Collaborative Model Editor

1. Domen

Collaborative Model Editor aplikacija predstavlja softversko rešenje pisano u *C#* programskom jeziku koja korisnicima omogućava kreiranje i iscrtavanje objekata (entiteta), kao i definisanje relacija između njih. Na taj način, ova aplikacija dozvoljava kreiranje modela koji se mogu koristiti u različitim fazama razvoja krajnjeg softverskog proizvoda. Sama aplikacija razvijena je u skladu sa konceptima klijent-server arhitekture, a sastoji se iz četiri projekta koji će kasnije svaki pojedinačno biti opisan. GUI je poprilično jednostavan, kako bi korisniku bilo omogućeno jasno i lako snalaženje i korišćenje aplikacije.

Od entiteta koji se mogu iscrtavati na radnoj površini aplikacije, korisnicima su dostupna dva tipa objekata koji su reprezentovani pravougaonicima crvene i plave boje, relacije između dva objekta i atributi samog objekta koji su reprezentovani labelama. Detaljnije o svakom entitetu i funkcionalnostima svakog od njih biće opisano kasnije.

S obzirom na to da je ovde reč o kolaborativnoj aplikaciji, neophodno je objasniti pojam kolaboracije. U opštem značenju, kolaboracija, odnosno kolaborativni rad, predstavlja zajednički rad većeg broja subjekata nad zajedničkim, deljenim resursom, pri čemu se subjekti mogu nalaziti na udaljenim lokacijama. U kontekstu ove aplikacije, to će značiti da više korisnika može raditi nad istim modelom, tj. više korisnika može izvršavati sve operacije koje aplikacija nudi kako bi prilagođavali kreirani model sopstvenim potrebama. Pri tome, osnovni zahtev ovog tipa aplikacija je da se održi konzistentnost deljenog resursa (modela – u primeru ove aplikacije), odnosno da svi subjekti koji rade nad tim deljenim resursom imaju potpuno istu sliku modela.

2. Opseg

Collaborative Model Editor kao aplikacija obezbeđuje različite funkcionalnosti korisnicima i daje jedno rešenje problemu koji je u današnje vreme veoma aktuelan u svetu informacionih tehnologija. Taj problem je izvršavanje Undo i Redo funkcionalnosti u okviru kolaborativnih aplikacija. Funkcije koje ova aplikacija korisnicima nudi su sledeće:

- ***Iscrtavanje objekata*** – podrazumeva da se na klik mišem preko panela, odnosno radne površine, iscrta objekat koji je prethodno selektovan. Ukoliko su selektovani pravougaonici crvene ili plave boje, oni će slobodno biti iscrtani. Da bi se iscrtala relacija, neophodno je da budu selektovana **tačno dva** objekta, a da bi se nekom objektu dodao atribut potrebno je da bude selektovan **samo taj** objekat. U smislu kolaborativne aplikacije, ova funkcionalnost neće imati nikakvog ograničenja i korisnici će moći da dodaju objekte slobodno, pri čemu će oni biti vidljivi svim ostalim korisnicima.

- ***Selektovanje objekata*** – već je pominjano kod prethodne funkcionalnosti aplikacije, a podrazumeva da se klikom na nacrtani objekat na radnoj površini

obezbeđuje prepoznavanje objekta koji je jedinstven u okviru same aplikacije, i na taj način se korisniku omogućavaju dodatne funkcije.

- ***Brisanje objekata*** – omogućava uništavanje objekta u kontekstu same aplikacije, a moguće je brisanje isključivo prethodno selektovanog objekta. Prilikom ove funkcije, postoje određena ograničenja s obzirom na činjenicu da je reč o kolaborativnom tipu aplikacije, a o tome će biti reči u daljem tekstu.

- ***Undo*** - podrazumeva poništavanje poslednje izvršene akcije. U skladu sa konceptima kolaboracije, ova funkcionalnost biće omogućena samo pod određenim uslovima, a korisnik može poništiti samo aktivnosti koje je on izvršio nad modelom kao deljenim resursom.

- ***Redo*** – podrazumeva ponovno izvršavanje prethodno poništene akcije. Korisniku je dozvoljeno da izvrši ovu funkcionalnost samo jednom, bez obzira koliko puta je prethodno pozvao funkciju *Undo*.

3. Arhitektura aplikacije

Kao što je prethodno pomenuto, sama aplikacija sastoji se iz četiri manja projekta, koji će svaki pojedinačno predstavljati posebne module. Ti moduli su:

