SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET

Diplomski rad

Pristup oblikovanju ontološki utemeljenog područnog znanja

Student: Šime Iglić

Mentor: doc.dr.sc. Ani Grubišić

Split, listopad 2013.

Sadržaj

1 Uvod

U posljednje vrijeme učenici se susreću s mnoštvom informacija koje moraju naučiti. Obujam gradiva koje učenici moraju savladati postaje sve veći, te se javlja problem kako učenik to sve može naučiti i na koje načine. Jedan od zanimljivijih načina na koji se ovaj problem može riješiti je primjena konceptualnih mapa koje na jedan sasvim drugačiji, to jest slikovitiji, način pomažu učeniku da nauči i zapamti velik broj informacija.

Posljednjih desetljeća ubrzani razvoj tehnologija i weba doveo je do razvoja sustava e-učenja. Svakodnevno se ti sustavi razvijaju i usavršavaju da bi se učenicima osiguralo što bolje učenje i poučavanje.

Sustavi e-učenja su se toliko razvili da su postali i inteligentni te mogu učenika poučavati kao što ga poučava stvarni učitelj u učionici, komunicirajući s učenikom tijekom cijelog procesa. Neki sustavi e-učenja imaju mogućnost prilagođavati se znanju učenika, te njegovu napretku.

Međutim, koliko god sustavi e-učenja bili napredni i inteligentni, ti sustavi moraju sadržavati neko područno znanje koje će učenik učiti i koje sustav e-učenja kao onaj koji poučava mora znati. Većinom je to područno znanje u obliku podataka uneseno u bazu podatka sustava e-učenja koja je neizostavni dio svakog sustava e-učenja. Kod područnog se znanja susrećemo s problemima: kojom je brzinom moguće unijeti područno znanje u bazu podataka sustava e-učenja, koje tehnologije i programske alate koristiti kako bi se to ostvarilo, koji se koraci moraju poduzeti da bi to bilo moguće, kolika je stručnost potrebna onoga koji to radi, te koliko je sve to moguće automatizirati i ubrzati koristeći neku od tehnologija i programskih alata. U želji za boljim razumijevanjem problema i eventualnom napretku u rješavanju istih pristupit će se rješavanju problema na najbolji način, ovisno fizičkoj izvedivosti i dostupnosti materijala i tehnologija.

Izvedivost projekta, kao i potencijalni rezultati otkrit će nam o koliko složenim problemima je riječ, a u slučaju pozitivnih rezultata govorimo o stvaranju izvrsne podloge za daljnja istraživanja i pokušaje unaprjeđenja izvršavanja zadataka.

Struktura ovog diplomskog rada je sljedeća: u prvom poglavlju upoznajemo se s problematikom ovog rada. U drugom poglavlju se opisuju inteligentni tutorski sustavi kao i sam problem područnog znanja, to jest problem njegova unosa u inteligentni tutorski sustav. U trećem poglavlju se opisuje ontologija, primjena ontologije, alati za kreiranje ontologije, načini učenja ontologije, problemi u učenju ontologije te jezik ontologije. U četvrtom poglavlju se opisuje praktični dio diplomskog rada te njegova realizacija kroz korake od početka do kraja. U petom poglavlju se daje zaključak, to jest osvrt na rad i ono što se u radu postiglo.

2 Inteligentni tutorski sustavi

Napretkom tehnologija napredovali su i sustavi e-učenja i to u na način da ti sustavi više ne koriste samo za učenje nego su postali toliko razvijeni da su u mogućnosti i poučavati učenike. Takvi sustavi se nazivaju inteligentni tutorski sustavi. Što su to inteligentni tutorski sustavi te koji se problemi javljaju kod rada sustava može se vidjeti u sljedećim poglavljima.

2.1 Definicija inteligentnih tutorskih sustava i područnog znanja

„Matt Lewis: Inteligentni tutorski sustav je sustav koji minimalno sadrži mogućnost simulacije čovjekovog načinima rješavanja problema u danom područnom znanju te poput „živog“ tutora ima sposobnost odvajanja područnog znanja od pedagogijskog“ (Stankov, 2010) . Inteligentni tutorski sustav se sastoje od sljedećih komponenti (Stankov, 2010):

- Modul stručnjaka (područno znanje) nosilac područnog znanja s kojim će učenik tijekom učenja i poučavanja komunicirati.
- Modul učenika (znanje učenika) obuhvaća sve aspekte stjecanja učenikova znanja i vještina u danom područnom znanju.
- Modul učitelja (tutorsko znanje) je modul koji je nositelj scenarija poučavanja i pedagogijskih znanja s kojima raspolaže „živi “ učitelj.
- Okruženje nastavnog procesa i sučelja učenika (modul komunikacije) koji sadrži elemente inteligentnog tutorskog sustava koji podržava učenikov rad u raznim situacijama te različite programske alate koji mu olakšavaju proces učenja i poučavanja.

Modul stručnjaka je najvažniji modul inteligentnog tutorskog sustava često nazivan i kralježnica inteligentnog tutorskog sustava. Da bi ovaj modul bio što bolje napravljen, a samim time i inteligentni tutorski sustav na što većoj razini zadovoljavao svoju namjenu, potrebno je što bolje oblikovanje i predstavljanje područnog znanja. Područno znanje je znanje o području koje je učeniku namijenjeno za učenje i poučavanje (Stankov, 2010) . Siromašno, slabo oblikovanje i prikazivanje područnog znanja dovodi do poteškoća u korištenju inteligentnih tutorskih sustava te samim time i do slabog znanja učenika, međutim nijedan sustav ne može posjedovati sve znanje iz nekog područja zbog velikog obujma znanja čak i kod nekog ograničenog područja. Bilo koji oblik predstavljanja znanja mora zadovoljiti sljedeće uvjete:

- Izravno modeliranje tehnika koje bi stručnjak primijenio u rješavanju problema s tog područja, čak i pod uvjetom siromašnije baze znanja jer je poučavanje jedino čovjekov način i tehnika rješavanja problema.
- Eksplicitno predstaviti strategije i tehnike rješavanja problema. Sustav mora predstaviti sve vjerojatne strategije čak i one koje su pogrešne da bi tutor mogao prepoznati što učenik pokušava učiniti.

Tri su pristupa za strukturiranje znanja u modulu stručnjaka inteligentnog tutorskog sustava (Stankov, 2010):

- Model „crne kutije“
- Model „prozirne kutije“
- Kognitivni ili spoznajni model

Kod modela crne kutije stručnjak crne kutije generira korektno ulazno – silazno ponašanje preko zadataka u područnom znanju i tako omogućava besprijekornu prosudbu. Međutim problem je što je njen mehanizam nepristupačan (Stankov, 2010).

Kod modela prozirne kutije temeljna metodologija u izgradnji modula stručnjaka zahtijeva inženjera znanja i stručnjaka područnog znanja koji je u stanju identificirati problemsko područje te iterativno testirati i unaprijediti sustav. Sustave temeljene na modelu prozirne kutije karakterizira obimnost i posebno izražena priroda znanja poput ljudskog razmišljanja (Stankov, 2010).

Kognitivni model je model koji je mnogo djelotvorniji u simulaciji čovjekovog načina rješavanja problema jer omogućava da modul stručnjaka oponaša čovjekovo razmišljanje i način upotrebe znanja. Vrijednost ovog pristupa se ogleda u mnogo prihvatljivijem načinu komuniciranja učenika i modula stručnjaka ali ovakav pristup ima i nedostataka. Razvoj kognitivnog modela je mnogo ograničenja i vremenski zahtjevnija zadaća nego li jednostavan razvoj ekspertnog sustava. Pokretanje računanja kognitivnih modela može biti izrazito računalno skupo. Jedan od problema je u tomu kolika količina detalja koje je potrebna unijeti u kognitivni model (Stankov, 2010).

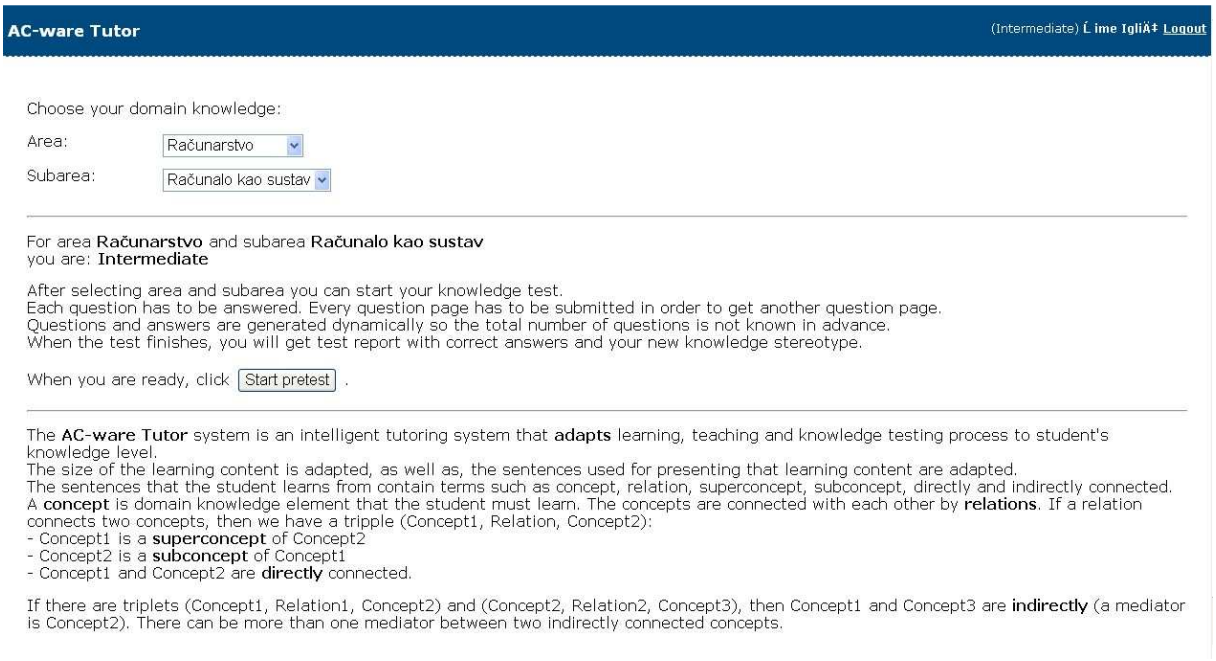
Kao što je u poglavlju naglašeno kod modula stručnjaka javlja se problem oblikovanja područnog znanja međutim to nije jedini problem koji se može javiti. Jedan od važnijih problema koji se javlja kod područnog znanja je kako to područno znanje unijeti u određeni sustav. Konkretni primjer problematike može se vidjeti kod sustava AC-ware Tutor-a.

2.2 AC-ware Tutor

AC-ware Tutor (Grubišić, 2012) (Slika 1) je model sustava za automatsko i dinamičko generiranje računalom oblikovanog nastavnog sadržaja, koji se prilagođava trenutnom znanju učenika. Ontološki pristup u opisivanju područnog znanja omogućava jednostavnu formalizaciju deklarativnog znanja korištenjem različitih alata koji podržavaju rad s konceptima i relacijama. Učenje na sustavu se odvija tako da sustav najprije preko inicijalnog testa provjeri znanje učenika te učeniku na osnovu pokazanog znanja pridruži stereotip. Stereotip može biti (Grubišić, 2012): početnik, novak, osrednji, napredni, stručnjak.

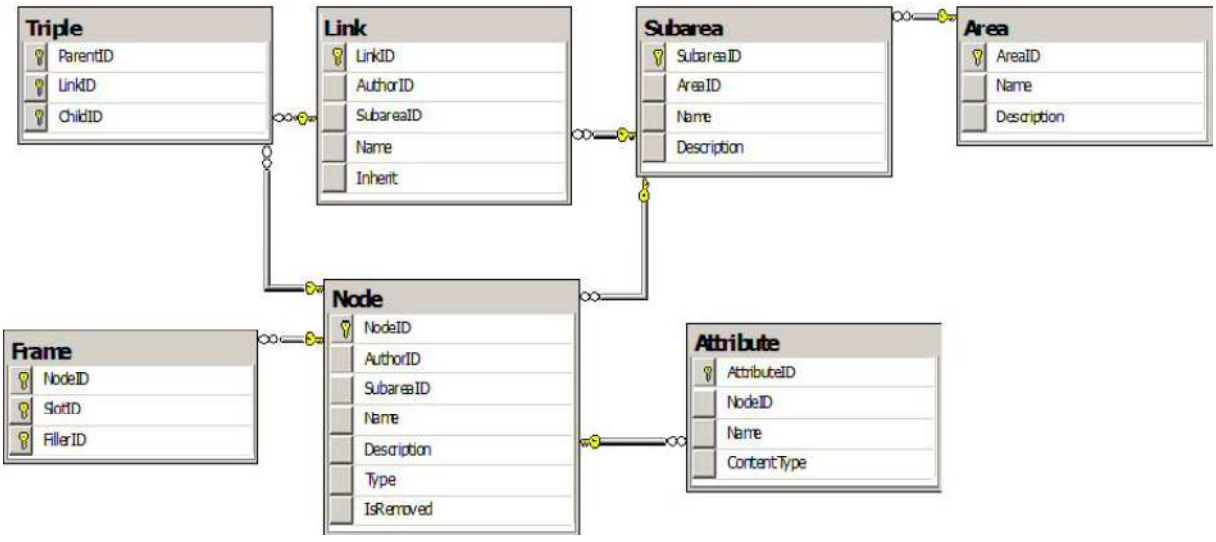
Stereotip koji je pridružen učenikovom znanju ovisi o njegovom pokazanom znanju na inicijalnom testu. Nakon što je učenik dobio stereotip učenik određeno vrijeme uči te nakon toga ponovno na isti način slijedi provjera znanja. U slučaju da učenik pokaže znanje on napreduje što se tiče stereotipa i automatski mu se količina gradiva za naučiti prilagođava njegovom stereotipu to jest povećava, ukoliko učenikovo znanje ostaje nepromijenjeno stereotip također ostaje nepromijenjen, ukoliko se

učenikovo znanje pogoršalo učenik dobiva stereotip niže razine te mu se automatski gradivo prilagođava i postaje lakše. Učenik sam odabire vrijeme koje će provoditi u učenju i do kojeg stereotipa želi ići.



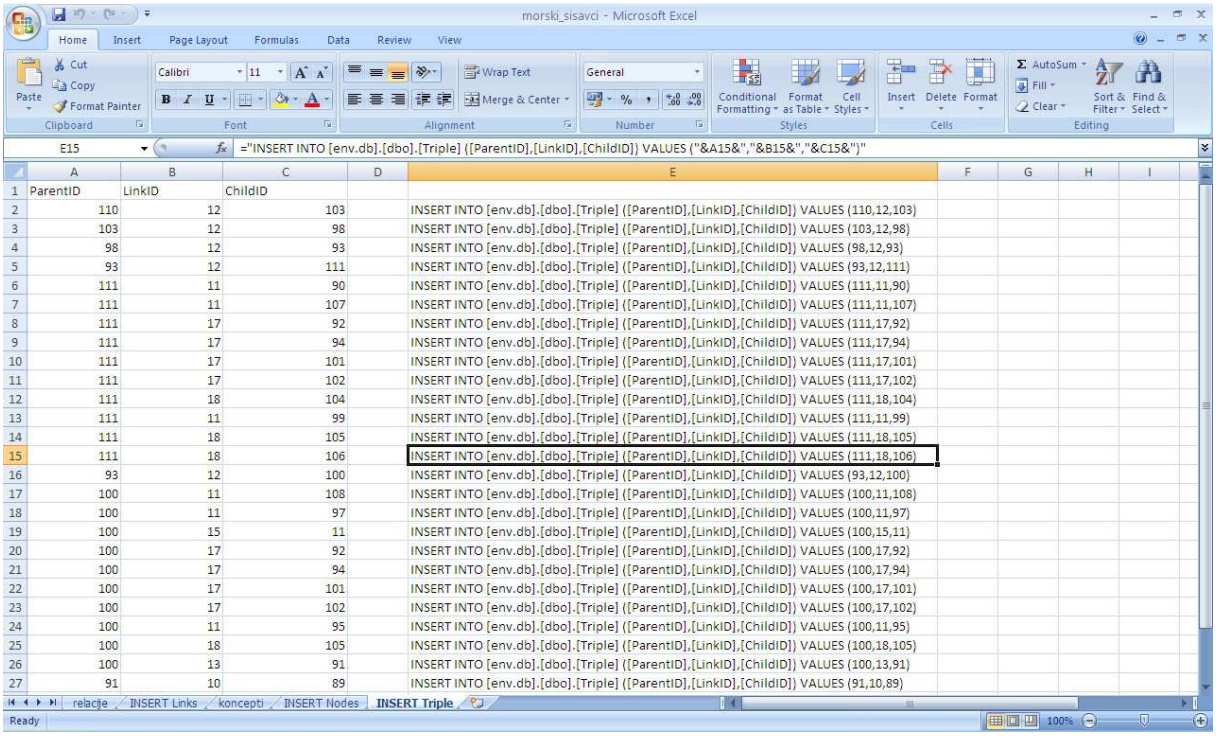
Slika 1: Prikaz korisničkog sučelja AC-ware Tutor-a

U nastavku će ukratko biti objašnjena problematika unosa područnog znanja u sustav AC-ware Tutor. Kao prvo važno je napomenuti da bi se područno znanje moglo unijeti u neki sustav taj sustav mora imati bazu podataka (Slika 2) koja se kod ovakvih sustava naziva i baza znanja u koju je to područno znanje pohranjeno i čiji se obujam može povećavati što znači da se baza sustava može nadopunjavati s novim znanjem.



Slika 2: Prikaz baze podataka područnog znanja

Jedan od problema unosa područnog znanja u sustav AC-ware Tutor je taj što je to vrlo dugotrajan proces koji zahtijeva mnogo vremena zato što se područno znanje unosi ručno koristeći MS Excel alat. Da bi se podaci unijeli u bazu podataka u MS Excel-u se svi podaci i naredbe za unos podataka u bazu trebaju unijeti ručno kao što je prikazano na slici ispod.



	A	B	C	D	E
1	ParentID	LinkID	ChildID		
2	110	12	103		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (110,12,103)
3	103	12	98		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (103,12,98)
4	98	12	93		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (98,12,93)
5	93	12	111		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (93,12,111)
6	111	11	90		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,11,90)
7	111	11	107		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,11,107)
8	111	17	92		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,17,92)
9	111	17	94		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,17,94)
10	111	17	101		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,17,101)
11	111	17	102		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,17,102)
12	111	18	104		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,18,104)
13	111	11	99		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,11,99)
14	111	18	105		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,18,105)
15	111	18	106		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (111,18,106)
16	93	12	100		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (93,12,100)
17	100	11	108		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,11,108)
18	100	11	97		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,11,97)
19	100	15	11		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,15,11)
20	100	17	92		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,17,92)
21	100	17	94		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,17,94)
22	100	17	101		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,17,101)
23	100	17	102		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,17,102)
24	100	11	95		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,11,95)
25	100	18	105		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,18,105)
26	100	13	91		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (100,13,91)
27	91	10	89		INSERT INTO [env.db].[dbo].[Triple] ([ParentID],[LinkID],[ChildID]) VALUES (91,10,89)

Slika 3: Primjer unosa podataka u tablicu trojke koristeći MS Excel

Na (Slika 3) je moguće vidjeti primjer ručnog unosa podataka u tablicu trojki koristeći MS Excel. Baza podataka područnog znanja AC-ware Tutora napravljena je u SQL programskom alatu. Da bi se problem unosa područnog znanja u sustav AC-ware Tutor olakšao i da bi se skratilo vrijeme samog unosa područnog znanja u sustav potrebno je ovaj proces automatizirati i ubrzati koristeći razne programske alate. Automatizacija unosa područnog znanja u sustav Ac-ware Tutor te načini ubrzanja njegovog unosa biti će objašnjeni u nekom od narednih poglavlja.

