ISPITNI ZADATAK

iz Softverskog inženjerstva

Filip Nikšić

7. rujna 2009.

Sadržaj

Poglavlje 1

Uvod

1.1 Tekst zadatka

Oznaka: SI_103

Razradite sustav koji omogućuje igranje igara tipa MUD preko Interneta ili LAN-a. Svaka igra (može ih biti više neovisnih na raspolaganju) se zbiva u nekom vremenskom i prostornom okruženju. (Npr. srednji vijek i magija, Drugi svjetski rat, znanstvena fantastika, suvremeno moderno okruženje s nekim zanimljivim detaljem itd.) Igrač na zahtjev mora dobiti sve raspoložive podatke koji mu olakšavaju igranje. Svaki igrač kreira i vodi jedan lik (ili više njih). Prilikom kreiranja lika igrač može donekle utjecati na njegove osobine koje se mogu mijenjati tokom igre u ovisnosti o aktivnostima kojima se lik bavi (sjetite se RPG igara). Uz likove koje vode igrači (tzv. PC likovi), u igri se pojavljuju i mnogi drugi. O njima, njihovim aktivnostima i osobinama računa vodi sam sustav. Igra omogućuje interakciju svih likova gdje god je to moguće (razgovor, telefoni, sukobi, ...) i komunikaciju svih igrača, neovisno o igri (sjetite se IRC-a ili ICQ-a). U igri se mogu pojavljivati i neki predmeti od kojih su neki interesantni zbog svoje vrijednosti (zlato, dragulji, ...), upotrebljivosti (voda, hrana, oružje), zanimljivosti (oružje s magičnim osobinama, napredna oruđa, ...) Neki predmeti mogu gotovo u potpunosti imati osobine živih bića.

Sustav može poslužiti i za opisivanje i prezentaciju nekog stvarnog okruženja, npr. turistički obilasci (ovdje mora biti izražen grafički element).

1.2 Struktura dokumenta

Prvo dajemo opis sustava u cjelini. Razrađujemo model klijent–server i opisujemo korišteni protokol za komunikaciju. Općenitost zadanih zahtjeva pokušavamo obuhvatiti vrlo

velikom konfigurabilnošću sustava: na administratoru servera je da specificira detalje o objektima i njihovim karakteristikama u svijetu u kojem se igra odvija. U tu svrhu na raspolaganju mu je ugrađeni skriptni jezik.

U ostatku rada posebno razrađujemo poslužitelj (poglavlje 3) i klijent (poglavlje 4). Poslužitelj zahtjeva nešto više pažnje zbog svoje kompleksnosti, dok kod klijenta ističemo jedino korisničko sučelje. U procesu izrade dokumenta korišten je objektno-orijentirani pristup i UML. UML dijagrami izrađeni su alatom BOUML [2]. Prototip korisničkog sučelja klijenta napravljen je korištenjem PyQt4 tehnologije [3] (kombinacija jezika Python i Qt grafičke biblioteke).

Poglavlje 2

Sustav u cjelini

2.1 Okruženje

Okruženje igre, tj. svijet koji modelira poslužitelj sastoji se od soba i prolaza. Pod pojmom soba podrazumijevamo bilo koji prostor u kojem se igrač može nalaziti. To može biti livada, kuća, rupa u zemlji i sl. Prolaz povezuje sobe i jednosmjeran je. Ukratko, svijet možemo zamišljati kao usmjereni graf kojem su čvorovi sobe, a bridovi prolazi.

U svijetu se mogu nalaziti likovi, predmeti i NPC-ovi. Likove stvaraju i njima upravljaju igrači spojeni na poslužitelj. Mogu ih voditi iz sobe u sobu koristeći prolaze, mogu uzimati predmete na koje pritom naiđu, trgovati i boriti se s drugim likovima ili NPC-ovima. Predmeti su razni objekti poput novca, oružja, hrane i sl. Nalaze se razasuti po sobama, ili u posjedu likova. NCP-ovi su likovi kojima upravlja sustav.

Svaki objekt (bilo soba, prolaz, lik ili predmet) ima određena svojstva. Primjeri svojstava za likove mogu biti inteligencija, snaga, spretnost, nivo, iskustvo, život, mana. Budući da su zahtjevi u zadatku toliko općeniti, sustav zahtjeva od administratora da sam specificira svojstva i na koji način ona utječu na objekte u igri.

2.2 Model klijent–poslužitelj

Prirodno se nameće model klijent–poslužitelj. Poslužitelj je konstantno pokrenut i predstavlja okruženje u kojem se igra odvija. Prihvaća konekcije klijenata (koji predstavljaju igrače) i upravlja njihovom interakcijom i komunikacijom. Stanje okruženja na početku rada učitava, a na kraju pohranjuje u bazu.

Klijent se sastoji od grafičkog sučelja na kojem dominira polje za unos teksta i polje u kojem pišu odgovori poslužitelja. Nakon što se spoji na poslužitelj, klijentu na raspolaganju stoji skup naredbi. Sva komunikacija odvija se prema protokolu: klijent upućuje

naredbu, poslužitelj je obradi i šalje odgovor.

2.3 MUD Client Protocol

Za komunikaciju između poslužitelja i klijenta služimo se protokolom baziranim na MUD Client Protocolu, verziji 2.1. U ovom odlomku ukratko opisujemo taj protokol. Detaljne specifikacije mogu se pronaći na stranici [1].

Komunikacija se ostvaruje izmjenom *mrežnih linija* (engl. *network lines*). To su jednostavno nizovi ASCII znakova koji izmjenjuju posredstvom TCP/IP protokola. Razlikujemo *in-band* i *out-of-band* linije. Prvo se odnosi na nestrukturirane linije koje mogu nositi bilo kakvu informaciju; poslužitelj ili klijent ih interpretira na neki uobičajen način. Drugo su strukturirane linije kojima se prenose *MCP poruke*. Njihova struktura zadana je kontekstno-slobodnom gramatikom.

```
<message-line> ::= '#$#' <message> EOL

<message> ::= <message-start>
            | <message-continue>
            | <message-end>
<message-start> ::= <message-name> <space> <auth-key> <keyvals>
```

Svaka out-of-band poruka počinje prefiksom `#$#`, na što se nastavlja poruka. Poruka ima svoje ime, ključ za autentikaciju i argumente. Ime služi za određivanje akcije koja se treba poduzeti. Najčešće će se raditi o naredbi koju igrač želi izvršiti. Ključ za autentikaciju se generira na početku komunikacije i služi kao svojevrsna rudimentarna zaštita. Argumenti su dani kao niz ključnih riječi s pridruženim vrijednostima.

```
<keyvals> ::= '' |  <space> <keyval> <keyvals>
<keyval> ::= <key> ':' <space> <value>

<key> ::= <simple-key> | <multiline-key>
<simple-key> ::= <ident>
<multiline-key> ::= <ident> '*'

<value> ::= <unquoted-string> | <quoted-string>
```

Ključna riječ je identifikator argumenta. Ukoliko završava zvjezdicom, to je signal da se vrijednost argumenta proteže kroz više linija. Inače se radi o jednostavnoj vrijednosti.

```
<message-continue> ::= '*' <space> <datatag>
                       <space> <simple-key> ':' ' ' <line>
<message-end> ::= ':' <space> <datatag>
<datatag> ::= <unquoted-string>
<line> ::= '' | <line-char> <line>
```

U slučaju višelinijskih vrijednosti, jednostavna vrijednost u početnoj poruci se ignorira (najčešće se stavi prazan string ""), a stvarna vrijednost iščitava se iz niza poruka koje slijede. Ako početna poruka sadrži višelinijsku vrijednost, mora u listi argumenata imati argument s ključnom riječi `_data-tag`. Taj argument se, zajedno s ključnom riječi višelinijske vrijednosti, koristi za označavanje nastavljajućih poruka. Semantika nastavljajućih poruka je sljedeća:

- Umjesto autentikacijskog ključa, nastavljajuća poruka ima *datatag*, vrijednost argumenta `_data-tag` iz početne poruke.
- Nastavljajuća poruka ima samo jedan par ključa i vrijednosti, i to ključ iz početne poruke i dio odgovarajuće višelinijske vrijednosti. Valja napomenuti da se sve nakon dvotočke i razmaka smatra dijelom višelinijske vrijednosti.

