Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“
Zrenjanin

PROJEKTOVANJE I IMPLEMENTACIJA SOFTVERA ZA PODRŠKU I PRAĆENJE RADA ŠAHOVSKIH KLUBOVA

- Diplomski rad -

Mentor: Prof. dr Radosav Dragica

Student: Miloš Kecman
Broj indeksa: 36/01-02
Smer: Profesor informatike

Zrenjanin, 2007.

Sadržaj:

1. Uvod

Projektovanje softvera je daleko osetljivija i važnija aktivnost u životnom ciklusu softvera od same izrade softvera. Objektno orijentisani pristup, koji je prikazan u ovom radu uvodi koncepte koji olakšavaju modelovanje realnog sveta i omogućavaju ponovnu upotrebu koda, ali samo pod uslovom da je softverski sistem dobro isprojektovan.
Dobar projekat softvera znači ne samo u potpunosti zadovoljenje trenutnih zahteva nego i sposobnost softvera da se lako prilagodi novim budućim zahtevima korisnika. Zato su fleksibilnost i lakoća održavanja softvera jedan od ključnih principa kojima se teži u objektno orijentisanom programiranju.

2. Metodologije

Osamdesetih godina programeri su počeli sa opisivanjem kontrolisanog procesa razvoja softvera, od opisa zadatka do održavanja gotovog proizvoda. Ovo je dovelo do stvaranja **strukturne metodologije** kao što je **SSADM** (Strukturna Sistem Analiza i Dizajn Metod). Metodologija predstavlja opis koraka kroz koje razvojni tim treba da prođe, a u cilju stvaranja visoko-kvalitetnog sistema [2].
Devedesetih godina, objektno orijentisano programiranje preuzima dominaciju i projektanti počinju da rade na **Objektno-Orijentisanim Metodologijama**, prijemčivijim objektno -orijentisanom programerskom stilu. Rane OOM uključuju *Booch* metod [Booch 93], *Objectory* metod [Jacobson 92] i OMT [Rumbaugh 91], nastaje i ***Rational Unified Process*** kao i metodologija koja sve više dobija na popularnosti ***eXtreme Programming*** [3] **.**

Metodologija je sistematski način rada. Ona je ponavljajući (repeticioni) proces koji omogućava postupak praćenja razvoja softvera od njegove najranije faze do održavanja instaliranog sistema. Metodologija specificira očekivani proizvod prateći njegov proces (kao i formu datog proizvoda).
Iako je većina metodologija dizajnirana da izađe na kraj sa timovima projektanata koji proizvode veliku količinu softvera, razumevanje osnova dobre metodologije je veoma bitno i za projektante koji rade na malim projektima. Razlog za to je :

- Metodologija pomaže pri nametanju discipline prilikom kodiranja.
- Prolaženjem kroz osnovne metodološke korake povećava naše razumevanje problema, poboljšavajući kvalitet našeg rešenja.
- Omogućava nam uočavanje konceptualnih i praktičnih grešaka pre nego što ih unesemo u izvorni kod.
- Na svakom nivou metodologija specifikuje naš sledeći korak.
- Metodologija nam pomaže da proizvedemo proširiv kod (koji se lako menja), kod koji se može koristiti za rešenje i drugih problema i koji je lakši za pronalaženje grešaka (jer je dobro dokumentovan).

Veliki razvojni projekti su takođe na dobiti:

- Dokumentacija: Metodologija obuhvata postupak dokumentovanja svake faze razvojnog truda, tako da za završeni sistem nije nepristupačno monolitan.
- Smanjenje vremena pristupa: Pošto su poslovni tokovi, aktivnosti, uloge i zavisnosti razumljiviji, javlja se manja mogućnost da proces stoji u redu za čekanje resursa.
- Povećava se šansa za isporukom na vreme i bez probijanja budžeta.
- Princip ponovljivosti: Pošto imamo dobro definisane aktivnosti, slični projekti bi trebalo da se isporučuju sa sličnim vremenskim rokom i sa sličnim cenama. Ako proizvodimo slične sisteme iznova za različite kupce možemo težiti ka metodologiji koja će se koncetrisati na jedinstvene aspekte poslednjeg projekta eventualno možemo automazizovati delove projekta i prodavati ih drugim proizvodnim timovima.

Dobra metodologija je sačinjena od :

- Planiranje: Odluka o tome šta treba da se uradi.
- Spiskovi: Mapiranje stvari koje treba učiniti.
- Sredstva: Proračunavanje ljudskih, softverskih, hardverskih i ostalih potrebnih resursa.
- Poslovni tokovi: Dizajniranje arhitekture sistema, modeliranje domena problema i planiranje projektantskog truda.
- Aktivnosti: Individualni zadaci sa poslovnim tokovima, kao što je testiranje komponenata, crtanje dijagrama klasa ili detaljnog use case dijagrama, ...
- Uloge: Odigrane u okviru metodologije (projektant, tester ili prodavac).
- Edukovanje: Odluka na koji način trenirati osoblje, ukoliko je nephodno, da bi odigrali svoje uloge; odluka na koji način će krajnji korisnici naučiti da koriste sistem.

2.1. **SSA**DM[1] – **S**trukturna **S**istem **A**naliza i **D**izajn **M**etod:

1. **Analiza sistema**
   Poznata i kao „Studija izvodljivosti“. Koristi Dijagram Toka Podataka za opis rada sistema i vizualizaciju poznatih problema.
   Obuhvata sledeće korake:
   a. **Razvoj *Business Activity Model*-a**. Poslovni događaji i pravila se istražuju kao ulaz u specifikaciju novog automatizovanog sistema.
   b. **Pronalaženje (Ispitivanje) i definisanje zahteva**. Identifikuju se problemi u vezi sa okruženjem koji će biti rešeni od strane novog sistema.
   c. **Ispitivanje procesiranja**. Istražuju se tokovi informacija u saradnji sa postojećim uslugama i opisuju u formi **M**odela **T**oka **P**odataka. U datom stadijumu MTP predstavlja tekuće servise (usluge) sa svim nedostacima.
   d. **Ispitivanje tekućih podataka**. U ovom koraku se identifikuje i opisuje struktura podataka sistema nezavisno od načina trenutnog skladištenja i organizovanja podataka.Ovaj korak rezultira kreiranjem modela podataka koji podržava tekuće servise (usluge).
   e. **Izvođenje logičkog pogleda tekućeg servisa**. Razvija se logički pogled na dati sistem kako bi se omogućilo razumevanje problema tekućeg sistema.

2. **Opis poslovne specifikacije**
   Poznata i kao logička etapa specifikacije sistema. Sastoji se iz dva dela. Prvi deo je **istraživanje postojećeg okruženja** i obuhvata identifikovanje zahteva sistema i modelovanje poslovnog okruženja. Modelovanje čini kreiranje DTP i **L**ogičke **S**trukture **P**odataka za procese i strukture podataka koji su deo sistema. Drugi deo se sastoji u **odabiru i izgradnji jedne od 6 Poslovnih Opcija Sistema** (Business Systems Options). Sledeći koraci su deo etape:
   a. **Definisanje BSO**. Identifikuje se broj mogućih rešanja sistema koji udovoljavaju definisanim zahtevima od kojih će korisnik načiniti odabir.
   b. **Odabir BSO**. Vrši se prikazivanje BSO korisnicima i zatim korisnik odabira željenu opciju.

3. **Detaljna specifikacija sistema**
   Poznata kao specifikacija zahteva. Obuhvata sledeće korake:
   a. **Definisanje zahtevanog procesiranja sistema**. Ovaj korak teži ka prilagođavanju zahteva kako bi reflektovali odabran BSO , opisali zahtevani sistem u pogledu tokova podataka sistema, i definisali korisničke uloge novog sistema.
   b. **Razvoj modela podataka**. Ovaj korak se odvija paralelno sa prethodnim. Logički model podataka datog okruženja se prosiruje kako bi podržao sva procesiranja u odabrana u BSO.
   c. **Izvođenje funkcija sistema**. Za vreme paralelnog definisanja podataka i procesa, dodatni događaji su identifikovani, što je dovelo do ažuriranja postojećih i definisanja novih funkcija.

---

[1] Definicija metode SSADM preuzeta sa http://en.wikipedia.org/wiki/Structured_Systems_Analysis_and_Design_Methodology

d. **Razvoj specifikacije korisničkih poslova**. Razvijen je Work Practice Model kako bi se dokumentovalo razumevanje poslova korisnika u potpunosti.
e. **Poboljšanje modela podataka.** Poboljšanje kvaliteta logičkog modela podataka datog sistema primenom relacione analize podataka (poznate i kao normalizacija)
f. **Razvoj prototipa specifikacije**. Opisuje odabrane delove sistema u prilagodljivoj formi demonstracije za korisnika.
g. **Razvoj specifikacija obrade** (procesiranja).
h. **Potvrda sistemskog cilja** (pravca sistema)

4. **Logički dizajn podataka**
   Poznat i kao logička specifikacija sistema. Vrši se odabir opcija koje su tehnički naj izvodljivije. Razvojno/implementaciona okruženja se specificiraju na osnovu ovog izbora.
   Faza je sastavljena iz :
   a. **Definisanja BSO**. U ovom slučaju identifikuju se i definišu mogući pristupi fizičkoj implementaciji kako bi se udovoljilo definiciji funkcija.Takođe se vrši validacija zahteva na nivou servisa za predloženi sistem u svetlu tehničkog okruženja.
   b. **Odabir BSO.** Ovaj korak obuhvata prezentaciju BSO-a korisnicima i odabir željene opcije.

5. **Logički dizajn procesa**
   Takođe poznat kao logička specifikacija sistema. Vrši se ažuriranje logičkog dizajna i procesa. Moguća je i specifikacija dijaloga.
   Obuhvata :
   a. **Definisanje korisničkih dijaloga**. Definiše se struktura svakog dijaloga.
   b. **Definisanje procesa ažuriranja**. Specificira se postupak ažuriranja za svaki događaj i definiše obrada greške za svaki događaj.

6. **Fizički dizajn**
   Cilj ove faze je specifikacija fizičkog dizajna podataka i procesa, korišćenjem rečnika i karakteristika odabranog fizičkog okruženja i inkorporiranih standarda.
   Delovi etape:
   a. Priprema za fizički dizajn
      - Korištenje pravila implementacionog okruženja;
      - Pregled preciznih zahteva za logičko i fizičko mapiranje;
      - Planski pristup;
   b. Kompletiranje specifikacije funkcija
   c. Inkrementalno i repetitivno razvijanje dizajna podataka i procesa.

Tri najbitnije tehnike koje se koriste u SSADM su:

1. **Logičko modelovanje podataka** – identifikacija, dokumentovanje, modelovanje zahteva sistema koji se dizajnira.
2. **Modelovanje toka podataka** - identifikacija, dokumentovanje, modelovanje prenosa podataka u informacionom sistemu.
3. **Modelovanje ponašanja entiteta** - identifikacija, dokumentovanje, modelovanje događaja koji utiču na svaki entitet i sekvenci u kojima se „okidaju“ događaji.

U prvom prilogu diplomskog rada možete pogledati seminarski rad iz softverskog inženjerstva „Informacioni sistem šahovskog kluba“[2] koji je osnova ovog diplomskog rada i koji je realizovan primenom konvencionalnog pristupa gde su modeli:

- Sturkturna sistemska analiza kao funkcionalni model sistema
- Model objekti-veze kao model baze podataka
- Model složenog objekta kao modela aplikacija

,gde je primenjen transformacioni životni ciklus:

- Analiza sistema (SSA)
- Specifikacija sistema (MOV)
- Projektovanje (Logičko i fizičko projektovanje baza podataka i programa)
- Implementacija (Baza podataka, kodiranje, testiranje)

Na pratećem CD-u su priloženi i dijagrami odrađeni u PowerDesigneru verzijama 6 (Process Analyst za .pam –dijagrami tokova podataka SSA) i 10 (Data Arhitekt – dijagrami logičke i fizičke baze podataka .cdm, .pdm MOV) kao i ostali dijagrami rađeni u Power Designeru 12.

[2] Predmetni nastavnik Prof. Dr Dragica Radosav, student Miloš Kecman; rad realizovan na T.F.“Mihajlo Pupin“ u Zrenjaninu septembra 2006.godine

2.2. Objektno orijentisana metodologija

Globalno govoreći, sve objektno orijentisane metodologije su slične, imaju slične faze i artifakte, ali postoje brojne male razlike.
Namera OOM nije da propisuje pravila: projektanti imaju brojne mogućnosti i slobode izbora oko korištenja dijagrama ili nekih drugih artifakta. Međutim, projektni tim mora izabrati jednu metodologiju i složiti se oko artifakta koje će stvarati, pre nego što počne bilo kakvo detaljno planiranje ili popis.
Generalno, svaka metodologija upućuje na:

- Filozofiju iza svake faze
- Poslovne tokove i individualne aktivnosti svake faze
- Artifakte koje treba proizvesti (dijagrami, tekstualni opisi i kod)
- Zavisnosti između artifakta
- Potrebu za modelovanjem statičke strukture i dinamičkog ponašanja

Statičko modelovanje obuhvata delovanja nad logičkim i fizičkim delovima sistema i načinu na koji će oni biti povezani. Dinamičko modelovanje je posvećeno izboru načina kolaboracije statičkih delova. Grubo govoreći, statički model opisuje konstrukciju i inicijalizaciju sistema, dok dinamički model opisuje kako će se sistem ponašati za vreme rada. Uglavnom se proizvede jedan statički i jedan dinamički model u toku svake faze modelovanja.

UML, RUP i XP
Godine 1996, *Jacobson* i *Rumbaugh* pridružili su se Rational Corporation (osnovane od strane *Booch*-a), i razvili set notacija koje su postale poznate kao ***Unified Modeling Language***[3] (UML). Neki projektanti svrstavaju UML kao notaciju za *brainstorming* i dokumentovanje na visokom nivou, drugi pak svrstavaju UML u slikovni programski jezik, koji generiše nov kod iz postojećeg koda.
Cilj jednistvenog jezika modeliranja (UML-a), kako su ga postovali njegovi autori, je višestruk:

- da predstavi kompletni sistem (ne samo softverski deo) korišćenjem OO koncepata;
- da uspostavi eksplicitnu vezu između koncepata i izvršnog koda
- da uzme u obzir faktore skaliranja koji su ugrađeni u kompleksne i kritične sisteme
- da bude sredstvo modeliranja upotrebljivo i za čoveka i za mašinu

Rational Unified Process je spiralni, iterativni, inkrementalni metod nastao od tvoraca UML-a. Može se reći da je to metodologija koja izdvaja najbolje aspekte UML-a.

Još jedna popularna metodologija je **eXtreme Programming** [*Beck 99*] smatra se brzom,efikasnom metodologijom jer dozvoljava i reaguje na promene. XP se odlikuje dvema osnovnim idejama: par programera i projektovanje navođeno testiranjem.

[3] U radu je prikazana UML notacija i njena konkretna implementacija kroz primere nad SPPR Šahovskih Klubova

Programeri rade, pre svega, na poboljšanju kvalieta softvera i na testiranju ali tako da test bude napisan i pre samog koda.

UML Standard koji se primenjuje kod objektno orijentisanog pristupa predviđa sledeće poglede na sistem, s tim što se u svakom pogledu sistem može opisati sa statičkog (strukturnog) i dinamičkog aspekta:

- **Pogled slučajeva korišćenja(Use-case view)** - predstavlja skup slučajeva korišćenja. Predstavlja se preko dijagrama slučajeva korišćenja - statički aspekt. Dinamički aspekt sistema se ovde daje bilo verbalnim opisom slučajeva korišćenja, ili formalnim opisom, preko dijagrama interakcije, dijagrama aktivnosti i dijagrama promene stanja.

- **Projektni pogled (Design view):** Statički aspekt sistema se ovde prikazuje preko dijagrama klasa i dijagrama objekata, a dinamički aspekt preko dijagrama interakcije, dijagrama promene stanja i dijagrama aktivnosti.

- **Procesni pogled (Process view)-** prikazuje "niti upravljanja" i procese preko kojih se ostvaruje konkurentnost i sinhronizacija procesa u sistemu. Preko ovoga pogleda prvenstveno se analiziraju performanse, skalabilnost i propusnost sistema. Predstavljaju se sa istom vrstom dijagrama kao i projektni pogled, samo se akcenat stavlja na "aktivne klase"- klase koje reprezentuju procese i niti.

- **Implementacioni pogled (Implementation view) -** prikazuje komponente i fajlove sa kojima se sistem realizuje. Statički aspekt se prikazuje dijagramima komponenti, a za dinamički se koriste dijagrami interakcije, dijagrami promene stanja i dijagrami aktivnosti.

- **Pogled razmeštaja (Deployment view) -** prikazuje sistemsko- hardversku topologiju. Prikazuje se distribucija hardverskih komponenti i instalacija softverskih komponenti na njima. Statički aspekt se opisuje dijagramima razmeštaja, a za dinamički se ponovo koriste dijagrami interakcije, dijagrami promene stanja i dijagrami aktivnosti.

2.3. Klasične faze u razvoju softvera

Postoje brojne faze koje su zajedničke za razvoj softvera, u zavisnosti od metodologije, počevši od snimanja zahteva pa do održavanja. Tradicionalni pristup omogućava prelaz sa jedne faze na drugu. Moderni, međutim, dozvoljava izvođenje svake faze više puta i u bilo kom redosledu. Sledeća lista objašnjava najčešće faze u izradi softvera [5].

1. Spisak zahteva[4]
   otkriva šta želimo da postignemo sa novim softverom i ima dva aspekta:
   - Poslovno modelovanje (Business modeling) – razumevanje konteksta u kom će novi softver raditi
   - Funkcionalna specifikacija – definisanje mogućnosti softvera. Šta će novi softver raditi, a šta neće tako da na osnovu toga možemo proceniti kada smo završili i da li smo zadovoljni novim softverom.

2. Analiza
   Omogućava da spoznamo „sa čime imamo posla". Pre nego što dizajniramo rešenje moramo biti svesni relevantnosti, svojstava i veza. Takođe moramo biti u stanju da dokažemo naše razumevanje. Ovo može uključiti kupce i krajnje korisnike jer oni jesu eksperti za datu oblast.

3. Dizajn
   U ovoj fazi razmatramo kako rešiti problem. Donosimo zaključke bazirane na iskustvu, proceni, intuiciji, kakav ćemo softver napisati i kako ćemo ga razviti. Sistemski dizajn razbija sistem na logičke podsisteme (procese) i na fizičke podsisteme (kompjuteri i mreže), odlučuje kako će mašine komunicirati, odabira korektnu tehnologiju za posao,... U dizajniranju podsistema odlučujemo kako prevesti logički podsistem u koristan,efektivan i praktičan kod.

4. Specifikacija
   Specifikacija je jedna od često ignorisanih i zanemarenih faza u procesu razvoja.Sam termin „specifikacija" se koristi u različitom kontekstu što zavisi od projektanata.Npr. izlaz iz faze „spisak zahteva" je specifikacija koja kazuje za šta je sistem osposobljen , izlaz iz analize je specifikacija koja nam ukazuje sa čime radimo „sa čim imamo posla".Specifikacija klase je čist, precizan opis korištenja komponenti našeg softvera, njihovog ponašanja ukoliko se koriste pravilno. Specifikacija se koristi u sledećim slučajevima:
   - Kao osnova za dizajniranje test softvera za ispitivanje sistema
   - Za demonstraciju korektnosti,ispravnosti softvera
   - Za dokumentaciju softverskih komponenti i okvira u kojima se mogu koristiti od strane trećeg lica
   - Za opis ponovnog korištenja koda u okviru drugih aplikacija

Faze koje spadaju u klasične faze razvoja softvera, a u ovom radu ih neću podrobno opisivati jesu testiranje, uvođenje i održavanje

[4] Neke od metodologija kombinuju Spisak zahteva i Analizu, dok druge kombinuju Analizu i Dizajn.

2.4. Softverski inženjering i metoda vodopada

Tokom 70'ih programeri su tražili najbolji način izrade softvera (pisanja softvera),kako bi na najprikladniji način zamenili *ad hoc* mehanizam sa skalabilnim i prenosivim metodologijama.
Javila se ideja [2] da izrada softvera može biti nalik izradi zgrade ili nekog drugog objekta. Međutim, odmah se moglo videti da ta analogija ne odgovara u potpunosti stvarnoj slici. Programeri moraju uz pomoć programskih jezika precizno saopštiti računaru šta treba da čini unoseći izjavu po izjavu, granu po granu, funkciju po funkciju, što je analogno sa inžinjerom koji gradi most i saopštava ciglama, čeliku kako da se ponašaju i na koji materijal i predmet treba da se oslone (kao što programer čini asembliranjem podataka), što naravno deluje smešno i pomalo zaglupljujuće.

Ovi nedostaci ipak nisu sprečili rastuću popularnost softverskog inženjerstva i metodologija koja se zasnivala na pretpostavci da razvoj softvera može biti sistematski i predvidiv. Ovo je dovelo do razvoja „Metodologije vodopada".
Razvoj zahvata klasične faze (Spisak zahteva,Analizu,Dizajn,...) koje se moraju završavati respektivno pre početka sledeće faze. Ovakav razvoj je lako isplanirati (svaki put je sličan plan razvoja) i proračunati (kompleksnost problema , broj projektanata koji rade i trajanje razvoja deli se sa brojem faza...) Metodologija vodopada nam omogućava da radimo sa projektantima koji su experti u različitim oblastima. Svi timovi stručnjaka rade svoj posao dok u potpunosti nisu sigurni da su ga odradili korektno, zatim dokumentuju svoj rad koristeći sopstveni žargon i notaciju, i prosleđuju ga sledećem timu u nizu (na kaskadi). Ipak, iako je dobra ideja u osnovi, metodologija vodopada nije realistična. Jer čak i da proračunamo trajanje projekta, pre pregledanja detalja problema, ne možemo znati na kakve ćemo poteškoće nailaziti svo vreme (loše odluke u dizajnu, pogubne greške, neadekvatna tehnologija, ili možda zemljotres).

