3. Koncepti Inženjerske informatike / Modeli mišljenja

U prošlosti su, kao što je u uvodu napomenuto, kompjuteri pre svega korišćeni za „računanje". Ime „Computer" (engl. to compute = računati) upravo to objašnjava. Stoga je evidentno da su u prošlosti za rad na računaru u prvom redu korišćeni modeli mišljenja koji su bili orijentisani na „računanje", tj. na numeričke operacije, a time i na algoritme. Stoga je klasični model mišljenja Informatike / Inženjerske informatike postavljen algoritamski.

Ali, kao što je već primećeno, računari su se poslednjih godina značajno promenili. Oni više nisu „računske mašine" već su više multifunkcionalno primenljivi sistemi za radno mesto (Oblasti primene su obrada teksta, grafika, banke podataka, sistemi bazirani na znanju, itd.). Zato je razumljivo da su se pored klasičnog „algoritamskog modela" razvili i drugi modeli mišljenja. Različiti, trenutno međusobno konkurentni modeli mišljenja moraju moderno obrazovanom inženjeru biti poznati; ovi će stoga u nastavku biti kratko predstavljeni.

3.1. Algoritamski model mišljenja

Osnovni princip rešavanja problema u klasičnom algoritamskom modelu mišljenja je, kao što i samo ime kaže, algoritam. Izrazom

Algoritam

se označava

tačan opis šematski/mehanički izvršivog računskog postupka,

pri čemu je reč „Algoritam" izvedena „akustički" iz imena poznatog persijskog matematičara

Al-Chowarizmi (oko 750. g. nove ere),

koji je napisao značajnu knjigu iz algebre.

Algoritam se označava kao niz naredbi (= elementarni nalozi ili imperativne operacije). Zbog upotrebe naredbi govori se takođe o imperativnom modelu razmišljanja.

Primer: **Euklidov algoritam** (jedan od najstarijih algoritama) za izračunavanje najvećeg zajedničkog delioca - NZD (eng: BGD – Biggest Common Divider, nem: ggT, tj. größter gemeinsamer Teiler) dva proizvoljna prirodna broja p i q $(p \geq q)$; na primer, NZD(24,18)=6.

Euklidov algoritam za NZD(p, q)

Korak 1: Podeliti p sa q ;
ostatak označiti sa r (naredbom Moduo);
Dobijanje ostatka znači celobrojno deljenje, na primer 24:18=1 sa ostatkom 6.

Korak 2: ako je $r=0$, tada je q traženi NZD i idemo na Kraj;
ako je $r \neq 0$, tada stavimo
$p = q$; $q = r$;
idemo nazad na Korak 1.

Konkretan slučaj (statički proces)

p = 378; q = 216; na početku algoritma važi:
p = 378; q = 216; r = neoznačeno

Korak 1: p : q = 378:216 = 1 sa ostatkom 162 → r = 162

Korak 2: r = 0 ? → ne
$p = q$ = 216; $q = r$ = 162
nazad na Korak 1;

Korak 1: p : q = 216:162 = 1 sa ostatkom 54 → r = 54

Korak 2: r = 0 ? → ne
$p = q$ = 162; $q = r$ = 54
nazad na Korak 1;

Korak 1: p : q = 152:54 = 3 a ostatkom 0 → r = 0

Korak 2: r = 0 ? → da
tj. NZD = q = 54
Kraj algoritma.

Ovaj jednostavni primer očigledno pokazuje da se algoritam sastoji od više potpuno uređenih koraka i da se unutar algoritma može skakati sa naredbe na naredbu (ponavljanja i iteracije su takođe mogući).

Kod velikih problema se odgovarajući algoritmi dobijaju koristeći

Analizu problema

koja predstavlja editovanje odgovarajuće postavke problema s obzirom na algoritamsko rešenje. Ovde je preporučljivo algoritme ili delove algoritama povezati u pregledne, veće celine, koje se često nazivaju „Procedure". Zato se, pored algoritamskog, tj. imperativnog načina mišljenja, govori i o proceduralnom načinu mišljenja, zavisno od toga iz koje ravni se problem posmatra.

Tek transformacija algoritma u jedan oblik razumljiv računaru pomoću specijalnog jezika (nekog programskog jezika), vodi konačno izrazu

Program.

Kako se pri izvršenju programa obrađuju podaci (brojevi, znakovi, simboli), sledeća jezička jednačina opisuje izraz „Program":

Program = Algoritam + Podaci,

ili, preciznije:

Program = { **Algoritmi (izraženi u nekom programskom jeziku)** + **odgovarajući podaci za taj algoritam**

Da bi se program funkcionalno izvršavao, moraju se ispuniti sledeći važni zahtevi u odnosu na algoritam i na od njega primenjenih podataka.

