunapalallisten reunatäsmäyspelien ratkaisuun, mutta pienillä muutoksilla sillä voi ratkaista myös reunatäsmäyspelejä, joissa reunapaloja ei käytetä. Algoritmi aloittaa reunatäsmäyspelin ratkaisemisen pelilaudan vasemmasta yläkulmasta. Tämän jälkeen suoritus jatkuu vasemmalta oikealle rivi kerrallaan.

Jokaisessa pelilaudan kohdassa on voimassa erilaisia paloille asetettavia rajoitteita. Sijoittamattomista paloista hyväksytään kulloiseenkin pelilaudan kohtaan ehdokkaiksi ainoastaan ne palat, jotka täyttävät kulloinkin voimassa olevat rajoitteet. Aloittaessaan vasemmasta yläkulmasta algoritmi valitsee kaikkien palojen listalta yhden palan kerrallaan. Se selaa paloja läpi, kunnes löytää palan, joka on määritelty pelilaudan kulmaan sopivaksi palaksi. Tällaisella palalla on siis kaksi sellaista sivua, jotka kuuluvat pelilaudan reunalle. Kun sopiva pala on löydetty, se sijoitetaan pelilaudalle. Jos pala ei ole oikeassa asennossa, sitä kierretään, kunnes se sopii paikoilleen.

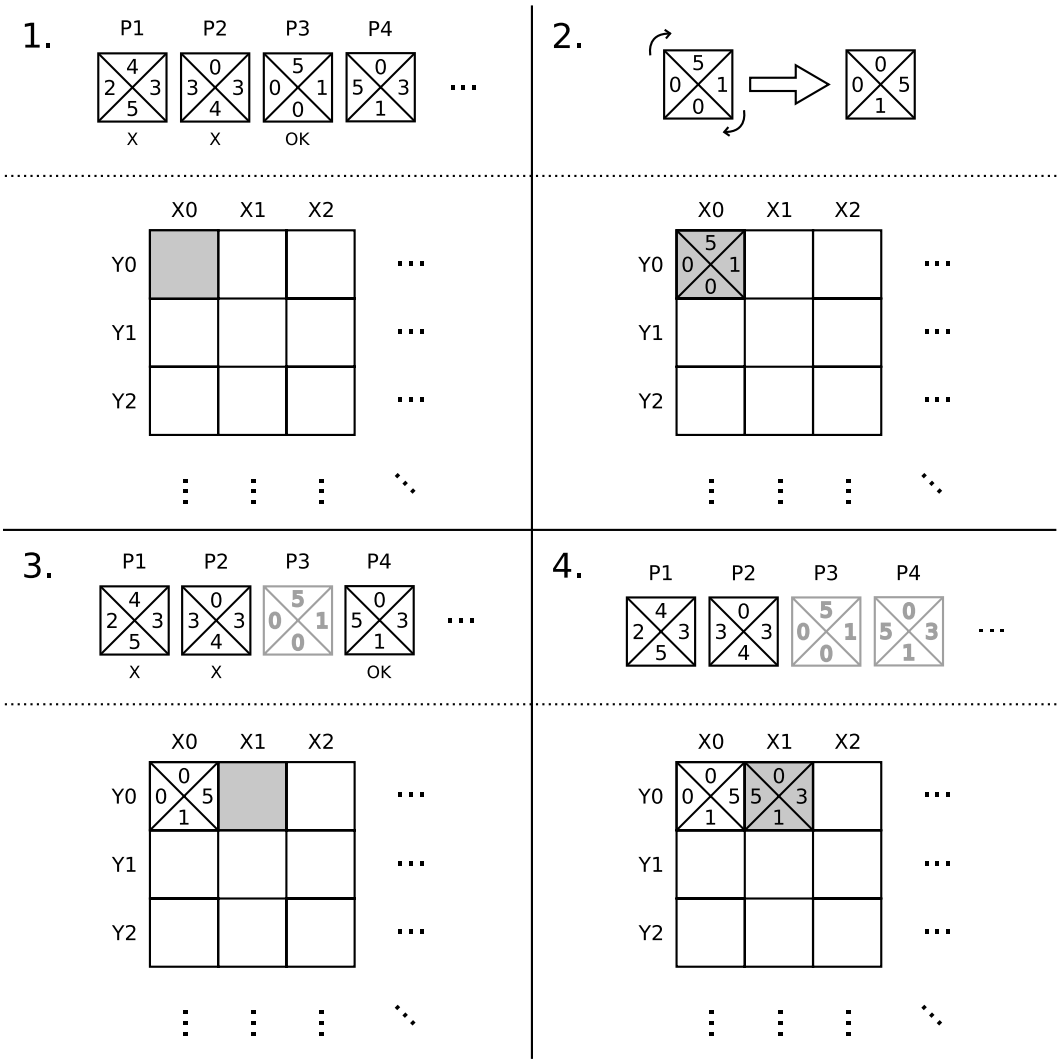
Nyt algoritmi siirtyy pelilaudalla seuraavaan ruutuun (oikealle). Koska tämä sijainti on pelilaudan yläreunassa, täytyy sijoitettavalta palalta löytyä yksi (ja vain yksi) sellainen sivu, jonka voi sijoittaa pelilaudan reunalle. Algoritmi käy jälleen käyttämättömiä paloja läpi niin kauan, kunnes se löytää pelilaudan reunalle sopivan palan. Tämän jälkeen se tarkistaa, löytyykö valitusta palasta sellaista sivua, joka on samanlainen kun edellisen paikalleen asetetun palan oikean puoleinen sivu. Jos käsiteltävästä palasta löytyy tällainenkin sivu, yritetään pala asettaa pelilaudalle. Jos se ei heti sovi, käännetään palaa ja kokeillaan uudestaan.

Näin edetään, kunnes päädytään ensimmäisen rivin viimeisen kohtaan. Tähän kohtaan tulevalta palalta vaaditaan jälleen pelilaudan kulmaan tulevan palan ominaisuudet. Kun kulmapala on löytynyt, jatketaan suoritusta seuraavan rivin ensimmäisestä ”lokerosta”.

Algoritmi jatkaa toimintaansa edellä kuvatulla tavalla huomioiden käsiteltävän kohdan ympärillä olevat mahdollisesti jo aikaisemmin asetetut palat. Haettavasta palasta täytyy siis aina löytyä sen viereisten palojen vastinsivut.

Jos algoritmi on ajautunut suorituksessaan siihen pisteeseen, ettei käsiteltävään kohtaan sovi yksikään jäljellä olevista paloista, suoritetaan peruuttaminen (engl. *backtracking*). Tämä tarkoittaa sitä, että algoritmi poistaa laudalta viimeksi asetetun palan ja asettaa poistetun palan tilalle seuraavan mahdollisen sopivan palan. Jos sopivaa palaa ei edelleenkään löydy, suoritetaan uusi peruuttaminen.

Jokaisen palayhdistelmän kokeileminen takaa sen, että jos algoritmi saa syötteekseen oikeelliset palat, se myös löytää ratkaisun näille paloille. Tämä tarkoittaa sitä, että jos algoritmi ei löydä yhtäkään ratkaisua annetuille paloille, ei näistä paloista voi tällöin koota täydellistä ratkaisua.



Kuva 3. Esimerkki käsiteltävän ratkaisualgoritmin toiminnasta.

Kuvassa 3 havainnollistetaan edellä kuvatun ratkaisualgoritmin toimintaa. Ensimmäisessä vaiheessa haetaan palaa, joka sopii pelilaudan vasempaan yläkulmaan koordinaatteihin (X0, Y0). Paloja siis selataan, kunnes löytyy sellainen

pala, jolla on kaksi pelilaudan reunaa vasten tulevaa sivua. Paloista P3 on ensimmäinen, joka täyttää tämän ehdon.

Toisessa vaiheessa pala asetetaan paikoilleen, mutta nyt huomataan, että pelilaudan ulkoreunaa vastaan tulevat sivut eivät ole oikeassa kohdassa. Palaa täytyy siis kiertää. Tässä tapauksessa yksi kierto riittää, ja pala sopii paikoilleen. Algoritmi siirtyy pelilaudan seuraavaan vapaaseen kohtaan koordinaatteihin (X1, Y0).

Kolmannessa vaiheessa etsitään sellaista palaa, jolla on yksi pelilaudan reunaa vasten tuleva sivu. Tämän lisäksi palasta täytyy löytyä vastinsivu edellisen asetetun palan kanssa. Pala P4 toteuttaa molemmat näistä ehdoista.

Neljännessä vaiheessa valittua palaa sovitetaan pelilaudalle, ja tällä kertaa kiertoa ei tarvita, sillä pala sopii suoraan paikoilleen.

Algoritmi 1 on pseudokoodinen listaus käsiteltävän ratkaisualgoritmin toiminnasta:

Algoritmi RTP-Ratkaisu(*A*, *P*, *sij*)

Syötteet: Pelilauta *A*, jossa paikat yhteensä *n* palalle. Taulukko *P*, joka sisältää *n* kappaletta paloja, joista pystyy kokoamaan täydellisen ratkaisun pelilaudalle A. Pelilaudan käsiteltävä sijainti *sij*. Algoritmin käynnistyessä *sij* == *alku*.

Tuloste: Pelilauta *A*, johon taulukon *P* kaikki palat on ladottu reunatäsmäyspelien sääntöjen mukaisesti.

```
RTP-Ratkaisu(A, P, sij)
begin
    if sij == loppu then
        return A
    for each pala P[i] in P do
        if pala P[i] täyttää sijainnin sij ehdot then
                for each kiertymä do
                        if pala P[i] sopii pelilaudan A sijaintiin sij then
                                merkitse pala P[i] käytetyksi
                                aseta pala P[i] pelilaudan A sijaintiin sij
                                tulos ← RTP-Ratkaisu(A, P, sij + 1)
                                if tulos != JATKA then
                                        return tulos
                                end if
                                poista pala P[i] pelilaudan A sijainnista sij
```

```
                    merkitse pala P[i] käyttämättömäksi
                    käännä palaa P[i]
                else
                    käännä palaa P[i]
                end if
            end for
        end if
    end for
    return JATKA
end
```

Algoritmi 1. Reunatäsmäyspelin ratkaiseva rekursiivinen algoritmi.

Koska algoritmi 1 käy suorituksensa aikana kaikki mahdolliset palojen sijoitukset läpi (kunnes se löytää jonkin oikean ratkaisun), perustuu sen toiminta näin ollen väsytysmenetelmään. Tämä väsytysmenetelmä on kuitenkin eräällä tavalla ”viisas”, sillä se käy rajoitteiden ansiosta monta kertaluokkaa vähemmän vaihtoehtoja läpi kuin ”tyhmä” väsytysmenetelmä, joka ainoastaan kokeilee järjestyksessä jokaista mahdollista sijoitusta välittämättä palojen välisistä suhteista.

Huomioitavaa on, että algoritmi 1 sopii niin reunapalallisten kuin reunapalattomienkin reunatäsmäyspelien ratkaisuun, vaikka itse algoritmin toteutus on tehty ainoastaan reunapalallisille reunatäsmäyspeleille.

4.3. Reunatäsmäyspelien luominen

Kaikki samankokoiset reunatäsmäyspelit eivät ole yhtä helppoja tai vaikeita ratkaista, vaikka näin voisi ehkä kuvitella. Reunatäsmäyspelien paloilla ei ole muita ominaisuuksia kuin niiden sivujen sisältämät tunnisteet, joten nämä tunnisteet määrittelevät täysin samankokoisten reunatäsmäyspelien väliset vaikeussuhteet.

Hyvä esimerkki on reunatäsmäyspeli, jonka jokaisen palan reunoilla käytetään vain yhtä ainoaa tunnistetta. Tällainen peli on täysin triviaali ratkaista, sillä jokainen pala käy mihin tahansa pelilaudan kohtaan missä tahansa asennossa. Toinen ääritilanne on sellainen, jossa jokaisella pelilaudan vastinsivuparilla on oma yksilöllinen tunnisteensa. Myös tämänkaltaisen reunatäsmäyspelin ratkaisu on lähestulkoon triviaali, sillä ratkaisu on täysin yksiselitteinen; jos ratkaisija löytää kaksi palaa, joiden jollain sivulla on sama tunniste, hän voi olla varma, että nämä palat ovat vierekkäin myös valmiilla pelilaudalla. Tällainen reunatäsmäyspeli muistuttaa läheisesti palapeliä.

Selvästi huomataan, että vaikeimmissa reunatäsmäyspeleissä palojen erilaisten tunnisteiden määrä on jossain näiden kahden ääripään välillä.

Reunapaloja käyttävissä reunatäsmäyspeleissä ratkaisun löytymisen vaikeuteen vaikuttaa yleisen tunnisteiden määrän lisäksi myös pelilaudan reunoille tulevien palojen tunnisteiden valinta. Esimerkiksi Eternity II -pelissä pelilaudan reunoille tulevien palojen niille sivuille, jotka koskettavat toisia reunapaloja, on valittu ainoastaan viittä eri tunnistetta (*ulkotunniste*). Muihin paloihin ja reunapalojen pelikentälle päin osoittaville sivuille tunnisteet on valittu seitsemäntoista eri tunnisteen joukosta (*sisätunniste*). Nämä seitsemäntoista tunnistetta eroavat pelilaudan ulkoreunaan tulevien palojen viidestä tunnisteesta. Yhteensä erilaisia tunnisteita on siis 22 kappaletta (ellei pelilaudan ulkoreunaa merkitseviä tunnisteita lasketa). Mitä todennäköisimmin tunneisteiden suhteet on valittu niin, että Eternity II on mahdollisimman vaikea ratkaista. [Scaus and Deville, 2008]

Myös identtisten ja symmetristen palojen määrällä on suuri vaikutus reunatäsmäyspelien vaikeuteen. Jos pelilaudalla on edes muutamia identtisiä tai symmetrisiä paloja, reunatäsmäyspelin mahdollisten (redundanttien) ratkaisujen määrä moninkertaistuu. Symmetristen ja identtisten palojen vähentäminen vaikeuttaa siis ratkaisun löytymistä huomattavasti.

Algoritmin 1 testausta varten luoduissa reunatäsmäyspelien ilmentymissä ei ole huomioitu lainkaan symmetrisiä eikä identtisiä paloja.

4.4. Ratkaisualgoritmin testaus

Tässä kohdassa testataan kohdassa 4.2. esitetyn ratkaisualgoritmin toimintaa käytännössä. Huomiota kiinnitetään myös siihen, miten reunatäsmäyspelin palojen tunnisteiden erilaiset lukumäärät vaikuttavat pelin ratkaisuaikaan.

Testattava reunatäsmäyspelien ratkaisualgoritmi on pyritty toteuttamaan mahdollisimman tehokkaaksi. Useimmin toistuvat operaatiot on optimoitu nopeuden maksimoimiseksi. Tehokkuuten pyrkimisestä huolimatta algoritmi pystyy käyttämään vain yhtä prosessoriydintä yhden reunatäsmäyspelin ilmentymää kohden. Algoritmi on toteutettu Java-ohjelmointikielen versiolla 1.6. Kaikki algoritmilla suoritetut testit ajetaan käyttäen kahta tavallista kannettavaa tietokonetta, joiden prosessoreina ovat Intel Core 2 Duo T7250 (2,0 GHz) ja Intel Core 2 Duo T7500 (2,2 GHz). Testit on hajautettu kahdelle eri tietokoneelle testien suorittamiseen kuluvan suuren ajan vuoksi. Molemmissa testitietokoneissa on kaksi prosessoriydintä, joten neljää eri ratkaisualgoritmin ilmentymää voidaan ajaa samanaikaisesti.

Testit ajetaan erillisissä sarjoissa, joiden parametrit eroavat toisistaan. Kussakin testisarjassa luodaan sata erillistä kunkin testin mukaista reunatäsmäyspeliä, jotka ratkaistaan peräjälkeen. Sadan testin jälkeen kunkin testisarjan tuloksista lasketaan suoritusaikojen tunnuslukuja. Saatuja lukuja verrataan

Ansóteguin ja muiden [2008] saamiin tuloksiin. Ansóteguin ja muiden tekemään tutkimukseen viitataan jäljempänä nimellä Ansóteguin tutkimus.

Testejä ajetaan pelilaudoilla, joiden koko on 6 x 6 palaa ja 7 x 7 palaa. Koon lisäksi testisarjat eroavat toisistaan peleihin luotavien palojen tunnisteiden määrässä. Alla on kaikkien testisarjojen ominaisuudet:

6 x 6 palan pelilauta, jossa 6 sisätunnistetta ja 2 ulkotunnistetta
7 x 7 palan pelilauta, jossa 6 sisätunnistetta ja 4 ulkotunnistetta
7 x 7 palan pelilauta, jossa 7 sisätunnistetta ja 2 ulkotunnistetta
7 x 7 palan pelilauta, jossa 7 sisätunnistetta ja 3 ulkotunnistetta
7 x 7 palan pelilauta, jossa 7 sisätunnistetta ja 4 ulkotunnistetta
7 x 7 palan pelilauta, jossa 8 sisätunnistetta ja 2 ulkotunnistetta.

Testisarjat ja testien määrät ovat samat kuin Ansóteguin suorittamissa testeissä, pois lukien reunapalattomat reunatäsmäyspelit, joita tässä ei testata ratkaisualgoritmin soveltumattomuuden vuoksi.

Taulukossa 1 on algoritmilla 1 saavutetut tulokset sekä taulukossa 2 Ansóteguin tutkimuksen tulokset.

	Pelilaudan koko (sisätunnisteiden määrä:ulkotunnisteiden määrä)					
	6 x 6 (6:2)	**7 x 7 (6:4)**	**7 x 7 (7:2)****	**7 x 7 (7:3)***	**7 x 7 (7:4)***	**7 x 7 (8:2)***
Mediaani (s)	1,55	302	200	324	38*	413
Keskiarvo (s)	3,25	1761	2291	603	128*	1095
Keskihajonta (s)	4,29	4186	7813	863	216*	2094
Minimi (s)	~0,00	0,53	0,45	4,88	0,30*	5,67
Maksimi (s)	23	26890	61924	5141	1347*	11134

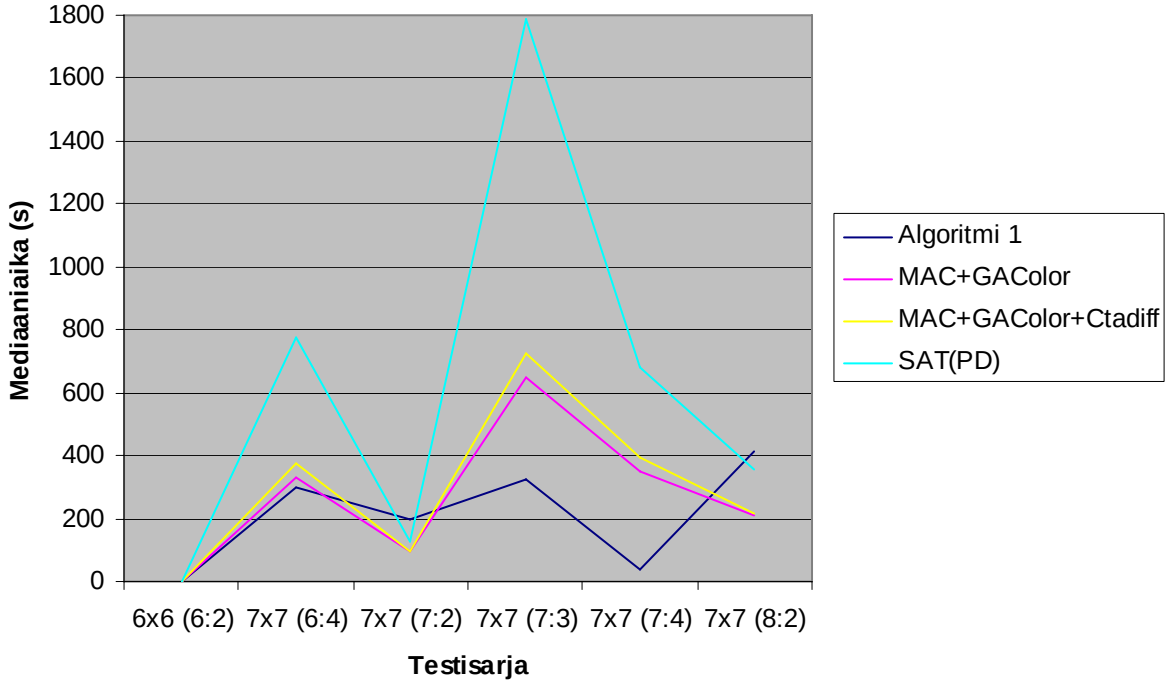
Taulukko 1. Tutkielmassa käsitellyn ratkaisualgoritmin testitulokset.[3]

Alla käytetty algoritmi ja taulukossa testiryhmän mediaanitulos (s)	**Pelilaudan koko (sisätunnisteiden määrä:ulkotunnisteiden määrä)**					
	6 x 6 (6:2)	**7 x 7 (6:4)**	**7 x 7 (7:2)**	**7 x 7 (7:3)**	**7 x 7 (7:4)**	**7 x 7 (8:2)**
PLA-DOM	15	520	18193	9464	581	8387
PLA-CHESS	0,5	5249	137	4181	6906	510
MAC+GAColor	0,94	328	96	646	348	208

[3] Yhdellä tähdellä merkityt testisarjat on suoritettu tietokoneella, jonka suorittimena on Intel Core 2 Duo T7500. Kahdella tähdellä merkityissä tuloksissa laskentaa suorittivat molemmat käytetyistä tietokoneista. Muut testisarjat on suoritettu tietokoneella, jonka suorittimena on Intel Core 2 Duo T7250.

MAC+GAColor+ CTadiff	0,73	377	94	727	395	216
SAT(P)	7,45	>20000	4418	>20000	7960	6465
SAT(PD)	0,55	777	125	1785	682	359
MAC$_b$ dom/deg	19	2415	>20000	>20000	3307	>20000
Minion	125	3463	>20000	>20000	4675	>20000

Taulukko 2. Ansóteguin tutkimuksen tulokset. [Ansótegui et al., 2008]



Kuva 4. Nopeimpien ratkaisualgoritmien mediaaniajat eri testisarjoilla.

5. Yhteenveto

Saadut tulokset eivät ole täysin vertailukelpoisia Ansóteguin tulosten kanssa, sillä Ansóteguin tutkimuksessa algoritmien testaukseen käytetty tietokone[4] on erilainen kuin tässä tutkimuksessa. Lisäksi tämän tutkimuksen testit on ajettu kahdella eri tietokoneella.

Testitietokone, jonka suorittimena on Intel Core 2 Duo T7250, sijoitti pelilaudalle keskimäärin 1200 palaa millisekunnissa. Toinen testitietokone suorittimenaan Intel Core 2 Duo T7500 sijoitti palojen pelilaudalle keskimäärin 1300 palan millisekuntivauhtia. Tietokoneiden välinen nopeusero oli näin ollen noin 8 %. Kahden erilaisen tietokoneen käyttö vääristää tuloksia hiukan, mutta saatujen tuloksien välinen suuruusluokka pysyy silti samana.

Ansóteguin tutkimuksessa reunatäsmäyspelien luomiseen käytetyn algoritmin yksityiskohdista ei ole myöskään tietoa, joten varmuudella ei voida sanoa,

[4] 64-bittinen AMD Opteron (1,8 GHz)

oliko testisarjoissa ratkaistavien reunatäsmäyspelien vaikeustaso samaa luokkaa. Näennäisesti vähäisellä vaikeustason muutoksella on erittäin suuri vaikutus reunatäsmäyspelin ratkaisuaikaan.

Saadut tulokset antavat ymmärtää kuvan 4 perusteella, että toteuttamani algoritmi vaikuttaa olevan ilmeisen nopea. Toisaalta viimeisessä testisarjassa (7 x 7 (8:2)) algoritmini oli selvästi hitaampi kuin Ansóteguin tutkimuksen nopeimmat algoritmit. Mielenkiintoista olisi nähdä, kasvaisiko ero esimerkiksi testisarjalla 7 x 7 (8:3). Testisarjan 7 x 7 (7:4) tulos herättää hieman hämmennystä, sillä sarjan mediaaniaika oli vain noin kymmenesosa Ansóteguin tutkimuksen saman testisarjan nopeimman algoritmin (MAC+GAColor) suoritusajasta. Algoritmini saattaa tosin soveltua erityisen hyvin reunatäsmäyspeleille, joilla on niinsanotusti soveliaat parametrit.

Kuten näemme, tulosten keskiarvoajat eroavat hyvin paljon mediaaniajoista, sillä tulosten hajonta on sangen suuri. Olisi ollut mielenkiintoista verrata saamiani keskimääräisiä ratkaisuaikoja Ansóteguin tulosten keskiarvoaikoihin, mutta näitä tietoja ei ollut valitettavasti saatavilla.

Koska jo melko yksinkertaisella periaatteella toimivalla reunatäsmäyspelin ratkaisualgoritmilla saatiin tässä tutkielmassa aikaan vähintäänkin kohtuullisia tuloksia, kehittyneemmillä menetelmillä saataneen ratkaisualgoritmin nopeutta parannettua mahdollisesti paljonkin. Nähtäväksi jää, onko mikään reunatäsmäyspelien ratkaisualgoritmi kyennyt vielä ratkaisemaan Eternity II -peliä.

Viiteluettelo

[Ansótegui et al., 2008] Carlos Ansótegui, Ramón Bèjar, César Fernàndez and Carles Mateu, Edge matching puzzles as hard SAT/CSP benchmarks. In: *Proc. of the 14th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* **5202**, Springer, 560-565.

[CNF, 2008] Conjunctive Normal Form – from Wolfram MathWorld. Available as http://mathworld.wolfram.com/ConjunctiveNormalForm.html. Checked 18.12.2008.

[Cormen et al., 2007] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms, 2nd ed.* MIT Press, 2007.

[Demaine and Demaine, 2007] Erik D. Demaine and Martin L. Demaine, Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity. *Graphs and Combinatorics* **23** (June 2007), 195-208.

[Eternity II, 2007] About Eternity II. Available as http://uk.eternityii.com/about-eternity2. Checked 13.12.2008.

[Gasarch, 2002] William I. Gasarch, Guest column: The P=?NP poll. *ACM SIGACT News* **33**, 2 (June 2002), 34-47 .

[Hopcroft et al., 2003] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman,

Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 2003.

[McAdam, 2004] Puzzle History. *American Jigsaw Puzzle Society*. Available as http://www.jigsaw-puzzle.org/jigsaw-puzzle-history.html. Checked 27.10. 2008.

[Piccione, 1980] Peter A. Piccione, In search of the meaning of Senet. *Archaeology* **33** (July/August, 1980), 55-58. Available as http://www.cofc.edu/~piccione/piccione_senet.pdf. Checked 10.11.2008.

[Scaus and Deville, 2008] Pierre Schaus and Yves Deville, Hybridization of CP and VLNS for Eternity II, *Actes JFPC 2008.*

[Takenaga and Walsh, 2006] Yasuhiko Takenaga and Toby Walsh, Tetravex is NP-complete. *Information Processing Letters* **99**, 5 (September 2006), 171-174.

[Tetravex, 1991] Tetravex – Wikipedia, the free encyclopedia. Available as http://en.wikipedia.org/wiki/TetraVex. Checked 17.11.2008.

Vahvan todentamisen mahdollisuudet ja haasteet kouluympäristössä

Aapo Laitinen

Tiivistelmä.
Suomalaisissa peruskouluissa ja toisen asteen oppilaitoksissa on käytössä runsaasti web-sovelluksia. Tässä tutkielmassa tarkastellaan salasanojen ongelmia henkilöllisyyden todentamisessa sekä tekijöitä, jotka joudutaan ottamaan huomioon suunniteltaessa vahvan todentamisen käyttöönottoa. Kouluympäristön erityisiä ongelmia ovat jaetuissa tiloissa sijaitsevat tietokoneet sekä korkeat kustannus- ja tehokkuusvaatimukset.

Avainsanat ja -sanonnat: tietoturva, pääsynvalvonta, sähköinen tunnistaminen, vahva tunnistaminen, todentaminen, vahva todentaminen, todennus, vahva todennus, oppilaitokset.
CR-luokat: J.1, K.6.5

1. Johdanto

Suomalaiset peruskoulut ja toisen asteen oppilaitokset ovat 2000-luvun alkupuolella ottaneet käyttöönsä laajan joukon web-sovelluksia. Vaikka verkko-opetus ja oppimisalustat ovat yhä verrattain vähäisessä käytössä, tapahtuu arvosanojen kerääminen, kurssi-ilmoittautuminen ja poissaolojen seuranta usein verkossa, ja verkkoviestinnästä on tulossa yksi kodin ja koulun yhteistyön kulmakivistä.

Tietoja verkkoon vietäessä pitää kuitenkin noudattaa varovaisuutta, sillä ilman minkäänlaisia suojaustoimenpiteitä tiedot olisivat periaatteessa puolentoista miljardin ihmisen nähtävillä. Kyse ei ole pelkästä hypoteettisesta uhasta ja käytännössä “vihollisia” löytyy jo saman rakennuksen sisäpuolelta. Nuorten näyttämishalu ja ihmissuhdepulmat kun voivat nykyisin purkautua tietomurtoina.

Ehkä tyypillisin suojaustoimenpide on tietojen näyttäminen vain henkilöille, joilla katsotaan olevan oikeus niiden näkemiseen. Jotta tämä voitaisiin toteuttaa käytännössä, pitää henkilön jotenkin pystyä todistamaan henkilöllisyytensä tietojärjestelmälle. Nykyisin tämä tapahtuu yleisimmin salasanalla tai mikäli luotettavampi todistaminen on tarpeen, verkkopankkitunnuksilla.

Tässä tutkielmassa tarkastellaan salasanojen ongelmia todentamismenetelmänä erityisesti peruskoulujen ja toisen asteen oppilaitosten (jatkossa lyhyesti koulujen) web-sovellusten yhteydessä ja esteitä, joita muiden menetelmien ja

vahvan todentamisen käyttöönottamiseen liittyy. Menetelmien teknisiä piirteitä tai niiden toteuttamiseen liittyviä seikkoja ei tässä tutkielmassa juuri esitellä.

2. Tausta ja termit

Henkilötietolaki (523/1999) sääntelee kaikkien suomalaisen yhteiskunnan toimijoiden, kuten yksityishenkilöiden, yhteisöjen ja laitosten sekä viranomaisten, oikeutta käsitellä henkilötietoja silloin, kun muissa laeissa ei ole kyseistä tilannetta koskevia erityissäädöksiä. Henkilötietojen käsittely edellyttää aina riittävien perusteiden täyttymistä. Koulun tapauksessa peruste muodostuu oppilaan ja koulun välisestä oppilassuhteesta, jota voidaan pitää eräänlaisena asiakassuhteena, sekä henkilökunnan ja koulun välisestä palvelussuhteesta. Tällöinkin koulu voi käsitellä ainoastaan sellaisia oppilaiden ja henkilökunnan henkilötietoja, jotka ovat tarpeen opetuksen järjestämiseksi. (Tietosuojavaltuutettu, 2001)

Koulun käsittelemiä henkilötietoja ovat esimerkiksi oppilaiden, opettajien ja muun henkilökunnan nimet, osoitteet ja henkilötunnukset; oppilaiden koe-, kurssi- ja loppuarvosanat sekä erityisopetuksen järjestämiseksi ja turvallisuuden takaamiseksi oleelliset tiedot terveydentilasta; opettajien ja muun henkilökunnan työhönottoa, työhistoriaa ja palkkausta koskevat tiedot. Joidenkin tietojen käsittely on välttämätöntä laissa määrättyjen velvoitteiden täyttämiseksi, jollainen on esimerkiksi poissaolojen seuraaminen ja niistä ilmoittaminen huoltajalle.

Henkilötietolain 32 § velvoittaa myös suojaamaan henkilötietoja "asiattomalta pääsyltä tietoihin ja vahingossa tai laittomasti tapahtuvalta tietojen hävittämiseltä, muuttamiselta, luovuttamiselta, siirtämiseltä taikka muulta laittomalta käsittelyltä". Tietoja suojataan toimenpiteillä, "[joiden] toteuttamisessa on otettava huomioon käytettävissä olevat tekniset mahdollisuudet, toimenpiteiden aiheuttamat kustannukset, käsiteltävien tietojen laatu, määrä ja ikä sekä käsittelyn merkitys yksityisyyden suojan kannalta".

Koulujen kannalta on myös oleellista, että julkisuuslain 24 § (Laki viranomaisten toiminnan julkisuudesta, 621/1999) määrää esimerkiksi koesuoritukset ja todistukset, jotka sisältävät oppilaan henkilökohtaisten ominaisuuksien sanallista arviointia, salassa pidettäviksi. Julkisuuslain 18 §:ssä edellytetty hyvä tiedonhallintatapa vaatii henkilötietolain tavoin mm. että "asiakirjojen ja tietojärjestelmien sekä niihin sisältyvien tietojen suoja, eheys ja laatu turvataan asianmukaisin menettelytavoin ja tietoturvallisuusjärjestelyin ottaen huomioon tietojen merkitys ja käyttötarkoitus sekä asiakirjoihin ja tietojärjestelmiin kohdistuvat uhkatekijät ja tietoturvallisuustoimenpiteistä aiheutuvat kustannukset".

Tietoturva (information security) tarkoittaa tietojärjestelmän ja tietojärjestelmään tallennetun tiedon suojelua tarkoituksellisia hyökkäyksiä vastaan erilai-

silla toimenpiteillä eli tietoturvatoimilla. Toimenpiteitä on kolme lajia: *estäminen* (prevention) pyrkii estämään hyökkäystä tapahtumasta, *havaitseminen* (detection) pyrkii tekemään tapahtuneen hyökkäyksen näkyväksi järjestelmän ylläpitäjille ja *vaste* (response) puolestaan pyrkii rajoittamaan vahinkoja, korjaamaan ne ja parantamaan toimenpiteitä jatkossa kohdattavien vastaavanlaisten hyökkäysten varalta. (Schneier, 2003)

Yksi tärkeimmistä hyökkäyksiä estävistä tietoturvatoimista on *pääsynvalvonta* (access control), joka pyrkii varmistamaan, että järjestelmään ja sen eri osiin pääsevät käsiksi ainoastaan oikeutetut henkilöt. Pääsynvalvonta voidaan jakaa kolmeen vaiheeseen: *tunnistaminen* (identification) tarkoittaa henkilön yksilöimistä nimellä tai numerolla, *todentaminen* (authentication) sen vahvistamista, että henkilö on tosiaan se, joksi hänet on tunnistettu, ja *valtuuttaminen* (authorization) oikeiden toimintojen sallimista ja epäämistä sen perusteella, keneksi henkilö on tunnistettu. (Schneier, 2003) On tärkeää huomata, että vaikka nämä termit ovat vakiintuneita englanninkielisessä tietoturvaa koskevassa tieteellisessä keskustelussa, niitä käytetään tietojenkäsittelyn ammattikirjallisuudessa ja suomenkielisissä teksteissä runsaasti toistensa synonyymeinä. Erityisen harmillista on, että suomalaisissa julkisen hallinnon julkaisuissa "tunnistamisella" tarkoitetaan lähes yksinomaan todentamista.

3. Todentaminen ja vahva todentaminen

Henkilöllisyyden todentamisessa käytettävät menetelmät jaetaan perinteisesti kolmeen ryhmään: henkilön tietämään tietoon perustuviin (something you know), henkilöllä olevaan esineeseen perustuviin (something you have) ja henkilön fyysisiin ominaisuuksiin perustuviin (something you are). (O'Gorman, 2003)

3.1. Tietoon perustuva todentaminen

Esimerkkejä tietoon perustuvista todennusmenetelmistä ovat salasana, tunnusluku (ns. PIN-koodi), henkilötunnus ja henkilöhistoriaan liittyvät tiedot (kuten syntymäaika, synnyinpaikka ja vanhempien nimet). Tällä hetkellä salasanat ja tunnusluvut ovat ylivoimaisesti hallitsevia todennusmenetelmiä, sillä niiden syöttäminen ei vaadi erityistä laitteistoa ja loppukäyttäjä voi valita oman salasanansa tai tunnuslukunsa, jolloin niiden toimittaminen on helppoa. Henkilötunnusta ja henkilöhistoriaan liittyviä tietoja käytetään lähinnä täydentämään muita menetelmiä esimerkiksi kadonneen salasanan uudistamisen yhteydessä, sillä henkilötunnus ja henkilöhistoriaan liittyvät tiedot löytyvät lukuisista yritysten ja yhteisöjen tietokannoista ja henkilöhistoriaan liittyviä tietoja henkilö on tyypillisesti paljastanut ainakin lähipiirilleen ja ehkä maailmallekin, vaikkapa Facebook-profiilissaan tai blogissaan. Salasanoilla ja tunnusluvuilla on kuitenkin merkittäviä ongelmia, joiden takia niille on alettu miettiä vaihtoehtoja.

Salasanat ovat helposti vakoiltavissa. Eräässä tutkimuksessa kokemattomat koehenkilöt pystyivät vakoilun kannalta optimaalisissa olosuhteissa vakoilemaan keskimäärin 3,65 merkkiä viidestä satunnaisesta merkistä koostuvasta salasanasta (Tari, Ozok & Holden, 2006). Salasanoja voidaan vakoilla myös teknisesti eli vakoiluohjelmilla, näppäinpainallukset tallentavilla laitteilla (keylogger) ja kameroilla. Ihmiset jakavat salasanojaan myös vapaaehtoisesti. Australialaisessa verkkopankkipalveluja koskevassa haastattelututkimuksessa salasanansa jakavat parit perustelivat tätä luottamuksen osoittamisena toista kohtaan, tarpeella hoitaa toisen asioita, toisen vähäisellä kiinnostuksella tietotekniikkaan ja hallinnollisten hankaluuksien välttämisellä (Singh et al., 2007). Salasana saattaa paljastua myös arvaamalla tai systemaattisen kokeilun seurauksena, sillä optimaalinen salasana eli satunnainen merkkijono on hankalasti muistettavissa ja ihmiset siksi mielellään valitsevat yksinkertaisia salasanoja tai salasanoja, joilla on jotain merkitystä heille.

Nämä ongelmat on tunnettu jo 1960-luvulta lähtien ja niitä on yritetty torjua salasanakäytännöillä (password policy). Käytäntö saattaa asettaa vaatimuksia esimerkiksi salasanan pituudelle, sen sisältämien merkkien monimuotoisuudelle (esimerkiksi että salasanan tulee sisältää sekä kirjaimia että numeroita) ja salasanan vaihtotiheydelle. (Morris & Thompson, 1979) Salasanakäytännöt kuitenkin tyypillisesti pahentavat muistettavuusongelmaa. Ratkaisuksi on ehdotettu muistisääntöjen käyttämistä salasanojen valintaan (esimerkiksi Yan, Blackwell, Anderson & Grant, 2000), salasanojen tallentamista huolellisesti säilytettävään muistikirjaan ja erilaisia graafisia salasanoja (joissa käyttäjä esimerkiksi napauttaa tiettyjä kohtia suuressa kuvassa tai poimii tietyt kuvat muiden joukosta, katso esimerkiksi Tari ja muut, 2006).

3.2. Esineeseen perustuva todentaminen

Esineeseen perustuvassa todentamisessa käyttäjä esittää esineen, jonka hallussapito todentaa hänen henkilöllisyytensä. Ajokortin esittäminen virastossa asioitaessa ja oven avaaminen avaimella ovat esimerkkejä esineeseen perustuvasta todentamisesta tietojenkäsittelyn ulkopuolella.

Älykortti (toimikortti, smart card) on mikrosirulla varustettu, luottokortin kokoinen kortti, joka suunniteltu siten, että mikrosirulle tallennetun salausavaimen kopioiminen on vaikeaa. Älykortti laitetaan tietokoneeseen liitettyyn lukijaan ja tunnusluku, jos sellaista edellytetään, syötetään tietokoneeseen tai lukijaan. Tämän jälkeen käyttäjä on todennettu kunnes hän ottaa älykortin pois lukijasta.

Salasanalaskuri on pieni, numeronäytöllä varustettu laite. Mallista riippuen se joko näyttää numeroista koostuvan salasanan jatkuvasti ja vaihtaa sitä kerran minuutissa tai näyttää uuden salasanan hetken ajan kun laitteessa olevaa painiketta on painettu. Salasana lasketaan laitteeseen tallennetun salausavaimen pe-

rusteella. Salasanalaskurin näyttämä salasana joko syötetään tavallisen salasanan sijasta tai sen lisäksi.

Avainlukulistoja on useita lajeja. Nykyisin suomalaisten verkkopankkien käyttämät listat koostuvat kahdesta numerosarakkeesta, joista ensimmäinen sisältää kasvavan luvun ja toinen satunnaiselta näyttävän luvun. Käyttäjätunnuksella ja tavallisella salasanalla kirjautumisen jälkeen annetaan käyttäjälle ensimmäisestä sarakkeesta löytyvä luku ja häntä pyydetään syöttämään viereinen luku. Pankit tarjoavat todennuspalveluja, joissa asiakas pystyy todentautumaan kolmannelle osapuolelle pankin avainlukulistan avulla.

Radiotunnistus (RFID) perustuu antennista ja mikropiiristä koostuvaan tunnisteeseen, joka on henkilöntodennussovelmissa yleensä upotettu luottokortin kokoiseen korttiin tai avaimenperään. Kun tunnisteen sisältävä esine viedään lukijan lähelle, vaihtavat tunniste ja lukija keskenään tietoa. Lukuetäisyys ja tietoturvaominaisuudet, kuten suojaus kopiointia vastaan, riippuvat tunnisteen mallista.

Henkilön tunnistamiseen kännykällä on erilaisia tapoja. Kännykkään voidaan lähettää tekstiviesti, jonka sisältämä avainluku henkilöä pyydetään syöttämään tietokoneeseen. Vaihtoehtoisesti henkilöä voidaan pyytää lähettämään tietokoneen näyttämä avainluku tekstiviestinä tiettyyn numeroon. Kännykkään voidaan myös asentaa ohjelmisto, joka toimii kuin salasanalaskuri. Lisäksi on olemassa *mobiilivarmenteita* eli SIM-kortille tallennettuja salausavaimia (Väestörekisterikeskuksen allekirjoittama kansalaisvarmenne tai operaattorin allekirjoittama yritysvarmenne). Kännykkää voidaan myös käyttää radiotunnistuksen tyylisesti NFC-teknologialla (Near Field Communication).

Salausavain voidaan tallentaa erillisen laitteen sijasta myös tietokoneelle. Tietokoneelle tallennettujen salausavainten käyttöä on hidastanut käyttöliittymien vaikeakäyttöisyys ja salausavainten tietokoneelta toiselle siirtämisen vaikeus. Microsoft CardSpace pyrkii tarjoamaan yhtenäisen käyttöliittymän tietokoneelle tallennettujen salausavainten hallintaan, hyödyntämiseen verkkopalveluissa ja sovelluksissa sekä siirrettävillä tallennusvälineillä säilyttämiseen.

3.3. Fyysisiin ominaisuuksiin perustuva todentaminen

Ihmisen henkilöllisyys voidaan todentaa myös *biometrisesti* eli hänen fyysisten ominaisuuksiensa perusteella. Todentamiseen voidaan käyttää sellaisia henkilön fysiologiaan tai käyttäytymiseen liittyviä ominaispiirteitä, jotka ovat *universaaleja* (jokaisella ihmisellä on ne), *erottelevia* (ne ovat tarpeeksi erilaisia eri ihmisillä), *pysyviä* (ne säilyvät tarpeeksi muuttumattomina ajan kuluessa) ja *kerättäviä* (ne voidaan mitata automaattisesti ja määrällisesti). Tunnetuin esimerkki biometrisestä todennuskeinosta ovat sormenjäljet, mutta niinkin erilaiset piirteet kuin kämmenenjälki, kasvojen lämpökuva, verkkokalvon verisuonet, ääni

ja näppäilyrytmi ovat esimerkkejä ehdotetuista keinoista. (Jain, Ross & Prabhakar, 2004)

Biometristen menetelmien etu tietoon ja esineisiin perustuviin menetelmiin nähden on se, että henkilö kantaa biometristä tunnistetta aina mukaan. Lisäksi joidenkin biometristen tunnisteiden kopioiminen ja matkiminen on hankalaa. Biometristen menetelmien haittoja ovat esimerkiksi se, että kerran kopioitua biometristä tunnistetta ei voi vaihtaa, että jotkin biometriset tunnisteet voidaan kopioida henkilön sitä tietämättä (esimerkiksi sormenjälki voidaan kopioida henkilön koskettamasta esineestä), että luettujen tietojen tulkitseminen on verrattain epätäsmällistä (mistä seuraa merkittävä väärän hyväksymisen ja hylkäämisen mahdollisuus) sekä biometristen tunnistuslaitteiden tuomat kustannukset.

3.4. Vahva todentaminen

Vahva todentaminen (strong authentication, multi-factor authentication) tarkoittaa henkilöllisyyden todentamista siten, että todennustapahtumassa käytetään useampaa kuin yhtä menetelmää ja kaikki käytetyt menetelmät eivät kuulu samaan mainituista kolmesta ryhmästä. Esimerkiksi todentautuminen Väestörekisterikeskuksen tuottaman kansalaisvarmenteen sisältävällä sirullisella henkilökortilla (vastaa entistä HST-korttia) edellyttää sekä kortin asettamista lukulaitteeseen että tunnusluvun syöttämistä. Samoin verkkopankkitunnuksilla todentaminen edellyttää sekä salasanaa että avainlukulistalta poimittua avainlukua.

Kahden erityyppisen menetelmän vaatiminen vaikeuttaa hyökkäyksiä todennusjärjestelmää vastaan, sillä hyökkääjän pitää käytännössä tehdä kaksi erilaista hyökkäystä. Ei siis riitä, että hän vakoilee salasanan vaan hänen pitää myös varastaa kortti, tai toisin päin. Tämä pidentää hyökkäykseen kuluvaa aikaa, suurentaa riskiä kiinnijäämiseen ja kasvattaa psykologista kynnystä tehdä hyökkäys pelkästään tilaisuuden tarjoutuessa (esimerkiksi jos potentiaalinen hyökkääjä näkee salasanan vahingossa tai löytää kortin lattialta).

3.5. Kertakirjautumisen, välittäjäpalveluiden ja todentamisen suhde

Kertakirjautuminen (single sign-on) tarkoittaa menetelmää, jonka avulla käyttäjä voi asioissa peräkkäin useissa sovelluksista tai palveluissa ilman että hänen tarvitsee todentautua jokaiseen niistä erikseen. Tieto hänen henkilöllisyydestään välitetään sovelluksesta tai palvelusta toiseen hänen liikkuessaan niiden välillä. Kertakirjautumismenetelmiä ovat esimerkiksi yhdysvaltalaisissa korkeakouluissa web-sovelluksia yhdistävä CAS, web-sovellusten välisen kertakirjautumisen teollisuusstandardiksi muotoutunut SAML ja paikallisverkoissa tietokoneelle kirjautumisen ja esimerkiksi levyjaon tai web-sovelluksen yhdistävä Kerberos. Samaan luokkaan voidaan lukea myös OpenID, joka todistaa käyttä-

jän "hallitsevan" tiettyä verkko-osoitetta, ja Facebook Connect, joka todistaa käyttäjän olevan tietty Facebook-käyttäjä.

Olemassa on myös välittäjäpalveluita, jotka "piilottavat" useita todentamismenetelmiä yhtenäisen rajapinnan taakse. Ideana on abstrahoida todentamismenetelmien erot siten, että web-sovelluksen kehittäjän näkökulmasta ei ole eroa sillä, kirjautuuko käyttäjä avainluvulla vai vaikkapa älykortilla. Teknisesti välittäjäpalvelun toteutus ei välttämättä eroa kertakirjautumispalvelusta eikä ole estettä sille, ettei välittäjäpalvelu voisi olla myös kertakirjautumispalvelu. Suomen julkishallinnossa on käytössä useita välittäjäpalveluja, joista tärkeimpiä ovat Kansaneläkelaitoksen, työ- ja elinkeinoministeriön ja verohallinnon yhteinen Tunnistus.fi ja eräiden kuntien ja ministeriöiden perustama VETUMA (Valtiontalouden tarkastusvirasto, 2008).

Kertakirjautuminen voi helpottaa asiointia useista web-sovelluksista koostuvassa ympäristössä ja välittäjäpalvelu voi helpottaa sovellusten kehittämistä, mutta on tärkeää muistaa, että kumpikaan näistä ei ole itsessään todentamismenetelmä, vaan ainoastaan mekanismi jolla loppukäyttäjän tunnistanut ja todentanut sovellus välittää tiedon onnistuneesta todentamisesta sitä tarvitsevalle sovellukselle. Tätä eroa ei aina tehdä ja kertakirjautuminen saattaa olla yksi sovellukseen tarjolla olevista "todentamismoduuleista" (authentication plugin) tai vastaavista. Syy tähän on se, että sovelluksen tai palvelun teknisen toteutuksen näkökulmasta kummatkin ovat komponentteja, joista saadaan ulos jollain tapaa todennettu käyttäjän tunnistava tieto.

4. Koulujen tietojärjestelmistä

Peruskoulun ja toisen asteen oppilaitoksen tietojärjestelmän ydin on kouluhallinto-ohjelmisto, jonka tärkein tehtävä on pitää kirjaa oppilaista ja heidän arvosanoistaan sekä tuottaa erilaisia tulosteita, tilastoja ja tiedonsiirtoja. Suomessa markkinoilla on kolme tuotetta: peruskouluissa ja toisessa asteessa käytettävät StarSoft Primus ja Nextime Procapita sekä ammattikouluissa (ja ammattikorkeakouluissa) käytettävä Logica Winha. Erääseen kouluhyvinvointia koskevaan kyselyyn vastanneissa kouluissa oli käytössä Primus 81 %:ssa, Procapita 12 %:ssa ja 5 % ei käyttänyt mitään kouluhallinto-ohjelmaa (Kuusela, Peltonen & Rimpelä, 2007). Kaikki tuotteet ovat client-server -pohjaisia työpöytäsovelluksia.

Verkon ylitse tapahtuva itsepalveluasiointi on yleistynyt myös kouluissa, jolloin kouluhallinto-ohjelmiston palveluja laajennetaan web-liittymän tarjoavalla laajennustuotteella tai sen rinnalle otetaan käyttöön erillisiä web-sovelluksia. Kouluissa käytettäviä web-sovelluksia on listattu taulukossa 1. Kouluissa käytettävät ohjelmat eivät luonnollisesti rajoitu kouluhallinto-ohjelmaan ja web-sovelluksiin vaan niiden lisäksi käytössä on sekä kouluihin suunnattua oh-

jelmistoa kuten opetusohjelmia ja opetusjärjestelyjen suunnitteluohjelmia sekä yleiskäyttöisempää ohjelmistoa kuten toimistopaketteja, työryhmäsovelluksia ja tilanvarausjärjestelmiä.

Tehtävä	**Sovellukset**
Arvosanojen kerääminen	- Wilma (Primuksen yhteydessä) - ProcapitaWeb (Procapitan yhteydessä) - WinhaWiivi (Winhan yhteydessä)
Kurssi-ilmoittautuminen	- Wilma (Primuksen yhteydessä) - WebValinta (Procapitan yhteydessä) - WinhaWille (Winhan yhteydessä)
Poissaolojen seuranta	- Wilma (Primuksen yhteydessä) - ProcapitaWeb (Procapitan yhteydessä) - WinhaWiivi (Winhan yhteydessä) - Helmi Reissuvihko
Kodin ja koulun yhteydenpito	- Wilma (Primuksen yhteydessä) - Helmi Reissuvihko - MUKAVA Kivahko
Verkko-opetus ja monimuoto-opetus	- WSOY Opit - Fronter - Moodle - Peda.net Veräjä, Oppimappi ja Verkkolehti
Lisäksi mm: - Peda.net OPSpro (opetussuunnitelmien tekeminen ja ylläpitäminen) - Erilaisia webmail-sovelluksia, sisällönhallintajärjestelmiä, keskustelufoorumeja, blogialustoja ja yhteisösovelluksia	

Taulukko 1. Eräitä kouluissa käytössä olevia web-sovelluksia.

4.1. Käyttäjät

Kuvassa 1 on kuvattuna koulun tietojärjestelmien käyttöön osallistuvia henkilöitä sekä heidän toimialueitaan.

Koulun tietojärjestelmiä hyökkäävänä henkilönä nähdään tavallisesti oppilas, sillä tietojärjestelmiä vastaan tapahtuvat hyökkäykset on helppo kuvitella jatkeena samoille nuorison koulua vastaan tuntemille antipatioille, jotka ilmenevät myös ilkivaltana ja kiusantekona, sekä keskinäiselle välienselvittelylle, joka ilmenee myös väkivallantekoina ja muuna kiusaamisena. Opettajilla ja koulun henkilökunnalla on työnsä puolesta laajemmat oikeudet tietojärjestelmien käyttöön, joten heidän tapauksessaan mahdollinen väärinkäyttö olisi enemmänkin aseman kuin tietojärjestelmän väärinkäyttöä. Yleensäkin oppilaiden, henkilökunnan ja täysin ulkopuolisten henkilöiden halua koulun tietojärjestelmää kohti hyökkäämiseen rajoittaa taloudellisen hyödyn puute.

Kuva 1. Tyypillisiä sellaisia peruskoulujen sidosryhmiä, joilla on käyttöoikeuksia koulun tietojärjestelmiin. (Stakes, 2008, Opetushallitus, 2007).

Suomalainen yhteiskuntaa on perinteisesti pidetty suhteellisen vahvana ns. identiteettivarkauksia vastaan. Suomessa henkilön tilinumeron tunteminen mahdollistaa ainoastaan rahan lähettämisen hänelle, kun taas Yhdysvalloissa yritykset voivat tilinumerolla veloittaa rahaa henkilöltä ilman erikseen pankille annettavaa suostumusta. Samoin henkilötunnuksen ei periaatteessa pitäisi riittää luoton hankkimiseen (Tietosuojalautakunta, 2008). Käytännössä on Suomessa kuitenkin paljastanut tapauksia, joissa pikavippiyhtiöiden kautta on varastetuilla henkilötiedoilla onnistuttu hankkimaan rahaa (Saarinen, 2008). Lisäksi henkilötiedot voivat olla kauppatavaraa myös sellaisenaan, jos tiedot ovat julkisuuden henkilöstä (BBC, 2008).

4.2. Käyttötilanteet

Käyttötilanne (käyttökonteksti, context of use) tarkoittaa sitä ympäristöä, jossa tuotteen todelliset loppukäyttäjät tulevat todellisessa elämässään sitä käyttämään. Joidenkin käytettävyyden määritelmien, esimerkiksi standardissa ISO 9241 (ISO, 1997) annetun, mukaan käytettävyys on käyttötilanteen funktio. Toisin sanoen, näiden määritelmien mukaan on mahdotonta puhua tuotteen käytettävyydestä yhtenä suureena, ainoastaan käytettävyydestä tietyissä käyttötilanteissa.

Standardi ISO 9241 määrittelee tuotteen käyttötilanteen koostuvan "käyttäjistä, tehtävistä ja välineistä (laitteistoista, ohjelmistoista ja materiaaleista) sekä fyysisestä ja sosiaalisesta ympäristöstä jossa tuotetta käytetään". Asiantuntijaarviointien apuvälineiksi on kehitetty varsin täsmällisiä jäsennyksiä (esimerkik-

si Maguire, 2001), mutta nämäkin lopulta perustuvat käyttäjien, tehtävien ja ympäristön kolmiyhteyteen (mikäli välineet lasketaan osaksi fyysistä ympäristöä).

Tietoturvan kannalta ovat käyttöympäristössä erityisen olennaisia kysymykset siitä, miten tietokoneita ylläpidetään, onko käyttötilanteessa läsnä muita henkilöitä ja keillä muilla henkilöillä pääsyä tilaan muina aikoina. Koulun käyttötilanteissa potentiaalinen uhri ja hyökkääjä ovat keskenään läsnä useissa tilanteissa. Oppilaat tekevät kurssivalintojaan ATK-luokassa toisten oppilaiden seurassa ja opettajat kirjaavat poissaoloja luokassa oppilaiden läsnä ollessa (tosin tietysti ainoastaan niissä kouluissa, jossa jokaisessa luokassa on tietokone). Samoin oppilas ja hänen huoltajansa asuvat tyypillisesti samassa taloudessa.

Tämä hyökkääjän ja uhrin läheisyys poikkeaa huomattavasti esimerkiksi verkkopankista. Verkkopankin salasanoja havitteleva henkilö sijaitsee usein toisessa maanosassa ja joutuu turvautumaan selain- tai sähköpostiohjelman haavoittuvuuksiin tai huijausviesteihin. Uhan torjumiseksi riittää ohjelmien tunnollinen päivittäminen ja terve skeptisyys. Kouluympäristössä on sen sijaan mahdollista kurkkia olan yli tai pöydän takaa ja samassa taloudessa pystyy puolestaan helposti asentamaan haittaohjelmia tai "lainaamaan" avainlukulistaa tai muuta tietoturvavälinettä. Huoltaja ja oppilas eivät aluksi vaikuta kovin todennäköiseltä uhrilta ja hyökkääjältä, mutta itse asiassa oppilas pystyy käyttämään huoltajansa tunnuksia esimerkiksi poissaolojensa peittelyyn, sillä poissaolojen seuranta on tärkeä koulun tietojärjestelmien käyttökohde (Kuusela, Peltonen & Rimpelä, 2007).

Läheisyys aiheuttaa ongelmia myös salasanakäytännöille. Salasanan vaihtaminen kerran kuukaudessa saattaa olla järkevä käytänne vahinkojen rajoittamiseksi, mikäli hyökkääjä voi saada salasanan tietoonsa vain kovan työn kautta tai vahingossa. Sen sijaan kouluympäristössä voi hyökkääjä yksinkertaisesti toistaa hyökkäyksen kerran kuukaudessa.

5. Menetelmien vertailuperusteita

Taulukossa 2 on listattu tärkeimpiä menetelmiä, jotka ovat koulujen käytettävissä todentamista varten. Muita julkisen hallinnon todennuspalveluja käsittelee kattavasti (Valtiontalouden tarkastusvirasto, 2008). Tässä tutkielmassa ei ryhdytä tekemään yksityiskohtaista vertailua eri todentamismenetelmien kesken, vaan tyydytään esittelemään erilaisia menetelmiä sekä vertailuperusteita.

Menetelmä	Myöntäjä	Välittäjä	Muut osalliset
Salasana	Koulu	-	-
Avainlukulista	Koulu	-	-
	Pankki (TUPAS)	-	-
		VETUMA	-
Avainluku tekstiviestillä	Koulu	-	Tekstiviestivälittäjä
Avainluku laskurilla	Koulu	-	Laskureita valmistavat lukuisat yritykset, tunnetuimpana ehkä RSA (SecurID)
Älykortti	Koulu	-	Älykortin valmistaja
	Poliisi / Väestörekisterikeskus (sirullinen henkilökortti)	-	-
		VETUMA	-
		Eräät pankit	-
Mobiilivarmenne	Koulu	-	Kansalaisvarmenne tai yritysvarmenne on mahdollista saada Soneran ja Elisan SIM-kortteihin
	Poliisi / Väestörekisterikeskus	-	
Radiotunnistus	Koulu	-	Lukijan ja tunnisteen valmistaja
Sormenjälki tms. biometria	Koulu	-	Lukijoita valmistaa mm. Microsoft
Tietokoneelle tallennettu salausavain	Koulu	-	-

Taulukko 2. Tärkeimpiä koulujen käytettävissä olevia todentamismenetelmiä.

5.1. Kustannukset

Tietoturvaloukkauksista aiheutuu kustannuksia, mutta niin aiheutuu myös tietoturvatoimista. Tietoturvatoimen taloudellista järkevyyttä voidaan arvioida esimerkiksi ALE-menetelmällä (Annualized Loss Expectancy, vuosittainen vahingon odotus). Siinä tietoturvariskin nykyinen keskimääräinen vuosittainen toteutumistiheys kerrotaan riskin toteutuessaan aiheuttamilla keskimääräisillä kustannuksilla. Samoin tehdään tietoturvatoimen toteuttamisen jälkeen odote-

tulle esiintymistiheydelle. Tietoturvatoimi on järkevä, jos lukujen erotus on suurempi kuin tietoturvatoimen vuosittainen kustannus. (CERT, 2006)

ALE-menetelmä tarjoaa selkeän mallin riskeistä keskustelemiseen, mutta sen käytännön soveltamisesta tekee pulmallista yhtälön tekijöiden arviointi. Harvinaisten tai vielä toteutumattomien riskien toteutumistiheyden arviointi, kuten myös riskin toteutumisen todellisten kustannusten määrittäminen, on vaikeaa (Schneier, 2008). Vielä vaikeampaa se on liiketalouden ulkopuolella. Koska kouluympäristössä suojeltaville tiedoille on vaikea asettaa rahallista hintaa, on sijoittamista tietoturvaan vaikea perustella. Koulun tehtävä on järjestää opetusta ja tietoturvaan käytetty raha on eräässä mielessä poissa tästä tehtävästä. Toisaalta koululla on velvollisuus suojella käsittelemiään henkilötietoja ja järjestää muutenkin turvallinen oppimisympäristö, joten joitain tietoturvatoimia voidaan pitää välttämättöminä.

Tietoturvatoimesta aiheutuu erityyppisiä kustannuksia. Ensimmäinen on tietoturvatoimen tekninen toteuttaminen tai integrointi (mikäli valmis kirjasto, komponentti tai ohjelmisto on jo olemassa), ellei tietojärjestelmä tue sitä jo hankintahetkellä. Seuraavia ovat aloituskustannukset, jotka saattavat olla esimerkiksi tietokonekohtaisia (kuten älykortinlukija tai sormenjälkilukija) tai henkilökohtaisia (kuten sirullisen henkilökortin hankkimismaksu tai salasanalaskurin hinta). Jos todentaminen tapahtuu kolmannen osapuolen tietojärjestelmän kautta, saattaa palvelulla olla vielä kuukausimaksu sekä todentamistapauskohtainen käyttömaksu. Myös henkilökunnan tai muiden käyttäjien koulutuksesta ja tukitapahtumista saattaa kertyä kustannuksia.

Taulukossa 3 on kuvattu eräiden todennusmenetelmien aloitusmaksu, kuukausimaksu ja käyttömaksu. Taulukossa nähdään paitsi että kustannukset vaihtelevat, myös se että kustannusten jakautuminen eri maksuihin vaihtelee.

Taulukossa 4 verrataan verkkopankkitunnistuksen ja sirullisen henkilökortin kokonaiskustannuksia erilaisilla todennustapahtumien määrillä, kun oletetaan että halutaan todentaa vahvasti sata henkilöä vuoden ajan. Verkkopankkitunnistuksessa arvioidaan edullisinta, vaikkakin epärealistista tapausta, jossa kaikki sata henkilöä ovat saman pankin asiakkaita. Henkilökortissa arvioidaan kalleinta, mutta taaskin epärealistista tapausta, jossa jokainen henkilö tarvitsee kortinlukijan. Vaikka henkilökortin aloituskustannukset ovatkin suuria, muuttuu se taloudellisemmaksi vaihtoehdoksi, jos kirjautumisia on yli 128 henkilöä kohden vuodessa. Menetelmiä ei siis voida asettaa järjestykseen kustannuksien mukaan tuntematta käyttömääriä ja käyttäjien ja asiointipäätteiden lukumääriä.

Menetelmä	Aloitusmaksu	Kuukausimaksu	Käyttömaksu
Salasana	-	-	-
Verkkopankkitunnistus (TUPAS), suoraan pankkien järjestelmästä (1)	n. 200 € / pankkiryhmä	n. 13 € / pankkiryhmä	n. 0,40 €
Sirullinen henkilökortti	alkaen n. 15 € / tietokone	0,40 € / henkilö (2)	-
OATH-standardin mukainen salasanalaskuri	n. 4 € / henkilö (3)	-	-
Kansalaisvarmenne kännykässä (4)	8,21 € + 13 € / henkilö (5)	1,90 € / henkilö	on (6)
Avainluku tekstiviestillä	-	-	alkaen 0,08 €
Koulun tulostama avainlukulista	-	-	alle 0,01 €
Sormenjälki	alkaen n. 35 € / tietokone (7)	-	-

Taulukko 3. Eräiden todentamismenetelmien kustannukset. Tiedot koottu lokaja joulukuussa 2008. Todentamismenetelmän käyttöönotosta ja käytöstä aiheutuvia henkilöstökuluja ei ole otettu huomioon, kuten ei myöskään tietojärjestelmien integrointikuluja. (1) Tiedot haettu Osuuspankin ja Säästöpankkien verkkosivuilta 17.10.2008, hinnat sisältävät ALV:n. Pankkiryhmiä on kaikkiaan seitsemän. (2) Laskennallinen arvo. Henkilökortin hankkiminen maksaa 40 € ja se on voimassa viisi vuotta. (3) Entrust IdentityGuard Mini Token OE. (4) Sonera Varmennekortti. Tiedot haettu Soneran verkkosivuilta 7.12.2008. Myös Elisa tarjoaa vastaavan palvelun. (5) Varmennekortin toimitusmaksu Soneralta ja rekisteröintimaksu poliisilta. (6) Tunnistuspalvelun hinnat määräytyvät asiakaskohtaisesti. (7) Microsoft Fingerprint Reader.

Tapahtumia vuodessa / henkilö	Verkkopankkitunnistus	Sirullinen henkilökortti
10	200+12*13+100*10*0,40 = 756	100*40+100*15 = 5500
100	200+12*13+100*100*0,40 = 4356	100*40+100*15 = 5500
1000	200+12*13+100*1000*0,40 = 40356	100*40+100*15 = 5500

Taulukko 4. Verkkopankkitunnistuksen ja sirullisen henkilökortin kokonaiskustannusten vertailua, kun halutaan todentaa vahvasti 100 henkilöä vuoden ajan.

Perusopetus on Suomessa maksutonta, mukaan lukien välttämättömät opetusvälineet. Tämä tarkoittaa, että oppilaita tai huoltajia ei voida vaatia hankkimaan laitteita tai muita välineitä koulun tietojärjestelmien turvallista käyttöä varten. Tietoturvatoimet tulee suunnitella niin, että riittävä turvallisuus saavutetaan ilman ylimääräisiä laitteita tai välineitä, tai että kyseiset laitteet tai välineet tarjotaan koulun tai muun tahon puolesta.

5.2. Käytettävyys

Osuuspankin op.fi-palvelun asiakas vierailee verkkopankissa keskimäärin kaksi kertaa viikossa (Juote, 2008). Sen sijaan opettaja saattaa joutua kirjautumaan kouluhallintojärjestelmään 5-10 kertaa päivässä. Jos yläasteen opettaja syöttää poissaolot tietojärjestelmään joka oppitunnin alussa ja vahvemman tunnistuksen takia prosessi kestää puoli minuuttia kauemmin, menettää oppilas laskennallisesti kahden päivän verran opetusta lukuvuodessa (laskelma 1). Tällainen laskelma on luonnollisesti hyvin karkea keino arvioida vaikutuksia (ei ole varmaa, käytettiinkö kyseinen puoli minuuttia aiemminkaan opetukseen, tai vaikka käytettiinkin, tapahtuiko silloin oppimista), mutta vastaavia laskelmilla on perusteltu tietojärjestelmien käytettävyyden parantamiseen sijoittamista talouselämässä.

Kaikkia tietojärjestelmän toimintoja ei tietenkään käytetä näin usein. Opettaja kirjaa poissaoloja päivittäin, mutta kurssinumerot kerran kurssin päätteeksi. Samoin perusopetuksen oppilaaksi ilmoittaminen tapahtuu vain kerran henkilöä kohden. Vaikka näissä toiminnoissa ei käytön tehokkuudella ole yhtä paljon väliä, on opittavuus puolestaan tärkeää, sillä käyttöön ei ehdi syntyä rutiinia. Tämä pätee myös sovelluksen käyttämään todentamismenetelmään.

Yläasteella on opiskeltava keskimäärin 30 vuosiviikkotuntia luokka-asteella. Vuosiviikkotunti tarkoittaa 38 opetustuntia. Opetustunnin pituus on 45 minuuttia.

$30 * 38 * 0{,}50 / 45 \approx 12{,}7$

Menetys lukuvuodessa on kaksi kuuden opetustunnin päivää.

Laskelma 1. Laskennallinen yläasteen koululaisen menettämä opetuksen määrä, kun oppitunnista puoli minuuttia käytetään muuhun kuin opetukseen.

5.3. Myöntäjät ja poikkihallinnollinen asiointi

Kouluissa käytettäväksi soveltuville todentautumismenetelmillä on eri myöntäjiä: viranomainen, pankki ja koulu. Myöntäjällä eli provisioijalla tarkoitetaan tahoa, joka luovuttaa todentamisvälineen loppukäyttäjälle (salasanan, älykortin tms.), tyypillisesti varmistettuaan aluksi loppukäyttäjän henkilöllisyyden eisähköisesti. Viranomaisten ja pankin myöntämillä välineillä on etunaan se, että henkilöllisyys voidaan todentaa sähköisesti ennen kuin henkilö on koskaan vie-

raillut koulussa. Toisaalta koulun myöntämät välineet ovat tasapuolisempia loppukäyttäjiä kohtaan: kukaan ei joudu maksamaan viranomaismaksua tai solmimaan pankkisuhdetta saadakseen koulun palvelun käyttöönsä.

Valtion IT-strategiassa on asetettu tavoitteeksi poikkihallinnollinen toiminta, joka julkisten sähköisten palvelujen osalta tarkoittaa esimerkiksi pyrkimystä poikkihallinnollisten sähköisten asiointipalvelukokonaisuuksien kehittämiseen. Näissä käyttäjän tulisi pystyä asioimaan useiden julkisen hallinnon tahojen kanssa yhdellä todentautumisella. Poikkihallinnollisen asioinnin odotetaan paitsi helpottavan asiointia, myös vähentävän kustannuksia. Poikkihallinnollisen asioinnin saattaa estää todentautumismenetelmän tekninen toteutus, sopimusehdot tai luottamuksen puute toisia tahoja kohtaan. Valtiontalouden tarkastusvirasto (2008) kritisoi pankkien tunnistuspalveluja ja VETUMA-palvelua puutteista poikkihallinnollisen asioinnin suhteen ja mainitsi Tunnistus.fi-palvelun hyvänä esimerkkinä poikkihallinnollisen asioinnin mahdollistavasta palvelusta.

