MRS

LAB 01

MRSLab 01 – Metodologija Razvoja Softvera – Vežba 01

ULM i Ciljevi UML-a
Strukturni dijagrami
Dijagrami ponašanja
Power Designer
Modelovanje u Power Designer-u

1. UML

UML je standardni jezik za izradu specifikacija, vizualizaciju, kreiranje i dokumentovanje delova sistema. UML (Unified Modeling Language) je kreiran od strane Object Management Group-e (OMG). Prva specifikacija UML 1.0 je izašla u januaru 1997. godine. On se razlikuje od drugih standardnih programskih jezika poput C++, Java, itd, jer predstavlja vizuelni jezik. Iako se UML uglavnom koristi za modelovanje softverskih sistema, on se može koristiti i za modelovanje drugih sistema.

UML nije tipičan programski jezik ali se može koristiti da se pomoću dijagrama generiše kod u različitim programskim jezicima. On je direktno povezan sa objektno orjentisanom analizom i dizajnom i dovoljno moćan da prikaže bilo koji koncept koji u njima postoji.

1.1. Ciljevi UML-a

Slika vredi hiljadu reči. Ova poznata misao se u potpunosti uklapa u suštinu diskusije o UML-u. Objektno orjentisani koncepti su predstavljeni dosta ranije od UML-a, pa u to vreme nije postojala standardna metodologija organizacije objektno orjentisanog razvoja sistema. UML se pojavio upravo da bi odgovorio na ove zahteve.

Postoji više ciljeva koji su doveli do razvoja UML-a ali je najvažniji da se definiše jezik za modelovanje opšte namene koji mogu koristiti svi koji se bave modelovanjem a koji će biti jednostavan za razumevanje i upotrebu.

UML dijagrami se ne kreiraju samo za ljude koji se bave kreiranjem softvera već i za poslovne korisnike, obične ljude i bilo koga ko želi da razume kako funkcioniše neki sistem. Ovaj sistem može ali i ne mora biti softverski.

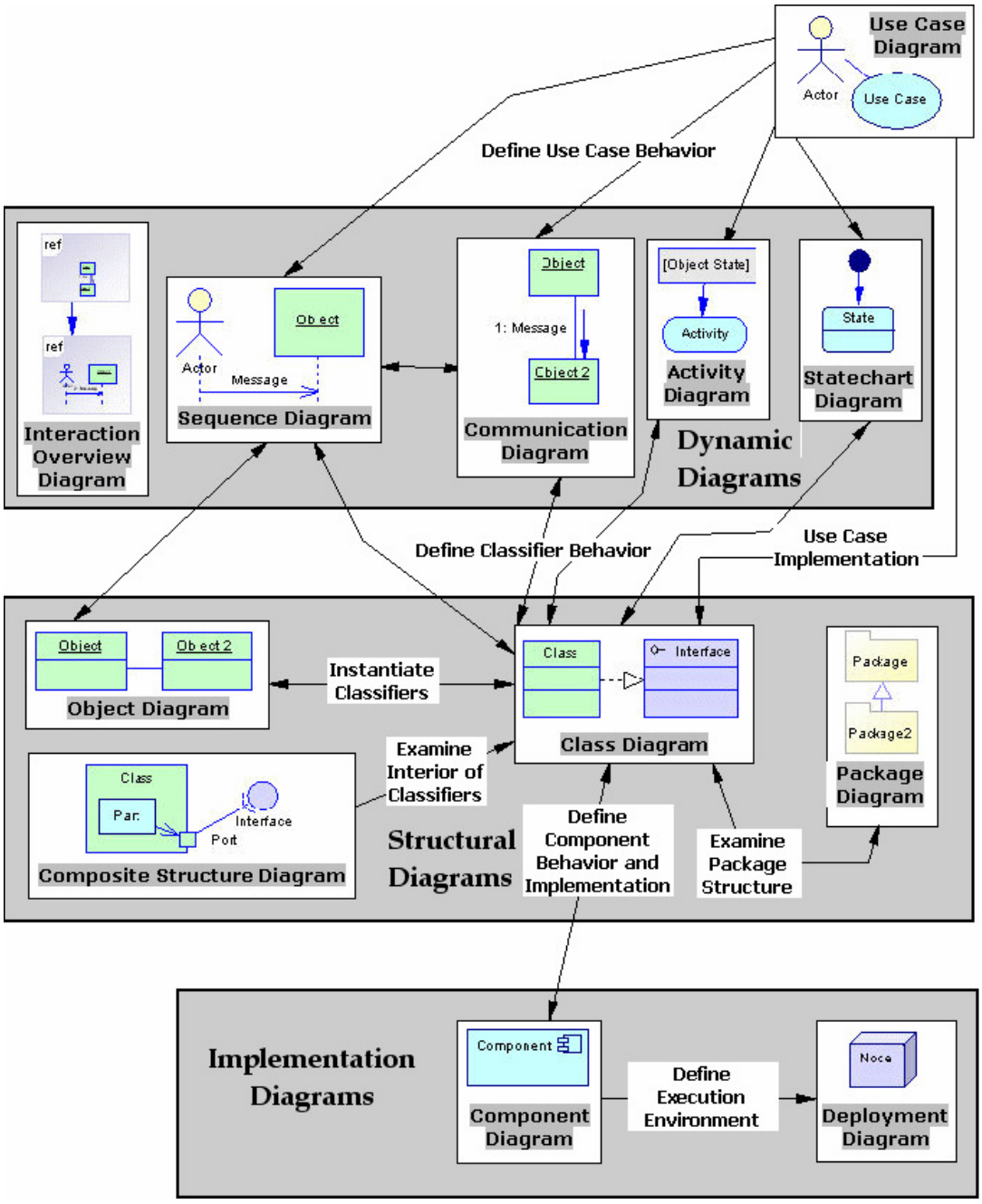
Kao zaključak može se reći da UML dijagrami predstavljaju jednostavan mehanizam za modelovanje različitih praktičnih sistema u današnjem kompleksnom okruženju.

