Tietojenkäsittelytiede **31** Marraskuu 2010 sivut 43–56
Toimittaja: Jorma Tarhio

Jype — visualisointi ja automaattinen arviointi ohjelmoinnin opetuksessa

Juha Helminen
Aalto-yliopisto
Tietotekniikan laitos
juha.helminen@cs.hut.fi

Tiivistelmä

Ohjelmoinnin oppimisen tueksi on kehitetty lukuisia ohjelmistoja. Useimmat näistä havainnollistavat ohjelmointiin liittyviä käsitteitä visualisoinnin keinoin. Toisen pääjoukon muodostavat automaattiset arviointi- ja palautejärjestelmät. Tutkimusten mukaan visualisointityökalut näyttäisivät tukevan oppimista tehokkaammin, jos ne aktivoivat oppijan pelkän passiivisen katselun sijaan. Tämä on innostanut tutkimaan visualisoinnin ja automaattisen arvioinnin sekä palautteen tiivistä yhdistämistä. Artikkelissa annetaan yleiskatsaus em. työkaluista ja esitellään Jype, uusi web-pohjainen ohjelmointiympäristö Pythonille. Jypessä yhdistyvät suoraviivainen yksinkertaistettu koodausympäristö, joka myös mahdollistaa visuaalisen virheenjäljityksen ohjelman tilan visualisaatioiden tukemana, ja automaattisesti arvioidut ohjelmointitehtävät.

1 Johdanto

Ohjelmointitaidon opetus on kriittinen osa tietotekniikan koulutusta ja ohjelmoinnin alkeet on yleensä yksi ensimmäisistä opintojaksoista tietotekniikan opetusohjelmissa. Ohjelmoinnin oppiminen on kuitenkin hyvin vaikeaa. Ennen kaikkea ohjelmoinnin oppiminen vaatii, että oppija muodostaa mielessään käsitetason ymmärryksen ohjelmointikielen suoritusmallista [11] sekä siitä, miten ohjelmointirakenteet ja -käsitteet kytkeytyvät siihen ohjaus- ja tietovuossa. Lisäksi on opittava käyttämään ohjelmatyökaluja, joilla hallitaan ohjelmoinnin eri osatehtäviä, kuten kääntämistä, testausta ja virheenjäljitystä.

Oppimistutkimuksissa yliopisto-opettajat kautta maailman ovat toistuvasti havainneet, että opiskelijat eivät ole omaksuneet tietoja ja kehittäneet taitoja, joita heiltä odotetaan, kun he ovat suorittaneet alkeisohjelmoinnin opintojaksonsa [49, 51]. Siksi ohjelmoinnin opetusmenetelmiä on tutkittu paljon, ja erityisesti ohjelmoinnin opetuksen tueksi on kehitetty monenlaisia ohjelmistoja, joilla pyritään tukemaan ja edistämään ohjelmointiosaamisen kehittymistä. Tässä artikkelissa annetaan yleiskatsaus ohjelmoinnin opetuksen ohjelmatyökaluista keskittyen erityisesti ohjelmien visualisointiin sekä järjestelmiin, joissa sitä on yhdistetty automaattiseen arviointiin. Lopuksi esitellään eräs uudempi järjestelmä, Jype, joka on web-pohjainen ohjelmoinnin opetustyökalu Pythonille, missä ohjelmointiympäristö on tiiviisti yhdistetty automaatti-

seen arviointiin.

Ohjelmoinnin oppimista tukevat ohjelmatyökalut keskittyvät yleensä johonkin tiettyyn oppimisen osa-alueeseen. Monet näistä työkaluista perustuvat ohjelmien visualisointiin. Pyrkimyksenä on havainnollistaa ohjelmoinnissa usein esiintyviä abstrakteja käsitteitä ja rakenteita. Näin toivotaan parannettavan ja nopeutettavan tietorakenteiden, algoritmien ja yleensäkin ohjelmien suorituksen ymmärtämistä. Myös itse ohjelmointi voi tapahtua kokonaan graafisilla operaatioilla tekstin kirjoittamisen sijaan. Toinen pääryhmä on erilaiset automaattisen arvioinnin työkalut. Näissä oppija suorittaa ohjelmointiin liittyviä tehtäviä ja järjestelmä antaa välitöntä palautetta vastausten oikeellisuudesta. Kolmas pääryhmä tässä karkeassa jaottelussa on erilaiset erikoisympäristöt, joissa yleiskäyttöiseen ohjelmointikieleen tutustutaan askelissa, tasoittain, kasvattaen käytettävissä olevaa kielen ominaisuuksien joukkoa vähitellen tai se on korvattu kokonaan jollain peruskäsitteiden opetukseen paremmin soveltuvalla rajoitetummalla opetuskielellä. Artikkelit [38, 41] antavat kattavan katsauksen ohjelmoinnin opetuksen ja oppimisen tutkimuksesta.

2 Visualisointi ja automaattinen arviointi ohjelmoinnin opetuksessa

2.1 Visualisointi ohjelmoinnin opetuksessa

Keskeinen tavoite alkeisohjelmoinnin opintojaksoilla on oppia lukemaan ja kirjoittamaan ohjelmia. Paitsi riittävä harjoittelu, avainasemassa on tarkka ymmärrys siitä, mistä ohjelman tila muodostuu ja miten ohjelman suoritus vaikuttaa siihen. Kuten kaikkien abstraktien rakenteiden kohdalla, erilaisia havainnollistuksia voidaan käyttää tiedon välittämiseen ja ymmärtämisen edistämiseen. Täten yleisesti käytetty menetelmä oppimisprosessin tukemiseen ohjelmoinnin opetuksessa on ohjelman tilan visualisointi. On olemassa lukuisia opetustyökaluja sekä aloittelijoille suunnattuja kehitysympäristöjä, jotka hyödyntävät visualisointia ohjelman tilan havainnollistamisessa. Järjestelmät voidaan karkeasti ryhmitellä neljään pääluokkaan: visuaalisen ohjelmoinnin ympäristöt, mikromaailmat, algoritmien visualisointi ja ohjelmien visualisointi. Seuraavaksi tarkastellaan kutakin ryhmää yksitellen ja mainitaan esimerkkeinä tunnettuja sekä uudempia ja edistyneimpiä järjestelmiä. Kirja [50] sisältää historiallisen katsauksen visualisointijärjestelmistä.