Kad su svi dijelovi svih višelinijskih vrijednosti poslani pomoću nastavljajućih poruka, potrebno je signalizirati kraj pomoću završne poruke: dvotočke iza koje slijedi *datatag*.

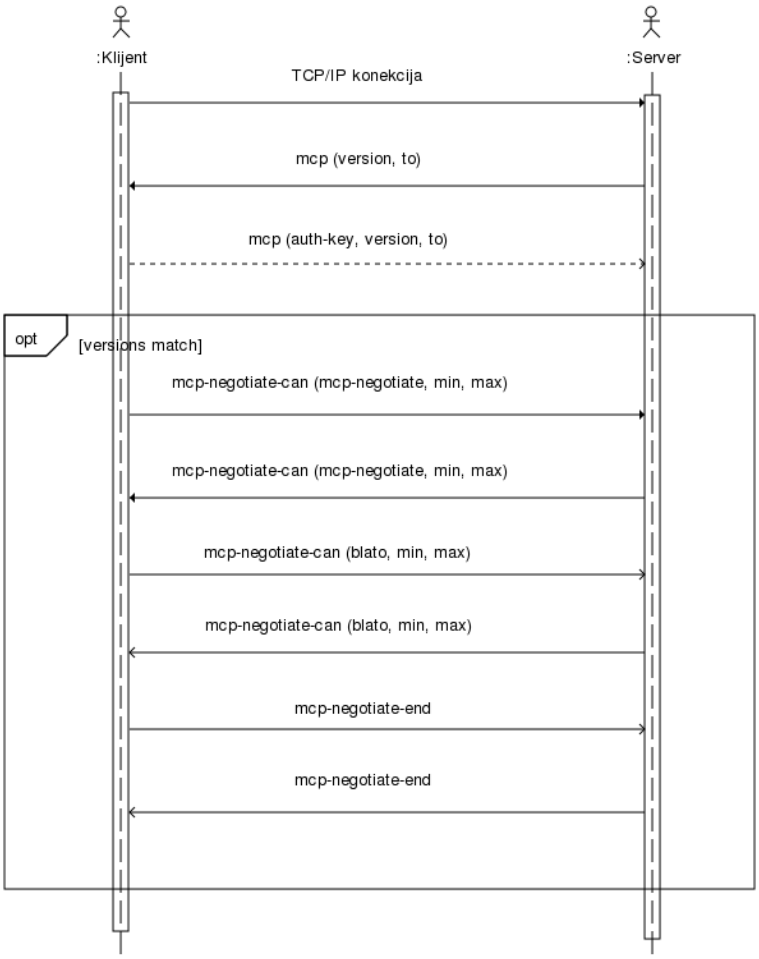
Pokažimo na primjeru kako izgleda poruka samo s jednostavnim vrijednostima:

```
#$#say 5d15e709 from: "Radagast the Brown" to: "Seraph Rascal"
               what: "0 je prirodan broj."
```

Pogledajmo sad primjer niza poruka kojima se prenosi višelinijska vrijednost:

```
#$#send 5d15e709 from: "Radagast the Brown" text*: ""
                 _data-tag: c3df
#$#* c3df text: Ovo je primjer teksta koji se
#$#* c3df text: proteže kroz nekoliko linija.
#$#* c3df text:
#$#* c3df text: Na ovaj način ne moramo koristiti navodnike
#$#* c3df text: za označavanje stringova. Možemo uključiti
#$#* c3df text: "specijalne" znakove. Sve nakon dvotočke i
#$#* c3df text: razmaka smatra se dijelom vrijednosti. To
#$#* c3df text:     znači da su i razmaci dio vrijednosti.
#$#: c3df
```

Sad kad smo ukratko objasnili strukturu poruka (više o tome na spomenutoj stranici [1]), objasnimo ukratko komunikacijski protokol prikazan na sequence dijagramu 2.1. Na početku se server i klijent moraju usuglasiti oko podržane verzije MCP protokola, moraju razmijeniti autentikacijski ključ i ustanoviti koje verzije *paketa poruka* podržavaju. U tu svrhu se koriste dvije vrste MCP poruka: `mcp` i `mcp-negotiate-can`.



Slika 2.1: MCP protokol

Komunikacija pokreće uspostavom konekcije od strane klijenta. Prije bilo kakve izmjene poruka server mora klijentu poslati `mcp` poruku s rasponom verzija MCP protokola koje podržava. Npr.:

```
#$#mcp version: 2.1 to: 2.1
```

Klijent po primitku poruke odgovara vlastitom `mcp` porukom u kojoj osim raspona verzija šalje i autentikacijski ključ koji se koristi u svim sljedećim porukama. Npr.:

```
#$#mcp authentication-key: 5d15e709 version: 1.0 to: 2.1
```

I server i klijent dalje koriste maksimalnu obostrano podržanu verziju MCP protokola ako takva postoji.

Slijedi faza dogovora oko podržanih verzija paketa poruka. Naime, MCP poruke svrstavaju se u pakete. Sve MCP implementacije moraju imati podržavati barem jedan paket: `mcp-negotiate`, i to barem verziju 1.0. Taj paket sadrži poruke `mcp-negotiate-can` i `mcp-negotiate-end`. Prva služi za dojavljivanje drugoj strani koji paketi (zajedno s rasponom verzija) su podržani. Naravno, server i klijent se moraju usuglasiti oko verzije samog `mcp-negotiate` paketa. Stoga jedan drugom šalju poruku oblika:

```
#$#mcp-negotiate-can 5d15e709 package: mcp-negotiate
                     min-version: 1.0 max-version: 2.0
```

Napomenimo da nijedna strana ne smije čekati `mcp-negotiate-can` poruku s verzijom `mcp-negotiate` paketa prije nego pošalje vlastitu, kako bi se izbjegao komunikacijski dead-lock. Međutim, mora se primiti verzija `mcp-negotiate` paketa prije daljnjih `mcp-negotiate-can` poruka s verzijama ostalih paketa.

Kad je verzija `mcp-negotiate` paketa uspostavljena (opet, uzima se maksimalna obostrano podržana verzija), može početi slanje `mcp-negotiate-can` poruka za ostale pakete. Obje strane to rade asinkrono. Svaka strana nakon što je primila `mcp-negotiate-can` poruku (i poslala svoju) za neki paket, može odmah početi slati poruke iz tog paketa.

Nakon što je poslala sve `mcp-negotiate-can` poruke, strana završava ovu fazu komunikacije porukom `mcp-negotiate-end`:

```
#$#mcp-negotiate-end 5d15e709
```

Važno je napomenuti da nijedna strana ne smije čekati `mcp-negotiate-end` poruku prije nego pošalje `mcp-negotiate-can` poruku za neki svoj paket.

U našem sustavu, server i klijent implementirat će verziju 2.1 MCP protokola i verziju 2.0 `mcp-negotitate` paketa. Uz to, dodajemo podršku za vlastiti paket `blato` koji sadrži naše poruke:

- `blato-login`

  Poruka koja služi za pridruživanje igraču već postojećeg lika. Obavezni argumenti su `username` i `password`. Zbog sigurnosti, password se ne prenosi kao otvoreni tekst, nego se prenosi hash vrijednost.

- `blato-create`

  Poruka koja služi za stvaranje novog lika.