2. UML dijagrami

Bilo koji kompleksni sistem se može najbolje razumeti tako što ćemo ga prikazati putem dijagrama ili slike. Ovi dijagrami imaju uticaj na lakše razumevanje. Jedan dijagram nije dovoljan da bi pokrio sve funkcionalnosti sistema. Zbog toga u UML-u postoje različiti dijagrami pomoću kojih je moguće pokriti ove funkcionalnosti.

Postoje dve široke kategorije UML dijagrama koje se dele u svoje podkategorije. Ove kategorije dijagrama su:

- Strukturalni dijagrami (Structural diagrams) – u ovu grupu spadaju i Implementacioni Dijagrami (Implementation diagrams)
- Dijagrami ponašanja (Behavioral diagrams, Dynamic diagrams)



Slika broj 1. Uopštena podela UML dijagrama [1]

2.1. Strukturalni dijagrami

Strukturalni dijagrami predstavljaju statičke aspekte sistema. Oni su predstavljeni pomoću klasa, interfejsa, objekata, komponenti i čvorova. U ovu grupu dijagrama spadaju:

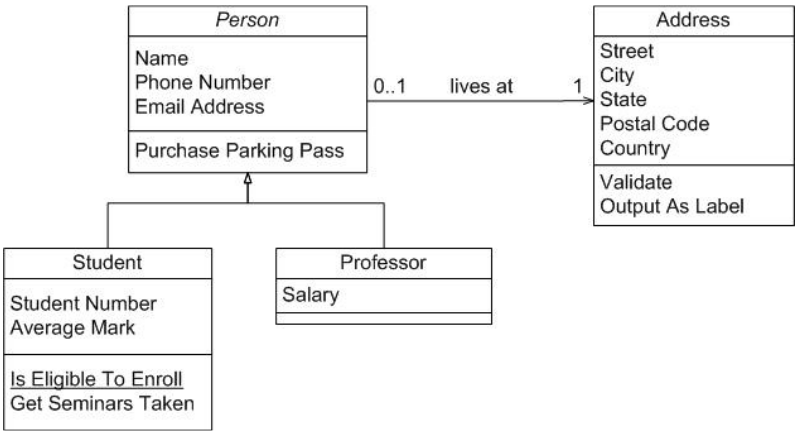
- Klasni dijagrami
- Dijagrami objekata
- Dijagrami komponenti (Implementation diagrams)
- Dijagrami rasporeda (Implementation diagrams)

2.1.1. Klasni dijagrami

Dijagrami klasa su statički dijagrami i predstavljaju statički pogled na aplikaciju. Oni se ne koriste samo za vizualizaciju, opisivanje i dokumentovanje različitih delova sistema, već i za konstruisanje izvršnog koda softverskih aplikacija.

Klasni dijagrami opisuju atribute i operacije klasa kao i ograničenja koja su postavljena pred sistem. Masovno se koriste u modelovanju objektno orjentisanih sistema jer predstavljaju jedine UML dijagrame koji se mogu direktno mapirati na objektno orjentisane jezike.

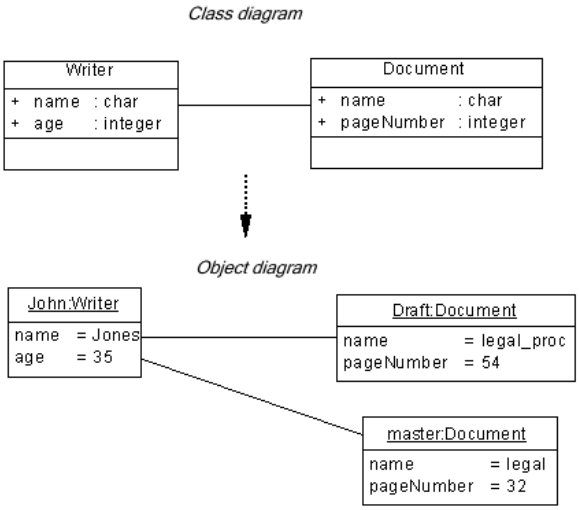
Ovi dijagrami prikazuju kolekcije klasa, interfejsa, asocijacija, saradnje i ograničenja.



Slika broj 2. Primer klasnog dijagrama [2]

