

POGLAVLJE

1

Uvod u UML

TEMPO U SAVREMENOM BIZNISU POSTAJE SVE BRŽI I BRŽI, SA VELIKOM POTREBOM DA SE POBEDE I PRIHVATE ZAHTEVI TRŽIŠTA. DANAS U DOBA E-BIZNISA, E-TRGOVINE I DRUGIH "E" SISTEMA "TRADICIONALNI" NAČINI RAZVOJA SISTEMA JEDNOSTAVNO NISU ODGOVARAJUĆI. DANAS SISTEM MORA BITI RAZVIJEN NA OSNOVU ZAHTEVA "INTERNET VREMENA". TAKOĐE, VEĆI TEMPO JE DOVEO DO VEĆE POTREBE DA SISTEMI KOJI SE RAZVIJAJU BUDU VEOMA FLEKSIBILNI. RANIJE JE KORISNIK MOGAO DA NA SVOJ ZAHTEV, POSLAT CENTRU ZA OBRADU PODATAKA, ČEKA OKO DVE GODINE KAKO BI DOBIO ODGOVARAJUĆU PROMENU. DANAS KORISNIK ŠALJE ZAHTEV ZA NEKOM PROMENOM IT ODELJENJU I UJEDNO ZAHTEVA ODGOVOR ZA NAJMANJE DVE NEDELJE! CIKLUSI RAZVOJA OD ŠEST NEDELJA, ZAHTEVI MENADŽMENTA, ZAHTEVI KORISNIKA I ČAK I ZAHTEVI XP (ENG. EXTREME PROGRAMMING) VODE KA SLEDEĆEM: PROMENE U SISTEMU MORAJU DA SE ODRADE BRZO!

Ovde je mesto gde UML (eng.Unified Modeling Language-Jedinstveni jezik modelovanja) stupa na scenu. UML je standardizovana notacija za modelovanje objektno-orientisanih sistema i osnovna platforma za brz razvoj aplikacija. U ovom poglavlju, opisacemo osnove UML-a, upoznaćemo koncepte objektno-orientisanog programiranja i pokazati Vam kako da koristite UML za razvoj Vaše aplikacije.

- Učenje o objektno-orientisanoj paradigm i vizuelnom modelovanju
- Istraživati tipove grafičke notacije
- Upoznati tipove UML dijagrama
- Razvijati softver koristeći vizuelno modelovanje

Uvod u objektno-orientisaniu paradigmu

Struktuirano programiranje je do skora bilo osnovna za razvoj softvera. Programeri su pisali standardne linije koda kako bi izveli operacije poput štampanja, a onda su taj kod kopirali u sve aplikacije koje su pisali, kad im je bilo potrebno da izvedu tu operaciju. Ovako se uspevalo smanjiti vreme razvoja novih aplikacija, ali su problemi nastali kad je bilo potrebno izmeniti kod koji je kopiran, zato što je izmena zahtevala da se to izvrši u svim aplikacijama u kojima je taj kod kopiran. *Objektno-orientisano* programiranje upravo ima za cilj da ove probleme koji su stvoreni strukturiranim programiranjem reši.

Primenjujući koncepte objektno-orientisanog programiranja, programeri kreiraju blokove koda koji se zovu objekti. Ovi objekti se onda koriste od strane različitih aplikacija. Ako jedan od objekata treba modifikovati, programer tu promenu treba da uradi samo jednom. Kompanije žure kako bi prilagodili ovu tehnologiju i integralsali je u njihove već postojeće aplikacije. Činjenica je da većina aplikacija koja se danas razvijaju su objektno-orientisane. Neki od programskih jezika, kao na primer Java, zahtevaju objektno-orientisaniu strukturu programa. Ali šta to u stvari znači?

Objektno-orientisana paradigma je drugačije viđenje aplikacija. Sa objektno-orientisanim pristupom, Vi aplikaciju delite na mnoštvo malih delova, ili objekata, koji su prilično nezavisni jedan od drugoga. Onda možete razvijati aplikaciju tako što ćete povezivati te objekte u jedinstvenu celinu. To je kao da gradite zamak pomoću granitnih blokova. Prvi korak je da napravite ili kupite osnovne objekte, različite tipove granitnih blokova. Kad imate sve te blokove možete ih složiti i napraviti sebi zamak. Kad jednom kupite ili napravite osnovne objekte, u svetu računara, možete ih jednostavno spojiti i kreirati novu aplikaciju.

U svetu struktuiranog programiranja, kreirati formu sa na primer list box-om, morate da napišete popriličan kod: kod koji kreira samu formu, kod koji kreira i "puni" list box i kod koji kreira OK dugme koje potvrđuje izabranu vrednost u list box-u. Primenom objektno-orientisanog programiranja Vi samo treba da koristite tri objekta: formu, list box i OK dugme. Dakle, nova paradigma je "stavi zajedno gomilu objekata, a onda se fokusiraj na ono što je specifično za pojedinačnu aplikaciju".

Jedna od osnovnih prednosti objektno-orientisanog programiranja je mogućnost izgradnje komponenti samo jednom i njihovo ponovno korišćenje. Kao što možete ponovo koristiti elemente za Vaš zamak igračku i napraviti umesto zamka kućicu, možete ponovo koristiti osnovne delove objektno-orientisanog dizajna i koda u računovodstvenim sistemima, u sistemima za vođenje magacina ili u drugim sistemima.