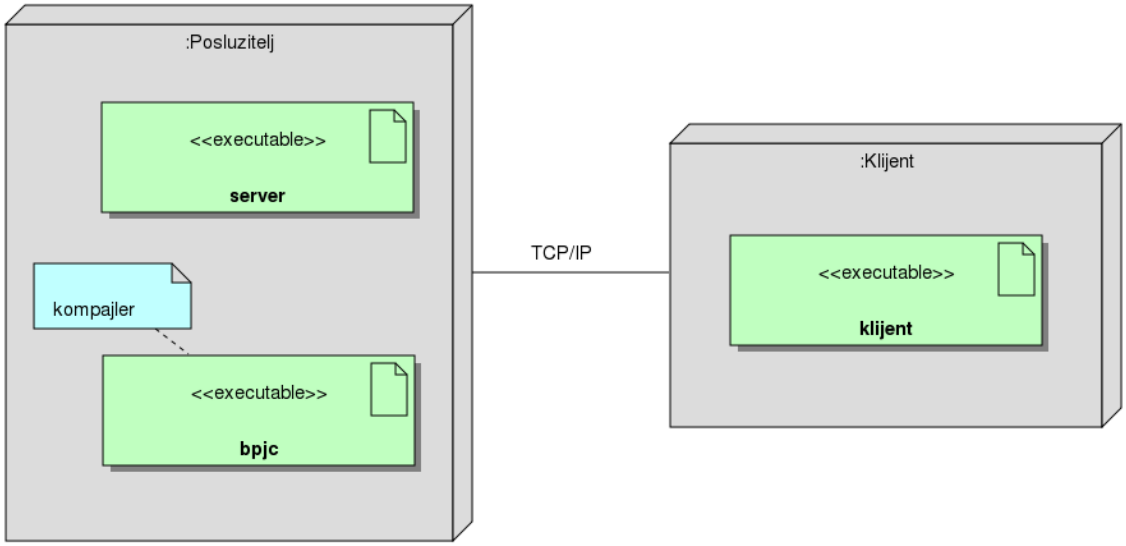
- `blato-action`

  Ova poruka ima obavezan argument `name` koji identificira akciju, tj. naredbu koju igrač želi izvršiti. Ovisno o akciji, mogući su dodatni argumenti.

2.4 Skriptni jezik

Skriptni jezik zasad ne specificiramo. Jedna mogućnost je razviti vlastiti jezik, a druga preuzeti neki od već postojećih, npr. Lua [4], i prilagoditi ga našim potrebama.

2.5 Deployment



Slika 2.2: Deployment dijagram

Poslužitelj se nalazi na računalu sa stalnom vezom na Internet. Klijenti se izvršavaju

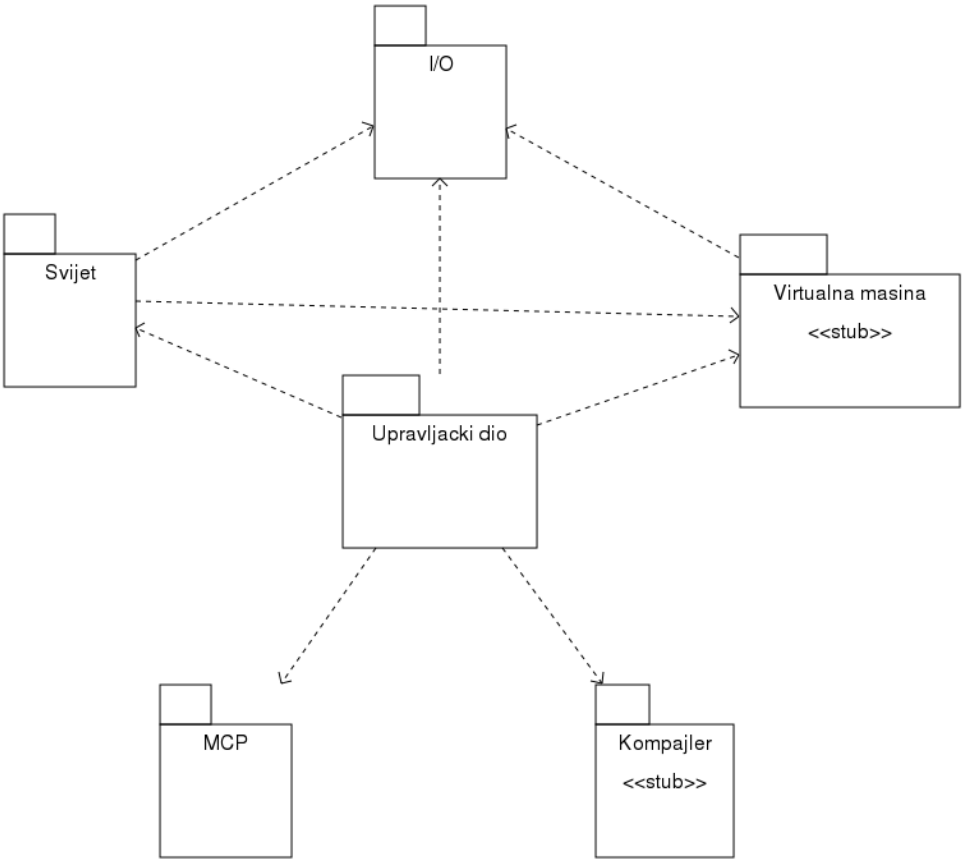
na bilo kakvim računalima s pristupom na Internet. Operacijski sustavi koji se podrazumijevaju moraju biti POSIX kompatibilni.

Kompajler izdvajamo u zasebnu izvršnu datoteku kako bi se mogao koristiti i kad server nije pokrenut.

Poglavlje 3

Poslužitelj

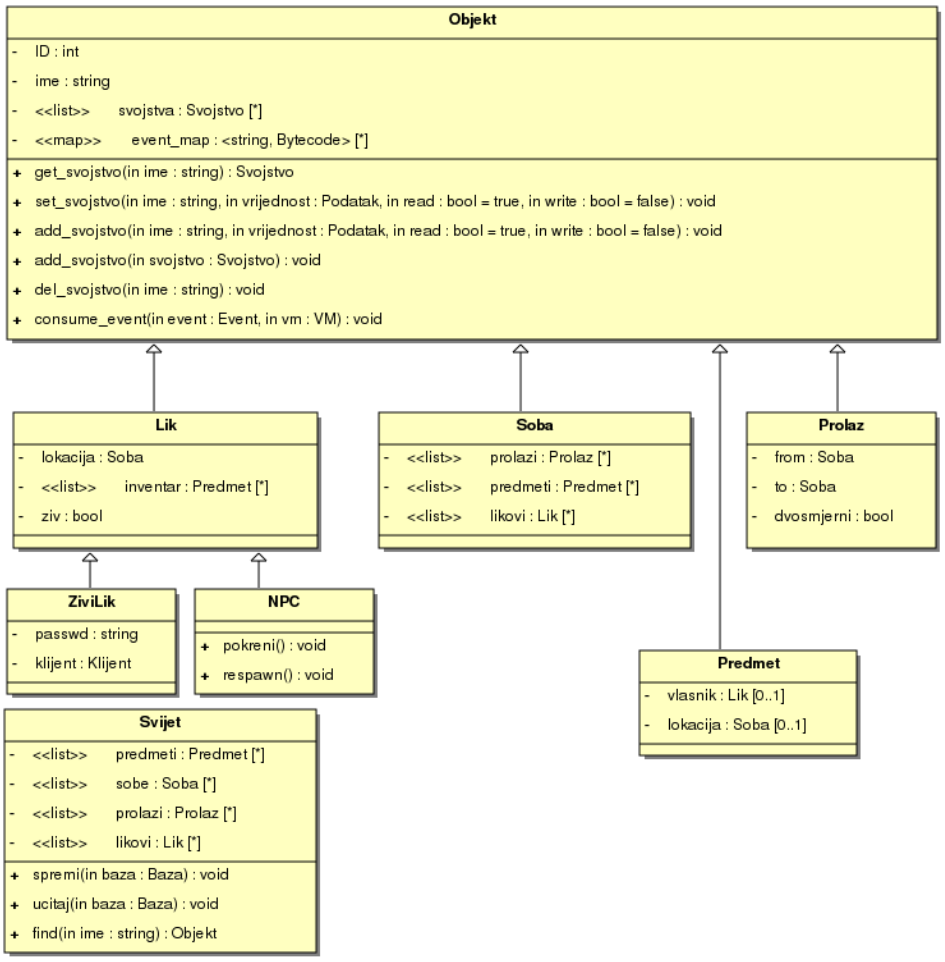
3.1 Package dijagram



Slika 3.1: Package dijagram

Na package dijagramu 3.1 vidimo pakete od kojih se server sastoji. Centralno mjesto zauzima upravljački dio. U njemu se nalaze klase Server i Klijent koje predstavljaju dvije glavne komponente sustava. U paketu Svijet nalaze se klase koje modeliraju svijet; više o njima kasnije. I/O paket sadrži barem jednu klasu: Baza, koja modelira vezu prema hard disku. Ovaj paket nećemo detaljnije specificirati jer sama izvedba pohrane podataka nije do kraja zamišljena. MCP paket sadrži klase koje služe za funkcioniranje MCP podsustava. Virtualna mašina i Kompajler također ostaju nespecificirani, osim nekih kasnijih natuknica što se ugrubo od njih očekuje.

