Tarkvara dünaamika

Reaalajasüsteem

Sellistes süsteemides on arvuti liides juhitava keskkonna ja inimese vahel. Piiri pealolevad süsteemid (mis on kahe keskkonna vahel) on üldiselt tehislikud.

Enamasti on juhitav keskkond olemas ja arvutisüsteem peab sobima selle keskkonnaga.

Enne ehitamist uuritakse juhitavat keskkonda. Arvuti paigutamisel sinna keskkonda need omadused aga muutuvad. Muutmise iseloomu pole võimalik ette ennustada. Oleks vaja valida võimalikult vähe keskkonda iseloomustavaid parameetreid, mis täpsustuvad projekti edenedes.

Nt. lennumasinad, mis ilma arvutita on lennuvõimetud.

„Mõistlik inimene üritab kohaneda maailmaga, mitte kohandada maailma enda järgi".

Ajakriitilised süsteemid

Arvuti tahab jälgida ja juhtida reaalse maailma füüsikalist protsessi. Osad protsessid on looduslikud (mida inimene üritab muuta), osad inimese loodud tehislikud protsessid. Arvuti tegevus ja kiirus peab olema suurem kui reaalse maailma protsessi muutumise kiirus.

Looduslikud protsessid toimuvad igaüks erineva kiirusega, seega on neid ka erinevalt vaja mõjutada.

Arvuti sekkumiseks protsessi tuleb kindlaks teha, millal on õige hetk. Matemaatiliselt võib selle õige hetke leidmine võtta kauem aega kui protsess ise töötab, seega on sekkumise hetk vaja leida ligikaudselt (kasutada aproksimeerimist).

26. veebruar 2003
UTC - Eksisteerib universaalne aeg. Baseerub Cs-kellal. Kehtestati tagantjärele. Kasutatakse üle maailma. Väline aeg. Võetakse mingite raadiosaatjatega. Kohalikes arvutivõrkudes püütakse sellega kelli sünkroniseerida. Tähtis sõjalistes rakendustes.
Probleem sõjanduses allveelaevadega – vee all ei saa sünkroniseerida kelli.
Sünkroniseerimine toimub nt. kord aastas, kui laev pinnale tõuseb. Algus – 1582, kui tuli Gregorianuse kalender.

Lokaalne aeg – Võib olla sama, mis universaalaeg. Kohandatud nt. suveajaga, päikseajaga. Arvuti riistvaraline kell, mida sünkroniseeritakse aegajalt. Seda aega hoitakse igas süsteemis (nt. akude pealt.).

Missiooniaeg – Suhteline aeg. Saab alguse mingist süsteemsest trigerist. Reaalajasüsteemis hetk, kui süsteem tööle pannakse. Alamsüsteemil võib olla omaette missiooniaeg. Kogu süsteemil võib ka üks ühine missiooniaeg olla. Kõik sõltub konkreetsest rakendusest. Pikkadel protsessidel on mõistlik missiooniaega tükeldada ja siduda konkreetse protsessiga.

Keerulisemad on need ajad, mis näitavad aja voolu regulaarsust.

Loks – kahe järgneva ajaühiku vaheline pikkus võib olla erinev (kahe kella ajaühiku pikkus võib olla erinev). Selle tõttu muutub süsteem ebastabiilseks. Kellad loevad kvartsi võnkumisi. Probleem on võnkumiste täpsuses ja nende sõltuvus vibratsioonist, temperatuurist jne. Loks on ühe kella sees oleva võnkumise mitteregulaarsuse mõõt. Sama protsess kahe kella puhul on uit (nähtus, mis tekib, kui liita kaks erinevat perioodilist funktsiooni kokku.). Vajalikud, et hinnata erinevate arvutivõrgus olevate kellade tööd. Vaja hinnata näiteks algoritmide töö täpsust.

Nihe – Suhteliselt regulaarne erinevus kahe kella vahel. (nt. käekell käib ööpäevas mingi ajaühiku ette). Muutub suhteliselt aeglaselt. Muutumise kiirust näitab triiv. Näitab kui kiiresti nihe muutub ajas.

Aeg ei ole tavalise inimese jaoks probleem, arvuti sees võib see aga suureks probleemiks kujuneda. Eriti veel siis, kui väljastpoolt ei saa kellasid sünkroniseerida.

Kogu aja sünkroniseerimise probleem on verifitseerija asi. See on inimene, kes peab tõestama, et kogu kood, mis tehakse, töötab süsteemi ulatuses õigesti. Verifitseerija ei saa omada arvutisüsteemi sisest vaadet, ta peab vaatama, et arvuti ja keskkond koos töötavad õigesti. Arvuti konfiguratsioon võib olla suvaline (kiiparvutist kuni suurte globaalsete võrkudeni (nt. keskkonnaseiresüsteemid)).

Realiseerija vaade piirdub ainult arvutisüsteemi sisese vaatega. Spetsifitseerija vaatab eemalt nii keskkonda kui arvutisüsteemi. Ta peab panema koos käima keskkonna, arvuti ja inimesed. On vaja infot keskkonna sees toimuvate protsessida dünaamika kohta. Kogu info tuleb toimetada arvuti sisse, projekteerides süsteemi aega ja ajakitsendusi.

Osa andmeid on vaja vaid anduritelt kokku korjata, osa aga keskkonnas välja kutsuda, et neid siis lugeda. (nt. Inimeksperdid. Eksperdid ei taha niisama oma teadmisi jagada.) Keskkonnast kogutud info on erinevate skaalade peal (mikrosekundist kuni aastani). Aega ei mõõdeta vaid minutite ja sekunditena. Ajakitsendusi võib anda edasi ka meetrites (tõkkepuude tõstmise ja langetamise süsteem. „Tõkkepuud alla lasta siis, kui rong on ülesõidust x meetri kaugusel". See on määratud keskmise väga raske rongi kiirusega märja ilma korral.). Vaja teisendada meetrites antud ajahinnang ja teisendada see sekunditesse või millisekunditesse.

Erinevad eksperdid annavad tõenäoliselt erinevad hinnangud. Seega on vaja leida kõige õigem ja süsteemi jaoks korrektseim hinnang. Ajaühikud on vaja omavahel sobitada. Ühe minuti üleminek teiseks minutiks on aeglane protsess ja see võtab mitu mikrosekundit aega. Vaja leida optimaalseim lahendus. Mitme väärtuse korral tuleb valida välja kõige mõistlikum väärtus. Mõistlik sõltub projekti eesmärgist. (nt. maksimaalse turvalisuse saamiseks tuleb

valida väärtus, mis annab parima turvalisuse astme. See väärtus ei pruugi aga sobida nt. maksimaalse kiiruse saavutamiseks.).

Paljud protsessid toimuvad viitega, protsessid on inertsed. Vaja protsesse ette ennustada. Mida paremini ennustatakse, seda täpsemini süsteem töötab. (nt. temperatuuri suurendamine ahjus. Energia juurdeandmine ei anna kohe temp. kasvu.).

Samuti tekib probleeme semantikaga. Tarbekeeles sõna „samaaegselt" on väga laia tähendusega. Samaaegsus tähendab tavainimese jaoks minuteid, vahel ka mitmeid minuteid. (nt. samaaegne kahe kraani keeramine, mis asuvad teineteiselt mitmekümne meetri kaugusel). Vaja kindlaks teha, mida see samaaegsus tegelikult tähendab. Vaja leida minimaalne ajaintervall, mille jooksul toimuvaid protsesse saab veel lugeda samaaegselt toimuvaiks.

Samaaegsus ei ole tegelikult programmeerija probleem. Kui aga süsteem töötab valesti, siis süüdistatakse programmeerijat.