Kako bismo prevazišli neke od ovih problema i kako ne bismo isporučili proizvod koji bi bio neprijatno iznenađenje za korisnika, moramo pronaći način da povežemo i uključimo korisnika u postupak projektovanja. Takođe moramo pronaći način da veoma brzo prikažemo funkcionalnost softvera korisniku kako bismo uz njegove korekcije završili postupak. Nažalost Metod vodopada je previše strog da bi dozvolio komunikaciju sa korisnikom i sproveo potrebne korekcije.
Metod vodopada pokušava da u samo jednom prolazu kroz faze proizvede kompletno rešenje.
Iako ima brojne mane, metodologija vodopada je korisna u sledećim slučajevima:

- Kada je potrebno isporučiti projekat koji se nebitno razlikuje od prethodno razvijenog projekta.
- Kao kostur za učenje drugih tehnika u softverskom inženjerstvu.
- Kao jedan prolaz kroz spiralnu metodologiju
- Kao kostur za podršku iterativne metodologije
- Za brz razvoj malih projekata sa malim brojem projektanata (Rapid Application Development, prototyping, production prototyping)

3. Stvaranje sistema

Pre nego što uopšte i počnemo sa razmatranjem softvera koji ćemo napisati, moramo istražiti poslove u kojima će se naš softver primenjivati – bez razumevanja posla, teško da možemo očekivati da će naš softver poboljšati poslovanje.

Kada smo razumeli poslove i dokumentovali naše shvatanje poslova u **business requirements** (poslovni zahtevi), moramo razmisliti o svrsi softvera u odnosu na korisnike. Bez razumevanja **system requirements** (sistemskih zahteva), rizikujemo da stvorimo kod koji neće biti nikome od koristi.

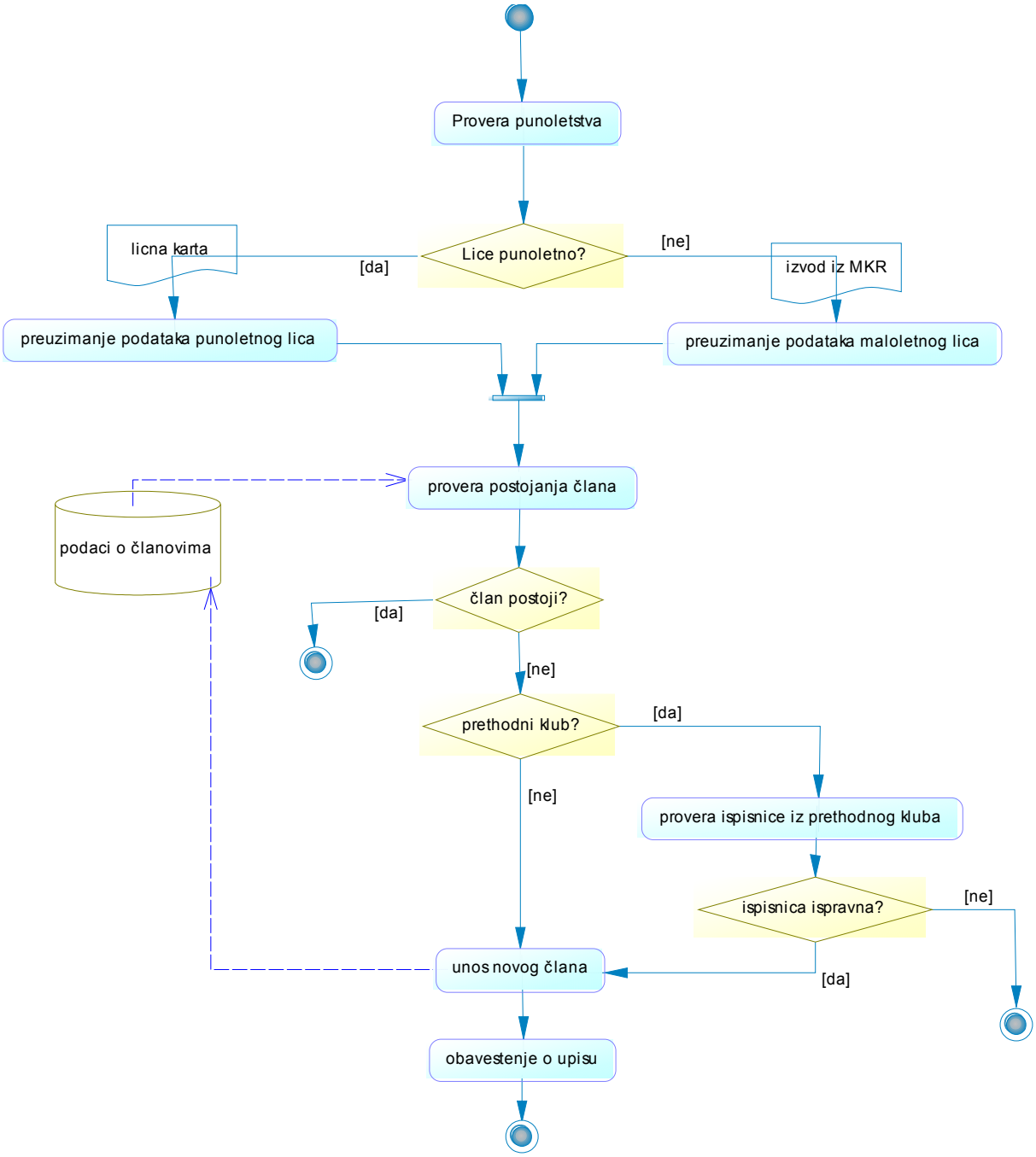
Ukoliko imamo sreće, pre nego što počnemo sa razvojem sistema, možemo dobiti detaljnu dokumentaciju od strane kupca („sponzora projekta“) sa solidnom osnovom i sadržajem.Ili možemo naš razvoj temeljiti na **mission statement** , kratko uputstvo o novom, željenom poslovnom pravcu.
Kao projektanti, moramo transformisati dokument korisničkih zahteva ili *mission statement* u kompletan, nedvosmislen opis sistema koji ćemo projektovati u standardni format koji kupci-sponzori mogu razumeti i odobriti.

3.1. Poslovni model

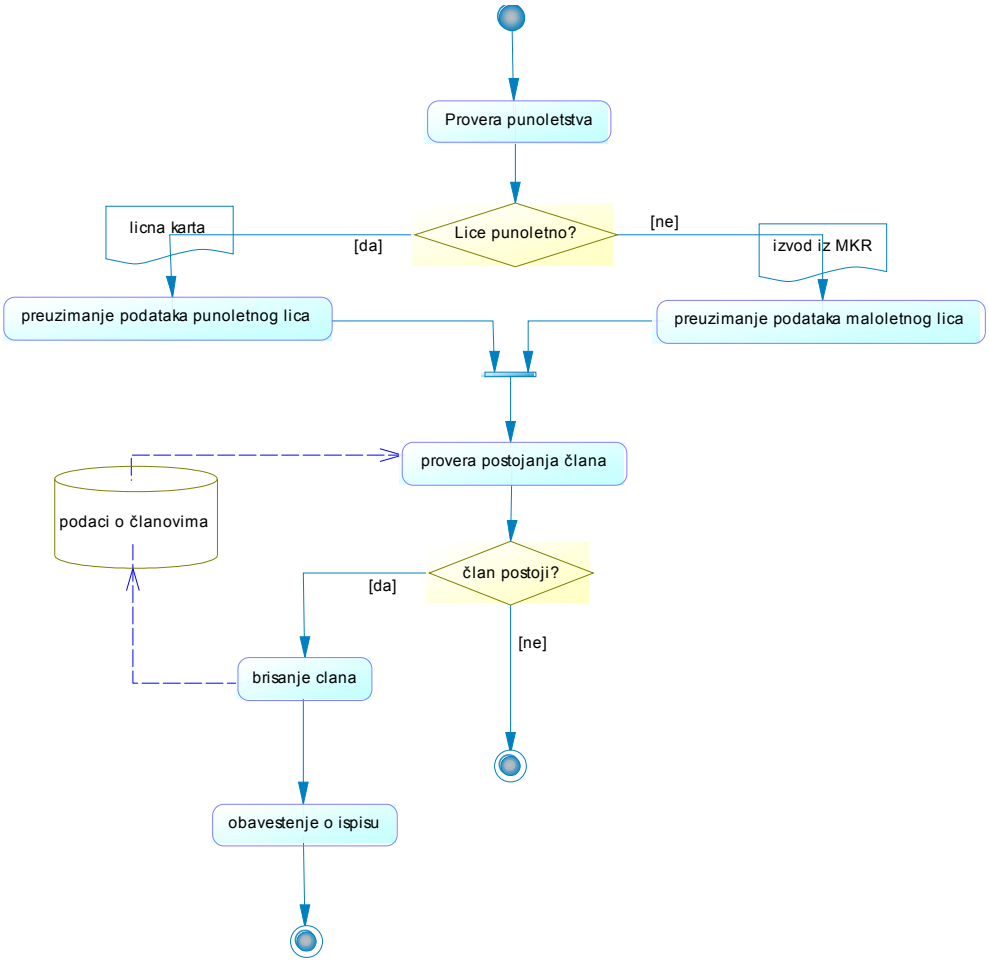
Business model (poslovni model) je prethodnica modeliranju funkcionalnosti sistema. Poslovni model može biti jednostavan kao dijagram klasa, pokazujući veze između poslovnih entiteta – ponekad se tretira kao **domain model** (područni model). Područni model može biti dovoljan za male projekte, ali za većinu projekata se izrađuje poslovni model koji prikazuje poslovanje celokupnog sistema, ili bar poslovanje nekih delova koji okružuju sistem koji projektujemo.
Business Process Model (BPM) se koristi za izradu poslovnog modela sistema i predstavlja složeniju alternativu Use Case modelima. BPM je konceptualni model koji omogućava opis poslovne logike i pravila sa korisničke tačke gledišta. Prikazuje interakciju između procesa, tokova, poruka i kolaboracionih protokola od jedne ili više početnih tačaka do nekoliko potencijalnih krajnjih tačaka.

Dat je prikaz nekoliko BPM dijagrama realizovanih u PowerDesigneru verzije 12.1.0.1913 na kojima je prikazan opis poslovne logike šahovskih klubova. Ostale dijagrame možete pronaći u prilogu br.2 na pratećem CD-u.

Evidencija igrača:

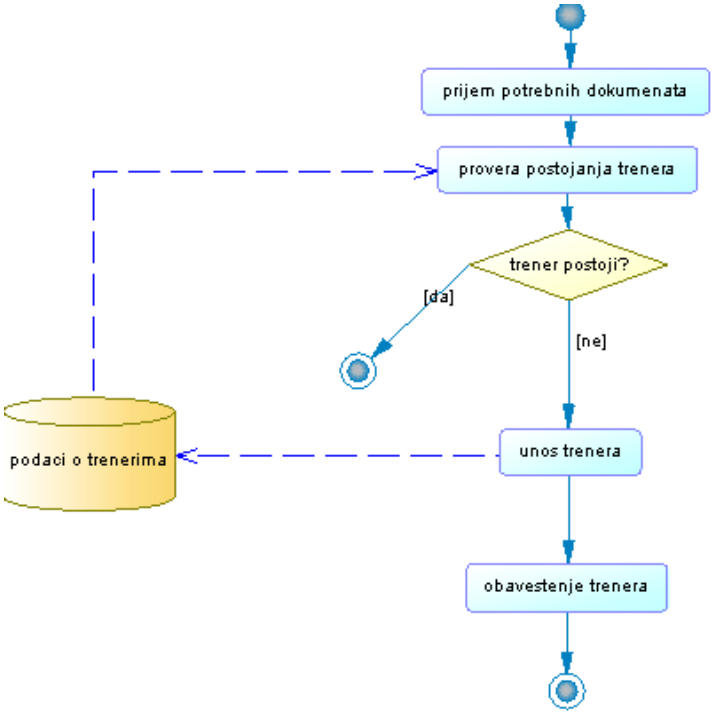


Slika1. Unos člana

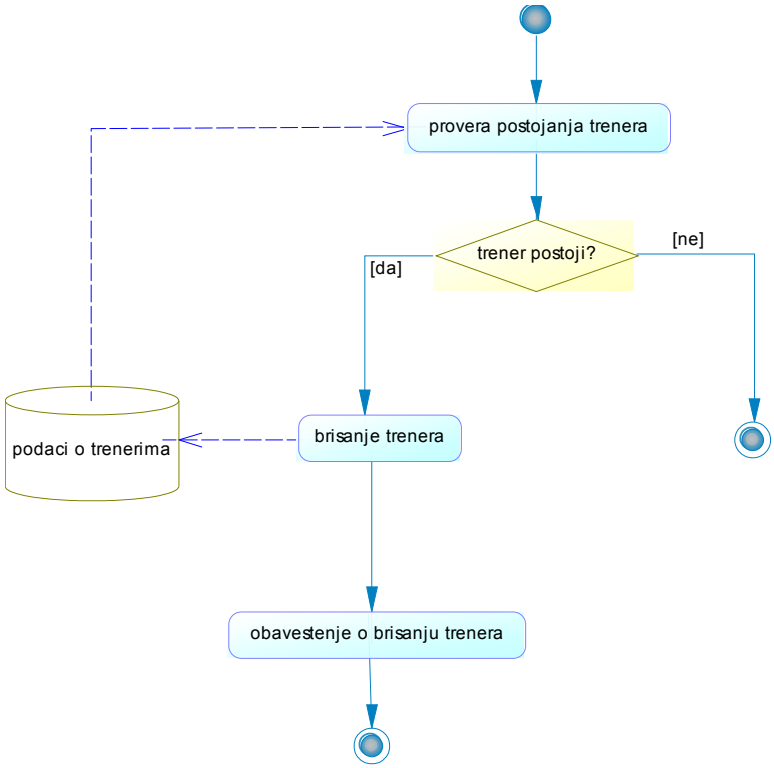


Slika2. Brisanje člana

Evidencija trenera:

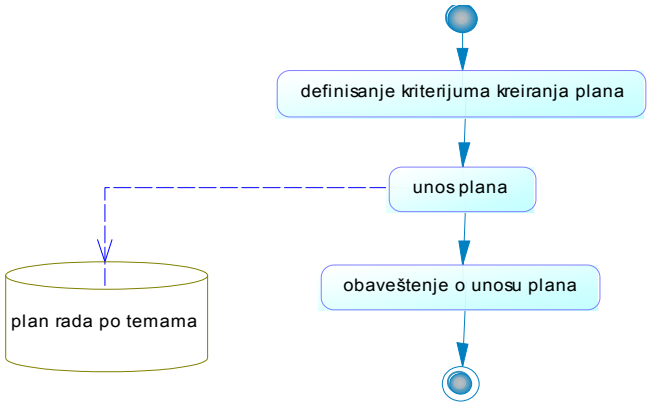


Slika3. Unos trenera

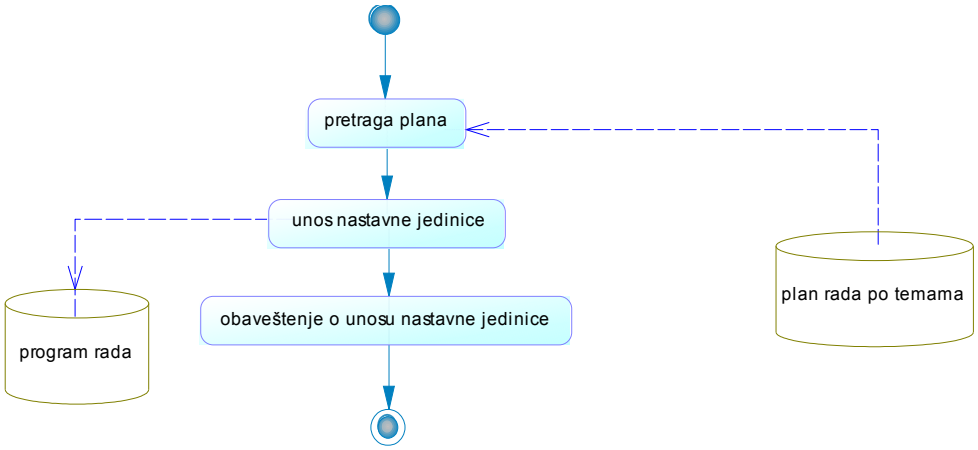


Slika4. Brisanje trenera

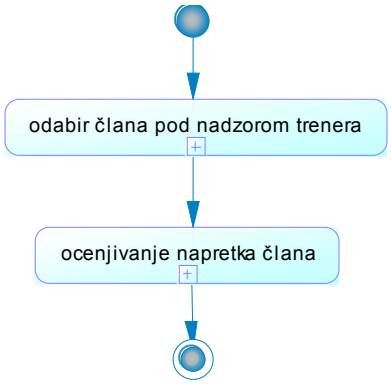
Plan i program:



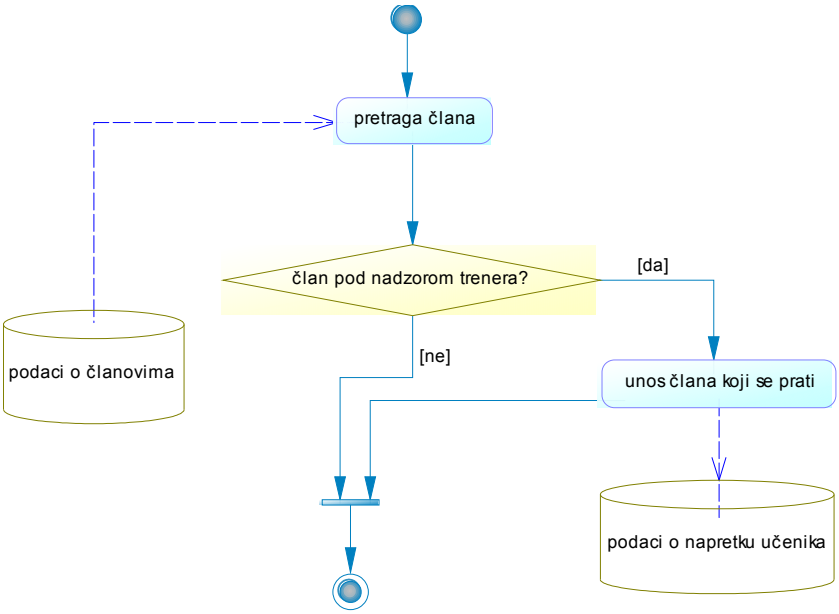
Slika5. Unos plana



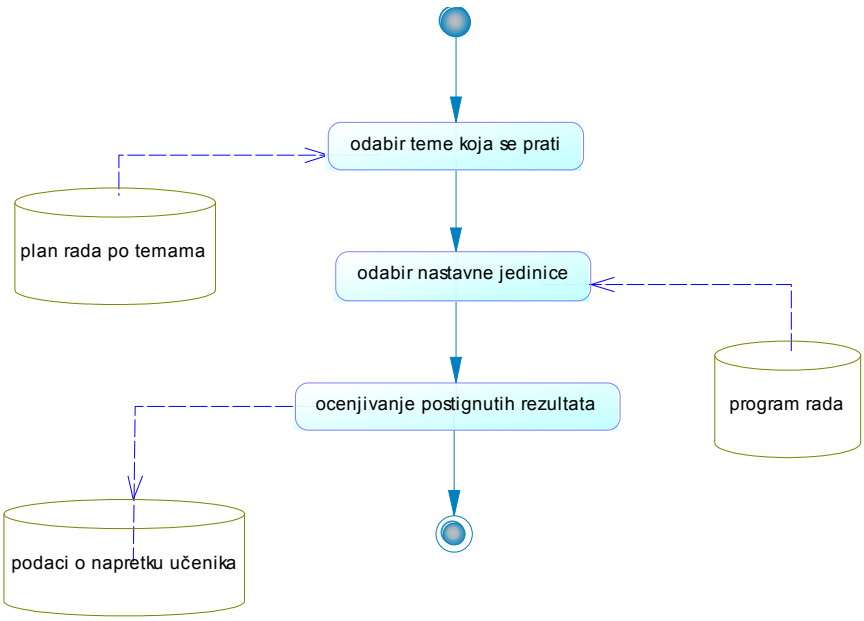
Slika6. Unos programa



Slika7. Unos napretka



Slika8. Odabir člana pod nadzorom trenera

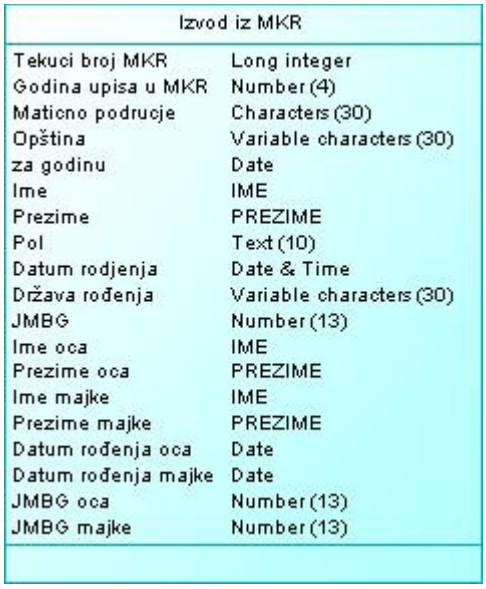


Slika9. Ocenjivanje napretka člana

***

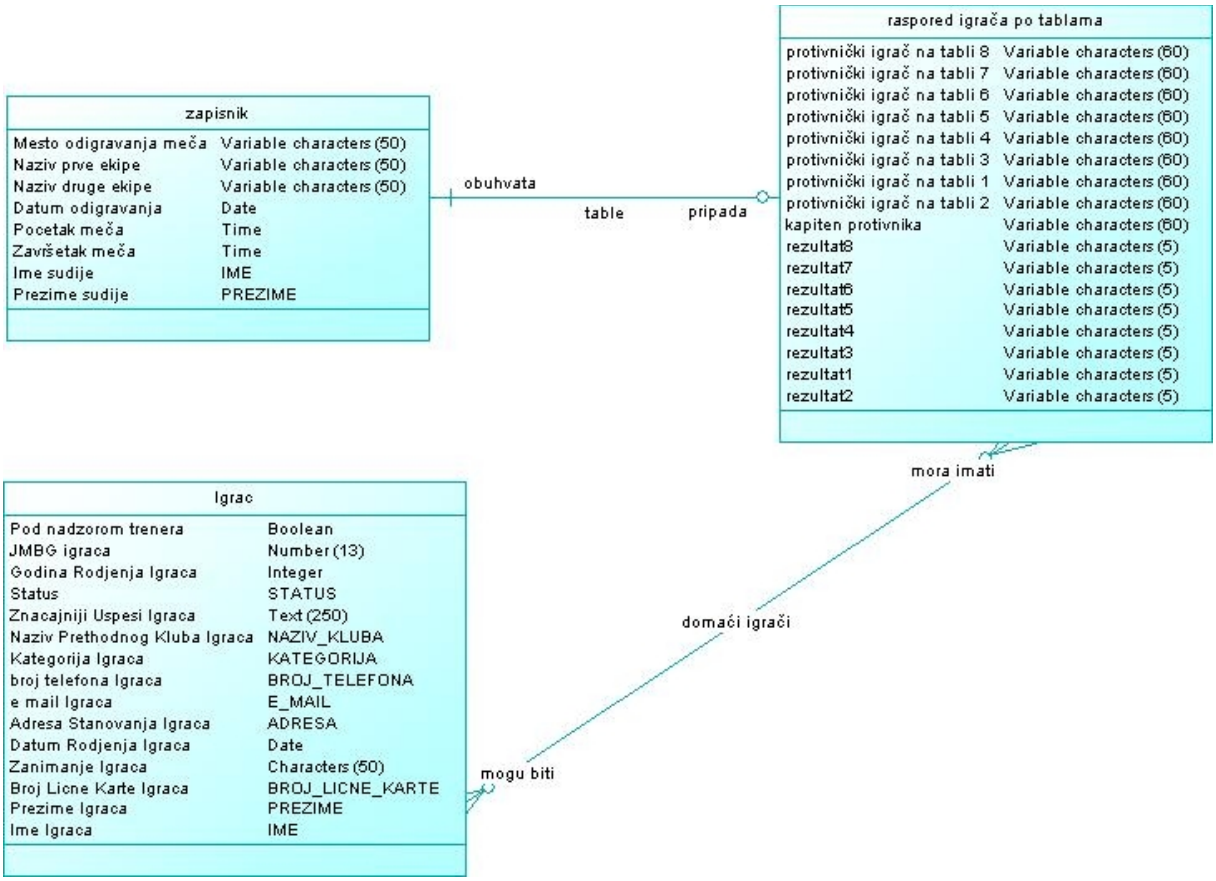
Dokumenti sistema

Dokumenti sistema predstavljeni konceptualnim modelima:

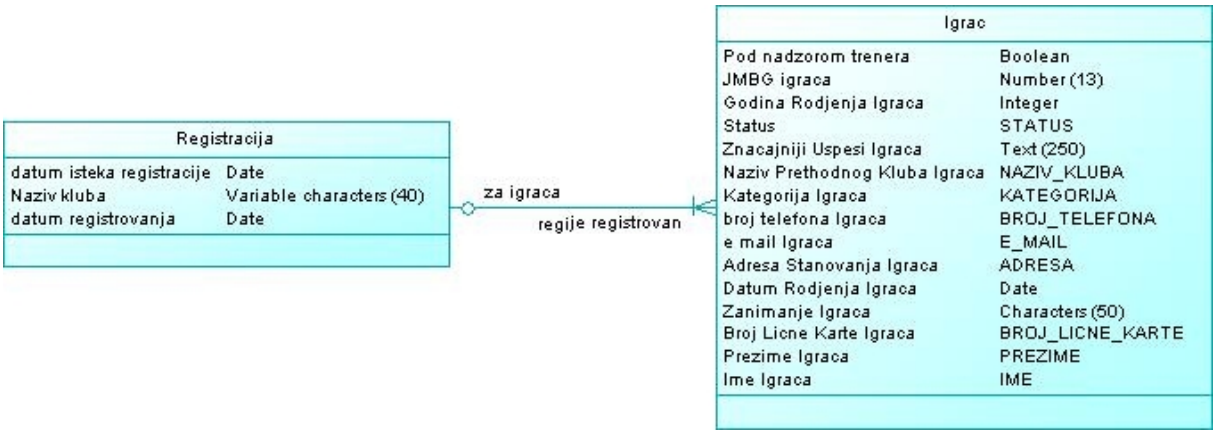


Slika10. Izvod iz matične knjige rodjenih

Slika11. Lična karta



Slika12. Zapisnici sa sudjenja



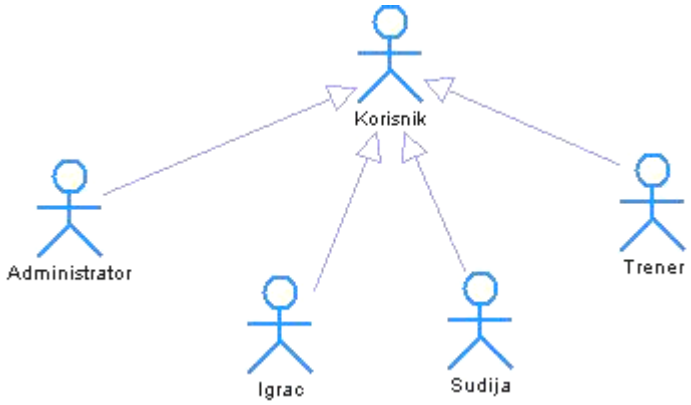
Slika13. Registracije igrača

Nakon dokumentovanja posla u BPM, opšte je prihvaćeno da se zahtevi sistema prezentuju i preko Use Case modala. Razlog za to je što se Use Case relativno lako kreira za prikaz sponzorima [2].

3.1.1. Use Case

Use Case, na ovom nivou (sistemskom), mora biti detaljniji i sa manje slobode od Use Case dijagrama na poslovnom nivou i treba da sadrži:

- listu učesnika (sa opisima)
- listu slučajeva korištenja (sa opisima)
- dijagrame slučajeva korištenja
- detalje slučajeva korištenja
- pregled slučajeva korištenja
- dodatne zahteve (sistemski zahtevi koji ne pripadaju nijednom posebnom Use Case dijagramu)
- Skicu korisničkog interfejsa
- Poboljšan rečnik pojmova
- Prioritete slučajeva korištenja



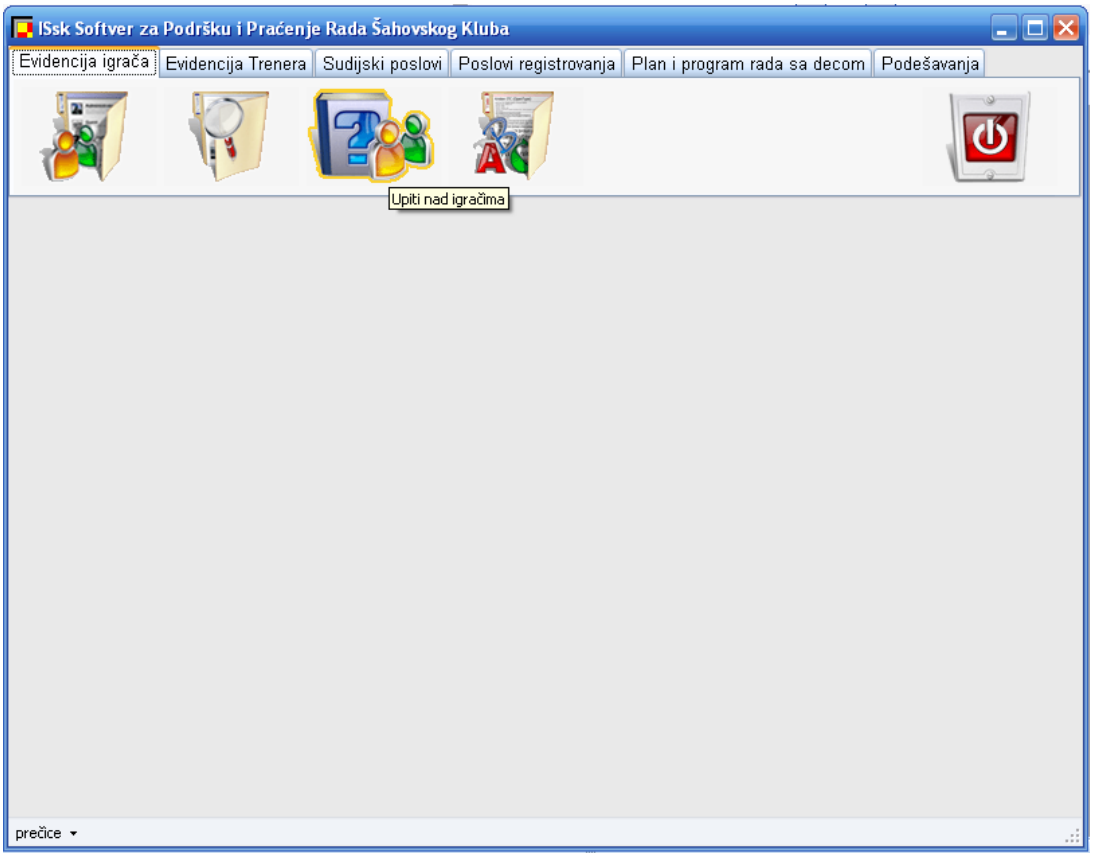
Slika14. Lista učesnika

U korisnike sistema spadaju:

1. **Administrator sistema** - koji ima apsolutan pristup aplikaciji
2. **Sudija** – ima pristup samo kartici sudijski poslovi (unos i pregled zapisnika, unos raspisa)
3. **Trener** – pristupa delu „plan i program rada sa decom“ (unos i pregled plana i programa, evidentiranje napretka učenika)
4. **Igrač** – pristup ograničen na pregled igrača, sudija, trenera, plana i programa

Verbalni opis slučajeva korištenja

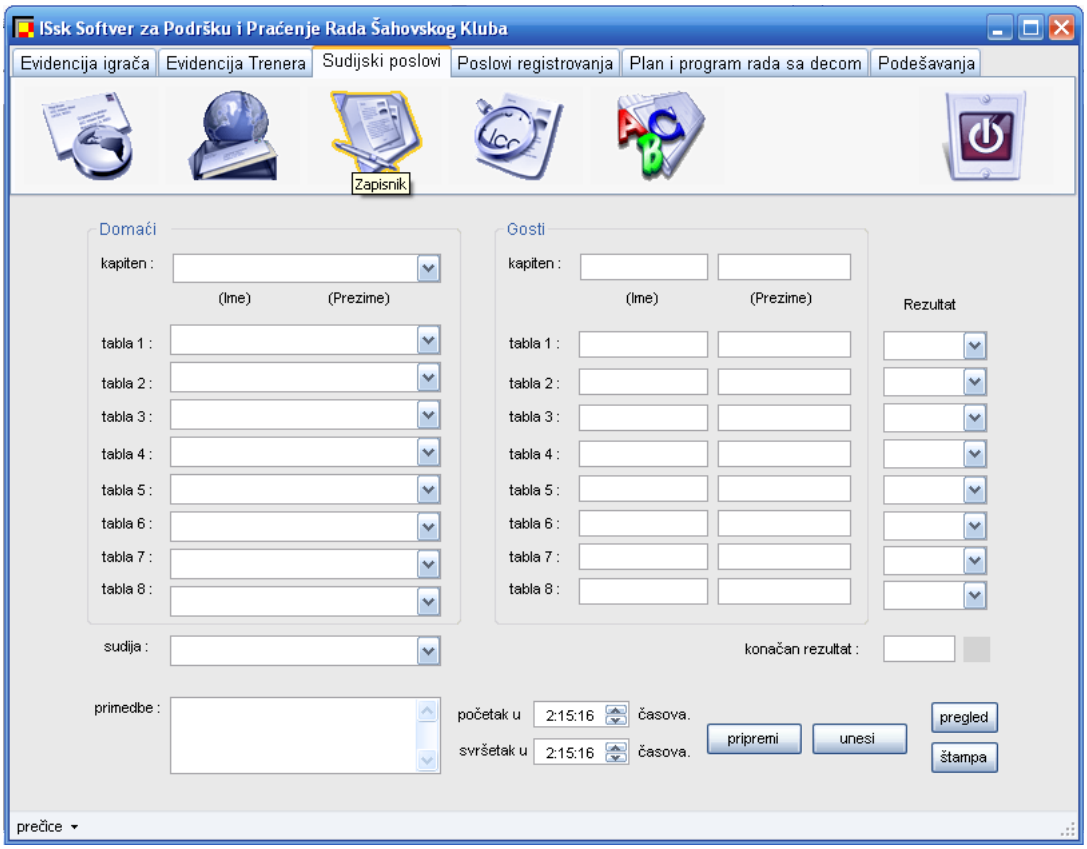
Slučaj korištenja: Glavni meni
Učesnik: Administrator
Svrha: Prikazuje glavni meni
Kratak opis: Sadrži menije za rad sa igračima, sudijama, trenerima, menije za obradu poslova suđenja i registrovanja



Slika15. Glavni meni

Slučaj korištenja: Obrada igrača
Učesnik: Administrator
Svrha: Prikazuju se forme sa opcijama za manipulaciju nad podacima igrača
Kratak opis: Unos igrača, brisanje, ažuriranje, slanje obaveštenja o upisu i ispisu, prikaz svih igrača

Slučaj korištenja: Obrada registrovanja
Učesnik: Administrator
Svrha: Prikazuju se forme sa opcijama za manipulaciju nad poslovima registrovanja
Kratak opis: Unos (skladištenje) obaveštenja o registraciji, slanje podataka na osnovu obaveštenja Šahovskom Savezu Vojvodine o igračima koje je potrebno registrovati, registrovanje kluba, uplata članarine, preuzimanje i skladištenje registracija (takmičarskih knjižica).



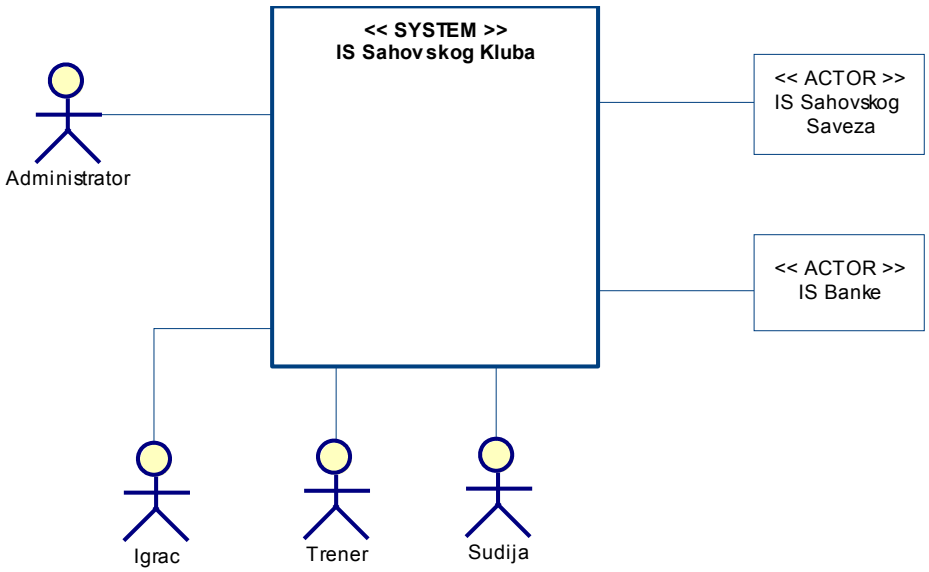
Slika16. Unos zapisnika

Slučaj korištenja: Obrada suđenja
Učesnik: Administrator, sudija
Svrha: Prikazuju se forme sa opcijama za manipulaciju nad poslovima suđenja
Kratak opis: Slanje zahteva za suđenjem sudijama, popunjavanje zapisnika sa suđenja od strane sudije i nadgledanje od strane administratora

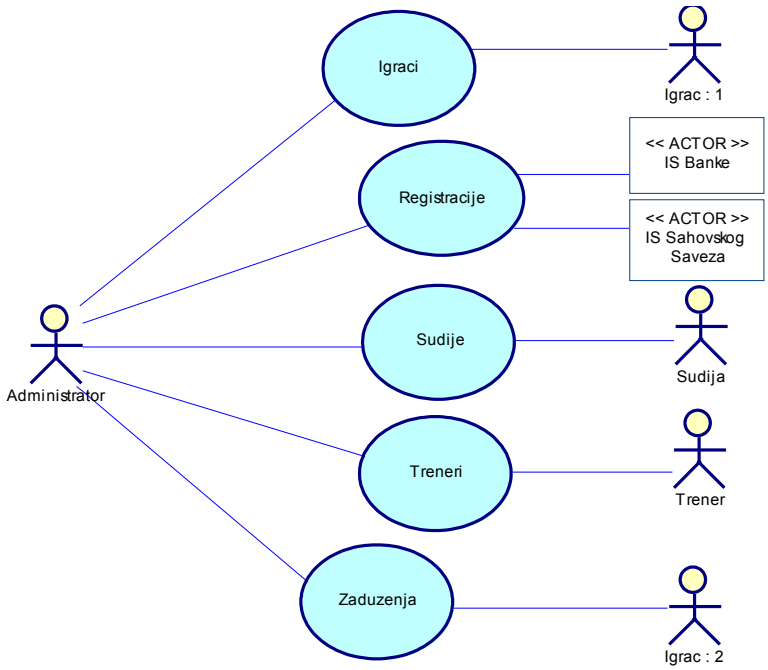
Slučaj korištenja: Poslovi trenera
Učesnik: Administrator, trener
Svrha: Prikazuju se forme sa opcijama za manipulaciju nad poslovima trenera
Kratak opis: Administrator evidentira trenere u bazu trenera, propisuje plan i program koji skladišti u bazu, a odatle ga prosleđuje treneru koji na osnovu plana i programa radi sa igračima i prati napredak koji beleži u bazu koju nadgleda administrator

Slučaj korištenja: Obrada zaduženja
Učesnik: Administrator
Svrha: Prikazuju se forme sa opcijama za manipulaciju nad poslovima zaduženja
Kratak opis: Unosi i briše zaduženja igrača