Loppukäyttäjän kannalta on kätevää, jos samalla todentamismenetelmällä voi todentautua useisiin eri palveluihin. Tällä on kuitenkin myös tietoturvavaikutuksia, sillä hyökkääjä saa samalla hyökkäyksellä pääsyn useisiin palveluihin. Esimerkiksi mikäli sirullisesta henkilökortista olisi tullut suunnitellun kaltainen "yleisavain" sähköiseen asiointiin, olisi sekä henkilökortin että tunnusluvun haltuunsa saaneella hyökkääjällä pääsy lähes kaikkeen uhrinsa viranomaisasiointiin, pankkipalveluihin ja yksityiseen viestintään. Todentamismenetelmien hajanaisuus voi siis osaltaan olla positiivinen piirre.

5.4. Avoimuus ja kilpailu

Todentamismenetelmät eroavat toisistaan myös avoimuuden suhteen. Avoimimpia ovat *avoimen ekosysteemin* ratkaisut, joissa useat toimijat tarjoavat keskenään yhteensopivia, samaan avoimeen standardiin perustuvia todentamisvälineitä. Toisin sanoen, järjestelmäntoimittajan toteutettua todentamismenetelmän tuen, pystyy koulu tai käyttäjä valitsemaan parhaiten tarkoitukseensa soveltuvan välineen. Näiden vastakohta ovat *suljetut* ratkaisut, joissa järjestelmäntoimittaja joutuu integroimaan todentautumisvälineiden valmistajan tarjoaman kirjaston tai solmimaan sopimuksen valmistajan kanssa saadakseen toteuttamisen vaatimat tiedot käyttöönsä. Tämä rajaa suljettujen ratkaisujen tuen vain sellaisiin ympäristöihin, joita valmistajan kirjasto tukee, tai sellaisiin järjestelmiin, joihin järjestelmäntoimittaja on valmis kirjoittamaan tuen tyhjästä alkaen.

Arjen tietoyhteiskunnan neuvottelukunnan sähköisen tunnistamisen kehittämisryhmän valmistelemat kansalliset linjaukset korostavat avoimia markkinoita ja kilpailua (Sähköisen tunnistamisen kehittämisryhmä, 2008). Valtiontalouden tarkastusvirasto suosittaa, että jatkossa julkisen hallinnon toimijat keskittyisivät hyödyntämään ja sertifioimaan vahvan todentamisen palveluja nii-

den tuottamisen sijaan (mm. Väestörekisterikeskuksen ylläpitämä HST-kortti ja sirullinen henkilökortti) (Valtiontalouden tarkastusvirasto, 2008).

5.5. Eettiset kysymykset

Todentamisen eettiset kysymykset liittyvät erityisesti biometriikkaan ja radiotunnistamiseen. Web-sovellusta varten kerättyä biometriikkoja sisältävää rekisteriä saatetaan käyttää myös muihin tarkoituksiin. Sama voi tapahtua myös web-sovellusta varten käyttöön luovutetulle radiotunnisteelle.

Monimutkaiset todentamisratkaisut saattavat lannistaa käyttäjiä, joiden tietotekninen osaaminen on heikkoa ja siten asettaa heidät eriarvoiseen asemaan heidän oman tai heidän lastensa koulunkäynnin seuraamisessa. Pyrkimys tietoturvan parantamiseen saattaa siis vahingossa syventää tietotekniikkataitojen osaamiskuilua (digital divide). Toisaalta käyttäjätunnus ja salasana eivät suinkaan ole kaikkein yksinkertaisin todentamismenetelmä, vaan esimerkiksi älykorttien käyttöönottaminen saattaisi jopa helpottaa sähköistä asiointia.

6. Vaihtoehtoisia ratkaisuja ja niiden ongelmia

Luvussa 2 esitellyt tunnistaminen, todentaminen ja valtuuttaminen eivät ole ainoita tapoja toteuttaa pääsynhallinta. Näiden lisäksi on olemassa esimerkiksi sijaintipohjainen pääsynhallinta, jota hyvin perinteisessä merkityksessä edustaa tietojen laittaminen ilmoitustaululle tilaan, johon on rajattu pääsy tai jota valvotaan. Internetissä tätä vastaa IP-pohjainen pääsynhallinta, jolla pystytään esimerkiksi rajaamaan tietyt tiedot näkyville vain oppilaitoksen tietokoneilta.

Normaali tapa toteuttaa pääsynhallinta web-sovelluksissa on, että käyttäjä tunnistetaan, todennetaan ja valtuutetaan istunnon aluksi ja valtuutus säilyy samana kunnes käyttäjä lopettaa istunnon tai sovellus suorittaa istunnon aikakatkaisun. Tämän sijasta voidaan käyttää useampitasoista todentamista, jossa ennen riskialttiimpien toimintojen sallimista edellytetään toista todentautumista eri menetelmällä eli vahvaa todentamista käytetään vain silloin, kun se on tarpeen. Verkkopankissa tämä usein tarkoittaa sitä, että tilin saldo näytetään salasanalla todentautumisen jälkeen, mutta päästäkseen näkemään tilitapahtumat tai tekemään tilisiirron joutuu syöttämään myös avainluvun.

Kumpaakin edellistä tavoista kutsutaan *käyttäjän todentamiseksi* (authenticating the user). Hienovaraisempi tapa toteuttaa pääsynhallinta on *toimenpiteen todentaminen* (authenticating the transaction). Tällöin päätös valtuutuksesta tehdään vasta juuri ennen toimenpiteen suorittamista ja kaikki mahdollinen tieto, mukaan lukien toimenpiteen yksityiskohdat, otetaan huomioon. Verkkopankissa tämä voisi tarkoittaa esimerkiksi sitä, että tilisiirto joka muistuttaa käyttäjän aikaisemmin tekemiä hyväksytään pelkällä salasanalla, tilisiirto tuntematomalle tilille edellyttää avainlukua ja tilisiirto Nigeriaan toteutetaan vasta, kun pankki on tehnyt varmistussoiton käyttäjälle.

Luvussa 2 viitattiin tietoturvatoimenpiteiden luokitukseen: estäminen, havaitseminen ja vaste. Tässä tutkielmassa on tarkasteltu ainoastaan vahvan todentamisen kautta tehokkaampia estämismenetelmiä. Joskus voi olla järkevämpää hyväksyä tietoturvaloukkauksia tulevan tapahtumaan ja pyrkiä tehokkaasti tunnistamaan ne ja toipumaan niistä. Hyvä esimerkki tästä on Wikipedia, joka sallii kenen tahansa tehdä muutoksia sivuille mutta tarjoaa käyttäjille työkaluja sivustolla tapahtuvien muutosten valvontaan ja säilyttää sivujen muutoshistorian, jotta töhrityt sivut on helppo palauttaa takaisin puhtaaseen tilaan. Koulujen tapauksessa ongelmana tosin on, että suojeltavana on erityisesti oppilaiden yksityisyys ja henkilötietojen paljastumisesta on vaikea toipua, sillä hyökkääjä on saattanut vaikkapa julkaista tiedot toisella sivustolla.

Lukuisat erilaiset käyttäjäryhmät ja tiheydeltään ja kriittisyydeltään vaihtelevat tehtävät viittaisivat yksityiskohtaisemman tietoturvahallinnon ja -suunnittelun tarpeeseen. Tällaista suunnittelua tehtäessä on ongelmana empiirisen tiedon vähyys. Otetaan esimerkiksi tavallinen osa salasanakäytäntöä, vaatimus salasanan vaihtamisesta 90 päivän välein. Perusteluksi esitetään usein että tämä pienentää keskimääräisen hyökkäysikkunan, eli ajan jona hyökkääjä pääsee kirjautumaan järjestelmään, 45 päivään. Mutta miksi juuri 45 päivän keskimääräinen hyökkäysikkuna on hyväksyttävä? Tai toisesta näkökulmasta, miksi juuri 90 päivän vaihtovälistä kertyvä haitta on hyväksyttävää? Samoin, miksi oletetaan, ettei hyökkääjä vain hanki uuttakin salasanaa tietoonsa samalla tavoin kuin vanhan? On kyseenalaista, onko siirtymällä yksinkertaisesta ja kansanviisauksiin perustuvasta tietoturvasuunnitelmasta yksityiskohtaiseen mutta yhä kansanviisauksiin perustuvaan suunnitelmaan saavutettavissa mitään etua.

7. Yhteenveto

Koulut ovat paikkoja, joissa tietokoneita käytetään toisten ihmisten katseen alla ja tämä tekee salasanoista lähtökohtaisesti ongelmallisia. Vahva todentaminen voisi osittain ratkaista tämän ongelman, mutta vahvojen todentamismenetelmien käyttöönottoa vaikeuttaa kouluympäristön asettamat korkeat vaatimukset kustannuksille ja tehokkuudelle. Toisaalta tietoturvatoimien tarkoituksenmukaisuutta arvioitaessa pitää niiden hyötyjä ja haittoja arvioida suhteessa suojeltavaan tietoon ja saattaa hyvinkin olla, että ongelmallisuudestaan huolimatta salasanat täyttävät nykyiset kouluympäristön tietoturvatarpeet.

Tämä tutkielma on suppea, pintapuolinen ja laadullinen kuvaus todentamiseen liittyvistä asioista suomalaisissa peruskouluissa ja toisen asteen oppilaitoksista, enkä näe siksi sopivaksi esittää suositusta vahvan todentamisen tai minkään tietyn todentamismenetelmän käyttöönoton puolesta tai vastaan. Nimeän sen sijaan kolme aihetta, joista toivoisin näkeväni tutkimuksia tulevaisuudessa. Ensimmäinen on kattava tilastollinen tarkastelu tietoturvaloukkauk-

sista ja niiden käsittelystä suomalaisessa koulumaailmassa. Toinen on mallien kehittäminen tietoturvatoimien taloudellisen järkevyyden päättelemiseksi voittoa tavoittelemattomissa organisaatioissa. Kolmas on tietoturvan kansanviisauksien entistä laajempi testaaminen käytännön kokeiden keinoin.

Viiteluettelo

BBC. (2008). Health worker sold Spears records. Retrieved December 7, 2008, from http://news.bbc.co.uk/2/hi/entertainment/7760549.stm.

CERT. (2006). The ROI of security. Retrieved October 10, 2008, from http://www.cert.org/podcast/notes/2roi.html.

Henkilötietolaki 523/1999. (1999). Noudettu 15.9.2008 Finlex-lakitietokannasta.

Jain, A. K., Ross, A., & Prabhakar, S. (2004). An introduction to biometric recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 14, 1, 4-20.

Juote, K. (2008). Op.fi - ajan hermolla. *OP-Pohjola -lehti* 3/08, pp. 18-19.

Kuusela, J., Peltonen, H., & Rimpelä, M. (2007). Poissaolot ja niiden seuranta. Hyvinvoinnin ja terveyden edistäminen peruskouluissa, pp. 105-112. Vammala: Opetushallitus.

Laki viranomaisten toiminnan julkisuudesta 621/1999. (1999). Noudettu 10.12.2008 Finlex-lakitietokannasta.

Maguire, M. (2001). Context of use within usability activities. *Int. J. Human-Computer Studies* 55, 453-483.

Morris, R., & Thompson, K. (1979). Password security: a case history. *Communications of the ACM*, 22, 11, 594-597.

O'Gorman, L. (2003). Comparing passwords, tokens, and biometrics for user authentication. Proceedings of the IEEE Volume 91, 2021-2040.

Opetushallitus. (2007). Laatua kodin ja koulun yhteistyöhön. Helsinki: Opetushallitus.

Saarinen, J. (2008). Huijarit nostivat 80 000 euroa pikavippejä väärillä henkilötiedoilla. Noudettu 7.12.2008 osoitteesta http://www.hs.fi/tulosta/1135241592714.

Schneier, B. (2003). Beyond fear: thinking sensibly about security in an uncertain world. New York: Copernicus Books.

Schneier, B. (2008). Security ROI: fact or fiction? Retrieved October 10, 2008, from http://www.csoonline.com/article/446866/Security_ROI_Fact_or_Fiction_.

Singh, S., Cabraal, A., Demosthenous, C., Astbrink, G., & Furlong, M. (2007). Password sharing: implications for security design based on social practice. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 895-904.

Stakes. (2008). Oppilashuolto ja koulun sosiaalityö. Sosiaaliportti. Noudettu 7.12.2008 osoitteesta http://www.sosiaaliportti.fi/fi-FI/lastensuojelunkasikirja/tyoprosessi/ehkaiseva_lastensuojelu/toimintamuotoja/oppilashuolto_ja_koulun_sosiaalityo/.

Sähköisen tunnistamisen kehittämisryhmä. (2008). Vahvan sähköisen tunnistamisen kansalliset linjaukset Suomessa. Noudettu 14.12.2008 osoitteesta http://www.arjentietoyhteiskunta.fi/files/89/Sahkoisen_tunnistamisen_kansalliset_linjaukset_080926_lopullinen.pdf.

Tari, F., Ozok, A. A., & Holden, S. H. (2006). A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. Proceedings of the Second Symposium on Usable Privacy and Security, 56-66.

Tietosuojavaltuutettu. (2001). Asiaa tietosuojasta 9/1999: kenellä on oikeus tallettaa ja käyttää henkilötietoja. Tietosuojavaltuutetun julkaisu, 21.12.1999; päivit. 1.6.2001.

Tietosuojalautakunta. (2008). Tietosuojalautakunnan päätös nro 1/22.1.2008.

Valtiontalouden tarkastusvirasto. (2008). Tunnistuspalveluiden kehittäminen ja käyttö julkisessa hallinnossa (Valtiontalouden tarkastusviraston toiminnantarkastuskertomus 161/2008). Helsinki: Edita Prima Oy.

Yan, J., Blackwell, A., Anderson, R., & Grant, A. (2000). The memorability and security of passwords - some empirical results. Retrieved December 10, 2008, from http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-500.pdf

WWW-sivujen tekninen saavutettavuus

Pasi Lampinen

Tiivistelmä.
Tutkielma käsittelee verkkosivuilla käytettäviä tekniikoita ja niiden saavutettavuutta. Webin muuttuminen staattisesta dynaamiseksi tuo mukanaan paljon etuja, mutta kaikille saavutettavan sisällön tuottaminen on entistä haastavampaa. Uusia tekniikoita käytettäessä tulisi huomioida erilaiset käyttäjäryhmät ja taata WWW-sivujen toimivuus kaikilla käyttäjillä. Tutkielmassa tarkastellaan myös erilaisia suosituksia verkkosivustojen saavutettavuudelle.

Avainsanat ja -sanonnat: Tekninen saavutettavuus, dynaaminen HTML, esteettömyyssuositukset
CR-luokat: C.2.6, C.4, H.5.2, K.4.2

1. Johdanto

Internetin kasvu on ollut huimaa. Sen luomat mahdollisuudet tiedonvälityksen ja yhdenvertaisuuden parantamisessa ovat olleet mullistavia. Internetin kautta tiedon välittäminen kaikille onnistuukin ennennäkemättömän helposti ja tehokkaasti, eikä kukaan varmasti halua jäädä paitsi tämä uuden tietoväylän mukanaan tuomista eduista. Aluksi WWW-sivut olivat kiinteitä, sisältäen tekstiä ja linkkejä. Pelkkä hypertekstin avulla tehty, staattisten dokumenttien tietoverkko ei kuitenkaan riittänyt kovin pitkäksi aikaa. Uusien tekniikoiden luomien mahdollisuuksien myötä Web-sivut ovat muuttuneet dynaamisiksi, toiminnallisuus ja interaktiivisuus ovat lisääntyneet runsaasti. WWW-sivujen ulkoasu, kiinnostavuus ja houkuttelevuus ovat kohonneet aivan uudelle tasolle.

Internetin lähtökohtana on olla yhdenvertainen, kaikille avoin tietoverkko. Internetissä toimivan World Wide Webin tuomat mahdollisuudet ihmisten ja yhteisöjen yhdistämisessä ovat olennainen osa lähes jokaisen arkipäivää. Saavutettavuus tarkoittaa sitä, että sivujen sisältö on kaikkien ulottuvissa. Käyttäjien erilaisuus pitää ottaa huomioon sivuja suunniteltaessa ja toteutettaessa ja esteet poistettava. Tietotekniikka ja Internet eivät enää ole vain pienen ryhmän etuoikeus, vaan ne ovat lähes kaikkien käytettävissä. Siksi minkäänlaisia oletuksia käyttäjistä ei tulisi sallia, vaan käyttäjien erilaisuuteen tulisi varautua.

Uudet tekniikat WWW-sivujen sisällön tuottamisessa saattavat osittain tehostaa tiedonvälitystä, mutta tuovat myös mukanaan ongelmia erilaisia käyttäjäryhmiä ajatellen. Ihmisten lähtökohdat Internetin käyttäjinä vaihtelevat hyvin paljon. Kaikilla tulisi olla mahdollisuus tiedon hankintaan Internetissä teknisestä tietämyksestä, iästä tai fyysisistä eroavaisuuksista riippumatta.

Internetin ollessa nykyään kaikkialla selailu ei välttämättä tapahdukaan enää perinteisesti vain tietokoneelta kotoa tai toimistosta. Informaation välittyminen WWW-sivujen kautta tulee onnistua niin matkapuhelimen pieneltä ruudulta kuin suuriresoluutioiselta videoprojektoriltakin. Yhteyksien nopeudet, prosessoritehot ja selaimet vaihtelevat paljon, eivätkä ne saisi nousta esteeksi tiedon välittymiselle Internetissä.

2. Saavutettavuus ja esteettömyys

2.1. Saavutettava WWW

Saavutettavuus (eli esteettömyys) tarkoittaa erilaisuuden huomioimista ja esteiden poistamista, jotta sisältö pysyy kaikille saavutettavana. Sillä pyritään luomaan kaikille samanlaiset ja yhdenvertaiset lähtökohdat esimerkiksi WWW-sivun sisällön lähestyttävyyteen. Samalla tavalla kuin rakennusta suunnitellessa otetaan huomioon liikuntarajoitteisten liikkuminen rakennuksessa, tulisi WWW-sivuilla oleva sisältö olla myös sokean ihmisen tavoitettavissa. Optimaalisella suunnittelulla sama reitti sopisi niin liikuntarajoitteisille kuin kävelevillekin, ja sama sivu suoraan niin sokeille kuin näkevillekin. Aina kuitenkaan tämä ei ole mahdollista, mutta tällöin olennaista on kuitenkin, että sisältö on kaikkien saavutettavissa jollain keinolla.

WWW-sivujen saavutettavuudella pyritään siihen, että käyttäjän kyvyillä tai rajoitteilla ei olisi merkitystä sisällön saavuttamisen kannalta. Oikein suunnitellut sivut tarjoavat kaikille mahdollisuuden hyödyntää sivuston tietoa ja toiminnallisuuksia. Sivustoja suunniteltaessa poikkeavan ulkoasun tavoittelu ja uusien tekniikoiden – välillä tarpeetonkin – käyttö ajavat usein esteettömän suunnittelun edelle. Yleisesti saavutettavuutta ei suunnittelijoiden keskuudessa pidetä edes kiinnostavana tai tärkeänä asiana, vaan se koetaan usein enemmänkin taakkana ja luovuutta rajoittavana tekijänä. [Wikipedia, 2008; Regan, 2004]

Lähes kaiken sähköistyessä ja Internetin tullessa yhä tärkeämmäksi osaksi jokaisen elämää esteettömyys nousee hyvin merkittäväksi kysymykseksi. Kuinka taata jokaiselle samanlaiset lähtökohdat Webin sisältämiin palveluihin ja sovelluksiin? WWW:n saavutettavuudesta ei tulisi missään nimessä tinkiä. Pääsy Internetiin ja mahdollisuus välttyä digitaaliselta kuilulta täytyykin olla jokaisen ihmisoikeuksia. Erilaiset käyttäjäryhmät huomioimalla taataan kaikille mahdollisuus olla osa tietoyhteiskuntaa. [Arrue et al., 2006]

2.2. Fyysiset rajoitteet

Internet ei ole enää vain vähemmistön, tietotekniikasta kiinnostuneiden yhteisön, etuoikeus. Internet on kaikkien ulottuvilla ja se on käytettävissä kaikkialla

kaiken aikaa. Käyttäjän rajoitteista tai teknisestä osaamisesta ei voi tehdä minkäänlaisia oletuksia. Esteettömällä suunnittelulla pyritään ottamaan erilaiset käyttäjäryhmät huomioon.

Usein esteetöntä WWW-suunnittelua tarkasteltaessa tarkoitetaan suunnittelua, jolla otetaan huomioon erityisryhmät, joilla on jokin fyysinen rajoite. Näkövammaisuuden eli sokeuden ja heikkonäköisyyden huomioiminen on yksi yleisimmistä esteettömyyden osa-alueista. Sokeita käyttäjäryhmiä ajatellen sivuston tulee toimia ruudunlukijoilla, joiden toimivuus riippuu yleisesti WWW-sivujen toteutuksen ja teknisten ratkaisujen esteettömästä suunnittelusta. Saavutettavuuden parantaminen heikkonäköisiä ja värisokeita ajatellen taas edellyttää ulkoasulta selkeyttä ja huomioita tiettyjen värien käytössä. Kuulovammaisille ongelmia aiheuttaa ääntä sisältävä materiaali. Tähän sisältöön tulisi tarjota tekstitykset tai tekstivastikkeet, tai sitten ääni ei saisi olla olennainen osa sisällön saavuttamisen kannalta. Kognitiivisista häiriöistä kärsiville esteettömien sivujen täytyy olla selkeät ja yhdenmukaiset. [Weakley, 2005]

2.3. Tekniset rajoitteet

WWW-sivuilla ja sivujen tuottamisessa käytetyt tekniikat ja teknologiat ovat kehittyneet Webin alkuajoista huikeasti. Multimedia on tullut olennaiseksi osaksi koristamaan ja monipuolistamaan sivustoja. Suunnittelijoiden nauttiessa uusien tekniikoiden luomista mahdollisuuksista visuaalisen sisällön luomisessa sivuille unohtuvat usein saavutettavuus ja käytettävyys. Webin tavoitteena tulisi kuitenkin olla yhteinen, avoin ympäristö, joka myös toimii kaikilla. Tekninen saavutettavuus on näin ollen noussut yhä suuremmaksi ongelmaksi. WWW-sivujen muuttuminen kiinteistä dokumenteista dynaamisiksi interaktiivisiksi sovelluksiksi vaatii entistä enemmän suunnittelua ja erilaisten käyttäjäryhmien huomioon ottoa saavutettavuuden takaamiseksi.

Saavutettava Web edellyttää WWW-sivujen toimimista laitealustasta ja Internet-yhteyden nopeudesta riippumatta. Uudet tekniikat vaativat sivujen selailuun käytettävältä laitteelta entistä enemmän prosessoritehoja. Sivujen kokojen kasvaminen ja varsinkin liikkuva kuva taas edellyttävät käytettävältä yhteydeltä suurta kaistanleveyttä. Vaikka Mooren lain mukaan prosessoritehot kaksinkertaistuisivatkin 18 kuukauden välein, ei jokaiselta käyttäjältä voi vaatia konetehoja kehityksen huipulta. Internetin yleistyminen mobiililaitteissa mahdollistaa nykyään WWW:n hyödyntämisen muualtakin kuin perinteiseltä tietokonetyöpisteeltä. Mobiililaitteiden laskentatehot eivät millään yllä tällä hetkellä pöytäkoneiden tasolle. Niiden tehot, tuki erilaisille tekniikoille ja yhteyksien nopeudet ovat hyvin vajavaiset, jotta WWW-sivujen sisältö olisi niiden avulla aina saavutettavissa. Saavutettavalla suunnittelulla pyritään mahdollistamaan WWW-sivujen käyttö kaikkialla selailuun käytettävästä laitteesta riippumatta.

Webin täytyy tulevaisuudessakin perustua laiteriippumattomuuteen, jotta sen universaali tasavertainen luonne säilyisi. Vaikka uudet tekniikat luovatkin mahdollisuuksia entistä visuaalisemmille ja monipuolisemmille WWW-sovelluksille, ei saavutettavuutta saisi kuitenkaan missään vaiheessa unohtaa. Suunnittelun ja saavutettavuuden tulisi kulkea kaiken aikaa käsi kädessä. Kaikilla toimivan sivuston ei missään nimessä tarvitse olla synonyymi tylsälle ja vanhanaikaiselle sivustolle. [Regan, 2004]

3. Esteellisiä tekniikoita

Uudet tekniikat ovat tulleet monipuolistamaan WWW-sivuja. Kokonaan staattiset dokumentit ovat osittain siirtyneet jo historiaan. Suosituimmat WWW-sovellukset ja -sivut ovatkin huomattavasti kehittyneempiä kuin perinteiset HTML-dokumentit. Web 2.0:n myötä koko Webin luonne on muuttunut toiminnallisemmaksi ja interaktiivisemmaksi. Uudet teknologiat ovat mahdollistaneet avoimen ja vapaan tiedonlevitysverkoston synnyn. Web-sivut ovat muuttuneet sovelluksiksi, jotka helpottavat elämää ja yhdistävät ihmisiä. Tekniikat tuovat mukanaan myös paljon ongelmia, ja sivut tulisikin suunnitella niin, että ne ovat myös kaikkien saavutettavissa. [Wikipedia, 2008a]

3.1. Skriptikielet

Komentosarjakielet eli skriptikielet ovat Web-sivujen dynaamisuuden takana. Niiden tulo osaksi WWW-sivuja on muuttanut paljon koko Webin luonnetta. WWW-sivun koodissa olevat komentosarjat ajetaan selaimessa, jolla saadaan paljon uusia ja nopeuttavia ominaisuuksia perinteiseen HTML:n verrattuna. Ennen staattiset sivut olivat näkyvissä yksi kerrallaan ja jokaisesta linkistä painaminen johti yhteydenottoon palvelimelle. Yhteyden ottaminen jokaisen toiminnallisen elementin painamisen yhteydessä hidasti WWW-sivujen käyttöä. Skriptikielillä saadaan WWW-sovellukset perinteisten tietokoneohjelmien kaltaisiksi ja nopeutetaan Webin toimivuutta.

Yleisimmin Web-ympäristössä käytetty skriptikieli on JavaScript. Se on alunperin Netscape Communicationsin kehittämä komentokieli, jota käytetään yleensä toiminnallisuuksien luomiseen HTML-sivuilla. Kieli mahdollistaa mm. uusien ikkunoiden aukaisun, lomakkeiden tarkistuksen ennen niiden palauttamista palvelimelle ja tyylien muuttamisen. Sitä käytetään joko suoraan HTML-koodiin upotettuna tai sisällytettynä erillisistä tiedostoista. JavaScriptin käyttö WWW-sivuilla on kasvanut voimakkaasti viime vuosien aikana. Vuonna 2007 59% sivuista sisälsi JavaScript-koodia, kun vuonna 2001 39% sivuista käytti kyseistä skriptikieltä. Dynaamisuus on siis olennainen osa Webiä tulevaisuu-

dessa ja skriptikielten käyttö vain lisääntyy entisestään. [Wikipedia, 2008b; Gibson, 2007]

Web 2.0:a pidetään seuraavan sukupolven Webinä. Toiminnallisuuden ja sovellusmaisuuden mahdollistajana on Ajax-tekniikka. Ajax on nimitys usean vanhan tekniikan yhteiselle käytölle. Sillä pystytään luomaan erilaisia toimintoja WWW-sivuille käyttämällä JavaScriptiä, XHTML:a, DOM:ia, XMLHTTPRequestia ja XML:a. Ajax mahdollistaa WWW-sivun muuttamista dynaamisesti. Käyttäjän ei tarvitse siirtyä välttämättä sivulta toiselle, jotta sivun sisältö muuttuisi. Ajaxin avulla pystytään välittämään tietoa palvelimelle halutessa vain osasta sivua, eikä koko sivua tarvitse ladata uudestaan. Tämä nopeuttaa WWW:n käyttöä ja lisää mahdollisuuksia toiminnallisuuden luomiseksi sivustoille. [Gibson, 2007]

Skriptikielet aiheuttavat hyödyllisyydestään huolimatta useita saavutettavuusongelmia. Kaikki selaimet eivät tue JavaScriptiä kunnolla tai ollenkaan. Esimerkiksi vanhoilla selaimilla skriptikielten toimivuus ei ole lainkaan taattua. Myös mobiililaitteet ja niiden käyttämät selaimet eivät aina tarjoa tukea skriptikielten avulla toteutetuille toiminnallisuuksille. Asiakkaan puolella eli WWW-sivun käyttäjän koneessa ajettavaa koodia voidaan myös pitää aina tietoturvariskinä. Varovaisimmat käyttäjät eivät välttämättä halua käyttää JavaScriptiä ollenkaan ja saattavat näin ollen sulkea JavaScriptin käytön pois selaimestaan. Tammikuussa 2008 Webin käyttäjistä 95% oli JavaScript-tuki selaimessa ja pitivät sitä päällä. Käyttäjistä 5% ei siis saavuta JavaScriptillä toteutettua toiminnallisuutta ja vuorovaikutteisuutta. [W3Schools, 2008]

Skriptikielten avulla toteutetut WWW-sovellukset eivät siis ole kaikkien saavutettavissa. Kielten käytön kuitenkin lisääntyessä ja toiminnallisuuden kasvaessa esteettömyyttä ei tulisi kuitenkaan unohtaa. Skriptikielillä ei tulisi rakentaa koko sivustoa, mikäli se ei ole tarpeellista. Koko WWW-sivuston toiminnallisuuden ja sisällön laskeminen skriptikielten varaan estää sen käytön joiltakin käyttäjäryhmiltä. Parhaimmillaan skriptikielet ovatkin yksittäisten toiminnallisuuksien lisäämisessä ja ymmärrettävyyden parantamisessa. Mikäli skriptikielet eivät ole jollain käyttäjällä käytössä, olisi sivuston toimittava aina jollain tavalla. Esimerkiksi noscript-elementillä pystytään määrittämään vaihtoehtoinen sisältö, mikäli käyttäjän selain ei skriptiä pysty suorittamaan. Sisältö pitäisi kuitenkin olla aina saavutettavissa, vaikka skriptikielet eivät toimisikaan.

3.2. Flash

Flash on Macromedian luoma kehitysympäristö, jota käytetään ensisijaisesti tuottamaan multimedia-sisältöä WWW-sivustoille. Nykysin Flashin omistaa Adobe, joka vastaa sen kehittämisestä. Flashin avulla pystytään lisäämään liikkuvaa kuvaa ja vuorovaikutteisuutta Web-sivuille. Yleensä sen avulla luodaan

HTML:llä toteutetun sivun sekaan osia, jotka sisältävät erilaisia multimediaelementtejä. Flashilla on kuitenkin mahdollista luoda kokonainen WWW-sivustokin.

Web-sivustoilla Flashia käytetään nykyään yhä useammin. Webin monipuolistuessa ja muuttuessa vuorovaikutteisemmaksi siitä on tullut olennainen osa sivustoja. Se helpottaa videon, äänen, animaatioiden ja pelien tuomista Internetiin. Flash on noussut suunnittelijoiden suosioon, koska se mahdollistaa sellaisen visuaalisen sisällön luomisen WWW-sivuille, joka perinteisellä HTML:llä ja CSS:llä oli huomattavasti vaikeampaa tai jopa mahdotonta. Liian usein Flash on silti vain huomionhakuisten mainosten ja sivustojen alkuintrojen julkaisuväline. Vuonna 2000 Jakob Nielsen oli sitä mieltä, että suurin osa Flash-materiaalista oli huonoa. Hänen mukaansa Flash tuo mukanaan paljon käytettävyysongelmia. [Nielsen, 2000]

Flashin käytössä onkin käyttäjän näkökulmasta useita ongelmia. Ensinnäkin se tarvitsee toimiakseen selaimeen asennettavan Flash Player -laajennuksen. Adoben tuodessa uusia versioita Flash Playerista käyttäjältä edellytetään aina päivitystä, jotta kaikki Flashilla tuotettu sisältö olisi saavutettavissa. Joillakin laitteilla, kuten mobiililaitteilla, tuki Flashille on usein vajavainen. Mikäli sivuston toimivuus ja sisältö on riippuvainen Flashista, se ei ole läheskään kaikille saavutettava. Vuonna 2008 Adoben mukaan 99% PC:llä Internetiä käyttävistä on asentanut Adobe Flash Playerin, eli Flash-sisältö on heidän saavutettavissaan. Kuitenkin eri versioiden toimivuus eri selaimilla ja käyttöjärjestelmillä on vaihtelevaa, mikä vähentää Flash-materiaalin saavutettavuutta. [Adobe, 2008]

Flashin avulla tuotettu multimediasisältö Web-sivulla on perinteiseen HTML-sivuun verrattuna hyvin raskas. Flashia sisältävien sivujen koot ovat suuremmat, mikä hidastaa huomattavasti WWW:n käyttöä, etenkin käyttäjillä, joilla on hidas Internet-yhteys. Flash-sivut ovat myös nostaneet Webin käyttäjien laitteiden tehovaatimuksia. Kun tekstin selaaminen onnistuu lähes millä tahansa laitteella, Flashin avulla koristeltu sivu vaikeuttaa WWW:n käyttöä vanhemmilla laitteilla huomattavasti. Flashin toki voi sulkea selaimesta pois, mutta mikäli sivuston sisältö, informaatio tai jokin oleellinen toiminto on Flashin sisällä, se ei ole tällöin kaikille saavutettava.

3.3. Video WWW-sivuilla

Videon määrä WWW-sivuilla on lisääntynyt Internet-yhteyksien nopeuksien noustua lähes kaikilla liikkuvan kuvan siirtämiseen vaadittavalle tasolle. Webissä videon avulla saadaan informaatiota nopeasti ja tehokkaasti levitettyä. Esimerkiksi Youtuben kaltaisten palveluiden välityksellä. Internet-yhteyksien ja tietokoneiden kehittyessä videon laatu Webissä paranee entisestään. Tällä hetkellä osa levitettävästä videomateriaalista, esimerkiksi elokuvatrailereista, on-

kin jo HD-laatuista. Tekniikan kehittyessä liikkuvan kuvan saavutettavuus vanhemmilla laitteilla ja hitaammilla yhteyksillä kuitenkin aiheuttaa ongelmia. Monissa tapauksissa videosta on tullut keskeinen ja ainoa osa sivun tiedonvälittäjänä, joka myös osaltaan vähentää sisällön saavutettavuutta. Sivustolta tulisikin löytyä videon lisäksi vaihtoehtoinen tapa välittää informaatio. [Reid and Snow-Weaver, 2008]

Jotta videon siirtäminen Internetissä olisi mahdollisimman nopeaa, täytyy video pakata pienempään tilaan. Pakkaamisessa pyritään siihen, että hävitetään kuvasta sellaista tietoa, jota ei silmällä voi nähdä. Videon sisältämä kuva- ja äänimateriaali pakataan erilaisten koodekkien avulla. Näistä syntyvät bittivirrat paketoidaan haluttuun säiliömuodon mukaiseen tiedostomuotoon. Kun pakattua videomateriaalia haluaa katsoa, tarvitaan kyseistä koodekkia purkamaan signaali katsottavaksi. Erilaisia paketointimuotoja eli videoformaatteja on useita, ja niiden saavutettavuus vaihtelee. Käyttäjä tarvitsee aina kyseisen videoformaatin vaatiman erillisen toisto-ohjelman ja koodekin, jotta videon katsominen selaimella onnistuu. Saavutettavan sisällön takaamiseksi video tulisikin olla tarjolla useammalla kuin yhdellä formaatilla. Yleisesti käytettyjä videoformaatteja ovat Quicktime, Windows Media ja RealVideo. Nykyään myös Flash Video on paljon käytetty formaatti, varsinkin palveluissa, joissa käyttäjät saavat itse julkaista videoitaan. Saavutettavuutta WWW-sivuihin upotetuissa videoissa haittaa myös videoformaattien eri versiot. Kun formaatin uusi versio julkaistaan, edellyttää se myös aina käyttäjältä päivityksiä, jotta kaikki kyseistä formaattia tukevat uusimmatkin videot olisivat saavutettavissa. [Sivonen et al., 2006]

Internet-yhteyksien nopeuksien kasvaessa, näyttöjen tarkkuuksien parantuessa ja tietokoneiden tehojen kasvaessa WWW:ssä voidaan levittää yhä tarkempaa videota. Videon resoluution noustessa ja kuvanlaadun parantuessa videon koko kuitenkin kasvaa nopeasti. Saavutettavuuden kannalta isojen ja tarkkojen videotiedostojen tulisikin olla saatavilla vaihtoehtona myös pienemmillä tarkkuuksilla. Kaikilla käyttäjillä, esimerkiksi mobiililaitteita käyttävillä, eivät prosessoritehot tai Internet-yhteyden nopeus riitä toistamaan korkearesoluutioisia videotiedostoja sujuvasti. Toisaalta Webissä videoiden katsomiseen käytettävä suoratoisto nopeuttaa katselua jonkin verran myös hitaammilla Internet-yhteyksillä. Suoratoisto mahdollistaa videon aloittamisen jo ennen kuin koko videotiedosto on ladattu. [Wikipedia, 2008c]

3.4. Näyttölaitteiden resoluutio

Tietotekniikan kehittyessä tietokoneiden näyttölaitteet ovat kokeneet paljon muutoksia. Näytöt ovat ohentuneet, resoluutiot kasvaneet ja värien määrä lisääntynyt. WWW:n selailu ei rajoitu kuitenkaan enää nykyään perinteisen tie-

tokoneen näytön ääreen. Matka- ja älypuhelinten resoluutiot ovat luokkaa 200x200 pikseliä, kun LCD-näyttöjen resoluutiot liikkuvat jo 1900x1200 pikselin tietämillä. Näin suuri ero näyttöjen tarkkuuksissa edellyttää WWW-sivujen suunnittelulta erityisiä huomioita, jotta sisältö pysyisi saavutettavana kaikilla näyttölaitteilla.

Vuonna 2008 enemmistö, eli 48% Web-selainten käyttäjistä, käytti 1024x768 pikselin resoluutiota. Tätä suurempaa resoluutiota käytti 38% käyttäjistä. Vuoden 2005 tammikuussa yleisin resoluutio oli jo 1024x768 pikseliä, mutta tuolloin vain 12% käytti tätä suurempaa tarkkuutta näytössään. Näyttöjen tarkkuuksien kehitys on siis selkeästi nousujohteinen. WLAN-tuen yleistyminen matkapuhelimiin ja muihin mobiililaitteisiin, sekä 3G-verkkojen laajentuminen mahdollistaa toisaalta WWW:n selailun mistä vain. Mobiililaitteiden näyttöjen resoluutiot ovat – ainakin toistaiseksi – huomattavasti perinteisiä tietokoneen näyttöjä pienempiä, mikä vaikeuttaa Webin käyttöä. Koska näyttölaitteiden resoluutioissa on niin suuria eroja, ei WWW-sivuja tulisi suunnitella vain tiettyä resoluutiota silmällä pitäen. Jotta sivusto pysyisi kaikille saavutettavana, ei minkäänlaisia oletuksia käyttäjän näyttölaitteesta saa tehdä. [W3school, 2008a]

Useasti sivujen suunnittelussa voidaan vaikuttaa siihen, miten sivusto toimii ja miten saavutettava se on erilaisilla resoluutioilla. Sivustoille graafisia elementtejä suunniteltaessa CSS:n avulla tulisi välttää kiinteitä kokoja, mikäli se on vain mahdollista. Koot tulisi aina ilmaista suhteellisesti, eikä tiettyjen pikseleiden mukaan, jotta sivuston ulkoasu toimisi mahdollisimman hyvin eri resoluutioisilla näytöillä. CSS mahdollistaa myös erilaisten tyylilomakkeiden luomisen erilaisille mediatyypeille. Mahdollisuus erilaisten ulkoasujen luomiseen esimerkiksi tietokoneen näytölle, pieninäyttöisille laitteille sekä projektoreille helpottaa erilaisten käyttäjäryhmien huomioonottoa. Kuvista ja videoistakin voidaan tarjota useita erikokoisia vaihtoehtoja eri resoluutioille. Kuvaa sisältävissä WWW-sivuissa jokaiselle resoluutiolle optimaalista versiota ei kuitenkaan ilman skaalautuvaa grafiikkaa periaatteessa voida mitenkään toteuttaa. Siksi on paljon suunnitteluvaiheesta kiinni, kuinka saada sivustosta niin saavutettava, jotta tiedonvälitys kaikille näyttölaitteesta riippumatta on kuitenkin taattu. [W3C, 2007]

4. Saavutettavuussuosituksia

World Wide Web on Internetissä toimiva kaikille avoin järjestelmä. Jokainen saa julkaista WWW:ssä lähes mitä tahansa. Minkäänlaisia pakotteita tai lakeja WWW-jukaisujen toimivuudessa ja saavutettavuudessa ei ole. Julkaisija saa itse päättää Web-sivujen toimivuudesta tai toimimattomuudesta. WWW:n laadun

ja yhdenmukaisuuden parantamiseksi on kuitenkin luotu erilaisia teknisiä suosituksia ja standardeja, joista suurimmasta osasta vastaa W3C.

4.1. W3C

WWW:n kasvaessa ja laajentuessa on Web-sivujen saavutettavuus noussut tärkeäksi asiaksi. Vuonna 1994 WWW:n kehittäjä Tim Berners-Lee perusti W3C:n eli World Wide Web Consortiumin Webin laadun varmistamiseksi. W3C on kansainvälinen yhtymä, joka kehittää ja ylläpitää Web-standardeja. Sen tavoitteena on protokollia ja suosituksia kehittämällä taata Webin kasvu myös tulevaisuudessa. W3C on antanut suosituksista erilaisia tarkistuslistoja, joiden avulla saavutettavuutta voidaan parantaa. Suositukset ovat periaatteita, joilla voidaan arvioida ja parantaa yleisellä tasolla esteettömyyttä internetympäristössä. WWW-sivujen teknisen toteutuksen yhdenmukaisuuden takaamiseksi standardit ovatkin elintärkeitä. Ilman yhteisiä standardeja WWW saattaisi lähteä nopeasti pirstaloitumaan, joka vaikeuttaisi niin Web-sivujen käyttäjiä kuin kehittäjiäkin. W3C:n ohjeet ovat vain suosituksia, joiden noudattaminen ei ole pakollista. Webin toimivuuden ja tulevaisuuden kannalta oleelliset standardit kuitenkin otetaan huomioon lähes poikkeuksetta kehittäjien keskuudessa. [Jacobs, 2008]

4.2. WAI

Webin saavutettavuusaloite (WAI) on yksi W3C:n toiminta-alueista. WAI:n pyrkimyksenä on parantaa yleisesti Web-saavutettavuutta. Sen tavoitteena on, että eri teknologiat tukevat saavutettavuutta nyt ja tulevaisuudessa. WAI kehittää ja ylläpitää ohjeistuksia niin sivujen sisällölle kuin selain- ja sivuntekoohjelmillekin. [W3C, 2008

WCAG (Web Content Accessibility Guidelines) on yksi WAI:n Webin esteettömyyttä edistävistä ohjeistoista. Sen ensimmäinen versio WCAG 1.0 ilmestyi vuonna 1999. Vuonna 2001 WCAG:in toisesta versiosta tuli ensimmäinen luonnos. Marraskuussa 2008 WCAG 2.0:n statukseksi tuli proposed recommendation (PR), joka on viimeinen vaihe ennen viralliseksi suositukseksi tuloa. WCAG sisältää ohjeistuksia WWW-sisällön saavutettavuudessa. Suosituksesta tuleva uusi versio lisää ja päivittää ensimmäisen version ohjeistuksia. [Caldwell et al., 2008; Nykänen, 2003]

WCAG määrittää suositusten avulla minkälaiset Web-sivujen tulisi olla, jotta niiden sisältö olisi mahdollisimman saavutettava. Se määrittelee esteettömän suunnittelun perusperiaatteiksi sulavan muuntautumisen eri käyttäjille, sivuston ymmärrettävyyden ja navigoinnin helppouden. WWW-sivujen sulava muuntautuminen takaa sivuston saavutettavuuden kaikille käyttäjän fyysisistä tai teknisistä rajoitteista huolimatta. [Crisholm et al., 1999]

WCAG 2.0 antaa saavutettavuuden parantamiseksi neljä yleistä periaatetta. Ensinnäkin sisällön tulee olla kaikkien hahmotettavissa. Sen mukaan kaikelle ei tekstiä sisältävälle sisällölle, kuten videolle tai äänelle, tulee olla vaihtoehto tekstin muodossa. Toinen periaate on, että käyttöliittymä ja navigointi ovat kaikkien käytettävissä helposti. Vaatimuksena on esimerkiksi, että kaiken tulisi toimia tarvittaessa näppäimistöltä ja sisällön saavuttaminen ei saisi olla aikaan sidottua. Kolmantena periaatteena on sisällön ja käyttöliittymän ymmärrettävyys. Siinä suositellaan, että teksti tehdään helposti luettavaksi ja ymmärrettäväksi. Sivuston toiminta tulisi suunnitella niin, että käyttäjä ennustaisi helposti mitä seuraavaksi tapahtuu ja välttyisi virheiltä. Viimeisenä periaatteena on sisällön kestävyys. WWW-sivujen sisällön saavutettavuuden parantamiseksi tulisi taata sivuston mahdollisimman hyvä yhteensopivuus eri selainten kanssa. Vaikka sivusto ei toimisikaan kaikilla selaimilla samalla tavalla, tulisi sisältö olla kuitenkin saavutettavissa vanhoilla, nykyisillä ja tulevilla selainohjelmilla. [Caldwell et al., 2008]

4.3. WAI-ARIA

Webin muuttuminen dynaamiseksi ja vuorovaikutteiseksi uusien tekniikoiden ja varsinkin Web 2.0:n myötä, edellyttää myös uusia suosituksia saavutettavuuden takaamiseksi. W3C:n saavutettavuusaloite (WAI) on julkaissut kokoelman ohjeita koskien dynaamisen WWW-sisällön saavutettavuuden lisäämistä. Accessible Rich Internet Applications (WAI-ARIA) sisältää ohjeita suunnittelijoille saavutettavuuden parantamiseksi Ajaxin kaltaisia tekniikoita käytettäessä. Rikkailla Web-sovelluksilla tarkoitetaan sovelluksia, jotka toimivat selaimessa ja niiden toiminta muistuttaa perinteisiä tietokoneohjelmia. Applettien ja skriptikielten käytön lisäännyttyä esteettömyyden huomioiminen on vaikeutunut. Perinteiseen staattiseen HTML:llä toteutettuun sivuun verrattuna uusilla tekniikoilla toteutettu Web-sivu voi muuttua kaiken aikaa. WAI-ARIA lisää määrittelyjä esimerkiksi sivun elementtien tilasta ja roolista, jotta verkkosivuston sisältö ja käyttöliittymä olisi kaikille saavutettavampi. ARIA sisältää ohjeita myös Web-selainten suunnittelijoille, jotta selaimet tukisivat myös mahdollisimman hyvin saavutettavuutta. [Reid and Snow-Weaver, 2008; Wikipedia, 2008]

5. Yhteenveto

Internetissä toimiva maailmanlaajuinen Web on kaiken aikaa kasvava järjestelmä. Teknologioiden kehittyessä WWW muuttuu entistä monipuolisemmaksi ja toiminnallisemmaksi. Kaikille avoin ja hyödyllinen järjestelmä edellyttää huolellista teknistä toteutusta. Toteutuksen lähtökohtana tulisi aina olla erilais-

ten käyttäjien tarpeiden ja rajoitteiden huomioiminen. Uusien tekniikoiden käyttö ei saa nousta saavutettavuuden edelle.

Internetin yleistyminen köyhimpiinkin maanosiin ja maihin tuo mukanaan haasteita. Fyysiset, tekniset ja taloudelliset eroavaisuudet Internetin käyttäjäryhmien välillä kasvavat. Webin yhdenvertaisen luonteen säilyttämiseksi vaaditaan suunnittelulta erityistä huomiota erilaisuuden huomioonottoon. Saavutettavalla suunnittelulla voidaan välttää maantieteellinen ja taloudellinen lokerointi Internetissä. Uudet tekniikat mahdollistavat tiedonvälityksen ja kommunikaation Webissä entistä helpommin ja tehokkaammin. Mutta mikäli ne sulkevat joiltain käyttäjäryhmiltä pois WWW:n tarjoamaa sisältöä ja toimintoja, täytyisi niiden käytön tarpeellisuutta harkita. Saavutettavuuden takaamiseksi sisältöön tulisi aina päästä käsiksi jollain tavalla, vaikka uudet tekniikat käyttäjällä eivät toimisikaan.

Saavutettavuuden parantamiseksi tehdään paljon työtä. Suositukset ja standardit antavat hyviä suuntaviivoja. Pelkät suositukset sinällään eivät kuitenkaan riitä pitämään WWW:tä laadukkaana. Suunnittelijoiden ja kehittäjien tulisi niitä noudattamalla taata Webin käytettävyys ja saavutettavuus kaikille käyttäjille nyt ja tulevaisuudessa. Tänä päivänä mahdollisuus saavutettaviin sisältöihin ja toimintoihin verkkosivustoilla pitää olla jokaisen ihmisoikeus.

Viiteluettelo

[Adobe, 2008] Flash Player Penetration. http://www.adobe.com/products/player_census/flashplayer/. Checked 1.12.2008.

[Arrue et al., 2006] Myriam Arrue, Markel Vigo and Julio Abascal, Architecture for personal web accessibility. In: *Proc. Of International Conference on Computers Helping People with Special Needs,* 2006, 120-127

[Caldwell et al., 2008] Ben Caldwell, Michael Cooper, Loretaa Guarion Reid and Gregg Vanderheiden, Web Content Accessibility Guidelines (WCAG) 2.0. http://www.w3.org/TR/WCAG20/. Checked 8.12.2008.

[Crisholm et al., 1999] Wendy Crisholm, Gregg Vanderheiden, Ian Jacobs, Web Content Accessibility Guidelines 1.0. http://www.w3.org/TR/WCAG10/. Checked 8.12.2008.

[Gibson, 2007] Becky Gibson, Enabling an accessible web 2.0. In: *Proc. of the 2007 International Cross-disciplinary Conference on Web Accessibility, 2007*, 1-6.

[Jacobs, 2008] Ian Jacobs, About the World Wide Web Consortium. http://www.w3.org/Consortium/. Checked 18.11.2008.

[Nielsen, 2000] Jakob Nielsen, Flash: 99% Bad. http://www.useit.com/alertbox/20001029.html. Checked 1.12.2008.

[Nykänen, 2003] Ossi Nykänen, W3C pähkinänkuoressa. http://www.w3c.tut.fi/reports/2003/0113aboutw3c/index.html. Tarkistettu 3.12. 2008.

[Regan, 2004] Bob Regan, Accessibility and design: a failure of the imagination In: *2004 International Cross-disciplinary Workshop on Web Accessibility (W4A)*, 2004, 29-37.

[Reid and Snow-Weaver, 2008] Loretta Guarino Reid and Andi Snow-Weaver, WCAG 2.0: a web accessibility standard for the evolving web. In: *Proc. of the 2008 International Cross-disciplinary Conference on Web Accessibility*, 2008, 109-115.

[Sivonen et al., 2006] Henri Sivonen, Kai Puolamäki ja Tero Tilus, Videotiedostomuotojen valinta Internet-käyttöön. http://www.effi.org/sananvapaus/videotiedostomuoto.html. Tarkistettu 2.12.2008.

[W3C, 2007] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. http://www.w3.org/TR/CSS21/media.html. Checked 2.12.2008.

[W3C, 2008] W3C Invites Developers to Implement WCAG 2.0. http://www.w3.org/2008/04/wcag20cr-pressrelease.html.en. Checked 3.12. 2008.

[W3Schools, 2008] Browser Statistics. http://www.w3schools.com/browsers/browsers_stats.asp. Checked 1.12.2008.

[W3Schools, 2008a] Browser Display Statistics. http://www.w3schools.com/browsers/browsers_display.asp. Checked 1.12.2008.

[Weakley, 2005] Russ Weakley, Lyhet johdatus esteettömyyteen. http://saavutettava.fi/artikkelit/lyhyt-johdatus-esteettomyyteen/. Tarkistettu 19.11.2008.

[Wikipedia, 2008] Web accessibility. http://en.wikipedia.org/wiki/Web_accessibility. Checked 18.11.2008.

[Wikipedia, 2008a] Web 2.0. http://fi.wikipedia.org/wiki/Web_2.0. Tarkistettu 1.12.2008.

[Wikipedia, 2008b] JavaScript. http://en.wikipedia.org/wiki/JavaScript. Checked 1.12.2008.

[Wikipedia, 2008c] Suoratoisto. http://fi.wikipedia.org/wiki/Suoratoisto. Tarkistettu 2.12.2008.

[Wikipedia, 2008d] Web Accessibility Initiative. http://en.wikipedia.org/wiki/Web_Accessibility_Initiative. Checked 9.12.2008.

Tietokantamalleista ja niiden käytöstä erilaisissa sovellusalueissa ja sovelluksissa

Anne Mikkonen

Tiivistelmä.
Ensiksi tämän tutkielman tarkoituksena on selvittää kirjallisuuden avulla hierarkkisen mallin, verkko-, relaatio- ja oliomallin rakenne, perusominaisuudet, rajoitteet sekä niiden mahdolliset heikkoudet tai puutteet. Toiseksi, tarkoituksena on antaa näkemys siitä, miten tietokantajärjestelmä liitetään käyttäjän käytettäväksi erilaisiin sovelluksiin PHP- ja Java-kielen avulla. Lopuksi kartoitetaan nykyaikaisempia tietokantamalleja hyödyntäviä sovellusalueita ja tietokantasovelluksia.

Avainsanat ja -sanonnat: Tietokantamalli, hierarkkinen malli, verkkomalli, relaatiomalli, oliomalli, tietokantasovellus.
CR-luokat: H.3, H.4, E.1

1. Johdanto

Tiedon tarve on lisääntynyt valtavasti eri aloilla, ja tiedon keräämistä ja tallentamista varten on kehitetty tietokantajärjestelmiä, jotka ovat nykyään kaikkien ulottuvilla. Tämän ansiosta ne ovat myös levinneet laajalti eri alojen käyttöön. Tietokantojen avulla tietoja tallennetaan tietyn rakenteen mukaisesti, jotta ne olisivat helpommin ja nopeammin saatavilla. Näitä erilaisia rakenteita kutsutaan tietomalleiksi, joilla tarkoitetaan suunnitelmaa tai luonnosta, joka kertoo, miten tieto on rakenteellisesti tallennettu [Powell, 2006].

Tietomallit ovat kehittyneet vuosikymmenien aikana aivan kuten muutkin tietotekniikkaan liittyvät asiat. Tutkielmassa kerrotaan lyhyesti neljän tietomallin – hierarkkinen malli, verkko-, relaatio- ja oliomalli – rakenteesta, perusominaisuuksista, niiden rajoitteista sekä heikkouksista. Katsauksen tarkoituksena on kuvata tietomallien kehitystä ja antaa käsitys niiden perusideasta. Lopuksi tutkielmassa kartoitetaan relaatio- ja oliomallia hyödyntävien tietokantojen soveltuvuutta eri sovelluksiin tai sovellusalueisiin.

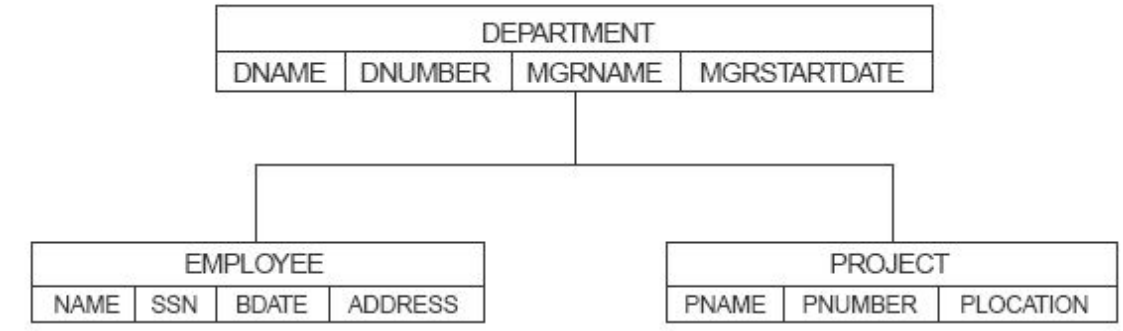
Relaatiotietokanta on eniten käytetty tietokanta sen helppouden ansiosta. Sille ei kuitenkaan ole esitetty varsinaista sovellusaluetta, johon se sopii parhaiten, vaan Elmasri ja Navathe [2007a] sanovat, että se on perinyt sovellukset ensimmäisiltä tietokannoilta eli hierarkkiselta ja verkkotietokannalta. Oliotietokanta on sen sijaan harvinaisempi, vaikka sillä on todettu olevan sovellusalue, suunnittelu- ja valmistustekniikkasysteemit, johon se sopii relaatiotietokantaa paremmin [Koslajda, 1995; Monk *et al.*, 1996].

2. Ensimmäiset tietomallit

Ensimmäisiä tietokantamalleja olivat hierarkkinen ja verkkomalli. Hierarkkinen malli oli näistä ensimmäinen, joka standardoitiin [Hinton and Hinton Jr., 1996]. Tällä mallilla on pyritty kuvaamaan ilmiöitä ja sen avulla ymmärtämään paremmin maailmaa. Sitä on ryhdytty käyttämään eri aloilla, kuten esimerkiksi yritysmaailmassa, jossa organisaation rakenne kuvataan sen avulla [Elmasri and Navathe, 1989]. Verkkomalli luotiin parantamaan hierarkkisen mallin heikohkoa suhdeominaisuutta.

2.1. Hierarkkinen malli

Hierarkkinen malli koostuu *kohteista* (entity) ja niiden välisistä *vanhempi-lapsi -suhteista* (parent-child relationship). Kohteisiin liittyy erityyppisiä ominaisuuksia, joiden avulla niitä kuvataan. Näiden ominaisuuksien joukkoa sanotaan *tietueeksi* (record). Jokaisella ominaisuudella on jokin yhteinen, sitä kuvaava nimi sekä tyyppi. Kohteiden väliset suhteet ovat tyyppiä 1:N, joka tarkoittaa, että kohteella voi olla vain yksi vanhempi ja useita lapsikohteita. [Elmasri and Navathe, 1989]



Kuva 1. Hierarkkinen tietokantamalli [Elmasri and Navathe, 2007b].

Hierarkkista mallia kuvataan yleensä puuna (ks. kuva 1), jonka ominaisuuksia ovat:

1. Yksi kohteista on juuri, joka ei ole minkään kohteen lapsi.
2. Kaikki muut kohteet paitsi juuri ovat lapsia, joilla on ainoastaan yksi vanhempi-lapsi -suhde.
3. Kohteella, joka on vanhempi, voi olla nolla tai enemmän vanhempi-lapsi -suhteita.
4. Kohdetta, jolla ei ole lapsia, kutsutaan lehdeksi (leaf).

Jos kohteella, joka on vanhempi, on useita lapsikohteita, ovat nämä kohteet jollakin tavalla järjestetty. Kaaviossa järjestys luetaan vasemmalta oikealle. [Elmasri ja Navathe 1989]

Todennäköistä kuitenkin on, että tiedon kuvaamisessa suhdetyyppi 1:N ei ole riittävä, koska yleensä tietokannassa olevat tiedot liittyvät monimutkai-

semmalla tavalla toisiinsa. Tällöin tarvitaan suhdetyyppiä N:M, joka voidaan kuvata lisäämällä tietokantaan joko uusia hierarkioita, mikä aiheuttaa redundanttisuutta tai virtuaalisia osoittimia riippuen siitä, miten kehittynyt tietokantajärjestelmä on kyseessä [Elmasri and Navathe, 1989].

Tietokannan monimutkaistuessa eheyden on kuitenkin säilyttävä, josta syystä tietokannalta vaaditaan joitakin rajoitteita, jotka Elmasri ja Navathe [1989] listaavat seuraavasti:

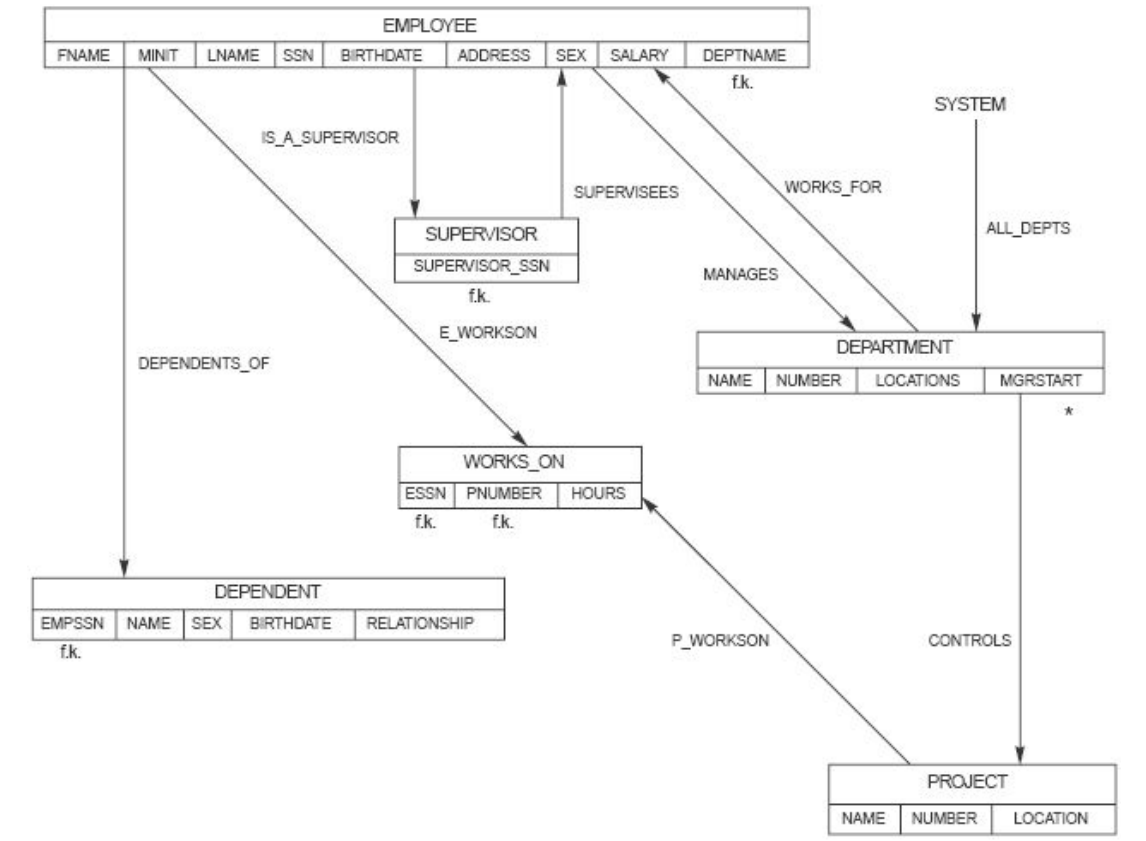
1. Lapsitietuetta ei voi lisätä, ellei sitä liitetä vanhempaan.
2. Lapsitietue voidaan poistaa itsenäisesti, mutta vanhempaa poistettaessa myös sen lapset ja jälkeläiset poistetaan.
3. Virtuaalisessa suhteessa virtuaalinen osoitin tulee olla liitetty olemassa olevaan vanhempaan, ja poistettaessa vanhempi, jolla on virtuaalinen osoitin lapseen, on ensin poistettava osoitin, jotta toiminto on sallittu.
4. Jos lapsella on useampia kuin yksi vanhempi, ja ne omaavat saman tietuetyypin, tulee lapsen tietue lisätä jokaisen vanhemman lapsitietueeksi.
5. Jos lapsella on useampi kuin yksi vanhempi, ja ne omaavat eri tietuetyypin, tulee vain yksi vanhempi liittää todellisella suhteella ja loput virtuaalisella osoittimella lapseen.

Rajoituksista johtuen hierarkkisessa mallissa on huomattu olevan joitakin ongelmia tai heikkouksia. Yksi suurimmista heikkouksista on se, että tietoa, joka on useampaan kertaan tietokannassa (ks. rajoite 4), voidaan päivittää, mutta itse tietokanta ei huolehdi, että kyseinen tieto päivittyy kaikkiin tietueisiin, vaan sen päivittäminen on jätetty lisäysohjelman huolehdittavaksi [Elmasri and Navathe, 1989]. Toinen ongelma saattaa syntyä siitä, että jonkin lapsen tietue halutaan lisätä tietokantaan ennen kuin tiedetään, mihin vanhempaan se liittyy. Tämä voidaan tosin ratkaista tallentamalla vanhempi-tietueeksi ns. valetietue [Hernandez, 2003]. Kolmas heikkous on se, että hierarkkinen malli ei sovellu mallintamaan tietoja, joiden välinen suhde ei varsinaisesti ole vanhempi-lapsi-suhde, vaan tiedot liittyvät jollakin muulla tavalla toisiinsa [Anumba, 1996].

Hierarkkisen mallin rajoitteiden ansiosta kehitettiin uusi siitä kehittyneempi malli, verkkomalli, jonka avulla pystyttiin esittämään kohteiden välisiä suhteita, jotka eivät välttämättä olleet hierarkkisia [Anumba, 1996].

2.2. Verkkotietomalli

Verkkotietomallin osat ovat tietueet ja niiden väliset *joukkorakenteet* (sets), joita voidaan kuvata kuvan 2 mukaisesti. Tietueet muodostuvat ominaisuuksista eli attribuuteista kuvaten kohdetta aivan samalla tavalla kuin hierarkkisessa mallissa. Näiden tietueiden väliset suhteet sen sijaan poikkeavat hierarkkisen mallin mukaisista vanhempi-lapsi -suhteista. [Elmasri and Navathe, 2007b]



Kuva 2. Verkkotietokantamalli [Elmasri and Navathe, 2007b].

Verkkotietomallissa suhteille eli joukkorakenteille on ominaista se, että niillä on nimi, *omistaja* (owner record) ja *jäsen* (member record). Suhde voi olla tyyppiä N:M, joten tämä poistaa redundanttisuuden, ja on tästä syystä kehittyneempi malli hierarkkiseen malliin verrattuna. Lisäksi verkkotietokantamallissa joukkorakenteella on yksi erikoistyyppi, jota kutsutaan *tietokantajärjestelmän omistavaksi joukkorakenteeksi* (system-owned set, singular) ja jonka omistajana on tietokantajärjestelmä. Tällä joukkorakenteella on kaksi tarkoitusta: toisaalta se luo tietokantaan toisen sisääntulon tarvitsematta tietää omistajana olevaa kohdetta, ja toisaalta se voi järjestää joukon eri tavoin, mikä antaa sille mahdollisuuden päästä käsiksi tietoihin eri järjestyksessä. [Elmasri and Navathe, 2007b]

Verkkotietomallin osatekijöiden takia sillä voidaan sanoa olevan ennemminkin vaihtoehtoja kuin rajoitteita. Vaihtoehdot voidaan jakaa koskemaan uuden jäsenen lisäystä ja sitä tilannetta, voiko jäsen olla olemassa ilman, että sillä on suhderakennetta omistajaan. Lisäyksellä on kaksi vaihtoehtoa, automaattisuus (automatic) tai manuaalisuus (manual), joista ensimmäisessä lisättäessä uusi jäsen lisätään sille myös automaattisesti sopiva joukkorakenne, ja jälkimmäisessä puolestaan joukkorakenne voidaan lisätä jälkikäteen manuaalisesti. Olemassaolon vaihtoehdot Elmasri ja Navathe [2007b] jakavat kolmeen: vaihtoehtoinen (optional), pakollinen (mandatory) ja kiinteä (fixed). Vaihtoehtoisessa uusi jäsen voi olla itsenäisenä ilman, että sitä liitetään mihinkään omistajaan tai

se voidaan liittää omistajaansa, joka voidaan myös vaihtaa tarvittaessa toiseen. Pakollisessa jäsen on pakko liittää johonkin omistajaan, mutta se voidaan vaihtaa myöhemmin toiseen. Viimeisessä eli kiinteässä jäsen liitetään kiinteästi omistajaansa joukkorakenteella, jota ei voi myöhemmin vaihtaa. Taulukko 1 yhdistää havainnollisesti nämä vaihtoehdot ja niiden käyttömahdollisuudet.

Taulukko 1. Joukkorakenteiden vaihtoehdot jäsenen lisäyksessä [Elmasri and Navathe, 2007b]

	Vaihtoehtoinen	**Pakollinen**	**Kiinteä**
Manuaalinen	Sovellusohjelma vastaa joukkorakenteen lisäyksestä. Voidaan yhdistää, jättää yhdistämättä tai yhdistää uudelleen toiseen omistajaan		
Automaattinen	DBMS lisää uuteen jäseneen joukkorakenteen automaattisesti. Voidaan yhdistää, jättää yhdistämättä tai yhdistää uudelleen toiseen omistajaan	DBMS lisää uuteen jäseneen joukkorakenteen automaattisesti. Voidaan yhdistää uudelleen toiseen omistajaan.	DBMS lisää uuteen jäseneen joukkorakenteen automaattisesti. Ei voida yhdistää uudelleen.

Verkkomallin monimutkaisuus on yksi sen pääheikkouksista, koska tietojen lisääntyessä myös niiden väliset suhteet luonnollisesti lisääntyvät ja vaikeuttavat tietokannan ylläpitoa ja käyttöä [Anumba, 1996].

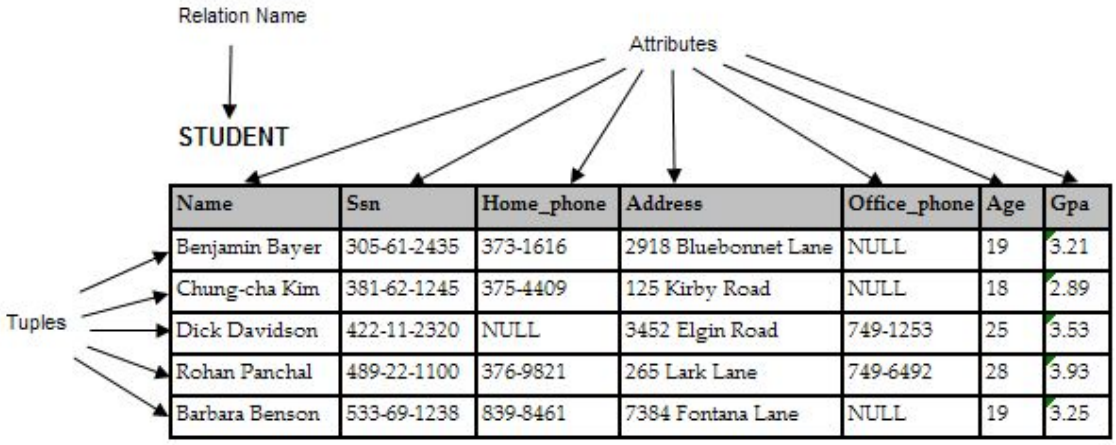
Vaikka verkkotietomalli oli jo kehittyneempi kuin hierarkkinen malli sen eihierarkkisten suhteiden kuvaamiskyvyn vuoksi [Anumba, 1996], se ei kuitenkaan ollut riittävä malli vastaamaan tietokannalta odotettuihin vaatimuksiin. Tietokannoista tehtävät kyselyt monimutkaistuivat siten, että haluttiin tehdä yksittäisiä tai tilapäisiä kyselyitä (ad hoc), joilla haettiin jokin tieto tietokannasta koskien esimerkiksi tiettyä aikaväliä [Oppel, 2004]. Paremmin yksittäisten kyselyjen aiheuttamien haasteiden vastaamiseen kehitettiin täysin eri periaatteella toimiva malli, relaatiomalli.