Nt. Muutujad oleks vaja väärtustada ajatemplitega (kehtivusintervall), mis näitab, kui kaua võib ühe muutuja väärtust, mis on saadud reaalsest keskkonnast, kasutada. (nt. temperatuuri mõõtmise kehtivusintervall on pikem, kui rõhu muutumise oma. Rõhk muutub kiiremini kui temperatuur.).

Oluline on ka kosteaeg – kui kiiresti peab süsteem reageerima mingile keskkonnas toimuvale sündmusele. (hetk, kui märgati keskkonnas toimunud muutust kuni hetkeni, millal saadeti reaktsioon keskkonda tagasi.).

Seega on vaja lisaks muutuja väärtuse leidmisele ja selle leidmise kiirusele arvutada välja ka selle väärtuse muutumise kiirus.

Kõige sellega tegelebki spetsifitseerimine.

Arvestada tuleks, et praktikas ei ole resolutsioon ehk eristatav aeg ebatäpne. Kui kella resolutsioon (ehk ajaarvestuse täpsus) on 1 minut, siis seda on võimalik mõõta ca. kaheminutilise täpsusega. See on tingitud mõõtmisveast. See ei ole üldse programmeerija probleem.

Ajakitsendusi realiseerides tuleb kokku puutuda aja semantikaga. Aeg võib olla pidev e. tihe või hõre e. diskreetne. Tiheda aja korral saab iga ajapunkti vahele panna veel kolmanda ajapunkti. Hõreda aja korral seda ei saa teha. Kui diskreetses ajas toimub kahe ajahetke vahel kolm sündmust, siis nende toimumise täpset hetke s.t. toimumisjärjekorda ei ole võimalik määrata.

Tihedat aega saab modelleerida reaalarvudega.

Universum on lõpmata suur. See on täidetud lõpliku arvu aatomitega. Seega on universum hõre. Nii on ka kogu ümbritsev keskkond hõre. Seega peaks olema ümbritsev keskkond diskreetne. Puudub igasugune tõestus, et keskkond on pideva iseloomuga. Ajaintervalli pikkus on antud juhul kõige väiksem modelleerija poolt eristatav ajaühiku pikkus. Seega ongi kogu aeg diskreeditud.

Kestvus – kahe sündmuse vaheline kaugus teineteisest. Parempoolne lõpp-punktist lahutatuna vasakpoolsest lõpp-punktist annabki kestvuse.

Igal programmil on tegelikult oma aeg. Kui töötab kümme programmi, siis tuleb rääkida ka kümnest erinevast ajakomplektist. Kõik programmid on omavahel seotud ja mõjutavad teineteist. Neid on võimalik jälgida, kuna programmide arv on lõplik. Omavaheliste

interaktsioonide kirjeldamiseks tuleb teada kõigi interaktsioonis osalevate komponentide (s.t. programmide) ajakomplekte.

Ajamudel peab olema nii lihtne kui võimalik. Samas peab ta olema piisav, et analüüsida kõiki ajastamisega seotud omadusi.

Ajastamist iseloomustavad:
- jõudlushinnangud (saab mõõta ajaga, mis on kogu süsteemis ühine)
- muutujate ja sündmuste kehtivusaegade hindamine
- üksikute huvipakkuvate osade interaktsioonide ajalise korrektsuse hindamine.

Viimased nõuavad igale osakesele oma ajasüsteemi (muidu ei ole võimalik neid hinnata).

Kogu süsteemi on lihtne projekteerida pidevas ajas, sest inimesed on sellega harjunud. Ühel hetkel on vaja minna diskreetsesse aega. Sel juhul kaotavad aga paljud matemaatilised teooriad kehtivuse. Seda ei saa jätta programmeerija teha, sest tal puudub vastav haridus ja kogemus.

Aeg võib olla:
- meetriline (saab mõõta kahe ajapunkti vahelist kaugust, sõltumata ajaarvestussüsteemist) või topoloogiline (kaugus pole oluline, tähtis on järjestus). Järjestuse aluseks võib olla ajaline mõõde, kuid võib ka mingi muu olla (nt. spektriline – kõige punasem enne, kõige sinisem viimasena).
- tõkestamata (nt. tavaline loendur) või tõkestatud (süsteemil on mingi maksimaalne aeg. Raske ennustada, kas maksimaalne aeg on piisav).
- lineaarne, hargnev, tsükliline või mittetsükliline.
- tingimuslik (vaadatakse hargnevat ajapuud, kus igas hargnemispunktis on mingisugused tingimused, mis määrab ära, millist haru mööda liikuda). See ei ole reaalajasüsteemides oluline, kuna üldiselt sellises süsteemis midagi valida ei saa. Väliskeskkond dikteerib käitumise.
- pööratav (nt. simulatsioon lahesõja ajal, kus kõik variandid mängiti enne tegelikku vägede liikumist läbi), rangelt kasvav või suhteline. Reaalajasüsteemides on sageli kõik eelmised variandid korraga kasutusel. (nt. bioloogias on aeg rangelt kasvav ja peaaegu mittepööratav – termodünaamiline aeg; psühholoogias on kasutusel vaid suhteline aeg. Aja alguspunkt nihkub, kõik mineviku sündmused on selged, kõik tuleviku asjad on ebaselged. See aitab kirjeldada inimteadust.)

Aja näited
- Tavaline andmetöötlussüsteem töötab lineaarses topoloogilises ajas. Kindlalt järjestikune. Oluline on sündmuste järjestus, mitte nende ajaline kestvus.
- Lineaarne meetriline diskreetne ja rangelt kasvav aeg on kasutusel algsetes reaalajasüsteemides. Kogu süsteemis oli ainult üks aeg, mis liikus kindlas suunas. Kõiki sündmusi sai kirjeldada vaid ühel ajateljel.
- Hargnev, topoloogiline või meetriline ja diskreetne aeg, mille peal töötavad mõned ajaloogikad. Eelmine aeg on antud ajaks teisendatav.
- Lahesõja simuleerimisel kasutati lineaarne diskreetne kasvavaid meetrilisi aegu. Lisaks veel mitu komplekti pööratavaid diskreetseid meetrilisi ja palju suhtelisi aegu. Simuleeritakse arvutivõrgu peal; igal programmil oma isiklik aeg, neid aegu on võimalik tagasi pöörata. Kui mingi sündmus mõjutab teisi ajas etteläinud sündmusi, tulevad viimased ajas tagasi soovitud punkti ja unustavad kõik vahepeal tehtu.

Reaalajasüsteemis peaks korraga olema kasutusel
- Pööratav aeg (nagu füüsikas)
- Rangelt kasvav aeg (nagu termodünaamikas või bioloogias)
- Suhteline aeg (nagu psühholoogias)

Vähemalt üks rangelt kasvav aeg peab olema meetriline. Sellisel juhul saab seda kasutada välisvaatleja.

Paljud mudelid kasutavad liiga lihtsaid ajakomplekte, mis kirjeldavad reaalajasüsteemi vaid osaliselt.

Pööratav aeg on realiseeritud mitmes tavalises programmis (Wordi Undo). Ka eriolukordade töötlemise süsteemis tuleb ajas tagasi minna, kui arvutamise käigus selgub, et mõned andmed on puudu ja nende saamiseks tuleb minna ja need hankida, seejärel teostada arvutused uuesti. Reaalajasüsteemis ei saa aega täielikult pöörata; kui süsteemist on juba reaktsioon välja läinud, siis seda enam tagasi võtta ei saa. Samuti võib mingi reaktsiooni väljaarvutamine reaalajasüsteemis võtta kauem aega, kui on lubatud ajakitsendustega. Ainus, mida sel juhul teha saab, on kinnitada, et vastus ei ole usaldusväärne.

Rangelt kasvavas ajas ei ole asjad enam pööratavad. Tavaliselt saab sel juhul protsessi pöörata tagasi kuni mingi hetkeni. Niikaua, kuni arvutisüsteemist mitte mingit infot välja ei ole antud, saab põhimõtteliselt protsessi pöörata (kui ei ületata lubatud aega). Kui signaalid on välja läinud, ei ole pööramine enam võimalik.

