Tietojenkäsittelytiede **22** Joulukuu 2004 sivut 21–34
Toimittaja: Antti Valmari

Muuttujien roolitutkimus

Pauli Byckling & Petri Gerdt & Seppo Nevalainen
Joensuun yliopisto
Tietojenkäsittelytieteen laitos
{Pauli.Byckling, Petri.Gerdt, Seppo.Nevalainen}@cs.joensuu.fi

Tiivistelmä

Muuttujien roolit on uusi käsite, jota voidaan hyödyntää mm. ohjelmoinnin alkeisopetuksessa. Tässä tekstissä esittelemme muuttujien roolikäsitteen perusajatuksen, roolipohjaisen ohjelmakoodianimaattorin PlanAnin sekä kolme muuttujien roolitutkimuksen osa-aluetta.

1 Johdanto

Muuttujien tyypillinen käyttö ohjelmoinnissa on ohjelmoijasta tai ohjelmointikielestä riippumatta varsin yleislaatuista. Muuttujien käyttötavat eivät ole satunnaisia tai tilapäisratkaisuja, vaan tunnusomaisia kaavoja ja menetelmiä, ”planeja”, jotka esiintyvät ohjelmissa yhä uudelleen. Käytännössä vain muutama kaava riittää kuvaamaan lähes kaikkien ohjelmissa esiintyvien muuttujien käyttöä.

Ohjelmissa toistuvat menetelmät ovat tavallisesti asiantuntijoiden omaamaa nk. ”hiljaista” tietoa, jota kokeneet ohjelmoijat käyttävät ohjelmoidessaan [14]. Tavallisesti tämä tietämys kehittyy ohjelmoijalle ajan ja kokemuksen myötä varsinaisen opiskeluvaiheen jälkeen tai sen myöhäisissä vaiheissa. Ohjelmoinnin opetus painottuu pitkälti ohjelmontikielten rakenteisiin ja opiskelijat joutuvat opiskeluvaiheessa omaksumaan korkeamman tason rakenteita lähinnä esimerkkiohjelmista. Sajaniemen esittelemä *muuttujien roolit* on uusi käsite, jolla asiantuntijoiden muuttujien käyttöön liittyvää hiljaista tietämystä voidaan esittää muodossa, jota voidaan hyödyntää mm. ohjelmoinnin alkeisopetuksessa [16].

Muuttujien roolitutkimuksen lähtöajatuksena on ollut kehittää pieni, mutta silti kattava joukko rooleja, jolla voidaan konkreettisesti kuvata kaikki alkeistasolla tavattavat muuttujat ja jota olisi mahdollista käyttää ohjelmoinnin opetuksessa [14]. Alunperin roolit löydettiin proseduraalisia ohjelmia tarkastellen, mutta havaintojemme mukaan samaa roolikäsitettä voidaan lähes sellaisenaan soveltaa myös funktionaalisiin ohjelmiin sekä olio-ohjelmiin. Samat roolit esiintyvät eri paradigmoissa, mutta esimerkiksi funktionaalisessa ohjelmoinnissa ne ovat parametrien (eivätkä muuttujien) ominaisuuksia. Roolit ovatkin ohjelmointikielestä ja jopa ohjelmointiparadigmasta riippumaton käsite.

Muuttujien roolitutkimus keskittyi alkuvaiheissaan ohjelmoinnin alkeisopetuksen ympärille, mutta on kehittynyt kattamaan myös osa-alueita, jotka ajavat muita tarkoitusperiä. Roolikäsitettä voi-

daan soveltaa mm. alkeistason ohjelmissa esiintyvien virheiden selittämiseen, ohjelmien visualisointiin ja suurten ohjelmien ymmärtämiseen. Seuraavassa esittelemme kolme toisistaan riippumatonta roolitutkimuksen osa-aluetta, jotka ovat: roolit ja strateginen ohjelmointitietämys, attentio ja ohjelmatietämys sekä automaattinen muuttujien roolien tunnistaminen. Esittelemämme osa-alueet ovat samalla kirjoittajien väitöstyöaiheita. Ennen roolitutkimuksen käsittelyä esittelemme lyhyesti seuraavissa kohdissa roolikäsitteen perusajatuksen sekä tutkimusmenetelmiä, joita sovellamme roolitutkimuksessa.

2 Muuttujien roolit

Tässä luvussa esittelemme lyhyesti roolikäsitteen perusajatuksen sekä yksittäiset roolit ja niiden väliset suhteet. Lisäksi käsittelemme ohjelmointiin liittyvän tietämyksen lajeja ja esittelemme roolipohjaisen ohjelma-animaattorin, PlanAnin.

2.1 Roolikäsite

Ohjelman tietovirtaa käsitellään muuttujien avulla. Yksittäisen muuttujan tarkoitus on toimia ohjelmassa dynaamisena tietoalkiona, jolle on luonteenomaista saada uusia arvoja muiden muuttujien ja ulkoisten tapahtumien seurauksena. Muuttujan rooli kuvaa muuttujan luonnetta ja käyttäytymistä sen elinkaaren aikana. Jokainen rooli kuvaa yhtä stereotyyppistä muuttujan käyttöä.

Seuraavat esimerkit havainnollistavat muuttujien roolien perusajatusta:

Taulukon tulostus:

```
for i := 1 to 12 do
  write(taulukko[i]);
```

Muuttujan `i` rooli on askeltaja.

Kokonaismäärän kerääminen:

```
sadesumma:=0
...
sadesumma:=sadesumma+sade
```

Muuttujan `sadesumma` rooli on kokooja.

Taulukon tulostusesimerkissä muuttuja `i` saa arvoikseen ennalta tiedetyt arvot 1:stä 12:een, kun muuttujan arvoa kasvatetaan for-silmukassa yhdellä. Muuttuja käy läpi, eli askeltaa, ennalta tiedossa olevan jonon. Muuttujalla on siis ohjelmassa rooli, jota kutsumme askeltajaksi. Toisen esimerkin muuttuja, `sadesumma`, saa uuden arvon kun sen omaan entiseen arvoon lisätään toisen muuttujan arvo. Muuttuja siis kerää kokonaismäärää useiden yksittäisten arvojen summana. Tällaista muuttujaa kutsumme rooliltaan kokoojaksi.

Seuraavat kymmenen roolia riittävät kattamaan 99 prosenttia kaikista aloittelijatason imperatiivisissa ohjelmissa esiintyvistä muuttujista [16]:

Kiintoarvo Muuttujan rooli on kiintoarvo, jos sen arvoa ei muuteta sen jälkeen kun siihen on ensimmäisen kerran luettu tai sijoitettu arvo.