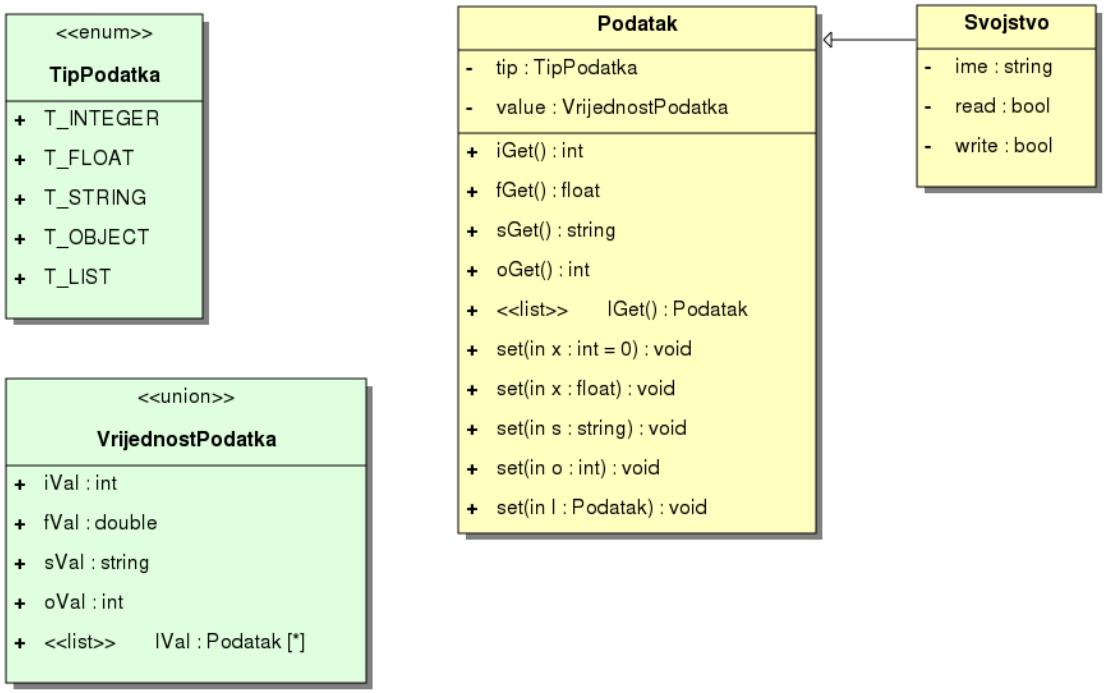
3.2 Paket Svijet



Slika 3.2: Klase u paketu Svijet

U paketu Svijet (dijagram 3.2) najvažnije mjesto zauzima hijerarhija nasljeđivanja u kojoj je bazna klasa Objekt. Praktički sve ostale klase su izvedene iz nje: Lik, čije su

specijalizacije ZiviLik i NPC, Soba, Predmet i Prolaz. Istaknuto mjesto među atributima ovih klasa ima lista svojstava. Klase koje modeliraju svojstva mogu se vidjeti na dijagramu 3.3.



Slika 3.3: Klase koje modeliraju svojstva

Ono što je bitno za sve krajnje specijalizirane klase je da implementiraju virtualnu metodu `consume_event()`. Ona, naime, određuje ponašanje objekata u slučaju da su na njima okinuti (triggerani) događaji koji su rezultat naredbi zadanih od strane igrača, ili NPC-ova.

Većina događaja ima pridruženu funkciju (spremljenu u bytecode obliku) koja se treba izvršiti ukoliko je dotični događaj iniciran. Pridruživanje je definirano mapom `event_map`. Na `consume_event()` metodi je da pošalje odgovarajući bytecode na izvršavanje virtualnoj mašini.

Neki događaji imaju predefinirano uobičajeno ponašanje koje je ugrađeno u samu metodu `consume_event()`. Npr. ako je objekt koji je instanca klase ZiviLik dobio na

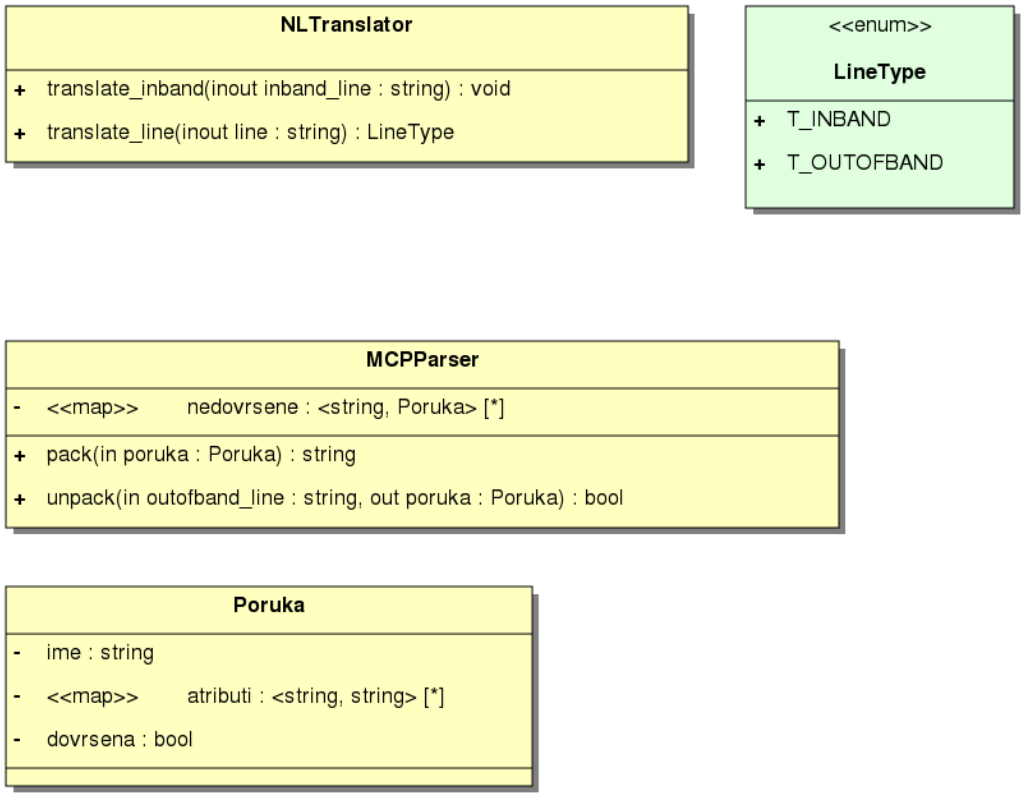
konzumaciju događaj

```
event: said who: "Radagast the Brown" what:"0 je prirodan broj."
```

onda njegova verzija metode može odmah reagirati slanjem poruke pridruženom klijentu. Stvar bi trebala biti uređena tako da za ovakve uobičajene događaje metoda prvo provjeri postoji li pridružena bytecode funkcija, a ako ne postoji, izvrši uobičajen slijed naredbi. Na taj način će administrator sustava imati mogućnost prema potrebi promijeniti uobičajeno ponašanje za uobičajene događaje poput onog kad igrač kaže nešto u sobi.

((Napomena: upotrijebljena notacija za događaj je proizvoljna, događaj je modeliran klasom Event iz paketa VM.))