3 Ontologija

Ontologija (engl. *ontology*) u računarskim znanostima podrazumijeva način predstavljanja znanja kojim se oblikuje promatrana domena imenovanjem koncepata u domeni i formiranjem relacija među njima na način da budu razumljivi i čovjeku i računalu (Prcela, 2010).

Ontologije se smatraju idealnim rješenjem za mnoge aplikacije kao npr. integracija baza podataka, peer-to-peer sustavi, elektroničko poslovanje, usluge weba. U računalnoj znanosti postoje razni podatkovni i pojmovni modeli koji se smatraju ontologijama (npr. UML modeli, XML sheme, formalne ontologije, itd.). Ontologije se zapisuju pomoću jezika za zapisivanje ontologija.

Za ontologiju možemo reći da se sastoji od:

- Skupa koncepata
- Skupa relacija ili veza
- Skupa instanci dodijeljenih pojedinom konceptu

Složene (engl. *heavyweight*) i jednostavne (engl. *lightweight*) ontologije su ontologije koje se mogu konstruirati različitim tehnikama modeliranja. Za ovakve ontologije je važno reći da mogu biti implementirane u različitim jezicima. Ontologije mogu prezentirati visoku razinu informacija (engl. *highly informal*) koje su korisne korisnicima ako su zapisane u prirodnom jeziku. Ontologije mogu biti ograničeno informativne (engl. *semi-informal*) ako se zapisuju u restriktivnoj i strukturiranoj formi prirodnog jezika. Polu-formalne (engl. *semi-formal*) ontologije su zapisane u umjetnom jeziku ili u nekom formalno definiranom jeziku kao što je OWL (engl. *Ontology Web Language*). Ontologije koje su zapisane u prirodnom jeziku nisu čitljive od strane računala.

Različite tehnike koje se upotrebljavaju u pojedinim područjima kao što su: programsko inženjerstvo i baza podataka u svrhu modeliranja koncepata te veza između koncepata mogu poslužiti za izgradnju jednostavnijih ontologija (Uschold & Gruninger, 1996, str. 12). Postoji nekoliko klasifikacija ontologija.

Guarino (Guarino, 1998) predlaže jednostavnu klasifikaciju tipova ontologija prema stupnju njihove ovisnosti o zadatku i tako razlikuje opće ontologije, domenske ontologije, ontologije za specifične zadatke i aplikacijske ontologije.

Opće ontologije su već dobro razrađene i postoji nekoliko općih ontologija koje se često koriste u istraživanjima, ali takve ontologije su obično neupotrebljive za neke konkretne zadatke koje zahtjeva određena domena ili aplikacija (Guarino, 1998).

Domenske ontologije su ontologije koje se mogu iznova upotrebljavati samo u domeni koju obuhvaćaju. Domenske ontologije sadrže vokabulare s konceptima iz domene i relacije koje postoje između njih, informacije o aktivnostima koje se odvijaju u domeni, te elementarna pravila koja vladaju u istoj. Razlika između domenskih i općih ontologija je u tome što koncepti u domenskim ontologijama predstavljaju specijalizaciju koncepata definiranih u općim ontologijama (Guarino, 1998).

Ontologije zadataka sadrže vokabulare vezane uz neki specifični zadatak ili aktivnost. Ovakve ontologije često nastaju specijaliziranjem koncepata iz općih ontologija. Vokabular takvih ontologija sadrži pojmove koji se upotrebljavaju pri rješavanju problema vezanih uz neki zadatak koji ne mora biti u području samo jedne domene (Guarino, 1998).

Aplikacijske ontologije ovise o aplikacijama koje ih upotrebljavaju. Aplikacijske ontologije sadrže sve definicije potrebne za oblikovanje modela znanja nužnog za određenu aplikaciju, te ovakve ontologije često proširuju ili specijaliziraju koncepte iz vokabulara domenskih ontologija ili ontologija specifičnih zadataka (Guarino, 1998).

S obzirom na podatke kojima se raspolaže, učenje ontologije se može podijeliti na nekoliko zadataka: fokus na učenje koncepata, fokus na učenje relacija između koncepata, učenje i koncepata i relacija u isto vrijeme. Formalno rečeno, proces učenja ontologije podrazumijeva mapiranja između komponenti ontologije, gdje su neke komponente već poznate, a neke nedostaju i njih želimo otkriti.

Prednosti ontologija (Đuraković & Verić, 2013):

- Koherentna navigacija, omogućava kretanje od koncepta do koncepta u strukturi ontologije
- Fleksibilne ulazne točke, jer svaka specifična perspektiva u ontologiji može se pratiti i dovesti u vezu sa svim njegovim povezanim konceptima
- Veze koje ističu relevantne informacije bez zahtjeva prethodnog znanja o domeni ili njezinoj terminologiji
- Sposobnost da predstavi bilo koji oblik informacija, uključujući i nestrukturirane, polu-strukturirane i strukturirane podatke
- Povezivanje koncepta
- Integriranje sadržaja pravilnim povezivanjem i mapiranjem koncepata

Nedostatci ontologija (Đuraković & Verić, 2013):

- Koncept koji je nov nepoznat je za mnoge korisnike tako da se često javljaju greške
- Problem mapiranja dobivene ontologije sa postojećim domenskim ontologijama,
- Problem gubitka semantike nakon transformacije

3.1 Primjena ontologije

Ontologije u informacijskim znanostima se ponajviše koriste u: semantičkom webu, umjetnoj inteligenciji i programskom inženjerstvu.

Semantički web je nadopuna, ekstenzija World Wide Web-a koji je nastao u potrazi za efikasnijim rješenjima za pronalaženje informacija. Semantički web omogućuje bolju suradnju između računala i korisnika. Ideja semantičkog weba je da se sve informacije koje se na web-u pojavljuju označe posebnim oznakama tako da računala mogu automatizirano povezivati podatke iz jednoga tipa informacija s nekim drugim tipom informacija.

Semantički web omogućava da webu dostupni izvori informacija mogu biti organizirani i korišteni semantičkim, a ne sintaktičkim ili strukturalnim metodama. Semantički web predstavlja skup

programa koji prikupljaju sadržaj sa weba te: procesira informaciju, razmjenjuje rezultate sa drugim programima na globalnoj razini (Jović, 2005, str. 4). Semantički web je osmislio Tim Berners-Lee.

Semantički web će donijeti strukturu smislenom sadržaju Web stranica gdje se ontologije izdvajaju kao ključni mehanizam za predstavljanje znanja Semantičkog web-a. Ontologija pruža mogućnost generiranja novih činjenica na osnovu činjenica koje su eksplicitno dane. Ontologije omogućavaju postizanje višeg stupnja funkcionalnosti Web-a. Razlozi sve češće primjene ontologija na Web-u su (Đuraković & Verić, 2013):

- Povećanje preciznosti u pretraživanju Web-a
- Povezivanje informacija na stranicama sa pridruženim im strukturama znanja i pravilima zaključivanja
- Razmjenu znanja između ljudi i programskih jezika
- Dijeljeno razumijevanje domena
- Pružanje strukture za prijenos informacija u Semantičkom web-u
- Mogućnost nadograđivanja i mijenjanja u skladu s potrebama

Umjetna inteligencija (engl. Artificial Intelligence) je disciplina koja se bavi oblikovanjem inteligentnih sustava koji implementiraju ona svojstva ljudskog ponašanja koja se smatraju inteligentnim (Matešić, 2010). Područja koja se bave istraživanjem umjetne inteligencije su: računarstvo, matematika, ekonomija, psihologija, itd.

Umjetnu inteligenciju dijelimo na slabu i jaku. Kod slabe umjetne inteligencije razvijaju se inteligentni sustavi kojima se dodaju smo neki oblici ljudskog ponašanja, dok kod jake umjetne inteligencije se razvijaju inteligentni sustavi kojima se u potpunosti dodaju sve inteligentne osobine ljudskog bića (Slika 4).

Slika 4: Primjer umjetne inteligencije

Programsko inženjerstvo primjena je sustavnog, disciplinirano i mjerljivog pristupa u razvoju, uporabi i održavanju programske podrške (IEEE, 2004, str. 24). Disciplina programskog inženjerstva obuhvaća: znanje, alate, metode za definiranje zahtjeva programske podrške koji omogućavaju obavljanja zadataka dizajna programske podrške, konstrukcije programske podrške, testiranja

i održavanja programske podrške. Programsko inženjerstvo je disciplina koja posjeduje znanje iz različitih polja kao što su: računarstvo, menadžment, matematika, upravljanje projektima, itd.

Primjena ontologije u razvoju programske potpore može doprinijeti optimiziranju procesa razvoja i korištenja programskog rješenja, ontologija može poslužiti kao predložak za optimiranje oblikovanja novog sustava tako što se ponovno koristi znanje zapisano u strukturi ontologije. U korištenju programa ontologija može biti komunikator među agentima i na taj način omogućiti njihovo bolje međudjelovanje.

3.2 Elementi ontologije

Gledajući širu sliku svijeta, to jest svemira u kojem se živi može se vidjeti da se cijeli svemir sastoji od elemenata koji se neophodni za njegovo postojanje. Također primjena elemenata kao nečeg bitnog može se naći u ontologiji koja te elemente opisuje.

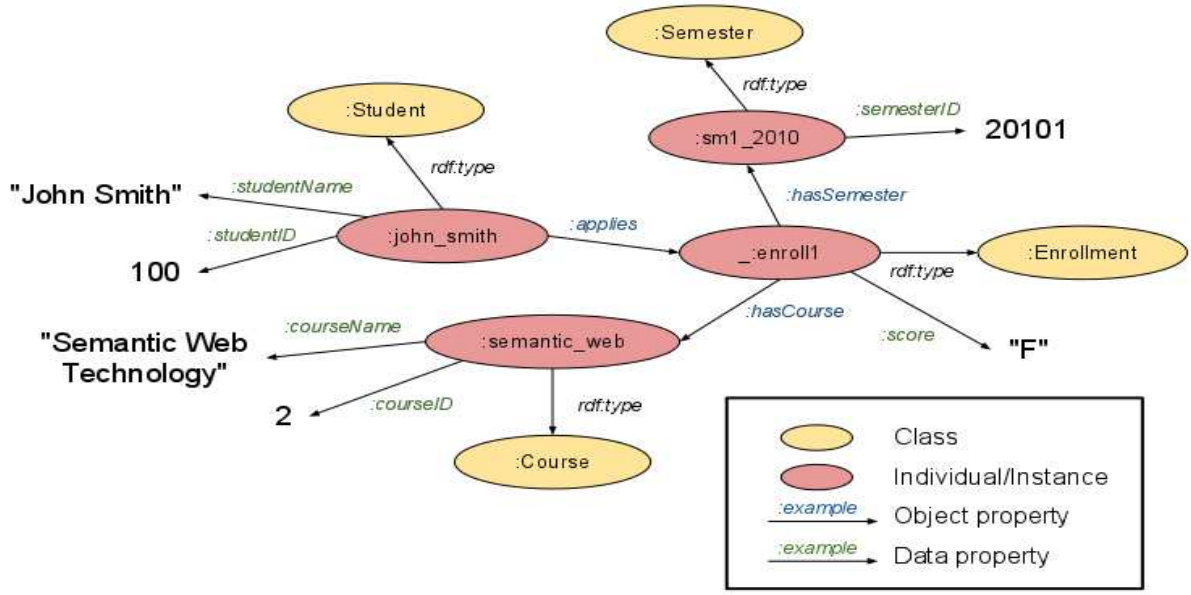
Osnovni elementi koje ontologije opisuju su (Slika 5):

- Individue: osnovne objekte „početne razine“
- Klase: zbirke ili tipove objekata
- Atribute: pripadajuća svojstva, pojave, karakteristike ili parametre koje objekt može imati ili distribuirati
- Odnose: način na koji se objekti odnose jedni prema drugima

Individue su osnovna, početna razina sastavnica ontologije koje u ontologiji mogu uključiti konkretne objekte poput ljudi, životinja, stolova, automobila, molekula i planeta. Ontologija ne mora uključiti individuu.

Klase su apstraktne grupe, skupine ili zbirke objekata. Mogu sadržavati individue, druge klase ili kombinaciju jednih i drugih.

Svi objekti u ontologiji mogu biti opisani tako da im pridružujemo atribute. Svaki atribut ima barem ime i vrijednost. Atribut se koristi pohranu informacija koje su specifične za objekt na koji se oslanja. Možemo ih koristiti za opisivanje odnosa između objekata u ontologiji. Relacija je atribut čija je vrijednost drugi atribut u ontologiji.



Slika 5: Grafički prikaz ontologije

3.3 Povijest ontologije

Ontologija potiče iz filozofske discipline koja je se procjenjuje da je stara kao i sam Aristotel, smatra se sinonimom za metafiziku. U to doba je bila nazvana prva filozofija, a izvodi se od grčke riječi **ontos** što u prijevodu znači biti to jest biće i **logos** što znači riječ ili govor (Legg, 2007). Takozvana filozofska ontologija je znanost o onome što jest, o vrstama i strukturama objekata, svojstava ,događaja, postupaka i odnosa u svim područjima stvarnosti (Smith & Welty, 2001). Materijalni objekt, Osoba, Vrijeme i Broj su neke od kategorija koje su predložili tradicionalni ontolozi. Aristotelovo djelo *Kategorije* predstavlja prvi pokušaj čovječanstva da odredi sustavnu formalnu ontologiju. Ovo je djelo značajno isprepleteno s Aristotelovom logikom. Razumijevanje pravila zaključivanja po kojima znanje razvrstano u određene kategorije može biti pretvoreno u kategorizirano znanje koje je za Aristotela neodvojivo od razumijevanja samih kategorija. Iako kasniji filozofi teže nastaviti istraživanja s ontologijom i formalnom logikom kao odvojenim disciplinama, formalna ih je ontologija u aplikacijama informacijske tehnologije bila prisiljena ujediniti. Tijekom godina istraživanja filozofske ontologije postoje mnoga neodgovorena pitanja o kojima se raspravlja i danas:

- Stupanj do kojeg se kategorije koje su smislili ljudi ontolozi mogu smatrati sveopće primjenjivima ili objektivnima u opreci s artefaktima u određenim kontekstima, kao što je kultura, vremensko razdoblje ...
- Do koje mjere treba kategorije jednog ontologa smatrati valjanima za sva vremena, a do koje su mjere djeljive u svim zajednicama?
- Mora li se sav ontološki posao raditi lokalno i opetovano?

Kasnih 50-tih godina ontologija svoju primjenu pronalazi u računalnim znanostima. Razvoj baza podataka i njihova složenost dovelo je do toga da su određeni problemi koje oblikovatelji baza podataka susreću zapravo ontološke naravi. Na primjer: što je osoba? Je li to entitet koji postoji kao cjelina u jednomu određenom vremenu ili je to entitet koji pokriva čitav "život osobe". Nekakva opća

standardizirana pojmovna shema smatra se poželjnom radi omogućavanja da se privremena rješenja ontoloških problema zaštite od računalnih programera koji bi te probleme koristili u zlonamjerne svrhe. Prema Tomu Gruberu (Gruber, 1993) značenje ontologije u kontekstu računalnih znanosti jest opis koncepata i odnosa koji mogu postojati za agenta ili zajednicu agenata specificirajući da je ontologija općenito rečeno set definicija formalnog rječnika. Reprezentacija entiteta, ideja i događaja zajedno s njihovim svojstvima i odnosima s obzirom na sistem kategorija je ono što je zajedničko ontologiji u računalnoj znanosti i filozofiji. Filozofi su manje zabrinuti oko utvrđivanja fiksnih, kontroliranih pojmova negoli istraživači u računalnim znanostima dok su računalni znanstvenici manje uključeni u raspravu o nekim nužnim početnim principima.

Kao što je i ranije rečena u to doba ontologija ima veliku primjenu i u umjetnoj inteligenciji. U kasnim 1970-im i ranim 1980-im, intenzivni istraživački napori bili su usmjereni u tzv. “ekspertne sustave”. Takvi sustavi su težili predstavljanju znanja neke domene tako da se na pitanja o toj domeni moglo odgovoriti na razini ljudskih stručnjaka. Danas ontologija ima primjenu u mnogim znanstvenim disciplinama među njima je najzanimljivija primjena ontologije u informatici.

3.4 Učenje ontologije

Učenje ontologije (engl. ontology learning) odnosi se na automatsko kreiranje i otkrivanje ontologija pomoću tehnika iz područja strojnog učenja ili otkrivanja znanja (Jurić, 2013). Poluautomatske metode kreiranja ontologija za razliku od manualnih omogućavaju kreiranje ontologija na mnogo široj razini sa znatno većom brzinom.

Jezici za prezentacije ontologija nalaze se između prirodnog jezika koji je razumljiv ljudima i formalnog jezika koji je razumljiv strojevima. Kako bi se taj problem riješio potrebno je i dalje dopustiti ljudima korištenje prirodnog jezika na način da se automatski mapiraju riječi i izrazi iz prirodnog jezika koji je razumljivi ljudima u ontološke koncepte. Znanje pohranjeno u ontologijama je deklarativno, eksplicitno i vrlo strogo. Znanje koje se izražava ljudskih govornim i pisanim jezikom je: implicitno, često nejasno ili čak nerazumljivo.

Kod otkrivanja znanja ontologija predstavlja model kojeg je potrebno prikazati u nekom hipotetskom jeziku (Jurić, 2013). S obzirom na podatke kojima se raspolaže učenje ontologije se može podijeliti na nekoliko zadataka:

- Fokus na učenje koncepata,
- Fokus na učenje relacija između koncepata
- Učenje i koncepata i relacija u isto vrijeme

Metode iz područja otkrivanja znanja usmjerene su na otkrivanje znanja pronalaženjem strukture unutar podataka. Metode iz područja otkrivanja znanja koriste za mapiranje podataka koji su nestrukturirani kao na primjer velike kolekcije dokumenata u ontologiji. Središnji problem u području otkrivanja znanja predstavlja skalabilnost prilikom čega je često potrebno manipulirati velikim količinama podataka. Koraci u ovom procesu su:

- Kreiranje koncepata
- Kreiranje relacija

- Popunjavanje ontologije instancama
- Konstruiranje ontologije
- Proširivanje ontologije novim sadržajem

Kod strojnog učenja tehnike su usmjerene na automatsko prepoznavanje i detekciju određenih uzoraka i pravila unutar nekog skupa podataka. Proces strojnog učenja zasniva se na indukciji, odnosno na donošenju nekih generalnih zaključaka nad nekim skupom podataka. Učenje nad takvim podacima može biti nadzirano ili bez nadzora. S aspekta učenja ontologija, metode iz ovog područja kao izvor imaju najčešće tekstualne dokumente. U području strojnog učenja već je nastao velik broj smislenih tehnika za induktivno učenje iz skupa podataka, ali većina ih je osim par iznimki uglavnom suprotna ideji učenja ontologija.

Tehnike strojnog učenja najprije pokušavaju razviti analitičke modele koji objašnjavaju podatke. Iz tog razloga se te tehnike najčešće koriste za potrebe predviđanja, odnosno za klasifikaciju novih primjera. Većina radova koji koriste tehnike strojnog učenja koriste: segmentiranje, klasificiranje, memorijsko učenje ili induktivno zaključivanje iz uzoraka unutar primjera.

3.4.1 Otvoreni problemi u učenju ontologije

Iako je u učenju ontologije napravljen velik pomak još uvijek postoje problemi koje je bitno razmotriti, u nastavku će biti navedeni osnovni problemi s kojima se susrećemo u učenju ontologije. Neki od problema u učenju ontologija su (Jurić, 2013):

- Učenje specifičnih relacija
- Učenje relacija višeg stupnja
- Učenje definicija
- Filtriranje pojmova
- Kreiranje ontologija visokog stupnja
- Vrednovanje, itd.