Visuaalisen ohjelmoinnin ympäristöissä periaatteena on tarjota yksinkertaistettu graafinen käyttöliittymä ohjelmien rakentamiseen tekstipohjaisen ohjelmoinnin sijaan. Ajatuksena on, että tekstipohjaisessa ohjelmoinnissa huomattava osa ajasta kuluu kieliopin kanssa taistellessa ja että näin oppijat voivat päinvastoin aluksi keskittyä paremmin ohjelmoinnin olennaisiin käsitteisiin sen sijaan, että takertuisivat ongelmiin monimutkaisen formaalin kielen muodostavien ohjelmointikomentojen kanssa. Joitain esimerkkejä ovat JPie [13], Karel Universe [3], RAPTOR [6] ja Alice [31].

Mikromaailmat rakentavat saman periaatteen pohjalle, mutta lähestymistapa on tarjota opetukseen suunniteltu erikoisohjelmointikieli, jonka rakenteet ovat yksinkertaisempia kuin tavallisessa yleiskäyttöisessä ohjelmointikielessä. Lisäksi tämä kieli on tiiviisti yhdistetty graafiseen maailmaan, jonka puitteissa ohjelmointikielellä toimitaan. Maailma sisältää olentoja ja esineitä, joita voidaan ohjata ja käsitellä kielen sisältämillä yksinkertaisilla ja tarkoitukseen suunnitelluilla komennoilla. Maailmat perustuvat teemoihin, joilla on fyysinen vastine, kuten robotin oh-

jaamiseen. Visualisointi tarjoaa siten lähes käsinkosketeltavan havainnollistuksen ohjelman suorituksesta ja tilasta. Joitain esimerkkejä ovat Karel the robot [37], Jeroo [47], PigWorld [28] ja greenfoot [17].

Algoritmien visualisointijärjestelmillä tuotetaan animaatioita algoritmeista. Näiden järjestelmien tarkoitus on mahdollistaa algoritmien keskeisten periaatteiden havainnollistaminen ohjelman suoritusta ja tietokoneen muistia korkeammalla abstraktiotasolla. Tämä perustuu tyypillisesti sellaisten tietorakenteiden kuten taulukkojen, puiden ja verkkojen käsitetason visualisaatioihin, jotka korostavat tietoalkioiden käsitteellisiä suhteita sen sijaan, että esitettäisiin niiden varsinainen sijainti muistissa. Nämä järjestelmät on pääasiassa suunniteltu käytettäväksi luennoilla tai sähköisessä oppimateriaalissa, sillä ne vaativat yleensä algoritmin annotointia manuaalisesti visualisointikomennoilla, animoinnin kirjoittamista alusta alkaen erityisellä algoritmianimaatiokielellä tai -kirjastolla tai sen rakentamista graafisella käyttöliittymällä. Joitain esimerkkejä ovat Animal [42], JHAVÉ [34], Alvie [8] ja TRAKLA2 [29].

Ohjelmien visualisoinnissa keskitytään havainnollistamaan varsinaisten ajettavien tai ajossa olevien ohjelmien muistin ja tietorakenteiden toimintaa. Niitä käytetään virheenjäljityksessä sekä ylipäätään ohjelmien ymmärryksen helpottamisessa. Ne voivat tarjota visualisaatioita monista eri osista ohjelman suoritusta kuten lausekkeiden evaluoinnista, muuttujasidonnoista ja kutsupinosta. Tyypillisesti järjestelmä mahdollistaa ohjelmakoodin suorittamisen askelittain, missä ohjelman tilan muuttumista voi seurata visualisaatioiden tukemana visuaalisten virheenjäljittimien tapaan ja visualisaatiot rakentuu kielen käsitteiden ympärille muuttujien, funktioiden ja luokkien tasolle. Joitain esimerkkejä ovat Jeliot 3[1] [30], jGRASP[2] [9] (Graphical Representations of Algorithms, Structures, and Processes), VIP[3] [52] (Visual InterPreter), ViLLE[4] [25] (the visual learning tool) ja Teaching Machine[5] [36].

2.2 Ohjelmointitehtävien automaattinen arviointi

Yksinkertaisen ohjelman kirjoittaminen, laajentaminen tai muokkaaminen ovat tyypillisiä tehtäviä ohjelmoinnin alkeiden opintojaksoilla. Tällaisten palautusten läpikäynti ja arvostelu on aikaavievää ja suurelta osin hyvin suoraviivaista. Siksi ohjelmointitehtävien arvioinnin automatisointiin on kehitetty monia järjestelmiä. Automaattinen arviointi mahdollistaa työkuorman pitämisen hallittavissa, kun opiskelijoita on paljon tai tehtäviä halutaan tarjota paljon. Järjestelmien tulisi tukea oppimista tarjoamalla riittävän hyvää palautetta oppijoille, jotta he pystyvät kehittymään itsenäisesti käytännön ohjelmointitehtävien kautta. Työkaluja on niin yksinkertaisista yksikkötestien ajamiseen perustuvista järjestelmistä älykkäisiin tutorointijärjestelmiin, jotka seuraavat ja mallintavat oppijan edistymistä ja tarjoavat sen perusteella vinkkejä ja ohjausta oppijalle. Pääasiassa ohjelmointitehtävien arviointi perustuu kuitenkin lopputuotteeseen, eli erityisesti tuotettuun ohjelmakoodiin. Ohjelmalta vaaditaan oikeaa toiminnallisuutta sekä hyvää rakennetta ja ohjelmointityyliä. Tyypillisesti automaattisilla menetelmillä keskity-

[1] `http://cs.joensuu.fi/jeliot`
[2] `http://www.jgrasp.org`
[3] `http://www.cs.tut.fi/~vip/en/`
[4] `http://ville.cs.utu.fi`
[5] http://www.engr.mun.ca/ theo/TM/

tään ohjelman oikeellisuuden arviointiin tutkimalla sen ajonaikaista käyttäytymistä. Lisäksi saatetaan arvioida esimerkiksi ohjelman tehokkuutta tai ohjelmointityyliä. Analyyseillä voidaan pyrkiä havaitsemaan vaikka globaalien muuttujien tarpeetonta käyttöä, huonoa muuttujien nimeämistä tai sisentämistä. Artikkeli [2] antaa katsauksen ohjelmointitehtävien automaattisesta arvioinnista.