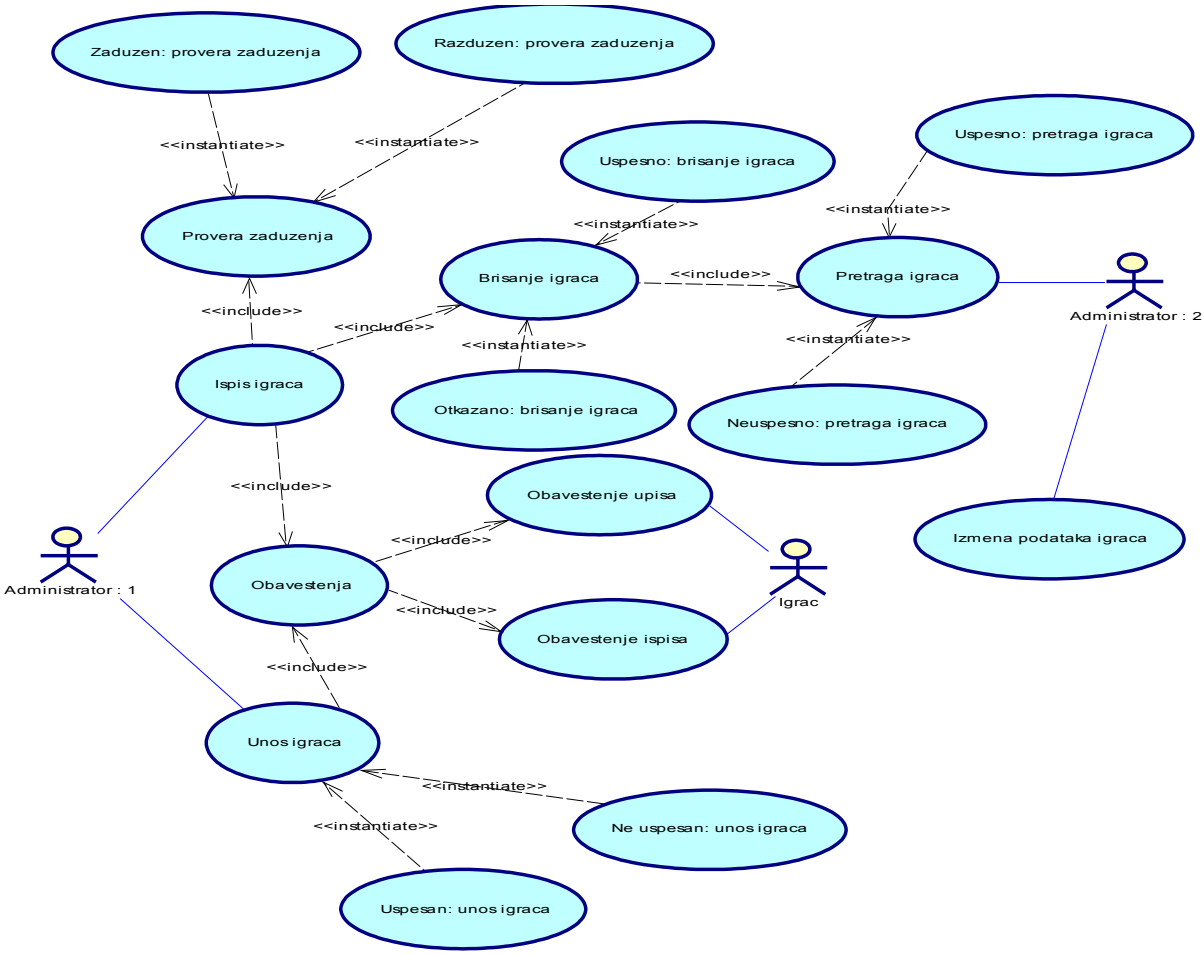
Dijagrami slučajeva korištenja



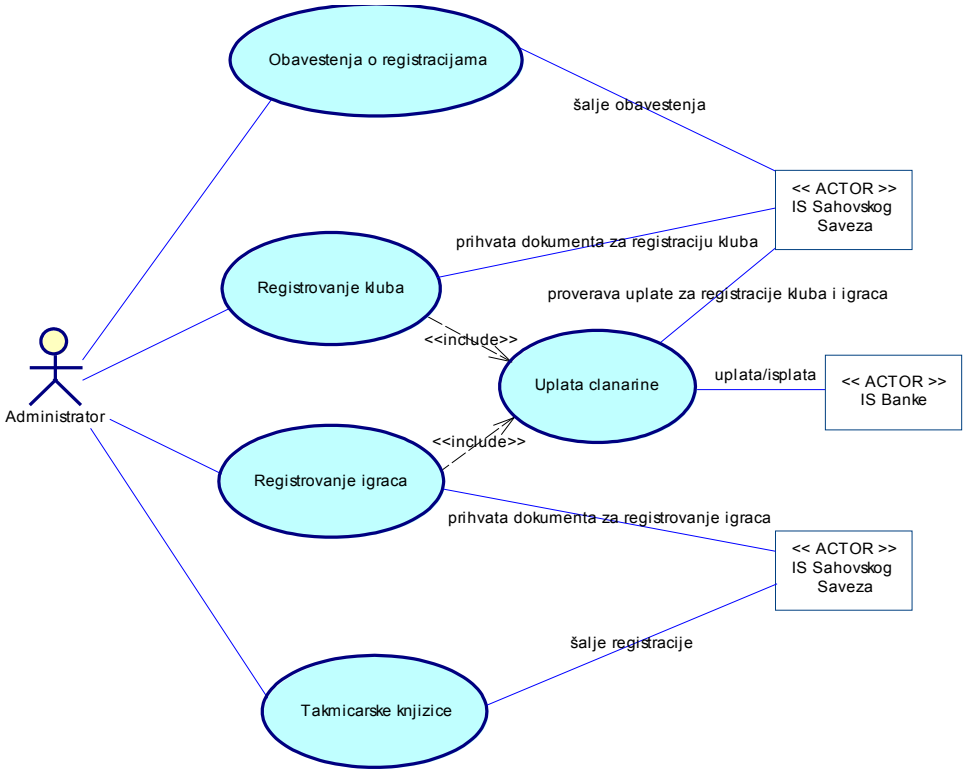
Slika17. I.S. Šahovskog Kluba



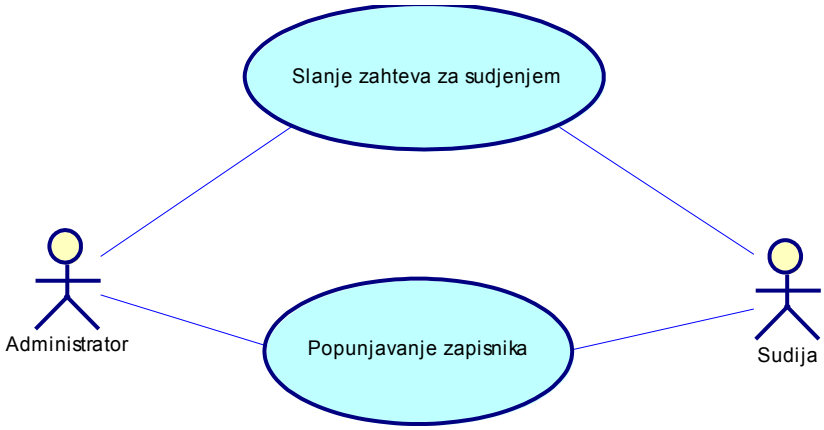
Slika18.- 1. Glavni meni



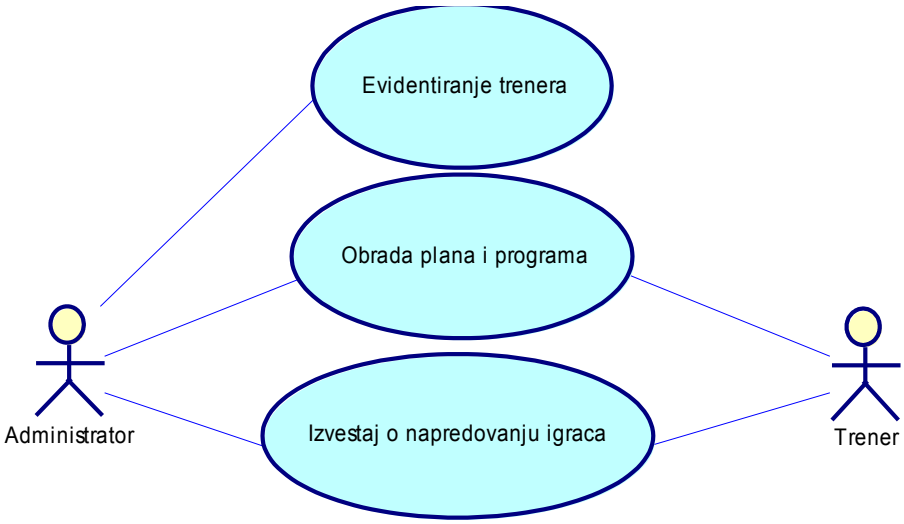
Slika19. -2. Obrada igrača



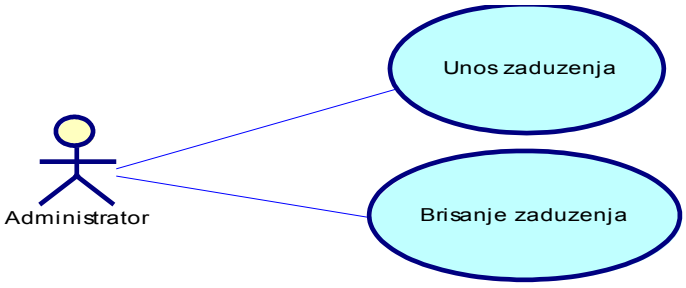
Slika20. -3. Obrada registrovanja



Slika21. -4. Obrada suđenja



Slika22. -5. Poslovi trenera



Slika23. -6. Obrada zaduzenja

4. Analiza

Ne možemo očekivati potpuno razumevanje problema na osnovu *Business Requirements Modela*, jer on opisuje postojeću praksu, a uvođenjem softvera očekujemo uvođenje nove prakse.
Čak iako imamo Use Case model (*System Use Case*) naše razumevanje problema je nekompletno jer *use case „*upravlja“ sa interakcijama između učesnika i granice sistema – sistem se posmatra kao crna kutija sa spoljašnjom vidljivošću. I pored toga Business
Requirements Modeling (omogućava razumevanje poslovnog konteksta) i System Requirements Modeling (definiše naš ugovor između sponzora) moraju biti urađeni.
Nakon odrađene statičke analize, experti će biti u mogućnosti da procene naše razumevanje poslovnih objekata pre nego što ono izvrši uticaj na naš dizajn (model statičke analize je koristan i za dizajniranje šeme baze podataka). Nakon odrađene dinamičke analize, možemo biti sigurni da naši analizirani objekti podržavaju traženu funkcionalnost sistema.

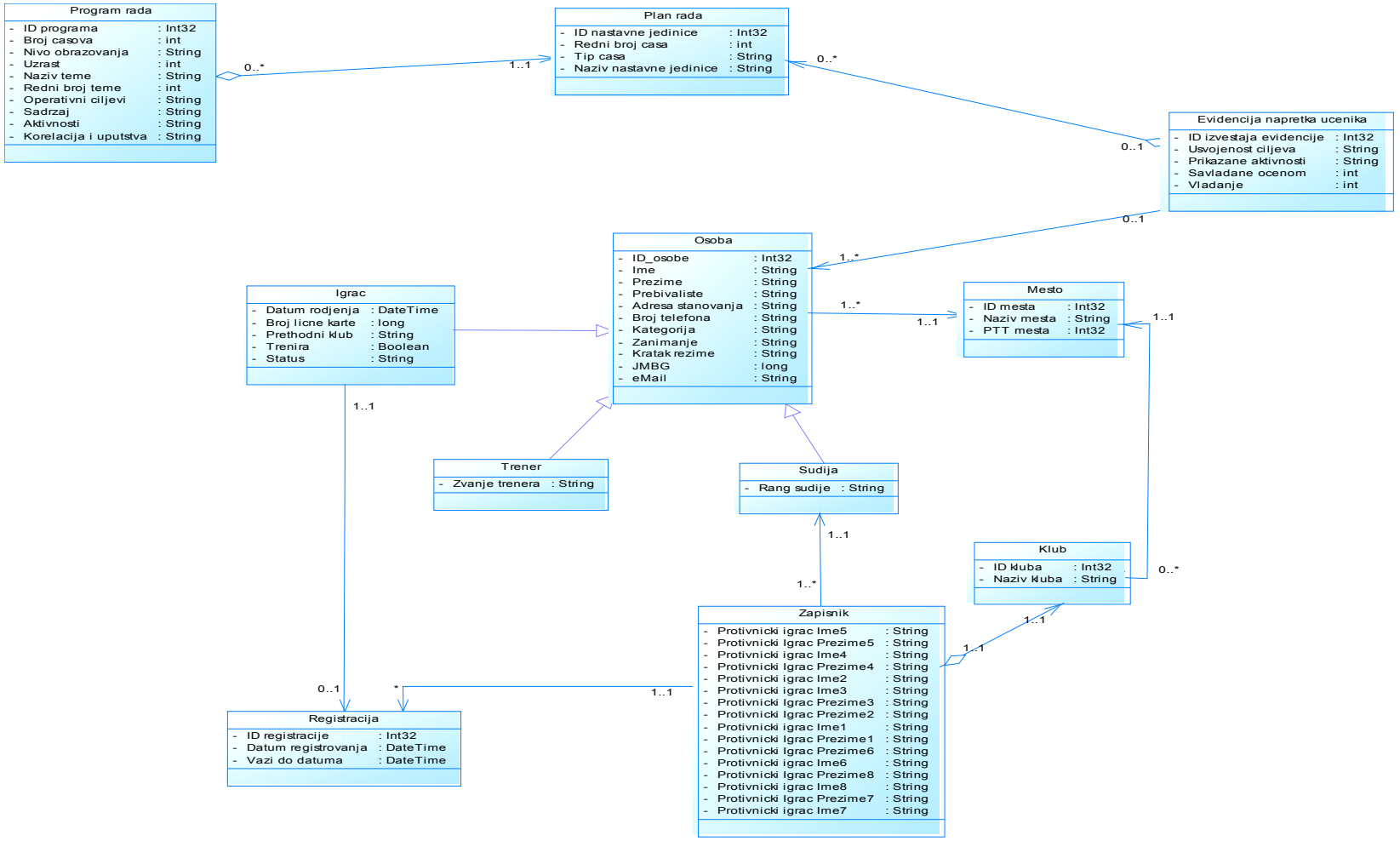
Proces analize može sadržati sledeće korake koji se mogu ponavljati dok se ne postigne zadovoljavajući nivo za projektante i „sponzore“.

- Koristi se model zahteva sistema da bi se pronašli kandidati klasa koji opisuju objekte koji mogu biti relevantni za sistem i zapisuju se na dijagramu klasa
- Pronalaženje veza (asocijacija, agregacija, kompozicija i nasleđivanje) za klase
- Pronalaženje atributa za klase
- Prolaženje kroz sistemski *use case* , proveravajući da li je podržan od strane objekata koje imamo, fino podešavanje klasa, atributa i veza odakle sledi da *use case* realizacija prouzrokuje operacije za kompletiranje atributa.
- Ažuriranje rečnika podataka i nefunkcionalnih zahteva ukoliko je neophodno.

4.1. Statička analiza

Statičko modelovanje uređuje logičke i fizičke delove sistema i način na koji će biti spojeni. Odnosno, opisuje kako ćemo konstruisati i inicijalizovati sistem.
Sastoji se od:

- identifikovanja klasa
- identifikovanja relacija među klasama
- crtanja klasnih i objektnih dijagrama
- kreiranja atributa



Slika24. Primer klasnog dijagrama modela analize

4.2. Dinamička analiza

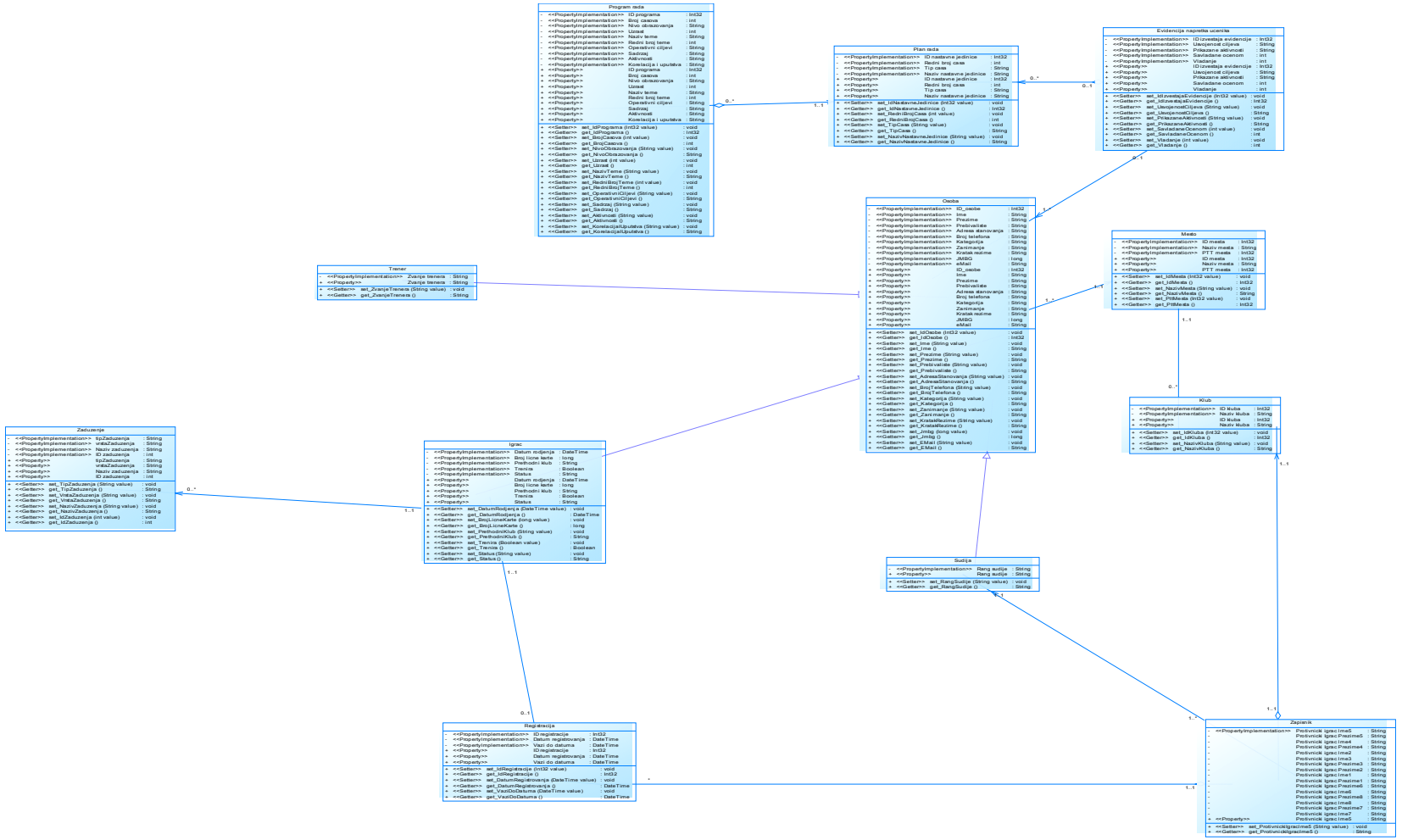
Dinamička analiza se izvodi iz sledećih razloga:

- Da bismo potvrdili da je naš dijagram klasa kompletan i ispravan, kako bismo mogli ispraviti na vreme greške ukoliko postoje: ovo podrazumeva dodavanje,brisanje i modifikovanje klasa, relacija, atributa i operacija.
- Da bismo bili sigurni da naše modelovanje, do ovog trenutka, može da se implementira u softver; u ovom koraku se naravno uključuju i experti (sponzori)
- Da bismo verifikovali funkcionalnost korisničkog interfejsa koji se pojavljuje u finalnom sistemu

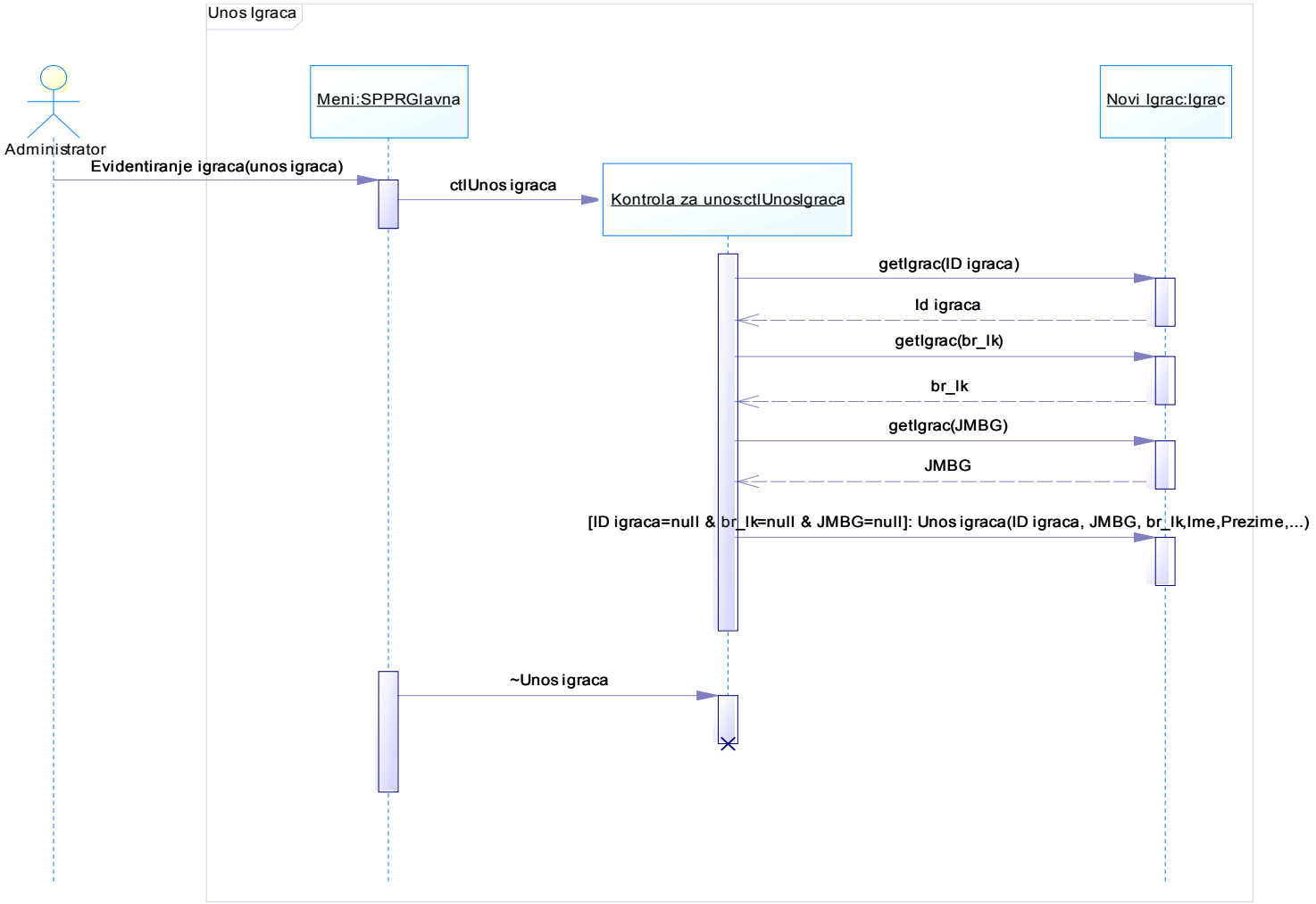
Dinamička analiza obuhvata:

- Use Case realizaciju – koriste se komunikacioni i sekvencni dijagrami
- Granice, entitete i kontrolere – koriste se komunikacioni dijagrami
    - učesnik – osoba ili sistem van granice sistema
    - granica – objekat na ivici sistema (između sistema i učesnika)
    - entitet – objekat u okviru sistema koji predstavlja poslovni koncept
    - kontrolor – uslužni objekat koji omogućava sledeće usluge
        - kontrolu sistemskih procesa
        - stvaranje novih entiteta
        - obnovu postojećih entiteta

