UNIVERZITET U BEOGRADU
FAKULTET ORGANIZACIONIH NAUKA

ZAVRŠNI RAD

Kreiranje specijalizovanih razvojnih okruženja pomoću NetBeans platforme

Mentor:
Prof dr. Vladan Devedžić

Student:
Ivana Jovičić 115/06

Beograd, 2010. godine

Sadržaj

1. UVOD

U ovom radu opisan je razvoj specijalizovanih razvojnih okruženja za određenu namenu pomoću NetBeans platforme. Objašnjen je pojam specijalizovanih razvojnih okruženja i prednosti koje ona nude. Analizirana su rešenja i istaknute prednosti koje u razvoju specijalizovanih razvojnih okruženja nudi NetBeans platforma. Primer implementacije dat je na aplikaciji za razvoj neuronskih mreža koja je deo *framework*-a Neuroph.

Cilj je prikazati kako se NetBeans platforma može koristi za razvoj specijalizovanih razvojnih okruženja. U radu će biti prikazane mogućnosti koje NetBeans platforma nudi i na koji način olakšava razvoj specijalizovanih razvojnih okruženja.

U poglavlju „Pregled relevantnih oblasti“ opisane su oblasti od interesa. Najpre je objašnjen koncept specijalizovanih razvojnih okruženja, a zatim predstavljene karakteristike NetBeans platforme i njenih osnovnih aplikacionih interfejsa koji su značajni za razvoj specijalizovanih razvojnih okruženja.

U poglavlju „Zahtevi i analiza“ definisani su zahtevi koje je potrebno ispuniti. Dat je pregled elemenata koji čine specijalizovano razvojno okruženje i koje je potrebno implementirati. Zatim je data specifikacija ovih elemenata i analizirani mogući načini implementacije pomoću NetBeans platforme.

U poglavlju „Projektovanje“ dato je rešenje za realizaciju elemenata specijalizovanog razvojnog okruženja pomoću NetBeans platforme. Dati su dijagrami klasa i sekvenci koji prikazuju osnovne klase potrebne za kreiranje novog tipa projekta, fajlova i komponenti grafičkog korisničkog interfejsa.

U poglavlju „Implementacija“ detaljno je prikazana implementacija elemenata specijalizovanog razvojnog okruženja. Dat je prikaz ključnih delova koda klasa neophodnih za implementiraciju novog tipa projekta, novih tipova fajlova koji pripadaju projektu i komponenti grafičkog interfejsa. Prikazani su postupni rezultati na aplikaciji tokom i na kraju implementacije svakog elementa specijalizovanog razvojnog okruženja, i dat je kratak rezime potrebnih koraka.

U poglavlju „Evaluacija“ istaknute su prednosti korišćenja NetBeans Platforme za implementaciju specijalizovanih razvojnih okruženja i prednosti koje su ostvarene implementiranjem elemenata specijalizovanog razvojnog okruženja za Neuroph aplikaciju.

U poglavlju „Zaključak“ dat je pregled šta je postignuto i koji su dalji pravci razvoja aplikacije.

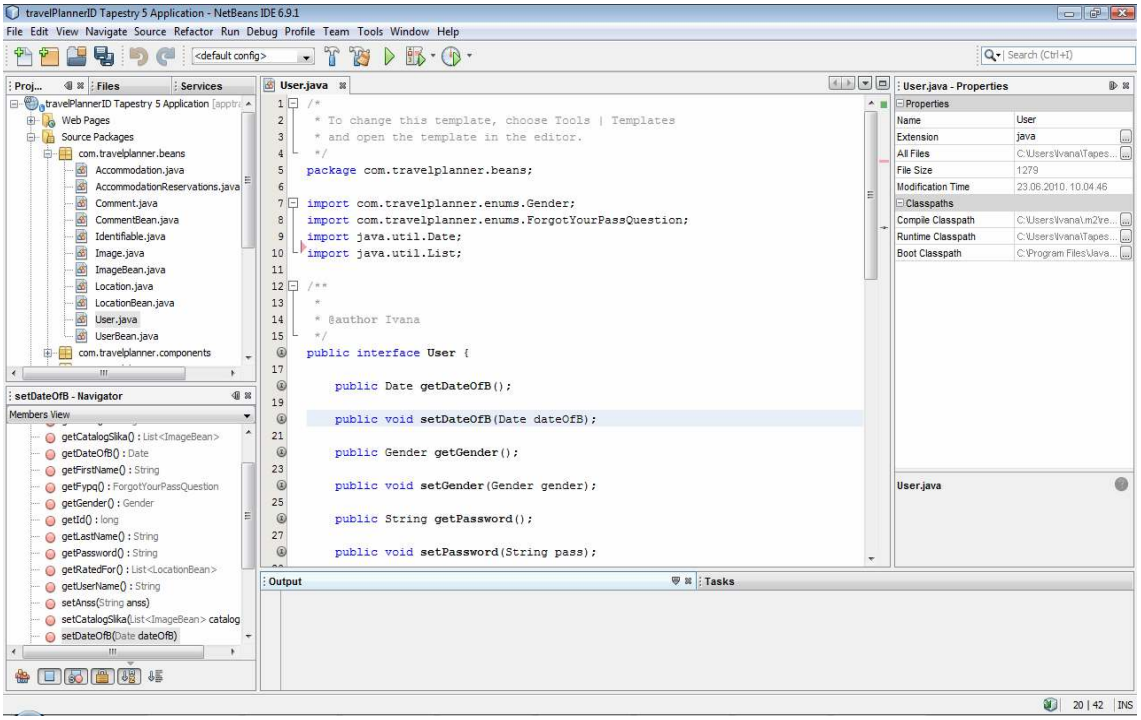
2. PREGLED RELEVANTNIH OBLASTI

2.1 Razvojna okruženja

Integrisana razvojna okruženja (*Integrated development environment – IDE*) u oblasti razvoja softvera objedinjuju razne alate koji se koriste prilikom kreiranja softvera čime omogućavaju jednostavniji rad i veću produktivnost.
Neke od tehničkih unapređenja koje obezbeđuje IDE za razvoj softvera su sledeće :

1. grafički korisnički interfejs za izvršavanje akcija i navigaciju;
2. grupisanje fajlova u koncept projekta;
3. integrisanje sa kompajlerom;
4. jednostavno praćenj erada programa (*debug*-ovanje);
5. mogućnost testiranja koda i performansi;
6. jednostavno generisanje dokumentacije projekta.

Prednosti korišćenja razvojnih okruženja su kraće vreme razvoja aplikacija, veći kvalitet i pouzdanost koda i standardizovani razvojni proces. Razvojna okruženja mogu podržavati rad sa jednim ili više programskih jezika. Jedno od tih razvojnih okruženja je NetBeans IDE.
Na slici 1. prikazan je standardni izgled NetBeans razvojnog okruženja.



Slika 1. NetBeans IDE

Specijalizovano razvojno okruženje predstavlja poseban tip razvojnih okruženja prilagođeno za rad sa specifičnim tipovima podataka, karakterističnim za određenu oblast. Zavisno od oblasti aplikacije moguće je kreirati nove tipove podataka, projekata i prilagoditi razvojno okruženje radu sa njima. Na ovaj način moguće je iskoristi prednosti koje nude postojeća razvojna okruženja za potrebe kreiranja posebne aplikacije. NetBeans platforma je jedno od mogućih rešenja za kreiranje specijalizovanih razvojnih okruženja.

2.2 Netbeans platforma

NetBeans platforma je *framework* zasnovan na *Swing* biblioteci namenjen razvoju desktop aplikacija. *Swing* biblioteka je standardna Java biblioteka koja sadrži komponente grafičkog korisničkog interfejsa. NetBeans platforma obezbeđuje osnovu za razvoj aplikacija, koju korisnik može da nadogradi i prilagodi svojim potrebama, pri tom stavljajući fokus na razvoj same domenske logike aplikacije umesto na korisnički interfejs.

U klijent-server arhitekturi pojam „*rich client*“ znači da se veći deo obrade podataka odvija upravo na strani klijenta. To su aplikacije koje su pogodne za dogradnju pomoću *plugin*-ova ili modula, što ih čini fleksibilnim za rešavanje različitih problema i prilagođavanju potrebama korisnika. Pogodne su za distribuciju i ažuriranje preko Interneta.

Većina desktop aplikacija ima slične tehničke zahteve, kao što su:

1. konzistentni korisnički interfejs;
2. mogućnost dograđivanja aplikacije;
3. prikaz podatka;
4. konfiguracioni sistem;
5. *help* sistem;
6. distribucioni mehanizam;
7. mogućnost online ažuriranja ;
8. podrška za rad na različitim operativnim sitemima.

Ispunjavanje ovih tehničkih zahteva za svaku novu aplikaciju oduzima previše vremena u razvoju ukoliko se stalno radi iznova za svaku aplikaciju ponaosob. NetBeans platforma nudi rešenje za sve pomenute tehničke zahteve. NetBeans platforma obezbeđuje opšti osnovni okvir za razvoj desktop aplikacija, npr. meni, *toolbar*, *status bar*, vizualizacija progresa, *help* sistem i slično. Svi elementi korisničkog interfejsa su izgrađeni na *AWT/Swing* biblioteci komponenti što ih čini pogodnim za dalju obradu ili integraciju sa drugim postojećim softverom zasnovanim na *Swing* biblioteci. Takođe *rich client* platforma olakšava kreiranje sistema *wizard-a*, upravljanje sistemom podataka, editorom podataka, internacionalizaciju *help* sistema i još mnogo toga.

NetBeans platforma, kao i sve aplikacije razvijene pomoću nje, je podeljena na module, odnosno ima modularnu arhitekturu. Modul predstavlja skup funkcionalno povezanih klasa. Pored klasa, modul sadrži i interfejs pomoću kojieg drugi moduli komuniciraju sa njim bez direktne zavisnosti. Moduli su opisani pomoću manifest fajlova i podataka u XML fajlovima. Zahvaljujući ovim opisima moduli ne moraju biti eksplicitno dodati NetBeans platformi, jer je na osnovu XML fajlova poznato koji su moduli na raspolaganju kao i kakve veze postoje između modula.

Sama NetBeans platforma se sastoji od skupa osnovnih modula, neophodnih za pokretanje i kreiranje standardnog korisničkog interfejsa aplikacije. Pored ovih, na raspolaganju su i mnogi drugi aplikacioni interejsi (API) koji olakšavaju proces razvoja aplikacije.

U osnovi NetBeans platforme i modularne arhitekture nalazi se „*NetBeans runtime container*“ koji je odgovoran za samo pokretanje, učitavanje, instaliranje i deinstaliranje modula u toku izvršavanja, kao i za samo pokretanje aplikacije. Sastoji se od sledećih modula: *Bootstrap, Startup, Module System, File System, Utilities.*

Neki od značajnih modula koji čine distribuciju NetBeans platforme su:

Module System API - omogućava modularnu strukturu NetBeans platforme i dinamičko dograđivanje.

Lookup API – odgovoran je za komunikacione mehanizme između modula i njihovo dinamičko povezivanje bez jakih zavisnosti tokom izvršavanja programa.

File System API –Omogućava pristup virtuelnom fajl sistemu NetBeans platforme, koji ima ulogu centralnog registra aplikacije, pruža mogućnost kreiranja novih foldera, fajlova i atributa fajl sistema. i obezbeđuje komunikaciju između modula aplikacije pomoću XML fajlova.

DataSystem API – omogućava predstavljanje osnovnih tipova fajlova u aplikaciji.

Nodes API – odgovoran za vizuelnu reprezentaciju osnovnih elemenata.

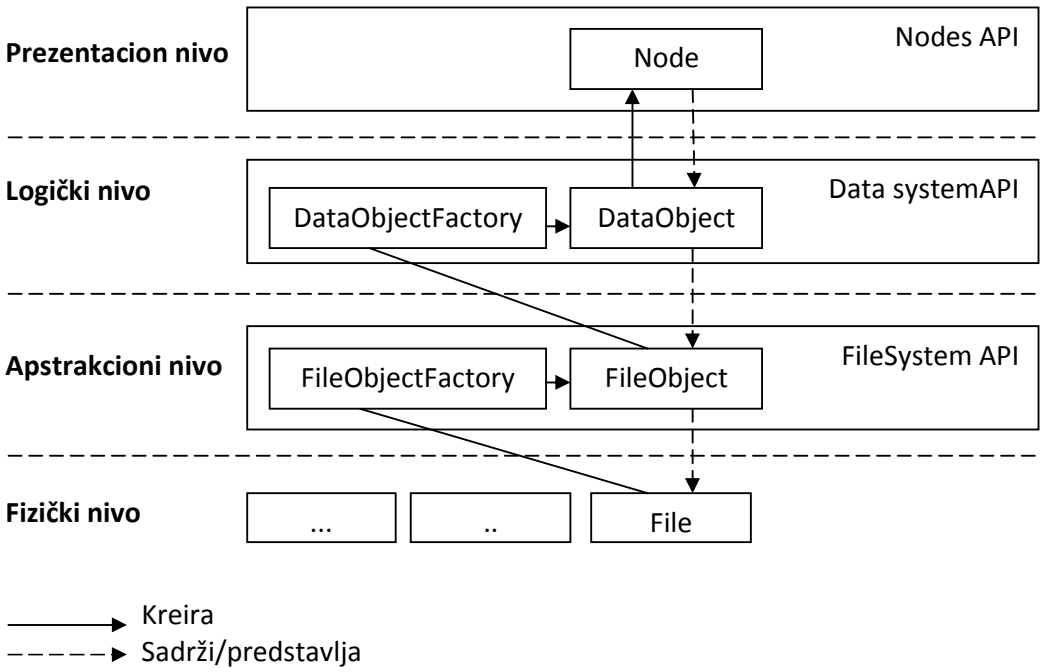
Explorer & Property Sheet API – omogućava obradu i prezentovanje nodova (osnovna komponenta Nodes API) u hijerarhiji pomoću komponenti iz *Netbeans Platform Swing* biblioteke.

Window System API - pruža mogućnost kreiranja aplikacije sa naprednim *window* komponentama kojima upravlja *windows manager*.

Actions API - omogućava dodavanje akcija elementima iz menija i *toolbar*-a.

Visual Library API – pruža mogućnost dodavanja vizuelnih alatki aplikaciji.

NetBeans platforma nudi brojna rešenja za kreriranje, upravljanje, manipulisaciju i obradu podataka. Ova rešenja su obezbeđena pomoću *File System, Data System i Node API*-a. Na slici 2 predstavljene su veze između ovih elemenata. Svaki od pomenutih *API*-va se nalazi na različitom nivou apstrakcije.



Slika 2. Arhitektura reprezentacije podataka u NetBeans Platformi

2.2.1 File system API

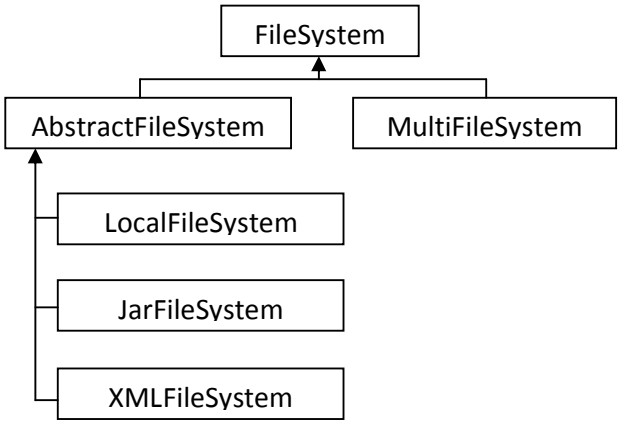
Fajl sistem kao integralni deo operativnog sistema je odgovoran za čuvanje, organizaciju i upravljanje fajlovima i datotekama. *FileSystem API* NetBeans platforme pruža slične mogućnosti. Pomoću njega moguće je manipulisati modulima aplikacije slično kao što se manipuliše aplikacijama na nivou operativnog sistema, tj neki modul aplikacije moguće je instalirati, deinstalirati, ukloniti ili obrisati kao što je to moguće raditi sa aplikacijom na operativnom sistemu. Na ovaj način je vrlo lako dodati ili ukloniti neku od alatki aplikacije, i time upravljati njenim funkcionalnostima.

FileSystem API je jedan od osnovnih modula NetBeans platforme, deo *runtime container*-a, tako da je uključen u svaku aplikaciju NetBeans platforme. Na slici 3 dat je osnovni dijagram klasa *FileSystem API*-a.

Opšte karakteristike fajl sistema specificirane su u apstraktnoj klasi *FileSystem*.

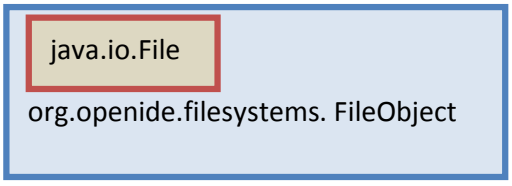
Apstraktna klasa *AbstractFileSystem* predstavlja superklasu za specifične implementacije (LocalFileSystem, JarFileSystem i XmlFileSystem).

Klasa *MultiFileSystem* dalje obezebeđuje *proksi* za složene fajl sisteme pri čemu se koristi kao superkalasa.



Slika 3. Hijerarhija klasa u FileSystem API-u

Kombinacija klasa *MultiFileSystem* i *XMLFileSystem* čini *System FileSystem*, koji predstavalja opšti registar konfiguracija.
Lookup predstavlja mehanizam za pronalaženje instance objekta. Prosleđivanjem *Class* objekta pomoću *Lookup*-a može se dobiti instanca te klase ili *null* ako nije do sada instancirana. Postoji globalni *Lookup* koji se koristi za pronalaženje objekata koji su registrovani u sistemu i obično su implementirani kao Singltoni. Takođe, neki objekti imaju metodu *getLookup*() koja omogućava drugim delovima koda da dođu do nekih specifičnih ponašanja tih objekata,npr. *Nodes* i *Project* objekti. Standardni *Lookup* je koristan kod jednostavnije registracije objekata u okviru sistema. Međutim ponekad je potrebno povezati dodatne podatke sa objektom. *System FileSystem* omogućava napredniji mehanizam za registraciju bilo kojih tipova podataka. Svaki modul koji bi trebalo da doda neki fajl ili objekat fajl sistemu treba da sadrži *layer.xml* fajl. Modul može ili ne mora da sadrži ovaj fajl, ali ne može da ima više od jednog. Ukoliko se moduli kreiraju pomoću *modul project template-a* u NetBeans IDE-u, *layer.xml* fajl se kreira automatski. U vreme pokretanja aplikacije sistem spaja sve layer fajlove modula i kreira integralni fajl sistem od manjih *XMLFfilesystem* fajlova. Pored za dodavanje elemenata u fajl sistem ovaj metod se koristi i za uklanjanje elemenata iz fajl sistema.