2.1.2. Dijagrami objekata

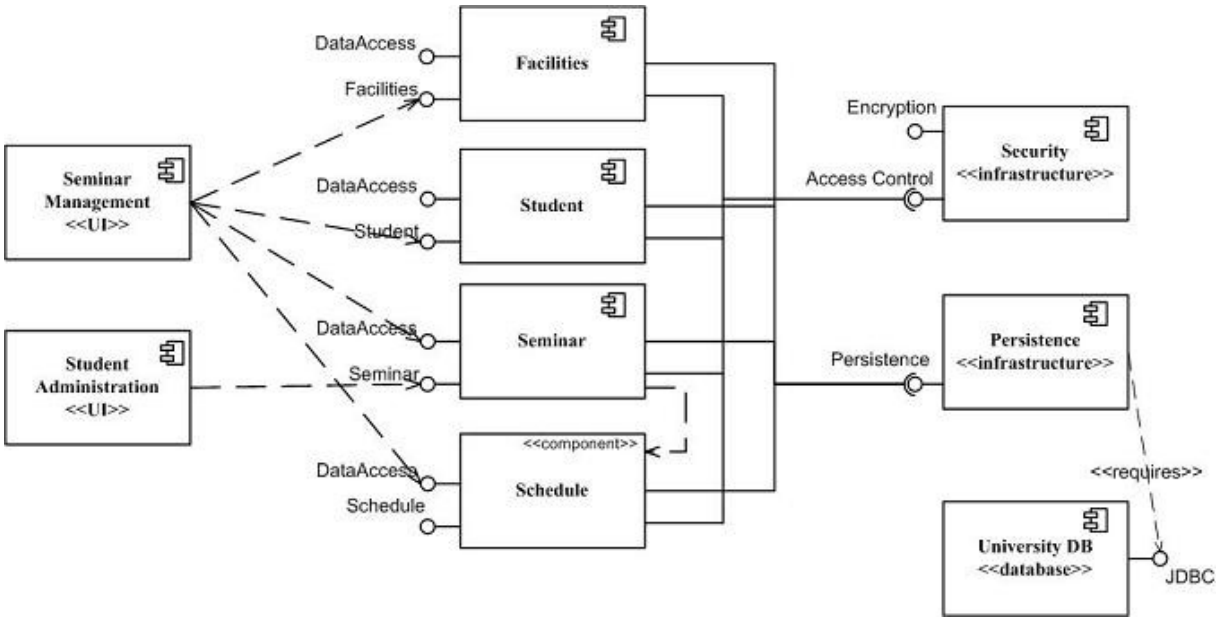
Dijagrami objekata su izvedeni iz klasnih dijagrama pa su objektni dijagrami zavisni od klasnih dijagrama. Objektni dijagrami predstavljaju instancu dijagrama klasa. Osnovni koncepti su slični i za klasni dijagram i za dijagram objekata. Dijagrami objekata takođe prikazuju statički pogled na sistem, ali ovaj statički pogled predstavlja sliku sistema u određenom momentu. Dijagrami objekata se koriste da iscrtaju skup instanci objekata i njihove međusobne veze.



Slika broj 3. Primer objektnog dijagrama [3]

2.1.3. Dijagram komponenti

Dijagrami komponenti su po svojoj prirodi i ponašanju drugačiji od klasnih dijagrama i dijagrama objekata. Oni se koriste za modelovanje fizičkih pogleda na sistem. Fizički aspekti su elementi poput izvršnih fajlova, biblioteka, dokumenata itd. koji predstavljaju sistem. Drugim rečima, dijagrami komponenti se koriste da prikažu organizaciju i odnose među komponentama u sistemu. Oni ne opisuju funkcionalnosti sistema već komponente koje se koriste da bi kreirali date funkcionalnosti.



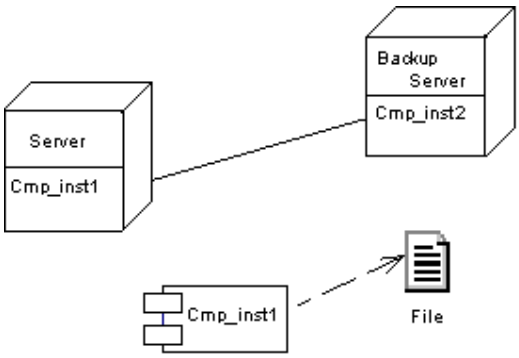
Slika broj 4. Primer dijagrama komponenti [4]

2.1.4. Dijagram rasporeda

Dijagrami rasporeda se koriste da prikažu topologiju fizičkih komponenti sistema, odnosno gde su razmeštene softverske komponente sistema. Oni se koriste da opišu statički prikaz rasporeda sistema. Ovi dijagrami se sastoje od čvorova i njihovih međusobnih odnosa.

Dijagrami komponenti i dijagrami rasporeda su blisko povezani. Dijagrami komponenti se koriste da opišu komponente a dijagrami rasporeda prikazuju kako su te komponente razmeštene po hardveru.

UML je dizajniran da se pretežno fokusira na softverske činioce sistema. Međutim, ova dva dijagrama su specijalni dijagrami koji se koriste za fokus na softverske komponente i hardverske komponente. Većina UML dijagrama se koristi za logičke komponente dok se dijagrami rasporeda koriste da bi se ukazalo na hardversku topologiju sistema.



Slika broj 5. Primer dijagrama rasporeda (Deployment diagram) [5]

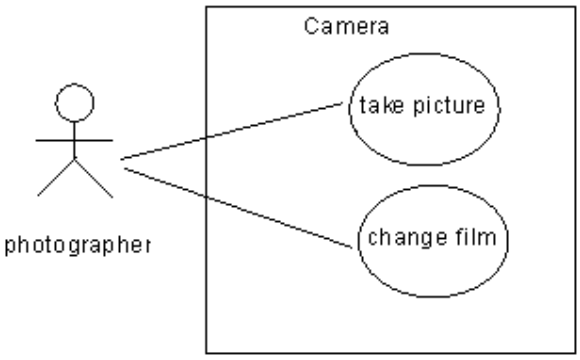
2.2. Dijagrami ponašanja

Svaki sistem može da ima dva aspekta: statički i dinamički. Na osnovu ovoga može se reći da se model smatra kompletnim kada su oba aspekta u potpunosti pokrivena. Dijagrami ponašanja prikazuju dinamičke aspekte sistema. Dinamičko ponašanje sistema predstavlja ponašanje sistema kada se on izvršava. U ovu grupu dijagrama spadaju:

- Dijagrami slučajeva korišćenja
- Sekvencijalni dijagrami
- Dijagrami saradnje
- Dijagrami stanja
- Dijagrami aktivnosti

2.2.1. Dijagrami slučajeva korišćenja

Svrha dijagrama slučajeva korišćenja je da prikažu dinamičke aspekte sistema. Međutim ova definicija je suviše opšta da bi opisala svrhu jer i drugi dinamički UML dijagrami (dijagrami aktivnosti, sekvencijalni dijagrami, dijagrami saradnje i dijagrami stanja) takođe imaju istu svrhu. Zbog toga ćemo sagledati određenu namenu ovih dijagrama koja ih razdvaja od drugih dijagrama za modelovanje dinamičkih osobina sistema.