Učenje specifičnih relacija bazira se na učenju specifične vrste relacija. Za neke domene jako su važne manje proučavane vrste relacija poput relacije dio-cjelina (Jurić, 2013).

Učenje relacija višeg stupnja bazira se, kako i sam naziv kaže, na relacije višeg stupnja kao što su na primjer relacija povjerenje (Jurić, 2013).

Učenje definicija bazira se na različitim oblicima od identifikacije i ekstrakcije novih definicija, odabiranja alternativnih definicija ili slaganja definicija od dijelova informacija. Za učenje definicija često se koriste relacije poput definira i odnosi se na (Jurić, 2013).

Filtriranje pojmova se odnosi na filtriranje nerelevantnih koncepata unutar teksta (Jurić, 2013).

Kreiranje ontologija visokog stupnja predstavlja proces kreiranja ontologija visokog stupnja iz podataka koje dobijemo korištenjem nekih od metoda učenja ostaje problematičan. Povezivanje lingvističkih entiteta s konceptima u ontologiji može se izvesti ručno ili automatski. Također spajanje i integracija ontologija ostaje i dalje jako zahtjevan zadatak (Jurić, 2013).

Većina ontologija nastala tehnikama učenja ontologija vrednovana je od strane ljudi. Važno je napomenuti da se kod vrednovanja javlja problem pronalaska stručnih ljudi za obavljanje ovog procesa te samim time vrednovanje predstavlja jako zahtjevan posao za čovjeka (Jurić, 2013).

3.5 Ontologija i konceptualne mape

Konceptualne mape (engl. Conceptual Maps) predstavljaju alat za brže i učinkovitije učenje. Zapravo to je tehnika koja pomaže da pojedinac zapamti što više podataka koji mu za nešto služe (Skokandić, 2008).

Velika primjena konceptualnih mapa se nalazi u obrazovanju. U obrazovanju konceptualne mape se koriste da bi učenici (Skokandić, 2008):

- Zapamtili veliku količinu gradiva
- Skratili vrijeme učenja
- Naučeno pohranili u dugoročno pamćenje

Konceptualne mape se još nazivaju i kognitivne mape. Utemeljitelj modernih konceptualnih mapa je profesor Tony Buzan koji je proučavao načine učenja svojih studenata. Zaključio je da učenici ne mogu dovoljno dobro učiti i naučiti samo iz svojih bilješki nego da bi konceptualne mape omogućile bolje i lakše učenje velikih količina gradiva. Konceptualnim mapama se potiče fotografsko pamćenje i vizualno razmišljanje jer na taj način se može postići da određeni podatak možemo zapamtiti gdje se nalazi i koji je podatak. U suprotnom se možemo sjetiti samo mjesta gdje se podatak nalazi ali ne i koji je podatak, što se lako može i provjeriti ako pojedinac napravi osobni pokus na način da određene podatke pamti na tradicionalni način da „buba ili uči na pamet“ gradivo i na način da koristi konceptualne mape. Kod prvog načina se može dogoditi da znamo mjesto gdje se sadržaj nalazi ali ga se ne možemo sjetiti, a kod konceptualnih mapa učenik se prisjeća podatka preko slika, zabilješki i boja. Vrlo je bitno napomenuti da sav tekst, riječi, rečenice koje su moguće, zamijenimo simbolima ili slikama.

Postoje dvije vrste konceptualnih mapa (Skokandić, 2008):

- Računalne konceptualne mape
- Ručne konceptualne mape

Kod stvaranja ručnih konceptualnih mapa potreban je samo: papir, olovka, boje i malo dobre volje za to sve nacrtati (Slika 6).

Kod računalnih konceptualnih mapa potrebno je računalo i alat koji omogućava crtanje konceptualnih mapa, dok umijeće u crtanju nije potrebno (Slika 7).

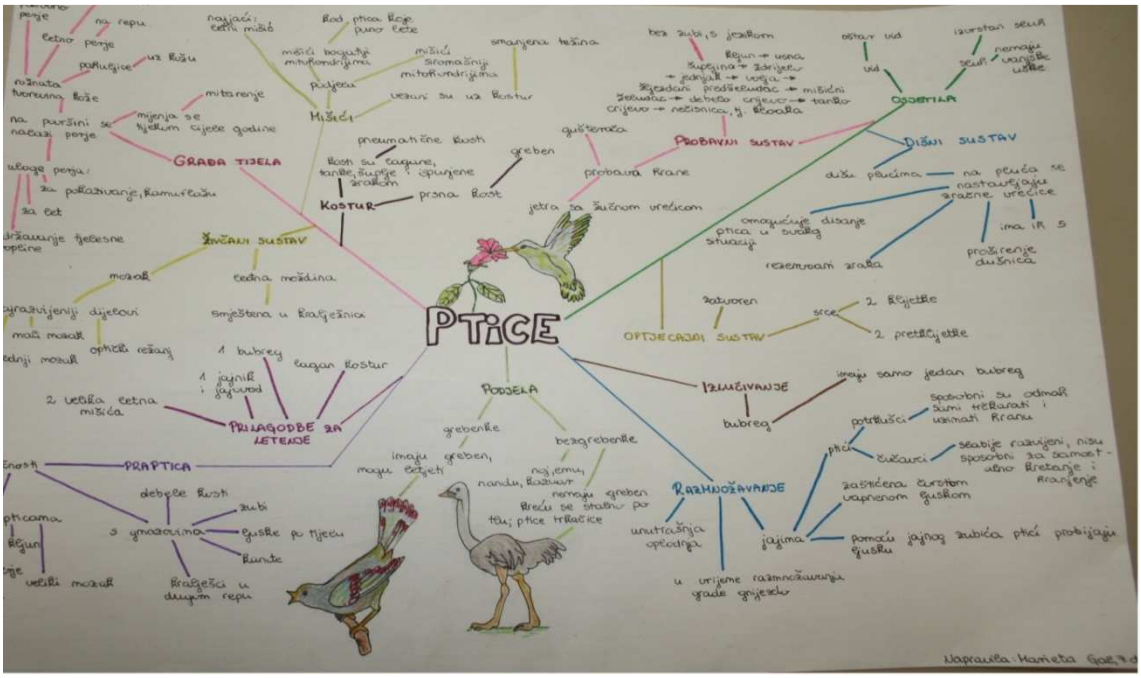
Prilikom izrade konceptualnih mapa potrebno se pridržavati sljedećih pravila (Skokandić, 2008):

- Početak ide od sredine i crta se crtež koji karakterizira temu na kojoj se radi
- Korištenje različitih simbola i veličina slova u cijelom dijagramu
- Odrediti i upisati ključne riječi
- Koristiti različite boje

- Razvijati vlastiti stil izrade umne mape
- Naglasiti najvažnije dijelove i koristiti asocijacije
- Svaka ključna riječ je samostalno zapisana

Prilikom izrade konceptualnih mapa postoje stvari koje treba izbjegavati kao što su (Skokandić, 2008):

- Previše boja(najmanje tri, najviše pet)
- Točke među linija
- Nejasna središnja tema
- Teme udaljene od riječi na koje se odnose
- Slova u različitim bojama



Slika 6: Primjer konceptualne mape nacrtane rukom (http://os-gjuro-ester-koprivnica.skole.hr/upload/os-gjuro-ester-koprivnica/images/static3/1405/attachment/primjer_umne_mape_2.JPG)

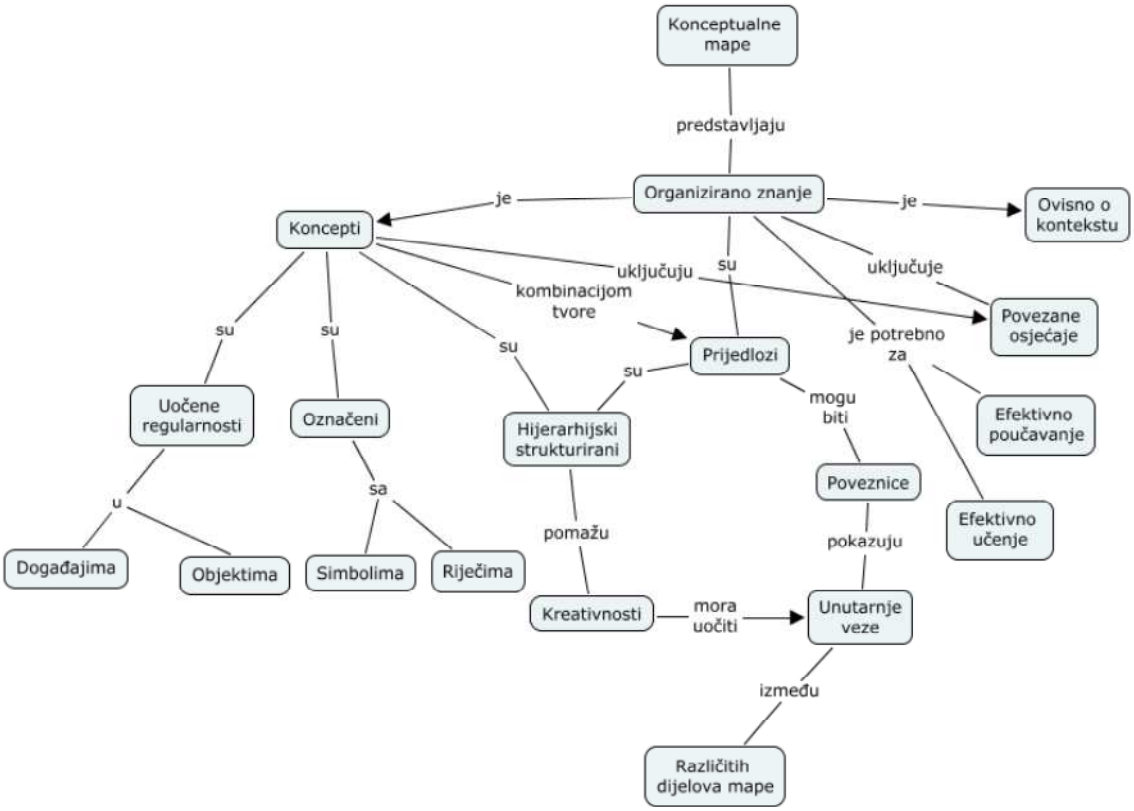
Učiteljima u nastavi ovo je odličan alat za:

- Mjesečno planiranje nastave
- Obradu problematičnih učenika
- Vođenje bilježaka

Pri izradi konceptualnih mapa u razredu većinom se koristi „Oluja ideja“ to jest stvaranje ideja i kreiranja konceptualnih mapa od strane grupe učenika gdje se ideje pojedinca ne kritiziraju već se unaprjeđuju, a cilj je istaknuti što veći broj ideja bez obzira na njihovu valjanost.

Glavna razlika između konceptualne mape i ontologije je to što konceptualna mapa odražava razmišljanje o jednom predmetu razgovora, a ontologije se sastoje od niza međusobno povezanih koncepata na temelju čega se može zaključiti kako za izradu ontologije treba izraditi barem onoliko konceptualnih mapa koliko ontologija ima različitih koncepata, a veze između konceptualnih mapa predstavljat će veze između koncepata u ontologiji (Prgomet, 2011). Konceptualne mape se najčešće izrađuju u nekom od alata predviđenih za to.

Sličnost konceptualne mape i ontologije je što oboje služe za strukturirano predstavljanje znanja. Konceptualna mapa koja nije razumljiva računalu ne može predstavljati ontologiju (Prgomet, 2011). Osim koncepta konceptualna mapa sadrži i relacije koje definiraju odnos među konceptima. Međutim, konceptualna mapa se organizira oko jednog pojma pa je možemo interpretirati kao modul ontologije koji je također ontologija te može predstavljati drugačiji pogled na sustav.

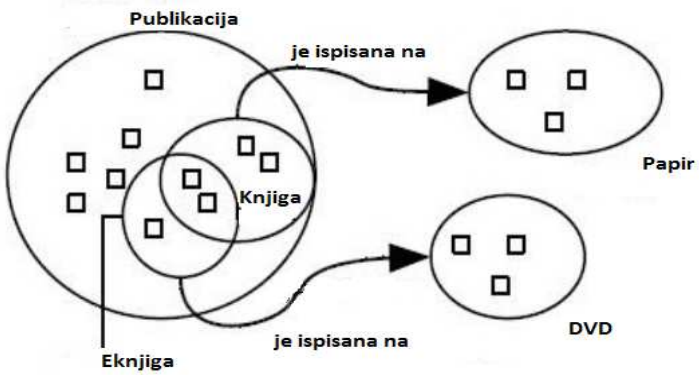


Slika 7: Primjer konceptualne mape kreirane programskim alatom (Prgomet, 2011)

3.6 OWL

Postoje mnogi ontološki jezici ali u narednom poglavlju će biti opisan OWL jezik. OWL (Web Ontology Language) je ontološki jezik namijenjen opisu pojmova i odnosa između pojmova te opisivanje ontologija na web-u, a to znači da informacije koje se nalaze u dokumentima su obradive od strane aplikacija i sadržaj je razumljiv korisnicima (Prcela, 2008). OWL je nastao iz ideje o semantičkom webu. OWL je izveden iz DAML i OIL ontoloških jezika. Informacije opisane ovim jezikom postaju znanje. OWL ima mogućnost izvođenja činjenica koje nisu eksplicitno navedene. OWL se koristi kao deklarativni jezik u predstavljanju znanja raznih domena primjene kao što su (Slika 8):

- Medicina
- Molekularna biologija
- Programsko inženjerstvo
- Konfiguracija složenih sustava, itd.



Slika 8: Primjena OWL-a

U OWL-u moguće je:

- Definirati sinonime za objekte, klase i svojstva
- Definirati karakteristike svojstava kao što su: tranzitivnost, simetričnost, inverzno svojstvo
- Definirati ograničenja na vrijednosti koje neko svojstvo može imati
- Definirati nove klase primjenom skupovnih operacija nad postojećim klasama.

U razvoju OWL-a moraju se uzeti obzir pretpostavke o različitim ontologijama i dozvola za kombiniranjem različitih ontologija. Može se dogoditi da neke od kombiniranih ontologija mogu biti proturječne, ali nove informacije ne smiju nikada opozivati postojeće, jedino se mogu dodavati. Kako bi se omogućila takva kompatibilnost OWL uvodi tri pod jezika za različite namjene (Prcela, 2008):

- OWL Full
- OWL DL
- OWL Lite

Potpuni pod jezik ili OWL Full upotrebljava osnovni OWL jezik koji dozvoljava njegovu kombinaciju na proizvoljan način s RDF-om i RDF Schemom. Prednost mu je njegova sintaktička i semantička kompatibilnost s RDF-om. Svaki legalni RDF dokument je također legalni OWL Full dokument. Svaki valjani RDF/RDF Schema zaključak je također valjani OWL Full zaključak ((Prcela, 2008), (Paunovic & Stokic, 2012)).

OWL DL (deskriptivna logika) je pod jezik od OWL Full-a. Ovaj pod jezik posjeduje neka ograničenja u vezi upotrebe konstrukata iz OWL-a i RDF-a. Prednost je osiguranje efikasne razumijevajuće podrške. Nedostatak je gubljenje potpune kompatibilnosti s RDF-om. RDF dokument se treba proširiti na određeni način i ograničiti na neki drugi način prije nego što bude legalni OWL DL dokument. Svaki legalni OWL DL dokument je legalni RDF dokument ((Prcela, 2008), (Paunovic & Stokic, 2012)).

OWL Lite je uglavnom namijenjen podržavanju klasifikacijske hijerarhije i jednostavnih ograničenja. Ovaj jezik je dobra polazna osnova za pravljenje ontoloških alata. OWL Full je proširenje OWL DL, koji je proširenje OWL Lite. Svaka OWL Lite ontologija je OWL DL i OWL Full ontologija. Svaka OWL DL ontologija je i OWL Full ontologija ((Prcela, 2008) , (Paunovic & Stokic, 2012)).

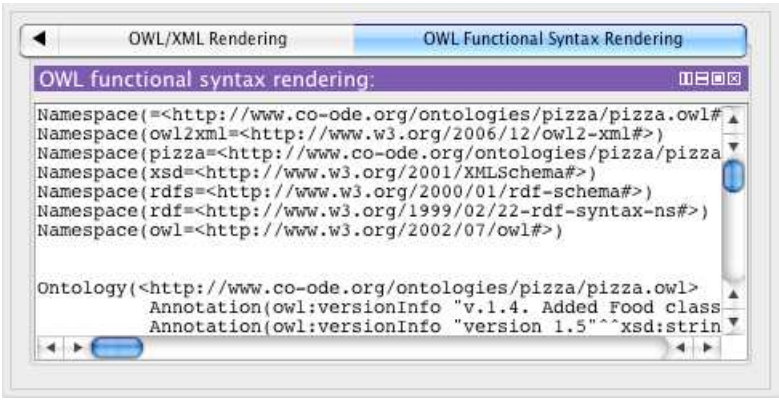
3.6.1 Opis OWL jezika

Što se tiče sintakse OWL jezika je izgrađen na RDF-u i RDF shemi te koristi RDF-ovu XML-baziranu sintaksu (Slika 9).

RDF/XML ne osigurava čitljivu sintaksu zbog čega su druge sintaksne forme također definirane za RDF:

- XML temeljenu sintaksu koja ne slijedi RDF konvencije i čime se osigurava veća čitljivost korisnicima
- Saženu sintaksu, koja je više kompaktna i čitljiva nego XML sintaksa ili RDF/XML sintaksa
- Grafičku sintaksu temeljenu na UML (Unified Modeling Language) konvenciji, koja je široko upotrebljavanja i predstavlja jednostavan način kojim korisnici postaju upoznati s OWL-om

Postoji apstraktna sintaksa koja je jednostavnija i više prilagođena ljudima, a moguće ju je koristiti za OWL DL i OWL Lite. XML/RDF sintaksa je službena OWL sintaksa namijenjena strojnoj obradi i može se koristiti za sve tri inačice OWL-a.



Slika 9: Prikaz OWL sintakse

Što se tiče zaglavlja (Slika 10) važno je napomenuti da su OWL dokumenti koji se nazivaju i OWL ontologije ujedno i RDF dokumenti (Pavic, 2013). Osnovni element u takvim dokumentima je *rdf:RDF* koji je također i početni element OWL dokumenta.

```
<rdf:RDF
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

Slika 10: OWL zaglavlje

OWL dokument započinje i nizom izjava koje služe za održavanje dokumenta. Te izjave su grupirane unutar *owl:Ontology* elementa koji sadrži komentare, oznaku verzije i uključivanje drugih ontologija (Slika 11).

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion
        rdf:resource="http://www.mydomain.org/uni-ns-old"/>
  <owl:imports
        rdf:resource="http://www.mydomain.org/persons"/>
  <rdfs:label>University Ontology</rdfs:label>
</owl:Ontology>
```

Slika 11: Prošireno zaglavlje OWL-a

Važan dio zaglavlja su i informacija o verziji ontologije odnosno njegovoj kompatibilnosti s prethodnim verzijama ontologije koje prikazuju sljedeći elementi:

- owl:versionInfo – općenita informacija o verziji (bilo brojčana, bilo znakovna)
- owl:backwardCompatibleWith – ova izjava definira prethodnu verziju s kojom je ontologija kompatibilna
- owl:incompatibleWith – označava da nije kompatibilna s nekom prethodnom verzijom

OWL klasa se definira korištenjem *owl:Class* elementa (Pavic, 2013). Kod OWL-a se podrazumijeva postojanje dvije specijalne klase:

- owl:Thing - najopćenitija klasa (sve je stvar) - sve klase su pod klase ove klase
- owl:Nothing - prazna klasa- svaka klasa je nad klasa ove klase

Pri definiranju proizvoljne klase moguće je koristiti dodatne elemente kojima se pobliže određuje klasa. Dodatni elementi proizvoljne klase su:

- owl:disjointWith - definira disjunkciju dviju klasa
- owl:equivalentClass - definira ekvivalenciju dviju klasa

OWL razlikuje dva tipa svojstava (Pavic, 2013):

- svojstvo objekata
- svojstvo tipa podataka (datatype)

OWL ne posjeduje predefinirane tipove podataka (cjelobrojni, float, niz znakova,...). OWL ne pruža specifičan način definiranja predefinirani tipova podataka nego dopušta korištenje redefiniranih tipova iz XML Scheme odnosno njegovih tipova podataka. Jedna od posebnosti OWL su podržane restrikcije nad svojstvima. Restrikcija se postiže uvođenjem elementa *owl:Restriction* koji se obavezno sastoji od elemenata :

- owl:onProperty – svojstvo koje se ograničava
- jedan od elemenata koje definiraju ograničavanje

Elementi koji definiraju ograničavanje mogu biti :

- owl:hasValue - posjeduje neku određenu vrijednost
- owl:allValuesFrom - posjeduje sve vrijednosti
- owl:someValuesFrom - posjeduje neke vrijednosti
- owl:minCardinality i/ili owl:maxCardinality (owl:cardinality)

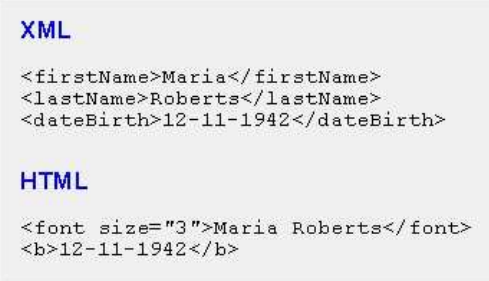
Enumeracija u OWL-u je definirana *owl:oneOf* elementom koji se koristi pri definiranju klase kao niz svojih elementa.