3. Nykyaikaiset tietomallit

Nykyään yleisin tietomalli on relaatiomalli, jonka ensimmäisenä esitteli IBM:n tutkija E.F. Codd 1970. Ensimmäinen kaupallinen tietokantajärjestelmä, joka perustui relaatiomalliin, tuli markkinoille 1980-luvun alkupuolella IBM:n ja ORACLE:n toimesta. Tämän jälkeen perässä ovat seuranneet mm. Microsoft SQL serverillä ja Accessilla. Oliomallin tietokantajärjestelmät ovat hieman tuntemattomampia, vaikka ne ovat tulleet markkinoille 1980-luvulla. [Elmasri and Navathe, 2007a; Oppel 2004]

3.1. Relaatiomalli

Relaatiomalli koostuu joukosta relaatioita, jotka voidaan määritellä matemaattisesti joukko-opin avulla [Elmasri and Navathe, 2007a]. Aikaisemmat mallit eivät hyödyntäneet matematiikkaa ja siksi tämä oli uutta. Relaatiolla on nimi ja vähintään yksi attribuutti, jolla on nimen lisäksi tyyppi ja muoto. Kun attribuuteilla on arvot, muodostuu *monikko* (tuple) eli taulun yksi rivi. Relaatio kuvataan kaksiulotteisena taulukkona, jota esittää kuva 3. Tietokannan relaatiot liitetään toisiinsa avaimien avulla, joiden ansiosta niiden välille muodostuu yhteys eli suhde.



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Kuva 3. Relaatio [Elmasri and Navathe, 2007a]

Relaatiomallin ominaisuuksiin kuuluu, ettei sen monikoiden järjestyksellä relaatiossa ole merkitystä. Siten voidaan sanoa relaatioiden olevan samat silloin, kuin niissä on samat tiedot tallennettu, vaikka sen tietueet olisivatkin eri järjestyksessä. Tämän mallin ominaisuutena voidaan pitää myös sitä, ettei se hyväksy moniarvoisia attribuutteja, vaan vaatii niiden olevan atomisia. [Elmasri and Navathe, 2007a]

Relaatiomallin rajoitteet johtuvat osin sen ominaisuuksista, mistä hyvänä esimerkkinä on se, ettei malli hyväksy moniarvoisia attribuutteja. Myös Attribuuttien arvoihin liittyy rajoitteita, joita kutsutaan avainrajoitteiksi ja NULL-

arvon rajoitteiksi. Ensimmäinen rajoite liittyy avaimiin, joiden arvot eivät saa olla samat kahdella tai useammalla monikolla. Jälkimmäinen rajoite puolestaan joko sallii tai ei salli attribuutille NULL-arvon. [Elmasri and Navathe, 2007a]

NULL-arvon salliminen ei saa kuitenkaan rikkoa entiteetin eheyttä. Entiteetin eheysrajoitteen mukaan avaimena olevan attribuutin arvo ei voi olla NULL sen yksilöivän ominaisuutensa vuoksi. Viite-eheysrajoitteen tarkoituksena on pitää yllä kahden relaation välistä konsistenssia, mikä tarkoittaa, että viitattaessa relaatiosta toisen relaation attribuutin arvoon täytyy sen arvon olla olemassa sen relaation jonkin monikon arvona, johon on viitattu. Relaatiolla voi lisäksi olla *semanttisia eheysrajoitteita* (semantic integrity constraints), jotka saatetaan voimaan jonkin toisen sovellusohjelman kautta tai yleisen kielen avulla, mikä on luotu käytettäväksi tähän tarkoitukseen, kuten mm. SQL:n liipaisimet. Semanttisesta rajoitteesta esimerkkinä on se, ettei seteliautomaatti anna rahaa, mikäli tilillä ei ole vähintään nostoa vastaavaa summaa rahaa Edellä esitetyssä esimerkissä tietokannan tilaa muutetaan muuttamalla tilin saldoa. Muuttamisen jälkeen tilan tulee säilyä eheänä. Tilan on pysyttävä eheänä tiedon päivittämisen tai muuttamisen ohella myös lisäyksen ja poistamisen jälkeen. [Elmasri and Navathe, 2007a]

Relaatiomallia käyttävät tietokantajärjestelmät ovat nykyään yleisessä käytössä erilaisissa sovelluksissa niin tuotannollisissa yrityksissä kuin palveluyrityksissä ja julkisissa laitoksissa. Sen heikkous on kuitenkin käsitellä monimutkaisempia tietotyyppejä kuten kuvia, kompleksisia piirustuksia sekä audiovisuaalisia tiedostoja [Oppel, 2004]. Lisäksi relaatiomallilla on rajallinen tiedon käsitteellistämiskyky ja sen perusominaisuus, redundanttisen tiedon omaaminen, vaatii normalisointioperaatioita [Anumba, 1996].

Koska relaatiomallin heikkous on käsitellä monimutkaisempia tietotyyppejä, objekteja sekä pidempiaikaisia transaktioita, tarvittiin uusi malli, oliomalli, joka sai paljon vaikutteita ohjelmoinnin puolelta, jossa olio-ohjelmoinnin käyttö lisääntyi [Elmasri and Navathe, 2007a].

3.2. Oliomalli

Olio on oliomallin perusosa, jolla on kaksi komponenttia, tila ja operaatiot, ja joka tallennetaan pysyvästi massamuistiin (secondary storage), josta tietokanta jakaa oliot useiden ohjelmien ja sovellusten käyttöön. Jotta oliot voidaan tallentaa tietokantaan itsenäisinä olioina, ne on yksilöitävä. Yksilöinti voidaan toteuttaa tietokantajärjestelmän yksilöintigeneroinnin avulla, mutta se voidaan myös asettaa manuaalisesti halutulla muuttujalla. Identifiointitunniste ei näy ulkopuoliselle käyttäjälle, vaan se hallinnoidaan järjestelmän sisäisenä toimintona. Koska identifiointitunnistearvon tulee olla uniikki, sen täytyy olla muuttuma-

ton, ja siksi objektin osoite ei voi olla yksilöivänä tekijänä. [Elmasri and Navathe, 2007a]

Tila perustuu olion määriteltyyn rakentajaan, joka voi olla tyypiltään alkeisosa (atom), monikko (tuple), joukko (set), lista (list), pakkaus (bag) tai taulukko (array). Atominen rakentaja määrittelee oliolle muuttujan arvon, joka voi olla tyypiltään tietokantajärjestelmien tukemia perustyyppejä, kuten integer, character, string ja boolean. Monikossa rakentaja on muotoa: $<a_1:i_1, a_2:i_2,\ldots, a_n:i_n>$, missä a on attribuutin nimi ja i olion yksilöintitunnus. Tyyppiä joukko oleva rakentaja on muotoa: $\{i_1, i_2,\ldots, i_n\}$, jossa yksilöintitunnukset voivat olla missä järjestyksessä tahansa. Lista esitetään muodossa $[i_1, i_2,\ldots, i_n]$, yksilöintitunnukset on järjestetty. Taulukko-tyyppi on yksidimensionaalinen taulukko, jonka koko on ennalta määrätty, kun taas lista on dynaaminen. Pakkaus on puolestaan verrattavissa joukkoon, mutta sillä erolla, että se voi sisältää duplikaatteja elementtejä. Alla on esitetty (esimerkki 1) muutamia erilaisia olioita, joiden rakentajat ovat erityyppisiä, ja jotka koostuvat identifiointitunnuksesta (i), rakentajan tyypistä (c) sekä olion tilasta (v). [Elmasri and Navathe, 2007a]

Esimerkki 1

O1 = (i_1, atom, 'Houston')
O2 = (i_2, atom, 5)
O3 = (i_3, atom, 'Research')
O4 = (i_4, set, {i_1, i_2, i_3})
O5 = (i_5, tuple, <Dname:i_3, Dnumber:i_2, Mrg:i_7, Locations:i_1, Employees:i_6, Projects:i_8>) jne.

Olioiden rakenne määritellään luokkakohtaisesti, ja luokkiin määritellään avaimien lisäksi myös suhteet toisiin luokkiin ja olioihin. Avaimet ovat attribuutteja, joiden arvot ovat uniikkeja muihin nähden aivan samalla tavalla kuin relaatiomallissa. Suhteet puolestaan voivat olla joko yksi- tai kaksisuuntaisia. Yksisuuntainen suhde määritellään attribuutilla ja kaksisuuntainen suhde lauseella esimerkin 2 mukaisesti. [Elmasri and Navathe, 2007a]

Esimerkki 2

Yksisuuntainen
attribute struct Dept_mgr {EMPLOYEE Manager, date Start_date}
Mgr;
Kaksisuuntainen
relationship DEPARTMENT Works_for
inverse DEPARTMENT::Has_emps;.

Olioilla on elinkaari, joka kuvastaa sitä, ovatko ne tilapäisiä vai pysyviä. Tilapäiset oliot ovat olemassa vain ohjelman käynnissäoloajan, ja niitä käytetään esimerkiksi pysyvien olioiden tietojen manipulointiin. Pysyville olioille voidaan antaa nimi, jolla niihin voidaan viitata ohjelmassa. Nimiä käytetään myös kohtina, joista päästään tietokantaan sisään. [Elmasri and Navathe, 2007a]

Jotta päästään suorittamaan tietokannassa toimintoja, ne on määriteltävä. Näitä toimintoja voivat olla mm. lisäys, poisto, päivitys tai näiden yhdistelmä sekä jonkin tiedon laskenta jostakin oliossa olevasta tiedosta, kuten ikä syntymäajasta. Oliomallissa kapselointi on yksi sen pääominaisuuksista, mutta tietokannassa täysi kapselointi heikentää sen käytettävyyttä, kuten tilapäisten kyselyjen tekemistä, joten kapselointia kevennetään näkyvillä attribuuteilla, joiden tietoihin on mahdollista päästä käsiksi suoraan ulkopuolelta. Koska vain osa olion toiminnoista on kapseloitu, on järkevää miettiä, mitkä toiminnot ne ovat. Yleensä olion tilaa päivittävät operaatiot on kapseloitu, jotta tietokannan eheys voidaan taata paremmin. [Elmasri and Navathe, 2007a]

Olio-ohjelmoinnista perityistä ominaisuuksista huolimatta Machura ja Jebb [1998] sanovat oliomallin heikkoudeksi sen monimutkaisuuden ja tiedon riippuvuuden.

4. Tietokantajärjestelmät ja niiden käyttämiä muita teknologioita

Edellä esiteltyihin tietomalleihin perustuen on markkinoilla tietokantajärjestelmiä, jotka on nimetty myös mallinsa mukaisesti hierarkkiseksi tietokantajärjestelmäksi, verkko-, relaatiotieto- ja oliotietokantajärjestelmäksi. Tietokantajärjestelmät kommunikoivat eri teknologioiden kanssa ja ovat myös yhteensopivia monien eri ohjelmointikielten kanssa.

Internetin yleistyessä myös elektroninen kaupankäynti (electronic commerce, e-commerce) sekä muut Internet-sovellukset yleistyivät. Internet-sivu toimii käyttöliittymänä, jonka avulla käyttäjät pääsevät käsiksi tietokannan tietoihin. HTML (Hypertext Markup Language) on laajalti käytetty kieli yhdessä PHP:n (Hypertext Preprocessor) kanssa toteutettaessa liittymää käyttäjän ja tietokannan välille, koska PHP:ssä on monia luokkakirjastoja, joiden avulla saadaan yhteys sekä voidaan tehdä kyselyitä tietokantaan. Tällä hetkellä PHP tukee taulukossa 2 esitettyjä tietokantoja [Achour *et al.*, 2008].

Taulukko 2. PHP:n tukemia tietokantoja [Achour *et al.*, 2008]

Dbase	DB++	FrontBase	filePro
Firebird/InterBase	Informix	IBM DB2	Ingres II
MaxDB	mSQL	Mssql	MySQL
Mysqli	Oracle OCI8	Ovrimos SQL	Paradox
PostgreSQL	SQLite	SQLite3	Sybase

The Java Database Connectivity (JDBC) on Java-kielen rajapinta, jonka avulla luodaan yhteys relaatiotietokantaan. Saadakseen yhteyden muodostuksen toimimaan tarvitaan myös ajuri, joita on neljää eri tyyppiä [Boger, 2001]:

1. JDBC/ODBC-silta, jonka avulla pääsee käyttämään kantaa, joka perustuu ODBC:iin (Open Database Connectivity),
2. Alustaan yksilöity JDBC-ajuri, joka usein on kirjoitettu C- tai C++-kielellä, eikä sovellu käytettäväksi yleisesti sovelluksissa, koska sitä ei voi ladata serveriltä selaimeen,
3. Universaali JDBC-ajuri, joka kommunikoi serverin komponenttien kanssa ja sovellukset voivat käyttää sitä suoraan ilman erillistä asennusta asiakkaan koneella,
4. Suora JDBC-ajuri, joka toimii suoraan tietokannan käytännön tasolla.

Näistä yleisimmin käytetyt ovat JDBC/ODBC-silta ja universaali JDBC, joista silta on edullisempi sen ilmaisen saatavuuden johdosta, mutta se on hitaampi ja joustamattomampi universaaliin JDBC:iin verrattuna [Boger, 2001].

Yhteyden luomisen jälkeen käyttäjä voi tehdä kyselyitä tietokantaan luokkien Statement, PreparedStatement ja CallableStatement avulla riippuen siitä, miten monimutkaisesta kyselystä on kyse sekä saada käyttöönsä kyselyn tulokset ResultSet -luokan avulla [Boger, 2001].

Oliotietokannat ovat vielä melko harvojen toimijoiden käytössä, joten Roger [2001] esittää yhteyden luomisen ObjectStore-tietokantajärjestelmään, joka tukee Java-kieltä hyvin. Ensin luodaan yhteys Session-luokan avulla ja sen jälkeen siihen liitytään join-metodilla. Tämän jälkeen tietokanta avataan Database-luokan open-metodilla. Tietokantaa avattaessa sille kerrotaan parametrin avulla, halutaanko tietokannasta vain lukea tietoja vai myös päivittää niitä. Transaction-luokan kautta tietokannasta päästään lukemaan ja päivittämään tietoja. Transaktio avataan ja sille kerrotaan, luetaanko vain tietoja vai halutaanko niitä myös päivittää, jotta tietokanta ymmärtää voiko olla useampi samanaikainen transaktio olemassa. Haluttujen toimintojen jälkeen transaktio suljetaan commit-metodilla, joka päivittää tietokantaan muiden nähtäväksi muutetut tai lisätyt tiedot. Lopuksi vielä suljetaan tietokanta ja yhteys, jottei tietokanta lukkiinnu.

5. Tietokantojen käyttämiä sovelluksia ja sovellusalueita

Kuten edellä on mainittu, tietokannat voidaan liittää sovelluksiin useamman eri teknologian avulla, jotta niitä voidaan hyödyntää monenlaisissa sovelluksissa. Tietokantoja on hyödyllistä käyttää sovelluksissa, jos useamman käyttäjän on tarve päästä tietoihin yhtäaikaisesti tai tiedot muuttuvat jatkuvasti. Ei kuitenkin ole täysin selvää, mikä tiettyä tietomallia käyttävä tietokanta soveltuu parhaiten mihinkin tarkoitukseen. Relaatiotietokantoja käytetään samoissa sovelluksissa kuin hierarkkista ja verkkomallia soveltavia tietokantoja, ja oliotietokantojen käyttö on yleistä konetekniikan suunnittelussa, multimedian tuottamisessa ja tuotantojärjestelmissä [Elmasri and Navathe, 2007a].

5.1. Esimerkkejä relaatiotietokantaa käyttävistä sovelluksista

Relaatiotietokanta on käytetyin tietokanta tietokantasovelluksissa [Elmasri ja Navathe, 2007a] sen helppokäyttöisyyden, yksinkertaisuuden ja tiukan matemaattisen perustan ansiosta [Stoimenov *et al*., 1999]. Relaatiomallin käyttökelpoisuutta ei ole kategorisoitu yksittäisiin sovelluksiin tai sovellusalueisiin, vaikkakin Stoimenov ja kumppanit [1999] sanovat, ettei se sovellu käytettäväksi eitavanomaisissa sovelluksissa, jotka tarvitsevat monimutkaisempia funktioita käsitelläkseen kompleksisimpia tietotyyppejä kuten pitkiä tekstejä, kuvia sekä audio- ja videotietoja.

Yksittäisiä sovelluksia, joissa käytetään relaatiotietokantaa hyödyksi, löytyy lukematon määrä. Yksi syy sovellusten runsaslukuisuudelle on Internetin hyötykäytön lisääntyminen esimerkiksi elektronisessa kaupankäynnissä. Tavanomaisiksi kaupankäynnin kohteiksi ovat yleistyneet muun muassa kirjat, elokuvat, CD-levyt, pelit, tietokonetarvikkeet ja jopa isommat tavarat kuten kodinkoneet.

Myös erilaiset varausjärjestelmät kuten kokoushuoneiden, liikuntavuorojen sekä muiden tilojen varausohjelmat voivat hyödyntää relaatiotietokantaa. Mikään ei myöskään estä relaatiotietokannan hyödyntämistä toimistosovelluksissa kuten esimerkiksi myyntitilausten käsittelyssä, laskutuksessa sekä kirjanpitojärjestelmissä.

Vaikka Internet antaa mahdollisuuden tehdä sovelluksia maantieteellisesti laajalle levinneelle käyttäjäkunnalle, se ei sulje pois web-selaimen hyödyntämistä yhden käyttäjän tarpeisiin tehdyssä sovellusohjelmassa. Yksi omakohtainen esimerkki tällaisesta sovelluksesta on Tampereen yliopiston käännöstieteen laitoksen tutkijalle kehitettävä elektroninen sanakirja, joka sisältää lääketieteen termejä ja niiden merkityksiä aikakaudelta 1375 - 1550.

Relaatiotietokantasovelluksen ei tarvitse myöskään olla jokaisen ulottuvilla ja mistä tahansa käytettävissä, vaan se voi olla jokaisen omalla koneella oleva ohjelma. Yksi tällaisista ohjelmista on Microsoft Windows Installer, joka hyö-

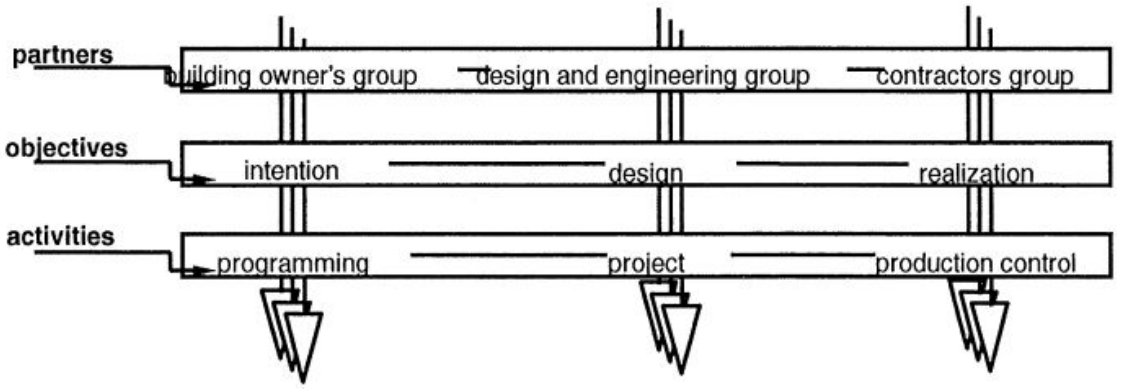
dyntää relaatiotietokantaa asennusohjeiden ja ohjelman poistossa tarvittavien tietojen säilytyksessä [Sun Microsystems, 2007].

5.2. Esimerkkejä oliotietokantaa käyttävistä sovelluksista ja sovellusalueista

Oliotietokantaa sovelletaan erilaisiin suunnittelu- ja valmistustekniikkasysteemeihin kuten CASE (The Computer Aided Software Engineering), yleisesti CAD (The Computer Aided Design) ja sen erityisalat MCAD (The Mechanical Computer-Aided Design), ECAD (The Electronical Computer-Aided Design), sekä CAM (The Computer-Aided Manufacturing), koska nämä tarvitsevat oliomallin erikoisominaisuuksia, kuten monimutkaisemman rakenteen ja monien suhteiden hallintaa, kohteen ja sen komponenttien käyttäytymisen mallinnusta sekä niiden varianttien ja versioiden hallintaa [Koslajda, 1995; Monk *et al.*, 1996]. Lisäksi oliotietokantojen käyttö on levinnyt tietoliikenneteknologiaan, terveydenhuoltoon sekä liiketoiminta-alaan kuten rahoituspalveluihin ja yleisesti liiketoimintaprosessien hallintaan [Machura and Jebb, 1998].

Tietojärjestelmien suunnittelussa ja toteutuksessa käytetään avuksi CASE-välineitä, jotka auttavat kahdessa eri vaiheessa, määrittely- ja suunnitteluvaiheessa sekä toteutusvaiheessa. Ensimmäisessä vaiheessa käytettyjä välineitä kutsutaan edustavälineiksi ja jälkimmäisiä taustavälineiksi, joita ovat esimerkiksi kääntäjät ja sovelluskehittimet. Pidemmälle viedyt välineet perustuvat tietokantaa, johon monimutkaisempia tietomalleja vaativat tiedot tallennetaan, jotta niitä voidaan käyttää uudelleen jonkin uuden, mutta samankaltaisen ohjelmistoprojektin suunnittelun ja toteuttamisen pohjana. [Haikala ja Märijärvi, 2006]

Kim ja Han [2003] toteuttivat prototyypin laivan suunnitteluohjelmasta, jossa käytettiin perinteisen kansioon tallentamisen sijasta oliomallin hyödyntämää tietokantaa, joka mahdollisti samanaikaisen pääsyn tietoihin.



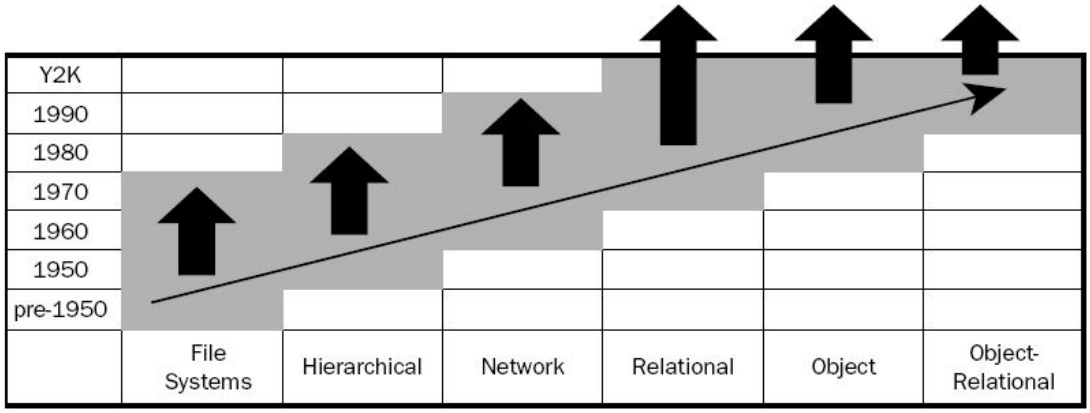
Kuva 4. Rakennusprosessi ja sen osanottajat [Ameziane, 2000].

Rakennusteollisuus on yksi hyvä esimerkki, jossa yhdistyvät sekä mekaanisten kohteiden tietokoneavusteisen suunnittelun (MCAD) ja tietokoneavusteisen valmistuksen CAM soveltaminen rakennettaessa tietojärjestelmää kyseiselle alalle. Ameziane [2000] kertoo, kuinka oliomallin käyttämää tietokantaa hyödynnettiin järjestelmän luomisessa, jossa yhdistettiin useamman käyttäjäryhmän pääsy samoihin tietoihin (kuva 4), jotta voitiin parantaa ja nopeuttaa rakennuksen valmistusprosessia aina rakennuksen suunnittelusta sen toteutukseen. Tämä mahdollisti myös sen, että asiakkaalle saatiin täydellinen kuvaus hänen tilaamastaan rakennuksesta.

Tietoliikennealalla oliotietokantaa hyödyntävät järjestelmät sisältävät asiakashallinnan, laskutusjärjestelmän, käyttötukijärjestelmän, etäaseman hallinnan sekä vieläpä ilmoitusjärjestelmän [Machura and Jebb, 1998].

6. Yhteenveto

Tietomallit ovat kehittyneet niiden tarvevaatimusten lisääntyessä (kuva 5), ja ne jaetaankin tutkielmassa ensimmäisiin ja nykyaikaisiin malleihin. Nykyaikaisista malleista E. F. Coddin 1970 esittelemä relaatiotietomalli on laajimmalle levinnyt ennen kaikkea sen helppokäyttöisyyden ansiosta. Oliomalli puolestaan on myöhemmin saanut suosiota kompleksisempien asioiden tallennuksessa.



Kuva 5. Eri tietokantamallien kehitys [Powell, 2006].

Relaatiomallia hyödyntäviä tietokantoja käytetään erilaisissa sovelluksissa, mutta niille ei sinänsä ole mitään erityistä sovellusaluetta, johon ne olisivat vahvimmin liitetty. Relaatiomalli soveltuu hyvin sovelluksiin, joiden tiedot ovat tyypiltään yksinkertaisia ja mallinnettavissa tietyllä hierarkkisuudella.

Oliomallin tärkeimpiä ominaisuuksia ovat sen hyväksymät monimutkaisemmat tietotyypit tallennettavissa tiedoissa sekä sen hyvä yhteensopivuus olio-ohjelmoinnin kanssa. Oliomallia hyödyntäviä ensimmäisiä sovellusalueita olivat suunnittelu- ja konetekniikkasysteemit sekä maantieteelliset tietojärjestelmät. Myöhemmin sovellusalueet laajenivat myös muihin sovelluksiin, kuten tietoliikenneteknologiaan, terveydenhuoltoon, tietokoneavusteiseen julkaisemi-

seen, rahoituspalveluihin sekä yleiseen liiketoimintaprosessien hallintaan [Machura and Jebb, 1998].

Koska markkinoille on tullut myös olio-relaatiomallia käyttävä tietokanta, olisi mielenkiintoista tutkia, miten yleinen tämä malli on sovelluksissa tai sovellusalueissa verrattuna aikaisemmin esitettyihin malleihin. Lisäksi eri tietokantojen testaaminen käytännössä esimerkiksi kyselynopeuden ja käytettävyyden kannalta kartoittaisi niiden eroja, rajoja sekä etuja ja sitä kautta myös niiden tehokkuutta eri sovelluksissa.

Viiteluettelo

[Ameziane, 2000] Farid Ameziane, An information system for building production management. *International Journal of Production Economics* **64**, 1-3 (March 2000), 345-358.

[Anumba, 1996] C. J. Anumba, Data structures and DBMS for computer-aided design systems. *Advances in Engineering Software* **25**, 2-3 (March-April, 1996), 123-129.

[Achour et al., 2008] Achour Mehdi, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy and Jakub Vrana, PHP Manual. Available at http://fi.php.net/manual/en/. Checked November 26, 2008.

[Boger, 2001] Marco Boger, *Java™ in Distributed Systems Concurrensy, Distribution and Persistence.* Wiley, 2001.

[Elmasri and Navathe, 1989] Ramez Elmasri and Shakant B. Navathe, *Fundamentals of Database Systems*. The Benjamin/Cummings, R, 1989.

[Elmasri and Navathe, 2007a] Ramez Elmasri and Shakant B. Navathe, *Fundamentals of Database Systems*. Addison-Wesley, 2007.

[Elmasri and Navathe, 2007b] Ramez Elmasri and Shakant B. Navathe, *Fundamentals of Database Systems Appendix D and E of Overview of the Hierarchial Data Model and Network Data Model*. Available at http://www.aw.com/elmasri Checked October 12, 2008.

[Haikala ja Märijärvi, 2006] Ilkka Haikala ja Jukka Märijärvi, *Ohjelmistotuotanto*. Talentum, 2006.

[Hernandez, 2003] Michael J. Hernandez, *Database Design for mere Mortals: a hands-on guide to relational database design.* Pearson Education, 2003.

[Hinton and Hinton Jr., 1996] Mary Hinton and Phillip R. Hinton Jr., Tutorial the LIMS database. *Laboratory Automation and Information Management* **31**, (February, 1996), 161-171.

[Kim and Han, 2003] Junhwan Kim and Soonhung Han, Encapsulation of geometric functions for ship structural CAD using a STEP database as native storage. *Computer-Aided Design* **35**, 13 (November 2003), 1161-1170.

[Koslajda, 1995] Tomasz Koszlajda, Implementation models of configuration management. *Information Sciences* **84**, 1-2 (May 1995), 129-138.

[Machura and Jebb, 1998] Marek Machura and Tim Jebb, Selected issues in object-oriented database design. *BT Technology Journal* **16**, 3 (July 1998), 193-204.

[Monk et al., 1996] Simon Monk, John A Mariani, Beshir Elgalal and Helen Campbell, Migration from relational to object-oriented databases. *Information and Software Technology* **38**, 7 (1996), 467-475.

[Oppel, 2004] Andrew J. Oppel, *Databeses Demystified – a Self-Teaching Guide.* McGraw-Hill, 2004.

[Powell, 2006] Gavin Powell*, Beginning Database Design.*Wiley, 2006.

[Stoimenov et al., 1999] Leonid Stoimenov, Antonija Mitrovic, Slobodanka Djordjevic-Kajan and Dejan Mitrovic, Bridging object and relations: a mediator for an OO front-end to RDBMSs. *Information and Software Technology* **25**, 25 (January, 1999), 57-66.

[Sun Microsystems, 2007] Sun Microsystems, Overview of the Windows Installer Technology. Available at http://support.microsoft.com/kb/310598 Checked December 7, 2008.

Ehrenfeuchtin ja Fraïssén peli - tietokantateoreettinen näkökulma

Piia Nieminen

Tiivistelmä

Ehrenfeuchtin ja Fraïssén peli on äärellisten mallien teoriassa sovellettu menetelmä, joka perustuu struktuurien samanlaisuuden tutkimiseen. Tässä tutkielmassa Ehrenfeuchtin ja Fraïssén pelin käyttökelpoisuutta tarkastellaan tietokantojen ja kyselykielten kannalta. Voidaan osoittaa, että Ehrenfeuchtin ja Fraïssén peli riittää karakterisoimaan ensimmäisen kertaluvun logiikan ilmaisuvoiman, johon tietokantateorian käytetyimpien kyselykielten ilmaisuvoima pohjautuu.

Avainsanat: Malliteoria, ensimmäisen kertaluvun logiikka, ilmaisuvoima, määriteltävyys.
CR-luokat: F.4.1

1 Johdanto

Äärellisten mallien teoria on äärellisiä struktuureita ja niiden ominaisuuksia tutkiva logiikan suuntaus, jonka sovellusalueet löytyvät suurelta osin tietojenkäsittelytieteestä. Itse asiassa äärellisten mallien teorian voidaan katsoa syntyneen tietojenkäsittelytieteen kehityksen tuomista haasteista 1970-luvulla [5]. Tällöin muun muassa vaativuusteoriassa ja tietokantateoriassa ilmenneet ongelmat edellyttivät, että malliteoreettisissa tarkasteluissa rajoituttiin äärellisiin struktuureihin. Äärellisten mallien teorian menetelmien juuret löytyvät matemaattiseen logiikkan pohjautuvasta klassisesta malliteoriasta. Äärellisten mallien teoria erottui omaksi tutkimusalueekseen, koska klassisen malliteorian tutkimuskentillä äärellisiin struktuureihin rajoittumista ei koettu mielenkiintoiseksi. Tämä johtui osittain siitä, että klassisen malliteorian tärkeimmät tulokset edellyttävät äärettömien struktuurien konstruointia [1].

Ehrenfeuchtin ja Fraïssén peli, lyhyemmin EF-peli, on poikkeuksellinen klassisen malliteorian menetelmä, sillä sen sovellusmahdollisuudet ulottuvat äärellisistä struktuureista äärettömiin struktuureihin. EF-peli sai alkunsa Roland Fraïssén menetelmästä kuvata struktuurien elementaarista ekvivalenssia. Myöhemmin Andrzej Ehrenfeucht muotoili Fraïssén algebrallisen hahmotelman peliteoreettiseen muotoon. EF-peli perustuu kahden struktuurin rakenteellisen samanlaisuuden tarkasteluun edestakaisin-menetelmällä (back-and-forth method), jossa kaksi pelaajaa valitsee vuorotellen tarkasteltavien struktuurien alkioita tiettyjen sääntöjen mukaisesti [5].

Tekijä on käsitellyt Ehrenfeuchtin ja Fraïssén peliä laajemmin ja matemaattisemmin pro gradu -tutkielmassaan [6]. Tämän tutkielman tarkoitus on syventää tietoa formaalien kielten ominaisuuksista ja ilmaisuvoimasta tietokantateorian kannalta, sillä tietokantateoriassa määriteltävyyskysymykset nousevat olennaiseen osaan. Tietokantoja käsitellessä on mahdollisesti kysyttävä: "Voidaanko käytettävällä kyselykielellä formalisoida kysely, joka selvittäisi tietokannasta halutut tiedot?" Koska tietokantateorian käytetyimmät kyselykielet pohjautuvat ensimmäisen kertaluvun logiikkaan, tutkielmassa keskitytään juuri ensimmäisen kertaluvun logiikan ilmaisuvoimaan.

Lukijan oletetaan tuntevan diskreetin matematiikan ja matemaattisen logiikan perusteet. Luvussa 2 käydään lyhyesti läpi ensimmäisen kertaluvun logiikan syntaksia ja semantiikkaa. Samalla selvitetään kyselykielten ja ensimmäisen kertaluvun logiikan yhteyttä. Luvussa 3 esitellään olennaisimmat samanlaisuuden käsiteet sekä Ehrenfeuchtin ja Fraïssén pelin kulku. Luku 4 käsittelee kyselyitä ja niiden määriteltävyyttä ensimmäisen kertaluvun logiikassa. Lisäksi neljännessä luvussa esitellään tutkielman tärkeimmät tulokset Ehrenfeuchtin ja Fraïssén pelin kannalta. Tutkielman päälähteiksi mainittakoon Libkinin teos *Elements of Finite Model Theory* [5] sekä Grädelin ja muiden teos *Finite Model Theory and its Applications* [4].

2 Valmistelevia tarkasteluita

Logiikat ja ohjelmointikielet ovat formaaleja kieliä, joiden syntaksi ja semantiikka määritellään aina täsmällisesti. Vielä 1960-luvulla tietokantojen kyselyt olivat vaikeasti kirjoitettavia ohjelmia, mutta 1970-luvulla uuden relaatiomallin synty johti uudenlaisten kyselykielten kehittymiseen [7, p. 6]. Prototyyppinä uusille kielille oli ensimmäisen kertaluvun logiikka, lyhyemmin FO-logiikka, johon myös laajalti käytössä oleva SQL perustuu. Tässä luvussa esitellään tiivistetysti FO-logiikan syntaksi ja semantiikka tietokantateoreettisesta näkökulmasta. Määritelmät pohjautuvat suurimmalta osin Libkinin [5] sekä Ebbinghausin ja muiden [2] teoksiin.

Tietokantojen esittämiseen tarvitaan luonnollisesti jokin kieli. Jotta kielellä voidaan ilmaista asioita, on sen sisällettävä erilaisia symboleita. FO-logiikan perustan muodostaa loogisten ja ei-loogisten symbolien aakkosto. Loogisiin symboleihin lukeutuvat

- muuttujat $v_1, \ldots, v_i$;
- konnektiivit $\neg$, $\vee$;
- sulkeet (,);
- eksistenssikvanttori $\exists$ ja
- identiteettisymboli $=$.

Loogisten symbolien joukko pysyy muuttumattomana. Sen sijaan ei-loogisten symbolien joukko, symboliaakkosto σ, saattaa vaihdella kontekstista riippuen. Tästä syystä voidaan sanoa, että symboliaakkosto σ määrittää ensimmäisen kertaluvun logiikan kielen. Jatkossa aakkostolla σ viitataan aina seuraavan määritelmän mukaiseen relationaaliseen aakkostoon, ellei toisin mainita.

Määritelmä 2.1. *Relationaalinen aakkosto* σ on ei-tyhjä äärellinen joukko vakiosymboleita $c_1, \ldots, c_n$ ja relaatiosymboleita $R_1, \ldots, R_n$. Jokaisella relaatiosymbolilla on paikkaluku $k \in \mathbb{N}^+$, joka kertoo relaation attribuuttien määrän.

Tavallisen kielen tapaan symboleita ei luetella mielivaltaisesti toinen toisensa perään, vaan asian ilmaisemiseksi symbolijonon on noudatettava tiettyjä sääntöjä. Symbolijonoja kutsutaan FO-kaavoiksi vain silloin, kun ne on muodostettu FO-logiikan kaavanmuodostussääntöjen mukaisesti. Kaavanmuodostussääntöjä varten tarvitaan termin käsite, jolla tarkoitetaan yleisesti predikaattilogiikoiden kaavojen yksilöitä eli subjekteja. *Termejä* ovat muuttujat ja aakkoston σ vakiot. Tavanomaisesti symbolit x, y, z, ... on varattu muuttujille, symbolit c_1, c_2, c_3, ... vakioille ja symbolit t_1, t_2, t_3, ... termeille. Käytetään merkintää $\mathrm{var}(\sigma)$ aakkoston σ muuttujien joukolle.

Määritelmä 2.2. Olkoon σ aakkosto. Määritellään FO-kaavat induktiivisesti:

(i) Jos t_1 ja t_2 ovat termejä, niin $t_1 = t_2$ on kaava.

(ii) Jos $t_1, \ldots, t_k$ ovat termejä ja R on k-paikkainen relaatiosymboli, niin $R(t_1, \ldots, t_k)$ on kaava.

(iii) Jos φ_1 on kaava, niin $\neg\varphi_1$ on kaava.

(iv) Jos φ_1 ja φ_2 ovat kaavoja niin $\varphi_1 \vee \varphi_2$ on kaava.

(v) Jos φ on kaava, niin $\exists x\varphi$ on kaava.

Atomikaavoiksi kutsutaan vain niitä kaavoja, jotka on muodostettu soveltamalla kaavanmuodostussääntöjä (i) ja (ii). Jatkossa käytetään standardeja lyhenteitä

$$\varphi \wedge \psi,$$
$$\varphi \Rightarrow \psi,$$
$$\varphi \Leftrightarrow \psi \quad \text{ja}$$
$$\forall x\varphi$$

merkintöjen $\neg(\neg\varphi \vee \psi)$, $\neg\varphi \vee \psi$, $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$ ja $\neg\exists x\neg\varphi$ sijasta tässä järjestyksessä.

Olkoon S joukko kaavoja. Jos joukon S kaavoista muodostetaan uusia kaavoja konnektiiveilla $\vee$, $\wedge$ ja $\neg$, niin uusia kaavoja kutsutaan S-kaavojen *Boolen kombinaatioiksi*.

Kaavassa $\neg\forall xRxy \wedge \exists xRxx$ kaikki muuttujan x esiintymät ovat kvanttorien vaikutuksen alaisia. Tällaisia muuttujien esiintymiä kutsutaan *sidotuiksi*. Kyseisessä kaavassa muuttuja y ei ole yhdenkään kvanttorin sitoma. Tästä syysyä muuttujan y esiintymää kutsutaan *vapaaksi*.

Määritelmä 2.3. Olkoon $\text{free}(\varphi)$ kaavassa φ esiintyvien vapaiden muuttujien joukko, jolloin

$$\begin{aligned}
\text{free}(\neg\varphi) &:= \text{free}(\varphi);\\
\text{free}(\varphi * \psi) &:= \text{free}(\varphi) \cup \text{free}(\psi), \text{ kun } * \text{ on } \wedge,\ \vee,\ \Rightarrow \text{ tai } \Leftrightarrow;\\
\text{free}(\forall v\varphi) &:= \text{free}(\varphi) - \{v\};\\
\text{free}(\exists v\varphi) &:= \text{free}(\varphi) - \{v\}.
\end{aligned}$$

Muuttujatermin x ainoa vapaa muuttuja on x. Vakiolla c ei ole vapaita muuttujia. Kaavan $t_1 = t_2$ vapaat muuttujat ovat termien t_1 ja t_2 vapaat muuttujat. Kaavan $R(t_1, \ldots, t_k)$ vapaat muuttujat ovat termien $t_1, \ldots, t_k$ vapaat muuttujat.

Kun kaavan φ vapaat muuttujat ovat muuttujien $x_1, \ldots, x_n$ joukossa, voidaan merkitä $\varphi(\overrightarrow{x})$. Kaavaa, jossa ei ole vapaita muuttujia, kutsutaan *lauseeksi*. Merkitään lauseita symboleilla Φ, Ψ, ...

Ensimmäisen kertaluvun kaava φ saa merkityksen, kun sen totuutta tutkitaan tietyssä struktuurissa. Tässä tutkielmassa tarkastellaan vain äärellisiä relationaalisen aakkoston struktuureita, jotka ovat verrattavissa relaatiomallin tietokantoihin.

Määritelmä 2.4. Olkoon σ relationaalinen aakkosto, jolloin σ-struktuuri $\mathcal{A} = \langle A, \alpha\rangle$ koostuu

- ei-tyhjästä joukosta A, jota kutsutaan struktuurin $\mathcal{A}$ universumiksi tai arvojoukoksi $\text{dom}(\mathcal{A})$, ja
- funktiosta α, joka kuvaa aakkoston σ symbolit X struktuurin $\mathcal{A}$ tulkinnalleen $\alpha(X) = X^{\mathcal{A}}$.

Nyt kaikille vakiosymboleille $c \in \sigma$ tulkinta $c^{\mathcal{A}}$ on struktuurin $\mathcal{A}$ alkio, eli $c^{\mathcal{A}} \in A$. Aakkoston σ k-paikkaisten relaatiosymbolien tulkinta $R^{\mathcal{A}}$ on struktuurin $\mathcal{A}$ k-paikkainen relaatio, jolloin $R^{\mathcal{A}} \subseteq A^k$. Usein struktuuri esitellään

käyttämällä merkintätapaa $\mathcal{A} = \langle A, \{c_i^{\mathcal{A}}\}, \{R_i^{\mathcal{A}}\}\rangle$, jolloin säästytään funktion α tulkintojen erilliseltä esittelyltä. Struktuuri $\mathcal{A}$ on äärellinen, jos sen universumi A on äärellinen joukko.

Kun relaatiotietokanta koostuu vain yhdestä kaksipaikkaisesta relaatiosta, voidaan sen rakenne kuvata verkolla [1, pp. 1-2].

Määritelmä 2.5. Olkoon $\sigma = \{E\}$, missä E on kaksipaikkainen relaatiosymboli. Aakkoston σ struktuuria $\mathcal{G} = \langle G, E^{\mathcal{G}}\rangle$ kutsutaan *verkoksi*, kun $E^{\mathcal{G}} \subseteq G^2$ on irrefleksiivinen sekä symmetrinen.

Verkon $\mathcal{G}$ alkioita kutsutaan *solmuiksi*. Pareja $g_1, g_2 \in G$ kutsutaan *särmiksi*, kun $(g_1, g_2) \in E^{\mathcal{G}}$ on voimassa. Jos solmut g_1 ja g_2 muodostavat särmän, kutsutaan niitä toistensa *naapureiksi*. Olkoon n positiivinen kokonaisluku ja olkoot $Eg_1g_2, Eg_2g_3, \ldots, Eg_{n-1}g_n$ verkon $\mathcal{G}$ relaatioita. Tällöin jono $g_1, \ldots, g_n$ muodostaa *polun* solmusta g_1 solmuun g_n. Sanotaan, että verkko $\mathcal{G}$ on *konnektiivinen* eli *yhtenäinen*, jos mielivaltaisesta solmusta $g \in G$ on olemassa polku jokaiseen solmuun $g' \in G$. Verkkoa, joka on irrefleksiivinen mutta ei symmetrinen, sanotaan *suunnatuksi verkoksi*.

Esimerkki 2.1. [1, p. 12] Koostukoon tietokanta kaikista maailman kaupunkien nimistä ja kaupunkien välisistä relaatioista siten, että kaupunki a on relaatiossa kanupungin b kanssa vain, jos kyseisten kaupunkien välillä on suora lentoyhteys ilman välipysähdyksiä. Tietokanta voidaan esittää aakkoston $\sigma = \{E\}$ suunnattuna verkkona $\mathcal{G} = \langle G, \alpha\rangle$, missä universumi G koostuu kaupungeista, ja missä funktio α kuvaa relaatiosymbolin E struktuurin $\mathcal{G}$ relaatioksi $E^{\mathcal{G}}$. Nyt relaatio $E^{\mathcal{G}}ab$ tarkoittaa, että on olemassa suora lento kaupungista a kaupunkiin b.

Termien arvo aakkoston σ struktuurissa $\mathcal{A}$ määritellään seuraavasti. Olkoon $x_1, \ldots, x_n$ vapaiden muuttujien jono, jolloin termille t määritellään arvo $t^{\mathcal{A}}(\overrightarrow{a})$, missä $\overrightarrow{a} \in A^n$. Termin t ollessa muuttuja x_i, termin arvo struktuurissa $\mathcal{A}$ on $t^{\mathcal{A}}(\overrightarrow{a}) = a_i$. Jos termi t on aakkoston σ vakiosymboli c, termin arvo struktuurissa $\mathcal{A}$ on vakion c tulkinta $\alpha(c) = c^{\mathcal{A}}$.

Ensimmäisen kertaluvun logiikan semantiikka eli merkitysoppi rakentuu toteuttamisrelaation $\vDash$ ympärille. Määritellään ensimmäisen kertaluvun kaavojen toteutuminen aakkoston σ mielivaltaisessa struktuurissa $\mathcal{A}$. Olkoon

$\overrightarrow{a} \in A^n$ jono vakioita ja olkoon $\varphi(\overrightarrow{x})$ FO-kaava. Kirjoitetaan $\mathcal{A} \vDash \varphi(\overrightarrow{a})$, jos kaava $\varphi(\overrightarrow{x})$ toteutuu struktuurissa $\mathcal{A}$ jonolla $\overrightarrow{a}$. Toisin sanoen kaava $\varphi(\overrightarrow{x})$ on tosi struktuurissa $\mathcal{A}$, kun sen jokainen vapaa muuttuja x_i tulkitaan alkioksi $a_i \in A$ kaikilla $1 \leq i \leq n$. Määritellään toteuttamisrelaatio $\vDash$ kaikille FO-kaavoille induktiivisesti:

- Olkoon $\varphi \equiv t_1 = t_2$. Tällöin $\mathcal{A} \vDash \varphi(\overrightarrow{a})$ pitää paikkansa, jos ja vain jos $t_0^{\mathcal{A}}(\overrightarrow{a}) = t_1^{\mathcal{A}}(\overrightarrow{a})$ on voimassa.
- Olkoon $\varphi \equiv Rt_1, \ldots t_k$. Tällöin $\mathcal{A} \vDash \varphi(\overrightarrow{a})$ pitää paikkansa, jos ja vain jos $t_1^{\mathcal{A}}(\overrightarrow{a}), \ldots, t_k^{\mathcal{A}}(\overrightarrow{a}) \in R^{\mathcal{A}}$ on voimassa.
- $\mathcal{A} \vDash \neg\varphi(\overrightarrow{a})$ pätee, jos ja vain jos $\mathcal{A} \vDash \varphi(\overrightarrow{a})$ ei pidä paikkaansa.
- $\mathcal{A} \vDash \varphi(\overrightarrow{a}) \vee \psi(\overrightarrow{a})$ pätee, jos ja vain jos $\mathcal{A} \vDash \varphi(\overrightarrow{a})$ tai $\mathcal{A} \vDash \psi(\overrightarrow{a})$ pitää paikkansa.
- Olkoon $\psi(\overrightarrow{x}) \equiv \exists y\varphi(y, \overrightarrow{x})$. Tällöin $\mathcal{A} \vDash \psi(\overrightarrow{a})$ pätee, jos ja vain jos $\mathcal{A} \vDash \varphi(a', \overrightarrow{a})$ pätee jollakin $a' \in A$.

Sanotaan, että struktuuri $\mathcal{A}$ on kaavan φ malli, jos kaava $\varphi(\overrightarrow{x})$ toteutuu struktuurissa $\mathcal{A}$ jollakin jonolla $\overrightarrow{a} \in A^n$. Jonosta $\overrightarrow{a}$ voidaan puhua tällöin myös tulkintajonona. Koska lauseissa ei ole vapaita muuttujia, merkitään $\mathcal{A} \vDash \Phi$, jos struktuuri $\mathcal{A}$ on lauseen Φ malli.

3 Ehrenfeuchtin ja Fraïssén peli

3.1 Samanlaisuus

Logiikassa on olemassa lukuisia samanlaisuuden käsitteitä, jotka viittavat asoiden yhtäläisyyksiin hieman eri näkökulmista. Ehrenfeuchtin ja Fraïssén pelin ymmärtämisen kannalta erityisen olennainen samanlaisuuden käsite on isomorfismi, jolla viitataan struktuurien alkioiden välisten suhteiden samanlaisuuten eli struktuurien rakenteelliseen yhtäläisyyteen. Jos struktuurista $\mathcal{A}$ muodostetaan toinen struktuuri $\mathcal{B}$ korvaamalla struktuurin $\mathcal{A}$ alkiot toisilla alkioilla siten, että alkioiden väliset suhteet eivät muutu, struktuurit $\mathcal{A}$ ja $\mathcal{B}$

ovat isomorfiset. Tässä vaiheessa esitellään myös toinen samanlaisuuden peruskäsite, elementaarinen ekvivalenssi, joka liittyy EF-pelin hyödyllisyyteen ensimmäisen kertaluvun logiikan ilmaisuvoiman kuvaajana. Elementaarinen ekvivalenssi kuvaa rakenteellisten yhtäläisyyksien sijaan struktuurien samanlaisuutta suhteessa käytettävään kieleen.

Määritelmä 3.1. Olkoot $\mathcal{A}$ ja $\mathcal{B}$ aakkoston σ struktuureita. Kuvaus $h : A \to B$ on isomorfismi struktuurista $\mathcal{A}$ struktuuriin $\mathcal{B}$, jos

(i) h on bijektio;

(ii) jokaiselle $c \in \sigma$ pätee $h(c^{\mathcal{A}}) = c^{\mathcal{B}}$ ja

(iii) jokaiselle k-paikkaiselle relaatiosymbolille $R \in \sigma$ ja jokaiselle jonolle $a_1, \ldots, a_k \in A$ pätee

$$R^{\mathcal{A}} a_1 \ldots a_k, \quad \text{jos ja vain jos} \quad R^{\mathcal{B}} h(a_1) \ldots h(a_k).$$

Jos on olemassa isomorfismi struktuurista $\mathcal{A}$ struktuuriin $\mathcal{B}$, käytetään merkitään $\mathcal{A} \cong \mathcal{B}$ ja sanotaan, että $\mathcal{A}$ ja $\mathcal{B}$ ovat isomorfiset.

Isomorfia säilyttää totuuden. Täten, jos σ-struktuurit $\mathcal{A}$ ja $\mathcal{B}$ ovat isomorfiset kuvauksen h suhteen, kaikille σ-lauseille Φ pätee $\mathcal{A} \vDash \Phi$, jos ja vain jos $\mathcal{B} \vDash \Phi$. Yleisemmin kaikille σ-kaavoille φ ja tulkintajonoille $a_1, \ldots, a_n$ pätee $\mathcal{A} \vDash \varphi(a_1, \ldots, a_n)$, jos ja vain jos $\mathcal{B} \vDash \varphi(h(a_1), \ldots, h(a_n))$. Katso [2, pp. 38-39].

Määritelmä 3.2. σ-struktuurien $\mathcal{A}$ ja $\mathcal{B}$ sanotaan olevan *elementaarisesti ekvivalentit*, jos kaikille σ-lauseille Φ pätee $\mathcal{A} \vDash \Phi$, jos ja vain jos $\mathcal{B} \vDash \Phi$. Merkitään tällöin $\mathcal{A} \equiv \mathcal{B}$.

Koska isomorfismi säilyttää totuuden, mielivaltaiset isomorfiset struktuurit $\mathcal{A}$ ja $\mathcal{B}$ ovat elementaarisesti ekvivalentit. Sen sijaan elementaarisesti ekvivalentit struktuurit eivät välttämättä ole isomorfiset [2, pp. 39-40]. Eherenfeuchtin ja Fraïssén pelin käyttökelpoisuus pohjautuu samanlaisuuteen, joka sijoittuu merkitykseltään jonnekin isomorfismin ja elementaarisen ekvivalenssin välimaastoon. Tätä varten tarvitaan isomorfismia lievempi ehto: osittainen isomorfismi. Esitetään määritelmä osittain Libkiniä [5, p. 27] mukaillen.

Määritelmä 3.3. Olkoot $\mathcal{A}$ ja $\mathcal{B}$ aakkoston σ struktuureita. Kuvaus p on *osittainen isomorfismi* struktuurista $\mathcal{A}$ struktuuriin $\mathcal{B}$, jos

(i) $\mathrm{dom(p)} \subset A$ ja $\mathrm{rg(p)} \subset B$;

(ii) p on injektio;

(iii) kaikille aakkoston σ vakiosymboleille c ja jokaiselle $a \in \mathrm{dom(p)}$ pätee

$$c^{\mathcal{A}} = a, \quad \text{jos ja vain jos} \quad c^{\mathcal{B}} = p(a);$$

(iv) kaikille aakkoston σ k-paikkaisille relaatiosymboleille R ja jokaiselle jonolle $a_1, \ldots, a_k \in \mathrm{dom(p)}$ pätee

$$R^{\mathcal{A}} a_1, \ldots, a_k, \quad \text{jos ja vain jos} \quad R^{\mathcal{B}} p(a_1), \ldots, p(a_k);$$

Käytetään merkintää $\mathrm{part}(\mathcal{A}, \mathcal{B})$ struktuurien $\mathcal{A}$ ja $\mathcal{B}$ välisille osittaisille isomorfismeille. Peliteoreettisissa tarkasteluissa on joskus luontevaa puhua parin $(\overrightarrow{a}, \overrightarrow{b})$ määrittelemästä osittaisesta isomorfismista, missä $\overrightarrow{a} = \mathrm{dom(p)}$ ja $\overrightarrow{b} = \mathrm{rg(p)}$.

Sisältäköön aakkosto σ ainoastaan relaatiosymboleita. Tällöin aakkoston σ struktuureita tarkasteltaessa osittainen isomorfismi säilyttää atomikaavojen totuuden. Olkoon $\varphi(x_1, \ldots, x_n)$ aakkoston σ mielivaltainen atomikaava ja olkoon struktuuri $\mathcal{A}$ kaavan $\varphi(x_1, \ldots, x_n)$ malli tulkintajonolla $\overrightarrow{a} \in A^n$. Olkoon kuvaus p osittainen isomorfismi struktuurista $\mathcal{A}$ struktuuriin $\mathcal{B}$. Tällöin on voimassa $\mathcal{A} \vDash \varphi(a_1, \ldots, a_n)$, jos ja vain jos $\mathcal{B} \vDash \varphi(p(a_1), \ldots, p(a_n))$ pitää paikkansa. Ekvivalenssi ei päde, jos aakkosto sisältää vakio- tai funktiosymboleita. Toisaalta, vaikka aakkosto σ sisältäisi vain relaatiosymboleita, osittainen isomorfismi ei säilyttäisi kvanttorillisten kaavojen totuutta [2, pp. 181-182].

3.2 Ehrenfeuchtin ja Fraïssén pelin kulku

Olkoon r positiivinen kokonaisluku ja olkoot $\mathcal{A}$ ja $\mathcal{B}$ relationaalisen aakkoston σ struktuureita. Struktuurien $\mathcal{A}$ ja $\mathcal{B}$ väliselle r-kierroksiselle EF-pelille käytetään merkintää $\mathrm{EF}_r(\mathcal{A}, \mathcal{B})$. Peli pelataan kahden pelaajan välillä, joista pelaaja I pyrkii osoittamaan, että struktuurit $\mathcal{A}$ ja $\mathcal{B}$ ovat erilaiset, kun

pelaaja II puolustaa struktuurien samanlaisuutta. Jokainen kierros pelataan noudattamalla seuraavia vaiheita:

(1) Pelaaja I valitsee joko alkion a_i stuktuurista $\mathcal{A}$ tai alkion b_i struktuurista $\mathcal{B}$, missä i merkitsee kierroksen lukua.

(2) Pelaaja II valitsee alkion jäljelle jääneestä struktuurista. Jos pelaaja I valitsee alkion $a_i \in A$, pelaaja II valitsee alkion $b_i \in B$ tai toisinpäin.

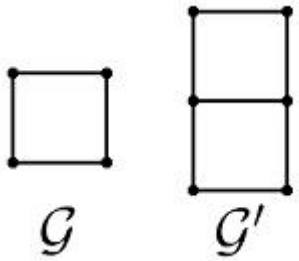
Pelaajan II on suoritettava valintansa siten, että jokaisella kierroksella i valinnoista $a_1, \ldots, a_i$ ja $b_1, \ldots, b_i$ muodostuva pari $(\overrightarrow{a}, \overrightarrow{b})$ määrittelee osittaisen isomorfismin struktuurien $\mathcal{A}$ ja $\mathcal{B}$ välillä. Tällöin alkio a_j kuvautuu alkiolle b_j kaikilla arvoilla $j = 1, \ldots, i$. Ellei pelaaja II pysty suorittamaan valintaansa näillä ehdoilla, peli päättyy pelaajan I voittoon. Jos pelaaja II pystyy suorittamaan r valintaa siten, että pelin päätyttyä pari $(\overrightarrow{a}, \overrightarrow{b})$ määrittelee osittaisen isomorfismin struktuurien $\mathcal{A}$ ja $\mathcal{B}$ välillä, pelaaja I häviää.

Strategia on joukko sääntöjä, joiden mukaan pelaaja suorittaa valintansa vastapuolen valinnoista riippuen. Strategiaa, jota noudattamalla toinen pelaaja voittaa huolimatta toisen pelaaja siirroista, sanotaan *voittostrategiaksi*. Vain yhdellä pelaajalla voi olla voittostrategia kerrallaan. Jos pelaajalla II on voittostrategia r-kierroksisessa pelissä, merkitään $\mathcal{A} \sim_r \mathcal{B}$.

Esimerkki 3.1. Jos pelaaja II tietää, että struktuurien $\mathcal{A}$ ja $\mathcal{B}$ välillä on olemassa isomorfismi $h : A \to B$, seuraavaa strategiaa noudattamalla hän voittaa huolimatta pelin pituudesta tai pelaajan I valinnoista.

($*$) Jos pelaaja I valitsee vuorollaan alkion $a \in \mathcal{A}$, pelaaja II valitsee seuraavaksi alkion $h(a)$. Jos pelaaja I valitsee alkion $b \in \mathcal{B}$, pelaaja II valitsee alkion $h^{-1}(b)$.

Esimerkki 3.2. Olkoot $\mathcal{G}$ ja $\mathcal{G}'$ kuvan 1 mukaisia verkkoja. Pelaajalla II on voittostrategia struktuurien $\mathcal{G}$ ja $\mathcal{G}'$ välisessä kaksikierroksisessa EF-pelissä, eli $\mathcal{G} \sim_2 \mathcal{G}'$. Pelaaja II voittaa pelin valitsemalla solmut siten, että valinnoille g_1 ja g_2 on voimassa $(g_1, g_2) \in E^{\mathcal{G}}$, jos ja vain jos valinnoille g_1' ja g_2' on voimassa $(g_1', g_2') \in E^{\mathcal{G}'}$. Sen sijaan kolmekierroksisessa pelissä pelaajalla I on voittostrategia, eli $\mathcal{G} \nsim_3 \mathcal{G}'$. Pelaaja I voittaa valitsemalla kolme solmua verkosta $\mathcal{G}'$ siten, että mikään näistä solmuista ei ole relaatiossa toisiin valittuihin solmuihin.



Kuva 1

Apulause 3.1. *Olkoot $\mathcal{A}$ ja $\mathcal{B}$ σ-struktuureita. Relaatiolla $\sim_r$ on muun muassa seuraavat ominaisuudet:*

(1) Jos struktuurit $\mathcal{A}$ ja $\mathcal{B}$ ovat isomorfiset, kaikille luonnollisille luvuille r pätee $\mathcal{A} \sim_r \mathcal{B}$.

(2) Jos $\mathcal{A} \sim_r \mathcal{B}$ pätee, niin myös kaikille lukua r pienemmille luonnollisille luvuille i pätee $\mathcal{A} \sim_i \mathcal{B}$.

(3) Olkoon M relationaalisen aakkoston σ struktuurien luokka. Relaatio $\sim_r$ on ekvivalenssirelaatio struktuuriluokan M yli.

Määritellään pelaajan II yleinen voittostrategia Grunbachin ja muiden tapaan [4, p. 36]. Määritelmä yhdistää EF-pelin Fraïssén algebralliseen hahmotelmaan elementaarisesta ekvivalenssista [2, pp. 185-191]. Samastetaan osittainen isomorfismi p joukkoon $\{(a, p(a))|a \in \text{dom(p)}\}$, jolloin voidaan merkitä $p \subseteq q$, kun q on osittaisen isomorfismin p laajennus.

Määritelmä 3.4. Olkoon r positiivinen kokonaisluku. Pelaajan II voittostrategia pelissä $\text{EF}_r(\mathcal{A}, \mathcal{B})$ on jono $P_0, P_1, \ldots, P_r$, missä jokainen P_i on sellainen ei-tyhjä joukko osittaisia isomorfismeja struktuurista $\mathcal{A}$ struktuuriin $\mathcal{B}$, että

(i) kaikille $i < r$, $p \in P_i$ ja $a \in A$ on olemassa $q \in P_{i+1}$ siten, että $p \subseteq q$ ja $a \in \text{dom(q)}$;

(ii) kaikille $i < r$, $p \in P_i$ ja $b \in B$ on olemassa $q \in P_{i+1}$ siten, että $p \subseteq q$ ja $b \in \text{rg(q)}$.

Kun pelaajalla II on voittostrategia pelissä $\mathrm{EF}_r(\mathcal{A}, \mathcal{B})$, määritelmä 3.4 takaa, että jokaisella kierroksella i on olemassa ei-tyhjä osittaisten isomorfismien joukko struktuurista $\mathcal{A}$ struktuuriin $\mathcal{B}$. Osittaisten isomorfismien ominaisuus (i) takaa, että pelaaja II löytää sopivan alkion struktuurista $\mathcal{B}$ silloin, kun pelaaja I valitsee alkion struktuurista $\mathcal{A}$. Ominaisuuden (ii) ansiosta pelaaja II löytää sopivan alkion struktuurista $\mathcal{A}$, jos pelaaja I valitsee alkion struktuurista $\mathcal{B}$.

4 Määriteltävyys

Klassisen malliteorian tutkijoiden kielteinen suhtautuminen äärellisiin struktuureihin on jossain määrin ymmärrettävää, sillä määriteltävyyden kannalta ensimmäisen kertaluvun logiikka ei käyttäydy äärellisten struktuurien parissa parhaalla mahdollisella tavalla. Toisaalta FO-logiikka on ilmaisuvoiman suhteen heikko, toisaalta liiankin voimakas. Voidaan todistaa, että jokainen äärellinen struktuuri on mahdollista kuvata vain yhdellä FO-lauseella [1, p. 13]. Lisäksi mikä tahansa äärellisten struktuurien luokka voidaan kuvata äärellisellä määrällä FO-lauseita [1, p. 14]. Tästä syystä struktuurien ominaisuuksien määriteltävyyttä tutkitaan struktuuriluokissa ja ominaisuuden määriteltävyys pyritään osoittamaan vain yhdellä FO-lauseella.

4.1 Kyselyt

Äärellisten mallien teorialle tyypillinen tapa on tarkastella ominaisuuksien määriteltävyyttä kyselyillä. Annetaan kyselylle loogisiin tarkasteluihin soveltuva täsmällinen määritelmä [4, p. 29]. Olkoon M aakkoston σ struktuuriluokka siten, että jos struktuuri $\mathcal{A}$ kuuluu struktuuriluokkaan M ja struktuuri $\mathcal{B}$ on isomorfinen struktuurin $\mathcal{A}$ kanssa, niin myös struktuuri $\mathcal{B}$ kuuluu luokkaan M. Toisin sanoen struktuuriluokka M on suljettu isomorfismin suhteen.

Määritelmä 4.1. Struktuuriluokan M k-paikkainen *kysely* on sellainen kuvaus Q, että

- $\mathrm{dom(Q)} = M$;

- $Q(\mathcal{A})$ on k-paikkainen relaatio struktuurista $\mathcal{A} \in M$;
- jos kuvaus h määrittelee isomorfismin $\mathcal{A} \cong \mathcal{B}$, niin silloin pätee $Q(\mathcal{B}) = h(Q(\mathcal{A}))$.

Struktuuriluokan M *Boolen kysely* määritellään sellaiseksi kuvaukseksi $Q : M \to \{1, 0\}$, että jos struktuurit $\mathcal{A}$ ja $\mathcal{B}$ ovat isomorfiset, niin silloin pätee $Q(\mathcal{A}) = Q(\mathcal{B})$. Boolen kysely voidaan liittää struktuuriluokan M aliluokkaan M', jolle pätee

$$\mathcal{A} \in M', \quad \text{jos ja vain jos} \quad Q(\mathcal{A}) = 1.$$

Tietojenkäsittelytieteessä kyselyillä viitataan eräänlaisiin muutos- ja hakukäskyihin, joilla tietokannan tietoihin päästään käsiksi. Määritelmän 4.1 kyselyä ja tietokantojen kyselyä ei ole syytä sekoittaa toisiinsa, vaikka yhteys kyselyiden välillä onkin. Tietokantojen hakukäsky ja kysely Q vastaavat merkitykseltään toisiaan. Sekä tietokantojen hakukäskyt että kyselyt Q antavat tietoja tarkasteltavasta struktuurista.

Esimerkki 4.1. [4, pp. 29, 31] Annetaan esimerkkejä verkon $\mathcal{G}$ kyselyistä.

- Jos halutaan poimia ne solmut, joilla on vähintään kaksi erillistä naapuria, esitetään yksipaikkainen kysely

  $$NE(\mathcal{G}) = \{g_1 \in G | \text{solmulla } g_1 \text{ on vähintään kaksi erillistä naapuria}\}.$$

- Jos halutaan poimia ne solmut, joiden välillä on kahden askeleen mittainen polku, esitetään kaksipaikkainen kysely

  $$PA(\mathcal{G}) = \{(g_1, g_2) \in G^2 | \text{on olemassa kahden askeleen mittainen polku solmujen } g_1 \text{ ja } g_2 \text{ välillä}\}.$$

- Verkon $\mathcal{G}$ universumin parillisuus voidaan selvittää Boolen kyselyllä

  $$EVEN(\mathcal{G}) = \begin{cases} 1, \text{ jos verkossa } \mathcal{G} \text{ on parillinen määrä solmuja} \\ 0 \text{ muussa tapauksessa} \end{cases}.$$

- Verkon $\mathcal{G}$ yhtenäisyys voidaan selvittää Boolen kyselyllä

  $$CONN(\mathcal{G}) = \begin{cases} 1, \text{ jos verkko } \mathcal{G} \text{ on yhtenäinen} \\ 0 \text{ muussa tapauksessa} \end{cases}.$$

Koska kyselyllä Q on mahdollista formalisoida struktuurien ominaisuuksia ja myös struktuurien alkioiden ominaisuuksia, voidaan ensimmäisen kertaluvun logiikan ilmaisuvoimaa tutkia kyselyiden avulla. Tätä varten on tiedettävä, mitä kyselyn määriteltävyys FO-logiikassa tarkoittaa.

Määritelmä 4.2. Olkoon M aakkoston σ struktuurien luokka.

- Struktuuriluokan M k-paikkainen kysely Q on FO-määriteltävä, jos on olemassa sellainen FO-kaava $\varphi(x_1, \ldots, x_k)$, että jokaiselle struktuurille $\mathcal{A} \in M$ pätee

$$Q(\mathcal{A}) = \{a_1, \ldots, a_k \in A^k | \mathcal{A} \vDash \varphi(a_1, \ldots, a_k)\}.$$

- Boolen kysely Q on FO-määriteltävä, jos on olemassa sellainen FO-lause Φ, että jokaiselle struktuurille $\mathcal{A} \in M$ pätee

$$Q(\mathcal{A}) = 1, \quad \text{jos ja vain jos} \quad \mathcal{A} \vDash \Phi.$$

Esimerkki 4.2. Seuraavat kyselyt ovat määriteltäviä ensimmäisen kertaluvun logiikassa.

- Esimerkin 4.1 yksipaikkainen kysely $NE(\mathcal{G})$ voidaan määritellä FO-kaavalla

$$\varphi_{NE}(x) \equiv \exists y \exists z(\neg(y = z) \wedge Exy \wedge Exz).$$

- Esimerkin 4.1 kaksipaikkainen kysely $PA(\mathcal{G})$ voidaan määritellä FO-kaavalla

$$\varphi_{PA}(x, y) \equiv \exists z(\neg(x = z) \wedge \neg(y = z) \wedge Exz \wedge Ezy).$$

Esimerkki 4.3. Tarkastellaan esimerkin 2.1 verkkoa $\mathcal{G}$. FO-kaava $\varphi(x, y) \equiv Exy \vee \exists z(Exz \wedge Ezy)$ määrittelee kaksipaikkaisen kyselyn, jonka vastauksena saadan sellaisten parien (a, b) joukko, että kaupungista a on lentoyhteys kaupunkiin b korkeintaan yhdellä välipysähdyksellä. Sen sijaan ei ole olemassa FO-kaavaa, joka määrittelisi kyselyn "Onko mahdollista lentää kaupungista a kaupunkiin b?"