Liikuva algusega suhtelist aega kasutatakse protsessidevaheliste interkatsioonide kirjeldamiseks. Kui kaks protsessi töötavad, mis vahetavad aegajalt infot, siis üks protsess ütleb teisele, kui vanu andmeid ta veel aktsepteerib. Andmete vanus ongi liikuva algusega suhteline aeg. Kui ühe protsessi andmete genereerimise aeg ei piisa, siis tuleks ühest protsessist käivitada kaks koopiat väikese nihkega. Sellisel juhul andmete saamise tihedus suureneb. Kui töötab ka ühest protsessist kaks koopiat, siis kahe protsessi vahel eksisteerib ikkagi üks ja seesama suhteline aeg hoolimata sellest, et ühest protsessist eksisteerib kaks koopiat.

Q-mudel
Selline mudel koosneb arvutusprotsessidest. Iga protsessi käivitus on rangelt kasvavas termodünaamilises ajas, mis liigub edasi diskreetsete hüpete kaupa. Iga protsessi arvutuskäigu sees võib aega pöörata eeldusel, et arvutuskäik lõpeb ettenähtud ajal ära. Rangelt kasvava ajaühiku sees on suhteline aeg, milles on üles loetletud antud protsessiga seotud (aja)parameetrid. Lisaks on kahe protsessi vahel veel suhteline aeg, mis määrab andmete vanuse (kehtivusintervall). Viimane aitab elimineerida andmeid, mille vanus ei võimalda neid enam kasutada. Samuti aitab see kooskõlastada mitme erineva programmi tööd.

Digitaalne filosoofia väidab, et maailmas juhuslikkust ei ole. Kõik asjad on määratud põhjuslike seostega. Juhuslikkus tekib sellest, et vaatleja ei tea kõiki põhjuslikke seoseid. Arvutiteadus baseerub algoritmidel. Algoritmide vahel on teada täielikud seosed. Iga algoritm peab tööd alustama mingitel kindlatel tingimustel. Samuti lõpetama mingitel kindlatel tingimustel. Algoritm peab olema lõpetatud, et algoritmi kohta saaks midagi öelda. Programmi hindamiseks peab see olema alati lõplik. See on arvutiteaduse dogma, mis kujunes välja 70. aastatel.

Kõik reaalajasüsteemid ja ka paljud muud rakendused töötavad kogu aeg. Sellistes süsteemides programmid oma tööd ei lõpeta. Programmid töötavad lõputus tsüklis. Alamprogrammid töötavad, lõpetavad, ootavad mingi aja ja siis alustavad uuesti tööd. Kogu süsteem aga oma tööd ei peata.

Hakati uurima, millised on projekteerimise ja realiseerimise raskused. Osaliselt selle tulemusena tekkis objekteorienteeritud programmeerimine. Sellest arenes välja komponenttehnoloogia, mis arenes edasi agentidel baseeruvaks tarkvara loomiseks.

Reaalajasüsteemid töötavad mittetäielikult määratud informatsiooniga. Selle pärast on võetud kasutusele aja mõiste, mille abil lihtsustatakse põhjuslikke seoseid. (Kui inimene elas koopas, siis käidi koos jahil, koos söödi, koos läks kõht tühjaks jne. Aeg ei olnud selle juures

oluline.) Planeerimisvajaduse tekkimisel oli vaja protsesse sünkroniseerida, sealt tekkis aja mõiste. Aja mõõtmine on kogu aeg täpsustunud.

Algoritmiteoorias pole aega vaja, vajalik on järjestus. Seega püüti 60.-70. aastatel arvutitest aega välja juurida. 80. alguses seoses objektide tekkimisega (mis ei mahtunud algoritmiteooria alla) ja reaalajasüsteemide arenguga ning rakenduste keerukuse kasvuga hakkas aeg arvutitesse tagasi tulema. Tekkisid vastavad ajateooriad. Vaja välja valida, millist ajamudelit kasutada. OMG ajamudel on praegusel hetkel enamlevinud, kuid paralleelselt eksisteerib ka teisi. Võimalikuks sai matemaatiliselt korrektse mudeli tegemine (Q-mudel). Q-mudel on empiiriline, kuid ta on esitatav 2. järku predikaatarvutuse kaudu. Seega on selle mudeli paikapidavust võimalik tõestada.

Uue paradigma vajalikkus:

- Arvuti rakendused muutusid kiiremini kui vastav teooria.
- Vaja leida põhjendus ajakitsenduste kasutamiseks ja rakendamiseks. Ajakitsenduste sissetoomine tagab Turingi masina teooria kasutamise võimalikkuse algoritmi enda sees.
- Selgus, et algoritmiteooria ei tööta reaalajasüsteemides. Ta baseerub staatilise keskkonna eeldusel (Nähtus tingib teooria loomise, mis tingib eksperimendi. Teooriat täpsustatakse täiendavate vaatlustega. Nähtus ise aga ei muutu! Algoritmi aluseks olev aksiomaatiline baas ei tohi muutuda algoritmi töö käigus. Kõik arvutuseks vajalikud teooriad peavad kehtima kogu arvutamise aja jooksul. See ei kehti reaalse ja tehismaailma piiril). Staatilise keskkonna tingimused kehtivad aga alati mingi kindla aja. See aeg ongi vaja kindlaks määrata.
- Ajalised kitsendused pannakse paika nn. „eksperthinnangutega". Keskkonna erinevad osad omavad aga erinevaid dünaamilisi karakteristikuid, mida on vaja hinnata. (Keeruline ülesanne.) See aga tähendab, et programm ei pruugi kõigis keskkonna osades ühtemoodi töötada.

Ka muudes eluvaldkondades on kitsendused sama levinud (helilooja peab arvestama teose loomisel nii orkestri kui ka publikuga – tegevus on ülimalt keeruline). Seega ei ole tarkvaraprojektid oma olemuselt palju keerulisemad kui muu valdkonna protsessid.

Mõtestatud tegevuse eelduseks on staatilise keskkonna säilimine. Seda saab realiseerida ajakitsendustega. Vaja projekteerida süsteem nii, et ajakitsendustest kinni peetakse. Ajakitsenduste sisseviimisel saab loota ekspertidele, uuringutele ja heale õnnele. Töö käigus on vaja kogu aeg oma esialgseid hinnanguid täpsustada. Esialgsete hinnangute põhjal tuleks teha formaalne mudel. Mudeli peal tuleks näidata ajakitsenduste paikapidavust. Detailide lisamisel saab paljusid hinnanguid täpsustada. Perioodiliselt on vaja töö käigus täpsustatud ajakitsendusi kontrollida. Et kõike seda läbi viia, peavad teooriad püsima muutumatuna. Kui teooria muutub, on samade tingimuste korral tulemused erinevad ja kontroll ei ole võimalik.

Väliskeskkonna erinevad osad ei käitu ühtemoodi. Ajakitsendused ja parameetrid võivad igal osal olla erinevad.

Paradigmad – üldine ideaalse arvutisüsteemi või selle osa arhitektuur. Tavaelus tähistab mõttemalli. (lõpetav *versus* mittelõpetav programm. „Lõpetav programm on hea, mittelõpetav on paha.")

Algoritmiteooria kohaselt ei ole ka objektorienteeritud programmeerimine algoritmiteooria järgi kirjeldatav. Põhjus selles, et objektidega kaasas olevaid meetodeid saab kutsuda välja suvalises järjekorras. Kõiki võimalikke meetodite kutsumise järjekordi on aga võimatu täpselt üles lugeda, sest kombinatsioonide arv on suur.