Iako je OWL jezik jako važan za rad s ontologijama, jedan od neizostavnih jezika kad se govori o ontologiji je XML o kojem će biti riječi u narednom poglavlju.

3.7 XML

XML je kratica za eXtensible Markup Language, odnosno jezik za označavanje podataka (Harold & Means, 2006). EXtensible označava da je jezik proširljiv de dozvoljava definiranje novih oznaka. Markup označava dodavanje specijalnog značenja podatku. Ovaj jezik definira sintaksu za označavanje podataka jednostavnim, svima razumljivim oznakama (engl. *tags*). XML bi kao jezik trebao biti razumljiv i ljudima i računalima.

Princip realizacije je vrlo jednostavan: odgovarajući sadržaj treba se uokviriti odgovarajućim oznakama koje ga opisuju i imaju poznato ili lako shvatljivo značenje. XML i njegova struktura je vrlo slična HTML-u, samo što XML oznaka definira podatak dok HTML oznaka definira izgled stranice što se vidi na (Slika 12).



Slika 12: Razlika XML-a i HTML-a

Danas je XML jezik vrlo raširen i koristi se za različite namjene (Harold & Means, 2006):

- Odvajanje podataka od prezentacije
- Razmjenu podataka
- Pohranu podataka
- Povećavanje dostupnosti podataka
- Izradu novih specijaliziranih jezika za označavanje
- Vektorska grafika
- Pozivanje udaljenih funkcija, itd.

XML omogućava pisanje vlastitih programa za rad s podacima u XML dokumentima i njihovu obradu. Ako sami radite program, koristite i vlastite oznake koje opisuju točno ono što je vama potrebno za rad. Za rad sa XML dokumentima možete upotrijebiti i gotov alat, kao što su čitači Weba (engl. Web browsers) ili programi za obradu teksta. Jedan dio alata može raditi sa svim XML dokumentima dok su drugi prilagođeni za podršku određenoj XML aplikaciji.

Što se tiče povijesti XML-a, važno je reći da je nastao iz različitih jezika pa tako 60-tih godina 20. stoljeća istraživači IBM-a počinju razvijati Generalized Markup Language (GML). Do GML-a se dolazi zbog problema s ogromnom količinom različite tehničke dokumentacije koja se za svaku posebnu

namjenu morala prepisivati i nanovo uređivati. Ideja je bila načiniti prvi šire korišteni jezik za označavanje podataka.

Kako se GML pokazao kao uspješan proizvod, razvoj u tom smjeru je nastavljen. 1986. objavljen je Standard Generalized Markup Language (SGML) (Harold & Means, 2006). Najveći nedostatak SGML-a je njegova veličina. Nastali jezik bio je opširan, složen za korištenje te skup u upotrebi. Razlog tome je što su autori pokušali pokriti svaku moguću primjenu jezika. Zbog tih osobina SGML nije bio raširen u praksi .

Iz SGML-a nastao je HTML (HyperText Markup Language). Za njegov razvoj izabran je jedan manji skup oznaka iz SGML skupa koji je bio primijenjen na formatiranje dokumenta. HTML je imao skup oznaka koje su opisivale osnovne dijelove dokumenta, a programi koji su tumačili strukturu takvih dokumenata bili su HTML preglednici. Nedostatak mu je mala količina oznaka koja nije bila dovoljna za neku veću upotrebu. Zbog potreba i želja tržišta te razvoja tehnologija proširivan je nestandardnim oznakama koje su služile za formatiranje sadržaja u smislu njegovog prikaza u internet pregledniku.

XML se razvija početkom 90-tih godina 20.stoljeća od strane World Wide Web Consortium-a. XML je zapravo jezik koji objedinjuje jednostavnost HTML-a i izražajnu snagu SGML-a. Razvojni tim je bio vođen nekim ciljevima kakav bi XML trebao biti, a to su:

- XML mora biti izravno primjenjiv preko interneta
- XML mora podržavati širok spektar primjena
- XML mora biti kompatibilan s SGML-om
- Mora biti lako pisati programe koji procesiraju (parsiraju) XML dokumente
- Broj opcionalnih značajki u XML-u mora biti apsolutno minimalan, u idealnom slučaju jednak nuli
- XML dokumenti moraju biti čitljivi ljudima, te u razumnoj mjeri jednostavni
- Standard mora biti specificiran što prije
- Dizajn XML-a mora biti formalan i precizan
- Kreiranje XML dokumenata mora biti jednostavno
- Sažetost kod označavanja dokumenta XML-om je od minimalnog značaja

Xml dokument se sastoji od (Harold & Means, 2006):

- Zaglavlja
- Sadržaja dokumenta

Primjer xml dokumenta moguće je vidjeti na (Slika 13).

```
<?xml version="1.0"?>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Slika 13: Primjer XML dokumenta

Zaglavlje (Slika 14) se nalazi na početku xml dokumenta i korisniku omogućava da vidi informacije o samom xml-u, to jest koja je verzija xml-a i standard po kojem je xml napravljen.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Slika 14: Primjer zaglavlja XML dokumenta

U sadržaju (Slika 15) xml dokumenta nalazi se sadržaj koji je omeđen xml oznakama. Svaki XML dokument mora imati jedan korijenski ili *root* element koji uokviruje kompletan sadržaj. Taj element opisuje XML dokument. Unutar korijenskog elementa ugniježđeni su svi ostali.

```
<poruka>
        ...
</poruka>
```

Slika 15: Prikaz sadržaja XML dokumenta

XML elementi mogu biti u odnosu (Harold & Means, 2006):

- roditelj-dijete (engl. parent-child)
- sestrinskom (engl. siblings)

Kod odnosa roditelj dijete (Slika 16) jedan je element hijerarhijski nadređen drugom.

```
<poruka>
 ...
 <naslov>Podsjetnik</naslov>
 ...
</poruka>
```

Slika 16: Roditelj-dijete odnos

U ovom primjeru element poruka je roditelj elementu naslov (nadređen je elementu naslov). Kaže se da je element naslov ugniježđen unutar elementa poruka. U XML-u elementi moraju biti pravilno ugniježđeni, što bi značilo da ne smije doći do preklapanja oznaka kao što je to vidljivo na (Slika 17).

```
<poruka>
 ...
 <naslov>Podsjetnik</poruka>
 ...
</naslov>
```

Slika 17: Primjer nepravilno ugniježđenih elemenata

Kod sestrinskog odnosa (Slika 18) elementi se nalaze na istoj razini. U primjeru ispod elementi: naslov i tijelo se nalaze na istoj razini.

```
<naslov>Podsjetnik</naslov>
<tijelo>Otiđi kupiti kruh</tijelo>
```

Slika 18: Primjer sestrinskog odnosa elemenata

Nazivi elemenata (Harold & Means, 2006):

- Mogu sadržavati: brojeve, slova i posebne znakove
- Moraju počinjati slovima
- Ne smiju sadržavati praznine
- Treba izbjegavati korištenje znakova kao što su -,/…
- Ne smiju počinjati tekstom xml ili XML

XML atributi nude podatke koji dodatno opisuju XML elemente. Podatci koji se navode kao atributi mogu biti zanimljivi čovjeku ili računalnom programu koji parsira XML dokument. Atributi imaju svoj naziv i vrijednost koja se navodi pod navodnicima (Slika 19).

