Inteligentni tutorski sustavi za poučavanje programiranja

Tonći Dadić
Fakultet prirodoslovno-matematičkih znanosti Sveučilišta u Splitu
Teslina 12, 21000 Split
E-mail: tonci.dadic@pmfst.hr

Sažetak - *Inteligentni tutorski sustav za poučavanje programiranja može značajno olakšati i unaprijediti tradicionalni nastavni proces. Učenik treba postići znanje svojstava jezika, vještinu oblikovanja programa i opću sposobnost rješavanja problema. Tutorsko poučavanje se provodi preliminarnim, interaktivnim ili refleksivnim pritupom. Specifičnost sustava za poučavanje programiranja je identifikacija i objašnjenje učenikovih semantičkih pogrešaka zavisnih zadatku. Kronološkim opisom karakterističnih sustava, s naglaskom na njihov doprinos, daje se pregled razvoja područja tijekom tri desetljeća. Odrednice oblikovanja planiranog sustava su cjelovitost poučavanja, jezična nezavisnost, te različiti pristupi naprednim i učenicima s poteškoćama. Nalazi se potrebnim istražiti mogućnost razumijevanja programa primjenom tehnike problema zadovoljenja ograničenja u uvjetima najverojatnijeg preslikavanja postignutog u namjeravani program.*

Ključne riječi - poučavanje programiranja, inteligentni tutorski sustav, razumijevanje programa.

1. Uvod

Inteligentni tutorski sustavi za poučavanje programiranja su interesantno područje primjene umjetne inteligencije. Od početka sedamdesetih do danas oblikovano je nekoliko desetina sustava i objavljeno stotine znanstvenih članaka, koji se odnose na proceduralne [6][15][23], funkcijske [2], logičke i objektne programske jezike [7]. Sustavi poučavaju učenike analizu problema, specificiranje programa, planiranje [16] i kodiranje. Pored toga, istraživači su posvetili pažnju vizualizaciji izvođenja programa i inteligentnom debugiranju.

Usprkos značajnom istraživačkom naporu investiranom u područje, relativno mali broj sustava se redovito koristi u nastavnom procesu, a većina se ne primjenjuje izvan institucije u kojoj su razvijeni [4]. Oblikovanje okruženja za učinkovito učenje programiranja je, još uvijek, izazovno istraživačko područje s mnogim otvorenim pitanjima [13].

Ovaj rad je organiziran u šest poglavlja. U 2. poglavlju se navode cilj i poteškoće učenja programiranja. Prikazuju se karakteristični sustavi, s osvrtom na inovacije kojima su doprinijeli području.

U trećem poglavlju se daju odrednice oblikovanja, a u četvrtom vrednovanje predloženog sustava. Zaključuje se rekapitulacijom svojstava predloženog sustava i prijedlogom smjera istraživanja.

2. Učenje programiranja i tutorski sustavi

Istraživači i stručnjaci, uključeni u proces poučavanja, programiranje nalaze kompleksnom misaonom aktivnošću kojom se definira apstraktan proces. Razumijevanje i vizualizacija apstraktnih procesa učenicima predstavlja problem kako pri učenju programiranja, tako i drugih područja sa sličnim značajkama.

2.1. Što učenik treba naučiti?

Linn i Dalbey [19] definiraju idealan lanac spoznajnog postignuća učenjem programiranja i predlažu ga standardom za usporedbu metoda nastave programiranja. Tri glavne karike lanca su:

- svojstva programskog jezika
- vještina oblikovanja programa
- opća vještina rješavanja problema

Svojstva jezika – u cilju zapisivanja programskog rješenja problema predmetnim jezikom, učenik treba razumjeti sintaksu, semantiku, te izražajne mogućnosti jezika.

Vještina oblikovanja programa - je znanje uporabe grupe tehnika čijom se primjenom

svojstva jezika kombiniraju u cjelinu kojom se rješava postavljeni problem. Vještina se temelji na znanju predložaka stereotipskih uzoraka koda koji komponiraju više svojstava jezika. Predlošci provode kompleksne funkcije, kao što su sortiranje, pronalaženje najmanjeg višekratnika dva cijela broja, brojanje riječi u zadanom tekstu itd. Programeri planiraju kombiniranje svojstava jezika i predložaka, dekomponiraju problem u dijelove, neovisno rješavaju svaki dio, te parcijalna rješenja povezuju u jedinstvenu cjelinu – program. Po oživotvorenju, testiranjem se utvrđuje korektnost programa.

Opća sposobnost rješavanja problema - dolazi do izražaja pri učenju novih formalnih sustava i postavlja se ciljnim postignućem nastave programiranja.