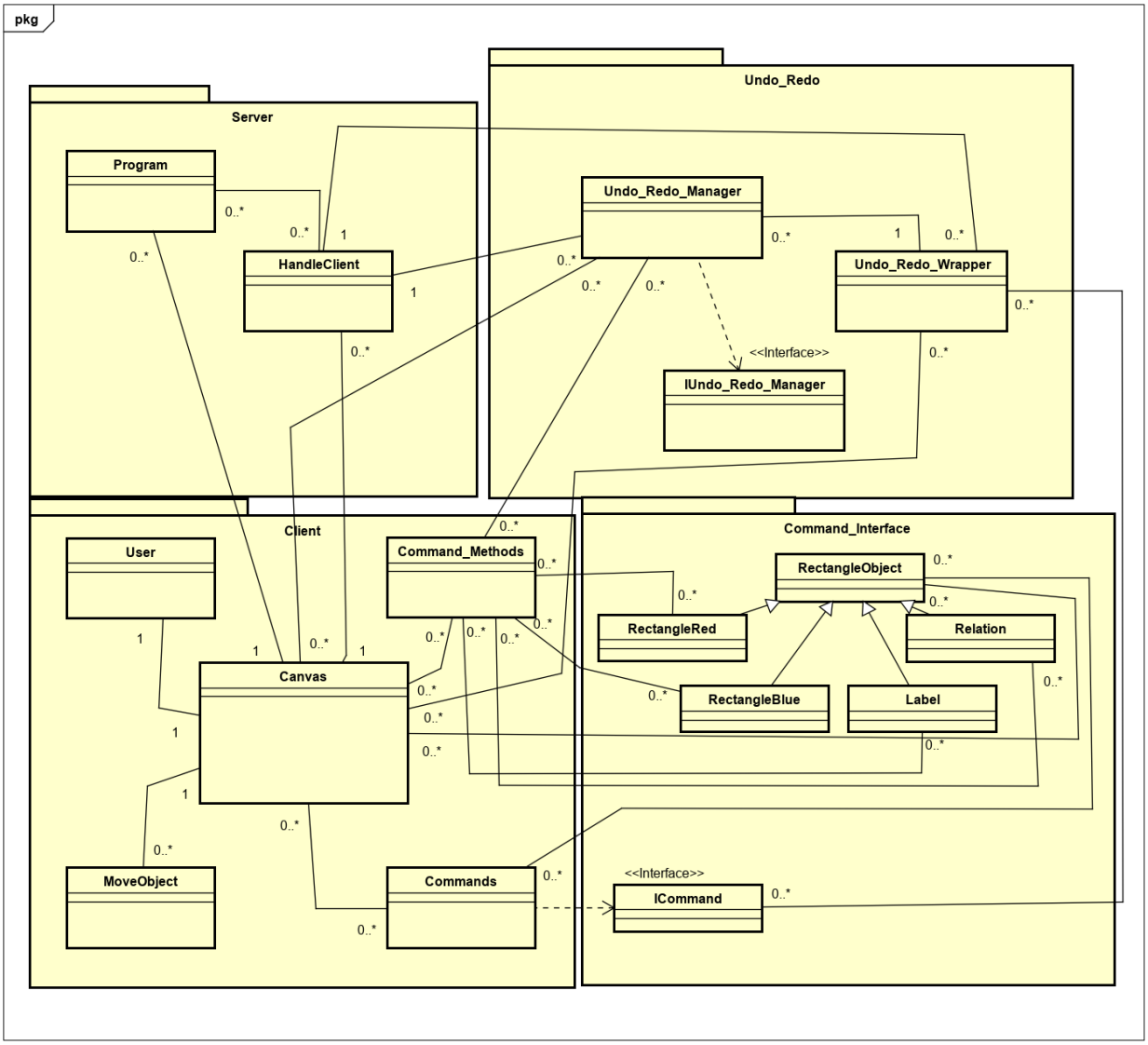
- ***Client*** – modul koji predstavlja glavnu komponentu GUI-a. Sadrži sve metode koje omogućavaju iscrtavanje objekata na radnoj površini aplikacije.

- ***Server*** - modul koji omogućava komunikaciju između klijenata. Svaka aktivnost koju klijent izvrši nad modelom, šalje se na server i tamo se obrađuje. Takođe, server pristigle aktivnosti od jednog klijenta šalje svim ostalim klijentima i na taj način obezbeđuje očuvanje konzistentnog stanja modela i kolaboraciju između različitih korisnika.

- ***Undo_Redo_Manager*** – modul koji rešava problem kolaborativnog izvršavanja Undo/Redo funkcionalnosti. Sadrži metode i provere koje su ključne za obezbeđenje iste slike modela kod svih klijenata.

- ***Command_Interface*** – modul koji sadrži modele svih objekata koji se mogu iscrtavati na radnoj površini i strukturu komandi. Komande će predstavljati objekte u koje će se injektovati aktivnosti nad samim modelom. Biće u jedinstvenom formatu i omogućavače obradu svih aktivnosti na isti način.