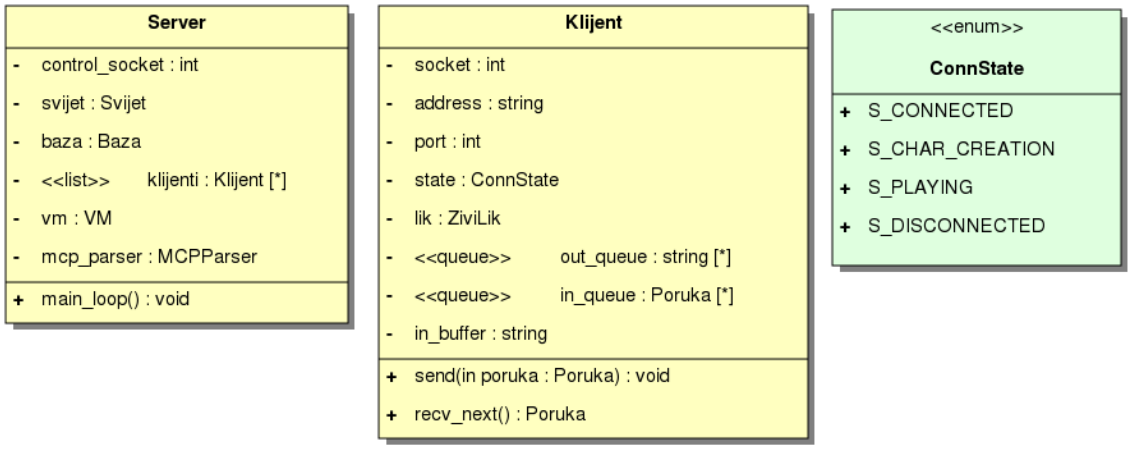
3.3 Paket MCP



Slika 3.4: Klase iz paketa MCP

U paketu MCP imamo klase NLTranslator, MCPParser i Poruka. Poruka modelira MCP poruku opisanu u odlomku 2.3. NLTranslator služi za prepoznavanje koje su mrežne linije in-band, a koje out-of-band i za obilježavanje in-band linija koje se spremaju za slanje mrežom (dodavanjem prefiksa `#$"` ako je potrebno; vidi [1]). MCPParser s jedne strane pretvara poruke u mrežne linije, a s druge strane iz mrežnih linija stvara poruke. Sjetimo se, poruke mogu biti poslane korištenjem više mrežnih linija, stoga MCPParser čuva nedovršene poruke i prosljeđuje ih preko parametra `poruka` metode `unpack()` tek kad su dovršene.

3.4 Upravljački dio



Slika 3.5: Upravljački dio

Dolazimo do glavnog, upravljačkog dijela. On se sastoji od klasa Server i Klijent. Server sadrži sve potrebne resurse. Njegova metoda `main_loop()` sadrži glavnu petlju

čijim se cikličkim izvršavanjem odvija rad servera. Što ona radi najbolje se vidi na activity dijagramu 3.6.

Klijent modelira udaljenog klijenta. Sadrži sve potrebne mrežne informacije kao i metode za slanje i primanje poruka.

3.5 Virtualna mašina i kompajler

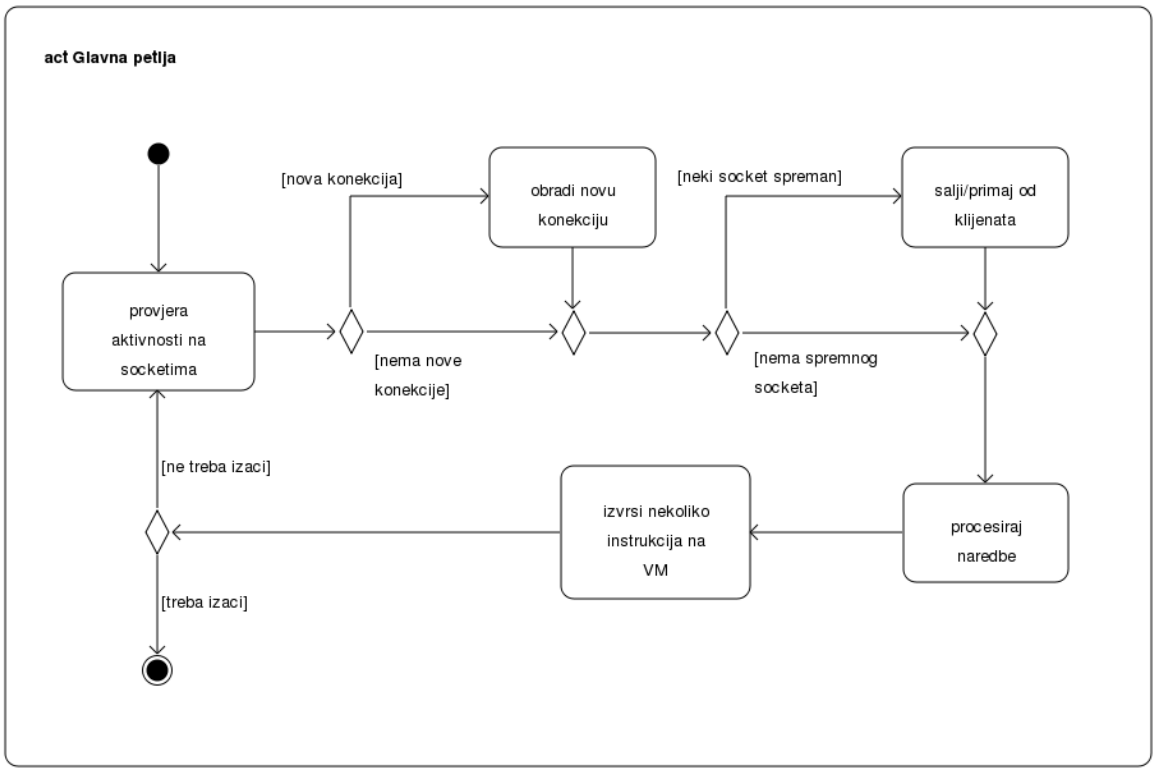
Virtualna mašina trebala bi se sastojati od stoga i nekoliko registara. Trebala bi imati neke uobičajene instrukcije poput ADD, MOV, JSR itd. Također, trebala bi imati mogućnost pozivanja "vanjskih" funkcija i pristup "vanjskim" objektima—vanjskim u odnosu na mašinu, tj. funkcijama i objektima samog sustava. Nadalje, trebala bi imati red u kojem bi stajale dretve spremne na izvršavanje. Trebala bi imati mogućnost izvršavanja po nekoliko instrukcija svake dretve kako bi sve one imale ravnopravan prioritet. (Opcionalno, moglo bi se uvesti nekoliko razina prioriteta.)

Kompajler zasad ostaje nerazrađen.

3.6 Dinamički i bihevioralni pogled

Prva bitna stvar što se tiče dinamičkog funkcioniranja servera je glavna petlja. Da bismo shvatili što se u njoj odvija, najbolje je pogledati dijagram 3.6. Objasnimo detaljnije aktivnosti koje se vide na tom dijagramu:

- Provjera aktivnosti na socketima – server prvo neko vrijeme provede čekajući nove konekcije i aktivnosti na socketima. Vidjeti POSIX funkciju `select()`.
- Ukoliko je došla nova konekcija, valja je obraditi: stvoriti novi objekt klase Klijent i inicijalizirati ga.
- Šalji/primaj od klijenata – ukoliko se POSIX funkcija `send()` ne bi blokirala na socketu nekog klijenta, pošalji koliko god možeš poruka iz njegovog izlaznog reda. Ukoliko se POSIX funkcija `recv()` ne bi blokirala na socketu nekog klijenta, primi koliko možeš u dolazni red.
- Obradom primljenih poruka od klijenata generirat će se naredbe koje je potrebno izvršiti. Izvrši ih.
- Izvedi nekoliko instrukcija na virtualnoj mašini. Naime, virtualna mašina ima red dretvi koje čekaju na izvršavanje. Svakoj dretvi posveti "nekoliko" virtualnih procesorskih ciklusa tako da ukupno vrijeme rada virtualne mašine ne bude preveliko.