Isti predlošci i proceduralne vještine su zajedničke mnogim ili čak svim formalnim sustavima. Dakle, pristupom u kojem će učenik naučiti predloške zapisane jednim formalnim sustavom i pravila transfera u novi, predmet tekućeg učenja, do izražaja dolazi smisleno učenje i aktivacija postojećeg znanja.

2.2. Što početnicima predstavlja problem?

Lemut i dr. [18] poteškoću učenja programiranja objašnjavaju potrebom primjene niza kompleksnih aktivnosti, koje početnik mora savladati istovremeno. Na primjer, program se testira izvođenjem, uz pažljivo odabrane rubne vrijednosti programskih ulaza, koji će rezultirati izvodenjem svih programskih puteva. Izbor rubnih vrijednosti ulaza zahtjeva poznavanje semantke programskih instrukcija. Nasuprot tome, programer početnik uči instrukcije, pa vrlo teško može samostalno odabrati takve ulaze.

DuBoulay [11] moguće izvore poteškoća nalazi u:

(1) *Orjentaciji* - općoj predodžbi učenika o programiranju i programu.

(2) *Apstraktnom stroju* - razumijevanju računalnog modela kojeg definira programski jezik.

(3) *Notaciji* - sintaksi i semantici jezika.

(4) *Strukturama* - znanju programskih konstrukcija kao kompoziciji instrukcija kojima se rješavaju određeni programski zahtjevi.

(5) *Pragmatici* - vještinama primjenjenim u izgradnji korektnog programa (planiranje, dekompozicija, kodiranje, testiranje, pronalaženje i otklananje pogrešaka).

2.3. Tradicionalna nastava programiranja

U tradicionalnoj nastavi programiranja, učitelj postupno objašnjava svojstva jezika kroz izabrane primjere. Razumijevanje instrukcija učenici produbljuju samostalno rješavajući postavljene zadatke, pri čemu učenikov program može sadržavati pogreške kategirizirane kao:

- sintaksne pogreške
- semantičke pogreške nezavisne zadatku
- semantičke pogreške zavisne zadatku

Pridjeljivanje cjelobrojnoj varijabli vrijednosti s tekućim zarezom ili ponavljanje koje nikada neće završiti su primjeri semantičkih pogrešaka nezavisnih zadatku. Semantičke pogrešake zavisne zadatku predstavljaju razliku ostvarenog i namjeravanog ponašanja programa.

Pored toga, učenik može zapisati program koji proizvodi korektne izlaze, ali uz primjenu lošeg stila. Cilj učenje programiranja nije postizanje isključivo korektnih izlaza, nego korektnog, stilski ispravnog programa.

Pogreške učeničkih programa u pravilu su individualne i vrlo različite. Učenička skupina istog tečaja može biti izrazito heterogena s obzirom na motivaciju, predznanje, sposobnost i stil učenja. Tradicionalni pristup nastavi programiranja, u kojem jedan učitelj treba ustanoviti i objasniti pogreške svakom učeniku, često pred učitelja stavlja izazovnu i napornu zadaću. Uvođenjem u nastavu programiranja tutorskog sustava, koji je u stanju locirati i objasniti individualne pogreške, proces učenja i poučavanja programiranja se značajno unapređuje i olakšava.

Vrednovanje LISP Tutora pokazuje da učenici ostvajuju približno isto postignuće učeći uz čovjeka-tutora 11,4 sati, računalnog-tutora 15 sati ili 40 sati u tradicionalnoj nastavi.

2.4. Temelji tutorskih sustava za poučavanje programiranja

Oblikovanje tutorskih sustava za poučavanje programiranja temelji se na (1) umjetnoj inteligenciji, (2) teoriji inteligentnih tutorskih sustava, (3) kognitivnoj psihologiji učenja programiranja i (4) prevođenju programskih jezika.

2.4.1. ITS kao inteligentan agent

S aspekta umjetne iteligencije, tutorski sustav se može promatrati kao inteligentan agent čiju

okolinu predstavlja učenik sa svojim znanjem programiranja. Percepcija stanja okoline se postiže kroz analizu ponašanja učenika pri rješavanju postavljenih programskih zadataka. Sustav djeluje na učenikovo znanje prikazom nastavnih jedinica čiji sadržaj nedostaje u njegovu znanju. Korisno djelovanje sustava se mjeri postignućem učenika u rješavanju programskih problema. Okolina u kojoj sustav djeluje je nedeterministička, sekvencijalna, stohastička i diskretna.

Djelovanje sustava se očituje kroz dvorazinsku povratnu vezu: stratešku i operativnu. Strateško djelovanje perceptira sveukupno učenikovo znanje i pronalazi razliku u odnosu na ciljno znanju, a operativno u fokus stavlja tekući zadatak, pronalazeći razliku između očekivanog i postignutog rješenja.

