Erkki Mäkinen (toim.)

Pieniä tietojenkäsittely-tieteellisiä tutkimuksia

Kesä 2007

TIETOJENKÄSITTELYTIETEIDEN LAITOS
TAMPEREEN YLIOPISTO

D-2007-11

TAMPERE 2007

TAMPEREEN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
JULKAISUSARJA D – VERKKOJULKAISUT
D-2007-11, SYYSKUU 2007

Erkki Mäkinen (toim.)

Pieniä tietojenkäsittely-tieteellisiä tutkimuksia

Kesä 2007

TIETOJENKÄSITTELYTIETEIDEN LAITOS
33014 TAMPEREEN YLIOPISTO

ISBN 978-951-44-7052-3
ISSN 1795-4274

Sisällysluettelo

Matkapuhelinten puhekäyttöliittymät

Juha Hjelm

Tiivistelmä.
Tutkielmassa käsitellään yleisesti puheen ominaisuuksia ja sen tuomia hyötyjä ja haittoja ihmisen ja tietokoneen välisen vuorovaikutuksen kannalta. Lisäksi selvitetään matkapuhelimen tuomia vaikutuksia erilaisiin käyttöliittymävaihtoehtoihin, mutta erityisesti puhekäyttöliittymiin.

1. Johdanto

Puheen käyttäminen tietokoneen ohjaamiseen ja toisaalta tietokoneen antama puhepalaute oli tuttua jo ennen nykyisiä tietokoneita monista sci-fi-kirjoista ja -elokuvista. Näissä tarinoissa tietokoneita pystyttiin komentamaan normaaleilla puhekomennoilla, ja niiden antama palaute oli usein lähes tai kokonaan kuin ihmisen puhetta. Puheen käyttämisellä tietokoneen ohjaamiseen saatiin tarinaan sopivaa futuristista otetta, mutta toisaalta sillä tehtiin tietokoneista inhimillisempiä.

Puheen käyttö tietokoneen käyttöliittymissä tekee tietokoneen käyttämisestä erilaisen kokemuksen kuin perinteisen hiiren ja näppäimistön käyttö. Jos tietokone pystyy tunnistamaan käyttäjän puheen luotettavasti ja tarkasti, tietokoneen käytöstä tulee paljon luonnollisempaa ja helpompaa. Käyttäjän ei tarvitse tietää paljoa tietokoneen käyttämisestä ja koneen teknisistä ominaisuuksista, jos koneelle voidaan ilmoittaa tavallisella puhekielellä, mitä haluaa tehdä. Toisaalta koneen antama puhepalaute tuo ihmisen ja tietokoneen väliseen vuorovaikutukseen uuden ulottuvuuden. Koneen antama puhepalaute on käyttäjän puheen tapaan ihmiselle luonnollinen tapa välittää tietoa ja monessa tapauksessa paljon tehokkaampikin tapa.

Puheen käyttäminen tietokoneen käyttöliittymissä ei ole tietenkään ongelmatonta. Monet ongelmista ovat teknisiä, kuten se, miten puheen tunnistuksesta ja puheen tuottamisesta saadaan luotettavaa ja tarkkaa. Asiaan liittyy myös käytettävyysongelmia, kuten ohjelman tilan ilmoittaminen käyttäjälle ja multimodaalisissa ohjelmissa käytetyn modaliteetin ilmoittaminen. Lisäksi täytyy ottaa huomioon esimerkiksi puheen julkisuus: minkälaisia asioita voidaan puhekäyttöliittymissä puhua.

Matkapuhelinten puhekäyttöliittymillä on monia tavallisten tietokoneiden puhekäyttöliittymien ominaisuuksista, mutta niiden pieni koko, erilainen käyttötapa ja käyttöympäristö tekevät niistä mielenkiintoisen ympäristön. Matkapuhelinten pieni koko ja siitä seuraava pieni näppäimistön ja ruudun koko vai-

keuttavat perinteistä näppäimistöön perustuvaa käyttötapaa, varsinkin uudemmissa puhelinmalleissa, joissa on paljon ominaisuuksia.

Tässä tutkielmassa tutkitaan puheen käyttöä matkapuhelimissa ja erityisesti keskitytään käyttäjän puheeseen ohjelmistolle annetuissa komennoissa. Ensin käydään läpi yleisellä tasolla puheen ominaisuuksia ja ongelmia, ja sen jälkeen käsitellään näitä erityisesti matkapuhelimissa. Työssä tutkitaan myös jo markkinoilla olevia puhekäyttöliittymillä varustettuja matkapuhelimia ja lopuksi yritetään selvittää alan tulevaisuuden näkymiä, nimenomaan minkälaisia puhekäyttöliittymäominaisuuksia tulevaisuuden puhekäyttöliittymien ominaisuuksia matkapuhelimissa ja niiden yleistymistä tavallisten kuluttajien käyttöön.

2. Puheen hyödyt

Puhe on erittäin tehokas kommunikointitapa. Chapaniksen [1975] tutkimuksessa puhe oli tehokkain kommunikaatiotapa interaktiivisessa ongelmanratkaisutehtävässä ihmisten välillä. Verrattuna muihin tapoihin, kuten kirjoittamiseen, puhe oli vähintään kaksi kertaa nopeampaa. Mutta on myös huomattu, että perinteisissä toimisto tehtävissä hiiren ja näppäimistön käyttö oli tehokkaampaa kuin puheen käyttö [Damper et al., 1996]. Toisaalta puhelimen pienempi näppäimistö ja näyttö voivat haitata käyttäjää niin paljon, että puheen käyttö voi olla nopeampaa.

Puheen ei aina tarvitse olla ainoa tietokoneen tai puhelimen käyttötapa, vaan käyttäjä voi vaihtaa eri käyttötapojen välillä tarpeen mukaan. Laitteen multimodaalisuus tuo lisää mahdollisuuksia ihmisen ja tietokoneen vuorovaikutukseen. Erityisesti tilanteissa, joissa on monia samanaikaisia tehtäviä, käyttäjä hyötyy multimodaalisesta käyttöliittymästä.

Puheessa käytätetyt ilmaukset voivat olla hyvin tehokkaita. Käyttäjä voi monesti lyhyellä lauseella antaa komennon paljon tehokkaammin kuin perinteisen hiiren tai näppäimistön käytöllä. Esimerkkinä voisi mainita samantyyppisten objektien valinta suuresta ja hajallaan olevasta joukosta. Käyttäjä voisi puheen avulla valita objekteja esimerkiksi koon tai värin mukaan, mikä vaatisi enemmän aikaa esimerkiksi hiiren avulla. Toinen, enemmän matkapuhelimiin liittyvä, esimerkki liittyy hakemistorakenteessa liikkumiseen ja eri työmoodien valinta: käyttäjä pystyy paljon helpommin valitsemaan objektin syvästä valikkorakenteesta puheen avulla.

Viimeisenä ja ehkäpä tärkeimpänä ominaisuutena, puhe on luonnollinen tapa kommunikoida ihmisille. Puhumaan opitaan jo pienenä lapsena, joten lähes kaikki matkapuhelinten käyttäjät pystyvät puhumaan ja ymmärtämään puhetta. Tehokkuus ei useinkaan ole tärkein asia ihmisille käyttötapaa valites-

sa. Puhe on monissa tapauksissa suosittu tietokoneen käyttötapa, vaikka se ei olisikaan tehokkain tai nopein tapa.

3. Puheen ongelmat

Puheella on myös ominaisuuksia, jotka voivat tuottaa ongelmia ohjelmia tehtäessä ja käytettäessä. Puheen hitaus tietyissä tilanteissa tuli jo mainittua, mutta muita joissakin tapauksissa negatiivisia ominaisuuksia ovat puheen väliaikaisuus ja peräkkäisyys Koska puhe on väliaikaista käyttäjän vastuulle jää muistaa, mitä on sanottu, ja lisäksi usein ohjelman tilan muistaminen. Puheen peräkkäisyys aiheuttaa sen, että informaatio annetaan puhujan haluamassa järjestyksessä, joten informaation vastaanottaja ei voi nopeasti päästä käsiksi itselleen tärkeään tietoon.

Puheella on myös monia rajoituksia ihmisen ja tietokoneen välisessä vuorovaikutuksessa. Suhm [2003] on kategorioinut ne kuuteen luokkaan: puheen tunnistaja, puhuttu kieli, ympäristö, ihmisen kognitio, käyttäjä ja laitteisto.

Puheen tunnistajaa täytyy useimmissa tapauksissa opettaa tunnistamaan käyttäjän puhetapaa, koska useimmissa tapauksissa käyttäjät puhuvat aina hieman eri tavalla: erilaiset aksentit, slangisanat, nopea tai hidas puhetapa. Vaikka puheen tunnistamisessa on menty eteenpäin, ei se ole vielä lähelläkään täydellistä ja kestää vielä pitkään ennen kuin kuka tahansa käyttäjä voi käyttää puheentunnistussovellusta luonnollista puhetta käyttäen suoraan, opettamatta sitä ensin.

Käytetty kieli vaikuttaa saatavilla oleviin puheen tunnistajiin. Suurille kielille löytyy enemmän halukkaita yrityksiä kehittämään puhekäyttöliittymiä, koska näille sovelluksille löytyy varmasti enemmän käyttäjiäkin. Suomenkielisiä puheen tunnistajia on kuitenkin tutkittu kohtuullisen paljon, suurelta osin eri yliopistoissa. Käytetty kieli voi kuitenkin vaikuttaa siihen, kuinka luonnollista kieltä käyttäjä voi käyttää puhuessaan laitteelle.

Ihmisten välinen puhe on hyvin spontaania ja hyvin usein siinä lopetetaan lauseita ja sanoja kesken ja käytetään eleitä ja ilmeitä puheen lisäksi. Puhuttaessa tietokoneelle tämä voi muodostua ongelmaksi. Sen lisäksi, että puheen täytyy olla selkeää, puheen täytyy myös koostua kokonaisista sanoista ja lauseista, jotta tietokone sen ymmärtäisi. Lisäksi puheessa on ominaista toisen päälle puhuminen. Koska normaalissa puheessa puhutaan usein toisen päälle, kun toisen keskustelijan puheen perusidea on tullut selväksi, täytyy tämä ottaa huomioon puhekäyttöliittymiä suunniteltaessa. Käyttäjä voi hyvinkin aloittaa puheensa ennen tietokoneen puheen loppumista, esimerkiksi tilanteissa, jossa tietokone luettelee vaihtoehtoja käyttäjälle.

Ympäristö vaikuttaa paljon puheen tunnistajan tarkkuuteen. Ympäristöissä jossa on paljon taustamelua, puheen tunnistaminen on vaikeampaa. Lisäksi jos taustalla on paljon muiden ihmisten puhetta, tietokone voi ottaa komentoja myös muilta ihmiseltä tulleilta ääniltä. Käytetyn laitteiston laskentateho, mikrofoni ja kaiuttimet vaikuttavat puhekäyttöliittymän toimintaan.

Käyttäjään ja ihmisen kognitioon liittyy puhekäyttöliittymien suurimpia ongelmia. Koska usein ei visuaalista tietoa ole saatavilla, puhekäyttöliittymissä täytyy luottaa käyttäjän muistiin: mitä ollaan tekemässä tai mikä on ohjelman tila.

Puhekäyttöliittymien suunnittelu on paljon vaikeampaa kuin tavallisten käyttöliittymien, koska suunnittelussa täytyy ottaa huomioon kaikki edelliset seikat normaalien hyvien suunnittelutapojen lisäksi.

Pitkien listojen ja taulukoiden kuvaaminen käyttäjälle on vaikeaa pelkästään puheen avulla. Uusien tutkimusten mukaan [Suhm et al., 2001] pitkien listojen esittely käyttäjälle voi joissain tilanteissa olla hyödyllistä puheen avulla. Näissä tapauksissa on huomattavaa, että käyttäjän ei tarvitse muistaa kaikkea koneen sanomaa tietoa, vaan hän hakee isosta määrästä tietoa muutaman tarvitsemansa datan. Jos taas käyttäjä haluaisi tietää kaikista koneen luettelemista asioista, olisi tiedon esittäminen graafisesti parempi vaihtoehto.

Koska puheen avulla käytävässä kommunikaatiossa täytyy keskustelijan itse muistaa kaikki tärkeät seikat, on huomattava, että liian pitkät dialogit voivat vaikeuttaa puhekäyttöliittymien käyttämistä. Dialogit pitäisi suunnitella niin, että käyttäjä pystyy muistamaan kaikki oleelliset seikat ja laitteen pitäisi auttaa käyttäjää tässä, esimerkiksi pitämällä sanotut lauseet tarpeeksi lyhyinä.

4. Matkapuhelinten erityisominaisuudet

Matkapuhelimet ovat monin tavoin erilaisia käyttöliittymäsuunnittelun kannalta perinteisiin pöytäkoneisiin verrattuna. Toisaalta niiden suunnittelussa voidaan käyttää joiltakin osin hyväksi kannettavien tietokoneiden ja PDA-laitteiden suunnitteluohjeita.

Matkapuhelimista on tehty pienempiä ja helpommin mukaan otettavia, joten käyttäjät pääsevät helpommin käsiksi erilaiseen informaatioon ja viihteeseen. Puhelimissa yhdistyy lukuisat erilaiset laitteet yhteen koneeseen: matkapuhelimessa voi olla tavallisten puhelimen ja tekstiviestien lisäksi sähköposti ja internet- palvelut ja pelejä, mp3-soitin, radio ja kalenteri. Vaikka puhelimiin on yhdistetty monia muita laitteita, niiden koko on koko ajan pienentynyt. Koska laitteiden valmistajat yhdistävät monia ominaisuuksia samaan laitteeseen, täytyy erityistä huomiota kiinnittää laitteen käyttöliittymään ja löytää käyttöliittymäsuunnitteluun myös uusia vaihtoehtoja. Useasti huomataan, että mitä

enemmän yhteen laitteeseen lisätään erilaista toiminnallisuutta, sitä vaikeampi sitä on käyttää, koska laite yrittää tarjota niin monia eri palveluita käyttäjälle, ettei tee niistä yhtään hyvin.

On huomattava, että matkapuhelimet ovat suunniteltu ensisijaisesti käytettäväksi puhelimena eikä kannettavana tietokoneena. Vaikka jotkut mallit ovat nykyään enemmän pienoistietokoneiden tyyppisiä, useimmat matkapuhelimet ovat ensisijaisesti puhelimen mallisia ja päätarkoitukseltaan puhelimia.

Matkapuhelimen koko rajoittaa näppäimistön ja ruudun kokoa, vaikka markkinoilla on joitain puhelinmalleja, kuten Nokian kommunikaattorit, joissa näppäimistö ja kuvaruutu rajoittavat käyttöä vähemmän kuin pienemmissä malleissa. Lisäksi matkapuhelimen näppäimistö rajoittaa käyttäjän keinoja manipuloida dataa.

Koska puhelimissa on ollut perinteisesti vähän muistia, ei niihin ole ollut mahdollista liittää kuin rajoitettu puhekäyttöliittymä, jos sitäkään. Tämä muistin vähäisyys rajoittaa vielä hyvän aikaa matkapuhelinten kehitystä, vaikka uusimmissa malleissa onkin jo varsin kehittyneitä puheentunnistusohjelmistoja.

Matkapuhelinta käytetään suureksi osaksi julkisilla paikoilla tai paikoissa joissa on muita henkilöitä paikalla. Tämä voi rajoittaa puheen käyttöä esimerkiksi seuraavissa tilanteissa:

1. Taustahäly voi estää matkapuhelinta kuulemasta kaikkia komentoja tai saada ottamaan komennon taustalta kuuluvista sanoista tai lauseista. Toisaalta käyttäjä ei välttämättä kuule puhelimen sanomia asioita, jos ympäristössä on liikaa melua.
2. Ehkäpä tärkein rajoite julkisilla paikoilla on salaisten tai arkaluonteisten asioiden käsittely matkapuhelimen puhekäyttöliittymän avulla. Vaikka olisikin kätevää antaa puhelimen lukea itselle tulleet sähköpostit, kaikissa tilanteissa se ei ole mahdollista, koska käyttäjä ei tiedä, sisältääkö posti arkaluontoista materiaalia.
3. Puhekäyttöliittymien käyttöön julkisilla paikoilla voi ainakin alkuvaiheessa vaikuttaa ihmisten halu puhua puhelimelleen. Puhelimen hands-free -laitteiston tapaan ihmisistä voi näyttää oudolta, että joku puhuu komentoja näennäisesti tyhjään ilmaan.

4.1. Puhe matkapuhelimissa

Puheentunnistusohjelmalla on mahdollista tehdä matkapuhelin helpommaksi käyttää. Mitä enemmän toiminnallisuutta laitteisiin lisätään sitä enemmän etua puhekäyttöliittymästä voi olla, koska puheen avulla puhelinta hallitakseen ei tarvitse käyttää kovin paljon laitteiden pieniä nappuloita tai muita hallintavälineitä. Puhekäyttöliittymän etu on, että käyttäjän ei välttämättä tarvitse katsoa

laitetta sitä käyttäessään, vaan se voi olla esimerkiksi taskussa ja käyttäjälle jää kädet vapaaksi muihin toimiin.

Lo ja Meng [2000] huomasivat, että valittaessa suuresta määrästä vaihtoehtoja puheen käyttö vähensi selvästi käyttäjän aikaa päästä käsiksi haluamaansa tietoon. Tutkimuksessa käytetty aika putosi eri näppäilyvaihtoehdoista riippuen 188 tai 86 sekunnista 60 sekuntiin. On kuitenkin huomattava, että asiaa on tutkittava enemmän luotettavien tietojen saamiseksi.

Tutkimuksessa oli toteutettu sääpalvelu ja pörssikursseista kertova palvelu. Näissä molemmissa huomattiin, että puheen käyttäminen nopeutti palvelun käyttöä.

Kuvassa 1 on esimerkki Voice Signalilta [2002] tekstiviestin lähettämisestä pelkän puheen avulla.

You: "SMS Message."
Phone: "Please say a name."
You: "Tom Stewart."
Phone: "Did you say: Tom Stewart?"
You: "Yes."
Phone: "Begin message. When you are done, say 'Message complete.'"
You: "Tom, I started reading the book you recommended. Interesting. Let's meet Tuesday a.m. to discuss. Message complete."
Phone says "Do you want to send this message now?"
You: "Yes."
Phone: "Sending message to Tom Stewart."

Kuva 1. Tekstiviestin lähettäminen puheen avulla [Voice Signal, 2002]

Esimerkistä nähdään kuinka käyttäjä lähettää yksinkertaisen viestin puhelimen muistissa olevaan numeroon. Kuitenkin puhelimen täytyy tunnistaa:

- Käyttäjän komento: tekstiviesti, Viesti valmis, Aloita viesti. Kun puhelin saa käskyn, se vaihtaa moodia käyttäjän käskyn mukaan.

- Kenelle viesti lähetetään: Tom Stewart. Puhelin etsii puhelinmuistiosta löytyykö sieltä henkilöä.

- Lähetetty viesti. Puhelimen täytyy pystyä tunnistamaan käyttäjän vapaata puhetta, että viesti voidaan tunnistaan tarkasti.

Käyttäjän puhekomennot näkyisivät myös puhelimen ruudulla, josta käyttäjä pystyy tarkastamaan lähetettävän viestin ennen lähetystä. Huomionarvoista edellisessä esimerkissä on puhelimen tekemät varmistukset käyttäjältä. Koska puheen tunnistus on niin virhealtista, täytyy käyttöliittymäsuunnittelussa ottaa huomioon myös mahdolliset väärin ymmärretyt käskyt. Tämän takia

edelliseen esimerkkiin voisi lisätä myös viestin korjauksen tai käskyn peruutuksen. Näiden lisäksi laitteen täytyy pystyä tuottamaan ymmärrettävää puhetta.

4.2. Erilaisia käyttötapoja

Puheen käyttö ei kuitenkaan ole paras vaihtoehto joka tilanteeseen. Joskus on helpompaa käyttää matkapuhelinta pelkästään näppäimistön kautta, ja käyttäjät eivät pidä, jos heidät pakotetaan käyttämään puhekäyttöliittymää sopimattomissa tilanteissa.

Mahdollisia käyttötapoja puheen käyttöön:

- Soittaminen puhelimella sanomalla nimi puhelinmuistiosta.
- Soittaminen puhelimella sanomalla haluttu puhelinnumero.
- Sähköpostin tai tekstiviestin saneleminen ja sen lähettäminen.
- Tapahtuman lisääminen kalenteriin puheen avulla
- Valikkorakenteessa liikkuminen tai ohjelman käynnistäminen.
- Musiikin valitseminen ja soittaminen puhelimen muistista sanomalla puhelimelle artistin nimi, kappaleen nimi, levyn nimi tai haluttu musiikkityyppi.

Näiden esimerkkien avulla on helppo päätellä, minkälaisia tehtäviä puhelimella voisi tehdä puheen avulla. Kohdassa 4.7 on esimerkkejä nykyisistä puhelimista ja niissä olevista puhekäyttöliittymistä.

Koska puhetunnistus vaatii paljon tehoa laitteelta, voi olla parempi tehdä rajoitettuja puheentunnistusohjelmia. Nämä ohjelmat tunnistavat vain määrättyjä sanoja ja niitä voidaan käyttää vain tietyissä tilanteissa, kuten näppäinlukon laittaminen päällä tai puhelimeen vastaaminen.

4.3. Dialogin hallinta

Yang ja Rothkrantz [2002] esitelevät SWAMP-sovellusta, joka perustuu järjestelmän ja käyttäjän välisiin dialogeihin. Ensin käyttäjä ilmaisee halunsa käyttää järjestelmää, yleensä jonkin komennon kautta. Tämän jälkeen järjestelmä vie käyttäjän erilaisten dialogien läpi yrittäen saada selville, mitä käyttäjä haluaa tehdä. Kun järjestelmä on kerännyt tarvittavat tiedot, haluttu toiminto suoritetaan. Järjestelmässä käyttöliittymä on jaettu kolmeen osaan:

1. Puheen tunnistus. Tämän osan on tarkoitus ymmärtää mitä sanoja käyttäjä järjestelmälle sanoo, ja muuttaa ne järjestelmän ymmärtämään muotoon.
2. Dialogi-komponentti. Tutkii puheen tunnistuksen tuottamaa tietoa, ja näiden perusteella yrittää selvittää mitä käyttäjä haluaa tehdä.
3. Puheen syntetisointi. Tarkoituksena tuottaa järjestelmän antama puhepalaute käyttäjälle.

Jokaisesta käyttäjän sanomasta asiasta järjestelmä yrittää selvittää onko kyseessä komento sille, vai käyttäjän muuten puhuma asia. Jos järjestelmä tunnis-

taa itselleen tulleen komennon, sen täytyy sen jälkeen selvittää onko tarpeeksi tietoa toiminnon suorittamiseen. Jos tietoa ei ole tarpeeksi, täytyy sitä kysyä lisää käyttäjältä, kunnes tietoa tehtävästä on tarpeeksi sen suorittamiseksi.

Puhekäyttöliittymää suunniteltaessa täytyy suunnitella kaikki tarvittavat dialogit, jotta järjestelmään voidaan sisällyttää tarpeeksi suuri sanavarasto. Tärkeää on selvittää, asettaako käyttäjä puhelimen kuuntelumoodiin vai kuunnellaanko käyttäjää koko ajan. Molemmissa on hyvät ja huonot puolensa. Jos käyttäjän täytyy asettaa puhelin kuuntelemaan itseään käsin, toimitaan puhekäyttöliittymän yhtä perusajatusta vastaan. Jos puhelin kuuntelee käyttäjäänsä koko ajan, nousee virheiden mahdollisuus huomattavasti.

Oviat ja Lunsford [2005] huomasivat, että puhekäyttöliittymää käyttäessä monet puhuvat myös itselleen samalla kun käyttävät laitetta. Joten sen lisäksi, että laitteen täytyy löytää taustamelun ja muiden ihmisten puheen seasta käyttäjän puhe, täytyy myös käyttäjän puheesta löytää oikeat laitteelle annetut käskyt. Huomattavaa on myös, että katseen seurannasta ei tässä tapauksessa ole kovin paljon hyötyä, koska monet käyttäjät katsoivat laitetta myös puhuessaan itselleen.

Ihmiset sopeutuvat hyvin keskustelukumppaninsa puhetyyliin normaalissa keskustelussa. Tätä voidaan käyttää hyväksi puhekäyttöliittymää suunniteltaessa, koska ihmiset käyttävät usein niitä sanoja ja lauseita, joita käytettiin myös aikaisemmin. Ohjelma käyttää vain muutamia useimmin käytettyjä sanontoja ja antaa käyttäjän sopeutua tilanteeseen. Vähitellen myös käyttäjä alkaisi käyttää ohjelman käyttämiä sanontoja ja lauseita. Tässä tapauksessa ohjelmalla ei tarvitsisi olla kovin laajaa sanastoa pystyäkseen toteuttamaan hyvän puhekäyttöliittymän.

4.4. Käyttöliittymämetaforat

Käyttäjiä voidaan auttaa suunnittelemalla käyttöliittymä jonkin käyttäjän tunnistaman metaforan mukaan [Howell et al., 2006]. Jos käyttäjille kuvaillaan valikkorakenne kuin se olisi samanlainen kuin Windowsin käyttämä, se voi auttaa ihmisiä, jotka ovat kyseistä käyttöjärjestelmää käyttäneet. Käyttäjät pystyvät visualisoimaan, miltä valikkorakenne näyttää mielessään, eikä heidän tarvitse käyttää niin paljon resursseja pelkästään valikon ja sen vaihtoehtojen miettimiseen.

Howell ja muut [2006] huomasivat, että valikon visualisointi auttaa käyttäjiä puhekäyttöliittymän käytössä. Käyttäjät, jotka pystyivät visualisoimaan valikkorakenteen, olivat muita tehokkaampia järjestelmän käytössä. Tietokoneiden graafisista käyttöliittymistä lainatut käsitteet ovat tuttuja suurimmalle osalle käyttäjistä, joten näiden käyttämisestä puhekäyttöliittymien suunnittelussa voisi olla suuri hyöty käytettävyyden kannalta.

Laite voi antaa käyttäjälle myös muuta äänipalautetta kuin puhetta [Pirhonen et al., 2002]. Jos palaute annetaan erilaisin äänimerkein, käyttäjä voi samaan aikaan esimerkiksi kuunnella musiikkia tai puhua puhelimessa. Tämä voi myös nopeuttaa puhelimen käyttöä, jos puhelin sanoisikaan että viesti lähetetty, vaan antaisi äänimerkin kuten esimerkiksi joissain tietokoneohjelmissa.

4.5. Puheen tunnistus muualla kuin puhelimessa

Koska muistia on vähän, Pakucs [2004] ehdottaa, että joissakin tapauksissa tarvittavat sanastot voitaisiin ladataan vasta niitä tarvittaessa. Esimerkiksi jos käyttäjä hakee ravintolan ruokalistaa, sen erikoissanasto ladataan puhelimeen vasta tarvittaessa. Näissä tapauksissa puhelimen ohjelmisto toimisi universaalina tulkkina erilaisille ohjelmille, joita käyttäjä voisi käyttää puhelimen kautta. Puhelimessa ei tarvitsisi välttämättä olla puheen tunnistajaa ja generoijaa, vaan näitä tarvittaessa puhelimella voitaisiin ottaa yhteys palvelimelle jossa nämä ohjelmistot löytyisivät.

On myös kehitetty multimodaalinen luonnollista puhetta käyttävä sovellus [Mitchell et al., 2006] joka käyttää tavallisia puhelimia. Sovellus ei ole asennettu puhelimeen, vaan käyttäjä ottaa siihen yhteyden puhelimellaan. Koska nykyisiin puhelimiin on vaikeaa tehdä kehittyneitä puhetunnistusohjelmaa, yksi vaihtoehto voisi olla, että käyttäjä ottaisi puhelimellaan yhteyden palvelimeen, jossa puhekäyttöliittymää voitaisiin käyttää. Koska esimerkiksi sähköpostia lähetettäessä ja vastaanotettaessa täytyy kuitenkin ottaa yhteys sähköpostipalvelimelle, voitaisiin samalla siellä hoitaa myös sähköpostien luku ja kirjoitus puhekäyttöliittymän avulla.

4.6. Järjestelmän opastus

Koska eri valmistajien laitteet suunnitellaan eri lailla ja niihin luultavasti tulee erilaiset käyttöliittymät ja puhetta käytetään eri lailla, tämä voi tuottaa ongelmia käyttäjille, jotka käyttävät eri valmistajien laitteita.

Puhekäyttöliittymää suunniteltaessa täytyy ottaa huomioon että joskus käyttäjän täytyy tehdä monia asioita samaan aikaan. Joskus järjestelmän täytyy myös pystyä keskeyttämään käyttäjän toimet tärkeämmän asian tullessa esiin. Esimerkiksi jos käyttäjä on sanelemassa tekstiviestiä, miten järjestelmän kannattaa keskeyttää käyttäjä jonkun soittaessa puhelimeen.

4.7. Nykyisiä matkapuhelimia

Tähän tutkielmaan tutkittiin Nokian matkapuhelimia ja niissä olevia ominaisuuksia puheen käyttämiseksi. Nokialla on uusimmissa malleissa jo ominai-

suuksia, joissa puhetta voidaan käyttää rajoitetusti eri toiminnoissa. Alla on kuvattu tarkemmin kaksi uudempaa mallia.

Nokian E65-mallissa on toteutettu äänikomennot valikoiden pikavalintoihin, näppäimistön lukitsemiseen ja profiileiden vaihtamiseen. Lisäksi puhelimessa on mahdollisuus puhuviin soittoääniin, jossa soittajan nimi sanotaan samalla kun puhelimen soittoääni soi.

Nokia E61i-mallissa on toteutettu samoja ominaisuuksia kuin E65-mallissakin: äänikomennot valikoiden pikavalintoihin, näppäimistön lukitsemiseen ja profiileiden vaihtamiseen ja puhuvat soittoäänet. Näiden lisäksi puhelimessa on myös tekstinlukuohjelma, jota voidaan käyttää tekstiviestien lukemiseen käyttäjälle. Ikävä kyllä pohjoismaisia kieliä ei ole vielä tuettu.

Molemmat mallit ovat hyvä esimerkki siitä, mihin matkapuhelinten puhekäyttöliittymissä ollaan menossa. Nämä kaksi mallia ovat yleisesti saatavilla, myös hinnan suhteen. Molemmissa puhelimissa on toteutettu jo yksinkertainen puheentunnistus ohjelmat, jotka helpottavat puhelimen käyttöä. Lisäksi mallissa E61i oleva tekstinlukuohjelma vapauttaa käyttäjän kädet ja katseen esimerkiksi tekstiviestiä luettaessa autossa.