Nykyiset ohjelmointitehtävien automaattisen arvioinnin järjestelmät ovat poikkeuksetta web-pohjaisia. Tyypillisesti oppija lähettää työnsä web-lomakkeella palvelinkoneelle arvioitavaksi ja saa palautteen vastaavasti web-sivuna. Ohjelmien arviointiin on karkeasti kaksi päälähestymistapaa: yksikkötestit ja ohjelman tulosteen vertailu. Tulostelähestymisessä annettu ohjelmakoodi ajetaan jollakin testisyötteellä ja tulostetta verrataan merkkipohjaisesti mallivastauksen tuottamaan tulosteeseen (esimerkiksi TRY [40], ASSYST [20], Goblin[6], RoboProf [10], BOSS [21], CourseMarker [18], Online Judge [7]). Ohjelman tulee siis tulostaa tiettyjä asioita tietyssä muodossa tietyssä järjestyksessä. Yksikkötestilähestymisessä annettua koodia tutkitaan kutsumalla osia siitä yksikkötestityyliin tarkastaen funktioiden/metodien oikea toiminta (esimerkiksi Scheme-robo [46], BOSS [21], Ludwig [48], Javala [27], WebTasks [43]). Web-CAT [12] sisältää hieman erilaisen lähestymistavan, jossa arviointiin sisällytetään oppijan itse kirjoittamat testit. Tässä asetelmassa oppija demonstroi itse, että ohjelma toimii sen sijaan, että järjestelmä testaa toimivuuden. Arviointi perustuu kolmeen tekijään: testien oikeellisuus, testien kattavuus ja testitulokset. Oppijan testit ajetaan mallitoteutukselle testien oikeellisuuden ja kattavuuden arvioimiseksi. Lisäksi testit ajetaan oppijan omalle toteutukselle. Perustelu menettelylle on, että jos testit ovat oikeat ja kattavat, niin myös ohjelman täytyy olla oikein, jos se läpäisee nämä testit.

Automaattisen arvioinnin hyötyjä ovat välitön palaute, palvelun hyvä saatavuus riippumatta ajasta tai paikasta sekä arvostelun puolueettomuus ja yhtenäisyys. Lähes poikkeuksetta nykyaikaiset arviointijärjestelmät mahdollistavat myös oppijoiden tehtäväpalautusten seuraamisen ja raporttien tuottamisen niistä. Automaattisen arvioinnin heikkouksia ovat yleensäottaen ihmistä heikompi palautteen laatu ja plagiarismi. On esimerkiksi hyvin vaikea arvioida ja antaa automaattisesti palautetta siitä, mistä mahdollisesta virhekäsityksestä jokin ohjelmaan jäänyt virhe johtuu. Plagiarismia on käytännössä mahdotonta estää, jos tehtävien tekemistä ei valvota, mutta sen havaitsemiseen on kehitetty monia järjestelmiä, jotka vertailevat palautuksia arvioiden, kuinka suurelta osin ne muistuttavat toisiaan. Tämän menetelmän luotettavuus toki heikkenee sen mukaan, mitä pienempiä tai muuten luonnostaan samankaltaisia palautukset ovat. Esimerkkejä tehokkaista nykyaikaisista järjestelmistä, jotka perustuvat ohjelmakoodin muuntamiseen merkkijonoalkioiksi [26], ovat YAP3 [53], JPlag [39] ja Plaggie [1]. Tällaiset järjestelmät eivät kompastu tavallisiin yksinkertaisiin leksikaalisiin tai rakenteellisiin muunnoksiin, joilla tyypillisesti yritetään peitellä plagiointia.

2.3 Ohjelmien visualisointi ja oppijan aktivointi

Kun oppijoille tarjotaan puitteet ohjelmien ja algoritmien tutkimiseen visuaalisesti, odotetaan heidän pystyvän hahmottamaan ohjelmien suoritukseen liittyvät käsitteet nopeammin ja syvällisemmin.

[6] http://goblin.tkk.fi/goblin/

On kuitenkin olemassa ristiriitaisia tutkimustuloksia sen suhteen, että edistävätkö visualisaatiot itsessään, kuten algoritmianimaatiot, todella oppimista. Tutkimukset antavat ymmärtää, että vuorovaikutuksen määrä visualisointia käytettäessä on merkittävä tekijä [14, 19, 35]. Passiivisten animaatioiden sijaan aktivoivat visualisointityökalut vaikuttavat tehokkaammilta. Erityisesti tämän tuloksen innoittamana on kehitetty monia järjestelmiä, jotka yhdistävät visualisoinnin ja automaattisen arvioinnin: oppijoille annetaan visualisaatioon liittyviä tehtäviä ja heidän vastauksistaan annetaan välitöntä palautetta automaattisen arvioinnin keinoin.

