METODOLOGIJA NAČRTOVANJA NAMENSKIH RT SISTEMOV

INFORMATICA 4/92

Keywords: Real-time systems, embeeded systems,
software design, structured design, multi-tasking

Peter Kolbezen
Laboratorij za računalniške arhitekture
Institut Jožef Stefan, Ljubljana

V članku je opisan splošen pristop k načrtovanju namenskih računalniških sistemov za delo v realnem času. Vpeljane so tehnike načrtovanja programske opreme takšnih sistemov. Obravnavani so principi, ki so osnovani na dveh metodah: metodi MASCOT in metodi strukturnega načrtovanja RT sistemov. Poudarek je na delitvi programske opreme na module in na izvajanju več opravil hkrati.

DESIGN METHODOLOGY OF EMBEEDED REAL TIME SYSTEMS. In the paper the general approach to the design of embeeded real-time systems is described. Software design techniques for these systems are introduced. The principles uderlying two methods, MASCOT and real-time structured design, are covered. Emphasis is given to dividing the software into modules and to the use of multi- tasking.

1. UVOD

Namenski računalniški sistem imenujemo sistem za delo v realnem času, ker omogoča krmiljenje naprav v okolju v realnem času. Zato je sistem z napravami fizično povezan preko senzorjev, ki zaznavajo dogodke v okolju, ki ga nadzoruje in krmili preko prikazovalnikov oz. aktuatorjev. Računalnik, ki je tako vgrajen v sistem (embeeded system), zaznava signale, ki se sprožijo glede na zunanje dogodke, in se mora nanje pravočasno in pravilno odzivati. To pomeni, da mora računalnik zaznati vse dogodke, čeprav se lahko pojavljajo istočasno, in izvesti dogodkom ustrezna opravila v predvidenem času. V nasprotnem primeru se obnašanje namenskega sistema poruši in posledice so lahko katastrofalne. V primerih, ko igra odzivni čas sistema posebej pomembno vlogo, govorimo o namenskih sistemih, ki delajo v strogem realnem čau (hard real-time system). Takšen sistem, ki ga imenujemo tudi trdi, je na primer sistem vodenja in nadzora letala, medtem ko je nadzor parkirišča primer t.im. mehkega sistema (soft real-time system). V slednjem primeru so časovne zahteve bistveno milejše, tako v pogledu odzivne hitrosti kot v posledicah (izguba parkirnine).

Pospešen razvoj RT sistemov sega nazaj v leta 1970 /BIB74/ in še vedno narašča. Namenske sisteme dandanes srečujemo na vsakem koraku. Na komercialnem področju so namenjeni dinamični obdelavi velikih količin podatkov, kot na primer pri rezervacijah letalskih vozovnic ali v bančnem poslovanju. Tudi zunaj komercialnih področij so takšni sistemi učinkoviti, na primer v bolnicah, knjižnicah, šolah. Posebej pomembno vlogo pa igrajo v industriji, kjer računalniki upravljajo procese z elektronsko obdelavo podatkov. Računalniki omogočajo hitrejšo in natančnejšo analizo operacij in spremenljivk, kot so temperatura, pritisk, kvaliteta produkcije ipd. Hkrati so učinkovitejši od človeka pri zaznavanju nemotenega in nevarnega obnašanja procesa. Kmalu so si utrli pot tudi na druga področja, zlasti na področje vodenja plovil in najrazličnejših vozil, na področje raziskav vesolja ipd.

Računalniška materialna oprema je lahko preprost mikroprocesorski sistem, v katerem mikroprocesor zaznava signale, ki se sprožijo glede na zunanje dogodke. Ker je obnašanje namenskega sistema pogojeno z realnim časom, mora tudi računalniški sistem izvajati nadzor pravočasno. To pomeni, da mora zaznati vse dogodke, čeprav se pojavljajo tudi istočasno in izvesti vsa opravila za zaznane dogodke v predvidenem času. Sicer se obnašanje sistema poruši in posledice so lahko katastrofalne. Ker postajajo te zahteve bolj in bolj kompleksne, v namenskih sistemih vse pogosteje srečujemo večprocesorske strukture. Te največkrat predstavljajo porazdeljene večprocesorske sisteme

/Coo86/. Procesorji takega sistema izvajajo posamezne akcije (opravila) sočasno. Vendar često tudi v takšnih sistemih število akcij presega število možnih virov. Problemi te vrste se rešujejo s posebnimi tehnikami.

Pri načrtovanju namenskih (embedded) računalniških sistemov za delo v strogem realnem času uporabljamo splošne pristope k načrtovanju RT sistemov.

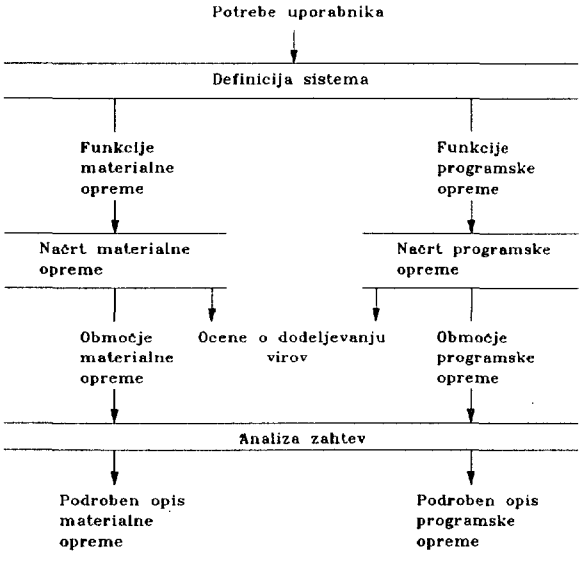
2. NAČRTOVANJE SISTEMA

Načrtovanju RT sistema se v splošnem ne razlikuje od načrtovanja namenskega sistema. V obeh primerih so faze načrtovanja razdeljene v dva dela:

- 1.faza: Faza planiranja sistema
- 2.faza: Faza razvoja

2.1. Faze načrtovanja

V fazi planiranja (slika 1) se raziščejo in natančno pojasnijo zahteve uporabnika, ki so potrebne za izdelavo podrobne specifikacije načrtovanega sistema in plana virov - ljudi, časa, naprav in razvojnih stroškov. V tej fazi (t.im. pripravljalni stopnji načrtovanja) se uvodno odločamo za porazdelitev funkcij med materialno in programsko opremo.



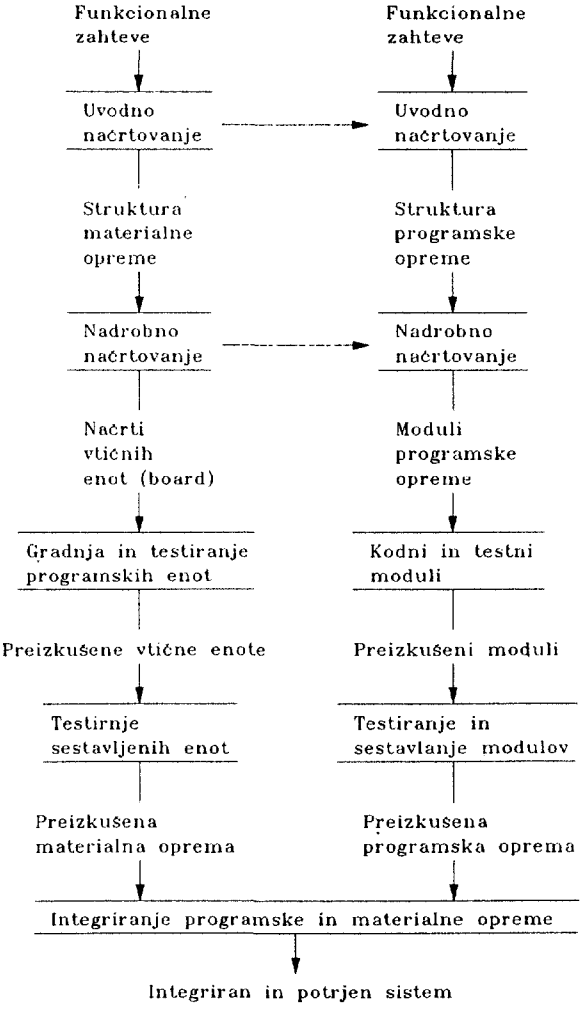
Slika 1. Faza planiranja.

V pripravljalni stopnji se najprej posvetimo izboru računalniške arhitekture. Odločamo se o eni od treh možnih variant:

- centralizirani sistem z enim samim centralnim računalnikom,
- hierarhičen sistem ali
- distribuirani sistem.

Rezultat te stopnje je dokument o specifikacijah sistema, ki jih izdela dobavitelj oziroma načrtovalec sistema. Izdela jih na osnovi dokumentov, ki vsebujejo zahteve, katerim naj bi sistem zadoščal. Zahteve poda uporabnik oziroma kupec sistema. V pripravljalni stopnji je bistveno, da so izdelani dokumenti popolni, detajlirani in povsem jasni. Izkušnje so namreč pokazale, da je pretežni del napak, ki se pojavljajo v končnem sistemu, posledica napačnih informacij v prvobitnih dokumentih, informacij z nejasnimi, nepopolnimi ali celo napačnimi specifikacijami.

Pogosto imamo skušnjavo, da rečemo "o tem vprašanju bo mogoče odločati kasneje". Posledica takšne odločitve lahko potegnejo za seboj tudi spremembe na drugih delih sistema, ki so že načrtani, pri naknadnem spreminjanju teh delov pa se pogosto vrinjajo napake. Priporoča se, da dokumentacija o specifikacijah sistema vsebuje tudi uporabniško dokumentacijo. Vsekakor pa mora biti taka dokumentacija natančno preverjena skupaj z



Slika 2. Stopnje razvojne faze.

uporabnikom (kupcem) že pred začetkom druge faze.

Razvojna faza je prikazana na sliki 2.

Cilj uvodne faze načrtovanja je dekompozicija sistema v množico specifičnih podopravil, ki se lahko obravnavajo ločeno. Načrtovanje v tej fazi poteka na visokem nivoju. To pomeni, da izhajamo iz visokonivojskih specifikacij. Rezultati te faze so globalne podatkovne strukture in visokonivojske programske arhitekture. Načrtovanje na tej stopnji zahteva, da obstaja tesna povezanost med načrtovalci materialne in programske opreme. To je še posebej pomembno za sisteme, pri katerih so možni popravki predhodnih odločitev o računalniški arhitekturi. Pri uporabi distribuiranega sistema so te vrste popravki potrebni v pogledu števila procesorjev, komunikacij (prepustnost, tip arhitekture), ipd. Na zaključku faze uvodnega načrtovanja mora biti izdelana kritična analiza celotne opreme, tako programske kot materialne.