5. Multimodaalisuus

Useimmissa tapauksissa puhekäyttöliittymän lisäksi puhelinta voidaan käyttää myös näppäimistön kautta. Näiden lisäksi puhelimen käyttämiseen voidaan käyttää myös kosketusta ja palautteen saamiseen myös tuntoaistia. Tuntoaistia käytetään puhelimissa jo esimerkiksi värinähälyttimen muodossa. Kosketusta voidaan käyttää esimerkiksi kosketusnäytön avulla jossa käytetään tiettyjä eleitä puhelimen käytössä. Multimodaalisuus eli puhelimen eri käyttötapojen yhdistäminen, on paras vaihtoehto tulevaisuudessa puhelimen käyttöliittymää suunniteltaessa. Multimodaalisia käyttöliittymiä suositellaan erityisesti mobiileille laitteille [Ringland and Scahill, 2003].

Multimodaaliset järjestelmät voidaan jakaa kahteen luokkaan: järjestelmiin, joissa käytetään yhtä modaalisuutta kerrallaan, ja järjestelmiin, jossa voidaan käyttää kaikkia modaliteetteja yhtä aikaa.

Käyttäjä voi vaihtaa moodia tarpeen mukaan. Esimerkiksi käyttäjä voi ottaa yhteyden manuaalisesti bussiaikataulutietokantaan ja sen jälkeen käyttää puhetta oikean aikataulun tai reittioppaan hakemiseen.

Mielenkiintoisempi vaihtoehto on, jos käyttäjä voi käyttää eri modaliteetteja samaan aikaan. Esimerkiksi käyttäjä voisi hotellin kirjautumislomaketta täyttäessään sanoa järjestelmälle halutut päivämäärät ja samaan aikaan valita ruudulta haluttu huonetyyppi näppäimistöä käyttäen. Tässä tapauksessa eri moodeja käytetään itsenäisesti. Toinen vaihtoehto on, jos eri moodeja käytetään yhdessä

saman tehtävän suorittamiseen. Karttaohjelmaa käyttäessään käyttäjä voisi esimerkiksi näyttää kartalta kahta pistettä ja käskeä puheen avulla järjestelmän etsimään parhaan reitin pisteiden välillä. Tässä esimerkissä mitään toimintoa ei voida suorittaa vain yhden moodin perusteella, vaan toimintoon vaaditaan molemmat.

Eri moodit soveltuvat erilaisiin tehtäviin eri tavoin. Muuttuva ympäristö tuo haasteita puhelimen käytössä. Multimodaalisuus voi olla parempi ratkaisu kuin yhtä moodia käyttävät laitteet, koska käyttäjä voi vaihtaa moodia ympäristön muuttuessa. Usein puhelinta käytettäessä tehdään myös muita tehtäviä, esimerkiksi ajetaan autoa. Näissä tapauksissa multimodaalinen laite tuo käyttäjälle vaihtoehtoja laitteen käytössä. Lisäksi multimodaalisissa laitteissa käyttäjä voisi valita omien mieltymysten mukaan käytetyn moodin, ja tämän takia samanlaista laitetta voisi käyttää monenlaiset ihmiset. Varmistuksissa ja virheen korjauksissa multimodaalisilla laitteilla voisi olla suuri etu, koska esimerkiksi puheen avulla virheen korjaaminen tekstiviestistä on paljon vaikeampaa kuin näytön ja näppäimistön tai kosketusnäytön avulla.

6. Yhteenveto

Puhekäyttöliittymät ovat käytössä jo monessa tietokoneiden sovelluksessa ja rajoitutussa käytössä matkapuhelimissa. Puheen käyttö matkapuhelimen ohjaamisessa ja palautteessa tuo omia ongelmiaan, mutta toisaalta sen tuomat edut ovat selkeät tietyissä tilanteissa. Matkapuhelimien käyttöliittymät pitäisi suunnitella multimodaalisiksi, jossa puheen käytöllä olisi oma osansa. Käyttäjälle pitäisi antaa mahdollisuus käyttää matkapuhelintaan tilanteeseen sopivalla ja käyttämän haluamalla tavalla.

Viiteluettelo

[Chapanis, 1975] Chapanis, A. Interactive human communication. *Scientific American* **232** (1975), 36-42.

[Damper *et al*., 1996] Damper, R., Tranchant and M., Lewis, S. Speech versus keying in command and control: effect of concurrent tasking. *International Journal of Human-Computer Studies* **45** (1996)337-348.

[Howell et al., 2006] Mark Howell, Steve Love and Mark Turner, Visualisation improves the usability of voice-operated mobile phone services *International Journal of Human-Computer Studies* **64** (2006) 754-769.

[Lo and Meng, 2002] Tin-Hang Lo and Helen M. Meng, WAP-speech: Deriving synergy between WAP and the spoken dialog interface. In: *Proceedings of the 11th International World Wide Web Conference, 2002.*

[Mitchell et al., 2006] Stella Mitchell, Christopher J. Pavlovski, Braam Smith, Harry Stavropoulos and David Wood, Multimodal natural language platform supporting cellular phones. *ACM SIGMOBILE Mobile Computing and Communications* **10** (2006) 34-45.

[Oviat and Lunsford, 2005] Sharon Oviatt and Rebecca Lunsford, Multimodal interfaces for cell phones and mobile technology. *International Journal of Speech Technology* **8** (2005)127–132.

[Pakucs, 2004] Botond Pakucs, Butler: a universal speech interface for mobile environments. In: *Mobile human-computer interaction – MobileHCI 2004* **3160**, 399-403.

[Pirhonen et al., 2002] Antti Pirhonen, Stephen Brewster and Christopher Holguin, Gestural and audio metaphors as a means of control for mobile devices. In: *Conference on Human Factors in Computing System*, 291 – 298.

[Ringland and Scahill, 2003] S.P.A. Ringland and F.J. Scahill, Multimodality — the future of the wireless user interface. *BT technology journal 2003* **21** (2003) 181-191.

[Suhm, 2003] Suhm, B. Towards best practices for speech user interface design. In: *EUROSPEECH-2003*, 2217-2220.

[Voice Signal, 2002] Voice Signal Technologies, Freedom of Speech: How Embedded Speech Recognition Benefits Mobile Device Users. Available as http:// www.digital-lifestyles.com/communication/wp.cfm?wpid=1126.

[Yang and Rothkrantz, 2002] C.K. Yang and L.J.M. Rothkrantz, Knowledge based speech interfacing for handhelds. In: *Proceedings of the Fifth International Conference on Text, Speech and Dialogs (TSD2002), Lecture Notes in Artificial Intelligence* **2610**, 405-412.

Suunnittelumallit erään sovelluksen näkökulmasta

Samuli Mäki

Tiivistelmä

Tässä tutkielmassa tutustutaan ensin yleisellä tasolla oliopohjaisiin suunnittelumalleihin. Muutamien oliopohjaisten suunnittelumallien soveltuvuutta arvioidaan tarkemmin tutkielmaa varten ohjelmoidussa sovelluskehyksessä.

Avainsanat ja -sanonnat: Suunnittelumallit, ohjelmistokehys, sovelluskehys
CR-luokat: D.1.5, D2.2, D.2.11

1. Johdanto

Suunnittelumalleista on tullut viimeisen vuosikymmenen aikana erottamaton osa ohjelmistotuotantoa. Suurten ohjelmistojen elinkaarien aikana kertyvistä kuluista yhä suurempi osa muodostuu näiden ylläpidosta ja pienien muutoksien tekemisestä. Mitä jäykempi ja kovakoodatumpi ohjelmisto, sen vaikeampaa on muuttaa jo valmista ohjelmistoa. Suunnittelumallien soveltamisen päätarkoitus on tuoda joustavuutta ohjelmistoihin, mikä helpottaa monimutkaisten ohjelmistojen suunnittelua ja ylläpitoa.

Tässä työssä lähestytään suunnittelumalleja puoliksi käytännön ja puoliksi teorian kannalta. Tutkielman ei siis ole tarkoitus olla lähellekään kattava katsaus suunnittelumallien maailmaan. Suunnittelumallit esitellään yleisellä tasolla lyhyesti niiden historian ja käsitteen määrittelyn kautta. Lisäksi esitellään mielenkiintoisesti suunnittelumalleihin liittyvät antisuunnittelumallit.

Toinen puolisko siirtyy käsittelemään suunnittelumalleja toteutetun sovelluskehyksen näkökulmasta. Sovelluskehys on korttipelejä varten suunniteltu ja ohjelmoitu kehys, josta erikoistamalla tulisi olla mahdollista saada aikaan kokonaisia korttipeliohjelmia. Viittä kehyksen suunnittelumallia tarkastellaan erityisesti niiden soveltuvuuden ja roolin kannalta.

Tulee pitää mielessä että suunnittelumallit eivät ole mitenkään kiveen kirjoitettu käsite, vaan niiden määritelmään vaikuttaa esimerkiksi ohjelmointikieli, jolla järjestelmä toteutetaan. Tässä tutkielmassa aihetta lähestytään pääosin olio-ohjelmointikielten näkökulmasta ja linjaukset perustuvat Gamman ja muiden [1995] ja Koskimiehen [2000] linjauksiin suunnittelumalleista.

2. Suunnittelumallit

2.1. Historia

Ohjelmistotuotanto ei ole alana vielä kovinkaan pitkäikäinen, vaan sen voidaan katsoa saaneen alkunsa 1960-luvulla. Suunnittelumallit eivät myöskään alun perin ole ohjelmistotuotannon kehittämiseen tarkoitettuja metodeja, vaan ne ovat rantautuneet sille muilta aloilta. Alun perin ajatuksen suunnittelumallien soveltamisesta esitti rakennusarkkitehti Christopher Alexander [Gamma et al., 1995]. Vaikka hänen ajatuksensa käsittelivätkin rakennusarkkitehtuurien suunnittelua, on idea suunnittelumallien nimeämisestä ja formaalista kuvaamisesta hyvin soveltuva myös ohjelmistotekniikan käyttöön.

Vasta 1980- ja 90-luvun vaihteessa heräsi ajatus soveltaa tätä ratkaisumallien ideaa myös olioperustaisten ohjelmistojen suunnittelussa, koska oli huomattu, että ohjelmistotuotannossa usein sovellettiin toistuvia ratkaisumalleja. Ward Cunningham ja Kent Beck ottivat käyttöön Alexanderin ideat SmallTalk-ympäristössä 1980-luvun lopussa. Tämän jälkeen aktiivisimmat suunnittelumallien tukijat perustivat ns. Hillside-ryhmän, joka alkoi järjestää PloP-konferensseja (Pattern Languages of Programs) vuodesta 1994. Tämän jälkeen vuonna 1995 ilmestyi suunnittelumallien saralla tärkeä merkkipaalu kun ”neljän kopla” julkaisi kirjansa Design Patterns [Gamma et al., 1995; Sierla, 2006].

Käytännössä suunnittelumallit eivät siis ilmaantuneet tyhjästä ohjelmistotekniikkaan, sillä suunnittelumallien avulla ratkaistavia ongelmia oli luonnollisesti aikaisemminkin ratkaistu. Tämä kuitenkin aikaisemmin — miksei osittain vielä tänä päivänäkin — nähtiin osana perinteisempää ohjelmointitekniikkaa. Uudentyyppisen ajattelun myötä suunnittelumalleja alettiin dokumentoida, mikä muutti niiden luonteen formaalimmaksi, kun aikaisemmin samaiset ratkaisut perustuivat ohjelmoijan omaan intuitiivisyyteen.

2.2. Määritelmä

Alexander [1977] määrittelee suunnittelumallin seuraavasti: ”Jokainen malli määrittelee ongelman, joka esiintyy uudelleen ja uudelleen ympäristössämme, ja sitten ydinratkaisun siihen, niin että sitä voidaan soveltaa miljoonia kertoja tekemättä kuitenkaan koskaan samaa ratkaisua kahdesti.”

Suunnittelumallit ovat nimettyjä ja hyvin dokumentoituja tapoja ratkaista tietty ongelma. Suunnittelumallin esittämä ratkaisu koostuu kokoelmasta olioita, luokkia ja niiden välisiä suhteita. Koskimies [2000] määrittelee ytimekkäästi suunnittelumallin tarkoittavan tapaa suunnitella (olio-) ohjelmiston osa tietyn usein esiintyvän ongelman ratkaisemiseksi.

Gamma ja muut [1995] luonnehtivat suunnittelumallien olevan pitkän kehityksen tulosta, kokeneiden ohjelmistosuunnittelijoiden hyväksi näkemiä ratkaisuja usein esiintyviin ongelmiin. Suunnittelumallit eivät edellytä ohjelmointiteknisesti mitään korkeamman asteen ”ohjelmointijippoja” vaan esittävät selkeän ja melko yksinkertaisen mutta abstraktin tavan ratkaista ongelmia. Suunnittelumallien dokumentoinnin ja nimeämisen myötä ne toimivat ikään kuin ohjelmiston suunnitteluun liittyvänä hiljaisen tiedon siirtämisenä. Tällöin kokemattomatkin suunnittelijat kykenevät hyödyntämään näitä hyväksi todettuja ratkaisuja.

Gamma ja muut [1995] kuvailevat yleisesti suunnittelumallin koostuvan seuraavista pääkohdista: sen nimi, ongelma, johon suunnittelumallia sovelletaan, kyseisen ongelman ratkaisu suunnittelumallin avulla ja suunnittelumallin soveltamisen seuraukset. Seuraavassa on eritelty edellä mainitut kohdat yksitellen.

Nimi.

> Nimeä voidaan käyttää ongelman kuvaamisessa. Sen tarkoituksena on kuvata samalla sekä ongelma että sen ratkaisu. Nimen tulisi olla havainnollinen, ja usein siihen liittyykin ilmeinen analogia reaalimaailmaan. Suunnittelumallien nimet tarjoavat ohjelmistokehittäjille yhteisen ongelmaan ja sen ratkaisuun liittyvän sanaston.

Ongelma.

> Kuvataan tilanne, johon suunnittelumallia sovelletaan. Tässä saatetaan kuvata luokka- tai oliorakenteita, jotka usein enteilevät huonoa ja joustamatonta suunnittelua. Joskus ongelman kuvaus sisältää myös listan edellytyksistä, joiden tulee täyttyä, jotta suunnittelumallia voidaan soveltaa.

Ratkaisu.

> Ratkaisu kuvaa suunnittelumalliin liittyvät elementit, niiden keskinäiset suhteet, vastuut sekä yhteistyön. Tässä kuvataan konkreettisen ratkaisun sijaan abstrakti ratkaisu, jolloin se on helpommin sovellettavissa eri konteksteissa.

Seuraukset.

> Seurauksilla tarkoitetaan suunnittelumallin soveltamisen tuloksia sekä mahdollisia kompromisseja, jotka joudutaan tekemään esimerkiksi tehokkuuden kustannuksella.

2.3. Suunnittelumallien luokittelu

Suunnittelumallit yleensä luokitellaan niiden käyttötarkoituksen perusteella kolmeen eri ryhmään: luontimalleihin, rakennusmalleihin ja käyttäytymismalleihin. Tätä luokittelua käyttävät sekä Gamma ja muut [1995] sekä Koskimies [2000].

Luontimallit (Creational patterns)

> Luontimallit abstrahoivat olioiden luomisprosessin, minkä avulla järjestelmät voidaan suunnitella riippumattomiksi siitä, miten niiden oliot luodaan [Gamma et al., 1995]. Tässä tutkielmassa esitellään tarkemmin luontimalleihin kuuluva Abstrakti tehdas (ks. alakohta 3.3.1).

Rakennemallit (Structural patterns)

> Rakennemallit huolehtivat olioiden välisistä suhteista ja siitä miten nämä muodostavat isompia kokonaisuuksia. Rakennemalleja käytetään myös suurten ohjelmistojen tehokkuuden ja muistin käytön optimoimiseen.

Käyttäytymismallit (Behavioural patterns)

> Nämä mallit huolehtivat algoritmeista ja olioiden välisistä vastuuketjuista. Olioiden ja luokkien määrittelemisen lisäksi käyttäytymismallit ottavat kantaa niiden väliseen kommunikaatioon. Myöhemmin näistä esitellään Tarkkailija (ks. alakohta 3.3.2), Tila (ks. alakohta 3.3.3) ja Strategia (ks. alakohta 3.3.3).

2.4. Antisuunnittelumallit

Antisuunnitelumallit ja niiden tunnistaminen on oleellisesti yhteydessä ns. oikeisiin suunnittelumalleihin. Tämä on käänteinen lähestymistapa oikeiden suunnittelumallien soveltamiseen verrattuna. Antisuunnittelumallit ovat huomattavasti enemmän yleisellä tasolla kuvattuja kuin suunnittelumallit. Antisuunnittelumallien tunnistamisen myötä voidaan havaita tarve soveltaa oikeita suunnittelumalleja. Seuraavassa on lueteltu antisuunnittelumallien tunnusmerkkejä, joista jokainen on tyypillinen ongelmatilanne, jonka ratkaisemiseen oikeat suunnittelumallit tähtäävät:

- koodin toistaminen
- ohjelman haarautuminen tietotyyppien perusteella
- kaikki mahdolliset vaihtoehdot kovakoodattu
- yliluokka tuntee jälkeläisensä
- epäselvät ja epäintuitiiviset rajapinnat

- ylisuuret luokat ja operaatiot
- paisunut ohjausluokka, joka sisältää paljon toisiinsa liittymättömiä attribuutteja ja operaatioita (ns. "Blob").

2.5. Mallien hyödyt ja haitat

Silloin kun suunnittelumallien avulla isojen ohjelmistojen suunnittelussa pystytään välttämään edellisessä luvussa listattuja antisuunnittelumallien ongelmia, on suunnittelumallien hyöty kiistaton. Suunnittelumallien hyöty korostuukin siis suurissa ohjelmistoissa, joille suunnittelumallit dokumentointeineen voivat tarjota helpommin ymmärrettävän rakenteen. Suunnittelumallit tuovatkin ohjelmointikieliä paremmin esille korkeamman tason rakenteita.

Yleisesti ohjelmistojen joustavuus lisääntyy huomattavasti suunnittelumalleja sovellettaessa. Ne lisäävät koodin uudelleenkäytettävyyttä, geneerisyyttä, sekä tekevät monien ratkaisujen dokumentoinnista selvempiä.

Tärkeä seikka suunnittelumallien soveltamiseen on että suunnittelija ymmärtää tarkasti, millaisissa tilanteissa mallia kannattaa käyttää. Suunnittelumalleja kuvattaessa luetellaankin tilanteita, joihin niiden soveltaminen sopii. Suunnittelumallin soveltamisen tulee aina lähteä tietyn ongelman identifioimisesta [Koskimies,2000].

Suunnittelumallien haittapuolet tulevat esiin tilanteissa kun konteksti, jossa niitä sovelletaan, ei olekaan ideaalinen suunnittelumallille. Tällaisessa tapauksessa suunnittelumalli – vaikka se ratkaisisikin halutun ongelman – lisää turhaan järjestelmän kompleksisuutta. Tilanteissa, joissa toteutettava ohjelma on pienehkö täsmäsovellus tiettyyn käyttötarkoitukseen, voi intuitiivinen ongelmanratkaisu olla suunnittelumalleja parempi tapa ratkaista ongelmia. Suunnittelumallit saattavat myös hajottaa loogisia kokonaisuuksia erilleen, mikä horjuttaa olioparadigman keskeistä ajatusta, olioiden analogiaa reaalimaailmaan, ja saattaa näin tehdä koodista ja suunnittelusta vaikeaselkoisempaa.

Suunnittelumallit lisäävät järjestelmän kerroksia ja samalla myös vähentävät tehokkuutta. Suunnittelumallien usein soveltama dynaaminen sidonta vaatii enemmän suorituskykyä. Kovakoodatut ratkaisut sisältävät usein suorat viitteet tarvittaviin olioihin ja tietotyyppeihin, mikä on tehokkaampi tapa ohjelman suorituksen kannalta. Nykyisellään tietokoneiden suorituskyvyn parantuessa kyseinen ongelma on merkittävä lähinnä rajallisempien resurssien ympäristöissä (matkapuhelimet, kämmenmikrot jne.) sekä kriittisissä sovelluksissa, joiden toiminnassa nopeus on tärkeässä roolissa (esimerkiksi järjestelmät, jotka tekevät lukuisia määriä iteraatioita).

Suunnittelumallien käytöstä saatava hyöty riippuu suunnittelijan ja ohjelmoijan malliin liittyvästä osaamisesta sekä mallista itsestään. Mikäli ohjelmoi-

jalle mallin toiminta ei ole täysin selvä, aiheuttaa sen soveltaminen vain lisäkustannuksia. [Vokác et al. , 2004].

3. Suunnittelumallit sovelluskehyksessä

3.1. Ohjelmistokehys

Ohjelmistokehyksellä tarkoitetaan olio-ohjelmoinnin yhteydessä kiinteästi toisiinsa liittyvien luokkien ja/tai rajapintojen kokoelmaa, joka toteuttaa tietyn ohjelmistoperheen perusarkkitehtuurin. Kehyksistä saadaan yksittäinen sovellus tai komponentti erikoistamalla. Kehys tarjoaa tätä tarkoitusta varten erikoistamisrajapinnan. Erikoistamisrajapinta ei yleensä ole eksplisiittisesti näkyvissä, vaan se annetaan epäformaalisti kehyksen dokumentoinnissa. [Koskimies, 2000]

Ohjelmistokehys määrää samalla myös siis koko ohjelmiston arkkitehtuurin; kokonaisrakenteen, jaon luokkiin ja olioihin, näiden vastuualueet sekä yhteistyön. Kehyksestä tehtävät erikoistukset ovat siis vahvasti sidoksissa kehyksissä tehtyihin ratkaisuihin.

3.2. Korttipeli-sovelluskehys

Sovelluskehys on eräänlainen ohjelmistokehys. Se on ohjelmisto, joka sisältää tietyntyyppisten sovellusten perusratkaisut. Tässä tapauksessa se on kehys, joka tarjoaa perustan erilaisten korttipelien ohjelmoimiselle.

Korttipeli-sovelluskehys on tyypiltään muunneltava kehys (white-box framework), mikä tarkoittaa sitä että sitä erikoistettaessa käytetään hyväksi perintämekanismia. Tällöin erikoistuksessa annetaan kehyksen luokille alaluokkia sekä voidaan määritellä niissä uudelleen operaatioita. Eräs toinen kehysarkkitehtuurityyppi on koottava kehys (black-box framework), joka kapseloi arkkitehtuuriratkaisut sisäänsä rajapintojen avulla käytettäväksi. Tähän verrattuna muunneltava kehys jättää erikoistukselle enemmän pelivaraa lisäten joustavuutta.

Ottaen huomioon sovelluskehyksen kontekstin ei ole tarvetta kapseloida liiaksi kehyksen luokkia. Samaten myös kehyksen ja sen erikoistuksen vähäisen iteroinnin vuoksi Korttipeli-sovelluskehys on pääosin muunneltava kehys. Koskimies [2000] toteaa monien kehysten olevan käytännössä edellä mainittujen tyyppien välimuotoja. Hän lisää myös että muunneltavat kehykset muokkautuvat yleensä vähitellen koottavaan suuntaan, kun niihin lisätään tarpeellisiksi havaittuja luokkia, jotka ovat sellaisinaan käytettävissä sovelluksissa.

3.3. Suunnittelumallit kehyksessä

Ohjelmistokehyksen joustavuutta ja yleisyyttä vaativa luonne suoranaisesti edellyttää suunnittelumallien tai vastaavien ratkaisujen soveltamista. Kehykset yleensä suunnitellaan kontekstin puitteissa mahdollisimman joustaviksi ja uudelleenkäytettäviksi. Kun halutaan jättää erikoistuksella paljon muunneltavia asioita, monia ratkaisuja ei voida kiinnittää liian sitoviksi.

Suunnittelumallit toimivat usein kahdessa päällekkäisessä osassa; kehyksessä sekä sen erikoistuksessa. Tällaisessa tilanteessa suunnittelumalli määrittelee erään kehyksen erikoistamiskohdan. Suunnittelumallien dokumenti antaa erikoistuksen tekijälle suuntaviivat erikoistuksen ohjelmointiin. Suunnittelumallit voidaan hyödyntää arkkitehtuurien rakennusosasina.

3.4. Tarkasteltavat suunnittelumallit

Seuraavassa esitellään Korttipeli-sovelluskehyksessä käytettäviä suunnittelumalleja.

3.4.1. Abstrakti tehdas (Abstract factory)

Tarkoitus

> Mallia käytetään kokonaisten toisistaan riippuvien olioperheiden luomisessa tarvitsematta tietää luotavien olioiden konkreettisia luokkia.

Motivointi

> Kehyksen erikoistuksen on kyettävä luomaan muun ohjelmiston kanssa (ja keskenään) yhteensopivia tuoteperheitä. Sovelluskehys ei kuitenkaan voi tietää siitä tehtävissä erikoistuksissa käytettäviä luokkia ennalta. Tässä korttipeleihin erikoistuvassa sovelluskehyksessä sen erikoistus tarvitsee monia erilaisia komponentteja, kuten

Soveltuvuus

> Abstrakti tehdas -suunnittelumalli soveltuu tilanteisiin kun toisiinsa liittyvät luokat on suunniteltu käytettäväksi yhdessä, mutta järjestelmän tulee silti olla riippumaton siitä, miten erikoistuksen luokat toteutetaan. Suunnittelumallia voidaan käyttää myös tilanteessa kun luodaan luokkakirjasto ja halutaan ainoastaan paljastaa luokkien rajapinnat.

Osallistujat

- AbstractFactory-rajapinta määrittelee operaatiot, joilla luodaan abstraktit oliot.

- Erikoistuksen tehdas toteuttaa edellä mainitun rajapinnan määrittelemät operaatiot.
- Abstraktit luokat määrittelevät tuoteperheen keskinäiset suhteet.
- Konkreettiset luokat perivät niitä vastaavat abstraktit luokat, ja toteuttavat näissä määrittelemättä jätetyt osat tai tarkentavat määriteltyjä.

Seuraukset

- Erilaisia erikoistuksia suunniteltaessa voidaan helposti vaihtaa luotava tuoteperhe säilyttäen yhteensopivuus muun kehyksen kanssa. Tämä onnistuu toteuttamalla abstrakti tehdas -rajapinta eri konkreettisella tehtaalla.
- Koottavan kehyksen (black-box framework) tapauksessa abstraktin tehtaan luomat oliot saadaan eristetyksi kapseloinnin avulla erikoistuksen luokista, jotka käyttävät niitä rajapintojen kautta.
- Abstraktin tehtaan käyttäminen kehyksessä edellyttää tarkkaa suunnittelua ja ennakointia, mitä järjestelmän kannalta tärkeitä luokkia erikoistus tulee tarvitsemaan, sillä malli ei tue tuotteiden lisäämistä.

3.4.2. Tarkkailija (Observer)

Tarkoitus

Suunnittelumalli mahdollistaa olioiden välisen kommunikaation, ilman että ne tuntevat toisiaan. Suunnittelumalli jakaa ohjelmistossa keskenään kommunikoivat osat tarkkailijoihin ja tarkkailtaviin.

Motivointi

Usein sovelluskehystä suunniteltaessa ei tiedetä kaikkia erikoistuksessa käytettäviä olioita, joiden on kuitenkin kyettävä kommunikoimaan keskenään. Suunnittelumalli mahdollistaa tarkkailtavan ”kuuntelun” ilman että tarkkailtava tuntee tarkkailtaviaan. Monissa tilanteissa oliolla saattaa olla monta tarkkailtavaa, jotka ovat kiinnostuneita tarkkailtavan tilan eri tapahtumista.

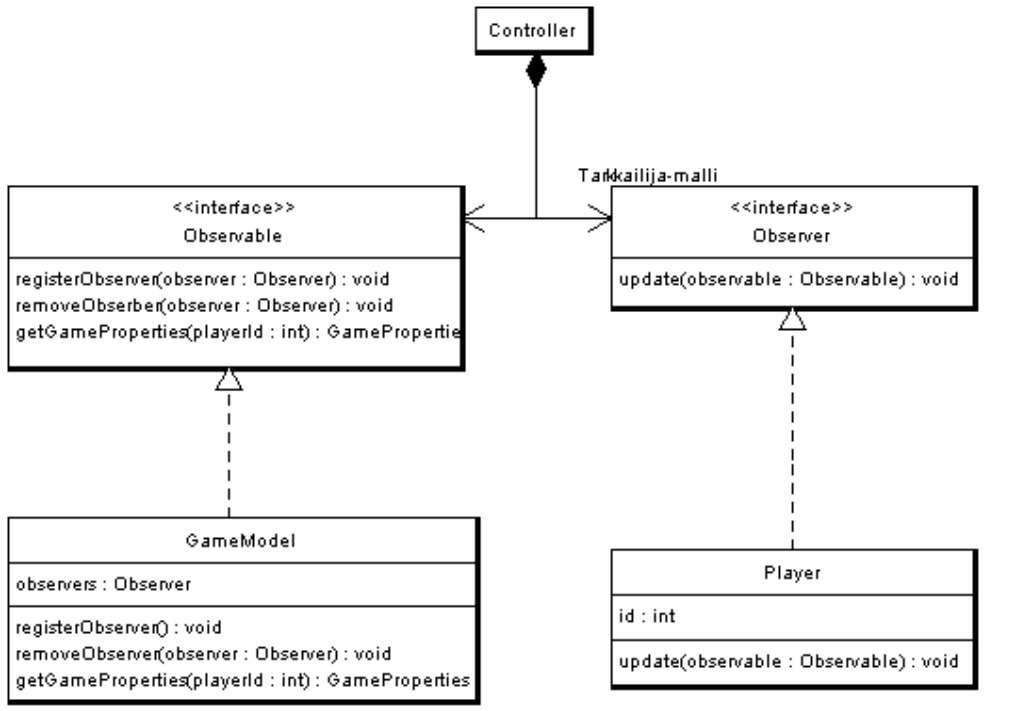
Soveltuvuus

Tarkkailija-suunnittelumallia voidaan käyttää kun järjestelmässä on kaksi tahoa, joista toinen on riippuvainen toisesta.

Osallistujat

- Tarkkailija toteuttaa Observer-rajapinnan, jonka kautta tarkkailtava ilmoittaa tarkkailtaville muutoksista niiden update-metodin kautta.

- Tarkkailtava toteuttaa Observable-rajapinnan, jonka kautta kuuntelija rekisteröityy kuuntelemaan tarkkailtavaa. (Ks. kuva 1.)



Kuva 1. Tarkkailija.

Seuraukset

- Tarkkailtavan ja tarkkailijan välillä on abstrakti liitos. Tarkkailtavalla on vain lista tarkkailtajista, joita se voi kutsua tilansa muuttuessa Observer-rajapinnan kautta.
- Uusia tarkkailijoita voi rekisteröityä kuuntelijoiksi milloin vain ilman, että itse tarkkailtavaa tarvitsee muuttaa.

3.4.3. Tila (State)

Tarkoitus

Olion käyttäytymistä voidaan muuttaa ajon aikana.