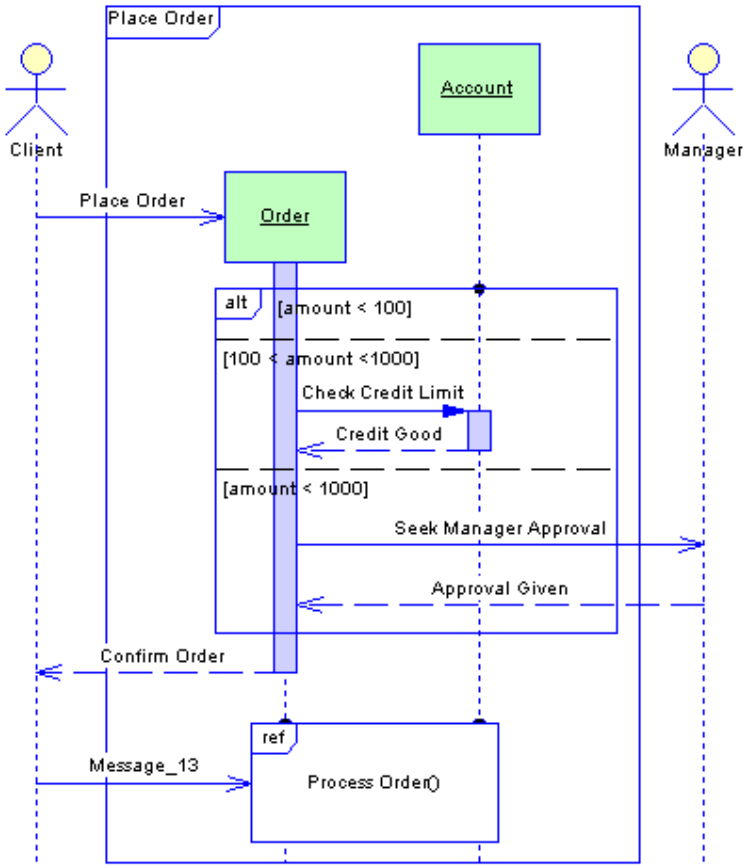


Slika broj 6. Primer dijagrama slučajeva korišćenja [6]

Dijagrami slučajeva korišćenja se koriste da se prikupe zahtevane funkcionalnosti sistema uključujući unutrašnje i spoljašnje uticaje. Dakle, kada se sistem analizira da bi se uvidele njegove funkcionalnosti, kreiraju se slučajevi korišćenja i identifikuju se učesnici. Oni se sastoje od učesnika, slučajeva korišćenja i njihovih međusobnih veza. Dijagram se koristi da modeluje sistem ili podsistem aplikacije. Jedan dijagram slučajeva korišćenja prikazuje određenu funkcionalnost sistema. Da bi se modelovao ceo sistem koristi se više ovih dijagrama.

2.2.2. Sekvencijalni dijagrami

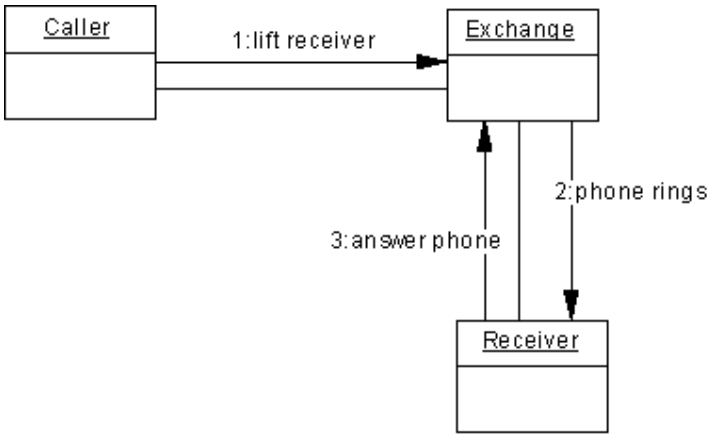
Sekvencijalni dijagram predstavlja tip dijagrama interakcije. Sekvencijalni dijagram prikazuje vremensku sekvencu poruka koje se šalju od jednog objekta ka drugom. On prikazuje redosled poruka. Svrha ovih dijagrama je da prikažu interakciju koja se dešava u sistemu. Interakcija u sistemu je veoma važna sa gledišta implementacije i izvršavanja.



Slika broj 7. Primer sekvencijalnog dijagrama [7]

2.2.3. Dijagrami saradnje

Dijagram saradnje takođe predstavlja tip dijagrama interakcije. Ovi dijagrami opisuju organizaciju objekata u sistemu koji učestvuju u razmeni poruka. Oni se sastoje od objekata i veza. Svrha dijagrama saradnje je slična sa svrhom sekvencijalnih dijagrama. Razlika je u tome što ovi dijagrami prikazuju organizaciju objekata i njihovu interakciju.

