Poglavlje 1

Uvod

Šta je UML?

Objedinjen jezik za modelovanje (engl. *Unified Modeling Language*, *UML*) jeste skup grafičkih notacija zasnovanih na jedinstvenom metamodelu, a služi za opisivanje i projektovanje softverskih sistema, naročito onih koji su napravljeni primenom objektno orijentisanih tehnologija. To je donekle pojednostavljena definicija. U suštini, UML ima nekoliko različitih značenja za razne ljude, što je posledica njegove istorije i različitih stavova o efikasnim postupcima softverskog inženjerstva. Zato je moj zadatak, u većem delu uvodnog poglavlja, da postavim osnove za ovu knjigu, objašnjavajući različite načine na koje ljudi vide i koriste UML.

Grafički jezici za modelovanje koriste se već dugo u industriji softvera. Nastali su zbog mišljenja da programski jezici nisu dovoljno apstraktni da bi olakšali rasprave o projektovanju softvera.

Uprkos činjenici da grafički jezici za modelovanje postoje dugo, njihova uloga u softverskoj industriji nije usaglašena. To neslaganje direktno utiče na shvatanje uloge jezika UML.

UML je relativno otvoren standard Grupe za upravljanje objektima (engl. *Object Management Group*, *OMG*), otvorenog udruženja kompanija. Grupa za upravljanje objektima pravi standarde koji olakšavaju saradnju, naročito među objektno orijentisanim sistemima. Ova grupa je možda najpoznatija po standardu CORBA (engl. *Common Object Request Broker Architecture*).

UML je nastao objedinjavanjem više objektno orijentisanih grafičkih jezika za modelovanje, koji su razvijeni kasnih osamdesetih i ranih devedesetih godina prošlog veka. Nakon pojavljivanja 1997. godine, UML je poslao tu vavilonsku kulu u istoriju. Zbog toga smo mu i ja i mnogi drugi projektanti iskreno zahvalni.

Načini korišćenja jezika UML

Ulogu jezika UML u razvoju softvera određuju različiti načini na koje ga ljudi koriste, a razlike su prenete iz drugih grafičkih jezika za modelovanje. Ove razlike dovode do dugih i komplikovanih rasprava o tome kako treba koristiti UML.

Da bismo rešili taj problem, Steve Mellor i ja smo, nezavisno jedan od drugog, izdvojili tri načina na koja ljudi koriste UML: za izradu skica, za izradu projekata i kao programski jezik. Po mom pristrasnom opažanju, ubedljivo je najčešća **izrada skica**. U toj primeni, projektanti koriste UML kao pomoćno sredstvo za opisivanje nekih aspekata sistema. Kao i u slučaju projekata, skice možete koristiti u razvoju ili povratnoj analizi. U **direktnom razvoju** (engl. *forward engineering*) UML dijagram se crta pre pisanja koda, dok se u **povratnoj analizi** (engl. *reverse engineering*) dijagram pravi prema postojećem kodu, da bi se olakšalo razumevanje.

Suština pravljenja skica je selektivnost. Skica napravljena direktnim razvojnim postupkom (engl. *forward sketch*), sadrži nekoliko problema koji će se javiti u kodu, i obično se crta tokom diskusije sa grupom ljudi. Cilj vam je da koristite skice da biste lakše saopštili ideje i alternative predstojećeg posla. Ne razgovarate o celokupnom kodu na kome ćete raditi, nego samo o značajnim pitanjima koja prvo hoćete da prenesete kolegama, ili o delovima projekta koje hoćete da vizuelizujete pre početka programiranja. Ovi sastanci mogu biti veoma kratki: za deset minuta se može razmotriti nekoliko sati programiranja, a za jedan dan se prodiskutuje o dvonedeljnom poslu.

U povratnoj analizi koristite skice da biste objasnili kako radi neki deo sistema. Ne prikazujete sve klase, već samo one o kojima je posebno zanimljivo i važno porazgovarati pre nego što zaronite u kôd.

Pošto je pravljenje skica prilično neformalno i dinamično, potrebno je da to uradite brzo i timski, pa se za to koristi obična tabla. Skice su korisne i u dokumentaciji, kada je u središtu pažnje obaveštenje, a ne potpunost. Za pravljenje skica koriste se najjednostavnije alatke, a ljudi se obično ne pridržavaju strogo svakog pravila jezika UML. Većinu UML dijagrama prikazanih u knjigama, uključujući i druge moje knjige, čine skice. Oni više služe selektivnom informisanju nego potpunom specificiranju.

Nasuprot skicama, **UML opisi projekata** su potpuni. U direktnom razvoju projektant pravi detaljan opis sistema, a programer na osnovu tog opisa piše kôd. Opis sistema treba da bude dovoljno potpun i da obuhvati sve odluke o projektu, a programer treba da bude u stanju da ga prati lako i bez mnogo razmišljanja.

Projektant i programer mogu biti ista osoba, ali je obično projektant stariji član tima koji projektuje za veći broj programera. Ovakav pristup je inspirisan drugim inženjerskim oblastima, u kojima inženjeri prave tehničke crteže, a zatim ih prosleđuju na realizaciju drugim kompanijama.

Projekat može obuhvatiti sve detalje ili samo određenu oblast. Obično se projekat razvija sve do interfejsa podsistema, a zatim se programerima prepušta razrada pojedinosti realizacije.

U povratnoj analizi, UML projekti treba da sadrže detaljne informacije o kodu u obliku papirne dokumentacije ili elektronske dokumentacije kojoj se pristupa pomoću interaktivnog grafičkog čitača. Projekti mogu prikazati svaki detalj neke klase u grafičkom obliku koji programer lakše razume.