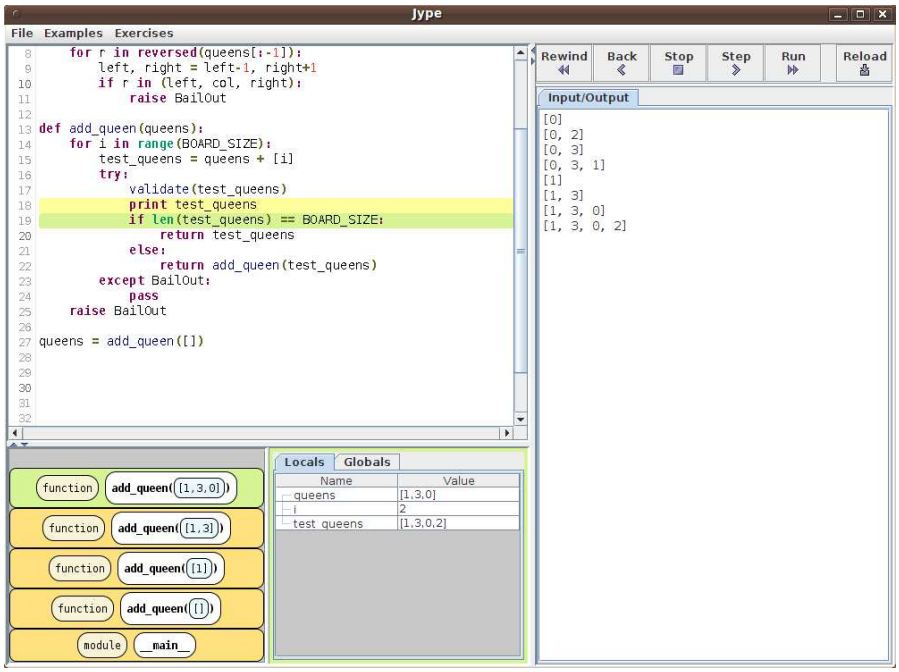
Tehtävät voivat esimerkiksi olla algoritmianimaatioon tai koodinanimaatioon pohjautuvia monivalintakysymyksiä [22, 25, 32, 33, 44, 45]. Tyypillisessä kysymyksessä oppijan tulisi ennustaa jonkin muuttujan arvo senhetkisen koodilauseen suorituksen jälkeen. Toinen esimerkki ovat automaattisesti arvioidut visuaaliset algoritmisimulaatiotehtävät, joissa oppijat rakentavat algoritmianimaatioita simuloimalla algoritmin suoritusaskelia. TRAKLA2:ssa [29] oppijat käsittelevät tietoalkioita tietorakenteiden visualisaatioissa raahaamalla ja valitsemalla niitä hiirellä ja lopputuloksen oikeellisuus arvioidaan vertaamalla luotuja tilasiirtymiä vastaaviin algoritmitoteutuksen tuottamiin siirtymiin. MA&DA:ssa [24] oppijat käsittelevät tietorakenteita valikoiden kautta. PILOT:ssa [4] verkkoalgoritmeja jäljitetään valitsemalla verkkojen kaaria.

Vain harvoissa jäjestelmissä, jotka näyttävät ohjelmavisualisaatioita, oppija aktivoidaan tehtävillä. Jeliot 3:lle on olemassa alustava toteutus ohjelman suoritukseen liittyville ennustustehtäville [32] eli tehtäville, joissa oppijan tulee vastata kysymyksiin ohjelman myöhemmistä tiloista nykyisen tilan (visualisaatioiden) perusteella: ”Mikä on muuttujan tulos arvo tämänhetkisen ohjelmarivin suorittamisen jälkeen?”. ViLLE-järjestelmässä voidaan luoda ohjelma-animaatioita, joihin voidaan liittää suoritukseen liittyviä monivalintakysymyksiä. Teaching Machine voi myös sisältää samantyylisiä kysymyksiä [5]. VIP on ainoa järjestelmä, joka yhdistää ohjelmointitehtävät ja ohjelmien visualisoinnin: integroitu ympäristö tarjoaa koodieditoriin ja visuaalisen virheenjäljittimen sekä antaa palautetta ohjelmien oikeellisuudesta.

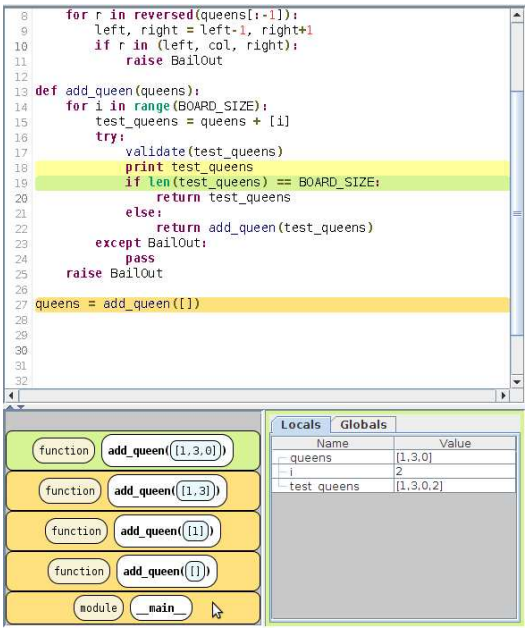
3 Jype

Useimmat nykyiset ohjelmien visualisaatioita näyttävät järjestelmät on rakennettu Java-kielelle, joka on ollut yleisin opetuskieli viime vuosina. Pythonin opetuksen tueksi ei ole vielä montaa ohjelmatyökalua. Joitain mikromaailmaan perustuvia ohjelmointiympäristöjä on kehitetty: rur-ple[7], Guido van Robot[8] ja Turtlet[9]. ViLLE [25] tarjoaa myös jotain tukea Pythonin suorituksen visualisointiin, sillä siinä Java-ohjelmakoodin suorituksen animaatio voidaan jossain määrin muuntaa vastaavaksi animaatioksi Pythonilla. *Jype* [15, 16] on uusi web-pohjainen ohjelmointiympäristö Pythonille, jossa yhdistyvät suoraviivainen yksinkertaistettu koodausympäristö, joka myös mahdollistaa visuaalisen virheenjäljityksen ohjelman tilan visualisaatioiden tukemana ja automaattisesti arvioidut ohjelmointitehtävät. Työkalu on suunniteltu ensimmäisen alkeisohjelmointikurssin tarpeisiin ja painopisteenä on siten lause- ja funktiotason ohjelmointi (ei oliolähtöinen kurs-

[7] `http://sourceforge.net/projects/rur-ple/`
[8] `http://gvr.sourceforge.net/`
[9] `http://www2.lut.fi/~jukasuri/Kilppari/`



Kuva 1: Jypen yleisnäkymä.