2.4.2. Inteligentni tutorski sustavi

U tipičnom inteligentnom tutorskom sustavu nalazimo module područnog eksperta, model učenika, modul učitelja i komunikacijski modul.

Ekspertni sustav "zna" korektno riješiti specificirani problem iz područja poučavanja. Pored toga, raspolaže zbirkom karakterističnih pogrešaka i u stanju je dijagnosticirati učenikovo rješenje.

Model učenika predstavlja bazu podataka o tekućem stanju učenikova znanja.

Računalni učitelj na temelju modela učenika odlučuje o napredovanju nastavnim sadržajem, bira primjerene zadatke i objašnjava pogreške.

Modul komunikacije implementira korisničko sučelje i podržava razmjenu informacija učenika sa sustavom.

2.4.3. Kognitivna psihologija učenja

Kognitivna psihologija naglašava značaj smislenog učenja pri usvajanju novog znanja. Aktivacijom postojećeg znanja, nove se informacije povezuju s onima koje učenik posjeduje u svojoj trajnoj memoriji. Mayer [21] argumentira kako konkretni modeli, bliski učeniku, kojima se apstraktni procesi računalnog programa čine vidljivim, pomažu razumijevanju tehničkih informacija. Razumijevanje svojstava jezika se može značajno unaprijediti, opisuju li se akcije programskih instrukcija prirodnim jezikom.

Sasvim općenito, pristupi tutorskog poučavanja mogu se svrstati u preliminarno, interaktivno i refleksivno [17].

Preliminarno poučavanje - tutor unaprijed priprema učenika kako će riješiti konkretan zadatak, najčešće navodeći rješenja sličnih problema. Tutor ne utiče na postupak rješavanja, a konačno rješenje se vrednuje kao korektno ili pogrešno, bez objašnjenja pogrešaka.

Interaktivno poučavanje – učenik pristupa rješavanju zadatka primjenjujući prezentirano opće znanje područja, bez prethodnih uputa vezanih uz konkretan problem. Tutor odmah intervenira, čim učenik nepovratno odstupi od putanje korektnog rješenja ili bezidejno odustaje od samostalnog traženja rješenja.

Refleksivno poučavanje – učenik samostalno rješava zadatak bez intervencija tutora, a potom tutor provodi reviziju konačnog rješenja i objašnjava pogreške.

Uspoređujući pristupe tutorskog poučavanja, uočava se potpuni izostanak operativne povratne veze u preliminarnom pristupu. Povratna veza refleksivnog pristupa često može biti zakašnjela, dopuštajući protezanje ranih pogrešaka cijelim programom, zbog čega se mogu kumulirati nove pogreške. Ispravak niza pogreški predstavlja tešku zadaću većini početnika.

Poučavanje daje najbolje rezultate primjenom interaktivnog poučavanja, uz povratnu vezu bez odgode, jer ne dopušta da učenik značajno odstupi s putanje korektnog rješenja. Nedostatak ovog pristupa, kada je jedini raspoloživ u sustavu, dolazi do izražaja kod učenika u naprednim fazama učenja.

2.4.4. Prevođenje programskih jezika

Teorija prevođenja programskih jezika pruža neophodnu osnovicu izgradnje prevoditelja, dijagnosticiranja sintaksnih pogrešaka, kao i semantičkih pogrešaka nezavisnih zadatku. Isto tako, neki sustavi temelje semantičku analizu zavisnu zadatku na apstraktnom semantičkom stablu, tokovima upravljanja programom i tokovima podataka.

2.5. Karakteristični tutorski sustavi

Specifična značajka tutorskih sustava za poučavanje programiranja je lociranje i objašnjenje semantičkih pogrešaka zavisnih zadatku. Kako ovakve pogreške predstavljaju relaciju između namjeravanog i zapisanog programa, pristupe semantičke analize razlikujemo prema izvoru znanja o namjeri.

Dinamičko i simboličko testiranje namjeru zapisuju očekivanim izlazima (numeričkim,

odnosno simboličkim), verifikacijski pristupi programskim specifikacijama, te aksiomatski logičkim izrazima, tj. tvrdnjama koje moraju biti zadovoljene u pojedinim točkama programa. Zajedničko obilježje ovih pristupa je nemogućnost preciznog lociranja i objašnjenja pogreške.

Pristupi automatiziranog razumijevanja programa dobro lociraju i objašnjavaju pogreške u prepoznatim segmentima programa. Namjeru zapisuju uzorcima koda, modelom rješenja primjenom produkcijskih pravila ili semantičkom mrežom. Prepoznati se mogu samo oni segmenti programa za koje postoje predviđeni uzorci.

Sustavi zasnovani na dijalogu, namjeru otkrivaju u interakciji s učenikom primjenom izbornika ili prirodnog jezika.