Askeltaja Askeltaja käy läpi arvoja jollain ennustettavalla tavalla.

Tuoreimman säilyttäjä Muuttuja on tuoreimman säilyttäjä, jos sen arvo on viimeisin jostakin tietystä joukosta läpikäyty arvo tai yksinkertaisesti vain arvo, joka on syötetty viimeksi.

Sopivimman säilyttäjä Sopivimman säilyttäjän arvo on ”paras” tai jollain muulla tavoin halutuin siihen asti läpikäydyistä arvoista, esimerkiksi suurin taulukosta löydetty luku.

Kokooja Kokoojan arvo kerääntyy kaikista siihen mennessä läpikäydyistä arvoista. Kokooja on esimerkiksi läpikäydyn kokonaislukutaulukon alkioiden sisältämistä arvoista yhteenlaskettu summa.

Muuntaja Muuntaja saa jokaisen uuden arvonsa aina samasta muilla muuttujilla suoritetusta laskutoimituksesta. Muuntaja siis tallettaa itseensä esimerkiksi laskutoimituksen välituloksen, jolloin sitä voidaan käyttää toisessa laskutoimituksessa tai tulostuksessa.

Yksisuuntainen lippu Yksisuuntainen lippu on Boolen muuttuja, joka ei saa enää alkuperäistä arvoaan sen jälkeen, kun se on kerran muuttunut. Yksisuuntaisella lipulla ohjataan esimerkiksi kontrollivuota tai tulostusta (kuten: `if (ok) then begin...`).

Seuraaja Seuraaja saa aina arvokseen jonkin tietyn toisen muuttujan vanhan arvon esimerkiksi taulukon tai listan läpikäynnissä (kuten: `edellinen=nykyinen; nykyinen=nykyinen->seuraava;` Muuttuja `edellinen` on rooliltaan seuraaja).

Tilapäissäilö Muuttuja on tilapäissäilö, jos sen arvoa tarvitaan aina vain hyvin lyhyen ajan. Tilapäissäilöä käytetään esimerkiksi lajittelujen vaihto-operaatioissa tallettamaan toinen vaihdettavista arvoista tilapäisesti.

Järjestelijä Järjestelijä on taulukko, jota käytetään siinä olevien tietojen uudelleen järjestämiseen sen jälkeen, kun taulukko on ensin alustettu joillakin arvoilla. Järjestelijä esiintyy tyypillisesti lajittelun yhteydessä.

Ylläolevat määritelmät ovat epätarkkoja ja kuvaavat ainoastaan kunkin roolin luonteen. Tarkat määritelmät samoin kuin ohjelmakoodiesimerkit kaikista rooleista löytyvät muuttujien roolien kotisivulta [15].

Kuvassa 1 on havainnollistettu muuttujien roolien keskinäisiä suhteita [6]. Literaali ja vakio ovat ohjelmointikielen rakenteita, muut kuvan solmut ovat rooleja. Roolien opettamisessa voidaan soveltaa kuvan järjestyksen mukaista konstruktiivista lähestymistapaa, jossa uutta tietoa rakennetaan aikaisemman, jo opitun tiedon päälle. Kuvan mukaan esimerkiksi tuoreimman säilyttäjä voidaan kuvata kiintoarvona, jonka arvon asetusta toistetaan silmukassa. Edelleen, mikäli silmukassa asetetut arvot muodostavat ennalta ennustettavan jonon, kuten esimerkiksi lukumäärän laskeminen tekee, on kyseessä askeltaja.

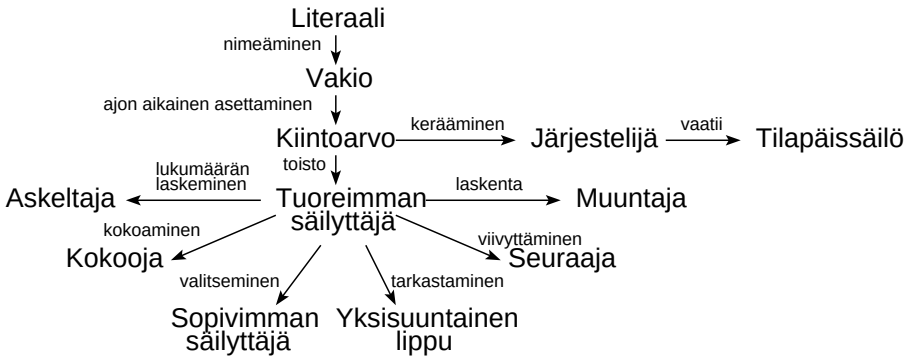
2.2 Ohjelmointitietämys

Ohjelmointiin liittyvä tietämys kattaa seuraavat kolme lajia:

Ohjelmointikieleen liittyvä tietämys Jonkin tietyn ohjelmointikielen syntaksi ja semantiikka (esimerkiksi kuinka sijoituslause kirjoitetaan ja mitä vaikutuksia sillä on).

Ohjelmatietämys Tietämys yksittäisestä ohjelmasta.

Ohjelmointitietämys Kuinka rakentaa ohjelmia käyttäen kyseiseen ohjelmointiparadigmaan liittyviä abstrakteja käsitteitä (kuten proseduraalisessa ohjelmoinnissa muuttujat, toisto jne.). Tärkein tietämyksen laji.



Kuva 1: Roolien välisiä suhteita [6].

Ohjelmoinnin alkeisopiskeluvaiheessa opiskelija joutuu kartuttamaan usean tason tietämystä. Ohjelmoinnin alkeisopetus keskittyy tavallisimmin ohjelmointikielen syntaksin ja semantiikan käsittelyyn, joka on matalimman tason tietoa. Korkeamman tason tietämys on usein opiskelijan oman mielenkiinnon ja itsenäisen opiskelun varassa, koska semanttista tietoa korkeampaa tietämystä ei yleensä eksplisiittisesti opeteta. Muuttujien roolit edustavat tätä korkeamman tason ohjelmointitietämystä. Roolikäsite on lisäksi yksinkertainen, tiivis ja intuitiivinen, joten se soveltuu hyvin opetusmetodiksi ohjelmoinnin alkeisopetukseen.