Motivointi

Korttipeleissä osa pelaajista voi olla ihmispelaajia ja osa tekoälyn ohjaamia pelaajia. Ihmispelaaja kuitenkin saattaa lopettaa pelin tai haluaa liittyä peliin kun peli on vielä kesken. Ohjelman olisi pystyttävä tekemään tämä muutos ajonaikaisesti ilman, että se vaikuttaa pelin kulkuun muuten kuin tämän vaihdettavan pelaajan kohdalta. Erikoistuksen tekijä voi haluta käyttää myös erilaisia tekoälyjä, mahdollisesti jopa ohjel-

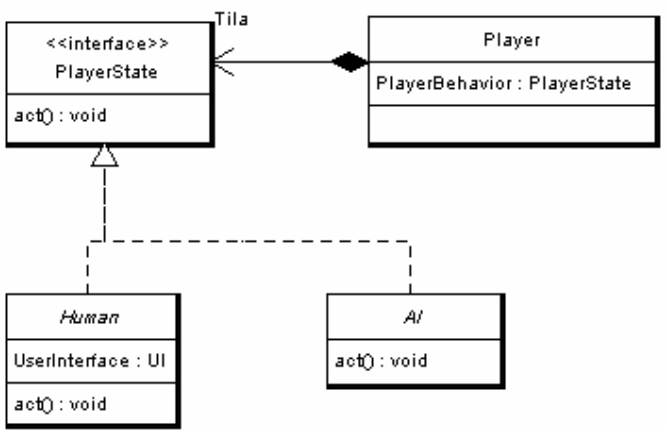
moida tekoälyn vaihtamaan omaa käyttäytymistään pelitilanteen mukaan.

Soveltuvuus

Malli soveltuu tilanteisiin kun olion käyttäytyminen riippuu sen tilasta ja sitä pitää pystyä muuttamaan ajon aikana. Tila-suunnittelumalli voi selkeyttää sellaisen olion rakennetta, jonka monilla operaatioilla on samantyyppinen ehtolauserakenne. Suunnittelumallin myötä olion tilaa voidaan käsitellä kuin toista oliota.

Osallistujat

- PlayerState-rajapinta määrittelee olion käyttäytymisestä vastaavat metodit; tässä tapauksessa vain act().
- Human-, ja AI-luokat toteuttavat PlayerState-rajapinnan ja määrittelevät tarvittavat metodit: AI tekoälyn ja Human kommunikoinnin käyttöliittymän kanssa.
- Player-luokka on yleinen pelaaja, joka tilansa myötä erikoistuu ihmispelaajan tai tekoälyn ohjattavaksi. (Ks. kuva 2.)



Kuva 2. Tila.

Seuraukset

- Malli sisällyttää olion käyttäytymisen sen tilaa kuvaavaan olioon.
- Koska kaikki tilaa koskeva koodi on aliluokissa, voidaan ne helposti vaihtaa toisiin vastaaviin, mikä helpottaa järjestelmän ylläpidettävyyttä.

3.4.4. Strategia (Strategy)

Tarkoitus

Malli määrittelee ja kapseloi vaihtoehtoisten algoritmien perheen. Asiakas voi halutessaan vaihtaa dynaamisesti käytettävää algoritmia perheen sisällä.

Motivointi

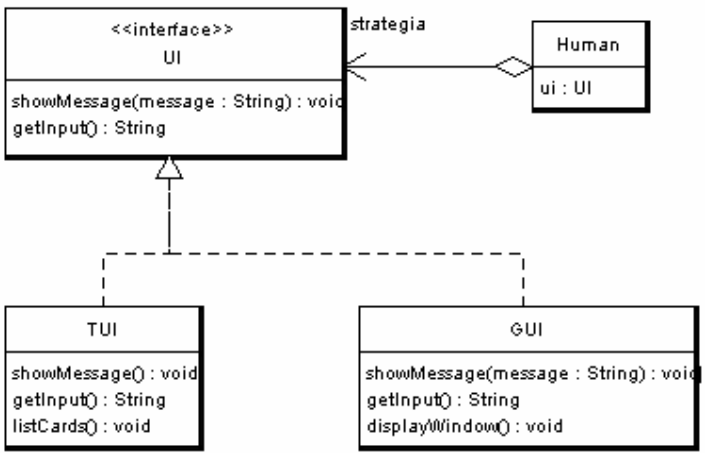
Monessa sovelluksessa on järkevää käyttää erilaisia algoritmeja saman tehtävän suorittamiseen. Olisi tietysti mahdollista sisällyttää algoritmit luokkaan, joka ne ajaa, mutta tämä tekisi usein luokasta liian monimutkaisen ja kömpelön. Korttipelin tapauksessa käyttöliittymätyyppiä ei haluta lyödä lukkoon, vaan kehyksen tulee tukea mm. komentorivikäyttöliittymää ja graafista käyttöliittymää.

Soveltuvuus

Suunnittelumallia voidaan käyttää tilanteissa, joissa muuten olisi monta erilailla saman toimenpiteen suorittavaa luokkaa tai vastaavia ehtolauseita yhden luokan sisällä. Strategia-rajapinnan taakse, sen toteuttaviin luokkiin, saadaan myös piilotettua käyttäjältä niiden toimintaan liittyvää dataa.

Osallistujat

- UI-rajapinta, joka määrittelee yleiset käyttöliittymän metodit.
- TUI-luokka toteuttaa rajapinnan metodit tekstipohjaisen käyttöliittymän avulla, ja GUI-luokka toteuttaa ne graafisessa käyttöliittymässä.
- Ihmiskäyttäjää edustava Player-luokka käyttää UI-rajapinnan kautta toista käyttöliittymistä. (Ks. kuva 3.)



Kuva 3. Strategia.

Seuraukset

- Mallilla voidaan korvata ehto- ja tapauslauseita.

- Kun algoritmit on kapseloitu ja jaettu omiin luokkiinsa, on niitä helppo laajentaa sekä vaihtaa.
- Mallin käyttäjän on oltava tietoinen kaikista algoritmeista käyttääkseen tehokkaasti suunnittelumallia hyväkseen.

3.4.5. Ainokainen (Singleton)

Tarkoitus

Mallin tarkoituksena on yksinkertaisesti taata että luokasta luodaan vain yksi ilmentymä.

Motivointi

On olemassa tilanteita, jolloin tietystä luokasta ei voi olla kuin yksi ilmentymä. Tämä voi seurata olion analogiasta reaalimaailman tai sitten halutaan vain ehkäistä ongelmia ja ristiriitoja, mitä tietyn luokan monet samanaikaiset ilmentymät voisivat tuoda. Korttipeli-sovelluskehyksessä on kyseessä jälkimmäinen; halutaan varmistaa, että abstraktin tehtaan perivästä erikoistuksen konkreettisesta tehtaasta luodaan vain yksi ilmentymä, jotta tuoteperhe pysyy yhtenäisenä.

Soveltuvuus

Malli soveltuu tilanteeseen, jolloin jostain luokasta tarvitaan vain yksi ilmentymä. Mallia voidaan käyttää myös tilanteessa, jolloin ilmentymää on pystyttävä laajentamaan aliluokkien avulla.

Osallistujat

- Ainokainen-malli joka on sisällytetty erikoistuksen konkreettiseen tehtaaseen. Siinä määritellän uniikki operaatio, jonka kautta asiakas saa luokan ilmentymän. Se siis vastaa itse ilmentymänsä luomisesta.

Seuraukset

- Ilmentymän luovassa metodissa voidaan kontrolloida ilmentymän luomista ja määritellä sille ehtoja.
- Pienellä muutoksella voidaan ilmentymien sallittu määrä muuttaa yhdestä mihin tahansa määrään.

Koodiesimerkki

```
private static HeartsFactory ref;
private HeartsFactory(){}
public static HeartsFactory getInstance(){
    if (ref == null)
     ref = new HeartsFactory();
    return ref;
}
```

Ilmentymän luominen oletusrakenninta käyttämällä estetään määrittelemällä se private-määreellä näkyväksi ainoastaan luokan sisällä. Luokka säilyttää myös staattista viittausta omaan ilmentymäänsä (jos sellainen on luotu). Metodi getInstance palauttaa tämän viitteen ja luo tarvittaessa luokan ilmentymän.

4. Yhteenveto ja loppupäätelmät

Nykypäivän ohjelmistotuotannossa korostuu tiimityön merkitys. Suunnittelumallien käyttö verrattuna epämääräisempään vain ohjelmointitekniikkana nähtävään tapaan ratkaista ongelmia edesauttaa tehokasta tiimityöskentelyä. Suunnittelumallien formaalin käyttämisen merkitys ohjelmistotuotannossa on monimuotoisempaa kuin vain ohjelmointitekniikan soveltaminen ongelmien ratkaisuun. Ne tuovat valmiit käsitteet ennen kenties vaikeammin kuvattaviin ongelmiin ja niiden ratkaisuihin, mikä helpottaa myös ryhmän sisäistä kommunikointia.

Suunnittelumallien tärkeä asema erityisesti kehysarkkitehtuureissa on kiistaton niiden mukanaan tuoman joustavuuden ja uudelleenkäytettävyyden myötä. Niiden käytössä kuitenkin tulee aina varmistaa, että niitä sovelletaan kontekstiin, mihin ne on tarkoitettu. On myös oleellista miettiä, kuinka tarpeellista uudelleenkäytettävyys ja joustavuus sovellukselle on, sillä jokaisessa tapauksessa se ei ole välttämätöntä.

Toteuttamassani sovelluskehyksessä eivät suunnittelumallit ole välttämättä esillä parhaalla mahdollisella tavalla, sillä työn luonteen vuoksi moni malli esiintyy kehyksessä osittain vain oman esiintymisensä, eikä välttämättä kehyksen kokonaisuuden, vuoksi. Joka tapauksessa toteutetunlaisen kehyksen luominen ilman suunnittelumallien tuomaa dynaamisuutta olisi huomattavasti vaikeampaa.

Suunnittelumallit eivät ole ohjelmistoarkkitehtuurissa mikään yleislääke ongelmien ratkaisemiseen. Koskimies on todennut, etteivät suunnittelumallit ole ”god given”, vaan niiden soveltamisen yhteydessä on aina sijaa terveelle järjenkäytölle.

Viiteluettelo

[Alexander, 1977] Christopher Alexander, *A Pattern Language: Towns, Buildings and Construction*. Oxford University Press, 1977.

[Gamma et al., 1995] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Koskimies, 2000] Kai Koskimies, *Oliokirja*. Satku - Kauppakaari, Jyväskylä, 2000.

[Sierla, 2006] Antto Sierla, Suunnittelumallien asema 2000-luvun olioperustaisessa ohjelmistokesityksessä. Pro gradu -tutkielma. Tampereen yliopisto, tietojenkäsittelytieteiden laitos.

[Vokác et al. ,2004] Marek Vokác, Walter Tichy, Dag I.K. Sjoberg, Erik Arisholm and Magne Aldrin, A controlled experiment comparing the maintainability of programs designed with and without design patterns – a replication in a real programming environment. *Empirical Software Engineering* **9** (2004), 149-195.

Sisäinen tietoturva

Timo Nikkari

Tiivistelmä.

Kun puhutaan tietoturvasta, niin monesti ajatellaan asiaa hyvin yksipuolisesti ja tietoturvan monet osa-alueet jäävät mainitsematta tai käsittelemättä. Tämä tutkimuksen tarkoitus on löytää tietoturvasta sisäisen tietoturvan osa. Sen tarkoitus on selventää ja ennen kaikkea tarkentaa tietoturvan kokonaisuutta. Lisäksi tässä tutkimuksessa pyritään saamaan selville se, miksi sisäisen tietoturvan tarve havaitaan vasta usein sen jälkeen, kun jonkun asteinen vahinko on jo syntynyt. Sisäinen ja ulkoinen muodostavat yhdessä toimivan tietoturvan.

Avainsanat ja -sanonnat: Tietoturva, sisäinen tietoturva, yrityksen tietoturvaratkaisut, tietoturvan rakentaminen.
CR-luokat: K.6.5

1. Johdanto

Jokaisella tietokoneen käyttäjällä on jonkinlainen käsitys siitä, mitä tietoturva tarkoittaa. Useimmiten peruskäyttäjälle tulee mieleen virushyökkäykset ja niiden torjuntaohjelmistot ja palomuuriohjelmistot. Toinen asia, josta paljon puhutaan, ovat sähköpostin roskapostit ja niiden suodattaminen. Samoin luottamuksellisten tietojen, kuten pankkiyhteystietojen, joutuminen rikollisten käsiin koetaan vaaraksi, josta pankit kyllä hyvin tiedottavat [Kyselytutkimus, 2006]. Nämä kaikki kuuluvat kuitenkin pääsääntöisesti ulkoisen tietoturvan piiriin. Ne henkilöt, jotka ovat työssä jossain suuressa yhtiössä, joilla on varaa panostaa kokonaisvaltaiseen tietoturvaan, tietävät ainakin jotain myös sisäisestä tietoturvasta.

Sisäisen ja ulkoisen tietoturvan suhdetta on kyllä pyritty määrittelemään, mutta määrittelyissä on paikoin huomattavia eroja. Sisäisen tietoturvan käsitettä on viime vuosien aikana ryhdytty tarkentamaan ja sisäiseen tietoturvaan liittyviä ratkaisun tarjoajia on syntynyt markkinoille. Tarjotuista ratkaisuista on havaittavissa, että monet palveluntarjoajat lähtevät käsittelemään asiaa ihmisen vaikutuksesta tietoturvaavan ja tarkemmin sanottuna oman henkilöstön toiminnasta, käyttäytymisestä ja siihen liittyvistä toiminnoista [Digitoday, 2007c; Digitoday, 2007b; Laaksonen et al., 2006].

2. Tutkimuksen rakenne

Tämän tutkimuksen tarkoituksena on koota lähinnä kotimaisesta kirjallisuudesta ja lehtiartikkeleista yhteen sisäiseen tietoturvaan käsittelevät asiat; aihetta käsitellään yritystoiminnan näkökulmasta. Sisäisellä tietoturvalla tarkoitetaan tässä tutkimuksessa lähtökohtaisesti yrityksen henkilökunnan tietoturvallisuuskäyttäytymistä yrityksen toimitiloissa tai etäkäyttönä yrityksen laitteilla ja ohjelmilla. Lisäksi ne henkilöt, joilla on perusteltu syy tai mahdollisuus päästä yrityksen tiloihin, kuuluvat sisäiseen tietoturvan piiriin. Määritykseen kuuluu osana fyysinen turvallisuus ja sen suojaaminen ja sen osalta tiedon luottamuksellisuus, eheys, käytettävyys, saatavuus, todennus, pääsynvalvonta ja kiistämättömyys.

Tutkimuksen tarkoitus on löytää vastaus siihen, mitä kirjallisuuden perusteella voidaan lukea sisäiseen tietoturvaan: ovatko löydetyt asiat pelkästään sisäistä tietoturvaa ja miksi sisäiseen tietoturvaan ei kiinnitetä huomiota?

3. Sisäinen tietoturva

Tässä luvussa määritellään ja jäsennetään sisäiseen tietoturvaan kuuluvat asiat. Aihetta tarkistellaan yritystoiminnan näkökulmasta ja yritetään löytää ne asiat, jotka yritystä perustaessa tai sisäistä tietoturvaa kehittäessä tulisi ottaa huomioon. Asioita ei ole tässä yhteydessä määritellä tärkeysjärjestykseen, vaan ne on esitetty yhtenevällä prioriteetilla. Asioita on tarkasteltu suomalaisen lainsäädännön ja toimintatapojen näkökulmasta.

3.1 Tietoturvapolitiikka, -suunnitelma ja -ohjeistus

Yritystoimintaa perustaessa olisi ensimmäisten joukossa syytä luoda tietoturvasuunnitelma, joka on perusturvallisuuden toteutusta ja ylläpitoa normaalioloissa koskeva suunnitelma. Suunnitelmassa esitetään organisaation tietoturvallisuustoiminnan tavoitteet, hallinto, tehtävät ja menettelyt, osoitetaan elintärkeät tietojärjestelmät ja määritellään niiden toipumisen edellyttämät toimet [Vahti 4/2003].

Sisäinen tietoturvasuunnitelma on yrityksen ylimmän johdon ja muutaman avainhenkilön luettavissa. Sitä tulee säilyttää hyvin suojatussa tilassa ja sen päivittäminen ajan tasalle säännöllisin väliajoin on tärkeää. Suunnitelmaa luotaessa on hyvä muistaa tietoturvan perus osa-alueet tai tavoitteet:

- Luottamuksellisuus (C)
- Eheys (I)
- Saatavuus (A)
- Todennus
- Pääsynvalvonta

- Kiistämättömyys.

Kolme ensimmäistä muodostaa helpon muistisäännöllä CIA (eng. Confidentiality-Integrity-Availability), joka muodostaa peruskehyksen suunnitelmalle. Kolme muuta ovat lisääviä tai edellisiä täydentäviä, mutta samalla hyvin tärkeitä osa-alueita.[Järvinen, 2002; Miettinen, 1999]

Sisäistä tietoturvaa luodessa luottamuksellisuudella tarkoitetaan sitä, että yrityksen tietoja pääsevät lukemaan vain ne työntekijät, joilla siihen on oikeus. Oikeus on määritelty usein yrittäjän toimesta, ja hänellä on vastuu ja tietämys siitä, kuka tietoa työtehtävissään tarvitsee. Todennusta tarvitaan, koska muuten järjestelmän mahdoton tunnistaa eri käyttäjät. Lisäksi tiedon tulee olla salattu siltä varalta, että esimerkiksi tietokone joutuu fyysisesti vääriin käsiin tai yrityksen työntekijä asentaa verkkoon vakoiluohjelman. Tällöin tietoa ei saa auki ilman oikeaa salasanaa. Eheydellä tässä yhteydessä tarkoitetaan sitä, että tiedostot ovat fyysisesti ehjiä. Tämän todentaminen ja varmistaminen on melko hankalaa, koska tiedosto ei ilmoita mitään hajottuaan, vaan sen huomaa vasta silloin, kun sitä tarvitaan. Varmuuskopiointi on hyvä keino löytää tiedostosta ehjä versio, mutta sen löytäminen voi vaatia pitkän ajan varmuuskopioiden säilyttämistä. Eheyden parantamiseksi hyviä keinoja ovat laadukkaiden komponenttien käyttö ja ohjeistus siitä, että kriittisimmät tiedot tulisi tallentaa aina kahteen eri paikkaan. Tietojen saatavuudella sisäisessä tietoturvallisuudessa tarkoitetaan tietojärjestelmien ja ohjelmien toimivuuteen. Tietojärjestelmät on hyvä ajoittaa niin, että ne toimivat työntekijöille työaikana. Muuten ne ovat pois käytöstä ja aikaa hyödynnetään tiedostojen varmistukseen. Saatavuus tarkoittaa myös sitä, että työntekijällä on oikeanlaiset ohjelmat työnsä suorittamiseen. Ylimääräisten ohjelmien asentaminen on riski. Ongelmaksi voi muodostua tilanne, jossa työntekijä tarvitsee arkistoitua tietoa, joka on tuotettu 10-15 vuotta sitten. Ohjelmisto ja laitteisto eivät välttämättä enää kohtaa tai tarvittavaa ohjelmaa ei enää ole yrityksessä olemassa.[Järvinen, 2002; Korvola et al., 1992]

Luottamuksellisen ja työntekijöiden oikeisiin tietoihin pääsemisen edellyttää todentamista. Tämä tarkoittaa, että jokaiselle työntekijälle luotava henkilökohtainen käyttäjätunnus. Käyttäjätunnus on aina syytä suojata salasanalla. Todentaminen sisäisissä järjestelmissä on sinänsä haastavaa. Liikkuminen eri tiedostojen, hakemistojen, palvelimien tai ohjelmistojen välillä työpäivän aikana on hyvin arkipäiväinen ilmiö. Pääsynvalvonta liittyy tähän hyvin olennaisesti. Yrityksen on tarkasti määriteltävä, kenellä on oikeus päästä mihinkin paikkaa. Pääsynvalvonta on hyvin usein ohjelma- tai tiedostokohtaista. Yleistä on myös, että sellaisiin palvelimiin rajoitetaan kokonaan pääsy, joista työntekijä ei tarvitse mitään tietoa. Pääsynvalvontaa kuuluu oleellisesti myös käytön seuranta. Järjestelmä pitää kirjaa siitä, kuka ja koska käyttää tiedostoja tai ohjelmaa [Mäntylä ja Kajava, 1996]. Kiistettävyyteen ei sinänsä liity sisäisen tietoturvan

kysymyksiä. Sen tarve tulee esiin lähinnä sähköisessä kaupankäynnissä [Järvinen, 2002]. Näin ollen sitä ei käsitellä tarkemmin tässä tutkimuksessa.

Jokaisessa yrityksessä tulisi olla tietoturvapolitiikka-käsikirja, joka on jokaisen työntekijän saatavilla ja johon tutustuminen on pakollista työsuhteen alkaessa. Tietoturvapolitiikan käsikirja on organisaation tasolla johdon hyväksymä näkemys tietoturvallisuuden päämääristä, periaatteista ja toteutuksesta. Tietoturvapolitiikka tai -strategia on kiinteä osa organisaation toiminta- ja tietohallintopolitiikkaa tai -strategiaa [Vahti 4/2003]. Tietoturvapolitiikan käsikirjassa käsitellään mm. miksi on päädytty tiettyihin sisäisiin ratkaisuihin ja miksi se on yritykselle tärkeää. Käsikirjassa on selvitetty tietoturvallisuuteen liittyvät vastuut ja työntekijöiden valvontaan liittyvät keinot ja syyt. Tietoturvapolitiikkaan on myös monissa tapauksissa kirjattu työntekijöiden koulutustarpeet tietoturvallisuudesta ja tietotaidon saattamisesta yrityksen vaatimalle tasolle. Yrityksen sopimuskäytäntö tietoturvan suhteen määritellään käsikirjassa. Tärkeää onkin, että käsikirjaa pidetään ajan tasalla. Sen jatkuva tarkastaminen ja sen saattaminen nykypäivän säädösten ja asetusten tasalle on hyvin ratkaisevaa sen toimivuuden ja hyödyllisyyden kannalta. Käsikirjaa luotaessa on tärkeää kuitenkin muistaa, että sen tarkoitus on olla yrityksen toimintaa tukeva, ei sitä haittaava tai hidastava. [Laaksonen et al., 2006]

3.2 Fyysinen suojaaminen

Fyysisellä turvallisuudella taataan organisaatiolle häiriötön ja turvallinen toimintaympäristö [Laaksonen, 2006]. Sillä tarkoitetaan niitä kaikki toimintoja tai asioita, joihin ei järjestelmän teknisillä suojauksilla tai ratkaisulla ole vaikutusta. Fyysinen tietoturvallisuus määritellään valtiohallinnon tietoturvakäsitteistön [Vahti 4/2003] mukaan henkilöiden, laitteiden, aineistojen, postilähetysten, toimitilojen ja varastojen suojaaminen tuho ja vahinkoja vastaan. Fyysinen turvallisuus sisältää muun muassa kulun- ja tilojen valvonnan, vartioinnin, palo-, vesi-, sähkö-, ilmastointi- ja murtovahinkojen torjunnan sekä kuriirien ja tietoaineistoja sisältävien lähetysten turvallisuuden.

3.2.1 Palo-, vesi- ja ilmastointiturvallisuus

Tietokoneita ja tietojärjestelmiä on turha suojata monenlaisilla viruksentorjuntaohjelmilla ja palomuureilla ulkopuolisilta hyökkäyksiltä, jos fyysinen turvallisuus ei ole hoidossa. Fyysisen turvallisuuden uhkatekijät ovat usein sattumanvaraisia, itsestään riippumattomia ja ennusteottamattomia. Niiden riskiä voidaan kuitenkin pienentää suojaamalla toimitilat mahdollisimman hyvin ainakin seuraavilta asioilta [Laaksonen et al., 2006; Järvinen, 2002; Paavilainen, 1998]:

- Varkaus; sarjanumero ja turvamerkintä
- Tulipalo tai lämpötilan liiallinen kohoaminen
- Vesivahinko ja kosteus
- Sähköhäiriö
- Pöly
- Työntekijän inhimillinen virhe.

Kaikille näille yhteistä on, että tieto voidaan säilyttää oikeanlaisella varmuuskopioinnilla(luku 2.4). Tulipalo-, vesi-, sähkö- ja ilmastointivahinkojen mahdollisuus on järkevää suojata vakuutuksella. Onnettomuuden sattuessa yrityksellä on mahdollisuus saada kohtuullinen korvaus menetetystä omaisuudesta tai toimitilojen uudelleen rakentamiskustannuksista. Ongelmana on kuitenkin, että vaikka vakuutus ja varmuuskopio palauttaisivat kaiken menetetyn, niin usein tällaisen vahingon seurauksena on toimintojen hetkellinen keskeytyminen. Toiminnan palauttaminen voi kestää vahingon laajuudesta riippuen useita kuukausia. Tähänkin on vakuutusyhtiöillä vakuutuksia [If], mutta siinä tapauksessa vakuutusyhtiö vaatii suojaamiselta korkeaa tasoa. Sammutusjärjestelmä ja muut sammuttamistarvikkeet on oltava asianmukaisessa tilassa. Tietokoneiden ja palvelimien sijoittamisessa on otettava huomioon, että tila on hyvin eristetty ja ilmastoitu. Välipohjan rakentaminen on hyvä keino estää vesivahinkoja. Kaiken tämän rakentaminen ja hankkiminen aiheuttaa huomattavia kuluja, jonka vuoksi aloittavalla tai pienellä yrityksellä voi olla vaikeuksia löytää siihen taloudellisia resursseja. [Laasonen et al., 2006; Kainomaa, 1984]

Pölyn muodostuminen on estettävissä säännöllisellä siivouksella ja pitämällä kaiken tilat vapaana kaikesta työhön liittymättömästä tavarasta, joka tuottaa pölyä. Laitteiden oikeanlainen sijoittaminen estää pölyn muodostamista ja ehkäisee vesivahinkojen syntymistä. Sähköhäiriö on seurauksena verkkovirrassa tapahtuneesta ylijännitteestä, esim. ukkonen. Se voi pahimmassa tapauksessa aiheuttaa jopa laiterikon tai verkkovirheen koko yrityksessä. Siitä suojaaminen on helpoin hoitaa USP-laitteilla eli ylijännitesuojalla tai isompien yritysten kohdalla varageneraattorilla. [Laaksonen et al., 2006; Paavilainen, 1998]

Varkauksia vastaan ja vakuutus yhtiötä varten on hyvä kirjata kaikkien koneiden sarjanumerot ylös ja säilyttää ne hyvin suojatussa paikassa. Koneeseen näkyvälle paikalle laitettu turvamerkintä heikentää sen laitonta jälleenmyyntiä. Turvamerkintä on hyvä sijoittaa niin, että sen poistaminen tai tuhoaminen herättää muissa huomiota. [Järvinen, 2002]

3.2.2 Kulunvalvonta

Työntekijöiden ja vierailijoiden kulunvalvonta on jo siksi keskeisessä roolissa, että oikeudellinen todistettavuus kasvaa ja parantaa näin huomattavasti yrityksen tilannetta mahdollisessa kiistatilanteessa. Valvomaton kulkeminen yrityksen tiloissa on aina riskitekijä. Ilman asianmukaista kulunvalvontaa monelta

muulta sisäisen tietoturvan osalta poistuu merkitys. Jos yrityksen tiloissa pystyy liikkumaan ”jälkiä jättämättä”, niin yrityksen on mahdoton selvittää, kuka on aiheuttanut tietovuodon tai tietoriskin, vaikka itse toiminto tai tietomurto havaittaisiinkin. Paavilainen [1998] onkin määritellyt ei-toivotulle tai ulkopuoliselle tunkeutumiselle kolme erilaista varautumismenetelmää, jolla voidaan kulunvalvonnassa varautua:

1. Valtuutus (authorization) määrää, kenellä on oikeus käyttää tiloja tai järjestelmiä
2. Tunnistus (identification) on toimintaa, jossa tarkistetaan kontrollikohteen identiteetti
3. Kulun- ja yhteydenvalvonta (access control) on toimintaa, jossa varmistetaan, että ainoastaan valtuutetut henkilöt pääsevät kontrolloidulle alueelle.

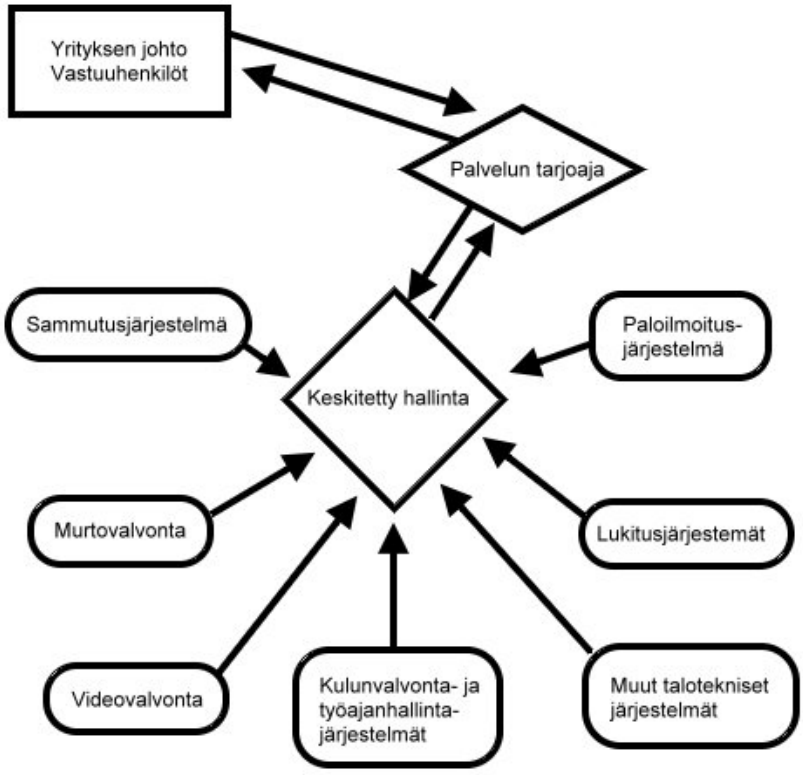
Kulunvalvonnan ei usein tarvitse olla yrityksessä huonekohtainen. Riippuen tietojen ja toimintojen arkaluontoisuudesta kulunvalvonta voi olla hyvinkin vyöhyke- eli aluekohtaista. Yrityksen tilat voidaan luokitella kuten tietoaineisto sen kriittisyyden mukaan. Näin pystytään rajaamaan esimerkiksi palvelinhuoneeseen pääsevien joukko niin, että vain siellä työskentelevillä on oikeus sinne päästä. Kulunvalvonnan keskeinen konkreettinen keino on oikeanlainen lukitseminen. Perinteisen lukot ja lukitusmenetelmät ovat jäämässä pois ja tilalle ovat tulleet sähköiset lukot, joiden turvallisuus on perinteisiä huomattavasti parempi ja joiden avulla kulunvalvontaa pystytään hallitsemaan tarvittaessa vaikka yhdeltä tietokoneelta.