Slika 4. Predstavljanje fajlova pomoću FileObject klase

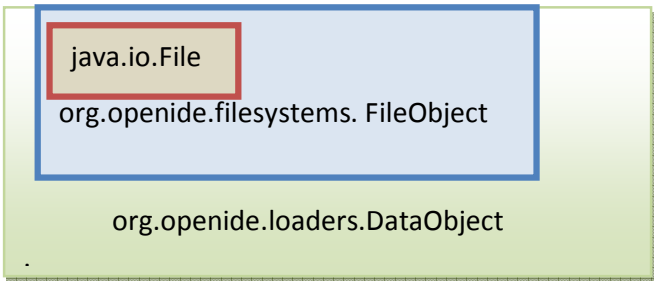
Podaci unutar fajl sistema(fajlovi, folderi) su predstavljeni pomoću klase *FileObject*. Ona predstavlja omotač za standardnu *java.io.File* klasu, što je prikazano na slici 4. Neke od osobina *FileObject*-a su sledeće:

1. Omogućava manipulaciju osnovnim operacijama nad fajlovima: pronalaženje, kreiranje, brisanje, preimenovanje fajlova ili foldera(`folder.createFolder("sub"), folder.createData("NewSource", "java")).`

2. Pruža mogućnost praćenja promena na folderima i fajlovima korišćenjem standardnih Java događaja (*FileChangeListener*).
3. Pomoću `FileObject.getMIMEType()` može se lako odrediti osnovni MIME tip fajla, koji se može koristiti za klasifikovanje.
4. Imaju atribute kojima je moguće dodeliti vrednosti, npr. putanja ikone za reprezentaciju fajla, imaju sopstvene input i output stream-ove koji se pozivaju metodama FileObject-a. Ukoliko je potreno kreirati FileObject od fajla na disku procedura je vrlo jednostavna:
   ```
   fileObject=FileUtil.toFileObject(new File("/neka/putanja/file.txt"));
   ```
   FileUtil takođe ima metode za rad sa ZIP ili JAR unosima.
5. Moguće je kreiranje sopstvenog fajl sistema za specijalizovane svrhe(implementiranjem AbstractFileSystem klase).

2.2.2 Data Systems API

DataSystem API obezbeđuje logički nivo koji se u smislu apstrakcije pristupa nalazi iznad *FileSystem API*-a. *DataObject* predstavlja omotač oko klase *FileObject*. Na slici 5 prikazana je struktura klasa *FileObject*, *DataObject* i *File* za predstavljnje fajlova. *DataObject* klasa omogućava pristup specifičnom tipu fajlova, za razliku od *FileObjecta* koji nije svestan tipova fajlova koje predstavlja. Redefinisanjem akcija odgovornih za editovanje i manipulaciju koje se vezuju za jedan *DataObject* moguće je kreirati funkcionalne tipove fajlova koji su specijalizovani za određeno okruženje.



Slika 5. Predstavljanje fajlova pomoću DataObject klase

Za kreiranje strukture fajlova u fajl sistemu NetBeans-a odgovorni su *DataLoaderPool*, *DataLoader* i *DataObject*.

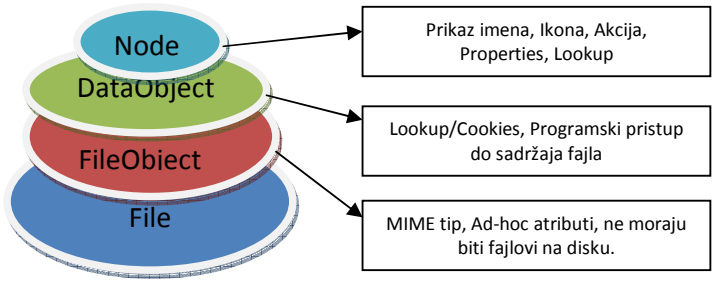
- **DataLoaderPool** je odgovoran za skeniranje fajlova u direktorijumima na disku, on odlučuje da li fajlovi relevantni ili ne, zatim ih dalje grupiše u relevantne celine i odlučuje kom tipu podataka pripada određeni fajl.
- **DataLoader** prilikom skeniranja svakom registrovanom „data loader-u“ postavlja pitanje da li bi dati podaci trebalo da se dalje obrađuju. Prvi *loader* koji prepozna fajl preuzima na sebe kreiranje odgovarajuće instance klase *DataObject*.
- **DataObject** predstavlja fajl u NetBeans-u.

Prednosti korišćenja ovog principa su sledeće:

1. moguće je povezivanje grupe fajlova i predstavljanje kao jednog data object-a;

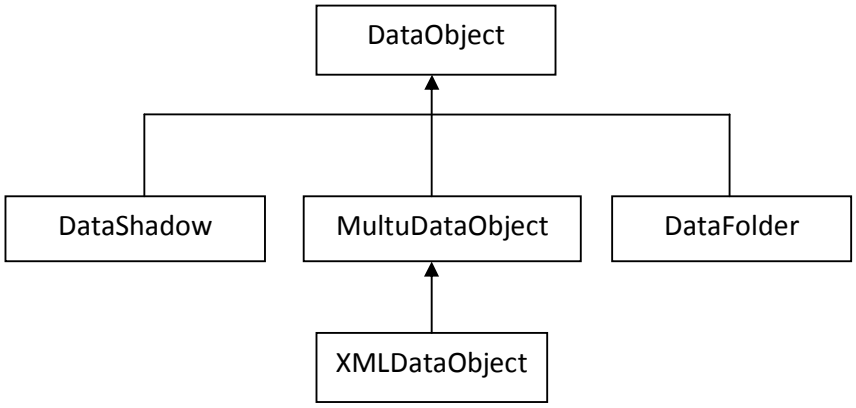
2. nerelevantni fajlovi kao što su *backup* fajlovi, test output se ne prikazuju na fajl sistemu pošto ih ni jedan loader ne prihvata. U *Explorer view*-u se prikazuju samo fajlovi relevantni za korisnika;
3. *DataObject* objekti sadrže brojna ponašanja koje obični fajl objekti nemaju, npr. mogu sadržati *cookies*-e koji obezbeđuju ponašanja za editovanje, mogu biti specificirani kao *template*-i, mogu proširiti standardne procedure izmene;
4. posebni pseudo fajlovi, kao što su *DataShadows*, se interpretiraju pomoću *DataSystem*-a, a ne *FileSystem*-a.

Data System API obezbeđuje skup superklasa koje omogućuju laku implementaciju *DataObject*-a i tipova podataka koji se koriste u aplikaciji. Neke od osnovnih klasa su DataObjectFactory*, DataLoader, MultiFileLoader, UniFileLoader, DataObject, MultiDataObject, MultiDataObjectEntry, FileEntry*.
DataObject je zadužen za kreiranje *Node* objekta, tj. reprezentacije podataka u okviru korisničkog interfejsa. Takođe, *DataObject* omogućava programski pristup do samog sadržaja fajlova. Na slici 6 Prikazane su uloge klasa *DataObject*, *FileObject* i Node.
DataLoader je odgovran za čitanje fajlova određenog MIME tipa, on obično obezbeđuje i omogućava sistemu da identifikuje fajlove sa određenom ekstenzijom ili sadržajem kao fajlove traženog tipa. Svaki individualni fajl se predstavlja pomoću klase *DataObject*.



Slika 6. Uloge FileObjecta, DataObjecta i Noda

DataObject je deklarisan pomoću apstraktne klase DataObject, međutim u praksi se kao superklasa najčešće koristi **MultiDataObject** klasa. Na slici 7 prikazana je hijerarhija klasa u *DataSystem API*-u. Ona implementira većinu metoda *DataObject* klase. Prednost *MultiDataObject* klase se ogleda u tome što za razliku od *DataObject* klase koja sadrži jedan primarni fajl(FileObject), može da vodi evidenciju o više FileObject-a, tj. sekundarnih fajlova. Ovi fajlovi najčešće predstavljaju neke zavisne fajllove, npr. kod *Form Editora*, jedan data objekat predstavlja .java, .form, i . class fajl, gde je java fajl primarni, a ostali su sekundarni. Po ovom principu učitavanja fajlova funkcioniše ***multi-file loader***. Ideja je da *loader pool* pri pregledanju fajlova, bez obzira da li prvo nailazi na primarni ili sekundarni fajl, prepozna da su iz iste grupacije, pronađe primarni fajl ukoliko postoji sekundarni i na kraju kreira *MultiDataObject*. Svaki od fajlova grupacije biće predstavljeni kao zasebni *entry*-i istog multi data objekta. Single-file loader predstavlja posebni tip *MultiFileLoader*-a, kada se prepoznaje samo jedan fajl. Ovaj princip je jednostavniji i češće se koristi pomoću *UniFileLoader-a*, npr. kod HTML fajlova.

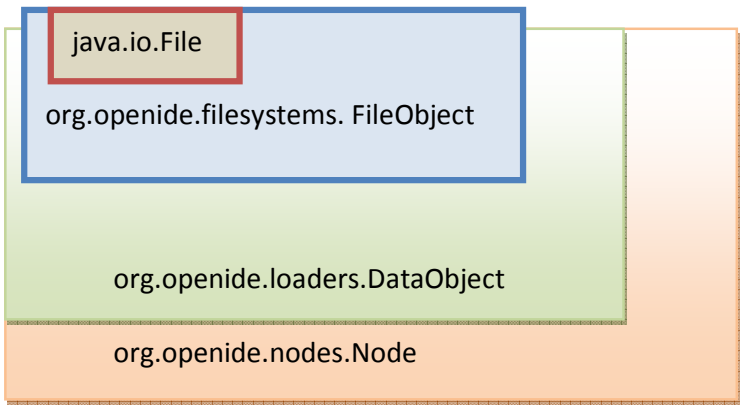


Slika 7. Hijerarhija klasa u DataSystem API-u

Kreiranje samog *DataObjecta* vrši se pomoću *DataObject.Factory-a*. U konstruktoru se prosleđuju primarni fajl i neka podklasa *DataLoader-a* (npr. MultiFileLoader). Pošto *DataObject* zna svoj tip podatak, takođe je zadužen i za kreiranje *Node-a* koji se koristi za njegovo predstavljanje u korisničkom interfejsu. Takođe, zahvaljujući poznavanju svog sadržaja *DataObject* obezbeđuje i specifične funkcionalnosti i *properties*-e.

2.2.3 Node System API

Nodes API je treći i najviši nivo u sistemu *NetBeans Resource Management*-a. *Resource Management* sistem je dogovoran za upravljanje fajlovima u memoriji i njihovom predstavljanju kao entiteta u korisničkom interfejsu. Nodovi su odgovorni za vizuelnu reprezentaciju i odgovarajuće ponašanje većine objekata u NetBeans-u. Mogu se koristiti za predstavljanje data objekata iz *DataSystems API-a* (tada obično služe kao omotač oko data objekta da bi obezebedili funkcionalniji korisnički interfejs), ili se mogu koristiti samostalno za posebne svrhe (svaka komponenta od alata u grafičkom korisničkom interfejsu predstavlja se pomoću nodova). Nodovi ne bi trebalo da se koriste za čuvanje podataka, već samo za njihovo prezentovanje. Podaci se obično čuvaju u data objektima ili pomoću drugih mehanizama za skladištenje. *Nodes API* kontroliše upotrebu i kreiranje nodova, omogućava akcije i kontroliše predstavljanje podataka u *Explorer* prozoru.



Slika 8. Predstavljanje fajlova pomoću Node klase

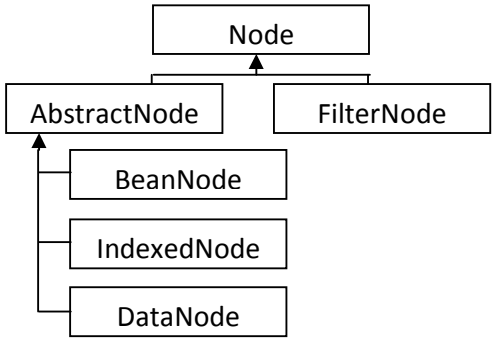
Nodovi predstavljaju proširenje koncepta *JavaBeans*-a, dodavanjem funkcionalnosti koje su značajne za *explorer* prikaz. *Explorer* prikaz je komponenta korisničkog interfejsa pomoću koga se predstavlja struktura projekta i sadržaj foldera. Korišćenjem *Explorer API* moguće je kreirati različite prezentacije hijerarhije nodova. Važna osobina nodova je da su oni „žive" komponente, tj. bilo kakve promene na njima će inicitati i promeni prikaza u *Explorer*-u. Nodovi najčešće imaju sledeće osobine:

- *naziv* koji je razumljiv korisniku;
- *opis* koji se može prikazati u *tooltip*-u;
- *ikonu* koja se prikazuje;
- *akcije* - koje se pojavljuju u popup meniju;
- *decu (children) ili podnodove* – svaki node ima org.openide.nodes.Children objekat koji može da krerira još *child* nodova, *properties* koji mogu biti modifikovani od strane korisnika pmoću Property Sheet-a, UI komponente koja je deo *Explorer API*-a i koja prikazuje pojedinost objekata.

Najčešće korišćeni tipovi nodova su sledeći:

1. Data nodovi, zasnovani na data objektima. Jednostavni primer je reprezentacija HTML fajla na disku, moguće je njegovo editovanje, razgledanje, i premeštanje. Kompleksniji primer je java klasa koja predstavlja formu. Takav nod bi imao jedno granjanje koje bi predstavljalo klase, metode i polja i još jedno koje bi predstaljalo hijerarhiju AWT/Swing komponenti(frame-ovi, paneli, dugmad, itd);
2. Data folder nodovi, predstavljaju paralelu java paketima;
3. Komponente palete – lista tabova, i komponenta na svakom tabu su nodovi, kao i komponente akcija na toolbar-u;
4. *Breakpoint* u debagger-u je takođe nod, kao i folder koji sadrži informacije o svim *brakpoint*-ovima;
5. Nod projekta sadrži različite fajlove koje su specifični za dati projekat, njegove akcije i ponašanja.

Osnovna ponašanja nodova su definisana Node superklasom. Podklase klase Node su prikazane na slici 5.



Slika 5. Hijerarhija Node podklasa

AbstractNode klasa obezebđuje najjednostavnije osobine *Node* klase, koristi se za instanciranje noda bez potrebe za implementiranjem i proširivanjem *Node* klase.
FilterNode kreira proxy nod koji delegira pozive metoda do originalnog noda. Ova vrsta nodova se koristi kada je potrebno prikazati podatke na više različitih načina.

BeanNode omogućava kreiranje noda od java objekta. BeanNode će pronaći sve JavaBean properties-e objekta i prikazati ih kao properties-e noda pomoću PropertySheet-a. ***IndexedNode*** kao što mu i ime govori pruža mogućnost uređivanja elemenata po datim indeksima.
DataNode je najčešće korišćen, kao što je već rečeno on se koristi za predstavljanje data objekata.

Svaki Node objekat ima svoj *Children* objekat, koji obezbeđuje kontejner za nodove decu, tj. podnodove. Objekat *Children* je odgovoran za kreiranje, uređivanje i strukturu nodova dece. Ovaj objekat se prosleđuje u konstruktoru AbstractNode klase. U najvećem broju slučajeva koristi se Children.Keys, međutim ako ne želimo da nod koji kreiramo ima dalje račvanje i podnodove, u konstruktoru se prosleđuje Children.LEAF. Pored njih na raspolaganju su i druge klase: Children.Array- super klasa za sve Children klase, Children.Map<T>- nodovi se sortiraju u mapu, Children.SortedArray – proširuje Children.Array sa Comparator-om, Children.SortedMap<T> proširuje Children.Map<T>sa Comparator-om.

Nodovi omogućavaju dodavanje akcija koje su osetljive na trenutno stanje. DataNode predstavlja akcije DataObjecta koji reprezentuje. DataLoader definiše MIME-specifičan folder u *layer* fajlu sa metodom *actionsContext*() gde su akcije registrovane. Kod nodova koji ne predstavljaju DataObjekte pomoću metode *getActions*() u Node klasi definišu se akcije konekst menija. Sve promene na nodovima mogu se pratiti pomoću *PropertyChangeListener*-a, kao i *NodeListener*-a. Pomoću *PropertyChangeListener*-a mogu se paratiti promene Node properties-a korišćenjem *getPropertySet*() metode. *NodeListener* se koristi za informisanje o promenama imena noda, ili izmenama na nodu roditelju ili detetu.

2.2.4 Explorer API

Explorer view predstavlja komponentu korisničkog interfejsa koja služi za prikaz nodova. Na NetBeans aplikacijama to je najčešće *Project* ili *File* sekcija sa leve strane glavnog prozora. Uloga Explorer view-a je da obezbedi grafičke komponente za predstavljanje nodova u hijerarhijskom prikazu, kroz *combo box*, menije itd. Takođe, ove komponente su odgovorne za *popup* menije nodova. *Explorer view* nema svest o tome šta reprezentuje, već samo obazbeđuje fizički korisnički interfejs pomoću *Swing* panela.

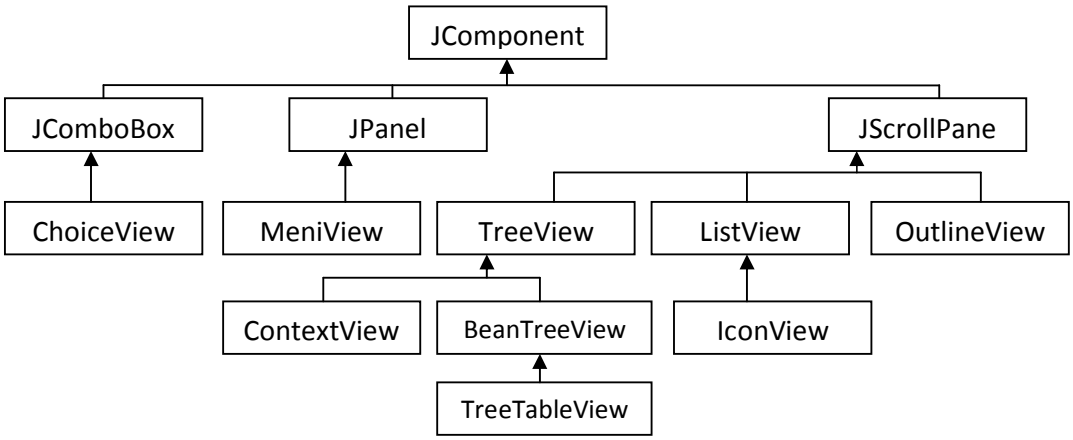
Najjednostavniji način za prikazivanje *Explorer* prozora je pozivom `NodeOperation.explore(...)` metode. Ona će u novom prozoru prikazati standardnu razgranatu drvoliku strukturu nodova. Za sva prilagođavanja prikaza neophodno je poznavanje osnovnih koncepata Explorer API-a:

1. *Topmost container- AWT/Swing* kontejner komponenta mora implementirati ExplorerManager.Provider. U NetBeans-u je to obično podklasa *TopComponent-a*. Ovaj kontejner omogućava deci, podklasama, pronalaženje *ExplorerManager-a*. *ExplorerManager* je zadužen za kontrolu prikaza i selekcija, obezebeđuje interfejs za selekciju, kao i sinhronizovanje različitih prikaza;
2. *Explorer* prikaz se dodaje glavnoj komponenti za prikaz kao dete. Nije potrebno neko posebno povezivanje sa *ExplorerManager*, kao što je već spomenuto pretragom

roditetljskih komponeti koje implementiraju *ExplorerManager.Provider*, vrši se poveivanje sa *ExplorerManager*-om;
3. Pozivanjem *ExplorerManager.setRootContext*() metode postavlja se nod za prikazivanje.