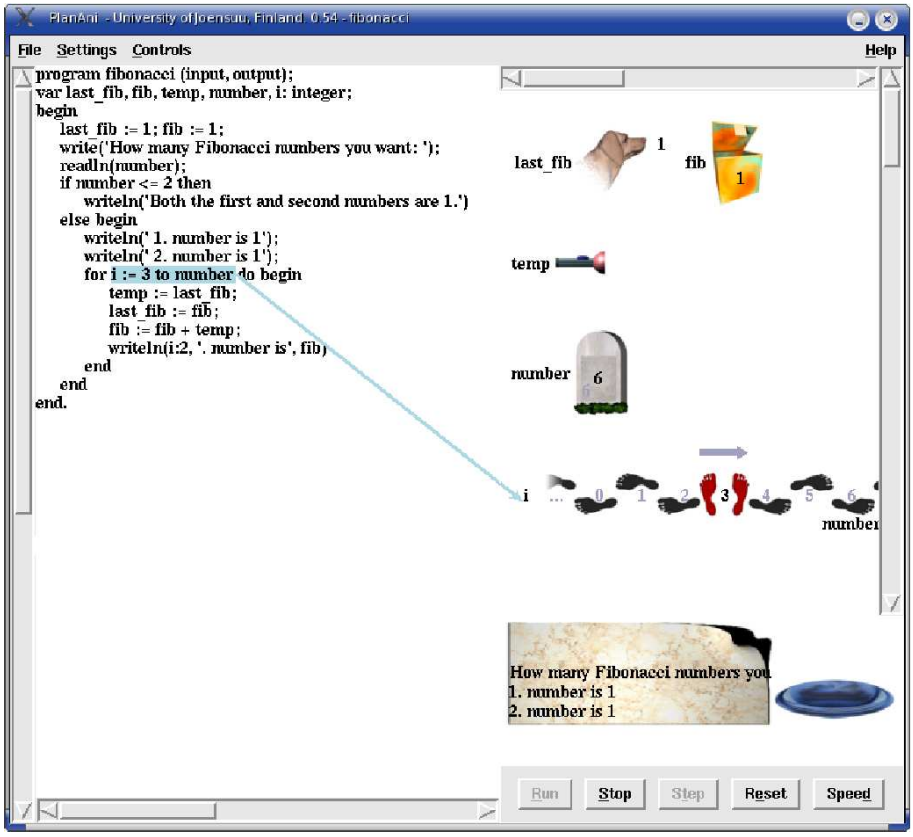
2.3 PlanAni

Muuttujien rooleja voidaan visualisoida Joensuun yliopistossa kehitetyn PlanAni-animaattorin [17] avulla. PlanAni on roolipohjainen ohjelma-animaattori, jossa jokaiselle roolille on oma visualisointinsa, roolikuvansa, jota käytetään kaikille tietyn roolin omaaville muuttujille. Muuttujien roolit ja muuttujiin kohdistuvat operaatiot esitetään animaattorin käyttäjälle kuvasymbolien avulla varsinaisen ohjelmakoodin rinnalla. Muuttujien lisäksi myös muuttujiin kohdistuvien operaatioiden animoinnissa hyödynnetään rooliriippuvaista esitystapaa. Animoinnin tavoitteena on tuoda ohjelmaan liitettävissä oleva asiantuntijatason hiljainen tieto vastaalkajien ulottuville.

Kuvassa 2 on PlanAni-animaattorin käyttöliittymän ruudunkaappaus. Käyttöliittymän vasemmassa puoliskossa näkyy animoitava ohjelmakoodi, jossa käsittelykohta on korostettu. Oikean puoliskon ylemmässä osassa sijaitsevat muuttujien roolikuvat. Vasemmalla sijaitsevan ohjelmakoodin käsittelykohta on yhdistetty nuolella tapahtumaan liittyvään muuttujaan. Oikeassa alakulmassa on tulostus- ja syöttöalueet, joita kuvaavat paperi ja lautanen. PlanAnin toimintaa hallitaan käyttöliittymän oikeassa alareunassa sijaitsevilla painikkeilla, joilla käyttäjä voi muun muassa käynnistää animaation ja säätää sen nopeutta.

PlanAni soveltuu hyvin ohjelmoinnin alkeisopetukseen ja sitä on käytetty ensimmäisen kerran vuonna 2002 järjestetyn opetuskokeilun yhteydessä. Kurssin harjoituksissa esimerkkiohjelmia visualisoitiin PlanAnin avulla. [16]

PlanAni on toteutettu Tcl/Tk:lla ja se on testattu Linux/Unix ja Windows NT



Kuva 2: PlanAni-animaattorin käyttöliittymän ruudunkaappaus.

-ympäristöissä. (PlanAni toiminee myös Mac-ympäristössä, testejä ei ole suoritettu tässä ympäristössä).

Lisätietoa muuttujien rooleista (mm. täydellinen roolien luettelo ohjelmakoodiesimerkkeineen) löytyy muuttujien roolien kotisivulta [15]. Myös PlanAni on vapaasti saatavilla samassa osoitteessa.

3 Tutkimusmenetelmistä

Tyypillisiä ohjelmoinnin psykologian tutkimuksissa käytettyjä menetelmiä ovat erilaiset kontrolloidut protokollakokeet, joita käytämme myös roolitutkimuksessa. Protokollakokeella tarkoitamme esimerkiksi koetta, jossa koehenkilön toiminta nauhoitetaan hänen suorittaessaan annetun tehtävän.

Keskeisiä kysymyksiä ovat mm. oppimisen laadun ja ohjelmontikäsitteiden ymmärtämisen mittaaminen. Ymmärryksen mittaamiseen ei ole yleispäteviä menetelmiä, mutta ohjelmoinnin tapauksessa menetelminä on käytetty esimerkiksi ongelmanratkaisutehtäviä (ts. ohjelmointitehtäviä), sekä vapaamuotoisia ohjelmakuvauksia [2]. Roolitutkimuksessa olemme käyttäneet molempia edellä maini-

tuista menetelmistä. Ohjelmointitehtävistä olemme varsinaisten valmiiden ohjelmien lisäksi keränneet nk. kirjoitusprotokollan, eli videoineet ohjelmakoodin kirjoitusprosessin tietokoneen ruudulta. Lisäksi olemme kehitelleet silmänliikekameran käyttöä ohjelmoinnin psykologian tutkimusmenetelmänä.

Yleispätevien menetelmien puuttumisen vuoksi kontrolloiduissa kokeissa saatujen tulosten analysointimenetelmien valinta ja kehittely vaatii harkintaa.

Ohjelmointitehtävien kirjoitusprotokollan analysointiin käytämme Ristin fokaalisen laajennuksen mallia [13]. Ristin mukaan ohjelmoija, joka ei omaa valmiita menetelmiä (plan) etenee ohjelman kirjoittamisessaan laajentaen yksittäisestä (tavoitteen kannalta olennaisimmasta) koodinpalasesta yksittäiseksi riviksi ja edelleen ohjelmointimenetelmäksi. Ohjelma siis syntyy nk. peruuttavan kehittämisen mukaisesti (backward development). Vastaavasti ohjelmoija, joka omaa tarvittavat menetelmät, pystyy palauttamaan menetelmän osat muististaan ohjelman ulkoasun mukaisessa järjestyksessä. Tällaista etenevää kehittämistä (forward development) kutsutaan skeeman laajennukseksi.