Grafički se arhitektura same aplikacije *Collaborative Model Editor* može predstaviti putem sledećeg dijagrama klasa:



Slika 1- Dijagram klasa cele aplikacije

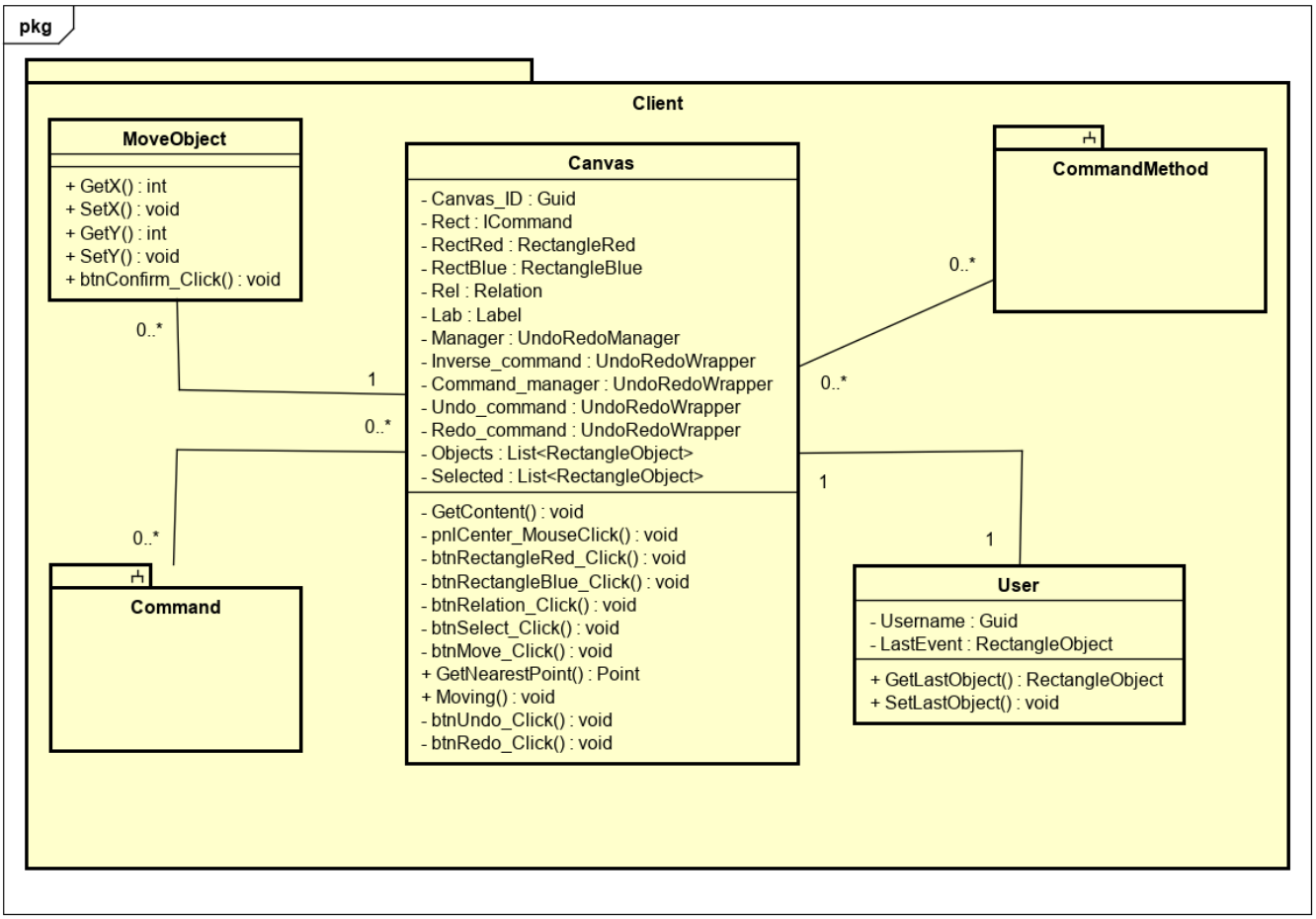
Kao što se sa dijagrama klasa može videti (Slika 1), navedena četiri projekta su međusobno povezana i između njih postoje različite vrste interakcija. Da bi komunikacija između projekata bila moguća, neophodno je postavljanje adekvatnih referenci u okviru svakog od projekata. S obzirom na to da je ova aplikacija razvijena u skladu sa konceptima klijent-server arhitekture, jasno je da jedan server može opsluživati više klijenata istovremeno. Sam server, pored opsluživanja klijenata, mora sadržati relevantnu i konzistentnu sliku modela kojoj će svaki klijentski program težiti, odnosno u slučaju postojanja određenih neusaglašenosti, slika na klijentu prilagodiće se serverskoj slici.

Command_Interface kao projekat je celina koja sadrži isključivo klase koje će predstavljati model objekata koji se mogu iscrtavati na klijentu u okviru metoda klase Canvas. U okviru Undo_Redo projekta, nalazi se logika za samo proveravanje mogućnosti izvršavanja funkcija

Undo i Redo. Metode u kojima je ta logika smeštena pozivaće se pri svakom pokušaju izvršavanja tih funkcija na klijentu, u cilju očuvanja konzistentne slike modela pri kolaborativnom radu.

