SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 57

Sustav za prilagodljivo poučavanje studenata izrađen u Web tehnologiji

Goran Pogačić

Zagreb, lipanj, 2010.

1. Uvod

E-učenje postaje sve raširenji način obrazovanja. Razvojem internet tehnologija i konstantnim povećavanjem broja korisnika interneta pojavljuje se sve veća potreba za učenjem preko interneta. Ideja ovog diplomskog rada je definirati teorijsku pozadinu i opisati konkretnu implementaciju sustava za e-učenje. Za razliku od većine sustava za e-učenje ovaj sustav bi trebao imati svojstvo prilagodljivosti. Sustav bi procjenjivao studenta te s obzirom na tip učenja studentu bi se prikazivao drugačiji sadržaj. Gradivo predmeta bi bilo podijeljeno na zasebne cjeline koje su u određenom preduvjetnom i hijerarhijskom odnosu. Za svaku cjelinu bi tada postojali pripadni dokumenti pisani za različite tipove učenja studenata. Ideja je da sustav vodi studenta kroz gradivo i adaptivno se prilagođava trenutnom znanju studenta kako bi najlakše savladao gradivo.

Sustav bi procjenjivao studentovo znanje kroz zadatke. Po završetku učenja svake cjeline provjeravalo bi se studentovo znanje. Postojeći sustav StudTest2 podržava dinamički generirane zadatke što ga čini idealnim mjestom za smještanje sustava za e-učenje. Osim što sustav procjenjuje studentovo znanje, kroz zadatke studenti bi dobili dobru povratnu informaciju o svojem znanju, što može puno pomoći prilikom pripremanja za ispite.

Za procjenjivanje studentovog tipa učenja koristi se podjela tipova učenja prema Kolbu [1]. Kolb je predložio postojanje dvije dimenzije karakteristika učenja koje se mogu lako prikazati u koordinatnom sustavu. Dobra strana Kolbove podjele je što prikaz u koordinatnom sustavu olakšava nastavnicima definiranje za koji tip učenja je dokument pisan. S druge strane, kvaliteta podjele se tek treba procijeniti.

Ovaj sustav bi se koristio na predmetu Digitalna logika na Fakultetu elektrotehnike i računarstva. Algoritmi i podatci potrebni za rad nalazili bi se u sustavu StudTest2. No studenti bi učili a nastavnici administrirali kroz sustav Ferko.

Postojeći slični sustavi uključuju Moodle (besplatna web-aplikacija koju nastavnici mogu koristiti za kreiranja efektivnih stranica za e-učenje [2]) i WebCT (programski alat koji se koristi za održavanje nastave na daljinu [3]). Oba sustava imaju prednosti i mane.

Razlog zbog kojeg radimo novi sustav je slaba fleksibilnost prilikom ispitivanja studenata kroz te sustave. Naime oni omogućuju samo slijedno prikazivanje već gotovih zadataka. Također ti zadatci se ograničavaju na ABC pitalice s jednim ili više točnih odgovora i pitalice tipa unos linije teksta. Sustav StudTest2 omogućava isto takve zadatke ali i puno fleksibilnije generičke zadatke. Zbog toga se čini boljim odabirom za implementaciju sustava za prilagodljivo poučavanje studenata.

U poglavlju *Model domene* opisan je način zapisivanja gradiva u sustavu pomoću strukture u obliku grafa. Na kraju poglavlja nalazi se dio koji je posvećen razradi načina zapisivanja modela unutar računala. Kroz poglavlje *Model studenta* definira se način modeliranja tipova studenata. U poglavlju *Algoritmi nad gradivom* opisani su algoritmi kojima će studenta, s obzirom na njegov tip, adaptivno poučavati gradivu. U drugom dijelu diplomskog rada opisana je implementacija sustava.

2. Model domene

Postojeće tehnologije koje rješavaju ovaj promjer su LOM i SCORM. LOM (engl. *Learning Object Metadata*) je standard koji specificira podatkovni model za opisivanje objekata za učenje [4]. SCORM (engl. *Sharable Content Object Reference Model*) je kolekcija standarda i specifikacija za e-učenje [5]. Glavni razlog za izgradnju vlastitog modela domene je jednostavnost. LOM i SCORM su složene specifikacije koje pokrivaju veliki broj slučajeva te se ne čine prikladnim. U nastavku je objašnjen model domene koji se koristi u ovom sustavu za adaptivno poučavanje.

Razmotrimo problem kroz pojednostavljeni primjer. Pretpostavimo da je potrebno izgraditi sustav koji poučava studente gradivu iz predmeta *Programiranje i programsko inženjerstvo*. Potrebno je pohraniti gradivo u sustav pa je u tu svrhu ono podijeljeno na cjeline koje student mora naučiti. Neka par odabranih cjelina budu *Programske petlje*, *Funkcije* i *Rekurzije*. Za razumijevanje pojedinih cjelina potrebno je razumjeti prvo neke druge cjeline. Na primjer, da bi student mogao učiti o rekurzijama mora prvo shvatiti koncept funkcije. Također neke cjeline sadrže manje podcjeline. Cjelina *Programske petlje* može sadržavati podcjeline za svaku vrstu petlje – *For*, *While* i *Do-While.* Svaka cjelina bi trebala imati skup materijala iz kojih student uči. Sustav treba i na neki način ispitati studenta naučeno gradivo pa za svaku cjelinu treba definirati i skup zadataka. A kao najvažnija osobina, je ideja da sustav različite studente različito poučava. Zbog toga sustav treba raspoznavati različite tipove studenata i imati materijale pripremljene posebno za svaki tip studenta.

Iz gornjeg jednostavnog primjera mogu se naslutiti glavni dijelovi sustava. Razmotrimo prvi problem – kako organizirati gradivo unutar sustava. U nastavku ovo ćemo nazivati *model domene*.

Model se koristi kada je iz domene problema potrebno izdvojiti samo ono što je bitno za rješavanje nekog problema. Domena problema je organizacija gradiva za učenje. Ona je usko vezana uz učenje pa da bi se izradio model domene mora se razmisliti o procesu učenja.

Kad se govori o učenju onda se najčešće govori o tome razumije li se ili ne neki dio gradiva. Ako se ne razumije, postavlja se pitanje zašto se ne razumije. U ovom ćemo radu dio gradiva koji student treba naučiti nazivati *segment gradiva*. Uvodimo dva tipa segmenta gradiva – *koncept* i *tema*.

Koncepti predstavljaju jednostavne segmente gradiva, atome, koji se ne mogu podijeliti na manje segmente. Točnije, unutar gradiva kojeg sustav treba poučavati ta podjela nije moguća ili nije bitna. S druge strane, tema predstavlja složeniji segment pa se samim time može podijeliti na druge manje segmente koji mogu biti složeni (teme) ili jednostavni (koncepti). Kako je to prirodan način ljudskog razmišljanja može se pretpostaviti da se većina gradiva može granulirati na ovakav način iako se ne može tvrditi da vrijedi za svako gradivo.

Postavlja se pitanje, postoji li kakav odnos između različitih segmenata gradiva? Ljudski slijed misli je u suštini linearan – ima početak i kraj. Zbog toga kada nešto učimo krenemo od jednostavnijeg prema složenijem gradivu. Istu rečenicu možemo reći na sljedeći način – prije nego što možemo shvatiti složenije gradivo moramo shvatiti ono jednostavnije. Očito je da postoji određena uvjetna veza između segmenata gradiva.

Zbog prirode domene problema u ovom radu ćemo prikazivati model domene pomoću grafa čija struktura je detaljno opisana u narednom poglavlju.

2.1. Struktura grafa modela domene

Kostur grafa modela domene čini usmjereni graf. Usmjerenom grafu je zatim dodano svojstvo da određeni tipovi čvorova mogu sadržavati druge čvorove. Ovako modificiran graf predstavlja graf modela domene.

Graf se sastoji od *čvorova* i *veza*. Čvorovi predstavljaju segmente gradiva koji trebaju biti naučeni. S obzirom na dva tipa segmenata gradiva postoje dva tipa čvora koji su jednako nazvani – *koncept* i *tema*.

Tablica 1 – tipovi čvorova u grafu modela domene

Tip čvora	Opis
Koncept	atomarni čvor
	označen punom crtom, ime pišemo unutra
Tema	sadrži podčvorove
	označena iscrtkanom crtom, ime pišemo izvana

U tablici 1 nalazi se opis značenja pojedinih tipova čvorova u grafu. Kako tema predstavlja složeni segment znanja ona se može podijeliti na manje segmente. Te cjeline su u grafu sadržane unutar teme pa ih nazivamo – *podčvorovi*. Podčvor može biti koncept ili tema. Prilikom crtanja grafa podčvorove crtamo unutar teme. Da bismo mogli reći da razumijemo temu jasno je da moramo razumjeti sve podčvorove.

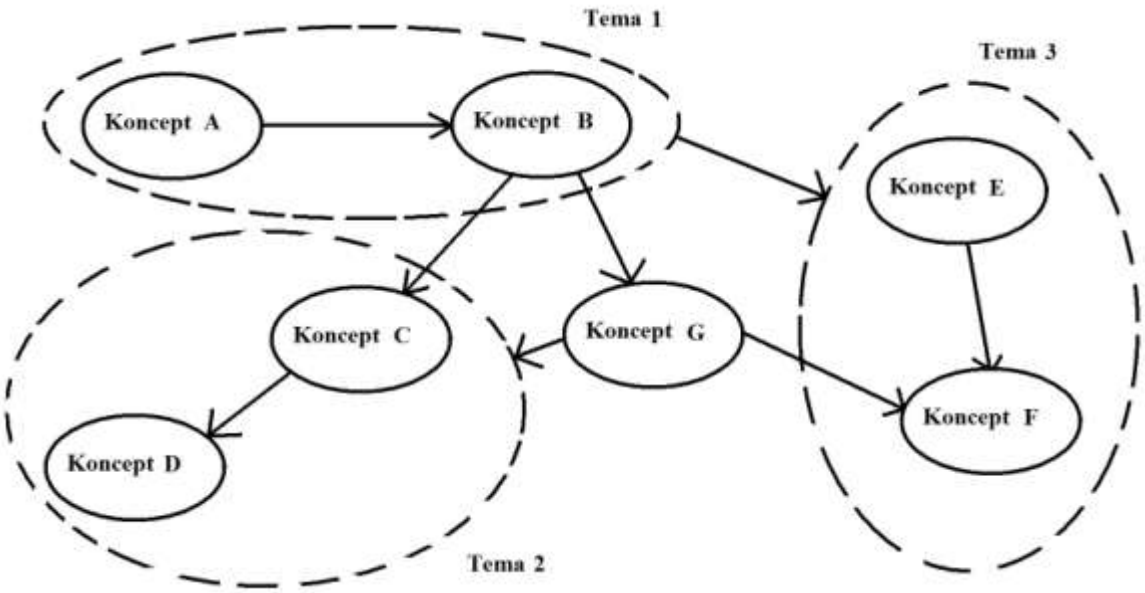
Veze su usmjerene što je označeno strelicama od jednog čvora prema drugom. Jedan primjer takve veze prikazan je na slici 1.

Slika 1 – primjer veze koncept-koncept

Veza na slici 1 znači da je A *preduvjet* od B odnosno, da se A treba naučiti prije nego što se prijeđe na B. Slika 1 je primjer povezivanja dva koncepta međutim veza u grafu je definirana nad svim kombinacijama koncepta i teme:

- koncept – koncept,
- koncept – tema,
- tema – koncept te
- tema – tema.

Složeni primjer grafa modela domene s nekim od gore spomenutih veza prikazan je na slici 2.

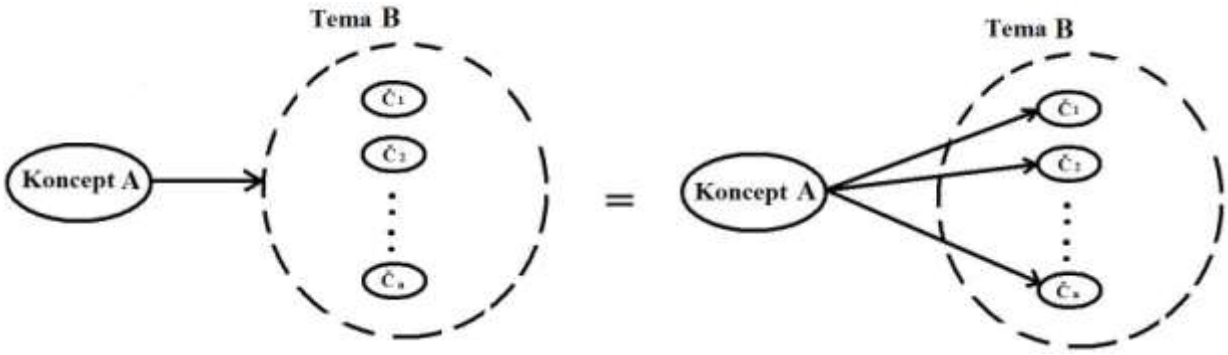


Slika 2 – primjer grafa modela domene

Pri tome stvaranje ciklusa nije dopušteno. To je zbog toga što postojanje ciklusa nije niti u prirodi ovakvog modela domene. Naime ako bi jedan koncept imao kao preduvjet neki drugi koncept a taj drugi istovremeno kao preduvjet prvi koncept to bi značilo da se za razumijevanje prvog mora znati drugi i obrnuto. To povlači da se ne mogu znati jedan bez drugoga, odnosno da zajedno predstavljaju nedjeljivu cjelinu. No to je suprotno pretpostavci da je koncept atom.