Ohjelmakuvausten analysoinnissa mielenkiinto ei kohdistu kuvausten oikeellisuuteen. Kuvauksen abstraktiotaso sekä tapa, jolla informaatio esitetään, ovat parempia mittareita kuvaamaan ihmisen mentaalimalleja, joten analysointiin tarvitaan erityisiä analysointimenetelmiä [2].

Useat aiemmat tutkimukset ovat soveltaneet ohjelmakuvausten analysointiin Penningtonin menetelmää [10], josta on muodostunut eräänlainen perusta ohjelmatietämyksen tutkimukselle. Roolitutkimuksen tarpeisiin olemme testanneet Goodin ohjelmakuvausten analysointimenetelmää (program summary analysis scheme) [3], joka pohjautuu Penningtonin menetelmään, mutta on hienojakoisempi ja tarkemmin määritelty. Menetelmä jakaantuu kahteen toisistaan riippumattomaan osaan: informaatiotyyppianalyysiin, jossa tarkastellaan mitä informaatiota ohjelmasta esitetään ja objektikuvausanalyysiin, jossa tarkastellaan miten yksittäisiin objekteihin kuvauksessa viitataan. Molemmissa analyyseissa luonnollisella kielellä annettu ohjelmakuvaus jaetaan pieniin osiin, jotka luokitellaan sisältönsä mukaan informaatiotyyppianalyysissa yhteentoista luokkaan ja objektikuvausanalyysissa seitsemään luokkaan. Varsinaisessa analyysivaiheessa ohjelmakuvauksen olemusta tutkitaan tarkastelemalla kuhunkin luokkaan sijoitettujen osien lukumääriä.

Olemme testanneet menetelmää luotettavuustestillä (inter-rater reliability test), jossa verrattiin kolmen toisistaan riippumattoman tutkijan samasta suomenkielisestä aineistosta suorittamien analyysien eroja ja yhtäläisyyksiä [2]. Tällä luotettavuustestillä pyrittiin testaamaan kyseisen analysointimenetelmän yleistettävyys sekä kartoittamaan jatkoa varten yhden analysoijan suorittaman analyysin luotettavuustaso. Luotettavuutta mitattiin laskemalla analysoijien välisten luokittelujen vastaavuuksia. Informaatiotyyppiluokittelussa päästiin 65,5 prosentin tarkkuustasoon vertailtaessa kaikkien kolmen luokittelijan luokituksia. Tapauksia, joissa vähintään kaksi luokittelijaa olivat valinneet samoin, oli 94 prosenttia kaikista tapauksista. Objektikuvausluokittelussa samat luvut olivat 73,2 ja 99,3.

Luotettavuustesti osoitti, ettei Goodin menetelmä ole täysin yksikäsitteinen, vaan vaatii lisää kehitystyötä. Luotettavuustestin yhteydessä pohdimme myös menetelmän pahimpia ongelmia. On huomattava, että luonnollisen kielen analy-

sointi on vaikeaa sen moniselitteisyyden, kontekstiriippuvuuden ja vivahde-erojen vuoksi. Lisäksi Goodin menetelmä on luotu englannin kieltä varten, joten menetelmän tarkkuus kärsii kielieroista. Nämä tosiseikat huomioiden totesimme Goodin menetelmän kuitenkin olevan tarkin olemassa oleva ohjelmakuvausten analysointimenetelmä ja näin olevan sopiva omiin tarkoitusperiimme. Olemme sittemmin soveltaneet Goodin menetelmää protokollakokeiden ohjelmakuvausten analysointiin.

Silmänliikekameroiden avulla voidaan rekisteröidä ihmisen katseen kohteita esimerkiksi tietokoneen näytöllä. Katseen kohteiden avulla voidaan arvioida ihmisen visuaalisen attention kohdistumista, vaikkakaan katseen kohde ja attention kohde eivät ole aina samoja. *Attentiolla* tarkoitamme tässä yhteydessä tarkkaavaisuutta tai huomiota, jota ihminen kohdistaa välittömiin aistien kautta saatuihin tai muistissa oleviin tietoihin. Ihmisen attention kohdistuminen tarjoaa informaatiota ihmisen kognitiivisista prosesseista, jotka ovat ohjelmoinnin psykologian tutkimuskohteita.

Silmänliikekameroita on aikaisemmin hyödynnetty erityisesti käytettävyystutkimuksissa sekä ihmisten haku- ja lukustrategioihin liittyvissä kognitiivisen psykologian tutkimuksissa. Ohjelmoinnin psykologian alaan liittyvissä tutkimuksissa silmänliikekameroita on hyödynnetty varsin rajallisesti. Tästä syystä suoritimme kontrolloidun kokeen [8], jonka avulla varmistuimme silmänliikekameroiden käyttökelpoisuudesta roolitutkimuksen yhtenä tutkimusmenetelmänä. Tutkimuksessamme tulemme käyttämään silmänliikekameroiden keräämää dataa tarkastellessamme ihmisten visuaalisen attention jakautumista näytöllä sijaitsevien merkityksellisten alueiden välillä.

4 Roolit ja strateginen ohjelmointitietämys

4.1 Tutkimuksen lähtökohta

Ohjelmointistrategiat ovat menettelytapoja, joita ohjelmoijat käyttävät lähestyessään ohjelmointiongelmaa. Strategiat voivat olla tiedostettuja, mutta yleisesti etenkin aloittelevilla ohjelmoijilla ne ovat alitajuisia toimintamalleja — ongelmanratkaisuun käytetään tavallisimmin sitä strategiaa, joka ”tuntuu hyvältä” tavalta lähestyä lähimpänä näköpiirissä olevaa osaongelmaa. Vasta-alkajien yleisin lähestymistapa onkin keskittyä kokonaisen ohjelmointiongelman sijaan sarjaan pienempiä osaongelmia aloittaen niistä yksittäisistä ohjelmariveistä, joita ensimmäiseksi havaitaan tarvittavan ongelman ratkaisussa.

Strategiat ovat usein myös opportunistisia. Tyypillisesti ongelmasta riippumatta pyritään käyttämään sitä strategiaa, jolla käsillä olevasta ongelmasta uskotaan päästävän yli mahdollisimman pienellä vaivalla. Aloittelevilla ohjelmoijilla kehittyneitä ohjelmointistrategioita on vielä vähän ja niiden soveltaminen on vaikeaa. Toiminnassa esiintyy usein jopa summittaista kokeilua — strategioiden pettäessä kokeillaan päästä nykytilasta lähemmäksi tavoitetilaa kirjoittamalla kokeilunomaisesti erilaisia ohjelmakoodirivejä, jotka eivät liity ongelman todelliseen ratkaisuun.