Kuva 2: Funktiokutsuja voi tarkastella nopeasti liikuttamalla hiiren kursoria pinokehysten päällä.

si). Sen tulisi tukea ohjelmien alkeiden oppimista tarjoamalla harjoitteluympäristö ohjelmakoodin kirjoittamiseen ja ohjelmien suorituksen jäljittämiseen. Teknillisessä korkeakoulussa tehty diplomityö [15] kuvaa tarkasti työkalun ensimmäisen version suunnitteluperiaatteet ja niiden perustelut sekä teknisen toteutuksen. Työkalun pohjalta on myös julkaistu konferenssiartikkeli [16], jossa tarkastellaan ohjelmatyökalun piirteitä opetuskontekstissa.

Kuvassa 3 on Jypen yleisnäkymä. Käyttöliittymä on jaettu neljään pääosioon. Vasemmassa ylänurkassa on komponentti koodin kirjoittamiseen ja muokkaamiseen. Sen alapuolella on visualisaatioalue, jossa havainnollistetaan suorituspinon ja muuttujien tiloja. Oikealla on tila tekstikomponenteille, jotka ovat välilehdissä. Tulostevirran sisältö näytetään yhdessä välilehdessä ja lisävälilehdet voivat näyttää tehtäväohjeita tai automaattisen arvioinnin palautteita.

3.1 Visualisointi

Ohjelman suoritusta voidaan tarkastella kahdesta päänäkökulmasta: ohjausvuo ja tietovuo. Jypessä ohjausvuon muutokset näytetään koodissa värittämällä parhaillaan suoritettava rivi vihreällä ja viimeksi suoritettu rivi keltaisella. Lisäksi suorituksen keskeyttävät poikkeukset korostetaan koodinäkymässä punaisella. Jype näyttää myös suorituspinon tilan yksinkertaisella visualisaatiolla. Laatikot näyttävät pinokehysten tyypit ja funktioille näytetään myös argumentit ja paluuarvot. Funktiokutsujen jonoa voi tarkastella myös liikuttamalla hiiren kursorin funktiokehysten päälle, jolloin koodinäkymästä korostetaan tämän funktiokutsun koodirivi kuten kuvassa 3.

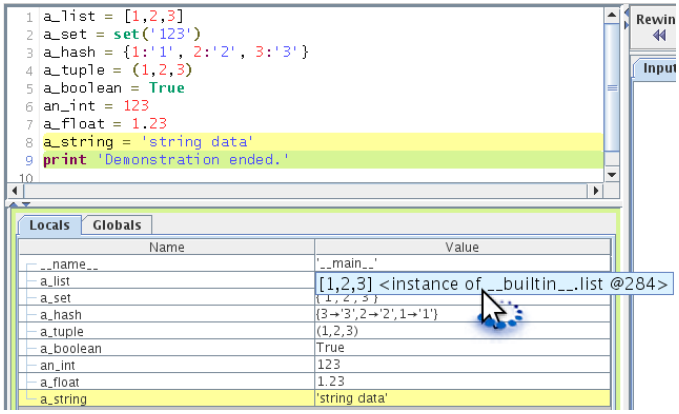
Kuvat 3.1 ja 3.1 näyttävät datavuon perusvisualisaation. Muuttujien nimet ja arvot esitetään yksinkertaisessa taulukkonäkymässä merkkijonoina. Kun uusi muuttuja esitellään kyseinen rivi korostetaan keltaisella värillä vastaten koodinäkymän väritystä. Kun olemassaolevaan muuttujaan sidotaan uusi arvo, vain arvon sarake korostetaan (kuva 3.1).

Kuvattujen perusvisualisaatioiden lisäksi Jypeä voidaan laajentaa vaihtoehtoisilla näkymillä. Tällä hetkellä Jype käyttää sisällytettyä tietorakenteiden visualisointikirjastoa näyttääkseen dynaamisen taulukon visualisaation Pythonin sisäänrakennetulle listatietotyypille (kuva 3.2). Koska Jypessä käytetty Python-toteutus voi suoraan kutsua ja käyttää Java-luokkia, voidaan myös käyttää visualisointikirjaston monia Java-toteutuksia tavallisista tietorakenteista visualisaatioineen. Kuvassa 3.2 on esimerkkinä binäärihakupuu. Muita kirjaston tietorakenteita ovat b-puu, punamusta puu, verkot, listat ja jonot. Jypeä voidaan siis käyttää myös algoritmianimaatioiden luomiseen Pythonilla käyttäen näitä rakenteita.

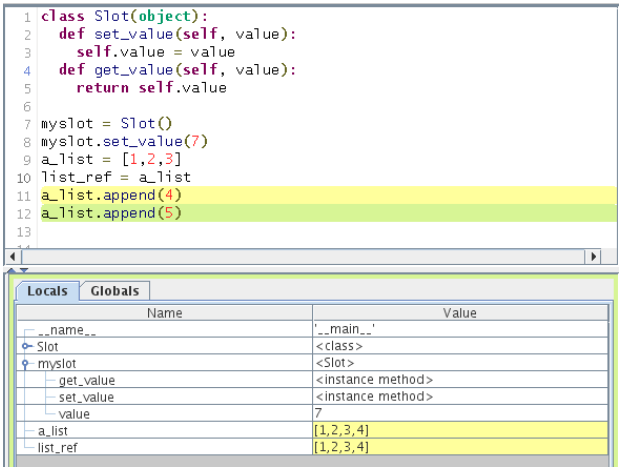
3.2 Toiminnallisuus

Ohjelmien suoritusta hallitaan Jypessä yksinkertaisilla painikkeilla, jotka näkyvät kuvassa 3: mene suorituksen alkuun, askella taaksepäin, pysäytä suoritus, askella eteenpäin ja aja ohjelma. Käyttäjä voi sujuvasti vaihdella koodin muokkauksen ja ohjelman ajamisen välillä ilman kääntämisaskelta – kun koodia muokataan kesken ohjelman ajon eli animaation, se keskeytetään automaattisesti. Suoritusta voi myös navigoida suorituspinon laatikoiden kautta. Valitsemalla laatikon hiirellä suoritus/animaatio siirtyy tähän kohtaan suoritushistoriaa, josta askeltamista voi taas jatkaa eteen- tai taaksepäin aivan normaalisti. Tätä toimintoa voitaisiin kutsua ’astumiseksi taaksepäin kutsujaan’.