U *Explorer API-u* na raspolaganju su različiti prikazi, zavisno od kojih će i struktura prikaza noda biti drugčija. Neki od prikaza su sledeći:

1. Najčešće korišćeni prikaz je *BeanTreeView* koji prikazuje nodove u razgranatoj drvolikoj strukturi.
2. *MeniView* - pomoću koga se može implementirati prikaz korišćenjem meni popup-ova, što čini prikaz manje statičnim.
3. *ListView* – prikazuje svu decu root noda kao Jlist. Može se posebno prilagoditi da prikazuje nodove u razgranatoj strukturi.
4. *ChoiceView* – prikazuje svu decu root komponente u JcomboBox-u.
5. *TreeTableView* - je kombinacija razgranatog drvolikog prikaza i tabele. Prilikom kreiranja ovog prikaza prosleđuje se niz Node.Property objekata. Ako neki nod u redu ima isti naziv kao u vraćeni Property.getName(), taj properti će biti prikazan u odgovarajućoj koloni.
6. *PropertySheetView* je UI komponenta *Explorera*. Prikazuje Propery Sheet i *properties* selektovanog noda koji omogućava izmenu istog.



Slika 6. Hijerarhija klasa različitih Explorer prikaza

Pored samog prikaza, *Explorer* je odgovoran za obezbeđivanje akcija i menija koji zavise od trenutnog konteksta (pomoću metode *getActions()*). Iako su različiti prikazi *Explorer*-a odgovorni za različite hijerarhije prikaza, neki aspekti prikaza se preslikavaju po određenim pravilima:

1. naziv i kratki opisi noda se prikazuju u pomoćnim alatkama korisničkog interfejsa;
2. za prikaz se koristi ikona definisana u nodu;
3. obično se prikazuju sva deca noda, po redosledu koji on sam određuje.
4. *Node.getActions(), Node.getContextActions(), Node.getDefaultAction(), Node.getContextMenu()* metode se koriste za izradu sistema za obradu događaja.
5. Node.getPropertySets() se koristi od strane *PropertySheet* prikaza.

6. Neki prikazi omogućavaju komande editovanje kao što su premeštanje, preimenovanje ili brisanje nodova korišćenjem metoda *Node.canRename(), Node.cloneNode(), Node.clipboardCut().*

2.2.5 Project API

Pomoću *ProjectAPI-a* moguće je definisati nove vrste projekata u okviru određenog razvojnog okruženja, tj. grupisati određene fajlove kao celinu i omogućiti jedinstvenu manipulaciju nad njima. Ovaj način tretiranja podataka olakšava rad korisniku. *Project window* prikazuje samo foldere i fajlove sa kojima će korisnik raditi. U *Files* prikazu je korisiniku omogućeno da vidi sve fajlove koji su sadržani u projektu. Da bi se skup podataka u nekom folderu prepoznao kao projekat potrebno je da sadrži podfolder specifičnog naziva pomoću koga će biti identifikovan, npr. kod netbeans projekata to je folder „nbproject".

Značajne klase za implementaciju novog tipa projekta su:

1. ***ProjectFactory*** zadužen je za kreiranje projekta. U ovoj klasi definiše se naziv foldera za prepoznavanje tipa projekta;
2. ***Project*** klasa je reprezentacija samog projekta;
3. logički prikaz projekta se obezbeđuje pomoću klase ***LogicalViewProvider***;
4. ***ProjectInformation*** obezbeđuje dodatne informacije o projektu;
5. ***ActionProvider*** obezbeđuje jednu ili više akcija za projekat;
6. pomoću ***CopyOperationImplementation*** ili ***DeleteOperationImplementation*** klasa moguće je jednostavno implementiranja osnovnih operacija editovanja nad projektom.

Korišćenjem dva *wizard-a* u NetBeans IDE-u, za kreiranje novog modul projekta i kreiranje novog template wizarda, – moguće je jednostavno kreiranje modula koji će sadržati uzorni primerak tehnologije kojom se bavi aplikacija na kojoj se radi. Ovaj uzorni primerak podataka može služiti kao šablon za projekt aplikacije koji se kreira pomoću *new project wizard-a*. Uslov je da se novi tip projekta najpre registruje i kreiraju sve neophodne klase za njegovo prepoznavanje. Zatim je moguće napraviti folder sa željenim skupom podfoldera ili podataka koje je potrebno da svaki projekat datog tipa ima. Obavezno je napraviti i folder sa nazivom tipa projekta po kome se prepoznaje. U sledećem koraku, pomoću *template wizarda* može se izabrati kreirani projekat kao uzorak aplikacije koji će biti generisan pri kreiranju projekta pomoću *new project wizarda.*

3. ZAHTEVI I ANALIZA

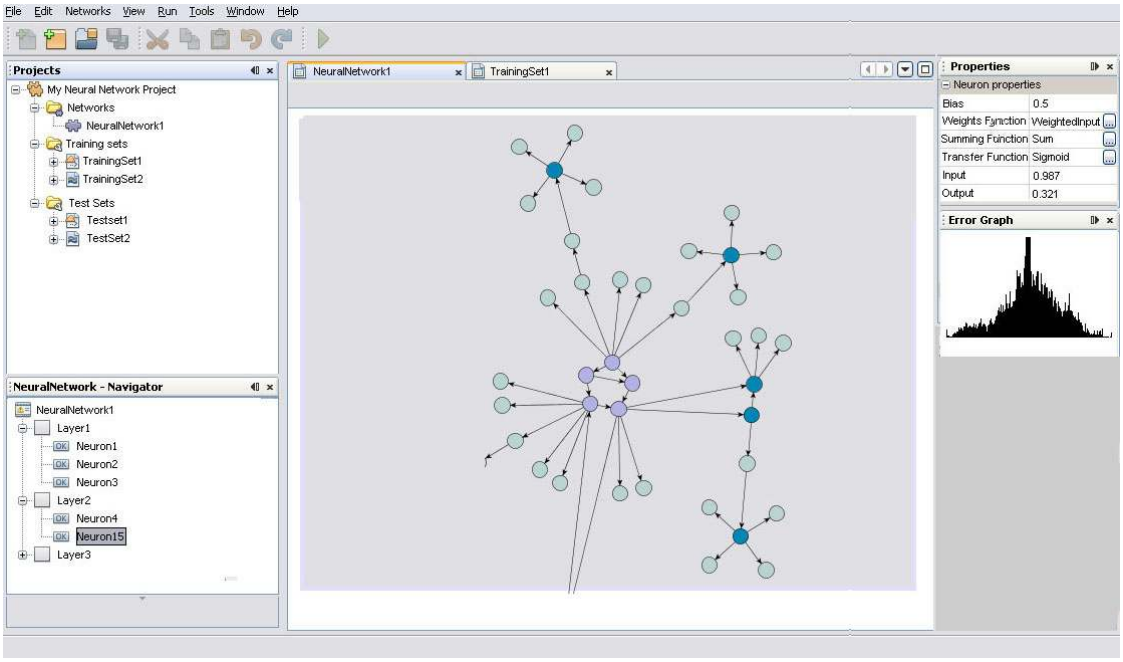
3.1 Definisanje zahteva

Za aplikaciju za rad sa neuronskim mrežama Neuroph potrebno je kreirati specijalizovano razvojno okruženje koje će da podrži odgovarajuće tipove projekata i projektnih fajlova, i obezbediti elemente korisničkog interfejsa za rad sa ovim projektima. Novim tipovima podataka potrebno je pridružiti akcije izmene i manipulacije koje su podržane za standradne tipove podataka i dodati specifične akcije vezane za neuronske mreže i skupove podataka za trening i testiranje. Potrebno je obezbediti mogućnost kreiranja fajlova i projekata pomoću *new project wizard-a* i osnovne šablone po kojima će projekti biti kreirani.

Aplikacija treba da ima *projects view*, *property view* i *navigator* elemente korisničkog interfejsa, tipične za razvojna okruženja..

- ***Projects view*** treba da omogući pregled aktivnog projekta sa hijerarhijskim prikazom sadržaja. Projekat se sastoji od foldera „Neural Networks“, „Training sets“ i „Test sets“;
- ***Property view*** bi trebalo da prikazuje parametre fajlova koji su značajni za rad korisnika;
- ***Navigator*** treba da prikaze strukturu fajlova. Kod neuronskih mreža potrebno je da se prikažu slojevi i neuroni u svakom sloju.

Na slici 7. data je početna idejna skica korisničkog interfejsa aplikacije.



Slika 7. Izgled aplikacije koji je potrebno postići

NetBeans platforma nudi gotove elemente korisničkog interfejsa (*Project view*, *Properties view* i *Wizard*-e) samo ih je potrebno prilagoditi za rad sa novim tipovima podataka. Ovi elementi su deo NetBeans razvojnog okruženja i imaju implementirane osnovne akcije za manipulaciju prozorima kao što su otvaranje, zatvaranje, premeštanje, minimiziranje ili maksimiziranje. Korišćenje gotovih komponenti značajno skraćuje i olakšava proces razvoja softvera, a rezultujuće komponente korisničkog interfejsa su pouzdane i funkcionalne.

Specijaliozovano razvojno okruženja podrazumeva podršku za kreiranje projekata specijalizovanog tipa, kreiranje specijalizovanih tipova fajlova koje projekat sadrži, obezbeđivanje akcija za izmenu i manipulaciju fajlovima i projektima, kao i prilagođavanje odgovarajućih alatki i grafičkih komponenti razvojnog okruženja novim tipovima fajlova. NetBeans platforma znatno olakšava proces razvoja specijalizovanih razvojnih okruženja pomoću gotovih rešenja koje nudi. Cilj je kreiranje specijalizovanog razvojnog okruženja Neuroph koja je deo *framework*-a za kreiranje i rad sa neuronskim mrežama.

Osnovni elementi koji čine razvojno okruženje zasnovano na NetBeans Platformi su :

Project Template	Obezbeđuje jednostavnu proceduru za kreiranje specifičnih tipova projekata, kao i standardnu strukturu projekta .
File Template	Podrška za kreiranje specijalizovanih fajlova koji pripadaju projektu.
Editor	Aplikacije zasnovane na NetBeans platformi mogu lako da obrade i prilagode NetBeans editor za rad sa specijalizovanim tipovima podataka. Može se koristiti nezavisno od NetBeans platforme.
Wizardi	NetBeans platforma obezebeđuje podršku za kreiranje i proširivanje *wizard-a* za kreiranje fajlova ili projekata. Zahvaljujući *wizard-ima* korisnikov rad sa kompleksnijim procedurama može biti znatno olakšan i skraćen.
Akcije	Akcije vezane za specifične tipove fajlova koje su osetljive na trenutni kontekst.
Upravljanje korisničkim interfejsom	Prozori, meniji, toolbar i druge prezentacione komponente koje su značajne i pre svega olakšavaju korisniku rad sa programom.
Menadžment podacima i prezentacijom podataka	Elementi korisničkog interfejsa za prezentovanje podataka i jednostavno editovanje od strane korisnika.

3.2 Povezivanje se elementima NetBeans Platforme

Project Template

Projekat predstavlja grupu logički povezanih fajlova. Pored čuvanja osnovih fajlova koji se tretiraju u aplikaciji, obezbeđuje i meta podatke o projektu. Kreiranjem novih tipova projekata pravi se omotač za fajlove specifičnog programa. U slučaju programa to su neuronske mreže i skupovi podataka za trening i testiranje.

Deklarisani tip projekta može imati sve osobine koje imaju standardni projekti NetBeans razvojnog okruženja. Takođe je moguće usaglasiti *windows* komponente i elemente menija sa novim tipom projekta. Podešavanjem koda u *layer.xml* fajlu za novi tip projekta omogućeno je kreiranje i nove *kategorije projekta* koja može predstavljati grupu specifičnih projekata. Time se mogu na pregledan način grupisati svi projekti aplikacije. Na primeru *Neuroph-a*, kreirana je sledeća kategorija *Neuroph* projekata:

1. **Neuroph project**, standardan *neuroph* projekat sa strukturom foldera *„Neural Networks“, „ Training Sets“* i *„Test Sets“*;
2. **ImageRecognition project** – projekat koji je prilagođen za rad sa alatkom za prepoznavanje slika pomoću neuronskh mreža. Nakon kreiranja projekta automatski se otvaraju odgovarajući alati;
3. **TextCharacter recognition project** – nakon kreiranja projekta automatski se otvaraju odgovarajući alati za prepoznavanj teksta pomoću neuronskih mreža;
4. **HandWriting recognition project** - nakon kreiranja projekta automatski se otvaraju odgovarajući alati za preoznavanje rukopisa.

Svakom tipu projekta možemo prilagoditi opis projekta, korake u iteratoru *wizard-a* za kreiranje projekta, kao i same akcije pri kreiranju.
Kreiranjem specifičnih tipova projekata korisnik može lakše da iskoristi potencijal aplikacije i unapredi svoj rad time što je manipulacija svim elementima projekata olakšana, jasno struktuirana i prilagođena konkretnim zahtevima.

Sadržaj projekta se prikazuje u *Project* prozoru. Kao štoje već rečeno, u *Project* prozoru prikazuju se samo fajlovi koji su relevantni za korisnikov rad na projektu. Isti tipovi fajlova su grupisani u zajedničkom folderu. Prikaz je hijerarhijski i može se prilagoditi promenama. U Project prozoru može istovremeno biti otvoreno više aktivnih projekata.

File template

Kreiranjem novog, specijalizovanog, tipa fajla obezbeđen je standardan način za čuvanje neuronskih mreža i skupova podataka za trening i testiranje. Sadržaj fajlova su serijalizovani objekti *NeuralNetwork* ili *TrainingSet* klasa. Moguće je registrovati novi tipa fajlova po ekstenziji ili po *XML root* elementu, ukoliko se radi sa XML fajlovima. U slučaju *Neuroph* programa fajlovi će biti registrovani po .nnet i .tset ekstenzijama. Kao i kod kreiranja projekata, i za tipove fajlova potrebno je kreirati novu kategoriju u kojoj se mogu nalaziti *wizard-i* za kreiranje različitih tipova mreža.

Pored standardnih akcija za koje je odgovoran *DataEditor*, novom tipu fajla se može dodati još specifičnih akcija. Takođe moguće je redefinisati osnovne akcije kao što su otvaranje, kopiranje, preimenovanje, čuvanje ili brisanje fajla. Za potrebe *Neuroph*-a reimplementirane su akcije otvaranja. Standardna operacija pri otvaranju fajla je otvaranje NetBeans editora i prikaz sadržaja fajla. Pošto je sadržaj fajlova koji predstavljaju neuronske mreže i podatke za trening i testiranje čine serijalizovani objekti odgovarajućih klasa, otvaranje nihovih sadržaja ne bi bilo od značajno za korisnika programa. Umesto toga, korišćenjem *Cookie* klasa potrebno je pri akciji otvaranja pozvati *Neuroph*-ov vizuelni grafički prikaz mreže.

Integracija sa grafičkim komponentama korisničkog interfejsa

Nakon registrovanja novih tipova podataka neophodno je za njih implementirati i ponašanja koja imaju standardni tipovi podataka. Kao što smo već spominjali Nodovi su odgovorni za vizuelno predstavljanje fajlova. Ako želimo da podesimo naziv, ikonu ili opis za kreirani tip fajla potrebno je da ta podešavanja izvršimo na nodu koji reprezentuje taj fajl.

Cilj je da omogućimo prikaz važnih podataka fajlova u properties prikazu. Za neke podatke implementiraćemo mogućnost izmene podataka pomoću properties prikaza kako bi korisniku olakšali manipulaciju sa fajlovima. Pomoću Navigatora potrebno je prikazati strukturu fajlova. Takođe, elementi koji su prikazani u navigatoru i čine neki fajl traba da budu sinhornizovani sa properties prikazom. Npr. za fajl Neuronske mreže, kada se u navigatoru prikazuju slojevi i neuropni mreže potrebno je za iste obezbediti prikaz osnovnih podataka u *Properties* prozoru. NetBeans platforma olakšava prilagođavanje grafičkih elemenata razvojnog okruženja konkretnom programu.

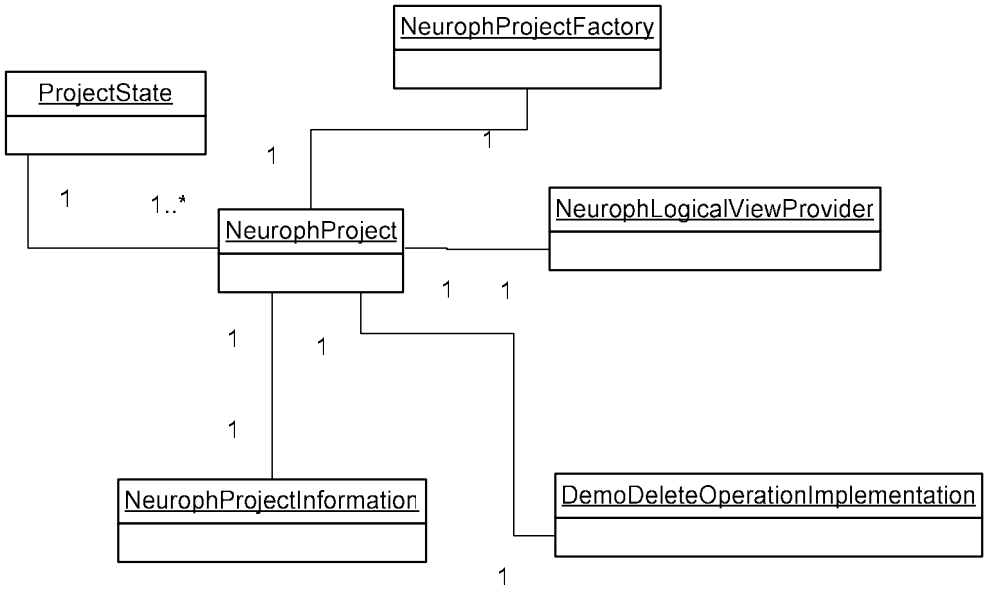
4. PROJEKTOVANJE

U ovom delu opisan je dizajn klasa elemenata koji čine specijalizovano razvojno okruženje. U Delu „Projektovanje“ dat je pregled i opis svih klasa koje su neophodne za kreiranje specijalizovanog razvojnog okruženja. Dijagramima klasa i dijagramima sekvenci prikazane su interakcije klasa koje čine osnovne elemente specijalizovanog razvojnog okruženja. NetBeans Platforme definiše osnovne interfejse ili apstraktne klase koje je potrebno implementirati za konkretne potrebe aplikacije.