No, kako se ova objektno-orientisana paradigma razlikuje od tradicionalnog pristupa u razvoju aplikacija? Tradicionalno, pristup u razvoju aplikacija vodi brigu o tome koje će informacije sistem da čuva i obrađuje. Primenom ovog pristupa, mi pitamo korisnika koje su mu informacije potrebne, dizajniramo bazu podataka koja će te informacije da čuva, izradujemo ekranске forme preko kojih će se vršiti unos podataka, i onogućavamo štampanje izveštaja koji sadrže informacije od interesa korisniku. Drugim rečima, mi se fokusiramo na informacije i manje obraćamo pažnju šta se radi sa informacijama ili kako se sistem ponaša. Ovaj pristup ce zove i pristup orijentisan ka podacima i korišćen je za razvoj hiljada aplikacija i sistema u prošlom periodu.

Modelovanje orijentisano ka podacima je dobro za razvoj baza podataka i čuvanje informacija, ali primena ovog pristupa u razvoju poslovnih aplikacija dovodi do određenih problema. Jedan od osnovnih izazova je taj što se zahtevi sistema vremenom menjaju. Sistem koji je orijentisan ka podacima može da promene u bazi podataka podnese lako, ali promene u poslovnim pravilima ili u ponašanju sistema nije lako implementirati.

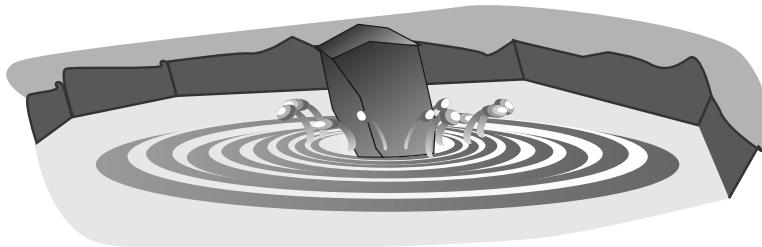
Objektno-orientisana paradigma daje odgovor na ovaj problem. Primenom objektno-orientisanog pristupa, mi se fokusiramo i na informacije i na ponašanje sistema. Naravno, mi sad možemo da razvijemo sistem koji je elastičan i fleksibilan na promene informacija i/ili ponašanja.

Dobit elastičnošću sistema može da se realizuje dobrim dizajnom objektno-orientisanog sistema. Ovo zahteva poznavanje određenih principa objektno-orientisane paradigme: enkapsulacije, nasleđivanja i polimorfizma.

Enkapsulacija

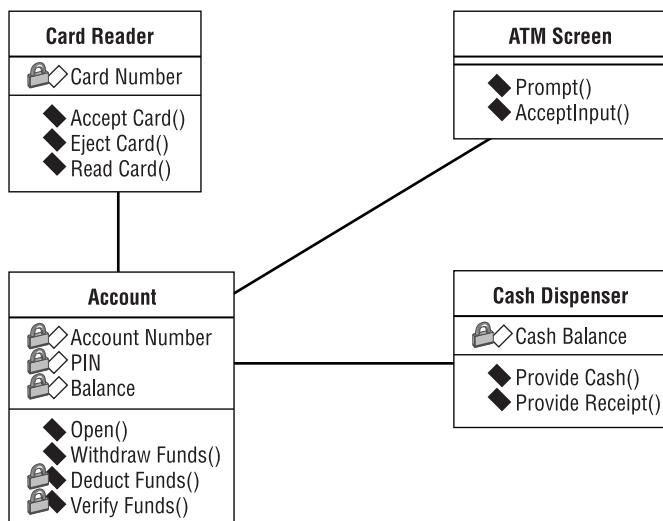
U objektno-orientisanim sistemima, mi kombinujemo deo informacije sa specifičnim ponašanjem koje ta informacija nosi. Onda mi te dve stvari upakujemo u objekat. To upućuje na enkapsulaciju. Drugi pogled na enkapsulaciju je da mi aplikaciju u stvari delimo na male delove koji su međusobno funkcionalno povezani. Na primer, imamo informacije koje se odnose na račun u banci, kao što je broj računa, saldo, ime stranke, adresa, tip računa, kamata i datum otvaranja računa. Mi takođe imamo i odgovarajuće ponašanje (operacije) vezano za taj račun: otvaranje, zatvaranje, depozit, isplata, promena tipa, promena stranke i promena adresu. Mi vršimo enkapsulaciju tih informacija i ponašanja (operacija) u objekat koji se zove račun. Kao rezultat imamo da će sve promene u bankarskom sistemu, a koje se tiču računa, biti jednostavno implementirane u objekat račun a preko njega za sve pojedinačne račune korisnika banke.

Dobit od primene enkapsulacije je i to što ograničava efekte promene sistema. Zamislimo sistem kao vodenu površinu a zahtevanu promenu kao veliki kamen. Vi ispustite kamen u vodu i-TRAS!- veliki talasi se formiraju u svim pravcima. Oni putuju preko te vodene površine i udaraju o obalu, vraćaju se nazad i sudaraju se sa ostalim talasima. Neki od talasa mogu i da prebace obalu i doprovode te vodene površine. Drugim rečima, udar kamena u vodu izazvao je veliki poremećaj. Ali ako našu vodenu površinu enkapsuliramo deljenjem u niz malih vodenih površina sa ogradama između njih, pa onda tu bacimo naš kamen, imaćemo kao i pre talase u svim pravcima. Ali ti talasi mogu da idu samo do prve ograde i onda se zaustavljaju. Dakle enkapsulacijom naše vodene površine neutralisali smo veliki poremećaj koji udar kamena može da izazove kao što je prikazano na slici 1.1.