Ovako definirana veza u grafu modela znanja naziva se *preduvjetna* veza. Preduvjetna veza između bilo koja dva čvora može se svesti na preduvjetnu vezu koja je striktno definirana samo između koncepata. Postoje tri situacije: koncept – tema, tema – koncept i tema – tema.

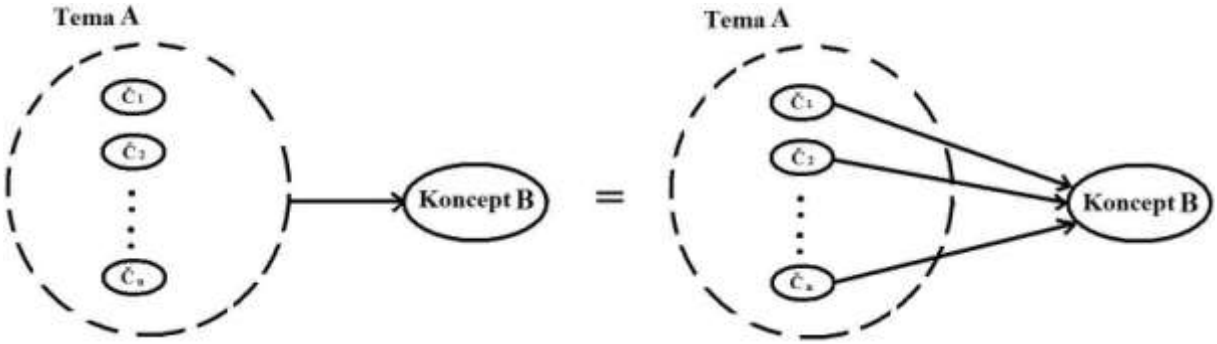
Na slici 3 vidimo situaciju kada je koncept preduvjet temi.



Slika 3 – preslikavanje iz veze koncept-tema u vezu koncept-koncept

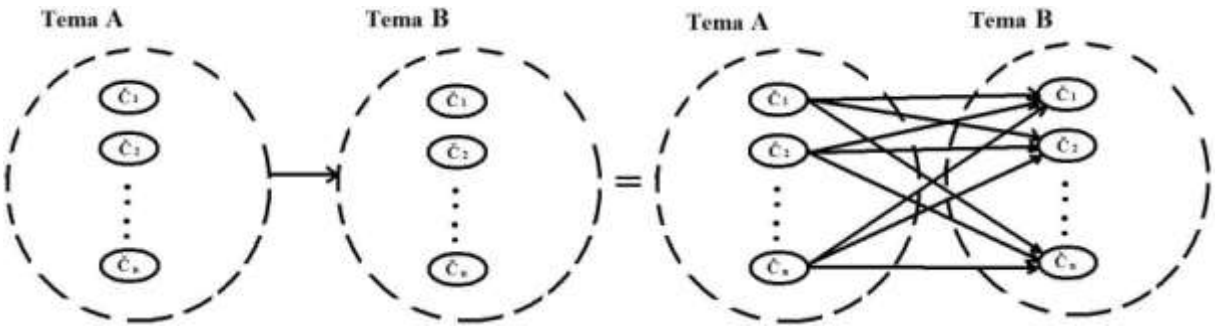
Značenje lijevog dijela slike 3 je da se za razumijevanje teme B mora razumjeti koncept A. To se može prevesti u sljedeće: za razumijevanje svakog podčvora koji se nalazi unutar teme B treba se razumjeti koncept A što je upravo ono što je prikazano na desnom dijelu.

Na slici 4 je prikazana situacija gdje je tema preduvjet konceptu.



Slika 4 – preslikavanje iz veze tema-koncept u vezu koncept-koncept

Da bi se razumio koncept B treba razumjeti temu A, kao što je to prikazano na slici 4. To znači da se mora razumjeti svaki podčvor unutar teme A da bi se razumio koncept B.



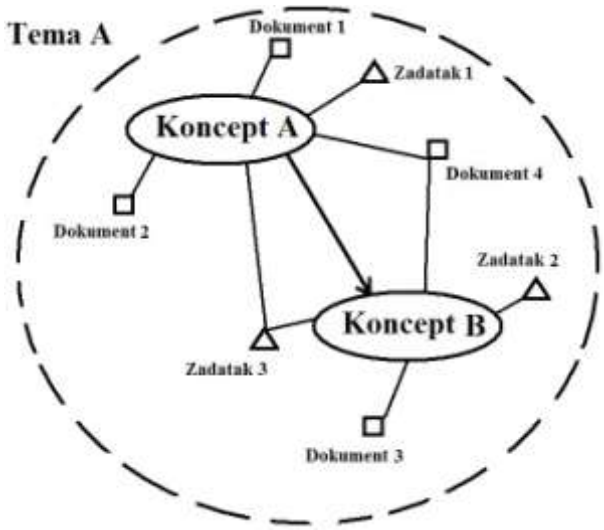
Slika 5 – preslikavanje iz veze tema-tema u vezu koncept-koncept

Slično kao i u prethodnim situacijama, svaki podčvor teme A mora se razumjeti prije nego što se može razumjeti bilo koji podčvor iz teme B.

Ukoliko je u situacijama prikazanim na slikama 3 do 5 podčvor tema postupak se ponavlja dok niti jedna tema nije direktno vezana za neki drugi čvor. Ovim postupkom se eliminiraju direktne veze između tema i drugih čvorova. Sadržajnost koncepta ili teme unutar druge teme pri tome je sačuvana.

Prethodnim razmatranjem definirana je struktura segmenata gradiva kojeg student treba naučiti. Pogledajmo kako je osnovni model još proširen dodatnim vezama. Svaki segment gradiva može imati na sebe povezane *dokumente* i *zadatke*. Dokumenti predstavljaju materijale iz kojih student uči o tom segmentu. Zadatci služe za provjeru je li student naučio određeni segment gradiva odnosno, daju ocjenu kvalitete znanja. Povezanost dokumenta ili zadatka s čvorom nazivamo *prekrivenost*. To znači da dokument objašnjava segment gradiva odnosno da zadatak ispituje znanje o segmentu gradiva. Jedan dokument ili zadatak može prekrivati više čvorova.

Radi preglednosti dokumenti i zadatci se preskaču prilikom crtanja grafa modela domene. Ukoliko je ipak potrebno vidjeti u samom grafu modela domene koje čvorove prekrivaju dokumenti i zadatci to se može napraviti kao što je prikazano na slici 6. Prilikom rada sa sustavom prikaz prekrivenosti se može ostvariti jednostavnim prikazom liste čvorova odabirom nekog dokumenta ili zadatka.



Slika 6 – graf modela domene s dokumentima i zadatcima

U tablici 2 dan je prikaz svih implicitnih i eksplicitnih veza u grafu modela znanja.

Tablica 2 – pregled svih vrsta veza u grafu modela domene

Vrsta veze	Opis nastanka	Značenje
preduvjetna veza	nastaje povezivanjem dva čvora u grafu	preduvjet za razumijevanje
sadržajnost u temi	nastaje smještajem nekog čvora unutar teme	smješteni čvor je dio te teme – podčvor
prekrivenost čvora nekim dokumentom ili zadatkom	nastaje povezivanjem dokumenta ili zadatka s nekim čvorom	dotični dokument ili zadatak služi za učenje odnosno provjeru znanja o segmentu gradiva kojeg taj čvor predstavlja

Ovim smo zaključili razmatranja modela domene. U nastavku ćemo detaljnije proučiti koje je sve podatke potrebno pamtiti u sustavu.

2.2. Metapodaci

Sustav bi trebao donositi odluke prilikom poučavanja studenata s ciljem najboljeg učinka. Za odluke potrebni su podatci pa u sljedećem poglavlju definiramo što sve sustav „zna“ o pojedinim dijelovima modela domene.

2.2.1. Segment gradiva

U tablici 3 dan je prikaz metapodataka koje sustav pamti za svaki segment gradiva. Skup metapodataka može se lako proširiti ukoliko nastane potreba za time.

Tablica 3 – metapodatci segmenta gradiva

Naziv	Opis
URI	jedinstveni identifikator pojedinog segmenta (Uniform Resource Identifier)
težina segmenta	mjera koja označava težinu savladavanja segmenta gradiva
lista preduvjeta	skup poveznica na sve segmente koji su preduvjeti dotičnom segmentu
težina preduvjeta	mjera koja označava u kolikoj mjeri student treba znati određeni preduvjet da bi mogao učiti dotični segment gradiva

Lista preduvjeta puni se automatski nakon povezivanja dva čvora u grafu modela domene. Dobro je uočiti da se u popis preduvjeta segmenta koji je koncept implicitno (automatski) mogu dodati koncepti koji nisu eksplicitno povezani na grafu. To je moguće zbog preslikavanja preduvjetne veze između tema i ostalih čvorova u preduvjetne veze samo između koncepata.

Težina segmenta predstavlja mjeru koja sustavu govori koliko je teško savladati segment gradiva. Ova mjera nema apsolutni opseg već treba samo biti relativna s obzirom na težine ostalih segmenata unutar predmeta. Vrlo važan metapodatak je težina preduvjeta. Ova mjera označava koliko je neki preduvjet bitan za određeni segment odnosno, koliko student dobro mora znati preduvjetni segment.

Segmenti koji su teme sadrže dodatno listu podčvorova.

2.2.2. Dokumenti i zadatci

Dokumenti predstavljaju materijale iz kojih student uči o nekom segmentu gradiva. Zadatci predstavljaju bilo kakvu provjeru koja vraća procjenu znanja studenta o tom segmentu.

Dokumenti sadrže dva metapodatka. Prvi je lista segmenata koje on pokriva. Drugi metapodatak je oznaka koja govori kojem tipu učenja studenta taj dokument pripada. Model studenta detaljno je obrađen u sljedećem poglavlju.

Kako za opisivanje nekog segmenta možda neće biti dovoljna samo jedna datoteka u ovom diplomskom radu pod dokument podrazumijevat ćemo *zip* arhivu koja sadrži sve datoteke koje čine dokument te posebnu datoteku *opisnik.txt*. U opisniku će se nalaziti informacije o tipu dokumenata, početnoj datoteci i sl. Primjer datoteke *opisnik.txt* se nalazi unutar isječka koda 1.

Isječak koda 1 – primjer *opisnik.txt* datoteke

```
vrsta=html
naziv=index.html
```

Zadatci sadrže samo jedan metapodatak koji je isti kao i kod dokumenata – prekrivenost segmenata.

3. Model studenta

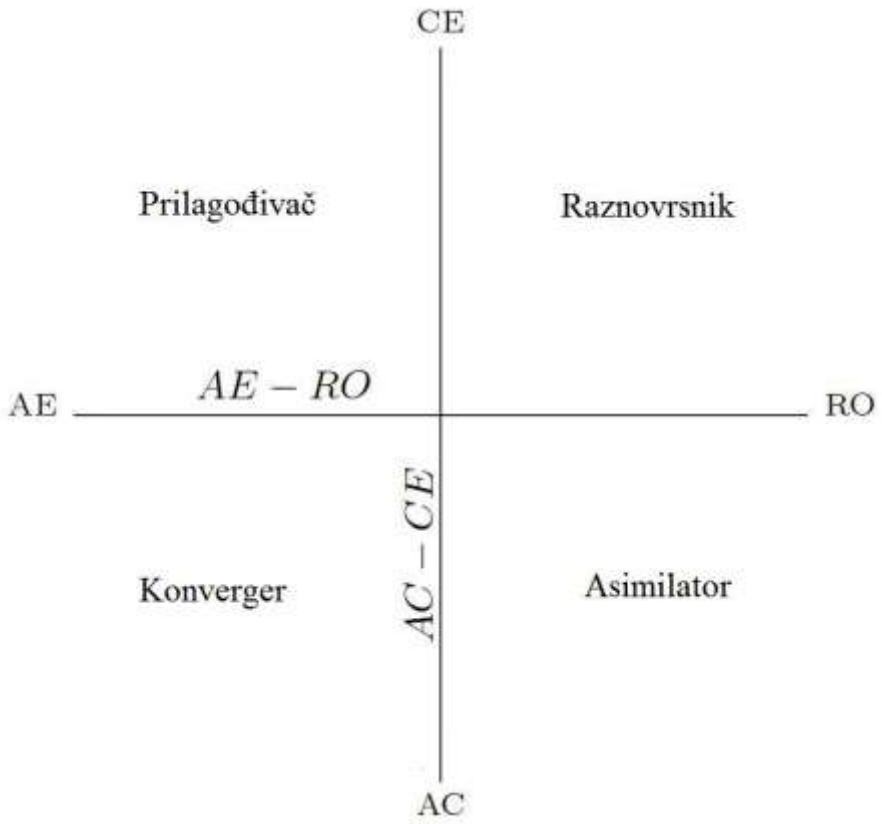
Cilj sustava je poučavanje studenata. Najveća kvaliteta sustava prilikom poučavanja treba biti sposobnost prilagođavanja studentima s različitim tipovima učenja. Prvi korak je definiranje koje će sve podatke sustav pamtiti o studentu – *model studenta*. Zadaća je modela omogućiti pridjeljivanje studentu određeni tip s obzirom na način učenja. Temeljem podataka o studentu sustav ima mogućnost različitog poučavanja.