Za izradu UML projekata potrebni su mnogo složeniji alati nego za izradu UML skica. U kategoriju alata za projektovanje spadaju specijalizovani alati za računarsko projektovanje softvera (engl. *computer-aided software engineering*, *CASE*), iako je CASE postao zloupotrebljen termin i proizvođači nastoje da ga izbegnu. Alati za direktan razvojni postupak omogućavaju crtanje dijagrama, a informacije o njima čuvaju u skladištu. Alati za povratnu analizu čitaju izvorni kôd, upisuju njegovo tumačenje u skladište i stvaraju dijagrame. Alati koji se mogu koristiti u obe vrste postupka poznati su kao **dvosmerni** (engl. *round-trip*).

Neki alati koriste izvorni kôd kao skladište, a dijagrame kao grafički prikaz koda. Ovi alati su mnogo čvršće povezani sa programiranjem i često su neposredno integrisani u razvojna okruženja. Njih obično zovem **jednosmerni** (engl. *tripless*) alati.

Granica između projekata i skica donekle je nejasna, ali se razlika, po mom mišljenju, zasniva na činjenici da su skice proizvoljno nepotpune i ističu značajne informacije, dok projekti teže sveobuhvatnosti, često sa ciljem da se programiranje svede na jednostavnu i uglavnom mehaničku aktivnost. Jednom rečju, rekao bih da se skice koriste za ispitivanje mogućih rešenja, dok se projekti moraju dosledno sprovesti.

Što više koristite UML, programiranje je sve više rutinsko, pa postaje očigledno da ga treba automatizovati. Mnogi CASE alati zapravo generišu kôd u nekom obliku, što automatizuje izgradnju većeg dela sistema. U izvesnim slučajevima dostižete tačku u kojoj celokupan sistem može biti opisan UML dijagramima i tako koristite **UML kao programski jezik**. Programeri crtaju UML dijagrame koji se neposredno prevode u izvršni kôd, a UML postaje izvorni kôd. Očigledno je da takva upotreba jezika UML zahteva posebno složene alate. (Osim toga, u ovom slučaju pojmovi direktnog razvoja i povratne analize nemaju smisla, pošto je UML dijagram izvorni kôd.)

Arhitektura zasnovana na modelu i izvršni UML

Kada se govori o jeziku UML, uglavnom se misli na **arhitekturu zasnovanu na modelu** (engl. *Model Driven Architecture*, *MDA*) [Kleppe i dr.]. U suštini, MDA je standardni pristup u korišćenju UML-a kao programskog jezika. Ovim standardom upravlja OMG, kao i UML-om. Pomoću okruženja za modelovanje prilagođenog arhitekturi MDA, proizvođači mogu napraviti modele koji rade i sa drugim srodnim okruženjima.

O arhitekturi MDA i jeziku UML često se govori u istom dahu, pošto je UML osnovni programski jezik arhitekture MDA. Naravno da ne morate koristiti MDA da biste koristili UML.

Arhitektura MDA deli razvoj u dve osnovne oblasti. Projektanti predstavljaju određenu aplikaciju praveći **model nezavisan od platforme** (engl. *Platform Independent Model*, *PIM*). To je UML model koji ne zavisi od tehnologije. Zatim se model nezavisan od platforme može alatima pretvoriti u **model za specifičnu platformu** (engl. *Platform Specific Model*, *PSM*). To je model sistema namenjen određenom izvršnom okruženju. Drugi alati zatim prave kôd za tu platformu na osnovu modela PSM. Model za specifičnu platformu može, ali ne mora, biti na jeziku UML.

Na primer, ako hoćete da napravite aplikaciju za vođenje magacina pomoću arhitekture MDA, prvo ćete napraviti model nezavisan od platforme. Ako hoćete da sistem radi na platformama J2EE i .NET, koristićete alate za pravljenje dva specifična modela, po jedan za svaku platformu. Zatim će drugi alati generisati kôd za te dve platforme.

Ako je postupak prelaska od nezavisnog modela, preko modela specifičnog za platformu, do izvršnog koda, potpuno automatizovan, UML se koristi kao programski jezik. Ukoliko se bilo koji od navedenih koraka obavlja ručno, koriste se UML projekti.

Steve Mellor se već dugo bavi UML-om i nedavno je skovao izraz **izvršni UML** (engl. *Executable UML*) [Mellor i Balcer]. Izvršni UML je sličan arhitekturi zasnovanoj na modelu, ali koristi nešto drugačije pojmove. Na sličan način počinjete s modelom nezavisnim od platforme, ekvivalentnim modelu u MDA. Međutim, u sledećem koraku koristite prevodilac modela (engl. *Model Compiler*) i pretvarate taj UML model u konačan sistem. Prema tome, ne koristi se model specifičan za platformu. Kao što izraz *prevodilac* nagoveštava, ovaj korak je potpuno automatski.

Prevodioci modela su zasnovani na ponovno upotrebljivim arhetipovima. **Arhetip** opisuje kako da neki izvršni UML model pretvorimo u

kôd za određenu programsku platformu. U slučaju aplikacije za magacin, kupili biste prevodilac modela i dva arhetipa (J2EE i .NET). Primenite arhetipove na izvršni UML model i dobićete dve verzije aplikacije za magacin.

U izvršnom UML-u ne koriste se svi standardni elementi jezika UML. Mnoge konstrukcije jezika UML smatraju se nepotrebnim i zato se ne koriste. Sledi je da je izvršni UML jednostavniji od potpunog UML-a.

Sve to dobro zvuči, ali koliko je realno? Po mom mišljenju, postoje dva problema. Prvi problem se tiče alata: da li su dovoljno razvijeni da bi obavili posao? To se vremenom menja; sigurno je da se oni, u trenutku kad ovo pišem, ne koriste previše i nisam video mnogo njih u upotrebi.