Nadrobno načrtovanje sistema je razbito na dve stopnji:

- dekompozicija sistema na module, in
- načrtovanje posameznih modulov.

Na prvi stopnji faze načrtovanja materialne opreme (MO) se zastavlja vprašanje strukture sistema. Med takimi vprašanji je n.pr. vprašanje vtičnih enot (boards) sistema. Odločati se moramo: ali naj bodo posamezne vtične enote namenjene samo analognim ali samo digitalnim vhodom, ali pa naj bodo vtične enote take, da bodo vsi vhodi (analogni in digitalni) koncentrirani na eni enoti? Drugo takšno vprašanje je n.pr., kakšen tip vodila naj bo uporabljen? Na drugi stopnji že načrtujemo posamezne vtične enote.

Prva stopnja programerskega inženirstva vključuje razpoznavanje aktivnosti, ki so med seboj na nek način povezane. Obstajajo heuristična pravila, ki načrtovalcu pomagajo pri odločitvah, kako razdeliti sistem na module /Pres82/. Izdelane heuristike dajejo različne poudarke posameznim metodologijam načrtovanja. V nadaljevanju bomo podali kratek opis nekaterih splošnih metodologij načrtovanja RT sistema.

2.2. Metodologije načrtovanja

Na kratko si bomo ogledali najbolj uporabljene metodologije, med katere sodijo: funkcionalni razcep, prekrivanje informacij, sklapljanje in povezovanje, ter razbitje.

Funkcionalni razcep (functional decomposition) je pristop, imenovan tudi "od vrha navzdol". Zagovarjajo ga Wirth in še nekateri. Predstavlja delitev programske opreme na module po funkcijah. To pomeni, da vsak modul izvaja le svojo, zanj specifično funkcijo.

Prekrivanje informacij (information hiding). Parnas /1972/ je bil velik nasprotnik funkcionalne dekompozicije. Zavzemal se je za metodo, ki je osnovana na takšnem prekrivanju informacij med moduli, da so v posameznem modulu prisotne vse možne informacije, ki jih modul uporablja /Par72/.

Razlike med obema pristopoma si oglejmo na naslednjem primeru. Vzemimo program, ki mora iz naprave prebrati blok besed nekega teksta. Besede v bloku morajo biti urejene po abecednem redu in se ne smejo podvajati. Tako urejen seznam besed moramo izpisati.

V primeru, da pristopimo k preprosti "funkcionalni dekompoziciji", razdelimo sistem na tri module: vhodni, urejevalni in izpisni. Vsi moduli imajo dostop do dodeljene podatkovne strukture, v kateri se nahaja tekst, in vsak od njih ve, iz katere seznama (linked list), zapisa (record) ali datoteke (file).

Parnas zagovarja delitev, pri kateri obstaja modul, imenovan upravljalnik pomnilnika (Store-Manager). Ta se ukvarja s shranjevanjem podatkov. Drugi moduli imajo možen dostop do podatkov le preko funkcij tega upravljalnika, ki lahko opravlja obe funkciji: vpisuje in izpisuje (besede). Prva funkcija omogoča, da se informacija shrani v pomnilnik in druga, da se pokliče iz pomnilnika (je dosegljiva), pri tem pa ostalim modulom ni potrebno vedeti, kako je pomnilnik organiziran. Prednost takega pristopa je, da med postopkom načrtovanja sprememba na enem modulu ne zahteva spremembe na drugem modulu.

Sklapljanje in povezovanje (coupling and cohesion) sestavljata dve metodi. Pri prvi metodi je prisotna težnja po minimalni sklopljenosti, pri drugi metodi pa težnja po maksimalni povezanosti med moduli, obe pa uporabljata heuristične pristope načrtovanja podatkovnega vodenja (data flow). Heuristiko in metodologije zanje so razvili Constantine, Steven /StMC74/ in Myers /Mye78/, /You79/.

Razbitje (partition) za minimizacijo vmesnikov je zasnovano na heuristiki, ki jo je predlagal DeMarco (1973) /BenL84/, in jo je mogoče kombinirati z metodami pretoka podatkov. Pravilno je menil, da je mogoče s transformacijo, ki jo je pokazal na diagramu pretoka podatkov, diagram razbiti v poddiagrame (module) tako, da se med njimi minimizira število povezav.

Pri RT sistemih so potrebne še dodatne heuristike, ki so značilne za posamezne module naslednjih kategorij:

- RT-HC: realni čas, trda omejitev (real-time, hard constraint)
- RT-SC: realni čas, mehka omejitev (real-time, soft constraint)
- IACT: pogovoren (interactive)

Uveljavljeni tipi programa (sekvenčni, večopravilni, program realnega časa) kažejo na težnje, da se zmanjša obseg programske opreme modula, ki pripada RT-HC kategoriji, saj je le-tega najtežje načrtati in testirati.

V pogledu načrtovanja programske opreme obstajajo bistvene razlike med RT in standardnimi sistemi na stopnji uvodnega načrtovanja in v postopkih načrtovanja na stopnji dekompozicije na module. Zato se bomo v nadaljevanju osredotočili prav na tovrstno problematiko, saj je načrtovanje, kodiranje in testiranje posameznega modula več ali manj podobno gradnji programske opreme standardnega tipa (non-real-time software).

2.3. Dokumentiranje specifikacij

Splošen primer dokumentiranja za fazo planiranja obsega dokumentacijo o:

- vmesniški (interface) opremi (povezava z okoljem)
- nadzorni napravi
- komunikaciji z operaterjem, ki vsebuje dokumentacijo o
  - prikazovalniku
  - nastavitvi krmilne naprave
  - posegih operaterja (tudi v smislu nastavljanja novih parametrov)
  - upravljavski informaciji (management information)
  - splošni informaciji

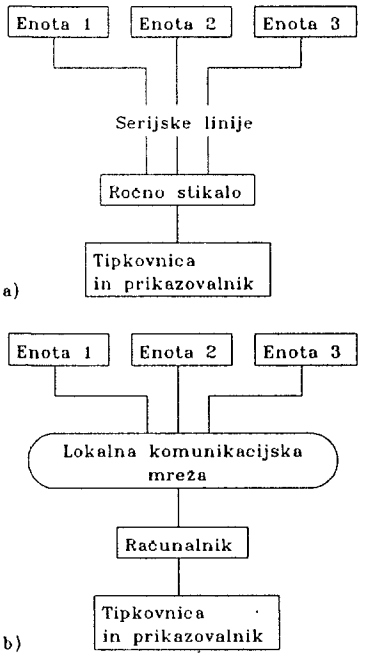
3. UVODNO NAČRTOVANJE

3.1. Načrtovanje materialne opreme

Ekonomski učinek materialne opreme računalnika v RT sistemu je odvisen od več činiteljev. Eden takšnih je gotovo tisti, ki bo odločal, ali naj bo računalnik na eni sami vtični enoti ali naj bo modularno grajen in naj uporablja standardno vodilo. Zahteve vplivajo tudi na izbiro 8-bitnega ali večbitnega mikroprocesorja, števila analognih in digitalnih signalov (kanalov), zahtevane hitrosti prenosov, konverzij ipd. Obstaja tudi ekonomski rizik pri izbiri števila potrebnih procesorjev: eno- ali več- procesorski sistem. Za primer si oglejmo dve možni konfiguraciji in ju poskusimo ovrednotiti.

Konfiguracija 1 (na sliki 3a) predstavlja preprosto rešitev materialne opreme z minmalnim številom procesorjev in preprostim mehaničnim stikalom, ki lahko poveže tastaturo z enim od 12-tih procesorjev. Vsak procesor bo moral vsebovati prikazovalnik ter vhodne programe in programe za procesiranje informacij.

Konfiguracija 2 (na sliki 3b) upošteva ločitev pravih kontrolnih funkcij od upravljalskih in operatorskih vhodno/izhodnih funkcij. V primerjavi s konfiguracijo 1 zahteva konfiguracija 2 dodatno materialno opremo, ki jo sestavljata poseben procesor in komunikacijska mreža. Pri tem je potrebno za vsakega krmilnega procesorja zagotoviti ustrezne pomnilniške kapacitete.



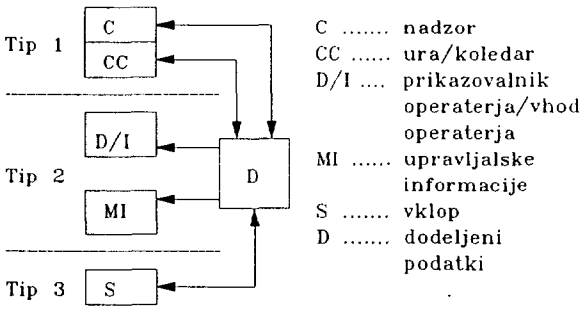
Slika 3. Mo'ne konfiguracije materialne opreme: a) Konfiguracija 1, b) Konfiguracija 2.

Del postopka načrtovanja MO je, da se oba možna pristopa k izbiri obeh konfiguracij na sliki 3 in drugi možni pristopi primerno ovrednotijo.

S podatkovnimi in časovnimi zahtevami (kot je n.pr. minimalni interval vzorčenjaja) je določena sistemska ura realnega časa. Potrebna je tudi odločitev, kakšni terminali naj bodo uporabljeni: specifični ali standardni terminali, koliko enih in koliko drugih ? To je odločitev, ki mora biti rezultat skrbne analize cen in prednosti, ki jih nudi posamezna rešitev.

3.2. Načrtovanje programske opreme

Programsko opremo RT-sistema iz prejšnega poglavja lahko razdelimo v segmente, ki so razvidni iz slike 4.



Slika 4. Konfiguracija programske opreme.

Krmilni modul (C modul) ima n.pr. **strogo omejitev** (hard constraint), saj mora vzorčevati najmanj vsakih 40 ms. V praksi je lahko ta omejitev nekoliko bolj ohlapna, recimo 40 ms ±1 ms s povprečno vrednostjo okrog 1 minute, kar zapišemo 40 ms ±0.5 ms. V splošnem čas vzorčevanja določimo z izrazom $T_s \pm e_a$. Zahteve lahko nekoliko omilimo, če n.pr. dopuščamo, da med 100 vzorci izpustimo en vzorec. Takšne omejitve so del testne specifikacije in jih moramo upoštevati tudi med testiranjem sistema.