4.2 EF-peli ja FO-logiikan ilmaisuvoima

Osoitetaan, että Ehrenfeuchtin ja Fraïssén peli riittää karakterisoimaan ensimmäisen kertaluvun logiikan ilmaisuvoiman. Lisäksi näytetään, kuinka EF-peliä tyypillisesti sovelletaan määrittelemättömyyden osoittamisessa. Tuloksia varten tarvitaan määritelmä matemaattisen logiikan tärkeästä käsitteestä, joka antaa tietoa FO-kaavojen rakenteesta.

Määritelmä 4.3. Olkoon φ FO-kaava. Määritellään kaavan φ kvanttoriaste $\mathrm{gr}(\varphi)$ induktiivisesti, jolloin

- $\mathrm{gr}(\varphi) = 0$, jos φ on atomikaava;
- $\mathrm{gr}(\varphi_1 \vee \varphi_2) = \max(\mathrm{gr}(\varphi_1), \mathrm{gr}(\varphi_2))$;
- $\mathrm{gr}(\neg\varphi) = \mathrm{gr}(\varphi)$;
- $\mathrm{gr}(\exists \mathrm{x}\varphi) = \mathrm{gr}(\varphi) + 1$.

Käytetään merkintää FO[r] sellaiselle FO-kaavojen joukolle, jossa jokaisen kaavan $\varphi \in \mathrm{FO[r]}$ kvanttoriaste on yhtäsuuri tai pienempi kuin r.

Esimerkki 4.4. [5, p. 33] Kaavajoukko FO[0] koostuu kvanttorittomista kaavoista eli atomikaavojen Boolen kombinaatioista. Oletetaan sitten, että φ on FO[r + 1]-kaava. Jos kaava φ muodostuu kaavojen φ_0 ja φ_1 Boolen kombinaatioista, on kaavojen φ_0 ja φ_1 oltava FO[r + 1]-kaavoja. Jos pätee $\varphi \equiv \exists x\psi(x)$ tai $\varphi \equiv \forall x\psi(x)$, kaava ψ on FO[r]-kaava. Täten jokainen FO[r + 1]-kaava on ekvivalentti muotoa $\exists x\psi(x)$ tai $\forall x\psi(x)$ olevien kaavojen Boolen kombinaatiolle, kun $\psi \in \mathrm{FO[r]}$.

Kvanttoriaste jakaa FO-kaavat pienempiin ja helpommin käsiteltäviin joukkoihin. Tästä syystä kvanttoriasteen avulla voidaan luoda joustavampi elementaarisen ekvivalenssin määritelmä, jonka avulla kahden struktuurin elementaarisen ekvivalenssin arviointi on mahdollista.

Määritelmä 4.4. Olkoon r luonnollinen luku. Sanotaan, että σ-struktuurit $\mathcal{A}$ ja $\mathcal{B}$ ovat *r-ekvivalentit*, $\mathcal{A} \equiv_r \mathcal{B}$, jos kaikille lauseille $\Phi \in \mathrm{FO[r]}$ pätee

$$\mathcal{A} \vDash \Phi, \quad \text{jos ja vain jos} \quad \mathcal{B} \vDash \Phi.$$

Voidaan myös sanoa, että struktuurit $\mathcal{A}$ ja $\mathcal{B}$ ovat elementaarisesti ekvivalentit kvanttoriasteeseen r saakka.

Huomaa, että relaatio $\equiv_r$ on määritelty käyttäen ainoastaan loogisia käsitteitä. Ehrenfeuchtin ja Fraïssén pelin hyödyllisyys perustuu tulokseen, jonka mukaan peliteoreettinen relaatio $\sim_r$ on yhtenevä relaation $\equiv_r$ kanssa. Todistus sivuutetaan (ks. [4, pp. 38-41]).

Lause 4.1 (Ehrenfeucht-Fraïssé)**.** *Olkoon r positiivinen kokonaisluku ja olkoot $\mathcal{A}$ ja $\mathcal{B}$ relationaalisen aakkoston σ struktuureita. Tällöin seuraavat kohdat ovat yhtäpitävät:*

- $\mathcal{A} \equiv_r \mathcal{B}$
- $\mathcal{A} \sim_r \mathcal{B}$.

Apulause 4.2. *Relaatiolla $\sim_r$ on äärellisen monta ekvivalenssiluokkaa. Jokainen relaation $\sim_r$ ekvivalenssiluokka voidaan määritellä yhdellä ensimmäisen kertaluvun lauseella $\Phi \in \mathrm{FO[r]}$.*

EF-peli on eheä ja täydellinen menetelmä ensimmäisen kertaluvun logiikan ilmaisuvoiman kuvaamiseen [4, p. 42]. Lauseeseen 4.1 ja apulauseeseen 4.2 nojautuen saavutetaan tulos, jonka perusteella kyselyn Q määriteltävyys voidaan karakterisoida Ehrenfeuchtin ja Fraïssén pelillä [4, p. 41].

Apulause 4.3. *Olkoon M aakkoston σ struktuurien luokka. Olkoon Q struktuuriluokan M Boolen kysely. Seuraavat kohdat (1) ja (2) ovat yhtäpitävät:*

(1) Kysely Q on FO-määriteltävä.

(2) On olemassa positiivinen kokonaisluku r siten, että kaikille struktuureille $\mathcal{A} \in M$ ja $\mathcal{B} \in M$ on voimassa: jos $Q(\mathcal{A}) = 1$ pitää paikkansa ja pelaajalla II on voittostrategia pelissä $\mathrm{EF}_r(\mathcal{A}, \mathcal{B})$, niin myös $Q(\mathcal{B}) = 1$ pitää paikkansa.

Koska ilmaisuvoiman tarkasteluissa ollaan kiinnostuneita pikemmin määrittelemättömyydestä kuin määriteltävyydestä, muotoillaan apulause 4.3 toisella tavalla.

Apulause 4.4. *Olkoon M aakkoston σ struktuurien luokka. Olkoon Q struktuuriluokan M Boolen kysely. Seuraavat kohdat (1) ja (2) ovat yhtäpitävät:*

(1) Kysely Q ei ole FO-määriteltävä.

(2) Jokaiselle positiiviselle kokonaisluvulle r on olemassa sellaiset struktuurit $\mathcal{A} \in M$ ja $\mathcal{B} \in M$, että pelaajalla II on voitttostrategia pelissä $\mathrm{EF}_r(\mathcal{A}, \mathcal{B})$, jos ja vain jos $Q(\mathcal{A}) = 1$ ja $Q(\mathcal{B}) = 0$ pätee.

Täten Boolen kyselyn Q määrittelemättömyyden todistamiseksi konstruoidaan sellaiset struktuurit $\mathcal{A} \in M$ ja $\mathcal{B} \in M$, että $Q(\mathcal{A}) = 1$ ja $Q(\mathcal{B}) = 0$ ovat voimassa. Tämän jälkeen osoitetaan, että pelaajalla II on voittostrategia pelissä $\mathrm{EF}_r(\mathcal{A}, \mathcal{B})$ kaikille positiivisille kokonaisluvuille r.

Lause 4.5. *Olkoon M äärellisten verkkojen luokka. Esimerkin 4.1 Boolen kysely $EVEN(\mathcal{G})$ ei ole määriteltävissä FO-logiikalla luokassa M.*

Todistus. Olkoon $\mathcal{G}_n$ epäyhtenäinen verkko, jolla on n solmua mutta ei yhtään särmää. Olkoon r positiivinen kokonaisluku, ja olkoot n ja m lukua r suurempia tai yhtäsuuria kokonaislukuja. Pelaajalla II on selvästi voittostrategia verkkojen $\mathcal{G}_n$ ja $\mathcal{G}_m$ välisessä r kierroksisessa EF-pelissä. Riittää, että valitaan parillinen luku n ja pariton luku m, jolloin apulauseen 4.4 perusteella väite pitää paikkansa [4, p. 42]. □

Palataan jälleen esimerkkiin 2.1. Halutaan tietää, onko mielivaltaisesta kaupungista yhteys kaikkiin kaupunkeihin välipysähdysten määrästä huolimatta. Toisin sanoen halutaan tietää, onko kaupunkien lentoyhteyksien tietokantaa kuvaava verkko yhtenäinen. Ensimmäisen kertaluvun logiikassa tämä ei kuitenkaan päde, sillä kysely $CONN(\mathcal{G})$ ei ole FO-määriteltävä.

Lause 4.6. *Olkoon M äärellisten verkkojen luokka. Esimerkin 4.1 Boolen kysely $CONN(\mathcal{G})$ ei ole määriteltävissä FO-logiikalla luokassa M.*

Todistus. Hahmotellaan todistus tyypillisellä EF-menetelmällä [3]. Oletetaan, että FO-lause Φ määrittelee Boolen kyselyn $CONN(\mathcal{G})$. Olkoon r lauseen Φ kvanttoriaste. Konstruoidaan sellaiset struktuurit $\mathcal{G}_1$ ja $\mathcal{G}_2$, että $CONN(\mathcal{G}_1) = 1$ ja $CONN(\mathcal{G}_2) = 0$.

Kun vähintään kolmen askeleen mittaisen polun alku- ja loppusolmu on sama, kutsutaan polkua *sykliksi*. Koostukoon verkko $\mathcal{G}_1$ syklistä S, jossa on $2n$ solmua. Olkoon verkossa $\mathcal{G}_2$ kaksi erillistä sykliä S_1 ja S_2, joissa kummassakin on n solmua. Riittävän suurella luvulla n - luvusta r riippuen - voidaan pelaajalle II määrittää voittostrategia.

On osoitettava, että pelaaja I ei hyödy verkon $\mathcal{G}_2$ epäyhtenäisyydestä. Itse asiassa toisistaan kaukana olevat verkon $\mathcal{G}_1$ solmut a ja a' ovat suhteeltaan samankaltaiset kuin verkon $\mathcal{G}_2$ solmut $b \in S_1$ ja $b' \in S_2$. Jotta pelaaja I hyötyisi verkon $\mathcal{G}_2$ epäyhtenäisyydestä, olisi hänen muodostettava polku solmusta a solmuun a', mitä pelaaja II ei pysty solmujen b ja b' välille muodostamaan. Tällaisen polun muodostaminen vaatisi, että pelattavassa EF-pelissä olisi enemmän kierroksia kuin r.

Oletetaan, että pelaaja I valitsee solmun $a_1 \in \mathcal{G}_1$. Pelaaja II vastaa valitsemalla mielivaltaisen solmun $b_1 \in S_1$. Jos pelaaja I valitsee seuraavaksi solmun $b_2 \in S_2$, pelaaja II valitsee solmun $a_2 \in \mathcal{G}_1$, joka on kaukana solmusta a_1. Nyt jos pelaaja I valitsee solmun $b_3 \in S_1$, pelaaja II valitsee solmun $a_3 \in \mathcal{G}_1$ läheltä solmua a_1. Kun luku n on riittävän suuri, tätä voidaan jatkaa r kierrosta siten, että valinnat $(\overrightarrow{a}, \overrightarrow{b})$ määrittelee osittaisen isomorfismin struktuurien $\mathcal{G}_1$ ja $\mathcal{G}_2$ välillä. Täsmällinen todistus vaatisi kaikkien valintamahdollisuuksien käsittelyä (ks. [3, pp. 3-4]). □

5 Yhteenveto

Äärellisten mallien teorian määriteltävyystuloksia on mielenkiintoista tarkastella tietokantateorian näkökulmasta. Koska kyselykielet pohjautuvat ensimmäisen kertaluvun logiikkaan ja relationaalisilla struktuureilla voidaan kuvata relaatiomallin mukaisia tietokantoja, määriteltävyystuloksia on helppo havainnollistaa relaatiomallin mukaisiin tietokantoihin viitaten. Ehrenfeuchtin ja Fraïssén peli on osoittautunut erinomaiseksi menetelmäksi karakterisoida FO-logiikan ilmaisuvoimaa. Peliä hyödyntäen on mahdollista osoittaa, että monet tietokannoillekin tyypilliset rakenteelliset ominaisuudet eivät ole määriteltävissä FO-logiikalla. Vaikka ensimmäisen kertaluvun logiikalla on ylivoimainen asema logiikoiden joukossa, äärellisten struktuurien parissa sen

ilmaisuvoima on osoittautunut melko heikoksi. Tästä syystä kyselykieliin on lisätty uusia ominaisuuksia, joiden myötä kielien ilmaisuvoimaa on kyetty parantamaan. Kyselykielten ja ensimmäisen kertaluvun logiikan yhteyttä on kuvattu erinomaisesti Vardin ja muiden luentomateriaalissa *Logic and Database Queries* [7]. Kyseisessä lähteessä esitellään havainnollisesti SQL-kielen kehitys ensimmäisen kertaluvun logiikasta.

Viitteet

[1] Ebbinghaus, H.-D., Flum, J.: *Finite Model Theory*. Springer, New-York (1999).

[2] Ebbinghaus, H.-D., Flum, J., Thomas, W.: *Mathematical Logic*. Springer, New York (1984).

[3] Ehrenfeucht-Fraïssé Games.
`http://www-cse.ucsd.edu/classes/sp08/cse205a/games.pdf`.
Checked 2008-10-23.

[4] Grädel, E., Kolaitis, P.G,. Libkin, L., Marx, M., Spencer, J., Vardi, M.Y., Venema, Y., Wenstein, S.: *Finite Model Theory and its Applications*. Springer, New York (2007).

[5] Libkin, L.: *Elements of Finite Model Theory*. Springer, New York (2004).

[6] Nieminen, P.: *Ehrenfeuchtin ja Fraïssén peli*. Tekeillä oleva pro gradu -tutkielma, Matematiikan ja tilastotieteen laitos, Tampereen yliopisto (2008).

[7] Vardi, M.Y., Barland, I., McMahan, B.: Logic and Database Queries (2006).
`http://www.cs.rice.edu/~tlogic/Database/all-lectures.pdf`.
Checked 2008-10-23.

Tietokonemadot

Niina Ojala

Tiivistelmä Tässä tutkielmassa esitellään erilaisia haittaohjelmia kuten matoja, viruksia, troijalaisia hevosia ja botteja. Pääpaino on kuitenkin tietokonemadoissa, joiden erilaisia luokitteluja tarkastellaan. Tutkielmassa esitellään myös kolmea kuuluisaa tietokonematoa. MyDoom valittiin, koska se oli nopeimmin leviävä mato vuonna 2004. Sasser taas pystyi aiheuttamaan vahinkoa eräiden tärkeiden yritysten toimintaan. Zotop edustaa uusimpia matoja.

Avainsanat: Haittaohjelma, tietokonevirus, troijalainen hevonen, botti, tietokonemato
CR-luokat: D.4.6

1 Johdanto

Tässä tutkielmassa esitellään tietokonematoja. Madot sekoitetaan usein tietokoneviruksiin, vaikka nämä käsitteet eivät olekaan synonyymeja. Tutkielman toisena tarkoituksena on esitellä matojen luokittelutapoja, joita on useita erilaisia. Tutkielmassa kerrotaan myös eräistä kuuluisista madoista. MyDoom oli kuuluisa siitä, että se oli nopeimmin leviävä mato vuonna 2004. Sasser taas pystyi aiheuttamaan vahinkoa eräiden tärkeiden yritysten toimintaan. Zotop puolestaan edustaa uusimpia matoja.

Tutkielmassa keskitytään erityisesti kertomaan tietokonemadoista, mutta ei niinkään niiden torjumisesta. Matojen ja muiden haittaohjelmien torjumisesta on paljon tietoa saatavilla, minkä vuoksi tässä tutkielmassa keskitytään pelkästään tietokonematoihin ja niistä kertomiseen, mutta niiden torjuntamenetelmät sivuutetaan.

Luvussa 2 käsitellään erilaisia haittaohjelmia kuten viruksia, matoja, troijalaisia hevosia ja botteja. Luvussa 3 esitellään matojen luokitteluja kohteen löytämisen, kuljettajan, aktivoimisen ja matojen aiheuttaman kuorman suhteen. Luvussa 4 kerrotaan kolmesta kuuluisasta madosta, jotka ovat MyDoom, Sasser ja Zotop. Luvussa 5 on yhteenveto koko tutkielmasta.

2 Haittaohjelmat

Haittaohjelma on yläkäsite kaikelle ei-halutulle koodille. Yksi määritelmä haittaohjelmalle voisi olla, että se on asennettu järjestelmään ilman järjestelmän ylläpitäjän lupaa. Se tekee järjestelmässä sellaisia toimintoja, joita järjestelmän ylläpitäjä ei hyväksyisi, jos tietäisi niitä siellä suoritettavan. Toisenlainen määritelmä haittaohjelmalle voisi olla, että yritys on halukas maksamaan tällaisen koodin poistosta tai estämään tällaisten koodien tunkeutumisen sen järjestelmiin. [5]

Ensimmäinen määritelmä haittaohjelmalle on hyvä, koska se tarkoittaa sitä, että haittaohjelma on jollakin tavalla haitallinen järjestelmän toiminnalle eikä haittaohjelman toimintaa voida hyväksyä. Toinen määritelmä ei sen sijaan ole kovinkaan hyvä. Se antaa ymmärtää, että ohjelma, jonka poistosta yritys ei ole valmis maksamaan, ei ole haitallinen järjestelmälle. Voisihan olla, että yritys ei ole kiinnostunut poistamaan jotain ohjelmaa, jos se ei aiheuta kovinkaan paljoa haittaa järjestelmälle. Tällaisia ohjelmia voisivat olla esimerkiksi erilaiset mainosohjelmat, jotka arvioivat käyttäjän toimintaa ja sen perusteella näyttävät nettiselaimessa juuri käyttäjälle kohdistettuja mainoksia. Mitenkään haitallisia ne eivät ole, mutta harvemmin kukaan antaa mainosohjelmalle luvan tarkkailla omaa toimintaa.

Yleensä kaikki virukset, madot, botit ja troijalaiset hevoset ovat tietynlaisia haittaohjelmia. On tietysti mahdollista, että joku näistä ei olisi haittaohjelma. Tämä olisi mahdollista, jos se ei aiheuttaisi haittaa järjestelmälle eikä suorittaisi haitallisia toimintoja siellä. Suurinta osaa viruksista ja madoista pidetään kuitenkin haittaohjelmina, koska ne aiheuttavat jollakin tavalla haittaa järjestelmälle.

2.1 Virus

Virus on haittaohjelma, joka saastuttaa olemassa olevan tiedoston tai ohjelman. Se tekee niihin muutoksia, jolloin niistä tulee haitallisia. Virus ei voi levitä ilman, että se tekee muutoksia olemassa oleviin tiedostoihin tai ohjelmiin. [5] Virus tarvitsee yleensä myös käyttäjän apua leviämisessä ja itsensä kopioimisessa [6]. Käyttäjän

tarvitsee siis lähettää edelleen viruksen saastuttama tiedosto toisille käyttäjille, jotta virus pystyisi leviämään.

Viruksen leviäminen on hidasta [12]. Tämä johtuu siitä, että virus tarvitsee käyttäjän apua levittämiseen. Tällöin viruksen leviämisvauhti on riippuvaista siitä, miten nopeasti käyttäjät lähettävät saastuneen tiedoston toisille käyttäjille ja kuinka monelle käyttäjälle he lähettävät tällaisen tiedoston.

2.2 Mato

Madolla tarkoitetaan haittaohjelmaa, joka leviää internetissä ilman ihmisen apua tai ihmisen avustuksella. Se voi tehdä itsestään kopioita ja tällä tavoin levittää itseään. Se voi myös käyttää hyväkseen sivuistoilla olevia tietoturva-aukkoja kohteensa saastuttamisessa. [6] Mato pystyy käyttämään useita erilaisia keinoja leviämisessä ja tietokoneen saastuttamisessa. Tämä tekee madosta hyvin vaarallisen ja tuhoisan.

Mato toisin kuin virus ei tarvitse saastutettavaa tiedostoa, vaan voi levitä itsenäisesti ilman ihmisen apua [5]. Madon ja viruksen määritelmät ovat hyvin lähellä toisiaan ja ne sekoitetaan helposti keskenään. Virus kuitenkin tarvitsee olemassa olevan tavallisen tiedoston, johon se tekee muutoksia ja tällä tavoin tekee siitä haittaohjelman. Mato taas voi levitä ilman tällaista tiedostoa ja se tarvitsee yleensä vähemmän käyttäjän apua leviämisessä. Mato voi myös levitä täysin ilman käyttäjän apua.

2.3 Troijalainen hevonen

Troijalainen hevonen on ohjelma, joka näyttää tarjoavan hyödyllisiä toimintoja käyttäjälle. Se sisältää kuitenkin piilotettuja ja haitallisia toimintoja. Käyttäjä huijataan käynnistämään hyödylliseksi luultu ohjelma, joka kuitenkin sisältää haitallista koodia. Tämä koodi aktivoituu, kun ohjelma on käynnistetty. Troijalaiset hevoset eivät kopioi itseään toisin kuin virukset ja madot. [6]

Matoihin ja viruksiin verrattuna troijalainen hevonen on selkeästi erilainen haittaohjelma, koska se tarjoaa hyödyllisiä toimintoja, jolla houkutellaan asentamaan ja ajamaan troijalaisen sisältämä ohjelma. Madot ja viruksethan pyrkivät käyttäjältä salaa tunkeutumaan järjestelmään, kun taas troijalainen hevonen esittää olevansa tarpeellinen ohjelma ja tekee järjestelmään tunkeutumisen käyttäjän luvalla.

2.4 Botti

Botti on haittaohjelma, joka suorittaa tietyn toiminnallisuuden. Hyökkääjä voi käyttää sitä omien tavoitteidensa saavuttamiseksi käyttämällä isäntäohjelmaa, jolla botin toimintaa voidaan kontrolloida. Botit eivät levitä itseään, mutta viruksia ja matoja voidaan käyttää niiden asentamiseksi järjestelmään. [9] Huomattavaa on, että erilaisia haittaohjelmia voidaan käyttää yhdessä saastuttamaan järjestelmä kuten bottien asentamisessa järjestelmään viruksen avulla. Käyttäjä voi poistaa viruksen, mutta häneltä saattaa jäädä bottiohjelma huomaamatta. Tällöin botti vain odottelee käyttäjän tietokoneelle hetkeä, jolloin hyökkääjä ottaa siihen yhteyttä ja alkaa käyttää järjestelmää omiin tarkoituksiinsa.

Botti varastaa käyttäjän nettiyhteyden ja identiteetin. Niitä käytetään suurina joukkoina hyökkäämään tiettyyn kohteeseen, jolloin ne voivat aiheuttaa suurempia tuhoja kuin yksittäiset haittaohjelmat. Botteja käyttävät pääasiassa järjestäytyneet rikolliset. He voivat uhkailla yrityksiä niiden hyökkäyksillä ja vaatia rahaa, jotta yritys säästyisi tällaiselta hyökkäykseltä. [5] Botit eroavat muista tässä tutkielmassa esitetyistä haittaohjelmista siinä, että niitä käytetään suurina joukkona hyökkäämään tiettyyn kohteeseen. Hyökkääjä pystyy myös kontrolloimaan niitä niiden vapaaksi päästämisen jälkeenkin.

3 Matojen luokittelua

Weaver ja kumppanit [12] esittelevät viisi erilaista matojen jaottelua, jotka perustuvat seuraaviin seikkoihin: kohteen löytäminen, kuljettaja, aktivoituminen, kuorma ja hyökkääjät. Mato voi kuulua useaan eri luokkaan riippuen siitä, millä tavoin se halutaan luokitella. Tässä luvussa esitellään kaikki muut paitsi hyökkääjät-luokittelu,

koska se ei liity itse matoihin, vaan kertoo niistä syistä, miksi jotkut kirjoittavat matoja.

3.1 Kohteen löytäminen

Madoilla on erilaisia keinoja löytää saastutettava kohteensa. Madot voivat etsiä uhrejaan tutkaamalla haavoittuvia tietokoneita. Ne voivat käyttää apunaan hyökkääjän luomaa listaa mahdollisesti haavoittuvista kohteista ja yrittää saastuttaa nämä kohteet. Madot voivat myös käyttää hyväkseen muiden tekemiä kohdelistoja kuten esigeneroitujen tai ulkopuolisen generoimien kohdelistojen tapauksessa. On myös mahdollista, että madot ovat passiivisia ja odottavat käyttäjän tulevan luokseen ja lataavan ne koneelleen.

Tapoja löytää kohde tutkaamalla on kaksi. Se on joko peräkkäistä tai mielivaltaista. Ensimmäisessä tapauksessa mato käy läpi osoitelohkon osoitteet järjestyksessä, kun taas jälkimmäisessä tapauksessa mato käy läpi osoitelohkon mielivaltaisessa järjestyksessä. Tutkaaminen on hyvin yleinen tapa löytää kohde. Mato etsii kohteistaan tunnettuja tietoturva-aukkoja.[12] Löydettyään tietoturva-aukon mato yrittää saastuttaa tietokoneen.

Tutkaamista rajoittaa kuitenkin haavoittuvien koneiden tiheys. Lisäksi virustentorjuntaohjelmat tunnistavat tällaisen käyttäytymisen helposti, koska se eroaa paljon normaalista tietoliikenteestä [12]. Nykyään virustorjuntaohjelmat ovat melko yleisiä, joten tällä tavalla leviävien matojen torjuminen on suhteellisen helppoa. Jotta tutkaamalla kohteensä löytävä mato voisi levitä, tulisi sen löytää ensin haavoittuva tietokone, jossa ei ole sitä tunnistavaa virustentorjuntaohjelmaa. Tämä voi olla nykyään hankalaa, vaikka vieläkään kaikissa tietokoneissa ei ole asiaan kuuluvia palomuureja tai virustentorjuntaohjelmistoja.

Mato voi löytää kohteensa esi-generoidun kohdelistan avulla. Hyökkääjä luo listan mahdollisista uhreista samalla, kun hän luo madon. Tällainen tapa löytää kohde vaatii suuren vaivan madon kirjoittajalta. [12] Tätä tapaa tuskin käytetään kovinkaan paljoa, jos ollenkaan johtuen suuresta vaivasta madon kirjoitusvaiheessa. Tosin se voi olla

nopea tapa levitä, kun madon ei tarvitse käydä läpi suurta joukkoa tietokoneita etsien haavoittuvuuksia, vaan se voi suoraan yrittää tunkeutua mahdollisten uhrien tietokoneelle. Madon tekeminen on kilpajuoksua virustorjuntaohjelmien ja haavoittuvuuksien paikkaamisen kanssa. Mitä kauemmin madon kirjoittajalta menee matoa luodessa, sitä vähemmän aikaa madolla on etsiä niitä tietoturva-aukkoja, joita ei ole paikattu.

Ulkopuolisen generoiman kohdelistan tapauksessa mato ottaa yhteyden metaserveriin. Metaserveri pitää listaa aktiivisina olevista palvelimista, jotka tarjoavat tiettyä palvelua, esim. pelipalveluita. [12] Tällainen hyökkäys voisi olla vakava, koska pelaajat luultavasti luottavat pelipalvelimeen, eivätkä he välttämättä osaa varautua siihen, että tällaiselta palvelimelta voisi tulla mato. Tällainen hyökkäys voisi olla hyvinkin nopea, jolloin tietoa pelaajille palvelimen saastumisesta ei saataisi tarpeeksi nopeasti, vaan suurin osa pelaajien tietokoneista voisi saastua.

Sisäisiä kohdelistoja hyväksikäyttävät madot etsivät tietokoneelta sovelluksia, jotka ylläpitävät listaa muista isäntätietokoneista, jotka tarjoavat haavoittuvia palveluita. Tällä tavalla leviävät madot eivät aiheuta saastuneen koneen liikenteeseen muutoksia, mutta maailmanlaajuinen tietoliikenne kasvaa. [12]

Passiivisesti kohteensä löytävät madot eivät etsi aktiivisesti uhreja, vaan odottavat käyttäjän ottavan niihin yhteyden. Ne saattavat myös luottaa käyttäjän toimintaan, joka voi auttaa matoa löytämään uusia kohteita. Tämä on hidasta, mutta se ei vaikuta yleiseen tietoliikenteeseen. [12] Tällä tavoin leviävää matoa on hankala huomata, koska se ei luo tietoliikenteeseen muutoksia. Tapa on todellakin hidas, sillä mato joutuu odottelemaan, että joku ottaa siihen yhteyden, mikä voi kestää hyvinkin kauan.

Passiivinen tapa löytää kohde on vähän samanlainen kuin troijalaisten hevosten käyttämä tapa levitä. Troijalaiset hevosethan odottavat, että käyttäjä lataa troijalaisen hevosen sisältämän ohjelman tietokoneelleen. Madot eivät kuitenkaan tarjoa käyttäjilleen hyödyllisiä palveluita ja madot pystyvät luomaan kopioita itsestään, mikä ei ole troijalaisille hevosille mahdollista.

3.2 Kuljettaja

Madot voivat levitä käyttämällä kuljettajana itseään, mutta ne voivat käyttää leviämiseen myös muita tapoja. Madot voivat käyttää toista kanavaa päättääkseen saastuttamisen tai upottaa itsensä normaalin tietoliikenteen sekaan ja mahdollisesti korvata oikeita viestejä väärillä viesteillä.

Itseään kuljettavat madot etsivät aktiivisesti saastutettavia tietokoneita [12]. Tämä on nopea tapa levitä, koska tällaisten matojen leviäminen ei ole riippuvainen käyttäjän toiminnasta, mikä usein on hidasta. Virustentrojuntaohjelmat voivat kuitenkin huomata itseään kuljettavan madon, koska se luo epänormaalia tietoliikennettä.

Madot voivat käyttää kuljettajanaan myös toista kanavaa. Tällöin ne tarvitsevat toisen yhteyskanavan saadakseen saastuttamisen päätökseen. Käyttämällä hyväksi RPC:tä madon saastuttama kone ottaa yhteyttä saastuttajaan TFTP:llä, jolloin madon loppuosa ladataan ja saastuminen on valmis. [12] RPC tarkoittaa sitä käytäntöä, jolla tietojärjestelmä voi kutsua toisessa tietojärjestelmässä olevia ohjelmia [2]. TFTP taas tarkoittaa tiedostojen siirtämisessä käytettävää käytäntöä [10]. Toisen kanavan tapauksessa sisääntulevan madon ei tarvitse olla suuri, koska se voi ladata loppuosan itsestään myöhemmin. Tällä tavoin virustentorjuntaohjelmat eivät huomaa sitä kovinkaan helposti.

Mato voi lähettää itseään normaalin tietoliikenteen mukana lisäten tai korvaten siinä kulkeviä viestejä. Tällaista toimintaa kutsutaan upottamiseksi ja sitä on vaikea huomata [12]. Virustentorjuntaohjelmat ja palomuurit pystyvät huomaamaan suhteellisen helposti poikkeavan tietoliikenteen. Korvaamalle normaalin tietoliikenteen viestejä madot voivat huijata virustentorjuntaohjelmia, koska tietoliikenteen määrä ei kasva. Kuitenkin tietokoneen toiminta voi muuttua ja virheitä ohjelmien toiminnassa voi esiintyä, kun mato on korvannut osan tietokoneen lähettämistä viesteistä. Käyttäjä voi alkaa tällöin epäillä koneen saastuneen. Mato voi lisätä omia viestejään normaalin tietoliikenteen sekaan, jolloin käyttäjä ei osaa epäillä tietokoneensa saastuneen. Kuitenkin virustentorjuntaohjelmat voivat helpommin huomata lisääntyneen tietoliikenteen.

3.3 Aktivoituminen

Löydettyään haavoittuneen tietokoneen matojen täytyy aktivoitua, jotta ne voivat aloittaa toimintansa ja saastuttaa kohteensa. Madot voivat aktivoitua useilla eri tavoilla. Ne voivat tarvita siihen ihmistä tai ihminen voi tehdä jonkin normaalin toiminnon, joka aktivoi madon. Niiden on mahdollista käyttää järjestelmän aikataulutettuja prosesseja tai ne voivat aktivoida itse itsensä.

Ihminen voi aktivoida madon. Tällöin se vaatii käyttäjältä madon sisältävän ohjelman suorittamisen, joka johtaa käyttäjän tietokoneen saastumiseen. Tämä on hidas tapa aktivoitua [12]. Madon kannalta pahimmassa tapauksessa käyttäjä ei käynnistä madon sisältämää ohjelmaa ollenkaan, eikä se pääse koskaan aktivoitumaan. Toisaalta taas, kun käyttäjä aktivoi madon, niin käyttäjä voi antaa palomuurille luvan sallia madon toiminnan. Näin voi käydä, kun käyttäjä luulee itse käynnistäneensä ohjelman ja kun palomuuri kysyy käyttäjältä, onko tämä sallittua, niin käyttäjä voi antaa siihin luvan, varsinkin jos käyttäjä ei ymmärrä palomuurin antamaa ilmoitusta.

Ihminen voi tehdä jonkin normaalin toiminnon, joka aiheuttaa madon aktivoitumisen. Tällainen toiminto voi olla esim. tietokoneen uudelleenkäynnistys. [12] Mato on päässyt jollakin keinolla tietokoneelle ja jokin käyttäjän harmiton toimi voi aktivoida sen. Käyttäjä ei voi mitenkään estää tässä tapauksessa matoa aktivoitumisessa, mutta hyvä virustentorjuntaohjelma voi sen tehdä ja estää matoa saastuttamasta tietokonetta.

Mato voi aktivoitua tietokoneen aikataulutettujen prosessien käynnistyessä. Tällaisia ovat esim. automaattiset ohjelmien päivitykset. [12]. Ainakin Windows-käyttöjärjestelmissä on paljon tällaista toimintaa. Automaattiset ohjelmapäivitykset ovat hyvä asia, sillä ne voivat paikata ohjelmissa esiintyviä tietoturva-aukkoja. Kuitenkin tällaisia päivityksiä voidaan käyttää väärin, varsinkin jos ne on toteutettu huonosti.

Mato voi aktivoitua itsestään. Itsensä aktivoivat madot alustavat itsensä suorittamaan erilaisia toimintoja hyväksikäyttämällä tunnettuja ja aina saatavilla olevia palveluita kuten IIS-verkkopalvelinta.

Tällaiset madot voivat liittää itsensä tähän palveluun tai ne voivat suorittaa komentoja käyttäen näiden palveluiden käyttöoikeuksia. [12]

3.4 Kuorma

Madon kuormalla tarkoitetaan kaikkea muuta kuin mitä mato tarvitsee levitäkseen [12]. Tässä luvussa esitellään muutamia kuormia, joita madoilla voi olla. Niitä voi olla vaikka kuinka paljon riippuen madosta ja sen kirjoittajan mielenkiinnosta. Tässä kohdassa esitellään takaoven avaaminen, tietojen tuhoaminen ja kryptaaminen sekä henkilökohtaisten tietojen etsiminen koneelta.

Tietokonemato voi avata takaoven [12]. Sen kautta järjestelmään voi päästä kuka tahansa tai mikä tahansa ja käyttää järjestelmää ilman, että tietokoneen käyttäjä tietää siitä. Tätä menetelmää käyttäen tietokonetta voidaan käyttää esim. roskapostin lähetykseen. Takaoven hyödyntäjä ei välttämättä ole sama henkilö, joka loi madon, vaan se voi olla myös toinen haittaohjelma.

Madot voivat etsiä henkilökohtaisia tietoja käyttäjästä kuten luottokortin numeroita ja lähettää ne madon tekijälle. [12] Tällöin hän voi hyväksikäyttää luottokortin numeroa. Tästä syystä omalla tietokoneelle ei pitäisi säilyttää mitään tarpeetonta henkilökohtaista tietoa kuten henkilötunnustaan. Jos näitä tietoja ei tietokoneella ole, niin mahdollisen madon hyökkäys ei voi saattaa niitä ulkopuolisten tietoon.

Madot voivat halutessaan tuhota käyttäjän tiedostoja [12]. Käyttäjälle voi aiheutua ongelmia riippuen siitä, kuinka tärkeitä tiedostoja mato on tuhonnut ja onko niistä olemassa varmuuskopioita. Tiedostojen tuhoaminen on aika harvinaista nykyään, mutta se on mahdollista. Matojen kirjoittajat eivät tosin hyödy tästä kovinkaan paljoa, mikä on varmasti vaikuttanut siihen, että tällaisia ominaisuuksia ei paljoa enää esiinny.

Matojen kirjoittajat voivat laittaa madon kryptaamaan käyttäjän tiedostoja. [12] Kryptaaminen tarkoittaa sitä, että tiedostot salakirjoitetaan niin, että ulkopuolinen ei voi niitä lukea. Tämä tapa

on paljon hyödyllisempi kuin tiedostojen tuhoaminen. Käyttäjä ei voi lukea tiedostojaan ja riippuen niiden tärkeydestä, hän voi haluta poistaa kryptauksen. Tällöin madon luoja voi yrittää kiristää käyttäjää maksamaan kryptauksen poistamisesta. Rahat saatuaan madon tekijä ei välttämättä purakaan kryptausta, vaan pitää rahat.

4 Kuuluisia matoja

Tässä luvussa esitellään kolme matoa. Ne ovat MyDoom, Sasser ja Zotop. MyDoom valittiin, koska se oli nopeimmin leviävä mato vuonna 2004. Sasser puolestaan valittiin, koska se pystyi aiheuttamaan vahinkoa eräiden tärkeiden yritysten toimintaan. Zotop valittiin, koska se aiheutti paljon tuhoa, mutta se on myös uusin näistä madoista. Zotop löydettiin vuonna 2005 [3], Sasser vuonna 2004 [11] ja MyDoom niin ikään vuonna 2004 [7].

4.1 MyDoom

MyDoom on mato, joka leviää sähköpostiviestien välityksellä. Se lähettää viestin, jossa se väittää olevansa yrityksen tietyltä tukiosastolta tuleva viesti. Se sisältää myös ajettavan tiedoston ja jos käyttäjä ajaa tiedoston, mato aktivoituu. Tämän jälkeen mato etsii käyttäjän kovalevyltä sähköpostiosoitteita ja lähettää itsensä näihin osoitteisiin. [8]

Weaverin ja kumppaneiden [12] matoluokituksissa MyDoom voisi olla kuljettaja-luokituksen perusteella itsensä kuljettaja, koska se lähettää itseään sähköpostiviesteissä. Aktivointiluokituksen mukaan MyDoom taas voidaan lukea kuuluvan käyttäjän avulla aktivoituviin matoihin, koska se tarvitsee käyttäjää aktivoimaan itsensä, mikä tässä tapauksessa tarkoittaa ajettavan tiedoston ajamista.

Brittiläinen tietoturvayhtiö torjui 100 000 kopiota MyDoom-madosta tunnissa ja 1.8 miljoonaa kopiota siitä 168 eri maassa [7]. MyDoom aiheutti valtavasti tietoliikennettä, vaikka monia sen kopioista pystyttiinkin estämään saastuttamasta kohdettaan. Ei olekaan ihme, että MyDoom nimettiin vuonna 2004 kaikkien aikojen nopeimmin levinneeksi madoksi.

Madoista on liikkeellä erilaisia versioita, jotka usein parantavat alkuperäistä matoa. Provos ja kumppanit [8] kertovat MyDoom.O:sta. Se parantaa alkuperäisen MyDoomin toimintaa siten, että mato etsii sähköpostiosoitteista verkkotunnuksia. Sitten se etsii hakukoneiden avulla verkkotunnusten perusteella lisää sähköpostiosoitteita, joihin se lähettää itsensä. MyDoom.O käyttää neljää erilaista hakukonetta, jotka olivat Google, Yahoo, Lycos ja Altavista.

MyDoom.O on huomattavasti vaarallisempi kuin alkuperäinen MyDoom. Etsimällä erilaisten hakukoneiden avulla lisää sähköpostiosoitteita se aiheutti huomattavaa kuormaa verkkoliikenteessä. Se pystyi myös levittämään itseään paljon nopeammin kuin alkuperäinen MyDoom, koska se haki lisää sähköpostiosoitteita hakukoneiden avulla.

4.2 Sasser

Sasser leviää tietoturva-aukon välityksellä ajettavana tiedostona. Se etsii tietokoneita sattumanvaraisten ip-osoitteiden perusteella. Jos mato onnistuu hyökkäyksessä, se lataa itse matokoodin hyökkääjän palvelimelta FTP:n avulla. Mato aiheuttaa tietoliikennettä tiettyihin tietokoneen portteihin. [4] Sasser saastuttaa Windows 2000/XP -käyttöjärjestelmää käyttäviä tietokoneita, mutta voi käyttää leviämisessa myös Windows 95/98/Me -käyttöjärjestelmiä. [11]

Sasser löytää Weaverin ja kumppaneiden [12] luokituksen mukaan kohteensa tutkaamalla, koska se etsii tietoturva-aukkojen avulla sopivaa kohdetta ja tutkaa mahdollista haavoittuvaa tietokonetta. Kuljettajaluokituksen mukaan, se taas on toisen kanavan käyttäjä, koska se käyttää FTP:tä saastuttamisen loppuun viemisessä.

Sasser pystyi sammuttamaan UK:n rannikkovartioston, Taiwanin postilaitoksen ja suomalaisten pankkien järjestelmiä [1]. Näiden yritysten tietoturvan pitäisi olla erittäin korkea, mutta siltikin ne saastuivat. Sasser onkin tämän takia varsin vaarallinen mato tai ainakin se pystyi yllättämään näiden organisaatioiden tietoturvan ja tunkeutumaan heidän järjestelmiinsä. Tästä syystä Sasser varmastikin tunnetaan yleisesti aika hyvin.

Sasser tuskin tulee olemaan ainoa mato, joka pystyy kaatamaan suurten yritysten ja pankkien tietojärjestelmiä. Tällaisia tahoja vastaan hyökätään, koska näiden järjestelmien kaatuminen aiheuttaa suurta kohua ja matojen kirjoittajat saavat kuuluisuutta onnistuessaan tässä. Kuuluisuuden lisäksi matojen kirjoittajan yrittävät päästä isojen yritysten ja pankkien järjestelmiin, koska siellä on paljon henkilötietoa yksityisistä ihmisistä ja heidän henkilötunnuksiaan, joita voidaan hyväksikäyttää rikollisissa toimissa. Saastuttamalla ison yrityksen järjestelmän, madon kirjoittaja voi saada käyttöönsä paljon tietokoneita, jotka voivat edelleen levittää matoa eteenpäin.

4.3 Zotop

Zotop käyttää hyväkseen tietoturva-aukkoja. Se aiheuttaa tietokoneen hidastumista sekä tietokoneen uudelleenkäynnistymistä. Päästyään tietokoneelle Zotop hakee itse matokoodin palvelimelta ja alkaa etsiä uhreja satunnaisien ip-osoitteiden avulla. Mato jättää jälkeensä takaoven ja se estää myös pääsyn tietyille virustentorjuntaohjelmien sivustoille.[3]

Zotop on aika samanlainen mato kuin Sasser. Erona Sasseriin on kuitenkin se, että Zotop jättää jälkeensä takaoven ja estää käyttäjien pääsyn tietyille virustorjuntaohjelmien sivustoille. Tämä voi olla aika tehokas suoja madolle säilyttää itsensä tietokoneella. Yleensä virustorjuntaohjelman joutuu lataamaan internetistä tai ainakin niiden päivitykset. Estämällä tietokoneilta pääsyn näille sivustoille Zotop voi jatkaa toimintaansa, mikäli tietokone pidetään päällä.

5 Yhteenveto

Haittaohjelma on yläkäsite kaikelle ei-halutulle koodille, joka yrittää tunkeutua järjestelmään. Haittaohjelman alakäsitteitä ovat virukset, madot, troijalaiset hevoset ja botit. Virukset tarvitsevat olemassa olevan tiedoston levitäkseen. Madot taas voivat levitä ilman sellaista. Troijalainen hevonen on ohjelma, joka näyttää tarjoavan hyödyllisiä toimintoja, mutta joka sisältää haitallista koodia. Botti varastaa tietokoneen nettiyhteyden sekä mahdollistaa hyökkääjän käyttää tietokonetta omien ilkeiden tavoitteidensa täyttämiseksi. Botteja käytetään suurina määrinä hyökkäämään yhteen kohteeseen.

Tietokonematoja voi jaotella hyvin monin eri tavoin kuten kohteen löytämisen, kuljettajan, aktivoitumisen tai sen kuorman perusteella. Se voi löytää kohteensa tutkaamalla tai etsimällä tietoturva-aukkoja. Mato voi myös etsiä mahdollisia kohteitaan erilaisten listojen perusteella, jolloin se yrittää hyökätä listalla olevalle tietokoneelle. Mato voi myös odottaa passiivisena, jotta käyttäjä tulisi hakemaan sen. Mato voi kuljettaa itseään tai käyttää muuta kanavaa saastuttamisen loppuun viemisessä kuin mitä se käytti tietokoneelle pääsemisessä. Se voi myös lähettää itseään normaalin tietoliikenteen mukana korvaten lähetettyjä viestejä väärillä viesteillä tai lähettämällä omia viestejä tietoliikenteen mukana.

Madon tarvitsee aktivoitua päästyään kohteeseensa. Se voi aktivoitua itse tai se voi tarvita ihmisen apua siinä. Käyttäjä voi myös tehdä jotain aivan normaalia toimintaa kuten käynnistää tietokoneen uudelleen, jolloin mato aktivoituu. Mato voi myös aktivoitua järjestelmän tunnettujen palveluiden käynnistyessä. Se voi tehdä muutakin kuin levitä. Se voi esim. aukaista takaoven, etsiä käyttäjän henkilökohtaisia tietoja, poistaa tiedostoja tai kryptata niitä.

Eräitä viimeaikaisista kuuluisista madoista ovat MyDoom, Sasser ja Zotop. MyDoom levisi sähköpostiviestien välityksellä. Se tarvitsi aktivoituakseen käyttäjän apua, jonka jälkeen se etsi käyttäjän tietokoneelta sähköpostiosoitteita, johon se lähetti itsensä. Sasser ja Zotop olivat aika samanlaisia ja ne kumpikin levisivät tietoturva-aukkojen avustuksella. Tämän jälkeen ne asensivat hyökkääjän palvelimelta matokoodin järjestelmään, jolloin se saastui. Zotop toisin kuin Sasser esti käyttäjän pääsyn virustorjuntaohjelmien kotisivuille ja aukaisi järjestelmään takaoven.

Viitteet

1. Anon: Sasser worm rises out of MS patch thicked. Computer Fraud & Security 2004, 5 (2004), 2.
2. CERT-FI: CERT-FI haavoittuvuustiedote 048/2003. https://www.cert.fi/haavoittuvuudet/2003/varoitus-2003-48.html. Tulostettu 15.12.2008

3. Cisco: Zotob worm information. http://www.cisco.com/web/about/security/intelligence/zotob-worm.html. Checked 19.09.2008.
4. F-Secure: F-Secure viruskuvaukset. http://www.f-secure.com/v-kuvaus/sasser.shtml. Vierailtu 19.09.2008.
5. Heiser, J.: Understanding today's malware. Information Security Technical Report 9 (2004), 47-64.
6. Kienzle, D, Elder, M.: Recent worms: A survey and trends. In: Proc. of 2003 ACM Workshop on Rapid Malcode, 1-10.
7. Legon, J.: Security firm: MyDoom worm fastest yet. CNN (2004). Available http://edition.cnn.com/2004/TECH/internet/01/28/mydoom. spreadwed/. Checked 23.11.2008
8. Provos, N., McClain, J., Wang, K.: Search worms. In: Proc. of the 4th ACM Workshop on Recurring Malcode (2006), 1-8.
9. Schultz, E.: Where have the worms and viruses gone? -new trends in malware. Computer Fraud & Security 2006 (2006), 4-8.
10. Sollins, K: The TFTP Protocol. http://www.isi.edu/in-notes/ien/ien133.txt. Checked 15.12.2008
11. Symantec: W32.Sasser.Worm http://www.symantec.com/security_response/writeup.jsp?docid=2004-050116-1831-99. Checked 19.9.2008.
12. Weaver, N.,Paxson, V.,Stanifor, S., Cunningham, R.: A taxonomy of computer worms. In: Proc. of the 2003 ACM Workshop on Rapid Malcode, 11-18.

Advanced Encryption Standard

Tuomas Pellonperä

Tampereen yliopisto

tuomas.pellonpera@uta.fi

Tiivistelmä Esittelen tässä artikkelissa AES-algoritmin. Luon yleissilmäyksen sen taustalla olevaan matematiikkaan ja kerron kohtalaisen yksityiskohtaisesti algoritmin toiminnasta.

1 Taustaa

Advanced Encryption Standard -standardi – lyhyesti AES – on Yhdysvaltojen hallituksen hyväksymä ja omaksuma lohkosalaaja. Tammikuussa 1997 Yhdysvaltojen Kansallinen teknologian ja standardien tutkimuslaitos[1] NIST ilmoitti haluavansa valita seuraajan *Data Encryption Standard* -salaajalle (DES), joka oli alkanut tulla haavoittuvaksi eri hyökkäyksille sen lyhyen avainpituuden ja tietokoneiden laskentakapasiteetin kasvun johdosta. Varteen otettavia seuraajaehdokkaita tuli 15, joista viisi valittiin toiselle kierrokselle. Lokakuussa 2000 NIST ilmoitti, että belgialaisten Joan Daemenin ja Vincent Rijmenin kehittämä Rijndael oli valittu voittajaksi avoimen ja läpinäkyvän valintaprosessin päätteeksi. Uusi standardi tuli voimaan toukokuussa 2002, mistä lähtien sitä on analysoitu intensiivisesti ja se otettu käyttöön maailmanlaajuisesti. [4]

[1] National Institute of Standards and Technology.

Yhdysvaltojen Kansallisen turvallisuuden virasto NSA[2] tarkisti kaikki toisen kierroksen finalistit ja luokitteli ne kaikki tarpeeksi turvallisiksi salaamaan Yhdysvaltojen hallituksen ei-salaisiksi luokiteltuja asiakirjoja. Kesäkuussa 2003 Yhdysvaltain hallitus ilmoitti, että AES on riittävän turvallinen salaamaan jopa erittäin salaiseksi luokiteltua aineistoa, kunhan avainpituus on vähintään 192 bittiä. [6] Ensimmäistä kertaa historiassa suuri yleisö voi siis käyttää NSA:n hyväksymää salausalgoritmia.

Vaikka termejä "AES" ja "Rijndael" käytetään arkikielessä toistensa synonyymeinä, tarkkaan ottaen ne eivät ole samoja. AES nimittäin määrittelee lohkopituudeksi 128 bittiä, kun sitä vastoin Rijndael määrittelee eri lohkopituuksia. (Kerron tarkemmin näistä eroista jäljempänä.) Otan tässä tutkielmassa vapauden käyttää termejä synonyymeinä keinona välttää liiallista toistoa. Haluan lukijan kuitenkin ottavan huomioon ja tiedostavan niiden vivahde-eron.

2 Matemaattiset perusteet

Kryptografiset protokollat ja algoritmit käsittelevät viestejä äärellisen avaruuden numeroina tai alkioina. Salaus- ja purkuoperaatiot muuttavat viestejä toisiksi viesteiksi siten, että operaatioiden tuloksena syntyneet viestit ovat saman äärellisen avaruuden alkioita. Tätä kutsutaan *sulkeutuvuusominaisuudeksi*. Tavallisilla aritmeettisilla laskutoimituksilla – yhteen, vähennys-, kerto- ja jakolaskulla – ei kuitenkaan välttämättä ole tätä ominaisuutta, minkä johdosta kryptografiset algoritmit käyttävät erityisiä

[2] National Security Agency.

algebrallisia struktuureja säilyttääkseen sulkeutuvuusominaisuuden. Näistä tärkeimpiä ovat ryhmät, kehät ja kunnat. [2]

AES-algoritmin operaatiot käsittelevät tietokoneen muistin tavuja. Koska yksi tavu on kahdeksan (8) bittiä, se voi ilmaista 256 ($= 2^8$) eri kokonaislukua. AES-algoritmin käyttämä algebrallinen struktuuri on nimeltään *äärellinen kunta* – jota kutsutaan myös Galois'n kunnaksi – ja sen tunnusluku on kaksi (2) ja ulottuvuus kahdeksan (8). Siitä käytetään merkintää $\mathbf{GF}(2^8)$. Rijndaelin kunnan alkiot ovat siis välillä 0–255, eli ne mahtuvat sopivasti yhteen tavuun.

2.1 Merkintöjä

Ennen kuin esittelen AES-algortimin äärellisen kunnan laskutoimituksia kerron, mitä käyttämäni merkinnät tarkoittavat. Tämä kohta hyödyntää pääasiassa lähdettä [3].

Merkintä

$$\{\, d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 \,\} \tag{2.1}$$

tarkoittaa yhtä tavua. Jokainen bitti d_i saa joko arvon nolla (0) tai yksi (1), kun $0 \leq i \leq 7$. Yllä oleva merkintä tarkoittaa siis binäärilukua. Sen voi lyhyemmin esittää kuusitoistajärjestelmän lukuna eli heksadesimaalilukuna. Esimerkiksi kymmenjärjestelmän luku 91 merkitään heksadesimaalilukuna

$$\{5B\}. \tag{2.2}$$

Koska AES kuitenkin pitää tavuja äärellisen kunnan alkioina, merkintä (2.1) yleensä tulkitaan korkeintaan seitsemättä astetta olevaksi polyno-

miksi

$$d = d_7x^7 + d_6x^6 + d_5x^5 + d_4x^4 + d_3x^3 + d_2x^2 + d_1x^1 + d_0, \tag{2.3}$$

jonka kertoimet siis ovat kunnan $\mathbf{GF}(2^1)$ alkioita.

Esimerkiksi { 10111001 } tarkoittaa siis polynomia $x^7+x^5+x^4+x^3+1$.

Merkintä

$$(b_0b_1b_2b_3 \ldots b_{n-1}) \tag{2.4}$$

tarkoittaa n:n tavun jonoa tai taulukkoa. Huomaa, että toisin kuin merkinnässä (2.1) merkinnässä (2.4) indeksointi kasvaa vasemmalta oikealle.

Sana (*word*) tarkoittaa 32 bitin – eli neljän (4) tavun – jonoa, jota käsitellään yhtenä kokonaisuutena tai neljän elementin taulukkona. Täten

$$w = (w_0w_1w_2w_3), \tag{2.5}$$

kun w_0, w_1, w_2 ja w_3 ovat tavuja ja w on sana.

Merkintä

$$\begin{bmatrix} b_{0,0}\ b_{0,1}\ b_{0,2}\ b_{0,3} \\ b_{1,0}\ b_{1,1}\ b_{1,2}\ b_{1,3} \\ b_{2,0}\ b_{2,1}\ b_{2,2}\ b_{2,3} \\ b_{3,0}\ b_{3,1}\ b_{3,2}\ b_{3,3} \end{bmatrix} \tag{2.6}$$

tarkoittaa 16 tavun (4×4) matriisia tai kaksiulotteista taulukkoa. Merkintä $b_{r,c}$ tarkoittaa r:nnen rivin ja c:nnen sarakkeen alkiota (tavua). Huomaa, että matriisin jokainen rivi ja sarake muodostaa muotoa (2.5) olevan sanan.

2.2 Yhteen- ja vähennyslasku

Rijndaelin käyttämässä Galois'n kunnassa alkioiden yhteen- ja vähennyslasku ovat identtiset ja helposti toteutettavissa; ne voidaan nimittäin määritellä yksinkertaiseksi xor-operaatioksi (jota merkitään symbolilla $\oplus$).

Olkoot a ja b Galois'n kunnan alkioita. Merkitään

$$\begin{aligned} a &= \{\, a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 \,\}, \\ b &= \{\, b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \,\}, \\ c &= \{\, c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0 \,\}. \end{aligned} \tag{2.7}$$

Tällöin yhteenlasku $c = a + b$ ja vähennyslasku $c = a - b$ määritellään

$$c_i = a_i \oplus b_i, \quad 0 \leq i \leq 7.$$

Nyt yhteen- ja vähennyslasku ovat *sulkeutuvia* eli $c \in \mathbf{GF}(2^8)$.

2.3 Kertolasku

Kertolaskun merkkinä käytetään symbolia '$\bullet$'. Polynomiesityksenä kertolasku kunnassa $\mathbf{GF}(2^8)$ tarkoittaa kahden polynomin tuloa modulo kahdeksatta astetta oleva *supistumaton polynomi*. Polynomi on supistumaton, jos se on jaollinen vain itsellään ja luvulla yksi.[3] AES-algoritmin supistu-

[3] Huomaa yhdenmukaisuus alkuluvun käsitteen kanssa.

maton polynomi on

$$m(x) = x^8 + x^4 + x^3 + x + 1, \tag{2.8}$$
$$= \{01\}\{1B\} \text{ heksadesimaalina.}$$

Kertolasku siis suoritetaan kertomalla polynomit a ja b termeittäin ja jakamalla tämä tulo polynomilla $m(x)$.

Kertolaskua ei kuitenkaan yleensä ohjelmoida yllä olevalla tavalla, vaan sen voi tehdä tehokkaammin käyttämällä hyväksi Rijndaelin kunnan tiettyjä potenssiinkorotusominaisuuksia, niiden pohjalta tehtyjä taulukkoja ja näistä tehtyjä hakuja. En kuitenkaan esitä tässä kirjoituksessa tuota algoritmia tämän yksityiskohtaisemmin, vaan pyydän asiasta kiinnostunutta lukijaa tutustumaan lähteeseen [5].

Rijndael-algoritmi ei sisällä kahden Galois'n kuntaan kuuluvan *muuttujan* välistä kertolaskua, vaan vain yksi muuttuja kerrotaan vakion kanssa [4, p. 53]. Tämä seikka yksinkertaistaa algoritmin ohjelmointia.

2.4 Kertolasku polynomilla x

Polynomi x vastaa kuusitoistajärjestelmän lukua $\{02\}$. Sillä kertominen on edellisessä kappaleessa esitellyn kertolaskun erityistapaus.

Kun polynomi (2.3) kerrotaan polynomilla x, tulos on

$$d_7x^8 + d_6x^7 + d_5x^6 + d_4x^5 + d_3x^4 + d_2x^3 + d_1x^2 + d_0x. \tag{2.9}$$

Kertolaskun $x \bullet b(x)$ tulos saadaan supistamalla (2.9) modulo $m(x)$ (katso (2.8)). Jos $d_7 = 0$, tulos on valmiiksi supistetussa muodossa; jos $d_7 = 1$, tu-

los supistetaan vähentämällä siitä $m(x)$ (kuten esitin kohdassa 2.2, vähennyslasku on siis yhtä kuin xor-operaatio). Täten kertominen polynomilla x voidaan toteuttaa siirtämällä d:n bittejä yhden askeleen vasemmalle ja suorittamalla sitten bittitason xor-operaatio luvulla $\{1B\}$.

Kertomista polynomilla x merkitään **xtime**(x). Tätä funktiota kutsumalla voidaan suorittaa kertolaskuja x:n korkeammilla potensseilla.

2.5 Tavukertoimiset polynomit

Tämä kappale käsittelee polynomeja, joiden kertoimet ovat kunnan $\mathbf{GF}(2^8)$ alkioita. Tällainen polynomi on korkeintaan kolmatta astetta ja muotoa

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0, \tag{2.10}$$

eli se voidaan esittää yhtenä muotoa (2.5) olevana sanana. Kahden tällaisen polynomin tulo on supistettava neljättä astetta olevalla polynomilla, jotta tulo olisi sulkeutuva. AES-algoritmissa tämä polynomi on $x^4 + 1$ siten, että

$$x^i \mod (x^4 + 1) = x^{i \mod 4}.$$

Jos tulon toinen polynomi $a(x)$ on kiinnitetty, laskutoimitus $a(x) \otimes b(x)$ voidaan kätevästi suorittaa matriisikertolaskuna

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} a_0\ a_3\ a_2\ a_1 \\ a_1\ a_0\ a_3\ a_2 \\ a_2\ a_1\ a_0\ a_3 \\ a_3\ a_2\ a_1\ a_0 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}, \tag{2.11}$$

jossa esimerkiksi

$$c_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$$

AES-algoritmissa on tuota seikkaa hyväksi käyttävä funktio **Rotl( )**, jonka kiinteässä polynomissa $a(x)$ on $a_0 = a_1 = a_2 = \{00\}$ ja $a_0 = \{01\}$. Toisin sanoen $a(x)$ on polynomi x^3. Sen tehtävä on muuttaa syötteenä annettu sana ($b_0b_1b_2b_3$) muotoon ($b_1b_2b_3b_0$), eli se kierrättää tavuja yhden pykälän vasemmalle. Myös **MixColumns**-muunnos käyttää kiinnitettyä polynomia ja matriisituloa. Siitä enemmän kohdassa 4.4

3 Suunnitteluperiaatteet

Rijndaelin kehittäjät Daemen ja Rijmen kertovat suunnitelleensa algoritmin käyttäen *Wide Trail Design* -strategiaa. [1] Koska en tiedä tälle termille hyvää suomennosta enkä edes usko sille olevan mitään vakiintunutta käännöstä, suomennan sen termillä WTD-suunnittelumalli tai -strategia.

Daemen ja Rijmen kuvaavat Rijndaelia "vaihtoavainlohkosalaajaksi" (*key-alternating block cipher*): se toistaa kierroksia, joissa vuorottelevat salausavaimen soveltaminen ja tästä riippumattomat kierrosmuunnokset.

WTD-suunnittelumalli pyrkii tekemään salaajat mahdollisimman vastustuskykyisiksi lineaarista ja differentiaalista salakirjoituksen purkua vastaan. Se pilkkoo kierrosmuunnoksen kahteen osaan – epälineaarisen ja lineaariseen muunnokseen (joista käytetään vastaavasti symboleja γ ja λ).

Turvallinen salaaja (a) tekee salausavaimen ja salatekstin suhteen mahdollisimman monimutkaiseksi ja vaikeatajuiseksi sekä (b) hälventää selvätekstin tilastolliset toistuvuudet salatekstin tilastollisuuteen.

Ensiksi mainittua ominaisuutta kutsutaan 'sekoitukseksi' (*confusion*) ja jälkimmäistä ominaisuutta 'hajaannukseksi' *diffusion*). Rijndaelin γ-muunnos tuottaa sekoitusta ja λ-muunnos puolestaan hajaannusta. Selvätekstin symbolin korvaaminen jollakin toisella symbolilla on sekoituksen pääasiallinen mekanismi. Rijndael käyttää tähän tarkoitukseen erityisesti suunniteltua korvauslaatikkoa (*S-box*). Hajaannus luodaan erityisillä lineaarisilla muunnoksilla.

4 Salausalgoritmi

AES-algoritmi saa syötteeksi 128 bittiä – eli 16 tavua – pitkän merkkijonon (lohkon) ja tuottaa samanpituisen tulosteen. Salausavaimen pituus on joko 128, 192 tai 256 bittiä.

Rijndael-algoritmi sallii sekä syöte-, tuloste-[4] ja avainpituuden olevan 128, 160, 192, 224 tai 256 bittiä. Tämä on näiden kahden algoritmin pääasiallinen ero. Vaikka olen tässä kirjoituksessa ottanut vapauden käyttää nimiä 'AES' ja 'Rijndael' synonyymeinä, esittelen nyt vain AES-algoritmin ja säädän kielenkäyttöni sen mukaiseksi.

AES-algoritmi koostuu eri muokkausvaiheita sisältävistä kierroksista, jotka muuttavat algoritmin syöte- eli selvätekstin salaustekstiksi. Tässä 'teksti' ei välttämättä tarkoita kirjallista tekstiä: koska AES käsittelee tietokoneen muistin tavuja, se voi vallan hyvin salata esimerkiksi videotie-

[4] Syöte- tulostepituudet ovat kuitenkin aina yhtä suuret.

dostoja ja ylipäänsä kaikkea ei-tekstuaalista dataa. Kierrosten lukumäärä N_r riippuu salausavaimen pituudesta N_k taulukossa 1 esitetyllä tavalla.

N_k	N_r
4	10
6	12
8	14

Taulukko 1. Kierroslukumäärä ja avainpituus.

N_k on siis salausavaimen pituus bitteinä jaettuna luvulla 32. Toisin sanoen N_k on avaimen pituus sanoina (katso määritelmä (2.5)). Koska lohkon pituus N_b on vakio AES-algoritmissa, $N_b = 4$ (eli lohkon pituus – 128 bittiä – jaettuna luvulla 32).

AES säilyttää kierrosten laskutoimitusten välitulokset $4 \times N_b$ -kokoisessa kaksiulotteisessa taulukossa (2.6), jota kutsutaan algoritmin *tilaksi*. Tilan koko on siis yhtä suuri kuin salattavan lohkon pituus. Algoritmin alussa se alustetaan salattavalla lohkolla; algoritmin aikana kierrosten muunnokset ottavat sen syötteekseen ja tallentavat tulosteensa sinne; algoritmin lopussa tila palautetaan lopputuloksena – salattuna lohkona.

Salausavainta ei käytetä sellaisenaan kierrosmuunnosten aikana, vaan algoritmin alussa se laajennetaan erityisellä avainaikataululla. Jokaisella kierroksella käytetään siten eri avainta. Viimeinen kierros eroaa aiemmista kierroksista siten, että sen aikana suoritetaan yksi muunnos vähemmän.

Pseudokoodi 1 kuvaa salausalgoritmin. Seuraavaksi esittelen tarkemmin avainaikataulun ja kierrosmuunnokset.

Pseudokoodi 1. Rijndaelin salausalgoritmi

```
Cipher(byte in[4N_b], byte out[4N_b], byte key[4N_k])
{
    byte state[4,N_b]
    word W[(N_r + 1)N_b]

    state = in
    KeyExpansion(key, W)
    AddRoundKey(state, W[0, N_b − 1])

    for (round = 1; round < N_r; round++)
    {
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, W[round∗N_b, (round+1)∗N_b − 1])
    }
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, W[N_r∗N_b, (N_r+1)∗N_b − 1])
    out = state
}
```

Pseudokoodi 2. Rijndaelin avainlaajennus

```
KeyExpansion(byte key[4N_k], word W[N_b(N_r + 1)], N_k)
{
    word temp

    i = 0
    while (i < N_k)
    {
       W[i] = word(key[4i], key[4i+1], key[4i+2], key[4i+3])
        i++
    }
    i = N_k
    while (i < N_b(N_r+1)]
    {
        temp = W[i−1]
        if (i mod N_k = 0)
            temp = SubWord(Rotl(temp)) ⊕ Rcon[i/N_k]
        else if (N_k > 6 & i mod N_k == 4)
            temp = SubWord(temp)
       W[i] = W[i−N_k] ⊕ temp
        i++
    }
}
```

4.1 Avainaikataulu

Pseudokoodi 2 esittää AES:n avainaikataulun. Tämä koostuu kahdesta komponentista – avainlaajennuksesta (*key expansion*) ja kierrosavaimen valinnasta. Koska joka kierroksella käytetään eri avainta, alkuperäinen N_k sanaa pitkä avain laajennetaan $N_k(N_r + 1)$ sanan pituiseksi taulukoksi. Tämän N_k ensimmäistä sanaa sisältävät alkuperäisen avaimen. Loput sanat muodostetaan suorittamalla eri operaatioita aikaisempiin sanoihin.

Rotl-muunnos siirtää sanan tavuja vasemmalle niin, että syötteeksi annettu sana ($a_0a_1a_2a_3$) muutetaan sanaksi ($a_1a_2a_3a_0$). Sanoja sisältävän kierrosvakiotaulukon **Rcon**[i] alkiot ovat muotoa ($x^{i-1}, \{00\}, \{00\}, \{00\}$), kun x^{i-1} tarkoittaa polynomin x (heksadesimaalina $\{02\}$) potensseja (katso kohta 2.4). Huomaa, että $i \geq 1$. **SubWord**-muunnos suorittaa **SubBytes**-operaation sanan jokaiseen tavuun.

4.2 SubBytes-muunnos

SubBytes on epälineaarinen tavujen korvausoperaatio (luvussa 3 mainittu γ-muunnos), jonka tarkoitus on tuottaa *sekoitusta*.

Periaatteessa tämä muunnos on yhdistetty funktio, jossa tilan tavu a kuvataan tavuksi b seuraavasti:

$$f(g(a)) = b,$$
$$g(a) = a^{-1},$$
$$f(a) = Ma + c.$$

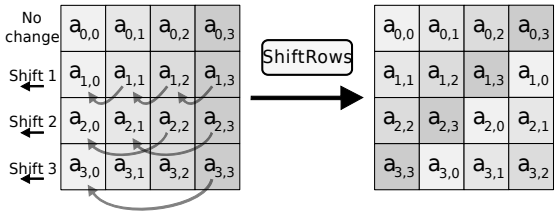
Funktio g kuvaa alkion a sen kertovaksi käänteisalkioksi kunnassa $\mathbf{GF}(2^8)$ (nolla-alkio kuvataan itseensä). Funktio f on affiininen muunnos, jossa M on 8×8 -matriisi, c on vektorina esitetty AES:n kunnan alkio $\{C6\}$ ja yhteenlasku on xor-operaatio $\oplus$.

Tämä operaatio ohjelmoidaan yleensä taulukkohauksi.

4.3 ShiftRows-muunnos

Tämä muunnos operoi tilan riveillä (jotka tulkitaan sanoiksi). Jokaisen paitsi ensimmäisen rivin tavuja siirretään tietty määrä vasemmalle. Tulosteen jokainen sarake koostuu neljän eri rivin yhdestä tavusta.

Kuva 1 havainnollistaa **ShiftRows**-muunnosta.



Kuva 1. AES ShiftRows-muunnos [6].

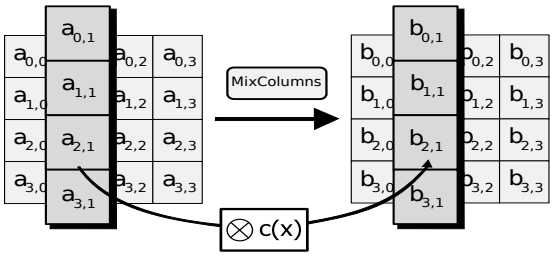
4.4 MixColumns-muunnos

MixColumns-muunnos operoi tilan sarakkeilla, jotka tulkitaan neljän termin polynomeiksi (2.10) ja kerrotaan modulo $(x^4 + 1)$ kiinnitetyllä polynomilla $c(x)$, kun

$$c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

Tämän muunnoksen voi suorittaa matriisitulona (2.11). Kuva 2 havainnollistaa **MixColumns**-muunnosta.