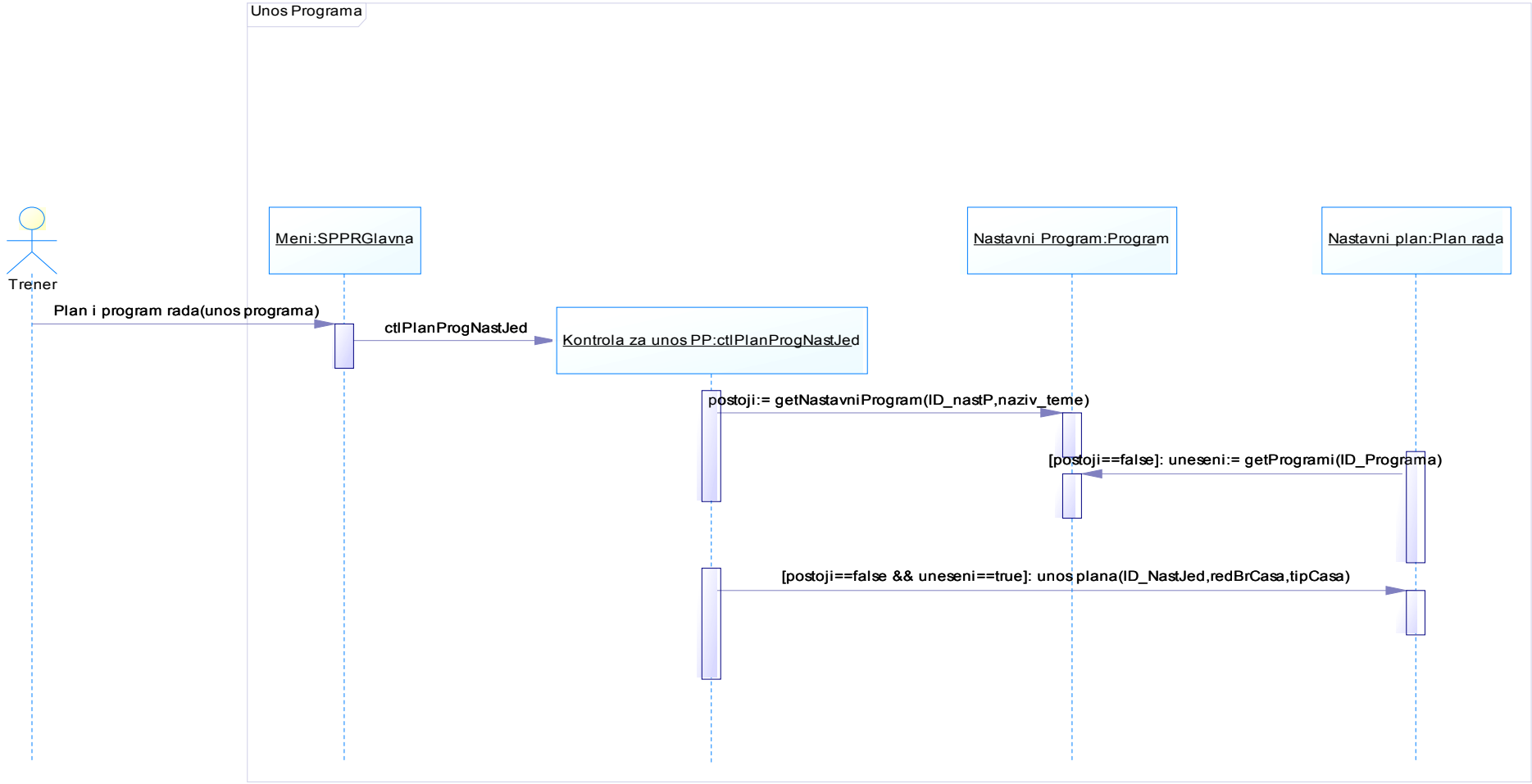
- Elemente komunikacionih dijagrama
- Dodavanje operacija klasama
- Odgovornosti
    - Svaki objekat treba da ima tačno jedan posao (ili ulogu) ako postoji objekat zadužen za više od jednog posla tada je očigledno da nam je potrebno više objekata. Za objekte sa jedinstvenim setom odgovornosti se kaže da imaju jaku koheziju, što je željeni cilj.
- Modelovanje stanja



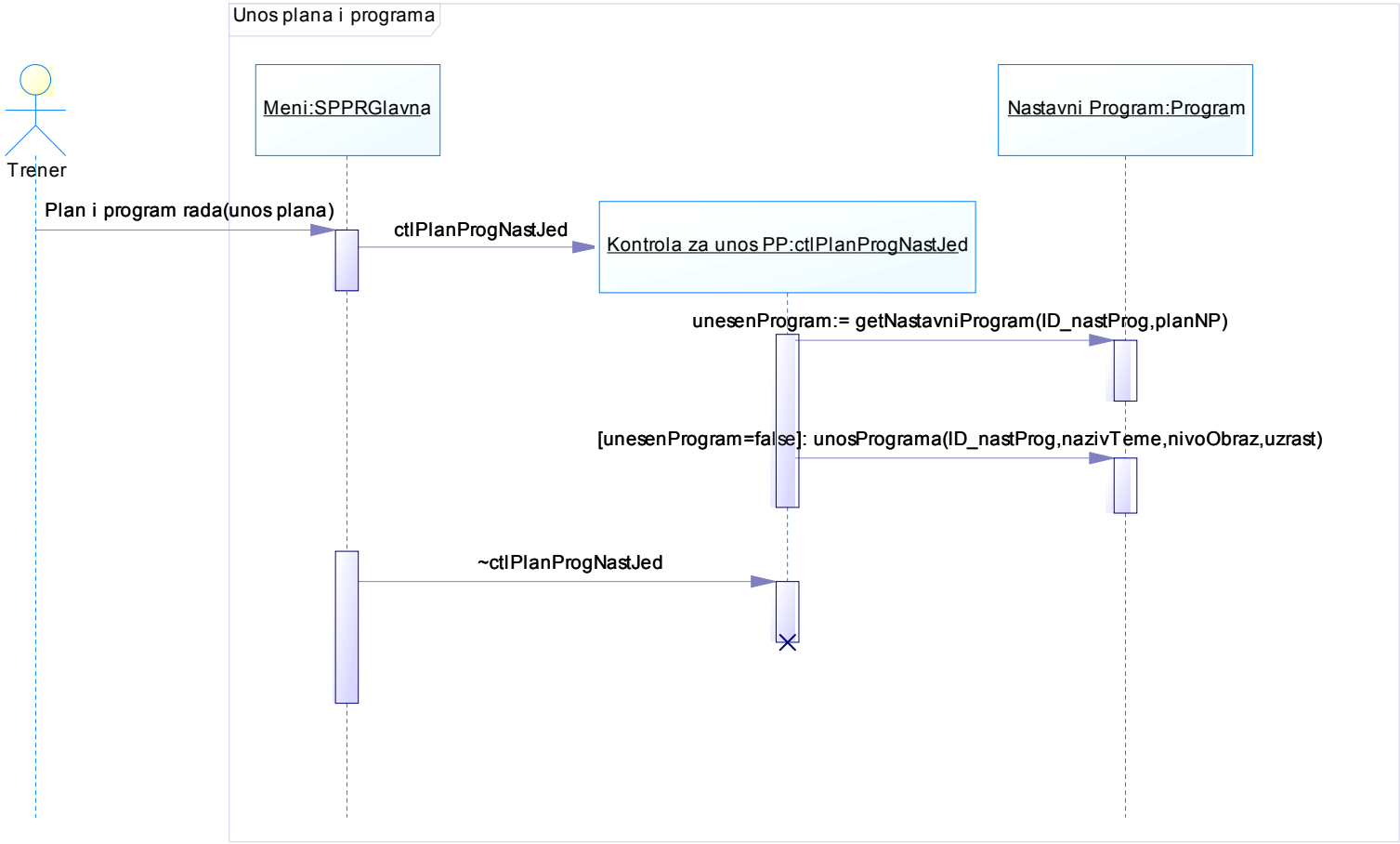
Slika25. Dijagram klasa sa operacijama



Slika26. Sequence diagram unosa igrača



Slika27. Sequence diagram unosa programa rada



Slika28. Sequence diagram unosa plana i programa

5. Projektovanje

Cilj ka kome se teži u etapi projektovanja jeste stvaranje dobrog – kvalitetnog proizvoda. Do kvalitetnog proizvoda se dolazi kombinacijom dobrog projekta i dobre realizacije.Ukoliko se bilo koji od ova dva uslova ne ispuni krajnji proizvod neće biti kvalitetan.

Na nastanak kvalitetnog softvera utiče nekoliko faktora koji se mogu grupisati u spoljašnje i unutrašnje faktore. Kvaliteti poput brzine i jednostavnosti korišćenja, a koje krajnji korisnici mogu direktno da iskuse, predstavljaju spoljašnje faktore kvaliteta; dok npr. modularnost i čitljivost softvera predstavljaju unutrašnje faktore, faktore koji su odlika izvornog teksta softvera. Može se reći da presudni uticaj imaju spoljašnji faktori jer korisnika ne zanima čitljivost izvornog koda već brzina kojom se kod izvršava, ali sa druge strane nije moguće (malo je verovatno) napisati kod koji se brzo izvršava a da unutrašnji faktori nisu korektno realizovani. Ključ spoljašnjih faktora leži u unutrašnjim.

U spoljašnje faktore ubrajaju se:

- **Korektnost** – *mogućnost softverskog proizvoda da tačno izvršava svoje zadatke koji su definisani njihovom specifikacijom*.
- **Robusnost** – *mogućnost softvera da na odgovarajući način reaguje na nenormalne situacije.*
- **Proširivost** – *lakoća prilagođavanja softvera promenama specifikacije.*
- **Višekratna upotreba** – *mogućnostsoftverskih elemenata da služe u konstruisanju više različitih aplikacija.*
- **Kompatibilnost** – *lakoća kombinovanja softverskih elemenata jednih sa drugim.*
- **Efikasnost** – *mogućnost softvera da što manje zahteva hardverske resurse.*
- **Prenosivost** – *lakoća prenošenja softvera u različita hardverska i softverska okruženja*
- **Jednostavnost upotrebe** - *jednostavnost upotrebe je lakoća sa kojom osobe razli itog obrazovanja i kvalifikacija mogu da nauče da koriste softverske proizvode i primene ih u rešavanju problema. Ona podrazumeva i jednostavnost instalacije, održavanja i nadgledanja*
- **Funkcionalnost** – *dijapazon mogućnosti koje pruža sistem*
- **Blagovremenost** – *mogućnost softvera da bude završen kada ga korisnici žele.*

5.1. Dizajniranje arhitekture sistema

Analiza i dizajn su veoma različite ideje, iako je granica između njih često zamagljena. Jasno razdvajanje analize i dizajna je dobra ideja koja nam omogućava da budemo sigurni da je problem dobro shvaćen pre nego što počnemo sa razmatranjem rešenja. Imajući to na umu, analiza predstavlja istraživanje problema dok dizajn predstavlja pronalaženje rešenja. Ne postoje striktna pravila za transformaciju modela analize u dizajn model.

U toku faze dizajniranja vršimo izbor tehnologije (npr. programski jezici, protokoli, DBMS,...) i moramo odlučiti u kojoj meri ćemo dozvoliti uticaj tehnologije na naš dizajn.
Što nam je dizajn generalniji time smo manje vezani za određenu tehnologiju i ne možemo iskoristiti sve prednosti konkretnih tehnologija.

Koraci u dizajnu sistema:

- odabir topologije sistema: na koji način će hardver i procesi biti distribuirani
- izbor tehnologije (programski jezici, baze podataka, protokoli,...)
- dizajniranje konkurentnosti : više korisnički, procesorski,računarski;
- dizajniranje sigurnosti: postoje mnogi aspekti sigurnosti koje moramo obezbediti i kontrolisati
- odabir delova podsistema: u većini slučajeva proizvodimo posebne aplikacije koje rešavaju određene probleme i moramo im obezbediti efektnu komunikaciju
- deljenje podsistema na slojeve ili druge podsisteme
- odluke o načinu komuniciranja mašina, podsistema i slojeva

5.1.1. Logička i fizička arhitektura

Fizička i logička arhitektura se veoma razlikuju. Logička arhitektura se zasniva na razdvajanju različitih tipova funkcionalnosti [8]. Naj poznatije logičko razdvajanje je na UI sloj, poslovni sloj, i sloj podataka koji se mogu nalaziti na istoj mašini ili na tri ne zavisne mašine – logička arhitektura ne definiše takve detalje.
Postoji izvesna veza između aplikacione logičke i fizičke arhitekture: logička arhitektura uvek ima veći ili jednak broj slojeva od fizičke arhitekture.
Ispravno dizajnirana logička n-slojna arhitektura obezbeđuje sledeće prednosti:

- logički organizovan kod
- lakšu manipulaciju
- bolje ponovno iskorištenje koda
- bolje iskustvo projektantskog tima
- visoku jasnoću prilikom kodiranja

Sa druge strane, ispravno dizajnirana fizička n-slojna arhitektura obezbeđuje sledeće prednosti:

- performanse
- skalabilnost
- nedostatak tolerancije

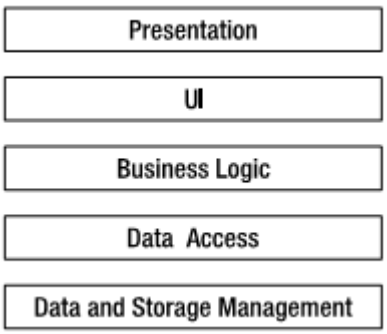
- bezbednost

Naravno, n-slojni dizajn (logički ili vizički) je kompleksniji od jednoslojnog dizajna. N-slojna arhitektura jedino pojednostavljuje proces kod velikih aplikacija ili kompleksnih okruženja.
N- slojnu arhitekturu bi trebalo koristiti u slučajevima kada:

- aplikacija je velika ili kompleksna
- aplikacija je jedna od mnogih sličnih ili srodnih aplikacija i prilikom kombinovanja nastaje velika ili kompleksna aplikacija
- okruženje je veliko ili kompleksno

, u suprotnob bi je trebalo zaobići.

Tradicionalno, najzastupljenija n-slojna arhitektura je bila sastavljena od 3 sloja: interfejsa, poslovne logike i sloja podataka.U novije vreme sve više se koristi 4 do 6 slojna logička arhitektura. U daljem tekstu je prikazana i objašnjena 5-slojna logička arhitektura.



Slika29. 5 – slojna logička arhitektura

Prezentacioni sloj (Presentation)
Sa Windows perspektive, sloj prezentacije i UI sloj su jedno isto: Predstavljaju (GUI) forme sa kojima su korisnici u interakciji.Međutim, Web perspektiva jasno razlikuje ova dva sloja. Web browser prezentuje sadržaj korisniku i prikuplja informacije od korisnika. U tom slučaju, sva interakciona logika se pokreće na web serveru, a ne na klijentskoj mašini.Treba još imati na umu da logički model mora pružiti podršku za pametne klijente i web bazirane klijente (pa čak i za mobilne telefone ili druge mobilne uređaje) tako da se u mnogim slučajevima prezentacioni logički sloj fizički razdvaja od UI logičkog sloja.

Korisnički interfejs (UI sloj – User Interface)
Sadrži logiku za odluku o tome šta će korisnik videti, navigacionu putanju, i način interpretacije korisnikovih akcija. U aplikacijama sastavljenim od Windows Formi tu logiku predstavlja kod iza formi. U stvari, kod se nalazi iza formi i u Web Form aplikacijama takođe, ali u ovom slučaju on može uključiti i kod koji se podiže na serverskim kontrolama; logički to je deo jednog sloja. U mnogim aplikacijama UI kod je veoma kompleksan. Kao prvo, mora da pruži odgovor korisničkim zahtevima.On takođe mora biti u interakciji sa logikom poslovnog sloja kako bi pružio validaciju korisničkih zahteva, da bi odradio svaki zatraženi posao ili bilo kakvu drugu poslovno baziranu akciju. Cilj je napisati UI kod koji će prihvatiti korisničke zahteve i dalje ih

prosleditи poslovnom sloju, gde će na kraju biti obrađen ili gde će na drugi način sa njim biti manipulisano. UI kod zatim mora odgovoriti korisniku prikazom rezultata koji predstavljaju korisnikovu interakciju sa poslovnim slojem. U .NET u UI kod je skoro uvek vođen događajima (event-driven), tako da se smatra više proceduralnim kodom i kodom baziranim na događajima nego objektno-orijentisanim kodom.

Poslovna logika (Business Logic)

Poslovna logika uključuje poslovna pravila, validaciju podataka, manipulaciju, obradu i sigurnost za aplikaciju.

Pravilo je, iako se sa tim pravilom ne slažu neki autori, da se sloj poslovne logike nalazi na odvojenom sloju od sloja na kome se nalazi UI kod. Moguće je određeni deo logike kopirati u UI kod kako bi se obezbedilo bogatije korisničko iskustvo, ali poslovni sloj mora implementirati svu poslovnu logiku, jer to je jedina svrha glavne kontrole i podržanosti. Poslovni sloj može biti vizički postavljen na klijentskoj radnoj stanici ili na serveru kako bi omogućio visok nivo interaktivnosti sa korisnikom. Da bi podržao neinteraktivne procese sloj poslovne logike se često smešta na aplikacioni server ili što je moguće bliže DBMS-u. Na sreću, moguće je uposliti jedan logički sloj na više fizičkih slojeva.

Rezultat toga je jednoslojni poslovni sloj koji je upošljen na klijentskoj radnoj stanici (ili Web serveru) i na aplikacionom serveru, što omogućava aplikaciji da obezbedi visok nivo interaktivnosti za vreme korisnikovog rada nad aplikacijom i efikasan *back-end processing* (pozadinska obrada) za ne aktivne procese.

Pristupni sloj (Data Access)

Kod iza pristupnog sloja podataka je u interakciji sa *Data Management* slojem i omogućava povraćaj, dodavanje, ažuriranje i uklanjanje informacija. Sloj pristupa u stvari ne upravlja i ne skladišti podatke; pre bi se moglo reći da obezbeđuje interfejs između poslovnr logike i baze podataka. U nekim slučajevima sloj pristupa će se naći na mašini koja je fizički odvojena od one na kojoj se nalazi sloj korisničkog interfejsa i/ili poslovne logike. U drugim slučajevima kod sloja pristupa će se naći na istoj mašini kao i poslovni sloj (ili čak UI) u cilju poboljšanja performansi. Logičko definisanje pristupnog sloja kao odvojenog sloja primorava separaciju između poslovne logike i bilo kakve interakcije sa bazom podataka (ili bilo kojim izvorom podataka). Ovakavo odvajanje obezbeđuje fleksibilnost prilikom kasnijeg odabira pokretanja koda pristupnog sloja na istoj mašini sa poslovnom logikom ili na drugoj mašini. Takođe omogućava promenu izvora podataka bez uticaja na aplikaciju.

Ovakvo odvajanje je korisno, između ostalog, i zbog toga što Microsoft ima naviku da menja tehnologije pristupnih slojeva na svake tri ili više godina, što nosi za sobom neophodno ponovno pisanje koda pristupnog sloja. (podsetimo se DAO, RDO, ADO 1.0, ADO 2.0, i sada ADO.NET). Izolacijom koda pristupnog sloja u poseban sloj uticaj ovih promena se ograničava na manje delove aplikacija.

Ponekad sloj pristupa može biti realizovan kroz seriju metoda koju koristi ADO.NET direktno kako bi povratio ili skladištio podatke. U drugim prilikama, sloj pristupa podacima je monogo kompleksniji, obezbeđujući apstraktniji ili čak meta podacima vođen način pristupa podacima. U takvim slučajevima, sloj pristupa može sadržati veoma kompleksan kod koji obezbeđuje ovakve apstraktne šeme pristupa podacima.

Smeštaj i upravljanje podcima (Data Storage and Management)
Ovaj sloj rukuje fizičkom kreacijom, povraćajem, ažuriranjem i brisanje podataka. *To je razlika u odnosu na sloj pristupa podacima koji zahteva* kreaciju, povraćaj, ažuriranje i brisanje podataka. Sloj za smeštanje i upravljanje podacima implementira ove operacije u kontekstu baze podataka ili seta fajlova,...

5.1.2. Implementacija logičke arhitekture

Petoslojnu logičku arhitekturu [8] je moguće konfigurisati u jedan, dva, tri, četiri ili pet fizičkih slojeva u cilju poboljšanja performansi, skalabilnosti, bezbednosti ili nulte tolerancije.

Optimalne performanse- pametni klijent
Ovaj način implementacije je korišten prilikom projektovanja i implementacije SPPR Šahovskih Klubova.

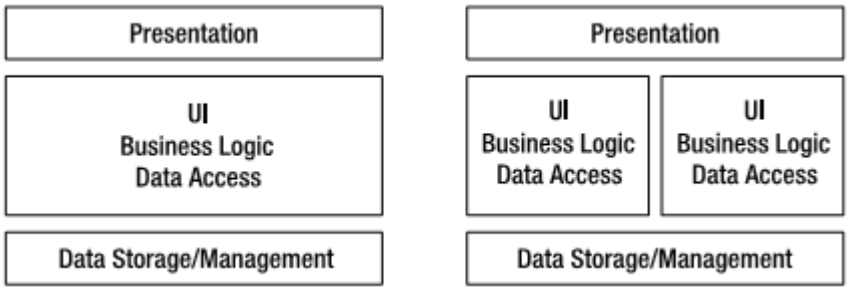
Implementacija obično podrazumeva Windows Forme za prezentacioni i UI sloj , sa kodom poslovne logike i sloja pristupa podacima koji se izvršava u istom procesu i obraća se *Access (JET)* ili *Microsoft SQL Server Express* bazi podataka.Činjenica da je sistem upošljen na samo jednom fizičkom sloju ne kompromituje logičku arhitekturu i odvajanje kao što je i prikazano na slici ispod.

Slika30. Petoslojna logička arhitektura instalirana na jednoj mašini

Slika31. Petoslojna logička arhitektura sa odvojenim DBMS

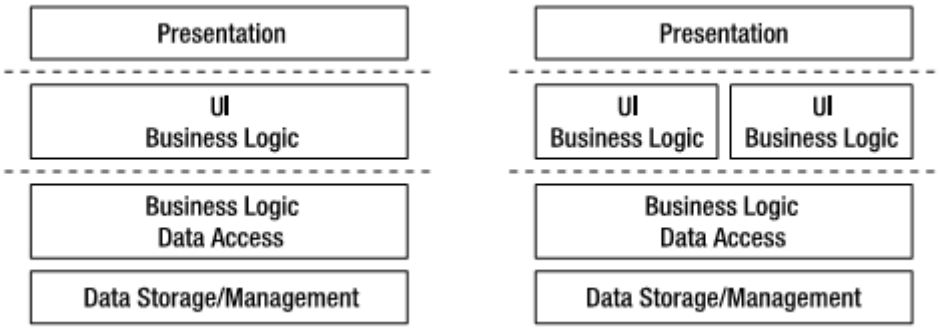
Postoje još neki načini implementacije logičke arhitekture:

- High-Scalability Smart Client
- Optimal Performance Web Client

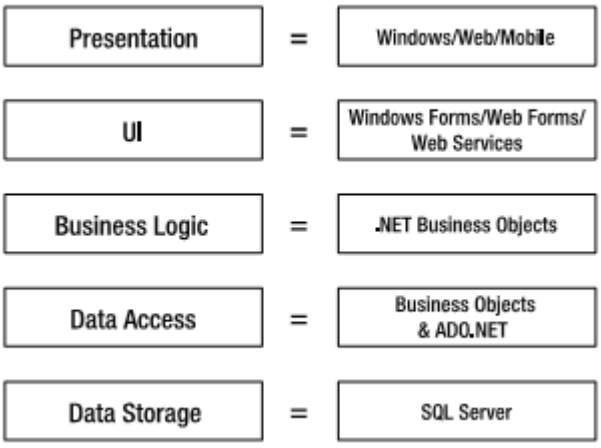


Slika32. Petoslojni model korišten u web aplikaciji i web farmama

- High-Security Web Client



Slika33. Petoslojni sigurnosni model primenjen na web aplikacijama i farmama



Slika34. Mapiranje logičkih slojeva prema tehnologijama

5.2. Konkurentnost

U mnogim sistemima, pogotovo mrežnim, se paralelno odvijaju mnogi procesi; takvi sistemi se nazivaju konkurentni sistemi.Da bismo obezbedili da sistem bude konkurentan moramo odgovoriti na sledeća pitanja:

- Kako obezbediti ažuriranje podataka u potpunosti pre nego što im neko uopšte pristupi
- Kako obezbediti da se ažuriranje ne odvija za vreme čitanja podataka koji treba da se ažuriraju

Ukoliko postoje konkurentne situacije koje mogu prouzrokovati poteškoće u sistemu, ne treba započinjati implementaciju dok ne bude postojala garancija da takva situacija neće ugroziti rad sistema.