4. Detaljan opis pojedinačnih projekata

- ***Client***



Slika 2 - Dijagram klasa modula Client

Kao što je već rečeno, deo aplikacije smešten u okviru *Client* projekta predstavlja ključni deo za realizaciju iscrtavanja svih objekata na radnoj površini. Svaki klijentski program (s obzirom na činjenicu da ih može biti više) sadržaće jedinstveno identifikaciono obeležje. To obeležje biće smešteno u property *Canvas_ID* u okviru klase *Canvas* i biće tipa Guid. Kao što je sa prikaza ovog modula vidljivo (Slika 2), glavna logika za iscrtavanje objekata i pozive funkcija koje to omogućavaju, biće smeštena u okviru klase *Canvas*.

Klasa *Canvas*, kao glavna klasa za vizuelno reprezentaciju svih funkcionalnosti aplikacije, obuhvata metode za obrađivanje događaja koji se dešavaju na odgovarajuće klijentske aktivnosti nad samom formom aplikacije. Pod tim podrazumevaju se aktivnosti korišćenja GUI od strane korisnika. Svaki od tih događaja menja stanje modela lokalno kod klijenta koji izvršava akciju, ali i na samom serveru. U skladu sa tim, svaka od ovih metoda sadrži deo slanja same aktivnosti na server kako bi se moglo izjednačiti stanje modela na svim klijentima.

Ova klasa, pored metoda koje obrađuju događaje, sadrži i metodu za izračunavanje najbližih tačaka između dva objekta (metoda *GetNearestPoint*), kao i metodu *GetContent* koja obezbeđuje kolaboraciju između različitih klijenata prikačenih na isti server.

GetContent je ključ dobre komunikacije klijenata sa serverom. Komunikacija ove dve komponente napravljena je tako da svaki klijent osluškuje šta se dešava na serveru. U slučaju da server šalje klijentu zahtev da određenu akciju izvrši, ova metoda taj zahtev obrađuje. Naime, kad se sa servera pošalje zahtev klijentu da se određena aktivnost izvrši nad modelom koji se nalazi na radnoj površini, taj zahtev dobija se u odgovarajućem formatu. Metoda *GetContent* vrši „raspakivanje“ pristiglog zahteva i na osnovu njega prepoznaje koju aktivnost treba da izvrši.

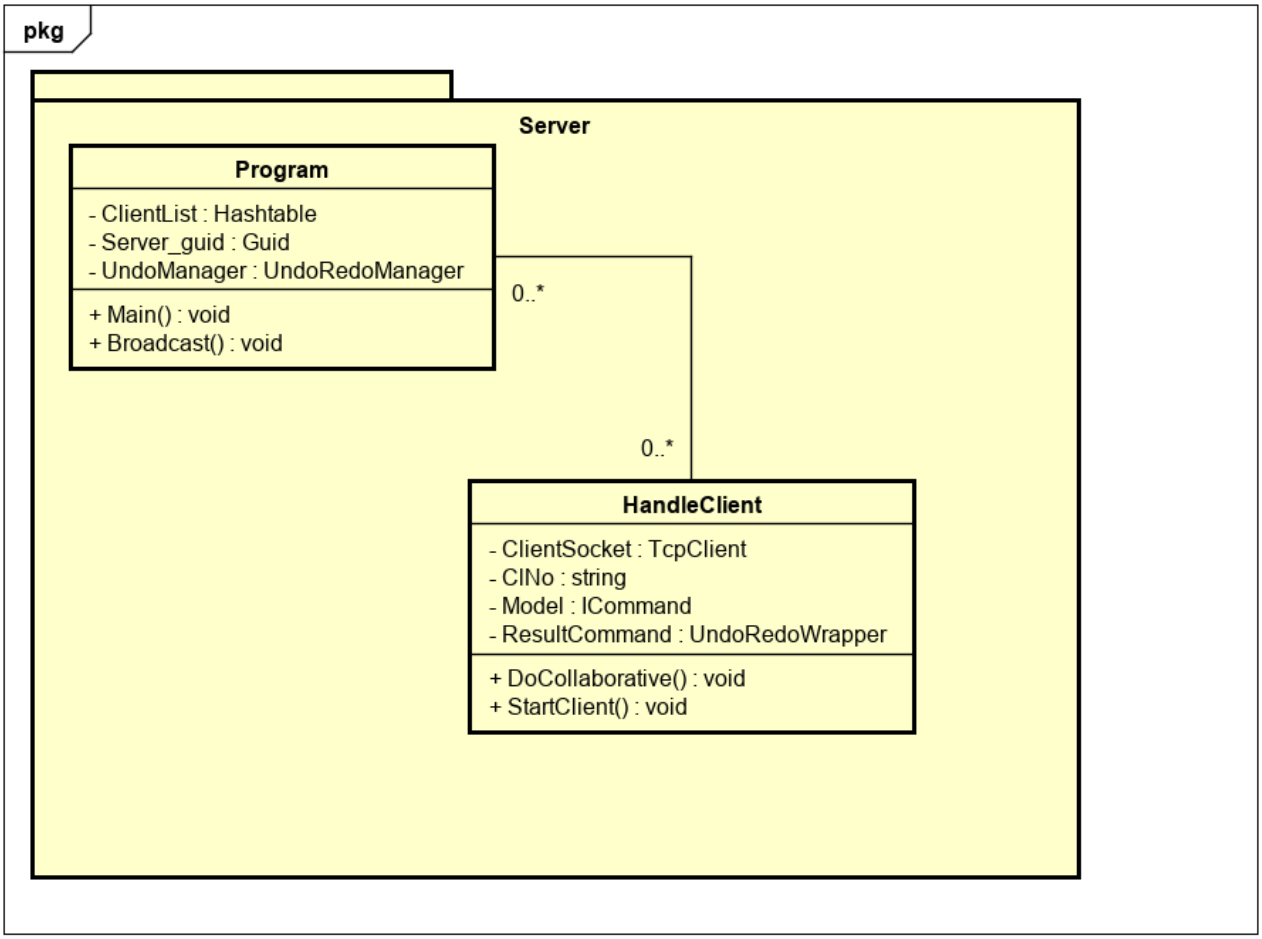
Klasa *User* modeluje korisnika aplikacije. Jedan korisnik aplikacije moći će da pokrene samo jedan klijentski prozor, i stoga će on uniformno moći da se identifikuje na osnovu *Username*-a korisnika. Takođe, property *Username* biće veoma bitan kako bi se pratilo koju aktivnost je izvršio koji korisnik.