Henkilöiden tunnistamiseen on vaihtoehtona useita eri ratkaisuja. Varmin keino on biometriset tunnistuskeinot [Digitoday, 2007a]. Biometrinen tunnistus on ihmisen automatisoitua tunnistusta jonkin fyysisen ominaisuuden perusteella. Tutuin esimerkki biometriasta on sormenjälkitunnistus, jossa tietokone tunnistaa ihmisen hänen sormenjälkensä perusteella. Muita biometrisen tunnistuksen menetelmiä ovat esimerkiksi kasvontunnistus, äänentunnistus ja silmän iiriksen tunnistus [Sisäasiainministeriö]. Perinteisempi keino tunnistamiseen on henkilöiden välinen tunnustautuminen, esimerkiksi yrityksen eteisessä on vastaanottovirkailija, joka tuntee kaikki yrityksessä työskentelevät henkilöt ulkonäöltä. Jos tunnustautumista ei tapahdu, niin silloin henkilön on todistettava henkilöllisyytensä virkailijalle. Sähköiset tunnustautumiset ovat nykypäivä huomattavasti yleistyneet. Toimikortit tai tunnusluvut tai niiden yhdistelmä ovat yleisimmin käytössä. Toimikortin suosiota on lisännyt siihen liitettävä ominaisuus, jolla yritys pystyy seuraamaan ja todentamaan työntekijän työajan esimerkiksi palkanmaksua varten. Toimikortin heikkoutena kuitenkin on, että se voi kadota tai se voidaan varastaa, jolloin kenellä tahansa on mahdollisuus liikkua yrityksen tiloissa toisen henkilön identiteetillä. Kuvalla ja/tai nimellä varustettu toimikortti on askel parempaan suuntaan, mutta ei silti yllä biomet-

risen tunnistamisen tasolle. Tunnusluvun huonona puolena on, että sen voi unohtaa tai oman muistamisen helpottamiseksi kirjata se ylös sellaiseen paikkaan, jossa se on yleisesti luettavissa. Tästä syystä turvallisin keino on sähköisessä tunnistamissa näiden yhdistelmä. Kulunvalvonnan hyvä apuna tai lisäominaisuutena toimii videovalvonta. Kaikki yrityksen kriittisissä paikoissa liikkuminen tallentuu videolle ja sen avulla voidaan tarvittaessa tunnistaa henkilö toimikortin väärinkäytöksen tai vastaavan sattuessa. Videolle tallentuva kulunvalvonta on hyvin usein hoidettu ulkopuolisen vartiointifirman toimesta ja siihen liittyy usein myös muita palveluja. [Paavilainen, 1998; Securitas; ISS]

3.2.3 Kokonaisvaltaiset turvallisuuspalvelut

Nykyisin vartiointiliikkeistä on tullut enenevässä määrin kokonaisvaltaisia turvallisuuspalvelun tarjoajia. Tätä täydentäväksi on myös konsultointiapua tietoturvallisuuteen liittyvissä asioissa saatavilla jopa ilmaiseksi [Digitoday, 2007d]. Yrityksellä valittavissa melkein kaikki mahdollinen, mitä fyysiseen tietoturvallisuuteen liittyy ja koota niistä itselleen sopiva paketti (ks. kuvio 1).



Kuvio1: Fyysisen tietoturvasuojaamisen palvelukokonaisuus

Pakettiratkaisun suurimpana etuna onkin sen yhdenmukaisuus ja helppous. Yritystoiminnan arkaluontoisuuden, toiminnan suuruuden, henkilökunnan

määrän ja toimitilojen sijainnin ja laajuuden perusteella on yrityksellä mahdollisuus valita sille sopivan kokonaisuuden. Yrityksellä on mahdollisuus keskittyä olennaisiin toimintoihin ja jättää suurimman osan fyysisen tietoturvan päivittäisistä asioita alan ammattilaisille.[Securitas; ISS]

3.3 Käyttäjätunnukset ja salasanat

Valtiohallinnon tietoturvakäsitteistön [Vahti 4/2003] mukaan käyttäjätunnus tarkoittaa yksilöitävää tunnistetta, jonka käyttäjän on sisäänkirjautumisen yhteydessä ilmoitettava. Käyttäjätunnus luodaan henkilön aloittaessa yrityksessä ja se tuhotaan, kun työsuhde loppuu. Käyttäjätunnuksen keskeisin tehtävä tunnistautumisen lisäksi on käyttöoikeuksien ja pääsynvalvonnan hallinta. Tämä on hyvin tärkeää, kun työntekijä liikkuu yrityksen sisäisessä verkossa ja sisäisen verkon kautta esimerkiksi internetiin [Kajava ja Remes, 2000]. Sen avulla on siis yrityksellä mahdollisuus määritellä jokaiselle työntekijälle ne oikeudet, joita hänen työtehtävänsä vaativat.

Käyttäjätunnus toimii tarvittaessa tietojärjestelmissä sähköisenä sormenjälkenä. Yrityksellä on mahdollisuus vaikka dokumenttitasolla määritellä loki. Lokilla tarkoitetaan tiedostoa, johon tehdään aikajärjestyksessä merkinnät tapahtumista ja niiden aiheuttajista. Loki kerätään yleensä automaattisesti ja samaan järjestelmään liittyviä lokeja voi olla useita, esimerkiksi vikaloki ja laskutusloki [Vahti 4/2003]. Näin ollen tarpeen tullessa on mahdollista esimerkiksi selvittää, kuka on tehnyt järjestelmään muutoksia, ketkä ovat käyneet lukemassa ko. tiedoston tai kuka on kopioinut tiedostot omaan käyttöönsä. Loki tiedostot ovat melkein aina piilotettuja normaalikäyttäjiltä ja muutosten tekeminen tai niiden tuhoaminen on mahdotonta.

Käyttäjätunnukset voidaan jakaa karkeasti kahteen ryhmään käyttöoikeuksien perusteella. Järjestelmän käyttöön tarkoitetut tunnukset ovat melkein jokaisella yrityksen työntekijällä ja johtajalla. Eri henkilöiden välillä toki on järjestelmä-, ohjelmisto- ja dokumenttikohtaisia eroja, ja ne muuttuvat aina työtehtävien ja sen vastuullisuuden mukaan, mutta käyttäjätunnuksen perusluonne on aina sama. Toinen ryhmä ovat ylläpitotunnukset. Ne ovat yrityksessä yleensä vain muutamalla avainhenkilöllä. Nämä henkilöt ovat yleensä tietohallinnon työntekijöitä tai yrityksen tieturvasta vastaava henkilö. Heillä on kokonaisvaltainen kuva yrityksen tietoteknisistä ja tietostrategisista ratkaisuista. Käyttäjätunnuksen luominen uudelle henkilölle on aina ylläpitotunnuksen omaavan henkilön tehtävä. Ilmoitus uuden tunnuksen luomisesta tulee aina tehdä kirjallisesti ja siinä on oltava esimerkiksi esimiehen allekirjoitus. Hyväksytty lomake on aina standardimuotoinen ja se tulee säilyttää niin kauan kuin työntekijä on yrityksen palveluksessa. Ylläpitotunnukset voivat olla myös ohjelmisto- tai järjestelmäkohtaisia. Varsinkin tässä tapauksessa sen suojaaminen ja suojeleminen on ensiarvoista, sillä nämä tunnukset eivät ole kenenkään henkilön nimellä ja

tunnistaminen voi olla sen myötä mahdotonta. Yleensäkin ylläpitotunnuksen ovat suuremman suojauksen ja salauksen tarpeessa, koska niiden joutuminen vääriin käsiin yleensä mahdollistaa vapaan liikkumisen yrityksen koko tietoverkossa.

Salasana on jokaiseen käyttäjätunnukseen liittyvä yksityinen, käyttäjän tiedossa oleva merkkijono. Tunnusta luodessa se on ylläpidon kanssa ennalta sovittu sana tai numerosarja, mutta käyttäjän ensimmäisen käynnistämisen yhteydessä järjestelmä vaatii sen vaihtamista.

Salasanoja luodessa on hyvä muistaa tiettyjä tietoturvallisuuteen liittyviä perussääntöjä. Salasanan yksityisyyden varmistamiseksi se on hyvä vaihtaa 3-4 kuukauden välein. Tämä jo siitä syystä, että verkon vakoiluohjelmat voivat kerätä erilaisia kirjautumis- tai lokitietoja, ja vaikka salasana näissä tiedoissa olisi suojattu, niin toistuvuus voi antaa salasanan selvittäjälle tarpeettoman edun. Vakoiluohjelmia voi asentaa ulkopuoliset tahot, esimerkiksi siivousfirman työntekijät, tai oman yrityksen epärehelliset työntekijät. Salasanan säilyttämistä ei koskaan suositella työpisteessä tai missään muussa paikassa, missä siihen voi olla jollakin toisella ihmisellä pääsy. Sitä ei missään olosuhteessa tule luovuttaa kenellekään, vaikka kyseessä olisi esimerkiksi pitkäaikainen työnkaveri. Se muodostaa tarpeettoman tietoturvariskin, jolla ei ole siihen rinnastettavaa hyötyä missään tilanteessa. Onkin suositeltavaa, että salasana olisi ainoastaan henkilön omassa päässä ja kirjaamista minnekään vältetään. [Järvinen, 2002; Miettinen, 1999]

Salasanan muodostaminen ei ole helppoa. Sen keksiminen sellaiseksi, että se olisi helppo muistaa, mutta samalla se olisi muille mahdollisimman vaikea arvattava, on haaste, johon työntekijät tarvitsevat apua ja ohjeistusta. Yrityksille salasanoja hallinnoivat ohjelmat ovat kehittyneet ja ne itsessänsä tekevät perustarkistuksia ennen kuin ne hyväksyvät käyttäjän salasanan. Salanoja muodostaessa ja niitä käyttäessä onkin hyvä muistaa Miettisen [1999] luoma tarkistuslista:

- ✓ Tietojärjestelmien salasanat ovat henkilökohtaisia. Pidä ne omana tietona, äläkä kerro niistä kenellekään ulkopuoliselle
- ✓ Käytä riittävän pitkiä salasanoja. Salasanan on oltava vähintään kahdeksan merkkiä pitkä, jotta sitä ei ole helppo arvata tai muuten selvittää
- ✓ Käytä vaikeasti arvattavia salasanoja. Salasanan tulee muodostua kirjaimista, numeroista ja erikoismerkeistä, mikäli tietojärjestelmä tämän hyväksyy
- ✓ Vaihda salasana riittävän usein. Sopiva vaihtoväli on muutaman kerran vuodessa. Mikäli epäilet salasanasi joutuneen ulkopuolisen haltuun, vaihda se välittömästi.

- ✓ Vältä salasanan kirjoittamista paperille. Jos se on aivan välttämätöntä, varastoi paperille kirjoittamasi salasanat sellaiseen lukittuun paikkaan, johon ulkopuoliset eivät pääse.
- ✓ Kun kirjaudut ensimmäistä kertaa sisään uuteen järjestelmään, vaihda käyttöösi annettu oletussalasana välittömästi
- ✓ Jos järjestelmä mahdollistaa käyttäjätunnuksen ja/tai salasanan varastoinnin järjestelmän muistiin. älä käytä tätä mahdollisuutta. Järjestelmän muistiin tallennetun käyttäjätunnuksen ja salasanan avulla kuka tahansa voi kirjautua sisään järjestelmään sinun käyttöoikeuksillasi.
- ✓ Jos käytät järjestelmää vieraan tietokonelaitteen kautta, varmistu, että kyseinen järjestelmä ei varastoi salasanaasi laitteen muistiin.
- ✓ Kun kirjoitat salasanaa, huolehdi siitä, että kukaan sivullinen ei näe sitä.

Muistilista on hyvä kirjata yrityksen tietoturvapolitiikkaan, jossa se on kaikkien työntekijöiden luettavissa. Uusien työntekijöiden kohdalla on aina hyvä velvoittaa lukemaan salasanaa koskeva ohjeistus tietoturvallisuuden takaamiseksi.

3.4 Varmuuskopiointi

Varmuuskopio on taltion, tiedoston tai ohjelman kopio, joka on tarkoitettu käytettäväksi, jos alkuperäinen menetetään vian tai vahingon takia [Vahti 4/2003]. Varmuuskopiointi on keino varmistaa, että varkauden, vahingon tai onnettomuuden sattuessa tiedoista ja ohjelmista on olemassa kopio. Sen tehtävänä on säilyttää tietoja ja ohjelmia useammassa kuin yhdessä paikassa ja useammalla kuin yhdellä tavalla. Varmuuskopiointiin ei ole olemassa universaalia sääntöä, vaan se riippuu siitä, kuinka vitaalia tai arkaluontoista tieto on. Kun ollaan tekemisissä esimerkiksi ihmisten henkilötietojen kanssa, niin varmuuskopiointi on lainsäädännössä määriteltyä. Jo enemmän kuin päivittäistä varmuuskopiointia tarvitaan esimerkiksi pankkialalla, koska kyseessä on toisen omaisuuden säilyttäminen ja sen todentaminen ilman varmuuskopiota on mahdotonta. Toki tällaisissa tapauksissa rinnakkaisia järjestelmiä on olemassa, mutta kaiken viimeisempänä keinona pitää olla eri paikassa säilytetty ja ajan tasalla oleva varmuuskopiointi [Digitoday, 2007b].[Miettinen, 1999]

Varmuuskopiointi on hyvin usein automatisoitu. Se ei vaadi käyttäjän jatkuvia toimia, vaan toimii annettujen speksien perusteella automaattisesti ja vaatii vain käyttäjän toimia virhetilanteissa. Jos varmuuskopiointi tapahtuu esim. CD-levylle, niin käyttäjän tehtäväksi jää vain levyn oikeanlainen säilyttäminen. Paavilaisen[1998] mukaan varmuuskopiointi voidaan jakaa kuuteen eri varmuusmenetelmään:

- Täydellinen varmuuskopiointi

- Kasvavan vedostuksen menetelmä
- Useiden versioiden menetelmä
- Varovaisen korvaamisen menetelmä
- Differentiaali tiedostoista
- Pelastusohjelma.

Täydellisessä varmuuskopioinnissa otetaan kaikista tiedostoista kokonaisuudessaan kopiot. Tämä menetelmä vaatii suuren tallennuskapasiteettia ja sitä ei käytetä yrityksissä, joissa on monta erillistä tietokantaa tai kannat ovat itsessänsä suuria. Kasvavan vedosten menetelmässä toimitaan aluksi samoin kuin edellisessä, mutta sen jälkeen varmuuskopiosta poistetaan ne tiedostot, jotka eivät ole muuttuneet edelliseen versioon verrattuna. Tämä on hyvä menetelmä juuri isoissa tietokannoissa. Useiden versioiden menetelmässä tietokannasta on useita yhtäaikaisia varmuuskopioita. Uuden kopiota ottaessa se ensin päivittyy yhteen paikaan täydellisesti ja vasta onnistuneen kopioinnin jälkeen siirtyy seuraavaan paikkaa. Periaatteessa tämä on sama asia kuin jatkuva varmuuskopiointi eli peilaus. Tämä on hyvä menetelmä mm. pankkialalla. Virhetilanteissa on tärkeä selvittää kopioiden virheettömyys. Varovaisen korvaamisen menetelmässä päivityksiä ei tehdä suoraan tietokantaan vaan tietokannan tai sen osan kopioon ja kun kaikki tarvittavat päivitykset on tehty, niin kopio korvaa varsinaisen varmuuskopion. Differentiaalin ottaminen tiedostoista tarkoittaa sitä, että kopio tehdään työtiedostoihin ja ne yhdistetään aika ajoin varsinaiseen kantaan. Viimeisenä menetelmänä on pelastusohjelman toteuttaminen ja sitä käytetään vasta silloin kun muut toipumiskeinot eivät auta. Se vaatii tietokannan tai järjestelmävastaavan manuaalisia toimenpiteitä ja hyvin korkeaa ammattitaitoa.

Monien vaihtoehtojen johdosta yrityksellä on mahdollisuus valita itselleen sopiva ja toiminnan tarpeisiin suhteutettuna järkevän varmuuskopiointimenetelmän. Kuten aiemmin jo on mainittu, varmuuskopio on ainut keino palauttaa tiedot vakavan onnettomuuden tai muun vastaavan vahingon jälkeen. Varmuuskopiot sisältävät yrityksen sellaista tietoa, mitä ei ole taloudellisesti mitattavissa ja niiden menettämistä ei pysty vakuutuksilla kattamaan.

3.5 Henkilöstöturvallisuus

Henkilöstöturvallisuus on henkilöstöön liittyvien tietoturvariskien hallintaa muun muassa toimenkuvien, käyttöoikeuksien ja koulutuksen avulla [Paavilainen, 1998]. Henkilöstöturvallisuus on yksi yritysturvallisuuden monista itsenäisistä osa-alueista. Sillä on kuitenkin monia yhtymäkohtia tietoturvallisuuteen. Tässä yhteydessä sillä henkilöturvallisuudella tarkoitetaan tietoturvallisuuden osa-aluetta, jossa tarkastellaan yrityksen tietojen ja tietojenkäsittelyn suojaamista ihmisten tahallisilta tai tahattomilta uhilta sekä ihmisten toimintaa tietoturvan varmistamiseksi[Miettinen, 1999]. [Kajava ja Leiwo, 1994]

3.5.1 Henkilöstön tietoturvakouluttaminen

Koulutuksen tavoitteena on tuoda henkilöstön tietoon se, millä tavalla yritys näkee tietoturvallisuuden ja motivoida työntekijöitä työskentelemään tehokkaammin yrityksen toiminnan edistämiseksi. Tehokkaan koulutuksen tulee olla jatkuvaa ja laadukasta siltä osin kuin se koskettaa työntekijän omaa työnkuvaa ja siihen liittyvää tietoturvaa. Koulutuksen tehokkuus riippuu siitä, kuinka hyvin henkilöstö motivoitunut [ITviikko, 2007a]. Motivaatiota voi olla kahdenlaista: sisäistä ja ulkoista [Laaksonen et al., 2006]. Sisäisellä tarkoitetaan sitä, että työntekijällä on aito innostus tietotekniikkaa kohtaan ja harrastuneisuus alaa kohtaan on työsuhdetta aiemmin opittua. Tällainen motivaatio on yrittäjälle tärkeää, koska se yleensä tarttuu muihin työntekijöihin. Ulkoisella motivaatiolla tarkoitetaan, että työntekijällä on luonnostaan kova kilpailuvietti, joka ilmentyy työnpaikalla siten, että keskinäinen kilpailu lisää tehokkuutta. Ulkoinen motivaatio usein tukee voimakkaasti sisäistä motivaatiota. [Laaksonen et al., 2006]

Onnistuneen koulutuksen pitää olla myös vaihtelevaa ja kouluttaja tulee olla sellainen, että työntekijä kokee, että koulutuksen tarkoitus on tukea ja vahvistaa opittuja taitoja tai tuoda täysin uutta tietoa alalta. Se ei siis saa olla sellaista, että työntekijä kokee koulutuksen saman kertaamiseksi ja jankuttamiseksi. Esimiehen tai yrittäjän tulee olla mukana vahvasti koulutustilauksissa ja koulutuksen tulee olla kaikille mahdollista.[Laaksonen et al., 2006]

3.5.2 Uuden työntekijän taustojen tarkistus

Uuden työntekijän palkkaamiseen liittyy aina tietoturvallisuusriski. Uuden työntekijän tullessa yritykseen, on aina tarkistettava työntekijän taustat. Taustojen tarkistus on yrityksen koosta riippumatta usein liian kevyttä ja yleisesti ei tiedosteta, mitä kuuluu taustojen tarkistukseen. [Miettinen et al., 2006]

Ilmaisia tarkistus keinoja on yleisten taustatietojen etsiminen. Helpoin tapa on etsiä internetistä, jossa henkilön aikaisemmista tekemisistä on merkintöjä. Ansioluettelon tarkistaminen on myös hyvin keskeisessä roolissa. Sen oikeellisuus ja kerrottujen referenssien tarkistaminen konkreettisesti esimerkiksi puhelimitse auttaa saamaan tietoa siitä, miten työntekijän aikaisempi työsuhde on löytynyt ja miten työntekijä on työnsä hoitanut. Maksullisia keinoja taustojen selvittämiseen ovat luottotietojen tarkistaminen ja suojelupoliisilla teetetty turvallisuusselvitys, jonka saaminen pk-yrityksen rekrytointitilanteessa on hyvin harvinaista. [Laaksonen et al., 2006; Miettinen et al., 2006]

3.5.3 Työsopimus ja työsuhteen loppuminen

Yrityksen ja työntekijän välinen työsopimus tulee aina olla kirjallinen. Siitä tulee minimissään olla yrityksen ja työntekijän tiedot, työsopimuksen laatu, lyhyt

kuvaus työtehtävästä ja palkanmaksun perusteet. Tietoturvallisuuteen liittyvissä työtehtävissä on hyvin usein myös lisätty työsopimukseen salassapitosopimus ja immateriaalisten oikeuksien omistus. Mitä tarkemmin työsopimuksessa asioita, tehtäviä, toimenkuvia ja oikeuksia on määritelty, sitä vahvemmalla työnantaja on, jos kiistatilanteita syntyy ja asioita joudutaan käsittelemään jopa oikeudessa. [Miettinen et al., 2006; ITviikko, 2007b]

Työ suhteen loppumiseen liittyvät asiat on järkevää selvittää työsopimuksessa. Silloin työntekijä tietää jo työsuhdetta aloittaessa, miten hänen tulee toimia työsuhteen loppuessa ja mihin tulee sitoutua. Tästä keskeisenä esimerkkinä on käyttäjätunnuksen ja salasanan luovuttaminen työnantajalle. Työnantaja samalla sitoutuu tuhoamaan kaikki työntekijän henkilökohtaiset ja henkilötiedot. [Laaksonen et al., 2006]

Alihankkijoiden työntekijöillä tarkoitetaan sellaisia yrityksen tiloissa liikkuvia henkilöitä, jotka eivät ole suorassa työsopimussuhteessa yritykseen. Näiden henkilöiden kohdalla on kuitenkin syytä noudattaa suurilta osin samoja etoja ja määräyksiä kuin omien työntekijöiden kohdalla. Esimerkiksi siivoojalla, vartijalla, postin jakajalla tai tavarantoimittajilla on täysin sama vaitiolovelvollisuus kaikesta näkemästään yrityksen tiloissa kuin omilla työntekijöillä.

3.5.4 Avainhenkilöriskit

Avainhenkilöllä tarkoitetaan pk-yrityksessä useimmiten yrittäjää itseään, samoin kuin muita henkilöitä, jotka hoitavat tehtäviä, mitkä ovat elintärkeitä yrityksen liiketoiminnan laajuuden, kannattavuuden ja jatkuvuuden kannalta. [If; Vahti 4/2003]

Avainhenkilöriskillä tarkoitetaan puolestaan sitä, mitä yritykselle tulee maksamaan, jos avainhenkilön työpanos yllättäen menetetään, joko tilapäisesti tai pysyvästi. Suoranaisten katemenetysten lisäksi on huomioitava myös uuden henkilön rekrytointikustannukset, perehdyttäminen, vajaa tuottavuus ja muun organisaation ajanmenetys sekä ylityökustannukset.[VTT; Kajava et al., 1996]
Tärkeä osa avainhenkilöriskin hallintaa on varahenkilöjärjestelmä, tulee olla harkittu ja jopa käytännössä testattu. Avainhenkilön vastuuseen kuuluu, että hän pitää varahenkilönsä ajan tasalla mahdollisen riskin toteutumisen varalta. Pk-yritysten ohuen organisaation takia varahenkilöjärjestelmä on niissä usein käytännössä hatara.[VTT; Kajava et al., 1996]

Avainhenkilökatastrofilla puolestaan tarkoitetaan riskiä, jossa menetetään kahden tai useamman avainhenkilön työpanos samalla kertaa. Näin voi käydä esimerkiksi liikenne- tai lento-onnettomuudessa. Yrityksellä tulisikin olla matkustussääntö, jonka mukaan usea avainhenkilö tai hänen varahenkilönsä ei saa matkustaa samassa liikennevälineessä.

3.5.5 Sisäisen tietoturvan valvonta

Sisäisellä valvonnalla varmistutaan siitä, että tietoturvapolitiikkaa noudatetaan ja annettuja tietoturvaohjeita sovelletaan työnteossa. Tämän lisäksi valvonnalla ylläpidetään ja tarkkaillaan niitä uhkia, jotka tiedetään olevan yrityksen sisäisessä tietoturvassa ja täyttävät tietoturvarikoksen tunnusmerkit [Leiwo ja Kajava, 1994]. Sisäinen valvonta pitää sisällään myös sen, että työntekijöiden toimintaa yrityksen järjestelmissä ja liikkumista sisäisessä tietoverkossa tarkkaillaan. Työntekijän normaalista poikkeavaan toimintaan pystytään reagoimaan hyvin nopeasti ja sen myötä tahalliset tai tahattomat tietovuodot pystytään estämään. [Pirnes et al., 2000]

Tällaisen tietoturvajohtamiseen on olemassa hyviä valvontatyökaluja. Niiden ei aina tarvitse olla kalliita suljettuja ratkaisuja, vaan esimerkiksi avoimen lähdekoodin ratkaisut ovat hyvin yleisiä tämän osalta. Valvonta tulee suorittaa aina niin, että se haittaa mahdollisimman vähän työntekijän päivittäisiä rutiineja, mutta on samalla tehokasta ja reagoivaa. Jos sen vaikutus näkyy, on työntekijöille korostettava, että tarkoituksena ei ole turha kyttääminen. [Laaksonen et al., 2006]

4. Johtopäätökset

Sisäiseen tietoturvaan toimintojen määrittäminen ei ole todellakaan helppoa. Koska mukana on paljon ihmisen toimintoja, niin asiat voidaan nähdä monesta näkökulmasta. Tämän voi myös havaita kirjallisuutta tutkiessa, sillä jokainen tunnisti kyllä sisäisen tietoturvan käsitteen, mutta niiden sisältö oli jokseenkin erilainen. Käsitettä oli myös käsitelty hyvin yleisellä tasolla ja se jätti paljon tulkitsemisen varaa. Myös aiheeseen sopivan kirjallisuuden löytäminen oli työlästä ja se pääsääntöisesti todella niukkaa.

Johdannossa esitetty määritelmä sisäisestä tietoturva osoittautuikin mielestäni hyvin sopivaksi. Kaikki asiat mitä sisäisen tietoturvan yhteydessä on mainittu kirjallisuudessa, sopivat ainakin soveltaen määritelmään. Tulkinnanvaraista tässä onkin se seikka, että monet tässä esitetyt asiat ovat seurausta ulkoisen tietoturvan kehittämisestä tai toimiakseen vaativat varmasti myös ulkoisen tietoturvan vahvaa olemassa olemista. Tietoturvan osa-alueet ovat riippuvaisia hyvin monessa tapauksessa toisistaan ja yksitellen käsiteltynä ovat lähes hyödyttömiä. Tässä kohtaan taas tullaan siihen kysymykseen, että kumpi on syy ja kumpi on seuraus. Mielestäni tämän tyylinen ajatus on vaarallista tietoturvan kohdalla, koska seurauksiin ei yleensä kiinnitetä huomiota, jos syytä ei havaita. Kokonaisuuden parantamiseksi onkin mielestäni selkeää, että jaottelu sisäiseen ja ulkoiseen on olemassa, koska se auttaa tietoturvaratkaisun kehittäjää selkeyttämään asioita ja useammat asiat tulevat esiin ennen kuin ne aiheuttavat vahinkoa.

Kuten aiemmin on jo todettu, yrityksen henkilöstö on tavalla tai toisella hyvin merkittävässä roolissa, kun puhutaan sisäisestä tietoturvasta. Sisäisen tietoturvan pettäminen voi olla seurausta tietämättömyydestä, huolimattomuudesta tai tahallisuudesta. Henkilöstön hallinta ja valvonta on hyvin vaikea hoitaa niin, että se itsessään ei anna aihetta uusien uhkien syntymiselle. Tärkeintä onkin, että asiat ja ratkaisut esitetään mahdollisimman selvästi ja avoimesti ja ne perustellaan aina henkilöstölle. Kun kyse on ihmisen toiminnasta ja ominaisuuksista on täysin selvää, että riskiä ei pystytä täysin sulkemaan pois millään keinoilla. Riskejä pystytään kuitenkin merkittävästi pienentämään ja niihin pystytään varautumaan niin, että negatiivinen vaikutus vahingon sattuessa liiketoimintaan on mahdollisimman pieni.

Kaksi keskeistä syytä sisäisen tietoturvan puutteellisuuteen ovat tietotaidon puute ja taloudelliset resurssit. Mitä pienempi yritys on, todennäköisemmin vastuu tietoturvasta jaettu useiden ihmisten kesken ja sitä hoitaa aina se henkilö, jolla omilta työtehtäviltä siihen jää aikaa. Tämä aiheuttaa sen, että kukaan ei hallitse kokonaisuutta ja kenelläkään ei ole kompetenssia hallita tietoturvallisuutta siinä määrin kuin se yrityksessä kuuluisi olla. Taloudellinen este tietoturvan kehittämiselle tulee useissa tapauksissa kysymykseen. Monen suomalaisen pk-yrityksen toiminta on sen kokoista, että yksinkertaisesti ei ole mahdollisuutta palkata tai ostaa tietoturvaosaamista. Sisäisen tietoturvan kehittäminen on osaltansa halpaa tai jopa ilmaista, mutta mukana on myös asioita, joiden oikeanlainen toteuttaminen on huomattava taloudellinen kustannus. Kun taloudelliset resurssit ovat rajalliset ja tietotaito on keskinkertaista, niin tällaisissa tapauksissa toiminnot ja ratkaisut kohdistetaan juuri ulkoiseen tietoturvaan kuten viruksen torjuntaan ja palomuuriin.

Tässä tutkimuksessa ei olla halutta ottaa kantaa siihen, kumpi näiden määrittelyjen perusteella on tärkeämpi tietoturvan osa-alue. Uskonkin, että siihen ei ole yhtä oikeaa vastausta olemassa. Tärkeintä onkin muistaa, että tietoturva – sisäinen ja ulkoinen – on hyvin keskeistä yritystoiminnassa ja sen pettäminen voi pahimmassa tapauksessa aiheuttaa yritystoiminnan loppumisen – oli se sitten kilpailevan yrityksen hakkeri tai oman yrityksen työntekijä, joka varasti kaikki yrityksen tiedot ja perusti oman kilpailevan yrityksen.

Viiteluettelo

[Digitoday, 2007a] Digitoday verkkolehti, Sveitsiläispankki luottaa yksinomaan biometriaan tunnustautumisessa, 8.5.2007, http://www.digitoday.fi/_page.php?_page_id=9&news_id=200711117 (Luettu 15.5.2007).

[Digitoday, 2007b] Digitoday verkkolehti, Deferon pk-yrityksille: varokaa vanhoja nauhoja, 2.4.2007, http://www.digitoday.fi/page.php?page_id=14&news_id=20078175 (Luettu 15.5.2007).

[Digitoday, 2007c] Digitoday verkkolehti, Tyytymätön työntekijä on vaaraksi yritykselle, 27.4.2007, http://www.digitoday.fi/page.php?page_id=14&news_id=200710342 (Luettu 15.5.2007).