Modeliranje studenta s obzirom na način učenja je vrlo popularan problem u današnjoj znanstvenoj zajednici. Od psihologije preko pedagogije do računalne znanosti, svugdje je postavljeno mnogo različitih teorija. Jedna od njih je Kolbova podjela stilova učenja. David Kolb identificirao je dvije dimenzije učenja, svaka sa dvije krajnosti. Jedna dimenzija uspoređuje razmišljanje naspram osjećanja dok druga dimenzija uspoređuje reflektiranje naspram djelovanja.

Prva dimenzija na jednom kraju ima apstraktnu konceptualizaciju (engl. *abstract conceptualization*, AC) gdje je učenje bazirano na korištenju koncepata za razumijevanje. Na drugom kraju ima konkretno iskustvo (engl. *concrete experience*, CE) gdje je učenje bazirano na osjećanju i specifičnostima trenutne situacije.

Druga dimenzija na svom jednom kraju ima reflektirajuće promatranje (engl. *reflective observation*, RO) gdje je učenje bazirano na promatranju i uzimanju u obzir različitih pogleda na istu stvar. Suprotni kraj druge dimenzije je aktivno eksperimentiranje (engl. *active experimentation*, AE) gdje se učenje ostvaruje djelovanjem i fokusom na pragmatične stvari.

Postojanje dvije dimenzije nam omogućuje prikaz u koordinatnom sustavu – obilježja AC-CE i RO-AE mogu se izračunati za svakog studenta i time jednoznačno odrediti točku u koordinatnom sustavu koja predstavlja tip tog studenta. Jedan primjer koordinatnog sustava prikazan je na slici 7.



Slika 7 – koordinatni sustav tipova učenja prema Kolbu

U tablici 4 prikazana su obilježja pojedinih tipova.

Tablica 4 – obilježja tipova učenja prema Kolbu

Tip učenja	Obilježja
Konverger	naglasak na razmišljanju i djelovanju, rješavanju problema, donošenju odluka i primjeni tehnički-fokusiranih ideja
Asimilator	preferiraju razmišljanje i reflektiranje, s fokusom na modele i integraciju informacija, manji fokus na ljude
Raznovrsnik	naglašeno osjećanje i reflektiranje, povezanost s imaginacijom, generiranjem ideja i fokusom na ljude
Prilagođivač	fokus je na osjećanju i djelovanju, brzo se prilagođavaju trenutnoj situaciji, metode pokušaja i promašaja, oslanjaju se na ljude za informacije

Prednost Kolbove podjele je mogućnost prikaza studenata kao točke u koordinatnom sustavu. To je izuzetno bitno jer prilikom izrade dokumenata nastavnik mora reći sustavu za koji je tip taj dokument prikladan. Kolbova podjela tipova učenja

može se nastavniku prikazati grafički, što svodi pridruživanje dokumenta nekom tipu na razinu jednog pritiska na miš.

Kolbova podjela je jedan od mogućih izbora. Interesantno bi bilo pogledati postojeće teorije o stilovima učenja kako bi se možda pronašle bolje podjele. One bi se zatim mogle implementirati i testirati.

4. Algoritmi nad gradivom

U prethodna dva poglavlja definirani su model domene i model studenta. Preostaje konstruirati algoritme temeljem kojih će studenti biti poučavani zadanom gradivu. Algoritam je po definiciji „recept“ [6] kako nešto napraviti. Algoritmi koji mijenjaju svoje ponašanje ovisno o dostupnim resursima se nazivaju *adaptivnim algoritmima* [7].

Očekivanja od sustava mogu se vidjeti kroz obrasce uporabe (engl. *use case*). U nastavku su navedena tri obrasca uporabe. Prvi obrazac govori o ponašanju sustava prilikom studentovog ulaska u sustav i pregledavanja gradiva nekog predmeta.

Obrazac uporabe 1 – početak rada sa sustavom, pregled predmeta

1. Student se prijavi na sustav.
2. Sustav prikazuje studentu popis predmeta.
3. Student odabire neki predmet.
4. Po odabiru predmeta student bira opciju učenja.
5. Studentu se prikaže popis svih tema i koncepata.

Student može odabrati za učenje ili temu ili koncept. Oba odabira povlače različite obrasce uporabe. Obrazac uporabe 2 opisuje studentov odabir nekog koncepta za učenje.

Obrazac uporabe 2 – studentov odabir učenja koncepta

1. Student odabire neki koncept koji želi naučiti.
2. Sustav na osnovu svojih podataka odlučuje ima li student dovoljno znanja da može učiti odabrani koncept (dovoljno znanja = svi koncepti/teme koji su preduvjeti imaju zabilježeno da ih student zna u onolikoj mjeri koliko zahtjeva veza od koncepta/teme preduvjeta prema konceptu koji je student odabrao).
    a. Ako nema dovoljno znanja, sustav ga pita želi li naučiti ono što ne zna ili da se to zanemari.
    b. Ako ima dovoljno znanja, sustav mu prikazuje dokumente koncepta ovisno o studentovom tipu i raspoloživim dokumentima.
3. Student čita dokumente i odgovora na zadatke vezane za odabrani koncept.
4. Sustav bilježi rezultat testiranja za dotični koncept.

Druga opcija studenta je odabir neke teme za učenje što je opisano u obrascu uporabe 3.

Obrazac uporabe 3 – studentov odabir učenja teme

1. Student odabire temu koju želi naučiti.
2. Sustav na osnovu svojih podataka odlučuje ima li student dovoljno znanja da može učiti odabranu temu (dovoljno znanja = svi koncepti/teme koji su preduvjeti imaju zabilježeno da ih student zna u onolikoj mjeri koliko zahtjeva veza od koncepta/teme preduvjeta prema temi koju je student odabrao).
    a. Ako nema dovoljno znanja, sustav ga pita da li želi naučiti ono što ne zna ili da se to zanemari.
    b. Ako ima dovoljno znanja, sustav stvara *put učenja* za tu temu (obuhvaća sve podčvorove unutar te teme) te mu prikazuje dokumente prvog čvora u tom generiranom putu.
3. Student čita dokumente i odgovara na zadatke vezane za dotični čvor.
4. Sustav bilježi rezultat testiranja za dotični čvor i prelazi na sljedeći čvor u *putu učenja*.

Iz prethodnih obrazaca uporabe vidi se da algoritam treba provjeravati zna li student sve potrebne preduvjete. Ukoliko ne zna, sustav će ga uputiti da nauči prvo ono što ne zna (ali mu pritom omogućiti i suprotno). Adaptivni dio algoritma je prikaz dokumenata s obzirom na tip studenta.

Najopćenitije gledano treba postojati funkcija koja će prikazati neki čvor iz grafa – segment gradiva. Sljedeći pseudokod opisuje tu funkciju.

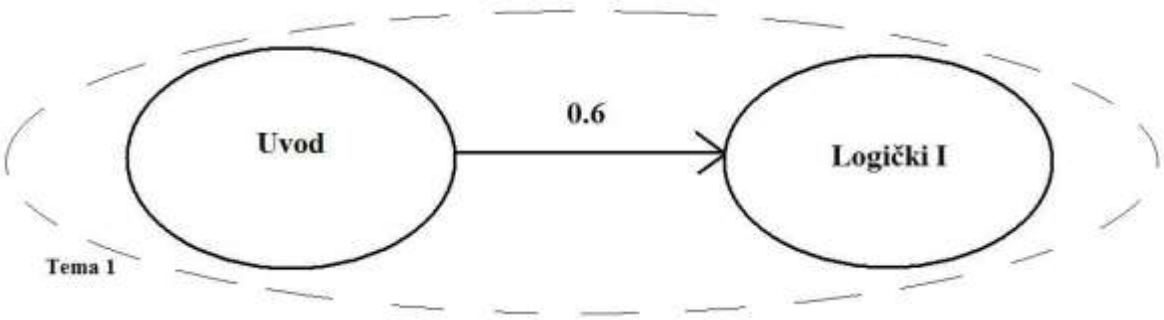
Funkcija 1 – prikazuje odabrani čvor

```
funkcija prikaziCvor(Cvor cvor) {
    radi {
        za svaki preduvjet od cvor {
            ako znanje studenta o preduvjet-u nije dovoljno
                dodaj preduvjet u neispunjeniPreduvjeti;
        }
```

```
            preskoci = false;
            ako neispunjeniPreduvjeti nije prazno {
                  prikazi neispunjeniPreduvjeti;
                  ako student kliknuo na „Preskoči“ {
                        preskoci = true;
                  }
            } inace {
                  preskoci = true;
            }

            ako preskoci == true {
                  PutUcenja put = genPutUcenja(cvor); /*ako je cvor koncept, unutar puta
      ce se samo on nalaziti, ako je tema, put ce obuhvaćati sve podčvorove teme */
                  prikaziPutUcenja(put);
                  break;
            } inace {
                  /* uvijek ide u dubinu 1 dokle god student to želi, moguća izmjena može
      biti sustav sam odabere neki preduvjet, npr. najlakši */
                  prikaziCvor(odabrani cvor iz liste neispunjenih preduvjeta);
                  continue;
            }
      } dok true;
}
```

Funkcija prvo provjerava jesu li svi preduvjeti ispunjeni. Kao što je bilo rečeno u poglavlju 2.2.1. Segment gradiva, uz preduvjet je vezana mjera koja označava koliko je preduvjetni segment bitan. Ukoliko student zna preduvjetni segment u većoj mjeri od one koja je zadana u popisu preduvjeta čvora tada sustav smatra da student zna dovoljno o preduvjetnom segmentu. Ukoliko preduvjeti nisu ispunjeni, sustav upućuje studenta na učenje čvorova o kojima ne zna dovoljno što student može ignorirati. Ukoliko preduvjeti jesu ispunjeni prikazuje mu se generirani *put učenja* nad tim čvorom. Put učenja predstavlja niz čvorova iz grafa modela domene kojim student treba proći da bi naučio odabrani čvor.

Pogledajmo na konkretnom primjeru funkciju *prikaziCvor()*. Graf modela domene možemo vidjeti na slici 8. Koncept *Uvod* predstavlja uvodnu cjelinu gradiva u predmetu Digitalna Logika dok koncept *Logički I* predstavlja cjelinu koja objašnjava sklop logičko I.



Slika 8 – primjer grafa modela domene

Iznad preduvjetne veze naznačena je težina preduvjeta. Ona nam govori da student mora znati 60% koncepta *Uvod* da bi mogao naučiti koncept *Logički I*. Pretpostavimo da student odabere iz popisa segmenata koncept *Uvod*. Kako taj koncept nema preduvjeta rezultat bi bio dokument koji opisuje koncept *Uvod*. Pretpostavimo da student zna koncept *Uvod* ocjenom od 0.5. Ako nakon toga iz popisa svih segmenata odabere koncept *Logičko I*, sustav će mu prikazati da nema dovoljno znanja o konceptima koji su preduvjeti odabranom konceptu, u ovom slučaju radi se o konceptu *Uvod*. Ukoliko student odabere segment *Tema 1*, kako taj segment nema preduvjeta, sustav će generirati put učenja za tu temu. Generiranje puta učenja objašnjeno je u nastavku.

Pogledajmo funkciju *genPutUcenja()*. Ulazni parametar može biti koncept ili tema. Ukoliko je koncept u pitanju onda je stvar trivijalna – generirani put učenja se sastoji samo od tog koncepta. No ukoliko je u pitanju tema (vidi: obrazac uporabe 3) onda algoritam treba generirati put po kojem će student proći kroz sve koncepte i teme unutar teme koju je odabrao za učenje. Početnu točku učenja definirat ćemo kao one koncepte koji nemaju nikakvih preduvjeta unutar te teme – ne trebamo ništa drugo naučiti prije nego što krenemo učiti te koncepte. Takve koncepte nazivamo *početni koncepti*. Pseudo kod je prikazan u nastavku:

Funkcija 2 – generira put učenja za odabrani segment

```
funkcija genPutUcenja(Cvor cvor) {
        PutUcenja put = new PutUcenja();
```

```
        ako cvor je Koncept {
                dodaj cvor u put;
        } inace {
                Tema tema = cvor;

                /* Radi unutar granica teme tema, za sve provjere preduvjeta ne gleda izvan
tema,  povezanost se gleda samo po konceptima. */
                komponentePovezanosti = pronadiKomponente(tema);
                sortirajKomponentePovezanostiPoTezini(komponentePovezanosti); /* s obzirom
na izračunatu težinu svake*/
                za svaku komponenta unutar komponentePovezanosti {
                        dok postoji cvor koji je unutar komponenta a nije unutar put {
                                Cvor cvor = pronadiSljedeciCvor(komponenta, put);

                                dodaj cvor u put;
                        }
                }
        }
        vrati put;
}
```

Ukoliko je odabrana tema, algoritam prvo analizira unutrašnju strukturu teme. Generiraju se sve komponente povezanosti unutar granica te teme. U poglavlju 2.1. Model Domene pokazano je da, bez obzira što se nalazilo unutar teme, cijelu strukturu možemo reducirati na veze samo između koncepata. Potrebno je naglasiti da se prilikom reduciranja ne gubi pripadnost segmenta temi. Za određivanje komponenti povezanosti može se iskoristiti neki postojeći algoritam za pronalaženje komponente povezanosti u jednostavnom usmjerenom grafu kao što je pretraživanje u širinu (engl. *breadth-first search*) [8] ili pretraživanje u dubinu (engl. *depth-first search*) [9].