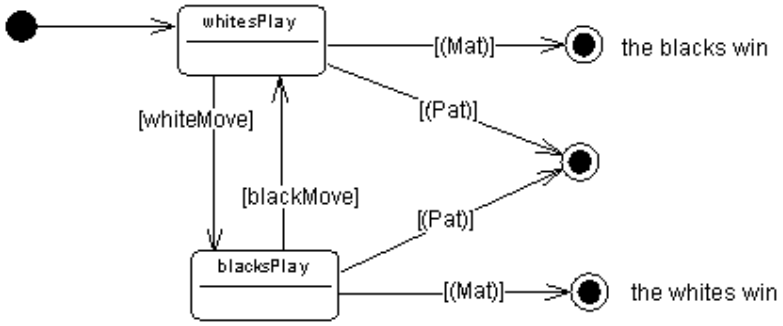


Slika broj 8. Primer dijagrama saradnje [8]

2.2.4. Dijagrami stanja

Dijagram stanja je jedan od pet UML dijagrama koji se koriste za modelovanje dinamičke prirode sistema. On definiše različita stanja objekta tokom njegovog životnog ciklusa. Ova stanja se mogu promeniti pomoću događaja. Dijagram stanja je koristan kod modelovanja reaktivnih sistema. Reaktivni sistem se može definisati kao sistem koji reaguje na spoljašnje ili unutrašnje događaje.

Dijagram stanja opisuje tok kontrole iz jednog stanja u drugo stanje. Stanja su definisana kao uslovi u kojima neki objekat postoji a koja se menjaju kada se neki događaj izvrši. Na osnovu ovoga se može reći da je najvažnija svrha dijagrama stanja da modeluju životni ciklus objekta od njegovog kreiranja do njegovog uništenja.

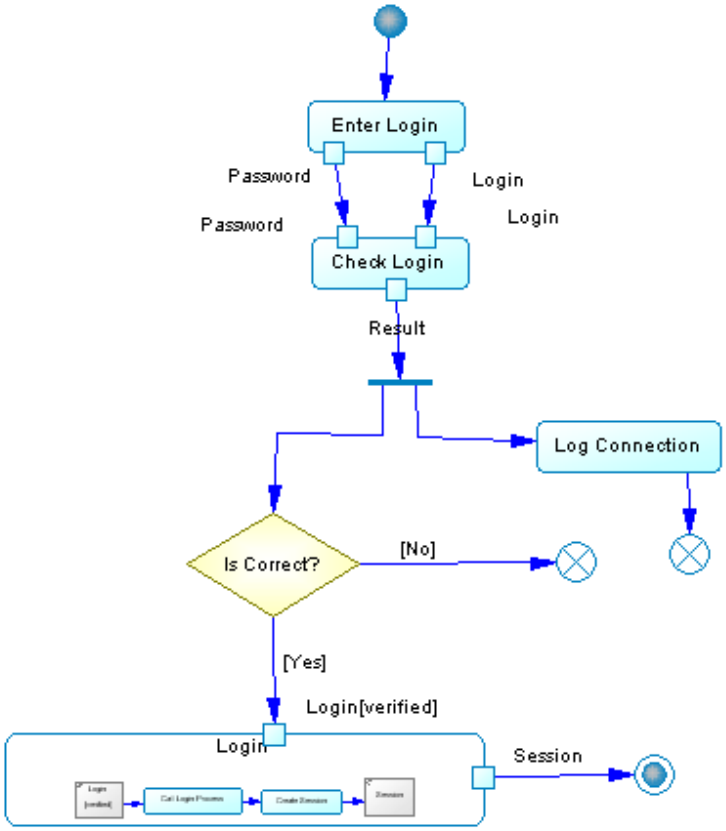


Slika broj 9. Primer dijagrama stanja [9]

2.2.5. Dijagrami aktivnosti

Dijagram aktivnosti je još jedan važan UML dijagram koji se koristi da opiše dinamičke aspekte sistema. Dijagram aktivnosti predstavlja dijagram koji prikazuje prelazak iz jedne aktivnosti u drugu aktivnost. Aktivnost se može opisati kao operacija sistema. Na osnovu ovoga možemo reći da kontrola prelazi iz jedne operacije u drugu. Ovaj prelazak može biti sekvencijalni, razgranat ili konkurentan. Dijagrami aktivnosti modeluju sve tipove kontrole toka koristeći različite elemente kao što je spajanje, razdvajanje, itd.

Osnovna svrha dijagrama aktivnosti je slična sa ostalim dijagramima koji modeluju dinamičko ponašanje sistema. Ostali dijagrami se koriste da prikažu tok poruka od jednog objekta do drugog dok se dijagrami aktivnosti koriste da prikažu prelaz iz jedne aktivnosti u drugu.



Slika broj 10. Primer dijagrama aktivnosti [10]