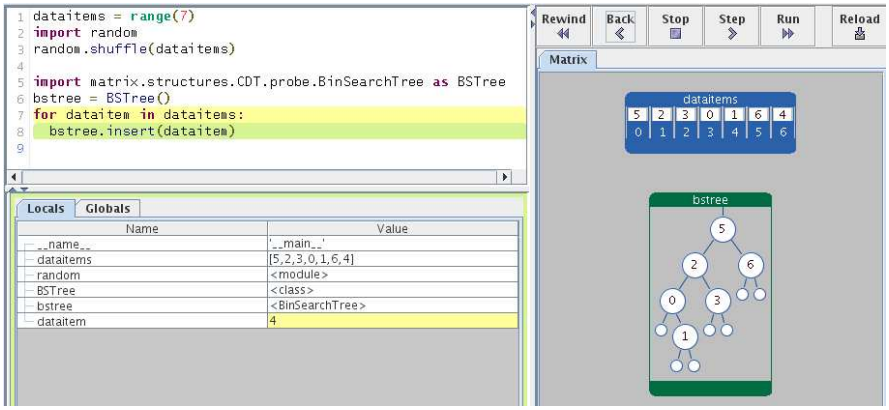
Tulostevirta on myös sidottu suorituk-



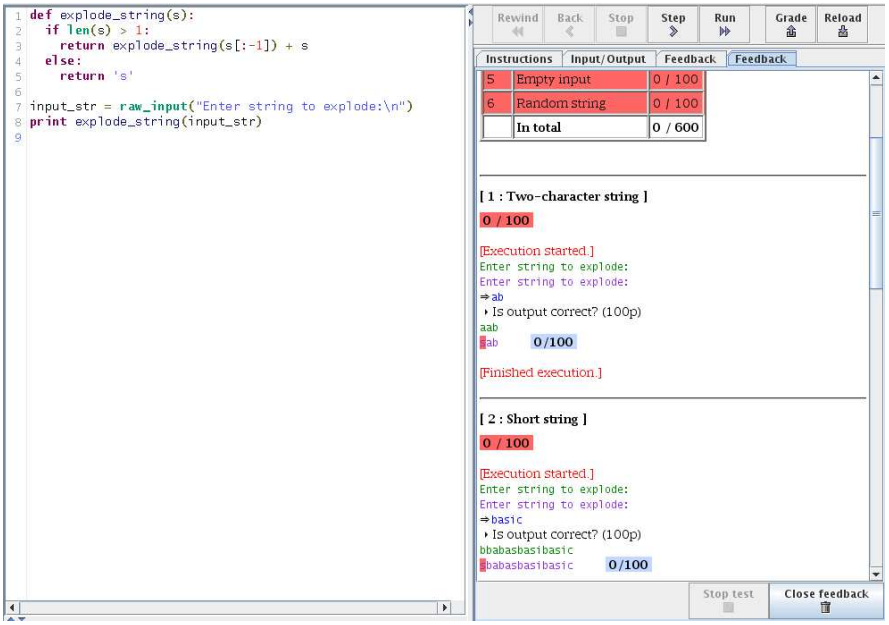
Kuva 3: Uusi muuttuja esiteltiin viimeksi suoritetulla rivillä kuten keltaiset väritykset ilmaisevat.



Kuva 4: Kahden muuttujan arvot muuttuivat (käsitteellisesti) viimeksi suoritetulla rivillä kuten keltaiset värityksen ilmaisevat. `list-ref` viittaa samaa listaolioon kuin `a-list`, johon lisättiin, joten olioviittausten tasolla muuttujasidonnat eivät muuttuneet.



Kuva 5: Pythonin listatietotyypistä, joka on visualisoitu dynaamisena taulukkona, lisätään alkioita binäärihakupuuhun.



Kuva 6: Tämän tehtävän palautteessa näytetään odotettu tuloste ja oppijan palautuksen tuottama tuloste allekkain eri väreillä. Palaute paljastaa ohjelmassa olevan järjestelmällisen virheen, missä tulosteen alkuun tulee aina virheellinen merkki.

sen animaatioon virheenjäljittämisen selkeyttämiseksi. Joten askellettaessa suoritusta taaksepäin myös tulosteet 'peruutetaan' takaisin aikaisempaan tilaan. Tämän on tarkoitus helpottaa erityisesti sellaisten ohjelmien virheiden jäljittämistä, joissa tuloste on keskeisessä osassa eli esimerkiksi syöte/tuloste-tyyppiseen vuorovaikutukseen perustuvissa yksinkertaisissa komentorivipohjaisissa ohjelmissa. Kun syötevirtaa luetaan, eli ohjelma odottaa syötettä esimerkiksi Pythonin sisäänrakennetulla `raw_input()`-funktiolla, tulostevirran tekstinäkymä muuttuu rivieditoriksi, jonka kautta syöte annetaan.