SLIKA 1.1 Enkapsulacija: Model vodene površine

Primenimo ovu ideju enkapsulacije na bankarski sistem. Predpostavimo da je menadžment banke doneo odluku da ako klijent ima račun kod banke, taj račun može da se koristi kao osnovu za njegov tekući račun. U neenkapsuliranom sistemu analiza se vrši široko na globalnom nivou. U osnovi, mi ne znamo gde su sve pozivi određenih funkcija sistema, tako da možemo da tražimo praktično svuda. Kada uočimo te pozive funkcija onda moramo da na osnovu zahteva koje one nose izvršimo određene promene u sistemu. Ako smo u ovome dobri uspećemo da pronademo oko 80 procenata poziva tih funkcija u sistemu. Kod enkapsuliranih sistema mi ne moramo da analiziramo ovoliko široko. Mi jednostavno posmatramo model našeg sistema i jednostavno lociramo gde je određeni poziv određene funkcije enkapsuliran. Kad lociramo funkciju u računu, mi promenu te funkcije vršimo samo tu, samo u tom objektu i naš zadatak je završen! Kao što se vidi na slici 1.2, promenu treba izvršiti samo u klasi Račun.



SLIKA 1.2 Enkapsulacija: Model banke

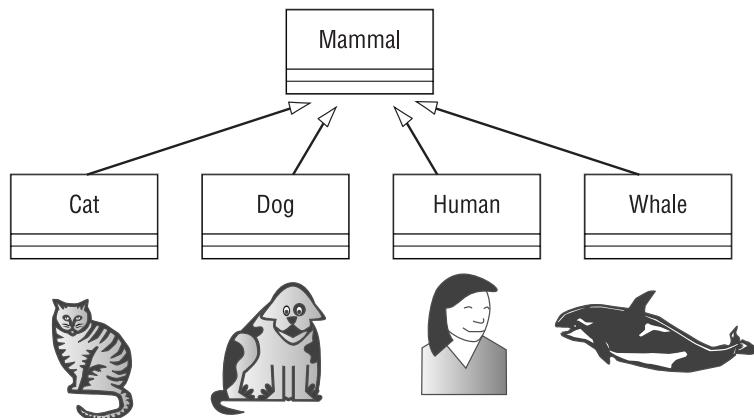
Koncept sličan enkapsulaciji je *skrivanje informacija*. Skrivanje informacija je sposobnost skrivanja nekih detalja o objektu od spoljašnjeg sveta. Za objekat spoljašnji svet je sve ono što je izvan njega samog kao objekta, čak i ako spoljašnji svet uključuje ostatak sistema. Skrivanje informacija ima za

posledicu istu dobit kao i enkapsulacija: fleksibilnost. Ovaj koncept ćemo istražiti više u Poglavlju IV "Klase i paketi".

Nasleđivanje

Nasleđivanje je drugi osnovni princip objektno-orientisanog koncepta. Ne, nema nikakve veze sa milionima evra koje ti je ostavila tetka iz Klagenfurta. Nasleđivanje ima više veze sa oblikom nosa i ušiju koje si dobio od majke i oca. U objektno-orientisanim sistemima, nasleđivanje je mehanizam koji koji omogućava da kreirate novi objekat na osnovu starog. Objekat *dete* nasleđuje sve osobine od objekta *roditelja*.

U stvarnom svetu se može videti dosta primera nasleđivanja. Postoji hiljade vrsta sisara: psi, mačke, ljudi, kitovi itd. Svaka od ovih vrsta ima neke jedinstvene karakteristike i neke koji su im zajedničke, kao na primer: da imaju dlaku, da su toplokrvni, da uzbajaju svoje mlade. U objektno-orientisanim terminima, postoji objekat *sisar* koji sadrži sve zajedničke karakteristike. Ovaj objekat je roditelj objektima deci ko što su: mačka, pas, čovek, kit itd. Objekat pas nasleđuje osobine objekta sisar i ima neke dodatne osobine koje ga detaljnije određuju, ko što su: trčanje u krug i zavijanje. Objektno-orientisana paradigma je ovu ideju nasleđivanja pozajmila iz stvarnog sveta, prikazano na slici 1.3, tako da ovaj koncept možemo primeniti na naš sistem.

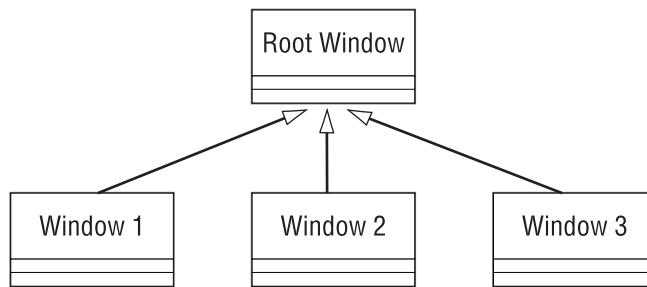


SLIKA 1.3 Nasleđivanje: Prirodni model

Jedna od osnovnih prednosti nasleđivanja je mogućnost jednostavnog održavanja. Kad se nešto promeni a tiče se svih sisara, moramo promeniti samo objekat roditelja, a objekti deca će te promene automatski naslediti. Ako sisari iznenada postanu hladnokrvni, promenu vršimo samo na objektu roditelju. Ostali objekti: mačka, pas, čovek, kit i druga deca objekti će automatski naslediti novu osobinu, hladnokrvnost.

U objektno-orientisanim sistemima, primer nasleđivanja možemo naći u prozorima. Recimo da imamo sistem sa 125 prozora. Jednog dana, kupac zahteva određenu poruku na svim prozorima. U sistemima, koji nemaju implementiran princip nasleđivanja, moramo da na svakom od 125 prozora izvršimo promenu. A ako je naš sistem objektno-orientisan mi ćemo primenom principa nasleđivanja seve prozore formirati na osnovu jednog koji bi bio objekat roditelj. Sad promenom

objekta roditelja, dodavanjem poruke, svi objekti će naslediti tu osobinu automatski kao što je prikazano na slici 1.4.