Zahtevi u odnosu na algoritam:

(1) Sve naredbe jednog algoritma moraju biti izvršive.
(naredbe, tipa, na primer, „izračunati NZD" ili „izračunati nule" su suviše opšte i stoga direktno neizvršive),

(2) Algoritam mora biti jednoznačan.
(tj. isti ulazni podaci moraju dati iste rezultate),

(3) Algoritam mora biti konačan.
(tj. beskonačne petlje kao konstrukcije daju nekorektan algoritam),

(4) Algoritam mora biti potpun.
(ne sme se primeniti pre nego što se prethodno potpuno ne definiše/proračuna; primer „nepotpunog scenarija":
Putnik 1: Ja želim da siđem kod X !
Putnik 2: Siđite jednu stanicu pre mene !),

(5) **Algoritam treba <u>uopšteno</u> definisati**.
(tj. on mora važiti za što više vrsta ulaznih podataka).

Zahtevi u odnosu na podatke:

(1) Podaci sa kojima rade algoritmi moraju se koristiti u saglasnosti tipa (celobrojni, realni, imaginarni, oznake, itd.) sa veličinama korišćenim u algoritmu.
(2) Kompozitne strukture podataka (entiteti, polja, liste, itd.) su moguće samo u okviru uskih definicija primenjenih programskih jezika.

Između algoritma i podataka pritom postoji značajna razlika, tj. kod algoritamskog modela mišljenja dominira algoritam. Podaci su samo pasivni elementi programa.

3.2. Drugi konkurentni modeli mišljenja

U algoritamskom modelu mišljenja su - kao što smo upravo videli – algoritmi centralni element modelisanja. Sa manje-više sekundarnim podacima se „nešto obavlja": algoritam se koncentriše na operacije, preuzima ulazne podatke na obradu i generiše rezultate, ali preuzeti podaci nisu direktno sastavni deo algoritma: prema tome, podaci se „samo" obrađuju. Algoritam je „trajan", dok su podaci „prolazni".

Algoritamska paradigma (tj. koncept i metodika rešenja) je stoga uvek primenljiva i čak izvanredno dobra kada se postavke problema mogu rešiti tako da se iz unapred datih podataka (ulaznih podataka) po jednom tačno utvrđenom postupku traženi rezultati mogu „izračunati". Vrlo mnogo matematičkih, inženjersko-tehničkih ili čak komercijalnih primena je u suštini tako formulisano (numerička postavka problema).

Detaljniji pogled na inženjerske probleme pokazuje da u toj oblasti postoji značajan broj drugačije postavljenih problema za koje je algoritamska paradigma samo delimično, ili nikako podesna. U tim slučajevima su pogodni drugi modeli razmatranja. Dajemo nekoliko karakterističnih primera:

Održavanje velikih struktura podataka

Kao zamena za tradicionalne stokove akata sređene u obliku registratura, uvedene su u inženjerstvu, u mnoge oblasti, banke podataka, u poslednje vreme zvane „Databank-Management-Systems", skraćeno „DBMS". Ovde se radi o dijametralnoj situaciji u odnosu na algoritamski model: podaci su dugotrajni i primarni, dok se algoritmi za održavanje i obradu (traženje, brisanje, memorisanje, izmenu, preuređenje) koriste, ali su od sekundarnog značaja, dakle „prolazni" su. Saglasno, ovde se govori o „modelu orijentisanom ka podacima".

Obrada znanja koje se može formalno definisati

Poslednjih godina se računari sve više primenjuju za obradu „znanja", ukoliko se ono može formalizovati. U inžinjerstvu se na primer može pokazati kako se od datih normi mogu formirati pravila u obliku „Ako - Tada" (tj. kao veze ili relacije između stvari koje mogu biti „tačne" ili „netačne"). Ova pravila se tada zajedno sa podacima nekog problema obrađuju pomoću algoritma zaključivanja (inferencne mašine), koji su ranije razvijeni za određene klase problema i time „potvrđeni". Rešenja problema se zatim obrađuju pomoću „logičkog zaključivanja", tako da se odgovarajući koncept rešenja označava kao „logički model razmišljanja". Na primer, programski jezik PROLOG podržava specijalne oblike logičkog modela na prirodni način. I ovde se pojavljuju algoritmi (na primer, u delovima pravila), ali ne oni, već logičke reprezentacije opisa problema, definišu koncept rešenja.