[Digitoday, 2007d] Digitoday verkkolehti, HP konsultoi pk-yrityksiä ilmaiseksi, 14.3.2007, http://www.digitoday.fi/page.php?page_id=50&news_id=20076464 (15.5.2007).

[If] If Vahinkovakuutusyhtiö Oy, 2007, http://www.if.fi/web/fi/commercial.nsf/noframes/0B68543550363CA1C1256CB7004A4AE4 (Luettu 27.4. 2007)

[ISS] ISS palvelut Oy, 2007, http://www.fi.issworld.com/view.asp?ID=1381 (Luettu 15.5.2007).

[ITviikko, 2007a] ITviikko verkkolehti, IT-opetus jää työkaverin harteille, 25.4.2007, http://www.itviikko.fi/page.php?page_id=46&news_id=200710046&rss=18 (Luettu 15.5.2007).

[ITviikko, 2007b] ITviikko verkkolehti, Käräjäoikeus kielsi rekrytoinnin SysOpenista, 11.4.2007, http://www.itviikko.fi/page.php?page_id=46&news_id=20078684&rss=18 (Luettu 15.5.2007).

[Järvinen, 2002] Petteri Järvinen, *Tietoturva & Yksityisyys*, WS Bookwell, Porvoo, 2002.

[Kainomaa, 1984] Seppo Kainomaa, *Tietoturva sekä Vahingot ja Väärinkäytökset ATK:ssa – Yleiskatsaus ja yhteenveto suoritetusta organisaatiokyselystä*, Tietotekniikan Kehittämiskeskus Ry., Helsinki, Helmikuu 1984.

[Kajava et al., 1996] Jorma Kajava, Sami J.P Heikkinen, Paavo Jurvelin, Tero Viiru ja Päivi Parviainen, *Tietojenkäsittelyn ulkoistaminen ja tietoturva – Information Security Research from Information Processing Outsourcing*, Working Papers Series B 42, Oulun yliopisto, Toukokuu 1996.

[Kajava ja Remes, 2000] Jorma Kajava ja Timo Remes, *Inranet Security from Organizational Point of View*, Working Papers Series B 59, University of Oulu, March 2000.

[Kajava ja Leiwo, 1994] Jorma Kajava ja Jussipekka Leiwo, *Tietoturvahenkilöstö oraganisaatiossa – Information Security Staff in Organizations*, Working Papers Series B 34, Oulun yliopisto, Joulukuu 1994.

[Korvola et al., 1992] Kaarlo Korvola, Teemupekka Virtanen, Rauli Parmes, Aulis Gerlander, Heikki Hietanen, Erkki Leiviskä, Kari Suvila ja Matti Tenhunen, *Valtiohallinnon tietoturvallisuuspäätös 6/1992*, 1992.

[Laaksonen et al., 2006] Mika Laaksonen, Terho Nevasalo ja Karri Tomula, *Yrityksen tietoturvakäsikirja – Ohjeistus, toteutus ja lainsäädäntö*, Oy Nordprint Ab, Helsinki, 2006.

[Leiwo ja Kajava, 1994] Jussipekka Leiwo ja Jorma Kajava, *Tietojenkäsittelyn Varmistaminen Yrityksen Turvallisuusjärjestelyjen Osana*, Working papers series B 31, Oulun yliopisto, Syyskuu 1994.

[Miettinen, 1999] Juha E. Miettinen, *Tietoturvallisuuden johtaminen – näin suojaat yrityksesi toiminnan*, Kauppakaari Oy, Helsinki, 1999.

[Miettinen et al., 2006] Tarmo Miettinen, Anssi Keinänen ja Miia Laukkanen, *Pientyönantajan työoikeudelliset ongelmat ja neuvontapalvelut,* Helsinki, 2006.

[Paavilainen, 1998] Juhani Paavilainen, *Tietoturva*, Gummerus, Jyväskylä, 1998.

[Pirnes et al., 2000] Jari Pirnes, Anssi Salmela ja Jorma Kajava, *Tietoturva ja Sisäinen Valvonta – Information Security and Internal Control*, Working papers series B 62, Oulun yliopisto, Marraskuu 2000.

[Securitas] Securitas Oy Turvallisuuspalvelut, 2007, https://www.securitas.fi/pages/secwebfi.nsf/sp?open&cid=Home (Luettu 15.5.2007).

[Vahti 4/2003] *Valtionhallinnon tietoturvakäsitteistö*, Valtionvarainministeriö, Helsinki, 2003 http://www.vm.fi/vm/fi/04_julkaisut_ja_asiakirjat/_01_julkaisut/05_valtionhallinnon_tietoturvallisuus/50903/50902_fi.pdf (Luettu 15.5. 2007).

[VTT] Pk-yrityksen henkilöriskit, 2000, VTT Automaatio, Turun kauppakorkeakoulu, Työterveyslaitos ja Tampereen teknillinen korkeakoulu(PK-RH hanke), http://www.pk-rh.fi/ftp/kalvot/kal-henkiloriskit.pdf (Luettu 15.5. 2007).

Java framework for WiFi-based Nabaztag device

Juha-Pekka Rajaniemi

Abstract

This paper is an introduction to Nabaztag/tag device and jNabServer framework that I have developed and published under LGPL license. Nabaztag/tag is second version Nabaztag that has been developed by French company named Violet. Violet is offering service to use with Nabaztag, but it has its flaws and restrictions due to commercial nature of the service. jNabServer is an alternative for Violet's services. You find a detailed explanation how Nabaztag works on low level and an introduction to start building your own jNabServer plugins.

Keywords: Nabaztag/tag, jNabServer
CR Categories: J.2, D.1.5, D.2.m

1. Introduction

jNabServer is an open-source server software which handles communication with Nabaztag/tags (Nabaztag, 2007), Wi-Fi enabled robotic rabbits that can be used as a physical user-agent in human-computer interaction.

jNabServer is written in j2SE 6.0 (Java Standard edition), so it is platform independent. The main idea is that people can develop their applications more freely on jNabServer platform by removing the need for connection to Violet's services as these services has no possibility of operating fast enough from all possible locations. This can be seen in an improvement from tens of minutes on Violet servers to milliseconds on local jNabServer at best cases. Also jNabServer opens a lot of possibilities as there is no need to have connection to the internet all the time.

Nabaztag/tag device can handle various forms of interaction, from voice to touch (button press), and from RFID 'sniffing' to ear movements. It can response by moving its ears, by displaying or changing the colour of its four LED lights. It can also play sounds which can be music, synthesized speech or other voices.

This paper is divided into two parts. First, I will describe Nabaztag/tags protocol and formats. Second, Chapter 3 contains information about jNabServer and examples of building simple jNabServer.

jNabServer has been produced at the University of Tampere (Department

of Computer Sciences, Speech-based and Pervasive Interaction Group). Tampere Unit for Computer-Human Interaction group hired me as intern to do research on Nabaztag communication protocol and develop a Java framework to operate Nabaztag/tags in local network. From this point onwards in this document I refer Nabaztag/tag as Nabaztag.

2. Basics of Nabaztag

Nabaztag receives code on its every boot. We call it the byte code and it is used to operate all internal actions what Nabaztag does. At the time of writing there is no tools released to modify these operations. We are explaining in detail how byte code operates inside Nabaztag. All this information is based on testing and there is still a lot of unknown issues. We are mostly targeting this documentation on communications in byte level.

Nabaztag has five leds, four in front and one in bottom, two moving ears which positions are tracked with optical sensors, a microphone, a speaker and an rfid-reader.

The RFID-reader reads tags of type ISO 14443 B (Dantan, 2007). These tags operate on 13,56mhz and has 4096-bit EEPROM. Nabaztag can read ID's of these tags.

Sounds recorded from microphone are encoded with IMA ADPCM (CCITT, 1990) before they are sent to server for processing.

When Nabaztag connects to a server for the first time after it has been powered on or rebooted it calls server to have send byte code to it. Byte code contains everything what Nabaztag needs to operate: what to do when a user input comes to device and how to operate the data coming from the server.

On the basic network configuration Nabaztag is working as a HTTP network client, which automatically ping calls the server with fixed interval. The interval can vary and it can be changed by sending PingIntervalBlock in the packet. Nabaztag can request the following five files from server: bc.jsp, locate.jsp, p4.jsp, record.jsp and rfid.jsp. These calls are hard coded in to the byte code. Nabaztag calls following files from server spontaneously or after received user input. (In these requests I'm using server IP 192.168.0.2 as an example):

1. GET http://192.168.0.2/vl/bc.jsp

After booting Nabaztag starts calling bc.jsp until it receives and processes the correct byte code from the server that has been set-up in configuration. Parameters for bc are shown in Table 1.

Parameter	Description
v	Version of Nabaztag
m	Mac address of Nabaztag (serial)
l	Unknown Mac Address (full of zeroes)
p	Unknown Mac Address (full of zeroes)
h	unknown

Table 1. The parameters for bc.jsp

2. GET http://192.168.0.2/vl/locate.jsp

When Nabaztag has successfully loaded the byte code it first calls locate.jsp which has to tell two IP's for Nabaztag. The first IP is the pingserver from where p4.jsp is called. The second value is broadcast server value. These value are separated with a newline character. Nabaztag will replace all "broadcast" strings from the Message Block with given IP when Nabaztag receives a message. The parameters used with locate.jsp are shown in Table 2.

Parameter	Description
sn	Serial of sending nabaztag
h	unknown

Table 2. The parameters for locate.jsp

3. GET http://192.168.0.2/vl/p4.jsp

p4.jsp is a basic ping call from server. The purpose is to tell to the server if any input has come from the user or to tell if the last message sent from server is successfully played. Information about these events are sent in sd parameter as can be seen in Table 3.

Parameter	Description
sn	Serial of sending nabaztag
v	The version of byte code loaded in to Nabaztag
st	Unknown, always has value 1
sd	Last action user has made on Nabaztag
tc	Unknown
h	Unknown

Table 3. The parameters for p4.jsp

Parameter sd can be numeric value in range from 0 to 8 with the following meanings

0. Nabaztag has received Time out from the server.
1. The user has clicked head button twice
2. Nabaztag has received message and completed executing it
3. The user has clicked head button once
4. Unknown
5. Unknown
6. Unknown
7. Unknown
8. Ear positions. When 8 is received, 'sd' is three characters long. The first character is 8 and then there are values for both ears.

4. POST http://192.168.0.2/vl/record.jsp

When the user keeps pressing the button on Nabaztag's head it starts voice recording. After the user has released button it sends record.jsp call to the server with ADPCM encoded audio data in its POST part of message. Record.jsp parameters are shown in Table 4.

Parameter	Description
sn	Serial of sending Nabaztag
v	version of byte code loaded in to Nabaztag
h	unknown
m	unknown

Table 4. The parameters for record.jsp

5. GET http://192.168.0.2/vl/rfid.jsp

When Nabaztag has read the correct type of RFID tag it sends rfid.jsp call to the server. It passes serial of sending Nabaztag, version of byte code and ID of the RFID. If Nabaztag reads ID wrong or ID is kept in the front of Nabaztag so that it repeatedly reads tag, then Nabaztag sends zero ID (sixteen zeros in a row). The parameters for rfid.jsp are shown in Table 5.

Parameter	Description
sn	Serial of sending nabaztag
v	version of byte code loaded in to nabaztag
h	unknown
t	ID of sniffed tag

Table 5. The parameters for rfid.jsp

2.1 Packets

The server responses to Nabaztag on demand when it request a ping command. The answer packets are custom byte arrays. These byte arrays start with byte 7F and end with FF and 0A bytes. In between there can be any amount of blocks. Blocks consists of three parts. First there is a byte telling the type of the block. Known block types are 03 , 04 , 09 and 0A. Next three bytes are reserved for telling the length of data in the blocks. In the length bytes last byte is the most significant byte and bytes are unsigned. After the length bytes there is the data for the block.

The example packet in Figure 1 contains two blocks. The first block is a type 03 which is coloured orange, has length of one and the value of its only byte is five. The next block is a type 09 which is coloured in green and it has a length of zero.

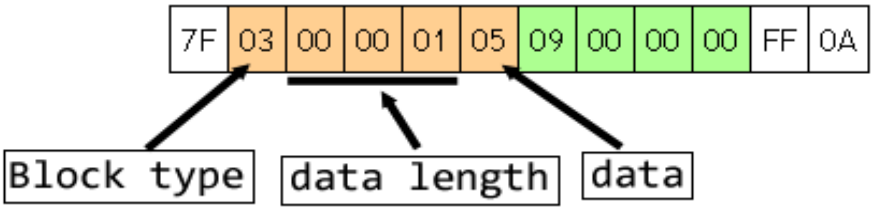


Figure 1. A sample answer packet

2.2 Blocks

Ping interval block (03)

Ping interval block length is always one. When this block is sent to Nabaztag it changes the ping interval to the given time in seconds.

Ambient block (04)

Ambient block sets light flashing patterns when there is no other messages playing. These lights can be interrupted by making the Nabaztag to perform some other operation like playing a sound. Ambient block is always between 22 to 24 bytes long. The length of a block tells how the nose led of Nabaztag is blinked. When the length is 22 bytes Nabaztag will not blink the nose led, with 23 bytes it blinks once every second and with the length of 24 bytes, Nabaztag double blinks its nose led every second. The byte 21 has right ear value what to change to when block start playing and the byte 22 has left ear value.

The data part of a block always starts with the bytes 7F FF FF FF. The values of bytes from 5 to 20 are unknown. These can be filled with zeroes.

Reboot block (09)

Reboot block is always built from the following four bytes: 09 00 00 00. This reboots Nabaztag and so it will call bc.jsp again for the byte code.

Message block (0A)

Message block is encrypted ASCII string message sent to Nabaztag. The encryption formula is the following:

$$C[i] = (B[i] - 0x2F) * (1 + 2 * C[i-1]) ,$$

where C is encrypted byte array, B is decrypted byte array and i is range from zero to length of byte data. As this algorithm gives encrypted value of next character depending on byte before current, there is a need for byte C[-1] which has constant value 35.

Message block contains commands that are separated from each other with a newline character. The known commands are shown in Table 6.

Com-mand	Format	Description
ID	ID <some number>	ID of the message. This way messages can have id's. Purpose is unknown.
CL	CL 0xAABBCCDDh	Set colour command. In format AA is number of led from 00 to 04, BB, CC and DD are colour values for red, green and blue in hex value.
PL	PL 0X	Set palette command. In format X is from 0 to 7. This sets palette colour for choreography files.
CH	CH <URL>	Choreography command. This command gets choreography file from given URL and plays it.
MU	MU <URL>	Play sound command. Plays sound from given URL.
MC	MC <URL>	Synonym for MU command.
ST	ST <URL>	Play stream command. Plays shoutcast streaming radio from given URL.
MW	MW	Wait command. Doesn't take any parameters. This command forces Nabaztag to finish given commands (e.g. playing sound) before going to the next one.

Table 6. Message block commands

2.3 Choreographies

Choreographies are files that can be played by calling them from message blocks with CH command. Choreographies contains led control and ear control. Each of the five leds can be controlled to have RGB values from 0-255. Ears can be moved to certain positions from 0x0 to 0x12 forwards or backwards. There is also a possibility to move an ear forwards a certain step count. These files are made from three parts. First there is a header which contains the length of bytes and in middle there is the data and at the end four zero bytes. In Figure 2 we have an example of simple choreography file where the data part is in orange colour. This data contains one command that tells the centre led to change into red:

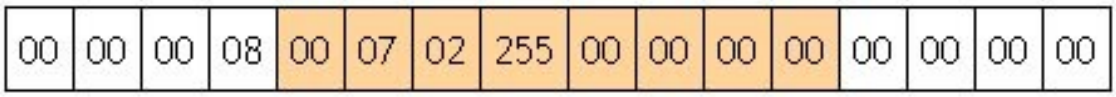


Figure 2. A sample choreography file

Data is filled with smaller command byte arrays. Choreography files are timed and this reflects also to the command bytes. The first byte of command bytes tells how long to wait before playing a new command from the beginning of the last command. The second byte tells the type of the command. After these two bytes there can be any amount of bytes relative to the command. In this example we have one command where the first byte on white background tells that the command is played instantly when the file is started to play. Next on blue background we have the type of command byte and in the end with orange colour we have the data of command. See Figure 3.

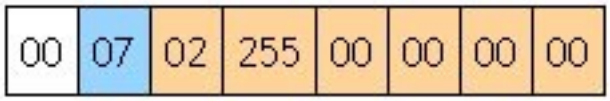


Figure 3. A sample choreography command

In what follows, we list the possible choreography commands. A tempo command sets how fast commands are played. Type byte is 1. Data part is build from one byte that tells tempo. At default this value is 10. Figure 4 shows an example that sets tempo to 15.

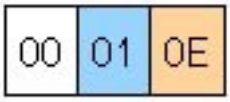


Figure 4. A sample tempo command

Led command sets colour of given led to RGB colour. Type byte is 7 and data is build in the following way: first byte tells led which colour to change from 0 to 4, next led is red value followed by green and blue bytes. Lastly there is two zero bytes that are being ignored. In Figure 5 we have an example which sets centre led red value to 255, green to 255 and blue to zero.

Figure 5. A sample led command

Ear command type byte is 8 and it consist of three data bytes. First one is

the ear that can be 0 (right) or 1 (left). Next is the ear position between 0x0 and 0x12 (0-18). And third byte is the direction of rotation, 0 (forward) or 1 (backward). In Figure 6 right ear is rotated 'forward up' position by rotating the ear backwards.

Figure 6. A sample ear command

Play random MIDI command is type 0x10 command. Plays random mini note. It has no data part. See figure 7.

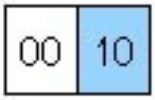


Figure 7. Play random midi command

Ear step command is type 0x11 command. Data is made from two bytes: first byte selects the ear 0 (right) or 1 (left) and next byte tells how many steps to go forwards. In Figure 8 we move left ear ten steps forwards.

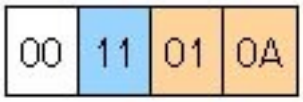


Figure 8. A sample ear step command

In addition to these commands there are "ifne" command that is a some sort of loop command and "attend" command that synchronizes choreography to sound. These two commands has not been tested and their functionality isn't sure yet.

3. jNabServer

jNabServer is Java based server software that is designed to handle operations which are introduced in the previous chapter.

jNabServer is build on top of very light weight HTTP-server. This means that all functions that jNabServer performs are communicated through HTTP-protocol. These functions include communication with Nabaztag and XML API. HTTP-protocol was chosen because Nabaztag uses it. There was no reason to start building our own communication layer. This means that all the data

coming into jNabServer can be either in parameters of requested URL or in POST data of message. XML API employs a slightly inconvenient way of using .xml ending files where you have to pass POST data.

This means that jNabServer is a framework. Actual functionality wanted for Nabaztag is done in form of plugins. XML API provides tools for making raw data request straight to the Nabaztag without interfering with the plugin. At all time a Nabaztag can have only one active plugin, and all commands coming from Nabaztag are passed to this plugin.

Usually instance of plugin's parent, the Nabaztag class is passed to the plugin as a parameter in constructor. This gives the plugin an ability to change itself to another plugin inside the Nabaztag class. This method is called state architecture.

Supported requests

jNabServer can handle any kind of requests, but most requests are of the following five types: bc.jsp, locate.jsp, p4.jsp, record.jsp and rfid.jsp. These requests must have string "/vl/" preceding to make server understand them. The server can pass local files by starting request with string "/local/". For example, if you would like to call c:\mysound.mp3 file from server you request "/local/c:\mysound.mp3" file from server. Also the server can pass choreographies from choreogoraphy library by starting command with "/choreographylibrary/". After this can be be added name of wanted choreography.

Architecture

As Nabaztag uses HTTP-protocol to communicate with jNabServer it means that network architecture is relative simple. Cycle of one HTTP request starts from Nabaztag. Nabaztag asks for one of the possible files from jNabServer. These request are more detailed explained in requests part of documentation. In jNabServer MicroServer socket server receives this call and passes it to new Worker class. Worker class parses request. If request is boot code or locate request it creates response and sends it to Nabaztag. After sending data connection is closed and Worker terminated. If request was any other type than boot code or locate it is passed to Nabaztag class. If Nabaztag has any pending forced packets it sends one of them and Worker is terminated. If Nabaztag class doesn't have any pending forced packets it parses request and passes call to correct method in plugin. After the correct method has been called Nabaztag tells plugin to send data. Figure 9 shows how requests circles in jNabServer.

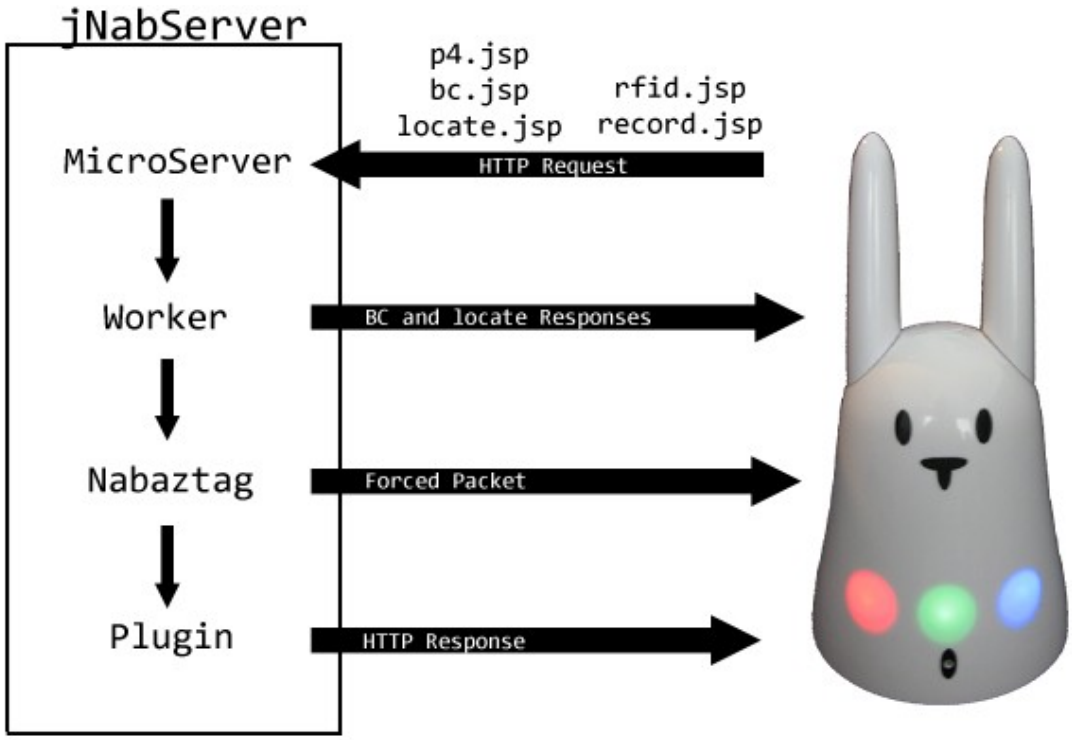


Figure 9. Cycle of one HTTP request coming from Nabaztag

3.1 Class descriptions and building a simple custom jNabServer

The purpose of **MicroServer**(*fi.uta.cs.nabaztag.server*) is to receive incoming socket connections from Nabaztag and pass them to Worker threads. To setup custom server one needs to create an instance of MicroServer and start it. MicroServer takes pingServer and broadcastServer parameters which usually is local IP address of server where jNabServer is running. Code fragment 1 is an example of the most basic implementation of jNabServer. The only problem in Code fragment 1 is that it doesn't do anything except sends boot code to Nabaztag.

```
import fi.uta.cs.nabaztag.server.MicroServer;

class myNabaztagServer {
        public static void main (String [] args) {
                MicroServer server = new MicroServer("192.168.1.100", "192.168.1.100");
                server.start();
        }
}
```

Code Fragment 1. A basic implementation of jNabServer.

Burrows(*fi.uta.cs.nabaztag*) is a container for all Nabaztags that jNabServer knows. Nabaztags can be added to server specifically by their serial or just let

them connect server when they are automatically added to known Nabaztags. The serial of Nabaztag can be found from bottom of device.

Nabaztag(*fi.uta.cs.nabaztag*). For every Nabaztag, the server knows there is class Nabaztag which contains the most basic information about Nabaztag: its serial and custom name. Nabaztag class takes care of passing requests to plugin that is set for it. Or if there is forced packets this class will send those packets and ignore incoming request.

NabaztagPlugin(*fi.uta.cs.nabaztag.plugins*) is an abstract class from which all plugins should be inherited. These classes can be passed to Nabaztag class to make Nabaztag to operate some custom actions.

With these three more classes we can expand myNabaztagServer example. In the example of Code fragment 2 the server will be loading example echo plugin that is in Code fragment 4. Now myNabaztagServer has functionality for Nabaztag with serial 000000000001.

```
import fi.uta.cs.nabaztag.Burrows;
import fi.uta.cs.nabaztag.Nabaztag;
import fi.uta.cs.nabaztag.server.MicroServer;

class myNabaztagServer {
        public static void main (String [] args) {

                Nabaztag myNab = new Nabaztag("000000000001");
                myNab.setPlugin(new EchoPlugin(myNab));
                Burrows.addNabaztag(myNab);

                MicroServer server = new MicroServer("192.168.1.100", "192.168.1.100");
                server.start();
        }
}
```

Code fragment 2. Extended basic implementation of jNabServer

ChoreographyLibrary(*fi.uta.cs.nabaztag*) is container for choreography files. When one calls in message block CH broadcast/chorlibrary/chorname this call will be passed to ChoreographyLibrary to find "chorname" from it and then to output to HTTP response.

Choreography(*fi.uta.cs.nabaztag*) classes are used to create choreographies. Once a choreography has been build with this class it can be saved to the

ChoreographyLibrary.

Packet(*fi.uta.cs.nabaztag.response*) is a container for blocks that are sent to Nabaztag. Packet has two uses. First it can change given blocks to binary array with generatePacket method. Another use for Packet is to parse a binary packet into message blocks. This can be done by feeding a binary array in to the constructor.

Block(*fi.uta.cs.nabaztag.response*) is the parent class of classes AmbientBlock, MessageBlock, PingIntervalBlock and RebootBlock. Block contains basic methods to get the type and size of a block.

AmbientBlock(*fi.uta.cs.nabaztag.response*) can set behaviour for Nabaztag when it is not receiving any commands. Currently it can change blinking speed of the nose led.

MessageBlock(*fi.uta.cs.nabaztag.response*) contains string data. Detailed information about the format is explained in Section 2.2. MessageBlock can encode and decode message block data to readable format. Class has encode() method that takes string as a parameter which will be encoded. In other hand getContent() method is used to decode given data.

PingIntervalBlock(*fi.uta.cs.nabaztag.response*) is block that can change ping interval time of Nabaztag. You will need to pass time in seconds to the constructor of PingIntervalBlock

RebootBlock(*fi.uta.cs.nabaztag.response*). Sending reboot block to Nabaztag causes it to reboot and ask boot code again.

Request(*fi.uta.cs.nabaztag.request*) is a class that parses incoming requests from Nabaztag. It also receives possible POST data when it is sent in record.jsp request.

3.2 Creating packets in jNabServer

All that is needed for classes to make packets are in fi.uta.cs.nabaztag.server.response package. Class Packet is a container for all blocks that are inherited from class Block. To build a packet you first need to create an instance of it and then use addBlock method to add blocks to the

packet. After all blocks has been added to the packet you can either send it to any nabaztag with nabaztag.forceNextPacket(Packet) or create byte array with Packet.generatePacket(). In Code fragment 3 we create a packet with few blocks. This example can be used only in plugins. In other places you can replace the last line with line nabaztag.forceNextPacket(p);.

```
//new packet instance
Packet p = new Packet();
//setting pinginterval to five seconds
p.addBlock(new PingIntervalBlock(5));
//generate byte array of packet
int [] byteArray = p.generatePacket();
```

Code fragment 3. A sample of creating packet.

There is two possible ways of using choreography files. You can either directly call a local .chor file to play or create choreography classes that can be requested from choreogoraphyLibrary. To request file named "myChor" from choreographyLibrary add following command to messageblock: CH broadcast/chorlibrary/myChor\n. If you are using SimpleMessageBlock you can call method playChoreographyFromLibrary(String name).
Nabaztag has only one plugin active and all commands coming from Nabaztag is passed to this plugin. If plugin knows which Nabaztag it belongs to (e.g. passing Nabaztag as parameter to plugin) then it can change the plugin to any other plugin. This method gives possibility to build a state machine.