Od CC modula (Clock/Calendar modul) se zahteva, da se ne izgubi noben urin impulz. Zato se mora aktivirati vsakih 20 ms (pri osnovni frekvenci 50 Hz). Takšno omejitev lahko spremenimo v **ohlapno** (soft constraint) z dodatno materialno opremo v obliki števnika, ki ga CC modul lahko čita in resetira. Omejitev je v tem primeru 1 sekundni odzivni čas s 5 sekundnim intervalom med dvema sosednjima odčitkoma števnika. Kolikšna je velikost števnika pri takšni omejitvi, si lahko izračunamo sami!.

D modul (operator display) ima strogo omejitev, tj. časovni interval odčitkov 5 sekund. Po občutku bi rekli, da takšen interval ni stroga omejitev, saj bi mora ustrezati povprečnemu času 5 sekund. Vendar je omejitev stroga, ker mora biti specificiran hkrati s povprečnim časom tudi maksimalni časovni interval, ki je n.pr. 10 sekund.

Mehke omejitve lahko pripišemo I modulu (operator input) in MI modulu (management information). O njih odločamo v soglasju s kupcem in oblikujejo del specifikacij v dokumentu zahtev.

Za S modul (start up) ni nujno, da dela v realnem času. Zato ga lahko obravnavamo kot standardni interaktivni modul.

Obstajajo različne aktivnosti, ki jih lahko delimo v več manjših. Kašna bo ta delitev in kako bodo potekale naslednje stopnje, je odvisno od splošnega pristopa k implementaciji. Obstajajo trije možni pristopi:

- enojen program
- sistem priviligiranih programov (foreground/background system - FG/BG system)
- večopravilni sistem

V nadaljevanju bomo obravnavali vsak pristop posebej.

3.3. Enojen program

Standardni pristop programiranja smatra posamezne module na sliki 4 kot procedure ali subrutine enojnega, glavnega programa:

```
begin Vklop;
  while not prekinitev do
    odčitavanje ure na vhodni liniji;
    if odčitek ure then begin
      ažuriranje ure in koledarja (CC);
      case true do
        čas za krmiljenje: krmiljenje (CO);
        čas za prikazovanje: ažuriranje
                    prikazovalnika (DU);.
        vhod operaterja: vnos podatkov (OI);
        zahteva za upravljanje: upravljanje
                              izhoda (MO);
      enddo
    end
  enddo
end.
```

Pri programiranju takšne strukture ni nikakršnih težav. Vendar se pri tem pojavlja več resnih časovnih omejitev, kot je n.pr. zahteva, da se mora CC modul aktivirati vsakih 20 ms, neglede na ostale module. Če so t_1,t_2,t_3,t_4 in t_5 maksimalni časi računanja, ki ustrezno pripadajo modulom CC, C, DU, OI in MO (management output), potem je za delujoč sistem postavljena zahteva, ki jo lahko izrazimo z izrazom

$t_1 + \max(t_1,t_2,t_3,t_4,t_5) \leq 20$ ms(Pomni: Vrednosti t_1,t_2,t_3,t_4 in t_5 morajo vključevati čas, ki je potreben za testiranje, in v t_1 mora biti vključen tudi čas za odčitavanje urinih impulzov).

Pristop k načrtovanju, ki je zasnovano na enojnem programu, lahko uporabljamo za preproste, majhne sisteme. Takšen pristop daje pregledno rešitev ob

minimalni materialni in programski opremi sistema, ki ga je lahko testirati. Z večanjem opreme pa narašča problem na stopnji nadrobnega načrtovanja. Pojavlja se težnja, da se program razbije na module. Problem ne narašča zaradi funkcionalnih razlik med moduli, ampak preprosto zato, da se lahko znotraj posameznih modulov programska oprema sčasoma tudi dopolnjuje. Glede na zahteve MO modula je pristop "enojnega programa" neprimeren. Uporabljiv je le, če zahteve spremenimo. Lahko pa tudi zahtevamo, da DU modul (display update module) razbijemo na tri module: prikazovanje datuma in časa, prikazovanje procesnih vrednosti in prikazovanje krmilnih parametrov.

3.4. Sistem priviligiranih programov

Vsekakor obstajajo prednosti, če so moduli s strogimi časovnimi omejitvami ločeni in neodvisno vodeni od modulov z ohlapnimi ali brez omejitev - manj je interakcij med moduli in manj tesnih časovnih situacij. Moduli s strogimi časovnimi omejitvami, ki so najbolj tesno povezani z zunanjimi procesi, tečejo v t.im. ospredju (foreground) in jih bomo imenovali **priviligirani moduli.** Moduli z ohlapnimi omejitvami ali brez njih tečejo v t.im. ozadju (background) in jih bomo imenovali **nepriviligirani moduli.** Priviligirani moduli, ali opravila, kot jih navadno imenujemo, imajo visoko prioriteto glede na opravila, ki tečejo v ozadju. Zato se uporabljajo posebni mehanizmi, ki omogočajo, da priviligirano opravilo prekine nepriviligirano opravilo.

Razdelitev opravil v FG in BG opravila navadno zahteva podporo RT operacijskega sistema, kakršno nudi že sistem RT/11. Tej zahtevi je mogoče prilagoditi tudi nekatere standardne operacijske sisteme. Tako CP/M omogoča preproste FG/BG operacije, če materialna oprema podpira prekinitve. FG opravilo je prekinitvena rutina in BG opravilo standardni program.

V primeru pristopa, ki je zasnovan na sistemu priviligiranih programov, lahko strukturo zgoraj opisanega enojnega programa modificiramo v strukturo, ki je deljena na dva sistemska dela: v ospredju, v ozadju.

v ospredju

```
if prekinitev then begin
  ažuriranje ure in koledarja (CC);
  if čas za krmiljenje (C) then krmiljenje (CO);
  RTI
```

Zahteva za delo v ospredju (FG področje) je strnjena v izrazu:

$$t_1, t_2 \leq 20 \text{ ms},$$

kjer je
t_1 = maksimalen čas za CC modul, in
t_2 = maksimalen čas za C modul.

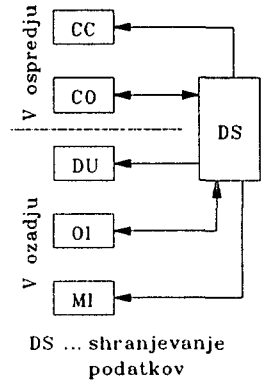
v ozadju

```
begin Vklop;
  while not prekinitev do
    case true do
      čas za prikazovanje: ažuriranje
                    prikazovalnika (DU);
      vhod operaterja: vnos podatkov (OI);
      zahteva za upravljanje: upravljanje
                              informacij (MI);
    enddo
  enddo
end.
```

Zahteve za delo v ozadju (BG področje) pa so:

1., max(t_3,t_4,t_5) 10 sek;
2., prikazovalni modul skanira povprečno vsakih 5 sek;
3., vhodni operator se odziva v času 10 sek.

Vidimo, čeprav so časovne omejitve bolj ohlapne, je ovrednotenje sistemskih zmogljivosti težje. Izvajati moramo meritve, ki so bolj zapletene, kot v primeru enojnega programa.



Slika 5. Povezanost programskih modulov FG/BG sistema.

Čeprav FG/BG pristop razbije **krmilno** strukturo v dva dela - FG in BG module, so posamezni moduli med seboj še povezani preko podatkovne strukture (slika 5).

Povezave med moduli skrbijo za dodeljevanje podatkovnih spremenljivk (krmilnih parametrov). Pri enojnem programu (imenovan tudi enoopravilni program) ni nobenih težav pri nadzorovanju dostopa do dodeljenih spremenljivk, ker je vedno aktivno samo eno opravilo (modul). V primeru FG/BG sistemskih opravil, pa so možne paralelne aktivnosti. Istočasno je lahko aktivno eno opravilo

iz FG in eno opravilo iz BG področja (oz. se opravili izvajata kvazi paralelno, če imamo enoprcesorski sistem).

V našem posebnem primeru lahko spremenljivke dodelimo krmilnim, prikazovalnim in vhodnim modulom brez posebnih težav, saj neko spremenljivko priredimo samo enemu modulu. Modulu operatorja vhodov priredimo krmilne parametre in množico kazalčnih spremenljivk, CC modulu podatkovne in časovne spremenljivke, ter krmilnemu modulu podatkovne spremenljivke procesa, ki ga sistem nadzira. Zaradi varnosti moramo vhodne podatke iz operatorja predhodno shraniti in jih prenesti dodeljenemu pomnilniku šele potem, ko so verificirani. V ta namen se uporablja posebna metoda shranjevanja vhodnih podatkov (parametrov). Če za take varnostne ukrepe ne poskrbimo, lahko npr. v telefoniji pri vzpostavljanju zveze pride do nezaželjenih prekinitev po drugi poti in s tem do nepravilnega ali nestabilnega nadzora zveze.

Prenos podatkov med FG in BG opravili je znan kot **kritična sekcija** programa, ki mora biti nedeljiva. To preprosto dosežemo s prepovedjo vseh prekinitev med prenosom podatkov.

Težava pri takem - sicer preprostem pristopu - je, da je pri večih ločenih modulih nezaželjeno, da ima vsak modul dostop do osnovne materialne opreme stroja in možnost spremiti status prekinitev. Praksa je pokazala, da je materialno opremo takšnih modulov teško načrtovati, kodirati in testirati, in da obstaja v povprečju večja možnost napak. Zato je priporočljivo, da je število takšnih modulov čim manjše.

Idealni prenosi morajo biti časovno prilagojeni krmilnemu modulu, kar pomeni, da morata biti operatorski in krmilni modul sinhronizirana ali se morata med seboj dogovarjati (rendezvous).

3.5. Večopravilen sistem

Oblikovanje programske opreme velikih RT sistemov in sistemov s pogostimi interakcijami je lahko bistveno lažje, če je mogoče delitev FG/BG še dalje deliti na več delov, kar omogoča rabo koncepta večih aktivnih opravil. Vsako aktivnost, ki se mora izvesti, smatramo že v začetni stopnji oblikovanja programske opreme za ločeno opravilo. Takšen (večopravilni) pristop se implicira v možnosti paralelnega izvajanja opravil, pri čemer število uporabljenih procesorjev v sistemu sploh ni bistveno.