4.1 Specijalizovani tip projekta

Navešćemo zahteve koje treba da ispuni određeni tip projekta, kako bismo mogli da povežemo funkcionalnosti sa klasama koje su odgovorne za njihovo izvršavanje.
Zahteve koji bi trebalo da obezbedi određeni tip projekta su sledeći:

1. integracija značajnih fajlova na jednom mestu, tj. omotač za fajlove od značaja;
2. odgovarajuća struktura specijalizovanih fajlova koji pripadaju projektu;
3. osnovne operacije editovanja projekta;
4. kreiranje odgovarajućeg logičkog prikaza fajlova;
5. kreiranje posebne kategorije projekata, koja se prikazuje u wizard-u za kreiranje novog projekta.



Slika 8. Dijagram klasa potrebnih za kreiranje novog tipa projekta

Na slici 8. Prikazane su klase od interesa potrebne za kreiranje novog tipa projekta i implementiranje osnovnih funkcionalnosti projkta tipa *Neuroph*. U daljem tekstu dat je pregled i opis značajnih klasa za.

NeurophProject

Svi projekti predstavljaju se pomoću *Project* interfejsa. Interfejs Project deklariše metodu *getLookup()* koja vraća Lookup i služi za interakciju drugih objekata sa projektom. Sve osobine i mogoćnosti projekta bi trebalo da budu izložene pomoću lookup-a. Project Infrastruktura najčešće „skriva“ originalni Project koji je ProjectFactory kreirao i izlaže samo omotač koji služi za prosleđivanje poziva ili akcija. Zbog ove osobine ne preporučuje se kastovanje Project objekta u neku implementacionu klasu. Sva ponašanja projekta su kontrolisana pomoću lookup-a. U implementaciji *getLookup()* metode ćemo kreirati Lookup i registrovati implementacije ActionProvider-a, ProjectLogicalView-a, DemoDeleteOperation-a, DemoCopyOperation, ProjectInformationImplementation-a kao i stanje prjekta tj. ProjectState. U Project interfejsu deklarisana je još i metoda *getProjectDirectory()* koja vraća direktorijum sa meta podacima i samim sadržajem. Klasa NeurophProject implementira interfejs Project za projekte tipa *Neuroph*.

NeurophProjectInformation

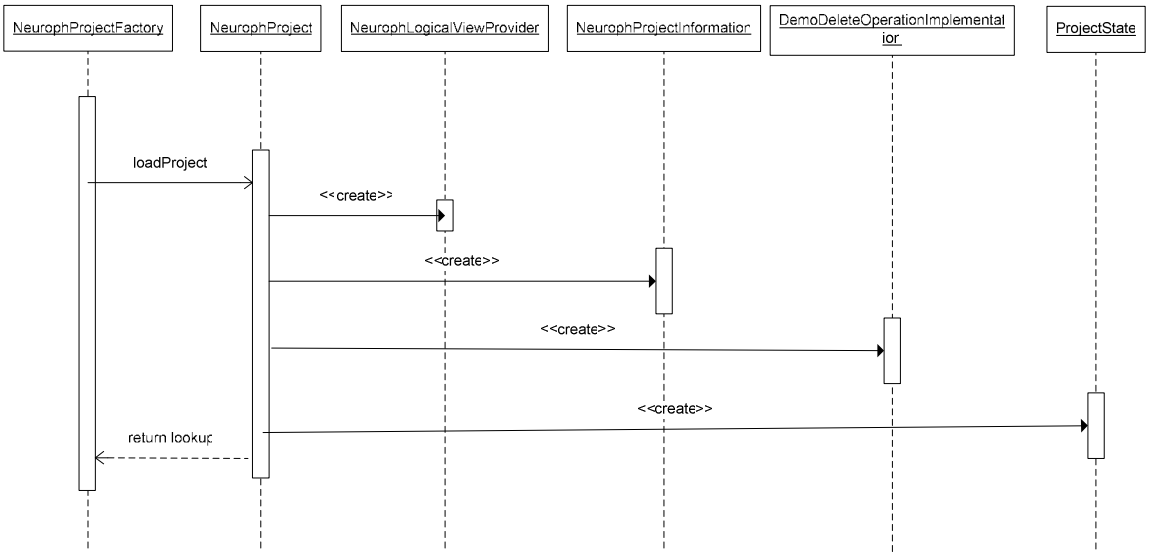
ProjectInformation interfejs omogućava kontrolisanje imena projekta koje se prikazuje, kao i ikonu projekta. Najčešće svi projekti istog tipa dele istu ikonu, međutim moguće je zavisno od meta podataka projekta da projekti imaju različite ikone. Klasa NeurophProjectInformation implementira interfejs ProjectInformation za potrebe projekta tipa *Neuroph*.

NeurophLogicalViewProvider

Interfejs LogicalViewProvider odgovoran je za kontrolisanje prikaza u *Projects* prozoru. Zavisno od semantike projekta moguće je prikazati različite nodove koji su odgovarajući za rad korisnika. Najčešće se prikazuju važni *src* paketi ili fajlovi. ikona i naziv Root noda projekta bi trebalo da se poklapaju sa istim u *ProjectInforamtion*-u. Kontekst meni će varirati zavisno od tipa projekta. Veći broj akcija mogu biti kreirani pomoću *CommonProjectActions* i *ProjectSensitiveActions* klasa. Files sekcija nije pod kontrolom project-a. On uvek prikazuje generisan sadržaj od vrha ka dnu drvoidnom strukturom. Interfejs *LogicalViewProvider* deklariše metode createLogicalView() i findPath(Node root, Object target). Klasa NeurophLogicalViewProvider implementira interfejs LogicalViewProvider za projekte tipa *Neuroph*. Za dati projekat pozivanjem metode createLogicalView() kreira se konkretna struktura nodova koja će biti prikazana.

NeurophProjectFactory

Interfejs *ProjectFactory* odgovoran je za kreiranje projekta u memoriji od direktorijuma na disku. Deklariše tri metode: isProject(FileObject projectDirecory) koja proverava da li se dati direktorijum odnosi na projekat koji taj ProjectFactory prepoznaje;
loadProject(FileObject projectDirecory)-kreira projekat, saveProject(Project project)-čuva projekat. Dijagramom sekvenci na slici 9. dat je prikaz interakcije pomenutih klasa prilikom kreiranja projekta. *NeurophProjectFactory* Implementira interfejs *ProjectFactory* za projekte tipa *Neuroph*.



Slika 9. Dijagram sekvenci kreiranja projekta

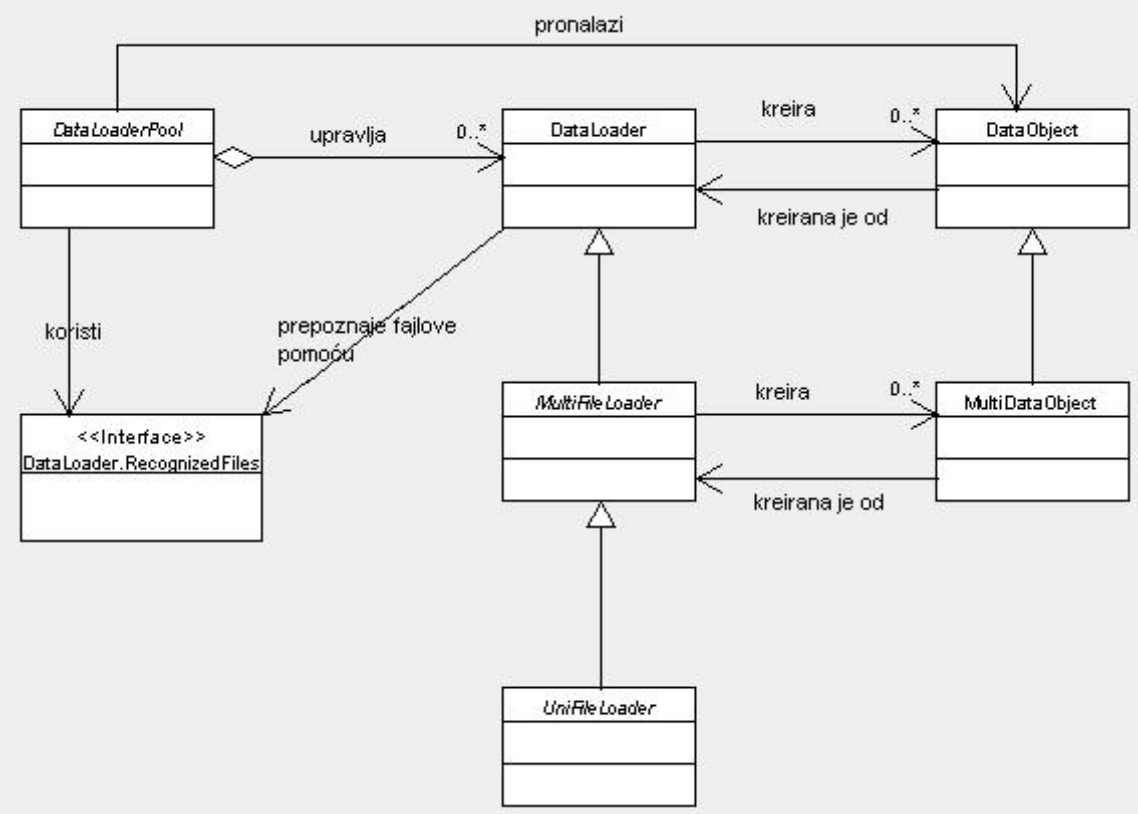
ProjectSample

Uzorak projketa koji će se koristi kao šablon za kreiranje novih tipova projekata, se jednostavno kreira pomoću dva *wizard-a* NetBeans-a : *„New Module Project“* i *„New Project Template“ wizarda*. Nekon kreiranja uzorka projekta i njegovog instaliranja u standardnom IDE-u biće moguće kreiranje novog tipa projekta na isti način kao i svaki drugi projekat u NetBeans-u. Neophodni su nam sledeći fajlovi:

neurophProject.zip	**Zipovani uzorak projekta**
neurophTemplateDescription.html	Kratak opis tipa projekta koji se prikazuje u wizardu
NeurophProjectTemplatePanelVisual.java	Prikazuje se u WizarduPanelu. Uzima podatke za kreiranje projekta.
NeurophProjectTemplateWizardIterator.java	Odgovoran za samo kreiranja projekta. Poziva WizardPanel.
NeurophProjectTemplateWizardPanel.java	Pomoću wizardDescriptor beleži aktivnosti kreiranja projekta i prati da li su podaci validni. Podatke vraća WizardIteratoru.

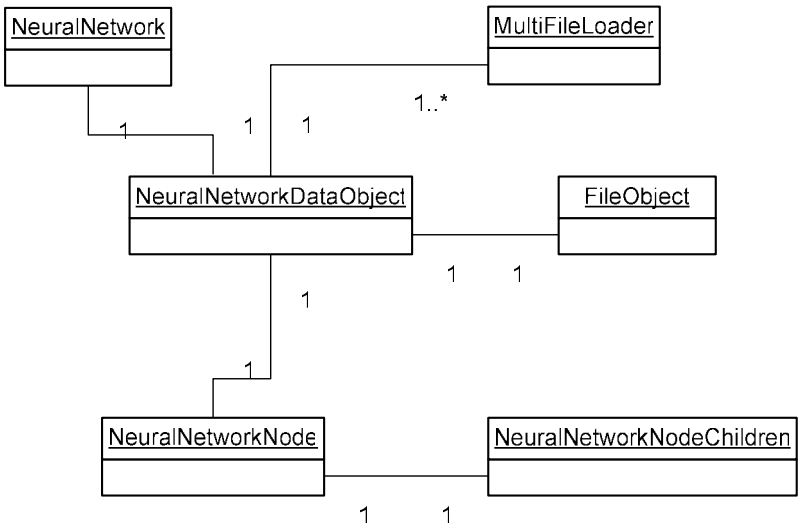
4.2 Specijalizovani tipovi projektnih fajlova

Za neuronske mreže i podatke za trening i testiranje potrebno je kreirati odgovarajuće reprezentacije u obliku fajlova koji pripadaju projektu. Ovi fajlovi će biti registrovani u IDE-u i njima će moći da se manipuliše kao i sa drugim fajlovim razvojnog okruženja. Na slici 10. Prikazana je interakcija klasa NetBeans Platforme *DataSystem API-a* potrebnih za prepoznavanje i kreiranje nekog tipa fajla.



Slika 10. Interakcija klasa Data System API-a

Na slici 11. Prikazan je dijagram klasa značajnih za kreiranje i predstavljanje fajla tipa *NeuralNetwork*. U daljem tekstu dat je opis ovih klasa.



Slika 11. Dijagram klasa za kreiranje i predstaljanje fajla tipa NeuralNetwork

NeuralNetworkDataObject

Klasa *MultiDataObject* odgovorna je za reprezentaciju *FileObject*-a. U konstruktoru prima *MultiFileLoader-a* koji je zadužen za prepoznavanje fajla specifičnog tipa i primarni fajl, tj. *FileObject*. *MultiFileLoader* prepoznaje fajlove koji predstavljaju neuronske mreže ili trening setova po njihovim ekstenzijama. U klasi *MultiDataObject* već su implementirane metode za manipulaciju sa data objektima. Klasa *NeuralNetworkDataObject* nasleđuje *MultiDataObject* klasu. Redefinisanjem metode *createNodeDelegate*() može se odrediti prikaz fajla u *Project* prozoru. U konstruktoru se prosleđeni fajl deserilaizuje nakon čega se dolazi do klase *NeuralNetwork*.

NeuralNetworkNode

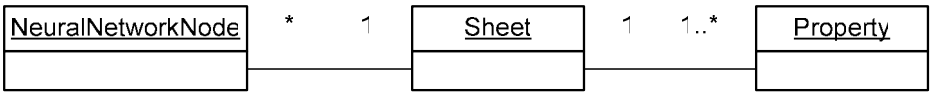
Klasa *AbstractNode* potrebna je prilikom kreiranja noda za reprezentaciju fajl. Klasa *NeuralNetworkNode* nasleđuje AbstractNode klasu. Iako se u *NeuralNetworkDataObject*-u formira nod za dati fajl, kreiranjem posebne klase *NeuralNetworkNode* može se koristiti isti nod za predstavljanje fajla u project prozorukao i u grafičkim komponentama korisničkog interfejsa. U suprotnom bi se morala ponavljati podešavanja nod delegata. Redefinisanjem metode *createSheet()* definisali smo parametre koji će biti prikazani u *Properties* prozoru. U ovoj klasi možemo dodati i akcije koje su specifične za fajl tip, kreiranjem privatnih podklasa koje implementiraju klasu *AbstractAction*.

NeuralNetworkNodeChildren

Klasa *Children.Keys* odgovorna je za kreiranje podnodova jednog noda. Deklariše metode *addNotify*() i *createNodes*(Object n). Metoda *addNotify*() se poziva kada se zatraže deca noda. Metoda *createNodes*(Object n) za prosleđeni objekat kreira niz nodova i vraća ga. Najčešće se koristi kada je potrebno jedan fajl predstaviti sa više nodova. Klasa *NeuralNetworkNodeChildren* nasleđuje klasu *Children*.*Keys*. Potrebno je redefinisati metodu *addNotify*() i *createNodes*(Object n).

4.3 Integracija sa komponentama grafičkog korisničkog interfejsa - *Properties* i *Navigator*

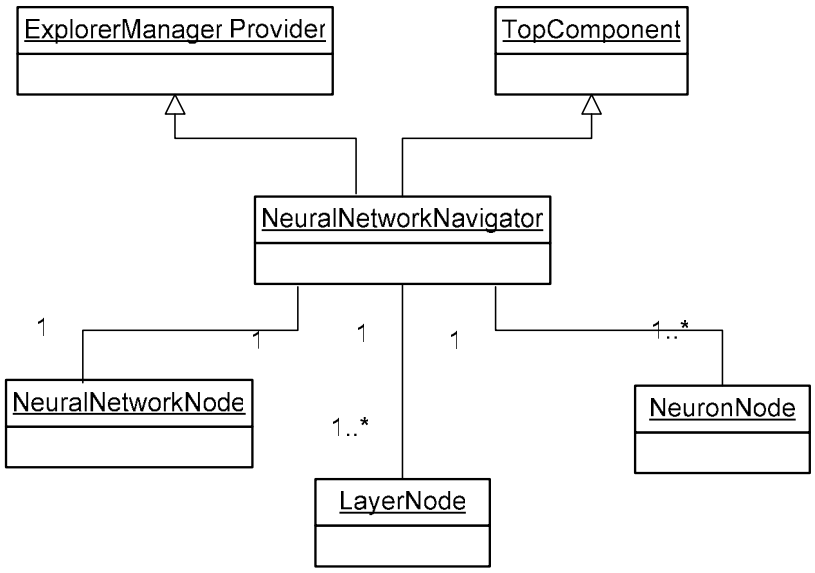
Kao što je već rečeno, osnovni element za vizuelnu reprezentaciju fajlova jeste Node. Da bi se novi tip fajla prilagodio *Properties* prozoru potrebno je da u klasi koja definiše nod za taj fajl, odredimo parametre koji će biti prikazani. Na primeru *Neuroph* aplikacije u klasi *NeuralNetworkNode* potrebno je redefinisati metodu *createSheet*(). U *Node API-u* se nalazi klasa *Sheet* koja predstavlja sve *properties* jednog noda koji će biti prikazani. U metodi createSheet() kreiraćemo *Property* objekte, svaki od ovih objekata će predstavljati neki parametar iz *NeuralNetwork* klase. Kreirane objekte potrebno je dodati u Sheet.Set. Pomoću *PropertyEditor* klase moguće je obezbediti editovanje vredosti preko *Properties* prozora. Kreiranjem novih editora, nasleđivanjem klase *PropertyEditorSupport*, moguće je obezbediti prikazivanja složenih tipova podataka(npr. *LearningRule* klasa), kao i posebna pravila pri editovanju. Na slici 12. Dat je prikaz odgovarajući dijagram klasa.