ShiftRows- ja **MixColumns**-muunnos tuottavat salaajaan *hajaannusta*.



Kuva 2. AES MixColumns-muunnos [6].

4.5 AddRoundKey-muunnos

AddRoundKey-muunnos suorittaa xor-operaation algoritmin tilan ja kierrosavaimen välillä.

Kierrosavaimia tarvitaan $N_k(N_r + 1)$ kappaletta, koska ennen ensimmäistä kierrosta suoritetaan yksi xor-operaatio. Loput operaatiot suoritetaan kierrosten viimeisenä muunnoksena.

AddRoundKey on ainoa salausavaimesta riippuvainen muunnos.

5 Purkualgoritmi

Algebrallisesti siistin rakenteen johdosta AES:n kaikilla kierrosmuunnoksilla on käänteismuunnos. Näin ollen salaus voidaan purkaa suorittamalla käänteisoperaatiot salausalgoritmiin nähden päinvastaisessa järjestyksessä, jolloin avainaikataulua ei tarvitse muuttaa.

AddRoundKey on itsensä käänteismuunnos (**InvAddRoundKey**); sehän on vain xor-operaatio. **ShiftRows** voidaan kääntää siirtämällä rivien tavuja tietty määrä oikealle (**InvShiftRows**): ensimmäistä riviä ei siirretä, toista riviä siirretään yksi askel, kolmatta riviä kaksi ja neljättä riviä kolme askelta. Sen sijaan **SubBytes**- ja **MixColumns**-muunnosten käänteisoperaatiot (vastaavasti **InvSubBytes** ja **InvMixColumns**) ovat hieman monimutkaisempia, mutta nekin saadaan algebrallisesti elegantiin asuun. En esittele tässä artikkelissa käänteismuunnoksia tämän tarkemmin. Ne on määritelty tarkasti lähteissä [4] ja [3].

Seuraavan kahden seikan ansiosta purkualgorimin käänteismuunnokset voidaan suorittaa myös samassa järjestyksessä kuin salausalgoritmin vastaavat "normaalit" muunnokset:

i) **InvSubBytes**- ja **InvMixColumns** ovat toisiinsa nähden kommutatiivisia, eli niiden suoritusjärjestys voidaan vaihtaa muuttamatta lopputulosta.

ii) **InvMixColumns**- ja **InvAddRoundKey**-muunnokset ovat lineaarisia. Olkoon **InvMixColumns**-muunnos funktio f^{-1}, tila t, kierrosavain k ja xor-operaatio $\oplus$. Tällöin lineaarisuus tarkoittaa, että

$$f^{-1}(t \oplus k) = f^{-1}(t) \oplus f^{-1}(k).$$

Nyt avainaikatauluun on tehtävä yksi muutos: kaikkiin kierrosavaimiin paitsi ensimmäiseen ja viimeiseen on suoritettava ylimääräinen **InvMixColumns**-operaatio.

6 Yhteenveto

Uutena salausstandardina AES-algoritmia käytetään nykyään laajassa mitassa. Sitä on analysoitu ja sitä analysoidaan kattavasti. AES on siitä harvinainen salausalgoritmi, että se voidaan esittää algebrallisesti erittäin siistissä muodossa. Se, voidaanko tuota järjestyksenmukaista muotoa käyttää hyväksi algoritmia vastaan hyökätessä, on tiiviin tutkimuksen kohteena.

Viitteet

1. Joan Daemen and Vincent Rijmen. Security of a wide trail design. In *Progress in Cryptology — INDOCRYPT 2002*, volume 2551 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2002.
2. Wenbo Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall, 2003.
3. National Institute of Standards and Technology, Computer Security Division, Computer Security Resource Center., `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`. *Specification for the Advanced Encryption Standard (AES)*, 2001. Federal Information Processing Standards Publication 197.
4. Vincent Rijmen and Joan Daemen. *The Design of Rijndael*. Springer, 2002.
5. Sam Trenholme. AES' Galois Field. `http://www.samiam.org/galois.html`, 2008. Checked 2008-11-20.
6. Wikipedia. Advanced Encryption Standard. `http://en.wikipedia.org/wiki/Advanced_Encryption_Standard`, 2008. Checked 2008-09-28.

Graafin kaarten risteymien minimointi neuroverkkoalgoritmeilla

Jukka Peltomäki

Tiivistelmä.
Graafien esittäminen kaksiulotteisena mahdollisimman vähäisin kaarien risteymin on NP-kova ongelma, jonka approksimointiin neuroverkot soveltuvat hyvin. Kyseisen ongelman ratkaisuun esittelen tässä kolmen neuroverkkoihin perustuvan algoritmin toimintaa ja eroja sekä tehokkuudessa että tarkkuudessa. Käsiteltävät algoritmit ovat Cimikowskin ja Shopen, Wangin ja Okazakin sekä Hen, Sýkoran ja Mäkisen algoritmit.

Avainsanat ja -sanonnat: Graafin planarisointi, risteymien minimointi, neuroverkko-optimointi, Hopfield-verkko.
CR-luokat: G.2.2, I.2.6

1. Johdanto

Graafien esittäminen kaksiulotteisena mahdollisimman vähin kaarien risteymin on laskettavuudeltaan NP-kova ongelma [Garey and Johnson, 1983], jonka likimääräiseen ratkaisuun on kehitetty useita algoritmeja. Ongelma voidaan ratkaista esimerkiksi perinteisellä [Jünger and Mutzel, 1993], geneettisellä [Comellas, 1992] tai metaheuristisella [Resende and Ribeiro, 1997] algoritmilla. Risteymien minimointiin on kehitetty myös useita erilaisiin neuroverkkoihin perustuvia algoritmeja, jotka soveltuvat hajautetun rakenteensa ansiosta rinnakkaislaskentaan erinomaisesti ja tarjoavat tehokkaan ratkaisuvaihtoehdon.

Esittelen kolme risteymiä minimoivaa neuroverkkoalgoritmia pääpiirteissään, ja vertailen niiden tehokkuutta ja tarkkuutta teoreettisella tasolla. Ensimmäiseksi esittelen Robert Cimikowskin ja Paul Shopen vuonna 1996 esittämän algoritmin, josta myöhemmin käytän sukunimiin perustuvaa nimitystä CS. Toiseksi esittelen Rong Long Wangin ja Kozo Okazakin vuonna 2006 esittämän algoritmin, josta käytän niin ikään sukunimiin perustuvaa nimitystä WO. Kolmanneksi esitän Hongmei Hen, Ondrej Sýkoran ja Erkki Mäkisen niin ikään vuonna 2006 esittämän HSM-algoritmin.

2. Lähtökohdat

Graafin piirustuksella tarkoitetaan graafin solmujen ja kaarien sijainnin määrittämistä tasolla. Sama graafi voidaan tietysti piirtää äärettömän monella eri tavalla. Risteymänumerolla tarkoitetaan tietyn graafin minkä tahansa piirroksen vähintä kaarien risteymien määrää. Lisäksi määritellään parittainen ja pariton

risteymänumero tarkoittamaan vähintä kaarien parillista tai paritonta risteymien määrää kaikista tietyn graafin piirroksista. Pariton risteymänumero on aina parillista pienempi, ja ne molemmat ovat risteymänumeroa pienempiä. Graafin pariton risteymänumero määrittää optimaalisen ratkaisun kyseisen graafin risteymien minimointiongelmaan. [Pach and Tóth, 2000].

Risteymien minimointialgoritmien testauksissa käytetään usein täydellisiä graafeja, koska niiden pienin mahdollinen risteymien määrä tiedetään. Täydellisessä graafissa sen jokainen solmu liittyy kaarella jokaiseen toiseen. [Weisstein, 2008c].

Kaikki kolme esittelemääni algoritmia antavat graafille ns. kaksisivuisen esityksen. Yleisesti graafeja voidaan esittää tietyn sivumäärän kirjoina, joka tarkoittaa jokaisen sivun vastaavan avaruudessa omaa kaksiulotteista puolitasoaan, jotka leikkaavat yhdessä viivassa eli ikään kuin kirjan sidoskohdassa. Graafin solmut asetetaan peräkkäin sidoskohtaan ja niiden väliset kaaret piirretään sivuille. Tietysti mitä enemmän sivuja kirjassa on, niin sitä vähemmin risteymin graafi todennäköisesti voidaan sillä esittää, mutta sitä vaikeampi kirja on piirtää. Usein graafi halutaan esittää kaksiulotteisesti, joten risteymien minimointi juuri kaksisivuisessa esityksessä on käytännössä tärkeää. Graafin kirjaa kutsutaan kirjan upotukseksi (*book embedding*) silloin, kun yksikään kaaripari ei risteä. [Enomoto *et al.*, 1999].

Kaksisivuisessa esityksessä graafin solmut ovat tasavälein vaakalinjalla, ja kaaret piirretään puoliympyröinä joko ylä- tai alakautta, eli tilanne voidaan kuvitella pöydälle vaakatasoon avattuna kirjana. Solmujen järjestys linjalla vaikuttaa useimmissa tapauksissa siihen, miten vähin risteymin kaaret on edes teoreettisesti mahdollista piirtää [Cimikowski and Shope, 1996]. Käsittelemäni kolme algoritmia eivät kuitenkaan järjestele solmuja, vaan minimoivat risteymiä pelkästään määrittämällä kumpaa puolta kunkin kaaren olisi paras kulkea. Cimikowski ja Shope toteavat käyttävänsä solmujen järjestämiseen Nicholsonin [1968] ahnetta algoritmia, mutta kahdessa muussa tutkimuksessa solmujärjestykseen ei oteta kantaa. Nicholsonin algoritmissa valitaan ensimmäiseksi asteluvultaan suurin solmu, jonka jälkeen valitaan laskevassa järjestyksessä loput solmut sen mukaan, kuinka monta kaarta niistä johtaa jo linjalle asetettuihin solmuihin.

Tässä tutkielmassa oletetaan tunnetuiksi graafeihin liittyvät peruskäsitteet. Graafin solmujen määrää merkitään jatkossa n:llä ja kaarien määrää m:llä.

3. Neuroverkoista yleisellä tasolla

Neuroverkot ovat kehittyneet pyrkimyksestä matemaattisesti jäljitellä ihmisen aivosolujen toimintaa. Matemaattiset mallit yksinkertaistavat oikeiden solujen toimintaa huomattavasti, ja tässä esittelemäni algoritmit eivät käytä uusimpia

oivalluksia hyväkseen, vaan ne pohjautuvat jo vuosikymmeniä sitten esiteltyihin yksinkertaisimpiin malleihin. Yksinkertaisistakin komponenteista pystytään kuitenkin rakentamaan monimutkaisia kokonaisuuksia, ja esimerkiksi CS:n käyttämistä McCullochin ja Pittsin jo vuonna 1943 esittelemistä binäärineuroneista voidaan rakentaa verkottamalla mikä tahansa Boolen funktio. Neuroverkkojen varsinainen kätevyys perustuu kuitenkin siihen, että ilmiön noudattaman funktion arvailun tai päättelyn sijaan voidaan neuroverkolle vain antaa esimerkkejä syötteistä ja halutuista tuloksista, jonka jälkeen annetaan verkon itse oppia niiden välinen yhteys. Esimerkiksi ihmiskasvojen tunnistukseen ei ole kukaan pystynyt kehittämään täsmällistä funktiota, mutta neuroverkolle voidaan esimerkkien avulla opettaa kasvojen tunnistus.

Tässä esittelemäni graafien risteymien minimointiin tarkoitetut algoritmit eivät kuitenkaan perustu opettamiseen, vaan niissä käytetään neuroneita pelkästään komponentteina ja toiminnallisuus annetaan valmiin perinteisen funktion muodossa. Varsinaisesti ne eivät opi mitään, vaan muistuttavat enemmänkin perinteisiä algoritmeja kuin yleensä tekoälyyn käsitteellisesti liitettyjä oppivia neuroverkkoja. Tavallaanhan algoritmin voidaan sanoa oppivan tietty ratkaisu tietyn graafin risteymien minimointiin, ja Hopfield-verkon energialuvun minimointia kutsutaan myös oppimiseksi. Yleisesti ottaen esittelemissäni kolmessa tutkimuksessa olevat mallit kuuluvat kuitenkin etsintä- ja optimointimenetelmiin, eivätkä ne käytä ohjattua, vahvistavaa tai ohjaamatonta oppimista. [Hertz *et al.*, 1991].

WO ja HSM perustuvat molemmat muunnelmaan Hopfieldin mallista. Hopfieldin neuroverkkomalli on John Hopfieldin vuonna 1982 kehittämä rekursiivinen neuroverkko, jossa signaalit voivat kulkea sekä eteen- että taaksepäin ja dynaamisuuden konvergenssi on taattu. Jokaista verkon tilaa kuvaa energialuku, joka saadaan energiafunktiosta. Toiminnan kannalta energiafunktion valinta on oleellista, koska siinä verkolle määritellään optimoitava ongelma. Verkko pyrkii aina pienentämään energiaansa, ja se väistämättä päätyy paikalliseen minimiin neuronien satunnaisen päivityksen myötä. Verkon voidaan sanoa olevan tasapainotilassa energialuvun ollessa paikallisessa minimissä, ja toisin sanoen ratkaisun olevan valmis. [Hopfield, 1982]. Hopfield-verkkoa käytettäessä on otettava ratkaisun optimaalisuutta arvioitaessa huomioon sen varma ajautuminen nimenomaan paikalliseen minimiin, koska se ei välttämättä ole lähelläkään globaalia minimiä eli optimaalista ratkaisua.

4. Cimikowskin ja Shopen algoritmi

Graafin risteymien minimointiin Cimikowski ja Shope [1996] ovat esittäneet yleisesti $O(n^3)$ aikakompleksisen neuroverkkoalgoritmin, jossa jokaista graafin

kaarta kohden tarvitaan kaksi neuronia: yksi kuvaamaan kaaren kulkua yläkautta ja toinen kuvaamaan kaaren kulkua alakautta. Olkoon V_{UPij} solmujen i ja j välistä yläkautta kulkevaa kaarta kuvaava neuroni ja V_{DOWNij} alakautta kulkevaa kaarta kuvaava. Neuronit ovat muodoltaan McCulloch-Pittsin [1943] neuroneita, eli aktivaatiofunktiona on askelfunktio [Weisstein, 2008a] ja mahdollisia tuloksia tasan kaksi – tässä tapauksessa ykkönen tai nolla. Kaari voi olla siis yhdessä neljästä neuronien määräämästä tilasta: jos sekä V_{UPij} että V_{DOWNij} ovat nollia, ei kaaren kulkureittiä ole vielä määritelty. Jos kumpi tahansa on ykkönen toisen ollessa nolla, kulkee kaari ykkösen antavan neuronin osoittamaa puolta. Jos molemmat ovat ykkösiä, on kyseessä virhe. Ykkönen V_{UPij} on vain sen syötepotentiaalin U_{UPij} ollessa nollaa suurempi, ja sama pätee neuronin V_{DOWNij} ja sen syötepotentiaalin U_{DOWNij} suhteen. [Cimikowski and Shope, 1996].

Neuronin uuden syötteen laskemiseen käytettyjä muutosfunktioita käytetään kahta erilaista: ylä- ja alakautta kulkevaa kaarta kuvaaville neuroneille molemmille omaansa. Käytännössä muutosfunktiot eri puolille ovat hyvin samanlaiset – laskennallisesti ne ovat täysin identtiset. [Cimikowski and Shope, 1996].

Cimikowskin ja Shopen algoritmissa neuronien itseispotentiaalit U_{UPij} ja U_{DOWNij} alustetaan satunnaisesti, mutta kuitenkin siten, että ne ovat avoimella välillä $]-\omega, 0[$, jossa ω on määrätty positiivinen reaaliluku. Tämän jälkeen iteroidaan, kunnes kaikille kaarille on löytynyt kulkureitti tai ennalta määrätty maksimiaika on täyttynyt. Iteraatiokierroksen aikana päivitetään askelfunktiolla jokaisen neuronin tulot ja lasketaan muutosfunktioilla neuroneiden uudet arvot. [Cimikowski and Shope, 1996].

5. Wangin ja Okazakin algoritmi

Wang ja Okazaki [2006] ovat esittäneet graafin risteymien minimointiin paranneltuun Hopfieldin malliin perustuvan neuroverkkoalgoritminsa. CS:stä poiketen he käyttävät neuroverkossaan yhden neuronin jokaista kaarta kohden. Kaaren neuronin ollessa nolla kiertää kaari alakautta, ja sen ollessa yksi, kiertää kaari yläkautta. Neuronien syötteestä laskettava tuloste saadaan sigmoidifunktiolla [Weisstein, 2008b]. WO:n energiafunktio käy läpi kaikki kaariparit, mistä muodostuu algoritmin sarjallinen ja yleinen dominoiva aikakompleksisuus $O(m^2)$.

Hopfield-verkon paikalliseen minimiin päätyminen estetään WO:ssa lisäämällä neuronin ajassa seuraavan arvon määrittävään muutosfunktioon niin sanottu stabiloiva kerrointermi, jolla kerrotaan neuronin itseispotentiaali, ja joka kasvaa tasapainotusprosessin edetessä avoimella välillä nollasta ykköseen. Verkon ollessa vielä kaukana tasapainotilasta, on stabilointitermikin pieni, joten verkko ikään kuin odottaa hieman useamman iteraation ajan ennen tasa-

painottumistaan. Verkon lähestyessä ratkaisua lähenevät myös stabilointitermit ykköstä, jolloin neuronien itseispotentiaalien päivitys lähenee normaalia Hopfield-mallin käsittelyä. [Wang and Okazaki, 2006].

Aluksi neuronien itseispotentiaalit arvotaan avoimelle välille nollasta ykköseen ja arvot päivitetään sigmoidi-funktiolla. Tämän jälkeen iteroidaan verkon tasapainotilan saavuttamiseen asti. Iteraatiossa käydään neuroneita satunnaisessa järjestyksessä läpi kaksi kertaa niiden määrän verran. Neuronin läpikäynnissä sille lasketaan muutosfunktiolla uusi itseispotentiaali, ja päivitetään sen tuottama arvo. Verkon energia-arvo päivitetään jokaisen neuronin käsittelyn yhteydessä. [Wang and Okazaki, 2006].

6. Hen, Sýkoran ja Mäkisen algoritmi

He, Sýkora ja Mäkinen perustavat niin ikään 2006 julkaisemansa algoritmin Hopfieldin [1982] malliin. WO-algoritmin tapaan myös HSM varaa vain yhden neuronin kaarta kohti, eikä kiertosuunnassakaan ole eroa. Myös tässä haetaan tasapainotilaa verkolle, joskin neuronien tulosteet lasketaan askelfunktiolla sigmoidifunktion sijaan. Askelfunktio antaa tulokseksi ykkösen vain aidosti positiivisilla syötteillä ja nollan ei-positiivisilla syötteillä.

HSM:n energiafunktiossa käydään läpi kaikki kaariparit ja summataan leikkaavien kaarien määrä [He *et al.*, 2006]. Juuri tämä parien läpikäynti aiheuttaa algoritmille sen dominoivan aikakompleksisuuden $O(m^2)$.

Neuroni- ja aikakohtainen muutosfunktio käy läpi kyseisenä ajanhetkenä kyseistä neuronia vastaavan kaaren kanssa risteävät kaaret ja vähentää neuronin nykyisestä syötearvosta risteymien määrän summan sopivalla kertoimella. Alapuolelta risteävässä tapauksessa muutosfunktio kasvattaa neuronin syötearvoa, ja yläpuolelta risteävässä tapauksessa vähentää sitä. [He *et al.*, 2006]. Toisin sanoen, mikäli jokin kaari risteää alapuolella hyvin monen toisen kanssa, kasvatetaan kaaren neuronin syötearvoa. Kun syötearvo on kasvanut positiiviseksi asti, määrää askelfunktio kaaren kulkemaankin yläkautta, jossa toivottavasti on enemmän tilaa.

Aluksi jokaisen kaarta vastaavan neuronin syöte arvotaan joko ykköseksi tai miinus ykköseksi, eli käytännössä kaaret vain piirretään satunnaisesti kulkemaan joko ylä- tai alakautta. Tämän jälkeen aloitetaan neuronien läpikäynti muutosfunktiolla, ja iterointikierroksia tehdään kunnes verkko saavuttaa tasapainotilansa, eli energiafunktio on ajautunut paikallisen miniminsä eikä sen arvo enää laske. Neuroneille alussa satunnaisesti annettu arvo vaikuttaa, mihin monista paikallisista minimeistä energiafunktio päätyy, joten koko kierrossarja suoritetaan muutamia kertoja uudestaan satunnaisarvoilla alustuksesta lähtien, jonka jälkeen tuloksista valitaan lopputulokseksi pienimmän energia-arvon

saanut kierros. [He *et al.*, 2006].

7. Tehokkuudesta

Huomattavaa kaikkien kolmen valitun algoritmin kohdalla on niiden rinnakkainen luonne. Jokaisessa algoritmissa yksittäisen neuronin laskenta on vakioaikainen tapahtuma, joten rinnakkaisuutensa ansiosta algoritmeista voidaan rakentaa tiettyyn valittuun graafin rajakokoon asti vakioaikaisia ratkaisijoita. Yleisesti ottaen uudemmat Hopfieldin malliin perustuvat HSM ja WO ovat aikakompleksisuudeltaan $O(m^2)$ ja CS asymptoottisessa mielessä vain hieman huonommin skaalautuva $O(n^3)$. Ajan pysyessä vakiona kasvaa tietysti vaadittujen prosessointiyksiköiden määrä – näiden algoritmien tapauksessa lineaarisesti kaarien suhteen. Vaativissa tilanteissa joutuu ajan ja laskentayksiköiden määrän – eli käytännössä rahan – suhteen tekemään kompromissin. Uudemmat WO ja HSM käyttävät CS:ään verrattuna vain puolet neuroneita, joten vakioaikaisessa ratkaisujärjestelmässä laskentayksiköitäkin tarvitsee vain puolet.

Nopeusero HSM:n ja CS:n välillä on merkittävä. Neuronit läpikäyvä yksittäinen iteraatiokierros sujuu HSM:ltä nopeammin, mikä suurelta osin selittynee CS:ään nähden puolittaisella neuronien määrällä. Nopeampien iteraatioiden lisäksi HSM myös tasapainottuu nopeasti ja selviää CS:ää selkeästi vähemmillä määrillä iteraatioita. Missään testitapauksessa HSM ei tarvinnut yli 20 iteraatiota, ja esimerkiksi kymmenen solmun täydellisestä graafista HSM selviää kolmella iteraatiolla, kun CS tarvitsee 21 kierrosta. [He *et al.*, 2006].

8. Tulosten tarkkuudesta

Tehokkuutensa lisäksi uudemmat WO ja HSM pääsevät myös CS:ää selkeästi tarkempiin tuloksiin. Tarkkuus tässä on sama asia kuin ratkaisun läheisyys teoreettisesti optimaaliseen ratkaisuun. HSM ratkoo kolme- ja neljärivisiä ruudukoita optimaalisesti ja nopeasti. Esimerkiksi 4x9-ruudukosta HSM saa aikaan täysin planarisoidun graafin 15 millisekunnissa, kun CS:ltä menee noin 19 sekuntia, ja risteymiä jää vielä 327. [He *et al.*, 2006].

Vaikka CS pääsee parhaimmillaan optimaaliseen ratkaisuun ainakin alle kolmentoista solmun täydellisillä graafeilla, jää se kyseisillä graafeilla silti keskimäärin noin kymmenisen prosenttia parhaasta ratkaisusta. Luonnollisesti CS:n optimaaliseen ratkaisuun pääsemisen todennäköisyys pienenee graafin koon kasvaessa. Viiden solmun täydellisen graafin yhden risteymän optimaaliseen ratkaisuun CS yltää joka kerta, mutta jo kuuden solmun graafissa se löytää optimaalisen ratkaisun vain 42 % todennäköisyydellä ja kymmestä kolmeentoista solmun tapauksissa se yltää optimaaliseen tulokseen vain alle viidesosassa testikertoja. WO ratkaisee testattuun kolmeentoista solmuun asti kaikki täydelliset graafit optimaalisesti joka kerta. [Wang and Okazaki, 2006]. HSM rat-

kaisee täydellisiä graafeja kymmeneen solmuun asti CS:n kanssa yhtä hyvin. [He *et al.*, 2006]

Uudemmat HSM ja WO ovat siis selkeästi CS:ää tehokkaampia ja antavat myös laadukkaampia ratkaisuja. Paremmuuden syitä lienevät ainakin uudempien algoritmien perustuminen Hopfieldin malliin sekä vain yhden neuronin käyttö kaarta kohden.

Suurin ero HSM:n ja WO:n välillä on niiden keino käsitellä Hopfield-verkon ominaisuus tasapainottua paikalliseen minimiin. HSM ei pyrikään estämään paikallisen minimin löytymistä, vaan käy tasapainottumisprosessin useasti läpi erilaisilla satunnaislähtöarvoilla, ja valitsee saavutettujen tasapainotilojen joukosta parhaan ratkaisun. WO puolestaan aloittaa verkkonsa tasapainottamisen pienin muutoksin, ja lisää muutosten suuruutta iteraatioiden edetessä. Tämä tietysti pidentää tasapainotusprosessia, mutta koska se myös estää liian huonoon paikalliseen minimiin päätymistä, ei tasapainotusta tarvitse tehdä kuin kerran.

9. Lopuksi

Viime vuosina on esitetty myös muita vastaavaan tarkoitukseen soveltuvia neuroverkkoalgoritmeja, joiden lähempi tarkastelu suhteessa kolmeen käsittelemääni olisi mielenkiintoista. Esimerkiksi López-Rodríguezin ja muiden [2007] algoritmi optimoi kaarien kulkusivun lisäksi myös solmujärjestyksen, ja se perustuu kahteen toisiaan syöttävään MREM-neuroverkkoon, joilla minimoidaan samaa energiafunktiota. Kawarabayashin ja Reedin [2007] algoritmi ei perustu neuroverkkoon, mutta se pystyy lineaarisessa ajassa selvittämään, onko graafissa korkeintaan tietty määrä risteymiä. Risteymien jäädessä kysytylle tasolle algoritmi myös piirtää graafin.

Neuroverkot soveltuvat graafien risteymien minimointiin hyvin, ja algoritmit ovat vielä viimeisen kymmen vuoden aikanakin selkeästi kehittyneet tehonsa ja oikeellisuutensa osalta. Myös sarjallisesti sovellettuina tässä esitellyt algoritmit ovat nopeutensa puolesta käyttökelpoisia. Mielenkiintoiseksi kysymykseksi jää myös HSM- ja WO-algoritmien tarkempi eroavaisuus.

Viiteluettelo

[Cimikowski and Shope, 1996] Robert Cimikowski, and Paul Shope, A neural network algorithm for a graph layout problem. *IEEE Trans. on Neural Networks* **7**, 2 (Mar. 1996), 341–346.

[Comellas, 1992] F. Comellas, Using genetic algorithms for planarization problems. In: I.C. Brezinski and U. Kulish (Ed.). *IEEE Computational and*

Applied Mathematics, Elsevier, 1992, 93–100.

[Enomoto *et al.*, 1999] Hikoe Enomoto, Miki Shimabara Miyauchi, and Katsuhiro Ota, Lower bounds for the number of edge-crossings over the spine in a topological book embedding of a graph. *Discrete Applied Mathematics* **92**, 2-3 (Jun. 1999), 149–155.

[Garey and Johnson, 1983] M.R. Garey and D.S. Johnson, Crossing number is NP-complete. *SIAM J. Alg. Disc. Meth.* **4**, 3 (Sep. 1983), 312–316.

[He *et al.*, 2006] Hongmei He, Ondrej Sýkora, and Erkki Mäkinen, An improved neural network model for the two-page crossing number problem. *IEEE Trans. on Neural Networks* **17**, 6 (2006), 1642–1645.

[Hertz *et al.*, 1991] John A. Hertz, Anders S. Krogh, and Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991.

[Hopfield, 1982] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Nat. Academy Sci., USA*, **79** (1982), 2554–2558.

[Jünger and Mutzel, 1993] Michael Jünger and Petra Mutzel, Solving the maximum weight planar subgraph problem by branch and cut. In: G. Rinaldi and L. Wolsey (Ed.). *Proceedings of the Third Conference on Integer Programming and Combinatorial Optimization (IPCO)* (1993), 479–492.

[Kawarabayashi and Reed, 2007] Ken-ichi Kawarabayashi and Bruce Reed, Computing crossing number in linear time. In: *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing* (Jun. 2007), 382–390.

[López-Rodríguez *et al.*, 2007] Domingo López-Rodríguez, Enrique Mérida-Casermeiro, Juan M. Ortíz-de-Lazcano-Lobato, and Gloria Galán-Marín, K-pages graph drawing with multivalued neural networks. In: *Artificial Neural Networks – ICANN 2007. Lecture Notes in Computer Science* **4669** (Sep. 2007), Springer, 816–825.

[McCulloch and Pitts, 1943] Warren S. McCulloch and Walter Pitts, A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology* **5**, 4 (Dec. 1943), 115–133.

[Nicholson, 1968] T.A.J. Nicholson, Permutation procedure for minimising the number of crossings in a network. *Proc. IEE* **155**, 1 (1968), 21–26.

[Pach and Tóth, 2000] János Pach and Géza Tóth, Which crossing number is it anyway? In: *Proceedings of the 39th Annual Symposium on Foundations of Computer Science* (1998), IEEE Press, 617–626.

[Resende and Ribeiro, 1997] Mauricio G.C. Resende and Celso C. Ribeiro, A grasp for graph planarization. *Networks* **29** (1997), 173–189.

[Wang and Okazaki, 2006] Rong Long Wang and Kozo Okazaki, Solving the minimum crossing number problem using an improved artificial neural network. In: *Advances in Machine Learning and Cybernetics. Lecture Notes in Computer Science* **3930** (May 2006), Springer, 797–803.

[Weisstein, 2008a] Eric W. Weisstein, Heaviside Step Function. *From Mathworld – A Wolfram Web Resourc,* http://mathworld.wolfram.com/HeavisideStepFunction.html. Checked 1.12.2008.

[Weisstein, 2008b] Eric W. Weisstein, Sigmoid Function. *From Mathworld – A Wolfram Web Resource,* http://mathworld.wolfram.com/SigmoidFunction.html. Checked 1.12.2008.

[Weisstein, 2008c] Eric W. Weisstein, Complete Graph. *From Mathworld – A Wolfram Web Resource,* http://mathworld.wolfram.com/CompleteGraph.html. Checked 1.12.2008.

NFC-tekniikan vaikutus matkapuhelinpalveluiden käytettävyyteen

Piia Perälä

Tiivistelmä.
Matkapuhelinpalveluiden yksi tärkeimmistä ominaisuuksista on käytettävyys, joka käyttäjän näkökulmasta mittaa palvelun arvoa. Tämän tutkielman aiheena on esitellä, kuinka NFC (Near Field Communication) -teknologia vaikuttaa matkapuhelinpalveluiden käytettävyyteen. NFC:n käyttäminen matkapuhelimissa on arvioitu lisäävän palveluiden käytettävyyttä. Tutkielman tarkoituksena on koota yhteen aihealueen erilaisia tutkimustuloksia ja niiden pohjalta identifioida NFC:n käytössä ilmenneitä ominaisuuksia, jotka vaikuttavat positiivisesti käytettävyyteen.

Tutkielman tarkoituksen ei ole mennä syvälle tutkimuksien sisältöihin, vaan keskittyä lähinnä tehtyihin löydöksiin. Jokainen käytetty tutkimus esitellään kuitenkin lyhyesti, jotta kohdealue pystyään paremmin hahmottamaan.

Avainsanat ja -sanonnat: Käytettävyys, matkapuhelinpalvelu, NFC (Near Field Communication), tunniste
CR-luokat: H.5

1. Johdanto

NFC:llä varusteltuja matkapuhelimia on ollut Suomen markkinoilla vuodesta 2004 alkaen. Ensimmäinen Nokian toimittama matkapuhelin, johon NFC integroitiin, oli malliltaan Nokia 6131. Lisäksi Nokia tarjosi lisävarusteena NFC-kuoria Nokia 3220 ja 5140 puhelimiin. [Nokia]

Matkapuhelimessa olevan NFC-ominaisuuden käyttö ei kuitenkaan yleistynyt käyttäjien keskuudessa. Yhtenä syynä yleistymättömyyteen voidaan pitää, että tuolloin NFC:tä käytettäviä matkapuhelinpalveluita oli tarjolla liian vähän, jotta käyttäjät olisivat omaksuneet uuden teknologian käytön. Muutamat suomalaiset matkapuhelinoperaattorit kuitenkin kehittivät testausmielessä sovelluksia, joissa hyödynnettiin NFC-teknologian käyttöä. [TeliaSonera, Elisa]

Tällä hetkellä NFC-matkapuhelimet ovat vielä harvinaisia, mutta tulevaisuudessa niiden oletetaan lisääntyvän. Nokia arvioi ABI Researchin tekemän tutkimuksen pohjalta, että NFC-puhelimien määrä maailmassa tulee nousemaan nykyisestä noin 9,59 miljoonasta vuoteen 2012 mennessä 293,45 miljoonaan [Nokia White Paper]. Näin ollen voidaan ajatella, että NFC:tä tukevien matkapuhelimien määrän kasvaessa, erilaiset matkapuhelinpalvelut, joissa NFC:tä käytetään, tulevat myös lisääntymään.

2. NFC ja sen mahdollisuudet osana matkapuhelinpalveluita

NFC on RFID:hen (radio frequency identification) pohjautuva radiotaajuisen etätunnistuksen hyvin lyhyillä etäisyyksillä mahdollistava tekniikka. Tällä hetkellä NFC on vielä suhteellisen tuntematon tekniikka, mutta erityisesti matkapuhelimissa sen käytön oletetaan tulevaisuudessa lisääntyvän. [Nokia White Paper]

NFC toimii 13.56 MHz taajuudella ja sen tiedonsiirtonopeus on 424 Kbits/s. Se tarjoaa elektronisten laitteiden välille mahdollisuuden kahdensuuntaiseen vuorovaikutukseen, jossa NFC-ominaisuudella varusteltu laite voi toimia sekä lukijalaitteena että tunnisteena. Kommunikaatio laitteiden välillä tapahtuu noin neljän senttimetrin etäisyydellä. Yhteys saadaan aikaan joko koskettamalla laitteita toisillaan tai viemällä laitteet toistensa läheisyyteen, jolloin tiedonsiirtoyhteys saadaan muodostettua. Yhteys on verrattavissa muihin tunnettuihin langattomiin tiedonsiirtotekniikoihin kuten Bluetooth tai Wi-Fi. Koska tiedonsiirto on mahdollista hyvin pienillä etäisyyksillä, NFC tarjoaa turvallisen tiedonsiirtoyhteyden laitteiden välille. [NFC-forum]

NFC-ominaisuus matkapuhelimissa mahdollistaan monia mielenkiintoisia mahdollisuuksia kehittää uusia matkapuhelinpalveluita. Vuorovaikutus NFC:llä voidaan jakaa neljään eri tasoon, jotka sisältävät vastaavan metaforan [Geven *et al.*, 2007]:

- Passiivisten kohteiden lukeminen. Esimerkiksi yhdensuuntainen kommunikointi julisteessa olevan tunnisteen kanssa. Tunnisteen kautta käyttäjä pystyy vastaanottamaan lisätietoja web-selaimen välityksellä. Metaforana on viivakoodinlukija.
- Palveluiden vahvistaminen. Esimerkiksi pääsyliput voidaan lukea päätteellä ennen sisäänpääsyä. Metaforana on paperilippu.
- Maksaminen. Laite korvaa maksamisessa käytetyt luottokortit ja käteisen rahan. Maksu voidaan veloittaa luottokortin tililtä tai matkapuhelinoperaattorin laskulla. Metaforana on luottokortti.
- Peer to Peer (vertaisverkko) -jakaminen. Kahdensuuntaista kommunikaatiota käytetään määrätyn tavoitteen saavuttamiseksi, esimerkiksi erilaisten digitaalisten tietojen jakamiselle ja vastaanottamiselle. Metaforana on kaapeli.

Tällä hetkellä yksi yleisimmistä teknologisista ratkaisuista matkapuhelinpalveluissa toteuttaa kosketusparadigmaa on NFC. Kosketusparadigmassa vuorovaikutus ympäristön kanssa on toteutettu tuomalla mobiilaite kontaktiin tai hyvin lähelle älykästä tunnistetta.

Kosketusta voidaan pitää hyvin luonnollisena tapana ihmiselle, sillä usein ihmiset koskettavat kohteita, joita haluavat käyttää. [Jaring *et al.*, 2007] Lisäksi

tutkimukset ovat osoittaneet, että kosketusparadigma on helppo oppia ja intuitiivinen käyttää. [Välkkynen *et al.*, 2006]

RFID:n perusajatus on, että kosketettavat kohteet on merkitty tunnisteilla, jotka sisältävät ns. tutkavastaanottimia, joiden sisältämät viestit voidaan lukea RFID-lukijoilla. Lukija voi tässä tapauksessa olla NFC-ominaisuudella varustettu mobiililaite.

3. Käytettävyyden määritelmät

Beven ja Cursor [1999] määrittelevät käytettävyyden olevan kuvaus korkeammasta laatutavoitteesta, jolla saadaan aikaan vaikuttavuutta, tehokkuutta ja tyytyväisyyttä. Tämä edellyttää paitsi helppokäyttöisyyttä myös aiheellista toiminnallisuutta, luotettavuutta, tietokoneen suorituskykyä jne. Lyhemmin voidaan todeta sen olevan synonyymi ”käytön laadukkuudelle”, joka on käyttäjän näkemys sovelluksen laadusta. Laadun saavuttaminen vaatii käyttäjäkeskeisen suunnitteluprosessin ja sopivan käytettävyysarviointitekniikan käyttöä ohjelmistoa arvioitaessa.

3.1. ISO 9241-11

ISO 9241-11 on kansainvälinen standardi, joka on laajalti käytetty viite tehtäessä käyttäjälähtöistä suunnittelua ja arviointia vuorovaikutteisille järjestelmille. [Jokela *et al.*, 2003] Standardi kuvaa käytettävyyttä neljästä eri näkökulmasta:

- Vaikuttavuus. Kuinka paikkansapitävästi ja täydellisesti käyttäjät saavuttavat määritellyn tavoitteen?
- Tehokkuus. Kuinka paljon resursseja on kulutettu, jotta saavutetaan tavoite?
- Tyytyväisyys. Käyttäjän tyytyväisyyttä laitteen tai järjestelmän käyttöön, tyytyväisyyttä vuorovaikutuksen sujuvuuteen ja sen tulokseen.
- Käyttöympäristö. Käyttöalue, jonka soveltuvuuteen vaikuttavat esimerkiksi käyttäjien ominaisuudet, tehtävät ja organisaatio

3.2. Jakob Nielsenin määritelmä käytettävyydelle

Jakob Nielsen on yksi tunnetuimista henkilöistä käytettävyystutkimuksen alueella. Hän on laajentanut ISO 9241-11 -standardia opittavuuden, muistettavuuden ja virheiden vähyyden kriteerillä. Käytettävyys voidaan määritellä viiden laatukomponentin kautta [Nielsen]:

- Opittavuus. Miten nopeasti ja helposti uusi vuorovaikutteisen laitteen tai järjestelmän käyttäjä oppii laitteen toimintalogiikan ja käyttämisen?
- Tehokkuus. (Ks. kohta 3.1).
- Muistettavuus. Kuinka helppoa jo aiemmin laitteen käytön oppineen henkilön on palauttaa mieleen laitteen käyttö ja sen toiminnallisuus?

- Virheet. Kuinka paljon virheitä käyttäjät tekevät, kuinka vakavia ne ovat ja kuinka helposti virheistä pystytään toipumaan?
- Tyytyväisyys. (Ks. kohta 3.1).

Lisäksi Nielsen on todennut, että on olemassa myös muita laatutekijöitä, joista yksi tärkeimmistä on hyöty. Hyöty viittaa vuorovaikutteisen järjestelmän suunnitteluun ja toimintaan: Tekeekö järjestelmä sen, mitä käyttäjä tarvitsee. Tämän pohjalta voidaan todeta, että järjestelmän käyttö on merkityksetöntä, jos sen toiminta ei vastaa käyttäjän odotuksia, vaikka sen käyttö olisikin helppoa. [Nielsen]

4. Tutkimuksia NFC-perustaisista matkapuhelinpalveluista

Tässä luvussa esitellään kolme tutkimusta, joissa erilaisia palveluita käytettiin NFC-ominaisuudella varustetulla matkapuhelimella. Tarkoituksena on antaa lyhyt kuvaus tutkimuksista sekä niiden tuloksista.

4.1. Reaalimaailman vuorovaikutuksen kokeminen – käyttäjäkokemuksia NFC kenttätutkimuksesta

Wienin yliopiston tutkimusyksikkö CURE (Center for Usability Research and Engineering) toteutti NFC:n käyttöön perustuvan tutkimuksen, jonka tarkoituksena oli pyrkiä määrittelemään käyttäjien kokemuksia, jotta ymmärrettäisiin paremmin NFC:n tarjoamat mahdollisuudet todellisissa vuorovaikutustilanteissa, joita käyttäjät kohtaavat jokapäiväisessä elämässään. Tutkimus tehtiin kenttätutkimuksena, jossa 75 käyttäjän ryhmä (50 opiskelijaa ja 25 työntekijää) Wienin yliopiston kampukselta käytti viittä erilaista NFC-palvelua matkapuhelimillaan [Geven *et al.*, 2007]:

1. Pääsy kampuksen luentosaleihin. Luentosalin oven lukko avattiin lukemalla tunniste matkapuhelimella. (Kuva 1)
2. Matkapuhelimen tietojen synkronisointi info-terminaalin kanssa. Tarkoituksena oli saada erilasia tietoja terminaalista sekä vastaanottaa uutisia lukemalla terminaalissa oleva tunniste. (Kuva 1)
3. Ostosten tekeminen kahviautomaatista. Lukemalla kahviautomaattiin sijoitettu tunniste, käyttäjällä oli mahdollisuus ostaa kahvia. (Kuva 2)
4. Maksaminen kahvilassa. Käyttäjien oli mahdollista maksaa ostoksena kahvilassa koskettamalla lukulaitetta, joka suoritti maksun. (Kuva 2)
5. Pelin lataaminen mainoksesta. Lukemalla mainokseen sijoitettu tunniste käyttäjien oli mahdollista ladata peli matkapuhelimiinsa.

Kuva 1: Lukon avaamiseen käytetty tunniste ja info-terminaali [Geven *et al.*, 2007]

Kuva 2: Kahviautomaatti ja kahvilassa maksaminen [Geven *et al.*, 2007]

Tuloksia varten käyttäjät tekivät NFC-palveluiden käytöstä päiväkirjoja sekä täyttivät online-kyselyn. Saatuja tuloksia tarkasteltiin kahdesta eri näkökulmasta: ensikokemus NFC:n käytöstä uusien käyttäjien näkökulmasta sekä millaisia käyttömalleja jatkuvassa NFC:n käytössä ilmeni.

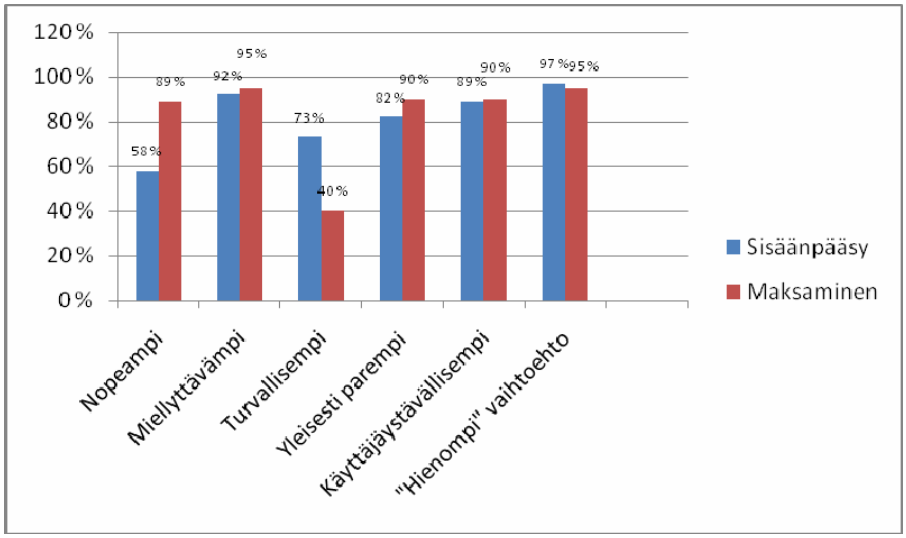
Päiväkirjoilla kerätyistä käyttäjiltä saaduista kokemuksista pystyttiin identifioimaan NFC:n käytön helpottavan käyttäjien jokapäiväistä elämää, esimerkiksi ostoksia oli mahdollista tehdä, vaikka lompakko oli jäänyt kotiin. Lisäksi palvelut koettiin käytännöllisiksi, esimerkiksi info-terminaalin käyttö luentojen välillä.

Käyttäjiltä löytyi myös negatiivisia käyttökokemuksia. Välillä NFC:n toiminta koettiin jossain määrin epävarmaksi, esimerkiksi lukkojen avaaminen ei aina toiminut, jolloin ovista kulkeminen estyi. Lisäksi aina ei ollut täysin selvää, kuinka NFC-palvelua tulisi käyttää tai kuinka toiminta tapahtui. Tästä esimerkkinä voidaan pitää epäselvyystilanteita maksua suoritettaessa (laitteen sijoittaminen) sekä outoa tunnetta siitä, ettei käyttäjän tarvinnut antaa komentoja maksun suorittamisen aloittamiseksi.

Kuitenkin 75 % käyttäjistä oli joko todella tyytyväisiä tai melko tyytyväisiä NFC-palveluihin (2 ensimmäistä vaihtoehtoa 5 asteen arvostelussa). Online-kysely, joka läheteltiin kaikille osallistujille (kyselyn palautti 60/75 osallistujas-

ta), ositti että 83 % osallistujista oli joko todella tyytyväisiä tai melko tyytyväisiä NFC-palveluihin. [Geven *et al.*, 2007]

Päiväkirjojen ja kyselyiden lisäksi käyttäjiä pyydettiin vertaamaan NFC:n avulla tapahtuvaa sisäänpääsyä ja maksamista normaalisti käytettäviin tapoihin. Kuvasta 3 voidaan nähdä, että NFC:n käyttö koettiin kaikin puolin paremmaksi vaihtoehdoksi kuin ”normaali” tapa.



Kuva 3: Arvio NFC-ratkaisun toimivuudesta verrattuna "normaaliin" tapaan [Geven *et al.*, 2007]

4.2. Mobiilisten ratkaisuiden työnkulun ja käytettävyyden parantaminen käyttämällä NFC-teknologiaa

Vuosien 2005-2007 aikana VTT (Valtion teknillinen tutkimuskeskus) toteutti tutkimuksen, jonka tarkoituksena oli identifioida kuinka NFC perusteinen systeemi parantaa mobiilisten ratkaisuiden työnkulkua ja käytettävyyttä. [VTT] Tutkimuksen kohteena oli kuusi pilottihanketta, joiden kesto vaihteli 8-12 viikkoon [Jaring *et al.*, 2007]:

1. Työajan hallinnointi

Kaikki pilottiin osallistujat olivat uimahallin ja kirjastojen huoltotyöntekijöitä. Ennen pilotin toteuttamista osanottajat käyttivät klassista kynä ja paperi -menetelmää työaikojensa seuraamiseen. Pilotin aikana he raportoivat työaikaansa seuraavanlaisella prosessilla: 1) Käyttäjä kosketti hänet yksilöivää ID- tunnistetta vahvistaakseen henkilöllisyytensä laitteen käyttäjänä. 2) Käyttäjä kosketti sijaintitunnistetta vahvistaakseen sijaintinsa. Vaiheet 1 ja 2 toistettiin, kun saavuttiin työpaikalle ja poistuttiin sieltä.

2. Ajoloki

Pilottiin osallistuneiden työ sisälsi paljon ajamista omalla autolla, joten he olivat oikeutettuja korvauksiin jokaisesta ajetusta kilometristä. Korvausten edellytyk-

senä oli matkalaskulomakkeen täyttäminen. Matkalaskulomakkeen täyttöä varten oli pidettävä ajolokia, johon täytettiin yksityiskohtaiset tiedot korvauksiin edellyttävistä matkoista. Pilottijärjestelmässä pyrittiin helpottamaan ajolokin tietojen keräämistä. Pilotin toimintaprosessi oli seuraavanlainen: 1) Käyttäjä kosketti toiminnan aloittavaa tunnistettaan, jolloin sovellus käynnistettiin automaattisesti. 2) Käyttäjä täytti tarvittavat kentät. 3) Tiedot toimitettiin järjestelmään.

3. Logistiikka

Pilottiin osallistujat olivat ravintolan lähettejä, joiden tehtävä oli toimittaa päivittäisiä aterioita vanhuksille. Ennen pilottia käytössä ei ollut virallista valvontaa, joka seuraisi toimituksia. NFC:tä käyttävässä pilotissa toimitusten seurantaprosessi oli toteutettu seuraavalla tavalla: 1) Käyttäjä kosketti hänet yksilöivää ID-tunnistetta, joka vahvisti hänen identiteettinsä. 2) Lähdettäessä reitille, käyttäjä kosketti oikeaa "aloita reitti x"-tunnistetta keittiön seinällä. 3) Kun ateria oli toimitettu vanhukselle, kosketti käyttäjä tunnistetta vanhuksen kotona, millä vahvistettiin ateria toimitetuksi. 4) Kun kaikki ateriat oli toimitettu, käyttäjä päätti reitin koskettamalla ravintolan keittiön seinällä olevaa "päätä reitti"-tunnistetta.

4. Siivous

Pilottiin osallistujien työ sisälsi siivousta sekä kokoushuoneissa että muissa tiloissa. Ennen pilottihanketta jokaisen yksittäisen huoneen siivoukseen käytetty aika oli karkeasti arvioitu siivoojan toimesta, jolloin todellista aikaa ei voitu jäljittää. Pilotissa jokaisen huoneen siivousaika raportointiin seuraavanlaisella prosessilla: 1) Käyttäjä kosketti hänet yksilöivää ID-tunnistetta, joka vahvisti hänen identiteettinsä. 2) Saapuessaan siivottavaan huoneeseen käyttäjä kosketti huoneessa olevaa tunnistetta. 3) Kun siivous oli päättynyt ja käyttäjä oli lähdössä huoneesta, hän kosketti samaa tunnistetta. 4) Työpäivän päätteeksi käyttäjä kosketti ID-tunnistettaan lopettaakseen työvuoronsa.

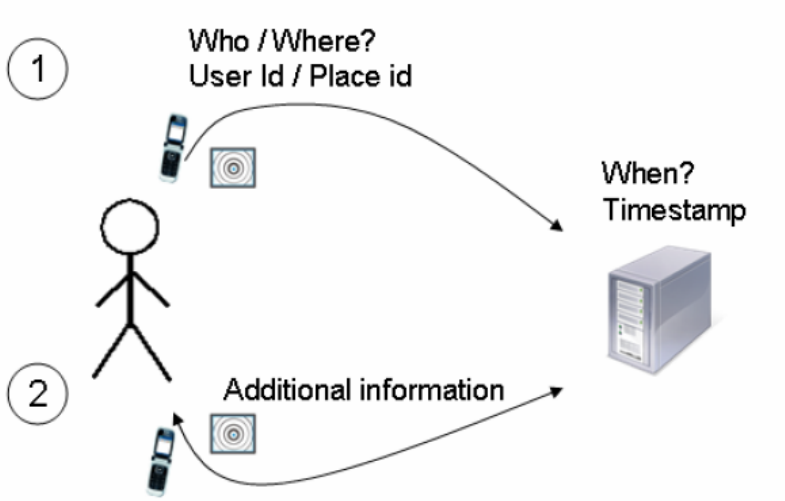
5. Mittarinlukija

Pilottiin osallistuja oli huoltomies, joka rutiininomaisesti raportoi sähkö- ja vesimittarilukemat. Ennen pilotin toteuttamista huoltomies käytti perinteistä kynä ja paperi -menetelmää kirjatakseen mittarin ID-numeron sekä sen lukeman. Tämän jälkeen lukemat siirrettiin Excel-laskentataulukon, joka toimitettiin vastuuhenkilölle. Pilotissa NFC-yhteensopivaa matkapuhelinta käyttämällä, huoltomies raportoi mittarin lukemat seuraavalla tavalla: 1) Huoltomies kosketti tunnistetta mittarin tunnistamiseksi. 2) Huoltomies syötti mittarin lukeman puhelimeen. Vaiheet 1 ja 2 toistettiin, kunnes kaikki mittarit oli luettu.

6. Ylläpito

Pilotissa kaksi kunnossapitotyöntekijää raportoi päivittäin tarkastuksistaan eri puolilla laitosta. Ennen pilotin toteuttamista raportointi tehtiin viikoittain sähköisen manuaalin kautta. Pilotissa NFC-perusteista ratkaisua käyttämällä huoltomiehet ilmoittivat tarkastuksista useita kertoja viikossa matkapuhelimen kautta vastaamalla järjestelmän lähettämiin kysymyksiin. Raportointiprosessi oli seuraavanlainen: 1) Päivän alussa käyttäjä kosketti hänet yksilöivää ID-tunnistetta, joka vahvisti hänen identiteettinsä. 2) Jokaisessa huoltopaikassa käyttäjä kosketti tunnistetta. 3) Käyttäjä vastasi järjestelmän lähettämiin kysymyksiin. 4) Työvuoron lopussa käyttäjä kosketti tunnistetta, joka päätti hänen työvuoronsa. Vaiheet 2 ja 3 toistettiin kaikissa tarkastuspaikoissa.

Kuvasta 4 voidaan nähdä peruskuvaus kaikissa piloteissa käytetystä toimintaprosessista.



Kuva 4: Peruskuvaus pilottien toimintaprosessista [Jaring *et al.*, 2007]

Pilottien tuloksia kerättiin haastattelemalla osallistujia henkilökohtaisesti ja sähköpostiin lähetettävällä kyselyllä. Haastattelussa osallistujat vastasivat avoimiin kysymyksiin koskien pilotin hyödyllisyyttä, käytön aikana ilmenneitä hyötyjä ja haittoja sekä tuntemuksia kosketusparadigman käytöstä. Tulosten pohjalta tutkijat pystyivät identifioimaan seuraavia työnkulkuun ja käytettävyyteen vaikuttavia tekijöitä NFC-palvelun käytöstä:

- Mahdollisuus pienentää tai poistaa päällekkäistä työtä
- Käyttäjien muistikuorman vähentäminen
- Virheiden vähentäminen (tarkkuuden paraneminen)

- Muistikuorman väheneminen
- Työvaiheiden väheneminen (esimerkiksi muutettaessa paperit sähköiseen muotoon)
- Järjestelmän käyttö mahdollisti paremmat valmiudet seurata työntekijöiden toimia
- Lisääntyneet mahdollisuudet havainnollistaa ja todentaa
- Ylimääräisen kulkemisen vähentäminen toimiston ja varsinaisen työtehtäväsuorituspaikan välillä
- Sovellukset tukivat käyttäjien tehtäviä ja tavoitteita
- Palvelut sopivat käyttäjille, koska ne olivat osa luonnollista toimintaa.

Lisäksi havaittiin, että kosketusparadigma koettiin helpoksi tavaksi saada informaatiota, tunnisteen koskettaminen sovellusten käynnistämiseksi koettiin mielekkääksi ja manuaaliseen kirjautumiseen käyttäjien ja paikkojen identifioimisessa ei ollut tarvetta.

Tutkimuksessa myös selvitettiin NFC:n mahdollisuutta ratkaista aikaisempia tehtävissä havaittuja ongelmia. Pilottien jälkeen voitiin todeta, että NFC:n käyttö:

- Mahdollisti pääsyn tietoihin ja sovelluksiin kentällä
- Mahdollisti reaaliaikaisen tiedon päivityksen
- Vähensi käyttäjien tekemiä virheitä
- Mahdollisti pääsyn reaaliaikaisiin ohjeisiin
- Vähensi käyttäjien muistikuormaa
- Mahdollisti henkilön läsnäolon varmistuksen määrätyssä paikassa.

4.3. Kosketusperustainen käyttöliittymä iäkkäille käyttäjille

Tutkimuksessa, jonka toteuttajana toimi VTT, kotihoidossa olevat vanhukset käyttivät NFC-perusteista palvelua. Palvelun kautta vanhusten oli päivittäin mahdollista tilata ja valita ateria, jonka kodinhoitajat heille toimittivat. Ennen sovelluksen toteutusta vanhuksilla ei ollut mahdollisuutta tehdä valintaa, vaan heille kaikille toimitettiin päivittäin sama ateria. Tästä syystä sovelluksen tarjoama valinnanvapaus toi vanhusten elämään myös lisäarvoa. Lisäksi palvelun kautta pystyttiin ateria perumaan pelkästään koskettamalla aterian peruttavaa tunnistetta. Aikaisemmin aterian peruminen oli tehtävä soittamalla ateriapalveluun määrättyyn aikaan mennessä.

Tärkein näkökulma tutkimuksessa oli kosketuspohjaisen käyttöliittymän sopivuuden tutkiminen vanhusten jokapäiväiseen elämään. Tutkimuksen kesto oli 8 viikkoa, osallistujien määrä oli 9 henkilöä ja osallistujien keski-ikä 76,6 vuotta. Useimmilla osallistujilla oli eritasoisia muistihäiriöitä, joiden vuoksi he tarvitsivat apua jokapäiväisessä elämässään.

Sovelluksen käyttöliittymänä toimivat NFC:n käytön mahdollistava matkapuhelin sekä muovisessa jalustassa oleva ruokalista (kuva 5), josta käyttäjä koskettamalla matkapuhelimella haluamaansa ruoka-annoksen tunnistetta, pystyi tekemään ruokatilauksen. [Häikiö *et al.*, 2007]

Week 37	A	B	C
MONDAY	Chicken sauce	Mushroom soup	No meal
TUESDAY	Sauce bolognese	Beef in peppercorn sauce	No meal
WEDNESDAY	Meat soup	Vegetable loaf	No meal
THURSDAY	Beef and potato bake	Salmon bake	No meal
FRIDAY	Game and bacon bake	Ham and vegetable soup	No meal

Kuva 5: Ruuantilausjalusta ja ruokalista [Häikiö *et al.*, 2007]

Tutkimustulosten saamiseksi käytettiin kolmea eri metodia: 1) Haastattelu, joka tehtiin jokaiselle osallistujalle ennen tutkimuksen aloittamista ja tutkimuksen päättymisen jälkeen. 2) Tarkkailu, joka toteutettiin harjoittelujaksossa (tarkoituksena oli selvittää kuinka osallistujat hyväksyivät ja oppivat uuden tekniikan käytön). 3) Osallistujien täyttämät päiväkirjat (perinteinen kynä ja paperi -menetelmällä).

Haastatteluissa monet vanhukset olivat ilmoittaneet, että matkapuhelimen näppäimistö oli liian pieni heidän käyttöönsä, sillä tärisevät kädet tai muutenkin heikentyneet hienomotoriset taidot aiheuttivat ongelmia matkapuhelimen käytössä. Tästä syytä osallistujat totesivat, että kosketusperustaisen käyttöliittymän etuna oli, että heidän ei tarvinnut käyttää matkapuhelimen näppäimiä. Osallistujat myös oppivat ja hyväksyivät nopeasti kosketuksen vuorovaikutustapana. [Häikiö *et al.*, 2007]

Malli, jota käytettiin ruuan tilaamiseen (kosketus tunnisteeseen ruokalistassa) ei tuottanut ongelmia osallistujille, vaikka heillä kaikilla oli jonkinasteisia muistihäiriöitä. Ongelmana kuitenkin nähtiin, että osalla käyttäjistä oli vaikeuksia löytää oikea etäisyys tunnisteen lukemiseen ruokalistasta. Lisäksi sovelluksen palautteen huomaaminen onnistuneesta tilauksesta tuotti joillekin osallistujille vaikeuksia. Tämä kuitenkin johtui usein matkapuhelimen näytön liian pienestä koosta eikä itse palautteenantotavasta.

Yleisesti tulosten perusteella voitiin todeta, että vanhusten asenteet uutta teknologiaa kohtaan olivat pääsääntöisesti positiivisia: he oppivat nopeasti kosketusperustaisen käyttöliittymän käytön ja pystyivät käyttämään sitä fyysisistä

ja kognitiivisista heikkouksista huolimatta. Lisäksi kosketus vuorovaikutustapana matkapuhelinsovelluksessa voitiin todeta olevan intuitiivinen ja luonnollinen tapa valita kohteita. Tästä syystä se sopii hyvin toimimaan ja integroitumaan sovelluksiin, joita vanhukset käyttivät jokapäiväisessä elämässään.

5. Johtopäätelmät ja yhteenveto

Tutkimustietoa NFC:tä käyttävistä matkapuhelinpalveluista on vielä rajallisesti saatavana, joten aihealueen tutkiminen asetti haasteita. Aihealueeseen viittaavaa tutkimusta on tehty esimerkiksi RFID:n käytöstä, mutta kuitenkaan niissä ei ole liitetty RFID:tä osaksi matkapuhelinpalveluita. Lisäksi useimmat tutkimukset NFC:n käytöstä matkapuhelinpalveluissa ovat salaisia. Tutkimukset on toteutettu matkapuhelinvalmistajien ja -operaattorien yhteistyössä, jolloin tutkimusjärjestelyihin ja -tuloksiin ei ole mahdollista tutustua tiedejulkaisuiden pohjalta. [TeliaSonera, Elisa]

Edellä mainituissa tutkimuksissa esiteltiin NFC-tekniikkaan perustuvia matkapuhelimella käytettäviä palveluita yhteensä 12 kappaletta. Palveluiden käyttöympäristöinä olivat lukkojen avaaminen, maksaminen, tiedonhaku, tunnistus sekä palvelun tilaaminen. ISO 9241-11 ja Nielsenin käytettävyyttä koskevien laatukomponentteihin nojaten [Jokela *et al.*; 2003, Nielsen] voidaan tutkimusten tuloksista identifioida seuraavia käytettävyyttä parantavia tekijöitä:

- Palvelun käyttö helpotti käyttäjien jokapäiväistä elämää
- Toteutetut palvelut olivat käytännöllisiä
- Verrattaessa palveluita vanhoihin käytössä olleisiin ratkaisuihin pystyttiin työmääriä ja päällekkäistä työtä vähentämään
- Käyttäjien muistikuorma väheni
- Käyttäjien tekemien virheiden määrä väheni
- Mahdollisti reaaliaikaisen tiedonseurannan ja päivityksen
- Toteutetut palvelut sopivat käyttäjille ja käyttöympäristöön
- Palvelut koettiin mielekkäiksi ja helpoksi käyttää
- Tekstin syötön matkapuhelimen näppäimistön kautta vähentyi (palvelusta riippuen ei tarvittu ollenkaan)
- Palvelun käyttö oli mahdollista, vaikka käyttäjillä oli fyysisiä tai kognitiivisia heikkouksia
- Palvelun ja tekniikan käyttö olivat nopeasti opittavissa
- Käytetty tekniikka oli nopeasti ja helposti omaksuttavissa.

	Vaikuttavuus	Tehokkuus	Tyytyväisyys	Käyttöympäristö	Opittavuus	Muistettavuus	Virheet
Helpotti jokapäiväistä elämää		X	X				
Palvelut käytännöllisiä				X			
Vähensi työmääriä verrattaessa vanhoihin ratkaisuihin		X					
Vähensi muistikuormaa		X	X				
Vähensi käyttäjien tekemiä virheitä							X
Tiedon seurannan mahdollisuus		X					
Tiedon tarkkuus	X						
Palvelut soveltuivat käyttäjille / tilanteisiin			X	X			
Tekstin syötön vähentäminen		X	X				X
Mielekäs ja helppo käyttää			X	X	X		
Käyttö mahdollista vaikka käyttäjällä fyysisiä tai kognitiivisia heikkouksia					X	X	

Taulukko 1: Käytettävyyttä parantavia tekijöitä

Taulukkoon 1 on kerätty tutkimuksista identifioidut tekijät, joilla on positiivinen vaikutus matkapuhelinpalveluiden käytettävyyteen. Taulukon ensimmäisestä sarakkeesta löytyvät identifioidut tekijät. Riviltä voidaan nähdä, mihin käyttävyyteen liittyviin laatutekijöihin löydöstä voidaan verrata.

Tutkimuksista saatujen löydösten perusteella voidaan todeta NFC:n käytön parantavan matkapuhelinpalveluiden käytettävyyttä tarjoamalla helposti opittavan ja omaksuttavan käyttöliittymäparadigman niin vanhemmille kuin nuorillekin käyttäjille. NFC:n käytön omaksuminen ei vaadi edeltävää tietoa tai kokemusta käytettyyn tekniikkaan. Myöskään heikentyneet motoriset taidot eivät tee kosketuskäyttöliittymän käytöstä vaikeaa. Yleistäen voidaan todeta,

että NFC mahdollistaa suoran ja luonnollisen vuorovaikutuksen, joka vähentää käyttäjien kognitiivista kuormitusta vähentämällä muistikuormaa.

Tutkimuksista voitiin myös huomata palvelun suunnittelussa huomioon otettavia tekijöitä. Palvelun käyttö tulisi suunnitella niin, että NFC:n tuomat parannukset käytettävyyteen pystyttäisiin hyödyntämään parhaalla mahdollisella tavalla. Käytettäessä palvelua ensimmäistä kertaa käyttöönoton ja vuorovaikutusmallin tulisi olla selkeitä käyttäjille. Lisäksi vuorovaikutuksen tilan näyttö tulisi olla selkeää annettaessa palautetta vuorovaikutuksen etenemisestä.

Kaiken kaikkiaan mahdollisuudet, joita NFC tuo matkapuhelinpalveluiden käyttäjille, ovat hyvin mielenkiintoisa HCI (Human Computer Interaction) -näkökulmasta. Teknologia tarjoaa mahdollisuuden realisoida kosketusparadigmaa sekä toteuttaa intuitiivisia, helppokäyttöisiä ja turvallisia langattomia palveluita [NFC Forum].

Viiteluettelo

[Beven and Curson, 1999] Nigel Bevan and Ian Curson, Planning and implementing user-centred design. In: *CHI '99 Extended Abstracts on Human Factors in Computing Systems*. ACM (1999) 137-138.

[Elisa] NFC in Helsinki transportation http://www.nfc-italy.com/?p=21 Checked 7.12.2008.

[Geven *et al.*, 2007] Arjan Geven, Peter Strassl, Bernhard Ferro, Manfred Tscheligi and Harald Schwab, Experiencing real-world interaction – results from a NFC user experience field trial. In: *MobileHCI '07: Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services*, ACM (Sept. 2007) 234-237.

[Häikiö *et al.*, 2007] Juha Häikiö, Arto Wallin, Minna Isomursu, Heikki Ailisto, Tapio Matinmikko and Tua Huomo, Touch-based user interface for elderly users. In: *MobileHCI '07: Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services,* ACM (Sept. 2007), 289-296.

[Jaring *et al.*, 2007] Päivi Jaring, Vili Törmänen, Erkki Siira and Tapio Matinmikko, Improving mobile solution workflows and usability using near field communication technology. In: *Ambient Intelligence, European Conference, AmI 2007, Lecture Notes in Computer Science* **4794** (2007), Springer 358-373.

[Jokela *et al.*, 2003] Timo Jokela, Netta Iivari, Juha Matero and Minna Karukka, The standard of user-centered design and the standard definition of usability: analyzing ISO 13407 against ISO 9241-11. In: *Proceedings of the Latin American Conference on Human-Computer Interaction*, ACM (2003) 53-60.

[NFC Forum] NFC Forum, 2008, http://www.nfc-forum.org/home. Checked 15.10.2008.

[Nielsen] Jakob Nielsen Biography http://www.useit.com/jakob/. Checked 7.12.2008.

[Nokia] Nokia Suomi http://www.nokia.fi/. Luettu 10.12.2008.

[Nokia White Paper] Nokia White Paper – Near Field Communcation, 2008, http://www.nokia.com/NOKIA_COM_1/Press/Mateals/White_Papers/pdf_files/White%20paper_Nokia_Near%20field%20communication.pdf Checked 15.10.2008.

[TeliaSonera] TeliaSonera http://www.teliasonera.com/press/pressreleases/item.page?prs.itemId=234201. Checked 7.12.2008.

[VTT] VTT – Valtion teknillinen tutkimuskeskus, 2008, http://www.vtt.fi/. Luettu 7.12.2008.

[Välkkynen et al., 2006] Pekka Välkkynen, Marketta Niemelä and Timo Tuomisto, Evaluating touching and pointing with a mobile terminal for physical browsing. In: *4th Nordic Conference on Human-Computer Interaction Changing Roles,* ACM (Oct. 2006) 14-18.

Toiminnanohjausjärjestelmän hankinta

Tomi Saarinen

Tiivistelmä.

Tutkielma käsittelee toiminnanohjausjärjestelmän hankintaa toimittajan ja tuotteen valitsemiseen asti. Tutkielmassa keskitytään toiminnanohjauksesta saataviin hyötyihin hankinnan perusteena, hankinnan valmistelun vaihe vaiheelta sekä lopullista toimittajan ja tuotteen valitsemiseen kilpailuttamisen avulla.

Avainsanat ja sanonnat: Toiminnanohjaus, ERP, tietojärjestelmä, hankinta, kilpailuttaminen.
CR-luokat: D.2.1, H.4.1, H.4.2, J.1, K.6.0,

1. Johdanto

Toiminnanohjausjärjestelmä (enterprise resource planning, ERP) eli yrityksen liiketoimintaprosessit yhdistävä ohjelmakokonaisuus on hyvin ajankohtainen. Alati verkostoituvilla markkinoilla yritysten on pystyttävä siirtämään tietoa tehokkaasti ja reaaliaikaisesti sekä yrityksen sisällä että ulkopuolella.

Tutkielmassani keskityn ERP-järjestelmän hankintaprosessiin toimittajan ja tuotteen valintaan asti. Aloitan kertomalla toiminnanohjauksesta yleisesti ja siirryn hankinnasta saataviin hyötyihin. Yksikään investointi ei ole perusteltu, ellei sillä saavuteta organisaatioon lisäarvoa. ERP:n hankinta mahdollistaa yrityksen liiketoimintaprosessien tehostumisen ja liikevaihdon kasvun.