Ohjelmointistrategiat kehittyvät järjestelmällisemmiksi ohjelmointitietämyksen ja -menetelmien karttuessa. Ne ovat eri tason asioita kuin yksittäiset menetelmät eli nk. planit. Strategialla kuvataan sitä, kuinka ohjelmoija soveltaa omaamiaan yksittäisiä planeja tai yleistä ohjelmointija menetelmätietämystään. Mikäli menetelmiä ei ole, ovat myös strategiat epämääräisiä.

Roolien avulla asiantuntijoiden menetelmätietämystä voidaan opettaa vasta-alkajille ja näin kartuttaa menetelmävarastoa entistä aiemmassa opiskelun vaiheessa. Lisäksi roolien opetuskäytöstä saadut ensimmäiset tulokset osoittavat, että roolitietämys auttaa varsinkin ohjelmien syvärakenteen ymmärtämisessä [16]. Syvärakenteen ymmärryksen katsotaan liittyvän asiantuntijuuteen — analysoidessaan ohjelmaa asiantuntijat pystyvät ymmärtämään ohjelman syvärakenteen vasta-alkajien keskittyessä lähinnä pintarakenteen tarkasteluun. Roolitietämys tuo siis vasta-alkajien ohjelmointitietämykseen suoraan sellaisia menetelmiä, jotka tavallisesti muodostuvat alitajuisesti vasta myöhemmässä vaiheessa kokemuksen ja harjaantumisen myötä. Roolitietämyksen voidaan siis olettaa edistävän myös ohjelmointistrategioiden kehittymistä.

4.2 Tutkimuksen tavoitteet

Tutkimusta muuttujien roolien vaikutuksesta ohjelmointistrategioihin voidaan hyödyntää eri tavoin. Yksi tärkeimmistä kiinnostuksen kohteistamme on tutkia, millä tavoin käytettynä roolit soveltuvat parhaiten ohjelmoinnin alkeisopetukseen. Tähän päämäärään päästäksemme joudumme aluksi selvittämään roolikäsitteen todelliset käytännön vaikutukset opetusmetodina.

Jotta voisimme tutkia, millaisia vaikutuksia roolikäsitteellä on strategisen ohjelmointitietämyksen kehittymisessä alkeistasolla, joudumme ensiksi tutkimaan asiantuntijoiden omaamia strategioita. Tarkoituksenamme on siis selvittää millaisia muuttujien rooleihin liittyviä strategioita ohjelmoinnin asiantuntijoilla on ja voidaanko niitä opettaa. Edelleen mielenkiintomme kohdistuu siihen, onko olemassa joitain erityisesti vasta-alkajille soveltuvia strategioita roolien käyttöön liittyen, mitä nämä strategiat mahdollisesti ovat, miten ne löydetään ja kuinka niitä voidaan parhaiten käyttää opetuksessa.

4.3 Tutkimustyö ja eteneminen

Muuttujien rooleja käytettiin ensimmäisen kerran opetuksen apuna syksyllä 2002 järjestetyssä opetuskokeilussa [16]. Kokeilussa yliopistotasoisen ohjelmoinnin alkeiskurssin osallistujat jaettiin kolmeen ryhmään: perinteinen, rooli- ja animointiryhmä. Perinteiselle ryhmälle annettiin perinteistä luennoista ja harjoituksista muodostuvaa opetusta. Toisin sanoen perinteisen ryhmän opiskelijoita opetettiin tavalla, jolla koko kurssin opetus olisi järjestetty ilman kyseistä opetuskokeilua. Rooli- ja animointiryhmien opiskelijat osallistuivat yhdessä erillisille luennoille, joilla muuttujien roolit esiteltiin systemaattisesti kurssin edetessä. Rooleja käytettiin opetuksen apuna luennoilla esimerkkiohjelmissa sekä kurssiin liittyvissä pakollisissa harjoituksissa. Animointiryhmän harjoituksissa esimerkkiohjelmia havainnollistettiin lisäksi PlanAni-animaattorilla [18]. Rooliryhmän ja perinteisen ryhmän harjoituksissa esitettiin samat esimerkkiohjelmat käyttäen Turbo Pascal -ohjelmointiympäristön virheenjäljitintä.

Kurssin edetessä yhteensä 39 kurssille osallistunutta opiskelijaa osallistui protokollakokeisiin, joita järjestettiin kurssin puolessavälissä sekä kurssin lopussa. Protokollakokeet koostuivat kahdenlaisista tehtävistä: yksin suoritetuista ohjelman ymmärrystehtävistä sekä pariohjelmointitehtävistä [5]. Ohjelmointitehtävistä kerättiin varsinaisten pariohjelmoinnin tuloksena syntyneiden ohjelmien lisäksi tehtävän suorituksen aikainen kirjoitusprotokolla sekä puheprotokolla (oh-

jelman kirjoittaminen videoitiin ja pareja kehotettiin keskustelemaan ohjelmoinnin aikana ohjelmaan ja ohjelmointiin liittyvistä asioista). Ohjelmien ymmärrystehtävässä taas henkilöitä pyydettiin valmiiseen ohjelmaan perehdyttyään antamaan vapaa sanallinen ohjelmakuvaus ohjelman toiminnasta ja tarkoituksesta.

Strategiseen ohjelmointitietämykseen liittyvä tutkimustyö keskittyy tässä vaiheessa pitkälti juuri edellä kuvatun protokolladatan analysointiin. Koska roolikäsite on uusi ja ainutlaatuinen tapa jäsentää ohjelmointitietämystä, olemme tutkimuksen alkuvaiheessa keskittyneet lähinnä tulosten analysointiin soveltuvien menetelmien etsimiseen, testaamiseen ja kehittämiseen (katso luku 3). Olemme soveltaneet Goodin menetelmää protokollakokeiden ohjelmakuvausten analysointiin tarkoituksenamme löytää mahdollisia eroavaisuuksia ryhmien välillä. Goodin menetelmän myötä suoritettu tilastollinen analyysi ei kuitenkaan ole tuottanut merkittäviä tuloksia ryhmien välisistä eroista, joten kyseisen aineiston analysointi jatkuu edelleen. Seuraavaksi sovellamme ohjelmakuvauksiin kvalitatiivisempaa analyysia.