Slika 3.6: Glavna petlja

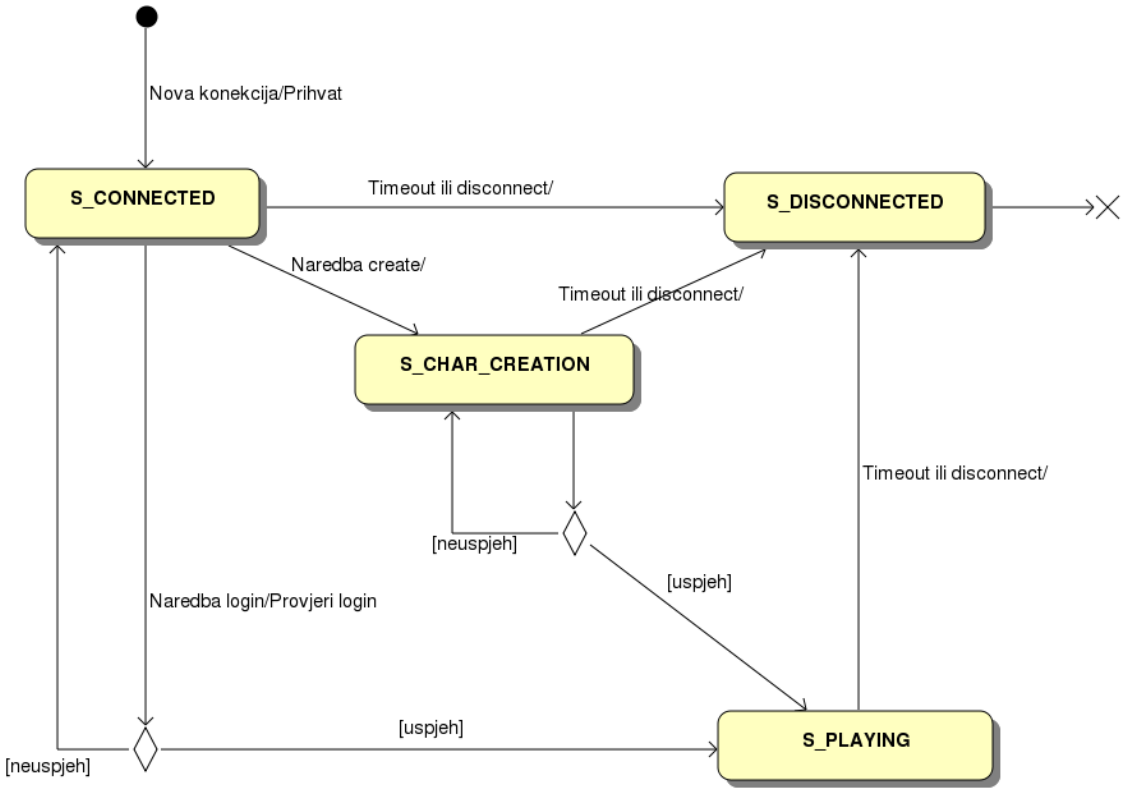
Ovo je nužno kako bi sve dretve dobile priliku za izvršavanje, a da dretva s eventualnom beskonačnom petljom ne zaokupira server unedogled.

Druga stvar je stvaranje novog lika za klijenta koji se upravo spojio. Budući da je administratoru ostavljena sloboda specificiranja vlastitih svojstava koje likovi mogu imati, mora mu biti omogućeno i da u skriptnom jeziku napiše vlastitu funkciju za stvaranje lika. Stoga u stanju kreacije lika (vidi state dijagram 3.7) dotična funkcija s odgovarajućim argumentima ide na izvršavanje na virtualnu mašinu.

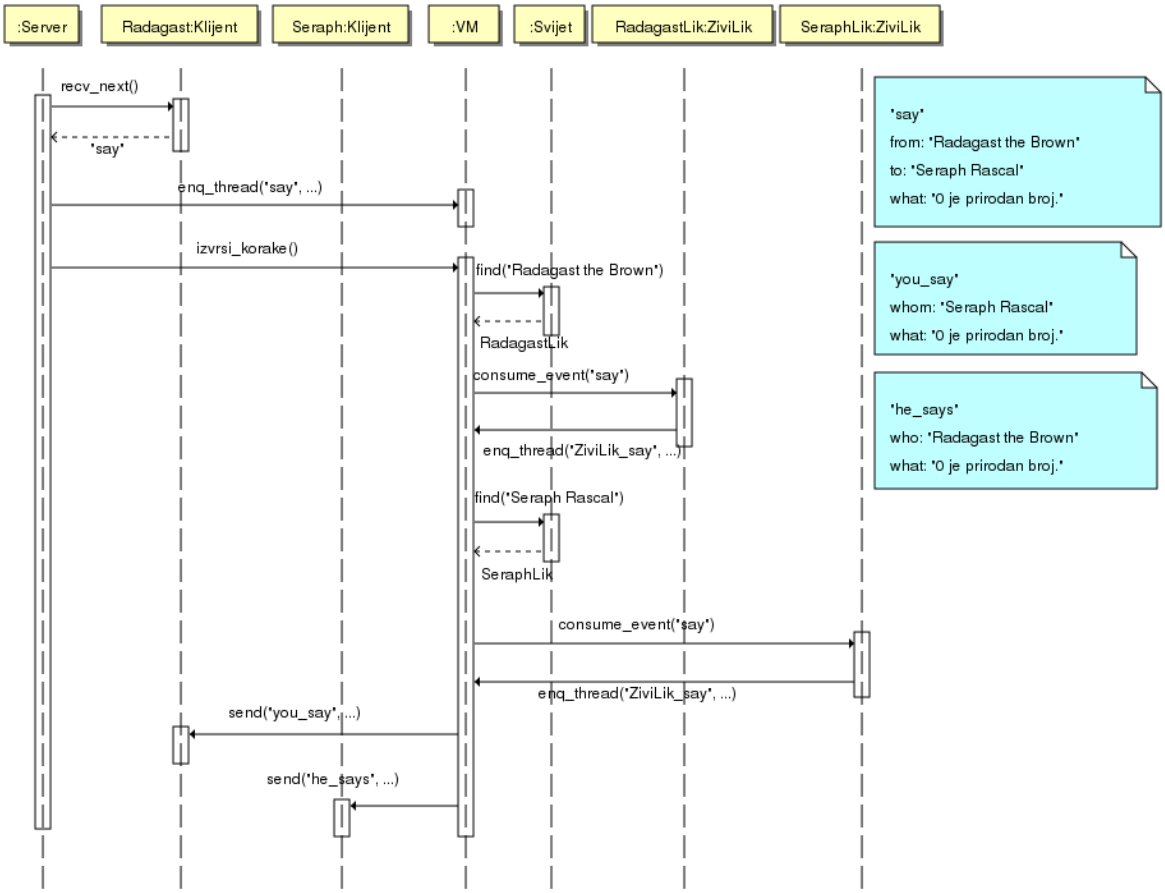
Treća bitna stvar je izvršavanje naredbi od strane igrača. Pogledajmo na zamišljenom scenariju što bi izazvala naredba

```
say from: "Radagast the Brown" to: "Seraph Rascal"
    what: "0 je prirodan broj."
```

Vidi sequence dijagram 3.8. Naime, čitav je slijed događaja pokrenut. Ovdje se lijepo vidi koliku zapravo ulogu virtualna mašina ima u svemu: enormnu.