SLIKA 1.4 Nasleđivanje: Model sa prozorima

U bankarskom sistemu, nasleđivanje možemo da koristimo za različite tipove računa koje u banci imamo. Neka naša zamišljena banka ima četiri različita tipa računa: oročeni račun, tekući račun, potvrda depozita, kreditna kartica. Ovi različiti tipovi računa imaju i neke zajedničke osobine. Svaki od njih ima broj računa, vrednost kamate, i ime vlasnika. Dakle možemo kreirati objekat roditelja koji treba da ima ove zajedničke karakteristike za sve račune. Objekat dete može da ima svoje jedinstvene karakteristike pored onih koje nasleđuje od objekta roditelja. Za tekući račun na osnovu koga je dobijen kredit, na primer imamo posebne karakteristike kao što su: limit i minimum mesečne rate. Potvrda depozita će imati datum dospelosti. Promene kod roditelja će imati uticaj na decu ali će promena kod objekta deteta ostati samo njegova.

Polimorfizam

Treći princip objektne orientacije je *polimorfizam*. Rečnik polimorfizam definiše kao pojavu različitih formi, stanja ili tipova. Polimorfizam znači imati mnoge forme ili implementacije pojedinačne funkcionalnosti. Kao i nasleđivanje polimorfizam srećemo i u stvarnom svetu. Dodeljivanjem osobine govora i zahtevom "Govori!" čovek kaže "Kako si?", pas "Av Av!", mačka "Mjau!" ili Vas najčešće ignoriše.

Govoreći terminima objektno-orientisane paradigme, ovo znači da možemo imati više implementacija jedne iste funkcionalnosti. Na primer, možemo da pravimo sistem za crtanje. Kada korisnik hoće nešto da nacrtá, bilo liniju, krug ili poligon, sistem pokreće komandu crtaj. Sistem sadrži mnoge tipove oblika koji svaki za sebe ima specifičan način crtanja. Kada korisnik hoće da nacrtá krug poziva se komanda za crtanje objekta krug. Koristeći polimorfizam sistem u toku rada "shvata" koji oblik treba nacrtati. Bez polimorfizma kod za funkciju crtanja bi izgledao ovako:

```

Function Shape.drawMe()
{
  SWITCH Shape.Type
  Case "Circle"
    Shape.drawCircle();
  Case "Rectangle"
    Shape.drawRectangle();
}
  
```

```
Case "Line"
Shape.drawLine();
END SWITCH
}
```

Sa polimorfizmom kod za crtanje bi samo pozvao funkciju `drawMe()` kao u primeru:

```
Function draw()
{
Shape.drawMe();
}
```

Svaki oblik (krug, linija, poligon, itd) će imati funkciju `drawMe()` za pojedinačan oblik.

Jedna od dobiti polimorfizma, kao i drugih osobina objektno-orientisanog pristupa, je jednostavnost održavanja. Šta se dešava, na primer, kad aplikacija treba da nacrtava trougao? U slučaju bez polimorfizma moramo da napišemo novu funkciju `drawTriangle()` za objekat tog oblika. Primenom polimorfizma mi objekat trougao crtamo pozivom funkcije `drawMe()`. Tako se obezbeđuje da se osnovna funkcija `draw()` na mora uopšte menjati.

Šta je vizuelno modelovanje?

Da Vi gradite novi sprat u Vašoj kući, verovatno ne biste počeli tako što bi samo kupili gomilu dasaka i prikivali ih sve zajedno sve dok ne ispadne kako treba. Slično biste bili više nego malo zabrinuti da je majstor koji radi posao odlučio da to odradi "površno" i bez planova. Vi biste želeli da pogledate planove koje biste pratili pre nego sto bi otpočeli sa radom. Šanse su da će nadogradnja trajati malo duže ovaj put. Sigurno ne biste želeli da se nadograđeni sprat sruši pri najmanjoj kiši.

Modeli završavaju isti posao za nas u svetu softvera. Oni su planovi za sistem. Planovi nam pomažu da isplaniramo nadogradnju pre nego što stvarno krenemo da dograđujemo našu aplikaciju, model nam pomaže da isplaniramo sistem pre nego što ga napravimo. Može sigurno da Vam pomogne kada su svi uslovi ispunjeni i sistem može da podnese uragan promena.

Kada sakupljate prohteve za Vaš sistem, Vi ćete uzeti sve potrebe korisnika i staviti ih među zahteve koje Vaš tim razume i koje može koristiti. Eventualno ćete hteti da uzmete ove zahteve i generišete kod iz njih. Formalno povezujući zahteve sa kodovima, možete osigurati da se zahtevi ustvari susreću sa kodom, tj. da se kod može lako nazad povezati sa zahtevima. Ovaj proces se zove *modelovanje*. Rezultat procesa modelovanja je mogućnost da se prate poslovni zahtevi do potreba za model koda, i obratno, bez problema tokom rada.

Vizuelno modelovanje je proces uzimanja informacija sa modela i njihovo grafičko prikazivanje korišćenjem niza standardnih grafičkih elemenata. Standardi su neophodni kako bi se izvukla bitna korist od modelovanja: komunikacija. Komunikacija između korisnika, informatičara (programera), analitičara, menadžera i svih onih koji su uključeni u projekat, je osnovna svrha vizuelnog modelovanja. Komunikaciju možete ostvariti korišćenjem nevizuelnih (tekstualnih) informacija, ali ipak morate priznati ljudi su bića koja vizuelno mnogo lakše prihvataju i uočavaju stvari. Mi imamo veću mogućnost da razumemo stvari ako su nam one prezentovane vizuelno preko modela nego u slučaju kad su nam prezentovane samo tekstom. Izradom vizuelnog modela sistema, možemo pokazati

kako sistem radi posmatrajući više nivoa. Možemo modelovati interakciju korisnika i sistema. Možemo modelovati interakciju pojedinih objekata sistema i samog sistema. Možemo čak modelovati i interakciju između sistema, ako to baš želimo.