As an example we define Code fragment 4, a plugin to create echo. There are three things it does. At first it records echo.wav file in onRecord method. At second it plays echo.wav on the next call that goes to Nabaztag with forceSendPacket method and the third one is that pressing the button changes the ping interval into two seconds. Changing ping time and playing sounds differ so that in playing a sound, Nabaztag is told to force play sound when next ping comes from it. This prevents Nabaztag to call the plugin at all and it just plays the sound. In onClick method in other hand returns the response to Nabaztag on the same ping call.

```
package fi.uta.cs.nabaztag.plugin;

import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Arrays;

import fi.uta.cs.nabaztag.Nabaztag;
import fi.uta.cs.nabaztag.server.request.Request;
```

```
import fi.uta.cs.nabaztag.server.response.Packet;
import fi.uta.cs.nabaztag.server.response.PingIntervalBlock;
import fi.uta.cs.nabaztag.server.response.SimpleMessageBlock;


/**
 * Purpose of this plugin is to make Nabaztag to say last
 * thing it has recorded. Plugins need to be inherited from
 * NabaztagPlugin class.
 *
 * @author Juha-Pekka Rajaniemi
 */
public class EchoPlugin extends NabaztagPlugin {

	//is the echo send?
	private boolean sendEcho;

	//give current nabaztag in constructor. This
	//give possibility to change plugins on the fly
	//optional.
	public EchoPlugin(Nabaztag n) {
			super(n);
			sendEcho = false;
	}


		/**
		 * When data has been sent to Nabaztag
		 */
	@Override
	public void dataSent() {
			//is the echo sent yet?
			if (sendEcho) {
					//if not then we create messageblock with id 600
					SimpleMessageBlock smb = new
SimpleMessageBlock(600);
					//tell jNabServer to play local file named echo.wav
					smb.playLocalSound("echo.wav");
					//tell jNabServer to wait for echo.wav to play
					smb.waitForCommands();
					//required method for SimpleMessageBlock to generate
block
					smb.finish();

					//then create packet which will be sent to Nabaztag
					Packet p = new Packet();
					//add created simple message block to packet
					p.addBlock(smb);
					//add another block which tells nabaztag to ping
					//jNabServer every second
					p.addBlock(new PingIntervalBlock(1));
```

```
                                        //tell plugin owner add packet to forced queue
                                        bunny.forceNextPacket(p);

                                        //echo has been sent
                                        sendEcho = false;
                                }


                                dataToSend = null;
        }

        /** What to do when user has double clicked */
        @Override
        public void onDoubleClick() {}

        /** When ears has been moved */
        @Override
        public void onEarsMove(int rightEar, int leftEar) {}

        /** When succesfully all messages has been sent */
        @Override
        public void onEndOfMessage() {}

        //variable to save last income data. This is because
        //Nabaztag keeps sending data for several times...
        private int[] lastData;


                /**
                 * When audio has been recorded parameter data has WAV raw
                 * data in array.
                 */
        @Override
        public void onRecord(int[] data) {
                        //don't do anything if data has been received already
                        if (lastData != null && Arrays.equals(lastData, data)) {
                                        return;
                        }

                        //write data to echo.wav
                        try {
                                        FileOutputStream fos = new
FileOutputStream("echo.wav");
                                        for (int i = 0; i < data.length; i++) {
                                                        fos.write(data[i]);
                                        }

                                        fos.close();

                                        lastData = data;

                                        sendEcho = true;
```

```
                        }
                        catch (IOException ioe) {}
        }


                /**
                 * When Nabaztag rfid this method is called
                 */
        @Override
        public void onRfid(String rfid) {}


                /**
                 * When button on Nabaztag head is clicked. Example
                 * how to send message message to nabaztag during same
                 * call.
                 */
        @Override
        public void onSingleClick() {
                        //create ping interval block
                        PingIntervalBlock pib = new PingIntervalBlock(2);
                        //add it to packet
                        Packet p = new Packet();
                        //add block to packet
                        p.addBlock(pib);

                        //nabaztag send dataToSend byte array to Nabaztag
                        //when this method has been processed. Packet has
                        //method generatePacket() which returns byte array
                        //version of packet.
                        dataToSend = p.generatePacket();
        }

        @Override
        public void onSingleClickWhilePlayingSound() {}

        @Override
        public void onTimeOut() {}


                /**
                 * When nothing matches then this method is run. Nabaztag's request
                 * is given as parameter
                 */
        @Override
        public void unknownRequest(Request r) {}

}
```

Code fragment 4. An example echo plugin for jNabServer

3.3 Build in XML API

In XML API client sends data in POST format to server. Server processes and responses in XML. Here is a list of possible calls to jNabServer

Requests

/xml/rawrequest.xml By passing data rawrequest.xml one can send packets and blocks straight from a client to Nabaztag. Nabaztag then processes given packets as forced packets and sends them in next call. In Code fragment 5 is DTD of XML that rewrequest.xml parses.

```
<!DOCTYPE RAWREQUST [

<!ELEMENT rawrequest (packet+) >
<!ELEMENT packet (ambientblock, pingintervalblock, messageblock, rebootblock) >
<!ELEMENT ambientblock ()>
<!ELEMENT pingintervalblock (#PCDATA)>
<!ELEMENT messageblock (#PCDATA)>
<!ELEMENT rebootblock (#PCDATA)>

<!ATTLIST packet serial CDATA #REQUIRED>
]>
```

Code fragment 5. DTD of XML that rawrequest.xml parses

/xml/jnabcommand.xml. This command can change the plugin of certain a Nabaztag or run methods on some plugin. This call can set the given Nabaztag's plugin to the given class. Currently setPlugin passes always the plugin's owner Nabaztag -class in the constructor. Another command is a method. It calls given methods from Nabaztag's plugin. No parameters can be given yet. In Code fragment 6 is DTD of XML that jnabcommand.xml parses.

```
<!DOCTYPE JNABCOMMAND [

<!ELEMENT jnabcommand (setPlugin+, method+) >
<!ELEMENT setPlugin (#PCDATA) >
<!ELEMENT method (#PCDATA)>

<!ATTLIST jnabcommand serial CDATA #REQUIRED>
<!ATTLIST method name CDATA #REQUIRED>
]>
```

Code fragment 6. DTD of XML that jnabcommand.xml parses

Responses

Both rawrequest and jnabcommand responses are same kind of messages.

In a case of success the response is following:

```
<response>
        <success>1</success>
</response>
```

and when there is an error this kind of XML is given:

```
<response>
        <error>
                <message>Error message from API</message>
        </error>
</response>
```

/xml/jnabserverinfo.xml is slightly different case compared to rawrequest and jnabcommand. This XML does not require client to send any data in POST format. It only responses with information about the server. DTD of returning XML file is shown in Code fragment 7.

```
<!DOCTYPE JNABSERVERINFO [

<!ELEMENT jNabServerInfo (registeredNabaztags+) >
<!ELEMENT registeredNabaztags (nabaztag*) >
<!ELEMENT nabaztag (plugin+, serial+)>
<!ELEMENT plugin (#PCDATA)>
<!ELEMENT serial (#PCDATA)>
]>
```

Code fragment 7. DTD of XML that jnabserverinfo.xml returns

Logger

Logger is a helper class built specially for the jNabServer. By default it just prints to the console all messages that are sent to it. To prevent the console messages to appear, Logger needs to have at least one listener set to it. This is done by using addListener() method to add a class as a listener. Listening class must implement fi.uta.cs.nabaztag.utils.LoggerListener interface.

Logger has one important method: addLog(). As a first parameter it takes the log message and as a second parameter an optional message level integer. By default all exceptions are at level 1 and all messages are at level 3.

4. Concluding words

This work has been mostly finding features with trial and error method and collection hints on various chats and forum topics on net. The new version of

Nabaztag (Nabaztag/tag) was uncharted territory as it wouldn't work in the same way as the original Nabaztag. After long testing I was able to build up jNabServer in the way I wanted it to be and it turned out to be really good tool to build Nabaztag programs. We tested jNabServer by building weakest link type game involving two Nabaztag's, a butler program that would pass saved messages when Nabaztag sniffed user RFID tag and dancing program (Rajaniemi & Antila, 2007) to test choreographies and sound capabilities.

Now that most of the basics of Nabaztag has been resolved and it is possible to build programs for it. As this project has been with close relation to the Companions project, which involves creating artificial friends for humans, I think it might be interesting to continue to that direction. For an example solution for the elder people that makes sure that they do their daily routines like cooking, taking medicines and cleaning. Also possibility of using multiple Nabaztag's in group working: saving notes, passing information to distant group members or even working as a voice over IP client. Another direction to go would be integration of jNabServer to smaller devices through j2ME. jNabServer could be running on mobile phone with WiFi capability it could be running inside of wireless router itself. Byte code is mystery also. Maybe I could build whole thing inside Nabaztag removing need of server making Nabaztag only possible to receive plugins to operate itself.

5. References

CCITT. (1990). 40, 32, 24, 16 kbit/s Adaptive differential pulse code modulation (ADPCM).

Dantant, J. (2007). ISO/IEC 14443 /: Proximity cards. Retrieved August 7, 2007, from http://www.waazaa.org/14443/.

Nabaztag. (2007, August 15). In Wikipedia, The Free Encyclopedia. Retrieved August 17, 2007, from http://en.wikipedia.org/w/index.php?title=Nabaztag&oldid=151348508.

Rajaniemi, J-P., & Antila, V. (2007). Nabaztag dancing. Retrieved August 17, 2007, from http://www.youtube.com/watch?v=QHh6PoPYM6g.

Grid-laskenta

Mikko Rantanen

Tiivistelmä

Tutkimuksessa käsitellään ensin gridien historiaa, sekä grid-sanan määritelmää. Seuraavissa osissa vertaillaan kahta erilaista arkkitehtuurin toteutusta, ja verkkotekniikoita gridien käytössä. Lopuksi käydään läpi lyhyesti tulevaisuuden näkymiä grid-teknologiassa.

Avainsanat ja -sanonnat: Grid, hajautettu laskenta, Globus Tolkit, Unicore
CR-luokat: C1, C.2

1. Johdanto

Gridit kehittyivät vuosien varrella ”suuren tieteen” tutkimustarpeista. 1980-luvulla kehittyi tutkimustyön ja laitteiden kehittymisen ansiosta parempia rinnakkaisen prosessoinnin menetelmiä. Tutkijat kehittivät algoritmeja, ohjelmia ja arkkitehtuureja, jotka tukivat samanaikaisuutta. Ohjelmistokehittäjät alkoivat kehittää monimutkaisia ohjelmistoja, jotka vaativat aina suurempia määriä laskentatehoa, ja rinnakkainen prosessointi yksittäisillä suurilla palvelimilla ei ollut enää ajallisesti järkevää. Jotkut tutkimusryhmät alkoivat etsiä ratkaisua hajauttamalla laskeminen usean koneen kesken. Tästä muodostui ensin klustereita, ja myöhemmin gridejä. [Berman et al., 2003]

Sana grid sekoitetaan usein klusterin kanssa sen samankaltaisuuden takia. Ero syntyy tietokoneiden ominaisuuksissa, sillä gridit ovat verkotettuja laajalle alueelle, ja niiden ominaisuudet ovat hyvin heterogeenisia. Sekaannusta tuo myös se, että gridit käyttävät usein klustereita osina verkostoaan. Kommunikointiin gridin sisällä ei voi luottaa, kun taas klustereissa on usein identtisiä tai hyvin samanlaisia laitteita, ja ne ovat verkotettu nopealla yhteydellä yhdessä samassa paikassa, ja niiden väliseen kommunikointiin voidaan luottaa. Klustereita pidetään yhtenä isona tietokoneena, missä eri yksiköt voivat keskustella keskenään prosessoinnin aikana. Gridit eroavat klustereista siinä, että niissä suoritettavat prosessointityöt ovat pakettimaisia, eli yksi gridissä oleva tietokone tekee sille annetun työn itsenäisesti, riippumatta siitä, mitä muut gridin sisällä olevat tietokoneet ovat tekemässä. [Foster and Kesselman, 1998; Berman et al., 2003]

Grid sai sanana alkunsa vuonna 1998, kun Foster ja Kesselman [1998] määrittelivät sen vapaasti suomennettuna seuraavasti:

"Laskennallinen grid on laite ja ohjelmistoinfrastruktuuri, joka mahdollistaa luotettavan, yhdenmukaisen, kaikkialle leviävän ja halvan pääsyn huipputason laskennallisiin voimavaroihin."

Myöhemmin Foster [2002] tarkensi gridin määritelmää kuvaamalla kolme asiaa, jotka rajaavat mikä gridistä tekee gridin:

1. Koordinoi resursseja, jotka eivät ole keskitetyn kontrollin alaisena... (Grid integroi ja koordinoi resursseja ja käyttäjiä, jotka sijaitsevat eri verkkoalueilla, esimerkiksi käyttäjän työpöytä- vs. keskitetty laskenta: eri hallinnollinen yksikkö samassa yhtiössä tai eri yhtiöitä; ja ratkaisee tietoturvan, menettelytavan, maksun ja jäsenyyden ongelman, jotka ilmentyvät tässä ympäristössä. Muuten kyseessä on paikallinen hallinnointijärjestelmä.)

2. ... käyttäen avoimia standardeja, yleisprotokollia ja rajapintoja ... (Grid on rakennettu monikäyttöisistä protokollista ja rajapinnoista, jotka ratkaisevat tärkeitä ongelmia kuten autentikaation, oikeudet, resurssien havaitseminen ja resurssien saatavuus. On tärkeää, että nämä resurssit ja rajapinnat ovat avoimesti kaikkien saatavilla ja standardoituja.)

3. ... toteuttaakseen laadultaan vaihtelevia palveluita. (Grid sallii käyttää sen aineksina olevia resursseja koordinoidulla tyylillä tarjotakseen vaihtelevan laatuisia palveluja. Tarkoittaen esimerkiksi vastausaikaa, suoritustehoa, saatavuutta ja tietoturvaa ja/tai usean resurssityypin jakamista vastatakseen käyttäjien monimutkaisiin vaatimuksiin.)

Gridit ovat siis ominaisuuksiltaan heterogeenisia, verkon kautta hajautettuja laajalle alueelle olevia yhtenäisiä systeemeitä, jotka käyttävät avoimia standardeja, yleisprotokollia ja rajapintoja. Niiden avulla ratkaistaan laskennallisesti vaikeita ongelmia pienemmässä ajassa, kuin niiden ratkaisu kestäisi perinteisillä menetelmillä.

2. Arkkitehtuureista

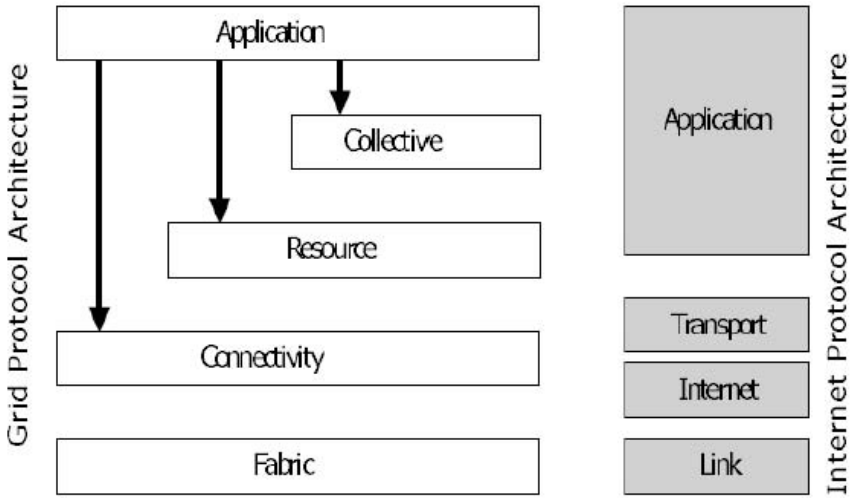
Funktioidensa puolesta gridit voidaan luokitella kolmeen eri luokkaan [Minoli, 2005]:

- Edellisessä luvussa mainittu laskennallinen grid, jossa koneet varaavat resursseja prosessori-intensiiviseen työhön.

- Prosessorien vapaita syklejä “saalistavat” gridit, jotka käyttävät hyväksi koneen vapaita hetkiä häiritsemättä sen mahdollista käyttäjää.
- Datagridit jotka antavat yhdistetyn rajapinnan kaikille datan säilytyspaikoille organisaatiossa, ja niiden läpi dataan voidaan tehdä kyselyitä, voidaan hallinnoida sekä varmistaa.

Arkkitehtuurin gridissä määrittelevät ensisijaisesti käytetyt protokollat, sillä se toimii verkotetussa ympäristössä. On tärkeää, että protokollat ovat standardoituja, jotta jatkuvuus, toimivuus, siirrettävyys ja koodin jakaminen säilyvät eri arkkitehtuurien välillä. Hyvin toteutettuna nämä funktiot voivat vähentää kustannuksia ohjelmiston suunnittelu- ja toteutusvaiheessa. Toinen arkkitehtuurin määrittelevä asia on käytettävät API:t (Application Programming Intergaces) ja SDK:t (Software Development Kits). [Berman et al., 2003]

Rakennetta voidaan parhaiten kuvata järjestelemällä komponentit kerroksiin. Kuvassa 1 on kuvailtu gridin arkkitehtuuria kerroksina, ylemmät kerrokset perivät ominaisuuksia alemmilta kerroksilta.



Kuva 1: Kerroksinen grid-arkkitehtuuri ja sen suhteet internetin protokolliin [Berman et al., 2003]

Tässä rakenteessa *Resource* ja *Connectivity* ovat keskeisin osuus; ne tekevät yksittäisten resurssien jakamisen helpommaksi. Protokollat näillä kerroksilla on suunniteltu niin, että ne voidaan toteuttaa monipuolisten resurssityyppien päälle, jotka ovat määritelty *Fabric* kerroksessa. Tämän kerroksen avulla voidaan rakentaa laajasta valikoimasta globaaleita palveluita ja ohjelmistospesifejä toimintoja *Collective* kerroksella. [Berman et al., 2003]

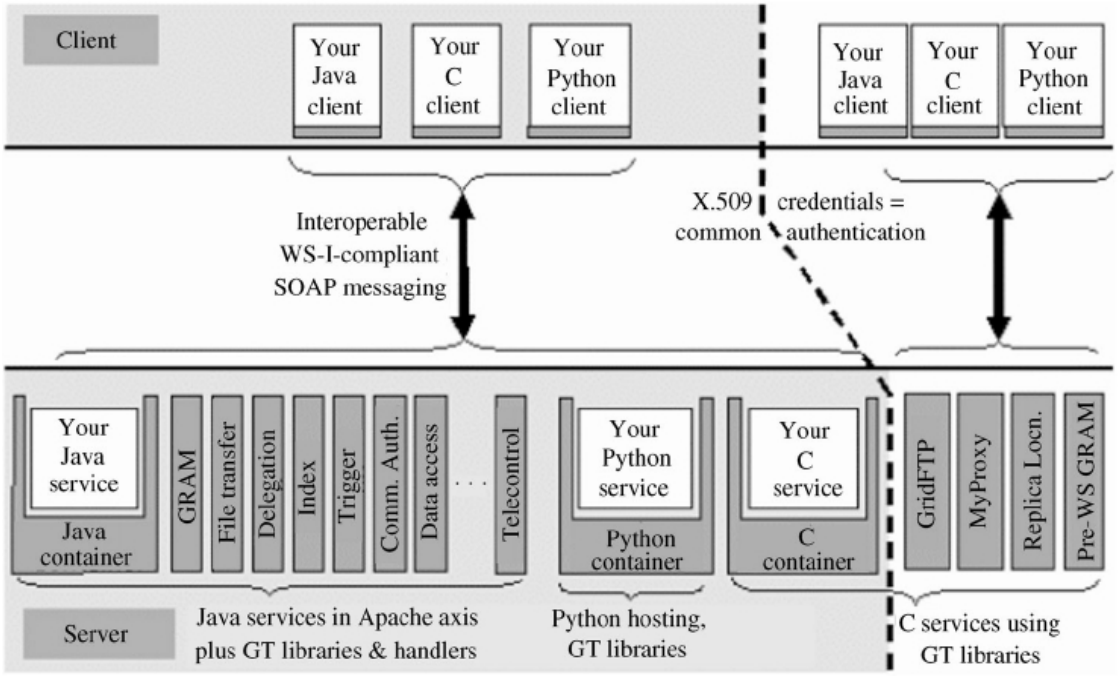
Tämä kerroksinen arkkitehtuurin kuvaus on niin yleinen, että se ei aseta rajoitteita suunnittelulle tai toteutukselle. *Application*-kerrokseen voi tämän abstraktion ansiosta luoda haluamansa ohjelmiston ilman, että koko arkkitehtuuria tulisi muuttaa. Tämä helpottaa ohjelmiston käyttöönottoa gridissä.

Seuraavaksi käyn läpi Globus Alliancen tuottaman Globus Toolkitin sekä UNICOREn arkkitehtuurit. Lopuksi vertailen niiden ominaisuuksia, protokollia ja rakennetta lyhyesti. Näiden kahden valinta tähän vertailuun vaikutti se, että sekä Globus Toolkit ja UNICORE ovat suosituimpia grid-arkkitehtuureja.

2.1 Globus Toolkit

Globus Alliance on kansainvälinen järjestö, joka tekee yhteistyötä usean yliopiston ja yrityksen kanssa kehittäen avoimen lähdekoodin ratkaisuja grid-infrastruktuuriin. Järjestö on kehittänyt työkalun nimeltään Globus Toolkit, jonka avulla on mahdollista luoda käyttäjälleen räätälöity järjestelmä, ilman suurta työtä grid arkkitehtuurin kanssa. Globus Toolkit käyttää Open Grid Services Architecture standardia, joka on palvelukeskeinen arkkitehtuuri. GT:n kehitys alkoi 1990-luvun loppupuolella, ja siinä on päästy jo neljänteen versioon.

Niin kuin kuvasta 2 voi nähdä, GT koostuu palvelimesta sekä gridiin osallistuvan asiakkaan puolesta. Palvelin sisältää infrastruktuuripalveluita, muun muassa toimeenpanon hallinta (GRAM), datan käsittely (GridFTP, RFT, OGSA-DAI), duplikaattien hallinta (RLS, DRS), monitorointi ja resurssien havaitseminen (Index, Trigger, WebMDS), valtuuksien hallinnointi (MyProxy, Delegation, SimpleCA) sekä instrumenttien hallinnnointi (GTCP). Useimmat näistä ovat Javan Web Servicesteknologiaa, mutta osa on toteutettu Pythonilla tai C-kielellä. Käyttäjän puolella on 3 säiliötä, joita voidaan käyttää arkkitehtuurin päälle rakennettavien ohjelmistojen ajamiseen. Kielivalintoina ovat Java, Python tai C. Säiliöt toimivat tietoturvan, hallinnoinnin, resurssien havaitsemisen, tilan hallinnan ja muiden mekanismien alustana, kun ohjelmistoissa halutaan käyttää näitä ominaisuuksia. [Foster, 2006]



Kuva 2: Kuvaus Globus Toolkitin arkkitehtuurista [Foster, 2006]

Joukko käyttäjäpuolen ohjelmistokirjastoja antaa ohjelmistoille mahdollisuuden kutsua GT:hen, sekä ohjelmistoon tehtyjä palveluita. Monissa tapauksissa useat rajapinnat tuovat eritasoisia tapoja hallita. Esimerkkinä GridFTP johon on toteutettu yksinkertainen konsolikäyttöliittymä, mutta myös kirjastoja sen datan hallinnointiin itsetehdyissä ohjelmistoissa.

Jo yksinään GT:n palvelut ovat käteviä, mutta yhdessä ne muodostavat yhdenmukaisen abstraktion ja mekanismin, jotta käyttäjät voivat olla vuorovaikutuksessa eri osiin samoilla tavoilla. Tämä yhdenmukaisuus edesauttaa monimutkaisten ja yhdessätoimivien systeemien rakentamista, ja koodin uudelleenkäyttämistä useilla tasoilla. [Foster, 2006]

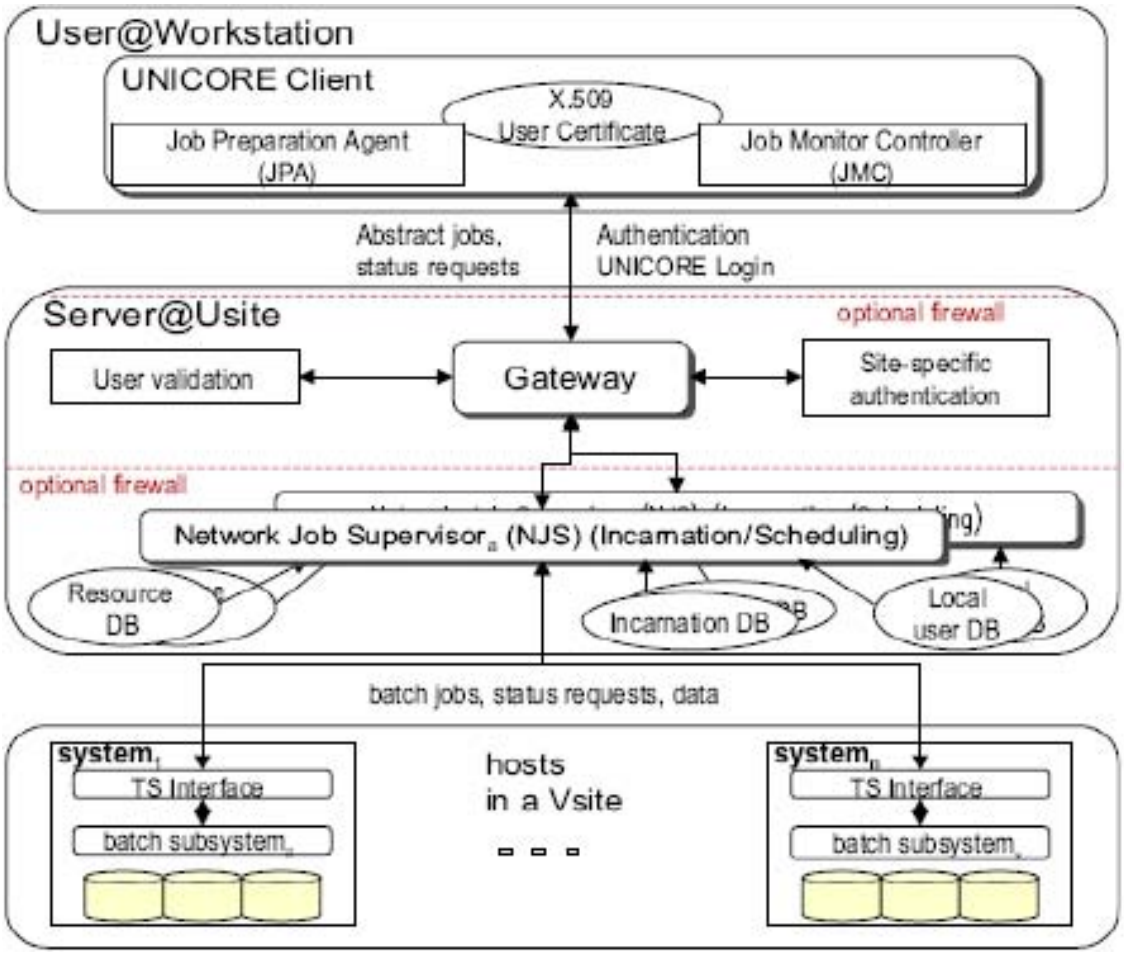
Globus Toolkit on käytössä useissa projekteissa ympäri maailmaa. Brown yliopistossa Yhdysvalloissa, sen päälle on rakennettu ohjelmisto, jolla on simuloitu ihmisten verisuonistojen toimintaa. Euroopan puolella CERN on hyödyntänyt GT:tä, suurien datamäärien prosessointiin hiukkaskiihdyttimestään. Näissä sekä useissa muissa projekteissa grid on ollut välttämätön, sillä saman laskentaan olisi kulunut tavallisilla klustereilla moninkertaisesti aikaa, tai jopa ei olisi ollut ajan ja resurssien takia järkevää.

2.2 UNICORE

Uniform Interface to Computing Resources -arkkitehtuuri on luotu Saksan opetusministeriön rahoituksella vuonna 1997. UNICORE on BSD-lisenssiä käyttävä avoimen koodin tuotos. Sen arkkitehtuuri koostuu kolmesta kerroksesta: käyttäjästä, palvelimesta ja kohde systeemistä. Käyttäjäkerroksella tarkoitetaan gridin ylläpitäjää, ja sitä edustaa graafinen käyttöliittymä, jolla käytetään hyväksi muiden kerroksien tarjoamia palveluita. Tämän käyttöliittymän avulla voidaan luoda ja kontolloida käyttäjän tietoturvaympäristöä, sekä valmistella, ja kontrolloida töitä, jotka suoritetaan UNICORE gridissä. Tietoturvaa on antamassa SSL (Secure Socket Layer) -protokolla, jota käytetään komponenttien välillä, jotka keskustelevat yleisten verkkojen ylitse.

Työt lähetetään gridiin luomalla Abstract Job Object. Se sisältää kuvauksia laskentaan sekä dataan liittyvistä tehtävistä, resurssi-informaatiota ja työnkulun spesifikaation. AJO:jen lähettämistä ja vastaanottamista kontrolloi Unicore Protocol Layer, joka on arkkitehtuurissa SSL-kerroksen päällä. Käyttäjän ei tarvitse välittää näiden protokollien toteutuksista, sillä graafisen käyttöliittymän avulla käyttäjää opastetaan kompleksien töiden muodostuksessa.

UPL välittää AJO:n serverin puolella olevan gatewayn läpi Network Job Supervisorille(NJS). NJS:t kontrolloivat virtuaalisia kohteita ja välittävät niille AJO:ssa määriteltyjä tehtäviä. NJS prosessoi AJO:n ja luo virtuaaliselle kohteelle sopivan työtehtävän. Kuvassa 3 näkyy miten NJS lähettää gatewayn läpi työtehtävän virtuaalisille kohteille. Alla olevassa kuvassa on arkkitehtuurista yksityiskohtainen kuvaus.[Romberg, 2002; Wikipedia, 2007]



Kuva 3: UNICORE-arkkitehtuurin yksityiskohtainen kuvaus [Romberg, 2002]

2.3 Vertailua

UNICORE eroaa Globus Toolkitistä siinä, että GT soveltuu paremmin systeemiin, jossa on mukana useita työpöytäkoneita, kun taas UNICOREa käytetään useammin supertietokoneiden kanssa, sen arkkitehtuurin soveltuvuuden takia. Kummassakin tietenkin voidaan käyttää supertietokoneita ja isoja klustereita. Molemmissa arkkitehtuureissa on omat hyvät puolensa, ja niitä on vaikea pistää järjestykseen, sillä käyttäjän tarpeet pitää ottaa huomioon mahdollisessa grid-arkkitehtuurin valinnassa. Kumpikin käyttää OGSA -konseptia, jossa kaikki on palvelukeskeistä.

Näiden kahden arkkitehtuuritoteutuksen pohjalta on toteutettu yli 40 grid-verkkoa ympäri maapalloa. [Globus Alliance, 2007; UNICORE, 2007]

3. Verkkotekniikoista

Useat gridit käyttävät arkkitehtuurin osanaan standardoitua salaustekniikkaa X.509 [RFC3280, 2002], joka on Grid Security Infrastructure:ssa (GSI) implementoituna. Globus Toolkit ja UNICORE kummatkin käyttävät sitä hyväkseen *Transport*-kerroksissaan. X.509 on eräänlainen "luottamuksen verkko", jossa kuka vain voi allekirjoittaa toisen avaimen sertifikaatin todistaen, että heihin voi luottaa. Yleensä allekirjoituksen tekevät luotetut järjestöt, näin taaten sertifikaatin aitouden. Salaustekniikalla voidaan asettaa gridin liikenteelle tietynasteinen luottamus.

Open Grid Service Architecture pyrkii olemaan *Transport*-kerroksen osaltaan aknostinen. OGSA käyttää verkkoliikenteessä Web Serviceja, jotka kommunikoivat XMLtekniikan avulla. [W3C, 2004]

Gridit toimivat ip-protokollien päällä, ja ovat siitä syystä tilattomia. Tästä seuraa ongelmia esimerkiksi pakettien katoamisella matkalla kohteeseensa. Arkkitehtuureihin on luotu QoS-tekniikoita joilla, muun muassa pakettien katoamista pyritään ehkäisemään. Useat toteutukset osaavat itsenäisesti pyytää uudestaan hukkuneita pakettejaan.

3. Yhteenveto

Gridien tulevaisuus näyttää hyvin valoisalta. Suurien datamäärien prosessoinnin ja analysoinnin tarve kasvaa eksponentiaalisesti. Yritykset ovat tulleet kehitystyöhön mukaan vasta viime vuosina, ja tämä on nopeuttanut arkkitehtuurin, protokollien sekä standardien kehittämistä. [Baker et al., 2005]

Lähivuosina on alkanut yleistymään usean ytimen prosessorit, tämä tulee varmasti vaikuttamaan grideistä löytyvään laskentatehoon. Muutaman vuosikymmenen sisään myös mahdollisesti Internet 2 -teknologia tulee antamaan grideille adrenaliinia, liikuttaen jopa 8,8 gigabittiä sekunnissa verkkojen ylitse. [Internet 2, 2007]

Gridit tulevat kaipaamaan standardointia jokaisella osa-alueellaan. Globus Toolkit sekä UNICORE ovat hyvin laajassa käytössä, ja kehitystyö jatkuu.