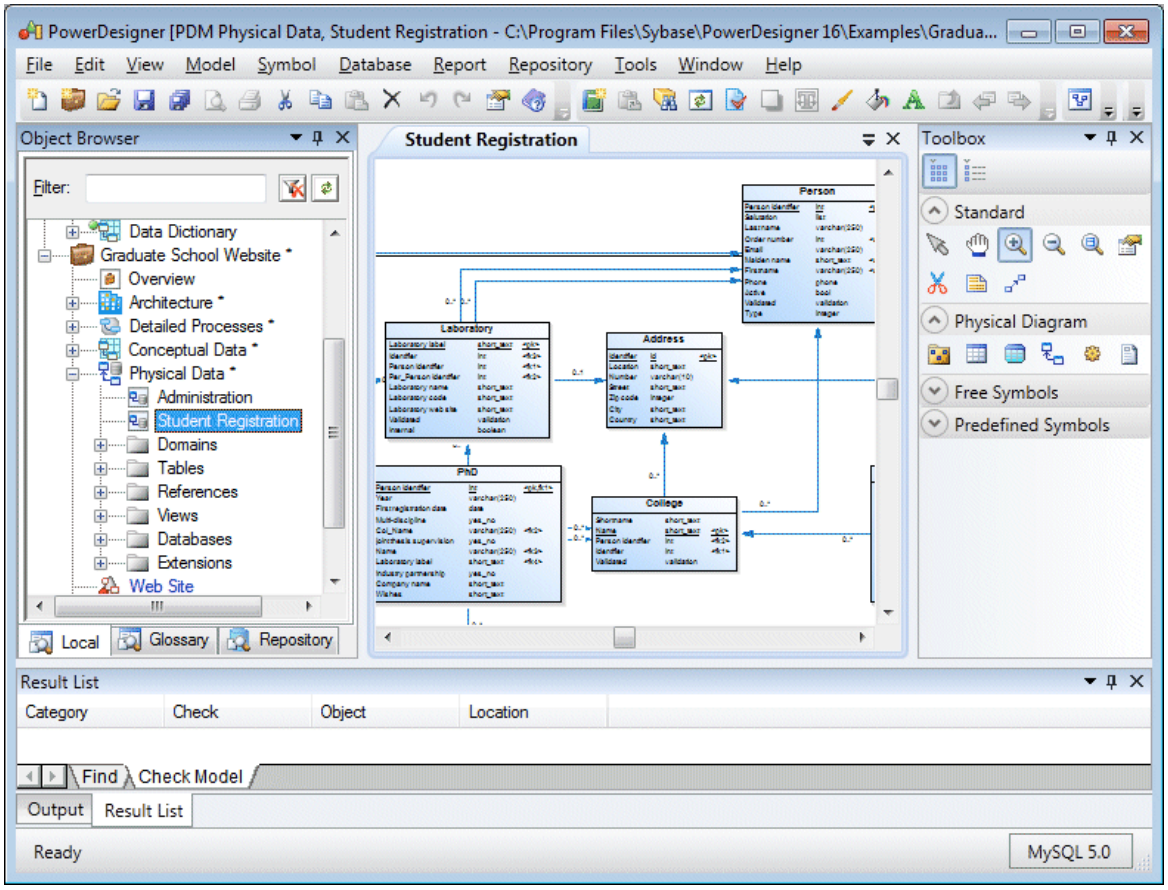
3. Power Designer

PowerDesigner predstavlja grafičko okruženje za modelovanje koje sadrži

- integrisano modelovanje kroz standardne metodologije i notacije
- automatsko generisanje koda kroz prilagodljive šablone
  - SQL (sa više od 50 podržanih DBMS)
  - Java
  - .NET
- moćne inženjerske mogućnosti za dokumentovanje i menjanje postojećih sistema
- prilagodljiva rešenja skladištenja (repository) sa visokim stepenom sigurnosti i mogućnostima praćenja razvoja po verzijama kao podrška višekorisničkom radu
- automatizovana mogućnost generisanja izveštaja
- proširivo radno okruženje koje omogućava dodavanje novih pravila, komandi i koncepata u postojeću metodologiju modelovanja i kodiranja

Tipičan izgled prozora PowerDesigner-a prikazan je na slici broj 11 i uočljive su sledeće komponente:

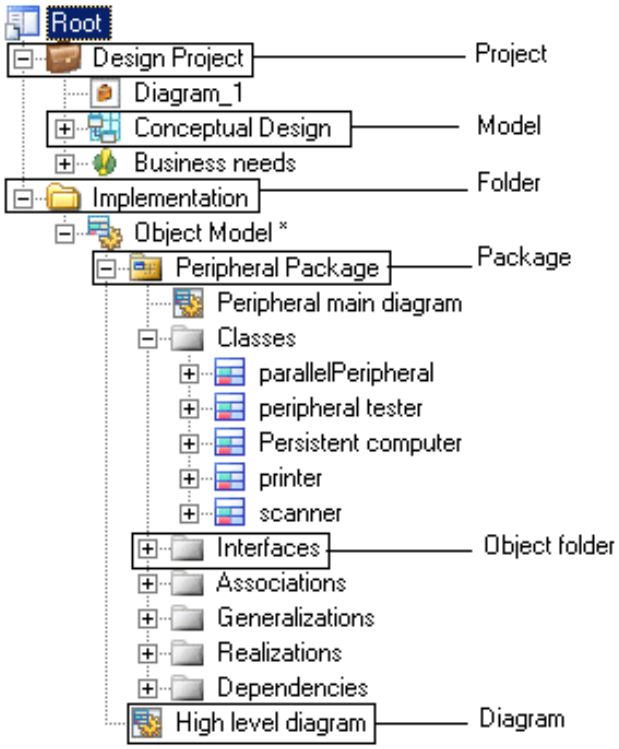
- Browser - prikazuje modele i objekte koji im pripadaju i omogućava brzu navigaciju kroz njih. Browser takođe poseduje i karticu koja omogućava pristup Repository-ju, u kom možemo čuvati sve naše modele i pripadajuće fajlove.
- Canvas - predstavlja osnovni prozor koji prikazuje trenutni dijagram modela ili izveštaj.
- Palette - predstavljaju grafičke alate koji pomažu da brzo kreiramo dijagrame modela. Prikazani alati će se razlikovati u zavisnosti od tipa dijagrama.
- Output window - prikazuje napredak bilo kog PowerDesigner procesa, kao što su kreiranje modela, provera modela ili kreiranje baze podataka iz modela.
- Result list - prikazuje rezultate pretraga ili provere modela.