Nakon kreiranja modela, možemo ih pokazati svim zainteresovanim grupama, a te grupe ljudi mogu sa modela da prikupljaju informacije koje su im od intresa. Na primer, korisnici mogu da se fokusiraju na interakciju koju oni ostvaruju sa sistemom na osnovu naših modela. Analitičari mogu da uvide interakciju objekata koji su u modelu dati. Informatičari mogu da uoče objekte koje trebaju da naprave i šta svaki od njih treba da obavlja. Inžinjeri koji treba da kasnije vrše testiranje aplikacije mogu da vizuelizuju interakciju između objekata i da im to pomogne u pripremi testova koji će biti zasnovani upravo na tim interakcijama. Menadžeri projekta mogu da ostvare uvid u ceo sistem i u interakciju pojedinih delova sistema. Šef cele organizacije može da uvidi, pregledom modela većeg nivoa, kako sistemi u njegovoj organizaciji interaguju međusobno. Sve u svemu, vizuelno modelovanje predstavlja moćan alat za prikazivanje svrhe sistema svim zainteresovanim grupama.

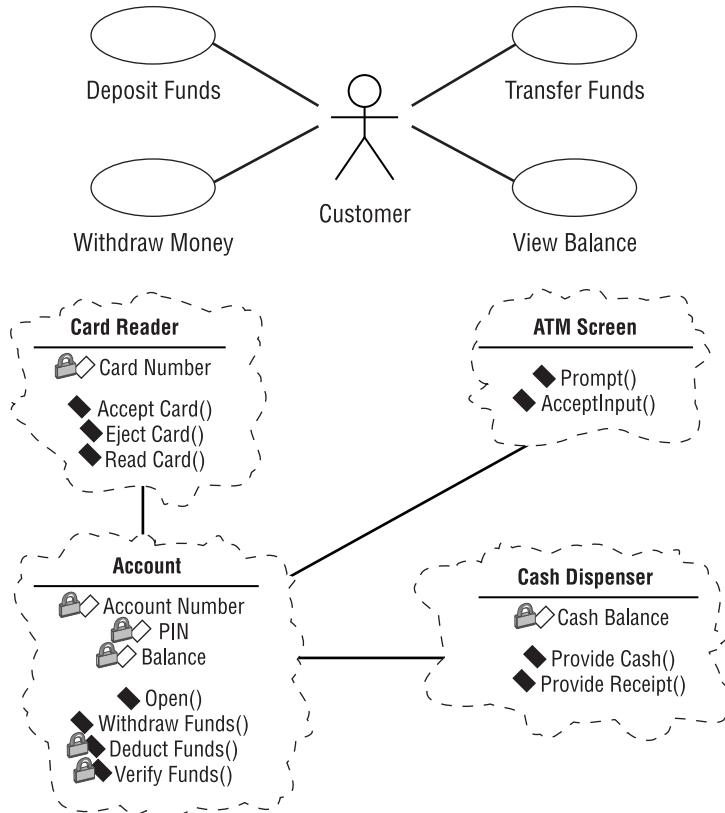
Sistemi grafičke notacije

Jedno od važnih pitanja u vizuelnom modelovanju je koju grafičku notaciju koristiti za opis različitih aspekata sistema. Ta notacija mora da bude poznata svim zainteresovanim grupama, u suprotnom model neće biti veoma koristan. Do sada su mnogi predlagali koju notaciju koristiti za vizuelno modelovanje. Neke od popularnih notacija, koje imaju jaku podršku, su Bučova notacija, OMT (eng. Object Modeling Technology) i UML.

Rational Rose podržava ove tri notacije, međutim, UML je standard koji je usvojen od većine korisnika i grupa za standardizaciju kao što su ANSI i OMG (eng. Object Management Group).

Bučova notacija

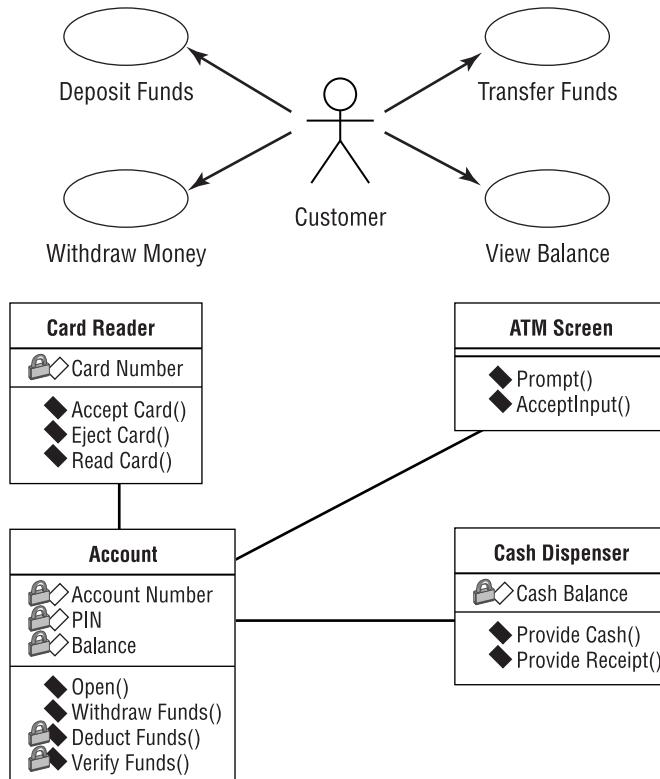
Bučova notacija je nazvana po njenom autoru Grady Booch-u iz kompanije Rational. On je napisao nekoliko knjiga u kojima govori o potrebi i prednostima vizuelnog modelovanja, pored toga on je i razvio notaciju grafičkih simbola za reprezentaciju različitih aspekata modela. Na primer, objekti u toj notaciji se prikazuju preko simbola koji podseća na oblak, ilustrujući činjenicu da objekat može biti skoro bilo šta. Bučova notacija, takođe, sadrži i niz različitih strelica za reprezentaciju više tipova veza između objekata. O ovim stvarima, objektima i relacijama između njih, čemo govoriti više u poglavlju IV, "Slučajevi upotrebe i akteri". Slika 1.5 je primer objekata i relacija prikazanih Bučovom notacijom.