U odjeljcima koji slijede, kronološki se prikazuju značajke izabranih tutorskih sustavi za poučavanje programiranja, s naglaskom na inovacije koje su uveli u područje.

2.5.1. BIP

Sustav BIP je razvijen na Sveučilištu Stanford 1976. za poučavanje BASIC-a primjenom preliminarnog pristupa [3]. Dok su prethodni sustavi prezentirali učeniku nastavnu materiju i postavljali zadatke nezavisno stanju njegova tekućeg znanja, BIP uvodi prilagodljivost modelu učenika.

BIP opisuje nastavni sadržaj usmjerenim grafom, gdje su čvorovi programske tehnike, vještine i zadaci. Lukovi povezuju svaki zadatak s vještinama koje primjenjuje, te vještine s programskim tehnikama. Mreža nastavnog sadržaja omogućuje individualizirani izbor zadatka prema modelu učenika i postavljenom cilju. Prema rješenju zadatka BIP ažurira model učenika vještinama i tehnikama koje su povezane sa datim zadatkom. BIP korektnost rješenja provjerava dinamičkim testiranjem.

2.5.2. LAURA

LAURA automatski debugira učeničke FORTRAN programe. Sustav je razvijena na Sveučilištu Caen, Francuska i predstavljen 1980. godine [1]. Glavna značajka sustava je prikazivanje programa usmjerenim grafovima čime su izbjegnute sintaksne varijacije teksta programa. Čvorovi grafa predstavljaju programske operacije (pridjeljivanje, testove uvjeta, ulaze, izlaze), a usmjereni lukovi tok upravljanja programom. Semantička analiza se provodi utvrđivanjem izomorfnosti grafova učenikovog i modela rješenja, uz primjenu normalizacije.

Preslikavanje čvorova grafa učenikova programa u čvorove grafa modela, započinje izgradnjom liste hipoteza. Postupkom «čvrsto» kodiranog rezonoranja se ažurira vjerojatnost primjenjivosti svake hipoteze. Vjerojatnost identičnosti dva čvora je veća, ako sadrže iste konstante, već povezane varijable, te isti broj nepovezanih varijabli.

2.5.3. PROUST

Sustav PROUST su razvili W.L. Johnson i E. Soloway 1984. na Sveučilišti Yale, New Haven, s ciljem da pomognu učenje Pascala i stjecanje opće vještine rješavanja problema [15].

Autori nalaze kako prethodnim metodama primjenjenim u sustavima za poučavanje programiranja, nedostaje detaljno razumijevanje relacija između postignutog teksta i namjere programa. Koristeći knjižnicu programskih planova, PROUST dekomponira cilj programa i generira stablo kojim predstavlja različita, moguća rješenja problema. Korektnost programa se analizira usporedbom učenikova s generičkim rješenjem. Kada se programi razlikuju, primjenjuju se operatori tzv. bug generatora prema deskriptivnoj teoriji učenja programiranja. Primjenjivost bug-generatora otkriva učenikovo pogrešno mišljenje koje je rezultiralo semantičkim pogreškama.

Autori ističu poteškoću neodređenosti, zajedničku svim netrivijalnim programima:

- dekompozicija problema nije jedinstvena
- isti program može biti pridružen većem broju dekompozicija problema

Vrednovanje PROUST-a pokazuje visoku učinkovitost lokalizacije pogrešaka (94%). Ipak, mnoge pogreške nije u stanju precizno identificirati i objasniti, možebitno zbog primjene refleksivnog pristupa poučavanja. Naime, zbog zakašnjele pomoći tutora, programi manje naprednih učenika kumuliraju niz zavisnih pogrešaka, pa ih PROUST nije u stanju razumjeti [17].

2.5.4. LISP tutor

LISP tutor su razvili John R. Anderson i Brian J. Reiser na Sveučilištu Carnegie-Mellon u Pittsburgu 1985. radi provjere postavki Andersonove ACT-R teorije [2]. Prema ovoj teoriji programiranje se najbolje uči u

interaktivnom pristupu s povratnom vezom bez odgode.

Glavna značajka LISP tutora je područni ekspertni sustav koji specificirane probleme može riješiti i zapisati LISP kodom. Pored toga, raspolaže zbirkom bugova koji se često nalaze u učeničkim programima. Podrška područnom ekspertu je GRAPES, ciljem ograničeni produkcijski sustav.

Sustav razlikuje modove planiranja i kodiranja programa. Dijalog s učenikom održava putem izbornika, koji nudi ispravne i neispravne planove. Autori su ovu tehniku dijagnoze učenikova znanja nazvali slijeđenjem modela *(eng. Model Tracing).*

Što će računalni učitelj kazati učeniku, kada mu preporučiti dodatno vježbanje, a kada ga uputiti na slijedeću lekciju odlučuje se uz pomoć računalnog tutora zasnovanog na produkcijskim pravilima.