Sljedeći korak algoritma je sortiranje komponenti povezanosti s obzirom na njihovu težinu. Težina komponente povezanosti je jednaka težini najtežeg koncepta kojeg

sadrži. Na kraju algoritam za svaku komponentu povezanosti gradi put učenja čvor po čvor. Funkcija koja pronalazi sljedeći čvor u putu učenja opisana je u nastavku.

Prilikom odabira sljedećeg čvora u obzir se uzima pripadnost unutar podtema (nalazi li se hijerarhijski koncept dublje u grafu ili ne), težina koncepta te udaljenost koncepata unutar komponente povezanosti. Najveću prednost imaju oni koncepti koji već imaju ispunjene sve preduvjete kako bi student slijedno učio o stvarima koje su, s obzirom na povezanost gradiva, najbliže. Zadnji se biraju početni koncepti jer oni predstavljaju početak neke druge grane u komponenti povezanosti. U slučaju nedeterminizma odabire se prvo onaj koncept koji je na najvišoj hijerarhijskoj razini. To znači da se studentu prvo nude za učenje oni koncepti koji su sadržani u manje različitih tema. Ukoliko se niti ovdje ne može jednoznačno odabrati neki koncept algoritam odabire onaj koji je najlakši.

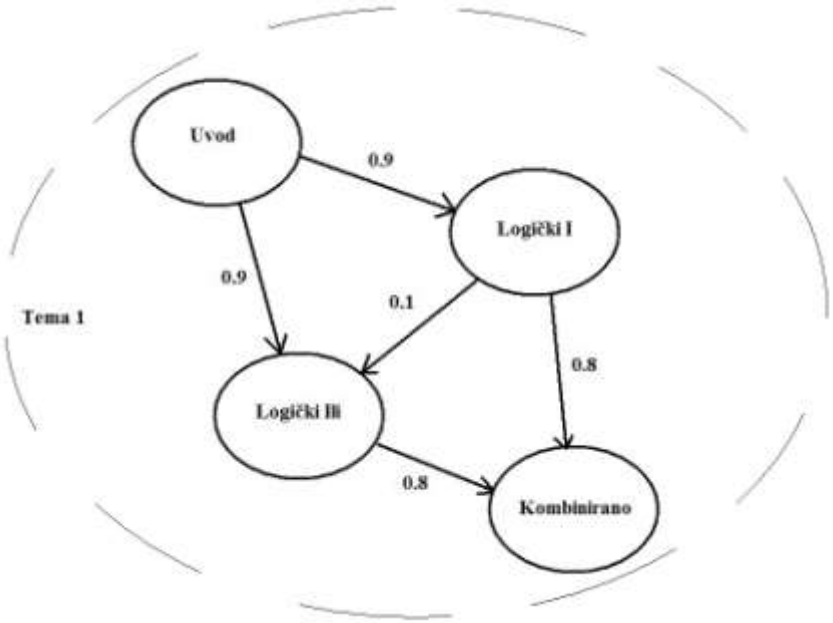
Funkcija 3 – pronalazi unutar komponente sljedeći čvor u putu učenja

```
funkcija pronadiSljedeciCvor (komponenta, put) {
    ako unutar put se nalaze svi podCvorovi od neke podTema a
            podTema se ne nalazi unutar put {
        vrati podTema;
    }
    za svaki koncept unutar komponenta {
        ako koncept unutar put {
            continue;
        }
        ako svaki preduvjet od koncept unutar put { // dodaje i početne koncepte
            dodaj koncept u sljedMoguciKoncepti;
        }
    }
    /* daje prednost konceptima koji nisu početni koncept kako bi što duže prolazilo po istoj
    grani neke komponente povezanosti */
    ako svaki postoji koncept unutar sljedMoguciKoncepti koji nije pocetni koncept {
        ako postoji samo jedan nepocetni koncept ciji su svi preduvjeti zadovoljeni {
            vrati taj koncept;
        } inace {
            ako ima samo jedan nepocetni koncept na najvisoj razini {
```

```
                                    vrati taj koncept;
                            }
                            vrati najlaksi nepocetni koncept;
                    }
            } inace {
                    ako ima samo jedan koncept na najvisoj razini {
                            vrati koncept s najvise razine;
                    }
                    vrati najlaksi koncept;
            }
}
```

Pravilo prednosti koncepata na višoj razini opravdano je prirodom grafa modela domene. Pretpostavimo da u nekoj temi postoji koncept A i podtema X u kojoj su sadržani neki drugi koncepti. Po prethodnom pravilu koncept A koji se nalazi na prvoj razini bit će odabran prije nekog drugog koncepta B koji se nalazi unutar podteme B (pod pretpostavkom da su preduvjeti za oba koncepta ispunjeni). Pretpostavimo da je takav odabir pogrešan, odnosno, da je taj drugi koncept B unutar podteme X pedagoški adekvatniji za učenje prije koncepta A. No, to bi značilo da je koncept B preduvjet koncepta A no to je suprotno pretpostavci da su preduvjeti od oba koncepta ispunjeni. Odabir koncepta A na višoj razini može se opravdati i činjenicom da je tema po samoj definiciji složenija od koncepta. Stoga, ako bi se odabrao koncept B ušlo bi se unutar složenijeg segmenta gradiva kojeg predstavlja podtema X prije nego što su pregledani jednostavniji segmenti. No, to je u suprotnosti s postavljenom pretpostavkom o tijeku ljudskog učenja u poglavlju 2 *Model domene* (od jednostavnijeg prema složenijem).

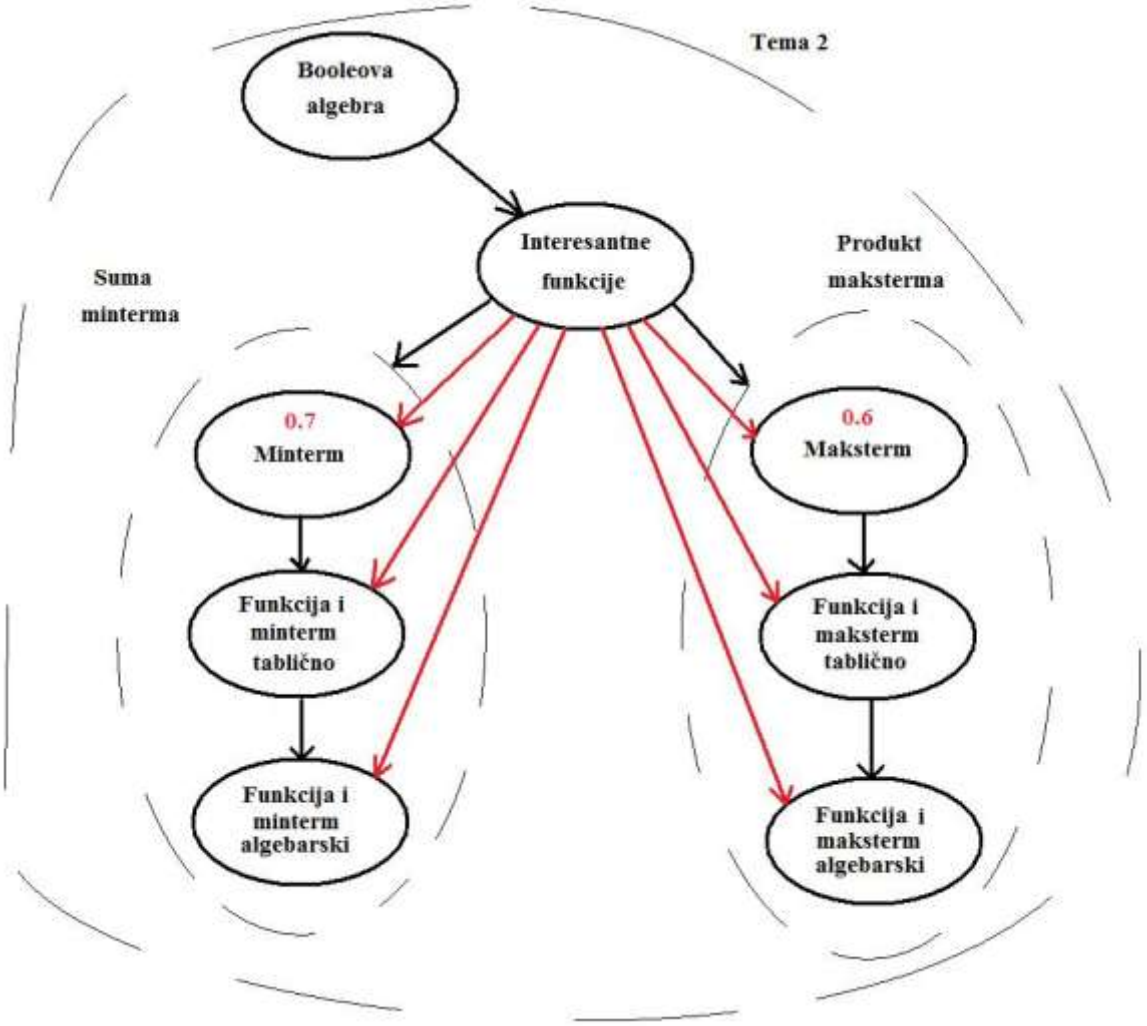
Pogledajmo na sljedećem primjeru generiranje puta učenja. Proširimo prvu temu Digitalne logike s dodatnim konceptima.



Slika 9 – proširenI primjer grafa modela domene prve teme na predmetu Digitalna logika

Ako student odabere segment *Tema 1* koji predstavlja temu, za nju će se generirati put učenja. Kako je algoritam isti za svaku komponentu povezanosti korišten je primjer sa samo jednom komponentom. Put učenja je na početku prazan. U skupu mogućih koncepata nalazi se samo koncept *Uvod* jer je to jedini koncept koji nema niti jedan preduvjet. Kako je to jedini koncept u skupu mogućih koncepata, iteracija je završena te je on dodan u put učenja. U sljedećoj iteraciji u skupu mogućih nalazi se jedino koncept *Logički I* jer jedino on ima sve svoje preduvjete već u putu učenja. Iteracije se nastavljaju dok na kraju ne dobijemo sljedeći niz segmenata: *Uvod*, *Logički I*, *Logički Ili*, *Kombinirano* i *Tema 1*.

Pogledajmo složeniji primjer prikazan na slici 10. Ovo je druga tema na predmetu Digitalna logika i pretpostavimo da je student odabrao nju za učenje.



Slika 10 – primjer grafa modela domene druge teme na predmetu Digitalna logika

Kao što je prikazano, ova tema ima dvije podteme *Suma minterma* i *Produkt maksterma*. Algoritam konstruira put učenja tako da prvo pronalazi komponente povezanosti. I u ovom slučaju postoji samo jedna komponenta povezanosti. Crvenom bojom su naznačene preduvjetne veze koje ne postoje u početnom grafu modela domene već su dodane prilikom traženja komponenti povezanosti (po pravilima opisanim u poglavlju 2.1. Struktura grafa modela domene, u ovom slučaju preslikavanje iz dvije veze koncept-tema gdje je koncept *Interesantne funkcije* a teme su *Suma minterma* te *Produkt maksterma*). Prva dva koncepta u putu učenja, *Booleova algebra* i *Interesantne funkcije*, su dodana na isti način kao u prethodnom primjeru. U sljedećoj iteraciji postoje dva moguća koncepta za put učenja: *Minterm* i *Maksterm*. Kako oba imaju sve preduvjete zadovoljene, oba su udaljena od zadnjeg dodanog koncepta jednako i oba se nalaze na istoj razini u hijerarhiji, izbor između njih se svodi na odabir lakšega. Na slici je crvenom

bojom naznačena težina za svaki od ova dva koncepta. Kako je definirano da je koncept *Minterm* lakši od koncepta *Maksterm* on se dodaje u put učenja. U sljedećoj iteraciji opet postoje dva moguća koncepta: *Funkcija i minterm tablično* i *Maksterm*. Ovo je slučaj kada presuđuje udaljenost od zadnjeg dodanog koncepta u put učenja. Kako je to bio koncept *Minterm* sljedeći odabrani koncept će biti koncept *Funkcija i minterm tablično* koji je samo jedan korak udaljen od njega za razliku od koncepta *Maksterm* koji je udaljen dva koraka. Konačni rezultat će biti sljedeći put učenja: *Booleova algebra*, *Interesantne funkcije*, *Minterm*, *Funkcija i minterm tablično*, *Funkcija i minterm algebarski*, *Suma minterma*, *Maksterm*, *Funkcija i maksterm tablično*, *Funkcija i maksterm algebarski*, *Produkt maksterma*.