Slika 12. Dijagram klasa za povezivanje fajla sa *Properties* prozorom

NeuralNetworkNavigator

Za implementaciju navigatora odgovorna je *NeuralNetworkNavigator* klasa. Ona nasleđuje *TopComponent* klasu koja je podklasa *JComponent* klase i prilagođenja je NetBeans *window* sistemu. Svaka *TopComponent* klasa ima *Lookup* i evidenciju o aktivnom nodu. Ove dve komponente određuju „kontekst selekcije", tj. koje će akcije biti omogućene ili koji će biti rezultati tih akcija. Pomoću *ExplorerManager*-a omogućen je hijerarhijski prikaz slojeva i neurona neuronske mreže u navigatoru. Za slojeve neurnoske mreže (*Layer* klasa) i neurone (*Neuron* klasa) napravićemo nodove koji će biti njihova reprezentacija na Navigatoru. Redefinisanjem metode *createSheet()*klase *AbstracNode* parametri ovih klasa takođe će biti prikazani u *Properties* prozoru. Na slici 13. Prikazane su pomenute klase potrebne za kreiranje Navigator prozora.



Slika 13. Dijagram klasa za kreiranje navigatora

5. IMPLEMENTACIJA

U ovom poglavlju opisana je implementacija specijalizovanog razvojnog okruženja pomoću NetBeans Platforme na primeru Neuroph-a. Kao razvojno okruženje i osnovni *framework* tokom implementacije korišćen je NetBeans 6.8. Kao što je objašnjeno u poglavlju „Zahtevi i analiza" osnovni elementi koje je potrebno implementirati prilikom kreiranja specijalizovanog razvojnog okruženja su:

- projekat tipa *Neuroph*;
- fajlovi tipa *NeuralNetwork* I *TrainingSet*;
- elementi grafičkog korisničkog interfejsa (*Navigator* I *Properties* prozor).

Za implementaciju korišćena su rešenja koje nudi NetBeans platforma data u delu "Projektovanje". Za svaki od elemenata koje je potrebno implementirati navedeni su koraci za implementaciju i najznačaniji segmenti koda.

5.1 Implementacija projekta tipa Neuroph

Prvi korak je kreiranje novog NetBeans modula koji će biti deo *Neuroph* programa i koji će sadržati sve fajlove i podešavanja koja su neophodna za kreiranje specijalizovanog razvojnog okruženja. Naziv modula je IDE. Na nivou ovog modula ćemo registrovati i nove tipove projekta i nove tipove fajlova. Zahvaljujući *File System API-u* i podešavanjim u *layer.xml* fajl-u koji pripada IDE modulu, sve specifikacije će biti primenjene na celoj aplikaciji. Najpre ćemo izvršiti registrovaciju novog tipa projekta, a zatim kreirati uzorak projekta i implementirati kreriranje novog projekta pomoću *wizard-a*. Za registraciju tipa projekta rekli smo da je neophodno definisati folder određenog naziva čije prisustvo će biti signal *ProjectFactory-ju* da se radi o projektu od značaja. Taj naziv folder će biti „neurophproject“ i njegovo ime se podešava u klasi koja implementira *ProjectFactory*. U našem primeru to je klasa *NeurophProjectFactory*:

```
public class NeurophProjectFactory implements ProjectFactory {

    public static final String PROJECT_DIR = "neurophproject";

    @Override
    public boolean isProject(FileObject projectDirectory) {
        return projectDirectory.getFileObject(PROJECT_DIR) != null;
    }
    @Override
    public Project loadProject(FileObject dir, ProjectState state)
throws IOException {
        return isProject(dir) ? new NeurophProject(dir, state) :
null;
    }



    @Override
    public void saveProject(final Project project) throws
IOException, ClassCastException {
        FileObject projectRoot = project.getProjectDirectory();
        if (projectRoot.getFileObject(PROJECT_DIR) == null) {
            throw new IOException("Project dir " +
projectRoot.getPath() +
                    " deleted," +
                    " cannot save project");
        }
    }
}
```

Slika 14. Segment koda klase NeurophProjectFactory

Pomoću parametra „PROJECT_DIR“ određen je naziv foldera koji određuje projekat tipa *Neuroph*. Metoda *loadProject* ispituje prisustvo ovog folder i kreira projekat tipa NeurophProject.

Klasa *NeurophProject* implementira interfejs *Project*. Kao što smo već spomenuli, u konstruktoru prima *FileObject* od kojeg će biti kreiran sam projekat. Najvažnija metoda *Project* interfejsa koju je potrebno implementirati je getLookup().

```
public class NeurophProject implements Project {
    private final FileObject projectDir;
    private final ProjectState state;
    private Lookup lookup;
    public NeurophProject(FileObject projectDir, ProjectState state)
{
        this.projectDir = projectDir;
        this.state = state;
    }
    @Override
    public FileObject getProjectDirectory() {
        return projectDir;
    }
  +  public FileObject[] getSubFolders() {...}

    @Override
    public Lookup getLookup() {
        if (lookup == null) {
            lookup = Lookups.fixed(new Object[]{
                        state,
                        new ActionProviderImpl(),
                        new DemoDeleteOperation(),
                        new DemoCopyOperation(this),
                        new Info(),
                        new NeurophProjectLogicalView(this),
                    });
        }
        return lookup;
    }
 +   private final class ActionProviderImpl implements
ActionProvider {...}
+     private final class DemoDeleteOperation implements
DeleteOperationImplementation {...}
+    private final class DemoCopyOperation implements
CopyOperationImplementation {... }
+    private final class Info implements ProjectInformation {...}
    }
```

Slika 15. Segment koda klase NeurophProject

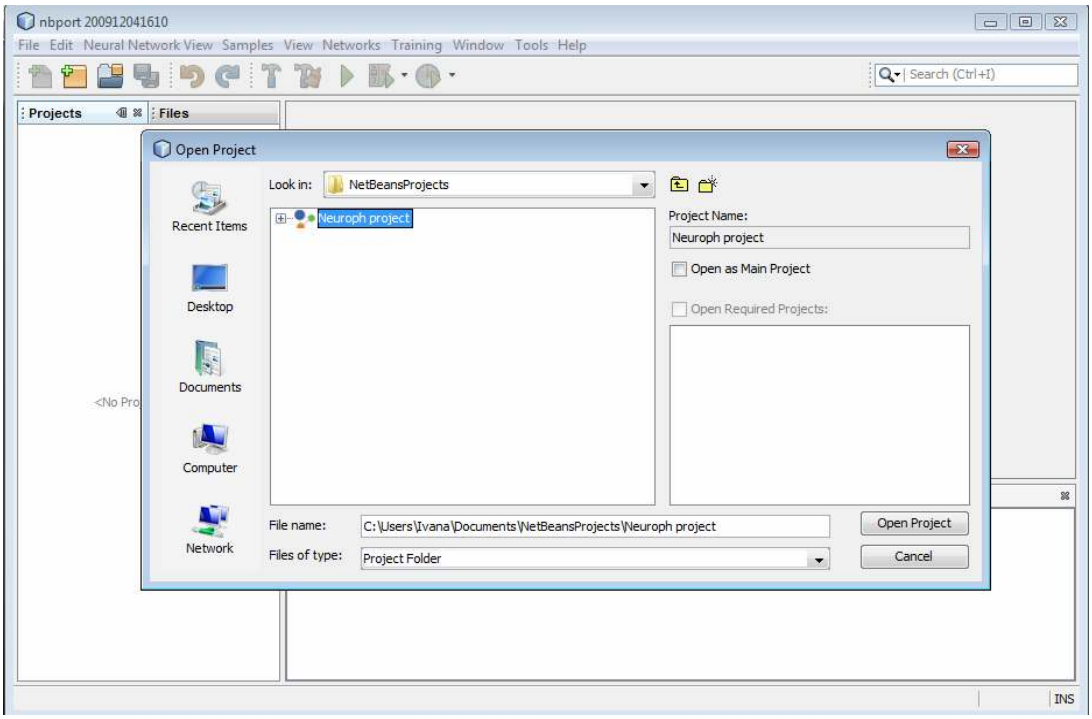
U metodi *getLookup()* smo registrovali osnovne osobine projekta. Implementiranjem *ActionProvider-a* na vrlo jednostavan način se projektu priključuju osnovne akcije. Metoda *getSubFolders()* je značajna za logički prikaz projekta. Ovu metodu koristi klasa *NeurophLogicalViewProvider*. Za vraćeni niz objekata klase *FileObject* kreiraju se nodovi i odgovarajuća struktura za prikaz. Najvažnija metoda klase *NeurpohLogicaViewProvider* je *createLogicalView()*. Za prosleđeni projekat ova metoda prolazi kroz sve podfoldere i kreira za njih *node* delegate. Zahvaljujuću *FileSystem API-u* i osobinama *FileObject-a* ispitivanje direktorijuma i podfoldera projekta je vrlo jednostavno. U metodi *createLogicalView()* možemo da vidimo primer kolaboracije *FileObject-a, DataObject-a* i *Node-a* tj. FileSystem *API, Project API* i *Node API*. Tu se vidi način na koji se delegiraju odgovornosti i uloge od predstavljanja projekta u memoriji do njegove reprezentacije u korisničkom interfejsu.

Za osnovni *root* folder projekta kreira se *ProjectNode* klasa, koja nasleđuje *FilterNode* i u njoj se vrše podešavanja ikone, naziva i akcija za node projekta.

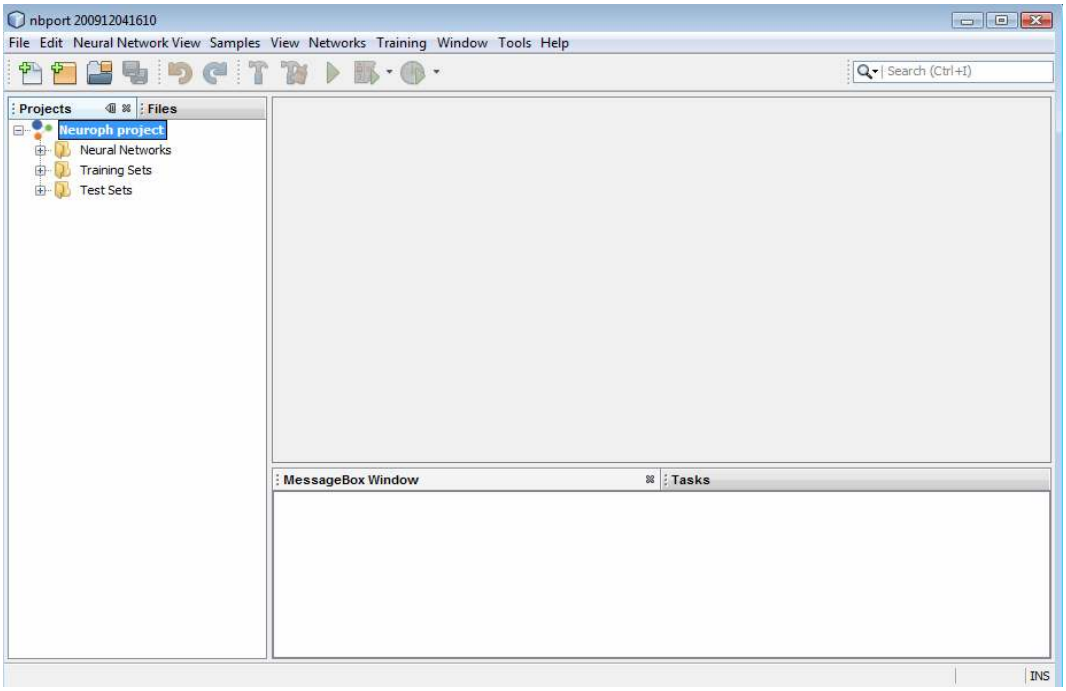
```
class NeurophProjectLogicalView implements LogicalViewProvider {
    private final NeurophProject project;
    public NeurophProjectLogicalView(NeurophProject project) {
        this.project = project;
    }
    @Override
    public org.openide.nodes.Node createLogicalView() {
        try {
           FileObject text = project.getTextFolder(true);
            FileObject[] fs = project.getSubFolders();
            DataFolder dfs1 = DataFolder.findFolder(fs[0]);
            DataFolder dfs2 = DataFolder.findFolder(fs[1]);
            DataFolder dfs3 = DataFolder.findFolder(fs[2]);
            DataFolder textDataObject = DataFolder.findFolder(text);
            Node realTextFolderNode =
textDataObject.getNodeDelegate();
            Node n = null;
            Node n2 = null;
            Node n5 = null;
            try {
                n = dfs1.getNodeDelegate();
                n2 = dfs2.getNodeDelegate();
                n5 = dfs3.getNodeDelegate();
            } catch (Exception ex) {
                Exceptions.printStackTrace(ex);
            }
            final Node finalnode3 = n;
            final Node finalnode4 = n2;
            final Node finalnode6 = n5;
            final ProjectNode tn = new
ProjectNode(realTextFolderNode, project);
                tn.getChildren().MUTEX.postWriteRequest(new
Runnable() {
                    public void run() {
                        tn.addSubNodes(new Node[]{finalnode3,
finalnode4, finalnode6});
                    }
                });
            return tn;
        } catch (DataObjectNotFoundException donfe) {
            Exceptions.printStackTrace(donfe);
            return new AbstractNode(Children.LEAF);
        }
    }
        private static final class ProjectNode extends FilterNode
{...}
}
```

Slika 16. Segment koda klase NeurophLogicalViewProvider

Sada je moguće videti prve rezultate prilikom pokretanja aplikacije *Neuroph* Moguće je učitati projekat tipa *Neuroph*. Svaki folder koji u sebi sadrži podfolder naziva *„neurophproject“* biće prepoznat kao projekat tipa *Neuroph*. Na slici 17. Dat je prikaz funkcionalnosti otvaranja projekta tipa *Neuroph*.



Slika 17. Otvaranje Neuroph projekta



Slika 18. Prikaz sadržaja Projekta

Da bi u osnovi projekta postojala struktura kao na slici 18 potrebno je da za *Neuroph* projekat napravimo željeni uzorak odnosno šablon. Na nivou modula kreiraćemo novi *Project template*, koji se nalazi u kategoriji *„Module Development“* NetBeans IDE file type-a. Potrebno je da selektujemo neki *Neuroph* projekat koji smo već kreirali. U tom projektu smo prethodno kreirali foldere *„Neural networks“*, *„Training sets“*, *„Test sets“*. Ovaj projekat i njegov sadržaj će biti osnova pri svakom kreiranju *Neuroph* projekta kroz *new project wizard*. Takođe, kao rezultat biće kreirani fajlovi neophodni za kreiranje neuroph projecta u new Project wizardu.

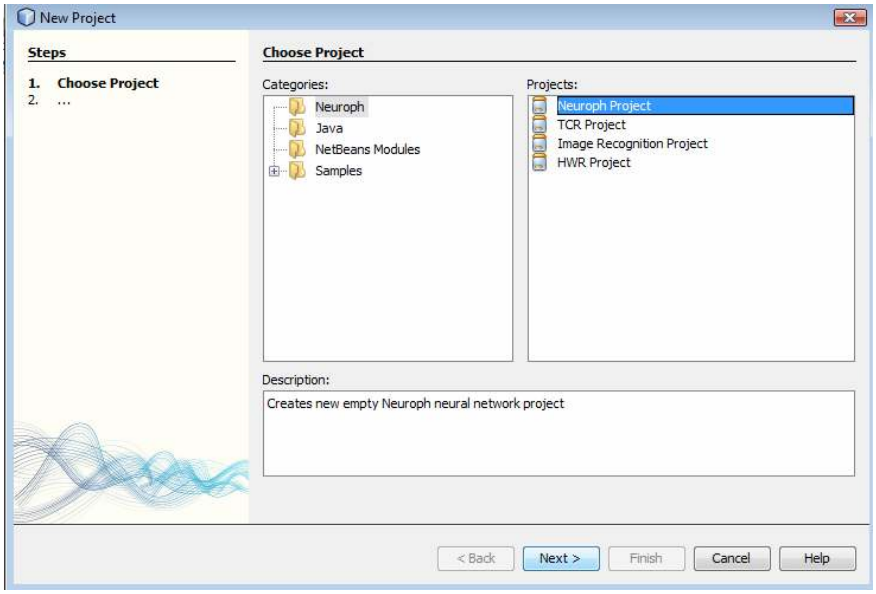
To su sledeću fajlovi : neurophProject.zip, neurophTemplateDescription.html, NeurophProjectTemplatePanelVisual.java, NeurophProjectTemplateWizardIterator.java, NeurophProjectTemplateWizardPanel.java. Fajl neurophProject.zip predstavlja *zipovan* neuroph projekat koji smo prilikom kreiranja Project template-a selektovali. U neurophTemplateDescription.html fajlu se nalazi opis projekta koji se pojavljuje prikom kreiranja projekta new Project wizardom. Fajlovi NeurophProjectTemplatePanelVisual.java, NeurophProjectTemplateWizardIterator.java i NeurophProjectTemplateWizardPanel.java su deo wizarda za kreiranje projekta, takođe se mogu prilagoditi kreiranju različitih podtipova projekta. Podešavanjima u layer.xml fajlu dodata je kategorija Neuroph i podrška za kreiranje Neuroph projekta, ImageRecognition, HandWrittingRecognition i ImageRecognition projekta. Na slici 19 Prikazana su podešavanja vezana za project template u layer.xml-u.

```
<folder name=”Templates”>
 <folder name="Project">
            <folder name="APISupport">
                <file name="neurophTemplateProject.zip"
url="neurophTemplateProject.zip">
                    <attr name="position" intvalue="400"/>
                    <attr name="displayName"
bundlevalue="org.neuroph.netbeans.ide.Bundle#Templates/Project/APISu
pport/neurophTemplateProject.zip"/>
                    <attr name="instantiatingIterator"
methodvalue="org.neuroph.netbeans.ide.project.type.NeurophProjectTem
plateWizardIterator.createIterator"/>
                    <attr name="instantiatingWizardURL"
urlvalue="nbresloc:\\org\\neuroph\\netbeans\\ide\\project\\type\\neu
rophTemplateDescription.html"/>
                    <attr name="template" boolvalue="true"/>
                </file>
            </folder>
            <folder name="Neuroph">
                <file name="neurophProject.zip"
url="neurophTemplateProject.zip">
                    <attr name="displayName"
bundlevalue="org.neuroph.netbeans.ide.Bundle#Templates/Project/APISu
pport/neurophTemplateProject.zip"/>
                    <attr name="instantiatingIterator"
methodvalue="org.neuroph.netbeans.ide.project.type.NeurophProjectTem
plateWizardIterator.createIterator"/>
                    <attr name="instantiatingWizardURL"
urlvalue="nbresloc:/org/neuroph/netbeans/ide/project/type/neurophTem
plateDescription.html"/>
                    <attr name="template" boolvalue="true"/>
```

```
                </file>
                <file name="neurophProject_1.zip"
url="neurophProject_1.zip">
                    <attr name="displayName" stringvalue="TCR
Project"
bundlevalue="org.neuroph.netbeans.ide.Bundle#Templates/Project/APISu
pport/neurophTemplateProject.zip"/>
                    <attr name="instantiatingIterator"
methodvalue="org.neuroph.netbeans.ide.project.type.NeurophProjectTem
plateWizardIteratorTCR.createIterator"/>
                    <attr name="instantiatingWizardURL"
urlvalue="nbresloc:/org/neuroph/netbeans/ide/project/type/neurophTem
plateDescription.html"/>
                    <attr name="template" boolvalue="true"/>
                </file>
                <file name="neurophProject_3.zip"
url="neurophProject_3.zip">
                    <attr name="displayName" stringvalue="HWR
Project"
bundlevalue="org.neuroph.netbeans.ide.Bundle#Templates/Project/APISu
pport/neurophTemplateProject.zip"/>
                    <attr name="instantiatingIterator"
methodvalue="org.neuroph.netbeans.ide.project.type.NeurophProjectTem
plateWizardIteratorHWR.createIterator"/>
                    <attr name="instantiatingWizardURL"
urlvalue="nbresloc:/org/neuroph/netbeans/ide/project/type/neurophTem
plateDescription.html"/>
                    <attr name="template" boolvalue="true"/>
                </file>
                <file name="neurophProject_2.zip"
url="neurophProject_2.zip">
                    <attr name="displayName" stringvalue="Image
Recognition Project"
bundlevalue="org.neuroph.netbeans.ide.Bundle#Templates/Project/APISu
pport/neurophTemplateProject.zip"/>
                    <attr name="instantiatingIterator"
methodvalue="org.neuroph.netbeans.ide.project.type.NeurophProjectTem
plateWizardIteratorIMR.createIterator"/>
                    <attr name="instantiatingWizardURL"
urlvalue="nbresloc:/org/neuroph/netbeans/ide/project/type/neurophTem
plateDescription.html"/>
                    <attr name="template" boolvalue="true"/>
                </file>   </folder>  </folder>    </folder>
```