Klasa *MoveObject* predstavlja pomoćnu formu koja će biti aktivna samo u situacijama pomeranja nekog od objekata. Pojaviće se u okviru GUI-a prilikom klijentskog pritiska na dugme *Move*, i na taj način će biti omogućeno menjanje stanje objekta.

U okviru paketa *Commands*, nalazi se veliki broj klasa (njih 18.) koje će služiti kako bi se same aktivnosti upakovale u odgovarajući format. S obzirom na arhitekturu same aplikacije i činjenicu da u opštem slučaju jedan server može opsluživati veći broj klijenata, ovim „pakovanjem“ aktivnosti omogućena je lakša obrada i prenošenje aktivnosti između klijenata i servera.

Konačno, klasa *CommandMethods* sadrži veliki broj različitih metoda koje će omogućiti izvršavanje aktivnosti na ostalim klijentima, odnosno onim klijentima koji nisu inicirali odgovarajuću aktivnost. Sve te metode pozivaju se iz metode *GetContent*, kada klijent osluškivanjem servera prepozna da treba da izvrši odgovarajuću operaciju.

- ***Server***



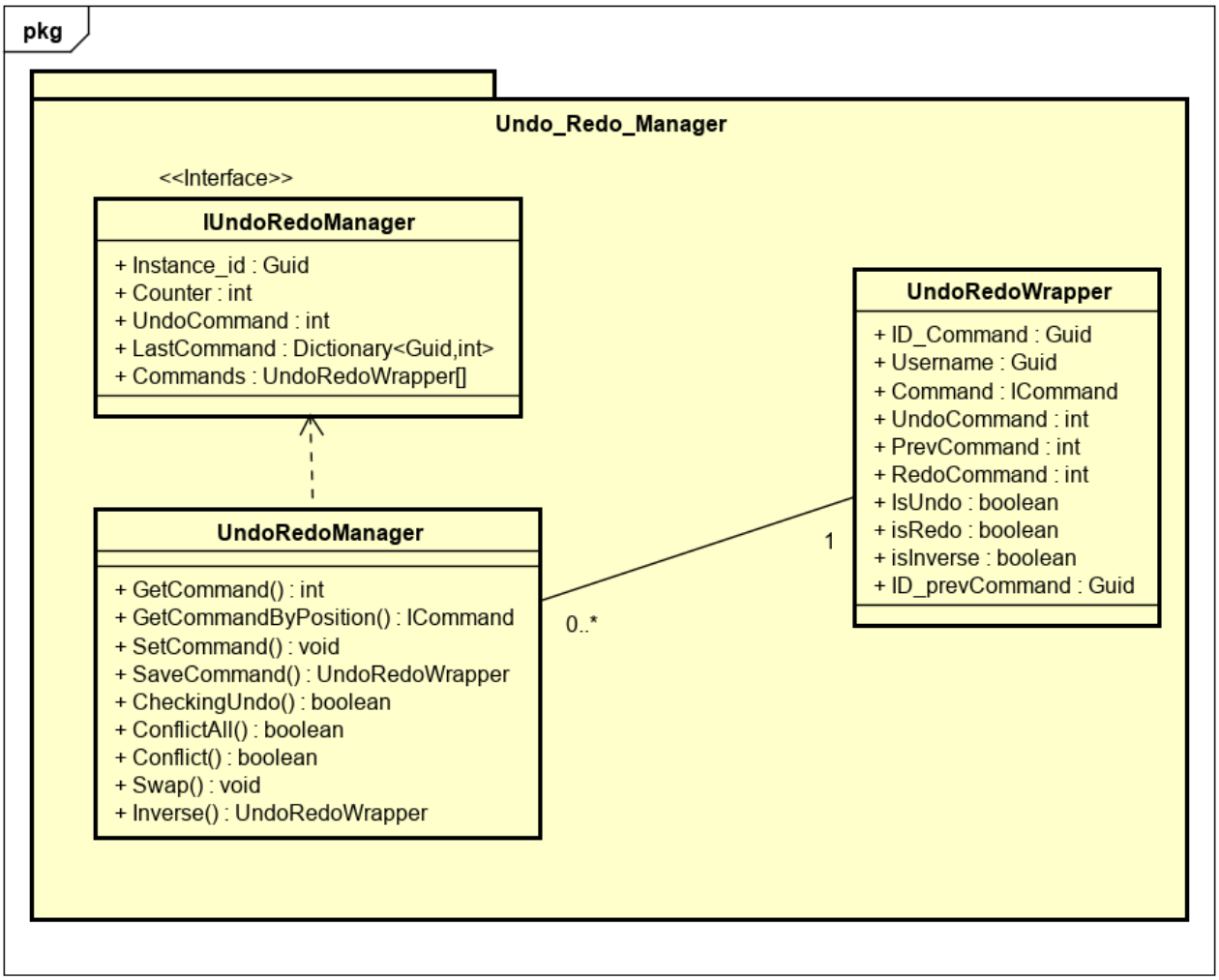
Slika 3 - Dijagram klasa modula Server

Server predstavlja ključnu komponentu za obezbeđenje konzistentnosti i kolaboraciju između različitih korisnika aplikacije. On se sastoji iz dve klase: *Program* i *HandleClient*.
Klasa *Program* namenjena je za pokretanje servera i podešavanje njegovih parametara, što se radi u okviru *Main* metode, kao i za slanje poruka svim klijentima kojima su povezani na serveru, u okviru *Broadcast* metode. Poruka koja se šalje klijentima sadrži aktivnost koja se treba izvršena, u formatu koji će svaki klijent lako razumeti.

Klasa *HandleClient* namenjena je za obrađivanje klijenata i njihovih poruka ka serveru. Naime, u okviru metode *StartClient* se vrši obrada svakog klijenta koji serveru pristupi i od tog trenutka omogućena je svaka dalja njegova komunikacija. Sve poruke koje klijent šalje ka serveru, odnosno sve aktivnosti koje klijent izvršava lokalno nad modelom, obrađuju se u okviru metode *DoCollaborative*. Ova metoda obezbeđuje proveru mogućnosti izvršavanja odgovarajuće aktivnosti, i „pakovanje“ same poruke koja će biti poslata ostalim klijentima ako se ona može izvršiti.