Arvutusmudel – formalism, milles kirjeldatakse arvutusprotsessi nende erinevates staadiumides (kuidas arvutused toimuvad). Näitab mitte ainult protsessi vaid ka algandmete esitusviisi ja ka protsessi enda ehitust. (nt. olekumudelid, CSP, CCS. CSP on mudel, kus kirjeldatakse üksikuid järjestikprotsesse mingis formalismis, mis töötavad perioodiliselt. Need protsessid vahetavad mingitel ajahetkedel andmeid. Sarnane Q-mudeliga. CCS-s (tekkis 1980) ei tehta mingeid eeldusi programmijupi enda kohta. Oluline pole see, mida ta

teeb, vaid see, mida ta teistele saadab. Sellist koodi vaadatakse CCS-s kui arvutusagenti. CCS-s jälgitakse agentide omavahelisi seoseid.)

Enne algoritmi valikut tuleks teha uuringud, kuidas probleemi lahendada. Probleem võib lahenduda lihtsamini kui algul arvatud, samuti võib olla probleem mittelahenduv.

LIMITS

Reaalajasüsteem on kogum omavahel suhtlevaid dünaamilisi süsteeme, millest üks on arvutisüsteem.
Reaalajatarkvara sellises süsteemis on realiseeritud nõrgalt seotud, korduvalt käivituvate, kuid oma olemuselt lõpetatud programmide näol. Viimased kaks simuleerivad lõputut protsessi. Tänu sellele on igat üksikut täitmist võimalik verifitseerida. Välisvaatlejale tundub, et programm on lõputu, kuid tegelikult on programm tsüklilise iseloomuga.

Reaalajasüsteemi tarkvara on üks terviklik mittelõpetav programm, millele on lisatud elususe (programm teeb kunagi midagi kasulikku), ohutuse (programm ei tee kunagi midagi kahjulikku sõltumata sellest, millal programm käivitati) ja ususe (ressurss eraldatakse kuitahes pika lõpliku aja jooksul – see ei toimi reaalajasüsteemides) printsiibid. Ausus realiseeriti esialgses Interneti protokollides – kõik sõnumid hoiti alles, püüdes need igal juhul kätte toimetada. Ruuterid ummistusid sõnumitest, mille adressaat oli igaveseks võrgust kadunud. Lisati ajakitsendus: kui sõnumit ei saa 4 päeva jooksul kätte toimetada, siis sõnum unustatakse. Ususe paradigma ei toimi reaalajasüsteemides.

05. märts 2003

- Alternatiivsed arvutusmudelid reaalajasüsteemides:
  - Lõplikud automaadid, olekudiagrammid ja atribuutautomaadid (esimene ja viimane on oma olemuselt interaktsioonimasinad).
  - Petri võrk (ei ole interaktsioonimasin, sest mäluefekti temas ei ole).

Interaktsioonmasinad jagunevad segmenteeritud ja mitmevoolisteks interaktsioonimasinateks. Interaktsioonimasin ei kasuta oma töös mitte x-i üksikut väärtust vaid kasutatakse andmevoogu, milles võib olla järjest lõpmatu palju väärtusi. Sarnaste sisendväärtuse korral ei pruugi väljundväärtus sama olla (sest arvutussüsteemis endas võib olla kasutusel mingi mäluelement). Sellises süsteemis ei toimu andmete töötlemine üksikute algoritmide kaupa. Mitmevoolised masinad omavad mitut sisend- ja mitut väljundvoogu.

Lõplik automaat võib ka genereerida lõpmatu väljundite jada. Kui tal puudub sisemine mälu, siis väljundid on kergesti esitatav sisendite kaudu, s.t. on kergesti esitatav ja analüüsitav.

- Teine arvutusmeetodite klass on protsessialgebrad ja ajaloogikad (enamik neist on taandatav Petri võrkudele.)

- CSP protsess on lähedane interaktsioonimasinatele.

- Abstraktsed andmetüübid ja OO arvutusmudelid (paradigma e. mõttemaailm võib olla objektorienteeritud, tegelik arvutusmudel on aga mitmevooline interaktsioonimasin).

Abstraktsed andmetüübid ja objektorienteeritud lähenemisviis on oma olemuselt samad, viimastele on lisatud vaid pärilikkuse omadused. Objektides on hulk meetodeid, mis on algebralises mõttes tavalised operatsioonid. (nt. kahe täisarvu või reaalarvu liitmine. Liitmine kui meetod omab ühesugust mõtet mõlema objekti korral).

Q-mudel

Kõige lähedasem sellele mudelile on abstraktsete protsesside omavahelise suhtlemise paradigma. Vaja kirjeldada kitsendused, mida on vaja selleks, et kogu süsteem toimiks ja nende protsesside omavahelise andmevahetuse tulemusena realiseeruks ette nähtud eesmärk.

Tavaliselt vaadatakse arvutusprotsessi kui muutumispiirkonnale funktsiooni väärtuste vastavusse seadmist (klassikaline algoritmiteooria). Sellest piisab andmetöötlussüsteemide loomiseks, kuna arvutusprotsessi aluseks oleva teooria eeldused ei muutu vastavusse seadmise protsessi käigus. Ühele muutumispiirkonna punktile vastab alati mingi kindel funktsiooni väärtus. See vastavus on püsiv ja see ei muutu kunagi.

Algoritmiteoorias on kasutusel sõnumivahetuseks kahte põhimõtet (FIFO ja LIFO). Teisi sõnumite edastamise ideoloogiaid ei ole. Garanteeritud on sõnumite kindel järjestus. Lisades siia juurde klassikaline aususe printsiip (et teised programmid ei trügi vahele ja ei paiguta sõnumeid valel ajal valesse kohta), ei teki mingeid probleeme. Probleeme võib tekkida siis, kui mõni programm ei taha saada esimest sõnumit vaid näiteks kolmandat sõnumit. Üks variant on lugeda ka eelmised sõnumid välja ja visata need minema (sel juhul on need sõnumid kõigi teiste protsesside jaoks ka kadunud). Teine võimalus on oodata, kuni järjekord jõuab teda huvitava sõnumini (ootamine võib aga osutuda lõpmatult pikalt). Seetõttu ei saa selliseid andmevahetuse mudelid reaalajasüsteemidele.

Reaalajasüsteemides ei ole protsesside töötamise järjekord kontrollitav (seega ei pruugi sõnumid LIFO ja FIFO skeemides olla õiges järjekorras).

Paljudel juhtudel sõltub reaalajasüsteemis arvutuse õigsus andmete vanusest (nt. temperatuuri ja rõhu mõõtmine). Seega on osade muutujate väärtusi vaja hoida erilise tähelepanu all. Seega on vaja teada sisendandmete ja väljundandmete vanust,

sõnumivahetuse algusaega, protsesside käivitus- ning töötamisaegu. Kahte viimast on võimalik kätte saada lihtsate vahenditega. Teisi ei saa lihtsat ajamudelit kasutades välja lugeda.

Probleeme saab elimineerida ajalipikute lisamisega sisendmuutuja väärtustele. Sisendmuutujate väärtuste hulk oleks vaja läbi korrutada ajahetkede hulgaga. See aga on keerulise ülesande asendamine veel keerulisemaga.

Kahe hulga korrutamine: <x,y> * <a,b> = <xa,xb,ya,yb>
Kahe lõpliku hulga korral on korrutis ka lõplik hulk.

Selline lähenemine võimaldab täiesti kindlalt eristada kahte ühe ja sama protsessi erinevat täitmist.

Kahe protsessi omavaheliseks andmevahetuseks luuakse nende vahele andmevahetuskanal. Andmete edastamiseks antakse teisele protsessile edasi viit andmete asukohale. Olgu kaks protsessi pi ja pj. Esimene genereerib andmeid teisele protsessile. Sel juhul nimetatakse kanaliks protsessi pi andmete projektsooni protsessi pj määramispiirkonnale. Lisada tuleb muidugi aeg, millises vahemikus genereeritud andmeid on vaja üle kanda. Samuti tuleb määratleda, millisel hetkel toimus ülekanne.