Neljännessä luvussa käyn läpi toiminnanohjausjärjestelmän hankintaan liittyviä valmisteluja. Mitä suurempaa järjestelmää ollaan hankkimassa, sitä enemmän projekti vaatii suunnittelua. Hyvin suunnitellun hankkeen onnistumismahdollisuudet ovat paljon suuremmat kuin hätäisesti suunnitellun.

Viidennessä luvussa kuvaan seitsenvaiheisen mallin ohjelmistotoimittajan ja ratkaisun valintaan. Malli etenee toimittajakandidaattien valinnasta ja karsinnasta aina lopullisen toimittajan valintaan ja sopimusneuvottelujen käynnistämiseen.

Lopuksi kiteytän olennaisimmat osat tutkielmastani yhteenvedon muodossa.

2. Toiminnanohjaus

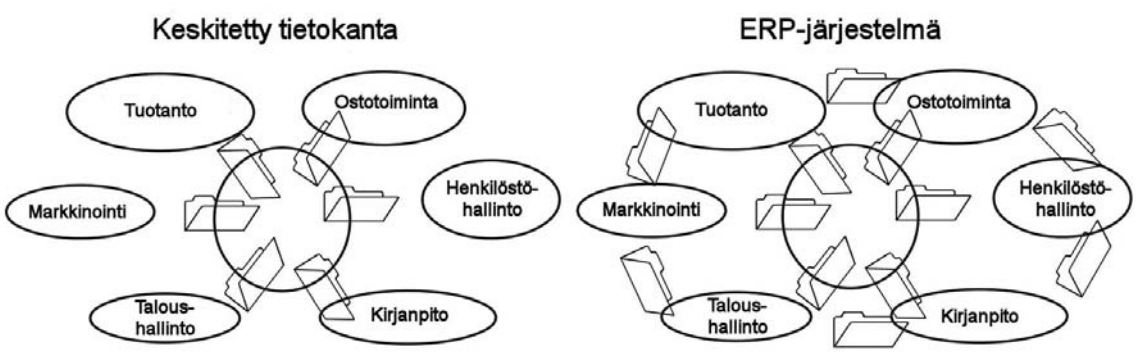
Organisaatiot verkostoituvat kovalla vauhdilla liiketoimintojen, liiketoimintayksikköjen ja yritysten kesken. Tietoa tarvitaan entistä enemmän ja nopeammin. Lisäksi reaaliaikainen seuranta on tullut välttämättömäksi. Tämä aiheuttaa haasteita tietojärjestelmille. Ratkaisuna on ERP-järjestelmät. Internet mahdollistaa tiedon kulun organisaatioiden välillä, ERP organisaatioiden sisällä.

Luvussa määritellään toiminnanohjaus käsitteenä sekä käydään läpi toiminnanohjauksen historiaa 1960-luvulta nykypäivään.

2.1. Toiminnanohjauksen määritelmä

Toiminnanohjauksella tarkoitetaan ohjelmakokonaisuutta, joka mahdollistaa automaattisen liiketoimintaprosessien hallinnan. Toiminnanohjauksen idea on integroida koko yrityksen resurssit yhdeksi kokonaisuudeksi. Integroinnilla tarkoitetaan linkittämistä, yhdistämistä ja yhteen liittämistä. Tietojen päällekkäisyys vähenee, aikaa säästyy ja operationaalinen tehokkuus kasvaa. Integrointi ei rajoitu pelkästään tietoon, vaan sitä voidaan käyttää myös liiketoimintojen uudelleenorganisoinnissa. [Nah, 2002]

Toiminnanohjaus eroaa perinteisestä tietokantakeskeisestä tietojärjestelmästä siten, että tietokantakeskeisessä järjestelmässä eri toiminnot tallentavat tiedon tietokantaan, josta se jaetaan. ERP-järjestelmässä toiminnot on linkitetty toisiinsa ja tietokantaan. Tämä tarkoittaa, että kokonainen ERP-järjestelmä mahdollistaa eri liiketoimintojen kuten tuotannon, taloushallinnon, kirjanpidon, henkilöstöhallinnon jne. yhteen liittämisen. ERP:n ydinajatus on tietokantakeskeisen järjestelmän tapaan tiedon jakaminen, mutta tietokantakeskeisestä järjestelmästä poiketen myös prosessien integrointi. Näihin kahteen ideaan perustuva ERP mahdollistaa todellisen resurssien optimoinnin ja näin ollen kustannustehokkuuden ja liikevaihdon kasvun. [Nah, 2002]

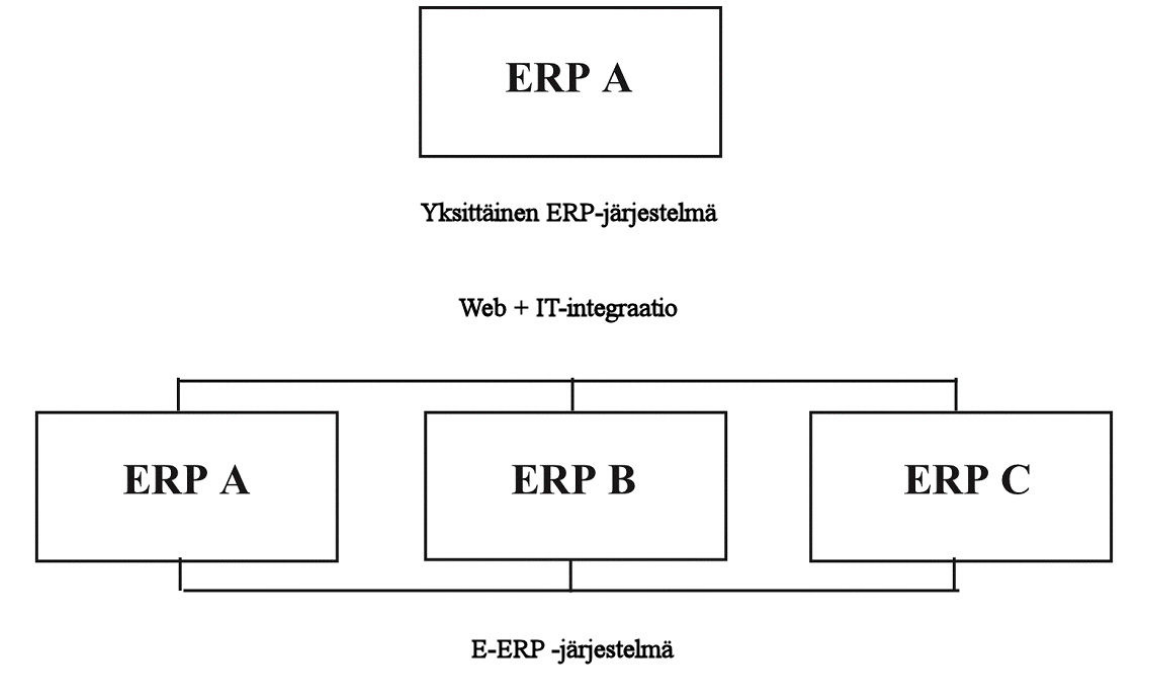


Kuva 1. ERP:n ja tavallisen tietokantajärjestelmän erot [Nah, 2002].

2.2. Toiminnanohjauksen historia

Toiminnanohjauksen kehittyminen seuraa melko läheisesti tietokoneiden ja ohjelmistojen kehitystä aikojen saatossa. 1960-luvulla yritykset alkoivat kehittää järjestelmiä varastoiden hallinnan automatisointiin. MRP-järjestelmät (material requirements planning, MRP) kehitettiin 1970-luvulla tuotannonohjausta varten. 1980-luvulla seurasi MRP II -järjestelmä (manufacturing resources planning, MRP II), jolla pystyttiin optimoimaan valmistusprosesseja. Järjestelmä sisälsi esimerkiksi työntekijäportaan- ja jakelun hallinnan, projektijohdon, taloushallinnon, henkilöstöhallinnon ja suunnittelun. Nämä järjestelmät olivat kuitenkin vielä heikosti integroituja yritysten muihin järjestelmiin. Varsinainen ERP julkaistiin 1980- ja 1990-luvun vaihteessa. Tällöin järjestelmä oli täydentynyt vielä laajemmilla integraatioilla. [Nah, 2002]

2000-luvun verkostoituneessa yhteiskunnassa yritysten on ollut pakko laajentaa järjestelmiään e-kaupan saralle. Perinteinen ERP on yritysten sisäistä käyttöä varten, kun taas e-kaupalla ja webillä laajennettu E-ERP-järjestelmä (extended ERP) luo lisäarvoa markkinoille ja toimialalle (kuva 2.). [Nah,2002]



Kuva 2. Perinteisen ERP-järjestelmän ja E-ERP-järjestelmän ero [Nah, 2002].

3. Hankinnan perusteet

Tietojärjestelmät ovat yritysten investointeja siinä missä muutkin investoinnit. Järjestelmillä pyritään saamaan samoja hyötyjä kuin tavallisilla prosessilaitehankinnoilla tai palveluyritysten toimitilainvestoinneilla. Tästä johtuen tietojärjestelmäinvestointeja tulee käsitellä samoin kuin muitakin investointeja. [Kettunen, 2002]

Tietojärjestelmien kehittämistä ja arviointia on kritisoitu 1990-luvun alusta alkaen, sillä niiden tuottavuus on ollut heikko. Tietojärjestelmän tuottavuutta on tosin vaikeampi mitata kuin esimerkiksi teollisuuden laitehankintoja. [Ruohonen ja Salmela, 2003]

Tässä luvussa käsittelen toiminnanohjauksen hyötyjä organisaatiolle. Lisäksi esittelen tutkimuksen toiminnanohjauksen hankinnan vaikutuksesta yrityksen suorituskykyyn ja liiketoimintaprosessien tehostumiseen.

3.1. Toiminnanohjauksen hyödyt

Organisaatio voi saada hyötyä ERP:n hankinnasta monella tapaa. Davenport [2000] jakaa hyödyt kolmeen alalajiin: säästöt uudesta lähestymistavasta työhön, säästöt vanhanaikaisesta järjestelmästä luopumisesta sekä liikevaihdon kasvattamisen edut.

ERP mahdollistaa säästöjä monella tapaa. Järjestelmällä voidaan automatisoida tehtäviä, jotka ovat aikaisemmin hoitaneet työntekijät. Lisäksi ERP mahdollistaa säästöjä prosesseja muuttamalla. Tietoa voidaan jakaa organisaation sisällä entistä vapaammin. ERP luo säästöjä myös tuotannossa esimerkiksi varastoinnin ja henkilöstöresurssien muodossa. [Davenport, 2000]

Kun organisaatio vaihtaa vanhasta järjestelmästä ERP-järjestelmään, loppuu vanhan järjestelmän päivittämisen tarve. Toisaalta myös ERP-järjestelmää täytyy päivittää, mutta esimerkiksi uuden moduulin lisääminen ERP-järjestelmään on vaivattomampaa. Monet organisaatiot vaihtoivat ERP-järjestelmään vuosituhannen vaihteessa, jotta tästä johtuvia laajoja ja kalliita korjauksia ei tarvinnut tehdä. [Davenport, 2000]

Liikevaihdon kasvu on merkittävä hyöty ERP-järjestelmää hankittaessa. ERP mahdollistaa asiakaspalvelun parantamisen, paremmat mahdollisuudet tilausten tekemiseen asiakkaille, kilpailukykyisemmän hinnoittelun ja paremman tuottavuuden. Nämä yhdessä mahdollistavat liikevaihdon kasvattamisen. [Davenport, 2000]

Ruohonen ja Salmela [2003] lisäävät listaan vielä muutamia tavoiteltavia hyötyjä: hallintokustannusten vähentäminen, toimitusten tarkkuuden paraneminen, raportoinnin ja valvonnan kehittäminen, lyhyemmät läpimenoajat sekä laadun paraneminen.

Wieder ja muut [2006] esittelevät tutkimuksessaan ERP:n hankinnan vaikutusta yrityksen toiminnan tehostumiseen. Tutkimus suoritettiin posti- ja puhelinkyselynä. Karsinnan jälkeen otoskooksi saatiin 106 Australiassa toimivaa yritystä. Yrityksistä 48 prosenttia oli kokonaan australialaisomistuksessa, 20 prosenttia pääosin australialaisomistuksessa, 23 prosenttia pääosin tai kokonaan ulkomaisessa omistuksessa. Loput 9 prosenttia yrityksistä ei vastannut tähän kysymykseen. Yritysten keskimääräinen liikevaihto oli 7,506 MAuD.

Tutkimuksessa keskityttiin tutkimaan yritysten suorituskykyä ja liiketoimintaprosessien tehokkuutta. Mittaaminen perustui yritysten tunnuslukuihin. Tutkimuksessa käytettiin suorituskyvyn mittaamiseen nettovoittomarginaalia ja maksuvalmiutta. Liiketoimintaprosessien tehokkuutta mitattiin voitolla jaettuna pitkäaikaisilla varoilla, myyntisaatavien kiertonopeudella ostovelkojen kiertonopeudella varaston kiertonopeudella ja liikevaihdolla. Kannattavuuden mittaamiseen käytettiin pääoman tuottoprosenttia, sijoitettua pääomaa ja käyttökatetta.

Tutkimuksessa kävi ilmi, että ERP-hanke alkaa merkittävästi vaikuttaa mitattuihin arvoihin odotettua myöhemmin, vasta 4-5 vuoden kuluttua käyttöönotosta. Yritysten ei tulisikaan odottaa nopeita tuloksia käyttöönoton jälkeen. ERP-hankkeesta saatava hyöty lisääntyy järjestelmän käytön oppimisen myötä. Mitä kauemman käyttöönotosta oli kulunut aikaa, sitä positiivisemmat olivat tulokset yritysten liiketoimintaan. Lisäksi toimitusketjunhallintajärjestelmällä täydennetyn ERP:n vaikutus liiketoimintaprosesseihin oli erityisen positiivinen.

3.2. Väärät hankintaperusteet

Kettusen [2002] mukaan osa yrityksistä on valinnut strategiakseen hankkia uudet tietojärjestelmät ensimmäisten joukossa ilman tarkempia tuottavuuslaskelmia. Ajattelumalli tähtää siihen, että uuden teknologian varhaisella käyttöönotolla voidaan saavuttaa kilpailuetua markkinoilla, vaikka uuden teknologian vaikutusta markkinoihin ei vielä tiedettäisikään.

Useimmiten väärät investoinnit tietojärjestelmiin johtuvat yrityksen IT-osaston kiinnostuksesta uusiin järjestelmiin. Mikäli osastolla on hyvin itsenäinen asema, eikä investointeja hyväksytetä ylemmällä tasolla, voidaan järjestelmiä ostaa ilman taloudellista pohjatarkastelua. Tietojärjestelmien rooli yritysmaailmassa on niin tärkeä, että kaikki tärkeimmät IT-hankkeet tulisi käsitellä yrityksen johtoryhmässä tai jopa hallituksessa saakka tarkkojen hyötyanalyysien sekä kannattavuuslaskelmien kanssa. [Kettunen, 2002]

4. Hankinnan valmistelu

Mitä suurempaa tietojärjestelmää ollaan hankkimassa, sitä enemmän projekti vaatii suunnittelua. Hankinnan valmistelu tuottaa hyväksytyn hankintasuunnitelman, jonka osia voidaan käyttää myöhemmin myös tarjouspyynnön pohjana. Valmistelun vaikutusta hankintaan ei pidä väheksyä, sillä mitä paremmin hankinta on suunniteltu, sitä tehokkaammin ja edullisemmin se onnistuu. [Tietotekniikan liitto, 2005]

Valmistelun käynnistää toimeksianto. Toimeksianto tulee yleensä organisaation liiketoiminta- tai tietotekniikkastrategiasta. Toimeksiannossa kuvataan lyhyesti, mitä ollaan hankkimassa ja miksi ollaan hankkimassa. Johdon tuki hankinnan ohjauksessa on erittäin tärkeä. [Tietotekniikan liitto, 2005]

Ennen hankintaprosessin käynnistämistä laaditaan selvitys nykyisestä infrastruktuurista, henkilöresursseista ja tietojärjestelmistä. Selvityksen pohjalta luodaan kuvaus nykyisestä toimintaympäristöstä. [Kettunen, 2002]

Kuvaus on tärkeä jatkon kannalta, sillä toimittajat voivat tarkastella toimintaympäristöä. Lisäksi toimittajat saavat tietoa mahdollisista kehitystarpeista projektiin liittyen. Myös toisiin järjestelmiin integroitaessa nykyisen toimintaympäristön kuvauksesta on hyötyä tietojärjestelmää suunniteltaessa. [Kettunen, 2002]

Tässä luvussa käsitellään hankinnan valmistelun vaiheet yksi kerrallaan.

4.1. Järjestelmävaatimukset

Järjestelmän keskeiset vaatimukset tulee määritellä valmisteluvaiheessa. Vaatimuksien lisäksi taustaksi tarvitaan usein tavoitetilan kuvaamista. Tavoitetilan toimintoja voidaan kuvata prosessikuvauksina, joista lisää myöhemmin. [Tietotekniikan liitto, 2005] Vaatimusmäärittelyssä voidaan kuvata myös ei-toiminnallisia vaatimuksia, kuten suorituskykyä, ylläpidettävyyttä ja tukipalvelujen saatavuutta [Kettunen, 2002].

Kettusen [2002] mukaan Vaatimusmäärittelyn laatiminen on tietojärjestelmäprojektin vaiheista kaikkein tärkein onnistuneeseen lopputulokseen vaikuttava tekijä. Ellei vaatimusmäärittelyä ole tehty kunnolla, eivätkä asiakas ja toimittaja ole sitoutuneet siihen kunnolla, on hankkeella huonot mahdollisuudet onnistumiseen.

Järjestelmävaatimusten kuvaaminen on yleensä valmisteluvaiheen työläin vaihe. Tähän saatetaan tarvita jopa kymmeniä tekniikan ja kohdealueen asiantuntijoita ja tulevia käyttäjiä. [Tietotekniikan liitto, 2005]

Projektin alussa osapuolilla saattaa olla hyvin erilainen käsitys lopputuotteesta. Jos prosessit ja käyttötilanteet on kuvattu riittävän selvästi, kommunikointi helpottuu. Alustava tietomalli ja kuvatut liittymät toisiin järjestelmiin helpottavat tilannetta entisestään. Vaatimusmäärittelyn taso ennustaa lopputuloksen tasoa. Kaiken toiminnan, myös hyväksymistestien, tulisi perustua vaatimusmäärittelyyn. [Tietotekniikan liitto, 2005]

4.1.1. Vaatimusmäärittelyn laatiminen

Vaatimusmäärittely jakautuu yleensä asiakkaan sekä toimittajan laatimaan vaatimusmäärittelyyn. Asiakkaan laatima määrittely kuvaa asiakkaan tarpeita järjestelmälle sisältäen olemassa olevan tietoteknisen infrastruktuurin, toiminnalliset tavoitteet hankittavalle järjestelmälle sekä rajaukset. [Kettunen, 2002]

Toimittaja yleensä täydentää vaatimusmäärittelyä tietoteknisillä ratkaisuilla sekä entisestään tarkennetuilla toiminnallisilla vaatimuksilla. [Kettunen, 2002]

Vaatimusmäärittelyn tarkan laatimisen merkitys korostuu tarjouksia pyydettäessä. Toimittajien tarjoukset projektin läpiviemiselle perustuvat juuri vaatimusmäärittelyyn. Mitä tarkempi vaatimusmäärittely on, sitä parempia ja vertailukelpoisempia tarjouksia toimittajaehdokkailta saadaan. [Kettunen, 2002]

Kettusen [2002] mukaan varsinaisen vaatimusmäärittelydokumentin tulisi sisältää seuraavat kohdat:

1. Rakennettavan palvelun yleiskuvaus
    - järjestelmästä saatavat hyödyt, käyttäjät, termistö
2. Rakennettavan palvelun toiminnalliset vaatimukset
    - syötetiedot, toiminnallisuudet, ulostulevat tiedot, toimintojen priorisointi
3. Projektin vaiheistus
    - kehitysmallista riippuen
4. Rajaukset
    - mitä ei tarvitse ottaa huomioon, mitä järjestelmä ei saa tehdä
5. Ympäristö
    - tietoteknisen ympäristön kuvaus
6. Palvelun integrointitarpeet
    - muiden järjestelmien kuvaukset, rajapinnat ja liittymät
7. Käyttäjämäärät ja skaalautuvuustarpeet
    - arvioidut käyttäjämäärät, tietomäärät ja aikatavoitteet
8. Tietoturva
9. Riskianalyysi
    - riskit teknologiassa, organisaatiossa ja toimittajaan liittyvät
10. Muut huomioitavat asiat
    - ylläpito ja tukipalvelut.

4.1.2. Prosessikuvaukset

Scheerin ja Habermannin [2000] mukaan prosessikuvaukset ovat tärkeitä erityisesti toiminnanohjausjärjestelmää hankittaessa. Liiketoimintaprosessit ovat hyvin monimutkaisia, jolloin prosesseja ei välttämättä voida kuvata summittaisella kuvauksella. Toimittajien tekemät referenssimallit antavat hyvän suunnan, joista prosessikuvaukset voidaan muokata omaan liiketoimintaan sopiviksi. Jos liiketoimintaprosessit ovat suunniteltu ja kuvattu kunnolla, ERP-järjestelmät voidaan määritellä prosessien mukaisiksi.

4.2. Perusarkkitehtuuri

Teknisten vaatimusten keskeisin osa on hankittavan järjestelmän teknisen arkkitehtuurin määritteleminen. Teknisen arkkitehtuurin määritteleminen kattaa teknisiä perusvalintoja, joita ovat muun muassa käyttöjärjestelmäympäristö, tietokantajärjestelmä, hakemistoratkaisut, ohjelmointikielet, erilaiset tietomuotojen standardit sekä rajapinnat. Tietotekniikan liitto [2005] lukee tärkeimmiksi arkkitehtuuriin vaikuttaviksi tekijöiksi:

- yrityksen olemassa oleva tietotekninen infrastruktuuri ja tietojärjestelmät
- tarvittavat yhteydet yrityksen asiakkaisiin, yhteistyökumppaneihin ynnä muihin ulkoisiin järjestelmiin
- käytettävissä ja saatavilla olevat henkilöresurssit ja palvelut
- hankittavalle järjestelmälle asetettavat vaatimukset.

Myös toimittaja voi tarkentaa tekniset vaatimukset. Tällöin tulisi kuitenkin kuvata yrityksen käyttämistä teknologioista tai muista syistä johtuvat rajoitukset. [Tietotekniikan liitto, 2005]

4.3. Kustannuslaskelmat

Vaatimusmäärittelyn jälkeen yrityksellä on yleensä tarkka kuva siitä, millainen järjestelmä tullaan rakentamaan ja millaisia infrastruktuurihankintoja joudutaan tekemään. Näiden pohjalta ei kuitenkaan voida vielä luoda täsmällistä budjettia, sillä se onnistuu vasta tarjousten vastaanoton jälkeen. Kuitenkin näillä tiedoilla voidaan määritellä hinta-arvio. [Kettunen, 2002]

Kustannuslaskelmia tehtäessä on pyrittävä huomioimaan kokonaiskustannukset, käyttöönotto ja järjestelmän käytöstä aiheutuvat kustannukset. Kustannuksista suurimman osan uskotaan muodostuvan vasta käyttöönoton jälkeen. Tällaisia kustannuksia ovat muun muassa käyttäjien koulutus, järjestelmän yl-

läpidon resurssit, ylläpitomaksut, mahdolliset ulkoistetut palvelinkustannukset sekä jatkokehityskustannukset.

Kustannuslaskelman jälkeen tulee tehdä laskelmat saatavien hyötyjen ja tuottojen suhteen. Laskemien perusteella voidaan päättää, mikä on hankkeen takaisinmaksuaika ja kannattavuus. Takaisinmaksuajan laskeminen voi olla joissain tapauksissa vaikeata, sillä saatavia hyötyjä voi olla vaikea muuttaa rahaksi. [Kettunen, 2002]

5. Ohjelmistotoimittajan ja ratkaisun valinta

Toimittajan valinta on monimutkainen prosessi. Organisaation tulisi käyttää toimittajakandidaattien, ohjelmistojen ja käyttöönottokumppaneiden valintaan korkeintaan kuusi kuukautta. Itse valinta ei suoranaisesti luo lisäarvoa organisaatiolle, joten valinta tulisi suorittaa mahdollisimman nopeasti. [Davenport, 2000]

Kettunen [2002] esittää ohjelmistotoimittajien kilpailuttamiseen ja valintaan kuusivaiheista mallia:

1. toimittajakandidaattien valinta
2. toimittajien esikarsinta
3. tarjouspyynnöt vaatimusmäärittelyyn pohjautuen
4. tarjousten arviointi ja pisteytys
5. jatkoneuvottelut ja tuotteeseen tutustuminen
6. toimittajan valinta.

Tietotekniikan liitto [2005] lisää vaiheisiin (7.) sopimusneuvottelut. Tässä luvussa esittelen kuvan 3 mukaisen seitsenkohtaisen mallin.

5.1. Toimittajakandidaattien valinta

Tietojärjestelmää ostettaessa toimittajakandidaattien löytäminen ei ole aivan yksinkertaista. Ostajat eivät välttämättä tunne osaajia, eivätkä siten löydä potentiaalisia toimittajaehdokkaita. Kettunen [2002] nostaa esille muutamia keinoja toimittajien etsimiseksi, joita ovat muun muassa internetin tietopankit sekä IT-alan julkaisut ja seminaarit. Erityisesti seminaareissa pääsee keskustelemaan alan yritysten edustajien kanssa ja saa näin luotua hyödyllisiä verkostoja.

Toimittajakandidaattien määrä kannattaa pitää aluksi suurena. Tämä mahdollistaa riittävän määrän kandidaatteja, joista voidaan valita varsinaiset toimittajaehdokkaat. Julkisissa hankkeissa tarjousprosessi on julkinen, joten toimittajia saadaan varmasti tarpeeksi ilman erityistä etsintää. [Kettunen, 2002]

Kuva 3. Ohjelmistotoimittajien kilpailuttamisen vaiheet. Mukaillen Kettunen [2002].

5.2. Toimittajien esikarsinta

Esikarsinta tulee kyseeseen tapauksissa, joissa toimittajakandidaatteja on paljon ja joissa osalla toimittajakandidaateista ei ole tarvittavaa osaamista ja uskottavuutta hankkeen toimittamiseen. Tällöin kilpailutusvaiheessa resursseja säästyy, ja ne voidaan keskittää potentiaalisten toimittajakandidaattien kanssa keskusteluun. Kettusen [2002] mukaan tietojärjestelmäprojektin kilpailuttamisessa voidaan ilman aikataulun venymistä kilpailuttaa enintään 5-6 toimittajakandidaattia

Yksityisen sektorin esikarsinta voidaan toteuttaa kolmella tavalla: sisäisenä kandidaattien arviointina, toimituskykyä kartoittavalla kyselyllä tai ulkopuolisen konsultin avulla. Sisäisessä arvioinnissa tietolähteenä ovat useimmiten yrityksen aikaisemmat projektit sekä yhteistyökumppaneiden kokemukset. Mikäli kartoitus suoritetaan kyselyllä, lomakkeen sisältönä voisi toimia esimerkiksi

seuraavat otsikot: yrityksen taustatiedot, yrityksen tarjoamat palvelut eriteltyinä, yrityksen osaaminen halutulta erityisalueelta, alihankkijoiden käyttö sekä projektinhallinta. Hankittaessa uudentyyppistä tietojärjestelmää konsulttien käyttö on perusteltua, mikäli omasta yrityksestä ei löydy tarvittavaa osaamista. [Tietotekniikan liitto, 2005]

Esikarsinnan jälkeen jäljellä tulisi olla 3-6 toimittajakandidaattia, joilta pyydetään tarjouspyyntö hankkeesta [Kettunen, 2002].

5.3. Tarjouspyynnöt vaatimusmäärittelyyn pohjautuen

Tarjouspyynnön tarkoitus on saada toimittajalta tarjouksen muodossa kirjallista tietoa ja sitoumus, joiden pohjalta voidaan valita toimittaja. Tarjouspyyntö sisältää tilaajan oman näkemyksen siitä, millainen järjestelmän tulisi olla ja miten se toteutetaan. Hyvä tarjouspyyntö on tiivis ja informatiivinen. Tarkemmat kuvaukset tulisi sijoittaa liitteisiin. Useammille toimittajille lähetettyjen tarjouspyyntöjen tulee olla informaatioltaan samankaltaisia, jotta tarjousten vertailu on tasapuolista. [Tietotekniikan liitto, 2005]

Tarjousten pyytäminen toimittajilta kannattaa tehdä huolellisesti. Hutiloiden läpiviety tarjousprosessi laskee tarjousten laatua ja vaikeuttaa niiden keskinäistä arviointia. Tarjouspyyntöjen kirjoittamiseen tulee varata tarpeeksi aikaa ja tarvittaessa tulee käyttää ulkopuolisia konsultteja sen tekemiseen tai kommentoimiseen. [Kettunen, 2002]

Tarjouspyyntö toimitetaan kirjallisena niille toimittajille, joilta tarjous halutaan. Tarjouspyyntö voidaan lähettää myös sähköisenä käsittelyn nopeuttamiseksi ja helpottamaan tarjouspyynnön sisäistä levittämistä toimittajan organisaatiossa. [Kettunen, 2002] Tarjouspyyntö voidaan julkaista myös julkisesti esimerkiksi lehdessä, mikäli hankinta halutaan suorittaa avoimena tarjouskilpailuna [Tietotekniikan liitto, 2005].

Tarjouspyynnön laadinta aloitetaan yrityksen esittelystä ja yleiskuvauksesta. Tässä kohtaa voi lyhyesti esitellä yrityksen toimialaa, henkilöstöä, liikevaihtoa ja asiakaskuntaa. [Kettunen, 2002] Yleiskuvauksessa annetaan toimittajalle kuva siitä millaiseen tarpeeseen järjestelmää ollaan hankkimassa. Yleiskuvauksen perusteella toimittaja voi arvioida hankkeen kiinnostavuutta ja vaativuutta omien intressien kannalta. [Tietotekniikan liitto, 2005]

Seuraavaksi tulisi kuvata järjestelmän tekniset vaatimukset. Tässä kuvataan tietotekninen ympäristö, jossa järjestelmän tulisi toimia, rajapinnat toisiin järjestelmiin ja järjestelmät jotka tulisi integroida rakennettavaan järjestelmään. [Kettunen, 2002]

Myös tarjouksen haluttu muoto voidaan kertoa tarjouspyynnössä. Toimittajalle on hyvä kertoa, mitä tietoja tarjouksesta on ehdottomasti löydyttävä. Myös hinnoittelun voidaan pyytää tietyssä muodossa. [Kettunen, 2002]

Tarjouspyyntö voi sisältää myös tiedon toimittajan valintaperusteista. Miten toimittaja valitaan ja mitkä ovat valintaperusteiden painotukset? Lisäksi projektin aikataulu tulee kuvata valintaprosessin osalta sekä projektin tavoitellun luovutusajankohdan osalta. [Kettunen, 2002] Toimittajalle on syytä antaa myös mahdollisuus esittää oma näkemyksensä aikataulusta ja sen realistisuudesta [Tietotekniikan liitto, 2005].

Toimitusta koskeviin vaatimuksiin kirjataan vaiheistuksen ja kokonaisaikataulun lisäksi päätöksentekopisteet, muut vaadittavat palvelut kuten tukipalvelut, toimittajan projektihenkilöstö lähinnä nimeämistasolla, työvälineet sekä projektointivaatimukset [Tietotekniikan liitto, 2005].

Tarjouspyynnössä voidaan ilmaista asiakkaan vaatimuksia sopimuskäytäntöihin. Asiakkaan sopimusvaatimuksia voivat sisältää esimerkiksi maksuihin, maksuehtoihin, takuuseen ja ylläpitoon liittyviä seikkoja. Tässä kohdassa mainitaan lisäksi käytettävät sopimusmallit, maksuehdot ja – aikataulu, ylläpidosta sopiminen, omistus- ja tekijänoikeuskysymykset sekä hinnoittelu. [Tietotekniikan liitto, 2005]

Tarjouspyynnössä voidaan eritellä myös toivomukset alihankkijoiden käyttöön liittyviin rajoituksiin. Myös tarjouksen voimassaolo ja mahdollisia lisätietoja antavat henkilöt tulee mainita. [Tietotekniikan liitto, 2005]

5.4. Tarjousten arviointi ja pisteytys

Laajan tarjouksen kirjoittaminen voi viedä viikkoja, joten tilaajan tulee olla kärsivällinen tarjouksen suhteen [Kettunen, 2002]. Tarjouksen tekemisen kuluvaan aikaan vaikuttaa toteutettavan järjestelmän koko, hankintasuunnitelman taso sekä hankinnan kokonaisvaiheistus [Tietotekniikan liitto, 2005].

Tarjoukset otetaan vastaan ja avataan samanaikaisesti jokaista tarjoajaa tasapuolisesti kohtelevalla tavalla. Tarjousten avaamisesta on suositeltavaa tehdä pöytäkirja. Tämä on suositeltava tapa erityisesti julkisella sektorilla. [Tietotekniikan liitto, 2005]

Tarjousten vertailun tarkoituksena on saattaa toimittajat ja heidän ratkaisunsa paremmuusjärjestykseen vastaanotettujen tarjousten perusteella. Vertailu tapahtuu etukäteen päätettyjen kriteerien avulla [Tietotekniikan liitto, 2005]. Arviointikriteerejä voivat olla muun muassa [Kettunen, 2002]:

- osaaminen vaaditulta alueelta
- referenssit ja näiden lausunnot

- toimituskyky ja aikataulujen hallinta
- projektin läpiviennin menetelmäosaaminen
- projektijohtamisen malli
- laatujärjestelmä
- tarjouksen yksityiskohtaisuus ja laatu
- tarjouspyynnön ja asiakkaan ongelman ymmärtäminen
- toimittajan toimialan tuntemus
- henkilökemiat toimittajan ja asiakkaan välillä.

Tarjoukset käydään ensin läpi vertaamalla niitä arviointikriteereihin. Tällöin karsitaan pois tarjoukset, jotka eivät vastaa tarjouspyyntöön riittävällä tarkkuudella [Kettunen, 2002]. Karsinnan jälkeen tarjoukset pisteytetään vertailukriteereittäin ja järjestetään yhteispistemäärien mukaan paremmuusjärjestykseen. Parhaita tarjouksia esitetään jatkoon ja näistä valitaan tarkemmalla vertailulla tarjouskilpailun voittaja. [Tietotekniikan liitto, 2005]. Tarjouskilpailusta pudonneille toimittajille tulee ilmoittaa putoamisesta. Lisäksi on suotavaa kertoa, miksi näin tapahtui. [Kettunen, 2002]

Tarjousvertailun suorittaa etukäteen valittu valintaryhmä. Valintaryhmän johtaja pitää huolen, että vertailussa pitäydytään ennalta päätetyissä valintakriteereissä. Ryhmän ohjeistaminen tapahtuu hankintasuunnitelman ja tarjouspyynnön pohjalta. Valintaryhmässä tulee olla riittävästi asiantuntemusta. Mikäli tarvittavaa asiantuntemusta ei ole, voidaan vielä tarjouksia vertailtaessa hankkia ulkopuolista asiantuntija-apua. Ongelmia saattaa tuottaa erityisesti hankinnan hyötyjen hahmottaminen, tekniset ratkaisut, systeemityön vaiheet sekä henkilöresurssit. Valintaryhmän on myös omattava osaamista ostamisesta kuten neuvottelutaitoa, sopimusosaamista, ohjelmistoliiketoiminnan periaatteiden tuntemista ja oman organisaation hankintaperiaatteiden tuntemusta. [Tietotekniikan liitto, 2005]

Tarjousten arviointi pisteyttämällä voi olla hyvinkin laaja ja aikaa vievä. Arviointi voi sisältää jopa kymmeniä arvioitavia kohtia. Pisteytyksessä on hyvä käyttää arviointitaulukkoa, joka laaditaan projektiin osallistuvien toimesta. Arviointitaulukkoon rakennetaan runko ja tämän jälkeen asetetaan kullekin kriteerille painoarvo. Painoarvojen tulee vastata tarjouspyynnön valintakriteerejä. Painokertoimiin ja annettuihin pisteisiin vaikuttavat osallistujien taustat ja preferenssit. Tästä johtuen pisteytys onkin vain yksi tuki kokonaispäätöksenteossa. [Kettunen, 2002]

Pisteytysten perusteella jatkoneuvotteluihin valitaan korkeintaan neljä toimittajaa. Parhaista tarjouksista tulee tehdä yhteenveto, johon kirjataan tarjous-

ten vahvuudet ja heikkoudet. Tätä yhteenvetoa käytetään pohjana tulevissa jatkoneuvotteluissa. [Kettunen, 2002]

5.5. Jatkoneuvottelut ja tuotteeseen tutustuminen

Jatkossa oleviin toimittajiin tutustutaan vielä henkilökohtaisten tapaamisten muodossa. Tapaamisella pystytään arvioimaan toimittajan ammattitaitoa ja kokemusta suhteessa toisiin toimittajiin. [Tietotekniikan liitto, 2005] Tapaamiseen on hyvä saada paikalle myös projektille suunniteltu projektipäällikkö. Tapaamisessa on syytä kartoittaa seuraavat asiat: [Kettunen, 2002]

- toimittajan nykytilanne ja tulevaisuuden kehitys
- projektiryhmä
- sopimukset
- projektin johtaminen
- muutosten ja riskien hallinta
- hinnoitteluperusteet ja työmääräarviot
- tarjouksen tarkennukset.

Tuotteen koekäyttö eli pilotointi on perusteellisin tapa tutustua toimittajaan ja ohjelmistoratkaisuun [Tietotekniikan liitto, 2005]. Pilotointi on erityisen hyödyllinen, jos ollaan ostamassa valmista tuotetta [Kettunen, 2005]. Pilotointi voidaan nähdä strategisen suunnittelun työvälineenä. Tällöin asiakas saa olennaista tietoa järjestelmästä ja näkee konkreettisesti, mitä on ostamassa. Toisaalta toimittajan näkökulmasta ymmärretään paremmin, mitä asiakas todella haluaa, ja voidaan nopeammin tarjota oikeaa tuotetta ja palvelua. Toimittajasta ja tuotteesta riippuen pilotointi voi olla pelkästään konsultointia tai tuotteen käyttämistä toimittajan demo-ympäristössä. [Mäkipää, 2006]

5.6. Toimittajan valinta ja sopimusneuvottelut

Lopullisen toimittajan valinta tehdään ensin valintatyöryhmässä. Työryhmä esittelee tekemänsä valinnan yrityksen johdolle, joka tekee lopullisen valinnan. Valintaa tehdessä tulee käyttää seuraavia päätöksenteon perusteita: tarjouksista tehty pisteytys, toimittajien vahvuudet ja heikkoudet, toimittajien tarjoukset ja valintaryhmän mielikuvat toimittajista. Näiden perusteella tehdään lopullinen valinta, yleensä kahden toimittajan väliltä. [Kettunen, 2002]

Suomessa on käytössä sopimusvapauden periaate. Tämä tarkoittaa sitä, että jokaisella on oikeus päättää, kenen kanssa, millaisin ehdoin, missä muodossa ja millaisia sopimuksia tekee. [Kettunen, 2002]

Perinteisesti sopimukset syntyvät toisen osapuolen antamalla tarjouksella, jonka toinen osapuoli hyväksyy. Osaan sopimuksista tarvitaan kuitenkin sopimusneuvottelut, jonka lopputuloksena allekirjoitetaan varsinainen sopimus. [Tietotekniikan liitto, 2005] Sopimuksen laatiminen on usein pitkä prosessi. Sopimuksen laadintaan ja neuvotteluun saattaa kulua projektista riippuen kuukaudesta puoleen vuoteen. Kettunen [2002] suosittelee käytettäväksi lakimiehiä projektisopimusta tehtäessä. Näin voidaan varmistaa, että sopimus on molempia osapuolia turvaava. Sopimus laaditaan yleensä toimittajan sopimusmallien pohjalta. Tällöin tulee olla tarkkana, että sopimus ei suojaa toimittajaa liikaa.

6. Yhteenveto

Toiminnanohjausjärjestelmän hankinta on aikaa vievä ja monimutkainen prosessi. Hankinta tulee suunnitella kunnolla, jotta edellytykset hyvään lopputulokseen ovat olemassa.

ERP-järjestelmä on useimmiten hyvin suuri investointi, jolloin hankinnasta on saatava yritykselle tarpeeksi lisäarvoa. Tällaista lisäarvoa ovat muun muassa asiakaspalvelun parantuminen, laadun parantuminen, liiketoimintaprosessien tehostuminen ja mahdollinen liikevaihdon kasvu. Tulee kuitenkin huomioida, että muutokset eivät tapahdu hetkessä. Hankinnasta aletaan saada konkreettisia hyötyjä vasta vuosien jälkeen käyttöönotosta.

Ohjelmistotoimittajan valinta on monivaiheinen prosessi. Prosessi aloitetaan valitsemalla useita toimittajaehdokkaita. Vaiheiden edetessä ehdokkaita karsitaan kunnes jäljellä on lopullinen toimittaja, jonka kanssa edetään sopimusneuvotteluihin.

Onnistuneen hankinnan lopputuloksena yritys saa käyttöönsä järjestelmän, joka mahdollistaa kilpailukyvyn säilyttämisen nykyaikaisilla verkostoituneilla markkinoilla. Organisaation sisäinen tiedonkulku nopeutuu ja reaaliaikainen reagointi muuttuviin tilanteisiin mahdollistuu.

Viiteluettelo

[Davenport, 2000] Thomas H. Davenport, *Mission Critical: Realizing the Promise of Enterprise Systems.* Harvard Business School Press, Boston, 2000.

[Kettunen, 2002] Sami Kettunen, *Tietojärjestelmän ostaminen – käytännön opas yrityksille*. WS Bookwell Oy, Porvoo, 2002.

[Mäkipää, 2006] Marko Mäkipää, *Tietostrategiat ja niiden muodostus*, Tampereen yliopisto, Tietojenkäsittelytieteiden laitoksen kurssi ja luentomateriaali (12. 2006).

[Nah, 2002] Fiona Fui-Hoon Nah, *Enterprise Resource Planning Solutions and management*. Idea Group Publishing, 2002.

[Ruohonen ja Salmela, 2003] Mikko J Ruohonen, Hannu Salmela, *Yrityksen tietohallinto*. Edita Publishing Oy, Helsinki, 2003.

[Scheer and Habermann] August-Wilhelm Scheer and Frank Habermann, Making ERP a success, *Communications of the ACM* **43**, 4 (2000), 57-61.

[Tietotekniikan liitto, 2005] *Tietojärjestelmän hankinta – ohjelmistotoimittajan ja –ratkaisun valinta.* Talentum Media Oy, Helsinki, 2005.

[Wieder et al., 2006] Bernhard Wieder, Peter Booth, Zoltan P. Matolcsy and Maria-Luise Ossimitz, The impact of ERP systems on firm and business process performance. *Journal of Enterprise Information Management* **19**, 1 (2006), 13-29.

Elektronisen kaunokirjallisuuden verkkojakelu

Oskari Salonen

Tiivistelmä.
Tässä tutkielmassa tarkastellaan lyhyesti elektronisen kaunokirjallisuuden lähihistoriaa, sekä tarkemmin sen verkkojakelua tänä päivänä. Tämän lisäksi selvitetään keskeisimpiä ilmiöön liittyviä haasteita sekä hyötykohtia.

Avainsanat ja -sanonnat: Elektroninen kaunokirjallisuus, digitaalinen kirjallisuus, verkkojakelu, WWW
CR-luokat: K.4, I.7.4

1. Johdanto

Elektroninen kirjallisuus on viime vuosien aikana nostanut profiiliaan. Markkinoille on ilmestynyt uudenlaisia laitteita ja palveluita, jotka ovat tuoneet elektronisen kirjallisuuden yhä suuremman yleisön tietoisuuteen. Kyseessä ei kuitenkaan ole, monista uusista teknologioista huolimatta, mikään täysin uusi ilmiö, sillä e-kirjallisuuden perusta niin formaattien, jakelukanavien kuin myös käytettävien oheislaitteiden suhteen on ollut olemassa jo hyvän aikaa.

Kirjallisuus on viimeistään 2000-luvulla kokenut samansuuntaisia muutoksia kuin muut mediat. Markkinointi ja jakelu tapahtuvat yhä useammin verkon välityksellä. Siitä huolimatta, että tällainen digitalisoituminen sekä verkottuminen voidaan nähdä luonnollisena osana tekstin evolutiivista prosessia (Hillesund, 2005), ei se kirjallisuuden kohdalla ole ollut mitenkään ongelmaton tai helppo muutos. Elektroninen kirjallisuus ja sen verkkojakelu mahdollistavat kuitenkin täysin uudentyylisiä toteutustapoja sekä toimintamalleja, jolloin olisi tärkeää tuntea ne syyt, jotka ovat verkkojakelun kohdalla aiheuttaneet toistaiseksi melko katkonaisia sekä lyhytkestoisia palveluita.

Vuosituhannen vaihteen tienoilla, kovimman it-buumin aikaan, elektronisen kirjallisuuden ennustettiin lyövän itsensä läpi tulevina vuosina. Kymmenen vuotta myöhemmin tilanne on kuitenkin pohjimmiltaan sama. Vaikka kehitystä on tapahtunut monellakin eri rintamalla, antaa lopullinen läpimurto vielä odottaa itseään. Revoluution sijaan elektronisen kirjallisuuden lähihistoriaa sekä nykytilaa kuvaakin paremmin termi evoluutio (Hillesund, 2005).

Tämän tutkielman puitteissa on tarkoitus selvittää elektronisen kaunokirjallisuuden www-jakelun nykytilaa, sekä siihen liittyviä haasteita ja mahdollisuuksia. Verkosta löytyy valtavan paljon erilaisia, laadultaan sekä volyymiltaan vaihtelevia sivustoja ja palveluita, eikä niiden kaikkien käsitteleminen ole mahdollista eikä mielekästä. Pyrkimyksenä onkin kartoittaa tärkeimmät sellaisista

kaunokirjallisuutta tarjoavista palveluista, jotka kuvaavat mahdollisimman kattavasti olennaisimpia jakelukanavia.

Tämän tutkielman kohdalla tärkeimpänä rajaavana tekijänä ovat kirjallisuutta verkossa tarjoavat palvelut, jotka määrittävät sisältönsä puolesta sen, kuuluvatko ne kaunokirjallisuuden piiriin. Fiktiivisen kirjallisuuden lisäksi aiheen lähestymistavaksi on valittu tietyllä tapaa tavallisen e-kirjallisuutta kuluttavan henkilön näkökulma. Tämä ilmenee lähinnä siten, että rajatulle käyttäjäkunnalle suunnatut palvelut, kuten erilaiset oppilaitosten verkkokirjastot, ovat pääosin jätetty käsittelyn ulkopuolelle. Tähän on kuitenkin joitain poikkeuksia, niiltä osin kuin tällaiset palvelut ovat kaikille avoimia.

Verkkopalveluiden lisäksi olen rajannut aiheen käsittelyä markkinoiden perusteella. Tässä tutkielmassa käsitellään ensisijaisesti englanninkielisen kaunokirjallisuuden sekä markkina-alueen verkkojakelua. Tämä johtuu aivan puhtaasti käsiteltävän materiaalin rajaamisesta, mutta myös osittain siitä, että monet alan innovaatiot ja uudet palvelut ovat vielä toistaiseksi melko Amerikkakeskeisiä. E-kirjallisuus on voimissaan myös Aasiassa, mutta tämän tutkielman puitteissa sitä ei käsitellä.

Elektronisen kirjallisuuden määritelmä on yhä edelleen varsin monitahoinen, ja siksi tutkielman alkuun on tarpeen täsmentää, mitä sillä tässä yhteydessä tarkoitetaan. Tätä seuraa tiivistetty katsaus kaunokirjallisuuden verkkojakelun historiaan sekä eri vaiheisiin. Seuraavaksi käsitellään ilmiön nykytilaa erityyppisten jakelukanavien kautta, jota seuraa katsaus käytössä oleviin formaatteihin, tekniikoihin sekä oheislaitteisiin. Tämän jälkeen aiheen käsittely jatkuu kaunokirjallisuuden verkkojakeluun liittyvillä ongelmilla sekä mahdollisuuksilla, jotka ovat monilta osin yhteneväisiä yleisen elektronisen kirjallisuuden kanssa.

2. E-kirjallisuuden määritelmä

Elektronisen kirjallisuuden monimuotoisuus on tehnyt e-kirjan tarkasta määrittelemisestä vaikeaa. E-kirjan eri ominaisuuksiin, erilaisin painotuksin pohjautuvia määritelmiä on lukuisia. Toimivan määritelmän luominen on haastavaa jo pelkästään siksi, että e-kirjalla voidaan viitata joko sisältöön, tiedostoformaatteihin, lukuohjelmiin tai lukulaitteisiin (Rao, 2004). Nopeasti kehittyvän sekä vaihtuvan teknologian keskellä, laajempaa hyväksyntää nauttivalta määritelmältä, joka lisäksi kestäisi aikaa, vaaditaan joka tapauksessa tiettyä riippumattomuutta eri laitteista sekä formaateista. Lopullista yksimielisyyttä siitä, mikä määrittelee e-kirjan, ei ole kuitenkaan vielä saavutettu, vaikka joitain varsin yleisesti hyväksyttyjä esityksiä aiheesta on tehty (Vassiliou & Rowley, 2008).

Tavallista painettua kirjaa jäljittelevä olemus on yksi varsin yleisesti e-kirjan ominaisuuksiin laskettu piirre (Armstrong, 2008; Vassiliou & Rowley, 2008). Armstrongin (2008) mukaan kirjan laajuudella tai kirjoitusmenetelmällä ei ole merkitystä, kunhan kirjaa luetaan millä tahansa lukulaitteella, jossa on näyttö.

Merkittävä osa nykypäivän elektronisesta kaunokirjallisuudesta on kuitenkin tavallisen romaanin pituisia sekä perinteisin menetelmin kirjoitettuja johtuen siitä, että ne ovat digitaalisia painoksia normaaleista printtimedian teoksista.

Kirjan pituuden kohdalla Armstrong (2008) tekee kuitenkin selkeän eron sen suhteen, että erilaiset lyhyet tekstijulkaisut kuten pamfletit eivät kuulu e-kirjojen piiriin, kuten eivät myöskään erilaiset elektroniset lehtijulkaisut.

Osa e-kirjan määrittelyistä pitää myös dynaamisia toiminnallisuuksia merkitsevinä tekijöinä. Varsin yleinen tämänkaltainen toiminto on hyperlinkitys (Vassiliou & Rowley, 2008). Tässä tutkielmassa tämä ominaisuus ei kuitenkaan ole ehdottoman olennainen tekijä, sillä osa käsiteltävistä e-kirjoista on tuotettu erilaisin valokuvausmenetelmin, eivätkä siksi sisällä mainitun kaltaisia ominaisuuksia.

Elektronisen kirjallisuuden lisäksi on tarpeen täsmentää kaunokirjallisuuden olemusta tämän tutkielman yhteydessä. Tarkka määritelmä siitä, mikä lasketaan kaunokirjallisuuden piiriin, on e-kirjan tapaan, ainakin paikoitellen, melko tulkinnanvaraista. Mihinkään kovin tarkkaan semanttiseen määrittelyyn ei ole tarvetta, joten todettakoon, että kaunokirjallisuus kattaa tällä kertaa siihen varsin yleisesti lasketut kirjallisuuden muodot. Näitä ovat kertomakirjallisuus, kuten romaanit sekä novellit, runous ja draama.

Elektronisen kaunokirjallisuuden piiriin voidaan laskea myös uudentyyliset kirjallisuuden alalajit, kuten hyperfiktio tai wikikirjallisuus. Näille tyypillisiä piirteitä ovat verkko ensisijaisena luomisympäristönä, tarinoiden dynaamisesti muuttuvat rakenteet, sekä useat eri kirjoittajat, jotka luovat teoksen yhteistyönä. Tämänkaltaisten teosten verkkojakelu jää kuitenkin tällä erää käsittelyn ulkopuolelle painopisteen ollessa perinteisemmissä kirjallisissa tuotoksissa. Myöskään CD- tai DVD-levyille talletetut e-kirjat eivät kuulu käsiteltäviin aiheisiin.

Huomionarvoista on myös se, että tähän tutkielmaan valikoidut verkkojakelukanavat sekä verkkopalvelut määräytyvät sen mukaan, mistä voidaan löytää edellä määritellyn kaltaista kaunokirjallisuutta. Toisin sanoen, käsiteltävät verkkojakelupalvelut tarjoavat varsin todennäköisesti myös muunlaista kirjallisuutta.

3. Verkkojakelun historiaa

Kaunokirjallisuuden verkkojakelu alkoi pienimuotoisesti melko pian WWW:n yleistymisen seurauksena. Lähestyttäessä 1990-luvun puoliväliä ensimmäiset kaunokirjallisuutta tarjoavat WWW-sivut alkoivat ilmestyä. Tällaisia olivat esimerkiksi Project Gutenbergin ensimmäiset verkkosivut vuonna 1994, sekä hieman myöhemmin, vuonna 1996, perustettu Internet Archive -sivusto. Verkkosivuja ja kirjallisuutta arkistoihin tallettava Internet Archive -projekti toimii yhteistyössä muiden vastaavien hankkeiden kanssa, kuten Project Gutenbergin sekö Million Books Projectin, ja onkin tavallaan kokoelma kirjakokoelmia (Connaway & Wicht, 2007).

Elektronisen kaunokirjallisuuden juuret juontavat kuitenkin paljon kauemmaksi historiassa, aikaan ennen WWW:tä. Vuonna 1971 Michael S. Hartin alulle panema Project Gutenberg, jonka ideana on digitoida, arkistoida sekä levittää mahdollisimman vapaasti yleisessä käytössä olevia teoksia, mainitaan usein ensimmäisenä varsinaisena e-kirjakokoelmana (Vassiliou & Rowley, 2008; Armstrong, 2008). Hieman myöhemmin, vuonna 1976, sai alkunsa myös toinen vastaavanlainen kokoelma, Oxford Text Archive. Oxfordin kokoelman on kuitenkin alun perin ollut tarkoituksena olla ensisijaisesti akateeminen resurssi (Vassiliou & Rowley, 2008), jolloin se on tarjonnut paljon muutakin kuin kaunokirjallisia tekstejä. Myöhemmin vastaavanlaisia, akateemisten laitosten yhteyteen kehitettyjä ilmaisia kokoelmia, on perustettu useita, kuten Virginian yliopiston kokoelma vuonna 1992 (Armstrong, 2008).

Vuosituhannen loppupuolella kiinnostus WWW:n kautta jaettaviin e-kirjoihin heräsi myös kaupallisessa mielessä. Markkinoille ilmestyi e-kirjojen lukemiseen kehitettyjä laitteita, kuten SoftBook sekä Rocket Ebook (Young, 2008), joiden ensimmäiset versiot ilmestyivät vuonna 1998. Laitteesta riippuen tyypillinen käyttötapa oli joko ladata e-kirja ensin tietokoneelle, josta se siirrettiin lukulaitteeseen, tai SoftBookin tapaan hyödyntämällä laitteen sisäistä modeemia, jolloin lataus tapahtui suoraan verkosta (Rao, 2001). Kiinnostusta e-kirjamarkkinoihin löytyi myös jälleenmyyjien sekä ohjelmistoyritysten parista. Barnes & Noble oli yksi varhaisista e-kirjoja myyvistä yrityksistä (Herther, 2005), Microsoftin sekä Adoben kehittäessä ohjelmistoja e-kirjojen lukemiseen.

Elektronisen kirjallisuuden alkutaivalta vaivasi markkinoiden pienuuden lisäksi standardien puute. Kirjoja oli tarjolla lukuisissa eri tiedostomuodoissa, jotka toimivat pahimmillaan ainoastaan yhdessä lukulaitteessa. Towle ja muut (2007) toteavatkin, että tämänkaltainen ”formaattisota” oli yksi syistä, miksi lukulaitteet eivät menestyneet. Myös monet kustantajat sekä jälleenmyyjät joutuivat 2000-luvun alussa toteamaan, että e-kirjojen ollessa kyseessä kuluttaja ja myyjä eivät kohdanneet riittävän tehokkaalla tavalla. Barnes & Noblesin pää-

omistuksessa ollut Mightywords.com e-kirjapalvelu lakkautettiin vuonna 2001 (Nawotka, 2001), kuten myös emoyhtiön oma e-kirjatuotanto kaksi vuotta myöhemmin (Herther, 2005). Aiemmin, vuoden 2001 aikana, suurista kustantajista Random House sekä Time Warner molemmat sulkivat erilliset elektronisia kirjoja kustantavat linjansa (Rose, 2001).

Siitä huolimatta, että monet suuret kirjakustantajat ja jälleenmyyjät eivät onnistuneet 2000-luvun alussa laukaisemaan menestyviä e-kirjapalveluita, markkinat olivat pienimuotoisuudestaan huolimatta olemassa. Just (2006) arvioi, että Yhdysvalloissa kaupallisesti tarjolla olleiden e-kirjojen määrä on kasvanut tasaista 20 prosentin vuosivauhtia, korkeimpien kasvupiikkien vuonna 1999 yltäessä 60 prosenttiin. IDPF-organisaation (International Digital Publishing Forum) keräämä data vuodelta 2003, e-kirjojen ollessa jo laskusuhdanteessa, kertoo 12–15 suurimman kustantajan tukkumyynnin olleen noin 21 prosentin vuosikasvun tasolla, rahallisen arvon ollessa noin 7,3 miljoonaa Yhdysvaltain dollaria (IDPF, 2008). Epäonnistuneissa e-kirjojen lanseerausyrityksissä onkin ollut useimmiten kyse liian suurista odotuksista sekä tavoitteista, koska markkinoiden todellista tilaa sekä kehitystä ei ole tunnettu.

Mukaan on mahtunut myös onnistumisia. Yksi tämän päivän suosituimmista elektronista kaunokirjallisuutta myyvistä palveluista on Fictionwise.com, joka on toiminut menestyksekkäästi jo vuodesta 2000. Kestävää kehitystä tapahtui myös e-kirjojen myynnin ulkopuolella. Vuonna 1999 julkaistiin ensimmäinen versio sittemmin yhä suositummaksi nousseesta OEBPS-määritelmästä (Open eBook Publication Structure Specification) (IDPF, 1999), joka on pyrkinyt luomaan laitteista sekä ohjelmistoista riippumattoman, standardinomaisen e-kirjaformaatin.

Mielenkiintoisen esimerkin siitä, millaisin menetelmin ja toimintamallein elektronista kaunokirjallisuutta on pyritty myymään, saa, kun tarkastelee yhden erittäin suositun kirjailijan, Stephen Kingin, kokeiluja e-kirjojen parissa. Vuonna 2000 King julkaisi teoksen *Riding the Bullet* kotisivullaan 1,99 dollarin hintaan, mikä osoittautui suureksi menestykseksi kirjan myydessä muutaman päivän aikana puoli miljoonaa kappaletta (Nawotka, 2008). Samana vuonna luku kerrallaan julkaistu, ”ilmainen” e-kirja *The Plant* osoittautui puolestaan vähemmän menestyksekkääksi. Kirjan jatkumisen ehtona oli se, että jokaista lukua kohti vähintään 75 prosenttia lukijoista lahjoittaisi vapaaehtoisesti yhden dollarin maksuna. Näin ei kuitenkaan pidemmän päälle käynyt, jolloin tarina jäi kesken (Towle, Dearnley & McKnight, 2007).

4. Ilmaiset e-kirjat

4.1 Kokoelmasivut

Kaunokirjallisuutta ilmaiseksi tarjoavia verkkosivuja on paljon, ja niiden laadullinen vaihtelu on melko suuri. Ilmaisia e-kirjoja tuotetaan verkkoon hyvin erilaisista lähtökohdista, yksittäisten harrastelijoiden toimesta aina suuriin akateemisiin digitoimishankkeisiin asti, jolloin on ymmärrettävää, että vaihtelua löytyy.

Ilmaisia elektronisen kirjallisuuden resursseja ehkä eniten määrittelevä tekijä on se, että niiden teosten tekijänoikeudet ovat umpeutuneet, jolloin valtaosa e-kirjoista on vanhempaa kirjallisuutta. Tämän lisäksi ilmainen e-kirja voidaan määrittää sen mukaan, missä määrin sitä voidaan hyödyntää. Berglund ja muut (2004) toteavat ilmaisuuden määritelmäksi sen, että e-kirjan hankkimiseen, lukemiseen, kopiointiin tai käyttöön ei liity mitään maksuja.

Seuraavassa esitellään, Google Books pois lukien, tällaisia palveluita, joista jokainen tarjoaa kaunokirjallisuutta hieman eri menetelmin sekä eri tarkoituksiin.

4.1.1 Project Gutenberg

Project Gutenberg (gutenberg.org) on vanhin ja tänä päivänä yksi suosituimmista kaunokirjallisuutta verkossa jakavista sivustoista. Joulukuussa 2008 se tarjoaa noin 27 400 (Project Gutenberg) ilmaista teosta kauno- sekä tietokirjallisuuden saralta, sisältäen pääosin e-kirjoja, mutta myös jonkin verran äänikirjoja.

Gutenberg–projektin ajatuksena on ollut tuottaa e-kirjat mahdollisimman laajalle käyttäjäkunnalle helposti saataville ja vielä siten, että tekstien käytettävyys säilyy kauas tulevaisuuteen. Tämä ilmenee käytännössä siten, että jaetut teokset ovat hyvin pelkistetyssä muodossa, ASCII-tekstin ollessa tärkein formaatti, jolloin kirjat ovat luettavissa lähes kaikilla tekstiä käsittelevillä ohjelmilla. Teoksia on saatavilla myös HTML-muodossa, sekä melko vähäinen määrä Adoben PDF-formaatissa.

Maailmanlaajuisen vapaaehtoistyön pohjalta toimivan projektin rinnalle on vuosien saatossa syntynyt vastaavia, maantieteellisesti hieman rajatumpia projekteja. Tällaisia ovat Project Gutenberg of Australia, Project Gutenberg DE saksalaiselle kirjallisuudelle, sekä Project Runeberg pohjoismaiselle kirjallisuudelle. Project Gutenbergin tuottamia e-kirjoja, hyödynnetään usein myös muiden ilmaisten e-kirjakokoelmien toimesta (Berglund et al., 2004). Tämä voidaan nähdä osoituksena projektin tietynlaisesta pioneerihengestä.

4.1.2 ManyBooks

Manybooks.net sivusto on yksi tällainen Project Gutenbergin työtä hyödyntävä palvelu. Joulukuussa 2008 sivusto mahdollisti noin 22 400 e-kirjan lataamisen, lukuisten tekstien ollessa peräisin Project Gutenbergin arkistoista.

ManyBooksin tärkein ominaisuus on kuitenkin se, miten palvelu kehittää Gutenbergin tekstejä. Lukuisten muunto-ohjelmien avulla sivustolle tuotetaan monipuolinen kokoelma e-kirjoja, joita voi ladata yleisimmissä tiedostoformaateissa. Palvelu tarjoaa e-kirjat suosituimpiin lukulaitteisiin kelpaavissa muodoissa, jopa Amazonin Kindle-laitteen yksinoikeudella käyttämässä AZW-formaatissa. Näiden lisäksi e-kirjoja tarjotaan muihin mobiililaitteisiin ja lukuohjelmiin soveltuvissa muodoissa, unohtamatta IDPF:n suosittelemaa epub-standardiformaattia.

4.1.3 MemoWare

Memoware.com sivuston ilmaisten e-kirjojen jakelu on keskittynyt hyvin voimakkaasti mobiililaitteiden ympärille. MemoWaren kohdalla kyseessä ovat vielä tarkemmin määriteltynä kämmentietokoneet (Personal Digital Assistant) sekä älypuhelimet (Smartphone), eikä palvelu tue suoranaisesti e-kirjojen lukulaitteita.

MemoWaren katalogista löytyy joulukuussa 2008 noin 6800 erilaista PDA–laitteisiin ladattavaa kaunokirjallista teosta. Noin 300 näistä teoksista on saatavilla Adoben PDF-formaatissa tai ASCII-tekstinä, mutta valtaosa teoksista on PDA-keskeisissä tiedostoformaateissa. Näistä yleisimmin palvelussa ovat edustettuna PalmReader, iSilo sekä TomeRaider formaatit.

4.1.3 Google Books

Google Books on yksi suurimmista kirjallisuuden digitointihankkeista, tuotettujen kirjojen lukumäärän yltäessä syyskuussa 2008 yli seitsemään miljoonaan (Leetaru, 2008). Kaikista kirjoista kuitenkin vain murto-osa on kokonaisuudessaan saatavilla. Kokonaisia kirjoja tuottava haku, jossa haetaan sekä otsikosta että kirjan koko sisällöstä osumia millä tahansa yleisimmistä englanninkielen artikkeleista, kuten *the* tai *a*, tuottaa noin 320 000 osumaa, joista vain osa on kaunokirjallisia teoksia. Google Booksin kokonaisten teosten määrän arviointi on vaikeaa, sillä palvelu ei tarjoa tähän mitään luotettavasti toimivaa menetelmää. Löydetyistä hakutuloksista vain muutama sata ensimmäistä on mahdollista selata läpi. Näistä kokonaisista teoksista vain pieni osa on mahdollista tallentaa PDF-muotoisena käyttäjän koneelle, minkä takia lukeminen tapahtuu selainikkunassa Googlen omalla sivulla.

Sellaisista kokonaisista teoksista, joiden tekijänoikeudet ovat umpeutuneet jo kauan sitten, Google tarjoaa varsin usein uusia, Google Books -palvelua varten tuotettuja versioita, jotka puolestaan on suojattu tekijänoikeuksin. Näiden teosten kohdalla ei voida puhua Berglundin ja muiden (2004) määrittelemästä ilmaisesta e-kirjasta.

Google Books eroaa monista muista e-kirjapalveluista siten, että se näyttää myös tekijänoikeuksien alaisia uusia kirjoja. Joissain tapauksissa näytetään koko teos, mutta useimmiten kyseessä on rajoitettu esikatselu kustantajan hyväksymänä (Leetaru, 2008). Käytännössä tämä ilmenee siten, että selainikkunassa luettavasta kirjasta on poistettu sivuja sieltä täältä, jolloin käyttäjä ei pääse lukemaan koko teosta. Rajoitetun esikatselun kirjoille on kuitenkin tyypillistä, että teoksista, joiden pituus on useita satoja sivuja, vain muutamia kymmeniä sivuja on poistettu.

Suuri osa Google Booksin löytämistä teoksista on kuitenkin pelkästään kansikuvan sekä tiivistetyn bibliografisen tiedon sisältäviä osumia, eivätkä lainkaan e-kirjoja. Leetaru (2008) näkeekin Google Booksin eräänlaisena mainosikkunana kustantajille, teossivujen sisältäessä linkkejä verkkokauppoihin, joista kirjan voi itselleen ostaa.

4.2 Kirjailijakohtaiset sivut

Oman pienen osansa ilmaisen kaunokirjallisuuden verkkojakelusta muodostavat kirjailijakohtaiset sivustot. Tällaiset ilmaiset verkkoresurssit, joihin on koottu tietyn kirjailijan tuotantoa, ovat muiden kokoelmasivustojen suhteen samanlaisia siten, että teokset ovat useimmiten vanhempaa kirjallisuutta.

Varsin tyypillisiä kirjailijakohtaisia sivustoja ovat esimerkiksi ambrosepierce.org, (Ambrose Pierce) sekä eldritchdark.com (Clark Ashton Smith) -sivustot, jotka kokoavat yhteen kirjailijoidensa, tässä tapauksessa 1800-luvun lopun ja 1900-luvun alun, kauhukirjallisuutta. Kirjailijakohtaisille sivustoille on tyypillistä se, että ne ovat pelkkien teosten kokoamisen sijaan enemmänkin kokonaisvaltaisia resursseja sisältäen biografioita, bibliografioita sekä muuta informaatioita kiinnostuksen kohteistaan. Edellä mainitut kaksi sivustoa kuvaavat tämänkaltaisia sivustoja hyvin myös siinä suhteessa, että e-kirjat ovat usein hyvin pelkistetyissä muodoissa, kuten HTML-tekstinä.

Uudempaa kaunokirjallisuutta on saatavilla myös jonkin verran ilmaiseksi. Verkosta löytyy kaunokirjallisuutta, joka ei ole löytänyt kustantajaa ja kirjailija on päättänyt julkaista teoksen ilmaiseksi, mutta myös e-kirjoja, joista on jo olemassa printtiversio. Tällaisesta on hyvä esimerkki yhdysvaltalaisen kirjailijan Nick Mamataksen Creative Commons -lisenssin alaisuudessa julkaisema teos

Move Underground, joka on ladattavissa esimerkiksi PDF-muodossa osoitteessa moveunderground.org.

5. Maksulliset e-kirjat

Uudempaa elektronista kaunokirjallisuutta lukeakseen joutuu pääsääntöisesti turvautumaan maksullisiin verkkopalveluihin. Saatavilla olevien kirjojen julkaisuajankohta onkin yksi selkeimmistä ilmaiset ja maksulliset palvelut toisistaan erottavista tekijöistä.

5.1 Fictionwise

Fictionwise.com on yksi suosituista elektronista kaunokirjallisuutta myyvistä palveluista. Oman palvelunsa lisäksi yhtiö hallinnoi sekä tuottaa sisältöä muihin omistamiinsa e-kirjapalveluihin, joita ovat ereader.com sekä ebookwise.com. Joulukuussa 2008 fictionwise.com sivustolla oli myynnissä noin 42 000 erilaista kaunokirjallista teosta.

Teosta ostettaessa valitaan, halutaanko kirja ostaa suojattuna, yhteen formaattiin sidottuna versiona vai suojaamattomana versiona, jonka formaatin voi valita myöhemmin. Osa kirjoista on myynnissä pelkästään suojattuina versioina, yleensä 3-4 eri tiedostoformaatissa, kun taas osa on saatavilla myös suojaamattomina versioina.