Značajnije pitanje je korišćenje jezika UML kao programskog jezika. Po mom mišljenju, UML vredi koristiti kao programski jezik samo ako postižemo značajno veću produktivnost nego s nekim drugim programskim jezikom. U to nisam ubeđen, na osnovu raznih grafičkih razvojnih okruženja koja sam ranije koristio. Čak iako se postiže veća produktivnost, potrebna je kritična masa korisnika da bi alati postali opšteprihvaćeni. To je velika prepreka. Poput mnogih starih korisnika Smalltalka, smatram taj jezik produktivnijim od trenutno vodećih jezika. Međutim, pošto je Smalltalk sada gurnut u stranu, ne koristi se u velikom broju projekata. Da bi izbegao sudbinu Smalltalka, jezik UML mora imati više sreće, čak i ako je moćniji.

Jedno od zanimljivih pitanja u vezi sa korišćenjem UML-a kao programskog jezika jeste modelovanje logike ponašanja. UML 2 nudi tri načina za modelovanje ponašanja: dijagrame interakcije, dijagrame stanja i dijagrame aktivnosti. Postoje grupe koje zastupaju korišćenje svakog od tih tipova dijagrama za programiranje. Ako UML postane popularan kao programski jezik, biće zanimljivo videti koje će od tih tehnika biti prihvaćene.

Prilikom razmatranja UML-a mnogi prave razliku između njegove primene za konceptualno modelovanje i za modelovanje softvera. Većini je bliska upotreba jezika UML za modelovanje softvera. Iz **perspektive softvera**, elementi jezika UML sasvim se dobro preslikavaju u elemente softverskog sistema. Kao što ćete videti, to preslikavanje nije regulisano nikakvim propisima ali – kada koristimo UML – govorimo o softverskim elementima.

Iz **perspektive koncepata**, UML predstavlja opis koncepata domena koji proučavamo. Dijagram ne mora da opisuje softverske elemente, pošto sami pravimo rečnik određenog domena.

Ne postoje nikakva čvrsta i kratka pravila o tumačenju jezika UML. Postoje zaista prilično široke mogućnosti njegove upotrebe. Neki alati automatski pretvaraju izvorni kôd u UML dijagrame, što UML čini alternativnim načinom posmatranja koda. To je, u velikoj meri, softverska perspektiva. Ako koristite UML dijagrame da biste, zajedno sa grupom računovođa, razumeli različita značenja termina *raspoloživa sredstva*, tada ste uglavnom u okvirima konceptualnog razmišljanja.

U prethodnim izdanjima ove knjige podelio sam softversku prespektivu na specifikaciju (interfejs) i realizaciju. Kroz praksu sam otkrio da je između njih previše teško povući jasnu granicu, pa mislim da nema potrebe dalje raspravljati o tome. Međutim, sklon sam da u svojim dijagramima uvek radije istaknem interfejs nego realizaciju.

Različiti načini korišćenja jezika UML vode do mnogih rasprava o značenju UML dijagrama i njihovom odnosu prema ostatku sveta. To posebno utiče na vezu između jezika UML i izvornog koda. Neki zastupaju gledište da UML treba koristiti za pravljenje projekta nezavisnog od programskog jezika, koji se koristi za realizaciju. Drugi veruju da je projekat nezavisan od jezika oksimoron, s posebnim naglaskom na „moron“.

Postoji i razlika u gledištima šta je suština jezika UML. Po mom mišljenju, većina korisnika UML-a, naročito onih koji ga koriste za pravljenje skica, vidi suštinu u dijagramima. Međutim, tvorci jezika UML smatraj da su dijagrami sekundarni, a da je metamodel suština. Dijagrami samo predstavljaju metamodel. Takav stav odgovara i onima koji koriste UML za pravljenje projekata, odnosno kao programski jezik.

Kad god čitate nešto u vezi sa UML-om, važno je da shvatite autorovu tačku gledišta. Tek tada možete razumeti često žestoke rasprave koje UML izaziva.

Pošto sam sve ovo napisao, treba da razjasnim svoje stanovište. UML koristim uglavnom za skice. Skice su mi korisne i prilikom razvoja i prilikom povratne analize, i u konceptualnoj i u softverskoj perspektivi.

Nisam posebno naklonjen detaljnim razvojnim projektima, pošto mislim da ih je veoma teško dobro napraviti i da se tako usporava razvoj. Razumno je praviti projekte do nivoa interfejsa podsistema, ali i tada treba očekivati da će se interfejsi menjati prilikom programiranja. Vrednost projekata dobijenih povratnom analizom zavisi od toga kako alat funkcioniše. Alat može biti od velike pomoći ako se koristi kao dinamički čitač, ali ako proizvodi velike dokumente time samo uništava šume.

Mislim da je UML kao programski jezik dobra zamisao, ali sumnjam da će ikada zaživeti. Nisam uveren da su grafički jezici produktivniji od tekstualnih za većini zadataka programiranja, a ako i jesu, veoma je teško da jezik bude široko prihvaćen.

Kao posledica mojih ubeđenja, ova knjiga se usredsređuje na upotrebu jezika UML za pravljenje skica. Srećom, za takvu primenu jezika može se napraviti sažet vodič. Ne bih mogao opisati druge načine korišćenja UML-a u knjizi ovog obima, ali zato knjiga predstavlja dobar uvod u druge, obimnije. Ako se zainteresovani za druge primene jezika UML, preporučujem da ovu knjigu čitate kao uvodnu, a zatim da pređete na ostale. Ako vas interesuju samo skice, ova knjiga može biti sasvim dovoljna.

Kako smo došli do jezika UML

Priznajem da sam zaluđen istorijom. Moja omiljena lagana literatura je dobra knjiga o istoriji. Svestan sam i toga da ona nije svima zabavna. Ovde govorim o istoriji, pošto mislim da je teško razumeti ulogu UML-a ako se ne shvati njegov nastanak.