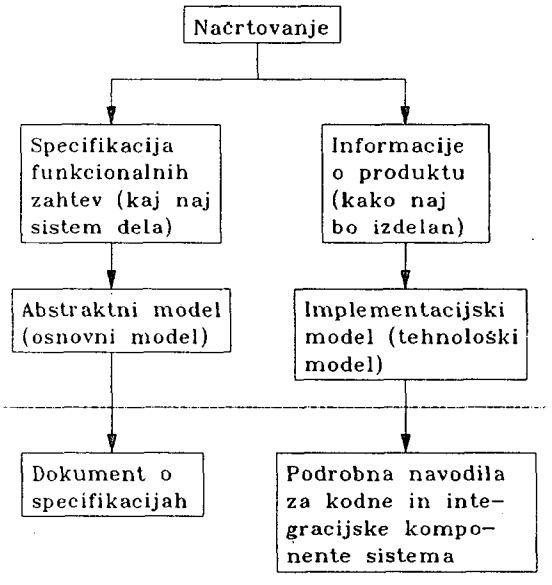
Za implementacijo večopravilnega sistema morajo biti dane naslednje možnosti:

- kreiranje ločenih opravil;
- razporejanje opravil med izvajanjem, navadno na prioritetni osnovi;
- dodeljevanje podatkov med opravili;
- sinhronizacija opravil med seboj in z zunanjimi dogodki;
- preprečevanje kvarnih vplivov enega opravila na drugega; in
- nadzor nad izvajanjem opravil (start/stop).

Naštete sposobnosti zagotavlja večina RT operacijskih sistemov. Predvsem jih zagotavljajo operacijski sistemi, ki so namenjeni večjim RT sistemom.

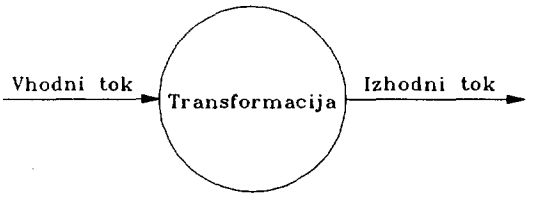
4. SPLOŠEN PRISTOP

Obravnava v zgornjih razdelkih je posvečena predvsem tistim tehničnim podrobnostim načrtovanja, ki imajo vpliv na oblikovanje arhitekture programske opreme. Učinkovitost načrtovanja pa postane zadovoljiva šele, če je možna abstrakcija sistema.



Slika 6. Abstraktni in izvedbeni model.

Oblikovanje velikih kompleksnih sistemov (za katere predpostavljamo, da so večopravilni) zahteva dva modela: abstraktnega in izvedbenega. Abstraktni model - ki se včasih nanaša na logični ali osnovni model - ne vsebuje nobene informacije o



Slika 7. Podatkovni pretok - osnovna notacija.

implementaciji; produkcijski ali izvedbeni model pa vsebuje tehnične detajle o delovanju sistema (slika 6).

Modela lahko razvijamo drug za drugim; tj. najprej razvijemo abstraktni in nato izvedbeni model. Bolj običajno pa je, da ju razvijamo sočasno, saj lahko odločitve o implementaciji na visokem nivoju vplivajo na odločive abstraktnega modela na nižjem nivoju.

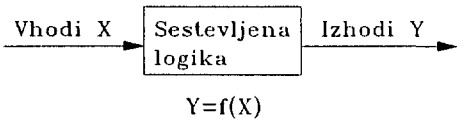
Razvoj abstraktnega modela, zasnovanega na specifikacijah, je odvisen od opisa sistema. Ta mora biti dovolj bogat idej, s katerimi je mogoče izraziti vse zahteve za načrtovanje sistema. Pri razvoju mora biti upoštevana metodologija, ki omogoča, da se abstraktni model prevede v izvedbeni model. Smernice razvoja se lahko opirajo na izsledke, dobljene s pomočjo heurističnih postopkov, ki jih bomo obravnavali kasneje.

Večino uporabnih zapisov in metodologij za načrtovanje RT sistemov je razvil Myers /1974/ na osnovi podatkovnega vodenja /May78/. Prvobitna temeljna predpostavka tehnike podatkovnega vodenja je, da lahko sistem ali podsistem predstavimo s podatkovno transformacijo (pretvorbo) kot osnovno notacijo (slika 7).

Podatkovni pretok je označen s puščico. Veljajo naslednje predpostavke:

1. Vhodni podatki za transformacijo in izhodni podatki so diskretne enote; vsaka enota predstavlja prevod (transakcijo);
2. Transformacija se proži na dospeli prevod;
3. Trajanje transformacije ne vpliva na natančnost operacije;
4. Izhod transformacije je odvisen le od tekoče transakcije. Na transformacijo ne vplivajo prejšni vhodni podatki. To pomeni, da transformacija nima nobenega notranjega zapisa o prejšni transakciji.

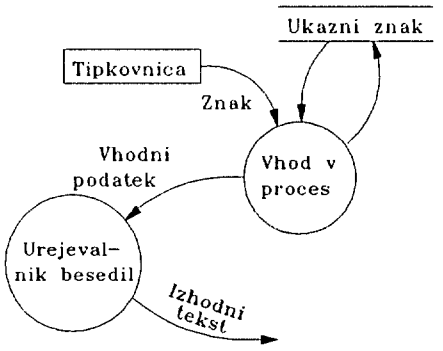
Model sistema je ekvivalenten sestavljenemu logičnemu vezju na sliki 8.



Slika 8. Sestavljen logični ekvivalent preprostega podatkovnega pretoka.

Preprost zapis, ki je zasnovan na podatkovnem vodenju, povzroča probleme pri predstavitvah številnih aplikacij, predvsem takšnih sistemov, ki so interaktivni. V veliki večini interaktivnih sistemov zahteva podatkovni vhod obravnavo različnih poti, ki so odvisne od prejšnih vhodnih podatkov. Pri modelu takšnih sistemov lahko uporabimo standardni zapis podatkovnega vodenja, če predhodne vhodne podatke, ki smo jih shranili, pred glavno sistemsko transformacijo dodamo novim vhodnim podatkom.

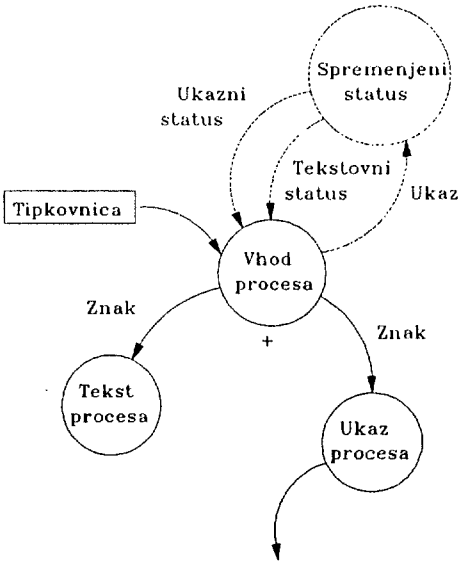
Primer takšnega pristopa je npr. sistem besednega urejevalnika. Iz tastature prihaja ali besedilo ali del ukaznega niza. Start ukaznega niza določa znak "ESC" in konec niza znak "CR". Rešitev kaže slika 9.



Slika 9. Ukazno procesiranje.

Vhodna transformacija prebere znak iz prikazovalnika in ga postavi pred znak, ki ga vzame iz pomnilnika ukaznih znakov. Ukazni startni znaka (ESC) in zadnji znak (CR) se z vhodno transformacijo shranita v pomnilnik. Podatek, ki ga sprejme glavna transformacija je tako ali ESC + znak, ali CR + znak, in transformacija procesiranja besed podatke ustrezno interpretira.

Alternativni pristop je takšen model sistema, da se vhodna transformacija zapomni, če je bil startni



Slika 10. Uporaba krmilne notacije

ukazni znak sprejet pri spremembi njenega notranjega stanja tako, da so kasneje sprejeti znaki obravnavani kot ukazi in prenešeni na ukazni procesor pred tekst procesorjem. Takšen model vidimo na sliki 10.

Poleg podatkovne transformacije je v modelu vpeljana dodatna notacija, ki označuje "ukazno" transformacijo. Ukazne akcije so označene s prekinjenimi črtami. Krmilni tokovi so dogodki, ki niso vezani na podatke. Krmilna transformacija je določena s tabelo prehajanja stanj.

V drugem pristopu je model sistema ekvivalenten sekvenčnemu logičnemu sistemu. Izhod je odvisen ne le od trenutnih vhodov, ampak tudi od notranjega stanja sistema, ki je funkcija prejšnih vhodov.

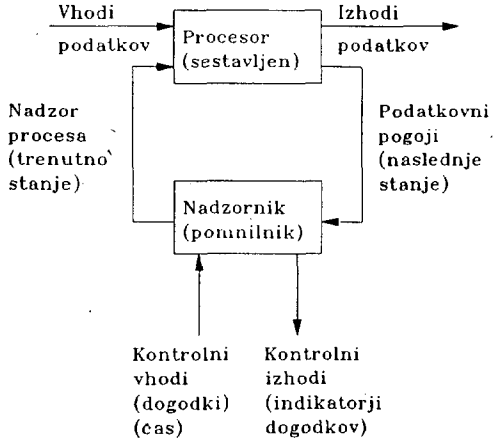
Študija RT sistemov kaže, da imajo le-ti dodatne značilnosti. Tipično ima RT sistem različne tipe vhodov in izhodov, katerih notacije predstavljajo:

1. diskretne dogodke, ki se lahko pojavljajo naključno;
2. zvezno spreminjajoče vrednosti podatkov; in
3. diskretne podatkovne vrednosti (ekvivalent transakcij)

Transformacije se lahko aktivirajo:

1., ob prihodu diskretne podatkovne vrednosti;
2., pri vnaprej določenih časovnih intervalih;
3., ko se pojavijo specifični notranji ali zunanji dogodki; ali
4., kontinuirano.

Spremembe notranjega stanja sistema so možne ne le kot posledica podatkovnih pogojev ampak tudi kot posledica zunanjih dogodkov. Hatley /Hat84/ je predlagal model, ki ga prikazuje slika 11. Krmilni vhodi in izhodi so tokovi dogodkov.