Edellä kuvatun analyysin ohella olemme analysointityössä keskittyneet myös pariohjelmointitehtäviin, joihin olemme tähän mennessä soveltaneet kvalitatiivisia analysointikeinoja pyrkien etsimään niitä ilmiöitä, joihin jatkossa kannattaa keskittyä. Olemme lähinnä tarkastelleet valmiissa ohjelmissa esiintyneitä virheitä sekä tarvittaessa puheprotokollaa päätelmien tukena. Analyysi on jo tuottanut mielenkiintoisia havaintoja, joihin keskitymme tutkimuksen jatkovaiheissa pyrkien vahvistamaan havaintoja myös tilastomenetelmin. Näistä havainnoista tulemme raportoimaan lisää lähitulevaisuudessa.

Pariohjelmointitehtävien kvantitatiiviseen analyysiin käytämme mukautettua versiota Ristin fokaalisen laajennuksen mallista. Roolikäsitteen tuntevat henkilöt omaavat valmiita muuttujien käyttöön liittyviä planeja, joiden kautta ohjelmointiongelman ratkaisua voidaan lähestyä. Roolituntemuksen omaavilla protokollahenkilöillä tulisi siis oletettavasti esiintyä enemmän etenevää kehittämistä kuin perinteistä opetusta saaneilla protokollahenkilöillä. Analyysissa Ristin menetelmää käytetään mukaillen. Siinä missä Rist tarkastelee ohjelmointia kokonaisuudessaan analysoiden ohjelmia sellaisenaan, tarkastelemme roolitutkimuksessa ohjelmointimenetelmiä yksittäisten muuttujien näkökulmasta. Näin ollen tämän tutkimuksen kannalta ”mielenkiinnoton” aineisto on tarkoitus jättää analyysin ulkopuolelle ja keskittyä tarkastelemaan sitä, kuinka protokollahenkilöt käyttävät muuttujiin liittyviä ratkaisuplaneja.

Pariohjelmointitehtävistä kerätyn ohjelmoinnin aikaisen puheprotokollan analysointiin emme vielä ole löytäneet soveliasta analysointimenetelmää. Vaikka puheprotokollan analysointiin on sinänsä olemassa runsaasti menetelmiä, on ohjelmoinnin aikaisen puheprotokollan kerääminen vielä harvinaista ja tarkoituksenmukaisten analysointimenetelmien kehittely vielä alkutekijöissään. Goodin menetelmä ei ainakaan sellaisenaan sovellu tällaisen puheprotokollan analysointiin. Puheprotokollaa onkin tähän mennessä käytetty ainoastaan kvalitatiivisen analyysin tukena. Tässä tarkoituksessa puheprotokolla toimii erinomaisena lisäinformaation lähteenä protokollahenkilöiden toiminnan tulkitsemisessa.

Edellä kuvattujen tutkimusmenetelmien avulla uskomme voivamme selvittää eroja vasta-alkajien ohjelmointitietämyksessä ja ohjelmointistrategioissa, mutta

olemme edelleen epävarmoja siitä, kuinka tarkasti näillä metodeilla voidaan mitata juuri niitä asioita, joihin kiinnostuksemme kohdistuu. Analysointimenetelmien sekä uusien koejärjestelyjen kartoittaminen ja kehitys jatkuu siis edelleen.

5 Attentio ja ohjelmatietämys

5.1 Tutkimuksen tavoitteet

Opetettaessa ohjelmointia vasta-alkajille voidaan hyödyntää erilaisia visualisointitekniikoita. Visualisointitekniikoiden avulla sekä ohjelmointikielen rakenteet että ohjelmarakenteet voidaan tehdä helpommin ymmärrettäviksi [4, 7]. Kolme ohjelmointiin liittyvän tietämyksen lajia on esitelty kohdassa 2.2. Ohjelmien visualisointia voidaan lähestyä minkä tahansa tietämyksen lajin kautta.

Petre ja Blackwell [11] esittävät, että visualisoinnin ei tulisi tapahtua ohjelmointikielen tasolla, koska yksittäisen ohjelmointiparadigman piirissä tapahtuva visualisointi, joka keskittyy ohjelmointikielen rakenteisiin, ei ole informatiivista. Tästä syystä tulisi suosia korkeamman tason ohjelmarakenteiden visualisointia [17].

Hyödynnettäessä visualisointitekniikoita täytyy ohjelmista valita ne piirteet tai ominaisuudet, joita tuodaan esille. Näillä visualisoinnin kohdistamiseen liittyvillä valinnoilla on vaikutusta visualisoinnin informatiivisuuteen. Visualisoinnit voivat erota toisistaan kohteiden valinnan lisäksi myös käytettävien symbolien ja erilaisten ulkoasuun liittyvien tekijöiden osalta. Esimerkiksi PlanAnissa roolia kiintoarvo havainnollistetaan hautakivisymbolilla. Ulkoasuun liittyviä tekijöitä ovat esimerkiksi käytettävien symbolien keskinäinen sijoittelu ja sijoittelu tietokoneruudulla.

Tutkimuksessamme tarkastelemme edellä esiteltyjä visualisointeihin liittyviä tekijöitä visuaalisen attention kohdistumisen kautta. Pyrimme etsimään vastauksia erityisesti seuraaviin keskeisiin kysymyksiin:

Missä määrin ja millä tavoin visualisoinnissa käytettävät erilaiset symbolit ja erilaiset ulkoasut vaikuttavat attention kohdistumiseen? Millainen vaikutus attention kohdistumisella on puolestaan muodostuvaan ohjelmatietämykseen?

Tutkimuksemme tavoitteena on kerätä yksityiskohtaista tietoa erilaisten roolikuvien, erilaisten roolikuvien sijoittelutapojen ja erilaisten roolikuviin kohdistuvien animaatioiden vaikutuksesta attention kohdistumiseen aloittelevien ohjelmoijien osalta. Tutkimme myös, kuinka nämä mahdolliset erot attention kohdistumisessa vaikuttavat ohjelmatietämyksen muodostumiseen.

5.2 Tutkimustyö ja eteneminen

Tutkimuksessamme käytämme PlanAni-animaattoria, jonka visuaalista ulkoasua vaihtelemme eri koetilanteissa. Koehenkilöiden katseen kohteet ruudulla mitataan silmänliikekameran avulla. Tällä tavalla keräämme tietoa koehenkilön visuaalisen attention kohdistumisesta. Koehenkilöiltä kerätään myös heidän käsityksiään tutkittavasta ohjelmasta. Tämä tapahtuu pääsääntöisesti ohjelmakuvausten avulla. Tarkastelemme ohjelmakuvauksia Goodin analysointimenetelmän avulla [3] ja pyrimme tällä tavalla tutkimaan ohjelmatietämyksen muodostumista. Goodin analysointimenetelmä on esitelty lyhyesti kohdassa 3.