Osamdesetih godina prošlog veka, objekti su počeli da napuštaju istraživačke laboratorije i napravili su prve korake ka „stvarnom“ svetu. Smalltalk se pretvorio u upotrebljivu platformu, a nastao je i C++. U tom periodu razni ljudi su počeli da razmišljaju o objektno orijentisanim grafičkim jezicima za projektovanje.

Ključne knjige o objektno orijentisanim grafičkim jezicima za modelovanje pojavile su se između 1988. i 1992. godine. Vodeće ličnosti bile su: Grady Booch [Booch, OOAD], Peter Coad [Coad, OOA], [Coad, OOD], Ivar Jacobson (Objectory) [Jasobcon, OOSE], Jim Odell [Odell], Jim Rumbaugh (OMT) [Rumbaugh, insights], [Rumbaugh, OMT], Sally Shlaer i Steve Mellor [Shlaer and Mellor, data], [Shlaer and Mellor, states] i Rebecca Wirfs-Brock (projektovanje vođeno odgovornostima) [Wirfs-Brock].

Svaki od navedenih autora nezvanično je predvodio grupu praktičara kojima su se dopadale njegove ideje. Sve te metode bile su veoma slične, ali su ipak obuhvatale izvestan broj malih, a uznemiravajućih, razlika. Elementi koji su predstavljali iste pojmove znatno su se razlikovali, što je često zbunjivalo moje klijente.

Tokom tog perioda previranja istovremeno se govorilo o standardizaciji, ali se to pitanje i zanemarivalo. Jedan tim iz grupe OMG pokušao je da se usredsredi na standardizaciju, ali je dobio samo otvoreno protestno pismo od svih ključnih metodologa. (Ovo me podseća na jedan stari vic: U čemu je razlika između metodologa i teroriste? – Sa teroristom se može pregovarati.)

Kataklizmički događaj koji je stvorio UML zbio se kada je Jim Rumbaugh napustio General Electric i pridružio se Grady Boochu u kompaniji Rational (sada je to deo IBM-a). Od početka je uočeno da udruženje Booch/Rumbaugh

može osvojiti odlučujući deo tržišta. Grady i Jim su objavili: „Rat metoda je završen – mi smo pobedili“, u osnovi najavljujući da nameravaju da standardizuju modelovanje na „Microsoftov način“. Izvestan broj drugih metodologa predložio je formiranje anti-Booch koalicije.

Grady i Jim su, do konferencije OOPSLA ‘95, pripremili prvi javni opis svog objedinjenog razvojnog postupka: to je bila verzija 0.8 dokumenta *Objedinjeni metod* (engl. *Unified Method*). Još je značajnije bilo to što su objavili da je kompanija Rational Software kupila firmu Objectory i da će se Ivar Jacobson pridružiti timu. Rational je priredio dobro posećenu zabavu da bi proslavio objavljivanje verzije 0.8 nacrta dokumenta. (Centralni događaj zabave bilo je prvo javno pevanje Jima Rumbaugha – svi se nadamo da je to bilo i poslednje.)

Sledeće godine se pojavio otvoreniji razvojni postupak. Rational je morao da uključi Ivarove ideje, a posvetio je vreme i drugim partnerima. Još je značajnije to što je grupa OMG odlučila da preuzme vodeću ulogu u razvoju objedinjenog razvojnog postupka.

Važno je razumeti zašto je uključena grupa OMG. Metodolozi su, kao autori knjiga, skloni visokom mišljenju o sebi. Međutim, ne verujem da bi krici pisaca knjiga ikada doprli do grupe OMG. Grupa je angažovana zbog buke koju su stvorili proizvođači alata, uplašeni da će standard kojim upravlja Rational pružiti alatima ove firme nepravednu prednost na tržištu. Proizvođači su podstakli OMG da nešto preduzme, pod izgovorom poboljšanja saradnje CASE alata. Tema je bila značajna i grupa OMG se bavila isključivo tom spregom. Postojala je ideja da se napravi objedinjen jezik za modelovanje koji će omogućavati razmenu modela između CASE alata.

Mary Loomis i Jim Odell vodili su prvu radnu grupu. Odell je jasno rekao da je spreman da prepusti svoj metod standardizaciji, ali da ne želi standard koji nameće Rational. U januaru 1997. godine, razne organizacije su priložile predloge standarda za metode da bi se olakšala razmena modela. Kompanija Rational je sarađivala s većim brojem drugih organizacija i, kao svoj predlog, objavila je verziju 1.0 dokumenta o jeziku UML, u kome se prvi put pominje naziv objedinjenog jezika za modelovanje.

Usledio je kratak period natezanja dok nisu spojeni različiti predlozi. Grupa OMG je usvojila verziju 1.1 kao svoj zvanični standard. Kasnije je standard izmenjen. Revizija 1.2 bila je u potpunosti kozmetička. Revizija 1.3 bila je značajnija. U reviziji 1.4 dodat je izvestan broj detalja u vezi s komponentama i profilima. U reviziji 1.5 dodata je semantika aktivnosti.

Uglavnom se smatra da su Grady Booch, Ivar Jacobson i Jim Rumbaugh tvorci jezika UML. O njima se uglavnom govori kao o Tri amigosa, mada šaljivdžije vole da ispuste prvi slog druge reči. (Igra reči – *three egos* umesto *three amigos* – kao aluzija na izražene individualnosti ove trojice, prim. prev.). Iako se oni

smatraju najzaslužnijim za UML, mislim da to nije sasvim pravedno. Notacija jezika UML nastala je u objedinjenom metodu Boocha i Rumbaugha. Od tada su nosioci posla uglavnom bili timovi grupe OMG. Tokom kasnijih faza, samo Jim Rumbaugh je potpuno bio posvećen ovom poslu. Po mom mišljenju, najveću zaslugu za UML imaju članovi tima koji su razvijali UML.