Slika broj 11. Interfejs Power Designer programa [11]

3.1. Browser

Browser omogućava hijerarhijski pogled na sve objekte modela. Izgled Browser-a je dat na sledećoj slici.



Slika broj 12. Power Designer Browser

Tipična hijerarhija objekata u stablu prikaza je sledeća:

- Workspace - početak svakog stabla prikaza u Browser-u je specifičan tip foldera koji se naziva workspace (radno okruženje). To je virtuelno okruženje koje sadrži i organizuje sve informacije i fajlove koje kreiramo prilikom stvaranja modela. Workspace omogućava da sačuvamo trenutno stanje modela koje možemo pokrenuti sledeći put kada pokrećemo PowerDesigner.
- Projekat - ponaša se kao kontejner za sve činioce koji su korišćeni u razvoju, omogućavajući da se oni sačuvaju kao jedna stavka u repository-ju. Svaki projekat sadrži dijagram koji automatski kreira i prikazuje zavisnosti i ostale veze između modela i drugih dokumenata.
- Folderi - workspace može da sadrži foldere koje definiše korisnik, a koji omogućavaju da se organizuju modeli ili drugi fajlovi u grupe. Npr. ako radimo na dva različita projekta, ali želimo da pristupamo i jednom i drugom iz istog workspace-a, možemo organizovati njihove fajlove upotrebom foldera.
- Modeli - su osnovne jedinice dizajna u PowerDesigner-u. Svaki model ima jedan ili više grafičkih pogleda koji se nazivaju dijagrami i određen broj objekata modela.
- Paketi - kada se radi na velikim modelima, možemo ih podeliti u manje "pod-modele" kako bi izbegli rukovanje sa velikim skupom stavki. Ovi pod-modeli se nazivaju paketi, a

mogu se koristiti kako bi dodelili različite zadatke ili oblasti od interesa pojedinim razvojnim timovima.

- Dijagrami - prikazuju interakciju različitih objekata u modelu. Moguće je kreirati više dijagrama kako u okviru jednog modela tako i u okviru jednog paketa.
- Objekti modela - ovo je opšti izraz koji se koristi da opiše sve stavke koje pripadaju modelu. Neki objekti modela, kao što su klase u objektno orjentisanom modelu, imaju grafičke simbole, dok se ostali, kao što su poslovna pravila, ne pojavljuju u dijagramima. Njima se može pristupiti jedino pomoću Browser-a ili liste objekata.
- Izveštaji - mogu se automatski generisati kako bi dokumentovali model.

3.2. Modelovanje u PowerDesigner-u

PowerDesigner omogućava jedinstven skup alata za modelovanje koji objedinjuju standardne tehnike i notacije modela poslovnih procesa, modelovanja podataka i UML modelovanje sa ostalim moćnim osobinama koje pomažu u analiziranju, dizajniranju, kreiranju i održavanju aplikacija. On takođe omogućava da se blisko integrišu dizajn i održavanje različitih slojeva unutar aplikacije, poslovni procesi, objektno orjentisani kod, XML rečnik i informacije o organizaciji baze podataka. Pružajući bogat skup modela na svim nivoima apstrakcije, PowerDesigner obogaćuje proces dizajna svih aspekata sistema od koncepta pa sve do primene. On ne nameće ni jednu metodologiju u procesu kreiranja softvera. Svaka kompanija može implementirati svoj specifičan redosled operacija, dodeliti odgovornosti i uloge, opisati koji alati treba da se koriste, koje provere se zahtevaju i koji dokumenti treba da se kreiraju u svakom koraku.

Tim koji radi na razvoju softvera obuhvata više inženjera sa različitim ulogama. Ove uloge uključuju analitičare posla, analitičare podataka i dizajnere, administratore baze, ljude koji rade razvoj sistema i testiranje, pri čemu će svaki koristiti različitu kombinaciju komponenti PowerDesigner-a.

- **Business Analysts (analitičari posla)** - definišu arhitekturu organizacije, poslovne zahteve i poslovne tokove posmatrane na visokom nivou apstrakcije. Oni mogu koristiti
    - **Enterprise Architecture Model (EAM)** - pomoću kojeg mogu prikazati sveobuhvatnu sliku organizacije, definisati njenu strukturu, analizirati funkcije, procese i tokove sa visokog nivoa apstrakcije. Ovi objekti se mogu povezati sa implementacionim objektima u bilo kom drugom modelu.
    - **Requirements Model (RQM)** - pomoću kojeg mogu definisati poslovne zahteve koji se mogu prevesti u tehničke zahteve. RQM opisuje projekat tako što prikazuje i tačno objašnjava koje funkcionalnosti se moraju implementirati tokom procesa kreiranja softvera, kao i ko je odgovoran za njih. Ovi zahtevi se mogu dodati bilo kom objektu u bilo kom drugom modelu kako bi se pratilo gde i kako su zahtevi ispunjeni.
    - **Business Process Model (BPM)** - pomoću kojeg mogu da definišu poslovne tokove na visokom nivou apstrakcije, objasne postojeće i nove sisteme i da simuliraju poslovne procese kako bi smanjili potrebno vreme i resurse i povećali prihod. BPM prikazuje poslovne procese u stvarnim poslovnim terminima. Može se koristiti kao dizajnerski alat za otkrivanje poslovnih potreba pomoću koga se one hijerarhijski organizuju i grafički prikazuju
- **Data Analysts and Designers (analitičari podataka i dizajneri)** - oni mapiraju tehničke zahteve na poslovne zahteve. Ulazeći dublje u analizu, mogu se definisati Use Case dijagrami koji se mapiraju na zahteve. Može se napisati funkcionalna specifikacija i

preciznije definisati priroda i detalji svakog procesa, aplikacije ili strukture podataka. Mogu se koristiti:

- **BPM**
- **Conceptual Data Model (CDM)** - koji predstavlja prikaz sistema nezavisan od platforme, dajući apstraktan pogled na njegove statičke strukture podataka. CDM omogućava realne normalizovane strukture podataka sa više-ka-više i nadređen-podređen tipom odnosa, i omogućava jasan prikaz podataka kroz sve sisteme, čineći informacije o sistemu dostupnim korisnicima, sistem arhitektama i analitičarima.