5.3. Sigurnost

Sigurnost je veoma širok termin koji se može podeliti na nekoliko aspekata:

- privatnost – skrivanje informacija i dostupnost samo privilegovanim
- autentičnost – poznavanje izvora informacije kako bismo odlučili da li možemo imati poverenja isti
- nepobitnost – omogućava da možemo sa sigurnošću tvrditi da je informacija pristigla sa određenog izvora
- integritet – moramo biti sigurni da pristigla informacija nije oštećena
- sigurnost – mora postojati mogućnost kontrole pristupa resursima

5.3.1. Kriptografija

Osnovni cilj kriptografije [4] je da omogući da dvoje ljudi, pošiljalac i primalac, mogu da komuniciraju preko nekog nesigurnog kanala, tako da protivnik ne može da protumači šta je poslato. Informacija koju pošiljalac želi da pošalje predstavlja izvorni tekst. On se procesom šifrovanja transformiše u šifrovan tekst i takav šalje primaocu. Protivnik prisluškivanjem dolazi do te poruke ali ne može da protumači sadržaj, dok primalac procesom dešifrovanja rekonstruiše izvorni tekst.
Postoje dve vrste kriptosistema simetričan i asimetričan. Kod simetričnog se koristi isti tajni ključ i za šifrovanje i za dešifrovanje, dok se kod asimetričnog koristi jedan, javni ključ za šifrovanje, a drugi, privatni ključ za dešifrovanje informacija. Mana simetričnih sistema je što je za prenos ključa potreban siguran kanal a prednost se ogleda u, i do šesnaest puta, jačim algoritmima za šifrovanje.

Dužina ključa direktno utiče na njegovu sigurnost, jer što je ključ duži to je veći broj kombinacija. Postoji još jedan uslov savršene sigurnosti, a to je korišćenje ključa samo jednom.

DES spada u simetrične sisteme i zasniva se na kodiranju nad blokovima dužine 64 bita. Ključ je takođe dužine 64 bita od kojih se za kodiranje koriste 56 bita a 8 služe za proveru parnosti. Algoritam prolazi kroz sledeće faze: inicijalna permutacija, zatim se blok deli na dva dela, levi i desni po 32 bita. Potom slede 16 rundi kombinovanja sa ključem. Spajaju se levi i desni blok u jedan, ponovo dužine 64 bita. Na kraju, sledi finalna permutacija koja odgovara inverznoj inicijalnoj permutaciji.

Detaljnije o simetričnim i asimetričnim kripto sistemima i njihovim implementacijama može se pronaći u [4]
Sledeći kod u programskom jeziku C# predstavlja konkretnu implementaciju DES algoritma:

```
FileStream outStream = File.Create("ISSK_admin.psw");
TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider(); //provajder
za kriptovanje
CryptoStream cs =
new CryptoStream(outStream,tdes.CreateEncryptor(),CryptoStreamMode.Write);
StreamWriter sw = new StreamWriter(cs);

sw.Write(original);
sw.Flush();
sw.Close();


FileStream outStreamSudija = File.Create("ISSK_sudija.psw");

CryptoStream csS = new
CryptoStream(outStreamSudija,tdes.CreateEncryptor(),CryptoStreamMode.Write);
StreamWriter swS = new StreamWriter(csS);

swS.Write(original2);
swS.Flush();
swS.Close();


FileStream fsKeyOut = File.Create("Encript_admin.key");  // enkriptovanje administratora

BinaryWriter bw = new BinaryWriter(fsKeyOut);
bw.Write(tdes.Key);
bw.Write(tdes.IV);
bw.Flush();
bw.Close();

FileStream fsKeyOutS = File.Create("Encript_sudija.key");// enkripcija sudije

BinaryWriter bwS = new BinaryWriter(fsKeyOutS);
bwS.Write(tdes.Key);
bwS.Write(tdes.IV);
bwS.Flush();
bwS.Close();



TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider();
                                                        //provajder za dekriptovanje

FileStream fsKeyIn = File.OpenRead("Encript_admin.key");  //cita kluc za dekriptovanje
sifre
BinaryReader br = new BinaryReader(fsKeyIn);
tdes.Key = br.ReadBytes(24);
tdes.IV = br.ReadBytes(8);

FileStream fsIn = File.OpenRead("ISSK_admin.psw");         //cita sifru

CryptoStream cs = new CryptoStream(fsIn, tdes.CreateDecryptor(),CryptoStreamMode.Read);
//dekriptuje sifru
```

```
StreamReader sr = new StreamReader(cs);

String vracenaSifra = sr.ReadToEnd();        //string "sifra" dobija vrednost passworda

fsKeyIn.Close();                      //zatvara objekat file strteam i oslobadja resurse
sr.Close();

FileStream fsKeyInS = File.OpenRead("Encript_sudija.key");  //cita kluc za dekriptovanje
sifre

BinaryReader brS = new BinaryReader(fsKeyInS);
tdes.Key = brS.ReadBytes(24);
tdes.IV = brS.ReadBytes(8);

FileStream fsInS = File.OpenRead("ISSK_sudija.psw");        //cita sifru

CryptoStream csS = new
CryptoStream(fsInS,tdes.CreateDecryptor(),CryptoStreamMode.Read);//dekriptuje sifru

StreamReader srS = new StreamReader(csS);
String vracenaSifraSudije = srS.ReadToEnd(); //string "sifra" dobija vrednost passworda

fsKeyInS.Close();                     //zatvara objekat file strteam i oslobadja resurse
srS.Close();
```

5.4. Dizajn podsistema

Vodeći se principima dizajniranja sistema, dizajn podsistema možemo definisati preko:

- dizajna klasa i polja poslovnog sloja koristeći klasni model iz postupka analize kao vodič
- načina skladištenja perzistentnih podataka i dizajna skladišta
- izgleda korisničkog interfejsa na osnovu skica proizvedenih u toku postupka analize
- prelaza kroz slučajeve korišćenja sa referencama na dizajn korisničkog interfejsa oslanjajući se na poslovne servise koji moraju biti podržani u srednjem sloju
- razvoja poslovni servisa u serverske objekte (distribuiran dizajn sistema)
- definisanja mera koje su potrebne za osiguranje konkurentnosti sistema (na osnovu poslovnih pravila koja kontrolišu pristup sistemu: korisničko ime, šifra,...) i paralelnog procesiranja

5.5. Mapiranje klasa iz modela analize u dizajn model

Za svaku dizajniranu klasu biramo nazive i tipove polja. Često polja proizilaze iz atributa ili asocijacija otkrivenih u toku analize. Pored atributa i asocijacija razmatra se još i nasleđivanje (odlučujemo da li ćemo ga zadržati).
Mapiraju se:

- operacije,
- tipovi promenljivih,
- tipovi polja,
- poželjno se obezbeđuju polja pristupa

```
...
private int count;
public int getCount() {
return count;
}
public void setCount(int c) {
count = c;
}
...
```

- klase,
- atributi,
- kompozicije,
- identifikuje se univerzalni identifikator,...

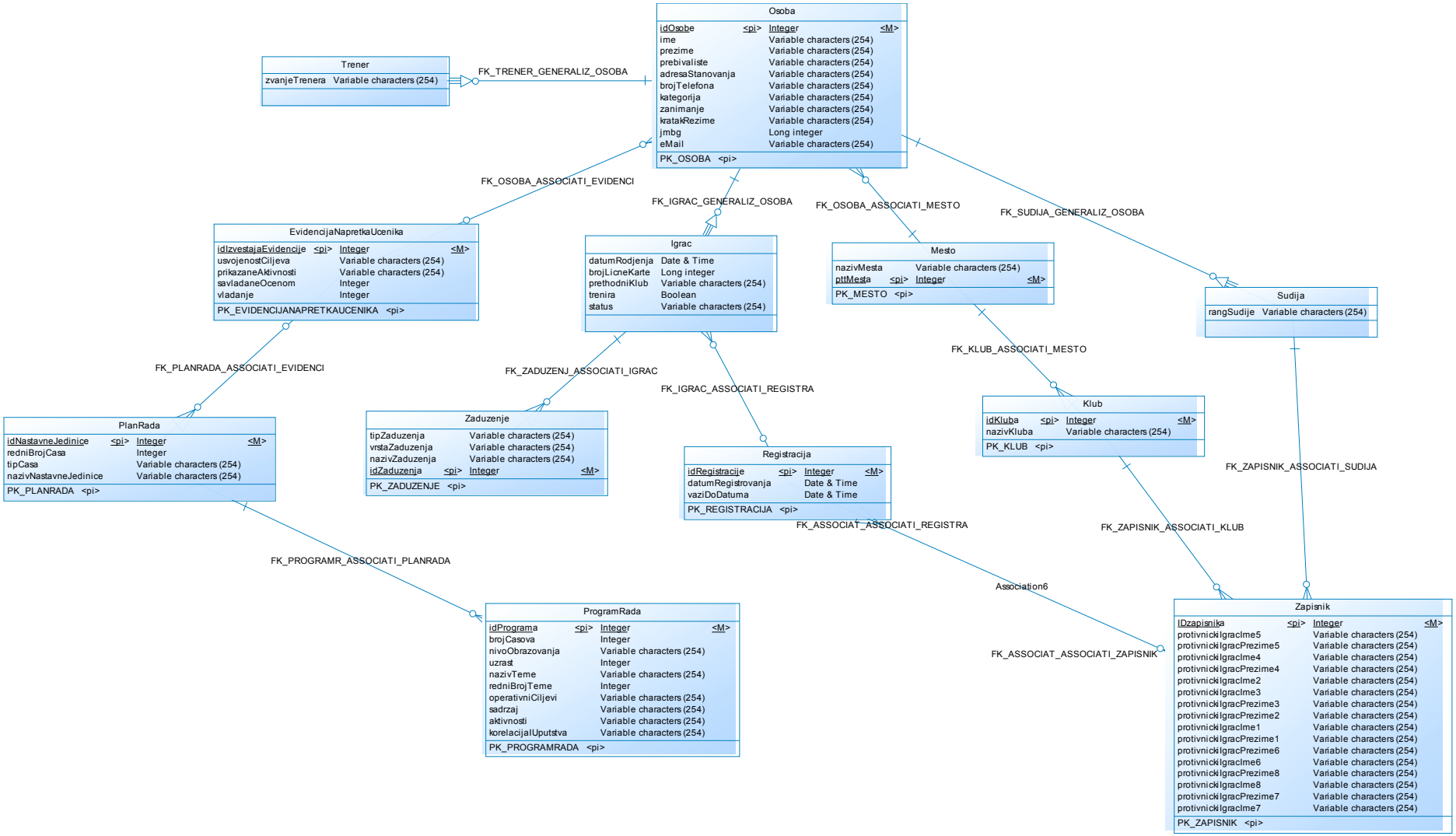
6. Modeliranje i dizajniranje baza podataka

Modeliranje je generalno skoncentrisano na pravljenje logičkih i fizičkih modela baza podataka. Logički model baze podataka je sastavljen od entiteta, atributa i uključuje relacije među različitim entitetima koje generalno mogu biti prinudne ili ne. Logički model je zasnovan na normalizovanom modelu koji je generalno u trećoj normalnoj formi. Sadrži mnoštvo elemenata koji sačinjavaju bazu, ali nije specifikovan prema konkretnom softveru ili konkretnoj implementaciji baze podataka.

Denormalizacija počinje sa fizičkim modelom baze. Logička baza se optimizuje za upite, specifične sisteme za upravljanje bazama podataka, i aplikacije koje mogu komunicirati sa bazom.

Dok je modeliranje proces koji je većinom fokusiran na prikazu baze, dizajn obuhvata čitav proces od kreiranja zahteva, poslovnih procesa, logičke analize, i fizičke konstrukcije baze do razvijanja same baze [1].

Na slici 35 i slici 36 dati su logički i fizički modeli baze podataka izrađeni u Power Designeru 12 gde je fizički model optimizovan prema MS SQL 2000 serveru.



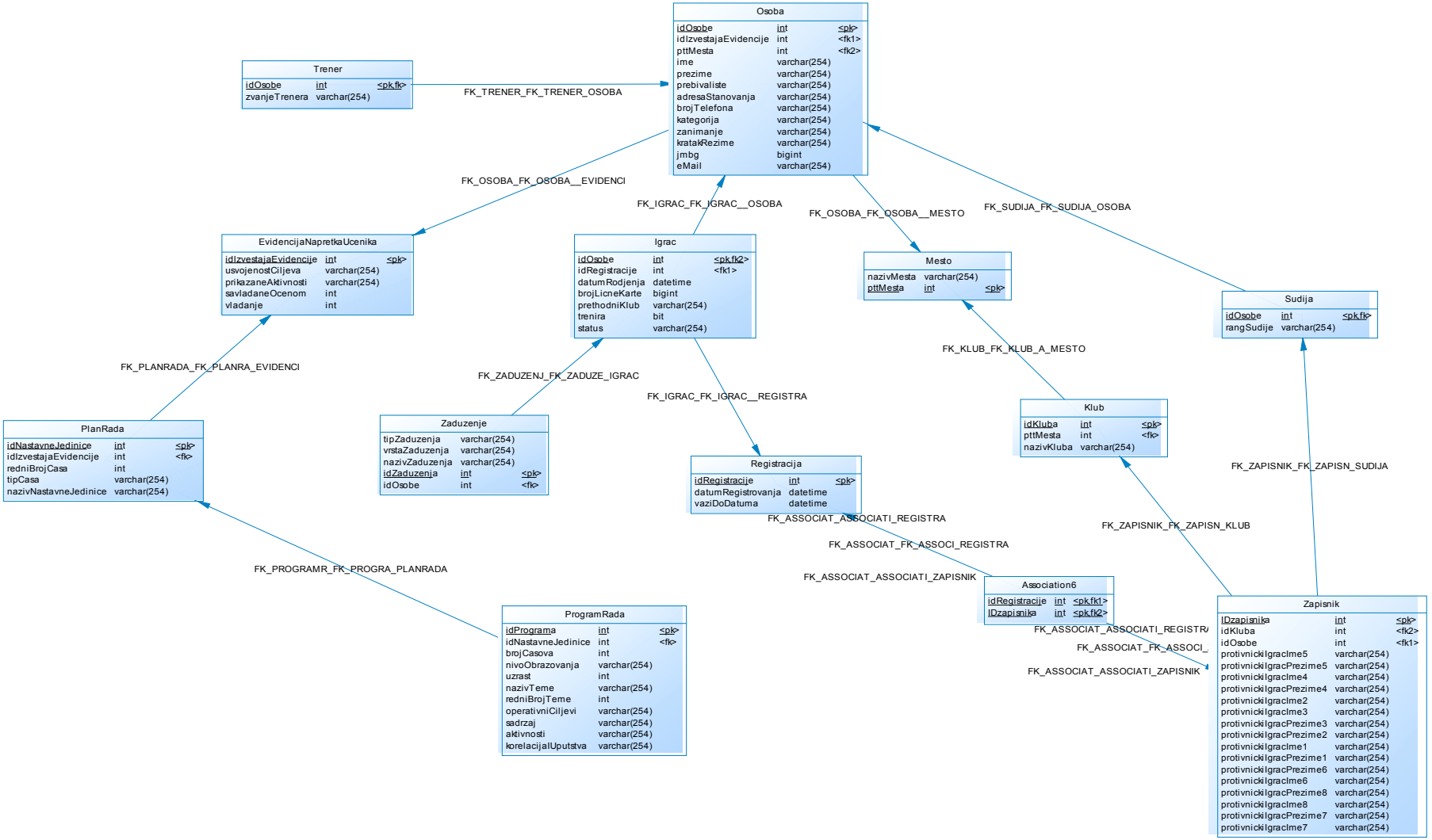
Slika35. Logička šema baze podataka

Ovim dijagramom predstavljene su i definisane klase objekta (tipovi entiteta) sa pripadajućim atributima i njihovim vrednostima iz odgovarajućih domena, primarnim ključevima pomoću kojih ostvaruju međusobne veze, kao i tip veza njihovih ograničenja.

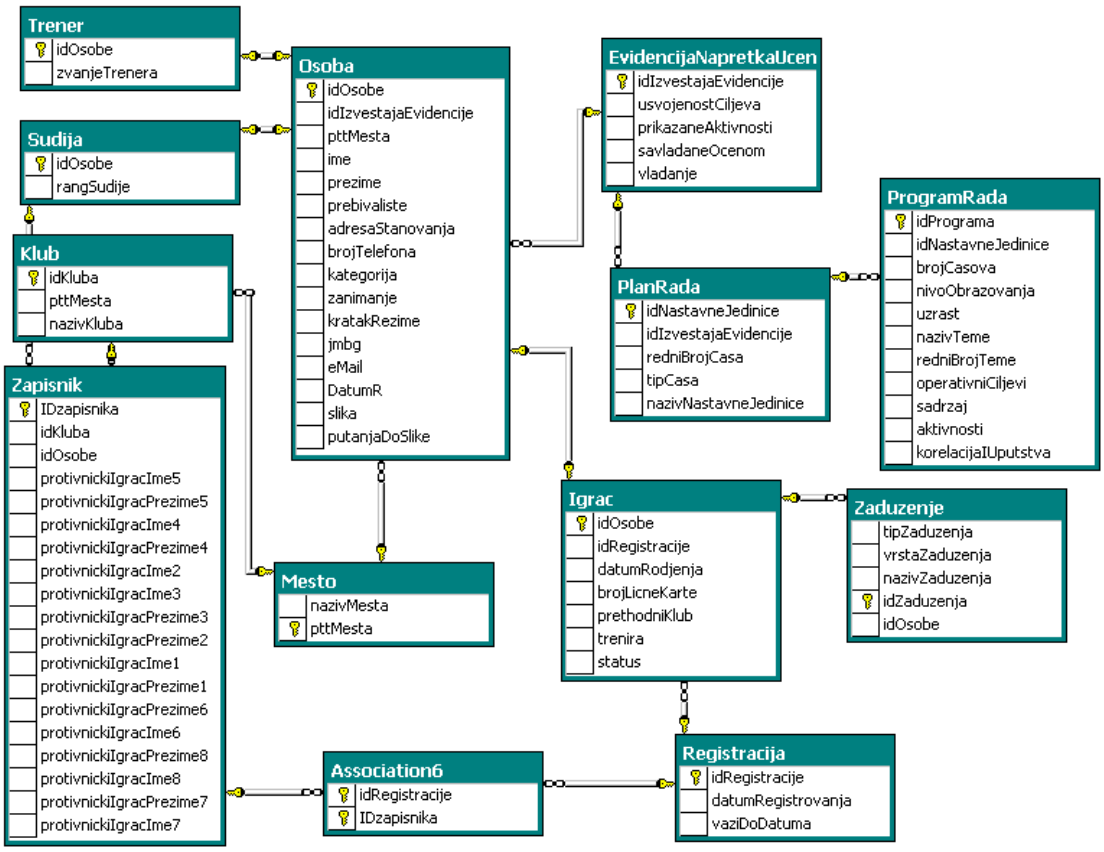
U fizičkom modelu podataka (PDM – Phisycal Data Model) se vrši dodavanje i definisanje alternativnih (semantičkih) ključeva u budućim tabelama, koji treba da obezbede eliminisanje redundanse i nedozvoljenih unosa podataka, a potom i ograničenja nad podacima u vidu referencijalnog integriteta.

Fizički model podataka mora biti potpuno tehnički (sintaksno) ispravan, jer se u protivnom neće izvršiti generisanje baze.
Baza podataka predstavlja jednu statičku sliku realnog sistema koja u svakom trenutku mora da ispunjava i zadovoljava određeni skup zadatih uslova i pravila integriteta, kako bi bila u konzistentnom, odnosno ispravnom stanju.
Ona u suštini predstavlja skup tabela podataka u koje su smešteni podaci povezani određenom problematikom.

Fizički model podataka prikazan je na slici 36.



Slika36. Fizička šema baze podata



Slika37. Šema baze podataka MS SQL 2000

U radu je korišten SQL Server kao RDBMS. SQL Server ili MS SQL vodi poreklo od Sybase SQL servera. Microsoft i Sybase su se udružili 1989 godine da bi, od svih stvari, razvili verziju SQL Servera za OS/2. SQL Server je preseljen u Windows NT 1993. u verziji 4.2. partnerstvo je raskinuto kada je izašla verzija 6.0. Od verzije 6.5. pa nadalje, SQL Server predstavlja samo Microsoft-ov proizvod. Veoma uspešna verzija 7.0 je u suštini kompletno prerađen proizvod, a ujedno je i prva verzija koja je bila dostupna za Windows 9x (praktično više nije bilo Sybase-ovog koda u SQL Serveru).Verzija SQL Servera, kojom je realizovan RDBMS aplikacije u ovom radu, je bila sledeća verzija nakon verzije 7.0

7. Opis aplikacije

7.1. Instalacija

7.1.1. Hardverski zahtevi

Za instalaciju i korišćenje programa na PC kompatibilnim računarima neophodno je ispuniti sledeće minimalne hardverske zahteve:

Minimalni hardverski zahtevi
Pentium III na 800 MHz
128 MB RAM memorije
Color SVGA monitor 15 inča
Slobodnog prostora na hard disk 60 Mb
CD ROM
Flopi disk drajv 3.5 inča
Miš

Preporučeni hardverski zahtevi
Pentium IV na 2.0 MHz
256 MB RAM memorije
Color SVGA monitor 17 inča
Slobodnog prostora na hard disk 60 Mb
CD ROM
Flopi disk drajv 3.5 inča
Miš

7.1.2. Softverski zahtevi

Program je moguće koristiti na PC računarima sa instaliranim WINDOWS XP operativnim sistemom. Potrebna minimalna rezolucija ekrana je 800x600 piksela. Takođe je potrebno instalirati Microsoft .NET Framework SDK v2.0. a za potrebe korišćenja baze podataka potrebno je imati instaliran MS SQL Server 2000.