Slika 11. Splošni model RT sistema (po Hatley-u, 1984)

Upoštevajoč gornje značilnosti RT sistemov, so bile razvite številne metodologije načrtovanja RT sistemov. Glavne od njih so:

1. MASCOT (modularni pristop v konstrukciji programske opreme delovanja in testiranja); vpeljala sta ga Jackson in Simpson /JaS75/.
2. DARTS (pristop k načrtovanju RT sistemov); Gomaa /Gom84/.
3. PAISLey (procesno orientiran, aplikativen, interpretativen, specifikacijski jezik) /Zav84b/.
4. Metoda za struktuirano analizo velikih RT sistemov; Hatley /Hat84/.
5. Struktuiran razvoj za RT sisteme; Ward and Mellor /WaM86/.

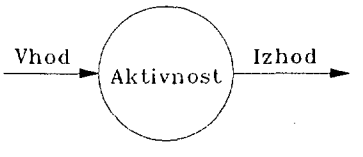
V nadaljevanju bomo obravnavali dve metodi: MASCOT in Ward/Mellor-jev strukturni razvoj.

5. MASCOT

Natančna (uradna) definicija metodologije MASCOT je podana v "The Official Handbook of MASCOT", MASCOT Suppliers Association. Poglede na različne aspekte metod najdemo v delih /JaS75/, /SiJ79/ in /Sim82/. Pri uporabi sistema MASCOT lahko preliminaro načrtovanje RT sistema interpretiramo kot konstruiranje virtualnega stroja, na katerem rešujemo dani problem. Virtualni stroj je zgrajen iz posebne množice abstraktnih enot /All81/. Konstruiranje takšnega stroja s pomočjo Module 2 je opisal /Bud85/. V naslednjih stopnjah načrtovanja se posvečamo implementaciji virtualnega stroja na realnem, ali na več realnih strojev, računalnikov. Ta metoda načrtovanja ne predpostavlja rabo določenega števila procesorjev.

Nadalje si oglejmo abstraktne entitete, kot so: aktivnost, komunikacija, kanal, nabiralnik in sinhronizacija.

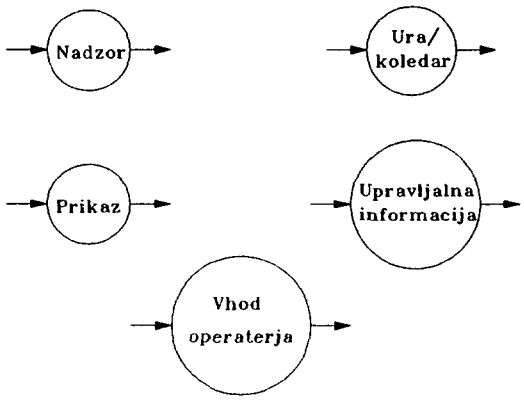
Aktivnost. Vsako aktivnost, ki se izvaja, označimo s krogom. Shematično je aktivnost prikazana na sliki 12, množica aktivnosti iz primera na sliki 4 pa na sliki 13.



Slika 12. Notacija aktivnosti.

Predpostavljamo, da virtualni stroj obravnava vsako aktivnost kot ločeno opravilo. Tako se le-te lahko nanašajo na opravila ali procese. Vendar je bolj zaželjeno, da jih do nadaljnega imenujemo aktivnosti, ker se na tej stopnji še ne odločamo o dejanski

strukturi opravil: več aktivnosti je lahko združenih v eno samo opravilo.



Slika 13. Aktivnosti iz primera na sliki 4.

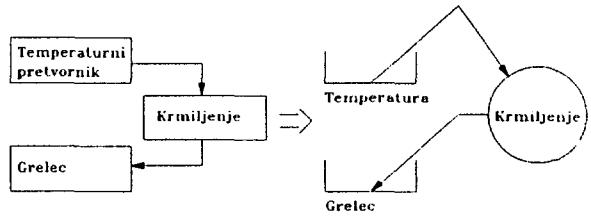
Vsekakor sistem na sliki 13 ni delujoč, saj manjkajo še posebne aktivnosti, ki urejajo odnose med posameznimi aktivnostmi (tj. komunikacije med aktivnostmi).

Komunikacije med aktivnostmi delimo na tri tipe:

- neposredni prenos med dvema akcijama;
- porazdeljevanje informacije večim akcijam; in
- sinhronizacijski signali.

Kanali. Učinkovit koncept za opisovanje direktnih komunikacij med aktivnostmi je kanal (podoben vodu - pipe). Poleg tega, da kanal povezuje dve aktivnosti, lahko združuje tudi neke pomnilniške lastnosti. Tako se lahko skozi kanal pretaka "hkrati" več posameznih (delov) informacij iz množice informacij. Posamezni deli informacije so navadno urejeni tako, da je prvi del informacije, ki vstopi v kanal, na drugem koncu kanala tudi prevzet. Na tej stopnji načrtovanja je skrb za implementacijo kanala nepotrebna. Obstaja pa več metod implementacije, ki zadevajo problematiko konkurenčnega programiranja. Zato jih v tem sestavku ne bomo obravnavali. Kanal označimo s posebnim simbolom: prečno črto med vhodom in izhodom (glej sliko 15!).

Nabiralniki. Fizikalne naprave (pomnilniki datotek, prikazovalniki ipd.) so naprave nadziranega okolja. Zamišljamo si jih kot nabiralnike. Nabiralnik (pool) imenujemo zbirko informacij, ki jo je mogoče čitati in/ali vanjo vpisovati iz večih aktivnosti sistema. Operacija čitanja informacije v nabiralniku ne briše ali spremeni. Tako je lahko vsaka informacija, ki je shranjena v nabiralniku, večkrat prečitana. Nabiralnik dela kot banka podatkov ali shramba informacij. Katerikoli del informacij v nabiralniku je enako lahko dosegljiv vsem aktivnostim, ki nabiralnik uporabljajo. Na stopnji načrtovanja tudi v tem primeru ni pomemben mehanizem ustvarjanja in delovanja nabiralnika - nabiralnik je preprosto abstraktna ideja. Pri nekaterih implementacijah so potrebni mehanizmi, ki ščitijo podatke pred okvarami in nadzorujejo uporabo informacij. Nabiralnik je simboliziran na sliki 14 kot nabiralnik temperaturnih vrednosti in krmilnih vrednosti grelnika.



Slika 14. Primer načrtovalne notacije (sistem nadzora tempera ture).

Preprost krmilni sistem na sliki 14 je sestavljen iz temperaturnega pretvornika, nadzorne naprave (krmilnika) in grelca. Krmilna naprava odčitava temperaturne podatke iz temperaturnega merilnega mostička, jih procesira in vpisuje v krmilnik grelca. Programski model obravnava temperaturni pretvornik in nadzirani grelec kot nabiralnik, krmilno napravo pa kot aktivnost.

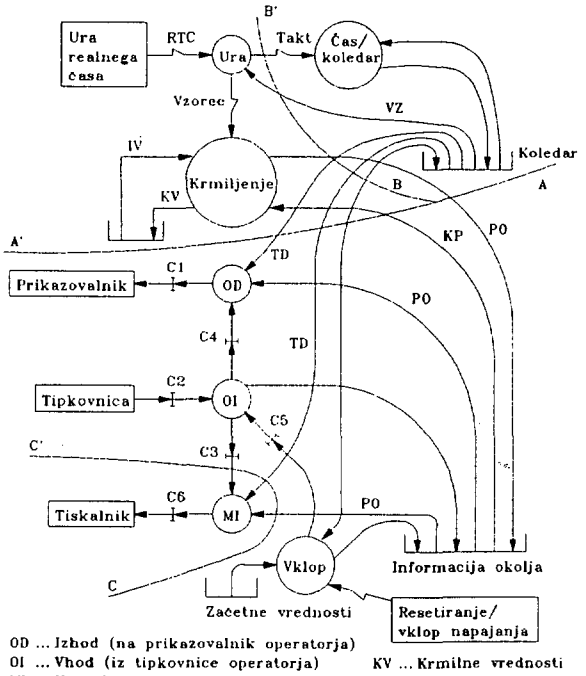
Sinhronizacija. V programih, ki zahtevajo potrditev, je potrebna sinhronizacija procedur. Ta zahteva je še posebej izrazita pri večopravilnem programu, v katerem se podatki razporejajo med več opravili. Poleg drugih aktivnosti so potrebne tudi aktivnosti za stop, start in zakasnitev (delay). Potrebne so tudi takšne aktivnosti, ki zahtevajo npr. vprašanje "Si pripravljen na sprejem podatka ?" ali vpračanje "Imaš kakšen podatek za mene ?". Aktivnostim ni dovoljeno, da se prekinejo, poženejo ali da zahtevajo vprašanja od drugih aktivnosti direktno. Pri razporejanju podakov bomo namreč videli, da je potrebna aktivnost, ki daje ukaz ali zahtevo za razpoznavanje stanja aktivnosti, kateri druga aktivnost nekaj ukazuje ali jo sprašuje. Direktna komunikacija signalov aktivnosti (ukaza ali zahteve) vodi k metodi sinhronizacije, ki ne omogoča verifikacije programa. To še ne pomeni, da direktne metode niso uporabljive: običajno se pogosto uporabljajo zastavice, ki so skupne, dosegljive večim aktivnostim, in označujejo, da se je izvršil nek poseben dogodek. Oglejmo si ta način na našem prejšnem primeru; za sinhronizacijo med CC in D aktivnostjo je potreben interval 5 sek; zastavico postavi CC vsakih 5 sekund, medtem ko D aktivnost v intervalih ugotavlja, če je zastavica že postavljena.

Uporaba te metode pri velikih sistemih prinaša več problemov. Predvsem je neugodno, da mora operator aktivnosti nepretrgoma pregledovati zastavico; in, če so prisotne še kake druge aktivnosti, mora operator aktivnosti poskrbeti, da D aktivnost čaka na ponovno testiranje zastavice. Potreben je nek mehanizem, ki omogoča, da D aktivnost čaka na dogodek (5 sekundni interval) in na CC aktivnost, ki signalizira, da se je dogodek pojavil.

Če zahtevamo, da imata proceduri WAIT(dogodek) in SIGNAL(dogodek) spodaj naštete lastnosti, potem lahko z njima opišemo želeno sinhronizacijo.

- WAIT(dogodek). Opravilo začasno ustavi aktivnost, čim se izvrši WAIT operacija. Neaktivnost traja do signalizacije dogodka, če se le-ta pojavi. Če pa je dogodek signaliziran že pred WAIT operacijo, se aktivnost takoj nadaljuje (tj. kontinuirano, brez čakanja).
- SIGNAL(dogodek). Operacija SIGNAL signalizira, da se je zgodil dogodek in tako omogoči, da se proces čakanja nadaljuje.