Ostamisen jälkeen kirja siirtyy käyttäjän palvelun sisäiseen kirjahyllyyn, josta se voidaan ladata tietokoneelle niin usein kun on tarve. Jos kirja ostettiin suojattuna versiona, voidaan se ladata ainoastaan ostamisen yhteydessä valitussa tiedostomuodossa. Jos kirja ostettiin suojaamattomana versiona, tarjoaa Fictionwise mahdollisuuden ladata se pöytäkoneita, lukulaitteita, kämmentietokoneita sekä älypuhelimia tukevissa tiedostomuodoissa, joita on yhteensä 12 erilaista.

Yksi Fictionwisen melko innovatiivinen ominaisuus on se, että palvelusta voidaan lähettää e-kirjoja suoraan kilpailevan yhtiön Amazonin Kindle-lukulaitteeseen. Säätämällä käyttäjän Amazon-tilin asetuksia voidaan Fictionwisen kirjahyllystä lähettää sähköpostiliitteenä e-kirja, yhteensopivassa Mobipocket-formaatissa, suoraan Kindle-laitteeseen. Siitä huolimatta, että e-kirjoja on saatavilla nykyisin yhä enemmän laiteriippumattomissa muodoissa, ei tällaista toiminnallisuutta, joka palvelee nimenomaan käyttäjän tarpeita, ilmene kovin paljoa kaupallisten palvelujen kohdalla

5.2 Amazon.com

Amazon.com-palvelun elektronista kaunokirjallisuutta koskeva osuus voidaan jakaa kahteen osioon. Toisen muodostaa Amazon Shorts -osio, joka myy lyhyttä kaunokirjallisuutta novellien muodossa. Toinen keskeinen osio on Kindle-kirjallisuuden palvelu, josta voi ostaa kirjoja Kindle-lukulaitteeseen.

Amazon Shorts -kokoelma tarjoaa mahdollisuuden ostaa novellimuotoista kirjallisuutta noin 2900 nimikettä kattavasta valikoimasta. Novelleja myyvä osio on varsin perinteinen siinä mielessä, että ostettuaan teoksen käyttäjä voi ladata sen tietokoneelleen joko Adoben PDF–formaatissa tai HTML-tiedostona. Tämän lisäksi ostetut kirjat talletetaan Amazonin omaan mediakirjastoon, josta niitä voi ladata myöhemmin.

Amazonin Kindle-kirjakauppa on puolestaan hyvin pitkälti Kindle–lukulaitteen ympärille rakennettu palvelu. Sivuston fiktio-osion valikoimissa on noin 88 000 teosta Kindle-painoksina, jotka ovat ladattavissa Amazonin yksinoikeudella käyttämässä AZW–formaatissa.

Kirjoihin on mahdollista tutustua ennen ostopäätöstä. Verkkokaupasta voi ladata lukulaitteeseen ilmaisen esikatselukappaleen, joka sisältää lyhyen tekstiosion teoksen alusta.

Kirjan ostamisen jälkeen se toimitetaan suoraan käyttäjän valitsemaan Kindle-lukulaitteeseen, mutta myös Amazonin mediakirjastoon myöhempää käyttöä varten. Kindle-kirjoja voi ladata mediakirjastosta myös käyttäjän koneelle, josta ne voidaan siirtää lukulaitteeseen.

6. Mobiililaitteet ja verkkojakelu

6.1 Sony Reader

Mobiililaitteiden kohdalla e-kirjallisuuden hankintatapoja on useita erilaisia. Lähes kaikki kämmentietokoneet, älypuhelimet kuin myös e-kirjojen lukulaitteet tukevat menetelmää, jossa kirja ladataan ensin tietokoneelle ja siirretään sen jälkeen joko langattoman yhteyden tai esimerkiksi USB-kaapelin avulla haluttuun laitteeseen. Sony Reader -lukulaitteen eri mallien kohdalla tällainen menetelmä on ainoa tapa siirtää teoksia (Gade, 2008). Käyttäjä voi kuitenkin valita, haluaako kirjansa hankkia verkkokaupoista, Sonyn oman e-kirjakaupan tarjotessa noin 35 000 fiktiivistä nimikettä, vai ilmaisista palveluista.

6.2 Amazon Kindle

Kindle-lukulaite tukee myös tietokoneen kautta tapahtuvaa kirjavälitystä. USB-yhteyden avulla laitteeseen on mahdollista siirtää e-kirjoja, joiden toimivuuden edistämiseksi Amazon on luonut sähköpostin avulla toimivan palvelun, joka

muuntaa tietyin rajoituksin lähetettäviä tiedostoja Kindlessä toimivaan AZW-formaattiin. Pientä maksua vastaan Amazon lähettää muunnetun tiedoston suoraan Kindleen, mutta tiedoston voi muuntaa AZW-muotoon myös ilmaiseksi, jolloin tiedosto palautetaan ja käyttäjä siirtää sen itse laitteeseen (Gade, 2008).

Näiden lisäksi Kindle mahdollistaa myös muita menetelmiä hankkia e-kirjoja verkosta. Laitteen avulla voidaan muodostaa langaton sekä maksuton Whispernet-yhteys Amazonin verkkokauppaan, josta e-kirjoja voidaan ostaa suoraan laitteeseen (Gingras et al., 2008).

Kindlessä on myös yksinkertainen WWW-selain, joka mahdollistaa e-kirjojen lataamisen verkkosivuilta suoraan laitteeseen. Toimiakseen kirjojen tulee olla joko AZW–formaatissa, suojaamattomina Mobipocket-tiedostoina tai pelkkänä tekstinä (Amazon, 2008).

6.3 PDA:t ja puhelimet

Kämmentietokoneiden sekä älypuhelimien kohdalla kaunokirjallisuuden verkkojakelu tapahtuu pääasiassa kahden eri kanavan kautta. Kuten lukuisat muut mobiililaitteet, myös kämmentietokoneet sekä puhelimet voidaan yhdistää useimmiten tietokoneisiin ja luodun yhteyden avulla pystytään siirtämään dataa laitteisiin.

Modernit PDA-laitteet sekä älypuhelimet sisältävät nykyisin kuitenkin lähes vakio-ominaisuutena Internet-yhteyden sekä WWW-selaimen. Tällä tavalla laitteesta voidaan suoraan selata erilaisia kaunokirjallisuutta jakavia verkkopalveluita, joista osa tarjoaa kirjallisuutta erityisesti kämmentietokoneisiin sekä puhelimiin muokatuissa formaateissa. WWW:n selaamisen yhteydessä voidaan lisäksi tehdä vielä erillinen jako selaimen kautta luettavaan kaunokirjallisuuteen sekä selaimen avulla laitteen muistiin ladattavaan e-kirjallisuuteen.

7. Formaatit ja suojaukset

7.1 DRM

Digital Rights Management (DRM) on yleistermi, jolla tarkoitetaan lukuisia erilaisia tekniikoita, joiden tarkoituksena on suojata digitaalista tietoa (Coyle, 2003). Elektronisen kirjallisuuden kohdalla DRM liittyy laite- ja ohjelmistovalmistajien sekä e-kirjoja tuottavien tahojen pyrkimyksiin estää teosten laitonta kopiointia sekä jakelua.

Erilaisia DRM-suojauksia on lukuisia. Suojausten tasosta riippuen niillä voidaan estää esimerkiksi e-kirjojen sisällön kopiointia, tulostamista ja muokkausta tai olennaisimpana, vaikuttaa e-kirjan toimivuuteen eri ohjelmistoissa ja

laitteissa. DRM-teknologiat eivät varsinaisesti estä tiedostojen kopiointia, vaan ne tekevät sen hyödyttömäksi (Coyle, 2003).

Menetelmiä DRM:n toteuttamiseen on useita. Amazon Kindlen kohdalla ostetut kirjat liitetään sarjanumeron avulla siihen Kindle-laitteeseen, joka on rekisteröity ostajan tiliin. Microsoft Readerin kohdalla ohjelman versio, yhdessä laitteiston tietojen kanssa, liitetään aktivoinnin avulla osaksi käyttäjän Microsoft Passport -tiliä, joka mahdollistaa suojattujen teosten lukemisen samanaikaisesti useassa eri Microsoft Reader -ohjelmassa. Näiden menetelmien lisäksi, DRM-suojattuja e-kirjoja on toteutettu esimerkiksi ostajan luottokortin numeroon pohjautuvina (Coyle, 2003).

7.2 Formaatit

Elektronista kirjallisuutta tarjotaan sekä kaupallisten että ilmaisten palvelujen kautta lukuisissa eri formaateissa. Eri formaattien aiheuttamaan yhteensopivuusongelmaan on kuitenkin kehitetty ratkaisuja: verkosta on mahdollista ladata lukuisia kääntöohjelmia, jotka muuntavat tiedostoja eri tiedostomuotoihin.

Vuoden 2007 syyskuusta lähtien IDPF-järjestön kehittämä epub-tiedostoformaatti hyväksyttiin virallisesti alan standardiksi (Nawotka, 2008). Kaikki e-kirjoja julkaisevat sekä tuottavat tahot eivät kuitenkaan vielä tue tätä XML-pohjaista formaattia. Suosituista valmistajista Sony tunnistaa epub-formaatin uusissa lukulaitteissaan sekä niiden ohjelmistoissa (Sony, 2008). Mobipocket Reader Desktop -ohjelmisto ei tue epub-tiedostoja suoraan, mutta pystyy uusimman version kohdalla muuntamaan tiedoston yhteensopivaan muotoon.

Adoben PDF (Portable Document Format) on varsin yleinen e-kirjaformaatti. Ohjelmistona PDF-tiedostojen lukemiseen toimii Adoben oma Adobe Reader, jota on saatavilla monipuolisesti niin pöytäkoneisiin, puhelimiin kuin kämmentietokoneisiin. Myös useimmat elektronisten kirjojen lukulaitteet, kuten Sony Reader, iLiad sekä tiedostonmuuntamisen jälkeen Amazonin Kindle tukevat PDF-formaattia.

Erityisesti kirjojen lukemiseen tarkoitetuilla ohjelmistoilla, kuten Mobipocket, Microsoft Reader tai eReader (Coyle, 2008) on myös omat formaattinsa. Tiedostot voivat olla joko DRM-suojatussa muodossa tai suojaamattomina ja niiden lukemiseen kehitetyt ohjelmat toimivat monipuolisesti erilaisissa tietokoneissa sekä älypuhelimissa. Lukuohjelmat ja niiden käyttämät tiedostot voivat olla sidottuja myös tiettyihin laitteisiin. Amazon Kindlen käyttämä AZW-formaatti sekä Sony Readerin oma LRF-tiedostomuoto toimivat ainoastaan valmistajien laitteissa.

Erityisesti ilmaisjakelussa olevaan elektroniseen kirjallisuuteen liittyvät HTML-formaatti sekä tekstimuodossa (.txt) olevat teokset. Lukemiseen sovel-

tuvia ohjelmistoja ovat tekstin kohdalla suurin osa tekstinkäsittelyohjelmista, HTML-tiedostojen soveltuessa parhaiten WWW-selaimiin.

Hieman harvemmin käytettyjä e-kirjamuotoja ovat erilaiset printtiteosten ulkoasua imitoivat formaatit. Adoben Flash-teknologiaan pohjautuvat e-kirjat ovat käytössä esimerkiksi British Libraryn sivustolla (Coyle, 2008), kuten myös Open Libraryn AJAX-pohjainen (Asynchronous JavaScript) e-kirja. Nämä formaatit näyttävät animaation avulla kääntyviä kirjan sivuja, joita voidaan selata hiiren avulla.

8. Verkkojakelun haasteet ja ongelmat

8.1 Tekijänoikeudet ja DRM

Tekijänoikeudet aiheuttavat monenlaisia haasteita e-kirjallisuuden verkkojakelun kohdalla. Vapaassa levityksessä (public domain) olevien teosten kohdalla joudutaan tasapainoilemaan eri maiden erilaisten lainsäädäntöjen suhteen. Yhdysvalloissa ja Euroopassa tekijänoikeus kestää yleisesti 70 vuotta tekijän kuoleman jälkeen, mikä on aiheuttanut teosten poistamisia esimerkiksi Project Gutenberg palvelusta (Towle et al., 2007).

Google Booksin käytäntö on myös ajankohtainen esimerkki e-kirjallisuuden sekä tekijänoikeuksien ongelmallisesta suhteesta. Googlen avulla tekijänoikeuksin suojatuista teoksista pystyy näkemään osan teoksen sisällöstä. Googlen mukaan tämä kuuluu ”reilun käytön” (fair use) piiriin (Leetaru, 2008), kirjailijoiden sekä kustantajien nähdessä asian toisin. Vuoden 2008 lokakuussa kuitenkin ilmoitettiin, että kiistan eri osapuolet ovat päässeet asiasta yhteisymmärrykseen ja Google voi korvauksia vastaan jatkaa sekä kehittää omaa käytäntöään (Economist, 2008).

DRM-suojaus aiheuttaa myös merkittäviä haasteita e-kirjojen myynnissä. Markkinoilla on myynnissä myös suojaamattomia e-kirjoja, mutta kaupallisten e-kirjojen kohdalla suojaus on hyvin yleinen käytäntö. Amazonin Kindle-kirjat ovat DRM-suojattuja ja sidottuja täysin siihen laitteeseen, johon ne ovat ostettu. Kindle-kirjojen lainaus, kopiointi sekä jälleenmyynti ovat kiellettyjä.

Kuluttajan oikeudet sen suhteen, missä laitteissa ostettuja e-kirjoja voi lukea tai voiko kirjoja lainata ja kopioida, ovat yhdessä tekijän saaman oikeudenmukaisen korvauksen kanssa haastava yhdistelmä. Towle ja muut (2007) näkevätkin DRM-suojausten ongelmallisuuden yhtenä merkittävänä tekijänä e-kirjojen toistaiseksi melko vaatimattoman menestyksen taustalla. Kuvaavaa tekijänoikeuksien ongelmallisuudesta onkin, että yksi tämän päivän suosituimmista kirjailijoista, Harry Potter sarjan luoja J.K. Rowling, on kieltänyt kokonaan kirjojensa elektronisten versioiden myynnin.

8.2 Formaatit

Hertherin (2005) mukaan koko teollisuuden hyväksymän standardin puute on selkeä ongelma mille tahansa tuotteelle. E-kirjallisuus ja sen verkkojakelu eivät ole tämän suhteen poikkeus. Siitä huolimatta, että jo kohta kymmenen vuoden ajan on ollut olemassa IDPF:n määrittelemä standardi sille, miten e-kirjat tulisi jäsentää, ovat formaatit yhä keskeinen ongelma.

IDPF-organisaation suosittelema epub-tiedostoformaatti sekä ISO standardisoitu Adoben PDF-formaatti ovat kaksi keskeistä muotoa, mutta suurista e-kirjapalveluista esimerkiksi Amazonin Kindle ei tue täysin vielä kumpaakaan.

Erilaisia verkossa jaettavia e-kirjaformaatteja on yhä edelleen paljon ja niiden yhteensopivuus eri lukulaitteiden ja ohjelmistojen kanssa on vaihtelevaa. Ongelmana on se, että käyttäjä joutuu valitsemaan lukulaitteen sekä käytetyn ohjelmiston perusteella, mitä kirjoja voi lukea, eri valmistajien e-kirjaformaattien ollessa monesti yhteensopimattomia. Youngin (2008) mukaan e-kirjallisuuden tulisikin pyrkiä kohti tilaa, jossa teosten lukeminen ei olisi kiinni laitteista.

8.3 Hinta, laatu ja nimikkeiden saatavuus

Elektronisten kirjojen hinta on yksi menestykseen vaikuttava tekijä. Towle ja muut (2007) näkevät ylihinnoittelun ongelmallisena, kustantajien halutessa tuotteilleen mahdollisimman hyvän tuoton. Tämä on keskeinen ongelma siinä suhteessa, että kuluttajat näkevät usein e-kirjan kustannuksiltaan halvempana tuottaa, minkä tulisi näkyä myös lopputuotteen hinnassa. Matalampaa hintaa voidaan perustella myös sillä, että ostettu e-kirja on periaatteessa vain ”vuokralla” käyttäjällään. Jälleenmyynti-, lainaus- tai muokkausoikeutta lukijalla ei siihen useimmiten ole. Verkosta ostettuun e-kirjaan liittyy myös aina riski siitä, millä varmuudella kirja toimii tulevaisuuden laitteissa sekä ohjelmistoissa.

Niissä tapauksissa, kun vertailukohtana on halvin printtiversio teoksesta, joka on yleensä pehmeäkantinen laitos, Amazon.comin elektroniset kirjat ovat tyypillisesti muutamia euroja paperiversioitaan halvempia. Näin ei kuitenkaan aina ole, ja osa teoksista on jopa samanhintaisia kuin printtiversionsa. Toisaalta myös selkeästi halvempia e-kirjoja löytyy.

Hinnat ovat erityisen vaihtelevia, kun tarkastelee eri e-kirjakauppojen valikoimia. Hinnoissa on jopa kymmenien eurojen eroja, ja e-kirjat ovat paikoitellen huomattavasti printtiversioitaan kalliimpia. Kuvassa 1 on pienimuotoinen hintavertailu Stephen Kingin teoksista. Hän on kirjailija, jonka e-kirjoja löytyy melko hyvin eri palveluista.

Teos	Amazon.com (printtikirjat)	Amazon Kindle	Fictionwise.com	Mobipocket.com	Sony eBook Store
Just After Sunset	16,8	9,99	10,79	17,99	11,99
Duma Key	9,99	9,99	10,79	17,99	9,99
Riding the Bullet	-	2	1,5	2,5	2,38
The Mist	6,99	5,59	4,19	6,99	6,64
Cell	9,99	7,99	5,99	9,99	8,99
The Stand	8,99	2,95	30	17,95	35
Carrie	-	1,75	19,5	17,95	22,75
Colorado Kid	5,99	3,99	2,99	4,99	4,74

Kuva 1 Stephen Kingin teosten hintavertailua Yhdysvaltain dollareina

Ilmaisten palveluiden kohdalla elektronisten kirjojen laatu on yksi olennaisimmista ongelmista (Berglund et al., 2004). Ilmaiskokoelmien kohdalla on myös usein tapana käyttää jo valmiiksi muiden tuottamia kirjoja, Project Gutenbergin laaja-alainen hyödyntäminen on tästä hyvä esimerkki. Tällainen kierrätys ja keskinäinen linkitys mahdollistavat huonolaatuisen e-kirjan leviämisen useisiin eri palveluihin.

Siitä huolimatta, että e-kirjojen määrä markkinoilla on viime vuosina jatkanut tasaista 20 prosentin vuosikasvuaan (Just, 2006), on e-kirjavalikoiman suppeus edelleen tosiasia. Suuntaa antavan kuvan kirjojen määrästä saa, kun tarkastelee Amazon.comin kaunokirjallisten Kindle-kirjojen lukumäärää suhteessa printtiteoksiin. Kindleen on mahdollista ostaa noin 88 000 teosta, kun paperille painettuja teoksia on myynnissä noin puoli miljoonaa.

Vaihtelua valikoiman suhteen on myös eri verkkokauppojen välillä. Amazonin omistuksessa oleva Mobipocket.com tarjoaa noin 50 000 kaunokirjaa ja Sonyn e-kirjakauppa noin 35 000 teoksen valikoiman. Suurimpienkin valikoimien kohdalla e-kirjat jäävät vielä toistaiseksi jälkeen tavallisten verkkokirjakauppojen tuotevalikoimasta.

9. Verkkojakelun hyödyt ja mahdollisuudet

Kaunokirjallisuuden verkkojakelun hyötykohdat eivät ole varsinaisesti erotettavissa elektronisen kirjallisuuden verkkojakelun hyödyistä. Kirjan sisältö ei ole avainasemassa, kun mietitään verkkojakelun potentiaalisia etuja ja mahdollisuuksia.

Verkkojakelulla on selkeästi mahdollisuus edistää merkittävällä tavalla kirjallisuuden kulttuurillista monimuotoisuutta. Modernit kirjat ovat lähes poik-

keuksetta jossain tuotantovaiheessa digitaalisessa muodossa ja sellaisenaan saatavilla, jolloin niitä ei ole edes tarvetta digitoida. Jos tekijänoikeudelliset seikat eivät estä julkaisua, moni sellainen teos, jolle ei ole enää riittävää kysyntää printtimarkkinoilla, olisi mahdollista julkaista elektronisessa muodossa. Rao (2001) näkee tässä mahdollisuuden jopa siihen, että koko ”poissa tuotannosta” (out of print) -aikakausi olisi mahdollista saattaa loppuun.

Sellaiset kirjat sekä kirjailijat, jotka eivät ole löytäneet itselleen kustantajaa, voivat hyödyntää myös verkkoa julkaisukanavana (Rao, 2004). Riippumatta siitä, päättääkö kirjailija jakaa kirjaansa ilmaiseksi vai maksua vastaan, tarjoaa WWW toimivan kanavan teoksen saattamiseksi lukijoilleen.

Verkkojakeluun tuotettujen teosten ei tarvitse kuitenkaan rajoittua pelkästään kaupallisiin teoksiin. Verkkojakelulla on jo osittain hyödynnettyä potentiaalia tuoda laaja-alaisesti saataville vanhoja sekä harvinaisia teoksia, joita ei muuten olisi mahdollista lukea.

Selkeänä hyötynä voidaan nähdä myös kirjan tuottamiseen tai hankintaan liittyvien kustannusten pieneneminen. Verkkojakelu tarjoaa helposti saatavilla olevia ilmaisia kirjakokoelmia, jolloin kustannukset kirjan lukijan kannalta ovat minimaaliset. Kaupallisella puolella e-kirjojen verkkojakelu poistaa myös tiettyjä kustannuksia. Kustantajan ei tarvitse enää maksaa teoksen painamiseen, varastointiin tai toimitukseen liittyviä kuluja (Rao, 2004).

Verkossa jaettujen e-kirjojen kohdalla kirjan hankkimisprosessista häviää myös toimitusaika (Towle et al., 2007; Rao, 2001). Oli kyseessä sitten ilmaispalvelusta hankittu tai ostettu elektroninen kirja, teos on käyttäjän saatavilla välittömästi. E-kirjoja myyvien palveluiden kohdalla saatavuuteen liittyy myös se hyvä puoli, että kerran ostettu teos on aina saatavilla. Esimerkiksi Amazonin sekä Fictionwisen kohdalla digitaalisen kirjahyllyyn talletettu teos on ladattavissa niin usein kuin haluaa, eikä sitä voi kadottaa.

10. Yhteenveto

Tässä tutkielmassa on esimerkkipalveluiden avulla selvitetty elektronisen kaunokirjallisuuden verkkojakelun nykytilaa, erilaisine muotoineen sekä jakelukanavineen. Sekä uutta että vanhaa kaunokirjallisuutta on saatavilla verkosta monipuolisesti erilaisissa formaateissa erilaisille alustoille.

Elektronisen kirjallisuuden verkkojakeluun liittyy kuitenkin lukuisia ongelmia aina sisällön määrän, tekijänoikeuksien kuin myös standardien laajemman hyväksynnän suhteen. Muun muassa nämä tekijät ovat aiheuttaneet, ja aiheuttavat yhä, haasteita sekä kirjojen tuottajille että kuluttajille.

Aivan viimeisten vuosien aikana erilaiset teknologiset kehitykset ovat vieneet e-kirjallisuutta suurin harppauksin eteenpäin. Tulevaisuudessa onkin mitä

todennäköisintä, että alati kehittyvä ja kasvava ala tulee tarjoamaan todellisen vaihtoehdon perinteiselle paperille painetulle kirjallisuudelle. Uusille sukupolville, joille WWW-palvelut sekä verkkojakelu ovat nykyisin jo arkipäivää, elektronisen kirjallisuuden omaksuminen tulee olemaan yhä tavanomaisempaa.

Elektronisten kirjamarkkinoiden kasvaessa myös alan tutkimus on entistä tärkeämpää. Tässä tutkielmassa hyvin suppeasti sekä pinnallisesti esiintuotuja ongelmia on kartoitettu aiemmin jonkin verran, mutta havaittujen, todellisten ongelmien mahdollisia ratkaisukeinoja huomattavasti vähemmän. Erityisesti kaupallisiin e-kirjoihin liittyvät ongelmat ovat ajankohtaisia ja niiden parissa onkin tutkimusalaa sekä pohdittavaa jatkoa ajatellen.

Viiteluettelo

Amazon (2008). Accessing basic web. Retrieved December 18, 2008, from http://www.amazon.com/gp/help/customer/display.html?nodeId=200137070&#downloading.

Armstrong, C. (2008). Books in a virtual world: the evolution of the e-book and its lexicon. *Journal of Librarianship and Information Science, 40(3),* 193-206.

Berglund, Y., Morrison, A., Wilson, R., & Wynne, M. (2004). An investigation into free ebooks. Retrieved December 18, 2008, from http://www.ahds.ac.uk/ litlangling/ebooks/report/FreeEbooks.

Connaway, L., & Wicht, H. (2007). What happened to the e-book revolution?: the gradual integration of e-books into academic libraries. *Journal of Electronic Publishing, 10(3).*

Coyle, K. (2003). The technology of rights: digital rights management. Retrieved December 18, 2008, from http://www.kcoyle.net/drm_basics.pdf

Coyle, K. (2008). E-reading. *Journal of Academic Librarianship*, *34(2)*, 160-162. Economist (2008, October 30). A new chapter. *The Economist*. Retrieved December 18, 2008, from http://www.economist.com/business/displaystory.cfm?story_id=12523914.

Gade, L. (2008, October 5). Sony reader prs-505. *MobileTechReview*. Retrieved December 18, 2008, from http://www.mobiletechreview.com/gadgets/Sony-Reader-PRS-505.htm.

Gingras, L., Escobar, M., Quintero, E., & Shah, T. (2008). Comparative analysis of e-readers: the sony reader prs-505. Retrieved December 18, 2008, from http://www.leannagingras.com/comparative-analysis.pdf.

Herther, N. K. (2005). The e-book industry today: a bumpy road becomes an evolutionary path to market maturity. *The Electronic Library*, *23(1)*, 45-53.

Hillesund, T. (2005). Digital text cycles: from medieval manuscripts to modern markup. *Journal Of Digital Information, 6(1).*

IDPF (1999). Open ebook publication structure specification version 1.0. Retrieved December 18, 2008, from http://www.idpf.org/oebps/oebps1.0/index.htm.

IDPF (2008). Industry statistics. Retrieved December 18, 2008, from http://www.idpf.org/doc_library/industrystats.htm.

Just, P. (2007). Electronic books in the USA – their numbers and development and a comparison to germany. *Library Hi Tech, 25(1),* 157-164.

Leetaru, K. (2008). Mass book digitization: the deeper story of google books and the open content alliance. *First Monday, 13(10).*

Nawotka, E. (2001, December 17). Mightywords closes down. *Publishers Weekly*. Retrieved December 18, 2008, from http://www.publishersweekly.com/ article/CA186715.html.

Nawotka, E. (2008). Our digital future. *Publishing Research Quarterly*, *24(2)*, 124-128.

Project Gutenberg (2008, December 10). Gutindex. Retrieved December 18, 2008, from http://www.gutenberg.org/dirs/GUTINDEX.ALL.

Rao, S. S. (2001). Familiarization of electronic books. *The Electronic Library*, *19(4)*, 247-256.

Rao, S. S. (2004). Electronic book technologies: an overview of the present situation. *Library Review*, *53(7)*, 363-371.

Rose, M. J. (2001, December 18). E-books live on after mighty fall. *Wired*. Retrieved December 18, 2008, from http://www.wired.com/culture/lifestyle/news/2001/12/49184.

Sony (2008, July 24). Reader digital book by Sony opens a new chapter on ebook formats. Retrieved December 18, 2008, from http://news.sel.sony.com/en/press_room/consumer/computer_peripheral/e_book/release/36245.html.

Towle, G., Dearnley, J.A., & McKnight, C. (2007). Electronic books in the 2003-2005 period: some reflections on their apparent potential and actual development. *Publishing Research Quarterly*, *23(2)*, 95–104.

Vassiliou, M., & Rowley, J. (2008). Progressing the definition of "e-book". *Library Hi Tech, 26(3),* 355-368.

Young, S. (2008). Beyond the flickering screen: re-situating e-books. *M/C Journal*, *11(4)*. Retrieved December 18, 2008, from http://dev.mediaculture.ci.qut.edu.au/ojs/index.php/mcjournal/article/viewArticle/61

Työryhmäominaisuuksien tuki pääsynvalvontaan

Jukka Similä

Tiivistelmä.
Verkkojulkaisujärjestelmissä on harvoin keskitytty työryhmätoimintojen toteutukseen, ja tämä heijastuu niiden pääsynvalvontamenetelmiin. Tässä tutkielmassa selvitetään, mitä työryhmäominaisuuksien toteuttaminen verkkojulkaisujärjestelmään vaatii pääsynvalvonnalta, ja vertaillaan eri pääsynvalvontamenetelmien soveltuvuutta työryhmätoimintojen pääsynvalvonnan toteuttamiseen.

Avainsanat ja -sanonnat: Pääsynvalvonta, verkkojulkaisujärjestelmä, RBAC, TMAC.
CR-luokat: D.4.6, K.6.5, H.4.1, H.3.4

1. Johdanto

Suuri osa nykyaikaisista verkkosivustoista toimii niin sanottujen julkaisujärjestelmien avulla. Nämä järjestelmät tarjoavat helppokäyttöisen käyttöliittymän ainakin yksittäisten verkkosivujen luontiin, sivujen päivittämiseen ja vanhojen sisältöjen poistamiseen. Yksi tällainen julkaisujärjestelmä on nimeltään Joomla. Monien muiden julkaisujärjestelmien tavoin Joomla on suunniteltu sivuston ylläpitäjän työkaluksi, ja pääylläpitäjä jakaa käyttöoikeuksia sisällön hallitsemiseksi myös sivuston rekisteröityneille käyttäjille. Käyttöoikeusrakenne nykyisessä Joomlassa on hierarkkinen, sillä käyttäjillä on yksi käyttäjäryhmä, johon he kuuluvat, ja ylemmät ryhmät sisältävät alempien käyttöoikeudet. Tämä rakenne ei salli kovin monipuolista pääsynvalvontaa, joka tarkoittaa sitä järjestelmää ja tietorakennetta, joka ohjelmiston taustalla ratkaisee, onko käyttäjällä oikeutta suorittaa pyydettyä toimintoa. Toiminto voi olla esimerkiksi kuvan lataaminen verkosta, artikkelin julkaiseminen, tekstin muokkaus tai vaikkapa vain julkaistujen artikkelien listaaminen. Erilaisia käyttöoikeuksien tallentamis- ja tarkistustapoja on tutkittu runsaasti, mutta avoimen lähdekoodin verkkojulkaisujärjestelmissä on käytetty lähinnä yksinkertaisia rakenteita, jotka on helppo ymmärtää ja joita on tehokas käyttää. Nämä yksinkertaiset rakenteet eivät kuitenkaan riitä, kun toteutetaan monipuolisia työryhmätoimintoja, joita nykyään halutaan lähes kaikkiin ohjelmistoihin verkkosivustot mukaan lukien. Esimerkki tällaisesta toiminnosta voi olla työryhmän sisäinen dokumenttipankki, johon pääsyä pitää voida rajoittaa käyttötavan mukaan, mutta jossa dokumenttien käyttöoikeuksia pitää voida jakaa myös työryhmän ulkopuolelle.

Työryhmätoiminnot vaativat pääsynvalvonnalta enemmän ominaisuuksia kuin perinteiset käyttäjätoiminnot. Myös verkkoympäristön erityispiirteet tuovat lisää vaatimuksia. Tilaton (stateless) ympäristö, jollainen verkkoympäristö on, ei salli käyttöoikeuksien pitämistä muistissa samalla tavalla kuin tilallinen (stateful) ympäristö. Tolone ja muut [2005] ovat tutkineet ryhmätyön pääsynvalvontavaatimuksia tilallisessa ympäristössä. Tässä tutkielmassa selvitetään, mitä ominaisuuksia verkkoympäristön työryhmätoiminnot tarvitsevat pääsynvalvonnalta ja mitkä pääsynvalvontamallit parhaiten tarkoitukseen soveltuvat. Pääsynvalvonnan tulee luonnollisesti tarjota sisällönhallinnan perustoimintojen (selaus, avaus, lisäys, muokkaus ja poisto) käyttöoikeustarkistukset helposti, ymmärrettävästi ja tehokkaasti. Lisäksi sen tulee tarjota työryhmäkomponenttien käyttöön sellaiset ominaisuudet, että käyttäjät voivat jakaa dokumenttien käsittelyoikeuksia keskenään, kuitenkaan poistamatta pääylläpitäjän mahdollisuutta hallita käyttöoikeuksia yleisellä tasolla. Lähtökohtana tutkielmassa on tämän vuoksi ollut *rooliperustainen pääsynvalvonta* (role based access control, RBAC) [Ferraiolo and Kuhn, 1992], jonka yksi tavoite on pääsynvalvonnan hallinnan säilyttäminen selkeänä. Tutkielmassa käsitellään rooliperustaisen pääsynvalvonnan eri variaatioita työryhmäominaisuuksien näkökulmasta ja vertaillaan niitä Joomlan nykyiseen pääsynvalvontaan. Variaatioiden ymmärtämiseksi on käsitelty myös niitä menetelmiä, joihin RBAC perustuu tai jotka se korvaa.

2. Pääsynvalvontamenetelmät

Tässä luvussa esitellään lyhyesti pääsynvalvontamenetelmät, jotka oleellisesti liittyvät rooliperustaiseen pääsynvalvontaan ja sen käyttämiseen työryhmäsovelluksissa tai Joomlassa. Käsitteitä *tietue* ja *olio* käytetään yleisesti tarkoittamaan toiminnan kohteena olevan tiedon yksittäistä ilmentymää, ja Joomla-verkkojulkaisujärjestelmän kannalta ne tarkoittavat samaa kuin tietokannan yhden taulun yksi rivi.

2.1. Perusmenetelmät

Pääsynvalvontamallit voidaan yleisesti jakaa *harkinnanvaraisiin* (discretionary access control, DAC) ja *ei-harkinnanvaraisiin* (non-discretionary access control, NDAC) malleihin sillä perusteella, voivatko käyttäjät kyseisessä mallissa jakaa käyttöoikeuksia eteenpäin. Harkinnanvarainen pääsynvalvonta [Hu et al., 2006] on yleisimmin käytössä oleva pääsynvalvontamalli. Se sallii käyttäjien määrittävän ja jakavan käyttöoikeuksia tietueille, joihin heillä itsellään on muokkausoikeus. DAC:n tärkeä ominaisuus on se, että tietueilla on aina omistaja (owner),

joka voi päättää tietueen käyttöoikeuksista ja jakaa niitä muille käyttäjille. Perinteinen esimerkki harkinnanvaraisesta pääsynvalvonnasta on UNIX-käyttöjärjestelmän tiedostojärjestelmän luku-, kirjoitus- ja suoritusoikeudet, joita käyttäjät voivat määrittää itse.

Pakollinen pääsynvalvonta (mandatory access control, MAC) on pääsynvalvontamalli, jossa minkä tahansa järjestelmällä suoritettavan toiminnon käyttöoikeus tarkistetaan keskitetysti. Aina kun käyttäjä yrittää käyttää jotain toimintoa, avata tietuetta tai esimerkiksi lukea kansion sisältöä, käyttöoikeus tarkistetaan käyttöoikeussäännöistä, joihin käyttäjällä ei ole mahdollisuutta vaikuttaa. [Hu et al., 2006.] Tämän vuoksi pakollinen pääsynvalvonta ei sovellu työryhmäohjelmistojen käyttöön ainoana pääsynvalvontamenetelmänä.

Harkinnanvarainen pääsynvalvonta toteutetaan usein *pääsynvalvontalistoilla* (access control lists, ACL). Tietueille liitetty pääsynvalvontalista sisältää tiedon siitä, kuka tai mikä saa käsitellä tietuetta ja miten. Tyypillisesti pääsynvalvontalistan rivi sisältää käyttäjän ja toiminnon, esimerkiksi (Alice, delete) ja tietueeseen liitettynä tämä tarkoittaa, että Alice saa poistaa kyseisen tietueen. [Wikipedia, 2008.] Pääsynvalvontalista ei rajoita listan rivin tekijätietoa tarkoittamaan vain käyttäjää, vaan se voi tarkoittaa myös esimerkiksi käyttäjäryhmää tai tietueen omistajaa.

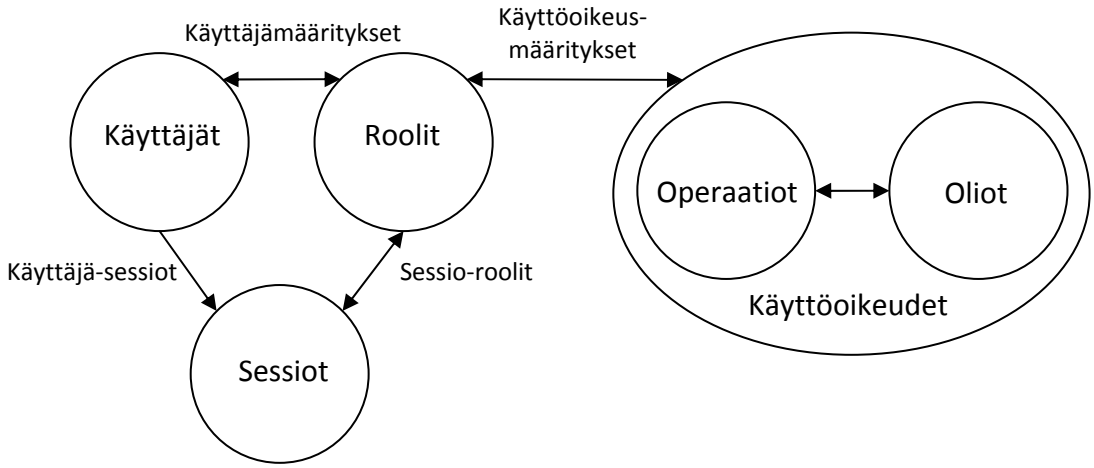
2.2. Kehittyneet menetelmät

Rooliperustainen pääsynvalvonta (RBAC) eroaa pakollisesta ja harkinnanvaraisesta pääsynvalvonnasta eniten siinä, että käyttöoikeussäännöt kohdistetaan toimintoihin tietueiden tai olioiden sijaan. Käyttöoikeudet määritetään rooleille, joita käyttäjällä voi olla useita. Osborn ja muut [2000] ovat osoittaneet, että RBAC:lla on mahdollista toteuttaa sekä pakollinen että harkinnanvarainen pääsynvalvontakäytäntö. Tähän tarvitaan RBAC96-mallin [Sandhu et al., 1996] mukaiset roolihierarkiat ja rajoittimet. Voidaan todeta, että ainakin teoriassa mikä tahansa nykyinen MAC- tai DAC-toteutus voidaan korvata RBAC:lla. Tosiasiassa tämä ei ole järkevää, sillä harkinnanvaraisen käytännön toteuttaminen RBAC:lla tarkoittaisi roolien ja rajoitusten määrän kasvua niin suureksi, että sen hallinta ei olisi tarkoituksenmukaista [Hu et al., 2006].

Aiempien pääsynvalvontamenetelmien ongelmana on ollut käyttöoikeuksien hallinnan hankaluus. Suurissa järjestelmissä yksittäiselle käyttäjälle liitetyt käyttöoikeudet on työlästä toistaa toiselle käyttäjälle, jonka tehtävä organisaatiossa voi olla sama, jolloin käyttöoikeuksien tulisi myös pääsääntöisesti olla samat. Rooliperustaisessa pääsynvalvonnassa näille käyttäjille asetetaan samat roolit ja käyttöoikeudet määritellään rooleille. Tällöin käyttäjän poistuessa organisaatiosta tai vaihtaessa tehtäviä hänet voidaan poistaa roolista, eikä käyttöoikeuksia tarvitse muuttaa. RBAC:hen sisältyy muitakin tärkeitä ominaisuuk-

sia, mutta tämä hallinnan helppous on sen lähtökohta ja syy, miksi se on kehitetty.

NIST-standardin [Ferraiolo et al., 2001] mukainen RBAC-viitekehys muodostuu neljästä kokonaisuudesta: *ydin-RBAC*, *hierarkkinen RBAC*, *staattinen velvollisuuksien erottaminen* (static separation of duty, SSD) ja *dynaaminen velvollisuuksien erottaminen* (dynamic separation of duty, DSD). Ydin-RBAC sisältää RBAC:n perusajatuksen – käyttäjät liitetään rooleihin, käyttöoikeudet liitetään rooleihin ja käyttäjät saavat käyttöoikeuksia niiden roolien kautta, joihin he kuuluvat. Kuvassa 1 esitetään ydin-RBAC:n rakenne.



Kuva 1: Ydin-RBAC:n rakenne [Ferraiolo et al., 2001]

NIST-RBAC:n mukaan järjestelmän tarvitsee toteuttaa vain ydin-RBAC ollakseen standardinmukainen, muut kokonaisuudet ovat valinnaisia ja niiden käyttäminen riippuu ominaisuuksien tarpeellisuudesta kyseisessä järjestelmässä. [Ferraiolo et al., 2001.] Ydin-RBAC:ssa määritellyt sessiot (kuvassa 1) pitävät kirjaa käyttäjällä aktiivisena olevista rooleista. Ne ovat käyttäjän käynnistämiä, ja niitä voi olla käyttäjällä monia. Yhdellä sessiolla on yksi käyttäjä, mutta siinä voi olla monta roolia aktiivisena, ja yhdellä roolilla voi olla monia sessioita. Käyttäjälle määritetyt käyttöoikeudet ovat ne käyttöoikeudet, jotka kuuluvat niille rooleille, jotka ovat aktiivisena kaikissa käyttäjällä aktiivisena olevissa sessioissa.

Hierarkkinen RBAC määrittää roolihierarkian joko yleisenä tai rajoitettuna. Yleisessä roolihierarkiassa roolilla voi olla useita vanhempia, joiden oikeudet periytyvät roolille. Rajoitetussa hierarkiassa vanhempia voi olla vain yksi. Pääsynvalvontamenetelmien ominaisuuksien vertailussa luvussa 4 rooliperustaisen pääsynvalvonnan ominaisuuksiin luetaan mukaan hierarkkinen RBAC.

Velvollisuuksien erottamiskomponentit (SSD ja DSD) tarkoittavat käytännössä samaa kuin RBAC96-mallin rajoitukset. Roolit voivat olla ristiriidassa toistensa kanssa, jolloin niitä ei saa olla samalla käyttäjällä määriteltynä (SSD)

tai yhdellä käyttäjällä samaan aikaan aktivoituna (DSD). Tällainen ristiriita voi syntyä esimerkiksi rooleille pankkilainan hakija ja myöntäjä. Lainan hakija voi olla töissä pankissa, josta on hakemassa lainaa, ja saattaa myöntää lainoja muille asiakkaille. Hän ei kuitenkaan saa hyväksyä omaa lainaansa. Tällöin roolit hakija ja myöntäjä eivät saa olla samaan aikaan aktivoituna.

RBAC:n laajennuksista huomioidaan *ryhmäperustainen pääsynvalvonta* (team based access control, TMAC) [Thomas, 1997] ja *kontekstitietoinen pääsynvalvonta* (context-based TMAC) [Georgiadis et al., 2001]. Ryhmäperustaisessa pääsynvalvonnassa käyttäjät liitetään työryhmiin ja näille työryhmille liitetään rooleja, joille sitten määritellään käyttöoikeudet. Thomas [1997] perustelee ryhmäperustaisen pääsynvalvonnan tarpeellisuutta kahdella pääsyyllä: pääsynvalvonnan tulee voida hallita käyttöoikeuksia yleisellä tasolla, mutta samaan aikaan yksityiskohtaisesti yksittäisille olioille, ja lisäksi ryhmätyöskentelyn tila pitää voida huomioida käyttöoikeuksia päätettäessä. Lisäksi ryhmäperustaisen pääsynvalvonnan tapa ryhmitellä käyttäjät työryhmiin, joille liitetään rooleja, on intuitiivinen ja soveltuu moneen organisaatioon suoraviivaisesti. TMAC määrittelee kaksi ryhmien yhteistyöhön liittyvää kontekstia, käyttäjäkontekstin ja oliokontekstin. Käyttäjäkonteksti tarjoaa tavan tunnistaa yksittäiset käyttäjät joilla on tietty rooli työryhmässä, ja oliokonteksti tarjoaa mahdollisuuden tunnistaa yksittäiset olioiden instanssit ryhmätyön tarpeita varten. [Georgiadis et al., 2001.] Kontekstitietoinen pääsynvalvonta hienosäätää ja yhdistää RBAC:n ja TMAC:n ominaisuuksia, ja se on kehittyneempi versio alkuperäisestä TMAC:n ideasta. Siinä kontekstitieto on oma osionsa pääsynvalvontamallissa, ja se voi sisältää myös muita tietoja kuin käyttäjä- tai oliokontekstin, esimerkiksi ajan tai paikan. Tässä tutkielmassa kontekstitieto jätetään vähemmälle huomiolle, sillä verkkojulkaisujärjestelmän käyttötarpeissa konteksti on yleensä tarkasti rajattu ja melko hyvin ennakoitavissa. Kontekstitietoinen pääsynvalvonta on myös vielä kehittyvä malli, jonka ongelmia ei ole selvitetty riittävästi [Chen and Crampton, 2008]. Ryhmäperustaisen pääsynvalvonnan idea työryhmistä ja niille liitettävistä rooleista on kuitenkin niin helposti omaksuttavissa, että se kannattaa huomioida vertailtaessa menetelmien soveltuvuutta, sillä Tolonen ja muiden [2005] huomioissa pääsynvalvonnan ymmärrettävyys on yksi tärkeä edellytys sen käyttöönotolle.

2.3. Joomlan nykyinen pääsynvalvonta

Joomla! on verkkojulkaisujärjestelmä, josta on keväällä 2008 ilmestynyt uusi, taaksepäin yhteensopimaton versio (Joomla! 1.5). Uuteen versioon on toteutettu yhteensopivuuskirjasto, jolla vanhan 1.0-version komponentit toimivat myös uudessa järjestelmässä. Pääsynvalvonnan kannalta tämä tarkoittaa, että käytössä on kaksi erillistä tapaa tarkistaa käyttöoikeudet. Joomlan nykyinen pääsyn-

valvontamekanismi on taaksepäin yhteensopivan vanhan version ja uuden keskeneräisen toteutuksen yhdistelmä. Vanha versio oli pakollinen pääsynvalvontamalli, jossa käyttäjäryhmien käyttöoikeuksia ei voinut muokata dynaamisesti. Käyttöoikeudet määriteltiin kovakoodattuna sen perusteella, mihin ryhmiin käyttäjä kuului, ja käyttäjä ei voinut kuulua kuin yhteen ryhmään. Uusi malli on toteutettu pääsynvalvontalistoilla, jotka teoriassa sallivat dynaamisen käyttöoikeuksien määrittämisen ja uusien käyttäjäryhmien luomisen. Toteutuksen keskeneräisyyden vuoksi uusi malli ei kuitenkaan edelleenkään salli kuin yhden käyttäjäryhmän käyttäjää kohti, eikä käyttöoikeuksia voi rajoittaa tai määrittää kuin ohjelmoimalla ne ohjelmiston komponentteihin. [Kennard, 2007.]

Työryhmätoiminnot vaativat pääsynvalvontamenetelmältä enemmän ominaisuuksia kuin perinteiset käyttäjätoiminnot. Pääsynvalvonnan tulee voida suojata minkä tahansa tyyppistä tietoa ja resursseja vaihtelevalla tarkkuudella. Sen pitää siis pystyä suojaamaan jaettu ympäristö ja sen käsittämät oliot, mutta myös tarjota mahdollisuus yksittäisten olioiden ja niiden ominaisuuksien hienovaraisempaan hallintaan. [Tolone et al., 2005] Joomlan uusi pääsynvalvontamenetelmä periaatteessa toteuttaa tämän vaatimuksen, mutta pääsynvalvonnalta vaaditaan lisäksi mahdollisuus määrittää käyttöoikeudet yleisellä tasolla [Tolone et al., 2005]. Joomlan pääsynvalvontatoteutuksella ei voida näitä molempia vaatimuksia toteuttaa samanaikaisesti, sillä yksittäisten olioiden hallintaan tehtävät rajoitukset vaativat pääsynvalvontalistaan omat merkintänsä samantasoisena kuin yleiset käyttöoikeudet. Tällä tavalla työryhmätoimintojen vaatimat käyttöoikeusmääritykset monimutkaistavat niiden hallintaa ja virheiden mahdollisuus kasvaa. Joomlan ACL-toteutus ei myöskään salli ajonaikaisten käyttöoikeusmääritysten tekemistä [Kennard, 2007], jota vaaditaan työryhmätoimintojen pääsynvalvonnalta [Tolone et al., 2005]. Jo näiden puutteiden vuoksi on syytä tutkia vaihtoehtoisia pääsynvalvontamenetelmiä, kun ollaan toteuttamassa työryhmätoimintoja Joomla-verkkojulkaisujärjetelmään.

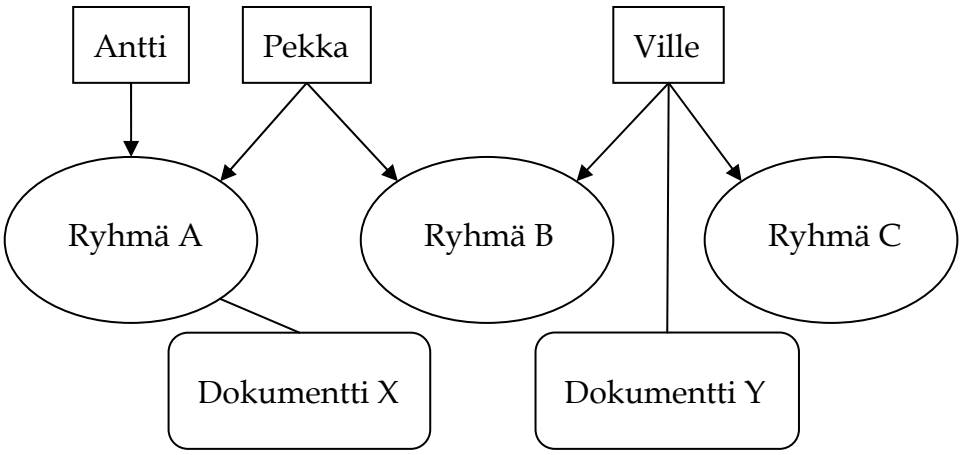
3. Pääsynvalvonnalta vaadittavat ominaisuudet

Tässä luvussa käydään läpi työryhmätoimintojen yleisiä vaatimuksia pääsynvalvonnalle. Tolone ja muut [2005] toteavat, että pääsynvalvontamenetelmän tulee olla niin yksinkertainen, että käyttäjät ymmärtävät sen, tai se sivuutetaan. Avoimen lähdekoodin järjestelmässä on huomioitava paitsi loppukäyttäjät, myös järjestelmän kehittäjät. Pääsynvalvontamenetelmän tulee olla niin yksinkertainen, että lisäosien kehittäjät haluavat mieluummin käyttää sitä kuin tehdä omat – mahdollisesti turvattomat – pääsynvalvontatarkistuksensa komponentteihinsa. Lisäksi kehittäjien tulee ymmärtää pääsynvalvonnan toiminta, jotta he

eivät vahingossa ohjelmoi turvallisuusaukkoja järjestelmään. Samaan aikaan pääsynvalvonnan tulee olla niin monipuolinen, ettei kehittäjille tule tarvetta tehdä omia toteutuksiaan menetelmän rinnalle.

3.1. Työryhmätoimintojen tuomat vaatimukset

Tolone ja muut [2005] ovat selvittäneet eri pääsynvalvontamenetelmien soveltuvuutta työryhmäympäristöön. He ovat muodostaneet työryhmäympäristön pääsynvalvonnan ominaisuuksista teoreettisen listan, jossa vaaditaan pääsynvalvonnalta muun muassa ”läpinäkyvää pääsyä sallituille käyttäjille ja vahvaa suojausta kielletyille käyttäjille joustavalla tavalla, joka ei rajoita yhteistyötä”. Tämän kaltainen yleinen lista on hyödyllinen tiettyyn pisteeseen asti, mutta eri pääsynvalvontamenetelmien hyödyllisyyttä arvioidessa tarvitaan myös konkreettisia esimerkkejä. Seuraava esimerkki havainnollistaa työryhmäominaisuuksien vaatimusten monimutkaisuutta. Kuvassa 2 esitellään organisaatiorakenne, jossa joukko käyttäjiä kuuluu erilaisiin ryhmiin. Sen jälkeen esitetään käyttäjien toiminnasta lista kysymyksiä, joihin pääsynvalvonnan tulisi pystyä vastaamaan.



Kuva 2: Yksinkertainen esimerkki työryhmistä

Kuvassa 2 ryhmään A kuuluvat käyttäjät Antti ja Pekka, ryhmään B Pekka ja Ville, ja ryhmään C vain Ville. Dokumentti X on tehty ryhmän A käyttöön ja dokumentti Y on Villen tekemä. Kysymykset, joihin pääsynvalvonnan tulee pystyä vastaamaan ovat seuraavat:

1. Saako Pekka lukea dokumentin Y?
2. Saako Ville muokata dokumenttia X?
3. Saako Antti tietää, ketä kuuluu ryhmään C?
4. Mitä dokumentteja Pekka saa muokata?
5. Kuka voi lukea dokumentin X?
6. Ketä kuuluu ryhmään C?

7. Mihin ryhmiin Pekka kuuluu?

Luonnollisesti pääsynvalvonnalle esitetään monimutkaisempiakin kysymyksiä, mutta edellä esitetyistä kysymyksistä saa kuvan siitä, miten työryhmätoiminnot vaikuttavat pääsynvalvonnan vaatimuksiin. Kysymyksistä voidaan myös huomata, että ne ovat eritasoisia. Ensimmäiset kolme kysymystä ovat muotoa ”saako käyttäjä suorittaa tietyn toiminnon”. Neljäs kysymys voidaan yleistää muotoon ”mitä toimintoja käyttäjä saa suorittaa”. Loput kysymyksistä ovat järjestelmän tietoja, joissa käyttäjätietoa ei tarvita vastauksen selvittämiseksi, mutta se tarvitaan, jotta tiedetään, voidaanko kysymykseen vastata – tavallinen käyttäjä ei välttämättä saa tietää ryhmään C kuuluvia käyttäjiä, vrt. kysymykset 3 ja 6. Tolone ja muut [2005] kutsuvat tätä metapääsynvalvonnaksi tai pääsynvalvonnan hallinnaksi. Pääsynvalvontaan pitää pystyä määrittämään ylläpitäjät, jotka voivat hallita käyttöoikeuksia eritasoisesti, tai tarvitaan erillinen pääsynvalvontamekanismi niiden hallitsemiseksi.

Pääsynvalvonnalle pitää myös pystyä kertomaan, kuka tai mikä saa käyttää dokumentteja. Kuvan 2 mukaisessa tilanteessa Villen pitää voida antaa Pekalle lukuoikeus dokumenttiin Y joko antamalla käyttöoikeus ryhmälle B tai suoraan käyttäjälle Pekka. Pääsynvalvontaan pitää voida asettaa käyttöoikeus siten, että kun dokumentti kuuluu ryhmään A, sitä voi lukea kuka tahansa, mutta jos se kuuluu ryhmään C, sitä voi lukea vain kyseisen ryhmän jäsenet. On turhaa yrittää keksiä esimerkkiä kaikista käyttötilanteista, mutta näillä edellä mainituilla esimerkeillä pyritään osoittamaan työryhmäympäristön vaatimusten monimutkaisuus.

3.2. Ympäristön aiheuttamat ongelmat

Työryhmien tuoma monimutkaisuus on ongelma itsessään, mutta se kasvaa suuremmaksi verkkoympäristön asettamien rajoitusten vuoksi. HTTP-protokolla, jolla verkkosivut toimitetaan käyttäjän selaimelle, on tilaton. Tämä tarkoittaa käytännössä sitä, että kahden yksittäisen sivulatauksen välillä ei ole yhteyttä. Tilaa seurataan evästeiden ja käyttäjäsessioiden avulla, ja muun muassa monessa Java-kielellä toteutetussa sovelluspalvelimessa oliot voidaan pitää muistissa sivulatausten välillä. Joomla!-verkkojulkaisujärjestelmä on ohjelmoitu PHP-kielellä, jolla yhden sivulatauksen hoitaa yksi prosessi. Mahdolliset muistiin ladatut pääsynvalvontatiedot häviävät prosessin päätyttyä, jolloin ne pitää ladata esimerkiksi tietokannasta joka sivulatauksella erikseen. Muistiin tallentamiseen on olemassa työkaluja kuten *memcached* [Fitzpatrick, 2008] tai *APC* [PHP Group, 2008], mutta ne eivät ole kovinkaan usein käytettävissä esimerkiksi webhotellipalveluissa. Joomlan tarkoituksena on toimia ympäristössä, jossa olioiden ja niiden käyttöoikeuksien pitäminen muistissa sivulatausten välissä

ei ole mahdollista, joten pääsynvalvonnan tulee olla riittävän tehokas käyttöoikeuksien lataamiseen tietokannasta joka sivulatauksella.

4. Pääsynvalvontamallien soveltuvuus työryhmätoimintaan

Tässä luvussa käydään läpi luvussa kaksi esitellyt pääsynvalvontamenetelmät ja verrataan niitä toisiinsa yksinkertaisuuden, ymmärrettävyyden, tehokkuuden ja ominaisuuksien puolesta.

4.1. Yksinkertaisuus ja ymmärrettävyys

Yksinkertaisin pääsynvalvontamalli [Tolone et al., 2005] on Lampsonin [1971] esittelemä *pääsymatriisi*. Se on käsitteellinen malli, jonka toteutuksia ovat muun muassa pääsynvalvontalistat. Pääsynvalvontalistojen käyttöä ja harkinnanvaraisen pääsynvalvonnan toteuttamista pääsynvalvontalistoilla puoltaa niiden yksinkertaisuus. Mallit ja niiden toiminta on helppo ymmärtää, mikäli järjestelmän monimutkaisuus pysyy hallittavissa mittasuhteissa.

Rooliperustainen pääsynvalvonta on monimutkaisempi käsitteiltään ja rakenteeltaan, ja kun toteutukseen otetaan mukaan hierarkia ja rajoitukset, pääsynvalvonnan toiminta voi olla monelle liian vaikea hahmottaa. Esimerkiksi Schwartz [2006a, 2006b] esittelee pääsynvalvontamallin, jota hän kutsuu rooliperustaiseksi pääsynvalvonnaksi. Kun mallia tutkitaan tarkemmin, huomataan kuitenkin, että kyseessä on pääsynvalvontalista, johon on yhdistetty pääsymatriisin käyttö, eikä varsinaisesti rooliperustainen pääsynvalvontamalli. Rooliperustaisen pääsynvalvonnan ongelmana ovatkin monet erilaiset toteutustavat ja jopa mielipide-erot pääsynvalvonnan toimintatavoista. Ryhmäperustaisen pääsynvalvonnan ymmärtämiseksi vaaditaan ensin rooliperustaisen pääsynvalvonnan sisäistäminen, joten se on vielä hankalampi vaihtoehto. Ryhmäperustaisen pääsynvalvonnan käyttäjä- ja oliokontekstit ovat vaikeaselkoisia käsitteitä, joiden toteuttamiseksi ja käyttämiseksi pitää ymmärtää pääsynvalvonnan toiminta syvällisesti. Ryhmäperustaisen pääsynvalvonnan ymmärtämistä kuitenkin helpottaa sen rakenne, joka kuvaa läheisesti reaalimaailman organisaatiorakenteita.

Vaikeimmin hallittava on kontekstitietoinen pääsynvalvonta, jossa kontekstitieto on mahdollisesti liian abstrakti käsite tavalliselle käyttäjälle tai jopa järjestelmäkomponenttien kehittäjille. Chen ja Crampton [2008] toteavat monimutkaisuuden olevan erilaisten kontekstitietoisten pääsynvalvontamallien yleinen ongelma. Lisäksi he kertovat, että tutkituissa kontekstitietoisissa malleissa voi esiintyä ristiriitatilanteita ja pääsynvalvontamäärittelyjen epämääräisyyksiä.

4.2. Tehokkuus

Tehokkain tapa käyttöoikeuksien hallintaan niiden tarkistusta ajatellen on tallentaa ne suoraan tietueen yhteyteen, kuten pääsymatriisilla voidaan tehdä. Verkkojulkaisujärjestelmässä juuri käyttöoikeuksien haun ja tarkistuksen tulee olla tehokasta, sillä niiden muokkaus on yleensä sikäli harvinainen toiminto, ettei sen tehokkuudella ole käytännön merkitystä. Pääsynvalvontalistat ovat yksinkertaisuutensa vuoksi yleensä tallennettu pienempään määrään tietokantatauluja, kuin rooli- tai ryhmäperustaisen pääsynvalvonnan tiedot, jolloin niiden voisi myös kuvitella olevan tehokkaampi tapa tarkistaa käyttöoikeudet.

Pääsynvalvontalistoilla voidaan kuitenkin toteuttaa monimutkaisia rakenteita ja niiden tehokkuus riippuu valitusta tallennustavasta. Yhdenlainen pääsynvalvontalista voi vastata kysymykseen ”kuka saa käyttää tätä tietuetta” vakioajassa, mutta kysymykseen ”mitä tietueita tämä käyttäjä saa käyttää” vastatakseen se joutuu käymään läpi vähintään lineaarisesti kasvavan kyselymäärän. Erilaisella pääsynvalvontalistalla tilanne voi olla päinvastainen. [Hu et al., 2006.] Monimutkainen ryhmähierarkia pääsynvalvontalistoilla toteutettuna saattaa olla tehottomampi kuin ryhmä- tai rooliperustaisella tavalla toteutettuna.

Rooli- ja ryhmäperustaisilla tavoilla käyttöoikeuksien haun tehokkuus ei muutu käyttäjämäärän kasvaessa, vaan niissä vaikuttaa lähinnä roolien määrä. Tällöin voidaan ajatella, että mitä suurempi järjestelmä on, sitä tehokkaampia nämä tavat ovat verrattuna muihin pääsynvalvontamalleihin. Kun näitä menetelmiä käytetään työryhmäominaisuuksien pääsynvalvonnan toteuttamiseen, saattaa roolien määrä kuitenkin riippua työryhmien määrästä [Wang, 1999]. Tällöin tehokkuus kärsii järjestelmän koon kasvaessa. Kontekstitietoinen pääsynvalvonta ei muuta tilannetta, sillä se laajentaa ryhmäperustaista pääsynvalvontaa hienojakoisemmaksi, ei yleisemmäksi työryhmien suhteen.

4.3. Ominaisuudet ja käyttöoikeuksien hallinta

Yksinkertaisimman pääsynvalvonnan, pääsymatriisin, ominaisuuksia on turha yrittää käyttää työryhmätoimintoihin. Sillä voidaan määrittää käyttöoikeudet yksittäiselle tietueelle tai käyttäjälle, mutta mihinkään sen monimutkaisempaan toimintoon ei kannata ryhtyä. Käyttöoikeuksien hallinta pääsymatriisilla kävisi liian monimutkaiseksi ja työlääksi jo pienessä organisaatiossa.

Pääsynvalvontalistoilla voidaan teoriassa toteuttaa riittävän monipuolinen pääsynvalvontamalli työryhmätoimintojen tarpeisiin yhdistämällä useampia pääsynvalvontalistoja samaan pääsynvalvontamalliin. Käyttöoikeuksien hallinta on tällöin vaikeampaa kuin rooli- tai ryhmäperustaisilla tavoilla, kun käyttöoikeuksia todennäköisesti joutuu määrittämään useaan listaan yhtaikaa. [Hu et

al., 2006.] Rooliperustainen pääsynvalvonta saattaa riittää työryhmätoimintojen käyttöön sellaisenaan, jos työryhmiä on rajattu, ennalta tiedetty määrä. Tällöinkin tarvitaan koko RBAC-standardin toteutus hierarkia ja velvollisuuksien erottaminen mukaan lukien. Joshi ja muut [2001] toteavat RBAC:n olevan erittäin hyödyllinen työryhmäominaisuuksien ja tietovirran hallintaan, mutta toteavat näihin vaadittavan kuitenkin RBAC:n laajennuksia.

Ryhmäperustainen pääsynvalvonta mahdollistaa ryhmien muokkaamisen roolien määrään koskematta, joten se on hyödyllinen laajennus työryhmätoimintojen toteuttamiseen rooliperustaisella pääsynvalvonnalla. Kontekstitietoinen pääsynvalvonta ei tarjoa lisäominaisuuksia Joomla-järjestelmän tarpeisiin, sillä sen ominaisuuksia voidaan hyödyntää lähinnä hajautetuissa työryhmäympäristöissä.

4.4. Hybridimalli

Työryhmäominaisuuksien toteuttamiseen parhaat toiminnot löytyvät ryhmäperustaisesta tai vastaavasta pääsynvalvonnasta. Niiden ongelmana on monimutkainen tietokantarakenne, josta haetaan tietoja mahdollisesti moninkertaisilla liitoslauseilla. Ne eivät ole riittävän tehokkaita käytettäväksi verkkosivustolla, jolla käyttöoikeudet pitää hakea tietokannasta jokaisella sivulatauksella. Voidaan arvioida, ettei mikään esitellyistä rakenteista ole kovin hyvä pääsynvalvontamalli Joomla-verkkojulkaisujärjestelmää varten sellaisenaan, vaan tarvitaan mahdollisesti useampi malli rinnakkain.

Hybridimalli, jossa esimerkiksi ryhmäperustaiseen pääsynvalvontaan yhdistetään pääsynvalvontalista tai yksinkertainen pääsymatriisi, voi olla hyvä lähestymistapa, jotta saavutetaan sekä tehokkuus että käyttöoikeuksien hallinnan ymmärrettävyys. Useiden rooliperustaiseen pääsynvalvontaan perustuvien hybridimallien työryhmäominaisuuksien tukea on tutkittu [Tolone et al., 2005; Wang, 1999] ja myös näiden tehokkuutta tulisi tutkia, jotta löydettäisiin verkkoympäristöön hyvin soveltuva standardimalli. Schwartzin [2006a, 2006b] pääsynvalvontalistoihin perustuva malli on tehokas ja se on käytössä todettu toimivaksi, joten hybridimalliin voisi ottaa Schwartzin mallista yksittäisten tietueiden käyttöoikeuksien käsittelyn ja liittää sen esimerkiksi ryhmäperustaiseen pääsynvalvontamalliin, jolla käyttöoikeuksien hallinta yleisellä tasolla on yksinkertaisempaa kuin pääsynvalvontalistoilla toteutetulla mallilla.

5. Yhteenveto

Tässä tutkielmassa on selvitetty, ettei Joomla-verkkojulkaisujärjestelmän nykyistä pääsynvalvontamallia voida tehokkaasti käyttää työryhmäominaisuuksien toteuttamiseen. Tämän lisäksi käytiin läpi rooliperustaisen pääsynvalvon-

nan periaatteet ja selvitettiin vaihtoehtoja, joilla työryhmäominaisuuksien pääsynvalvonta voitaisiin toteuttaa. Eri pääsynvalvontamallien ominaisuuksia vertaamalla todettiin, ettei mikään käsitellyistä malleista sovellu verkkoympäristöön työryhmäominaisuuksien käyttöön sellaisenaan, vaan parempi vaihtoehto olisi jokin monimutkaisen ja yksinkertaisen mallin yhdistelmä. Tällaisten hybridimallien tehokkuutta tulisi erityisesti tutkia erilaisissa ohjelmointiympäristöissä huomioiden myös muistinhallinnan rajoitukset verkkoympäristössä.

Viiteluettelo

[Chen and Crampton, 2008] Liang Chen and Jason Crampton, On spatio-temporal constraints and inheritance in role-based access control. In: *Proc. of the 2008 ACM Symposium on Information, Computer and Communications Security* (March 2008), 205-216.

[Ferraiolo et al., 2001] David F. Ferraiolo, Ravi Sandhu, Seban Gavrila, D. Richard Kuhn and Ramaswamy Chandramouli, Proposed NIST Standard for role-based access control. *ACM Trans. Inf. Syst. Secur.* **4**, 3 (August 2001), 224-274.

[Ferraiolo and Kuhn, 1992] David F. Ferraiolo and D. Richard Kuhn, Role-based access control. In: *Proc. of the NIST-NSA National (USA) Computer Security Conference* (1992), 554-563.

[Fitzpatrick, 2008] Brad Fitzpatrick, Memcached: A distributed memory object caching system. http://www.danga.com/memcached/ (Checked 4.12.2008).

[Georgiadis et al., 2001] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos and Roshan K. Thomas, Flexible team-based access control using contexts. In: *Proc. of the 6th ACM Symposium on Access Control Models and Technologies* (2001), 21-27.

[Hu et al., 2006] Vincent C. Hu, David F. Ferraiolo and Richard D. Kuhn, Assessment of Access Control Systems. *NIST Interagency Report* **7316** (September 2006).

[Joshi et al., 2001] James B. D. Joshi, Walid G. Aref, Arif Ghafoor and Eugene H. Spafford, Security models for web-based applications. *Commun. ACM* **44**, 2 (February 2001), 38-44.

[Kennard, 2007] James Kennard, *Mastering Joomla! 1.5 Extension and Framework Development*. Packt Publishing, 2007.

[Lampson, 1971] Butler Lampson, Protection. In: *5th Princeton Symposium on Information Science and Systems* (1971), 437-443. Reprinted in: *ACM Operat. Syst. Rev.* **8**, 1 (1974), 18-24.

[Osborn et al., 2000] Sylvia Osborn, Ravi Sandhu and Qamar Munawer, Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.* **3**, 2 (May 2000), 85-106.

[PHP Group, 2008] The PHP Group, Alternative PHP Cache. http://www.php.net/apc/ (Checked 4.12.2008).

[Sandhu et al., 1996] Ravi Sandhu, Edward Coyne, Hal Feinstein and Charles Youman, Role-based access control models. *IEEE Computer* **29**, 2 (February 1996), 38-47.

[Schwartz, 2006a] Baron Schwartz, How to build role-based access control in SQL. http://www.xaprb.com/blog/2006/08/16/how-to-build-role-based-access-control-in-sql/ (Checked 30.9.2008).

[Schwartz, 2006b] Baron Schwartz, Role-based access control in SQL, part 2. http://www.xaprb.com/blog/2006/08/18/role-based-access-control-in-sql-part-2/ (Checked 30.9.2008).

[Thomas, 1997] Roshan K. Thomas, Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments. In: *Proc. of the 2nd ACM Workshop on Role-Based Access Control* (1997), 13-19.