Notacije i metamodeli

Jezik UML definiše notaciju i metamodel. **Notacija** je skup grafičkih elemenata koji se koriste u modelima; to je grafička sintaksa jezika za modelovanje. Na primer, notacija dijagrama klasa definiše kako se predstavljaju elementi i koncepti kao što su klasa, asocijacija i kardinalnost.

Naravno, ovo dovodi do pitanja šta se tačno podrazumeva pod asocijacijom, kardinalnošću, pa čak i klasom. Njihova uobičajena upotreba nameće neke neformalne definicije, ali većina ljudi očekuje veću preciznost.

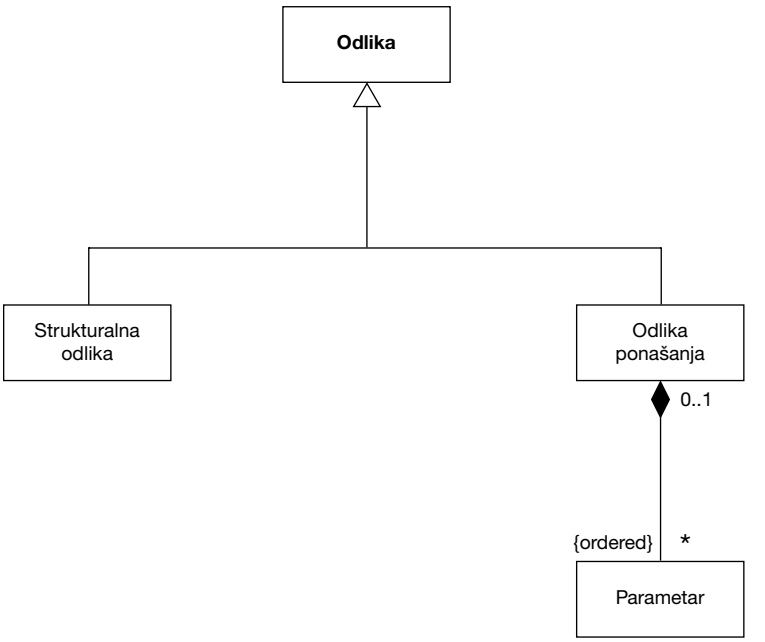
U formalnim metodama preovlađuju stroge specifikacije i precizni projekti . U takvim tehnikama projekti i specifikacije predstavljeni su primenom neke vrste predikatskog računa. Takve definicije su matematički stroge i vrlo su precizne. Međutim, značaj tih definicija nipošto nije univerzalan. Čak i ako možete da dokažete da neki program zadovoljava matematičku specifikaciju, ne postoji način da dokažete da ta matematička specifikacija odgovara stvarnim zahtevima sistema.

Najveći broj grafičkih jezika za modelovanje nije strog, a njihova notacija više odgovara intuiciji nego formalnoj definiciji. U celini, od toga nema naročite štete. Metode mogu biti neformalne, ali ih većina ljudi smatra korisnim, a upravo to je važno.

Međutim, metodolozi traže načine da povećaju strogost metoda bez narušavanja njihove upotrebljivosti. Jedno rešenje je da se definiše **metamodel**: to je dijagram, obično dijagram klasa, koji definiše koncepte jezika.

Slika 1.1, delić metamodela jezika UML, pokazuje odnose između odlika. (Ovaj primer treba da vam nagovesti kako izgledaju metamodeli. Neću ni pokušavati da ga objašnjavam.)

U kojoj meri metamodel utiče na korisnika notacije za modelovanje? Odgovor uglavnom zavisi od načina korišćenja. Korisnik koji pravi skice uglavnom ne obraća pažnju na metamodel, za razliku od onoga koji pravi projekte. Metamodel je od vitalnog značaja za korisnike UML-a kao programskog jezika, pošto definiše njegovu apstraktnu sintaksu.



Slika 1.1 *Delić metamodela jezika UML*

Ljudi koji razvijaju UML uglavnom su zainteresovani za metamodel, naročito zbog njegovog značaja za korišćenje UML-a kao programskog jezika. Pitanja notacije se često nalaze na drugom mestu, što treba da imate na umu ako ikada pokušate da se upoznate sa zvaničnom dokumentacijom.

Što dublje zalazite u detalje upotrebe jezika UML, shvatate da vam treba mnogo više od grafičke notacije. Zato su alati za korišćenje tog jezika vrlo složeni.

U ovoj knjizi nisam preterano strog. Dajem prednost tradicionalnim metodima i oslanjam se pretežno na vašu intuiciju. To je prikladno za sažetu knjigu poput ove, čiji je autor uglavnom naklonjen upotrebi skica. Ako hoćete da koristite strožije metode, uzmite detaljniju knjigu.