```
<poruka>
        <za font_face="Arial" font_size="10">Pero</za>
        <od font_face="Arial" font_size="10">Kate</od>
        <naslov font_face="Arial" font_size="14">Podsjetnik</naslov>
        <tijelo font_face="Arial" font_size="10">Otidi kupiti kruh</tijelo>
</poruka>
```

Slika 19: Prikaz XML atributa

Prilikom rada u XML dokumentu moraju se poštivati i određena sintaksna pravila kao što su (Harold & Means, 2006):

- Svaki XML element mora imati početnu i završnu oznaku
- Svi elementi moraju biti pravilno ugniježđeni
- XML oznake su osjetljive na velika i mala slova
- Svaki XML dokument mora imati korijenski element koji uokviruje cijeli sadržaj
- Vrijednosti atributa se navodi pod navodnicima
- Komentari se prikazuju pomoću posebne oznake
- Svaki xml dokument počinje zaglavljem u kojem je navedena verzija XML-a i kodna stranica
- Znakovi koji u XML-u imaju posebno značenje u tekstu se moraju prikazivati na poseban način, znakovnim ili numeričkim kodovima koji na početku imaju znak & (engl. ampersand), a na kraju točku-zarez

Prednosti XML-a (Harold & Means, 2006):

- Jednostavno je čitljiv čovjeku i računalu
- XML dokument je obična tekstualna datoteka čitljiva na svakoj platformi koja može čitati tekstualne podatke
- Podržava Unicode i omogućuje prikaz teksta na svim danas poznatim jezicima
- Format je samodokumentirajući. Oznake opisuju sadržaj koji se nalazi unutar njih.
- Ima stroga sintaksna pravila tako da je jednostavno kontrolirati ispravnost nastalog dokumenta.
- Međunarodno prihvaćen standard. Mnogi proizvođači programa su ga prihvatili i koriste u svojim proizvodima
- Hijerarhijska struktura je pogodna za opisivanje mnogih sadržaja
- Kompatibilan je sa SGML-om

Nedostaci XML-a (Harold & Means, 2006):

- Sintaksa je redundantna i opširna što može zamarati i zbunjivati osobu koja čita XML dokument.
- Zbog opširnosti računalni program koji obrađuje dokument mora obraditi veliku količinu podataka što će ga djelomično usporiti
- Da bi dokument bio dovoljno dobro "samoopisan" nazivi oznaka moraju biti dovoljno precizni što dovodi do dugih naziva
- Redundancija i velika količina podataka stvaraju velike zahtjeve za propusnosti mreže
- Programi koji obrađuju XML podatke su dosta složeni jer moraju obrađivati velike količine ugniježđenih podataka u više razina
- Nedostatak formalno propisanih formata za podatke može stvarati probleme ako sudionici u razmjeni nisu dobro opisali (npr. da li se decimalni brojevi prikazuju s decimalnom točkom ili zarezom)

Postoje dvije verzije XML-a (Harold & Means, 2006):

- XML 1.0
- XML 1.1

XML 1.0 inicijalno je kreirana 1998. godine. Do danas je imala nekoliko malih promjena. Širom je prihvaćena os strane korisnika u svijetu te se još i danas preporučuje za korištenje. Bazira se na filozofiji da je zabranjeno sve što nije dozvoljeno.

XML 1.1 inicijalno je objavljena 2004. godine i ima određena svojstva koja olakšavaju rad programima na velikim računalima. Njezin pristup je da je dozvoljeno sve što nije zabranjeno. Na takav način omogućuje se korištenje svih budućih Unicode znakova koji će se bilo kada u budućnosti definirati. Zbog svoje raširenosti uglavnom se još uvijek koristi verzija XML 1.0 jer je zadovoljavajuća za većinu korisnika. Korisnici koji imaju problema s ograničenjima verzije XML 1.0 uglavnom su već prešli na korištenje verzije XML 1.1.

3.8 Alati za konstruiranje ontologija

Brzi razvitak ontologije doveo je jednako tako i do brzog razvoja alata za kreiranje ontologija, u sljedećem poglavlju će biti riječi o alatima za kreiranje ontologije. Alati za kreiranje ontologije moraju osigurati sljedeće zahtjeve:

- Izvlačenje znanja to jest prikupljanje znanja od korisnika
- Odziv: mogućnost pretrage ontologija na temelju pojmova, relacija, sredstava od strane korisnika
- Uređivanje
- Vrednovanje

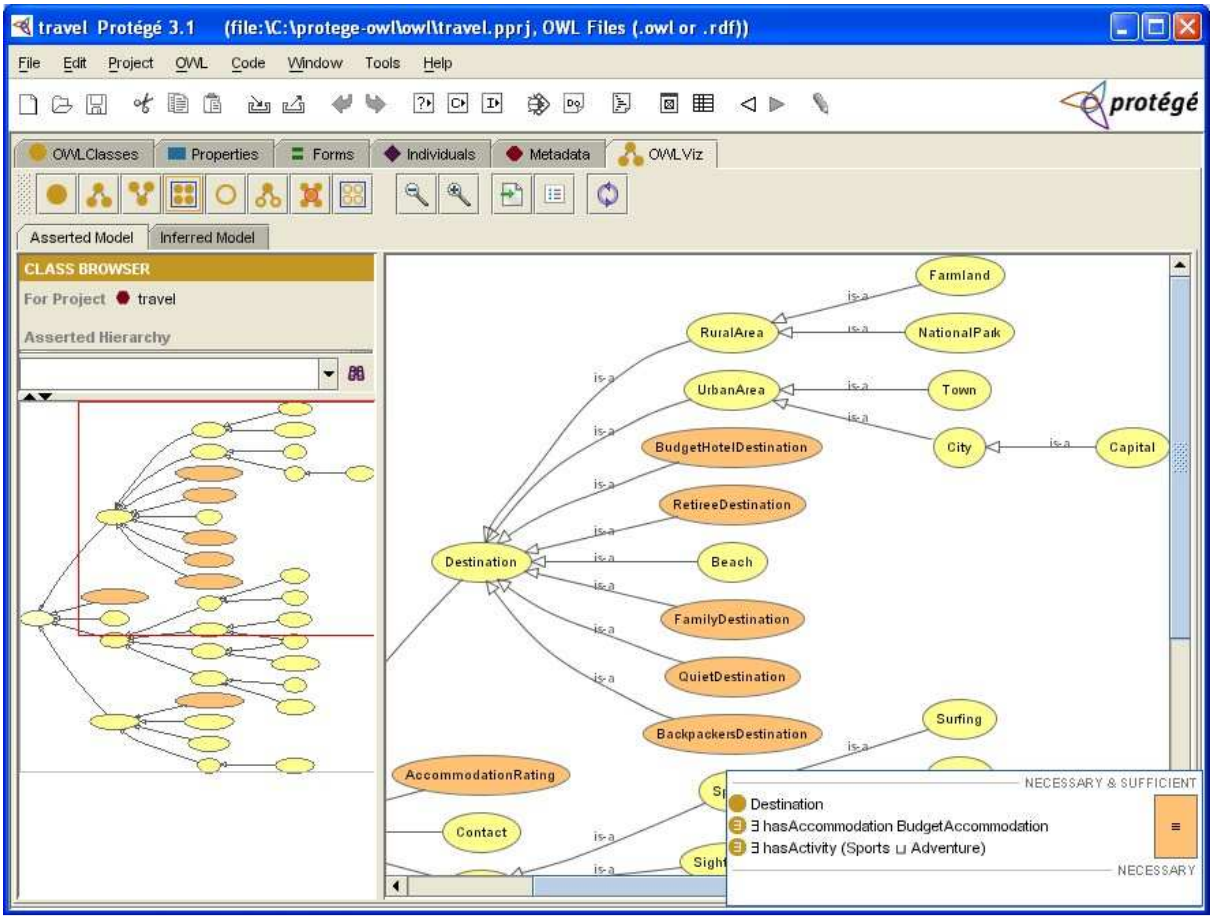
U nastavku će biti opisani sljedeći alati:

- Protégé
- Altova SemanticWorks
- SMORE/SWOOP

- TopBraid Composer

3.8.1 Protégé

Protégé (http://protege.stanford.edu/) je besplatan, alat otvorenog koda za konstrukciju ontologija te rad s njima. Podržava dva bitna načina modeliranja ontologija putem Protégé-Frames i Protégé-OWL editora (Slika 20). Ontologije stvorene u Protégé-u se mogu pohraniti u većem broju formata kao što su RDF(S), OWL te XML formati. Pisan je u Javi. Budući da je pisan u Javi, prenosiv je i može se izvršavati na svim računalima koja imaju Java Virtual Machine. Podržan je od mnogobrojne zajednice raznovrsnih korisnika. Koristi se za rješavanje problema povezanim sa znanjem u područjima kao što su biomedicina, obavještajni poslovi te korporacijsko modeliranje. Nedavno je zabilježeno više od 100000 registriranih Protégé korisnika.

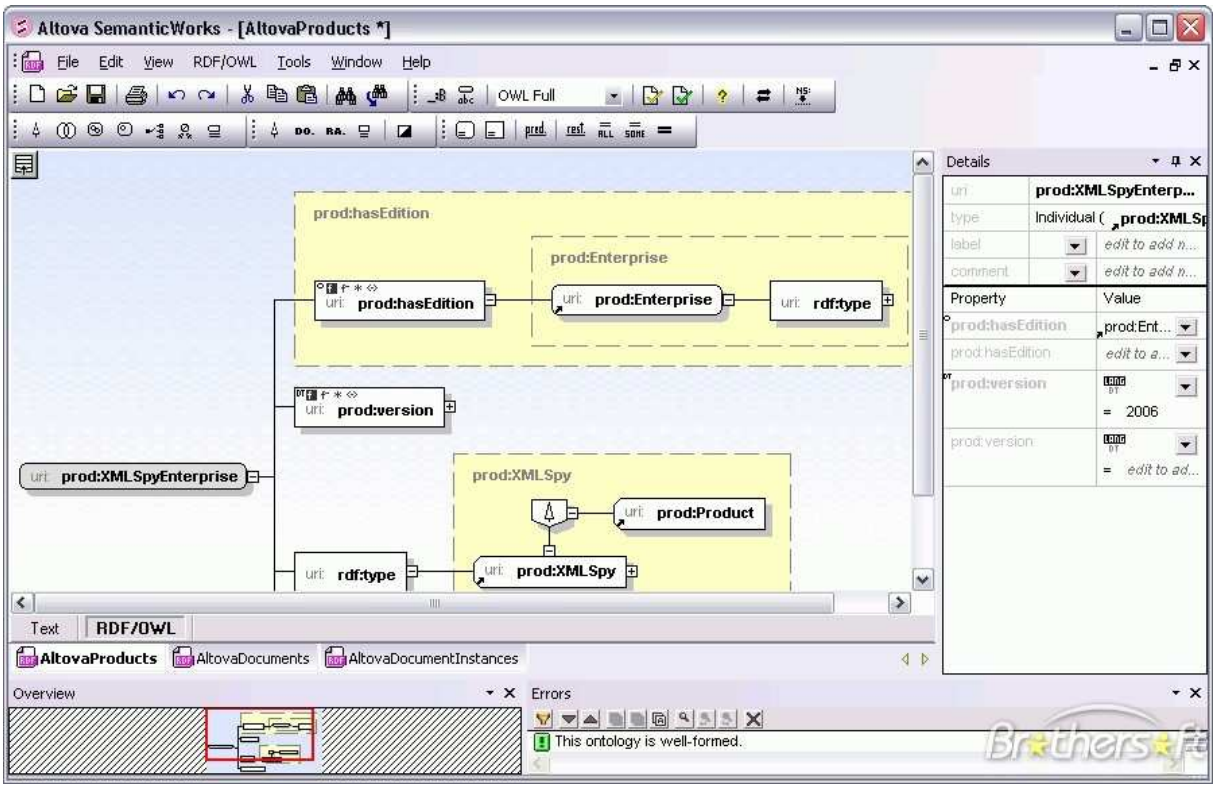


Slika 20: Izgled korisničkog sučelja Protégé-a

3.8.2 Altova SemanticWorks

Altova SemanticWorks (http://www.altova.com/products/semanticworks/rdf_owl_editor.html) je alat koji omogućava veliku fleksibilnost kod stvaranja i uređivanja ontologije. Trenutno se smatra jednim od najboljih ontoloških alata. Alat omogućava korisniku pronaći bilo koju grešku u radu te ju je jednostavno ispraviti, alat također omogućava kreiranje kompleksnih ontologija te njihov vizualni

prikaz koristeći napredni sustav pomoći te intuitivan sustav kratica i ikona (Slika 21). Alat je primjeren i za početnike koji posjeduju barem osnovno znanje o ontologijama. Što se izbornika tiče sličan je Protégé-u, provjera sintaksne i semantičke ispravnosti je jednostavna, također ga odlikuje i bogat repozitorij pomoći i uputa koji omogućavaju još brže snalaženje u ovom alatu.



Slika 21: Izgled korisničkog sučelja Altova SemanticWorks-a

3.8.3 SMORE / SWOOP

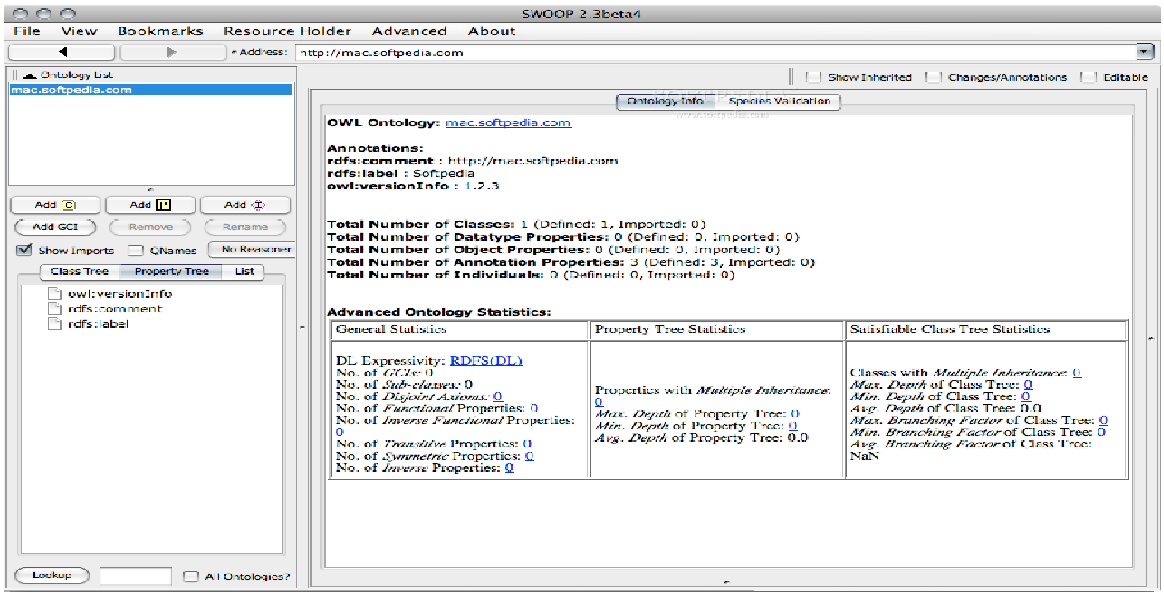
SMORE i SWOOP (https://code.google.com/p/swoop/) alati su alati za uređivanje ontologija koji korisnicima dopuštaju kreiranje i uređivanje ontologija koristeći jednostavno sučelje (Slika 22). I jedan i drugi alat su slični samo što SMORE ima integriranu komponentu web preglednika. Ova funkcija omogućava korisniku pristup internetu preko alata te samim time kreiranje ontologije na webu. SWOOP nudi i bogatiji skup opcija kao što su :

- Ispravljanje pogrešaka
- Automatsko particiranje
- Paletu upita

Također je važno napomenuti da je alat otvorenog koda kojeg korisnici mogu nadograđivati po svojim potrebama. SMORE i SWOOP su efikasni alati koji omogućavaju korisniku:

- Opis strukture
- Pregled izvornog koda
- Kornjača pregled
- Strukturirani pregled klasa

Što se tiče korištenja ovi alati su malo zahtjevniji za korištenje zbog nedostatka u dizajnu korisničkog sučelja i uputa.



Slika 22: Izgled SWOOP korisničkog sučelja

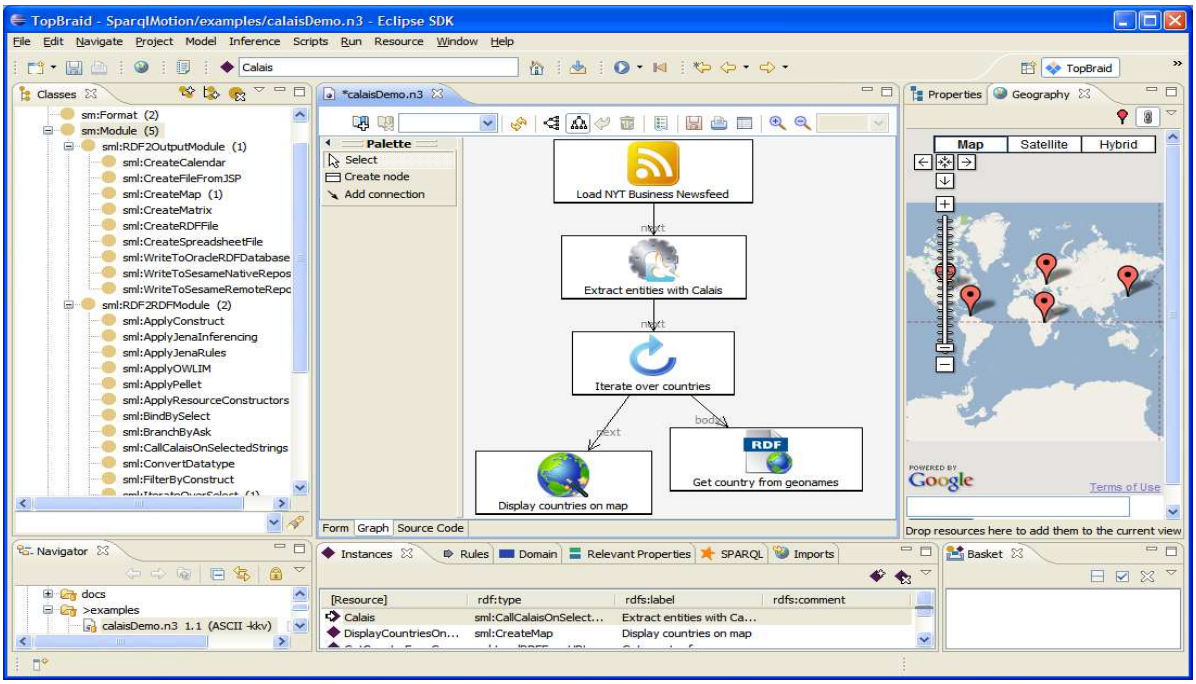
3.8.4 TopBraid Composer

TopBraid Composer (http://www.topquadrant.com/products/TB_Composer.html) je najmoćniji alat za modeliranje i izgradnju semantičkih aplikacija. Potpuno je u skladu s W3C standardima te nudi potpunu podršku za: razvoj, upravljanje i testiranje ontologije i povezanih podataka. TopBraid Composer (Slika 23) pruža podršku za razvoj semantičkih rješenja koja se mogu integrirati u različite aplikacije i izvore podataka. Također se koristi za:

- Razvoj ontoloških modela
- Konfiguriranje izvornih podataka
- Stvaranje semantičkih web usluga
- Kreiranje korisničkih sučelja

Alat pruža opsežnu pomoć i upute koji pokrivaju teme kao što su:

- Uređivanje ontologije i RDF podataka
- Razvijanje semantičkih web aplikacija
- Alat za razvoj aplikacija
- Rad s XML datotekama i tablicama
- Razvoj Web podataka
- Integracija podataka



Slika 23: Izgled korisničkog sučelja TopBraid-a

4 Programski modul za oblikovanje ontološki utemeljenog područnog znanja

Unos područnog znanja u inteligentni tutorski sustav u ovom slučaju AC-ware Tutor kao i velika vremenska zahtjevnost nameće se kao jedan od problema čije rješenje bi uvelike pomoglo onima koji se tim problemom susreću. Kao rješenje ovoga programa nameće se potreba za korištenjem nekoliko programskih alata pomoću kojih bi se ovaj proces automatizirao te samim time skratilo vrijeme unosa područnog znanja u sustav. Važno je napomenuti da se ovaj proces za sada ne može u potpunosti automatizirati upravo zbog potrebe za korištenjem različitih alata ali se u mnogome može ubrzati. Neki od programskih alata koji se mogu koristiti da bi se ovaj problem riješio su: alat za kreiranje područnog znanja CmapTools, C# kao alat u kojem će se realizirati programski modul, SQL u kojem je napravljena baza sustava AC-ware Tutor-a. Da bi se proces unosa područnog znanja olakšao i ubrzao potrebno je:

- Osigurati spajanje na xml datoteku
- Izdvojiti podatke iz xml datoteke
- Sprema podataka u bazu podataka

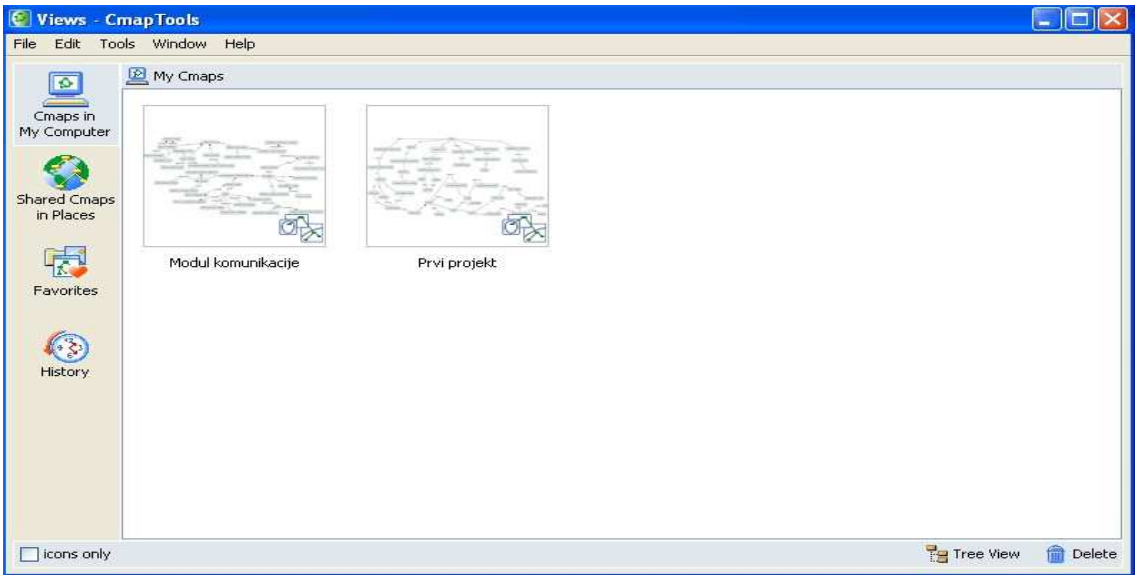
Spajanje na xml datoteku je potrebno iz razloga jer se na taj način omogućava brzo iščitavanje područnog znanja koje je kreirano u CmapTools-u. Nakon spajanja na xml datoteku potrebno je izdvojiti sve podate koje tvore područno znanje iz razloga. Nakon izdvajanja podataka potrebno je te izdvojene podatke unijeti u bazu područnog znanja AC-ware Tutor-a . Realizacija ovakvog projekta je zahtjevna i nije izvediva odjednom te je rad na ovom projektu bilo potrebno podijeliti u nekoliko manjih koraka koji omogućavaju lakši rad i bržu realizaciju projekta. Koraci u realizaciji projekta su sljedeći:

- Izraditi konceptualnu mapu u nekom od alata koji to omogućavaju
- Exportati konceptualnu mapu napravljenu u CmapTools-u u XML datoteku
- Spojiti se na XML datoteku
- Učitavanje podataka iz XML-datoteke
- Izdvajanje duplih podataka
- Spajanje na bazu podataka sustava
- Spremanje podataka u bazu

Za oblikovanje područnog znanja korišten je programski alat CmapTools. Korištenje ovog alata se odabralo zbog jednostavnosti korištenja alata te posjedovanje sučelja koje je vrlo dobro dizajnirano i lako za snalaženje. Jednostavnost rada i snalaženje prilikom korištenja alata omogućava i najvećem laiku da se snađe u ovom alatu, važno je napomenuti da alat pruža korisniku sve što mu treba za brzo i jednostavno kreiranje područnog znanja. Još jedna prednost ovog alata i razlog korištenja je brzo exportanje područnog znanja koje je grafički prikazano u xml datoteku, bez koje nastavak rada ne bi bio moguć.

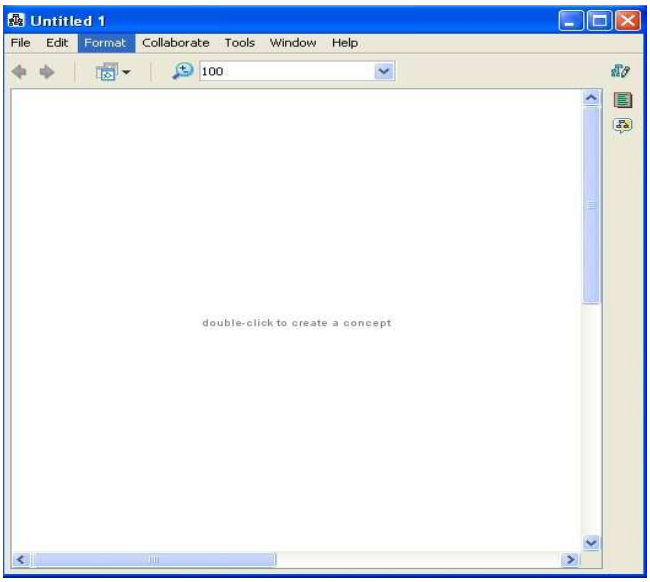
IHMC CmapTools (NOVAK & CAÑAS, 2006) je alat razvijen na IHMC institutu (Florida Institute for Human and Machine Cognition) čija područja djelatnosti obuhvaćaju prezentaciju znanja u formi pogodnoj za računalnu obradu, istraživanje i razvijanje standarda za razvoj semantičkog Interneta -

Interneta koji pruža bogatiji i bolje povezani sadržaj. IHMC CmapTools služi za izradu konceptualnih mapa u kojima je svaki entitet u izrađenoj konceptualnoj mapi moguće dodatno pojasniti podatkovnom poveznicom na novu konceptualnu mapu, video, sliku ili dokument. Spremanje dokumenata je moguće na dva načina. Prvi način je da se dokument spremi na vlastito računalo, a drugi način omogućava da se dokument spremi na nekom od Cmap servisa na kojima je omogućeno dijeljenje vlastitih mapa ostalim korisnicima, ovo spremanje radi na principu da se pri svakom spremanju mape generira njena web stranica na nekom od servera. Također je dostupna i stranica koja omogućava pretragu svih konceptualnih mapa i njihovog sadržaja. Stranicu je moguće posjetiti na linku: http://www.cmappers.net/. Kod spremanja važno je napomenuti da alat omogućava spremanje dokumenta u različitim formatima poput: PNG, PDF, HTML, CXL, XTM/XCM, IVLM, TEXT.



Slika 24: Korisničko sučelje CmapTools-a

Na (Slika 24) se vidi izgled korisničkog sučelja CmapTools-a. Korisničko sučelje koje korisniku omogućava izradu konceptualnih mapa izgleda malo drugačije i moguće ga je vidjeti na slici ispod.



Slika 25: Prikaz sučelja za kreiranje konceptualne mape

Kao što je na (Slika 25) moguće vidjeti koncept se kreira dvostrukim klikom na radnu površinu alata. Nakon što se kreira koncept automatski se pojavi izbornik koji omogućava uređivanje fonta teksta koji će biti upisan u koncept, njegovo poravnanje te još mnoge druge opcije. Veze među konceptima se dobivaju jednostavnim povlačenjem linije od jednog koncepta do drugog.

Većina programskog modula realizirana je u programskom jeziku C#-u koji je na neki način poveznica između xml datoteke i baze podataka AC-ware Tutor-a. C# uzet za realizaciju programskog modula zbog njegove velike primjene u programiranju te omogućavanja korisniku rješavanja i najzahtjevnijih problema.

C# je najnoviji objektno orijentirani programski jezik koji je doživio razne nadogradnje i stalno se razvija. Nastao je i razvijao se pod vodstvom Andersa Hejlsberga i njegovih suradnika. Danas je glavni jezik .NET platforme koja je, pak, glavni Microsoftov adut za razvoj programske podrške. Svaki ozbiljan programer na Microsoftovoj platformi ne može pobjeći od .NET-a, a time ni od C#-a. .NET (dot-net) je ime najmodernije platforme koja će imati velike utjecaje na programiranje u budućnosti. U nazivu je osnovna poruka koju nosi ova tehnologija: dostupnost u svakom trenutku na svakom mjestu. .NET daje realne osnove da postane osnovna platforma razvoja modernih aplikacija. Radni okvir .NET razvijen je sa ciljem da osigura okruženje za razvoj svih modernih aplikacija na Windows sustavu.

C# je dobro opremljen za izradu Web servisa, nanovo upotrebljivih komponenti na internetu koje udovoljavaju otvorenim standardima. Web servisi aplikacijama omogućuju upotrebu usluga distribuiranih širom Weba, što pojednostavljuje razvoj i povećava ponovnu upotrebljivost koda.

Za rad s bazom podataka AC-ware Tutor-a korišten je SQL koji je nezaobilazan programski jezik za rad s bazom podataka.

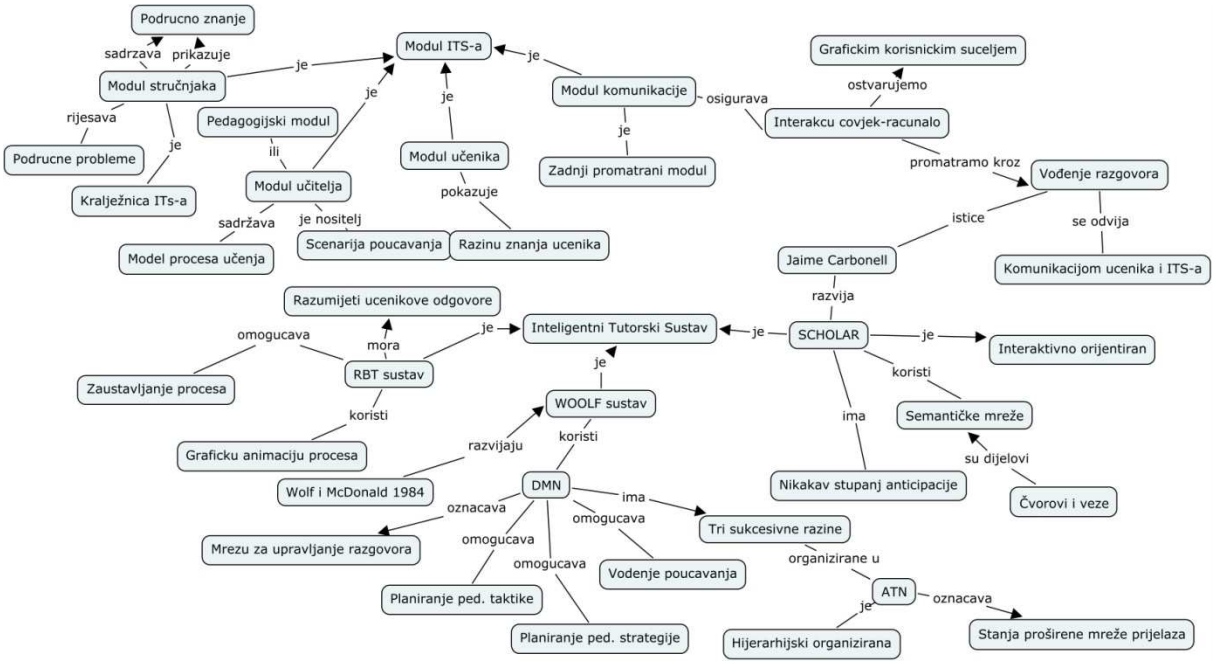
Microsoft SQL (engl. **S**tructured **Q**uery **L**anguage) je programski jezik koji je dizajniran za upravljanje podacima u relacijskim sustavima za upravljanje bazom podataka (RDBMS). Svrstavamo ga u neproceduralne programske jezike jer za razliku od programskog jezika C opisuje što želimo dobiti kao rezultat, a ne kako doći do njega. SQL ima naredbe za sveobuhvatan rad s relacijskom bazom podataka:

- Kreiranje tablice
- Unos podataka u tablice
- Brisanje podataka iz tablica
- Naredbe za definiranje baze podataka(DDL-Data Definition Language)
- Naredbe za manipulaciju podacima(DML-Data Manipulation Language)
- Upravljačke naredbe

4.1 Oblikovanje ontologije područnog znanja

U ovom poglavlju govorit ćemo o područnom znanju koje je kreirano u CmapTolols-u. Područno znanje koje je korišteno u ovom radu je modul komunikacije (Stankov, 2010) iz područja inteligentnih

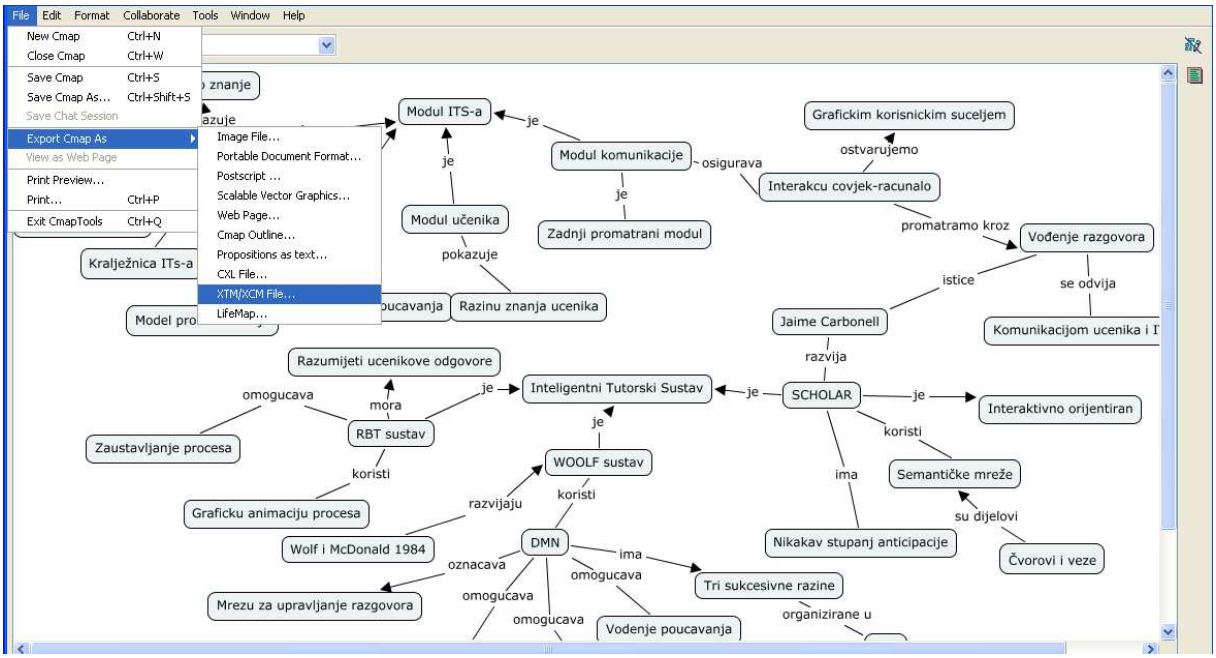
tutorskih sustava. Ovaj modul je uzet kao primjer iz razloga jer je vrlo bitan u radu inteligentnog tutorskog sustava i važno je da ga učenici točnije studenti dobro razumiju. Modul komunikacije je zapravo model koji osigurava nesmetanu i ispravnu komunikaciju učenika i inteligentnog sustava. Jedino dobro osmišljena komunikacija i oblikovano korisničko sučelje poboljšava sposobnosti ovakvih sustava u suprotnom dobra komunikacija te samim time i učenje na ovakvim sustavima bilo bi kompliciran. Primjer konceptualne mape iz izrađene u CmapTools-u prikazan je na sljedećoj slici (Slika 26).



Slika 26: Primjer konceptualne mape kreirane u CmapTools-u

4.2 Prikaz mape koncepata u XML datoteci

Nakon što se u CmapTools-u napravi konceptualna mapa ona se može exportati ili spremiti u raznim formatima kao što je u prethodnom poglavlju napomenuto. U narednom tekstu će biti objašnjeno kako napraviti export konceptualne mape u xml datoteku te kako ona izgleda. Izgled ovakve xml datoteke je malo drugačiji nego obične xml datoteke iz razloga što alat sam generira xml datoteku pa ona izgleda drugačije nego ona koju kreira korisnik. Na (Slika 27) je moguće vidjeti export konceptualne mape kao xml datoteke.



Slika 27: Prikaz spremanja dokumenta u CmapTools-u

Važno je napomenuti da kod odabira formata export-a korisnik odabere XTM format. Nakon što je ovaj proces završen alat šalje povratnu poruku korisniku da je spremanje uspješno završeno. Kao što se može vidjeti prilikom export-a u xml datoteku , datoteka se zapravo export-a u XTM datoteku koja je sadržajem zapravo XML datoteka. XTM dolazi od **X**ML **T**opic **M**aps format.

Pošto je u prethodnom tekstu objašnjeno kako dobiti xml datoteku iz CmapTools alata u narednom tekstu će biti prikazan izgled xml datoteke i u kratkim crtama opisan sadržaj xml datoteke. Izgled xml datoteke može se vidjeti na sljedećoj slici.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <topicMap id="1KQKNS4W6-XSNK61-53" xmlns="http://www.topicmaps.org/xtm/1.0/" xmlns:xlink="http://www.w3.org/1999/xlink">
  - <topic id="1KQKQGSJW-2366K8F-26L">
    - <instanceOf>
        <subjectIndicatorRef xlink:type="simple" xlink:href="http://cmap.coginst.uwf.edu/#concept" />
      </instanceOf>
    - <baseName>
      - <baseNameString>
          <![CDATA[ Tri sukcesivne razine ]]>
        </baseNameString>
      </baseName>
    </topic>
  - <topic id="1KQKQW7M3-1V712KD-2JG">
    - <instanceOf>
        <subjectIndicatorRef xlink:type="simple" xlink:href="http://cmap.coginst.uwf.edu/#concept" />
      </instanceOf>
    - <baseName>
      - <baseNameString>
          <![CDATA[ Graficku animaciju procesa ]]>
        </baseNameString>
      </baseName>
    </topic>
  - <topic id="1KQKPXNN6-24924MG-1NF">
    - <instanceOf>
        <subjectIndicatorRef xlink:type="simple" xlink:href="http://cmap.coginst.uwf.edu/#concept" />
      </instanceOf>
    - <baseName>
      - <baseNameString>
          <![CDATA[ Inteligentni Tutorski Sustav ]]>
        </baseNameString>
      </baseName>
    </topic>
```