Tammikuussa 2004 suoritimme kokeen [8], jossa vertailimme kolmen erilaisen silmänliikekameran soveltumista

ohjelmoinnin psykologian tutkimukseen. Kokeen tarkoituksena oli kerätä yleistä kokemusta silmänliikekameroiden käytöstä sekä analysoida, kuinka silmänliikkeiden tutkiminen soveltuu osaksi ohjelmoinnin psykologian tutkimuksen metodologiaa. Kokeesta saadut tulokset vahvistivat käsityksemme siitä, että silmänliikekameroiden avulla voidaan kerätä käyttökelpoista silmänliikedataa ohjelmoinnin psykologian tutkimusta varten.

Ensimmäinen visualisointeihin ja ohjelmatietämyksen muodostumiseen keskittyvä koe suoritetaan syksyllä 2004. Kokeessamme tulee olemaan koehenkilöinä kaksi aloittelevien ohjelmoijien ryhmää, joista toinen ryhmä tutkii ohjelmia Turbo Pascalin virheenjäljittimen avulla ja toinen ryhmä PlanAni-animaattorin avulla. Tarkastelemme koehenkilöiden visuaalisen attention kohdistumista ohjelmien tutkimisen aikana sekä niitä mentaalisia malleja, joita koehenkilöille syntyy tutkittavista ohjelmista. Tavoitteenamme on kokeen avulla selvittää, onko attention kohdistumisessa eroja eri työkaluja käyttäneiden ryhmien välillä.

6 Automaattinen roolien tunnistaminen

6.1 Tutkimuksen tavoitteet

Ihmisen kognitio ei ole eksaktia ja kognitiiviset rakenteet ovat epämääräisiä. Muuttujien roolit ovat ohjelmoijien käyttämiä kognitiivisia rakenteita, joilla ei ole tarkkaan määriteltyjä ominaisuuksia. Roolit ovat väljiä kuvauksia muuttujien käyttäytymisestä ja ne voivat poiketa toisistaan eri henkilöillä. Esimerkiksi matemaatikko osaa ennustaa Fibonaccin lukujonon tuottamat luvut ja mieltää lukujonon läpikäyvän muuttujan askeltajaksi. Fibonaccin lukujen määritelmää tuntematon vasta-alkaja ei välttämättä näe lukujen välistä yhteyttä ja voi antaa lukuja talletavalle muuttujalle roolin kokooja, koska muuttuja näyttää kokoavan aikaisemman laskennan tuloksia.

Ohjelmointikielet puolestaan ovat eksakteja. Ne perustuvat tarkkaan määriteltyihin kielioppeihin, jotta niitä on mahdollista tulkita tietokoneen avulla. Roolikäsite luo mielenkiintoisen mahdollisuuden tarkastella ihmisen kognitiota, koska eksaktilla ohjelmointikielellä tuotettuja ohjelmia on mahdollista analysoida automaattisesti. Automaattinen roolianalyysi on luonteeltaan epäeksaktia, koska kognitio ei ole täsmällistä.

Automaattinen roolianalyysi tutkii ohjelmia ja yrittää määrittää ohjelmakoodissa esiintyvien muuttujien roolit. Analyysin tavoitteena on löytää yhteys ohjelmakoodin ja ohjelmoijan kognitiivisten rakenteiden välille. Aiomme hyödyntää automaattista roolitunnistusta PlanAni-ohjelma-animaattorissa mahdollistaaksemme automaattisen ohjelmien animoinnin. Muita sovelluskohteitamme ovat laajojen ohjelmien ylläpitosovellukset ja roolitutkimuksen aputyökalut. Uskomme, että automaattista roolien tunnistamista voidaan hyödyntää myös erilaisissa virheenjäljitysympäristöissä ja ohjelmakoodin kirjoittamiseen käytettävissä sovelluksissa.

6.2 Tutkimustyö ja eteneminen

Kehitämme menetelmää löytää automaattisesti ohjelmakoodissa esiintyvien muuttujien roolit. Automaattinen roolianalyysi jakautuu kahteen vaiheeseen, jotka ovat oppimisvaihe ja tunnistusvaihe. Oppimisvaiheessa roolianalyysiä pohjustetaan luomalla rooli-informaatiolla varustettujen ohjelmien avulla tietokanta, jota käytetään automaattisessa tunnistamisessa. Tunnis-

tamisvaiheessa analyysisovellus saa syötteeksi ohjelmia, joiden muuttujille se etsii tietokannan avulla parhaiten sopivat roolit. Kuvailemamme automaattinen roolien tunnistus on luonteeltaan koneoppimistehtävä.

Ohjelmien tietovirta-analyysi on eräs keino analysoida ohjelmia [9]. Siinä ohjelma kuvataan tietovirtakaavion avulla: ohjelma jaetaan itsenäisiin osiin, joiden välillä kontrolli siirtyy ohjelman suorituksen aikana. Tutkimuskohteena ei ole suorituskelpoinen ohjelma, vaan ohjelmakoodi. Ohjelman käännösaikainen optimointi on tyypillinen tietovirta-analyysin sovelluskohde.

Automaattisen roolianalysaattorin teknisen toteutuksen ytimenä on kääntäjän [1] ja tietovirta-analyysisovelluksen yhdistelmä. Analysaattori käsittelee syötettään kuten kääntäjä jäsentäen ohjelmakoodin syntaksipuuksi ja muodostaen tietovirtakaavion. Tämän jälkeen se käyttää erilaisia tietovirta-analyysimenetelmiä tutkiakseen kuinka muuttujat syöteohjelmassa käyttäytyvät. Sovellus kerää jäsentämisen ja tietovirta-analyysin aikana tietämystä muuttujista. Kutsumme muuttujista kerättyä tietoa muuttujien tietovirtaominaisuuksiksi. Tietovirtaominaisuudet kuvaavat, mitä muuttujalle tapahtuu sen elinkaaren aikana, esimerkiksi kuinka usein muuttujan arvo muuttuu, kuinka muuttujan elinkaarensa aikana saamat arvot eroavat toisistaan ja riippuuko muuttujan arvo joidenkin muiden muuttujien arvoista.

Oppimisvaiheessa analysaattori saa syötteekseen ohjelmia, joissa muuttujien roolit on merkitty koodiin kommenttien sisään. Aluksi analysaattori kartoittaa ohjelman muuttujien tietovirtaominaisuudet. Sitten se poimii muuttujien roolit syöteohjelmasta ja yhdistää jokaisen muuttujan roolin sitä vastaaviin tietovirtaominaisuuksiin. Oppimisvaiheen lopuksi sovellus yleistää usean saman roolin omaavan muuttujan tietovirtaominaisuudet tietokantaan. Tietokantaan kootaan roolien kuvaukset tietovirtaominaisuuksien kautta ilmaistuna.