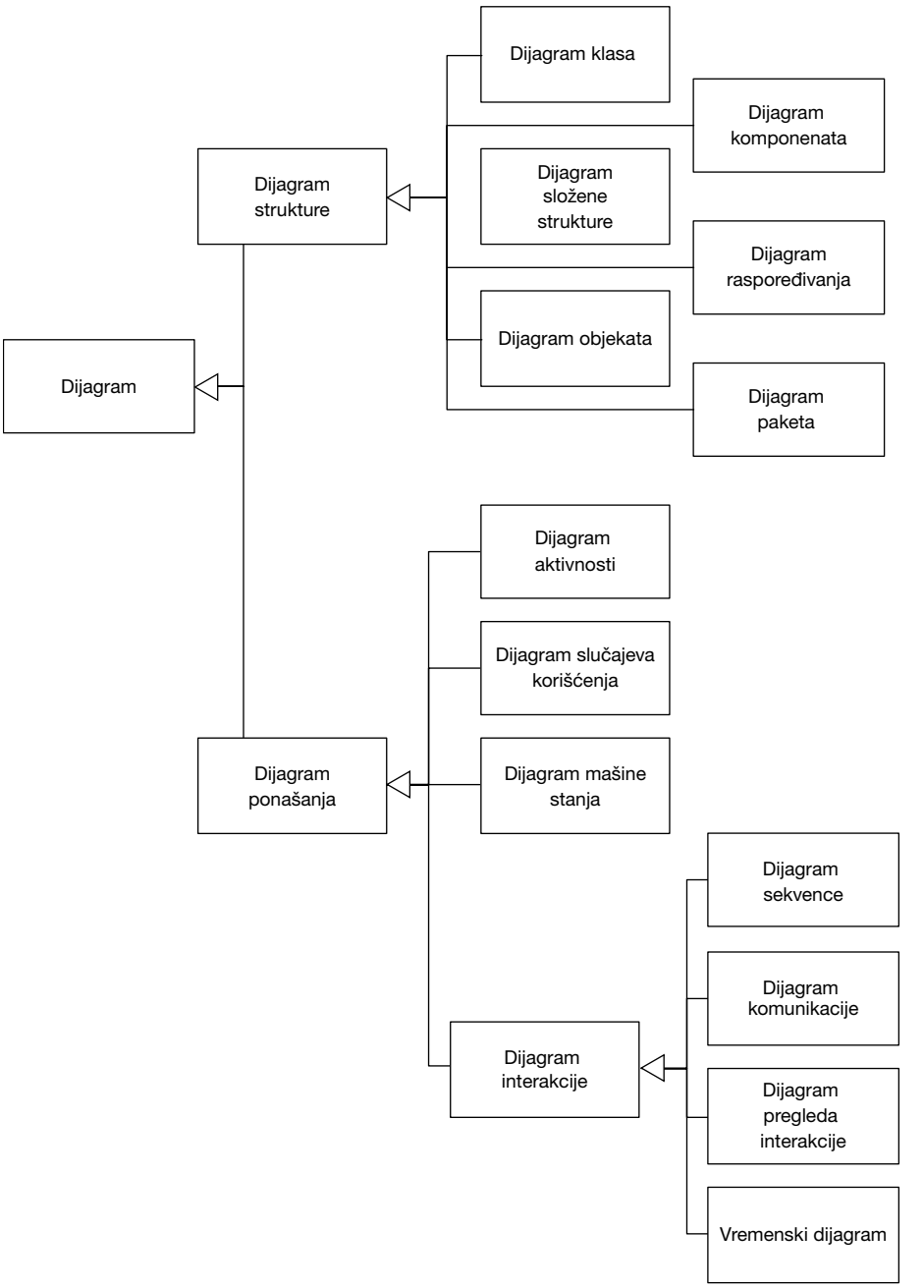
Dijagrami jezika UML

UML 2 opisuje 13 zvaničnih tipova dijagrama nabrojanih u tabeli 1.1 i klasifikovanih kao što je prikazano na slici 1.2. Iako navedeni tipovi dijagrama

predstavljaju način na koji mnogi ljudi pristupaju jeziku UML i na koji sam organizovao ovu knjigu, autori jezika UML ne vide dijagrame kao središnji deo jezika. Zato tipovi dijagrama nisu naročito strogi. Često je ispravno da koristite elemente jednog tipa dijagrama u drugom dijagramu. Standard jezika UML ukazuje na to da se neki elementi uglavnom crtaju na dijagramima određenih tipova, ali to nije strogo propisano.

Tabela 1.1 *Zvanični tipovi dijagrama jezika UML*

Dijagram	Poglavlja knjige	Namena	Uvedeno u verziji
Aktivnosti	11	Proceduralno i paralelno ponašanje	UML 1
Klasa	3, 5	Klase, odlike i veze	UML 1
Komponenata	14	Struktura i veze komponenata	UML 1
Komunikacije	12	Interakcija između objekata; naglasak na vezama	Dijagram kolaboracije verzije UML 1
Mašine stanja	10	Kako događaji menjaju objekat tokom njegovog postojanja	UML 1
Objekata	6	Primeri konfiguracije instanci	Nezvanično u verziji UML 1
Paketa	7	Hijerarhijska struktura tokom prevođenja	Nezvanično u verziji UML 1
Pregleda interakcije	16	Kombinacija dijagrama sekvence i aktivnosti	Novo u verziji UML 2
Raspoređivanja	8	Raspoređivanje artefakta po čvorovima	UML 1
Sekvence	4	Interakcija između objekata; naglasak na sekvenci	UML 1
Složene strukture	13	Razlaganje klasa tokom izvršavanja	Novo u verziji UML 2
Slučajeva korišćenja	9	Interakcija korisnika i sistema	UML 1
Vremenski	17	Interakcija između objekata; naglasak na vremenskoj promeni	Novo u verziji UML 2



Slika 1.2 *Klasifikacija tipova dijagrama jezika UML*

Kakvi dijagrami su ispravni?

Na prvi pogled, trebalo bi da bude jednostavno odgovoriti na ovo pitanje: ispravni su dijagrami koji su u specifikaciji definisani kao dobro formirani. U praksi je odgovor nešto složeniji.

Značajan deo ovog pitanja je da li UML ima opisna ili propisana pravila. Jezikom koji ima **propisana pravila** (engl. *prescriptive rules*) upravlja zvanično telo koje ustanovljava šta jeste a šta nije dozvoljeno u tom jeziku i koja su značenja iskaza tog jezika. **Opisna pravila** (engl. *descriptive rules*) jezika možete razumeti posmatrajući njihovu primenu u praksi. Pravila programskih jezika uglavnom su propisana, a ustanovljava ih grupa za standardizaciju ili vodeći proizvođač softvera, dok su pravila prirodnih jezika, kao što je engleski, opisna i njihova značenja su usvojena dogovorom.