Jypessä voi olla kahdenlaista sisältöä – esimerkkiohjelmia ja harjoituksia. Esimerkkiohjelmien tarkoitus on mahdollistaa tiettyjen rakenteiden ja käsitteiden opiskelu pedagogisesti perusteltujen esimerkkien kautta suorituksen askeltamisen tukemana. Nämä voivat toimia valmisteluna tehtävälle, jossa tietoja harjoitellaan ja testataan. Tehtävien tarkoitus on mahdollistaa oppijoiden päästä nopeasti ja helposti kokeilemaan pienten ohjelmien kirjoittamista välittömän automaattisen palautteen tukemana (kuva 3.2). Palaute koostuu vähintään hyväksytyistä ja epäonnistuneista testeistä, joissa on kuvattu odotettu lopputulos ja opiskelijan ohjelman tuottama tulos. Testien kirjoittaja voi lisäksi liittää jokaiseen testiin kuvauksen, mitä testissä tehdään, mikä sitten antaa vihjeen mahdollisesta ohjelman ongelmasta. Tulosteita verrattaessa opiskelijan ohjelman virheellinen tuloste korostetaan.

Kiteytettynä oppija käyttäisi järjestelmää seuraavalla tavalla: (1) oppija lukee tehtävänannon sekä mahdollisen muun tehtävään liittyvän oppimateriaalin ja alkaa tehdä kokeiluja tehtävään liittyvillä ohjelman käsitteillä ja rakenteilla käyttäen hyväksi mahdollisesti tehtävänannossa annettua valmiskoodia – tätä tutkiskelua tukee mahdollisuus suorittaa ohjelmaa askel askeleelta tutkien tilan muuttumista, (2) seuraavaksi oppija yrittää ratkaista tehtävän ja tekee omat testinsä ajamalla sitä ja vertaamalla tulosta odotettuun, (3) kun oppija on vakuuttunut, että hänen ratkaisunsa toteuttaa tehtävänannon vaatimukset, hän lähettää sen arvosteltavaksi saadakseen palautetta esityksestään, (4) jos ohjelmassa on virheitä, oppija jatkaa jäljittämällä virheitä käyttäen hyödykseen saatua palautetta sekä visualisaatioita ja mahdollisuutta askeltaa koodia joustavasti eteen- ja taaksepäin, kunnes on jälleen vakuuttunut ohjelman oikeellisuudesta ja jatkaa askeleesta 3.

3.3 Tekninen toteutus

Jype on toteutettu Javalla käyttäen Swing-käyttöliittymäkirjastoa ja on siten suhteellisen järjestelmäriippumaton. Sitä voidaan ajaa Java Applettina tai Java Web Start -sovelluksena webin ylitse, sekä myös paikallisesti asennettuna sovelluksena. Jype tarjoaa ainoastaan ympäristön ohjelmasuoritusten tarkasteluun ja ohjelmointitehtävien ratkontaan, ja se onkin tarkoitettu käytettäväksi osana jotain kurssinhallintajärjestelmää, jota käytetään palautusten tallentamiseen ja hallinnointiin. Tällä hetkellä Jype sisältää integraation TRAKLA2-järjestelmään [29]. Se mahdollistaa kurssin luomisen Jype-tehtävistä ja henkilökohtaiset käyttäjätilit pisteiden tallentamiseen ja seuraamiseen. Jype on rakennettu seuraavia avoimen lähdekoodin kirjastoja käyttäen: Jython[10], Matrix [23] ja jEdit[11].

[10] `http://www.jython.org/`
[11] `http://www.jedit.org/`

4 Yhteenveto

Tutkimusten mukaan visualisointityökalut näyttäisivät tukevan oppimista tehokkaammin, jos ne aktivoivat oppijan pelkän passiivisen katselun sijaan. Tässä artikkelissa annettiin yleiskatsaus ohjelmoinnin opetuksen ohjelmatyökaluista keskittyen erityisesti ohjelmien visualisointiin sekä järjestelmiin, joissa sitä on yhdistetty automaattiseen arviointiin. Jype on uusi web-pohjainen ohjelmoinnin opetustyökalu Pythonille, missä ohjelmointiympäristö on tiiviisti yhdistetty ohjelmavisualisaatioihin ja automaattisesti arvioituihin ohjelmointitehtäviin.

Viitteet

1. A. Ahtiainen, S. Surakka, and M. Rahikainen. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, pages 141–142. ACM Press, New York, NY, USA, 2006.
2. K.M. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005.
3. J. Bergin. Karel Universe drag & drop editor. In *ITICSE '06: Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 307–307, 2006.
4. S. Bridgeman, M.T. Goodrich, S.G. Kobourov, and R. Tamassia. PILOT: An interactive tool for learning and grading. *SIGCSE Bulletin*, 32(1):139–143, 2000.
5. M. Bruce-Lockhart, P. Crescenzi, and T. Norvell. Integrating test generation functionality into the Teaching Machine environment. *Electronic Notes in Theoretical Computer Science*, 224:115–124, 2009.
6. M.C. Carlisle. Raptor: a visual programming environment for teaching object-oriented programming. *Journal of Computing in Small Colleges*, 24(4):275–281, 2009.
7. B. Cheang, A. Kurnia, A. Lim, and W.C. Oon. On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121–131, 2003.
8. P. Crescenzi and C. Nocentini. Fully integrating algorithm visualization into a CS2 course.: a two-year experience. In *ITiCSE '07: Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 296–300, 2007.
9. J.H. Cross, T.D. Hendrix, and L.A. Barowski. Integrating multiple approaches for interacting with dynamic data structure visualizations. In *Proceedings of the 5th Program Visualization Workshop*, pages 3–10, 2008.
10. C. Daly and J.M. Horgan. An automated learning system for Java programming. *IEEE Transactions on Education*, 47(1):10–17, 2004.
11. B. Du Boulay, T. O'Shea, and J. Monk. The black box inside the glass box: Presenting computing concepts to novices". *International Journal of Human–Computer Studies*, 51(2):265–277, 1999.
12. S. Edwards. Using test-driven development in the classroom: providing students with automatic, concrete feedback on performance. In *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications EISTA*, volume 3, 2003.
13. Kenneth J. Goldman. An interactive environment for beginning Java programmers. *Science of Computer Programming*, 53(1):3–24, 2004.
14. S. Grissom, M.F. McNally, and T. Naps. Algorithm visualization in CS education: Comparing levels of student engagement. In *Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 87–94, 2003.
15. J. Helminen. Jype – an education-oriented integrated program visualization, visual debugging, and programming exercise

tool for Python. Master's thesis, Department of Computer Science and Engineering, Helsinki University of Technology, March 2009.