Kanal võimaldab üle kanda kõik andmed, mis ühest protsessist tekkinud on. Ülekande mahu piiramiseks on sisse toodud kanalifunktsiooni mõiste. Kanalifunktsioon valib ülekantavatest andmetest mingi alamhulga, mis tegelikult teise protsessi määramispiirkonda projitseeritakse.

Q-mudel võimaldab ajahetkede hulga abil ette anda käivitusmustri. Kõigi protsesside väljundid varustatakse ajalipikutega, mis määravad ära protsessi käivitushetke. (Protsessi algus kirjeldab arvutamise aluseks olevate andmete vanust; protsessi lõpp on aga muutuv ja võib sõltuda mitmes tegurist).

Q-mudel kombineerib analüütilist ja mitteformaalset analüüsi. Formaalsel analüüsil on teada teisendused, kanali kirjeldus ja kanali funktsioon. Selle info põhjal saab ehitada protsessi animaatori. Teisenduse taga olevat algoritmi ei ole vaja teada. Oluline on see, kui kaua tegevus kestab, milliseid andmeid vajatakse ja kellele andmed saadetakse. Animatsiooni abil on võimalik täpsustada ajaparameetreid.

Q-mudel soodustab koostööd tarkvara loojate, süsteemidisainerite ja juhtimisteoreetikute vahel. Q-mudeli korral tekib koostöö alus kohe pärast kasutajanõuete fikseerimist. Samuti sunnib Q-mudel arvestama projektis ohutust, veakindlust ja usaldusväärsust.

väärtuste väljastamisel tuleks neile lisada ka usaldatavuse lipik. Kui seda ei ole, siis ei saa teised protsessid neid andmeid vahetult kasutada (käivituvad eriolukordade töötlemise süsteemi osad, veaparandused jne.) Kogu see protsess tuleb samuti projekti lisada. Q-mudeli korral on põhjus, miks neid teisendusi projekti lisada, ilmutatult näha. Eriolukordade töötluse lisamine muudab tavaliselt kogu esialge süsteemi parameetreid, seega on mõistlik eriolukordade töötlust arvestada kohe projekteerimise alguses.

Q-mudel *versus* andmevoomudel

Andmevoomudel on samuti mitteformaalne meetod. Ei ole teada, millises järjekorras toimub alamprotsesside käivitamine ja täitmine andmevoomudeli sees. Sellise mudeli korral saab kasutada paralleeltöötlust. Iga protsess võib töötada eraldi riistvara peal; vaja on vaid organiseerida andmevahetuse erinevate protsesside vahel. Puudulikult on aga fikseeritud andmevoogude semantika.

Kõik andmed on Q-mudelis ajalipikutega, andevoogude semantika on täpselt fikseeritud. Võimalik on ajas selektiivne andmeside (see ei ole võimalik andmevoomudeli korral). Kogu seda infot on võimalik kirja panna reaalaja UML-is. Seda infot saab kasutada jõudlusanalüüsis ja planeerimisel (projekteeritud klassimudel paigutatakse hajusalt arvutivõrku). Olemasolevate vahenditega ei saa teostada interaktsioonide ajalist analüüsi.

Q-mudel ja HRT-HOOD

HRT-HOOD on üks populaarsetest objektorienteeritud tööriistadest, mis praeguseks asendatud UML-ga. Sisse on võimalik viia mõned ajakitsendused, neid on võimalik aga sisse viia alles füüsilise projekteerimise etapis (objektide paigutamine protsessoritele). Uuritakse käivitusperioodi, minimaalset sõnumi saabumise aega (minimaalne kahe käivituse vaheline periood). Q-mudel käsitleb aga perioodilisi ja juhuslikke korduvtäitmisi suhteliselt sarnaselt.

Reeglina ei pööra teised mudelid tähelepanu ajafaktorile. Neid kasutatakse andmetöötluse modelleerimiseks. Kõikide ajamudelite puhul on aga oluline aja keerukuse aste, mida neis mudelites kasutatakse. Lihtsamates mudelites saab mõõta sündmuse kestust ja kaugust. Ei ole võimalik uurida sündmustevahelisi interaktsioone.

Enamikke mudeleid, mis ei käsitle ajafaktorit, on võimalik teisendada Q-mudeliks, lisades juurde vajalikud ajaparameetrid. Kogu teisendus ei saa olla aga ühene, sest Q-mudelist tagasiteisendamisel ei pruugi saada esialgset mudelit.

12. märts 2003
Q-mudeli korral on lisatud protsessile käivitushetkede hulk, mis näitab, millisel hetkel käivitunud protsessiga on tegemist.

Kanalifunktsioon piirab kanali läbilaskevõimet nii, et tarbija saab öelda, millise ajatempliga andmeid ta saada soovib.

Iga objekt on seotud abstraktse andmetüübiga. Abstraktsel andmetüübil ei ole pärilikkuse omadust, objektidel aga on.

UML ja muud objektorienteeritud kirjeldused on tegelikult interaktsioonimasinad või multivoo-interaktsioonimasinad selle asemel, et olla Turingi masinad.

Q-mudel on ka multivoo-interaktsioonimasin.

Süsteemi käitumise määrab ära objektide omavaheline interaktsioon, mitte aga algoritmid, mis iga konkreetse objektiga seotud on.

Q-mudeli korral üritatakse alguses saada esialgu kaudseid hinnanguid protsesside täitmiseks kuluva aja kohta. Füüsilist projekteerimist saab alustada kohe pärast kasutajanõuete kirjeldamist. Ajakitsendusi tuleb täpsustada projekteerimise käigus korduvalt. Vigu avastatakse seeläbi varem, kuid ajakitsendusi on vaja see-eest pidevalt kontrollida.

Enamik olemasolevaid meetodeid ei sobi kõikide reaalajasüsteemides eksisteerivate ajaparameetrite tõestamiseks. Meetodite valik sõltub konkreetsest ülesandest ja sellest, mida on vaja tõestada.

Kui traditsioonilises süsteemis seatakse igale sisendile vastavusse mingi väljund, siis uue paradigma kohaselt siseneb arvutisse mitu sisendmuutujate voogu ja väljundis eksisteerib mitu väljundmuutujate voogu. (ingl. k. stream processing) Nt. mobiiltelefon, mis töötab niikaua, kuni ta on sisse lülitatud; sõltumata sellest, kas telefoni kasutatakse või mitte. Ei saa aga eksisteerida andmebaasi, kuhu suubub pidev sisendandmete voog (andmebaas saab täis).

Aja omadusi saab uurida
- Ajaliste Petri võrgud (Petri võrgud tekkisid 1960. aastatel)
- Ajaloogikatega
- Protsessialgebradega

Q-mudeli korral loobutakse ühtsest ja universaalsest ajast. Q-mudelis eksisteerib käivitushetkede hulk. Igale protsessile saab anda erinevad käivitushetkede hulgad, mis aga eksisteerivad erinevates ajaarvestussüsteemides. Igat komponenti arvutisüsteemis saab vaadata tema isiklikus ajas.

Protsess on Q-mudeli keskne osa, mis peaks kirjeldama algoritmi, selle kogumit; objekti või selle kogumit (agent). Lisades ajafaktori, saab protsesside erinevate täitmiskordade vahel vahet tegema.

Süsteem peab olema pidevalt töövalmis, ta peab ära tundma signaalid, mis on mõeldud temale ja õige signaali korral asuma vastavat protsessi täitma.

dom p – määramispiirkond, mis sisaldab süsteemis liikuvaid andmeid. Antud määramispiirkond võib tekkida mingite teiste protsessida väärtuspiirkondadest. Kõik süsteemi sisenevad andmed on varustatud ajalipikutega, mis näitab, millal need andmed on tekkinud.