Slika 19. Segment koda u layer.xml fajlu neophodan za kreiranje template projekata

Kao rezultat iz *Neuroph* aplikacije moguće je kreiranje različtih projekata, što je prikazano na slici 20.



Slika 20. Kreiranje Neuroph projekata pomoću wizarda

5.1.1 Rezime koraka za kreiranje projekta tipa Neuroph

1. **Kreiranje novog modula IDE.**

- Potrebno je izabrati *File > New Project*. Kategorija NetBeans Modules, iazbrati Module.
- Prilikom konfiguracije modula u wizardu potrebno je izabrati generisanje Layer.xml fajla - *„Generate XML Layer“* .
- Kreirani modul će koristi sledeće *API-eve*: *Datasystems API, Dialogs API, File System API, Nodes API, Project API, Project UI API, UI Utilities API, Utilities API*. Potrebno kreirati zavisnosti na ove module. Desnim klikom na modul, zatim izborom „*Properties*“ otvara se *Project Properties* dijalog. U panelu „Libraries“ potebno je dadati pomenute module.

2. **Kreiranje osnovnih klasa za reprezentovanje projekta.**

- **Kreiranje Project Facotory-a.**
  Kreirati novu klasu *„NeurophProjectFactory“*. Ova klasa mora da implementira interfejs *ProjectFactory*. Definisati String parametar PROJECT_DIR i dodeliti mu vrednost ”neurophproject”. Kao što smo objasnili prisustvo ovog parametra će određivati projekat tipa neuroph. Implementirati metodu *loadProject(FileObject dir, ProjectState state), isProject(FileObject projectDirectory) i saveProject(Project project).*
- **Kreiranje Project-a.**
  Kreirati novu klasu *„NeurophProject“* koja implementira interfejs Project. U konstruktoru klasa prima FileObject i ProjectState objekte. Najvažnija metoda za reimplementiranje je getLookup(). Ona vraća lookup projekta u kome su registrovane sve mogućnosti projekta – stanje(ProjectState), operacija brisanja(DemoDeleteOperation), operacija kopiranja(DemoCopyOperation), informacije o projektu(Info), logički prikaz(NeurophProjectLogicalView).

Klase *Info*, *DemoDeleteOperation* i *DemoCopyOperation* kreiraćemo kao privatne klase u okviru klase *NeurophProject*. Klasa Info implementira interfejs *ProjectInformation* i definiše opis, naziv i ikonu projekta. Klasa DemoDeleteOperation implementira interfejs DeleteOperationImplementation, a klasa *DemoCopyOperation* implementira interfejs *CopyOperationImplementation*.

- **Kreiranje LogicalVIew-a.**
  Kreirati novu klasu *NeurophProjectLogicalView*. Ova klasa implementira interfejs *LogicalViewProvider*. Potrebno je implementirati metodu *createLogicalView()* koja za dati projekat i njegove foldere kreira node delegate.

3. Kreiranje uzorka aplikacije

- Desnim klikom na modul IDE i izborom New otvara se *New File* dijalog. U kategoriji *Modul Development* potrebno je izabrati *New Project Template*. Prilikom konfigurisanja projekta za template prvo potebno je izabrati neki neuroph projekat, tj. neki folder sa podfolderom „neurophproject“. Ovaj folder sa svojim sadržajem će biti osnova za kreiranje projekta pomoću new project wizarda. Sledeći korak je određivanje imena za prikaz template-a i njegove kategorije. Za sada se može ostaviti kategorija „*APISupport*“.
- Nakon ovog koraka u *layer.xml* fajlu generisan je kod za registraciju novih tipova projekata u aplikaciji. Kreiranjem podfoldera *Neuroph* u folderu „*Templates*“ *layer.xml* fajla kreira se kategorija *Neuroph* u new project *wizard-u*. Premeštanjem generisanog koda za povezivanje projekata u ovaj folder biće omogućeno kreiranje projekata iz kategorije *Neuroph*.

5.2 Implementacija fajlova tipa NeuralNetwork i TrainingSet

Na primeru *Neuroph* aplikacije potrebno je da napraviti nove tipove fajlova koji će biti reprezentacija klasa *NeuralNetwork* i *TrainingSet*. Pošto je veći deo implementacija fajlova tipa *NeuralNetwork* i fajlova tipa *TrainigSet* isti, u daljem radu pokazana su sva podešavanja samo na primeru neuronskih mreža. Najpre je potrebno da registrovati novi tip podataka. Registrovanje se vrši pomoću *new file type wizarda*, kategorija *module development*, u NetBeans IDE-u. Prvi korak je određivanje ekstenzije fajla i MIME tipa. Prepoznavanje fajl može biti po ekstenziji ili po XML root elementu. Za neuronske mreže MIME tip će biti „text/x-nnet“, a ekstenzija .nnet. Najvažnija klasa za kreiranje novog fajla neuronske mreže je *NeuralNetworkDataObject*.

```
public class NeuralNetworkDataObject extends MultiDataObject {

    public NeuralNetworkDataObject(FileObject pf, MultiFileLoader
loader) throws DataObjectExistsException, IOException {
        super(pf, loader);
        nnfile = pf;
        nnetwork = new NeuralNetwork();
        nnetwork = readFile(pf);
        cookies = getCookieSet();
        cookies.assign(NeuralNetwork.class, nnetwork);
        //  cookies.add((Node.Cookie) DataEditorSupport.create(this,
getPrimaryEntry(), cookies));
```

```
        cookies.add((Node.Cookie) new
NeuralNetworkEditor(getPrimaryEntry()));
        cookies.add(this);
    }
    NeuralNetwork nnetwork;
    FileObject nnfile;
    CookieSet cookies;

    @Override
    protected Node createNodeDelegate() {
        DataNode node = new DataNode(this, Children.LEAF,
getLookup());
        node.setShortDescription("Name is " +
getLookup().lookup(NeuralNetwork.class).toString());
        node.setDisplayName(nnfile.getName());
        return node;
    }
    public NeuralNetwork getNnetwork() {... }
    public void setNnetwork(NeuralNetwork nnetwork) {... }
    public NeuralNetwork readFile(FileObject test) {...}
    @Override
    public Lookup getLookup() {
        return getCookieSet().getLookup();
    }
}
```

Slika 21. Segment koda klase NeuralNetworkDataObject

Konstruktoru se prosleđuje *FileObject* koji predstavlja sam fajl neuronske mreže u memoriji i *MultiFileLoader* koji je odgovoran za prepoznavanje fajla ekstenzije .nnet. Prosleđeni *FileObject* se deserijalizuje kako bi se dobila klasa *NeuralNetwork* koju sam fajl predstavlja. U konstruktoru takođe treba dodati u cookieSet-u akcije za manipulaciju sa fajlom. Standardno to su akcije *DataEditor* klase (kopiranje, katovanje, brisanje, otvaranje, preimenovanje). Umesto *DataEditor-a* za neuronsku mrežu kreirali smo novi *editor* , tj. *NeuralNetworkEditor* klasu.

```
public class NeuralNetworkEditor extends OpenSupport implements
OpenCookie, CloseCookie {

    public NeuralNetworkEditor(NeuralNetworkDataObject.Entry entry)
{
        super(entry);
        nnetdataobject = (NeuralNetworkDataObject)
entry.getDataObject();
    }
  NeuralNetworkDataObject  nnetdataobject;

    @Override
    protected CloneableTopComponent createCloneableTopComponent() {
        VisualTopComponent tc = new VisualTopComponent();
        tc.getDefault().open();
        return tc;
    }
```

```
    @Override
    public void open() {
        VisualTopComponent.getDefault().open();
        NeuralNetwork nnet = nnetdataobject.getNNetwork();
       FileObject [] trainingsetfolder =  (FileObject[])
ActiveProject.getInstance().getActiveProject().getSubFolders();
        Vector<TrainingSet> traingSets = new Vector<TrainingSet>();
        FileObject [] trsetch =trainingsetfolder[1].getChildren();
        for (int i = 0; i < trsetch.length; i++) {
            FileObject fileObject = trsetch[i];
           traingSets.add(readTSFiles (fileObject));
        }

EasyNeuronsViewController.getInstance().openNetworkViewFrame(nnet,
traingSets);
    }

    public TrainingSet readTSFiles(FileObject path) {.. }

    @Override
    public boolean close() {... }
}
```

Slika 22. Segment koda klase NeuralNetworkEditor

Nasleđivanjem *OpenSupport* klase i *OpenCookie* interfejsa redefinisana je akcija otvaranja fajla. Kao rezultat sada se otvara *GraphView* prikaz neuronske mreže i prozor *NetworkViewFrame*. Kao što je rečeno, sadržaj fajla neuronske mreže je serijalizovana neuronska mreža tako da njen prikaz u editoru ne bi bio od veliog značaja za korisnika. Takođe, moguće je da korisnik greškom napravi izmene u sadržaju fajla i time onemogući deserijalizvaciju neuronske mreže pri učitavanju. Za otvaranje *NetworkViewFrame*-a neophodno je prosleđivanje aktivne neuronske mreže i svih skupova podataka za trening i testiranje te mreže. Najsigurniji način za uzimanje postojećih training setova je direktno iz aktivnog projekta. Zahvaljujući *Lookup* mehanizmu, evidencija o aktivnom projektu ili fajlu je prilično jednostavna. Recimo da bismo došli do aktivnog projekta potrebno je da uradimo sledeće :

```
private NeurophProject nresult =
Utilities.actionsGlobalContext().lookupResult(NeurophProject.c
lass);
```

Alternativa za ovaj pristup je korišćenje *LookupListener-a*. Njegovu primenu ćemo prikazati kod implementacije *Navigator-a*.

Za samo registraciju novog tipa fajla potrebni su nam i sledeći fajlovi:

- **NeuralNetworkResolver.xml koji** mapira .nnet i .NNET ekstenzije MIME tipa, kako bi *DataLoader* mogao da prepozna ove tipove fajlova. Sadržaj rezolvera je sledeći:
  <MIME-resolver>
  <file>
  <ext name="nnet"/>
  <ext name="NNET"/>
  <resolver mime="text/x-nnet"/>

```
            </file>
        </MIME-resolver>
```

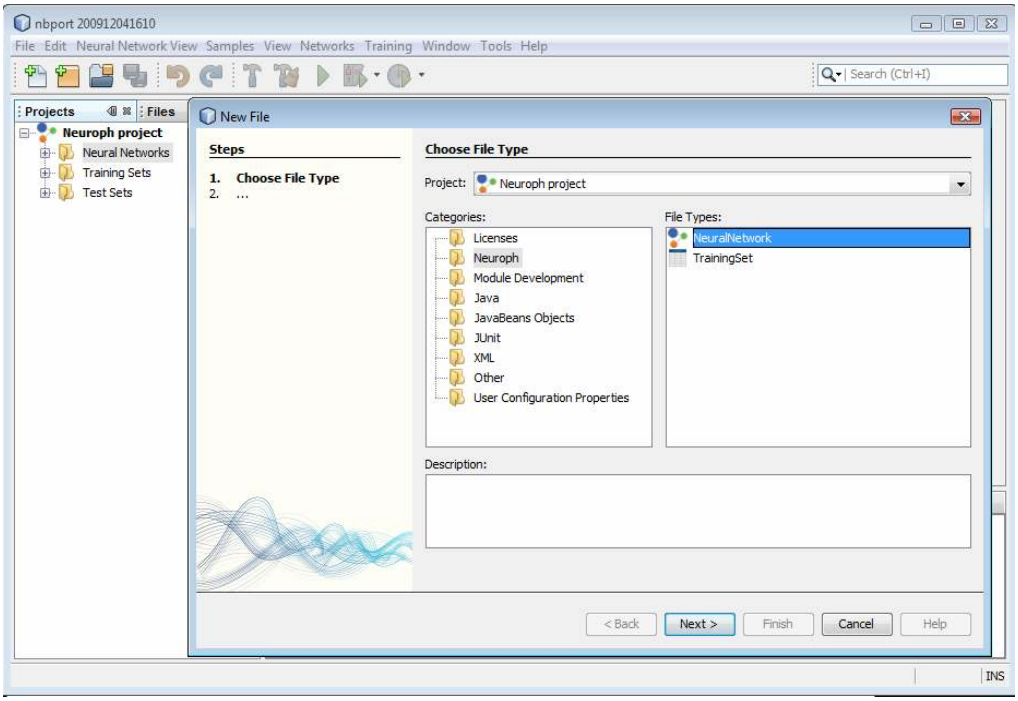
- **NeuralNetworkTemplate.nnet** predstavlja osnovu fajla koja će biti kreirana. Može biti prazan ili imati sadržaj koji će biti intuitivan za korisnika da započne sa radom.

U layer.xml fajlu IDE modula je dodat kod za povezivanja template fajlova za neuronske mreže i training setove, kao i evidencija o MIME rezolverima.

```
<folder name="Services">
        <folder name="MIMEResolver">
            <file name="NeuralNetworkResolver.xml"
url="NeuralNetworkResolver.xml">
                <attr name="displayName"
bundlevalue="org.neuroph.netbeans.ide.Bundle#Services/MIMEResolver/N
euralNetworkResolver.xml"/>
            </file>
            <file name="TrainingSetResolver.xml"
url="TrainingSetResolver.xml">
                <attr name="displayName"
bundlevalue="org.neuroph.netbeans.ide.Bundle#Services/MIMEResolver/T
rainingSetResolver.xml"/>
            </file>
        </folder>
    </folder>
    <folder name="Templates">
        <folder name="Neuroph">
            <file name="NeuralNetwork.nnet"
url="NeuralNetworkTemplate.nnet">
                <attr name="template" boolvalue="true"/>
            </file>
            <file name="TrainingSet.tset"
url="TrainingSetTemplate.tset">
                <attr name="template" boolvalue="true"/>
            </file>
        </folder><-folder>
```

Slika 23. Registrovanje tipova fajlova u layer.xml

Pošto je u template delu *layer.xml-a* dodata kategorija *Neuroph* i u njoj fajlovi tipa *NeuralNetwork* i *TrainingSet*, pomoću *new file wizarda* biće moguće kreiranje ovih fajlova.