Tunnistamisvaihe muistuttaa oppimisvaihetta. Analysaattori saa syötteekseen ohjelman, jossa ei ole rooli-informaatiota. Sovellus kartoittaa syöteohjelman muuttujien tietovirtaominaisuudet samalla tavoin kuin oppimisvaiheessakin. Sen jälkeen analysaattori vertailee muuttujien tietovirtaominaisuuksia tietokantaan talletettuihin roolien tietovirtaominaisuuksiin ja etsii kullekin muuttujalle parhaiten sopivan roolin.

Oppimisvaiheessa käytetty aineisto vaikuttaa analysoijan tietokantaan talletettaviin roolimääritelmiin. Eri ohjelmoijaryhmien tuottamien ohjelmien käyttäminen johtaa erilaisiin tietokantoihin.

Toteutamme automaattista roolien tunnistusta osana PlanAnia. Toteutamme sovellusta Tcl/Tk:lla ja hyödynnämme Yeti-moduulia [12], joka on Yacc:in kaltainen jäsentäjägeneraattori.

7 Yhteenveto

Tässä tekstissä olemme esitelleet muuttujien roolit -käsitteen. Muuttujien roolit on uusi käsite, joka kuvaa muuttujien stereotyyppistä käyttöä ja edustaa asiantuntijoiden omaamaa ohjelmointiin liittyvää hiljaista tietoa. Muuttujien rooleja voidaan hyödyntää ohjelmoinnin alkeisopetuksessa tuomalla asiantuntijoiden hiljaista tietoa vasta-alkajien ulottuville. Vain kymmenen roolia riittää kattamaan 99 prosenttia kaikista aloittelijatason imperatiivisissa ohjelmissa esiintyvistä muuttujista.

Muuttujien rooleja visualisoidaan Joensuun yliopistossa kehitetyn PlanAni-animaattorin avulla. PlanAnissa jokaiselle

roolille on olemassa oma visualisointinsa — roolikuvansa — jota käytetään kaikille tietyn roolin omaaville muuttujille.

Olemme esitelleet kolme roolitutkimuksen osa-aluetta: roolit ja strateginen ohjelmointitietämys, attentio ja ohjelmatietämys, sekä automaattinen muuttujien roolien tunnistaminen. Rooleihin ja strategiseen ohjelmointitietämykseen liittyvä tutkimus tarkastelee muuttujien roolien vaikutusta strategisen ohjelmointitietämyksen muodostumiseen aloittelijaohjelmoijien kohdalla. Attention ja ohjelmatietämyksen tutkimuksessa selvitämme erilaisten roolikuvien ja niiden sijoittelun vaikutusta attention kohdistumiseen ja sitä kautta ohjelmatietämyksen muodostumiseen. Kolmas tutkimuksen osa-alue pyrkii kehittämään automaattisen roolien tunnistamisen menetelmiä.

Tätä työtä tukee Suomen Akatemia määrärahalla nro: 206574.

Viitteet

[1] Aho A. V., Sethi R., Ullman J. D. (1988) *Compilers. Principles, Techniques and Tools*. Addison-Wesley, Reading, Massachusetts.

[2] Byckling P., Kuittinen M., Nevalainen S., Sajaniemi J. (2004) An Inter-Rater Reliability Analysis of Good's Program Summary Analysis Scheme. *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004)*. Institute of Technology Carlow, Ireland, 170–184.

[3] Good J. (1999) *Programming Paradigms, Information Types and Graphical Representations: Empirical Investigations of Novice Program Comprehension*. Ph.D. thesis, University of Edinburgh.

[4] Hundhausen C. D., Douglas S. A., Stasko J. D. (2002) A Meta-study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing 13*, 259–290.

[5] Kuittinen M, Sajaniemi J. (2003) First Results of an Experiment on Using Roles of Variables in Teaching. *EASE & PPIG 2003, Papers from the Joint Conference at Keele University 8th–10th April 2003*. Keele, U.K., 347–357.

[6] Kuittinen M., Sajaniemi J. (2004) Teaching Roles of Variables in Elementary Programming Courses. ITiCSE 2004, *Proceedings of the 9th Annual Conference on Innovation and Technology in Computer Science Education*, Leeds, UK, June 2004. Association for Computing Machinery, 57–61.

[7] Mulholland P. (1998) A Principled Approach to the Evalution of SV: A Case Study in Prolog. *Software Visualization — Programming as a Multimedia Experience*. The MIT Press, J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, Eds., 439–451.

[8] Nevalainen S., Sajaniemi J. (2004) Comparison of Three Eye Tracking Devices in Psychology of Programming Research. *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004)*. Institute of Technology Carlow, Ireland, 151–158.

[9] Nielson F., Nielson H. R., Hankin C. (1988) *Principles of Program Analysis*. Springer-Verlag, Heidelberg.

[10] Pennington N. (1987) Comprehension Strategies in Programming. Olson G. M., Sheppard S., Soloway E. (toim.), *Empirical Studies of Programmers: Second Workshop*, Ablex Publ. Co., 100–113.

[11] Petre M., Blackwell A. F. (1999) Mental Imaginery in Program Design and Visual Programming. *International Journal of Human-Computer Studies 51*, 1, 7–30.

[12] Pillhofer F. (2002) YETI — Yet Another TCL Interpreter. Internet WWW-sivu, URL: http://www.fpx.de/fp/Software/Yeti/ (elokuu, 2004).

[13] Rist R. S. (1989) Schema Creation in Programming. *Cognitive Science 13*, 389–414.

[14] Sajaniemi J. (2002) An Empirical Analysis of Roles of Variables in Novice-Level Procedural Programs. *Proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02)*, Arlington, VA, September 2002. IEEE Computer Society, 37–39.

[15] Sajaniemi J., The Roles of Variables Home Page. Internet WWW-sivu, URL: http://www.cs.joensuu.fi/∼saja/var_roles/. (lokakuu, 2004).

[16] Sajaniemi J., Kuittinen M. (painossa) An Experiment on Using Roles of Variables in Teaching Introductory Programming. *Computer Science Education*.

[17] Sajaniemi J., Kuittinen M. (2003) Program Animation Based on the Roles of Variables. *Proceedings of ACM 2003 Symposium on Software Visualization (SoftVis 2003)*, San Diego, CA, June 2003. Association for Computing Machinery, 7–16.

[18] Sajaniemi J., Kuittinen M. (2004) Visualizing Roles of Variables in Program Animation. *Information Visualization* 3(3), 137–153.