SLIKA 1.5 Primer simbola Bučove notacije

Object Management Technology (OMT)

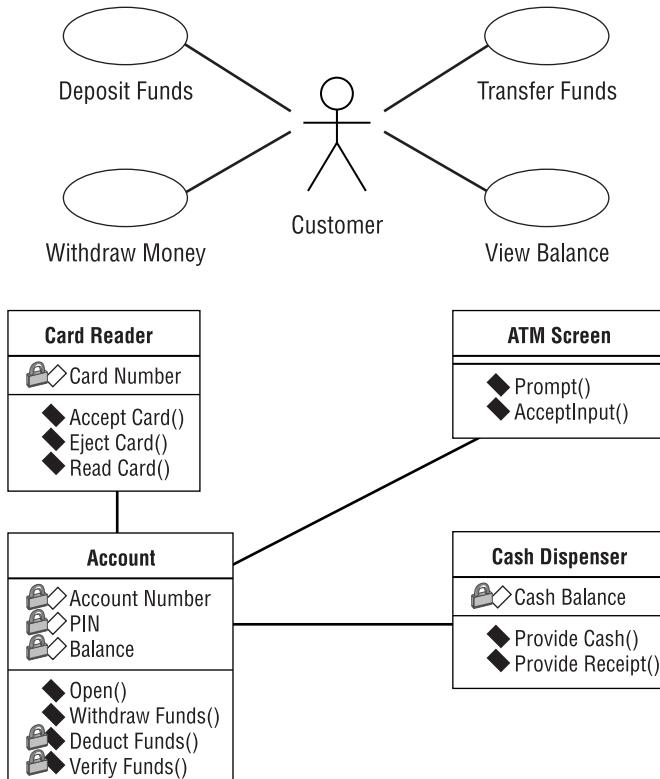
Tvorac OMT notacije je dr James Rumbaugh, koji je napisao nekoliko knjiga o analizi i dizajnu sistema. U knjizi "Objektno-orientisano modelovanje i dizajn" (Prentice Hall, 1990), Rumbaugh govori o važnosti modelovanja sistema preko objekata koji reprezentuju komponente stvarnog sveta. OMT koristi jednostavniju grafičku reprezentaciju nego Buč za opis sistema. Primer objekata i relacija prikazanih OMT notacijom je dat na slici 1.6.



SLIKA 1.6 Primer simbola OMT notacije

Unified Modeling Language (UML)

UML notacija je posledica saradnje Grady Booch-a, dr James Rumbaugh-a, Ivar Jacobson-a, Rebecca Wirfs-Brock, Peter Yourdon-a i mnogih drugih. Jacobson je naučnik koji je pisao o skupljanju zahteva sistema u transakcione pakete koji se nazivaju *slučajevi upotrebe*. O slučajevima upotrebe ćemo detaljnije govoriti u poglavljju IV. Jacobson je takođe razvio metod dizajna sistema koji se zove "*Objektno-orientisane softverske tehnike (OOSE)*" koji se fokusira na analizu sistema. Booch, Rumbaugh i Jacobson, koje popularno zovu "tri amigosa", zajedno rade u kompaniji "Rational Software" na standardizaciji i usavršavanju UML-a. UML simboli su veoma slični notacijama Buča i OMT, ali uključuju i elemente drugih notacija. Slika 1.7 pokazuje primer UML notacije.



SLIKA 1.7 Primer simbola u UML notaciji

Objedinjavanje metoda, koji su kasnije nazvani UML, počelo je 1993. Svaki od "tri amigosa" UML-a unosio je ideje drugih metodologija u UML. Zvanična unifikacija metodologije je nastavljena kasnije 1995, verzijom 0.8 UM-a (Unified Method). UM je dorađen i promenjen u UML 1996. Kasnije 1997 UML je dorađen i dat OTG-u (eng. Object Technology Group) a i mnoge kompanije su prihvatile. OMG je 1997 objavio UML 1.1 kao industrijski standard.

Proteklih godina, UML je evoluirao dodavanjem novih modela kao što su web zasnovani sistemi i modeli za modelovanje podataka. Poslednja objava UML-a je 1.3 koja se pojavila 2000. Specifikacija za UML 1.3 može se naći na web sajtu OMG-a (www.omg.org). U ovoj knjizi je korišćena verzija UML-a 1.3.

UML dijagrami

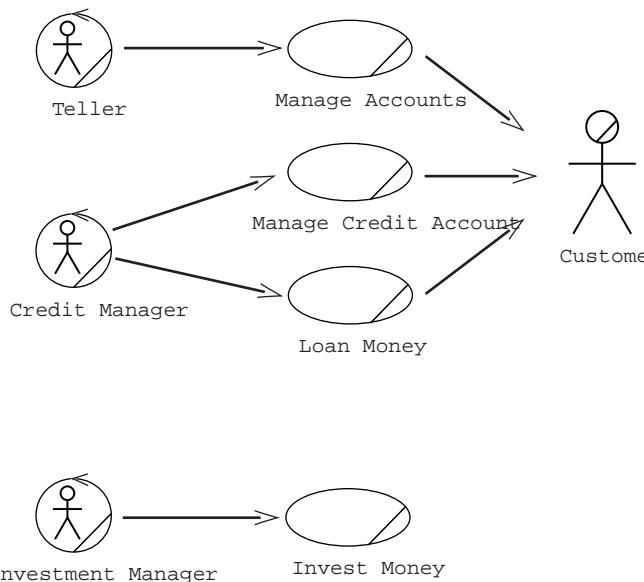
UML daje korisnicima nekoliko različitih tipova dijagrama koji opisuju različite aspekte sistema. Rational Rose podržava razvoj većine ovih dijagrama, kao na primer:

- Dijagrami slučajeva upotrebe poslovnog procesa
- Dijagrami slučajeva upotrebe
- Dijagrami aktivnosti
- Dijagrami sekvenci
- Dijagrami saradnje
- Dijagrami klasa
- Dijagrami stanja
- Dijagrami komponenti
- Dijagrami razmeštaja

Ovi dijagrami ilustruju različite aspekte sistema. Na primer, dijagrami kolaboracije pokazuju zahtevanu interakciju između objekata po redu kojim se neka funkcionalnost sistema ostvaruje. Svaki od dijagrama ima svoju svrhu i ciljnu grupu.