Slika 24. Kreiranje fajla tipa NeuralNetwork

Veoma bitno podešavanje u *layer.xml-u* pri registraciji fajlova je u folderu „*Loaders*“. Potrebno je da dodamo loadere za fajlove neuronske mreže i podataka za trening i testiranje. Na osnovu deklarisanog MIME tipa, „text/x-nnet“ kreiraćemo folder „text“ i podfolder „x-nnet“ u kome će biti naznačene akcije koje se vezuju za dati tip i neophodni podaci za prepoznavanje fajlova .nnet i povezivanje sa *DataObject-om*. U folderu „*Factories*“ se vrše ova podešavanja.

```
  <folder name="Loaders">
        <folder name="text">
            <folder name="x-nnet">
                <folder name="Actions">
                    <file name="org-neuroph-netbeans-ide-
OpenVisualWindowA.shadow">
                        <attr name="originalFile"
stringvalue="Actions/Edit/org-neuroph-netbeans-ide-
OpenVisualWindowA.instance"/>
                        <attr name="position" intvalue="100"/>
                    </file>
                    <file name="org-openide-actions-
CopyAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/Edit/org-openide-actions-CopyAction.instance"/>
                        <attr name="position" intvalue="400"/>
                    </file>
                    <file name="org-openide-actions-
CutAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/Edit/org-openide-actions-CutAction.instance"/>
                        <attr name="position" intvalue="300"/>
```

```
                    </file>
                    <file name="org-openide-actions-
DeleteAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/Edit/org-openide-actions-
DeleteAction.instance"/>
                        <attr name="position" intvalue="600"/>
                    </file>
                    <file name="org-openide-actions-
FileSystemAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/System/org-openide-actions-
FileSystemAction.instance"/>
                        <attr name="position" intvalue="1100"/>
                    </file>
                    <file name="org-openide-actions-
OpenAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/System/org-openide-actions-
OpenAction.instance"/>
                        <attr name="position" intvalue="100"/>
                    </file>
                    <file name="org-openide-actions-
PropertiesAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/System/org-openide-actions-
PropertiesAction.instance"/>
                        <attr name="position" intvalue="1400"/>
                    </file>
                    <file name="org-openide-actions-
RenameAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/System/org-openide-actions-
RenameAction.instance"/>
                        <attr name="position" intvalue="700"/>
                    </file>
                    <file name="org-openide-actions-
SaveAsTemplateAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/System/org-openide-actions-
SaveAsTemplateAction.instance"/>
                        <attr name="position" intvalue="900"/>
                    </file>
                    <file name="org-openide-actions-
ToolsAction.shadow">
                        <attr name="originalFile"
stringvalue="Actions/System/org-openide-actions-
ToolsAction.instance"/>
                        <attr name="position" intvalue="1300"/>
                    </file>
                </folder>
                <folder name="Factories">
                    <file name="NeuralNetworkDataLoader.instance">
                        <attr name="SystemFileSystem.icon"
urlvalue="nbresloc:/org/neuroph/netbeans/ide/images2.gif"/>
                        <attr name="dataObjectClass"
```

```
stringvalue="org.neuroph.netbeans.ide.NeuralNetworkDataObject"/>
                        <attr name="instanceCreate"
methodvalue="org.openide.loaders.DataLoaderPool.factory"/>
                        <attr name="mimeType" stringvalue="text/x-
nnet"/>
                    </file>
                </folder>
            </folder>
```

Slika 25. Dodavanja Loader-a za fajlove neuronske mreže

Zahvaljujući registrovanim akcijama, sa fajlovima neuronske mreže moguće je manipulisati kao sa uobičajnim fajlovima NetBeans IDE-a, npr. pri promeni parametara neuronske mreže biće automatski aktivirnao *Save* dugme u toolbar-u. Korišćenjem alatki NetBeans –a olakšan je proces razvoja i obezbeđene su pouzdane akcije za manipulisanje fajlovima.

5.2.1 Rezime koraka za kreiranje fajlova tipa NeuralNetwork

1. **Registracija novog tipa fajlova**

- Desni klik na IDE modul, izbor *New > File Type*.
- „*File Recognition*“ panelu kao MIME tip potrebno je upisati „text/x-nnet“, a zatim selektovati opciju „By Filename extension“ čime smo odredili da se fajlo prepoznaju po ekstenziji. U polje „*Extension*“ potrebno je upisati .nnet i .NNET. Kliknuti na Next.
- U panelu „*Name, Icon and Location*“ potrebno je dodati prefix naziva klasa koje će predstavljati fajl, u našem slučaju „*NeuralNetwork*“. Takođe, moguće je odrediti ikonu za tip fajla i lokaciju generisanih fajlova. Kliknuti na *Finish*.
- Pomoću new *File Type wizard-a* kreirani su fajlovi: *NeuralNetworkDataObject.java*, *NeuralNetworkResolver.java*, *NeuralNetworkTemplate.nnet*, a modifikovani fajlovi *project.xml*, *Bundel.properties* i *layer.xml*.

2. **Prilagođavanje osnovnih klasa koje reprezentuju fajl specifičnim potebama**

- **Modifikovanje NeuralNetworkDataObjecta**
  U konstruktoru *NeuralNetworkDataObject* potrebno je deserijalizovati primljeni *FileObject* da bi se došlo do klase *NeuralNetwork*. Takođe, u konstruktoru za dati tip *DataObject-a* pridružićemo *cookies* akcije i *NeuralNetworkEditor* kako bismo redefinisali akcije otvaranja i zatvaranja.
  Potrebno je redefinisati metodu *createNodeDelegate()* za kreiranje noda koji će predstavljati dati fajl.
- **Modifikovanje NeuralNetworkTemplate**
  Predstavlja početni sadržaj fajla. Prilikom kreiranja nnet fajla u sadržaj će biti serijalizovana prazna neuronska mreža.
- **Modifikovanje layer.xml fajla**
  NeuralNetworkTemplate fajlovi su nakon wizarda smešteni u kategoriju „Other“ new file wizarda. Dodavanjem podfoldera *Neuroph* u folder *Templates* u layer.xml-u i premeštanjem generisanog koda za neuralnnetworktemplate u ovaj folder može se kreirati nova kategorija za kreiranje fajlova.

3. **Kreiranje editora za data objekat**

- **Kreiranje NeuralNetworkEditor klase**
  Potrebno je kreirati novu klasu naziva *NeuralNetworkEditor* koja nasleđuje klasu *OpenSupport* i imeplementira interfejs *OpenCookie* i *CloseCookie*. Potrebno je redefinisati metode *open()* i *close()*.

4. **Kreiranje Noda koji predstavlja data objekat**

- **Kreiranje NeuralNetworkNode klase**
  Potrebno je kreirati novu klasu *NeuralNetworkNode* koja nasleđuje klasu *AbstractNode*. Redefinisanjem metoda *getIcon*() i *getName*() vrše se podešavanja izgleda fajla u project prozoru,a redefinisanjem metode *getActions()* mogu se pridružiti dodatne akcije nodu fajla.

5.3 Implementacija integracije sa komponentama grafičkog korisničkog interfejsa-Properties i Navigator

Za reprezentaciju *NeuralNetwork* klase potrebno je kreirati posebnu klasu, *NeuralNetworkNode*. Ona nasleđuje klasu *AbstracNode*. Redefinisanjem metoda *getIcon(), getOpenedIcon(), getActions()* definisana su osnovna ponašanja noda. U konstruktoru se prosleđuje *NeuralNetwork* klasa. Za vrednosti parametara ove klase biće prilagođen prikaz u properties prozoru. Za ovo podešavanje potrebno je redefinisanje metode *createSheet()*. Neophodno je kreirati *Property* objekte, pridružiti ime odgovarajuće editore prikaza i na kraju kreirane property-je dodati *Sheet.Set-u* koji će biti prikazan u *Properties* prozoru. Za prikaz jednostavnih tipova podataka kao što je *String* nije potrebno dodavati posebne *PropertyEditor-e*. U slučajevima kada prikazujemo složene tipove podataka potrebno je da naznačimo specifičan editor za taj tip parametra, kako što smo uradili na primeru input, output ili layers propertise-a. Za ove parametre kreirani su posebni editori nasleđivanjem klase *PropertyEditorSupport*. To su klase *TransferFunctionEditor*, *OutputNeuronsEditor, LayerEditor, InputNeuronsEditor*. Da bi vrednosti parametara bile preuzete potrebne su odgovarajuće get metode u *NeuralNetwork* klasi. Na slici 26. prikazan je deo koda klase *NeuralNetworkNode* sa fokusom na implementaciju metode *createSheet*().

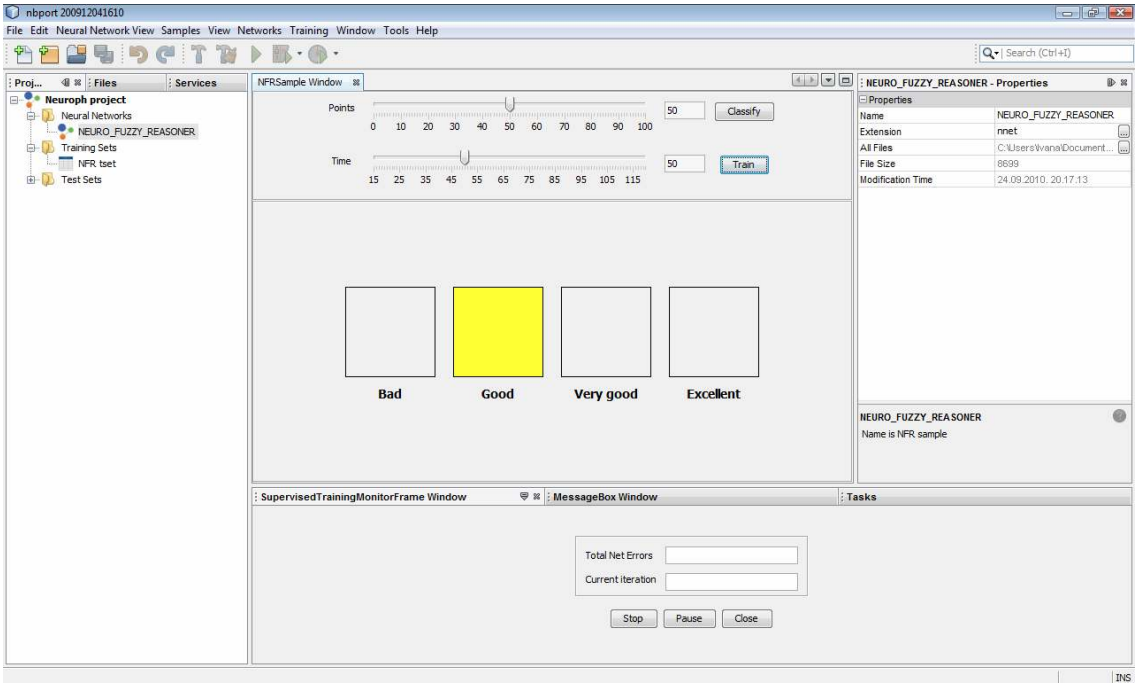
```
public class NeuralNetworkNode extends AbstractNode implements
PropertyChangeListener, LookupListener {
    public NeuralNetworkNode(NeuralNetwork neural) {
        super(new NeuralNetworkNodeChildren(),
Lookups.singleton(neural));
    }
    @Override
    public Image getIcon(int type) {... }
    @Override
    public Image getOpenedIcon(int i) {... }
  @Override
    public Action[] getActions(boolean popup) {...}
   @Override
    protected Sheet createSheet() {
        Sheet sheet = Sheet.createDefault();
        Sheet.Set set = Sheet.createPropertiesSet();
```

```
        NeuralNetwork obj = getLookup().lookup(NeuralNetwork.class);
        try {
            Property type = new PropertySupport.Reflection(obj,
NeuralNetworkType.class, "getNetworkType", null);
            PropertySupport.Reflection input = new
PropertySupport.Reflection(obj, LearningRule.class,
"getLearningRule", null);
            PropertySupport.Reflection output = new
PropertySupport.Reflection(obj, LearningRule.class,
"getLearningRule", null);
            PropertySupport.Reflection layers = new
PropertySupport.Reflection(obj, LearningRule.class,
"getLearningRule", null);
            type.setShortDescription("Neural Network Type");
            input.setPropertyEditorClass(InputNeuronsEditor.class);

output.setPropertyEditorClass(OutputNeuronsEditor.class);
            layers.setPropertyEditorClass(LayerEditor.class);
            type.setName("type");
            input.setName("InputNeurons");
            output.setName("OutputNeurons");
            layers.setName("Layers");
            set.put(type);
            set.put(input);
            set.put(output);
            set.put(layers);
        } catch (NoSuchMethodException ex) {
            ErrorManager.getDefault();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        sheet.put(set);
        return sheet;
    }
    public void propertyChange(PropertyChangeEvent evt) {
       if ("name".equals(evt.getPropertyName())) {
            this.fireDisplayNameChange(null, getDisplayName());
        }
        if ("someOther".equals(evt.getPropertyName())) {
            this.fireDisplayNameChange(null, getDisplayName());
        }
    }}
```

Slika 26. Segment koda klase NeuralNetworkNode

Kao rezultat kreiranja noda za reprezentovanje fajlova neuronske mreže, noda za prezentovanje trening seta i sinhronizovanja sa properties prozorom izgled aplikacije je prikazan slikom 27. U primeru je kreirana mreža I trening set primera "*Neuro Fuzzy Reasoner*".



Slika 27. Izgled aplikacije nakon kreiranja nodova i sinhronizovanja sa properties prozorom

Kreiranje Navigatora

U aplikaciji "*Neuroph*" za implementaciju navigatora je korišćena *TopComponent* klasa. Klasa *TopComponent* je deo *Windows API-a*, i predstavlja osnovnu komponentu za vizuelni prikaz u NetBeansu. *Navigator* je *singleton* komponenta. Kreiranje *TopComponent*-e moguće je pomoću *Window Component wizard*-a. U ovom slučaju automatski će biti kreiran kod za implementaciju klase kao singltona i biće kreirani sledeći fajlovi: NavigatorClassTopComponent.java, NavigatorClassTopComponentSettings.xml i NavigatorClassTopComponentWstcref.xml koji su odgovorni za otvaranje prozora, evidentiranje pozicije i omogućavanje premeštanja prozora na aplikaciji. Ovi fajlovi su registrovani *u layer.xml* fajlu.

Klasa *NavigatorClassTopComponent* će nasleđivati *TopComponent* klasu i implementirati *LookupListener*, *ExplorerManager.Provider* interfejse. Pomoću *LookupListener-a* mogu se pratiti trenutno selektovani fajlovi, ukoliko se radi o fajlu tipa NeuralNetwork kreiraće se prikaz pomoću *ExplorerManager.Provider-a*. Za svaki fajl neuronske mreže u *Navigator-u* će biti prikazani svi slojevi neurona I neuroni iz svakog sloja. Da bi se prikazali slojevi i neuroni moramo kreirati nodove. Kreiranjem nodova mogu se prikazati parametari slojeva i neurona u *properties* prozoru.

```
public final class NavigatorClassTopComponent extends TopComponent
implements LookupListener, ExplorerManager.Provider {

    private static NavigatorClassTopComponent instance;
    private static final String PREFERRED_ID =
"NavigatorClassTopComponent";
    private final ExplorerManager mgr=new ExplorerManager();
```

```
    public NavigatorClassTopComponent() {
        initComponents();

setName(NbBundle.getMessage(NavigatorClassTopComponent.class,
"CTL_NavigatorClassTopComponent"));

setToolTipText(NbBundle.getMessage(NavigatorClassTopComponent.class,
"HINT_NavigatorClassTopComponent"));
        associateLookup(ExplorerUtils.createLookup(mgr,
getActionMap()));
        setName("NeuralNetwork - Navigator");
    }
    private void initComponents() {…}

    public static synchronized NavigatorClassTopComponent
getDefault() {
        if (instance == null) {  instance = new
NavigatorClassTopComponent();  }
        return instance;
    }
    public static synchronized NavigatorClassTopComponent
findInstance() {
        TopComponent win =
WindowManager.getDefault().findTopComponent(PREFERRED_ID);
        if (win == null) {

Logger.getLogger(NavigatorClassTopComponent.class.getName()).warning
(
                    "Cannot find " + PREFERRED_ID + " component. It
will not be located properly in the window system.");
            return getDefault();
        }
        if (win instanceof NavigatorClassTopComponent) {
            return (NavigatorClassTopComponent) win;
        }

Logger.getLogger(NavigatorClassTopComponent.class.getName()).warning
(
                "There seem to be multiple components with the '" +
PREFERRED_ID
                + "' ID. That is a potential source of errors and
unexpected behavior.");

        return getDefault();
    }

    @Override
    public int getPersistenceType() {… }

    @Override
    public void componentOpened() {
        result =
Utilities.actionsGlobalContext().lookupResult(NeuralNetwork.class);
        result.addLookupListener(this);
        resultChanged(new LookupEvent(result));
```

```
        resultts =
Utilities.actionsGlobalContext().lookupResult(TrainingSet.class);
        resultts.addLookupListener(this);
        resultChanged(new LookupEvent(resultts));
    }

    private Result<NeuralNetwork> result;
    private NeuralNetwork activenn;
    private Result<TrainingSet> resultts;
    private TrainingSet activets;

    public void resultChanged(LookupEvent le) {
        Lookup.Result localresult = (Result) le.getSource();
        Collection<Object> coll = localresult.allInstances();
        if (!coll.isEmpty()) {
            for (Object nnn : coll) {
                if (nnn instanceof NeuralNetwork) {
                    activenn = (NeuralNetwork) nnn;
                   createView();
                   } } }

public void createView() {
        NeuralNetworkNode nn = new NeuralNetworkNode(activenn);
        Node[] nodvec = null;
        String name = nn.result.getName();
           if (activenn.getLayers().size() != 0) {
                nodvec = new Node[activenn.getLayers().size()];
                for (int i = 0; i < activenn.getLayers().size();
i++) {
                    Layer layer = activenn.getLayers().get(i);
                    Node nlayer = new LayerNode(layer);
                    nlayer.setName("layer " + String.valueOf(i +
1));
                    Node neuronnode = new
NeuronNode((layer.getNeuronAt(0)));
                    neuronnode.setName(layer.getNeurons().size() + "
neurons");
                    nlayer.getChildren().add(new
Node[]{neuronnode});
                    nodvec[i] = nlayer;
                }
                nn.setName(name);
                mgr.setRootContext(nn);
                mgr.getRootContext().getChildren().add(nodvec);
            }
    }

    public ExplorerManager getExplorerManager() {… }
    String name;
}
```

Slika 28. Segment koda klase NavigatorClassTopComponent.

Kao što smo spomenuli klase koje nasleđuju ***TopComponent*** klasu implementirane su kao *singleton*-i. Pozivanje jedinstvene instance obezbeđeno je pomoću metoda *getDefault()* i

findInstance(). Metoda *findInstance()* pomoću *WindowManager*-a pretražuje da li je instancirana *TopComponent-a* naziva klase, u ovom slučaju *„NavigatorClassTopComponent“*, u slučaju da nije, poziva se *getDefault()* metoda i privatnim konstruktorom klase instancira se komponenta. Prilikom nasleđivanja klase *TopComponent* potrebno je redefinisati metode *componentOpened()* i *componentClosed()*. One definišu ponašanja komponente pri otvaranju i zatvaranju.