Slika 28: Izgled XML datoteke

Kao što se iz (Slika 28) vidi xml datoteka se sastoji od zaglavlja koje daje informacije o samoj xml datoteci kao što su: verzija xml-a te standard u kojem je xml napisan. Također je moguće vidjeti i id-ove koji omogućavaju da korisnik može dobiti uvid u podatke kao što su koncepti i veze koji ih povezuju. Elementi od kojih se sastoji xlm datoteka su sljedeći:

- topicRef: referenca na glavni element
- subjectIndicatorRef: referenca na indikator područja
- instanceOF: element koji ukazuje na glavnu klasu
- topicMap: element konceptualne mape
- topic: glavni element
- baseName:ime elementa
- baseNameString: ime stringa kojeg sadrži
- association: glavna veza, itd

4.3 Spajanje na XML datoteku

U prethodnom poglavlju je objašnjeno kako iz CmapTools-a dobiti xml datoteku dok će u ovom poglavlju biti ukratko objašnjeno kako se pomoću C# spojiti na xml datoteku.

Da bi se XML datoteci pristupilo preko nekog programskog jezika potrebno je znati adresu na korisnikovom računalu gdje se ta datoteka nalazi. Pristupanje XML datoteci koristeći programski jezik C# moguće je vidjeti na (Slika 29).

```
System.Xml.XmlTextReader reader = new System.Xml.XmlTextReader(@"C:\Documents and Settings\Šime\Desktop\Prvi projekt.xml");
```

Slika 29: Prikaz pristupa XML datoteci preko C#

4.4 Učitavanje podataka iz XML-datoteke

Nakon spajanja na XML datoteku glavni zadatak je preko programskog jezika C# izvući podatke. Podaci koji su potrebni za rad su:

- Koncepti
- Veze ili relacije
- Trojke smislena cjelina koja se sastoji od koncepata i veza

Da bi rad s podacima te pristup njima bio lakši korist će se liste koje se definiraju na početku programa te će se kasnije puniti traženim podacima. U ovom primjeru liste su korištene da bi se izbjeglo dupliciranje koda te da bi se povećala čitljivost koda. Definiranje liste se vidi na (Slika 30).

```
public partial class Form1 : Form
{
    public List<string> koncepti;
    public List<string> veze;
    public List<string> konceptiBezDuplikata;
    public List<string> vezeBezDuplikata;
    public List<string> trojke;
    public List<string> trojkeID;
    public Form1()
    {
        InitializeComponent();
        koncepti = new List<string>();
        veze = new List<string>();
        konceptiBezDuplikata = new List<string>();
        vezeBezDuplikata = new List<string>();
        trojke = new List<string>();
        trojkeID = new List<string>();
        nadiKoncepteVezeTrojke(); // pospremi sve informacije u liste
    }
```

Slika 30: Definiranje liste u C#

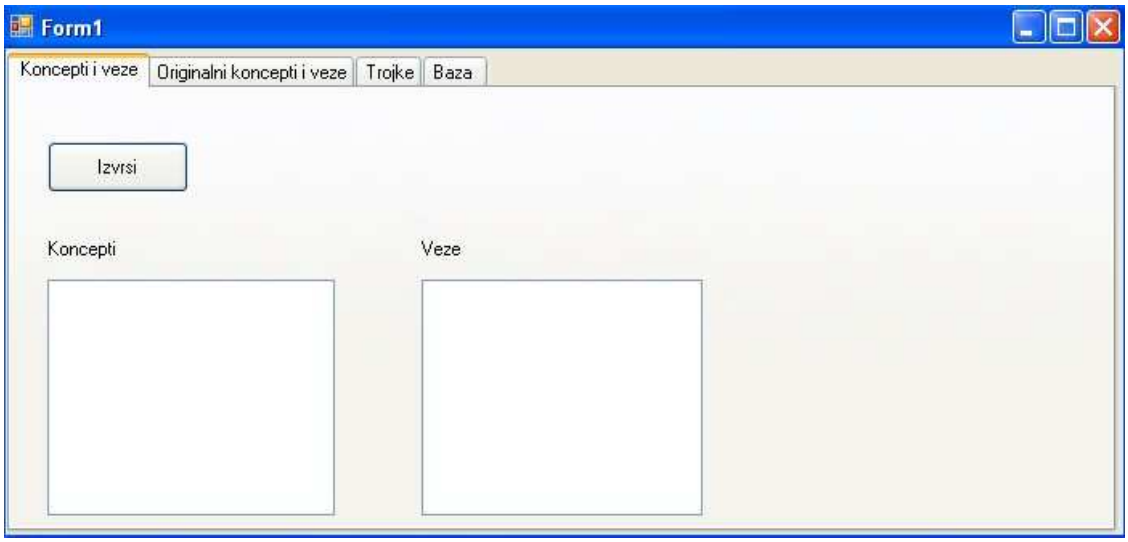
Iako je zadatak pronaći koncepte, veze i trojke, koncepti i veze će se tražiti preko trojki iz nekoliko razloga:

- Zbog jednostavnosti dobivanja koncepata i veza iz trojki
- Zbog ekonomičnosti to jest smanjenja potrošnje vremena u pisanju koda
- Zbog čitljivosti koda i lakšeg snalaženja u kodu

Važno je napomenuti da je aplikacija rađena u Windows aplikaciji koja se zasniva na form klasi i koja omogućava grafički prikaz onoga što se radi. U da bi se problem pronalaženja koncepata, veza i trojki uspješno riješio u radu su korišteni objekti:

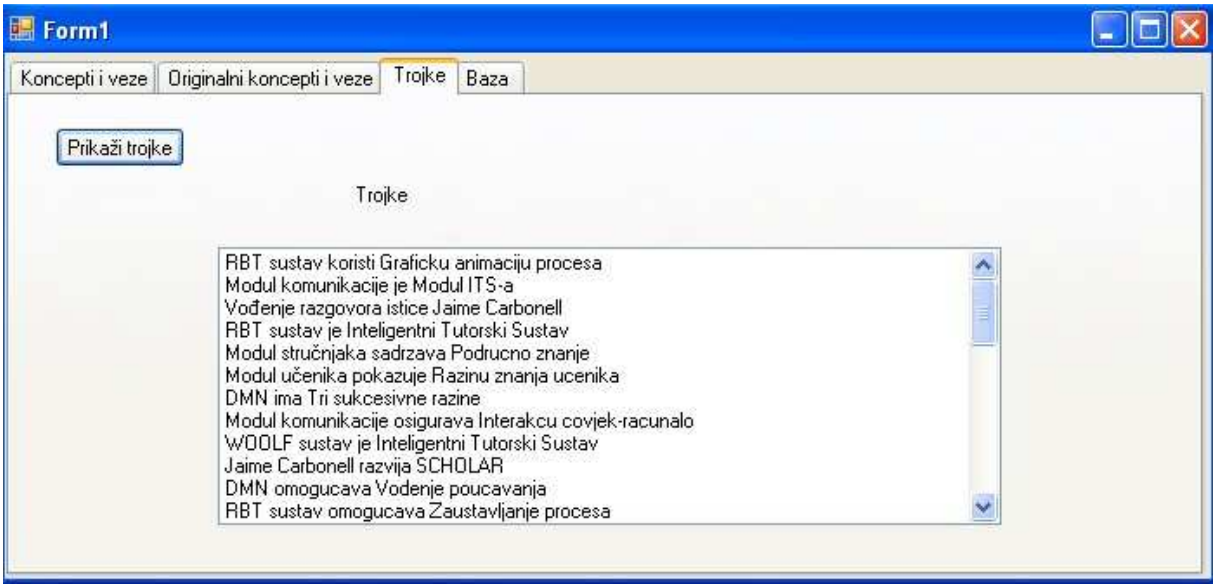
- Button
- Label
- Listbox
- Tabcontrol

Nakon pokretanja aplikacije korisnik je u mogućnosti vidjeti grafički prikaz onoga što je radio ili takozvanu formu.



Slika 31: Izgled aplikacije prilikom pokretanja

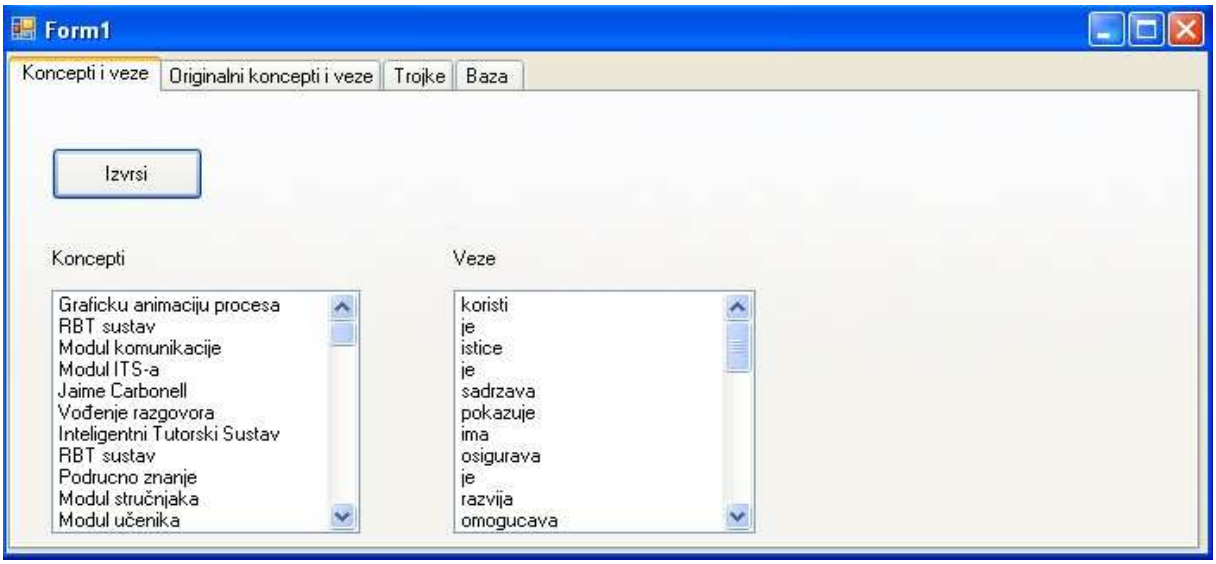
Prilikom pokretanja aplikacije (Slika 31) može se vidjeti da se aplikacija sastoji od izbornika koji prilikom klika na jednu od opcija izbornika omogućava korisniku kretanje po aplikaciji i izvršavanje onog dijela u aplikaciji koji korisnik odabere. Redoslijed izvršavanja programa i prikaza rješenja na zaslonu nije biran te korisnik o njemu sam odlučuje ovisno o njegovim željama i potrebama. Kako bi korisnik izvršavanje programa mogao vidjeti na ekranu potrebno je kliknuti na botun onoga što želi vidjeti inače to nije moguće. Prikaz podataka omogućava listbox. Prikaz trojki moguće je vidjeti na (Slika 32).



Slika 32: Prikaz trojki u C#-u

4.5 Izdvajanje duplih podataka

Prilikom izrade konceptualnih mapa često se događa da se pojedini koncepti i veze ponavljaju više puta to jest dupliciraju. U ovom koraku izrade aplikacije zadatak je izbaciti duple koncepte i veze. Dupli koncepti i veze se nastoje izbaciti zbog toga što se ti podaci spremaju u bazu podataka, a prilikom spremanja koncepata i veza u bazu podataka izdvajanjem duplih podataka nastoji se optimizirati zauzeće baze podataka.