Takođe, možda najbitnija uloga servera u svetu kolaborativnih aplikacija je da on mora čuvati relevantno stanje modela i svaki klijent mora težiti da ima istu sliku modela kao što se nalazi na samom serveru.

- ***Undo_Redo_Manager***



Slika 4 - Dijagram klasa modula Undo_Redo_Manager

S obzirom na to da je kluč ove aplikacije rešavanje problema izvršvanja funkcija *Undo* i *Redo* u kolaborativnim uslovima, može se reći da je ovo najbitniji modul aplikacije. Sva logika za rešavanje tog problema smeštena je u okviru klase *UndoRedoManager*. Ona implementira interfejs *IUndoRedoManager* koji sadrži određena obeležja za identifikaciju svakog objekta klase *UndoRedoManager*. Svaki od objekata sadržaće i *Dictionary LastCommand* koji će sadržati poslednje izvršene komande svakog korisnika aplikacije i niz svih izvršenih

komandi nad određenim modelom (niz *Commands*). Komanda predstavlja aktivnost nad modelom (npr. iscrtavanje, brisanje, pomeranje itd.).

UndoRedoWrapper predstavljaće „omot" u koji će svaka komanda biti smeštena. Takva struktura bitna je iz razloga što će svaka komanda u nizu svih komandi (*Commands* iz interfejsa *IUndoRedoManager*) imati odgovarajuće pokazivače i property-e koji će je jedinstveno označavati i uticati na rešavanje problema *Undo* i *Redo* funkcionalnosti. Naime, za svaku komandu će eksplicitno biti naznačeno da li je ona *Undo* komanda, *Redo* komanda, invertovana komanda ili regularna komanda na osnovu property-a *IsUndo*, *IsRedo* i *IsInverse*. Pored toga, komanda će imati i odgovarajuće pokazivače:

- *PrevCommand* – ukazivaće na prethodnu komandu koju je napravio taj korisnik.
- *UndoCommand* – ukazivaće na komandu za koju je izvršena funkcija *Undo*, a u svakom drugom slučaju ovaj pokazivač neće pokazivati ni na jednu komandu iz niza.
- *RedoCommand* – ukazivaće na komandu koja će se izvršiti pozivanjem *Redo* funkcije, u slučaju ako je prethodno izvršena *Undo* funkcija.

Na osnovu ovih obeležja, kreirani „omot" biće pogodan za izvršavanje svih provera mogućnosti izvršavanja određene aktivnosti u kolaborativnom svetu.