Dijagrami slučajeva upotrebe poslovnog procesa

Ovi dijagrami se koriste kako bi se prikazala funkcionalnost organizacije kao celine. Oni daju odgovor na pitanja "Koji se to posao obavlja?" i "Zašto gradimo sistem?". Oni se intenzivno koriste tokom modelovanja poslovnog procesa kako bi shvatili suštinu sistema i kako bi postavili temelj za izradu konkretnih slučajeva upotrebe. Primer pojednostavljenog dijagrama slučajeva upotrebe poslovnog procesa za neku finansijsku instituciju prikazan je na slici 1.8.



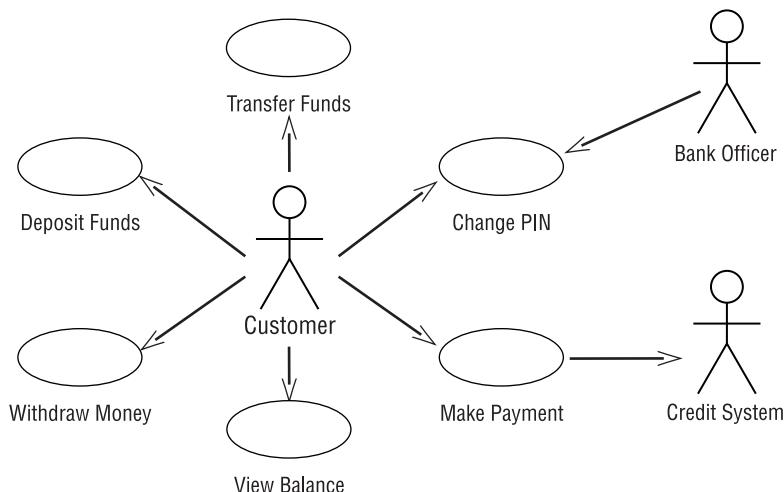
SLIKA 1.8 Dijagram slučaja upotrebe poslovnog procesa za neku finansijsku instituciju

Dijagrami slučajeva upotrebe poslovnog procesa se crtaju iz perspektive organizacije. Oni ne prave razliku između ručnih i automatizovanih procesa. (Dijagrami slučajeva upotrebe, koji će biti kasnije razmatrani, fokusiraju se samo na automatizovane procese). Dijagrami slučajeva upotrebe poslovnog procesa prikazuju interakciju između slučajeva upotrebe poslovnog procesa i aktera u poslovnom procesu. Slučajeva upotrebe poslovnog procesa reprezentuju procese u toku samog posla, a akteri poslovnog procesa reprezentuju uloge koje se u interakciji javljaju, kao na primer kupci ili trgovci. Drugim rečima, akteri poslovnog procesa prikazuju bilo koga ili bilo šta izvan poslovnog procesa što sa njim interaguje, oni ne prikazuju uloge ili radnike unutar poslovnog procesa. radnici unutar poslovnog procesa se prikazuju radnicima poslovnog procesa o kojima se govori u poglavlju III, "Modelovanje poslovnog procesa".

Dijagrami slučajeva upotrebe

Dijagrami slučajeva upotrebe prikazuju interakciju između slučajeva upotrebe i aktera. Slučajevi upotrebe reprezentuju funkcionalnost sistema i zahteve sistema sa perspektive korisnika. Akteri reprezentuju ljude ili sisteme koji omogućavaju ili dobijaju informacije od datog sistema, oni su stubovi sistema. Dijagrami slučajeva upotrebe prikazuju koji akter vrši iniciranje pojedinih slučajeva upotrebe, prikazuju i da akteri dobijaju informacije od tih slučajeva upotrebe. U suštini dijagrami slučajeva upotrebe mogu da prikažu zahteve sistema.

Dok dijagrami slučajeva upoterebe poslovnog procesa ne vode brigu o tome šta je automatizованo u procesu, dijagrami slučajeva upotrebe se fokusiraju samo na automatizovane procese. Ne postoji veza jedan prema jedan između slučajeva upotrebe poslovnog procesa i slučajeva upotrebe. Na primer, jedan slučaj upotrebe poslovnog procesa može da zahteva 30 slučajeva upotrebe za implementaciju tog procesa. Primer dijagrama slučajeva upotrebe za Bankomat sistem prikazan je na slici 1.9.



SLIKA 1.9 Dijagram slučajeva upotrebe za Bankomat sistem

Ovaj dijagram slučajeva upotrebe pokazuje interakciju između slučajeva upotrebe i aktera u bankomat sistemu. Na primer, korisnik bankarskih usluga inicira nekoliko slučajeva upotrebe: Isplata, Stanje depozita, Transfer novca, Izvrši plaćanje, Stanje i Promeni PIN. Neke od veza treba i posebno napomenuti. Službenik banke takođe može da inicira Promeni PIN slučaj upotrebe. Slučaj upotrebe Izvrši plaćanje je povezan strelicom sa akterom Kreditni sistem. Spoljni sistem može kao u ovom slučaju da ima ulogu aktera jer je Kreditni sistem u takvom odnosu sa sistemom Bankomat. Strelica koja ide od slučaja upotrebe ka akteru govori o tome da slučaj upotrebe obezbeđuje neke informacije potrebne akteru. U ovom slučaju slučaj upotrebe Izvrši plaćanje obezbeđuje podatke o kreditnoj kartici kreditnom sistemu.