LISP tutor sadrži oko 325 produkcijskih pravila planiranja i pisanja LISP programa, te još 475 pravila pogrešnih rješenja. Učinkovito je dijagnosticirao između 45 i 80% učeničkih pogrešaka. Vrednujući sustav, učenici su zamjerili veliki broj izbornika, koji prethode svakom koraku oblikovanja programa.

2.5.5. INTELLITUTOR II

INTELLITUTOR II je web orjentiran sustav kojeg je razvio tim Haruki Uena u periodu 1989-99. na Sveučilištu Tokio Denki, Japan [23]. Refleksivno poučava programske jezike C i Pascal. Sastavnice sustava su GUIDE, uređivač izvornog koda programa sa sustavom pomoći, te ALPUS, koji podržava razumijevanje programa utemeljeno na bazi znanja algoritama, programskih tehnika, uporabe varijabli i karakterističnih bugova.

Ključnu ulogu u detekciji učeničkih pogrešaka igra algoritamski orjentirana baza znanja, gdje su algoritmi predstavljeni hijerarhijskom strukturom HPG organiziranom u semantičkoj mreži. Razumijevanje učenikova programa se postiže podudaranjem uzoraka programa s predlošcima baze znanja kroz četiri koraka: (1) prevođenjem učenikova C ili Pascal programa u apstraktni jezik AL, (2) normalizacijom učenikova AL programa, (3) identifikacijom uloge varijabli primjenom podacima vođenog rezoniranja i (4) identifikacijom procesa u programu udruživanjem HPG čvorova s odgovarajućim segmentima učenikova programa.

2.5.6. PROPL

Sustav je razvijen na sveučilištu Pittsburg, 2003. godine [15]. Autori H.C. Lane i K. VanLehn nalaze dekompoziciju problema najvećom barijerom u svladavanju vještine programiranja. Dekompoziciju vide kao postupak pronalaženja programskih komponenti rješenja koje uključuju ciljeve, sheme i objekte. Ciljeve definiraju svrhom, odnosno, što je potrebno postići prema opisu zadatka, sheme su metode kojima se ciljevi postižu, a objekti podaci koje programer treba u rješenju.

Vodeći učenika u smjeru utvrđivanja navedenih komponenti, PROPL s učenikom održava interaktivni dijalog prirodnim jezikom. Ukoliko je učenikov odgovor očekivan i ispravan, sustav prirodnim jezikom zapisuje jasno razumljiv opis cilja, metode za njegovo postizanje, kao i tekući korak programa pseudokodom. Na prepoznati, neispravan odgovor sustav postavlja novo pitanje koje učenika usmjerava prema ispravnom rješenju.

PROPL-ov primarni izvor znanja je knjižnica konstrukcije dijaloga. Kreira ju autor nastavnog sadržaja, definirajući za svaku situaciju pitanje i dvije liste očekivanih riječi odgovora. Jedna lista sadrži očekivane riječi ispravnih, a druga neispravnih odgovora. Sukladno listi s većim brojem pogodaka, odgovor se nalazi ispravnim, neispravnim ili nedovoljno jasnim.

2.5.7. BITS

Sustav BITS, razvijen na Sveučilištu Regina, Kanada, objavljen je 2006. godine [6]. BITS je web orjentirani sustav za adaptibilno vođenje učenika nastavnim sadržajem. Na temelju modela učenika može preporučiti ciljeve učenja i generirati najbolji slijed napredovanja.

Nalazeći nemogućim egzaktno odrediti učenikovo znanje, autori primjenjuju Bayesian mrežu, kao dobro definiran okvir vjerojatnosnog zaključivanja. BITS nastavni sadržaj posvećen reduciranom jeziku C++ prikazuju Bayesian mrežom, te ju prema učenikovim odgovorima na kviz pitanja, ažurira vjerojatnosnim faktorima znanja.

2.6. Analiza postojećih sustava

U cilju usporedbe tutorskih sustava, definiraju se atributi:

Cjelovitost poučavanja – polazi od Linn-Dalbey standarda za usporedbu metoda poučavanja

programiranja i određuje u kojoj mjeri pojedini sustav doprinosi učenikovom napredovanju karikama spoznajnog lanca.

Jezična nezavisnost - podrazumijeva raspoloživost baze programskog znanja nezavisne konkretnom jeziku, pri čemu sustav na temelju svog «znanja» svojstava jezika, generira jeziku usmjereni nastavni sadržaj.

Stupanj razumijevanja programa – kvantificira sposobnost sustava u lociranju i identifikaciji semantičkih pogrešaka u realnim učeničkim programima.