Slika 3.7: Stanja Klijenta

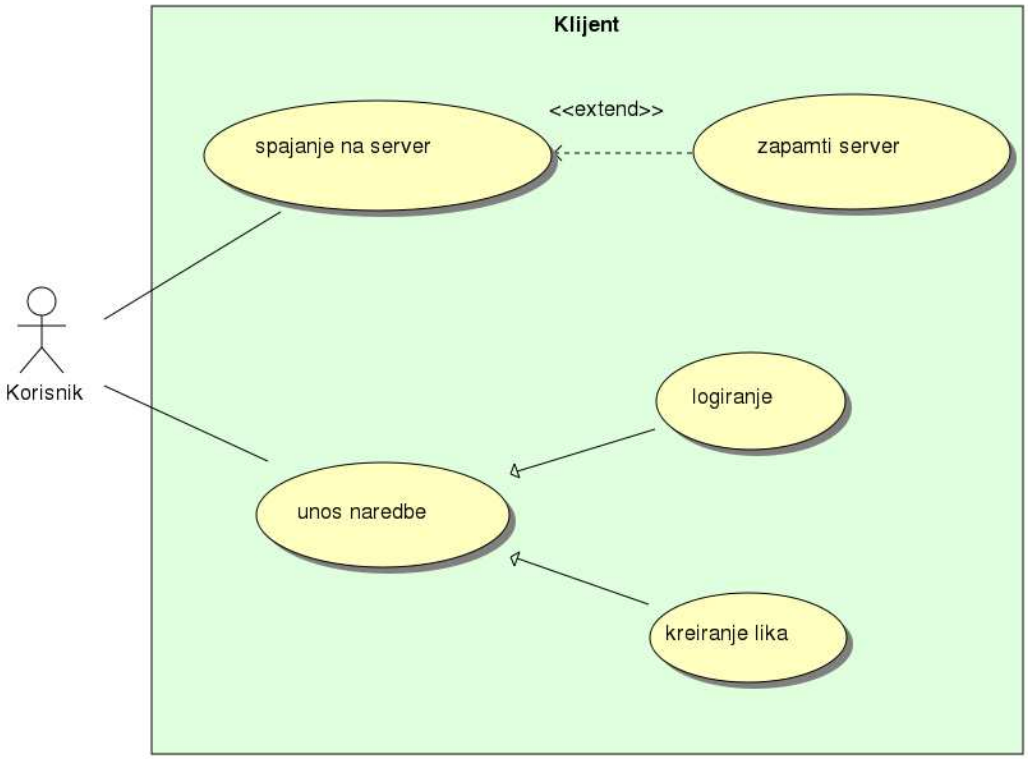


Slika 3.8: Scenarij izvršavanja naredbe

Poglavlje 4

Klijent

4.1 Use case dijagram

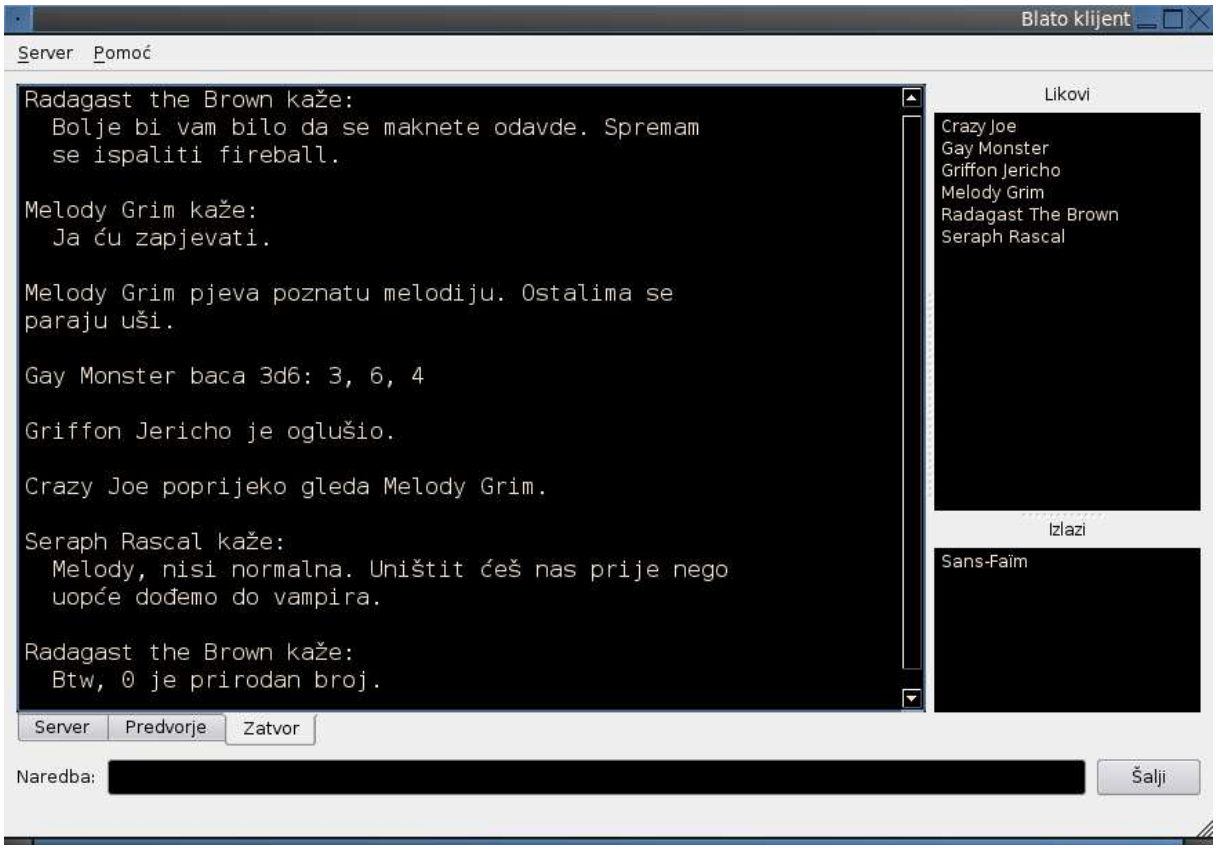


Slika 4.1: Use case dijagram

Po pokretanju klijentskog programa korisnika dočekuje sučelje kakvo se vidi na slikama 4.2 i 4.3. Biranjem stavke "Spoji se" iz menija "Server" otvara se dijalog za spajanje na server (slika 4.4). Korisnik ima mogućnost pohrane učestalo korištenih servera kako ne bi morao svaki put upisivati adresu i port za spajanje.

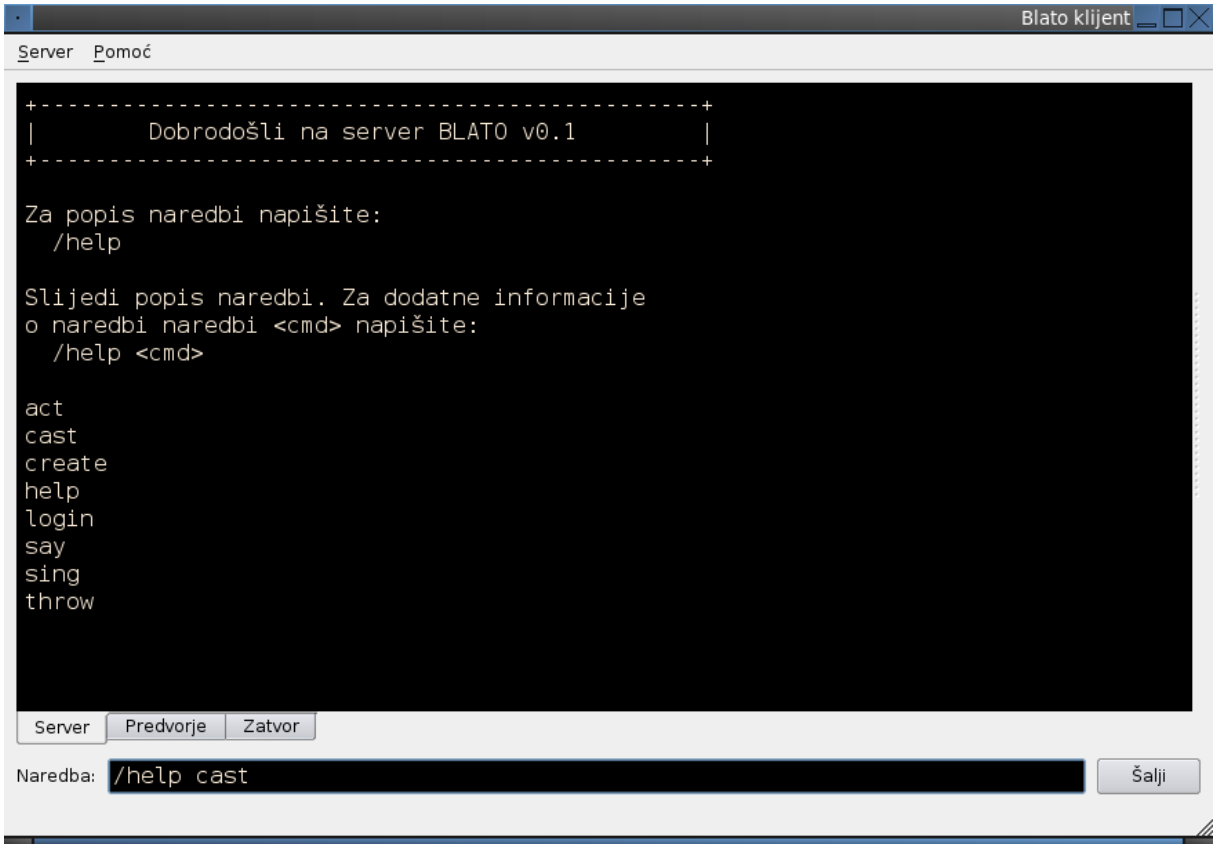
Kad se spoji na server (slika 4.3), korisnik se mora logirati kao već postojeći lik u igri, ili mora stvoriti novog lika. Na administratoru servera je da osigura naredbe za ove akcije (primjeri funkcija u skriptnom jeziku bit će isporučeni tako da može njih iskoristiti). Kad je logiran kao postojeći lik, može započeti s unosom naredbi koje definiraju njegovu interakciju sa ostatkom virtualnog svijeta.