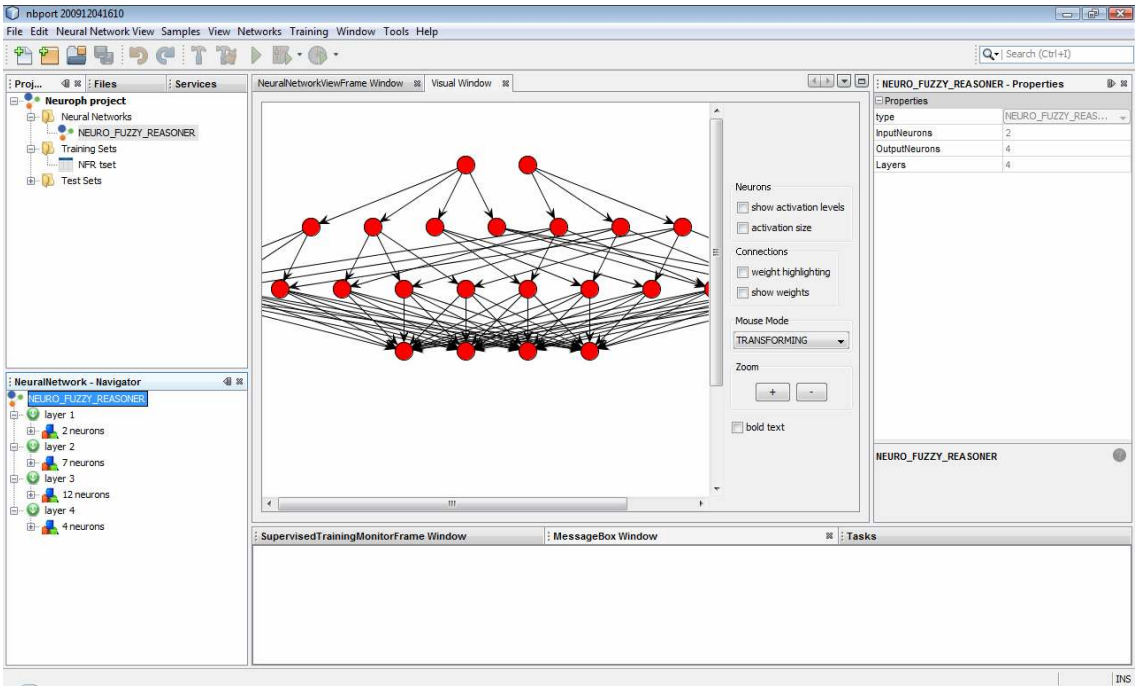
Implementiranjem *LookupListener-a* moguće je pratiti sve promene selektovanih fajlova. Redefinisanjem metode *resultChanged()* ispitujemo da li je selektovani fajl tipa neuronska mreža. Ako jeste kao *rootContext ExplorerManager*-a postavlja se selektovani fajl. Hijerarhijski prikaz u *Navigator-u* sastojaće se još od slojeva (Layer-a) neuronske mreže i broja neurona u svakom sloju. U metodi *createView* kreiraju se za datu neuronsku mrežu ispituje se broj slojeva i neurona i kreiranju se odgovarajuću nodovi (LayerNode i NeuronNode). U *Properties* prozoru za neurone se prikazuje transfer funckcija, greška, ulaz i izlaz. Samo je Transfer funkcija složeni tip podatka i za nju se kreira novi *PropertyEditor*, tj. *TransferFunctionEditor*.

Da bi *TopComponent-a* bila prikazana neophodno je registrovati fajl u *layer.xml-u*. Navigator će se u aplikaciji pojaviti tek pri kreiranju ili otvaranju prve neuronske mreže, do tada neće biti vidljiv. Ove akcije je potrebno registrovati u *Window* folderu. Takođe, komponentu je potrebno registrovati i u meniju aplikacije,tj. u *window* opcijama umesto standardnog navigatora. Korekcije u *layer.xml* su prikazane na slici 29.

```
  <folder name="Actions">
    <folder name="BpelNodes"/>
        <folder name="Window">
            <file name="org-neuroph-netbeans-ide-navigator-
NavigatorClassAction.shadow">
                <attr name="component"
methodvalue="org.neuroph.netbeans.ide.navigator.NavigatorClassTopCom
ponent.findInstance"/>
                <attr name="displayName"
bundlevalue="org.neuroph.netbeans.ide.navigator.Bundle#CTL_Navigator
ClassAction"/>
                <attr name="instanceCreate"
methodvalue="org.openide.windows.TopComponent.openAction"/>
            </file>
        </folder>
    </folder>
<folder name="Menu">
            <file name="NavigatorClassAction.shadow">
                <attr name="originalFile"
stringvalue="Actions/Window/org-neuroph-netbeans-ide-navigator-
NavigatorClassAction.instance"/>
            </file>
</folder>
```

Slika 29. Segment koda potrebnog za registraciju Navigatora u layer.xml fajlu.

Izgled aplikacije nakon implementacije navigatora za neuronske mreže i properties prozora za layer i neuron elemente prikazan je na slici 30.



Slika 30. Završni izgled aplikacije

5.3.1 Rezime koraka za integraciju fajla tipa NeuralNetwork sa elementima korisničkog interfejsa

1. **Integracija sa Properties prozorom**

- **Modifikovanje NeuralNetworkNode klase**
  Redefinisanjem metode *createSheet()* klase *NeuralNetworkNode* obezbeđuje se prikaz parametara u properties prozoru. Najpre je potrebno kreirati Sheet objekat, zatim *Sheet.Set* objektu dodati *Property* objekte koji predstavljaju parametre od značaja za dati tip fajl. U sličaju *NeuralNetwork* fajla to će biti *Property* koji predstavljaju tip mreže, ulazne neurone, izlazne neurone, i broj slojeva.
- **Kreiranje PropertyEditora**
  Za *property-e* ulaznih, izlaznih neurona i slojeva potebno je kreirati nove *PropertyEditore* pošto predstavaljaju složene tipove podataka. To će biti *InputNeuronEditor*, *OutputNeuronEditor* i *LayerEditor* klase. Sve one će nasleđivati *PropertyEditorSupport* klasu. Redefinisanjem metode *getAsText()* vraća se string vrednost parametra koji treba da se prikaze.

2. **Kreiranje Navigatora**

- **Kreiranje TopComponent klase.**
  Desnim klikom na modulu IDE biramo opciju *New > ModuleDevelopment > WindowComponent*. U „*BasicSettings*“ panelu potebno izabrati poziciju prozora, izabraćemo opciju navigator. U panelu „*Name, Icon and Location*“ podešava se naziv ikona i paket za smestanje klase. Generisani su fajlovi *NavigatorTopComponent.java*, *NavigatorTopComponent.form*, *NavigatorTopComponentSettings.xml*, *NavigatorTopComponentWstcref.xml*. U xml fajlovima zabeležena su podešavanja vezna za poziciju i aktivnost komponente pri paljenju aplikacije, tu je i moguće vršiti promene podešavanja.
- **Modifikovanje NavigatorTopComponent klase.**
  Klasa *NavigatorTopComponent* nasleđuje klasu *TopComponent*. Potrebno je redefinistati *componentOpened()* i *componentClosed ()* metode koje definištu ponašanje komponente prilikom otvaranja i zatvaranja.
- **Povezivanje sa ExplorerManager-om**
  Da bi klasa *NavigatorTopComponent* mogla da obezbedi hijerarhijski prikaz sadržaja aktivne mreže mora da implementira *ExplorerManager.Provider* klasu. Pored toga u konstruktoru *NavigatorTopComponent* klase potrebno je pomoću *associateLookup()* metode pridružiti *ExplorerManager* objekat *TopComponenti*.
- **Povezivanje sa LookupListener-om**
  NavigatorTopComponent klasa implementira interfejs *LookupListener*. Redefinisanjem metode *resultChanged()* proverava se selektovan fajl. Ukoliko se radi o fajlu neuronske mreže taj fajl se podešava kao *rootContext explorer managera*. Za prikaz *Layer* i *Neuron* elemenata neuronske mreže potrebno je kreirati nodove kako bi bili prikazani u navigatoru.
- **Kreiranje LayerNode i NeuronNode klasa.**
  Obe klase će nasleđivati *AbstracNode* klasu. Podešavanjem metoda *getIcon(), setName()* i *openIcon()* prilagođava se prikaz elemata u *Project* prozoru. Redefinisanjem metode *createSheet()* za parametre *Layer* i *Neuron* klasa kreiraju se *Property* objekti. Dodavanjem *Sheet.Set-u* povezan je prikaz parametara u *Properties* prozoru.

6. EVALUACIJA

Kreiranjem specijalizovanog razvojnog okruženja za *Neuroph* aplikaciju pomoću NetBeans platforme obezbeđeno je sledeće:

1. projekat tipa Neuroph sa jasnom strukturom u kome su smešteni svi fajlovi značajni za rad sa Neuroph aplikacijom;
2. jedinstven način čuvanja i prezentovanja podataka koji su specifični za neuronske mreže;
3. standardizovan način kreiranja fajlova – pomoću *new project* ili *new file wizarda*;
4. osnovne akcije za manipulaciju sa fajlovima – kopiranje, brisanje, preimenovanje, premeštanje, otvaranje, čuvanje i zatvaranje;
5. rad sa projektima i fajlovima neuroph aplikacije je kao i sa drugim standardnim tipovima podataka. Time je korisniku olakšano korišćenje aplikacije jer je rad intuitivan i jednostavan;
6. prilagođene elemente korisničkog interfejsa za prikaz i editovanje specifičnih fajlova- *Navigator* i *Properties* prozor.

Kreiranjem specijalizovanog razvojnog okruženja za Neuroph aplikaciju pomoću NetBeans Platforme korisniku je omogućen lakši rad sa aplikacijom.

Na tom primeru pokazano je kako je korišćenjem NetBeans Platforme moguće kreirati aplikacije jasno definisane strukture. Time je moguće jednostavnije prilagođavanje programa budućim zahtevima. Neke prednosti NetBeans platforme iskorišćene u kreiranju specijalizovanog razvojnog okruženja su sledeće:

- NetBeans platforma obezbeđuje jednostavno rešenje za kreiranje, upravljanje, manipulisanje i prezentovanje podataka. Rešenje je obezbeđeno pomoću *File System*, *Data System* i *Node System API*-a. Pošto se svaki od njih nalazi na zasebnom apstrakcionom sloju, jasno su podeljene odgovornosi i uloge koje imaju u radu sa fajlovima. Ako u daljem razvoju bude potrebno izmeniti izgled nekog fajla u korisničkom interfejsu jasno je da će se te prepravke praviti u klasi koja predstvalja njegovu *Node* reprezentaciju. Time će se izmenom u jednoj klasi promeniti sva pojavljivanja fajla u korisničkom interfejsu, npr. u Project, Properties ili Navigator prozoru.

- Jednostavna izmena i prlagođavanje elemenata aplikacije. Izmenom konfiguracionih fajlova *File System API*-a, najčešće *layer.xml* fajla, moguće je prilagoditi podešavanja fajlova, komponenti ili prozora trenutnim zahtevima aplikacije.

- NetBeans platforma nudi osnovne funkcionalnosti window komponenti kao što su miniziranje/maksimiziranje, dock/undock, premeštanje pozicije prozora aplikacije. Programiranje ovih funkcionalnosti bi oduzelo dosta vremena. Korišćenjem već postojećih funkcionanosti korisničkog interfejsa moguće je posvećivanje više vremena razvoju poslovne logike aplikacije.

- Lako upravnjanje *data* objektima. Pomoću *Explorer & Property Sheet API*-a obezbeđeno je nekoliko *Swing* komponenti za prikaz nodova. Zahvaljujući ovom *API*-u izmenom jedne linije koda moguće je promeniti prikaza modela nodova na vrlo brz i efikasan način. Promena prikaza koji je kreiran pomoću standardnih Swing komponenti trajao bi znatno oduže, Npr. JTree model se potpuno razlikuje od JList modela. Prebacivanje prikaza sa jednog na drugi bi značilo ponovno pisanje modela.

- Jednostavno dodavanje ili uklanjanje funkcionalnosti aplikacijie. Zahvaljujući Sistemu *FileSystem* sve funkcionalnosti koje su dostupne u aplikacije omogućene su preko različitih modula. Uklanjanjem ili dodavanjem određenih modula lako je upravljati funkcionalnostim aplikacije.

- Efikasno upravljanje selektovanim fajlovima. Prilikom kreiranje specijalizovanih razvojnih okruženja dosta pažnje teba posvetiti akcijama koje zavise od trenutnog konteksta aplikacije, tj. od selektovanih podataka. Praćenje aktivnog projekta ili fajla pomoću NetBeans platforme vrlo je jednostavno zahvaljujući *Lookup* mehanizmima.

Pored NetBeans Platforme alternativa za kreiranje *Rich Client* aplikacija *je Eclipse RCP* (*Rich Client Platform*). Dok se Eclipse RCP kroz SWT i Jface biblioteke zasniva na sopstvenim idiomima i konceptima, NetBeans platforma se kompletno oslanja na standardni *Java API* kroz AWT i Swing integrišući koncepte i komponente iz *Java Standard Edition*.

SWT (The Standard Widget Toolkit) koristi različite osnovne biblioteke na raličitim platformam. Zbog ove osobine neke funkcionalnosti koje su podržane na jednoj platformi ne moraju biti podržane na drugoj, ili imaju različita ponašanja. Npr. elementi grafičkog korisničkog interfejsa ne moraju izgledati isto na *Windows* i *Macintosh* operativnim sistemima. Da bi se kreirala aplikacija koja je predviđena za rad na različitim platformama potrebno je više vremena i prilagođavanja. NetBeans platforma je izgrađena kompletno na *Javi*. Window sistem je izgrađen samo pomoću komponenti *Swing* biblioteke.

Kreiranje specijalizovanog razvojnog okruženja bez korišćenja *Rich Client Platform framework-a* bi bilo znatno otežano. Kreiranje svih elemenata grafičkog intefejsa bi oduzelo mnogo vremena. Za svaku komponentu bi bilo potrebno implementirati mogućnosti editovanja i manipulacije koje su pomoću NetBeans platforme obezbeđene. Takođe, za kreiranje i regostrovanje novih tipova projekata ili fajlova ti trebalo obezbediti nov mehanizam za skeniranje fajlova na disku i prepoznavanje reigistrovanih tipova. Ovaj mehanizam je u NetBeans platformi omogućen pomoću *LoaderPool-a.* NetBeans platforma nudi wizarde pomoću kojih je standardizovan način kreiranja novih projekata i fajlova. Kreiranje sličnih komponenti samo pomoću *Swing* komponenti bi zahtevalo dosta vremena da bi postignute funkcionalnosti bile kao kod komponenti NetBeans platforme.

Zahvaljujući ovim i još mnogim drugim osobinama koje izlaze iz okvira ovog rada, NetBeans platforma nudi pouzdanu osnovu za razvoj aplikacije koja je stabilna, koja ima sve neophodne elemente za rad sa podacima i koja je pogodna za dograđivanje.

7. ZAKLJUČAK

U ovom poglavlju dat je pregled postignutih rezultata kao i mogući dalji pravci razvoja i primene aplikacije.

Osnovni doprinos

Svrha rada bila je da se pokaže proces razvoja specijalizovanog razvojnog okruženja za određenu namenu korišćenjem NetBeans platforme. U okviru *Neuroph frameworka* kreirana je aplikacija koja predstavlja razvojno okruženje za rad sa neuronskim mrežama.
U radu su opisane sve faze razvoja aplikacije ovog tipa korak po korak, i time sam rad predstavlja svojevrsno uputstvo za razvoj razvojnih okruženja ovog tipa, koje se može primeniti za razvoj osnove raznih drugih razvojnih okruženja.

Mogućnosti primene

Specijalizovana razvojna okruženja objedinjuju sve važne funkcionalnosti u jednu namensku celinu i korisniku olakšavaju rad.
Razvijena aplikacija za rad sa neuronskim mrežama omogućava kreiranje specijalizovanih projekata ovog tipa, koji sadrže sve potrebne elemente projekata ovog tipa (neuronske mreže i skupove podataka za trening i testiranje), kao i manipulaciju tim elementima.
Metode date u ovom radu mogu se primeniti za kreiranje specijalizovanih razvojnih okruženja u drugim oblastima.

Dalji pravci razvoja

Razvijena aplikacija predstavlja odličnu osnovu za dalji razvoj dodatnih funkiconalnosti, koje se takođe mogu zasnivati na standardnim elementima NetBeans platforme. Zahvaljujući modularnj arhitekturi NetBeans platforme, moguće je realizovati razne dodatke u vidu dodatnih modula i *plugin-ova*.
Dalji koraci u razvoju *Neuroph* aplikacije pomoću NetBeans platforme mogu biti:

- razvoj funkcionalnosti za podršku celom procesu treniranja i testiranja neuronskih mreža;
- integracije sa Java razvojnim okruzenjiem tj. Java IDE, čime bi se kreiralo kompletno Java okruženje za rad sa neuronskim mrežama, koje omogućava potpunu kontrolu nad svim elemenima *Neuroph frameworka* i istovremeni rad na nivou grafičkog korisničkog interfejsa i izvornog koda.

8. LITERATURA

[1] Heiko Böck, „The Definitive Guide to NetBeans™ Platform“,Apress, 2009.

[2] Tim Boudreau, Jaroslav Tulach, Geertjan Wielenga, „Rich Client Programming Plugging into the NetBeans Platform“, Prentice hall, 2007.

[3] Jürgen Petri, „NetBeans Platform 6.9 Developer's Guide“, Packt Publishing, 2010.

[4] Adam Myatt, Brian Leonard, Geertjan Wielenga, „Pro NetBeans™ IDE 6 Rich Client Platform Edition“, Apress, 2008.

[5] FileSystems API Javadoc, http://bits.netbeans.org/dev/javadoc/org-openide-filesystems/org/openide/filesystems/doc-files/api.html#diagram_fileobject

[6] DataSystem API Javadoc, http://bits.netbeans.org/dev/javadoc/org-openide-loaders/org/openide/loaders/doc-files/api.html

[7] NodeSystem API Javadoc, http://bits.netbeans.org/dev/javadoc/org-openide-nodes/org/openide/nodes/doc-files/api.html

[8] ExplorerSystem API Javadoc, http://bits.netbeans.org/dev/javadoc/org-openide-explorer/org/openide/explorer/doc-files/api.html

[9] ProjectSystem API javadoc, http://bits.nbextras.org/dev/javadoc/org-netbeans-modules-projectapi/org/netbeans/api/project/package-summary.html

[10] Declarative MIME type Resolvers, http://bits.netbeans.org/dev/javadoc/org-openide-filesystems/org/openide/filesystems/doc-files/HOWTO-MIME.html

[11] New project template, NetBeans Platform Learning Trail, http://wiki.netbeans.org/BuildSystemHowTo#How_to_Write_a_Project_Type_.28Generally.29

[12] New project type, NetBeans Platform Learning Trail, http://platform.netbeans.org/tutorials/nbm-projecttype.html

[13] New project sample, NetBeans Platform Learning Trail, http://platform.netbeans.org/tutorials/nbm-projectsamples.html

[14] File Type Integration, NetBeans Platform Learning Trail, http://platform.netbeans.org/tutorials/nbm-filetype.html