WAIT je procedura, ki kontinuirano odčitava dogodkovni status kanala ali nabiralnika, SIGNAL pa procedura, ki vanju vpisuje informacijo. V primeru ene same centralne procesne enote, ne more ena aktivnost stalno odčitavati neko spremenljivko in druga aktivnost spremeniti status



Slika 15. Uvodni (preliminarni) načrt programske opreme grelnega sistema.

spremenljivke za prekinitveno aktivnost. Zato je za izvajanje WAIT procedure, ki teče na realnem stroju, potrebna drugačna metoda; na uvodni stopnji načrtovanja se nam ni potrebno ukvarjati z vprašanjem, kako proceduro WAIT implementiramo.

Poudariti je treba, da informacija o posameznem dogodku ni nujno omejena na dve aktivnosti; to pomeni, da lahko več aktivnosti čaka na isti dogodek. Seveda lahko dogodek signalizira ena sama aktivnost.

Pri nadzoru opreme RT sistema je lahko dogajanje na opremi zelo pomembno. Takšni dogodki morajo biti SIGNALizirani s pomočjo zunanjih prekinitev, ki predstavljajo nekakšne sinhronizacijske signale. Dogodek, ki je SIGNALiziran z zunanjo prekinitvijo, mora aktivirati operacijo "čakanje" (WAIT). Sinhronizacijski signali so predstavljeni s simbolom impulza, vidni na sliki 15. Naprave (v materialni opremi) so označene s pravokotnimi bloki.

PRIMER

Implementacijo obravnavane notacije si lahko ogledamo na sliki 15. Predstavlja sistem ogrevanja s toplim zrakom. Osnovni načrt tega sistema je podan na sliki 4. Mogoče ga je načrtati na več načinov. V našem primeru je načrtan tako, da je vmesnik okolja predstavljen z nabiralnikom in prikazovalnik operaterja kot zunanja enota, ki je preko kanala povezana z aktivnostmi prikazovalnika.

6. NADROBNO OBLIKOVNAJE PO

Naslednja stopnja načrtovanja programske opreme (PO) je delitev sistema na segmente, kot kaže slika 15. Sistem razčlenjujemo postopoma. Delimo ga na vse manjše dele znotraj vsakega novega segmenta. Način, kako izvesti delitev, je odvisno predvsem od vaje in izkušenosti načrtovalca, ki lahko upošteva nekatere splošne smernice:

1. Grupiranje tistih aktivnosti, ki imajo podobne časovne karakteristike.
2. Grupiranje tistih aktivnosti, ki so tesno povezane z zunanjo opremo (okoljem).
3. Grupiranje aktivnosti tako, da je število kanalov, ki povezujejo posamezne grupe med seboj, čim manjše.

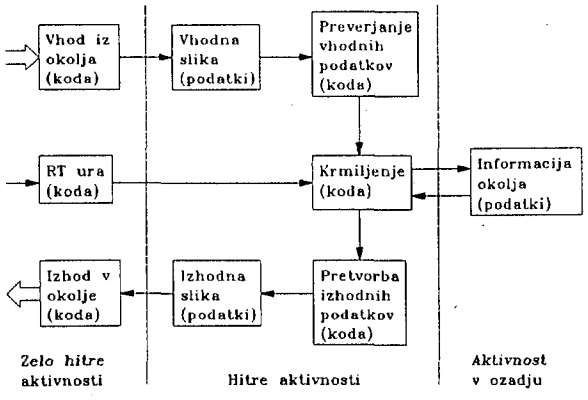
Z uporabo tretjega pravila dobimo razdelitev, ki je na sliki 15 označena s pikčastimi črtami in oznako AA'. Če sedaj uporabimo še pravilo 2, razdelimo gornji del diagrama tako, kot kaže pikčasta črta BB', s pravilom 1 pa še spodnjo polovico diagrama tako, kot kaže pikčasta črta CC'. V naslednjem razdelku bomo bolj na drobno obravnavali povezavo A'BB'

med segmentom, ki vsebuje urine aktivnosti, in segmentom, ki vsebuje aktivnosti krmiljenja.

Najpreje bomo bolj podrobno obravnavali funkcije, ki se morajo izvajati s pomočjo aktivnosti znotraj segmenta. Te aktivnosti so:

1. URA (CLOCK)
   (a) strežba urine prekinitve;
   (b) prekinitve števnika urinih taktov za generiranje verige urnega takta (TICK);
   (c) štetje verig za generiranje vzorcev (SAMPLEs); in
   (d) preverjanje, če se spremeni VZOREC_INTERVAL
2. KRMILJENJE (CONTROL)
   (a) čitanja in krmiljenje podatkov procesa v okolju;
   (b) izračuna krmilnih vrednosti;
   (c) izhodnih krmilnih vrednosti za okolje;
   (d) vnašanja podatkov OKOLJE_PODATKI v nabiralnik; in
   (e) preverjanje, če so NASTAVITVENE_TOČKE ali KRMILNI_PARAMETRI spremenjeni;

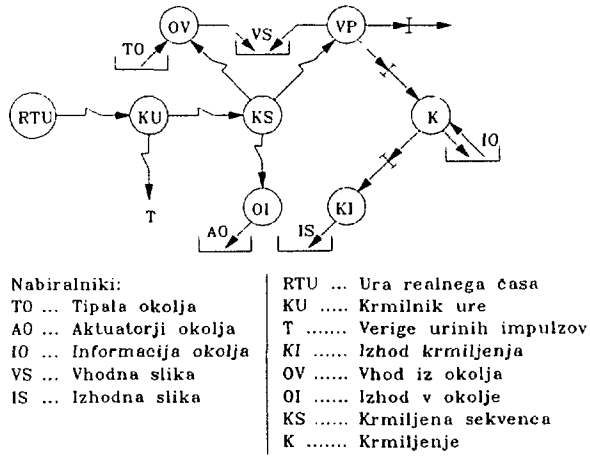
Nadaljne razbijanje programske opreme ure je nesmiselno. Iz funkcij ostalih aktivnosti pa je razvidno, da je le-te mogoče grupirati v tri grupe. Z okoljem sta tesno povezani le dve aktivnosti: zajemanje (t.j. sprejemanje) podatkov in oddajanje krmilnih vrednosti. Obe aktivnosti sta skupaj s prekinitveno rutino časovno najbolj kritični. Zato pripadata grupi z najvišjo prioriteto (t.j. grupi tistih aktivnosti, ki se morajo aktivirati zelo hitro). Glede na notacijo sistema (slika 17) je mogoče preostale aktivnosti krmiljenja nadalje grupirati tako, kot kaže slika 16.



Slika 16. Shema delitve krmilne programske opreme.

V vsakem intervalu vzorčevanja se ob taktu RT ure zajamejo podatki na vhodu iz okolja, t.im. "vhodna podatkovna slika". Nato se istočasno s krmilno informacijo shrani v izhodno podatkovno polje, t.im. "izhodna podatkovna slika", ki se pošlje nazaj v nadzorovano okolje. Informacija, shranjena v izhodni podatkovni sliki, predstavlja krmilne vrednosti, izračunane na osnovi predhodne vhodne podatkovne slike. To pomeni, da vsakemu intervalu vzorčenja sledijo nove izhodne podatkovne vrednosti. Prednost takšnega pristopa je, da sistem zagotavlja krmilne izhode v natančno določenih intervalih vzorčenja. Čas t_c, ki je določen s časom izračuna krmilnih vrednosti, mora zadoščati kriteriju t_c t_s in nič ne de, če t_c variira, le da je manjši od t_s. Krmilnik mora biti načrtovan tako, da kompenzira mrtvi čas, ki se pojavlja med vzorčenjem. Zato mora biti ta čas upoštevan že v modelu nadzorovanega okolja. Iz slike 16 je jasno, kako se lahko pri RT krmiljenju uporabi več procesorjev. Možne so paralelne operacije OKOLJE_VHOD (OV), OKOLJE_IZHOD (OI) in KRMILJENJE (K), ki lahko tečejo na ločenih procesorjih.

Nadaljno delitev aktivnosti krmiljenja predstavlja grupa, sestavljena iz operacij VHOD_PREVERJANJE (VP), KRMILJENJE_IZRAČUN (KI) in IZHOD_PRETVORBA (IP). V preprostih sistemih ta delitev ni nujna, v splošnem pa se mora vhod iz nadzorovanega okolja testirati na pogoje alarmiranja in na karakteristike tipal okolja (smer, napake, ipd. /Ise84/). Izhod na nadzorovano okolje mora biti prilagojen posebnim karakteristikam aktuatorjev. Načrtovalec se mora sam odločiti, ali smatra te operacije kot del aktivnosti OKOLJE_VHOD oz. OKOLJE_IZHOD ali kot ločene aktivnosti VHOD_PREVERJANJE in IZHOD_PRETVORBA, kot je prikazano na sliki 16. V primerih, da imamo samo eno krmilno zanko, je slika razčlenitve modula krmiljenja preprostejša (slika 17).



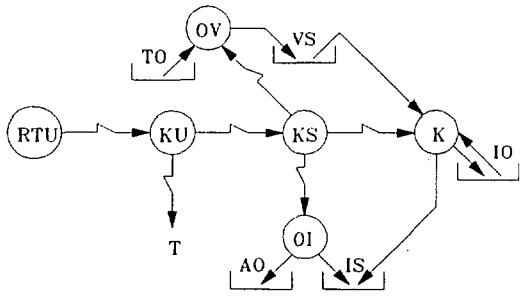
Slika 17. Razdelitev modula krmilne sekcije.

Vsako delitev sistema na sliki 15 moramo še naprej deliti v manjše module. Z vsakim novim diagramom

aktivnosti, ki ga na ta način dobimo, pa bo slovar podatkov bogatejši.