Kõikidele väärtustele võib olla antud kehtivusintervall. Tavaliselt määrab selle andmete tarbija. Väärtuste kehtivus sõltub ka konkreetsest kasutusalast (rõhu väärtused protsessi juhtimiseks vs. väärtused logifaili tarbeks). Sobilike andmete filtreerimiseks kasutatakse

kanalifunktsiooni. Ei ole mõistlik võtta kogu info süsteemi sisse, et seejärel seda filtreerima hakata (andmete ülekanne süsteemi võtab aega, sest sidekanalid on aeglased). Üle kanda tuleks vaid andmed, mis on sobiliku vanusega.

Q-mudelis eeldame, et igal protsessil on oma protsessor (ülesande lihtsustamiseks). Mahukaima protsessi korral võib ühe protsessi täitmiseks kasutada mitut protsessorit, milles kõigis töötab üks koopia antud pikast protsessist.

Käivitushetkede hulk
Eeldused

- T(p) elementide hulk peab olema täielikult järjestatud (elemente on lõplik arv). Projekteerimise käigus ei hakata fikseerima, mitu korda üks protsess võib käivituda, kuna see pole oluline. Protsess võib töötada aastaid. Elemendid on sel juhul järjestatud alati kasvavalt. Mitme koopia käivitamisel ühest protsessist tuleb käivitushetked valida nii, et oleks võimalik mõõteriistadega vahet teha, milline koopia millal käivitati. (Sest tegemist on diskreetsete ajahetkedega!)
- Iga täielikult järjestatud hulk omab minimaalset elementi. See element peaks omama väärtust null ja see võiks olla kõigi protsesside käivitushetkede hulgas ühine (nn. süsteemi külmstart, universumi algus jne.).
- Mitte-Zenoni omadus. Igas lõplikus ajaintervallis saab protsessil olla vaid lõplik arv käivitushetki. On vaja garanteerida, et varem alustanud protsessi andmed jõuaksid tarbijani enne kui hiljem alustanud protsessi omad.

Protsesside käivitamisel tuleb arvestada, et protsesside pikkused ei pruugi olla kogu aeg samad. Ühe ja sama protsessi koopiate täitmise kestus võib olla erinev!

T(p) defineerimine

- Tuleb üles lugeda kõik käivitushetked. (See on keeruline lõpmatu arv kordi käivituva protsessi korral).
- Viidata mingi sündmuse tekkimisele, mille esinemine käivitaks antud protsessi.
- Viidata juba defineeritud protsessi käivitushetkede hulgale.

Eeldatakse, et protsessid käivituvad perioodiliselt. Ülesande täitmiseks vajalike protsessi koopiate arv võib ületada riistvara võimalused. Saab projekteerida süsteemi, kus iga üksiku protsessikoopia täitmiseks kuluv aeg on lõplik ja ülevalt tõkestatud. See aitab planeerida riistvara ressursside jaotamist. Piiratud riistvararessursside korral aitab protsessi algoritmi lihtsustamine, mis tagab lühema käivitusaja või käivitushetkede vahelise aja suurendamine.

Protsessi olek
Protsess on kirjeldatud erineva hulga olekutega. Algolekust pääseb lõppolekusse tavaliselt üle mitme vaheoleku. Q-mudelis ei ole protsessi sisemist struktuuri tavaliselt teada, kuna see pole oluline. Kuna Q-mudeli protsess võib oma olemuselt olla multivoo-interkatsioonimasin, ei saa seda alati kirjeldada tavalise olekumudeliga. Kasutatav olekumudel peab suutma arvestada protsessi paralleelsust. Olulised on alg- ja lõppolek. Süsteemi kirjelduse lihtsustuse huvides ei oma vaheolekud erilist tähtsust. Täpsuse vajaduse kasvades on alati võimalik suur protsess jagada alamprotsessideks ja näidata ära ka vaheolekud. Olulised on vaid protsessi olekumuutujad, mis tekivad pärast tema lõppemist. Protsessi väljundväärtus (olekumuutujate väärtus) jääb muutumatuks senikaua, kuni uus koopia antud protsessist seda väärtust muudab.

26. märts 2003

Laborite ajad
E: 14:00 – 17:30
K: 08:00 – 12:30
Maikuust N: 14:00 – 17:30

Q-mudeli kanalid
Operatsioonid, mis teevad ühe protsessi väärtuste piirkonnad kättesaadavaks teistele protsessidele. Toimub ühe protsessi väärtuspiirkonna väärtuste ülekandmine teise protsessi muutumispiirkonda. Kanalite kasutamisel võivad tekkida mäluprobleemid. Kanalifunktsiooni antakse ette kõik väärtused, mis lähteprotsessist tekkinud on. Seega oleks mõistlik ette anda vaid sobilikud väärtused (piirangud), milliseid väärtusi kanalisse antakse. Kanalioperatsiooni tehakse vaid nendel ajahetkedel, mis on määratud käivitushetkede hulgaga. Protsessid võivad aga töötada erinevates aegades, mis teeb keeruliseks andmete vanuse määramise (millised andmed on veel piisavalt värsked ja millised mitte).

Paljud traditsioonilised loogikad (ajaloogika) ei suuda antud andmesidet formaalset kirjeldada ja analüüsida. (Putkast üleeilset ajalehte tooma minnes tuleks nende loogikate alusel iga kord seletada, miks just seda lehte vaja on). Antud andmesidet on võimalik kirja panna teist järku predikaatarvutuse abil. Esimest järku predikaatarvutus seda ei võimalda.

Kanaliga seotud protsessid ei pea töötama semisünkroonselt (ühe programmi lõppedes käivitub järgmine programm või protsess). Protsessid võivad töötada sünkroonselt (mõlemad protsessid alustavad tööd ühel ja samal ajal; paralleeltöötlus). Enamik kanaliga seotud protsesse töötab asünkroonselt (nende käivitushetked on teineteisest sõltumatud). Viimane ei ole tavalises andmetöötluses enamalt jaolt lubatud. Sellist täitmisviisi ei eksisteeri tavapärases arvutiteaduses.

Kanalifunktsioon defineeritakse tarbija poolt. Tarbija ütleb, milliseid andmeid tal vaja on. Tarbija ja tootja ei pea teadma teineteise nõudeid ega nime; kogu suhtlemine toimub läbi kanali, mis moodustab kahe protsessi vahel ülekandeliidese. Kanali kaudu võib transportida lisaks soovitud andmetele ka muud infot. Oluline on, et tarbija saaks seda, mida ta küsib.

Tootja toodab andmeid, mis saadetakse ringpuhvrile, kes kontrollib andmete vastavust soovitule ja saadab sobivad andmed tarbijale edasi (tarbija võib saata kinnituse andmete korrektsuse kohta).

Ringpuhvri omadused
Puhvri pikkus on oluline. See on määratud kanalifunktsiooniga. Puhver on jagatud segmentideks, mida täidetakse järgemööda. Igas järgmises segmendis on järjest vanemad andmed. Kui puhver saab täis, siis vanimad andmed kustutatakse. Tarbija soovib väljavõtet mingi sobiva vanusega olekumuutujate väärtustest (ajalipikud, mis andmetega kaasas on, on tootja ajas). Puhvri pikkus peab olema ühe võrra suurem kui kanalifunktsiooni poolt määratud andmete vanus. Tootjal ja tarbijal on sõltumatud puhvri viidad, millede omavahelise nihke suurus on määratud tarbija poolt soovitud andmete vanusega. Asünkroonse protsessi puhul ei ole aga antud nihe üheselt määratud, see muutub teatava funktsiooni järgi.