Analizirijući postojeće sustave, nalazi se izostanak cjelovitosti poučavanja programiranja. Iako različiti sustavi podržavaju većinu sastavnica lanca učenja programiranja, ne postoji sustav koji ih ujedinjuje. Brusilovsky takvo stanje objašnjava iznimnim naporom od nekoliko čovjek godina koji je potrebno uložiti u istraživanje i razvoj svake komponente cjelovitog sustava [4]. Postojeći sustavi ne poklanjaju dovoljnu pažnju učenju svojstava jezika i vrednovanju učenikova znanja na toj razini. Nalazi studije [19] argumentiraju da je ovoj karici učenja programiranja potrebno posvetiti punu pažnju.

Sustavi se fokusiraju na planiranje i dekompoziciju problema, te automatsko debugiranje koje podrazumijeva tutorsko lociranje i dijagnosticiranje semantičkih pogrešaka. Važna odlika programera je samostalno lociranje i otklananje pogrešaka u realnom programu. Zamjećuje se izostanak učenja i vježbanja ove važne vještine pod nadzorom tutorskog sustava.

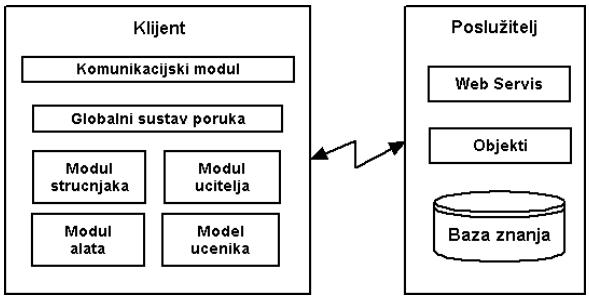
Postojeći sustavi su, uz neke izuzetke, posvećeni jednom programskom jeziku. Niti jednom sustavu se ne može pripisati atribut jezične nezavisnosti.

3. Odrednice oblikovanja sustava

Polazeći od sveukupnog ljudskog znanja u ovoj oblasti, oblikovanje idealnog sustava koji bi u potpunosti zamijenio čovjeka-tutora se može ocijeniti utopijskim planom. Ipak, izgradnja sustava kao komplementa tradicionalnoj nastavi i osnovice budućeg napredovanja prema idealnom, čini se realnim ciljem i ostvarivom zadaćom.

3.1. Poslužitelj – klijent arhitektura

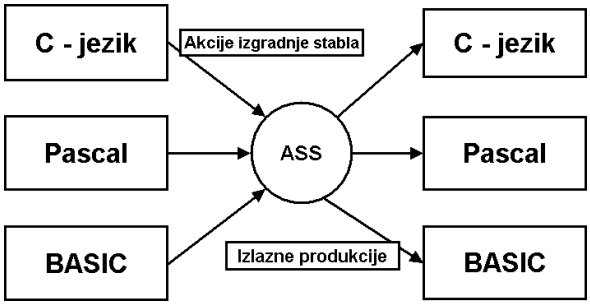
Predloženi sustav se temelji na poslužitelj – klijent arhitekturi s osloncem na web servise [9]. Arhitektura iskorištava obradnu snagu klijenta, radi analize učenikovih aktivnosti u realnom vremenu, te centraliziranu bazu znanja i konfiguraciju alata (uređivača teksta, jezičnog prevoditelja).



Slika 1. Klijent – poslužitelj arhitektura sustava

3.2. Jezična nezavisnost

Sustav je jezično nezavisan. Novi jezik poučavanja se uvodi definiranjem leksičkog analizatora i gramatike zapisane u Bachus-Naur formi, obogaćenoj akcijama za izgradnju apstraktnog semantičkog stabla. Nastavni sadržaj je jedinstven, a sustav ga dinamički prilagođava izabranom programskom jeziku.



Slika 2. Međujezično prevođenje

Analiza programa se provodi na razini jedinstvenog apstraktnog semantičkog stabla. Primjenom izlaznih gramatičkih produkcija pridruženih čvorovima stabla, sustav može zapisati program ciljnim jezikom. Sustav izgrađuje izlazne produkcije analizom gramatike jezika [10].

3.3. Cjelovitost poučavanja

Sustav cjelovito poučava sve sastavnice programskog znanja. Sintaksu jezika poučava primjenom scenarija računalnog trenera, interaktivno analizirajući program znak po znak, kako ga učenik zapisuje [8]. Uvođenjem apstraktno-pojmovnog modela, definiranog programskim jezikom, sustav u fazi izvođenja programa korak po korak, verbalizira akcije svake instrukcije i prikazuje učinak izvođenja na

stanje varijabli [9]. Učenikovo znanje semantike programskih instrukcija se provjerava zadacima jednostavne reformulacije i trasiranja programa. Trasiranje zahtjeva da učenik misaono simulira izvođenje programa i predvidi vrijednosti varijabli u pojedinim koracima programa. Učenike u naprednoj fazi učenja, sustav vodi u samostalnom testiranju, pronalaženju i korekciji logičkih pogrešaka.