[Tolone et al., 2005] William Tolone, Gail-Joon Ahn, Tanusree Pai and Seng-Phil Hong, Access control in collaborative systems. *ACM Comput. Surv.* **37**, 1 (March 2005), 29-41.

[Wang, 1999] Weigang Wang, Team-and-role-based organizational context and access control for cooperative hypermedia environments. In: *Proc. of the 10th ACM Conference on Hypertext and Hypermedia: Returning to Our Diverse Roots* (1999), 37-46.

[Wikipedia, 2008] Wikipedia, Access control list. http://en.wikipedia.org/wiki/Access_control_list/ (Checked 14.11.2008).

Creating 3d worlds with physics

Arttu Tamminen

Conclude

This paper explains how to use Ogre 3D graphics engine with PhysX physics engine.

Keywords: 3d engine, physics

CR Classification: I.3.4

1. Introduction

This paper explains how to use a 3d engine with a physic engine. I have chosen 3d engine to be Ogre 1.6 and physic engine to be Physx 2.8.0. However, NxOgre is used to wrap PhysX functionality easily to use with Ogre. Ogre is an open source graphic engine developed by Ogre Team, which has members around the globe. PhysX is a free physic engine, but still used by major commercial companies. It is developed first by NVIDIA. NxOgre is a wrapper developed by professional programmer, Robin Southern. NxOgre wraps almost all functionality of PhysX for Ogre. NxOgre makes it easy to use PhysX by hiding complicated functionality. NxOgre is in its beta phase, but it is constantly developed. I am using NxOgre 1.0 in this paper. It is the most stable version and seems to also to be last one of this kind of wrapper. In this paper I will ex plain and comment a simple program using Ogre and NxOgre. That program can be used as a framework for simple 3d physical programs or just to demonstrate the power of 3d engine with physics.

In Chapter 2 I tell about tools that I have been using when developing 3D-worlds with physics. In Chapter 3 I tell briefly how to use Ogre3D, explaining only the basic functionality. In Chapter 4 I explain how the example program is constructed from the viewpoint of physics. In Chapter 5 I briefly explain how the physical abilities work.

2. Tools

This chapter contains a short description of tools and libraries used in the sample program. Every tool and library is free and can be downloaded from the Internet. Installing and configuration these tools and libraries are not covered in this paper, because this paper concentrates on physics. Information about installing and configuration can be gathered from the Internet.

2.1. Ogre

Ogre is an open source 3d engine developed by Ogre Team. Ogre is just a plain rendering system, so it doesn't include anything related to physics or sounds, which one might include in the concept of a 3d engine. But it still includes Object Oriented Inputsystem (OIS). OIS is used to make Ogre to communicate with keyboard, mouse and joystick, because a 3d engine system is often for playing games where communications between input devices and graphics drawn to the screen are important. [OgreTeam, 2008] Installing and setting up Ogre for developing 3d graphical programs is quite easy but out of the scope of this paper.

2.2. PhysX

PhysX is a real-time physics engine SDK developed by NVIDIA. The engine is freely available for both Windows and Linux systems, which makes it a good choice to be used with Ogre. PhysX is also used by professional video game developers like Sony. [Wikipedia, 2008] PhysX has a wrapper called NxOgre, which is mostly supported in the Ogre community. A wrapper is an interface which hides the complicated parts of using a tool and makes the use of the tool much easier.

2.3. NxOgre

NxOgre is a physics connector library between Physx engine and Ogre3D. It makes the use of PhysX with Ogre quite easy and it binds Ogre objects together with PhysX objects. [Southern, 2008] As Ogre, NxOgre requires compiling the NxOgre source code and setting up PhysX and few environmental variables. This paper just introduces using NxOgre with Ogre, without any information about installing and setting up NxOgre. Information covering up installing NxOgre is found at NxOgre homepage.

2.4. Framework

There will be example program to demonstrate using Ogre 3D and NxOgre together. It is Microsoft Visual Studio 2005 project and it is available at http://www.cs.uta.fi/~at77902/thesis/PhysicsOgre.zip. I use Visual Studio 2005 purely because Ogre 3D software development kit is made especially for Visual Studio. Of course, source codes can be used with any other development environment. Used programming language is C++, because of its efficiency. But also Ogre is available in few other programming languages too. Used Ogre SDK is version 1.6.0 which is the newest one at the moment of writing this paper. PhysX version is 2.8.1 and NxOgre version is 1.0'21 which is the newest and last version using Ogre. After that NxOgre became independent of Ogre. When

running this framework, mouse is used to turn to camera and buttons W, S, A, D are used to move camera. Spacebar and Q can be used to create more objects. Pressing P-button makes one of the objects to move towards checkpoints and O-button will put that object to sleep. This will be explained in detail in Chapter 4.

2.5. Visual Debugger

PhysX SDK is delivered with Visual Debugger which can be used for debugging physical objects. This program only shows wireframe of physical objects. Applying forces and torque within this program are also possible. Using this program with the framework is complimentary, but heavily recommend, because in this framework there are few object which has no visualization, so they can only been seen within Visual Debugger. Using Visual debugger is useful when all physical objects can been seen. Example project's checkpoints and ball shaped objects can been seen in Figure 1.

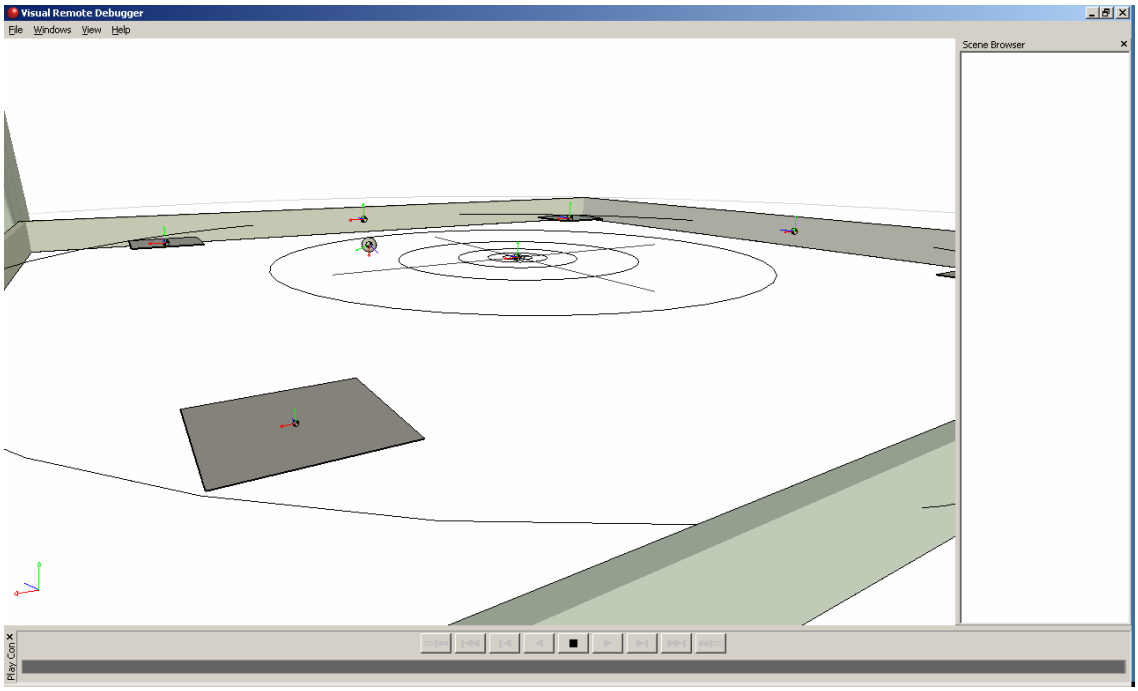


Figure 1. Visual Debugger showing example project

3. Short introduction to Ogre 3D

Because Ogre 3D is object-oriented system, the first thing to do is creating Root. Root is the root class for everything in Ogre 3D. [OgreTeam, 2008a] It takes care of plug-ins, DirectX and OpenGL, configuration files and logging system. After creating the root RenderWindow has to be created. RenderWindow shows rendered objects. It can be window or maximized depending on purpose. DirectX or OpenGL is used to render everything on the RenderWindow. If program is

executed at the moment, only blank window is opened without any content. At this point SceneManager is about to be created. SceneManager is an important part of Ogre 3D, because it controls everything seen on RenderWindow. Every visible object is made and handled by SceneManager. But using NxOgre SceneManager is not that important, because NxOgre takes care of using SceneManager and visible objects. There are also many other managers in Ogre 3D, but to keep the presentation simple, they are omitted in this presentation. [Junkers, 2006]

3.1. Camera and Viewport

The next things to be created are Camera and Viewport for the Camera. SceneManager is used to create Camera. There are few options for Camera which can be altered, depending on what kinds of views are needed. Plain camera is not that useful and therefore ViewPort is needed for the Camera. ViewPort is like a lens for Camera. There are also few options for ViewPort, like color and aspect ratio. After creating ViewPort it has to be attached to the Camera.

3.2. Resources

At this point everything which affects to the view is done. It is time to load resources which are used within the program. This means mainly mesh-files, which are 3d models and their textures, but also different kinds of files can be used. Primary function at this point is to load "resource.cfg". This file contains paths for the meshes and textures. This file can be loaded by using Ogres Configfile-class and then iterating through file and adding meshes and textures to Resourcemanager. Resourcemanager takes care of all meshes and textures.

3.3. Listeners

After reading and iterating meshes and textures, frame listener has to be registered before we can prepare to build up the scene. Frame listener is an Ogre class that will take care of keyboard and joystick. Frame listener has to be inherited from Ogres Frame listener; therefore user's own class must implement virtual functions frameStarted and frameEnded. Capturing keyboard and mouse events in frameStarted and frameEnded functions is possible and even recommended. In the sample program there are few keyboard events and mouse movements captured.

3.4. Light and Skybox

Now it is possible to start creating the scene and physical objects. First, it is unavoidable to create some light for the world, so that everything which is created is also visible. Creating light can be carried out using SceneManagers setAmbi-

entLight, which makes ambient light for world. Shadows are set up with setShadowTechnique method. There are four different kinds of shadow techniques, all of them are for different situations. In this project stencil additive shadow type is used. This shadow type is used because it is simpler than modulative. In this example there are none Entities which are Ogres way to represent mesh-files, but when if one will create Entities this would be propriety time to create them. Because in this paper only physical objects are used, no Entities are covered. After applying light, skybox is also included. Skybox is a huge cube attached around the camera. The cube has textures which represent sky or space, for example. And when camera is moving this cube also moves, creating of illusion of endless sky or space.

4. Structure of the program

This chapter describes the structure of the program. The program is constructed using the object-oriented paradigm. This means that the program is made of objects, which are instanced from classes. The program is written in C++ and without mentioning there is header-file for every source-file, and two header files without source-files. The header-files are not described here, but source-files are. There is Physic-class created, which holds most of the functionality.

4.1. Physic

Physic-class has almost all functionality of the program. It is created in the main source-file with default constructor and after that different values, variables and objects are passed with various methods. The constructor creates the instance of Root-class, which is the foundation of Ogre. It also shows the dialogue to the user to select DirectX or OpenGL. SceneManager and RenderWindow are also made within the constructor. After the construction, the camera has to be made with method createCamera. This method takes three parameters: the name, position and direction of the camera.

After the camera is made, the viewport of the previously made camera has to be constructed. Physic-object has methods createViewport what makes viewport for the camera. This method only takes background colour as parameter. This colour is usually just plain black.

Next Physic-object has setupResources method that reads file “resources.cfg”. This file contains information about where all the resources are. In this program it is used original “resources.cfg”-file from Ogre, but added few new resources.

Then the frame listeners are registered. They make interface between Ogre and human-computer interaction devices, such keyboard and mouse. The frame listeners will be explained in detail further.

CreateScene method creates only two objects. These are skybox and plane. Plane is as its name says plane which has texture on it and nothing else. In this program the plane serves as floor. Skybox is added to make this little more realistic. Since these two things do not exist in physical world, the physical objects cannot collide with these, expect for the plane which has exactly the same position as floor, which is made in the next method. Physical objects do not collide with the plane, although it seems like that does happen. In this method one should make all objects which don't have to interact with other objects. Usually in games, say, grass is this kind of object.

Then next method used is createPhysics which creates all physical objects. But in the beginning of the method the physical world is created. This is where NxOgre shows its usefulness: NxOgre hides the complicated usage of PhysX SDK, creating physical world needs one method only. This is done by calling the constructor of World-class. It takes one parameter, which is a normal string. String contains options, and in this case it has only one option which is "time-controller: ogre". This means that the user does not have to take care of timing. At this point an empty physical world exists. The world itself cannot contain any physical objects, but in the world there can be scenes, which contain objects [Southern, 2008a]. Creating the scene is also quite straightforward. It takes three parameters, the first one is the name for the scene, the second one is the world where the scene will exist and last one is a string which has options. In this case options are related to gravity, floor and renderer. Gravitation can be assigned with setGravity method which takes a vector as parameter. This vector describes direction and force. Using floor means that objects cannot have position where y is lower than zero. Setting "renderer: ogre" makes Ogre to render every physical object that has visualization. It is possible to make more than one scene, but they cannot interact with each other.

4.1.1. ActorGroups

ActorGroups are used to check collisions between different objects. The collision detection can be used for different kind of things, for example, does an object leave a certain scene or just take some action when a collision happens. In this case three groups are made, "sphereGroup", "checkPointGroup" and "wallGroup". Each of these groups consists of one or more objects. SphereGroup has only one ball shaped object which the user can control. "CheckpointGroup" has four checkpoints in array, what ball shaped object should touch in order. And wallGroup has four walls and when a ball shaped object

hits wall, it will be transported back to the last know checkpoint. For every group collision detector-class has to be registered, on every collision certain methods implemented in collision detector are used. This will be explained in greater detail in Section 4.3.

4.1.2. Convex meshes and triangle meshes

In the running example only one convex mesh is used, so it is loaded with addMeshas method. Also all triangle meshes are loaded in the same fashion. Convex meshes and triangle meshes are both an efficient way to presentin 3d objects, as seen in Figure 2. Convex meshes consist of simple convex polygons and triangle meshes consist of triangle polygons.[Wikipedia, 2008c] If used triangle meshes, they have to be static. Convex meshes can be used also static and dynamic. But using triangle meshes and convex meshes require much more computing power than using primitives.

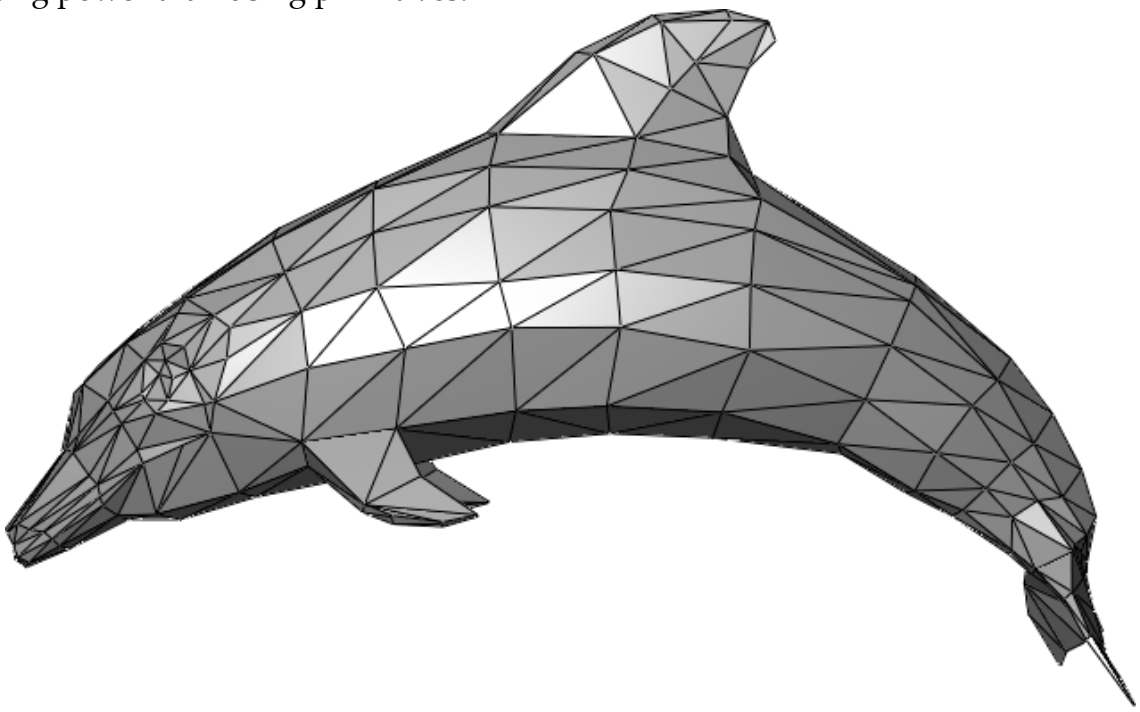


Figure 1. Dolphin represented as triangle mesh

In NxOgre convex meshes and triangle meshes can be used to represent some model more exactly than using just primitive shapes. Convex shapes and triangle shapes can be converted from normal Ogre model files by using application named Flour. [Southern, 2008b] Using Flour is not cover in this paper.

In NxOgre objects can be made using createActor or createBody methods. CreateActor creates physical object without any visualization in Ogre and createBody creates physical object with visualization in Ogre. Both of these have their own purposes. In our example actors are used for invisible walls and checkpoints and bodies for everything else. CreateActor methods parameters

are the name and the minimum bounding rectangle. These rectangles dimensions are the same as the minimum rectangle that can be fitted over the model. [Wikipedia, 2008d] In our example every dimensions of the model are already known or only convex shapes are used. Because using convex shape the user does not have to worry about the size of model. The normal way of using this minimum bounding rectangle is to create normal Ogre node of the desired model, to get information from that node and then to delete the useless node. That method is especially useful when reading information about the objects, for example, from XML-file.

At this point physic-class has done almost everything which was needed. Now it has to run the main loop where Ogre takes care of rendering everything visible in screen.

4.2. OgrePhysicFrameListener

OgrePhysicFrameListener is a listener class which listens keyboard, mouse and joystick. So it keeps track of every single action done with those peripherals. In this program only keyboard and mouse are considered. In the constructor of OgrePhysicFrameListener it is told which windows the listener should listen. In the sample program there is only one window which needs to be listened. After that InputManager has to be made, it takes care of two classes which are mouse listener and keyboard listener. It is also important to tell mouse listener sizes of listened window. After that instance of mouse listener and keyboard listener has to register to OgrePhysicFrameListener. The same kind of procedure occurs also when using collision detectors.

Because OgrePhysicFrameListener is inherited from Ogres own frame listener, there will be two methods that can be used to catch signals from keyboard and mouse. These two methods are frameStarted and frameEnded, as you can tell from the name first one occurs when Ogre is starting to renderFrame and second one after rendering has been done. Therefore mouse and keyboard are captured and checking which buttons have been pressed will physic-class generate more objects for example.

4.3. CollisionDetector

Collision detector is used to detect collisions between two dynamic objects. It has three types of collisions: when a collision is started, when a collision is ended and when a collision is occurring. In every of these one can check what objects are causing the collision.

Many dynamic applications are not concerned with the forces that occur when bodies collide or touch. On the other hand, the most interesting applications are concerned with collision detection – for the cannon ball to bounce on

the ground or an articulated body to fall down stairs, the SDK must be provided with the shapes of the bodies that can touch, and also the properties of the surfaces (e.g., how slippery they are). This collision detection functionality can range from very simple (the cannon ball is simply dependent on whether the ball is above or below ground level) to quite complex (in the case of the articulated body falling down the stairs). A body falling down the stairs is more complex because it involves many intersecting shapes. For example, the SDK may need to collide a convex against a mesh, and must resolve the system in a way to maintain the constraints within the body. [AGEIA, 2007]

In our program collision detection is used to make on object to go through every checkpoint in order. This is done by creating an array which holds positions for four checkpoints. At the beginning the first element is taken from the array. Then using Body's moveTowards method the object moves towards the first checkpoint, which is the recently taken first element. The object reaches this position and collides with the object. After that collision detector's onStartTouch method is occurring. That makes sure that the new checkpoint is not taken from the array, if the object is colliding with a wrong object. Otherwise, the next element is taken from the array. According to this information a new checkpoint will be taken from the array. And again moveTowards method can be used; this will go on forever, because a new checkpoint will be assigned every time an object is hitting the correct checkpoint. Also in the collision detector is checked if the object is colliding with walls. When object is hitting a wall, it will be moved back to last know checkpoint.

5. Physical explanations

In the physical engine, an object can be a primitive, for example, a cube or a sphere. An object can also be compound, i.e., constructed from primitives. If the model is more complex, then triangle and convex shapes can be used. Force can be added in every object in the physical world. This force can move the object. Adding constantly more force makes an object to have acceleration. [Beer & Johnston Jr, 1990] Also adding torque is possible. This is done by using addTorque method. Adding torque does add only angular speed for the object, which does not make it move, but spin. Also in this case adding more force constantly makes spinning faster and faster, therefore acceleration works also in this case. Adding these two forces together rolling motion can be acquired [Beer & Johnston Jr, 1990]. This force is used with sphere object which can be controlled to move towards checkpoints.

Also gravity is affecting all dynamical objects, which is pushing them towards floor or ceiling, depending on how gravity is defined. Every object also has friction which affects speed of objects.

6. Conclusion

Using 3d engine with physic engine is quite easy with tools Ogre3D, PhysX and NxOgre. And also it is really creative. There are an infinite amount of applications where physics and 3d graphics could be used.

The physical engine provides us with plenty of more features, like using different materials for objects. This would cause different type frictions and elasticity Also compound object can be used to make much more complicated objects. The engine also offers wheels for making vehicles and pulleys and joints. Also PhysX offers using clothes and soft bodies. There are many features beyond these which are not covered in this paper, because that would take so much time and it would be huge research to make.

References

[AGEIA, 2007] AGEIA Technologies Inc, AGEIA PhysX SDK 2.7, this help file is released with PhysX SDK only.

[Beer & Johnston Jr, 1990] Ferdinand P. Beer & E. Russel Johnston Jr, *Vector Mechanic for Engineers,* McGraw-Hill, 1990.

[Junker, 2006] Gregory Junker, *Pro Ogre 3D Programming*, Apress, 2006.

[OgreTeam, 2008] Ogre3D Manual, http://www.ogre3d.org/docs/manual/. Checked 5.11.2008.

[OgreTeam, 2008a] Ogre3D API, http://www.ogre3d.org/docs/api/html/. Checked 5.11.2008.

[Southern, 2008] Robin Southern, The official FAQ of NxOgre, http://www.ogre3d.org/phpBB2addons/viewtopic.php?t=6454&sid=7a55dafb16d963a9015a70312e28da3b, Checked 5.11.2008.

[Southern, 2008a] Robin Southern, Short Guide to NxOgre, http://www.nxogre.org/shortguide/. Checked 16.12.2008.

[Southern, 2008b] Robin Southern, Flour, http://get.nxogre.org/flour/. Checked 16.12.2008.

[Wikipedia, 2008] Wikipedia, PhysX, http://en.wikipedia.org/wiki/PhysX. Checked 5.11.2008.

[Wikipedia, 2008a] Wikipedia, Object-Oriented Programming, http://en.wikipedia.org/wiki/Object-oriented_programming. Checked 16.12.2008.

[Wikipedia, 2008b] Wikipedia, Triangle Mesh, http://en.wikipedia.org/wiki/Triangle_mesh. Checked 16.12.2008.

[Wikipedia, 2008c] Polygon Mesh, http://en.wikipedia.org/wiki/Polygon_ mesh. Checked 16.12.2008.

[Wikipedia, 2008d] Wikipedia, Minimum Bounding Rectangle, http://en.wikipedia.org/wiki/Minimum_bounding_rectangle. Checked 16.12.2008.

.

Suunnittelumalleista ja niiden soveltamisesta pelisuunnittelussa

Minna Viitanen

Tiivistelmä.
Tämän tutkielman tarkoituksena on selittää yleisellä tasolla, mitä suunnittelumalleilla tarkoitetaan olio-ohjelmoinnin yhteydessä ohjelmistotuotannossa. Erityisesti tutkielmassa perehdytään suunnittelumallien käyttöön peliohjelmoinnissa ja esitellään muutamia aiheesta tehtyjä ja meneillään olevia tutkimuksia. Lopuksi keskitytään vielä hieman suunnittelumallien käytöstä saatujen hyötyjen ja vastaavasti mallien soveltamisen ongelmakohtien tutkailuun.

Avainsanat ja -sanonnat: suunnittelumalli, pelisuunnittelumalli, olio-ohjelmointi, ohjelmistosuunnittelu, pelisuunnittelu.
CR-luokat: D.1.5, D.2.11, K.8.0

1. Johdanto

Tässä tutkielmassa on tarkoitus selvittää, mitä suunnittelumallit ovat, miten niitä sovelletaan ja erityisesti sitä, miten niitä voidaan käyttää pelien ohjelmistosuunnittelussa hyödyksi. Tämä tutkielma on lähinnä alan kirjallisuuteen ja aihepiirin viimeaikaisiin artikkeleihin perustuva katsaus suunnittelumalleihin peliohjelmoinnissa.

Tutkielman luvussa 2 esitellään suunnittelumallien käsitteen merkityksiä ja selvitetään, miten suunnittelumalleja yleisesti voidaan kuvata ohjelmistosuunnittelun keskeisissä dokumentaatioissa. Luvussa 3 pohditaan tarkemmin pelisuunnittelumalleja ja tutustutaan pelisuunnittelumalleihin yleisellä tasolla määrittelemällä käsitteitä ja yleisiä toimintatapoja. Luvun 3 lopussa syvennytään myös enemmän pelisuunnittelun tutkimukseen, erityisesti sovellettujen suunnittelumallien osalta. Luvussa 4 keskitytään suunnittelumallien hyötyjen arviointiin sekä mahdollisten haittojen havaitsemiseen. Luku käsittelee suunnittelumallien soveltamisesta suunnittelussa johtuvia ongelmakohtia ja antaa myös toisaalta argumentteja suunnittelumallien käytön puolesta.

2. Mitä ovat suunnittelumallit ja miten ne esitetään

Suunnittelumallit, joista saatetaan käyttää myös joissain tapauksissa nimitystä ohjelmistomallit, pyrkivät kertomaan yleisiä, geneerisiä, linjoja siitä, minkälainen ratkaisu parhaiten sopii käsillä olevaan ongelmaan [Rintala ja Jokinen, 2001]. Suunnittelumallit ovat yksinkertaisesti sanottuna yleisiä ratkaisumalleja usein esiintyviin, yleisiin ohjelmistosuunnittelun olio-ohjelmointiin liittyviin

suunnitteluongelmiin. Tarkemmin voidaan määritellä suunnittelumallit hyväksi todettujen ratkaisujen dokumentoiduiksi kuvauksiksi tietyistä ohjelmistosuunnittelun ongelmista, jotka koostuvat ongelman kuvailusta sekä sen yleismuotoisesta ratkaisusta.

Suunnittelumallit eivät koskaan ole uusia keksintöjä, ideoita tai tarkoituksenmukaisesti alusta lähtien tiettyä tarkoitusta varten suunniteltuja malleja, vaan ne ovat tunnettuja ja testattuja ratkaisuja, jotka on ”löydetty” olemassa olevista ohjelmistoista eri menetelmin. Rintala ja Jokinen [2001] määrittelevät suunnittelumallit olio-ohjelmoinnin uudelleenkäytön säännöstöiksi, jotka kertovat, miten jokin yleisempi ongelma voidaan ratkaista jonkin tietyn oliojoukon avulla.

Suunnittelumallien johtava idea perustuu havaintoon, jonka mukaan samankaltaiset ratkaisut luokkarakenteen ja luokkien yhteistyön suunnittelussa näyttävät toistuvan [Haikala ja Märijärvi, 2002]. Suunnittelumalli identifioi siihen osallistuvat luokat ja niiden ilmentymät keskittyen johonkin tiettyyn ohjelmistotuotannon suunnitteluongelmaan [Gamma et al., 1995]. Vaikka suunnittelumalli ei ole koskaan riippuvainen mistään tietystä ohjelmointikielestä, mallin kuvauksessa voidaan kuitenkin antaa havainnollistavia esimerkkejä jonkin ohjelmakoodin avulla. Usein esimerkiksi C++-ohjelmointikieltä on käytetty esimerkkikoodin esittämiseen. Koodin uudelleenkäyttö ei kuitenkaan ole mallien varsinainen tarkoitus, vaan kyse on enemmänkin luokka- ja oliorakenteen uudelleenkäytöstä. Luokkien toteutus ei, ainakaan välttämättä, ole osa mallia [Haikala ja Märijärvi, 2002].

Suunnittelumallien soveltamisalue vaihtelee. Tavallisesti suunnittelumallit ovat sovellusalueesta riippumattomia, mutta mikään ei estä antamasta myös tiettyyn alueeseen liittyviä malleja. Toisaalta myös suunnittelumallin soveltamistaso sovelluksessa voi vaihdella. Monet suunnittelumallit liittyvät arkkitehtuuritason ratkaisuihin, kun toiset taas voivat koskea yksityiskohtaisen tason ratkaisuja. Samaakin suunnittelumallia voidaan soveltaa useilla eri tasoilla. [Koskimies, 2000]

2.1. Suunnittelumallien idea ja ominaispiirteet

Suunnittelumallit ovat tuttuja ja tärkeitä työvälineitä ohjelmistosuunnittelijoille. Mallit mahdollistavat tietämyksen ja suunnittelukokemuksen siirtämisen kokeneemmilta suunnittelijoilta uusille helpolla tavalla. Osaaminen siis säilyy näin, vaikka projektien henkilöt vaihtuisivatkin. Hyvin nimetyt ja dokumentoidut mallit muodostavat koko ajan uutta käsitteistöä suunnittelijoille. Toisin sanoen, suunnittelumallit antavat suunnittelijoille yhteisen kielen, jolla suunnittelusta voidaan keskustella korkeammalla abstraktiotasolla kuin esimerkiksi pelkästään luokkien tasolla [Haikala ja Märijärvi, 2002]. On tärkeää, että doku-

mentaatio on esitetty mahdollisimman selkeässä muodossa ja että suunnittelumalleilla on yleisesti tunnetut, kuvaavat nimet [Koskimies, 2000].

On kuitenkin olemassa myös tilanteita, jolloin suunnittelumalleja ei välttämättä kannata käyttää. Mallit eivät sovi aina jokaiseen ohjelmistoprojektiin, tai projektin osaan. Suunnittelumallien soveltamisen tulee aina lähteä tietyn ongelman identifioimisesta [Koskimies, 2000]. Johtuen siitä, että mallit ovat dokumentoituja ratkaisuja tiettyihin ohjelmistoteknisiin ongelmiin, voi olla hyvinkin mahdollista, ettei yksikään olemassa oleva malli ole soveltuva juuri kyseisen ohjelmiston apuvälineeksi.

Vaikka alun perin suunnittelumallit kehitettiin täysin toista alaa varten, nykyisin suunnittelumallien käsite on varsin keskeinen osa myös olio-ohjelmointiin liittyvää ohjelmistokehitystä. Alkuperäinen tarkoitus suunnittelumalleille liittyi talonrakennuksen arkkitehtonisiin ratkaisuihin. Näin pohjaksi otettu ohjelmointiparadigma voi vaikuttaa siihen, mitkä ratkaisut on järkevää esittää suunnittelumalleina [Koskimies ja Mikkonen, 2005].

Suunnittelumalleja ei ole tarkoitettu koko ohjelmiston kuvailuun, mutta silti niillä voidaan halutessaan parhaassa tapauksessa kuvata suurikin osa tietystä ohjelmistosta. Yksi suunnittelumalli koskee aina useita ohjelman osia, kuten komponentteja, rajapintoja ja luokkia, jotka on järjestetty kyseiseen ratkaisuun jollain tietyllä tavalla. Suunnittelumallin käyttö ei kuitenkaan varsinaisesti tuo järjestelmään uutta toiminnallisuutta, vaan se parantaa ohjelmiston jotain laatuominaisuutta, kuten esimerkiksi sen muunneltavuutta, ylläpidettävyyttä tai uudelleenkäytettävyyttä [Koskimies ja Mikkonen, 2005].

Suunnittelumalleilla määritellään rajapintoja tunnistamalla niiden elementtejä ja niiden kautta siirrettävää dataa. Suunnittelumallissa voidaan myös ilmaista, mitä rajapintaan ei tule sisällyttää. Mallit määrittelevät pääsääntöisesti myös erilaisia suhteita rajapintojen välille. [Gamma et al., 1995]

Suunnittelumalleilla voidaan saavuttaa joustavuutta, ylläpidettävyyttä ja muunneltavuutta, mutta myös päinvastoin voidaan monimutkaistaa suunnitelmaa ja syödä ohjelmistolta sen suorituskykyä. Suunnittelumallin soveltaminen on perusteltua silloin kun erityisesti halutaan joustavuutta ja uudelleenkäytettävyyttä. Ylläpidettävyyden kanssa voi tulla vaikeuksia sovelluksen kasvaessa ja riippuvuuksien monimutkaistuessa. Tällöin ei usein uskalleta muuttaa sovelluksesta mitään, kun pelätään, että tullaan samalla rikkoneeksi jotain muuta koodissa. Selkeillä rajapinnoilla voidaan erottaa objektien toteutus ja vuorovaikutus muihin objekteihin nähden toisistaan, jolloin toteutuksesta tulee itsenäinen muusta sovelluksesta. Suunnittelumalleista on tässä tilanteessa erityisesti hyötyä.

Suunnittelumallien avulla voidaan ratkaista laajennettavuusongelmia. Luokan laajentaminen eristyksissä on helpompaa, jos luokka ei riipu monista muista luokista [Gamma et al., 1995]. Jo suunnittelumallien arviointi pakottaa ohjelmistokehittäjät miettimään ja pohtimaan yhdessä suunnitelmaansa, usein myös samalla tuoden esiin prosessin ongelmakohdat [Crawford and Kaplan, 2003].

Suunnittelumallit tukevat skaalautuvuutta ja voivat parantaa järjestelmien luotettavuutta. Suunnittelumallit voidaan validoida korkealla tasolla ja liittää ja varhaisessa vaiheessa suunnitteluprosessiin. Viimeistellyn ohjelmiston toimitus loppukäyttäjille on myös ohjelmistokehityksen tavoitteena projekteilla. Tähän vaiheeseen suunnittelumallit eivät kuitenkaan välttämättä tuo niin paljoa lisäarvoa kuin edellä mainittuihin muihin osa-alueisiin. [Crawford and Kaplan, 2003]

2.2. Suunnittelumallien kuvaus

Suunnittelumalli kuvataan systemaattisena, rakenteisena dokumenttina, jossa on aina tietyt osat. Dokumentin tulisi periaatteessa antaa kaikki se informaatio, jonka mallin soveltaja tarvitsee [Koskimies ja Mikkonen, 2005]. Dokumentin olennaisimpiin osiin kuuluvat muiden muassa mallin nimi, suunnitteluongelman kuvaus, ratkaisun esittely ja sen rakenne sekä mallin soveltamisen edut, haitat ja mahdolliset sovellusesimerkit. Esimerkki suunnittelumallista voi olla vaikkapa ”miten esitetään osista rekursiivisesti muodostuva olio siten, että kaikki oliot voidaan käsitellä samalla tavalla” [Koskimies, 2000].

Mallin nimi identifioi mallin, helpottaa suunnittelijoiden välistä kommunikointia mallista keskusteltaessa ja kasvattaa suunnittelusanastoa. On oleellista, että nimi on mahdollisimman kuvaileva, mutta kuitenkin kohtuullisen lyhyt, jotta se voidaan muistaa helposti. Nimen on oltava yksiselitteinen, eikä yhdellä mallilla tulisi olla useita nimiä. Tämä ei kuitenkaan täysin aina toteudu.

Kuvailtava ongelma on yleinen suunnitteluongelma, joka esiintyy toistuvasti useassa järjestelmässä. Suunnittelumallin ratkaisema ongelma pyritään esittämään siten, että se kattaa kaikki mahdolliset mallin sovellukset [Haikala ja Märijärvi, 2002]. Ongelman kuvauksen yhteydessä kerrotaan yleensä myös, milloin suunnittelumallia voidaan soveltaa tai milloin sen soveltamista tulisi välttää. Ongelmakuvaus voi sisältää listan olosuhteista, joiden tulee täyttyä ennen kuin mallia on järkevää soveltaa. Ongelmaa ja tällaista niin kutsuttua ”ongelmayhteyttä” voidaan havainnollistaa erilaisin esimerkein.

Mallin kuvauksen tärkein osa, ratkaisu, esitetään myös mahdollisimman yleisessä muodossa ja se kuvaillaan yleensä sanallisesti ja esimerkiksi antamalla ratkaisulle rakenne vaikkapa UML-kaaviota tai pseudokoodia apuna käyttäen. Kaavio havainnollistaa muun muassa malliin osallistuvien olioiden keskinäistä kommunikointia. Ratkaisun tulee periaatteessa olla riippumaton ohjelmointi-

kielestä. Lopuksi mallin kuvauksessa voidaan vielä esitellä soveltamisen etuja ja todettuja haittoja, eli soveltamisen seurauksia, jotta suunnittelijat voivat arvioida, kannattaako kyseessä oleva malli ottaa huomioon omassa projektissa. Mainittavia seurauksia ovat sellaiset, jotka liittyvät merkittävästi järjestelmän joustavuuteen, laajennettavuuteen tai siirreltävyyteen [Gamma et al., 1995].

Usein mallin yhteydessä esitetään myös esimerkkikoodia toteutuksen havainnollistamiseksi. Toteutusta voidaan havainnollistaa myös vaihtoehtoisesti, tai koodin lisäksi, antamalla esimerkkejä suunnittelumallin aikaisemmasta soveltamisesta todellisissa järjestelmissä. Joskus on tapana myös esitellä ”sukulaismalleja”, eli malleja, jotka ovat jollain tapaa samankaltaisia kuin dokumentissa esitelty.

Gamman ja kumppaneiden [1995] esittelemässä sapluunassa suunnittelumallien kuvaamiseksi mainitaan seuraavat kohdat, jotka tulisi esittää dokumentissa: mallin nimi ja luokittelu, tarkoitus, muut nimet, motivointi, sovellettavuus, rakenne, osallistujat (eli luokat ja objektit), vuorovaikutus, seuraukset, toteutus, esimerkkikoodi, tunnetut käyttökohteet ja sukulaismallit. Monet suunnittelumallien kuvaukset, varsinkin pelisuunnittelumallien tapauksessa, jättävät osan näistä kohdista pois tarpeettomina.

3. Suunnittelumalleista pelisuunnittelussa

Suunnittelumallit ovat käsitteenä yleisesti ajateltu ongelmanratkaisutyövälineiksi. Pelisuunnittelussa tämä ei ehkä ole paras mahdollinen määritelmä suunnittelumalleille. Pelien yhteydessä mallit eivät nimittäin aina välttämättä keskity ongelman ja sen ratkaisun esittämiseen sellaisenaan. Björk ja Holopainen [2004] uskovat, että pelisuunnittelumallit tarjoavat kuitenkin hyvän pohjan sille, miten voidaan jäsentää sitä tietämystä pelattavuudesta, jota voidaan sitten käyttää hyödyksi sekä pelien suunnittelussa että analysoinnissa. He määrittelevät pelisuunnittelumallit näin: ne ovat puolimuodollisia, toisistaan riippuvaisia kuvauksia yleisesti ja toistuvasti ilmenevistä pelisuunnitelman osista, jotka koskevat pelattavuutta. Puolimuodollisuudella tarkoitetaan tässä, että malleilla on olemassa rakenne ja että ne voidaan erottaa toisistaan ja myös, että on mahdollista tunnistaa suhteita pelisuunnittelumallien ja yleisen pelisuunnittelun välillä.

Pelien suunnittelumallit liittyvät aina jollain lailla toisiinsa. Voidaan jopa sanoa, että ne ovat sukua toisilleen. Jotkin suhteet ovat yleisempiä kuin toiset. Yhden pelin sisältämillä malleilla voi olla myös useita erilaisia suhteita keskenään. [Björk and Holopainen, 2004]

Suunnittelumallit tunnistetaan peleistä joko tarkoituksellisen etsinnän tuloksena tai ne voidaan myös löytää ”vahingossa”. Pelisuunnittelumalleja on

vaikea suunnitella täsmällisesti, sillä ne ovat useimmiten varsin korkean tason malleja [Björk and Holopainen, 2004].

3.1. Pelisuunnittelumallien soveltaminen

Pelisuunnittelumallien yhteinen käsitejoukko tarjoaa arvokasta tukea menetelmien muokkaamiseen. Mallien käyttö voidaan monissa tapauksissa yhdistää ja räätälöidä tiettyihin tosimaailmaan käyttötarkoituksiin. Pelisuunnittelumallien toteutus voidaan jakaa karkeasti kahteen kategoriaan: analyysiin ja suunnitteluun. Analyysi vaatii olemassa olevan pelin, prototyypin tai pelin kuvailevan suunnitteludokumentin, jotta voidaan tutkia, mitä suunnittelumalleja pelissä esiintyy. Suunnittelu voi viitata peli idean, käsitteen tai kuvauksen luomiseen käyttämällä pelisuunnittelumalleja tai formalisoimalla peli-idean tai pelin käsitteen rakenteisemmaksi kuvaukseksi. [Björk and Holopainen, 2004]

Pelisuunnittelumallien analyysi pitää sisällään pelisuunnittelumallit peleissä, peliprototyypit tai suunnitteludokumentit. Pelisuunnittelumallien löytäminen peleistä voidaan tehdä testaamalla pelien pelattavuutta, joko tekemällä se itse tai tarkkailemalla muita testaajia. Suunnitteludokumenttien kanssa tämä ei ole mahdollista, mutta pelien kuvauksia analysoimalla on mahdollista tehdä rakenneanalyysiä pelisuunnittelumallien tunnistamiseksi pelissä. Rakenneanalyysi tehdään suunnitteludokumenttien lisäksi myös staattisista pelikuvauksista ja prototyypeistä, kuten ohjekirjoista tai koodista. Tämä joustavuus sallii pelitestaamisen ja rakenneanalyysin yhdistämisen, jolloin voidaan suorittaa tehokkaampia ja luotettavampia tarkasteluja. [Björk and Holopainen, 2004]

Mallien toteutus pelisuunnittelussa sallii suunnittelijoiden hallita pelissä esiintyviä seikkoja. Pelisuunnittelumallit ovat hyödyllisiä useille potentiaalisille käyttäjäryhmille, joilla on perustavanlaatuisesti erilaiset työskentelytavat. Malleja käyttämällä näiden ryhmien välinen kommunikointi myös helpottuu. Voidaan toki väittää, että mallien käyttö vähentää pelisuunnittelun luovuutta tai että mallien käyttö johtaa tilanteisiin, joissa kaikki pelit noudattavat samaa kaavaa ja muodostuvat stereotyyppisiksi. Soveliaampi näkemys mallien käytöstä on verrata sitä esimerkiksi taiteellisiin suoritteisiin: taiteilijoilla on parempi mahdollisuus luoda jotain uudenlaista, kun muiden taiteilijoiden työ on heille tuttua. [Björk and Holopainen, 2004]

Pelisuunnittelumallit tukevat suunnittelua auttamalla luomaan ideoita, kehittämään rakenteisesti pelikäsitteitä, kehittämään suunnitteluryhmien jäsenten välistä kommunikointia ja ratkaisemaan pelaamisen suunnitteluongelmia. Kunhan pelikonseptit ovat selvillä, pelisuunnittelumalleja voidaan käyttää kehittämään ja rakenteistamaan sitä. Pelisuunnittelumallit sisältävät tietoa siitä, miten peleissä on saavutettu tietyn tyyppinen pelattavuuden vuorovaikutus, jotta sit-

ten voidaan esimerkiksi poistaa pelistä ei-haluttua tai ei-tarkoitettua vuorovaikutusta. [Björk and Holopainen, 2004]

Pelisuunnittelumallit ovat suhteellisen uusi työkalu tietokonepelien analysoimiseksi. Björk ja Holopainen [2004] kokosivat ja luokittelivat 296 pelisuunnittelumallia, niiden vuorovaikutussuhteet ja sovellusmahdollisuuksia. Nämä antavat melko kattavan kehyksen pelaamisen elementtien ja vuorovaikutuksen analyysille ja yleisen sanaston pelin suunnitteluelementtien ja käsitteiden kuvailuun. Eräs Björkin ja Holopaisen ehdottama käyttötarkoitus on tietokonepelikokoelmien kategorisointi niiden suunnittelumallien yhtäläisyyksien ja eroavaisuuksien mukaan, jotta voitaisiin tunnistaa tai ymmärtää genrejä. [Loh and Soon, 2006]

3.2. Pelisuunnittelumallien tutkimuksesta

Tässä kohdassa esittelen joitain pelisuunnittelumallien ominaisuuksia ja kerron niiden soveltamisesta eri tilanteissa. 2000-luvun puolella on tehty ja käynnissä useita tutkimuksia pelien yhteyteen liitetyistä suunnittelumalleista. Tässä perehdymme hieman näistä tutkimuksista irrotettuihin oleellisiin ideoihin ja ajatuksiin.

3.2.1. Käyttäytymisen liittäminen verkkopelin virtuaalimaailmaan

Pellensin ja muiden [2008] tutkimus on liittynyt käyttäytymisen lisäämiseen X3D/VRML-virtuaalimaailmoihin ja sen haasteisiin. He pohtivat mm. työkalutuettua suunnittelun lähestymistapaa käyttäytymisen luomiseksi objekteille verkon X3D-maailmoissa. Tämä lähestymistapa pohjautuu niin kutsuttuihin generatiivisiin suunnittelumalleihin sekä graafiseen merkintätapaan, joiden avulla käyttäytyminen voidaan määritellä luomalla malleista ilmentymiä ja muokkaamalla niitä asiaankuuluvasti. Eri malleja yhdistelemällä voidaan luoda monimutkaisempaa käyttäytymistä. Tämän lähestymistavan avulla kokeneet suunnittelijat voivat määritellä uusia käyttäytymismalleja ja tuoda ne graafisena esityksenä myös muiden saataville täysin käyttövalmiina.

Vaikka Pellens ja kumppanit myöntävät käsittävänsä hyvin, että monimutkaisen käyttäytymisen toteuttamisessa tulee olemaan vaikeaa välttää skriptaamista kokonaan, heidän esittämänsä uusi lähestymistapa pyrkii minimoimaan sen mahdollisimman suuressa määrin. Lähestymistapa tukee virtuaaliympäristöjen kehitystä lisäämällä kehitysprosessiin erityisen käsitteellisen mallintamisen vaiheen. Käsitteellisellä mallintamisella tarkoitetaan sitä toimintaa, jolla voidaan luoda teknologiasta riippumattomia rakennusmalleja järjestelmille. Mallinnuksen aikana ei tarvitse miettiä yksityiskohtia, joten virtuaalitodellisuuksista ei tarvita mitään taustatietoa mallien luomiseksi.

Pellens ja muut ovatkin siis laajentaneet esittämäänsä lähestymistapaa käsittämään myös suunnittelumallien merkintätavan, jotta voitaisiin saada kokeneilta suunnittelijoilta tietämystä ja helpottaa käyttäytymismääritysten uudelleenkäyttöä. Tästä seuraa, että vähemmän asiantuntevat voivat mallintaa käyttäytymistä korkealla abstraktiotasolla joutumatta mallintamaan kaikkia alemman tason käyttäytymiselementtejä. Perinteisestihän suunnittelumallit kuvataan tekstidokumentteina ja käytetään valmiita pohjia, jolloin soveltajan on käännettävä kuvaus toteutukseksi, mikä voi viedä aikaa ja olla virhealtista. Ratkaisuksi on ehdotettu generatiivisten suunnittelumallien käsitettä. Ne ovat muuten samankaltaisia kuin säännölliset suunnittelumallit, paitsi että ne poistavat mallien toteuttamisen taakan käyttäjältä. Sittemmin tämä käsite on laajennettu visuaalisgeneratiivisiksi suunnittelumalleiksi ilmaisemalla ne käyttäen graafista merkintätapaa ja integroimalla ne olemassa olevaan graafiseen käyttäytymisen, animaation, mallinnuskieleen.

Pellens ja muut tekevät eron kahden oleellisen mallikategorian välille: käyttäytymis-/vuorovaikutusmallit ja rakennemallit. Käyttäytymismallit koskevat todellisia toimintoja (algoritmeja), joita objekti voi suorittaa virtuaaliympäristössä. Vuorovaikutusmallit koskevat todellisen käyttäjän ja objektien vuorovaikutusta ja objektien välistä vuorovaikutusta virtuaaliympäristössä. Rakennemalli taas on tavallista mallia yhtä abstraktiotasoa korkeampi, vuorovaikutus- tai käyttäytymismalli ja koskee sen määrittelyä, miten useita malleja voidaan koota muodostamaan laajemman rakenteen.

Pellensin ja muiden esityksessä suunnittelumalli esitetään useina eri tavoin yhdisteltävinä graafisina elementteinä. Elementit edustavat mallille oleellisia komponentteja, siihen liittyviä objekteja ja niiden käyttäytymisiä. Todellisen käyttäytymisen toteuttavat algoritmit piilotetaan mallin käyttäjältä. Ne annetaan erikseen kokeneempien kehittäjien laatimassa mallikirjastossa. Tätä lähestymistapaa, joka mahdollistaa mallien käyttämisen käyttäytymisen mallintamisessa, kutsutaan nimellä CoDePA, Conceptual Design Pattern Approach, eli vapaasti käännettynä, käsitteellinen suunnittelumallien lähestymistapa. Mallit ilmaistaan niin, että niitä voidaan soveltaa useissa eri tilanteissa. Käyttäytymismallin käyttöönottoprosessi koostuu mallin valinnasta ja sen muokkaamisesta. Kun malli on valittu, sen koostavat elementit ja yhteydet pudotetaan automaattisesti piirtoalustalle, jonka jälkeen suunnittelija voi nimetä komponentteja ja muokata mallia antamalla parametreille uusia arvoja ja säätämällä mallin komponenttien määrää.

Pellens ja muut ovat luoneet suunnittelumallikirjaston, joka sisältää malleja eri kategorioista. Useimmat malleista ovat käyttäytymismalleja, mutta myös rakennemalleja on olemassa. Kirjoittajat toteavat vielä, että näiden koottujen

suunnittelumallien joukko ei tule koskaan olemaan täydellinen, sillä uusia malleja voidaan löytää milloin vain lisää ja lisätä kirjastoon. [Pellens et al., 2008]

3.2.2. Dynaamisesti laajennettavissa olevien virtuaaliympäristöjen puitteiden rakentaminen

Alexandre [2003] kertoo erityisistä puitteista, jotka on luotu dynaamisesti laajennettavissa olevien verkkoympäristöjen rakentamista varten. Kyseessä olevat puitteet ovat sellaisia, jotka auttavat sovelluskehittäjiä luomaan virtuaalimaailmasovelluksia Javalla kooten komponentteja ja käyttäytymistä helposti uudelleenkäytettävällä tavalla.

Suunnittelumalleilla voidaan kuvailla näiden puitteiden tarkoitus. Ne voivat antaa sovellusohjelmoijille tavan käyttää kehystä tarvitsematta ymmärtää sitä, miten se tarkalleen ottaen toimii ja opettaa kehyksen useita suunnitteluyksityiskohtia. Alexandre esittelee tärkeimpinä kaksi oleellista suunnittelumallia: MVC ja Abstrakti tehdas.

MVC, Model-View-Controller, eli malli-näkymä-ohjain -malli on suunnittelumalli, jota käytetään laajasti ja josta on paljon tietoa alan kirjallisuudessa. MVC-mallin ”näkymä” esittää ”mallin” sisältämän datan. Näkymän vastuulla on pitää yllä esityksen johdonmukaisuutta mallin muuttuessa. Ohjain liittää näkymän ja mallin yhteen. Se reagoi niihin tapahtumiin, jotka voivat johtaa mallin muuttamiseen.

Abstrakti tehdas -malli, joka on myös Gamman ja kumppaneiden [Gamma et al., 1995] kirjassa esitelty, antaa taas rajapinnan toisiinsa liittyvien tai toisistaan riippuvien objektien perheiden luomiselle määrittelemättä niiden konkreettisia luokkia. Asiakkaat kutsuvat luontioperaatioita saadakseen aikaan luokan ilmentymiä, mutta eivät ole tietoisia käyttämistään konkreettisista luokista. Tämä sallii tehtaan kapseloida se, miten objekteja luodaan. [Alexandre, 2003]

3.2.3. Lautapeli- ja tietokonepohjaisten strategiapelien vertailu suunnittelumallien avulla

Loh ja Soon [2006] käyttävät pelien suunnittelumallilähestymistapaa vertailukohtana eräälle suositulle tosiaikaiselle strategiatietokonepelille ja eräälle toiselle suositulle strategiselle lautapelille. He toteavat, että jo tiedossa olevia eroavaisuuksia voidaan käyttää apuna, kun yritetään muokata muista ei-tietokonepeleistä menestyviä tietokonepelisuunnitelmia. Ne voivat olla hyödyllisiä myös, kun yritetään muokata olemassa olevista tietokonepohjaisista peleistä tai simulaatioista hienompia parantamalla käytettävien välineistöjen vahvuuksia.

Olemassa olevat lautapelit ovat usein olleet inspiraationa tietokonepelien peli-ideoille ja -käsitteille. Tällainen ei kuitenkaan aina välttämättä toimi, mikä

voi johtua esimerkiksi siitä, että yritetään kopioida alkuperäistä peliä liian tarkkaan ilman mitään muutoksia, jotka ottaisivat huomioon tietokonepohjaisen formaatin vahvuudet ja rajoitukset verrattuna alkuperäiseen. Loh ja Soon pyrkivät tunnistamaan pelisuunnittelun eroavaisuudet, jotka sallivat pelien yhteisen ydinpelattavuuden säilyttää viehättävyytensä välineistön eroista huolimatta.

Loh ja Soon vertailevat kahta strategiapeliä. Samankaltaista näissä peleissä on muun muassa moninpelin mahdollisuus ja pelin päämäärät, jotka liittyvät muiden pelaajien eliminointiin simuloidun sodankäynnin seurauksena, sekä pelaamisen keskittyminen strategisten päätösten tekoon näiden tavoitteiden saavuttamiseksi. Pelien välinen vertailu paljastaa merkittävästi enemmän pelisuunnittelumalleja, joita esiintyy tietokonepelissä, mutta ei lautapelissä. Tämä viittaa kasvavaan pelauselementtien määrään tietokonepelissä, ja erojen tutkiminen peilaa pelaamisen syvyyden ja monimutkaisuuden kasvuun. Lisäksi useilla jaetuilla pelisuunnittelumalleilla, vaikkakin läsnä molemmissa peleissä, oli huomattavia eroavaisuuksia niiden toteutusasteissa (tietokonepelissä monimutkaisemmin ja syvällisemmin kuin lautapelissä).

Analyysiensä pohjalta Loh ja Soon [2006] ovat päätelleet, miten tunnistetut pelisuunnittelumallien erot osoittavat mukaelmiin, jotka hyödyntävät tietokoneen eroavaisuuksia ja lisämahdollisuuksia verrattuna lautapelimuotoon. Vaikkakaan yksittäinen vertailu ei ole riittävä, jotta voitaisiin päätellä tulosten oikeellisuutta tai yleistä sovellettavuutta, lähestymistavan menestys samankaltaisuuksien ja erojen tunnistamisessa osoittaa lähestymistavan toimivuuden myös pidemmälle vietyjen vertailujen kanssa.

3.2.4. Suunnittelumallien käytön motivointia tietokonepelitutkimuksen avulla

Gestwick [2007] puolestaan esittelee tietokonepelin case-tutkimuksen, jota voidaan käyttää suunnittelumallien tarpeen motivointina. Esiteltävänä on yksinkertainen peli, joka edustaa kaupallisten pelien toiminnallisuutta, kuten animaatioita, törmäysten havaitsemista jne.

Huolimatta aktiivisesta malliyhteisöstä ja jatkuvasta innostuksesta malleihin liittyen sekä käytännössä että tietojenkäsittelyn opetuksessa, suunnittelumalleja koskevia väärinkäsityksiä on runsaasti. Gestwickin henkilökohtainen kokemus osoittaa, että opiskelijat eivät usein pysty käsittämään mallien olemusta yksittäisistä esimerkeistä. Uskomus on, että malleja voidaan opettaa parhaiten esittämällä ne oikeassa ohjelmistossa, joka sisältää usean mallin yhteistyötä.

Kaupallisesti menestyvät pelit on yleensä toteutettu C tai C++-kielellä. Suunnittelumallit tuovat hyötyjä selkeyteen ja uudelleenkäytettävyyteen, mutta

eivät välttämättä suoritustehoon. Vaikka suunnittelumallisanasto ei puutu todellakaan peleistä, sen huolenaiheita on toistuvasti syrjäyttämässä tarve nopeuteen, mikä johtaa yhteen tarkoitukseen virtaviivaistettuun koodiin. On myös tärkeää tehdä ero pelisuunnittelun ja peliohjelmoinnin välille. On ollut yrityksiä luokitella ”suunnittelumallit” pelisuunnittelussa, mutta nämä eroavat olio-ohjelmistokehityksen suunnittelumalleista. Koska oliotekniikoita käyttävä peliohjelmointi on ilmentymä olio-ohjelmistotuotannosta, voidaan kohtuullisesti odottaa perinteisten mallien olevan sovellettavia. Lähes kaikilla peleillä on yhteinen arkkitehtuuri, huolimatta toteutuksen kielestä tai käytetyistä kirjastoista. [Gestwicki, 2007]

3.2.5. Oliopohjaiset suunnittelumallit pelien kehittämisen apuna

Ampatzoglou ja Chatzigeorgiou [2006] esittelevät oliopohjaisia suunnittelumalleja pelien kehittämiseen. Koska pelien nopea evoluutio vaatii suurta joustavuutta, koodin uudelleenkäytettävyyttä ja alhaisia ylläpitokustannuksia, on suunnittelumallien hyötyjen tutkimiseksi laadullinen ja määrällinen arviointiprojekti ”open source” eli avoimen koodin projekteista käynnistetty. Vaikka pelien kehittäminen on vahva teollisuudenala, tutkimuskenttä on vielä lapsenkengissään. Tämä saa pelien ohjelmointiammattilaiset vaatimaan parempia kehitysmenetelmiä ja ohjelmiston kehitystekniikoita. Ero pelien ja muiden ohjelmistomuotojen välillä on se, että pelien kehitysryhmät koostuvat useiden eri erikoisosaamisalojen ihmisistä, kuten käsikirjoittajat, pelin pääsuunnittelija, erilaiset ohjelmoijat, muusikko, äänitehosteryhmä, graafikot ja testaajat.

Suunnittelumallien käyttö pelikehityksessä on avoin tutkimuskenttä. Kirjallisuudesta ei ole helppoa löytää mallikatalogeja, joita voitaisiin käyttää ”yleisinä ratkaisuina yleisiin ongelmiin” peleissä. Tällaiset katalogit tekisivät kommunikoinnin suunnittelijoiden välillä helpommaksi ja tällaisten ohjelmien dokumentaation ymmärrettävämmäksi. Suunnittelumalleja peleille voidaan lähestyä kahdesta eri perspektiivistä: malleina, joita käytetään pelimekaniikan, eli pelattavuuden ja pelin sääntöjen, kuvailuun ja oliosuunnittelumallien käyttönä pelien ohjelmoinnissa.

Ampatzoglou ja Chatzigeorgiou [2006] arvioivat oliosuunnittelumallien käyttöä pelikehityksessä tarkastelemalla kahta avoimen lähdekoodin peliä. Tulokset vertailusta olivat lähes identtiset ja osoittavat, että mallit voivat olla hyödyllisiä, kun otetaan huomioon ylläpidettävyys. Tutkimuksessa peliversio, joka sisältää mallin, on vähentänyt monimutkaisuutta verrattuna aiempaan versioon ilman mallia. Lisäksi mallien soveltaminen tapaa lisätä ohjelmiston koheesiota. Kontrastina tälle, kun projektien koko on kasvanut mallin sisältämässä versiossa, on pelien kehittyvän luonteen vuoksi seurauksena uskomus, että suunnittelumallien soveliasta käyttöä tulisi rohkaista peliohjelmoinnissa.

4. Suunnittelumallien hyödyt ja haitat

Suunnittelumallien käyttö ei aina välttämättä ole täysin perusteltua. Vaikka mallit ovat periaatteessa hyödyllisiä ohjelmistosuunnittelussa ja ongelmien ratkaisussa, eivät ne aina päde kaikkiin tilanteisiin. Suunnittelumallien käyttö saattaa esimerkiksi kasvattaa liikaa ohjelmiston kokoa tai tehdä ohjelmistoarkkitehtuurista turhan monimutkaisen. Kunkin suunnittelumallin haitat tulisikin aina luetella suunnittelumallin kuvauksessa [Koskimies ja Mikkonen, 2005], jotta voidaan sitten arvioida sen käytön kannattavuutta. Suunnittelumallin kuvauksen ”seuraukset” -osiossa onkin tapana kertoa ko. mallin soveltamisen hyödyistä ja haitoista.

Siinä missä suunnittelumalli yleensä edistää jotain ohjelmiston laatuominaisuutta, se voi samalla todennäköisesti heikentää jotain toista ominaisuutta. Esimerkiksi jos edistetään ohjelmiston muunneltavuutta ja uudelleenkäytettävyyttä, saatetaan samalla tulla heikentäneeksi suorituskykyä. [Koskimies ja Mikkonen, 2005]

Koska suunnittelumallit eivät ole esitysmuodostaan riippuvaisia mistään ohjelmointikielestä, niiden sovellukset tapaavat hävitä varsinaisessa koodissa ellei niitä erikseen kuvata dokumentoinnissa tai koodissa olevissa kommenteissa [Koskimies ja Mikkonen, 2005]. Tämä tarkoittaa, että suoraan koodia lukemalla ei pystytä mitenkään näkemään siinä mahdollisesti käytettyä suunnittelumallia, sillä suunnittelumalleja ei koskaan anneta suoraan minään tiettynä koodina. Jos kuitenkin tahdottaisiin koodissa ilmaista käytetty suunnittelumalli, on siitä merkittävä tiedot kommentteina.

Koska suunnittelumalleja alkaa nykyisellään olla jo melkoinen määrä, on niiden kaikkien tunteminen ja muistaminen yhdelle suunnittelijalle täysi mahdottomuus. Kukaan ei voi ehtiä opetella koko ajan laajentuvaa malliavaruutta. Mallien muistamista ja opettelua haittaa myös usein niiden sidonnaisuus englannin kieleen – vaatii erinomaista kielitaitoa saada oikea mielleyhtymä esimerkiksi mallin kummallisesta nimestä [Rintala ja Jokinen, 2001].

Monet nimeävät suunnittelumallien oleellisimmaksi hyödyksi mallien tarjoaman tietämyksen välittämisen uusille suunnittelijoille ja muille alan työskentelijöille. Kun kaikki tuntevat suunnittelumallit, helpottuu myös asiantuntijoiden välinen kommunikointi. Aikaa ja vaivaa säästyy niin henkilökohtaisissa ajatteluprosesseissa, dokumentaatiossa kuin keskusteluissa kun tiettyyn malliin voidaan viitata vain sen nimeä käyttäen, selittämättä sen tarkemmin, mitä tarkoitusta ko. suunnittelumallia on tarkoitus käyttää. Suunnittelumallit ovat erinomainen tapa kerätä talteen organisaatioissa olevaa sovellusaluekohtaista tietoa, joka näin säilyy, vaikka ihmiset vaihtuvatkin [Rintala ja Jokinen, 2001].

Suunnittelumallit tarjoavat myös uusia, korkeamman tason rakenneabstraktioita, joita yhdistelemällä ohjelmistoja voidaan rakentaa [Koskimies ja Mikkonen, 2005]. Pyritään siis seuraamaan kehitystä ja havaitsemaan toistuvia rakenteita, jotka voidaan sitten nimetä ja esittää tietyn muotoisina. Eli parhaimmillaan suunnittelumalli on abstrakti suunnitteludokumentti, joka voidaan ottaa käyttöön ohjelmiston suunnittelussa heti, kun suunnittelija tunnistaa ongelmasta kohdan, johon malli sopii [Rintala ja Jokinen, 2001].

Se että suunnittelumallit eivät ole mitenkään riippuvaisia mistään tietystä ohjelmointikielestä, voidaan kenties toisten mielestä laskea haitaksi, mutta useimmat pitänevät sitä pelkästään hyvänä asiana. Mallien yleinen ja abstrakti muoto sallii niille paljon laajemman sovellusalueen kuin jos malli olisi tarkoitettu vain tietyn ohjelmointikielen ongelmanratkaisuvälineeksi. Erilaisia toteutusesimerkkejä ei toki mikään estä antamasta myös koodina suunnittelumallin kuvauksen yhteydessä, jottei selvitys jäisi liiankin yleisluonteiseksi mieltää suunnittelijalle.

Suunnittelumallit helpottavat menestyneiden suunnitelmien ja arkkitehtuureiden uudelleenkäyttöä. Suunnittelumallit auttavat valitsemaan suunnitteluvaihtoehtoja, jotka tekevät järjestelmästä uudelleenkäytettävän ja välttämään vaihtoehtoja, jotka siitä tinkivät. Suunnittelumallit voivat parantaa olemassa olevien järjestelmien dokumentaatiota ja ylläpidettävyyttä, esittämällä luokan, objektien vuorovaikutuksen ja niiden tarkoituksen täsmällisen määrittelyn. [Gamma et al., 1995]

Pelisuunnittelussa suunnittelumallien hyödyt voidaan todeta esimerkiksi, kun esitellään peliä yleisölle. Pelisuunnittelumalli antaa rakenteisen kuvauksen suunnitelmasta ja tiettyjen valintojen selitykset voidaan tehdä joko viittaamalla muihin vastaaviin peleihin, joissa on käytetty samoja suunnittelumalleja tai kuvailemalla, miten mallin korvaaminen jollain toisella voisi muuttaa pelin pelattavuutta. Pelisuunnittelumallit tarjoavat siis keinon tarkistaa pelin staattinen määritelmä aina, kun siitä tulee epävarmuutta. Kommunikoinnin parantuminen voi myös auttaa pelien suunnittelua ja luomista monella tasolla. [Björk and Holopainen, 2004]

Suunnittelumalleista voidaan saada suurin hyöty dokumentoimalla ne huolellisesti aina, kun uusia havaitaan. Oleellista on nimetä suunnittelumalli kuvaavasti ja yksiselitteisesti, jotta se jää sitä käyttävien tahojen mieleen. Dokumentointia helpottaa yleisesti käytetyt ”säännöt” siitä, mitä tulee kirjata ylös. Hyvän dokumentointiesimerkin antaa jo Gamman ja kumppaneiden kirja [Gamma et al., 1995], jossa määritellään, mitä kaikkea kuvauksen tulisi sisältää sekä annetaan useita esimerkkejä. Dokumentointiin on määrämuotoisten do-

kumenttipohjien lisäksi kehitetty muodollisempia menetelmiä – niin sanottuja mallikieliä (pattern language). [Rintala ja Jokinen, 2001]

5. Yhteenveto

Tässä tutkielmassa olen pohtinut suunnittelumallien merkitystä ja niiden soveltamisen kannattavuutta. Yleisesti ajatellen uskoisin, että suunnittelumallien hyödyt ovat merkittävästi suuremmat kuin mahdolliset haitat. Tosin on tietysti otettava huomioon, että suunnittelumallin käyttöönotto vaatii tarkkaa pohdintaa ja ongelman määrittelyä, jotta voidaan arvioida, onko se kannattavaa. Olen erityisesti keskittynyt pelien suunnittelun suunnittelumallien olemukseen ja esitellyt muutamia tutkimuksia aiheesta. Pelisuunnittelumallien tutkimus on vielä nykyisellään lapsenkengissään, sillä siihen suhtaudutaan ehkä vielä jossain määrin epätieteellisenä tutkimuskohteena, mutta tutkimusala lienee kuitenkin nousujohteisessa suosiossa.

Viiteluettelo

[Alexandre, 2003] Thomas Alexandre, Using design patterns to build dynamically collaborative virtual environments. In: *Proc. of the 2nd International Conference on Principles and Practice of Programming in Java* (Jun. 2003). Computer Science Press, 21-23.

[Ampatzoglou and Chatzigeorgiou, 2006] Apostolos Ampatzouglou and Alexander Chatzigeorgiou, Evaluation of object-oriented design patterns in game development. *Information and Software Technology* **49**, 5 (May 2007), 445-454.

[Björk and Holopainen, 2004] Staffan Björk and Jussi Holopainen. *Patterns in Game Design*. Charles River Media, 2004.

[Crawford and Kaplan, 2003] William Crawford and Jonathan Kaplan, *J2EE Design Patterns*. O'Reilly & Associates, 2003.

[Gamma et al., 1995] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Gestwicki, 2007] Paul V. Gestwicki, Computer games as motivation for design patterns. *ACM SIGCSE Bull.* **39**, 1 (Mar. 2007), 233-237.

[Haikala ja Märijärvi, 2002] Ilkka Haikala ja Jukka Märijärvi, *Ohjelmistotuotanto*. Talentum, 2002.

[Jones and Boyle, 2007] Ray Jones and Tom Boyle, Learning object patterns for programming. *Interdisciplinary Journal of Knowledge and Learning Objects* **3**, (2007), 19-28.

[Koskimies, 2000] Kai Koskimies, *Oliokirja*. Suomen ATK-kustannus, 2000.

[Koskimies ja Mikkonen, 2005] Kai Koskimies ja Tommi Mikkonen, *Ohjelmisto-arkkitehtuurit*. Talentum, 2005.

[Loh and Soon, 2006] Shanming Loh and Seah Hoch Soon, Comparing computer and traditional games using game design patterns. In: *Proc. of the 2006 International Conference on Game Research and Development* (Dec. 2006), 237-241.

[Narsoo and Mohamudally, 2008] J. Narsoo and N. Mohamudally, Identification of design patterns for mobile services with J2ME. *Issues in Informing Science and Information Technology* **5** (2008), 623-643.

[Pellens et al., 2008] Bram Pellens, Olga De Troyer and Frederic Kleinermann, CoDePa: A conceptual design pattern approach to model behaviour for X3D worlds. In: *Proc. of the 13th International Symposium on 3D Web Technology* (Aug. 2008). ACM, 91-99.

[Rintala ja Jokinen, 2001] Matti Rintala ja Jyke Jokinen, *Olioiden ohjelmointi C++:lla*. Talentum, 2001.