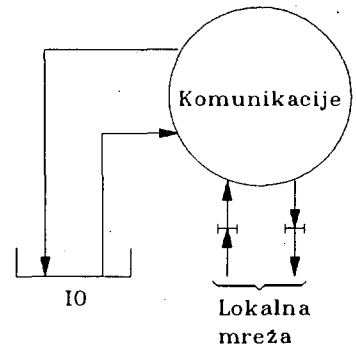
7. PREGLED NAD NAČRTOVANJEM

Nad načrtovanjem, ki vsebuje pogoste intervale v procesu načrtovanja, moramo imeti zelo dober pregled. Ta je sestavljen iz pregleda nad materialno in programsko opremo. Načrtovanje programske opreme ne more prehitevati odločitev v pogledu materialne opreme (npr. števila uporabljenih procesorjev). Programske strukture omogočajo že na uvodni stopnji načrtovanja identificirati aktivnosti, ki se lahko izvajajo paralelno (čeprav bo verjetno neka sinhronizacija med njimi potrebna). Cena računalniške materialne opreme sistema bo lahko v primeru na sliki 3 zaradi izbire procesorja za vsako aktivnost posebej prevelika. Po specifikacijah primera mora vsaki od 12-ih enot pripadati vsaj en procesor in enote morajo biti identične. Možni sta dve konfiguraciji materialne opreme, ki sta pokazani na sliki 4. V primeru konfiguracije 1 se morajo aktivnosti, podane na sliki 17, in aktivnosti, ki pokrivajo potrebe operaterja in upravljanja, izvajati na enem procesorju. V primeru konfiguracije 2, so potrebna sredstva komuniciranja z lokalno mrežo. Modul na sliki 17 se tedaj modificira tako, kot kaže slika 19. Odločitev bo vplivala na posebne komunikacijske mehanizme, uporabljene med aktivnostmi, in na dodeljevanje le-teh. Na primer, pri odločitvi, da uporabimo en procesor za aktivnosti na sliki 18 pomeni, da aktivnosti KRMILJENJE_SEKVENCA, OKOLJE_VHOD, KRMILJENJE_IZRAČUN in OKOLJE_IZHOD smatramo kot eno opravilo, ki ga lahko programiramo takole:



Slika 18. Poenostavljena delitev modula.

```
PROCEDURE KrmilnoOpravilo;
  REPEAT
    Wait(takt);
    OkoljeVhod;
    Okoljelzhod;
    Krmiljenjelzračun:
  UNTIL stop;
END KrmilnoOpravilo;
```



Slika 19. Modifikacija, ki omogoča komunikacijo z mrežo.

8. MASCOT SISTEM

MASCOT predstavlja prvi poskus formalizacije načrtovanja RT sistemov. Podpira jezik CORAL 66, ki je bil tedaj uporabljen v skladu z zahtevami del na projektih obrambe V.Britanije. Ker CORAL ne podpira večopravilne sisteme, naj bi MASCOT to pomnjklivost odpravil.

MASCOT ima dodatne prednosti, ki so bile že omenjene. Te se nananšajo na grupiranje aktivnosti v opravilih in na prilagajanje operacijskega sistema karakteristikam opravil.

Bistvene pomanjklivosti sistema MASCOT so:

- Slabo podpira velike sisteme ali sisteme, ki tečejo na večih procesorjih.
- MASCOT ni struktuiran ("flat" sistem). Vse informacije morajo biti prisotne na enem samem nivoju. Tako ne podpira t.im. metodologijo načrtovanja od zgoraj navzdol (top-down design methodology).
- Sinhronizacijski mehanizmi so omejeni: predvsem, opravil ni mogoče sinhronizirati tako, da se izvajajo v določenih urinih intervalih.

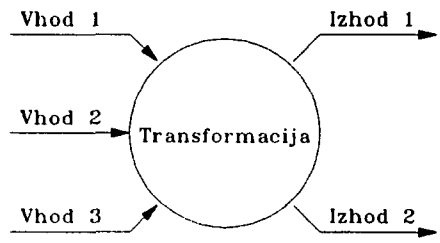
Nekatere od naštetih pomanjklivosti naj bi bile kasneje odpravljene z razvojem sistema MASCOT 3 /Sim84/. Tudi sistem DARTS, ki ga je razvil Gomaa v letih 1984 in 1986 je podoben MASCOTu. Slednji omogoča številne abstrakcije, s katerimi je mogoče zgraditi model načrtovanega RT sistema.

9. STRUKTURNI RAZVOJ

Struktuiran razvoj RT sistemov sta vpeljala Ward in Mellor leta 1986.

Dobro vtečena metoda za načrtovanje sistemov, ki ne delajo v realnem času, je t.im. "metoda pretoka podatkov" (dataflow method). Načrtovalec opazuje

program z vidika prenosa podatkov. Osnovni zapis, ki se uporablja pri tej metodi, je prikazan na sliki 20.



Slika 20. Notacija podatkovnega pretoka.

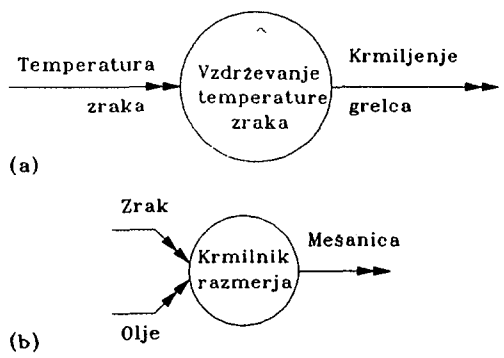
Tu se vhodni podatkovni tokovi transformirajo v določene izhodne podatkovne tokove. Transformacija je ekvivalentna akciji, ki jo poznamo v sistemu MASCOT. Pri tem se predpostavlja, da je podatek, ki se obdeluje, časovno diskreten. To pomeni, da je določen samo v specifičnem trenutku, v ostalem času pa je nedoločen ali ima vrednost 0.

Metoda je široko uporabljena pri načrtovanju transakcij, ki se nanašajo na obdelavo podatkov. Takšen tip sistema je na primer program, ki izvrši preklic bančnega računa. Vsak preklic je ločen diskreten dogodek (transakcija), med katerim podatek ni določen.

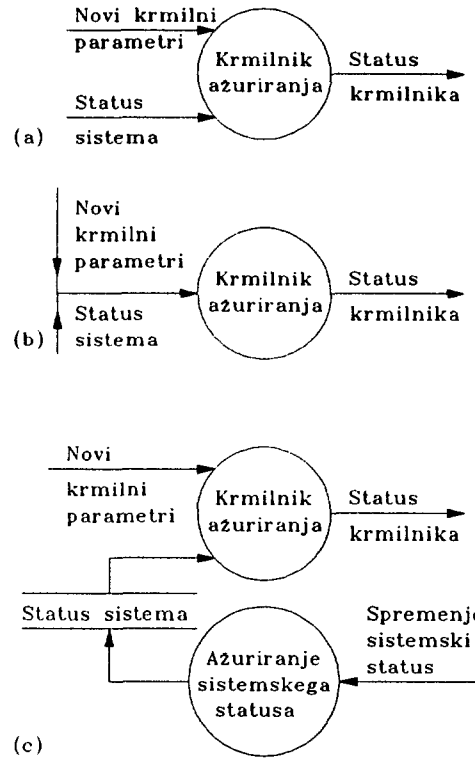
RT sistemi ne vsebujejo samo časovno diskretnih podatkov. Logični in krmilni sistemi imajo časovno zvezne podatke, kot tudi dogodkovne podatke (logične signale). Pri časovno zveznem podatku smatramo, da je podatek določen na neprekinjenem časovnem spektru (Pomni, da se pri načrtovanju podatek celo med vzorčenjem obravnava kot zvezni podatek). Ker obravnavamo različne tipe podatkov in dogodkov, mora biti notacija podatkovnega pretoka ustrezno razširjena.

9.1. Podatkovna transformacija

Primere časovno kontinuiranih transformacij kaže slika 21. Dvojna puščica na linijah pretoka podatkov označuje neprekinjeno naravo podatkov.



Slika 21. Časovno zvezni podatkovni pretok.



Slika 22. Sinhroni podatkovni pretoki:
a) dvoumni diagram
b) spojeni tokovi (merged flows)
c) uporaba podatkovnega pomnilnika

Pri diagramu časovno diskretne transformacije (slika 22) je možna dvoumnost, saj lahko diagram interpretiramo na dva različna načina: Po prvem načinu se transformacija izvrši tedaj, ko se pojavita oba vhoda hkrati; po drugem načinu pa je transformacija možna tudi v primeru, da se vhoda ne pojavita istočasno. Vhod, ki se pojavi prvi, se shrani. Ko se pojavi še drugi in sta tako prisotna oba hkrati, je izpolnjen pogoj, da se transformacija izvrši. Da ne more priti do opisane dvoumnosti, je bil vpeljan koncept sinhrone transformacije.

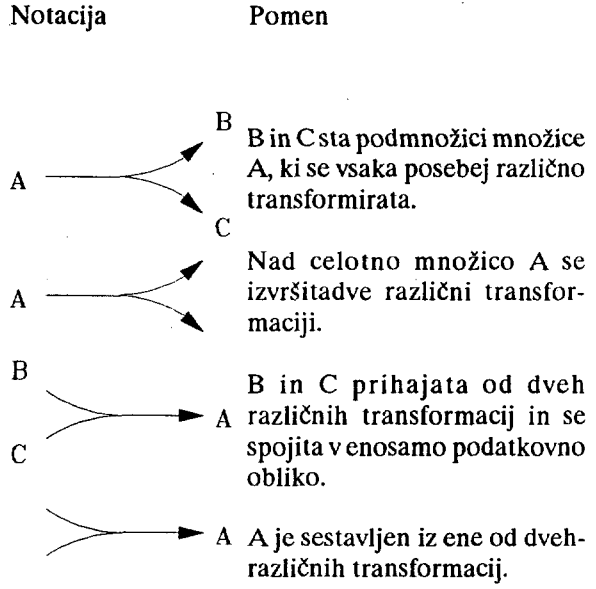
Pri sinhroni transformaciji moramo upoštevati naslednje dogovore:

- Možen je samo en diskreten vhod (vhodi za shranjevanje podat kov in vzbujanaja /glej spodaj/ niso upoštevani)
- Diskretni vhod je lahko sestavljen. Da se transformacija izvrši, morajo biti prisotni vsi elementi.
- Če obstaja več diskretnih izhodov, se morajo le-ti medsebojno izključevati.

Če upoštevamo gornja pravila, lahko transformacijo na sliki 22a predstavimo kot transformacijo na sliki 22b s sestavljenim vhodom ali kot transfor-

macijo na sliki 22c s podatkovnim pomnilnikom, ki pomni status sistema.