Viiteluettelo

[Minoli, 2005] Daniel Minoli, *A Networking Approach to Grid Computing*, Wiley,

2005.

[Baker et al., 2005] Mark Baker, Amy Apon, Clayton Ferner and Jeff Brown, Emerging grid standards, *Computer* **38** (April 2005) 43-50.

[Berman et al., 2003] Fran Berman, Geoffrey C. Fox and Anthony J. G. Hey, *Grid Computing: Making the Global Infrastructure a Reality*, Wiley, 2003.

[Foster, 2002] Ian Foster, What is the Grid? A Three Point Checklist. Available as http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf. Checked 17.5.2007.

[Foster, 2006] Ian Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, In: *IFIP International Conference on Network and Parallel Computing* **21** (July 2006), 513-520.

[Foster and Kesselman, 1998] Ian Foster and Carl Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1998.

[Globus Alliance, 2007] Projects with Globus Alliance Participants. Available as http://www.globus.org/alliance/projects.php. Checked17.5.2007.

[Internet2, 2007] Internet2, Buiding Tomorrow's Internet. Available as http:// www.internet2.edu/. Checked17.5.2007.

[Minoli, 2005] Daniel Foster Minoli, *Neworking Approach to Grid Computing.* Wiley, 2005.

[RFC3280, 2002] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Available as http://tools.ietf.org/html/ rfc3280. Checked17.5.2007.

[Romberg, 2002] The UNICORE Grid Infrastructure, *Special Issue on Grid Computing of Scientifc Programming Journal*, **10** (2002) 149 -157.

[UNICORE, 2006] Projects. Availble as http://www.unicore.org/index.php?id= projects. Checked 17.5.2007.

[W3C, 2004] Web Services Architecture, W3C Working Group Note 11 February 2004. Availble as http://www.w3.org/TR/ws-arch/. Checked 17.5.2007.

[Wikipedia, 2007] Wikipedia, UNICODE. Availble as http://en.wikipedia.org/ wiki/UNICORE. Checked 17.5.2007.

Ensimmäisen ohjelmointikielen valinnasta

Maria Salo

Tiivistelmä.
Tämä tutkielma käsittelee ohjelmoinnin alkuopetuksessa käytettäviä ohjelmointikieliä. Erityisesti pohditaan, mitkä ohjelmointikielet ovat suosittuja opetuskieliä ja mitkä asiat vaikuttavat ohjelmointikielen valintaan. Suosituimmat ohjelmointikielet opetuksessa ovat nykyisin C++ ja Java. Ohjelmoinnin alkeiden opettamiseen käytettävän kielen valintaan vaikuttaa välillä kielen yksinkertaisuus ja toisinaan taas kielen käyttö kaupallisissa sovelluksissa. Ohjelmoinnin opetustapaa pidetään usein tärkeämpänä kuin opetuskieltä.

Avainsanat ja -sanonnat: ohjelmoinnin opetus, ensimmäinen ohjelmointikieli, C, C++, Java, Python
CR-luokat: K.3.2

1. Johdanto

Tämä tutkielma käsittelee ohjelmoinnin alkeiskursseilla käytettäviä ohjelmointikieliä, ja sitä, miksi näitä tiettyjä kieliä käytetään. Onko jokin kieli muita parempi ohjelmoinnin aloittamiseen? Mikä tästä kielestä tekee paremman? Vaikuttaako opetusmenetelmä ohjelmointikieltä enemmän, sekä onko kielen valinnalla muita vaikutteita, kuten esimerkiksi työnantajien vaatimukset tietyn kielen osaamisesta?

2. 1970-luku

Laajimmin käytettyjä ”oikeita” ohjelmointikieliä olivat 1970-luvulla BASIC, FORTRAN ja COBOL [Taylor, 1977]. 1950-luvun loppupuolella julkaistu FORTRAN oli helpompi vaihtoehto käytössä olleille konekielille. FORTRAN vähensi kirjoitettavan koodin määrää niin paljon, että kielen suosio nousi nopeasti. COBOL (COmmon Business-Oriented Language) julkaistiin 1959 käytettäväksi kaupallisissa sovelluksissa. BASIC (Beginner's All-purpose Symbolic Instruction Code) kehitettiin 1960-luvun alussa nimenomaan aloitteleville ohjelmoijille.

Taylor kertoo [1977], kuinka hän ja muut Columbian yliopiston opettajat olivat tulleet siihen tulokseen, että ohjelmoinnin opettamisen aloittamiseen

sopii parhaiten hänen kehittämänsä FPL (First programming language). FPL ei ole ”oikea” ohjelmointikieli, vaan sillä ohjelmointi tapahtuu paperilla symbolien ja tekstin avulla. Taylorin mukaan suurimpia ongelmia ”oikeiden” kielten, kuten BASIC- tai FORTRAN-kielen, käytössä opetuksessa ovat mm. se, että näillä kielillä kirjoitettujen ohjelmien kääntäminen tapahtuu tietokoneella. 1970-luvulla monella ei ollut mahdollisuutta tietokoneen käyttöön, ja Taylorin mukaan ei ole kohtuullista olettaa, että ohjelmoinnin opiskelijoiden täytyisi voida käyttää tietokonetta.

Vuonna 1986 Cunniff ja muut [1986] tutkivat, tekevätkö FPL-kielellä ohjelmoinnin aloittaneet erilaisia virheitä kuin Pascal-kielellä aloittaneet. Tutkimuksen mukaan FPL-ohjelmoijat tekivät vähemmän virheitä, ja ne virheet joita tehtiin, eivät olleet yhtä vakavia kuin Pascal-opiskelijoilla. Tutkimuksessa todettiin, että graafisia kieliä, kuten FPL-kieltä, voitaisiin käyttää aloittelijoiden yleisten virheiden kitkemiseen.

1970-luvulla kehitettiin myös MABEL-kieli (MAnitoba BEginner’s Language). Se kehitettiin ohjelmointikieleksi niille, jotka eivät ole koskaan ennen ohjelmoineet [King *et al.*, 1978]. Kielen kehityksessä pidettiin mielessä tietyt vaatimukset, joita kielelle oli asetettu. Sen tulisi mm. olla yksinkertainen, ohjelman toiminnot tulisi olla helppoa ymmärtää, vaikka ohjelmaa ei olisi erityisemmin kommentoitu/dokumentoitu ja kielen tulisi olla helppo opettaa. MABEL-kieli jäi kuitenkin ilmeisesti vain tähän yhteen versioon, eikä sitä otettu käyttöön opetuksessa laajemmin.

3. 1980-luku

1980-luvun puolessavälissä Pascal oli yleisin ohjelmoinnin alkeiskursseilla käytetty ohjelmointikieli [Brilliant and Wiseman, 1996]. Se oli helppo kieli aloittelijoille ja Pascal-ympäristö tarjosi tarpeeksi tukea aloitteleville ohjelmoijille. Ohjelmointikielen huonoiksi puoliksi luettiin kuitenkin se, että opiskelijat alkoivat turvautua ohjelmointiympäristön tukeen liikaa ja tämä aiheutti ongelmia siirryttäessä muiden käytännöllisempien ohjelmointikielien pariin. Myös Pascalin heikko menestys kaupallisena ohjelmointikielenä aiheutti sen korvaamisen muilla kielillä. Kilpailu työmarkkinoilla oli kovaa ja haluttiin opettaa kieltä, joka auttaisi opiskelijoita työnhaussa myöhemmin.

Muita ohjelmoinnin aloittamisen opetuksessa käytettyjä kieliä 1980-luvulla olivat Ada, Scheme, C ja C++. Ada käytti Pascalin vahvoja puolia, mutta siitä

puuttuivat sen heikot puolet. Huonoiksi puoliksi Adassa luettiin mm. se, että kieli oli liian laaja.

C-kielen opetuksesta aloittelijoille oltiin montaa mieltä. Moni oli sitä mieltä, että jos opiskelija ei opi C-kieltä, on hän altavastaajana työmarkkinoilla, sillä C oli kieli, jonka osaamista moni työnantaja vaati. Samasta syystä moni opiskelija oli motivoituneempi opiskelemaan C-kieltä. Myös monilla myöhemmistä tietojenkäsittelykursseista vaadittiin C-kielen taitoa.

Toisaalta taas moni oli C-kielen opetusta vastaan. C:tä pidettiin liian monimutkaisena kielenä aloittelijoille muun muassa sen sisältämien osoittimien vuoksi [Gupta, 2004], ja toiset olivat sen opetusta vastaan juuri siksi, että se oli niin suosittu ja kaikki käyttivät sitä [Brilliant and Wiseman, 1996]. Myös C:n kapseloinnissa ja tiedon piilottamisessa oli monien mielestä puutteita.

Vuonna 1985 julkaistua C++-kieltä pidettiin parempana vaihtoehtona C-kielelle, sillä vaikka sekään ei ole ongelmaton, monet ongelmista tulevat C-kielestä, joka toimii C++:n pohjana. Vaikka osa ongelmista tulikin C++-kieleen C:n mukana, sopii C++ paremmin opetuskieleksi kuin C.

4. 1990- ja 2000-luvut

1990-luvulla yhä useampi opinahjo siirtyi opettamaan C- tai C++-kieltä, vaikka moni kuitenkin jatkoi Pascalin tai Adan opetusta. Uusia kieliä kehitettiin ja kehitetään yhä, mutta näistä ehkä tunnetuimmiksi tulivat Java ja Python.

Vuonna 1995 Sun Microsystems julkaisi uuden Java-kielen. Java on monin tavoin C++-kielen kaltainen, mutta keskittyy enemmän olio-ohjelmointiin. Java ei sisällä moniperintää tai osoittimia, kuten C++, jonka vuoksi sen katsotaan olevan hieman yksinkertaisempi kieli opettaa. Toisaalta taas olio-ohjelmointi koetaan vaikeaksi opetusaiheeksi aloitteleville ohjelmoijille [Cecchi *et al.*, 2003].

Java nousi nopeasti suosituksi kieleksi myös kaupallisissa sovelluksissa. Koska kieli oli suosittu ja työnantajat alkoivat vaatia sen osaamista, monet opinahjot siirtyivät Javan opettamiseen. C- ja C++-kielet eivät kuitenkaan menettäneet suosiotaan, sillä niiden suoritusajat ovat Javaa parempia, ja niiden käyttö on yhä hyvin yleistä. Monet eivät myöskään halunneet luopua muun muassa osoittimista tai moniperinnästä.

Guido van Rossumin kehittämä ja vuonna 1991 julkaistu skriptikieli Python alkoi saada suosiota opetuskielenä 1990-luvun lopulla ja 2000-luvun alussa. Pythonin eduiksi on laskettu, että se muistuttaa hyvin paljon tavanomaisia ohjelmointikieliä [Zelle, 1999] ja se tukee monia ohjelmointiparadigmoja eli -suun-

tauksia, mukaan lukien olio-ohjelmointia. Pythonia opetuskielenä tukevat myöntävät kyllä, että opiskelijoiden tulisi myöhemmin oppia muita tavanomaisia ohjelmointikieliä, mutta ovat sitä mieltä, että ohjelmoinnin aloittamiseen Python on yksi parhaista kielistä.

5. Opetuskielten kärkeä

On tullut selväksi, että on olemassa ohjelmointikieliä, joiden katsotaan olevan muita parempia ohjelmoinnin aloittamiseen. Syitä näihin valintoihin on monia: kielen yksinkertaisuus, mutta toisaalta myös monipuolisuus; saatavilla oleva dokumentaatio sekä ohjelmointiympäristöt; kielen suosio kaupallisissa sovelluksissa. Jos jotakin kieltä käytetään laajalti kaupallisissa sovelluksissa, opinahjot katsovat olevansa velvoitettuja opettamaan tätä kieltä, jotta opiskelijoilla on paremmat valmiudet työelämään.

Joidenkin mielestä ensimmäisen ohjelmointikielen ei tarvitse olla varsinainen ohjelmointikieli, sillä sen avulla on vain tarkoitus saada opiskelijat innostumaan ja ymmärtämään perusasiat [Taylor, 1977; Zelle 1999]. Toisten mielestä taas on hyvä saada opiskelijat alusta asti ”oikean” ohjelmointikielen pariin, jotta esimerkiksi olio-ohjelmoinnin periaatteet iskostuvat aloittelijan mieleen [Zolman].

Wexelblat [1980] vertaa ensimmäisen ohjelmointikielen vaikutusta ohjelmointitaitoihin äidinkielen vaikutukseen ihmisen ajatuksenkulkuun. Hänen mukaansa ensimmäinen ohjelmointikieli vaikuttaa pysyvästi henkilön ohjelmointitapaan, ja tämän vuoksi ensimmäisen kielen valinnassa tulisi olla huolellinen.

Seuraavissa kohdissa käsitellään kieliä, jotka ovat tulleet esille nykyajan suosituimpina kielinä ohjelmoinnin aloittamisessa.

5.1. C++

C++ on C-kielen pohjalta kehitetty ohjelmointikieli, joka tukee monia ohjelmointiparadigmoja. Tämä on luettu sekä sen hyödyksi että haitaksi [Brilliant and Wiseman, 1996]. Moni tukee C++-kielen käyttöä ohjelmoinnin opetuksessa, koska sen avulla voidaan ensin opetella proseduraalista ohjelmointia ja myöhemmin opettaa samalla kielellä olio-ohjelmoinnin periaatteita. Sama asia luetaan myös C++:n haitaksi, sillä koska C++ ei ole puhdas olio-ohjelmointikieli, se ei mitenkään pakota käyttäjää käyttämään luokkia. [Kölling *et al.*, 1995]

Opiskelijoiden on siis helpompaa tehdä virheitä olio-ohjelmointia opetellessa huomaamattaan.

5.2. Java

Javan sopivuutta ohjelmoinnin aloittamiskieleksi perustellaan muun muassa sillä, että Javasta on jätetty pois C- ja C++-kielistä tutut osoittimet ja moniperintä, jotka aiheuttivat ongelmia aloitteleville ohjelmoijille. [Hadjerrouit, 1998; Gupta, 2004] Lisäksi Javassa on automaattinen roskien keruu, eli käyttäjän ei tarvitse huolehtia muistinhallinnasta. Näiden muutosten avulla päästään eroon C++-kielen suurimmista ohjelmointivirheiden lähteistä. Osoittimet ovat kuitenkin tärkeitä ja hyvin tehokkaita välineitä ohjelmoinnissa ja niiden hallitseminen on hyödyllistä kaikille ohjelmoijille.

Javan haitoiksi on luettu sen ”liiallinen sanojen käyttö” (excess verbosity) [Gupta, 2004] ja perustoimintojen syntaksin monimutkaisuus [Hadjerrouit, 1998; Ross, 2000]. Hadjerrouitin mukaan aloittelevilla ohjelmoijilla on ollut vaikeuksia esimerkiksi Javan puolipisteiden sijoittelun ja toistorakenteiden kanssa.

5.3. Python

Python on alun perin kehitetty skriptikieleksi, mutta tietokoneiden tullessa nopeammiksi ja graafisten käyttöliittymien tullessa tärkeämmiksi, on siitä tullut varteenotettava kilpailija ”oikeille” ohjelmointikielille [Zelle, 1999].

Pythonin syntaksi on yksinkertainen: lauseet loppuvat rivinvaihtoon, puolipisteitä ei tarvita. Myöskään Javasta tai C++:sta tuttuja aaltosulkeita ei tarvita esimerkiksi if-rakenteen aloittamiseen ja lopettamiseen, vaan sen koko määräytyy sisennysten mukaan. Näin Python myös pakottaa käyttäjät sisentämään koodia, mikä taas tekee koodista luettavampaa [Donaldson, 2003]. Python myös tukee useita ohjelmointiparadigmoja, mukaan lukien olio-ohjelmointia vähän samalla tavoin kuin C++.

Pythonin puutteiksi luetaan sen tehottomuus verrattuna varsinaisiin ohjelmointikieliin. Zelle kuitenkin hylkää tämän asiaankuulumattomana, kun kyseessä ovat ohjelmointia aloittamassa olevat opiskelijat. Hänen mukaansa opiskelijoiden tekemillä ohjelmilla ei ole aikavaatimuksia ja niitä tuskin ajetaan muutamaa kertaa useammin, joten niiden tehokkuudella ei ole suurta merkitystä.

6. Ei sittenkään kieli, vaan opetus?

Mitä jos ensimmäisen ohjelmointikielen valinta ei olekaan tärkeää? Ehkä ohjelmoinnin oppimiseen vaikuttaakin enemmän se, miten sitä opetetaan. Ainoa asia, josta ollaan samaa mieltä kielen suhteen on, että ensimmäinen ohjelmointikieli ei saa olla mammuttimainen kieli, jolla voi tehdä kaikkea mahdollista, mutta jonka oppiminen vie vuosia [Howell, 2003]. Tällaisia kieliä lukuun ottamatta melkein kaikilla kielillä voidaan opettaa ohjelmoinnin perusteet.

Howell [2003] on sitä mieltä, että ohjelmoinnin perustana on ongelmanratkaisu, eikä kielen syntaksin opetteleminen. Ulloa [1980] käsittelee samantapaista lähestymistapaa. Ensin opiskelijoita opetetaan ratkaisemaan ongelmia esimerkiksi pseudokoodia käyttäen. Näin opiskelijoiden ei tarvitse käyttää aikaa yksityiskohtien kuten muuttujien määrittämisen tai tietorakenteiden pohtimiseen, vaan he voivat keskittyä algoritmin rakenteeseen. Vasta peruskurssin loppupuolella algoritmeja alettaisiin muuttaa toimiviksi ohjelmiksi.

Ohjelmoinnin aloituskurssien yhteydessä on kielen valinnan lisäksi keskusteltu ohjelmointiparadigman valinnasta. Böszörményin [1998] mielestä ei ole olemassa yhtä parasta ohjelmointiparadigmaa. Hänen mielestään yliopistoissa ei pitäisi opettaa vain olio-ohjelmointia, joka on nykyaikana suosituimpia ohjelmointiparadigmoja, vaan opiskelijoille pitäisi opettaa muitakin käytäntöjä. Vaikka yhtä tapaa pidetään nyt parhaana, ei aina ole ollut niin ja tuskin niin tulee olemaan tulevaisuudessakaan. Böszörményi myös vastustaa ohjelmoinnin aloittamista nimenomaan olio-ohjelmoinnilla, sillä se kasaa turhia paineita opiskelijoille, jotka eivät välttämättä ymmärrä olio-ohjelmoinnin periaatteita kovinkaan helposti.

Huonon opetuksen seuraukset ovat huomattavasti suuremmat kuin huonon ensimmäisen ohjelmointikielen [Wexelblat, 1980]. Opiskelija ei ole oppinut erottamaan itse ratkaisua ongelmaan siitä, miten ratkaisu on tehty tietyssä kielessä. Usein tehty virhe on siirtyä ongelmanasettelusta suoraan koodin kirjoittamiseen ilman minkäänlaista suunnitteluvaihetta. Tästä seuraa, että ohjelmoija aloittaa tekemällä pieniä osia ohjelmasta ja toivoo, että saa kokonaisuuden toimimaan jotenkin.

Howell [2003] kertoo yhden pidetyimmistä ja tehokkaimmista oppimiskeinoista olleen valmistellut “labrat” eli harjoitustunnit. Opiskelijat oppivat ohjelmointia ja ohjelmointikieltä paremmin käymällä läpi valmiita esimerkkejä ja ohjelmoimalla itse opettajan ollessa läsnä vastaamassa kysymyksiin. Myös palautteen saanti palautetuista harjoituksista on tärkeää. Jos palaute saadaan no-

peasti ja ennen seuraavan harjoituksen palauttamista, on hyvin todennäköistä, että mahdolliset virheet eivät pääse toistumaan ja niistä päästään helpommin eroon.

7. Yhteenveto

Ohjelmoinnin opettamiseen käytetyn ohjelmointikielen valintaan on alusta asti vaikuttanut ohjelmointikielten menestys kaupallisissa sovelluksissa. Jos kieltä on alettu käyttää yleisesti, työnantajat ovat vaatineet sen osaamista työntekijöiltään. Tämän vuoksi opinahjot ovat katsoneet velvollisuudekseen opettaa kieltä, joka auttaa opiskelijoita työnsaannissa. Toisaalta monet ovat sitä mieltä, että ohjelmoinnin aloituskielen ei tarvitse olla laajassa käytössä oleva ohjelmointikieli. Heidän mielestään aloituskielen tulisi olla sellainen kieli, joka on helppo oppia, jotta opiskelijat voivat keskittyä ohjelmoinnin perusasioihin kuten ongelmanratkaisuun ilman, että heidän tarvitsee miettiä kielen tarkkaa syntaksia.

Ohjelmointikielellä ei välttämättä ole väliä, jos ohjelmoinnin opetus on muuten kunnossa. On ollut keskustelua ohjelmointiparadigman ja oikeanlaisten opetustekniikoiden valinnasta. Ohjatut ohjelmointiharjoitukset ovat osoittautuneet tehokkaiksi ja pidetyiksi, ja nopea palautteen saanti tehdyistä harjoituksista on tärkeää virheiden kitkemiseksi.

Viiteluettelo

[Brilliant and Wiseman, 1996] Susan S. Brilliant and Timothy R. Wiseman, The first programming paradigm and language dilemma. In: *Technical Symposium on Computer Science Education, Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, (1996), ACM Press, 338-342.

[Böszörményi, 1998] László Böszörményi, Why Java is not my favorite first-course language. *Software - Concepts & Tools* **19**, 3 (December 1998), 141-145.

[Cecchi *et al.*, 2003] Leonardo Cecchi, Pilu Crescenzi and Gaia Innocenti, C : C++ = JavaMM: Java. In: *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java*, (2003), Computer Science Press, 75-78.

[Cunniff *et al.*, 1986] N. Cunniff, R. P. Taylor and J. B. Black, Does programming language affect the type of conceptual bugs in beginners' programs? A

comparison of FPL and Pascal. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (1986), ACM Press, 175-182.

[Donaldson, 2003] Toby Donaldson, Python as a First Programming Language for Everyone. Presented at Western Canadian Conference on Computing Education 2003. Available as http://www.cs.ubc.ca/wccce/Program03/papers/Toby.html. Checked 15.7.2007.

[Gupta, 2004] Diwaker Gupta, What is a good first programming language? *Crossroads* **10**, 4 (August 2004), 7.

[Hadjerrouit, 1998] Said Hadjerrouit, Java as first programming language: a critical evaluation. *ACM SIGCSE Bulletin* **30**, 2 (June 1998), 43-47.

[Howell, 2003] Kris Howell, First computer languages. *Journal of Computing Sciences in Colleges* **18**, 4 (April 2003), 317-331.

[King *et al.*, 1978] P. R. King, G. Cormack, G. Dueck, N. Jung, G. Kusner and J. Melnyk, MABEL: A Beginner's Programming Language. *ALGOL Bulletin* **43** (December 1978), 54-83.

[Kölling *et al.*, 1995] Michael Kölling, Bett Koch and John Rosenberg, Requirements for a first year object-oriented teaching language. In: *Technical Symposium on Computer Science Education, Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, (1995), ACM Press, 173-177.

[Ross, 2000] Rocky Ross, Going Backwards: Introductory Programming Languages. *SIGACT News* **31**, 4 (December 2000), 65-73.

[Taylor, 1977] Robert P. Taylor, Teaching programming to beginners. In: *Technical Symposium on Computer Science Education, Proceedings of the Seventh SIGCSE Technical Symposium on Computer Science Education*, (1977), ACM Press, 88-92.

[Ulloa, 1980] Miguel Ulloa, Teaching and learning computer programming: a survey of student problems, teaching methods, and automated instructional tools. In: *ACM SIGCSE Bulletin* **12**, 2 (July 1980), 48-64.

[Wexelblat, 1980] Richard L. Wexelblat, The consequences of one's first programming language. In: *Symposium on Small Systems, Proceedings of the 3rd ACM SIGSMALL symposium and the First SIGPC Symposium on Small Systems*, (1980), ACM Press, 52-55.

[Zelle, 1999] John M. Zelle, Python as a First Language. Presented at 13th Annual Midwest Computer Conference, March 1999. Available as Wartburg College. Available as http://mcsp.wartburg.edu/zelle/python/python-first.html. Checked 15.7.2007.

[Zolman] Leor Zolman, Tech Trends - Industry Articles. Why Java should be your first programming language [online]. Thomson Course Technology. Available as http://www.course.com/techtrends/java_0899.cfm. Checked 15.7.2007.

LTSP-päätejärjestelmä yrityksissä ja kouluissa

Tuomas Tauriala

Tiivistelmä.

Tässä tutkielmassa tutustutaan Linux Terminal Server Project -projektin toteuttamaan, avoimeen lähdekoodiin perustuvaan LTSP-päätejärjestelmään sen toimintaperusteisiin sekä yleisesti Linux-käyttöjärjestelmän soveltuvuuteen yritys- ja koulukäytössä.

Avainsanat ja -sanonnat: päätejärjestelmät, Linux, thin client

CR-luokat: C.2.4, C.5.5

1. Johdanto

Koulut ja yritykset asettavat sähköisille työympäristöilleen jokseenkin samanlaisia yleisvaatimuksia: työympäristön on oltava ennen kaikkea luotettava, turvallinen ja helppokäyttöinen. Lisäksi työympäristön ylläpito ja valvonta on oltava helppokäyttöistä ja työympäristön jatkokehitys sekä käyttötuki on oltava taattu vuosiksi eteenpäin. Nämä vaatimukset ovat perinteisesti kallistaneet vaa'an kaupallisten ratkaisujen puolelle, mutta 1990-luvun alussa syntyneen avoimen Linux-käyttöjärjestelmän yleistyessä ovat myös avoimeen lähdekoodiin perustuvat ratkaisut alkaneet saada jalansijaa omistautuneiden vapaaehtoisprojektien tukemina.

Suuret tietojärjestelmät, joiden päätehtävä on sähköisen käyttöympäristön tarjoaminen (työpöytäjärjestelmät), voidaan jakaa karkeasti kahteen eri ryhmään sen mukaan, mihin järjestelmän yksikköön kohdistuu sovellusten suoritustaakka.

Työpöytäkoneiden suorituskyvyn räjähdysmäisen kasvun tuloksena nykypäivänä kouluissa ja työpaikoilla ylivoimaisesti yleisin ratkaisu on nk. fat client -järjestelmä, jossa sovelluksia ajetaan työasemakohtaisesti - vain kriittiset tieto-resurssit ovat yhteisillä palvelimilla. Tämä ratkaisu asettaa kuitenkin resurssi- ja kustannuspaineita järjestelmän ylläpitoon: ylläpitäjien on huolehdittava kaikkien työasemien huoltamisesta joko yksi kerrallaan, tai työasemien on oltava niin yhtenäisiä että niiden asentamiseen voidaan käyttää yhtenäistä levyimagea. Ratkaisuna ongelmaan on suosiotaan lisäämässä jo 1980-luvulla laajalti käytössä ollut thin client -konsepti, jossa käyttäjät suorittavat sovelluksensa yhteisellä palvelimella päätelaitteiden kautta. Tässä ratkaisussa päätelaitteita ei tarvitse erikseen päivittää - niissä ei tarvitse olla edes massamuistia, vaan yksittäinen

päätelaite voi käynnistettäessä ladata ohjelmistonsa verkon kautta keskuspalvelimelta.

Tässä tutkielmassa pohditaan siis paitsi LTSP-päätejärjestelmää, myös yleisessä mielessä thin client -järjestelmän etuja nykyiseen fat client -arkkitehtuuriin verrattuna.

2. Työpöytäjärjestelmäratkaisujen perusjako

Työpöytäjärjestelmä on tietojärjestelmä, jonka pääasiallinen käyttötarkoitus on tarjota käyttäjilleen sähköinen työympäristö sovelluksineen. Työpöytäjärjestelmät voidaan karkeasti jakaa thin client ja fat client -järjestelmiin sen mukaan, mihin järjestelmän perusyksikköön kohdistuu sovellusten suoritustaakka.

1980-luvulla, ennen kuin varsinaisia graafisia käyttöliittymiä oli olemassakaan, tietokoneet olivat suuria ja kalliita. Tietokoneiden ja tietojärjestelmien yleistyessä yhä useampi kuitenkin tarvitsi tietokonetta työssään. Ratkaisuna ongelmaan kehitettiin päätejärjestelmät, joissa käyttäjät käyttivät muutamaa suurta keskuskonetta yksinkertaisten ja halpojen päätteiden kautta. Päätteet tarvitsivat vain vähän tai eivät lainkaan ylläpitoa, joten järjestelmän ylläpitäjät saattoivat keskittyä palvelinkoneiden hoitamiseen. Päätekäyttö oli aikaisempaan käyttöön verrattuna erittäin nopeaa ja kustannustehokasta. [Van Vleck, 2001]

1990-luvulla mikroprosessorit alkoivat pienentyä ja sen myötä työpöytätietokoneiden tehot alkoivat kasvaa. Keskuspalvelimet kävivät nopeasti liian hitaiksi verrattuna tehokkaisiin ja halpoihin työpöytäkoneisiin. Lisäksi markkinoille tuotiin graafisia käyttöjärjestelmiä kuten Applen MacOS sekä Microsoftin Windows, jotka yksittäinen käyttäjä saattoi vaivattomasti asentaa omaan tietokoneeseensa ja käyttää koneellaan haluamiaan sovelluksia. Palvelimista saatiin enemmän irti keskittämällä niihin ainoastaan yhteisiä, kriittisiä tietoresursseja.

Fat client -järjestelmän heikkoudet havaittiin jo hyvin varhaisessa vaiheessa, kun yksittäisiä työasemia piti alkaa ylläpitää. Tätä varten piti usein palkata lisää henkilökuntaa. Käyttöjärjestelmien alttius tietokoneviruksille sekä peruskäyttäjien tietotaidon puute erityisesti tietoturvan osalta kohdisti myös uusia vaaroja tietojärjestelmiin.

Kun 2000-luvun puolella mikroprosessorien tehokehitys on jälleen hidastumassa, ovat moniytimiset palvelimet ja raskaat moniajojärjestelmät jälleen tuomassa tehoedun palvelinten puolelle. Lisäksi paluuta päätepohjaiseen järjestelmään on tukemassa yritysten vanhin motivoija, raha. Ilmaiset, pelkästään ylläpidollisia kustannuksia asettavat avoimen lähdekoodin järjestelmät tarjoavat erityisesti pienistä resursseista kärsiville organisaatioille houkuttelevan pohjan tehokkaan työpöytäjärjestelmän tarjoamiseen käyttäjilleen. Myös jatkuvasti

kasvava virusten ja tietoturvaongelmien määrä motivoi organisaatioita etsimään nykyistä turvallisempia käyttöympäristöratkaisuja.