Nakon generiranja puta učenja, algoritam kreće od prvog čvora te prikazuje njegove dokumente. Nakon što ih student pročita ispituje se njegovo znanje o tom čvoru. Prelazi se na sljedeći čvor te se postupak ponavlja. Ukoliko u sustavu postoji podatak da student već posjeduje potrebno znanje o čvoru on se preskače.

Funkcija 4 – prikazuje put učenja studentu

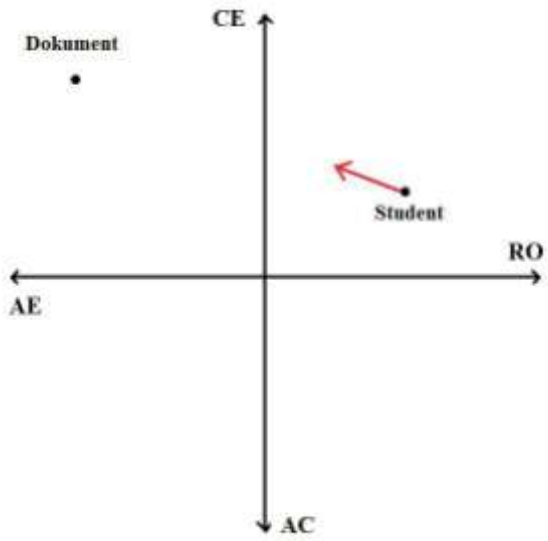
```
funkcija prikaziPutUcenja(PutUcenja put){
        za svaki cvor unutar put {
                ako student zna cvor {
                        continue;
                }

                /* student može u bilo kojem trenutku odabrati opciju „Preskoci“ i otici na sljedeci čvor */
                prikaziDokument(cvor);/*ovisno o podacima o studentu prikazuju se odgovarajući dokumenti (npr, više teorijski ili više praktični) */
                ispitajStudentaOCvoru(cvor);    /* zapisuje rezultat testa kao znanje studenta o čvoru, osvježuje podatke o studentu ako je potrebno*/
        }
}
```

Funkcija *prikaziDokument()* odabire onaj dokument definiran za dotični čvor koji najviše odgovara tipu učenja studenta. Ukoliko čvor predstavlja koncept onda se traže samo oni dokumenti koji pokrivaju jedino taj koncept. Ukoliko je čvor tema onda se traže svi dokumenti koji pokrivaju više podčvorova te teme.

Određivanje da li dokument odgovara nekom tipu učenja je ostvareno usporedbom definiranih koordinata dokumenta s koordinatama studenta u koordinatnom sustavu Kolbovih tipova učenja. Prikazuje se onaj dokument čije su koordinate najbliže trenutnim koordinatama studenta.

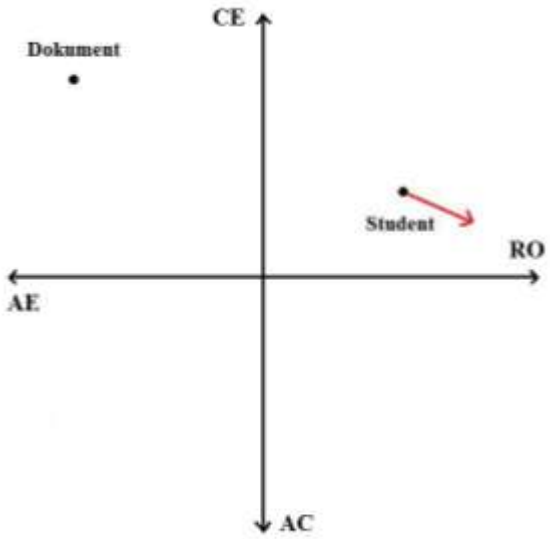
Nakon što student završi proučavanje prikazanog gradiva pokreće se ispitivanje tog gradiva. To obavlja funkcija *ispitajStudentaOCvoru()*. Ukoliko je student uspješno naučio gradivo njegov tip se pomiče u smjeru tipa učenje kojeg dokument najviše zastupa. Ukoliko nije uspješno naučio gradivo studentov tip se pomiče u smjeru onog tipa učenja kojeg dokument najmanje zastupa. Na taj način tip studenta se približava njegovom stvarnom tipu učenja te će mu u budućnosti prije biti prikazani dokumenti koji mu više odgovaraju.

Na slici 10 prikazano je što se događa sa studentovim modelom ukoliko uspješno nauči neki segment.



Slika 11 – student je uspješno naučio segment pomoću danog dokumenta

Model studenta se pomiče prema modelu dokumenta. Ukoliko student nije uspio naučiti segment pomoću prikazanog dokumenta to znači da mu takav tip dokumenta ne odgovara pa se model pomiče u suprotnom smjeru kao što je prikazano na slici 11.



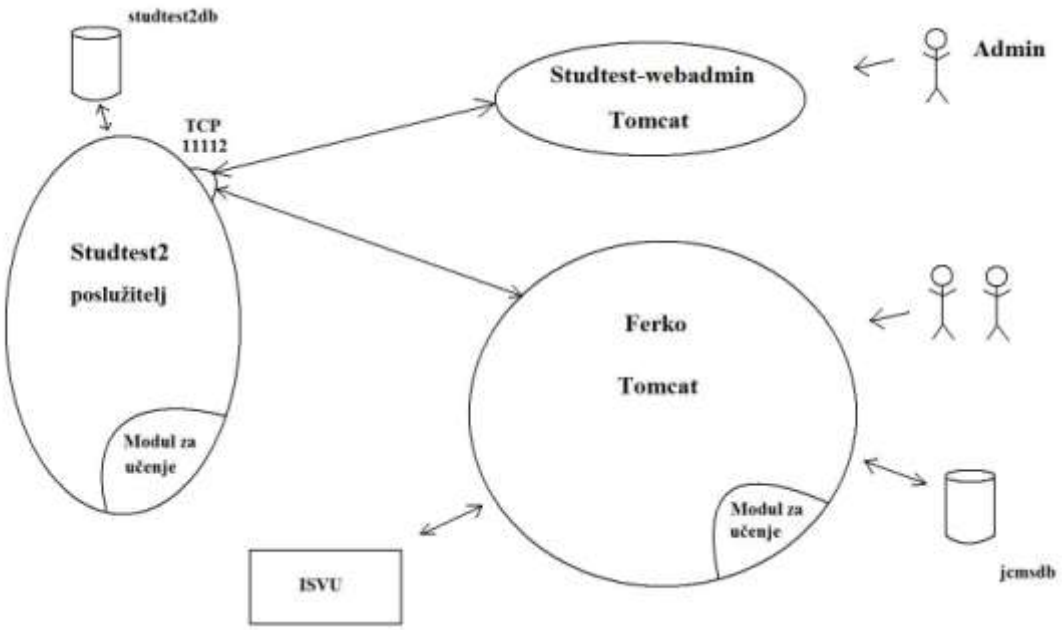
Slika 12 – student nije uspio naučiti segment pomoću danog dokumenta

Koliko se model studenta pomiče po koordinatnom sustavu je varijabla koja se može mijenjati. Treba naglasiti da pomak studentovog modela u slučaju neuspjeha treba biti manji nego u slučaju uspjeha. To proizlazi iz činjenice da do neuspjeha može doći zbog mnogo razloga koji i ne moraju imati veze s prikazanim dokumentom no do uspjeha može doći samo ako je student shvatio segment pomoću dokumenta.

5. Implementacija sustava

Ovo poglavlje govori o implementaciji sustava za prilagodljivo poučavanje. Opisat će se sve korištene tehnologije, korišteni postojeći sustavi te detalji implementacije opisanih algoritama.

Modul za poučavanje (nazvan *TeachMe*) koji sadrži sve spomenute algoritme ugrađen je u postojeće sustave *Ferko* i *StudTest2*. Odnos između ova dva sustava može se vidjeti na slici 13.



Slika 13 – StudTest i Ferko

Osim sustava StudTest2 i Ferko postoji i web aplikacija *Studtest-webadmin*. Ona služi za administraciju sustava StudTest2. Dijelovi modula za poučavanje se nalaze i u sustavu StudTest2 i u sustavu Ferko. Točnije, Ferko služi samo kao sučelje prema korisnicima a svi podaci i algoritmi nalaze se u sustavu StudTest2. Na taj način ostvaruje se veća modularnost odnosno, može se koristiti neki drugi sustav kao sučelje. Također, jedan od razloga zašto su svi algoritmi smješteni u sustav StudTest2 je taj što se u bazi podataka *studtest2db* nalaze svi podaci o različitim zadatcima vezanim za neko gradivo. Ti zadatci se definiraju kroz aplikaciju Studtest-webadmin aplikaciju i njihovi opisnici se

pohranjuju u bazi podataka StudTest2. Kako postoji modul za testiranje koji koristi istu strukturu znanja odlučeno je da glavna baza podataka bude ona u sustavu StudTest2 a ne ona u sustavu Ferko radi smanjenja opterećenja komunikacije između ova dva sustava te centralizacije podataka i programskog koda.

5.1. Sustav StudTest2

Sustav StudTest2 danas predstavlja jedan od temeljnih infrastrukturnih sustava koji omogućava provjeru znanja studenata kroz niz različitih tipova zadataka, te je svoju punu snagu pokazao dolaskom Bolonje na FER, gdje je korišten na kolegiju Digitalna logika za ispitivanje studenata na laboratorijskim vježbama i domaćim zadaćama. Nakon početnih podešavanja, sustav se pokazao vrlo skalabilnim, te je bez problema podnosio opterećenje od 120 paralelnih ispita na laboratorijskim vježbama, odnosno preko 400 pisanja domaćih zadaća [10].

Neke od mogućnosti koje sustav StudTest2 nudi su definiranje različitih vrsta provjera znanja – definiranje vremenskih ograničenja, broj mogućih otvaranja, pristupan unos rješenja, itd. Također, moguće je definirati politike prolaska na ispitu. Važno svojstvo za modul poučavanja je mogućnost definiranja različitih tipova zadataka kao što su standardne ABC pitalice s jednim ili više točnih odgovora, pitalice tipa unos linije teksta, pitalice tipa unos više linija teksta te korištenja *appleta* za prikaz zadataka (bogato grafičko sučelje). Dinamički zadatci najvažnije su svojstvo sustava StudTest2 zbog kojeg se isti čini idealnim za modul poučavanja jer omogućuje fleksibilno upravljanje zadatcima koji će služiti za provjeru studentovog znanja.

Sustav StudTest2 se koristi na predmetima Umjetna Inteligencija i Računalna Grafika na Fakultete elektrotehnike i računarstva u obliku zadaća i testova. Pokazalo se da korištenje dinamičkih zadataka osim brze povratne informacije o uspjehu, smanjuje prepisivanje te pomaže studentima u provjeravanju vlastitog znanja prije ispita. [11]

U nastavku se nalaze detalji implementacije na strani sustava StudTest2 koji uključuju bazu podataka, algoritme i komunikaciju prema Ferku.

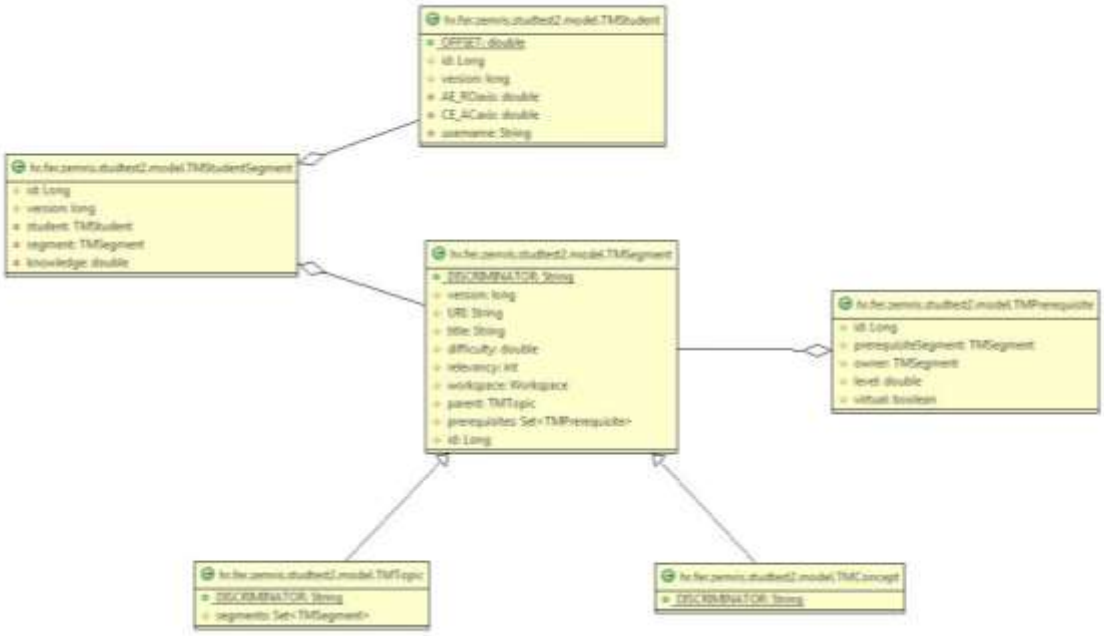
5.1.1. Biblioteka Hibernate i pripadni model podataka

Objektno orijentirani programski kod u sinergiji s relacijskom bazom zna biti vrlo složen i kompliciran, pogotovo u današnje vrijeme kada aplikacije imaju veliki broj linija koda. Biblioteka *Hibernate* je implementacija objektno-relacijskog preslikavanja. On za programera preslikava podatke iz objektne reprezentacije unutar programskog koda u relacijsku reprezentaciju unutar npr. SQL baze podataka [12]. Radi se o programskom alatu otvorenog koda. Osim preslikavanja Hibernate omogućuje podatkovne upite i usluge dohvata informacija. Umjesto da programer sam piše SQL upite i dobivene rezultate pretvara u objekte, Hibernate to radi za njega. Kako su sustavi Studtest2 i Ferko veliki sustavi, Hibernate je dobar izbor radi smanjivanja složenosti koda. Već postojeći podatkovni model sustava StudTest2 napravljen je kroz Hibernate.