UML je prilično precizan, pa se može očekivati da ima propisana pravila. Međutim, često se smatra softverskim ekvivalentom projekata iz drugih inženjerskih disciplina, a ti projekti nisu zasnovani na propisanim notacijama. Nijedna grupa za standarde ne određuje šta su dozvoljeni simboli na strukturnom tehničkom crtežu, već se notacija usvaja dogovorom. Ni zvanično telo za standardizaciju ne može rešiti taj problem, pošto stručnjaci za određenu oblast ne moraju slediti sve što to telo kaže. Osim toga, jezik UML je tako složen da je njegov standard često podložan različitim tumačenjima. Čak se ni vodeći stručnjaci za UML, koji su pregledali ovu knjigu, ne slažu oko tumačenja standarda jezika.

Ovo pitanje je značajno i za mene koji pišem ovu knjigu, i za vas koji koristite UML. Ako hoćete da razumete UML dijagram, važno je da shvatite da razumevanje standarda jezika ne daje potpunu sliku. Ljudi usvajaju konvencije, kako u branši, tako i u okviru pojedinačnog projekta. Shodno tome, iako standard jezika UML može biti osnovni izvor informacija, ne može biti i jedini.

Smatram da su za većinu korisnika pravila objedinjenog jezika za modelovanje opisna. Standard jezika UML ima najveći, ali ne i isključivi, uticaj na značenje jezika. Mislim da će to naročito doći do izražaja u verziji 2, koja uvodi neke konvencije notacije, suprotstavljene definicijama verzije 1 ili uobičajenoj upotrebi jezika UML, a doprinosi i složenosti. Zato u ovoj knjizi nastojim da prikažem UML onako kako ga vidim: i standarde i uobičajenu upotrebu. Kada budem morao da napravim razliku, koristiću izraz **konvencionalna upotreba** da bih ukazao na nešto što nije deo standarda, ali je, po mom mišljenju, u širokoj upotrebi. Koristiću izraze **standardni** ili **normativni** za ono što odgovara standardu. (Za one koji se bave standardima, izraz normativni označava nešto što se mora

ispoštovati da bi odgovaralo standardu. Tako je izraz nenormativni zgodan način da se kaže da je nešto strogo zabranjeno po standardu jezika UML.)

Kada posmatrate neki UML dijagram, setite se opšteg principa jezika da se bilo koja informacija može **izostaviti** sa dijagrama. Izostavljanje može biti opšte (skrivanje svih atributa) ili pojedinačno (ne prikazuju se, na primer, neke tri klase). Zato iz dijagrama ne možete izvoditi zaključke o elementima kojih nema. Ako nedostaje kardinalnost, ne možete zaključiti koje bi vrednosti bila. Čak i ako metamodel jezika UML ima podrazumevanu vrednost, na primer [1] za atribute, a ne vidite je na dijagramu, to može biti zato što se vrednost podrazumeva ili zato što je namerno izostavljena.

Postoje opšte konvencije, na primer ta da svojstva sa većim brojem vrednosti predstavljaju skupove. Takve podrazumevane konvencije ću istaći.

Važno je da ne gubite previše vremena na usklađenost sa standardom ako UML koristite za skice ili projekte. Važnije je da imate dobar opis sistema, a ja bih dao prednost dobrom projektu na nezvaničnom jeziku za modelovanje nego lošem projektu na zvaničnom jeziku. Naravno, najbolje je ono što je dobro a standardno, ali je bolje da uložite svoju energiju u dobar projekat nego da brinete o sitnim pojedinostima UML-a. (Naravno, ako koristite UML kao programski jezik morate poštovati pravila, inače se program neće ispravno izvršavati!)

Značenje jezika UML

Iako specifikacija detaljno opisuje dobro formiran UML, ona ne govori mnogo o značenju UML-a izvan uzvišenog sveta metamodela, što je neobično. Ne postoji nikakva formalna definicija o preslikavanju UML-a u bilo koji programski jezik. Kada vidite UML dijagram, ne možete *tačno* reći kako bi izgledao ekvivalentan kôd ali možete imati *okvirnu ideju* o tome kako bi on izgledao. U praksi je to dovoljno korisno. Razvojni timovi često uspostavljaju i koriste sopstvene konvencije, pa treba da upoznate i njih.