7.1.3. Postupak instalacije

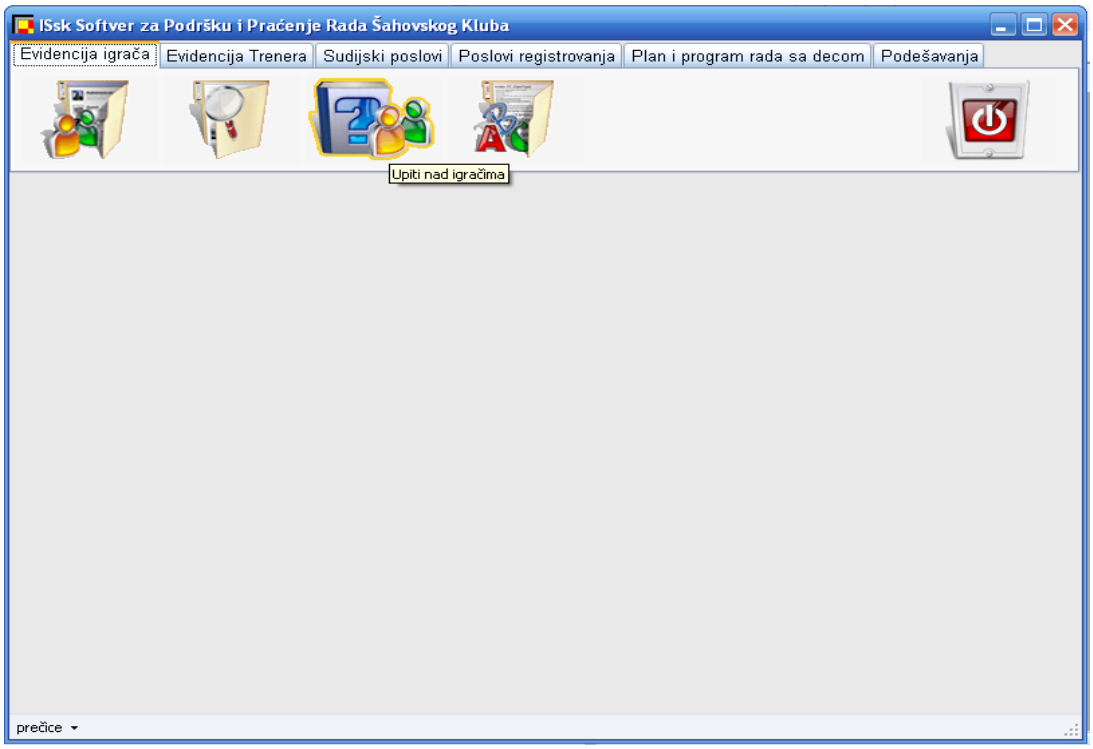
Za instalaciju programa potrebno je sa CD-a ili disketa pokrenuti fajl Setup.exe koji vrši automatsku instalaciju programa u folder koji bira sam korisnik tokom instalacije. Setup kreira ikonicu u Start meniu, na radnoj površini i omogućava deinstalaciju iz kontrolne table / Dodaj Ukloni programe.

7.2. Korišćenje (uputstvo za upotrebu)

7.2.1. Struktura aplikacije (programa)

Program je organizovan u formi standardnog Windows menija što omogućava korisniku brzo i lako kretanje kroz program, a naročito je omogućeno jednostavno unošenje podataka u bazu, izmena i brisanje postojećih, kao i izrada i štampanje neophodnih izveštaja. Program poseduje i Pomoć koja omogućava opis svih funkcija programa: menija (podmenija), rad sa formama programa, preglede i načine generisanja izveštaja.

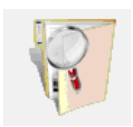
Nakon startovanja aplikacije pojavljuje se sledeći uvodni ekran sa osnovnim opcijama:



Slika38. Glavni meni

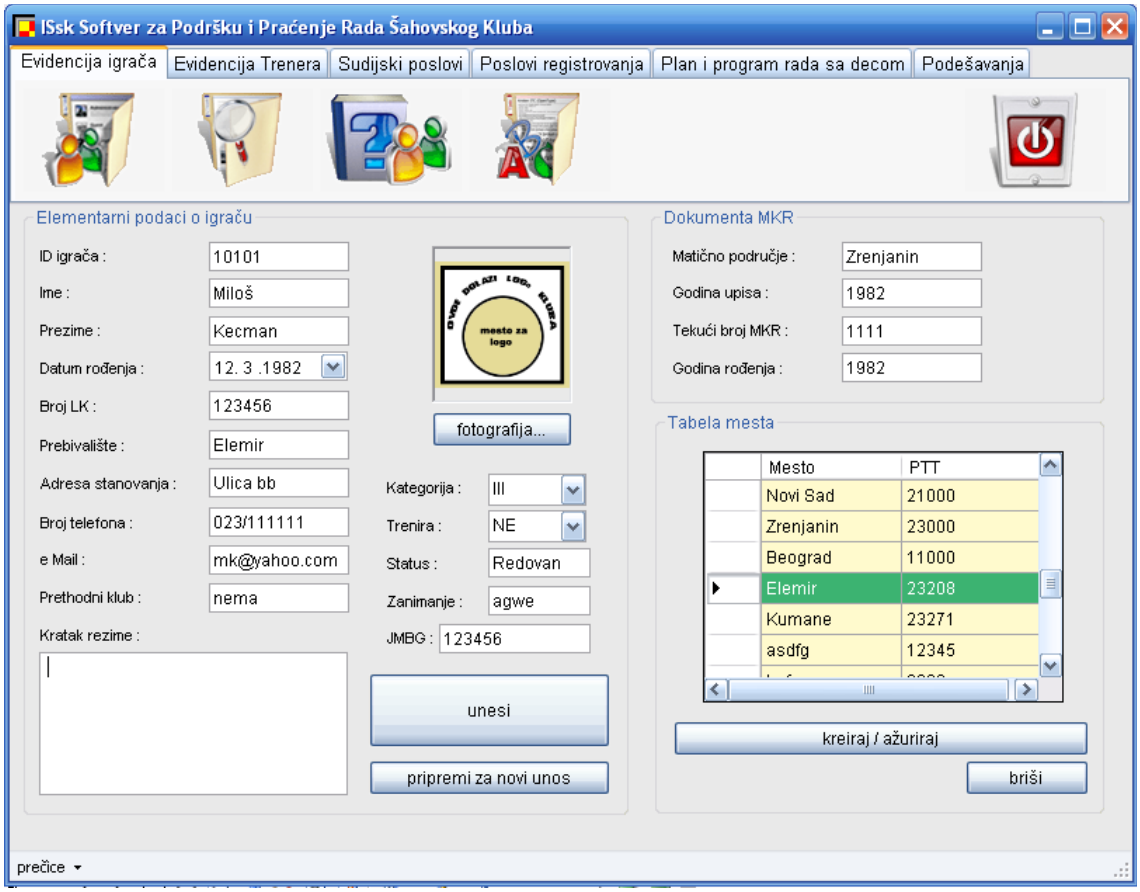
Prvo što se može uočiti je tabControl sa šest kartica. Svaka od kartica nosi ime koje opisuje opcije za manipulaciju nad podacima.

Evidencija igrača sadrži opcije za:

 Unos igrača ,  Pregled igrača,  Upiti nad igračima,

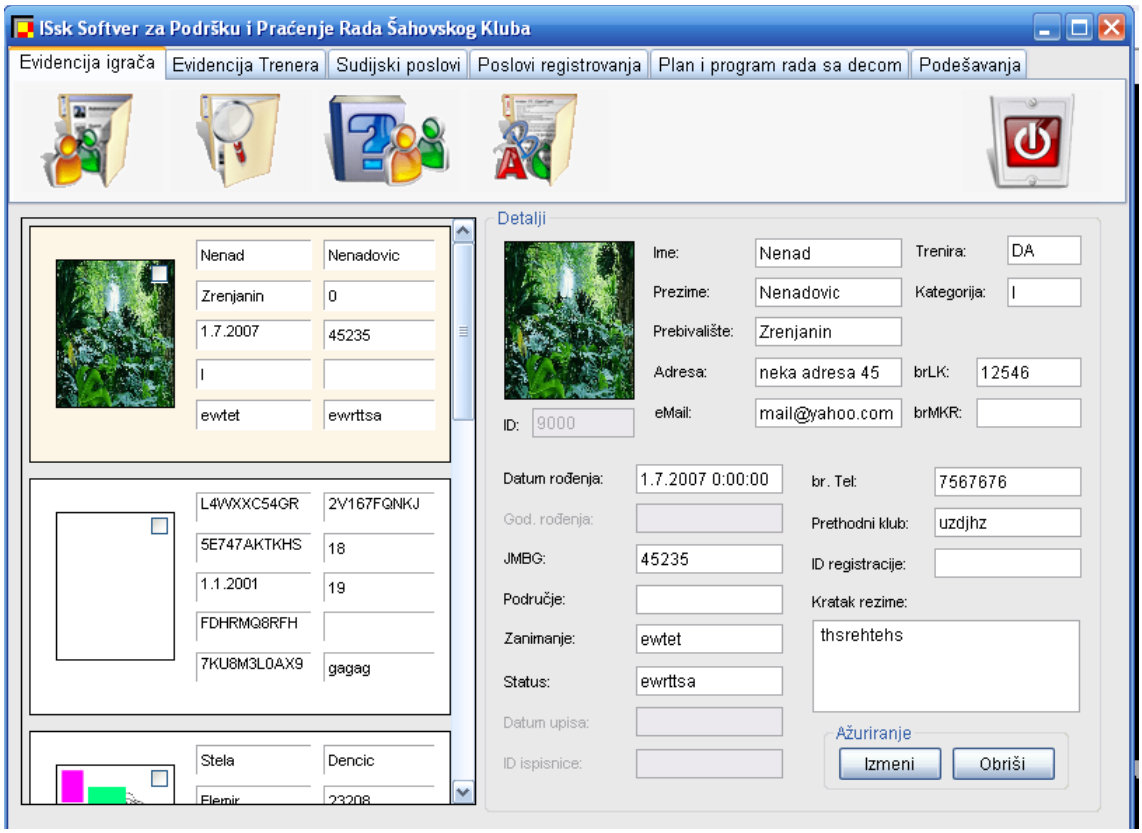
 Zaduženja igrača i  Izlaz

Klikom na *Unos igrača* ikonicu kreira se nova kontrola sa poljima za unos teksta tj. podataka o igraču. Pored polja za unos teksta na kontroli su još i polje za unos slike-aktiviranje *open file dialoga* pritiskom na dugme fotografija unosi se slika u bazu (njena putanja), dugmad za unos podataka iz svih polja u bazu ili čišćenje polja,a sa desne strane kontrole se nalazi DataGrid kontrola koja prikazuje sva mesta koja su u bazi i omogućuje odabir istih (klik na ćeliju mesta ili ptt-a automatski unosi naziv mesta u polje prebivalište).Preko ove kontrole, a pozivom nove forme pritiskom na dugme *kreiraj/ažuriraj* moguće je uneti novo mesto, ili izbrisati staro pritiskom na dugme *briši.*



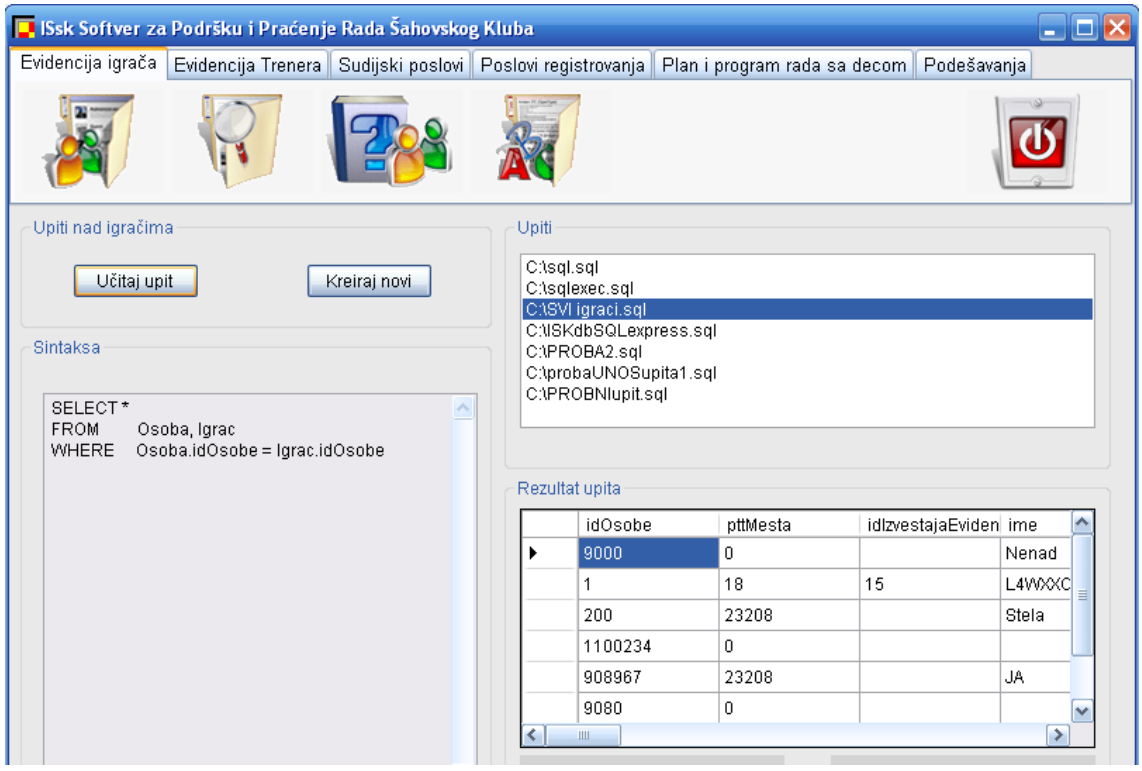
Slika39. Kontrola za unos igrača

Odabirom *Pregled igrača* pojavljuje se kontrola kao na slici 40. Sastoji se iz dve kontrole. Sa leve strane se nalazi kontrola u kojoj se nalaze kratki opisi svih igrača u bazi – „lične karte“ Odabirom neke od njih u desnom delu kontrole se prikazuju detaljni podaci igrača. Kontrole na levom delu ekrana-kratak pregled, je moguće čekirati-označiti kako bi se nakon desnog klika miša mogli poslati svi podaci, koji su vezani za čekiranu kontrolu, eMail-om ili odštampati.



Slika40. Kontrola za pregled

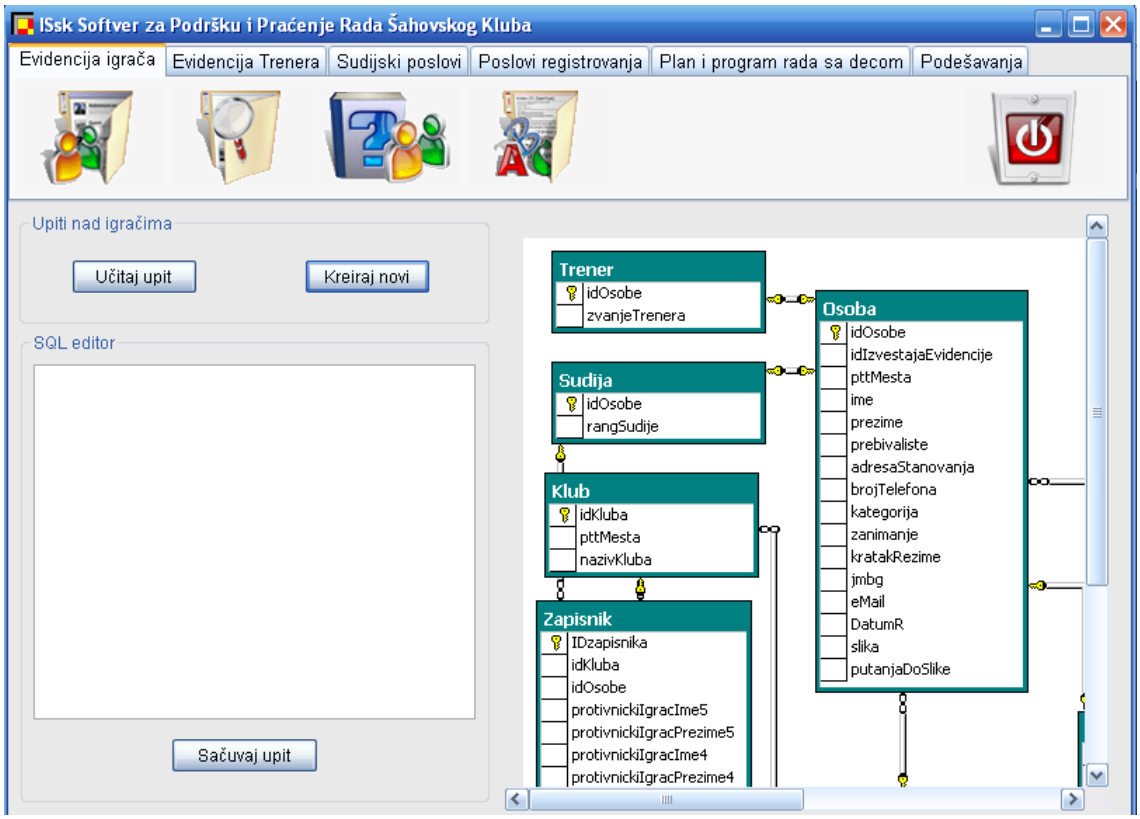
Na sledećoj slici 41 može se videti na koji način se upiti realizuju.



Slika41. Kontrola upita(učitavanje)

Klikom na dugme *Učitaj upit* pojavljuje se *open file dialog* i selektuju se željeni upiti koji se prikazuju u listBoxu (desno gore). Selektovanjem bilo kog od ponuđenih upita iz listBoxa u tekstualnom polju u delu sintaksa prikazuje se SQL upit koji vraća rezultate koji se prikazuju u DataGrid kontroli.

Kontrola koja omogućuje kreiranje novog upita nakon pritiska na dugme *Kreiraj novi* prikazana je na slici 42.

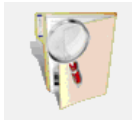


Slika42. Kontrola upita(snimanje)

Na desnoj strani kontrole se nalazi slika šeme baze podataka kako bi se mogla stvoriti slika o uređenju i hijerarhiji baze nad kojom se izvršava upit. SQL kod se unosi u SQLeditor i čuva pod nekim imenom sa ekstenzijom .sql

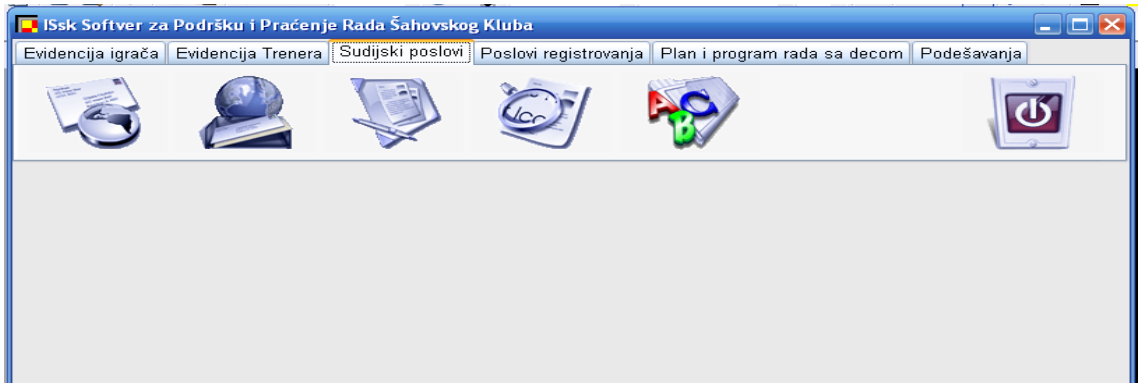
Evidencija trenera sadrži opcije:

Slika43. Unos trenera

 Unos trenera,  Pregled trenera,  Upiti nad trenerima i  Izlaz.

Klik na bilo koju opciju učitava kontrole za unos, pregled i upite nad trenerima – koje su prethodno objašnjene.