Hibernate se brine za transparentnu perzistenciju tzv. *POJO* objekata (*engl. Plain Old Java Objects*). Kolekcije objekata tipično su spremljene u kolekcije poput *Set* ili *List*. Objekti između kojih postoje određene relacije mogu se konfigurirati da kaskadno prenose operacije jedni na druge.

Model podataka

Model se sastoji od nekoliko grupa razreda. Svi razredi imaju prefiks 'TM' u svojem imenu kako bi označavali da se radi o dijelovima modula za poučavanje (TM je kratica za TeachMe). Prvu grupu među njima čine razredi koji predstavljaju čvorove u grafu modela domene. Tako postoje razredi *TMSegment*, *TMTopic* i *TMConcept*. Drugu skupinu čine razredi *TMStudent* i *TMStudentSegment* koji predstavlju model studenta i njegovo trenutno znanje. Treću skupinu čini razred *TMDocument* koji sadrži sve podatke o dokumentima, kojem modelu studenta pripada i koje segmente pokriva. Odnos između ovih razreda možete vidjeti kroz dijagram razreda prikazanim na slici 14.



Slika 14 – Dijagram razreda

Svaki razred sadrži varijable *id* i *version*. *Id* je jedinstveni identifikator svakog objekta (odnosno *n-torke* u relacijskoj bazi). *Version* služi kao oznaka verzije pojedinog objekta u slučaju istovremenog pristupa bazi podataka od strane više korisnika a u svrhu sprječavanja istovremenih izmjena.

Uz navedene razrede postoje još dva razreda. Oba su vezna uz same algoritme stvaranja puta učenja u grafu. *TMLearningPath* je razred koji sadrži put učenja (niz segmenata). *TMConnectedComponent* je razred koji služi za stvaranje komponenti povezanosti u grafu kada se sve veze preslikaju u koncept – koncept veze.

Unutar Hibernate-a koristi se naziv entitet (*engl. entity*) za sve objekte koji mogu samostalno postojati i kojima se može manipulirati neovisno o drugim objektima. S druge strane, svi oni objekti koji su dio nekih drugih objekata nazivaju se komponentama (*engl. component*).

Anotacije

Preslikavanje je ostvareno uporabom Java anotacija. Anotacije u programskom kodu su metapodatci napisani u posebnoj sintaksnoj formi i predstavljaju dio izvornog koda. Anotacije postoje za razrede, metode, varijable, parametre i pakete.

Pomoću anotacija definiraju se također i odnosi između različitih razreda. Postoje JPA i Hibernate anotacije. JPA (engl. *Java Persistance* API) je razvojni okvir koji omogućava objektno relacijsko preslikavanje [13]. Unutar biblioteke Hibernate postoje implementacije za JPA anotacije ali također i dodatne posebne Hibernate anotacije. Želimo koristiti samo JPA anotacije kako bi kod ostao što više modularan. Na taj način moguće je kasnije zamijeniti Hibernate s nekom drugom tehnologijom.

Postoji nekoliko bitnih anotacija koje su korištene u svim razredima u modelu podataka. Anotacija *@Entity* koja deklarira razred kao entitet (perzistentni POJO razred). *@Id* deklarira identifikator entiteta. *@Table* omogućuje definiranje tablice u koju će se dotični razred preslikati. Ako nema ove anotacije koristi se ime samog razreda kao ime tablice.

Isječak koda 2 – pojednostavljeni prikaz razreda TMStudent

```
@NamedQueries({
	@NamedQuery(name="TMStudent.findByUsername", query="select stud
from TMStudent as stud where stud.username=:studUsername")
})

@Entity
@Table(name="tm_students")
public class TMStudent {

	protected Long id;
	protected long version;

	private double AE_ROaxis;
	private double CE_ACaxis;

	private String username;

	@Id @GeneratedValue
	public Long getId() {
		return id;
	}

	@Version
	public long getVersion() {
		return version;
	}

	@Column(nullable=false)
	public double getAE_ROaxis() {
		return AE_ROaxis;
	}

	@Column(nullable=false)
	public double getCE_ACaxis() {
		return CE_ACaxis;
```

```
	}

	@Column(nullable=false, unique=true, length=250)
	public String getUsername() {
		return username;
	}

}
```

Anotacije nam također omogućavaju definiranje gotovih upita prema bazi podataka. Oni se definiraju kroz anotaciju *@NamedQuery*. U sustavu StudTest2 postoje razredi koji služe za pozivanje ovakvih upita – *PersistanceUtil* te *PersistanceServices*.

Isječak koda 3 – primjer metode koja poziva gotove upite

```
public static TMStudent getStudentByUsername(String username){
	List<TMStudent> list =
(List<TMStudent>)PersistanceUtil.getEntityManager().createNamedQuery("TMS
tudent.findByUsername").setParameter("studUsername",
username).getResultList();
	if(list==null || list.isEmpty()) return null;
	return list.get(0);
}
```

Relacijske baze podataka ne podržavaju nasljeđivanje. Postavlja se pitanje kako preslikati hijerarhijski odnos između razreda. Postoje tri rješenja:

- tablica po razredu,
- tablica po hijerarhiji te
- tablica po podrazredima.

Odabrana je strategija tablice po hijerarhiji. Ovakav način preslikavanja definira jednu tablicu koja sadrži sve objekte u hijerarhiji bez obzira na njihov tip. Tipovi se razlikuju pomoću dodatnog diskriminantnog stupca (svaki tip ima svoju definiranu vrijednost tog stupca). Odabran je baš ovaj način zato što će se prilikom rada sustava segmenti vrlo često dohvaćati iz baze. U slučaju druge dvije strategije dolazilo bi do spajanja različitih tablica prilikom svakog dohvata što bi usporilo sustav.

Isječak koda 4 – primjer preslikavanja tablicom po hijerarhiji

```
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@org.hibernate.annotations.Entity(dynamicInsert = true, dynamicUpdate =
true)
@DiscriminatorColumn(name = "discriminator", discriminatorType =
DiscriminatorType.CHAR)
@DiscriminatorValue(TMSegment.DISCRIMINATOR)
public abstract class TMSegment {...}
```

Vrlo važne anotacije su *@ManyToOne* i *@OneToMany* koje služe za definiranje odnosa između razreda. Kao što im sama imena kažu definira se N:1 odnosno 1:N odnosi. Dodatni argument tih anotacija je *fetch=FetchType.Lazy* čime se definira tzv. *lazy* dohvat objekata iz baze. Na taj način neće se dohvatiti cijeli graf znanja iz baze ako se koristi samo nekoliko segmenata u tom trenutku. Ovo uvelike poboljšava performance sustava. Uz te anotacije koriste se i anotacija *@JoinColumn* koja definira preko kojeg stupca se obavlja spajanje između tablica.

Postoji još jedna anotacija koja je često korištena a to je *@Transient*. Ona služi za označavanje varijabli koje ne trebaju biti preslikane u bazu podataka a potrebne su za algoritme. Primjer takve situacije je oznaka je li preduvjetna veza između segmenata virtualna ili nije (prilikom preslikavanja svih tipova veza u koncept – koncept veze).

5.1.2. Komunikacija sa sustavom StudTest2

Na slici 8 može se vidjeti da postoji jasno definiran način komunikacije sa sustavom StudTest2. To se ostvaruje pomoću „naredbi“ (engl. *Command*) koje su zapravo java razredi. Za svaku naredbu se definira broj i tip ulaznih argumenata i broj i tip izlaznih argumenata što onda korisnik te naredbe mora poštivati. Svaka naredba koristi postojeću funkcionalnost za serijalizaciju osnovnih tipova u binarni zapis. Primjer jedne takve naredbe je naredba za dohvat podataka o segmentima (skraćena radi ilustracije).

Isječak koda 5 – primjer naredbe za komunikaciju sa sustavom StudTest2

```
/**
 * This is <b>segment data</b> command.
 * ...
 * <li>if status=0<ol><li>one string containing error message, and that
is all</li></ol>
 * ...
 */
public class Command043 implements CommandProcessor,
```

```
@Override
public Command process(Command command) throws IOException{
      ...

}
```

U *javadoc* komentaru naredbe potrebno je definirati točno što naredba očekuje kao ulaz i kakav će biti odgovor. Unutar same naredbe izračunava se rezultat, u ovom slučaju dohvat traženih segmenata iz baze podataka.

Kako je komunikacija između sustava StudTest2 i klijenta (u ovom slučaju sustava Ferko) radi performansi svedena na binarizaciju bilo je potrebno smisliti način kako prenositi podatke s jednog u drugi sustav. Stoga je dogovoren protokol prilikom prijenosa segmenata. Prenose se samo osnovni podatci koji uključuju identifikator, verziju, uri, naslov, težinu segmenta i važnost segmenta. Popisi roditelja, djece i preduvjeta se u ovom slučaju ne prenose već postoje posebne naredbe koje upravo njih dohvaćaju.

5.2. Sustav Ferko

Ferko je sustav za upravljanje kolegijima usmjeren prema studentima i djelatnicima Fakulteta kako bi im olakšao svakodnevne radnje vezane za Fakultet, poboljšao međusobnu komunikaciju i unaprijedio iskustvo učenja, te bio pristupačan svima. Osnovne mogućnosti za studente su jednostavan pristup informacijama o fakultetskim obavezama i aktivnostima, pristupačne i kvalitetne provjere znanja (e-provjere). Također omogućuje detaljan uvid u rezultate provjera, jednostavnu i efikasnu komunikaciju s nastavnim osobljem pa i međusobno između studenata te jednostavan pristup nastavnom materijalu. Osnovne mogućnosti za nastavno osoblje su lakše stvaranje studentskih rasporeda i prosljeđivanje informacija i obavijesti studentima, ispitivanje studenata pomoću provjera znanja (pismenih i e-provjera) u vidu domaćih zadaća, bliceva i izlaznih testova te komunikacija sa studentima radi povratnih informacija [14]. Zbog svega navedenog sustav Ferko se čini idealnim mjestom za implementaciju modula za prilagodljivo poučavanje.

5.2.1. Razvojni okvir Struts2

Apache Struts2 je besplatni razvojni okvir (engl. *framework*) otvorenog koda za izradu web-aplikacija. Za razliku od konvencionalnih internet stranica gdje je veliki dio sadržaja statičan, web-aplikacije sadrže dinamički generirane odgovore. Web-aplikacija može interaktivno komunicirati s bazom podataka i s dijelovima koji su odgovorni za poslovnu logiku prilikom stvaranja odgovora. Programski kod, iako u većim aplikacijama može biti vrlo složen, kroz razvojni okvir Struts2 se razdvaja i na taj način postaje mnogo čitljiviji. To se ostvaruje kroz *MVC arhitekturu* (*Model-View-Controller*). *Model* predstavlja poslovnu logiku odnosno bazu podataka, *View* predstavlja dizajn web stranica dok *Controller* predstavlja navigacijski kod. Postoje tri glavne komponente u razvojnom okviru Struts2: kod koji će biti zadužen za posluživanje zahtjeva preslikanog na standardni URI (engl. *uniform resource identifier*), kod koji će biti zadužen za odgovor odnosno, prijenos kontrole nekom drugom resursu koji dovršava odgovor te biblioteka oznaka (engl. *tag*) koja pomaže kod izgradnje web-aplikacije.

Akcije

Svaki zahtjev od klijenta pokreće neku akciju na poslužitelju. Te akcije su u obliku java razreda a koji će razred biti pozvan, preslikava se u posebnom *xml* dokumentu (jcms.xml). Preslikavanje se može vidjeti na sljedećem primjeru.