- **Database Administrators (administratori baze podataka)** - koriste dobro definisane strukture podataka da optimizuju, denormalizuju i kreiraju baze podataka. Koristi se:
  - **Physical Data Model (PDM)** - koji predstavlja realnu bazu podataka i pridružene objekte koji se izvršavaju na serveru sa kompletnom informacijom o strukturi fizičkih objekata kao što su tabele, kolone, reference, trigeri, ugrađene procedure, pogledi i indeksi. PDM se može koristiti da generiše kod baze za bilo koju od 50 podržanih relacionih baza podataka. PDM se može kreirati inverznim inženjeringom iz skripti ili iz živog servera kroz standardnu ODBC konekciju. Držeći se PDM i CDM modela, osiguravamo da će krajnja implementacija u potpunosti odgovarati zahtevima sistema.
  - **Logical Data Model (LDM)** - koji se može ponašati kao most između CDM i PDM. Tehnički precizniji u odnosu na CDM, LDM omogućava da prikažu mnoge više-ka-više i nadređen-podređen odnosi, da se denormalizuju strukture podataka i da se definišu indeksi, bez navođenja određene RDBMS.
  - **Information Liquidity Model (ILM)** - koristi se od strane osoba koje su zadužene za kopiranje baze. On omogućava globalni prikaz kopiranja informacija iz centralne baze u jednu ili više udaljenih baza podataka.
- **Developers (inženjeri)** - pišu tehničku specifikaciju u RQM i modeluju aplikaciju definišući objektne strukture i ponašanje, i objektno/relaciono mapiranje. Oni koriste:
  - **Object-Oriented Model (OOM)** - koji koristi standardne UML dijagrame i notacije da predstavi objekte i njihove interakcije. To može biti forma inverznog inženjerstva koja se koristi da generiše kod za Java-u, .NET i mnoge druge jezike. Jaka integracija sa BPM, CDM i PDM može u velikoj meri pojednostaviti održavanje i razvoj sistema.
  - **XML Model (XSM)** - se može koristiti da grafički modeluje kompleksne strukture XML fajlova. Njihovi dijagrami i pogledi u obliku stabla daju globalan i šematski prikaz svih elemenata iz dokumenta.
- **Team Leaders (vođe projekata)** - imaju interes za sve tipove modela, i žele da osiguraju da su svi zahtevi, objekti i dokumenti međusobno povezani. Važno je da poslednja verzija dokumenata bude dostupna svim učesnicima u kreiranju softvera. Pomoću **Report Editor**-a se može automatizovati kreiranje detaljnih izveštaja (u RTF ili HTML formatima) za pojedinu ili sve komponente sistema. Može se koristiti:
  - **Free Model (FEM)** - za kreiranje dijagrama koji će objasniti arhitekturu sistema i aplikacija, scenarije upotrebe, dijagrame tokova i ostale grafičke prikaze.
- **Testers (testeri)** - koriste RQM, CDM i ostale modele, zajedno sa dizajn dokumentima kako bi razumeli kako aplikacija treba da radi i kako je kreirana.

4. Pitanja za obnavljanje gradiva / vežbu

1. Šta je UML?
2. Koji su ciljevi UML-a?
3. Kako možemo podeliti UML dijagrame?
4. Čemu služe strukturalni dijagrami i kako ih možemo podeliti?
5. Čemu služe dijagrami klasa?
6. Čemu služe dijagrami objekata?
7. Čemu služe dijagrami komponenti?
8. Čemu služe dijagrami rasporeda?
9. Čemu služe dijagrami ponašanja i kako ih možemo podeliti?
10. Čemu služe dijagrami slučajeva korišćenja?
11. Čemu služe sekvencijalni dijagrami?
12. Čemu služe dijagrami saradnje?
13. Čemu služe dijagrami stanja?
14. Čemu služe dijagrami aktivnosti?

5. Resursi

[1] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.stf.powerdesigner.eclipse.docs_15.0.0/html/clug/clugp3.htm
[2] http://www.agilemodeling.com/artifacts/classDiagram.htm
[3] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.infocenter.dc38086.1520/doc/html/rad1232632572941.html
[4] http://agilemodeling.com/artifacts/componentDiagram.htm
[5] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.stf.powerdesigner.docs_12.1.0/html/clug/clugp316.htm
[6] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.stf.powerdesigner.docs_12.1.0/html/clug/clugp157.htm
[7] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.infocenter.dc38086.1520/doc/html/rad1232633002992.html
[8] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.stf.powerdesigner.docs_12.0.0/html/clug/clugp176.htm
[9] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.infocenter.dc38086.1510/doc/html/rad1232633009242.html
[10] http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.infocenter.dc38086.1510/doc/html/rad1232633006132.html
[11] http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc38093.1610/doc/html/rad1232023402894.html