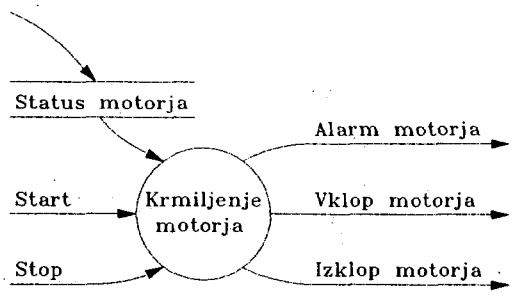
Podatkovni pomnilnik, ki je na sliki označen z dvema paralelnima črtama, kaže, da je podatek med dvema časovno diskretnima podatkoma zadržan. Pomen ostalih notacij t.im. spojenih podatkovnih pretokov je dan na sliki 23.



Slika 23. Notacije podatkovnih pretokov.

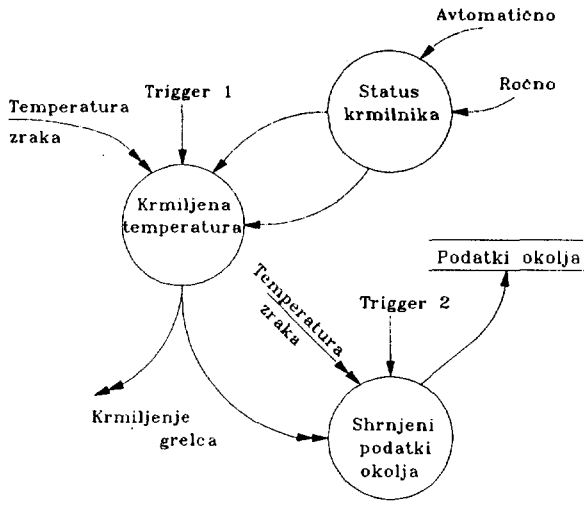
9.2. Krmilne transformacije in pozivi

Notacija krmilnih transformacij je pokazana na primeru na sliki 24. Dogodkovni pretoki in transformacije so označeni s prekinjenimi črtami. Vsaka sprememba krmiljenja je vezana na dogodek. Podatki o dogodkih okolja (ki v našem primeru predstavljajojo status motorja) se hranijo v posebnem pomnilniku.



Slika24. Krmilni podatkovni pretoki.

Koncept pozivov omogoča razmejitev podatkovnih in dogodkovnih pretokov. Ločimo tri pozive: ENABLE, DISABLE in TRIGGER. Njihova uporaba je pokazana na sliki 25. Pozivi se ne obravnavajo kot vhodi v pretvorbo (transformacijo), ampak preprosto kot stikala, ki so uporabljena zato, da se transformacija vklopi ali izklopi. Poziv TRIGGER požene transformacijo in se lahko uporablja za transformacijo časovno zveznih podatkov v časovno diskretne podatke.



Slika 25. Podatkovni pretok, ki kaže uporabo pozivov.

9.3. Pregled metod

Sistem podpira omejeno grafično notacijo za predstavitev splošne strukture RT sistema. Pravila, povzeta v spodnji tabeli 2, omogočajo preprosto preverjanje povezav vmesnikov med različnimi

Transformacija	Vhodi	Izhodi
Podatki	Podatkovni pretok	Podatkovni pretok
	Pozivi	Dogodkovni pretok
Krmiljenje	Dogodkovni pretok	Dogodkovni pretok
	Poziv	Poziv

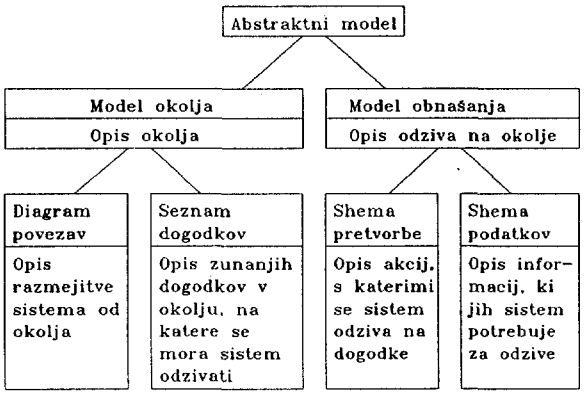
Tabela 1. Pregled vhodov in izhodov transformacij.

transformacijami. Ker so podatkovni pretoki jasni, je pri odločanju o delitvi sistema na module preprosta tudi uporaba načrtovalne heuristike za minimizacijo podatkovnih pretokov med posamez-

nimi deli sistema. Pri tem se moramo zavedati, da notacija upošteva spojene podatkovne pretoke in zato lahko enojna linija predstavlja kompleksno podatkovno strukturo in kompleksni vmesnik.

9.4. Zgradba modela

Opisano formalno metodo sta izdelala Ward in Mellor /WaM85/. Metoda sloni na dveh modelih: abstraktnem in izvedbenem. Modela se gradita sočasno, ker lahko visoko-nivojske izvedbene nadrobnosti vplivajo na nižje nivoje abstraktnega modela. Posamezne faze gradnje abstraktnega modela so razvidne iz slike 26.



Slika 26. Abstraktno modeliranje po Ward/Mellor-ju.

Na prvi stopnji se sistem določa glede na povezave z okoljem. Pri ten se uporablja diagram povezav (povezovalna shema) in seznam dogodkov.

Na drugi stopnji modeliranja se razvija vedenjski model (model obnašanja). Ta model kaže, kako se sistem odziva na aktivnosti v okolju. Razviti ga je mogoče na dva načina: po shemi transformacije ali po shemi podatkov.

Vsaka transformacija je oštevilčena. Z razčlenitvijo posameznih transformacij se lahko število množic diagramov povečuje, dokler operacije znotraj vsake transformacije niso več deljive. Pretvorbena specifikacija pripada vsaki transformaciji diagrama.

Pri načrtovanju opravila (kot enojnega opravila) lahko uporabimo običajne podatkovno pretočne metode. Alternativno metodo pa predstavlja uporaba nekega ustreznega višjega programskega jezika, n.pr. Module 2 /Bud85/.

10. ZAKLJUČEK

Obravnavali smo tri pristope k načrtovanju programske opreme RT sistemov, ki so zasnovani na enojnem programu, na priviligeranih programih ali na izvajanju več opravil hkrati. Zadnja metoda je najbolj splošna. Pred ostalima dvema ima prednost, ker uporablja abstraktne pojme (nabiralnike, kanale, signale in aktivnosti v MASCOT notaciji; ali podatkovne in dogodkovne pretoke s podatki in krmilnimi transformacijami v podatkovno pretočni notaciji) in omogoča načrtovalcu sistema, da se osredutoči na uporabniške vidike problema. Na kasnejši stopnji načrtovanja, ko je ugotovljena istovetnost različnih aktivnosti, obstaja možnost, da se načrtovalec opre na eno od ostalih dveh metod, če katera od njih omogoča boljše rešitve.

Metoda se v prvih korakih načrtovanja programske opreme ogradi od nadrobnih rešitev materialne opreme. Notranja struktura aktivnosti se lahko načrta tako, kot da so sekvenčno programirane. Dejanske izvedbene podrobnosti nabiralnikov, kanalov, signalov, aktivnosti podatkovnih in krmilnih transformacij postanejo ločen problem. Aktivnosti (ali transformacije) se lahko načrtujejo kot sekvenčni programi z uporabo standardnih tehnik /Pres82, Coo90/.

V novejšem času prevzemajo vse večjo vlogo CASE orodja. Z njimi lahko sistematično oblikujemo programsko in materialno opremo skozi celotni razvojni cikel. CASE orodja, ki so danes v uporabi, se najpogosteje opirajo na Ward/Mellor in Hatley/Pirbhai razširitvijo Yourdonove metodologije /HaP87, You88/. Primerjavo obeh metod najdemo v delu /KoK92/. Hatley/Pirbhai metodologija je predvsem zmogljivejča na nivoju specifikacij sistema in kot taka dobro dopolnjuje Ward/Mellor metodologijo. Skupaj omogočata učinkovito strukturno analizo opreme namenskih RT sistemov, ki smo jo v pričujočem delu poskušali pokazati na opisanih formalnih pristopih načrtovanja.

11. LITERATURA

/All81/ S.T.Allworth, Introduction to Rial-time Software Design. Macmillan, 1981.

/BenL84/ Bennett and D.A.Linkens (eds), Real-Time Computer Control. Peter Peregrinus, Stevenage, 1984.

/Bud85/ D.Budgen, Combining MASCOT with Modula-2 to aid the engineering of real-time systems. Software: Practice and Experience, 15(8), pp.767-93.

/Coo86/ J.E.Cooling, Real-Time Interfacing. Van Nostrand Reinhold (UK), ISBN 0-442-31755-7, 1986.

/Coo90/ J.E.Cooling, Software Design for Real-Time Systems. Chapman and Hall, ISBN 0-412-341-808, 1990.

/Gom84/ H.Gomma, A software design method for real-time systems. Communications ACM, 27(9), pp.938-49,1984.

/Hat84/ D.J.Hatley, A structured analysis method for large real- time systems. The Heap, DESIG Structured Lnguages, 7(2), pp.21- 38, 1984.

/Ise84/ R.Iserman, Process fault detection based on modelling and estimation methods-a survey, Automatica 20(4), pp.387-404.

/JaS75/ K.Jackson and H.R.Simpson, MASCOT-A modular approach to software construction, operation and test. IRE Tech.Note, No.778.

/KoK92/ B.Koroušić and P.Kolbezen, Razširitvi Yourdonove SADT metodologije za razvoj namenskih sistemov, Zbornik ERK 92, Portorož, B: 27-30, 1992.

/May78/ G.J.Myers, Composte Structured Design. Van Nostrand Reinhold, 1978.

/Par72/ D.L.Parnas, On the criteria to be used in decomposing systems into modules. Communications ACM 15(12), pp.1053-66, 1972.

/Pres82/ R.S.Pressman, Software Enginnering: a Practitioner's Approach. McGraqw-Hill, 1982.

/StMC74/ W.P.Stewens, G.I.Myers and L.L. Constantine, Structured design. IBM Systems J.13,p.15, 1974

/Sim84/ H.R.Simpson, MASCOT 3. IEE Colloquium, Paper No.3, 1984.

/WaM86/ P.T.Ward and S.J.Mellor, Structured Development for Real- Time Systems. Englewood Cliffs, NJ:Yourdo Press, ISBN: 0-13- 854787-4, 1985.

/You88/ E.Yourdon, Modern Structured Analysis. A Prentice-Hall Campany, 1988.

/You79/ E.N.Yourdon, Classics in Software Engineering. Yourdon Press, 1979.

/Zav84b/ P.Zave, An overview of the PAISLey project-1984. Tech. Rep., AT and T Bell Laboratories, 1984.