Isječak koda 6 – primjer mapiranja akcije

```
<action name="TeachMeSHome"
class="hr.fer.zemris.jcms.web.actions2.course.teachMe.TeachMeSHome">
            <result type="tiles">v2.course.teachMe.studentsHome</result>
</action>
```

Definira se ime akcije, razred koji će poslužiti tu akciju te rezultat po završetku. Rezultati će biti detaljnije opisani u nastavku. Ime akcije se koristi u URI-u koji pokreće tu akciju. Tako će akcija iz isječka koda 4 biti pokrenuta pomoću sljedećeg URI-a:

https://ferko.fer.hr/ferko/TeachMeSHome.action

Razvojni okvir Struts2 se nakon toga brine da analizira URI i pronađe odgovarajuću akciju. U ovom slučaju to je sljedeća akcija:

Isječak koda 7 – primjer razreda koji predstavlja akciju

```
@WebClass(dataClass=TeachMeSHomeData.class)
public class TeachMeSHome extends Ext2ActionSupport<TeachMeSHomeData>{

	private static final long serialVersionUID = 1L;

	@WebMethodInfo
	public String execute(){
		TeachMeSHomeService.showHome(getEntityManager(), data);
		return null;
	}

	public void setCourseInstanceID(String courseInstanceID){
		data.setCourseInstanceID(courseInstanceID);
	}
}
```

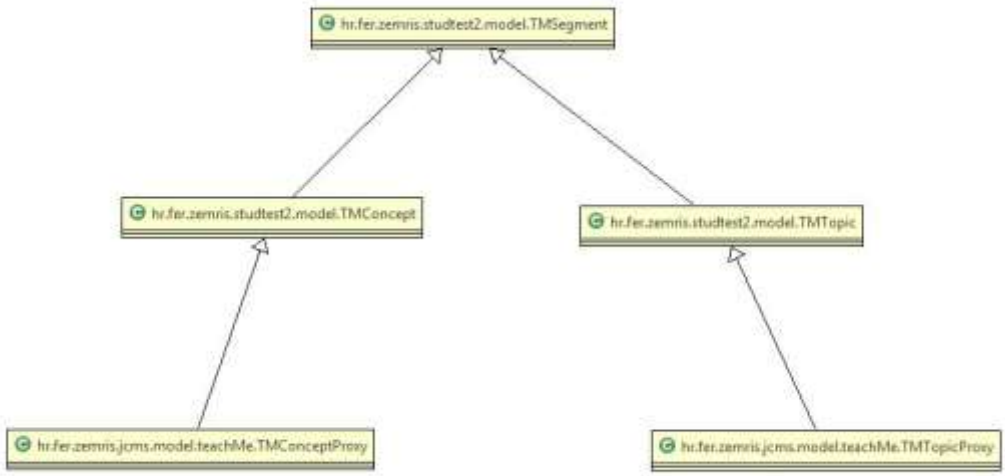
Ovdje se također koriste anotacije no ove anotacije nisu dio razvojnog okvira Struts2. One su definirane unutar sustava Ferko. *@WebClass* označava da se radi o razredu koji predstavlja nekakvu akciju. Svaka akcija mora imati pridruženi razred koji služi za prijenos informacija. U ovo slučaju radi se o *TeachMeSHomeData* kao što je to i naznačeno. Svaka akcija sadrži *execute()* metodu koja se, ukoliko nije drugačije naznačeno, poziva prilikom pokretanja akcije. Ukoliko se želi pozvati neka druga metoda onda se to mora eksplicitno tražiti unutar URI-a.

Sloj usluge

Ferko je izgrađen na taj način da je sloj usluge potpuno izdvojen u posebne razrede. Na taj način se postiže bolja modularnost koda. U isječku koda 5 vidimo da se u execute metodi akcije poziva razred *TeachMeSHomeService*. Taj razred sadrži svu logiku koja stoji iza akcije TeachMeSHome. Sve ulazne podatke dohvaća iz pripadajućeg *data* razreda, obavlja potrebne radnje nad njima, generira izlazne podatke koje također zapisuje u pripadajući *data* razred.

U sloju usluge obično se dohvaćaju podatci iz baze podataka. Kako se svi podatci modula za poučavanje nalaze na sustavu StudTest2 potrebno ih je prvo dohvatiti. Ovdje Ferko koristi prije spomenute naredbe sustava StudTest2 kako bi ostvario komunikaciju s njime. Dohvaćen odgovor pozvane naredbe se interpretira na unaprijed dogovoren način. Svi algoritmi izvode se na strani sustava StudTest2 pa sustav Ferko većinom dobivene podatke samo prikazuje klijentu.

Za razliku od sustava StudTest2 gdje se direktno iz baze dohvaćaju objekti, sustav Ferko radi s tzv. *proxy* razredima. *Proxy* razred je u općem slučaju razred koji ne sadrži podatke već referencu na podatke. Ovaj obrazac je pogodan kada nije potrebno dohvaćati odmah sve podatke već tek kada oni zatrebaju. Tada *proxy* objekt traži preko reference zatražene podatke.



Slika 15 - *proxy* razredi u modulu za poučavanje

U ovom slučaju postoje razredi *TMTopicProxy* i *TMConceptProxy*; oba su podrazredi pripadajućih pravih razreda. Kao što je u poglavlju 5.1.2. Komunikacija sa sustavom StudTest2 pokazano dohvaćaju se samo osnovni podatci o segmentima a ne i reference. Ti osnovni podatci se pohranjuju u *proxy* objekte i koriste normalno kao da radimo s pravim *TMTopic* i *TMConcept* objektima. Ukoliko je potrebno doći do preduvjeta ili podsegmenata, *proxy* objekti detektiraju da nemaju te podatke te se pokreće pozivanje naredbe sustava StudTest2 koja dohvaća podatke o traženim segmentima, koji su onda opet *proxy* objekti.

Isječak koda 8 – *proxy* razred TMConceptProxy

```
public class TMConceptProxy extends TMConcept {

	private boolean initialized = false;

	private TMSegmentFetcher fetcher;

	@Override
```

```
        public Set<TMPrerequisite> getPrerequisites(){
                if(!initialized){
                        prerequisites = fetcher.getPrerequisites(this);
                        initialized = true;
                }
                return prerequisites;
        }
}
```

Dohvaćanje podataka različitih segmenata ugrađeno je u razred *TMSegmentFetcher*. On sadrži internu priručnu memoriju (engl. *cache*) u kojem sadrži segmente korištene tijekom obrađivanja trenutne akcije. Do pozivanja naredbi sustava StudTest2 doći će samo ako se traži segment koji još nije bio tražen, odnosno za koji *TMSegmentFetcher* ne sadrži već podatke.

Isječak koda 9 – skraćeni prikaz razreda TMSegmentFetcher

```
public class TMSegmentFetcher {
        private List<TMSegment> segments;

        public TMSegment getSegment(String URI) {
                ...
                // pozivanje naredbi StudTest-a
                ...
        }

}
```

Razvojni okvir Apache Tiles

Nakon što je akcija pokrenuta i nakon što su svi rezultati pripremljeni, potrebno je korisniku to i prikazati. Koristi se *Apache Tiles*, razvojni okvir koji se bazira na predlošcima, izgrađen na način da se što više pojednostavi razvoj korisničkih sučelja web-aplikacija. Razvojni okvir Apache Tiles omogućavaju autoru da definira fragmente stranice koji se onda mogu slagati u kompletnu stranicu prilikom izvođenja. U kodu se povezuju s jednostavnim naredbama (npr. engl. *include*) kako bi se izbjeglo dupliciranje zajedničkih elemenata za sve stranice ili se pridružuju drugim predlošcima kako bi se razvili složeniji predlošci. Na taj način se kroz cijelu stranicu na jednostavan način može ugraditi konzistentan izgled [15].

U svakom *data* objektu koji sadrži sve korisničke rezultate postoji varijabla *result* koja je po tipu niz znakova. Razvojni okvir Struts2 zatim gleda vrijednost te varijable koja je preslikana na određeni predložak. Preslikavanje se nalazi u već spomenutom xml dokumentu *jcms.xml* koji se može vidjeti u isječku koda 5. No tamo samo piše ime traženog predloška a ne i gdje se on nalazi i od čega se sve sastoji. Te informacije se nalaze unutar drugog xml dokumenta *tiles.xml*.

Isječak koda 10 – dio *tiles.xml* dokumenta

```
<definition name="v2.course.teachMe.studentsHome" extends="v2.course">
            <put-attribute name="cbody" value="/WEB-
INF/pages/v2/course/teachMe/StudentsHome.jsp" cascade="true" />
</definition>
```

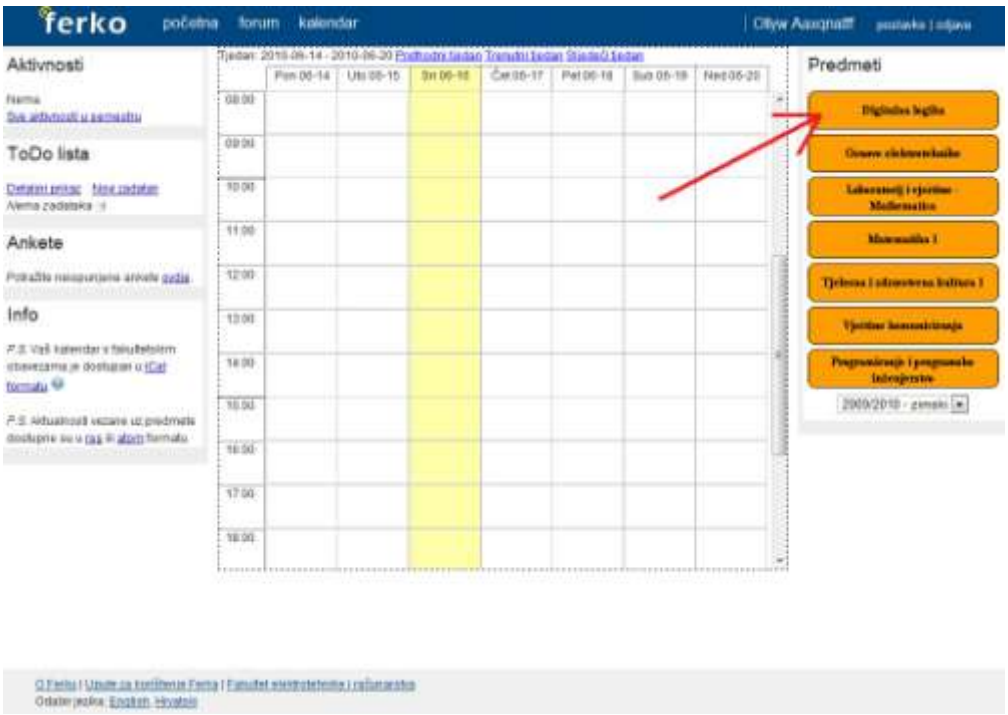
Iz primjera se vidi da se koristi nasljeđivanje kako bi se isti izgled stranice očuvao i ovdje. Definiran je i relativan put do same *jsp* stranice koja će se prikazati.

Dokumenti u sustavu Ferko

U poglavlju 2.2.2. Dokumenti definirali smo dokumente vezane za pojedine segmente. Zbog arhitekture oba sustava odlučeno je da se dokumenti nalaze na Ferku. Ovo je jedini dio modula za poučavanje koji se nalazi na Ferku (osim koda koji generira prikaz samom korisniku). Iako ovo smanjuje fleksibilnost modula (da se recimo umjesto Ferka koristi neki drugi sustav za prikaz rezultata algoritama) ovo je odlučeno kako bi se smanjila količina komunikacije između sustava Ferko i sustava StudTest2 koja je već i sada intenzivna.

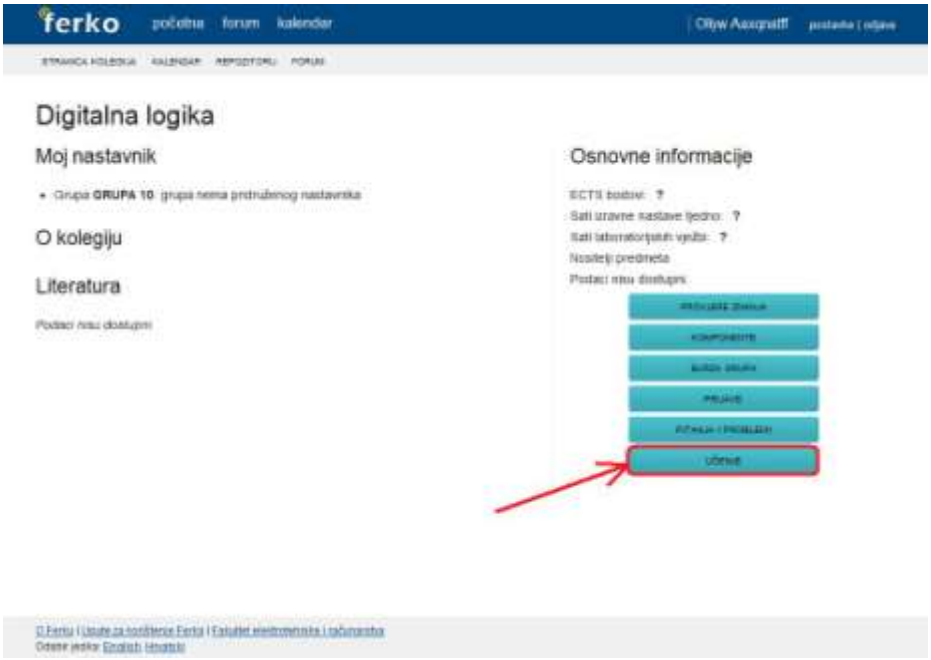
6. Modul za prilagodljivo poučavanje

Modul za prilagodljivo poučavanje nalazi se unutar sustava Ferko. Studenti nakon prijave na sustav prvo odabiru predmet koji žele učiti.