Puhvrisse kirjutamise ja sealt lugemise protseduur sõltub kanali tüübist. Kanalis kulub aega side peale kanalis ja puhvri korrastamiseks (viitade muutmine ja puhvri korrektsuse kontroll). Antud hilistumine on transporthilistumine.
Kogu andmevahetuses on viited tingitud sellest, et tarbija ootab andmeid, sõnumi kodeerimisest ja saatmisest ja sünkroniseerimisest.

Nullkanal

Kõik teised kanalid olid punkt-punkt-kanalid (tootja ühest pordist tarbija ühte porti). Nullkanal korraldab mitme protsessi samaaegset käivitamist, andmevahetusega ta ei tegele.

Kui ühes sõlmes olev protsessor avastab, et käivitada on vaja mingi programm, tuleb antud signaal saata kõikides teistes sõlmedes asuvatele protsessoritele. Sellest tekib viide. Lisaks on vaja aega, et protsessor antud toimingu ära jõuaks teha. Antud viide sõltub kasutatavast operatsioonisüsteemist, riistvarast jne. Signaali saabumisel antakse süsteemi katkestus, mis käsib hetkel käimasoleva protsessi peatada. Kui käimasoleva protsessi prioriteet on suurem, ei pruugi protsessi peatumine hetkeliselt toimuda. Lisaks on vaja ära kopeerida registrite sisu, mis peatatava protsessiga seotud olid. Samuti võib olla vajadus laadida käivitatav protsess mällu. Kogu protsess võtab aga aega.

/--/
09. aprill 2003

Tarkvara verifitseerimine
Süsteemi kirjeldus on olemas. Q-mudeli korral saab analüüsida mitmeid ajalisi omadusi. Q-mudel on ainuke, millega saab analüüsida relatsioone (ta on multivoo-interaktsioonimasin).

Viimasel ajal on kodeerimise protsess lihtsustunud (kasutada mooduleid jne.). Mõnedes arenduskeskkondades olemas ka koodigeneraatorid. Kodeerimine ei ole seega enam eriline probleem. Probleemiks on projektide erinevate etappide verifitseerimine ja kontrollimine. Praegusel momendil on peaküsimus spetsifikatsioonide õigsuse kontroll. Natuke pööratakse ka tähelepanu kasutajanõuetes olevate vastuolude avastamisele. (Ei ole otsene verifitseerimine, kuid on seotud formaalse analüüsiga).

Verifitseerida võib kasutades kahte või ühte keelt. Ühes kirjeldatakse süsteem, teises aga omadused, mida on vaja tõestada. Ühekeelse lähenemise puhul tehakse kogu töö ühes keeles.

Uurimisobjektiks on tarkvara koos riistvaraga või tarkvara koostöö mistahes välismaailma füüsikaliste protsessidega. Seega on kasutusel sundparalleelsus. Oluline ei ole algoritmide ega koodi verifitseerimine. Huvitavad algoritmist moodustunud struktuuride omaduste analüüs.

Enamik kaasaegses tehnoloogias ettetulevaid probleeme on põhimõtteliselt arvuti abil lahendatavad. Küsimus on selles, palju on selleks vaja võimsust ja aega. Valides välja algoritmi ja ehitades valmis süsteemi, võib selguda, et seda algoritmi ei saa kasutada (mõttetu töö).

Kahekeelses lähenemises on üheks keeleks arvuti poolt täidetav keel, mis konstrueerib teise keelde minevat teksti. Teine keel on väidete tõestamise keel, mille abil saab kirjeldada neid nõudeid; teise keele abil on võimalik mitmesuguste väidete tõestamine.

Teiseks keeleks võivad olla ajaloogikad või protsessialgebrad. Kakskeelset lähenemist kasutatakse programmikoodide tõestamiseks.

Kakskeele süsteemi tarbeks peab olema
- Arvutusmudel (mis kirjeldab andmevahetuse ja sünkroniseerimise mehhanisme)
- Teine keel, mille abil teostatakse arvutusmudeli poolt väljastatud sündmuste analüüsi. Siin on võimalik formuleerida ka teoreeme.
- Teoreemide tõestamiseks vajalik tõestussüsteem.
- Otsustusprotseduure, mille abil tehakse kindlaks, kas mudeli poolt antud sündmused omavad piisavalt tõendeid selleks, et teoreeme lugeda tõesteks või mitte.

Ühekeelsel lähenemisel ei looda eraldi sündmusi genereerivat mudelit. Programmi tõestamiseks ei pea sel juhul programmi realiseerima. Kasutatakse spetsifikatsioonide, projektide ja kasutajanõuete tõestamiseks. Tõestamine toimub siin empiiriliselt. Spetsifikatsiooni teisendatakse järjest keerulisemaks kuni jõutakse lõpp-projektini; seejuures püütakse säilitada spetsifikatsiooni tõesust.

Milneri arvutus
Eesmärk on saavutada süsteemi kirjeldus selliselt, et nende käitumise üle oleks võimalik matemaatiliselt arutleda jäädes seejuures ühe keele piiridesse. Arutlust rakendatakse nii programmidele kui andmestruktuuridele, hiljem ka riistvarale; välja jäetakse loodusliku maailma nähtused. Oluline ei ole see, kas funktsioonid töötavad süsteemi sees õigesti. Samuti oli ebaoluline ajastamine.

Kõik süsteemi protsessid on omaette objektid; nende omavaheline suhtlemine piirdub omavaheliste sõnumite saatmise ja vastuvõtmisega.

Arvutus baseerub

- Vaatlusel. Keegi jälgib töötavat süsteemi, et oleks võimalik otsustada, millist käitumist on võimeline väline vaatleja nägema. Iga väline vaatleja võib eksida selle interpreteerimisel, mida ta näeb.
- Sünkroniseeritud tegevustel. Iga paralleelsüsteemi osa on ehitatud sõltumatutest arvutusagentidest, mis vahetavad andmeid. Agent on objekt, mis täidab mingit ülesannet. Tal endal puudub otsustusõigus. Agent sarnaneb meetoditele objektorienteeritud programmeerimises. Ainult proaktiivsetel agentidel on selline valikuvabadus.

Süsteemi ehitamine seisneb agentide omavahelises ühendamises (s.t. panna nad omavahel infot vahetama). Nad peaksid olema võimelised ka iseseisvalt teineteist avastama (Plug&Play). Need on proaktiivsed agendid. Milneri arvutuse juures seda ette nähtud ei ole, kuigi kogu teooria kehtib ka proaktiivsete agentide juures.

Süsteemi ei ole võimalik jälgida, kui temaga ei olda interaktsioonis. Igasugune vaatamine ja jälgimine on võimalik vaid jälgitavate protsesside mõjutamise teel.

Võrreldes Petri üldise võrguteooriaga on tema arvutustes välistatud samaaegselt toimuvate sündmuste vaheline andmeside. Seda isegi juhul, kui üks protsess lõpetab märgatavalt varem kui teine. Kuigi ei keelata otseselt ära paralleelseid relatsioone, kuigi neid tegelikkuses ei kasutata. Paralleelsuse tunnuseks on agentide sõltumatu töötamise võimalus teineteisest. Kuna Milner tegeleb ühe vaatlejaga, siis ei ole võimalik rakendada paralleeltöötlust. Üks vaatleja ei saa samaaegselt infot vahetada kahe agendiga korraga. Kuna infovahetus kahe agendiga ei saa toimuda ühel ajal, siis järelikult ei ole võimalik väita, et agendid töötasid samaaegselt. Paralleelsus oleks võimalik juhul, kui eksisteeriks sama palju abivaatlejaid kui on agente. Abivaatlejad registreeriksid kõik agentide käivitused, mille abil oleks hiljem võimalik samaaegsust analüüsida. NB! Milneri arvutuses aega ei eksisteeri, seega ei ole võimalik ajalipikuid vmt. kasutada. Kogu süsteemis on vaid üks aeg – välise vaatleja aeg.