Klasa *UndoRedoManager* sadrži veći broj metoda koje predstavljaju logiku za proveru mogućnosti izvršavanja korisničkih aktivnosti i obezbeđenje jedinstvene slike modela na svakom od klijentskih programa u kolaborativnom radu. U narednom delu biće objašnjena svaka funkcija ove klase:

1) *GetCommand* – metoda koja vraća indeks poslednje komande koju je određeni korisnik izvršio. Kao ulazni parametar prihvata *Username* korisnika, i na osnovu njega pronalazi poslednju komandu iz Dictionary-ja *LastCommand*.

2) *GetCommandByPosition* – metoda vraća konkretnu komandu iz *Commands* niza svih komandi na osnovu prosleđene pozicije u nizu.

3) *SetCommand* – prilikom svakog izvršavanja neke komande (odnosno aktivnosti), poziva se ova metoda. Ona postavlja izvršenu komandu kao poslednju komandu koju je korisnik izvršio u okviru *LastCommand*-a – menja stanje ovog Dictionary-ja.

4) *SaveCommand* – metoda koja izvršava snimanje komande u niz svih komandi *Commands*. Ovo je najbitnija metoda u okviru ovog modula. U telu ove metode se postavljaju gore objašnjeni pokazivači (*PrevCommand*, *UndoCommand*, *RedoCommand*) na osnovu određenih uslova i provera. Snimanjem komande u niz, inkrementira se obeležje Counter koje predstavlja broj elemenata u nizu. Jednom upisane komande u niz nikada se ne brišu. Metoda kao ulazni parametar prima samu komandu, a rezultira na izlazu upakovanim „omotom" komande (UndoRedoWrapper) koji će u suštini biti jedinica razmene podataka između klijenta i servera.

5) *CheckingUndo* – metoda koja proverava da li je izvršen *Undo* neke komande na osnovu definisanih pokazivača u okviru metode *SaveCommand*.

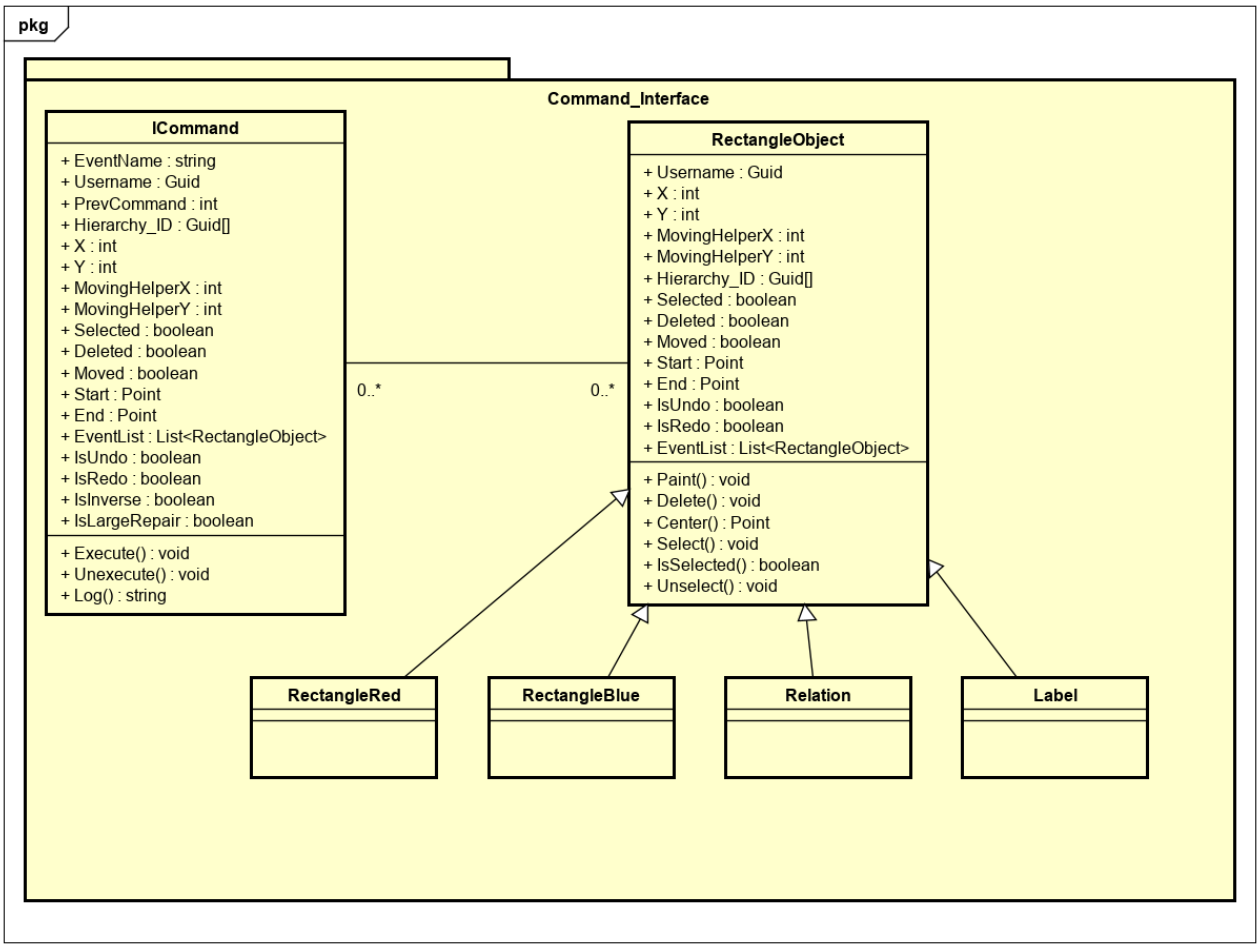
6) *ConflictAll* – metoda koja obezbeđuje konzistenciju slike modela na različitim klijentima. Pozivanjem metode *Conflict*, ova metoda će proći kroz čitav niz komandi i proveriti da li je određenu aktivnost moguće izvršiti. Ukoliko će željena aktivnost narušiti konzistentnost, odnosno pojaviće se konflikt između komandi, ta aktivnost neće biti izvršena.