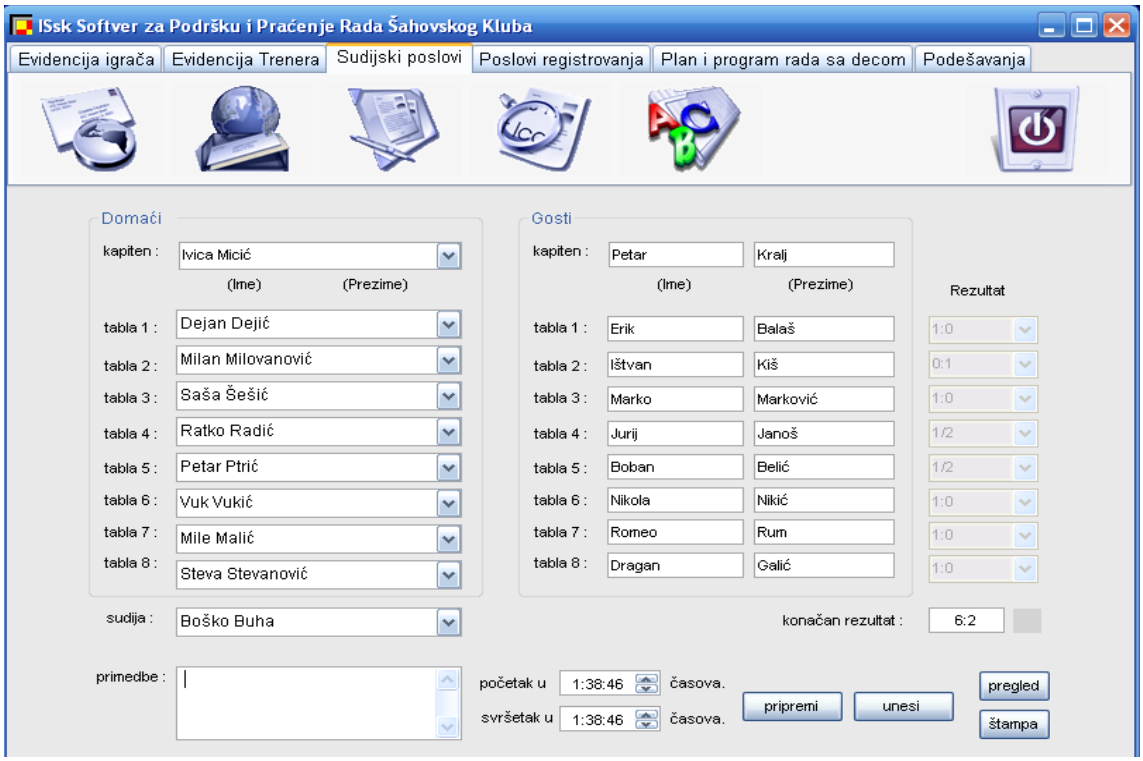
Sudijski poslovi sadrži:



Slika44. Sudijski poslovi

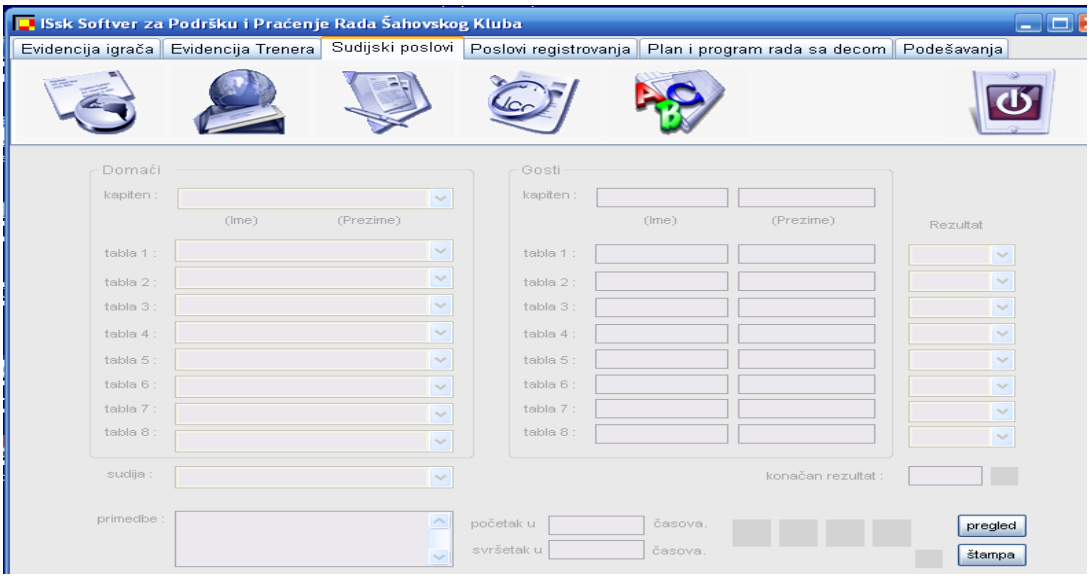
 Preuzimanje tabele sudija,  Slanje zahteva za suđenjem,  Unos zapisnika,  Pregled zapisnika,  Raspisi

Klikom na *Preuzimanje tabele...* ili *Slanje zahteva...* pojavljuje se kontrola za slanje email-a.
Unos zapisnika prikazuje sledeće:



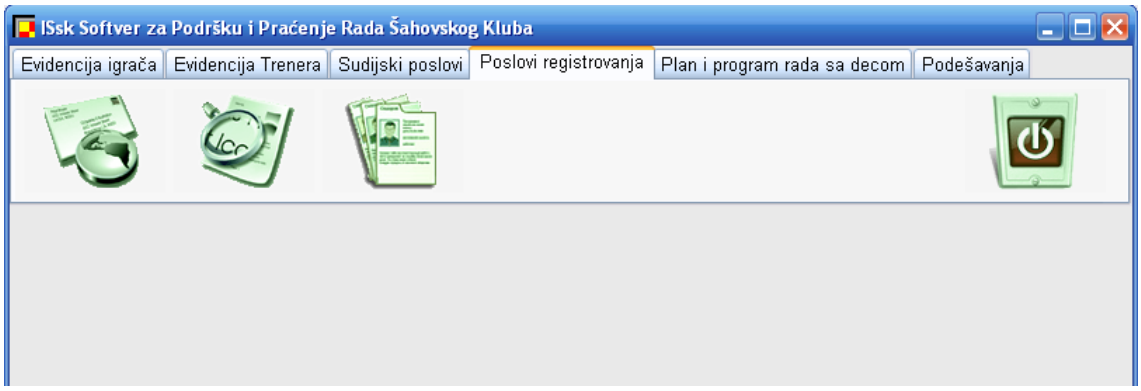
Slika45. Unos zapisnika

Sadrži polja za unos igrača i rezultata. Slično *Unosu zapisnika* pregled takođe ima sličnu strukturu polja samo što ta polja imaju svojstvo samo za čitanje.



Slika46. Pregled zapisnika

Poslovi registrovanja sadrže:

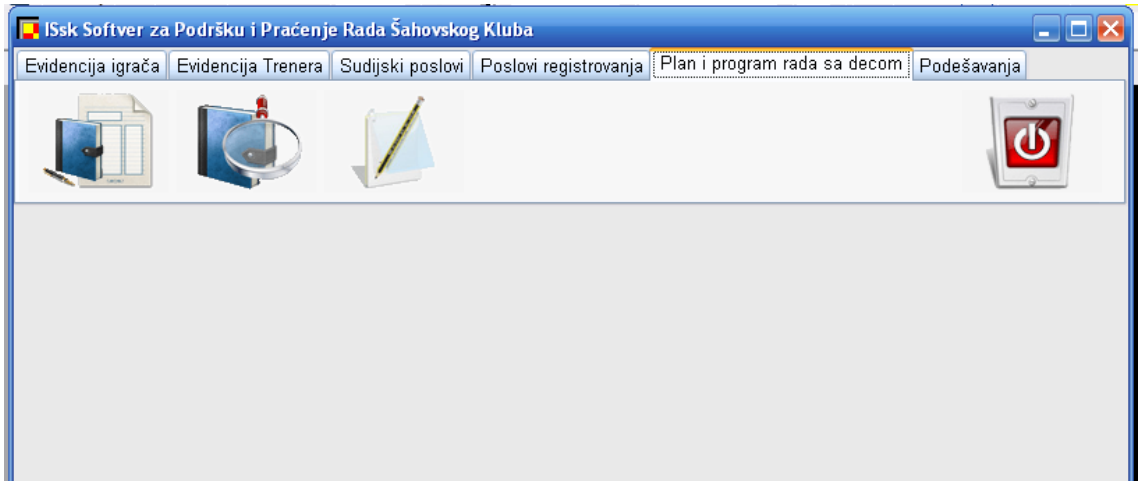


Slika47. Poslovi registrovanja

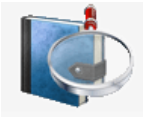
 Preuzimanje obaveštenja o registracijama,  Pregled obaveštenja

 Pregled registracija

Plan i program rada sa decom sadrži:



Slika48. Plan i program

 Unos plana i programa,  Pregled plana i programa,

 Evidentiranje napretka učenika

Podešavanja sadrži:

 Prilagođavanje aplikacije,  About,  Pomoć

Slika49. Podešavanje aplikacije

8. Realizacija

Objašnjenje realizacije će biti predstavljeno putem programskog koda koji je pisan u programskom jeziku C#, razvojnom okruženju Microsoft Visual Studio.Net 2005.
Zbog velikog broja programskih redova, prikazana je samo neka od realizacija pojedinih procedura.

Prikaz koda forme koja se prikazuje Intro aplikacije:

```
public partial class TransparencyForm : Form
    {
        public int brojacKill = 0;
        SPPRGlavna parent;
        public TransparencyForm(SPPRGlavna parent)
        {
            InitializeComponent();
            this.parent = parent;


            timer1.Interval = 200;//timer1 upravlja providnosti forme (daje "takt")
            timer1.Enabled = true;//po pokretanju je startovan

            timer2.Interval = 300;//1000;//timer2 se stara o unistenju aplikacije
            timer2.Enabled = false;//inicijalizovan je na 1sec i trenutno neaktivan
        }

        private void TransparencyForm_Paint(object sender, PaintEventArgs e)
        {
            // Podesava region forme (u ovom slucaju na kruzni oblik)

            System.Drawing.Drawing2D.GraphicsPath myGraphicsPath =
                new System.Drawing.Drawing2D.GraphicsPath();

            myGraphicsPath.AddEllipse(0, 0, this.Width, this.Height);
            this.Region = new Region(myGraphicsPath);
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            if (this.Opacity == 1)
            {
                timer1.Enabled = false;
                timer2.Enabled = true;
            }
            this.Opacity = this.Opacity + 0.035;
        }

        private void timer2_Tick(object sender, EventArgs e)
        {
            if (brojacKill == 1)
            {
                timer2.Enabled = false;
                parent.Opacity = 1;
                this.Dispose();
            }
            brojacKill += 1;
        }

    }
```

Prikaz koda klase koja sadrži podešavanje za SQL konekciju kao i neke osnovne procedure za manipulisanje nad podacima. Ovu klasu koriste sve kontrole koje komuniciraju sa bazom.

```
public class AppKonekcije
    {
        public SqlConnection konekcija;

        public AppKonekcije()
        {
            // kreiranje konekcije
    konekcija = new SqlConnection("server=.;database=SPPRdbms;Trusted_Connection=yes");
        }


        public void konektujSe()
        {
            try
            {
               //konektovanje
                konekcija.Open();
            }
            catch (SqlException sqlEx)
            {
 MessageBox.Show(sqlEx.Message, "Greška", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }


        public void zatvoriKonekciju()
        {
            try
            {
               // diskonekcija
                konekcija.Close();
            }
            catch (SqlException sqlExClose)
            {
                MessageBox.Show(sqlExClose.Message, "Greška", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }


        public DataSet ucitajPodatke(DataSet dsDestinacije, string komanda,
SqlDataAdapter daPunjenjaDsDestinacije, string nazivTabele_u_DS_dest)
        {
            dsDestinacije = null;
            try
            {

                daPunjenjaDsDestinacije = new SqlDataAdapter(komanda,konekcija);

                dsDestinacije = new DataSet();
                daPunjenjaDsDestinacije.Fill(dsDestinacije, nazivTabele_u_DS_dest);
                return dsDestinacije;
            }
            catch (Exception)
            {
                return dsDestinacije;
            }
        }
```

```
        public int izbrojUnose(string imeTabeleKojojSeBrojeUnosi,SqlConnection
konekcija)
        {
            //string naziv = ""; ;
            try
            {
                int ukupnoUnosa = 0;
                konektujSe();
                //pravi se selekt komanda
                string komandaBroja = "Select Count(*) FROM  " +
imeTabeleKojojSeBrojeUnosi + "";

                SqlCommand komanda = new SqlCommand(komandaBroja, konekcija);
                SqlDataReader reader = komanda.ExecuteReader();
                if (reader.HasRows)
                {   //Ukoliko reader ima neke redove
                    while (reader.Read())
                    {
                        ukupnoUnosa=Convert.ToInt32(reader.GetValue(0).ToString());

                    }
                }
                reader.Close();
                zatvoriKonekciju();
                return ukupnoUnosa;

            }
            catch (Exception excep)
            {
                zatvoriKonekciju();
                MessageBox.Show(excep.Message);
                return 0;
            }
        }

        public void Brisi_prost_Unos(string kolonaKljuca,string kljucTabele,string
nazivTabele)
        {
            //Pravi se DELETE komanda
            string CommStr = "Delete From "+nazivTabele+" Where "+kolonaKljuca+" LIKE '"
+kljucTabele+ "'";
            try
            {
                konektujSe();
                SqlCommand Comm = new SqlCommand(CommStr, konekcija);
                Comm.ExecuteNonQuery();
                MessageBox.Show("Uspešno ste izbrisali željeni unos");

            }
            catch (Exception g)
            {
                MessageBox.Show(g.Message);
                MessageBox.Show("Što znači da pokušavate da obrišete iz tabele "
+nazivTabele+ " celokupan unos pomoću "+kolonaKljuca+", ali vam je zabranjeno brisanje
jer se "+kolonaKljuca+" ,tj. podaci, nalaze u relaciji sa nekom drugom tabelom!");
            }
            finally
            {
                zatvoriKonekciju();
            }
        }


    }
```

Dat je deo koda iz ctlPregled u kom se vidi način učitavanja niza kontrola

```
public void generisanjeIgraca()
        {
            ctlView.ctlView.brK = 0;
               // broje se igrači kako bi se generisao tačan broj kontrola „lične karte“
                brojOsoba = appKon.izbrojUnose("Igrac", appKon.konekcija);

                newPanel = new ctlView.ctlView[brojOsoba + 1];

                DataSet dsIgraci = new DataSet("Igraci");
                string komandaPunjenjaIgraca = "SELECT DISTINCT  Osoba.idOsobe,
Osoba.pttMesta, Osoba.ime,Osoba.prezime,Osoba.prebivaliste, Osoba.kategorija,
Osoba.zanimanje, Osoba.jmbg, Osoba.eMail, Igrac.datumRodjenja, Osoba.adresaStanovanja,
Igrac.trenira, Igrac.brojLicneKarte, Igrac.status, Igrac.prethodniKlub,
Osoba.kratakRezime, Igrac.idRegistracije, Osoba.brojTelefona,Osoba.putanjaDoSlike FROM
Osoba INNER JOIN Igrac ON Osoba.idOsobe = Igrac.idOsobe";
                SqlDataAdapter daPunjenjaIgraca = new
SqlDataAdapter(komandaPunjenjaIgraca, appKon.konekcija);
                daPunjenjaIgraca.Fill(dsIgraci, "Igraci");

                try
                {
                    int x = 5, y = 5, i;
                    for (i = 0; i < newPanel.Length - 1; i++)
                    {
                        newPanel[i] = new ctlView.ctlView();
                        newPanel[i].Location = new System.Drawing.Point(x, y);
                        y += 155;
                        newPanel[i].Name = i.ToString();

                       if (dsIgraci.Tables["Igraci"].Rows[i]
["putanjaDoSlike"].ToString()!="")
                         newPanel[i].pictureBox1.Image =
Image.FromFile(dsIgraci.Tables["Igraci"].Rows[i]["putanjaDoSlike"].ToString());

                        //Vezivanje za dataSet

                        newPanel[i].lblIme.DataBindings.Clear();
                        newPanel[i].lblIme.DataBindings.Add(new Binding("Text",
dsIgraci, "Igraci.Ime"));

                        newPanel[i].lblPrezime.DataBindings.Clear();
                        newPanel[i].lblPrezime.DataBindings.Add(new Binding("Text",
dsIgraci, "Igraci.prezime"));

                        newPanel[i].lblPrebivaliste.DataBindings.Clear();
                        newPanel[i].lblPrebivaliste.DataBindings.Add(new Binding("Text",
dsIgraci, "Igraci.prebivaliste"));

                        newPanel[i].lblPtt.DataBindings.Clear();
                        newPanel[i].lblPtt.DataBindings.Add(new Binding("Text",
dsIgraci, "Igraci.pttmesta"));
                        newPanel[i].lblJMBG.DataBindings.Clear();
                        newPanel[i].lblJMBG.DataBindings.Add(new Binding("Text",
dsIgraci, "Igraci.jmbg"));
                        newPanel[i].lblStatus.DataBindings.Clear();
                        newPanel[i].lblStatus.DataBindings.Add(new Binding("Text",
dsIgraci, "Igraci.status"));
                        newPanel[i].lblZanimanje.DataBindings.Clear();
                        newPanel[i].lblZanimanje.DataBindings.Add(new Binding("Text",
dsIgraci, "Igraci.zanimanje"));

                        newPanel[i].BindingContext[dsIgraci, "Igraci"].Position = i;

                        this.panPregled.Controls.Add(newPanel[i]);
                        newPanel[i].BringToFront();
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
        }
```

Primer listBox IndexChanged osobine

```
private void listUpiti_SelectedIndexChanged(object sender, EventArgs e)
        {
            // učitavanje kroz stream teksta u textBox
            string sqlUpit = "";
            StreamReader sr = new StreamReader(listUpiti.Text);
            sqlUpit = sr.ReadToEnd();

            textBox3.Text = sqlUpit;

            //Izvršavanje upita

            DataSet dsSqlUpiti = new DataSet();

            try
            {
                appKon.konektujSe();
                SqlDataAdapter daSqlUpiti = new SqlDataAdapter(sqlUpit,
appKon.konekcija);

                daSqlUpiti.Fill(dsSqlUpiti, "Upiti");

                dataGridView1.DataSource = dsSqlUpiti;
                dataGridView1.DataMember = "Upiti";
                dataGridView1.Refresh();
            }
            catch (Exception excs)
            {
                MessageBox.Show(excs.Message);
            }
            finally
            {
                appKon.zatvoriKonekciju();
            }

        }
```

Prikaz koda koji vodi računa o „matematici“ u sudijskim zapisnicima.

```
public void rezultatK(string selektovanKombo)
        {
            switch(selektovanKombo)
            {
                case "1:0":
                    {
                        zbirDomacih += 1;

                    }
                    break;
                case "0:1":
                    {

                        zbirGosti += 1;
                    }
                    break;
                case "1/2":
                    {
                        zbirDomacih += 0.5;
                        zbirGosti += 0.5;
                    }
                    break;
                default:
                    break;
            }
textBox21.Text = zbirDomacih.ToString() + "" + ":" + "" +
zbirGosti.ToString();
        }
```

Kod klase koji vodi racuna o kreiranju tj. ažuriranju XML konfiguraconog fajla za aplikaciju:

```
class writeXMLmeniConfig
    {
        public static Int32
writeBgColorPicBox=MeniInit.bgColorPicBox.ToArgb();
        public static Int32
writeBgColorTab=MeniInit.bgColorTab.ToArgb();
        public static Int32
writeBgColorApp=MeniInit.bgColorApp.ToArgb();
        public static Boolean prikaziLogo;
        public static string putanjaLogoa=MeniInit.putanjaLogo;

        public void meniWriteXML()
        {

            XmlWriterSettings settings = new XmlWriterSettings();

            settings.Indent = true;

            settings.IndentChars = ("    ");

            settings.ConformanceLevel = ConformanceLevel.Document;

            settings.CloseOutput = false;

            settings.OmitXmlDeclaration = false;

            // (2) Create XmlWriter object

            XmlWriter writer =
XmlWriter.Create(Application.StartupPath + @"/SPPR.xml",settings);

            writer.WriteStartDocument();

            writer.WriteStartElement("S.P.P.R.SK");

               try
                {


writer.WriteElementString("bojaTabCtl",writeBgColorTab.ToString());

                    writer.WriteElementString("bojaPicBoxMouseMove",
writeBgColorPicBox.ToString());

                    writer.WriteElementString("bojaApp",
writeBgColorApp.ToString());


writer.WriteElementString("prikaziIntro",prikaziLogo.ToString());

                    writer.WriteElementString("putanjaLogo",
putanjaLogoa);

                 }

                catch (Exception ex)
                {
```

```
                    MessageBox.Show(ex.Message);

                }


            writer.WriteEndElement();

            writer.Flush();  // Flush any remaining content to XML
stream

            writer.Close();

        }
        }
```

9. Zaključak

Danas, čak i u većim šahovskim klubovima, retko postoje informacioni sistemi za praćenje rada. Pa čak i ako postoje veoma su zastareli. Potrebno je uvođenje novih sistema, što će u mnogome doprineti povećanju produktivnosti rada.

Upravo zbog napretka informatičkih tehnologija, koja sve više prodire u sfere života i rada ljudi, neizbežno je okrenuti se uvođenju modernih sistema, koji će povećati produktivnost rada kroz automatizovan način obrade podataka.

Korišćenjem ovakvih sistema moguće je upravljati procesom rada kroz opcije efikasnijeg ažuriranja i pregleda podataka. Važno je istaći integrisanost sistema, jer su sve funkcije i podaci integrisani u relacionoj bazi podataka, gde korisnici pomenutih sistema imaju prikladan korisnički interfejs koji je prilagođen samim korisnicima radi bržeg savladavanja i nesmetanog obavljanja posla.

10. Literatura

[1] Eric J. Naiburg ,Robert A. Maksimchuk „UML for Database Design“ ,July 01,2001

[2] Mike O'Docherty „Object-Oriented Analysis and Design“, John Wiley & Sons 2005

[3] By Dr. Neil Roodyn „eXtreme .NET: Introducing eXtreme Programming Techniques to .NET Developers“ ,Addison Wesley Professional, December 2004

[4] Pejić Jovan „Kriptografski i stenografski postupci“, diplomski rad, Tehnički fakultet "Mihajlo Pupin", Zrenjanin, 2006.

[5] Dr. Radosav Dragica, „Softversko inženjerstvo I“, Tehnički fakultet "Mihajlo Pupin", Zrenjanin, 2005.

[6] Dr. Radosav Dragica, „Softversko inženjerstvo II“, Tehnički fakultet "Mihajlo Pupin", Zrenjanin, 2005.

[7] Dr.Radulović Biljana, Ljubica Eremić, Kazi Zoltan, „Odabrana poglavlja projektovanja informacionih sistema“, Tehnički fakultet "Mihajlo Pupin", Zrenjanin, 2002.

[8] Rockford Lhotka „Expert C# 2005 Business Objects“ , Apress 2006

[9] Stephen C. Perry „Core C# and .NET“ , September 2005

11. Prilozi

Propratni CD sadrži:

1. izvorni kod softvera za praćenje i podršku rada šahovskih klubova, instalacionu verziju navedenog softvera, dokumentaciju koja se nalazi u formatu *MS Word 2003,* kao i potrebne projekte (bpm, oom, cdm, pdm i mdb) fajlove.
2. seminarski rad iz softverskog inženjerstva „Informacioni sistem šahovskog kluba“ – kompletna dokumentacija sa instalacionim fajlom.