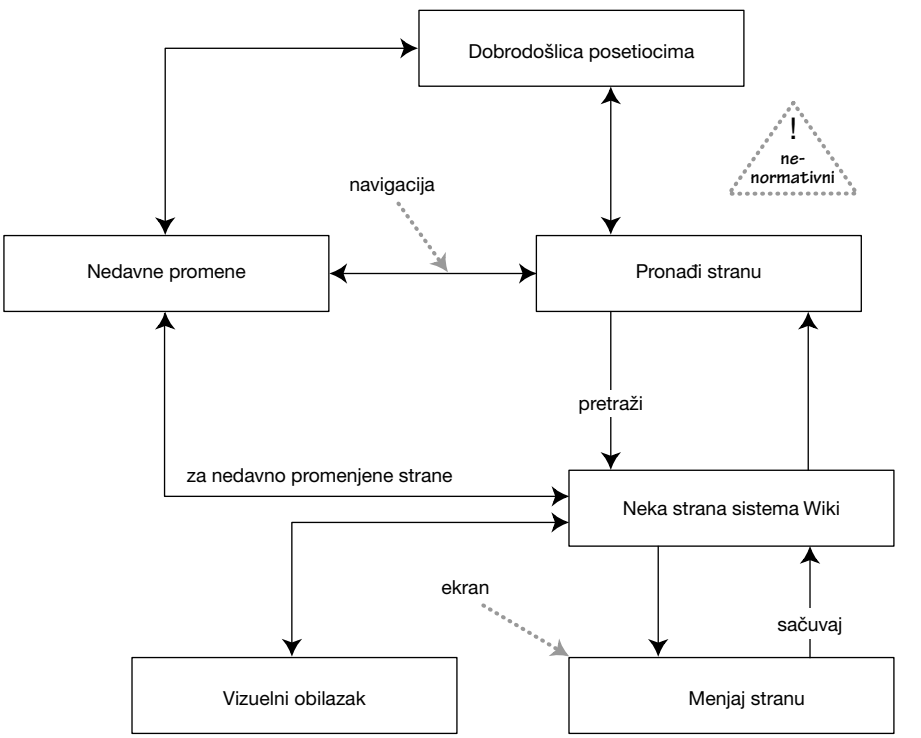
UML nije dovoljan

Iako UML propisuje veći deo različitih dijagrama koji pomažu da se definiše aplikacija, on ne daje kompletnu listu svih korisnih dijagrama koji bi vam mogli

zatrebati. U mnogim situacijama mogu biti korisni različiti dijagrami, te ne bi trebalo da oklevate da iskoristite dijagram koji ne pripada jeziku UML ako nijedan od UML-ovih ne odgovara vašim potrebama.

Na slici 1.3 prikazan je dijagram toka ekrana (engl. *screen flow diagram*), koji obuhvata različite ekrane korisničkog okruženja i načine prelaska s jednog na drugi. Već godinama srećem i koristim dijagrame tokova ekrana. Do sada sam video samo grubu definiciju njihovog značenja – u jeziku UML ne postoji ništa slično, a smatram da su veoma korisni.

Tabela 1.2 prikazuje tabelu odlučivanja (engl. *decision table*), još jedno sredstvo koje rado koristim. Tabele odlučivanja su dobar način da se prikažu složeni logički uslovi. Isto možete postići dijagramom aktivnosti, ali je tabela kompaktnija i jasnija u složenijim slučajevima. Postoje mnogi oblici tabela odlučivanja. Tabela 1.2 podeljena je u dva odeljka: uslovi su iznad, a posledice su ispod dvostruke linije. Svaka kolona pokazuje kako određena kombinacija uslova dovodi do odgovarajućeg skupa posledica.



Slika 1.3 *Neformalan dijagram toka ekrana za deo sistema Wiki (http://c2.com/cgi/wiki)*

Tabela 1.2 *Tabela odlučivanja*

Specijalan kupac	X	X	D	D	N	N
Prioritetna porudžbina	D	N	D	N	D	N
Međunarodna porudžbina	D	D	N	N	N	N
Uplaćeno	$150	$100	$70	$50	$80	$60
Posebna dostava	•	•	•			

Na nestandardne dijagrame nailazićete u različitim knjigama. Ne oklevajte da isprobate tehnike koje izgledaju prikladno za dati projekat. Ako su vam korisne, upotrebite ih. Ako nisu, zanemarite ih. (Naravno, isti savet važi i za dijagrame jezika UML.)

Odakle početi s jezikom UML

Niko ne razume niti koristi UML u celosti, čak ni njegovi tvorci. Najveći broj ljudi koristi mali podskup jezika UML. Morate pronaći podskup koji odgovara vama i vašim saradnicima.

Ako tek učite UML, preporučujem da se prvo usredsredite na osnovne oblike dijagrama klasa i dijagrama sekvence. Ovi dijagrami se najčešće koriste, a smatram da su i najkorisniji.

Kada savladate osnove, počnite da koristite napredniju notaciju dijagrama klasa, a pogledajte i druge tipove dijagrama. Eksperimentišite s dijagramima i ustanovite koliko su vam korisni. Ne ustručavajte se da odbacite tipove dijagrama koji vam ne pomažu u poslu.

Preporučena literatura

Ova knjiga nije potpun i konačan referentni priručnik za UML, a još manje za objektno orijentisanu analizu i projektovanje. Literatura je obimna i mnogo toga bi valjalo pročitati. Tokom razmatranja pojedinačnih tema, navodim i druge knjige u kojima možete da pronađete opširnije informacije. Navešću neke knjige o jeziku UML i objektno orijentisanom projektovanju.

Kao i za sve preporučene knjige, treba da proverite za koju verziju jezika UML su napisane. Nijedna knjiga izdata do juna 2003. ne opisuje UML 2.0, što ne

iznenađuje, pošto se mastilo kojim je ispisan standard tek bilo osušilo. Knjige koje preporučujem su dobre, ali vam ne mogu reći da li su, niti da li će biti izmenjene i dopunjene za standard verzije 2 jezika UML.

Ako ste početnik u korišćenju objekata, preporučujem vam s uvodnu knjigu koja mi je omiljena: [Larman]. Dobro je da sledite autorov pristup strogo vođen odgovornostima.

Zaključnu reč o jeziku UML potražite u zvaničnoj dokumentaciji standarda, ali ne zaboravite da je ona napisana za zaluđene metodologe. Lakše svarljivu verziju standarda pronaćićete u knjizi [Rumbaugh, UML Reference].

Ako vam trebaju detaljni saveti o objektno orijentisanom projektovanju, mnoge dobre tehnike možete naučiti iz knjige [Martin].

Preporučujem da pročitate knjige o projektnim obrascima, koje će vas odvesti dalje od osnovnih pojmova. Pošto je rat metoda završen, najveći deo zanimljivih knjiga o analizi i projektovanju bavi se obrascima (strana 27).