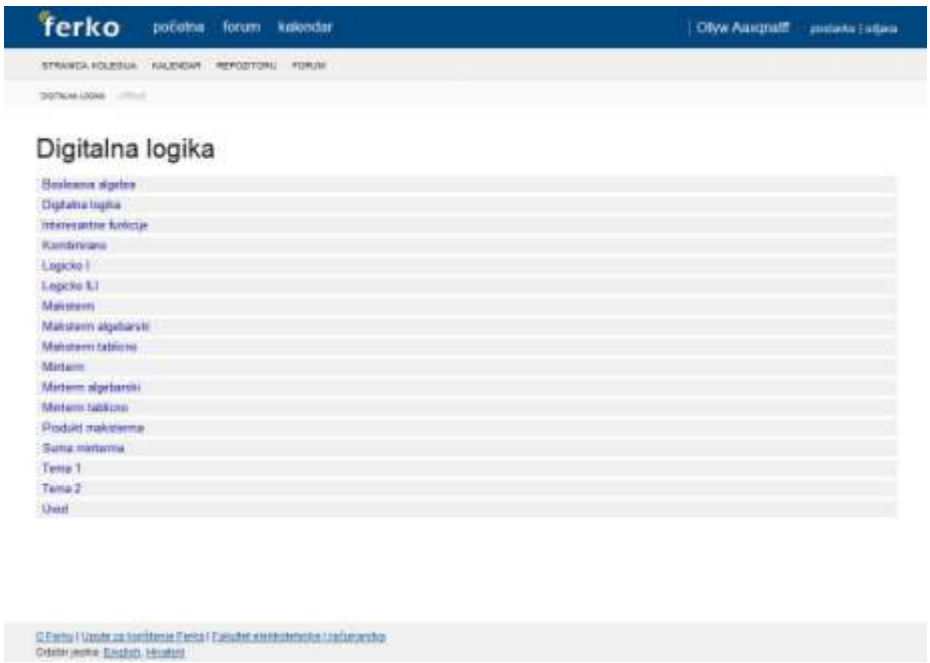
Slika 16 – prikaz studentovih predmeta unutar Ferka

Ukoliko je omogućen modul za prilagodljivo poučavanje za taj predmet na desnoj strani nalaziti će se odgovarajuća poveznica. Pritiskom na nju student će započeti učenje gradiva tog predmeta.



Slika 17 – prikaz poveznice do modula za prilagodljivo poučavanje

Odabirom poveznice *Učenje* započinje se učenje gradiva predmeta. Studentu se prvo prikazuje abecedni popis svih segmenata na predmetu. Alternativna ideja je prikaz izgrađenog puta učenja nad cijelim gradivom.



Slika 18 – prikaz svih segmenata unutar predmeta

Odabirom nekog segmenta prikazuje se stranica za poučavanje. Na lijevoj strani prozora nalazi se generirani put učenja zajedno s ispisanim postocima znanja studenta za svaki segment. Na desnoj strani prikazan je dokument za prvi segment u putu učenja. Primjer dokumenta za koncept *Uvod* je prikazan na slici 19.

Već smo naučili da su mintermi Booleove funkcije koje u tablici kombinacija vrijednost 1 poprimaju za samo jednu kombinaciju. Primjerice, funkcija od dvije varijable ima 4 minterma:

A	B	m0	m1	m2	m3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

// način označavanja - malo slovo plus indeks

Promatramo li lijevi dio tablice kao binarno zapisane brojeve, u prvom retku čitamo 00, što odgovara broju 0 - minterm m0 za tu kombinaciju poprima vrijednost 1 dok za sve ostale kombinacije poprima vrijednost 0. U drugom retku nalazi se kombinacija 01 što odgovara broju 1 - minterm m1 za tu kombinaciju poprima vrijednost 1 dok za sve ostale kombinacije poprima vrijednost 0. U trećem retku nalazi se kombinacija 10 što odgovara broju 1 - minterm m2 za tu kombinaciju poprima vrijednost 1 dok za sve ostale kombinacije poprima vrijednost 0. Konačno, u četvrtom retku nalazi se kombinacija 11 što odgovara broju 1 - minterm m3 za tu kombinaciju poprima vrijednost 1 dok za sve ostale kombinacije poprima vrijednost 0. Uočimo također da je svaki minterm zapravo funkcija logičko-I. Tako je m0=A'*B', m1=A'*B, m2=A*B' a m3=A*B.

Kako općenito doći do zapisa i-tog minterma funkcije od n varijabli? Prisjetimo se: minterm je funkcija logičko-I (tj. produkt) koji poprima vrijednost 1 samo za jednu kombinaciju varijabli - i to upravo onu određenu indeksom "i". Primjerice, kako glasi zapis minterma m17 funkcije f(A,B,C,D,E,F)? Kako je ovo funkcija od 6 vraijabli, prikažimo broj 17 kao 6-bitni binarni broj: 17 == 010001k, pri čemu svaki bit odgovara pojedinoj varijabli gledano s lijeva na desno. Kako je minterm produkt, između naprosto napišemo znak *, pa dobijemo:

0 * 1 * 0 * 0 * 0 * 1 * 0

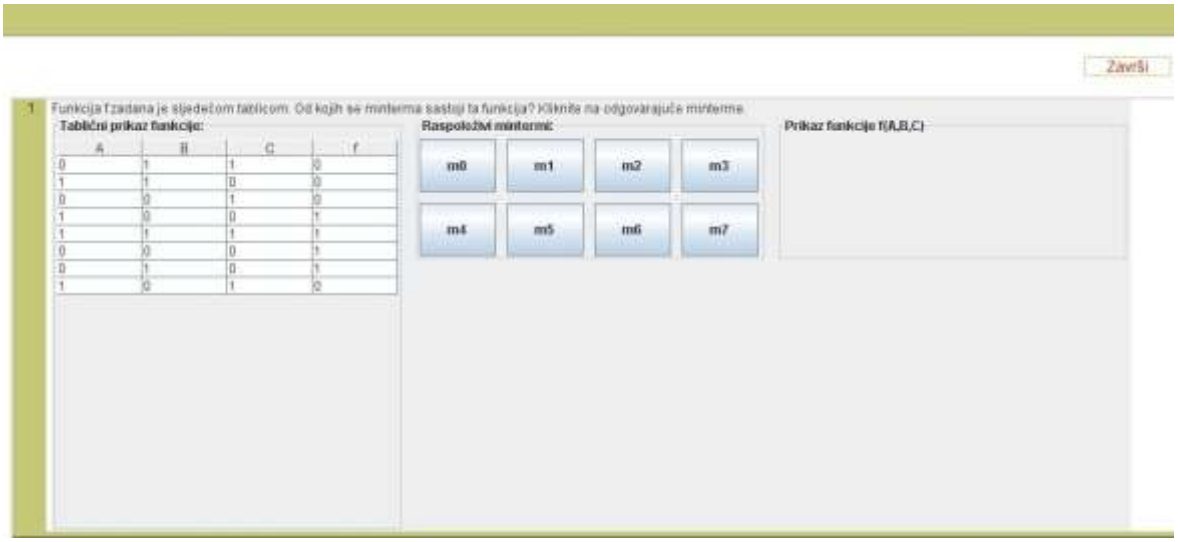
što je očito 0. Kako za ovu kombinaciju naš minterm mora poprimiti vrijednost 1, da bi umnožak bio jedan, svi njegovi dijelovi moraju biti jedan, a to znači da nule moramo komplementirati:

0' * 1 * 0' * 0' * 0' * 1 * 0' = 1 * 1 * 1 * 1 * 1 * 1 * 1 = 1

Kako smo komplementirali vrijednost varijabli A, C, D i E, slijedi da je zapis minterma m17 funkcije f(A,B,C,D,E,F) jednak A'*B*C'*D'*E'*F.

Slika 19 – primjer dokumenta za koncept *Minterm*

Tokom učenja student može odabrati iz padajućeg izbornika na vrhu stranice neki drugi dokument pisan za drugačiji tip učenja. Nakon proučavanja dokumenta student odabire poveznicu *Ispitaj me* na dnu stranice. Odabirom te poveznice otvara se zadatak koji ispituje studentovo znanje o dotičnom konceptu. Na slici 20 prikazan je zadatak koji ispituje znanje o konceptu *Funkcija i minterm tablično*.



Slika 20 – prikaz zadatka za koncept *Funkcija i minterm tablično*

Ukoliko student točno odgovori na zadatak sustav mu prikazuje dokument sljedećeg segmenta u putu učenja. Ukoliko nije točno odgovorio na zadatak sustav ga vraća na početni dokument. Alternativno rješenje je omogućiti studentu da odabere između rješavanja zadatka dok ga ne riješi točno i povratka na dokument.

7. Zaključak

Ovaj diplomski rad se sastoji od dva dijela. U prvom dijelu obrađena je teorijska pozadina jednog adaptivnog edukacijskog sustava. Prvo je prikazan način pohrane gradiva u sustav – model domene. Zatim je prikazan jedan mogući model studenta baziran na Kolbovoj podjeli tipova učenja. Potom su obrađeni algoritmi koji povezuju model domene i model studenta u adaptivno poučavanje studenata. U drugom dijelu objašnjena je sama implementacija početnih ideja kroz modul za poučavanje unutar dva postojeća sustava – StudTest2 i Ferko.

Kritični dijelovi sustava su generiranje puta učenja te korigiranje tipa studenta nakon ispitivanja. Iako logička valjanost generiranja puta učenja u ovom trenutku može djelovati ispravno ona će biti vidljiva tek nakon prvih testiranja. Problem raspoznavanja pravog tipa učenja studenta je najveći izazov kod adaptivnih edukacijskih sustava. Predviđa se da će model studenta i pripadajući adaptivni algoritam zahtijevati najviše vremena prilikom buduće analize.

8. Literatura

[1] *Learning Styles and Personality Types of Computer Science Students at a South African University.* Galpin, Vashti, Sanders, Ian i Chen, Pei-yu. 2007., SIGCSE Bulletin Volume 39 Issue 3, str. 201-205.

[2] Moodle.org [Mrežno][Citirano 9. lipnja 2010.]

http://moodle.org/

[3] WebCT [Mrežno] [Citirano 9. lipnja 2010.]

http://webct.carnet.hr/wct/owebctu.html

[4] *Draft Standard for Learning Object Metadata*. Skupina autora. 2002. Institute of Electrical and Electronics Engineers Standards Draft

[5] SCORM – Wikipedia [Mrežno] [Citirano 9. lipnja 2010.]

http://en.wikipedia.org/wiki/Sharable_Content_Object_Reference_Model

[6] Wikipedija. *Algoritam.* [Mrežno] [Citirano: 9. lipnja 2010.]
http://hr.wikipedia.org/wiki/Algoritam

[7] Wikipedia. *Adaptive_algorithm.* [Mrežno] [Citirano: 9. lipnja 2010.]
http://en.wikipedia.org/wiki/Adaptive_algorithm

[8] Breadth-first search – Wikipedia. [Mrežno] [Citirano 9. lipnja 2010.]
http://en.wikipedia.org/wiki/Breadth-first_search

[9] Depth-first search – Wikipedia [Mrežno] [Citirano 9. lipnja 2010.]
http://en.wikipedia.org/wiki/Depth-first_search

[10] jCourseware@ZEMRIS [Mrežno] [Citirano: 9. lipnja 2010.]
http://java.zemris.fer.hr/jCourseware/7.jsp

[11] Computer-Based Knowledge, Self-Assessment and Training. Čupić, Mihajlović. 2009., Engineering Education Volume 26 Number 1, str. 111-125.

[12] Documentation – Hibernate – JBoss Community [Mrežno] [Citirano: 9. lipnja 2010.] http://www.hibernate.org/docs.html

[13] Java Persistance API – Wikipedia [Mrežno] [Citirano 9. lipnja 2010.]

http://en.wikipedia.org/wiki/Java_Persistence_API

[14] jcms – Trac [Mrezno] [Citirano: 9. lipnja 2010.]

http://morgoth.zemris.fer.hr/trac/jcms/wiki

[15] Apache Tiles – Framework – Home [Mrezno][Citirano: 9. lipnja 2010.]

http://tiles.apache.org/framework/index.html

9. Sažetak

Sustav za prilagodljivo poučavanje studenata izrađen u Web tehnologiji

U okviru ovog diplomskog rada proučava se teorijska pozadina adaptivnog generiranja i kombiniranja materijala za učenje, načini zapisivanja znanja i njegova obrada. Izrađeni su algoritmi koji rade nad znanjem u cilju donošenja odluka o prezentaciji gradiva studentu. Definiran je model studenta s obzirom na Kolb-ove tipove učenja. Cijeli modul poučavanja ukomponiran je unutar dva sustava – StudTest2 i Ferko. U sustavu StudTest2 se nalazi baza podataka zajedno sa svim algoritmima dok Ferko služi samo za prikaz rezultata studentima i nastavnicima.

Ključne riječi: *e-učenje*, *adaptivno*, *poučavanje*, *web-aplikacije*, *StudTest2*, *Ferko*

This thesis examines theoretical background of generating and combining adaptive learning materials, methods of recording knowledge and its processing. Algorithms are designed that operate over knowledge that make decisions about its presentation to the student. Student model is defined with respect to Kolb's learning types. Whole teaching module is build within two systems – StudTest2 and Ferko. Database along with all the algorithms is located in StudTest2, while Ferko only serves to display results to students and teachers.

Key words: *e-learning*, *adaptive*, *teaching*, *web aplications*, *StudTest2*, *Ferko*