Analizom dijagrama slučajeva upotrebe možemo da prikupimo dosta informacija. Ovi dijagrami prikazuju opštu funkcionalnost sistema. Korisnici, menadžeri, analitičari, inžinjeri koji su zaduženi za obezbeđivanje sistema kvaliteta, i svi zainteresovani mogu pomoći ovih dijagrama da shvate koji je u stvari zadatak sistema.

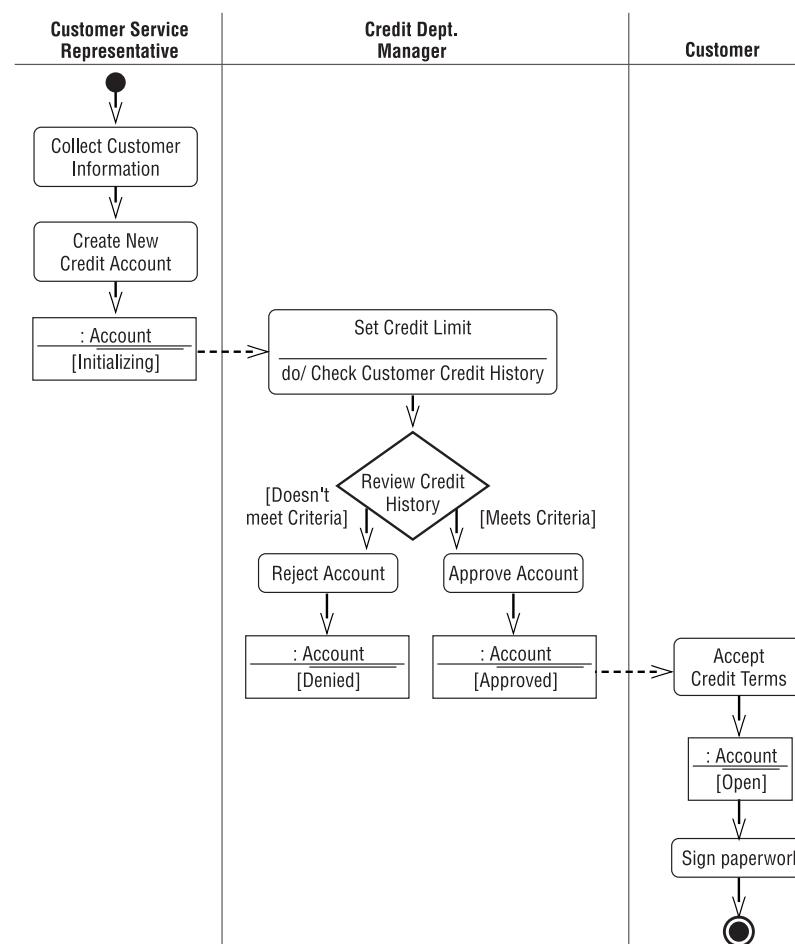
Dijagrami aktivnosti

Dijagrami aktivnosti prikazuju tok funkcionalnosti u sistemu. Oni se mogu koristiti u modelovanju procesa za prikazivanje toka samog procesa. Mogu se, takođe, koristiti i u procesu prikupljanja zahteva kako bi se prikazao tok događaja kroz konkretni slučaj upotrebe. Ovi dijagrami definišu gde počinje tok procesa, gde se završava, koje se aktivnosti u njemu odvijaju i u kom redu. Aktivnost je zadatak koji se obavlja u toku procesa.

Struktura dijagrama aktivnosti je slična dijagramima stanja, o kojima ćemo kasnije pričati u ovom poglavljju. Primer dijagrama aktivnosti je dat na slici 1.10. Aktivnosti u dijagramu su prikazane preko zaobljenih pravougaonika. Prikazani su i koraci koji se vrše kroz konkretni tok procesa. Objekti koji su obuhvaćeni tokom poslovnog procesa prikazani su pravougaonikom. Postoji i početno stanje koje prikazuje početak toka procesa i završno stanje koje prikazuje kraj procesa. Tačke donošenja neke odluke su prikazane rombom.

Pregledom isprekidanih linija može se utvrditi tok objekata kroz dijagram. Tok objekta prikazuje koji objekti se koriste ili kreiraju konkretnom aktivnišću i kako objekti menjaju stanje kroz tok procesa. Pune linije, koje se zovu prelazi, pokazuju kako se aktivnosti odvijaju jedna za drugom kroz proces. Na dijagram se po potrebi može dodati i više detalja za prelaze opisujući okolnosti pod kojim ase određena aktivnost odvija ili ne.

Dijagram aktivnosti može se podeliti po vertikali u takozvane plivačke staze. Svaka od plivačkih staza ima drgačiju ulogu u toku procesa. Pregledom aktivnosti u konkretnoj plivačkoj stazi možemo doći do obaveza uloge te plivačke staze. Pregledom prelaza između aktivnosti, u različitim plivačkim stazama, možemo saznati ko ima potrebu sa kim da komunicira posmatrajući određene aktere sistema sa svojim odgovornostima za funkcionalisanje tog sistema. Sve ovo su veoma vredne informacije kad pokušavamo da modelujemo ili razumemo poslovni proces. Dijagram aktivnosti ne treba kreirati za svaki tok procesa, ali uz pomoću njih je idealno razumeti kompleksne tokove procesa.



SLIKA 1.10 Dijagram aktivnosti za otvaranje računa