3.4. Semantička analiza programa

Sustav razlikuje napredne i učenike s poteškoćama [22]. Prvi mogu nesmetano eksperimentirati s programom, jer se njihovi programi semantički analiziraju na poziv, u fazi prevođenja. Programi učenika s poteškoćama se analiziraju interaktivno, tehnikom slijeđenja modela uz primjenu Hartmanove dekompozicije [14].

Semantička analiza zavisna zadatku temelji se na primjeni tehnike problema zadovoljenja ograničenja *(eng. Constraint Satisfaction Problem)*. Ograničenja se definiraju automatski, prema modelu rješenja zadatka. U odnosu na poznatu primjenu ove tehnike [24], kojom se analiziraju isključivo korektni programi, istražuje se primjenjivost u uvjetima najvjerojatnijeg programa, tj. pronalaženja preslikavanja učenikova programa u model s minimalnim brojem prekršenih ograničenja.

Slikovito kazano, razumijevanje korektnog programa, pronalaženjem programskih konstrukcija prema bazi znanja, je poput traženja igle u plastu sijena. Prepoznavanje nekorektnog programa nalikuje traženju predmeta u plastu sijena koji najviše nalikuje igli, uz identifikaciju akcija čijom se primjenom taj predmet transformira u iglu.

U cilju minimizacije prostora pretraživanja, za svaki se programski zadatak, uz pomoć računalnog eksperta, generira model rješenja. Model se zapisuje posebno razvijenim programskim jezikom [10], koji omogućuje zapisivanje koraka rješenja različitim algoritmima i algoritamskim varijacijama, uz primjenu mogućih instrukcija oživotvorenja. Isto tako, jezik omogućuje zapisivanje pogrešnih algoritamskih koraka, te hijerarhije ciljeva i pripadnih im podciljeva. Ciljevi odgovaraju identifikatorima predložaka uzoraka koda primjenjenih u rješenju.

4. Vrednovanje predloženog sustava

Tijekom zimskog semestra 2007/08. sustav je korišten u nastavi kolegija Programiranje I, na Fakultetu prirodoslovno-matematičkih znanosti, Sveučilišta u Splitu. Tri grupe s ukupno 29 studenata su učili osnovne pojmove programiranja uz primjenu programskog jezika QBasic. Od toga 16 (55%) studenata nije imalo prethodnih programskih iskustava, dok je 13 (45%) učilo programiranje u srednjoj školi od jednog do osam polugodišta.

Na kraju tečaja studenti su odgovorili na upitnik s 29 pitanja, birajući odgovore stupnjevane Likertovom skalom. Pitanja su se odnosila na motivaciju, predznanje i dojmove o predloženom sustavu.

Tablica 1. Pomoć sustava pri učenju

Stupanj	*Odgovora*	*%*
1. Vrlo štetna	0	0.0
2. Štetna	0	0.0
3. Ne može odlučiti	3	10.3
4. Korisna	4	13.8
5. Vrlo korisna	22	75.9

Zaključuje se, da je sustav ispunio osnovnu zadaću, jer je 26 (89.7%) studenata odgovorilo kako im je sustav pomogao učiti programiranje.

Pored navedenog, studenti su na sedam stupanjskoj Likertovoj skali označili svoj nalaz o korisnosti pojedinih sastavnica poučavanja *(1 vrlo štetno, 4 neutralno, 7 vrlo korisno).*

Tablica 2. Stupnjevi korisnosti

Komponenta	*Median*	*Prosjek*	*Rang*
Pojmovni model	7	6.2	4..7
Napredovanje tečajem	5	5.0	1..7
Sintaksna pomoć	7	6.2	4..7
Semantička pomoć	7	6.7	6..7

Neki studenti nalaze štetnim napredovanje nastavnim sadržajem pod kontrolom sustava, preferirajući potpunu slobodu izbora nastavnih jedinica, čak i onda kada ne raspolažu (prema nalazima sustava) neophodnim predznanjem. Prema očekivanju, studenti prihvaćaju i visoko vrednuju pojmovni model, sintaksnu i semantičku pomoć.

5. Zaključak

Predloženi sustav poklanja punu pažnju svim sastavnicama cjelovitog programskog znanja, što

ga uz potpunu jezičnu nezavisnost čini jedinstvenim. Apstraktno – pojmovni model, definiran programskim jezikom, pomaže vizualizirati i razumjeti apstraktne procese. Vođenjem učenika postupkom oblikovanja, kodiranja i testiranja programa, od izbora testnih ulaza do lociranja pogreške, učenik se priprema za samostalan rad u realnom okruženju programiranja.