Nykyinen päätejärjestelmä kilpailee skaalautuvuudessa sekä käytön sekä ylläpidon helppoudessa: tietojärjestelmässä voi olla useita sovelluspalvelimia, joissa on kaikissa käytössä samat sovellukset. Istunnon aloittaessaan käyttäjä yhdistetään siihen sovelluspalvelimeen, jolla on eniten vapaita resursseja. Sovelluspalvelimien toimintahäiriöt eivät estä järjestelmän käyttöä, sillä vialliset palvelimet voidaan irrottaa jakelusta häiriön korjaamisen ajaksi. Päätejärjestelmä soveltuu niin kymmenen kuin tuhannenkin tietokoneen palvelemiseen ilman erityisten ohjelmistojen asentamista. Yksittäisiä työasemia ei tarvitse huoltaa (muuten kuin fyysisissä laitevioissa), koska pääte lataa ohjelmistot keskuspalvelimelta käynnistyessään.

Fat client -järjestelmässä perusongelmana on tietoturva sekä yhteisten resurssien jakaminen: organisaation on joka tapauksessa hankittava palvelin yhteisten tietoresurssien säilyttämiseen. Tämän lisäksi on huolehdittava jokaisen työaseman ylläpidosta. Se voi olla vaikeaa, varsinkin kun työasemalaitteistoa harvemmin uusitaan kerralla. Jos työasemalaitteisto on ominaisuuksiltaan vaihtelevaa, eivät ylläpitäjät voi käyttää yhteisiä levyimageja työasemien asennukseen. Tämä aiheuttaa merkittävää aikaresurssien tuhlausta [Connolly, 2004].

Tietoturva on nykyaikana erittäin tärkeää, sillä useimmat käyttäjät tarvitsevat Internet-yhteyttä. Tämän vuoksi tietokoneet ovat alttiina Internetistä tuleville hyökkäyksille. Tietoturvaongelmien paikkaaminen vaatii yleensä järjestelmän ohjelmistojen päivittämistä sekä käyttäjien valistamista. Fat client -työasemien päivittämisessä on edellä mainitut ongelmat, mutta vielä suurempi ongelma on peruskäyttäjien tiedon sekä osaamisen puute. Peruskäyttäjältä ei oikeastaan voida edes olettaa, että he tietäisivät kaiken, mitä nykyään turvalliseen Internetin käyttöön tarvitsee tietää. Tämän vuoksi järjestelmänvalvojat yleensä rajoittavat käyttäjien oikeuksia merkittävästi, mikä aiheuttaa ongelmia organisaatioiden sisällä. Tämä on hyvin ymmärrettävää, sillä peruskäyttäjää ei kiinnosta tietotekniikan tuottamat lieveilmiöt kuten tietoturvaongelmat - hän haluaa ainoastaan tehdä työnsä mahdollisimman tehokkaasti ja helposti.

3. Linux työpöytäkäytössä

GNU/Linux on Linus Torvaldsin vuonna 1992 alulle panema avoimeen lähdekoodiin perustuva käyttöjärjestelmä. Se perustuu vanhaan Minix-käyttöjärjestelmään, joka taas on Andrew Tanenbaumin vuonna 1987 lähinnä yliopistokäyttöön julkaisema Unix-käyttöjärjestelmävariantti.

Kuten kaikki Unix-pohjaiset järjestelmät, myös Linux on suunniteltu pääasiassa palvelinkäyttöön. Se muistuttaa edelleen rakenteeltaan suuresti Unix System V:tä tai BSD:tä, vaikkakin se sisältää paljon myöhemmin käyttöä helpotta-

maan tehtyjä lisätoimintoja kuten Plug 'n Play -tuen sekä huomattavasti Unixia kehittyneemmän graafisen toiminnallisuuden. Filosofialtaan Linux on myös hyvin paljon Unixin ja BSD:n kaltainen. [Garrels, 2003]

Suurin osa Linuxille tarjolla olevasta ohjelmistosta perustuu Unixin tavoin avoimeen lähdekoodiin. Tämä johtuu osaksi siitä, että ohjelmat on kehitetty GNU Projectin kehittämällä C-kääntäjällä tai siihen perustuvilla ohjelmilla. Tämän vuoksi useat ohjelmistot on useimmiten julkaistu GNU Public License -lisenssin alaisina, joka pakottaa kehittäjät julkaisemaan myös tulevat versiot sekä muunnokset avoimena lähdekoodina.

Aluksi Linux oli vain yksinkertainen käyttöjärjestelmäydin ja joukko yksinkertaisia sovellusohjelmia, jotka soveltuivat ainoastaan palvelinkäyttöön. Yleistymisen ja erityisesti graafisen käyttöliittymän (X11R6, XFree/86) kehityksen myötä Linuxiin alkoi kuitenkin ilmestyä myös paljon graafisia ohjelmia, sekä toisin kuin Applen ja Microsoftin vastaavissa ohjelmistoissa, useita eri vaihtoehtoja graafisiksi käyttöliittymiksi.

2000-luvulla Linux löi tiensä läpi lopullisesti yritysten palvelimiin saavutettuaan kaupallisia yrityksiä kuten Red Hat, SuSE ja Novell, jotka alkoivat tuottaa yrityksille Linux-pohjaisia ratkaisuja. Oppilaitoksille avoimen lähdekoodin järjestelmä on jo alhaisten kustannustensa vuoksi houkutteleva, mutta yrityksissä se on saavuttanut suosiota vasta, kun Linux-ratkaisuihin on alettu todella sijoittamaan.

Yleisimmin työpöytäkäytössä tarvitut sovellusohjelmat ovat toimistosovelluksia: tekstinkäsittely, taulukkolaskenta, esitysgrafiikka, kaavionpiirto sekä mahdollisesti jokin tietokantasovellus. Tällä hetkellä yleisin toimistosovelluskokonaisuus on Applen Mac OSX:llekin tarjolla oleva Microsoft Office. Linuxissa on mahdollista saada Microsoftin ohjelmistopaketti toimimaan Wine-ohjelman (Windows Emulator) avulla, mutta sille on tarjolla myös oma, avoimeen lähdekoodiin perustuva OpenOffice-toimisto -ohjelmapaketti. OpenOffice tarjoaa tällä hetkellä tekstinkäsittely-, taulukkolaskenta-, esitysgrafiikka-, tietokanta- sekä piirto-ohjelman ja matemaattisten kaavojen suunnitteluun tarkoitetun ohjelman. OpenOfficen sovellukset on lokalisoitu lähes 70 eri kielelle, ja käännöstyö on jatkuvaa. OpenOffice-sovellukset osaavat käsitellä paitsi omaa Open Document -formaattiaan, myös Microsoft Officen asiakirjoja sekä Adoben PDF-asiakirjoja.

Muita työpöytäkäytössä tarvittavia sovelluksia ovat yleensä piirto- sekä erilaiset verkkosovellukset. Linuxille ei tällä hetkellä ole saatavilla Adoben ohjelmistoja (InDesign, Photoshop, Pagemaker yms.), mutta sille on tarjolla GNU-projektin tuottama ohjelma nimeltä Gimp, joka ominaisuuksissa jatkuvan kehityksen myötä lähellä Photoshopin tasoa. Gimp osaa myös käsitellä lähes kaikkia nykyisin käytettyjä kuvaformaatteja.

Verkkosovelluksista Linuxille on tarjolla kaikki perussovellukset: Mozilla-projektin tuotteet (Firefox web-selain, Thunderbird-sähköpostiohjelma sekä Sunbird-kalenteriohjelma) ovat saatavilla natiivisti Linuxille.

Graafisia käyttöliittymiä on tarjolla useita, mutta yleisimmin käytetyt ja myös kattavimmat ovat KDE-projektin tuottama K Desktop Environment sekä Gnome-projektin tuottama Gnome. Myös XFCE-projektin tuottama Xfce4 on melko suosittu kevyempi työpöytämanageri. Useimmat nykyisistä Linux-jakeluista käyttävät yhtä tai useampaa edellä mainituista käyttöliittymistä.

Hyvän työpöytäjärjestelmän on tarjottava myös kattava ja helppokäyttöinen multimediatoiminnallisuus. Linuxissa on edelleen ongelmana erilaisten suljettujen ja kaupallisten video- ja ääniformaattien tuki: useimmat Linux-jakelijat kieltäytyvät liittämästä tukea suljetuille formaateille jakeluunsa sillä perusteella, että niiden liittäminen rajoittaisi esimerkiksi oppilaitosten ja yritysten mahdollisuutta käyttää kyseistä jakelua lisenssiongelmien vuoksi. Kohtuullisella osaamisella on mahdollista laajentaa Linux-työpöytäjärjestelmä tukemaan myös kaupallisia formaatteja, mutta tällöin on oltava selvillä mahdollisista lisenssikysymyksistä. Linuxille on tällä hetkellä natiivisti saatavilla Internetissä yleisesti käytetty Macromedian Flash-ohjelmisto.

Yksi fat client -järjestelmän ehdoton etu on käyttäjän mahdollisuus liittää työasemaansa rajaton määrä erilaisia henkilökohtaisia massamuisteja kuten kiintolevyjä, CD-asema sekä USB-muisteja. Näiden käyttö ei ole aiemmin ollut mahdollista päätejärjestelmissä sopivan rajapinnan puutteessa. Nykyisissä LTSP-järjestelmän versioissa on tuki päätekohtaisille USB-muisteille sekä muille oheislaitteille kuten CD-asema, ulkoinen kiintolevy, skanneri tai tulostin. Ulkoiset massamuistit liitetään automaattisesti käyttäjän kotihakemistoon, joten erillistä erikoisosaamista niiden käyttöön ei tarvita. Tukea päätekohtaisille lisälaitteille kehitetään myös jatkuvasti.

Tietoturvan osalta Unixin perilliset ovat aina olleet Microsoftin markkinajohtajatuotteita parempia, pelkästään jo niiden suhteellisen harvinaisuuden ja luontaisen turvallisuuden vuoksi. Linuxissa käyttäjä toimii aina rajatuin oikeuksin siten, että mahdolliset virukset tai vakoiluohjelmat eivät pysty asentumaan tietokoneeseen ilman käyttäjän myötävaikutusta. Tämä johtuu käyttöjärjestelmäytimen toimintafilosofiasta. Toisin kuin Windowsissa, peruskäyttäjää ei kannusteta käyttämään pääkäyttäjän oikeuksia muuten kuin ohjelmistoja päivitettäessä tai järjestelmän asetuksia muuttaessa. Nykyisissä Linux-jakeluissa näiden toimintojen hoitamiseen ovat jakelujen toimittajat kehittäneet sovelluksia, jotka hoitavat esimerkiksi ohjelmistopäivitysten hakemisen ja asentamisen automaattisesti.

Linuxille on tiettävästi olemassa muutamia viruksia, mutta niiden levinneisyys on erittäin vähäinen. Uusia tietokoneviruksia kehitetään vuodessa noin 10000 [Lo, 2006], joista yli 90 % on tehty Microsoft Windows -käyttöjärjestel-

mälle sekä sen ohjelmistoille. Linuxin vähäistä virusten määrää käytetään usein perusteena sille, ettei Linux tarvitse aktiivista virussuojausta. Tämä ei ole kuitenkaan enää nykypäivänä totta, ja antivirusohjelmien valmistajat tarjoavat ohjelmistojaan myös Linux-käyttöjärjestelmälle.

Windows-käyttöjärjestelmän sovellusten ylläpito on hankalaa johtuen Windowsin jo lähtökohtaisesti kaupallisesta luonteesta. Windows ei tarjoa keskitettyä ohjelmistonhallintaa, koska se olisi kaupallisessa järjestelmässä käytännössä mahdotonta toteuttaa. Ohjelmistojen tuottajia on valtava määrä, eivätkä läheskään kaikkia halua alistua Microsoftin sovellushallintajärjestelmään. Microsoftin sovellushallintajärjestelmään osallistuminen olisi hyvin todennäköisesti myös maksullista, mikä käytännössä kuihduttaisi ilmaisohjelmistojen tuottamisen kokonaan. Edes Microsoft ei kuitenkaan halua tätä, sillä se karkottaisi nuoret ja aloittelevat ohjelmistokehittäjät Windowsin piiristä. Ratkaisuna on järjestelmä, jossa yksittäisen ohjelmiston tuottaja on vastuussa myös ohjelmistonsa puhtaasta ja oikeaoppisesta asentamisesta. Reaalimaailmassa tämä on johtanut siihen, että asennusohjelmistojen laatu vaihtelee suuresti ja heikkolaatuinen asennus- tai poisto-ohjelma saattaa tuhota käyttöjärjestelmän ydinkomponentin tai jaetun kirjastotiedoston siten, että muut sovellusohjelmat lakkaavat toimimasta. Tätä puutetta kompensoimaan Microsoft on kehittänyt Windows File Protection -työkalun, mutta se ei tarjoa sataprosenttista suojaa. Lisäksi se kasvattaa hukkatilan määrää palauttamalla järjestelmään turhaksi käyneitä tukikirjastoja, joita mikään sovellus ei enää käytä.

Useimmissa nykypäivän Linux-jakeluissa sovellusohjelmien sekä käyttöjärjestelmäkomponenttien hallinta on toteutettu keskitetyllä työkalulla. Tämä on mahdollista, koska vapaan ohjelmiston filosofian mukaan kaikki ohjelmat tulisi olla ilmaiseksi ja vapaasti saatavilla siten, että kuka tahansa voi noutaa ohjelman lähdekoodin, kääntää sen itse ja pakata haluamaansa muotoon. Tämän ansiosta Linux-jakelijat ovatkin pakanneet ohjelmistot uudelleen itselleen sopivaan muotoon voidakseen toimittaa ne oman Linux-jakelunsa pakettihallintaohjelmiston kautta. Nykyään suosituimmat pakettihallintaohjelmistot ovat Red Hatin ja Novellin käyttämä RPM sekä Debian-projektin sekä Ubuntun käyttämä APT. Molemmat ovat erittäin pitkälle kehitettyjä ja osaavat asentaa, päivittää sekä poistaa paketit tarkasti. Molemmat valvovat myös tarkasti ohjelmistojen riippuvuuksia toisistaan, ja osaavat asentaa aina tarvittavat tukikirjastot järjestelmään. Voidaan siis sanoa, että Linuxissa sovellusohjelmien hallinta on Windowsia kehittyneemmällä tasolla ja käyttäjän näkökulmasta helppohoitoisempaa.

4. LTSP-järjestelmän käyttöönotto ja ylläpito

LTSP-järjestelmän käyttöönottoon on useita tapoja. Ubuntu GNU/Linux tarjoaa varianteissaan valmiin ja helppokäyttöisen perusasennuspaketin [UbuntuWiki, 2007]. Edubuntu-jakelussa Ubuntun paketoima LTSP-järjestelmä asennetaan oletuksena, muissa Ubuntu-varianteissa sen saa asennettua komennoilla

```
sudo apt-get install ltsp-server-standalone openssh-server
sudo ltsp-build-client.
```

Mikäli käyttöön tarvitaan kustomoidumpaa LTSP-järjestelmää, täytyy se asentaa käsin Linux-ympäristöön. LTSP-järjestelmän osat ovat

- DHCP-palvelin, joka jakaa päätteille IP-osoitteet sekä tiedon, mistä pääte voi käynnistyessään ladata ohjelmistonsa.
- TFTP-palvelin, joka tarjoilee päätteelle ohjelmistopaketin sen käynnistyessä.
- NFS-palvelu, joka tarjoaa käynnistyneelle päätteelle perusohjelmat joiden avulla pääte muodostaa yhteyden LTSP-sovelluspalvelimeen.
- X-palvelu, joka tarjoilee päätteille graafiset istunnot kyseisellä sovelluspalvelimella.
- Lisäksi on joukko pienempiä demoneja, jotka huolehtivat päätteisiin kytkettyjen massamuistien sekä oheislaitteiden toiminnasta sovelluspalvelimen kanssa sekä multimedian toistamisesta päätteessä.

LTSP-järjestelmän asentamiseen on olemassa useita dokumentaatioita, mutta yleensä kannattaa seurata joko käytössä olevan Linux-jakelun toimittajan dokumentaatiota tai LTSP-kehitysryhmän dokumentaatiota. Kaikki dokumentaatio on saatavilla verkossa ilmaiseksi. [LTSPWiki, 2007]

5. Linux kouluissa

Lukio- ja korkeakouluasteella Linuxia on kokeiltu jo muutama vuosi, koska ylempien asteiden oppilaitoksissa on usein kyseisestä käyttöjärjestelmästä luonnostaan kiinnostuneita opettajia ja oppilaita. Joissakin oppilaitoksissa Linuxin käyttö saattaa olla jopa kokonaan oppilasvetoista [Hällfors & Teräs, 2002]. Erityisen kiinnostavaksi Linuxin tekee sen ilmaisuus sekä lähdekoodin vapaa saatavuus. Perus- ja toisen asteen oppilaitokset joutuvat yleensä tekemään ATK-hankintojaan melko pienellä budjetilla, joten ohjelmisto- ja ylläpitokulujen minimointiin kiinnitetään paljon huomiota. Suurin este Linuxin laajamittaisemmalle käyttöönotolle onkin osaamisen puute: koulu haluaa ohjelmistoratkaisua tehdessään varmistua siitä, että ohjelmistolle on olemassa jatkuva kehitys ja lähituki. Jos opettajakunnassa ei löydy osaamista, sitä joudutaan ostamaan koulun ulkopuolelta. Oppilaiden varaan ei voi laskea jatkuvan vaihtu-

vuuden vuoksi - vaikka asiansa osaavia oppilaita olisikin saatavilla tiettyinä vuosina, voi tilanne muuttua.

Mikroluokkansa ohjelmistoratkaisua tehdessä koululla ei ole etusijalla niinkään käyttöjärjestelmä tai sen yleisyys, vaan enemmänkin sovellustarpeet. Ohjelmistoratkaisun on pystyttävä tarjoamaan kaikilta osin helppokäyttöinen kokonaisuus. Yleisimmin koulujen tietotekniikan opetuksessa tarvitaan toimistoohjelmia (tekstinkäsittely, taulukkolaskenta ja esitysgrafiikka) sekä Internet-asiakasohjelmia (WWW-selain ja sähköpostiasiakas). Taide- tai teknisten aineiden erityisopetus asettaa vielä lisävaatimuksia (kuvankäsittelyohjelmat ja sovelluskehittimet). Hällforsin ja Teräksen [2002, 4] mukaan nykyiset Linux-ohjelmistojakelut pystyvät tarjoamaan nämä toiminnot opetuskäyttöön riittävällä tasolla.

Pelkkä sovellusohjelmien saatavuus ei kuitenkaan vielä riitä. On muistettava, että valtaosassa maailman tietokoneista (ml. kaupalliset yritykset, valtion järjestelmät sekä yliopistot) on edelleen käytössä Microsoft Windows ja sen sovellukset. Nämä kaksi käyttöjärjestelmää eroavat toisistaan fundamentaalisesti syvemmillä tasoilla. Niiden sovellusohjelmistot eivät ole yhteensopivia keskenään, ellei kyseistä ohjelmistoa ole toteutettu molemmille käyttöjärjestelmille erikseen. Myös käyttöjärjestelmien toteutuksessa on filosofisia eroja. Nämä erot kuvastuvat myös sovellusohjelmiin. Esimerkiksi perusasteen koulun tavoite on opettaa tietokoneen peruskäyttöä. Lapsi voi helposti oppia käyttämään Linuxia, mutta hänen pitää pystyä hyödyntämään oppimiaan taitoja myös Windows-käyttöjärjestelmän käytössä. Tämä on kouluille suuri huolenaihe.

Lähes kaikki ohjelmistot julkaistaan ainakin ensiksi pääosin englanninkielisinä. Lokalisointi on aikaa vievä prosessi, ja se maksaa yleensä paljon. Linux, kuten sen isäjärjestelmät Unix ja Minix, ovat natiivisti englanninkielisiä. Linuxiin on nykyään tarjolla melko kattava suomenkielinen lokalisaatio, mutta yksittäisissä sovellusohjelmissa lokalisoinnin laatu saattaa olla heikkoa tai jopa olematonta. Erityisesti perusasteen kouluille on tärkeää, että sovellusohjelmistot ovat saatavilla oppilaiden äidinkielellä. Lukio- ja amk-tasolla tätä vaatimusta ei ole.

Suurin kustannus ohjelmistosta ei koidu ohjelmiston hankinta, vaan ohjelmistojen päivittämisestä ja uudelleenkoulutuksesta. Koululle on tärkeää, että (yksi tai jopa useampi) mikroluokallinen tietokoneita voidaan päivittää nopeasti uusimpiin ohjelmistoversioihin. Sekä Windows että nykyaikaiset Linux-jakelut mahdollistavat tämän automaattisten asennusominaisuuksiensa puolesta. Molemmilla järjestelmillä on mahdollista mennä myös astetta pidemmälle nk. thin client -ratkaisuihin, joissa yksittäinen työasema ei sisällä itse sovellusohjelmia, vaan ne suoritetaan keskuspalvelimella. Näin ohjelmistojen päivittäminen jokaiseen työasemaan ei ole tarpeen.

Paitsi oppimiseen, koulu tarvitsee sovelluksia myös hallinnon tarpeisiin. Tällä hetkellä Linuxille ei ole tarjolla kattavaa kouluhallinto-ohjelmistoa. Hällforsin ja Teräksen [2002, 5] mukaan suomalaisen Starsoft Primus -ohjelmiston palvelinosaa voidaan ajaa Linux-käyttöjärjestelmässä. Linux-käyttöjärjestelmän mukana tarjottavalla WINE-ohjelmistolla voidaan ohjelmistovalmistajan mukaan nykyään ajaa myös Primuksen asiakasohjelmaa [Starsoft, 2007].

Microsoftin käyttöjärjestelmät ovat aina kärsineet tietoturvaongelmista. Tämä on luonnollinen seuraus Windows- ja DOS-käyttöjärjestelmien suuren suosion vuoksi. Linux on markkinoilla suhteellisen uusi tulokas, ja se on rakenteellisten ominaisuuksiensa vuoksi huomattavasti turvallisempi kuin Windows. Toisin kuin yrityksissä, koulujen on varauduttava myös omien käyttäjien suorittamia hyökkäyksiä kohtaan. Linux on asiantuntevalle murtautujalle erityisen houkutteleva kohde. Perusasennuksen jälkeen konfiguroimaton Linux-käyttöjärjestelmä ei ole riittävän turvallinen sisäisten hyökkäyksien osalta.

Ulkomailla Linux on saanut ehkä vielä suurempaa suosiota kuin kotimaassaan. Erityisesti köyhissä maissa kuten Intiassa nähdään Linuxin potentiaali erittäin pienellä budjetilla toteutettavassa opetusympäristössä. Sen avulla oppilailla on mahdollisuus ilmaiseksi ja laillisesti saada käyttöönsä erittäin pystyviä ja käyttökelpoisia sovellusohjelmistoja. Heillä on myös mahdollisuus tuottaa itse uusia ohjelmia ja oppia näin uran kannalta ensiarvoisen tärkeä ohjelmoinnin taito [Biggs, 2003].

Linux-ohjelmistotalot näkevät käyttöjärjestelmän potentiaalin opetuskäytössä, ja ovat erityisesti ulkomailla tarjonneet osaamistaan jopa ilmaiseksi köyhiin maihin käyttöjärjestelmän suosion levittämiseksi. Kolmannen maailman maat saavat usein lahjoituksina vanhoja ATK-laitteita, joita voidaan käyttää vielä muutama vuosi Linuxin kanssa tietotekniikan perus- ja erityisopetukseen.

6. Linux yrityksissä

Linux-pohjaisia palvelinratkaisuja on yritysmaailmassa käytetty jo muutama vuosi. Uusimpana villityksenä ovat HP:n ja virtualisointiratkaisuja tarjoavan VMwaren allianssin tuottamat virtualisointi- ja high availability -järjestelmät. Kuitenkin työpöytäympäristönä Linux on melko epäsuosittu. Suurin syy tähän lienee edelleen melko konservatiivisessa yritysmaailmassa vallitseva epäluottamus Linuxin kykyihin varteenotettavana työpöytäkäyttöjärjestelmänä. Monet yritykset elävät edelleen fat client -järjestelmien ajassa, joten on ymmärrettävää että niille ajatus työasemien muuttamisesta Linux-koneiksi kuulostaa pelottavalta.

Linux-pohjaiset fat client -järjestelmät tuskin tulevat lyömään itseään läpi kaupallisella puolella, mutta käyttäjäystävälliset, turvalliset ja helposti ylläpidettävät Linux-päätepalvelinratkaisut saattavat herättää yritysten kiinnostuk-

sen. Kehitys on menossa hyvää vauhtia kaupalliseen suuntaan. Tällä hetkellä LTSP-järjestelmien mahdollisuuksista on olemassa jo useita menestystarinoita oppilaitospuolella ja yritysmaailma seurannee kehitystä jatkuvasti.

Yritykselle on ensiarvoisen tärkeää järjestelmän ylläpidon ja kehityksen jatkuvuus sekä tietoturva. Tietoturvaa Linux tarjoaa jo nykyisellään: Työntekijöiden pääsyä järjestelmään voidaan rajoittaa entistä helpommin keskitetyllä tunnushallinnalla ja palvelukohtaisella auktorisoinnilla. Koska yksittäinen työasema lataa vähät perussovelluksensa verkosta, työasemakohtaista päivitysprosessia ei tarvita. Myös riski yksittäisistä saastuneista työasemista on käytännössä olematon, koska käyttäjät eivät pysty asentamaan omia ohjelmiaan työasemiinsa. Kun kaikki sovellukset ajetaan keskitetyllä palvelimella, ohjelmien päivitys on helppoa ja kaikille käyttäjille voidaan tarjota välittömästi uusimmat versiot eri sovelluksista. Lisäksi voidaan kontrolloida paremmin käyttäjien toimintaa järjestelmässä ja näin estää haitallinen käyttö.

Suurin este LTSP-järjestelmien käyttöönotolle yrityksissä tuntuukin olevan kaupallisten ratkaisujen puute. Yritykset uskovat rahan voimaan, eivätkä useimmat ole valmiita palkkaamaan omaa tukihenkilökuntaa järjestelmän ylläpitoon siinä määrin kuin LTSP-järjestelmä nykyisellä kehitystasollaan edellyttää. Koska osaavaa konsultti- tai outsourcing-palvelua ei ole olemassa, yritykset tyytyvät odottamaan. Tässä olisikin markkinarako uudelle innovatiiviselle IT-liiketoiminnalle. Muutamat suomalaiset yritykset tarjoavatkin jo perustason mikroluokkaratkaisuja kouluille, mutta nämä tuskin vastaavat yritysten tarpeisiin.

Viimeisen ongelman asettavat yritysten kustomoidut ohjelmistot, jotka on usein toteutettu Windows-käyttöjärjestelmälle. Näiden toimivuus Linuxissa on suuri kysymysmerkki. Useimmiten ne eivät toimikaan, jolloin käyttöjärjestelmän vaihtaminen saattaa käydä mahdottomaksi. Toki samankaltaisia sovelluksia tehdään uusina Linuxillekin ja pelkästään Suomessa on kyseiseen tarpeeseen toimivia ohjelmistotaloja pilvin pimein, mutta ajatus kalliilla hankitun sovelluksen hylkäämisestä konseptimuutoksen tai periaatteen vuoksi ei ole missään nimessä kannattavaa. Niin kauan kuin yritykset pysyttelevät 90-luvulla kehitetyissä erikoissovelluksissa, Linuxilla tuskin on suurempaa suosiota työasemakäytössä.

7. Yhteenveto

GNU/Linux on lupaava tulokas työpöytäjärjestelmämarkkinoilla. Se tarjoaa Microsoftin ja Applen vastaavia tuotteita paremman tietoturvan, käyttäjähallinnon sekä ylläpitomahdollisuudet. Helppokäyttöisyydessä Linuxilla on kuitenkin vielä parannettavaa. Perinteisessä fat client-tyyppisessä järjestelmässä Linux tuskin tulee saavuttamaan suurempaa suosiota, mutta pääteperustaisissa

konsepteissa Linux tarjoaa erittäin pitkällä olevan pohjan perustason toimistokäyttöön sekä opetukseen.

Sovellusohjelmapuolella Linuxille, Windowsille ja Mac OSX:lle natiivisti toteutettu OpenOffice-paketti kilpailee perustoiminnoissa samalla tasolla Microsoftin Office-tuoteperheen kanssa. Myös verkkosovellusten osalta Linux ja Windows ovat samalla tasolla Mozilla-tuoteperheen kanssa. Erikoissovellukset ovat suurin ongelma, koska niiden toimivuus Linuxissa on epävarmaa, eikä Linuxille natiiveja toteutuksia ole juurikaan saatavilla.

LTSP-päätejärjestelmä tarjoaa helposti käyttöön otettavan, mutta kuitenkin skaalautuvan ja vapaasti muokattavissa olevan thin client -konseptin, joka on jo nyt saavuttanut suosiota eri asteen oppilaitoksissa. Erityisesti köyhissä maissa, joissa oppilaitoksilla on todella pieni budjetti, LTSP tarjoaa oppilaille mahdollisuuden täysipainoiseen tietokoneen käyttöön, jollaista heillä ei ennen ole ollut. Myös Suomessa useat koulut ovat toteuttaneet LTSP-projektikokeiluja [Välimaa, 2005] ja siten levittävät GNU/Linux-käyttöjärjestelmän suosiota tulevissa sukupolvissa.

Viiteluettelo

[Biggs, 2003] John D. Biggs, Linux in education: Putting Linux in classrooms around the world. *Linux Journal* 95/2002.

[Connolly, 2004] Jim Connolly, Face-off: Thin clients vs. fat clients. Available at http://searchwinit.techtarget.com/ originalContent/0,289142,sid1_ gci943333,00.html.

[Garrels, 2007] Machtelt Garrels, Introduction to Linux, 05/2007, sivut 7-9. Saatavissa http://www.tldp.org/LDP/intro-linux/intro-linux.pdf

[Hällfors ja Teräs, 2002] Panu Hällfors ja Arto Teräs, Linux kouluissa -työryhmän näkemyksiä Linuxin käytöstä Suomen kouluissa vuonna 2002. FLUG (Finnish Linux User Group). Saatavissa: http://www.flug.fi/linuxkouluissa/ raportit/linux-kouluissa-2002-raportti.pdf.

[LTSPWiki, 2007] LTSP Wiki, Integrating LTSP-5 to your favourite Linux distro, Available at http://wiki.ltsp.org/twiki/bin/view/Ltsp/IntegratingLtsp.

[Lo, 2006] Kwan Lo, Renew and Update Your Antivirus Software Today, Available at http://www.selfseo.com/story-11606.php.

[Starsoft, 2007] Starsoft Oy, Primus -kouluhallinto-ohjelmiston vähimmäisvaatimukset. Saatavissa http://www.starsoft.fi/home/index.php/fin/content/ view/full/3960.

[UbuntuWiki, 2007] UbuntuLTSP Quick Install, Available at https://help.ubuntu.com/community/UbuntuLTSP/LTSPQuickInstall.

[Van Vleck, 2001] Tom Van Vleck, Home Terminals, Multiplexed Information and Computing Systems 9.7.2001. Available at http://www.multicians.org/terminals.html.

[Välimaa, 2005] Alpo Välimaa, Noormarkun LTSP-projekti, Saatavissa http://edu.noormarkku.fi/ltsp/.