4.2 Grafičko sučelje

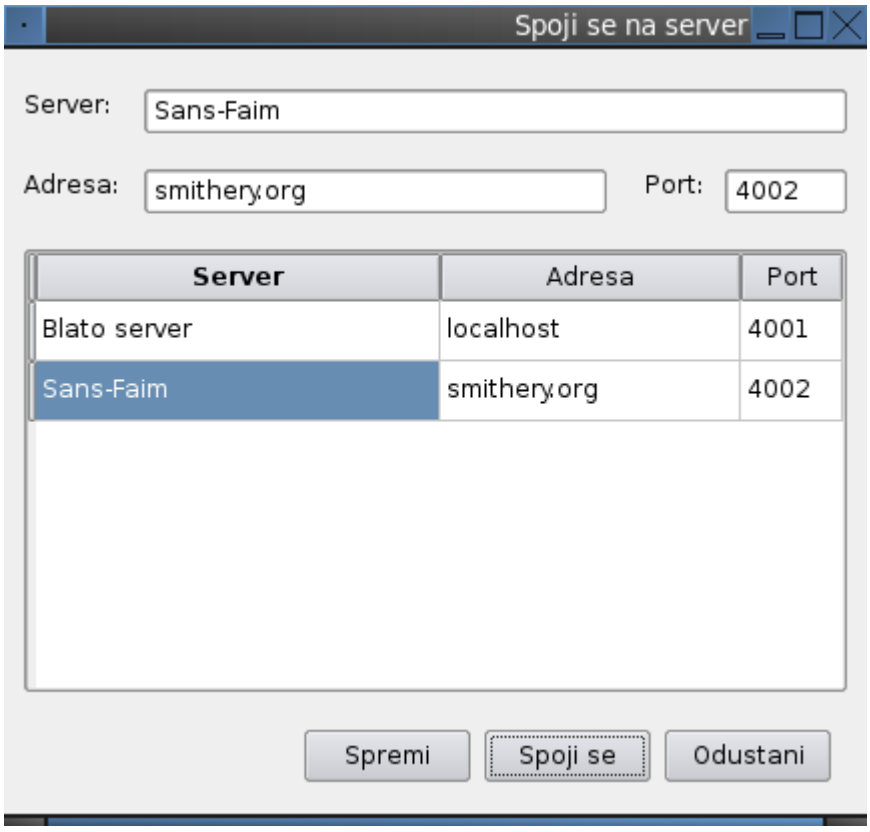


Slika 4.2: Sučelje klijenta (1)

Na glavnom prozoru dominira veliko polje na koje se ispisuju informacije koje dolaze od servera. Ukoliko je igrač logiran kao postojeći lik i nalazi se u nekoj sobi, s desne strane ima popis likova koji su s njim u sobi, kao i popis izlaza (tj. prolaza) iz sobe. Igrač naredbe unosi u za to predviđeno polje za unos.



Slika 4.3: Sučelje klijenta (2)



Slika 4.4: Dijalog za spajanje

Poglavlje 5

Rječnik

administrator Osoba koja se brine za konfiguraciju i pokretanje servera. Zadužena je za definiranje svojstava i skripti potrebnih za funkcioniranje svih objekata u igri.

argument Sastavni dio MCP poruke. Sastoji se od ključne riječi i vrijednosti. MCP poruka može imati nula, jedan ili više argumenata.

Baza Klasa koja modelira I/O vezu prema hard disku. Nije detaljnije specificirana.

`blato` MCP paket koji sadrži MCP poruke specifične za sustav.

bytecode Niz instrukcija za virtualnu mašinu.

ConnState Enumeracija, sadrži moguća stanja u kojima se objekt klase Klijent može naći, s obzirom na fazu konekcije u kojoj se klijent nalazi. Moguće vrijednosti su: S_CONNECTED, S_DISCONNECTED, S_CHAR_CREATION i S_PLAYING.

`consume_event()` Metoda klasa izvedenih iz klase Objekt zadužena za reagiranje objekta u slučaju okidanja događaja koji ga se tiču.

datatag Specijalan argument višelinijske poruke, služi za identifikaciju poruke u nastavljajućim i završnim linijama.

Event Klasa iz paketa VM koja modelira događaj. Nije detaljnije specificirana.

`event_map` Statička mapa klasa izvedenih iz klase Objekt, imenima događaja pridružuje bytecode koji se mora izvršiti na virtualnoj mašini u slučaju okidanja.

in-band linija Nestrukturirana mrežna linija.

klijent Komponenta sustava—izvršna datoteka koja se izvršava na klijentskom računalu.

Klijent Klasa koja modelira klijenta.

ključna riječ Dio MCP poruke koji služi za identifikaciju argumenta.

ključ za autentikaciju Ključ koji klijent šalje serveru kao dio `mcp` poruke, koristi se kao sigurnosna komponenta svake sljedeće poruke koju server i klijent razmijene.

kompajler Dio sustava koji prevodi skriptni jezik u bytecode. Izdvojen je u zasebnu izvršnu datoteku.

Lik Klasa koja modelira likove u svijetu. Iz nje su izvedeni ZiviLik i NPC.

LineType Enumeracija, modelira tip mrežne linije. Moguće vrijednosti su: T_INBAND i T_OUTOFBAND.

`main_loop()` Metoda klase Server, sadrži glavnu petlju koju server ciklički ponavlja tokom svog izvođenja.

MCP MUD Client Protocol, protokol koji se koristi za izgradnju MUD sustava i aplikacija.

`mcp` MCP naredba kojom počinje komunikacijski protokol između servera i klijenta.

`mcp-negotiate` MCP paket s porukama `mcp-negotiate-can` i `mcp-negotiate-end` koje služe za usuglašavanje servera i klijenta oko podržanih paketa poruka i njihovih verzija.

MCP paket Skup MCP poruka (pritom se misli na klase poruka).

MCP poruka Strukturirani entitet koji se koristi u komunikaciji servera i klijenta MCP protokolom, služi prenošenju informacija. Ovaj pojam koristi se za konkretnu MCP poruku, ali i za klasu poruka.

MCPParser Klasa koja služi za konstrukciju MCP poruka iz out-of-band linija i obrnuto.

mrežna linija Niz ASCII znakova koji se prenosi u komunikaciji MCP protokolom.

NLTranslator Klasa koja služi za razlikovanje in-band od out-of-band linija.

NPC Klasa izvedena iz klase Lik. Modelira lika kojim upravlja sustav, za razliku od klase ZiviLik koja modelira lika kojim upravlja klijent.

out-of-band linija Strukturirana mrežna linija, služi prenošenju MCP poruka.

Podatak Klasa koja modelira vrstu i vrijednost podatka koji određuju neko svojstvo. Moguće vrste podataka su int, float, string, objekt i lista podataka.

Poruka Klasa koja modelira MCP poruku.

Predmet Klasa koja modelira predmet u svijetu, izvedena iz klase Objekt.

Prolaz Klasa koja modelira prolaz između dvije sobe, izvedena iz klase Objekt.

server Komponenta sustava koja se izvršava na centralnom računalu sa stalnom vezom na Internet.

Server Klasa koja modelira server.

skriptni jezik Ugrađeni programski jezik kojim se definira ponašanje pojedinih objekata u sustavu.

Soba Klasa koja modelira sobu, tj. prostor u/na kojem se likovi mogu nalaziti.

Svijet Klasa koja modelira svijet, tj. okruženje u kojem se igra odvija.

Svojstvo Klasa koja modelira svojstvo, karakteristiku pojedinog objekta u svijetu.

virtualna mašina Dio sustava koji simulira procesor na kojem se izvršavaju bytecode instrukcije nastale prevođenjem programa pisanih u skriptnom jeziku.

VM Klasa koja modelira virtualnu mašinu.

vrijednost Sastavni dio argumenta MCP poruke.

ZiviLik Klasa izvedena iz klase Lik, modelira lika kojim upravlja klijent, tj. živi igrač.

Bibliografija

[1] The MUD Client Protocol, Version 2.1, The protocol specification, http://www.moo.mud.org/mcp/mcp2.html

[2] BOUML, http://bouml.free.fr/

[3] PyQt4, http://www.riverbankcomputing.co.uk/software/pyqt/intro

[4] Lua, programski jezik, http://www.lua.org/