16. J. Helminen, L. Malmi, and A. Korhonen. Quick introduction to programming with an integrated code editor, automatic assessment and visual debugging tool – work in progress. In *Proceedings of the 9th Koli Calling International Conference on Computing Education Research*, Joensuu, Finland, October 2009.
17. P. Henriksen and M. Kölling. greenfoot: Combining object visualisation with interaction. In *OOPSLA '04: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 73–82, 2004.
18. C.A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas. Automated assessment and experiences of teaching programming. *Journal on Educational Resources in Computing (JERIC)*, 5(3), 2005.
19. C.D. Hundhausen, S.A. Douglas, and J.T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, 2002.
20. D. Jackson and M. Usher. Grading student programs using ASSYST. *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, pages 335–339, 1997.
21. M. Joy, N. Griffiths, and R. Boyatt. The boss online submission and assessment system. *Journal on Educational Resources in Computing (JERIC)*, 5(3), 2005.
22. V. Karavirta and A. Korhonen. Automatic tutoring question generation during algorithm simulation. *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, pages 95–100, 2006.
23. A. Korhonen, L. Malmi, P. Silvasti, V. Karavirta, J. Lönnberg, J. Nikander, K. Stålnacke, and P. Ihantola. Matrix – a framework for interactive software visualization. Research Report TKO-B 154/04, Laboratory of Information Processing Science, Department of Computer Science and Engineering, Helsinki University of Technology, Finland, 2004.
24. M. Krebs, T. Lauer, T. Ottmann, and S. Trahasch. Student-built algorithm visualizations for assessment: flexible generation, feedback and grading. *SIGCSE Bulletin*, 37(3):281–285, 2005.
25. M.J. Laakso, E. Kaila, T. Rajala, and T. Salakoski. Define and visualize your first programming language. In *8th IEEE International Conference on Advanced Learning Technologies*, pages 324–326, 2008.
26. T. Lancaster and F. Culwin. A comparison of source code plagiarism detection engines. *Computer Science Education*, 14(2):101–112, 2004.
27. T. Lehtonen. Javala – addictive e-learning of the Java programming language. In *Proceedings of the 5th Annual Finnish / Baltic Sea Conference on Computer Science Education*, pages 41–48. University of Joensuu, November 2005.
28. R. Lister. Teaching java first: Experiments with a pigs-early pedagogy. In *ACE '04: Proceedings of the 6th Conference on Australasian Computing Education*, pages 177–183, 2004.
29. L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, 2004.
30. A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing programs with Jeliot 3. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 373–376, 2004.
31. B. Moskal, D. Lurie, and S. Cooper. Evaluating the effectiveness of a new instructional approach. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, pages 75–79, 2004.

32. N. Myller. Automatic generation of prediction questions during program visualization. *Electronic Notes in Theoretical Computer Science*, 178:43–49, 2007.
33. T.L. Naps. JHAVÉ – addressing the need to support algorithm visualization with tools for active engagement. *IEEE Computer Graphics and Applications*, 25(6):49–55, 2005.
34. T.L. Naps, J.R. Eagan, and L.L. Norton. JHAVÉ—an environment to actively engage students in web-based algorithm visualizations. In *SIGCSE '00: Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, pages 109–113, 2000.
35. T.L. Naps, G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, et al. Exploring the role ofvisualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, 2003.
36. T.S. Norvell and M.P. Bruce-Lockhart. Teaching computer programming with program animation. In *Proceedings of the 2004 Canadian Conference on Computer and Software Engineering Education*, 2004.
37. R.E. Pattis. *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons, 1994.
38. A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. *SIGCSE Bulletin*, 39(4):204–223, 2007.
39. L. Prechelt, G. Malpohl, and M. Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11):1016–1038, 2002.
40. K.A. Reek. The TRY system - or - how to avoid testing student programs. *SIGCSE Bulletin*, 21(1):112–116, 1989.
41. A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.
42. G. Rössling and B. Freisleben. ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages & Computing*, 13(3):341–354, 2002.
43. G. Rössling and S. Hartte. WebTasks: online programming exercises made easy. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 363–363. ACM, New York, NY, USA, 2008.
44. G. Rössling and G. Haussge. Towards tool-independent interaction support. *Proceedings of the Third Program Visualization Workshop*, pages 110–117, 2004.
45. G. Rössling and S. Schneider. An integrated and "engaging"package for tree animations. *Electronic Notes in Theoretical Computer Science*, 178:69–78, 2007.
46. R. Saikkonen, L. Malmi, and A. Korhonen. Fully automatic assessment of programming exercises. *SIGCSE Bulletin*, 33(3):133–136, 2001.
47. D. Sanders and B. Dorn. Jeroo: a tool for introducing object-oriented programming. In *SIGCSE '03: Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 201–204, 2003.
48. S.C. Shaffer. Ludwig: an online programming tutoring and assessment system. *SIGCSE Bulletin*, 37(2):56–60, 2005.
49. B. Simon, R. Lister, and S. Fincher. Multi-institutional computer science education research: A review of recent studies of novice understanding. In *36th Annual Frontiers in Education Conference*, pages 12–17, 2006.
50. J. Stasko. *Software visualization: Programming as a multimedia experience*. MIT Press, 1998.
51. J. Tenenberg, S. Fincher, K. Blaha, D. Bouvier, D. Chinn, S. Cooper, A. Eckerdal, H. Johnson, R. McCartney, et al. Students designing software: a multi-

national, multi-institutional study. *Informatics in Education*, 4(1):143–162, 2005.

52. A.T. Virtanen, E. Lahtinen, and H.M. Järvinen. VIP, a visual interpreter for learning introductory programming with C++. In *Proceedings of The 5th Koli Calling Conference on Computer Science Education*, pages 125–130, 2005.
53. M.J. Wise. YAP3: improved detection of similarities in computer program and other texts. *SIGCSE Bulletin*, 28(1):130–134, 1996.