7) *Conflict* – na osnovu obeležja *Hierarchy_ID* svake komande proverava se da li su komande u konfliktu. Konflikt između komandi pojaviće se ako je želja da se određena aktivnost izvrši nad nekim postojećim objektom, pri čemu će ta aktivnost narušiti potencijalno konzistentnost modela. Taj deo provere i očuvanja konzistentnosti biće izvršen pomoću obeležja *Hierarchy_ID*, koje će biti objašnjeno u kasnijem delu dokumenta.

8) *Swap* – metoda koja će zameniti pozicije dvema komandi (komandi koja se želi izvršiti i njene prethodne komande) u okviru pomoćnog niza komandi. Koristi se prilikom provere konflikta i u slučaju da ne postoji konflikt između komandi, komande će se u pomoćnom nizu zameniti. Ukoliko izvršena komanda može da se zameni sa svim ostalim komandama, zasigurno će ukazivati da njeno izvršavanje neće izazvati nikakav konflikt, odnosno neće dovesti do nekonzistentnog stanja modela, te će se ona slobodno izvršiti.

9) *Inverse* – metoda koja na osnovu prethodno definisanih pokazivača pravi inverznu komandu. Inverzna komanda praviće se samo u slučaju izvršvanja Undo funkcionalnosti. Inverzna komanda će omogućiti izvršavanje Undo-a, odnosno prepoznavanje koju aktivnost treba izvršiti da bi se ta funkcija uradila.

- ***Command_Interface***



Slika 5 - Dijagram klasa modula Command_Interface

Command_Interface predstavlja svojevrsnu biblioteku u okviru koje se nalaze modeli samih objekata koji se mogu iscrtavati i interfejs koji će implementirati sve komande koje se nalaze u okviru *Client* projekta.

RectangleObject je apstraktna klasa koja će definisati opšti model objekata koji će se iscrtavati. Sve metode u okviru ove klase su apstraktne, što znači da će konkretna implementacija metoda biti u okviru klasa koje nasleđuju *RectangleObject* klasu. To su klase: *RectangleRed, RectangleBlue, Relation, Label*. Svaka od tih metoda klase RectangleObject (*Paint, Delete, Center*...) implementira se na različit način u okviru različitih klasa.

Sa aspekta obeležja, na osnovu slike (Slika 5) se može videti da su klase *ICommand* i *RectangleObject* veoma slične. To je iz razloga što se kompletna komunikacija u okviru aplikacije, tj. komunikacija između klijenata i servera, odvija se posredstvom komandi kao

uniformne strukture koja je reprezentovana interfejsom *ICommand*. Dakle, svaka komanda u sebi sadržaće detaljne informacije o samom objektu za koji se određena aktivnost izvršava.

Pored samih parametara koji su neophodni za identifikaciju svakog objekta u okviru radne površine, interfejs komande sadrži i dodatne parametre koji su od ključnog značaja za njenu direktnu identifikaciju u sistemu. Najbitniji od njih je već pomenuti *Hierarchy_ID*.

Hierarchy_ID je niz Guid vrednosti koji jedinstveno identifikuje svaki objekat u sistemu. Već je rečeno da se svi objekti iscrtavaju na radnoj površini (Panel-u), kao i da se u okviru svakog iscrtanog pravougaonika crvene ili plave boje (koje reprezentuju klase *RectangleRed* i *RectangleBlue*) mogu dodati atributi (reprezentovani klasom *Label*). Na taj način, formira se hijerarhija objekata pri čemu će se na prvom nivou, odnosno u korenu hijerarhije, nalaziti radna površina, na narednom nivou biće pravougaonik određene boje, dok će na trećem nivou biti atribut. Takva hijerarhijska struktura modeluje se pomocu opisivanog obeležja *Hierarchy_ID*.

Kao što je već pomenuto u prethodnom delu, prilikom proveravanja konflikta između komandi, glavnu ulogu imaće baš ovo obeležje. Izvršavanje komandi nad objektima na različitim nivoima hijerarhije, može usloviti konflikt. Pri tome, bitno je naglasiti da svaka komanda preuzima vrednost ovog obeležja od samog objekta za koji je vezana.