Obrada simboličkih izraza

Želja da se u oblasti tehničkih primena matematički izrazi obrađuju u zatvorenoj analitičkoj, tj. simboličkoj formi, vodi ka takozvanom „funkcionalnom modelu". Tipičan primer je diferenciranje jedne funkcije jedne promenljive:

$f(x) = x^3 + x^2$ (Izlazni problem)

$f'(x) = 3x^2 + 2x$ (Rešenje u simboličkom obliku)

Osnova funkcionalnog modela je uopšteni matematički izraz za funkciju (odatle „funkcionalni"), koji – slično algoritmu u algoritamskom modelu – preuzima centralnu ulogu u funkcionalnom modelu. LISP je, na primer, programski jezik koji radi isključivo sa funkcijama; rešenja se ovde predstavljaju kao smisaono uređeni niz poziva funkcija.

Objektno-orijentisani model

Razmotreni slučajevi objašnjavaju kako su drugi modeli, pored algoritamskih, pogodni za rešavanje na računaru. Posebno u inženjerstvu postoje važne postavke problema, koje – izvan oblasti gore navedenih modela – zahtevaju jedan nezavisan model, koji – kako se sada čini – do sada navedene modele može načiniti „suvišnim". Ovde se radi o postavkama problema kod kojih se sa stvarima (objektima) opšte vrste „nešto radi". Ovde uvedeni model biće označen kao objektno-orijentisani model.

Primeri objektno-orijentisanog modela

Simulacija

Zadatak simulacionih programa je da opišu ponašanje objekata (na primer stanje napona i deformacija nosača, tj. nosećih konstrukcija) pomoću kompjutera.

CAD

Zadatak CAD (Computer Aided Design – Projektovanje pomoću računara) je vizuelna obrada modela zgrada ili terena, koji se sastoje od različitih „objekata", kojima se može dizajnirati konstrukcija i koji mogu sadržati kompleksne informacije.

Komunikacija čovek – mašina (interaktivna grafička korisnička ravan)

Programi na korisničkoj ravni nemaju izraženu svrhu da postignu „računske rezultate" već mnogo više operišu „objektima" u obliku „prozora", „ekranskih simbola", „prekidača", „klizača", itd. Tipični primeri za to su grafičke podrške standardnih operativnih sistema Windows 3.1, 3.11, Windows 95/98, Windows NT, itd.

Pri primeni gore navedenih programa brzo se ustanovilo da je pogodno operacije i podatke predstaviti i videti kao jedinstvo i nasuprot klasičnom algoritamskom modelu ili modelu orijentisanom ka podacima spojiti ih u „objekte". Tako za objekte važi sledeća rečenička jednačina:

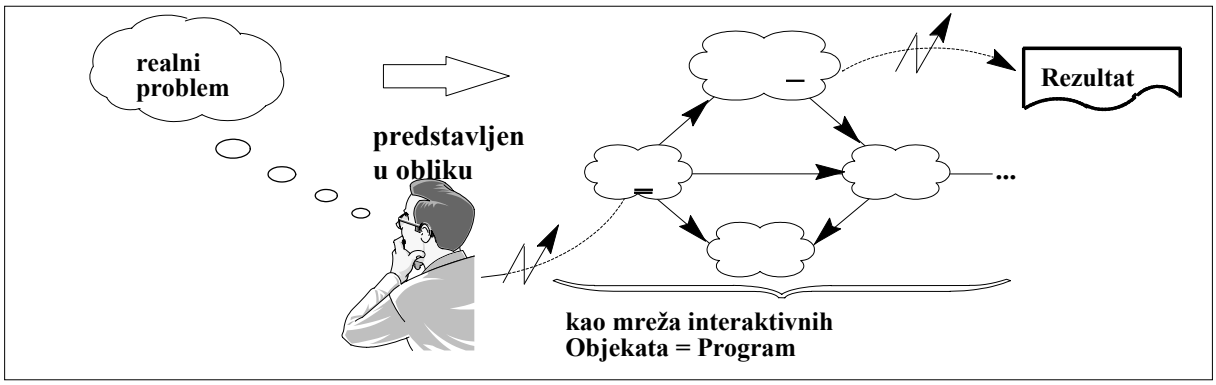
Objekat = Operacije + Podaci.

Objekti se, prema tome, ravnopravno i istovetno sastoje od podataka i operacija, koje se izvode nad tim podacima. Ili, drugačije rečeno: podaci jednog objekta su nerazdvojno povezani sa odgovarajućim operacijama. To vodi tome da podaci nisu više samo „pasivni", već su postali „aktivni".

Pri tome su objekti povezani sa drugim objektima pomoću **veza (asocijacije, relacije, linkovi).** Preko tih veza mogu se objekti kao elementi povezivati sa drugim objektima. Izvršenje programa u objektno-orijentisanom modelu se sastoji u tome da se aktivni objekti **izmenom novosti (message passing, Nachrichtenaustausch)** obraćaju ostalim objektima, generišu nove objekte, brišu nepotrebne objekte ili menjaju postojeće objekte.