Nalazi se potrebnim istražiti mogućnost razumijevanja programa primjenom problema zadovoljenja ograničenja uz najvjerojatnije preslikavanje učenikovog u program modela rješenja.

6. Literatura

[1] Adam A, Laurent, J. P. LAURA, a System to Debug Student Programs. Artificial Intelligence 15, 1980. p. 75-122.

[2] Anderson J. R, Reiser B. The LISP Tutor. Byte 10; 1985.

[3] Barr A, Beard M, Atkinson R. C. The computer as a tutorial laboratory: the Stanford BIP project. Man-Machine Studies 8, 1976. p. 567-596.

[4] Brusilovsky P. Intelligent Learning Environments for Programming, Proceedings of AI-ED'95, 7th World Conference on Artifcial Intelligence in Education. Washington, DC; 1995.

[5] Brusilovsky P. L. Intelligent Tutor: Environment and Manual for Introductory Programming. Educational and Training Technology International; 1992. p. 26-34.

[6] Butz C. J, .Hua S, Maguire R. B. A web-based bayesian intelligent tutoring system for computer programming. Web Intelligence and Agent System, Vol. 4; 2006. p. 77-97.

[7] Chee Y. S, Tan J. T, Chan T. Applying cognitive apprenticeship to the teaching of Smalltalk in a computer based learning environment. Proceedings of 7th International PEG Conference; 1993. p. 569-588.

[8] Dadic T, Stankov S, Rosic M. Prototype Model of Tutoring System for Programming. Proceedings of ITI Conference; 2006.

[9] Dadic T, Stankov S, Rosic M.Meaningful learning in the Tutoring System for Programming. Proceedings of ITI Conference; 2008.

[10] Dadic T. Programski jezik za definiciju rješenja programskog zadatka u tutorskom sustavu za poučavanje programiranja proceduralnim jezicima, seminarski rad, 2008.

[11] DuBoulay B, O'Shea T, Monk J. The Black Box Inside the Glass Box. In Studying the Novice Programmer, Editors Soloway E, J. C. Spohrer, Lawrence Erlbaum Associates, Publishers; Hillsdale, New Jersey; 1989. p. 431.

[12] Eisenberg M. B, Johnson, D. Learning and Teaching Information Technology Computer Skills in Context. US Federal government; 2003. http://www.libraryinstruction.com/ info-tech.html.

[13] Guzdial M. Programming Environments for Novices. In Computer Science Education Research, edited by Sally Fincher and Marian Petre. Springer-Verlag; 2004.

[14] Hartman J. Understanding natural programs using proper decomposition. In Proceedings of the International Conference on Software Engineering, Austin, Texas; 1991. p. 62-73.

[15] Johnson W. L, Soloway E. PROUST: Knowledge based program understanding. Proceedings of the 7th international Conference on Software Engineering; Orlando; 1984. p. 369-380.

[16] Lane H. C, VanLehn K. Coached Program Planning: Dialogue-Based Support for Novice Program Design. Proceedings of the Thirty-Fourth Technical Symposium on Computer Science Education (SIGCSE); 2003. p. 148-152.

[17] Lane H. C. Natural Language Tutoring and the Novice Programmer, doctoral dissertation, University of Pittsburgh, 2004.

[18] Lemut, Dettori, Du Boulay: Cognitive Models and Intelligent Environments for Learning Programming, Springer-Verlag, NJ, USA, 1993.

[19] Linn M.C, Dalbey J.: Cognitive Consequences of Programming Instruction, in Studying the Novice Programmer, Editors Soloway E, J. C. Spohrer, Lawrence Erlbaum Associates, Publishers; Hillsdale, New Jersey; 1989. p. 57-81.

[20] Lister R. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. SIGCSE Bulletin, Vol. 36, Issue 4., December 2004. p. 119-150.

[21] Mayer R. E. The Psychology of How Novices Learn Computer Programming. In Studying the Novice Programmer, Editors Soloway E, J. C. Spohrer, Lawrence Erlbaum Associates, Publishers; Hillsdale, New Jersey; 1989. p. 129.

[22] Perkins D.N, Hancock C, Hobbs R, Martin F, Simmons R. Conditions of Learning in Novice Programmers. In Studying the Novice Programmer, Editors Soloway E, J. C. Spohrer, Lawrence Erlbaum Associates, Publishers; Hillsdale, New Jersey; 1989. p. 261.

[23] Ueno, H. Integrated intelligent programming environment for learning programming. IEICE Transactions on information and systems; 1994. p. 68-79.

[24] Woods S. G, Quilici A. E, Yang Q. Constraint-Based Design Recovery for Software Reengineering: Theory and Experiments. Kluwer Academic Publishers, Norwell, Massachusetts; 1998.