Slika 33: Prikaz koncepata i veza u C#-u

Na (Slika 33) se vidi kako izgleda kada se izdvoje koncepti i veze koji su dupli, a nakon izdvajanja duplih koncepata i veza dobiju se originalni koncepti i veze što se vidi na (Slika 34).

Slika 34: Prikaz originalnih koncepata i veza u C#-u

4.6 Spajanje na bazu AC-ware Tutora

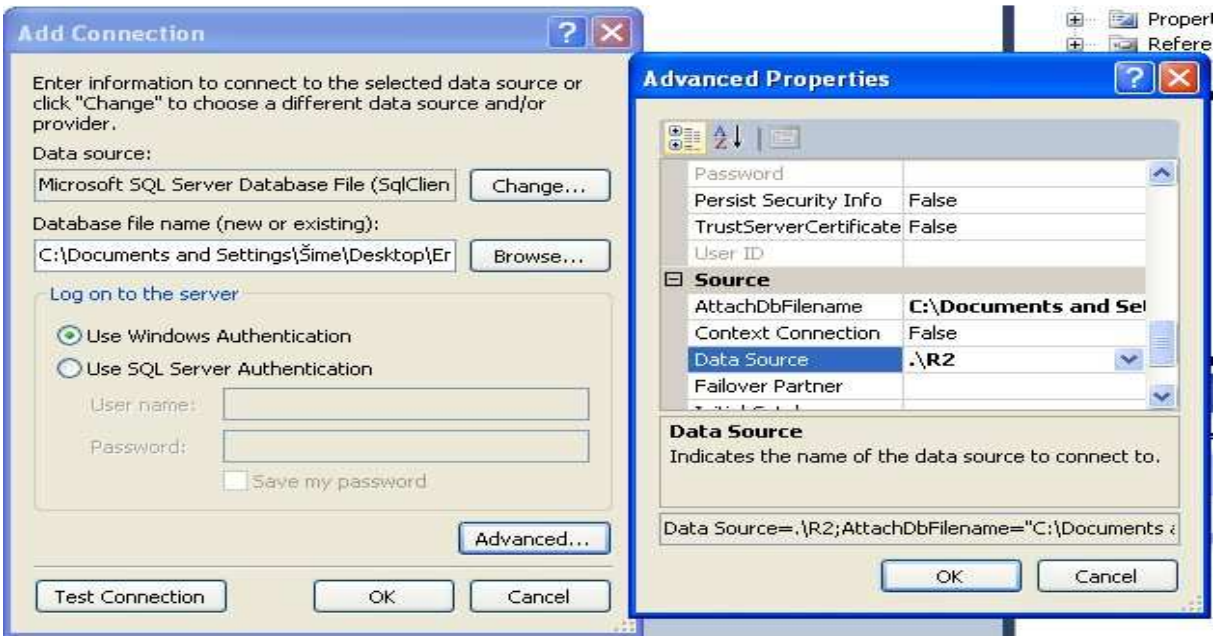
Nakon izdvajanja originalnih podataka potrebno je te podatke spremiti u bazu podataka. Baza podataka sustava je napravljena u Microsoft SQL programskom jeziku što je i ranije napomenuto. Prije samog spremanja podataka u bazu napraviti će se kratki uvod o koracima koji su potrebni za spajanje na bazu.

Da bi se spojili na bazu potrebno je:

- instalirati SQL programski jezik na vlastito računalo (Slika 35)

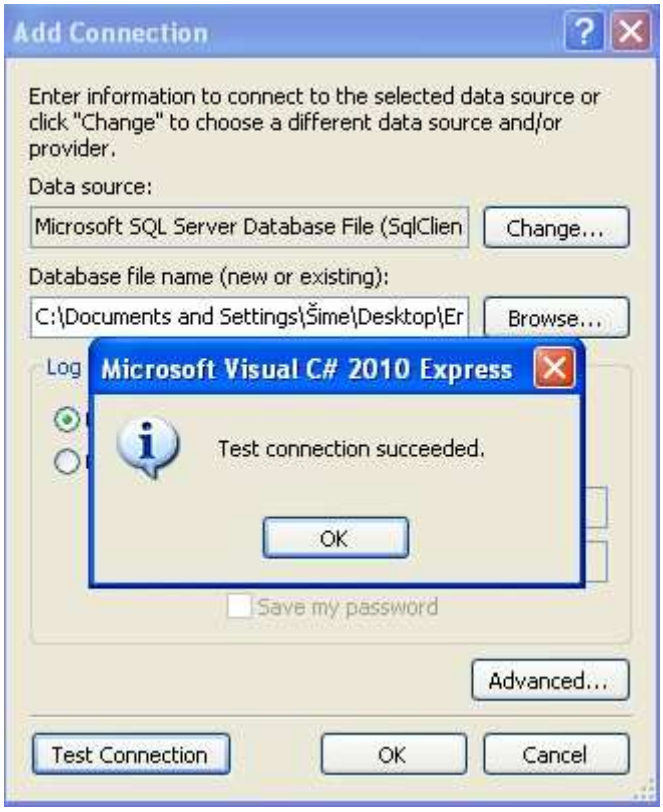
Slika 35: Instalacija SQL-a

- kreirati novu konekciju te u postavkama izvor podataka(engl. Data Source) postaviti na .\R2 da bi konekcija bila moguća (Slika 36)



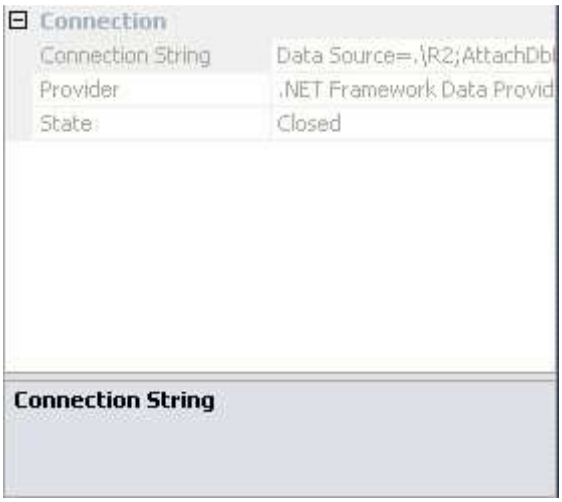
Slika 36: Prikaz postavki za konekciju na bazu podataka

- testiranje konekcije (Slika 37)



Slika 37: Testiranje konekcije

- preuzimanje konekcijskog stringa (Slika 38)



Slika 38: Prikaz konekcijskog stringa

4.7 Spremanje podataka u bazu

Da bi bilo moguće spremanje podataka u bazu nekog sustava najprije se mora vidjeti od čega se baza podataka sustava sastoji, konkretno koja su imena tablica te koje podatke te tablice sadrže. Baza podataka sustava AC-ware Tutora se sastoji od tablica (Grubišić, 2012):

- Triple
- Link
- Node
- Attribute
- Area
- Subarea
- Frame

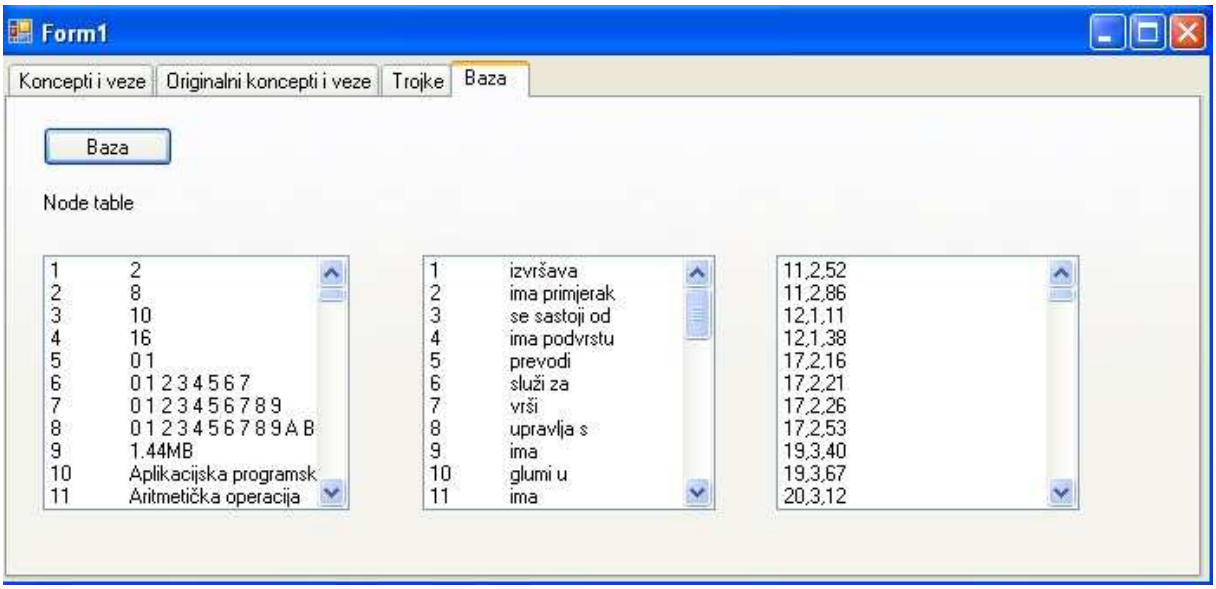
Tablice koje su važne za ovu aplikaciju su: trojke, koncepti i trojke. Podaci koji se unose u tablicu koncepata i veza su:

- AuthorID
- SubareaID
- Name
- Inherit
- Description
- Type
- IsRemoved

Važno je napomenuti da su svi podaci osim imena fiksni i unaprijed zadani dok je vrijednost imena zapravo vrijednost veza i koncepata koji su dobiveni iz xml datoteke te su prikazani u listboxu. Id-ovi koncepata i veza se automatski generiraju prilikom unosa podataka u tablicu. Kasnije se ti id-ovi spremaju u tablicu trojki. Pristup i manipulacija podacima radi se jednostavnim upitima koristeći SQL naredbe pod pretpostavkom da je korisnik spojen na bazu sustava na način koji je u prethodnom poglavlju objašnjen.

Da bi se podaci podatci unijeli u bazu podataka koristi se naredba INSERT dok se naredbom SELECT učitavaju podaci iz baze i prikazuju korisniku. Naredbom SELECT se omogućava provjera ispravnosti unesenih podataka te da bi se omogućio kasniji unos id-ova u bazu podataka. ***Napomena: prilikom unošenja trojki u bazu mora se paziti da se trojke dobivene preko podataka unesenih iz XML datoteke mogu samo jednom unijeti u bazu podataka u suprotnom programski jezik C# javlja pogrešku.***

Prikaz podataka iz baze može se vidjeti na (Slika 39).

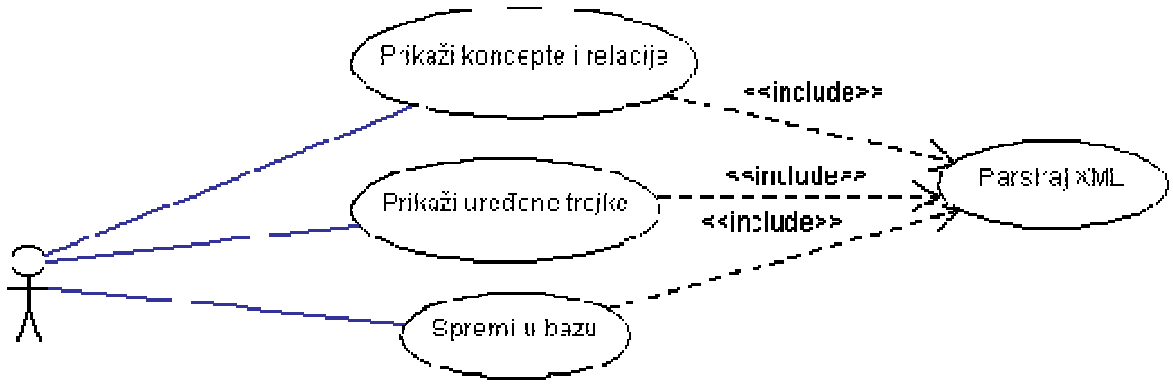


Slika 39: Prikaz podataka iz baze sustava AC-ware Tutor-a

4.8 Arhitektura modula za oblikovanje ontološki utemeljenog područnog znanja

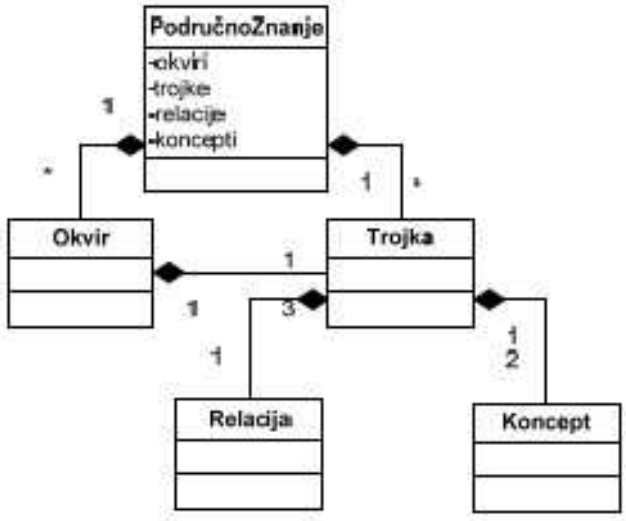
Arhitektura praktičnog dijela diplomskog rada će u ovom poglavlju biti pojašnjena pomoću UML dijagrama. Osnovni UML dijagrami koji će biti prikazani su: dijagram slučajeva korištenja (engl. use case diagram) i dijagram klasa (engl. class diagram).

Struktura praktičnog rada ili aplikacije se vidi kroz opis sudionika i funkcionalnosti koje koriste sudionici sustava. Jedini sudionik ovog sustava je korisnik koji može biti svatko tko se imalo razumije u rad na računalu. Korisnik može vidjeti: koncepte i relacije te uređenje trojke koje su smislena cjelina koja se sastoji od veze i koncepata. Nadalje korisnik može spremiti određene podatke u bazu podataka sustava AC-ware Tutora (Slika 40).



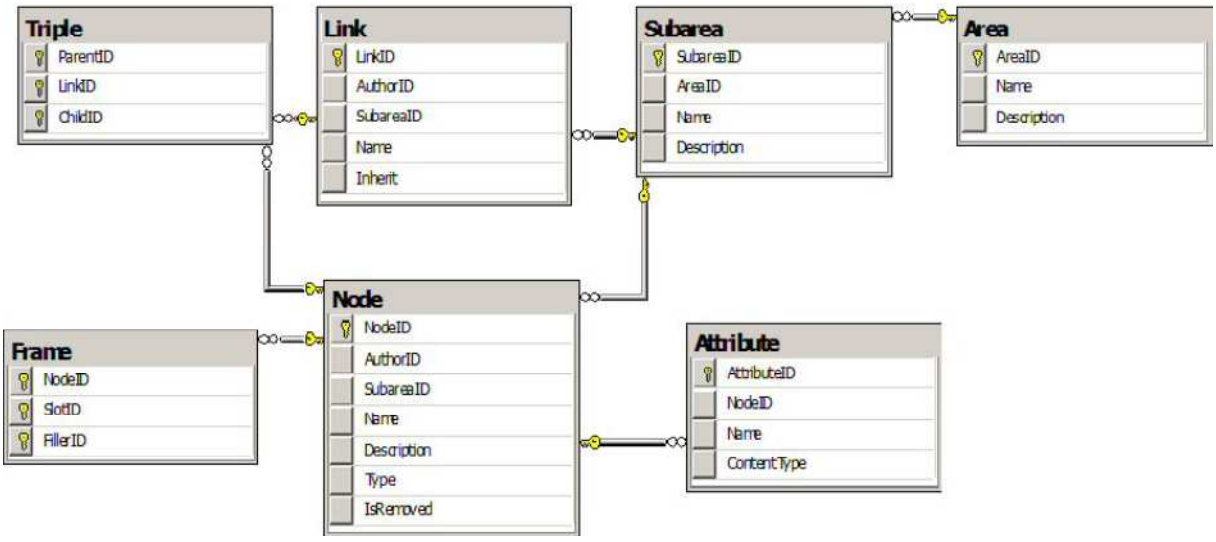
Slika 40: Dijagram slučajeva korištenja

Dijagram klasa predstavlja vizualnu specifikaciju objekata koji postoje u sustavu i odnose između njih. Njima se mogu specificirati konceptualni i implementacijski dijelovi sustava. U ovom slučaju dijagram klasa (Slika 41) je također rađen preko UML alata kao što je rađen dijagram slučajeva korištenja.



Slika 41: Dijagram klasa

Model podataka sustava AC-ware Tutor može se vidjeti na (Slika 42). Naglasak se stavlja na tablice koje su važne u ovom radu, a to su: koncepti (engl. Node), veze (engl. Link) te trojke (engl. Triple). Važnost ovih tablica je u tome što se u ove tablice zapravo sprema područno znanje sustava AC-ware Tutor-a.

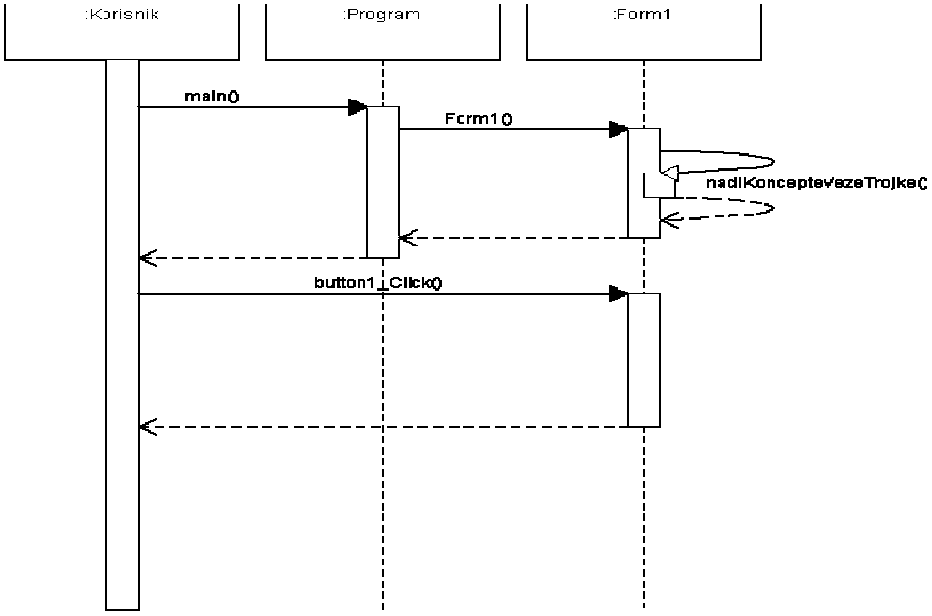


Slika 42: Model podataka AC-ware Tutor-a

Sljedeći UML dijagram koji će biti obrađen i opisan je dijagram redoslijeda (engl. Sequence diagram) koji zapravo opisuje ponašanje aplikacije koje se prikazuje kroz sinkronu razmjenu poruka te prikazuje redoslijed događaja koji se javljaju. Dijagram redoslijeda je povezan s dijagramom slučajeva korištenja jer se svaka funkcionalnost dijagrama slučajeva korištenja može opisati i razraditi

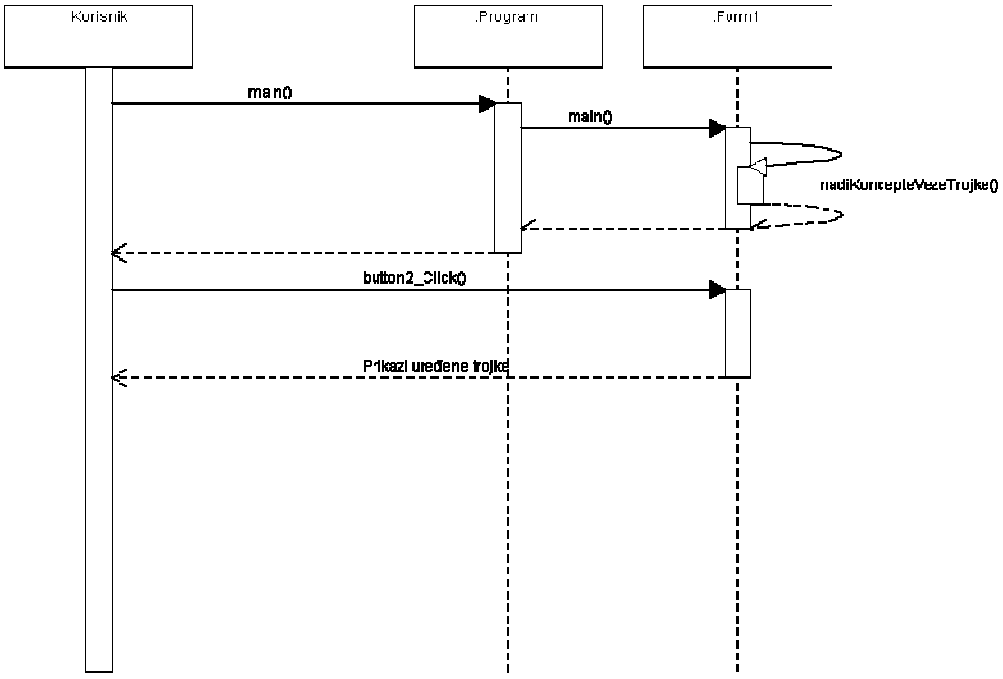
dijagramom redoslijeda te se kaže da je dijagram redoslijeda zapravo realizacija dijagrama slučajeva korištenja pod pretpostavkom da je dijagram slučajeva korištenja već nacrtan.