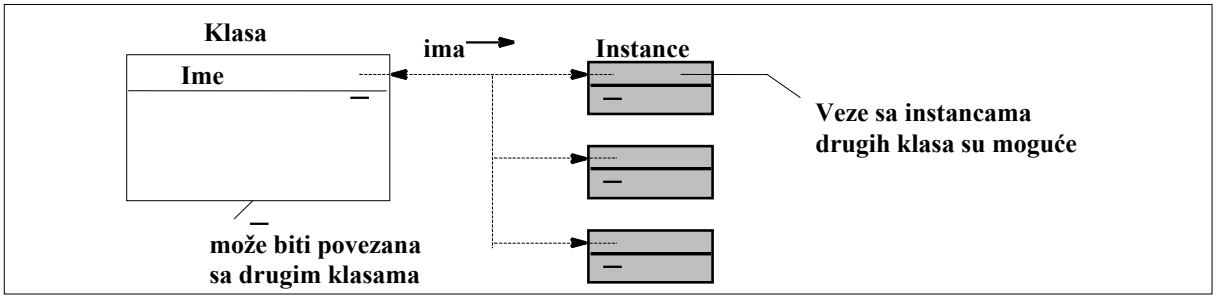
U zaključku imamo: u objektno-orijentisanom modelu problemi se rešavaju tako što se rešenje problema dobija komunikacijom u mreži međusobno komunikaciono povezanih objekata. Pri tome se simuliraju virtuelni „sociološki modeli" realnog sveta, gde se na sličan način (uporediti, na primer, scenarije pacijent/lekar, investitor/arhitekt/inženjer), komunikacijom rešavaju problemi.

Grafički se postupak može, prilično uprošćeno, ilustrovati na sledeći način:



U objektno-orijentisanim modelima pojavljuju se opet algoritmi: oni igraju kao **operacije,** tj. **metode** važnu ulogu – samo u drugačijem smislu nego kod algoritamskih modela. Algoritmi se predstavljaju u objektno-orijentisanim paradigmama kao metode, koje u poređenju sa algoritamskim modelom pokazuju neke specifičnosti, koje će u nastavku biti objašnjene.

Objekti iste vrste se pri modelisanju neke postavke problema spajaju u jednu nadređenu, apstraktnu strukturu, takozvanu **klasu objekata,** pri čemu su moguće različite forme grupisanja. Pri tome **klase** predstavljaju planove gradnje. Pri modelisanju praktičnih rešenja problema objekti realnog sveta se (automatski) izvode i generišu iz klasa i radi boljeg razlikovanja od svojih klasa označavaju kao **entiteti** (eng: **entity**, nem: **Instanzen**). Grafički se odnos između entiteta i klasa može predstaviti na sledeći način:



Pored utvrđivanja dodeljenih klasa i entiteta, posebno važnu ulogu igraju veze (tzv. relacije, asocijacije) sa drugim klasama i entitetima, pošto one sadrže važne sadržaje sa značenjem (semantika).

Objektno-orijentisani model omogućava da se vrlo kompleksni problemi obuhvate fleksibilno i koncizno; to je i razlog što objektno orijentisane tehnike dobijaju sve više na značaju i popularnosti pri postavci inženjerskih problema (CAD/CAE, Informacioni sistemi, Multimedijalna tehnika, itd.). Sve navedene oblasti su po svojoj strukturi upravo predodređene objektno orijentisanom pristupu. Šta više, u međuvremenu su razvijeni moćni objektno orijentisani programski jezici i razvijena njihova podrška, od kojih se posebno mogu navesti jezici **Object C**, **SMALLTALK**, **C++**, **Eiffel** i pre svih, **Java**.

Iz prethodnih razmatranja je očigledno da je **objektno-orijentisani model** pre svega nezavisan od nekog određenog računara (u smislu mašine). Sa druge strane, on je usko povezan sa tehničkom realizacijom, pošto se objekti i njihovi sadržaji (varijable, metode, relacije, itd.) moraju u računaru odslikati. Primena objektno orijentisanih modela stoga zahteva – isto kao i primena drugih modela – jedno osnovno poznavanje hardvera, ali takođe i znanje o tome, kako se hardver može pokrenuti, tj. poznavanje takozvanih operativnih sistema. S obzirom na činjenicu da se objektna orijentacija realizuje u vezi sa Internet-računarstvom, preporučljivo je takođe, čak i više nego kod drugih poznatih modela, poznavanje hardverskih i softverskih osnova strukture računarskih mreža.

„Hardver" i „Operativni sistemi" čine sadržaj sledeće dve glave. Tek kada najvažnije činjenice iz ove dve oblasti (Hardver → tehnička informatika; Operativni sistemi → praktična informatika) budu objašnjeni, može se preći na obradu objektno orijentisanih modela konkretnim **programskim jezikom JAVA**.