Süsteemi arendamisel tuleks jälgida tema evolutsiooni. Oleks vaja tõestada, et kõik tema varasemad omadused on säilinud arengu käigus. Milneri arvutus võimaldab seda.

Püüd on fikseerida teoreemid samas keeles, milles on kirjeldatud ka Q-mudel.
Esimesel tasemel näidatakse ära, et süsteemi ajaline käitumine on õige; teisel tasemel hakatakse süsteemi kirjeldust täpsustama ja muutma.

Verifitseerimise ja katsetamise vahel valitakse enamikul juhtudel alati katsetamine.
Verifitseerimine toimub etapiliselt:

- Vaatleme üksikuid elemente, kanaleid ja nende protsesse.
- Vaatleme kahe protsessi vahelisi interaktsioone.
- Protsesside grupi käitumise analüüsimine.

Protsesse iseloomustab:

- Protsessi käivitusaeg. Kui käivitusaeg on liiga pikk, tuleb protsessi lihtsustada või hankida täiendavat riistvara.
- Andmete tarbimine. Enne protsessi töö lõpetamist on vaja teiste protsesside käest andmed kätte saada.
- Kanalifunktsioon. Aeg on kanalifunktsioonis vastassuunaline. (Andmete vanus kasvab ajas tagasi liikudes.)
- Igal protsessil peab olema oma isiklik ajahetkede ja käivitushetkede hulk. See võib olla realiseeritud ka sünkroonkanali kaudu oleva viida abil mõnele teisele käivitushetkede hulgale.

Kõik protsessid võivad olla korduvkäivitatavad. Käivitushetked fikseeritakse esimese käivituse ja keskmise perioodi määramisega. Juhusliku perioodi korral antakse ette ka lubatud tolerants.

Tavalises programmeerimises ei pöörata tähelepanu töötavate protsesside koopiate arvule. Tavaliselt ei eksisteerigi üle ühe koopia. Reaalajasüsteemis on käivitajaks aga välismaailm, mistõttu võib tekkida olukord, kus korraga on töös ühe protsessi mitu koopiat. Perioodiliselt käivitatav protsess ei pruugi oma tööd lõpetada enne uue käivituskutsungi ilmumist. (Tingitud sundparalleelsusest, sest välismaailm dikteerib olukorra). Koopiate arvu võib vähendada kas protsessi töö lühendamisega või riistvara laiendamisega.

Varem alustanud koopia peaks ka varem lõpetama, mis ei pruugi alati nii olla. See on tingitud protsessi töötamise aja varieeruvusest.

16. aprill 2003

Välistada tuleb see, et hiljem alustanud protsess lõpetab enne kui varem alustanud protsessikoopia.

(t2+ksii(p1*t2))-(t1+(ksii(p1*t1)) >= ksii(t2+alfa(p))-(t1+beeta(p)) >= tmin(p)-(beeta(p)-alfa(p)) > 0
Siit: tmin > beeta(p)-alfa(p)

Protsessidevaheline interaktsioon

- Kahe pordi vahel võib olla ainult üks kanal.
- Igal protsessil võib olla ainult üks sisendport, kuhu siseneb poolsünkroonne kanal. Ühte porti võib suunduda ka mitu poolsünkroonset kanalit (sel juhul saab protsessi käivitada mitme erineva protsessi poolt). Ühe pordi korral on võimalik summeerimise teel välja arvutada käivitushetkede hulgad. Mitme pordi korral seda teha ei saa.

  Käivitamisel loetakse sisendportidest kanalites olevaid andmeid. Petri kanalis on andmed vaid juhul, kui teda käivitatakse. Kui oleks lubatud kahte erinevasse porti suubuvad petri kanalid, siis käivituse korral teistes petri kanalites andmeid ei ole. See on aga määramata olukord, mida ei tohi tekkida.
- Kui kaks käivitust satuvad ekvivalentsintervalli sisse, siis loetakse nad üheaegseteks ja käivitatakse vaid üks protsess. Kui aga käivitushetked ei lange ekvivalentsiintervalli sisse, siis käivitatakse protsessist mitu koopiat.
- Sünkroonkanalite kaudu seotud protsessidest, mis moodustavad sünkroonkobara, võib vaid üks protsessidest omada käivitushulkade vektorit.

Kahekaupa koos töötavad protsessid
Probleemid selliste protsesside korral sõltuvad kanalitüübist.

Sünkroonkanalite korral:

- Ühendus ei tohi mõjutada teise protsessi ajakitsenduste täitmist.
- Sünkroonkanali ühendus ei mõjuta tarbijaprotsessi ajakitsendusi, kui tootja protsess on lõpetanud enne, kui algab andmete tarbimine. Sel juhul on garanteeritud alati kõige värskemate andmete tarbimine.
- Protsess võib tarbida ka iseenda andmeid vanematest käivitustest. Sellise tarbimise korral ei tohi lubada pisematki ootamist omaenese andmete taga. Vastasel korral kasvaks tekkinud hilinemine piiramatult.

Petri kanalite korral:
Tootja protsess käivitab oma lõpetamisega tarbijaprotsessid.
Probleem: Kas kõik tootja protsesside käivitussignaalid viivad tõepoolest tarbijaprotsessi käivitamiseni, või ignoreeritakse mõningaid signaale mingil põhjusel?

Olgu situatsioon, kus ühte protsessi saab käivitada mitu erinevat teist protsessi (tarbija porti suubub mitu petri kanalit). Tarbijaprotsessi käivitamise tagab signaal, mis saabub esimesena ühest neist kanaleis ja ei jää samas tarbijaprotsessi ekvivalentsiintervalli sisse. Ülejäänud signaalid aga jäetakse kõrvale. Sisendandmed nende kanalite peal on tavaliselt samaväärse iseloomuga, kuid erinevad võib-olla oma väärtuste poolest. (Nt. temperatuuri mõõtmine mitme erineva anduriga)

Asünkroonsed kanalid
Tarbija- ja tootjaprotsess käivitatakse üksteisest sõltumatult. Tarbija tööks on sellegipoolest vajalik tootja informatsioon. Sellist lähenemist ei kasutata tavalises programmeerimises. Andmete hilistumine on kahe protsessi andmevahetuse korral erinev (mis ei ole seotud andmete transpordiga). See on seotud sellest, et tarbija ja tootja protsesside käivitushetked „ujuvad" teineteise suhtes.

Protsessidegrupi käitumine
Sünkroonsed kobarad
Üheks eesmärgiks informatsioonitupikute avastamine (üks protsess peab ootama, kuni teine protsess on töö lõpetanud; tööd ei saa jätkata, kuna puuduvad andmed või need andmed on vananenud). Tavalises programmeerimises kasutatakse sel juhul vanemaid andmeid. Süsteemides, mis on aga ajatundlikud, seda teha ei tohi.

Tupikud võivad tekkida ainult sünkroonsetes kobarates (ühelgi teisel juhul ei käivitu protsess enne, kui andmed pole saadaval). Kobaratest tuleb välja selekteerida lihtprotsesside jadad, mis on omavahel ühendatud sünkroonsete kanalitega. Jadad võivad olla ka sellised, kus jada esimene ja viimane element langevad kokku (silmused). Sellised silmused ongi tupikute tekkimise eelduseks.

Probleemid, mis tekivad protsessikobarate kasutamisel:
- Kui suuri ekvivalentsiintervalle kobarates kasutada. (Protsesside käivitamine üle võrgu üheaegselt – vajalik lõpmata täpne kellade sünkroniseerimine.) Küsimus ei ole, kas sünkroniseerida vaid selles, kui täpselt sünkroniseerida. (Defineerib süsteemi nõrgad nõuded.)
- Milline peab kobarates olema samaaegsusintervall. (Defineerib süsteemi ranged nõuded.)

Poolsünkroonsed kobarad
Ei erine oma olemuselt sünkroonsest kobarast.