Na (Slika 43) se može vidjeti realizacija dijagrama redoslijeda za funkcionalnost prikaza koncepata i relacija.



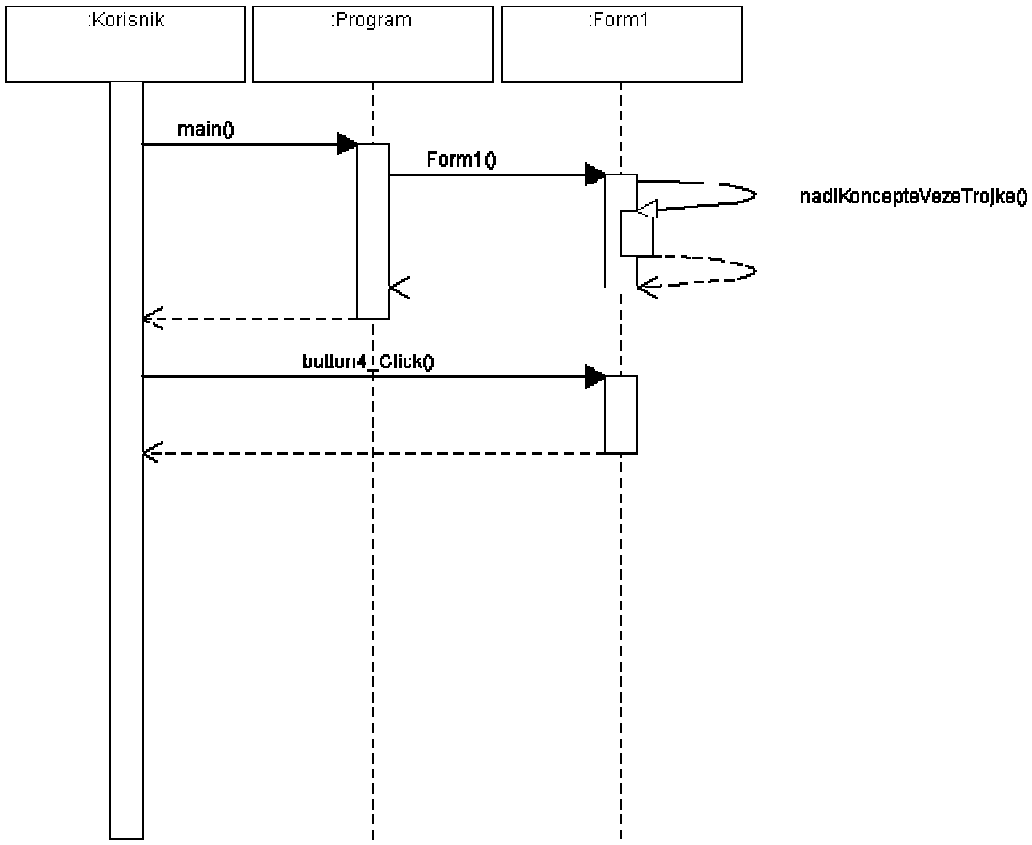
Slika 43: Dijagram redoslijeda za prikaz koncepata i relacija

Na (Slika 44) se vidi realizacija dijagrama redoslijeda za funkcionalnost prikaži uređene trojke.



Slika 44: Dijagram redoslijeda za prikaz uređenih trojki

Posljednja funkcionalnost za koju je realiziran dijagram redoslijeda je spremi u bazu i može se vidjeti na (Slika 45).



Slika 45: Dijagram redoslijeda za spremi u bazu

5 Zaključak

Prilikom realizacije rada pojavilo se nekoliko problema koji su na kraju uspješno riješeni i rad je priveden kraju. Prvi problem se javljao usred nedovoljno dobrog razumijevanja ontologije i konceptnih mapa, te je oblikovanje područnog znanja išlo malo sporije, ali nakon određenog vremena taj je problem riješen.

Drugi problem je bio s „izvlačenjem“ podataka iz xml datoteke, jer je xml datoteka kreirana od strane programskog alata i izgleda drugačije nego kad je kreira korisnik. Da bi se podaci dobili iz xml datoteke korisnik mora dobro poznavati izgled dane xml datoteke, te na temelju toga dobiti neke zaključke i vidjeti kako su elementi povezani s obzirom na strukturu xml datoteke i njezinu veličinu. Problem je što je xml datoteka jako duga i malo je teže pratiti njenu strukturu i uočiti neke zakonitosti. Da bi se jako brzo shvatilo kako su podaci organizirani u xml potrebno se orijentirati na id-ove jer jedino je tako moguće uočiti neke zakonitosti u jako kratkom roku.

U ovom diplomskom radu bilo je potrebno koristiti razne programske alate. Bez obzira na korištenje raznih programskih alata potrebno je mnogo vremena da se područno znanje oblikuje i da se podaci unesu u bazu podataka. Oblikovanje područnog znanja oduzima najviše vremena i za sada ga je nemoguće ubrzati i automatizirati, ali možda se i taj problem riješi nekim naprednim programskim alatom koji bi omogućio automatsku izradu područnog znanja. U svakom slučaju aplikacija koja se razvijala u ovom radu ubrzava rad korisnika i smanjuje vrijeme potrebno da se područno znanje prenese u neki sustav. Aplikacija omogućava automatizaciju nekih dijelova gdje je omogućena uštedu vremena i manji napor od strane onoga koji radi područno znanje.

6 Literatura

1. Damir Jurić: Metode učenja ontologija-trenutno stanje i problemi, s Interneta, http://www.fer.unizg.hr/_download/repository/KvalifikacijskiIspitDamirJuric.pdf, 09.09.2013
2. Đuraković A.; Verić V.: Semantički web, s Interneta http://www.mathos.unios.hr/~adjurako/semantickiweb/Semanti%C4%8Dki%20web.pdf, 09.09.2013
3. Gruber, T.R. (1993). A translation approach to portable ontology specifications. Knowlege acquisition, 5(2), pp. 199-220.
4. Grubišić, A.: Model prilagodljivoga stjecanja znanja u sustavima e-učenja, FER Zagreb, 2012
5. Guarino N.: Formal ontology and Information systems, s Interneta, http://www.mif.vu.lt/~donatas/Vadovavimas/Temos/OntologiskaiTeisingasKoncepcinisModeliavimas/papildoma/Guarino98-Formal%20Ontology%20and%20Information%20Systems.pdf, 20.09.2013
6. IEEE: Guide to the Software Engineering Body of Knowledge, s Interneta http://www.inf.ed.ac.uk/teaching/courses/seoc/2006_2007/resources/SWEBOK_Guide_2004.pdf, 20.09.2013
7. Legg C.: Ontologije na semantičkom webu, s Interneta, http://www.hkdrustvo.hr/datoteke/838/vbh/God.53(2010),br.1, 19.09.2013
8. Matešić M.: Umjetna inteligencija, s Interneta, http://www.mensa.hr/glavna/misli-21-stoljeca/479-umjetna-inteligencija-uvod, 19.09.2013
9. Novak J. D., Cañas A. J.: The Theory Underlying Concept Maps and How to Construct and Use Them. Technical Report IHMC CmapTools 2006-01 Rev 2008-01
10. Paunović L., Stokić A.: Uticaj ontologija na funkcionalnost web-a, s Interneta http://www.infoteh.rs.ba/rad/2012/RSS-8/RSS-8-3.pdf, 20.09.2013
11. Pavić K.: Semantički web, s Interneta, http://www.zemris.fer.hr/predmeti/krep/ Pavic.pdf, 09.09.2013
12. Prcela M.: Predstavljanje ontologija na webu, s Interneta http://www.fer.unizg.hr/_download/repository/Marin_Prcela_SW.pdf, 20.09.2013
13. Prgomet M.: Neformalni oblici prikaza znanja u Internet okruženju, s Interneta, http://www.splithorizont.com/pdf/mentalna_mapa_diplomski.pdf, 09.09.2013
14. Programsko inženjerstvo,s Interneta, http://hr.wikipedia.org/wiki/Programsko_in%C5%BEenjerstvo, 19.09.2013
15. Scott M. W.: XML za programere, Elliotte Rusty Harold, 2006

16. Skokandić A.: Umne mape, s Interneta, http://www.cool-school.net/index.php?ucitelj=askokandic&view=221, 19.09.2013

17. Smith B., Welty C.: Ontology: Towards a New Synthesis, s Interneta http://www.cs.vassar.edu/~weltyc/papers/fois-intro.pdf, 20.09.2013

18. Stankov S.: Inteligentni tutorski sustavi: teorija i primjena, PMF Split, 2010

19. Uschold M., Gruninger M.: ONTOLOGIES: Principles, Methods and Applications , s Interneta, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.111.5903&rep=rep1 &type=pdf, 20.09.2013

20. XML, s Interneta, http://hr.wikipedia.org/wiki/XML, 09.09.2013

7 Prilozi

7.1 Prikaz koda aplikacije

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;
using System.Data.SqlClient;


namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public List<string> koncepti;
      public List<string> veze;
      public List<string> konceptiBezDuplikata;
      public List<string> vezeBezDuplikata;
      public List<string> trojke;
      public List<string> trojkeID;
      public Form1()
      {
         InitializeComponent();
         koncepti = new List<string>();
         veze = new List<string>();
         konceptiBezDuplikata = new List<string>();
         vezeBezDuplikata = new List<string>();
         trojke = new List<string>();
         trojkeID = new List<string>();
         nadiKoncepteVezeTrojke(); // pospremi sve informacije u liste
      }


      private void nadiKoncepteVezeTrojke()
      {
         System.Xml.XmlTextReader reader = new System.Xml.XmlTextReader(@"C:\Documents and
Settings\Šime\Desktop\Modul komunikacije.xml");

         while (reader.Read())
         {
            // ako je trenutni element asocijacija
            if (reader.Name.Equals("association") && (reader.NodeType == XmlNodeType.Element))
            {
               // premjesti se do prvog topicRef - nađi id veze
               reader.ReadToFollowing("topicRef");
```

```
            string idVeza = reader.GetAttribute("xlink:href").Substring(1);// potrebno je uzimati substring od
prvog znaka nadalje, budući da id-evi u asocijacijama počinju sa znakom #

            // premjesti se do drugog topicRef - nađi id prvog koncepta
            reader.ReadToFollowing("topicRef");
            string idKoncept1 = reader.GetAttribute("xlink:href").Substring(1);

            // premjesti se do drugog topicRef - nađi id drugog koncepta
            reader.ReadToFollowing("topicRef");
            string idKoncept2 = reader.GetAttribute("xlink:href").Substring(1);

            // preko id-eva nađi vrijednosti koncepata i veze
            System.Xml.XmlTextReader reader2 = new System.Xml.XmlTextReader(@"C:\Documents and
Settings\Šime\Desktop\Modul komunikacije.xml");

            string veza = "";
            string koncept1 = "";
            string koncept2 = "";
            string trojkeIDkoncept1 = "";
            string trojkeIDkoncept2 = "";
            string trojkeIDveza = "";
            while (reader2.Read())
            {
              if (reader2.Name.Equals("topic") && (reader2.NodeType == XmlNodeType.Element))
              {
                // provjeri je li trenutni id topica jednak nekom od id-eva elemenata trojke
                string idTopic = reader2.GetAttribute("id");
                if (idTopic.Equals(idVeza))
                {
                  reader2.ReadToFollowing("baseNameString");
                  veza = reader2.ReadString();
                  veze.Add(veza); // dodaj vezu u listu veza
                  trojkeIDveza = idVeza;
                }
                else if (idTopic.Equals(idKoncept1))
                {
                  reader2.ReadToFollowing("baseNameString");
                  koncept1 = reader2.ReadString();
                  koncepti.Add(koncept1); // dodaj koncept u listu koncepata
                  trojkeIDkoncept1 = idKoncept1;
                }
                else if (idTopic.Equals(idKoncept2))
                {
                  reader2.ReadToFollowing("baseNameString");
                  koncept2 = reader2.ReadString();
                  koncepti.Add(koncept2); // dodaj koncept u listu koncepata
                  trojkeIDkoncept2 = idKoncept2;
                }
                else
                {
                  continue;
                }

                // ako su nađeni svi elementi trojke, ispiši ih, prijeđi na drugu asocijaciju
                if (veza.Length != 0 && koncept1.Length != 0 && koncept2.Length != 0)
                {
                  trojke.Add(koncept1 + " " + veza + " " + koncept2);
```

```
                    trojkeID.Add(koncept1 + "|" + veza + "|" + koncept2);
                    break;
                }
            }
        }
    }
}

konceptiBezDuplikata = makniDuplikate(koncepti);
vezeBezDuplikata = makniDuplikate(veze);
}


// funkcija koja iz liste miče duplikate
private List<string> makniDuplikate(List<string> lista){
    List<string> listaBezDuplikata = new List<string>();

    foreach (string element in lista)
    {
        // ako nova lista ne sadrži element poslane liste, stavi ga u novu listu
        if (!listaBezDuplikata.Contains(element)){
            listaBezDuplikata.Add(element);
        }
    }
    return listaBezDuplikata;
}
private void button1_Click_1(object sender, EventArgs e)
 {
    foreach(string koncept in koncepti){
        listBox1.Items.Add(koncept);
    }
    foreach (string veza in veze)
    {
        listBox2.Items.Add(veza);
    }
 }

private void button2_Click(object sender, EventArgs e)
 {
    foreach (string koncept in koncepti)
    {
        listBox6.Items.Add(koncept); // prikaži koncepte s duplikatima
    }
    foreach (string veza in veze)
    {
        listBox4.Items.Add(veza); // prikaži veze s duplikatima
    }
    foreach (string koncept in konceptiBezDuplikata)
    {
        listBox5.Items.Add(koncept); // prikaži koncepte bez duplikata
    }
    foreach (string veza in vezeBezDuplikata)
    {
        listBox3.Items.Add(veza); // prikaži veze bez duplikata
    }

 }
```

```
private void button3_Click(object sender, EventArgs e)
    {
      //foreach (string trojka in trojkeID)
      foreach (string trojka in trojke)
      {
        listBox7.Items.Add(trojka); // prikaži koncepte bez duplikata
      }
    }


private void button4_Click(object sender, EventArgs e)
    {

      SqlConnection thisConnection = new SqlConnection(@"Data
Source=.\R2;AttachDbFilename=C:\Documents and
Settings\Šime\Desktop\Env.Database\env.db.mdf;Integrated Security=True;Connect Timeout=30;User
Instance=True");
      thisConnection.Open();

      foreach (string koncept in konceptiBezDuplikata)
      {
        SqlCommand cmd = new SqlCommand("INSERT into Node values
(1474,4,@Name,1,0,2)",thisConnection);
        cmd.Parameters.AddWithValue("@Name", koncept);
        cmd.ExecuteNonQuery();
      }

      SqlCommand thisCommand2 = thisConnection.CreateCommand();
      thisCommand2.CommandText = "SELECT NodeID,Name FROM Node";
      SqlDataReader thisReader = thisCommand2.ExecuteReader();
      Dictionary<string, string> koncepti = new Dictionary<string, string>();
      while (thisReader.Read())
      {
        string id = thisReader["NodeID"].ToString();
        string name = thisReader["Name"].ToString();
        koncepti.Add(id,name);
        listBox8.Items.Add(id+"\t"+name);
      }
      thisReader.Close();
      thisConnection.Close();

      SqlConnection Connection = new SqlConnection(@"Data Source=.\R2;AttachDbFilename=C:\Documents
and Settings\Šime\Desktop\Env.Database\env.db.mdf;Integrated Security=True;Connect Timeout=30;User
Instance=True");
      Connection.Open();

      foreach (string veza in vezeBezDuplikata)
      {
        SqlCommand cmd = new SqlCommand("INSERT into Link values (1474,4,@Name,0)", Connection);
        cmd.Parameters.AddWithValue("@Name", veza);
        cmd.ExecuteNonQuery();
      }

      SqlCommand thisCommand3 = Connection.CreateCommand();
      thisCommand3.CommandText = "SELECT LinkID,Name FROM Link";
      SqlDataReader thisReader2 = thisCommand3.ExecuteReader();
      Dictionary<string, string> veze = new Dictionary<string, string>();
```

```
        while (thisReader2.Read())
        {
            string id = thisReader2["LinkID"].ToString();
            string name = thisReader2["Name"].ToString();
            veze.Add(id, name);
            listBox9.Items.Add(id + "\t" + name);
        }
        thisReader2.Close();
        Connection.Close();


        SqlConnection Connection3 = new SqlConnection(@"Data
Source=.\R2;AttachDbFilename=C:\Documents and
Settings\Šime\Desktop\Env.Database\env.db.mdf;Integrated Security=True;Connect Timeout=30;User
Instance=True");
        Connection3.Open();

        foreach (string trojka in trojkeID)
        {
            SqlCommand cmd = new SqlCommand("INSERT INTO Triple([ParentID],[LinkID],[ChildID]) VALUES
(@ParentID,@LinkID,@ChildID)", Connection3);
            string[] podaci = trojka.Split('|');
            string unos1 = "";
            string unos3 = "";
            string unos2 = "";
            string unos = "";
            string prviKoncept = podaci[0];
            string veza = podaci[1];
            string drugiKoncept = podaci[2];
            unos1 = koncepti.First(x => x.Value == prviKoncept).Key;
            unos2 = unos + veze.First(x => x.Value == veza).Key;
            unos3 = unos + koncepti.First(x => x.Value == drugiKoncept).Key;
            cmd.Parameters.AddWithValue("@ParentID", unos1);
            cmd.Parameters.AddWithValue("@LinkID", unos2);
            cmd.Parameters.AddWithValue("@ChildID", unos3);

            cmd.ExecuteNonQuery();
        }

        SqlCommand Command3 = Connection3.CreateCommand();
        Command3.CommandText = "SELECT ParentID,LinkID,ChildID FROM Triple";
        SqlDataReader Reader3 = Command3.ExecuteReader();
        while (Reader3.Read())
        {
            listBox10.Items.Add(Reader3["ParentID"] + "," + Reader3["LinkID"] + "," + Reader3["ChildID"]);
        }
        Reader3.Close();
        Connection3.Close();
    }
  }
}
```

7.2 Izgled aplikacije prilikom pokretanja