Esimene diskett

Sissejuhatus reaalajatarkvaratehnikasse LAP 5711

Abimaterjal loengute koduseks analüüsiks.
koostas L.Mõtus

Peamised raamatud:

1. H.Kopetz “Real-time systems; Design principles for distributed embedded applications”, Kluwer Academic Publishers, Boston/Dordrecht/London, 1997, 338 pp., ISBN 0-7923-9894-7
2. P.T.Ward, S.J.Mellor “Structured development for Real-time Systems” vol.1, Prentice-Hall, 1985, 156 pp, ISBN 0-13-854787-4 025
3. S.R. Schach “Classical and Object-Oriented Software Engineering”, IRWIN (a Times Mirror Higher Education Group, Inc), 1996, 603, ISBN 0-256-18298-1
4. J.Rumbaugh, et al “Object-oriented modeling and design”, Prentice-Hall, 1991, 500 pp., ISBN 0-13-629841-9
5. I.Sommerville “Software Engineering”, Addison-Wesley Publishing Company, 1992, 649 pp., ISBN 0-201-56529-3
6. A.Behforooz and F.J.Hudson “ Software Engineering Fundamentals”, Oxford University Press, 1996, 661 pp., ISBN 0-19-510539-7
7. L.Motus, M.G.Rodd “ Timing analysis of real-time software”, Elsevier Science Publishing/Pergamon, Oxford, 1994, 212 pp., ISBN 0 08 0420265 (hardcover), ISBN 0 08 0420257 (flexicover)

Viited täiendavatele allikatele on antud tekstis.

Õppematerjaliga saab tutvuda URL aadressil
http://www.dcc.ttu.ee/Automaatika/LAP/LAP5711/,
samas on kopeeritavad WORD.7 failid.

Viimati täiendatud

August 1999

Üldinfo loengukursuse kohta

Eesmärk: Selgitada reaalajasüsteemide erinevust traditsioonilistest andme- ja infotöötlus-süsteemidest. Anda lühiülevaade tarkvaratehnika alustest – projekti juhtimine, riski analüüs, elutsüklid. Selgitada reaalajatarkvara loomisel tekkivate probleemide ja traditsioonilise tarkvaratehnika meetodite võimaluste mittetäielikku vastavust. Peamised vaadeldavad tarkvaratehnika tööriistad baseeruvad poolformaalsete -- struktuuranalüüsi mudelid ja objekt-orienteeritud mudelid -- tarkvara projekteerimise meetodite näitel. Praktilised harjutused arvutiklassis toimuvad UML-ga ühilduvas Artisan Real-time Studio tarkvaratehnika keskkonnas. Loengusarja kokkuvõtvas osas juhitakse tudengite tähelepanu käsitlemata jäänud tarkvara projekteerimis- ja realiseerimisprotsessi olulistele aspektidele, millel võivad olla saatuslikud tagajärjed reaalajatarkvara töövõimele. Käsitlemata jäänud omadused vajavad spetsiaalseid teooriaid ja tööriistu, mida tutvustavad järgnevad kursused (LAP 5712 Tarkvara dünaamika, LAP 5713 Reaalajasüsteemid (projekt), LAP 8714 Tarkvaratehnika keskkonnad).

Eeltingimused: Tutvus informaatika ja tarkvaratehnika põhitõdedega. Arvutikasutamisoskus normaalsel tasemel. Vähemalt kaks aastat ülikooli üld- ja alusõpet (soovitavalt informatsioonitehnika valdkonnas). Algteadmised süsteemitehnikast.

Orienteeruv ajakulu loengusarja läbimiseks:

Loengud toimuvad sügissemestril, loenguid on 3 tundi nädalas ja juhendajaga harjutustunde on 1 tund nädalas (kokku 16 nädalat). Lisaks loengutele ja juhendajaga harjutustele on vajalikud iseseisvad harjutused tarkvarakeskkonna kasutamise kogemuste saamiseks (keskmiselt 10 tundi) ja iseseisev töö kodus/laboris (keskmiselt 20 tundi) koduülesannete lahendamiseks.

Eksamiks valmistumine võtab täiendavalt 30 tundi, et saada positiivset (hindest 1 ehk “nõrgast” paremat) tulemust. Eksam on kirjalik, vastata tuleb 10-le küsimusele, kõikvõimalikke mittebioloogilisi abivahendeid võib eksamil kasutada. Eksamiküsimustele vastamise aja ülemine piir on 120 minutit. Eksamitulemused saab teada orienteeruvalt 10 päeva jooksul alates eksamile järgnevast päevast. Kolme päeva jooksul (tavaliselt), alates eksamitulemuste ilmumisest teadetetahvlile võib eksamihinde protestida, s.t. tulla suulisele vestlusele kui suudate põhjendada hinde mittevastavust eksamitöös kirjutatule.

Eksamile pääsemise tingimused:

1. Edukalt tehtud kontrolltöö reaalajasüsteemide ja reaalajatarkvara põhialuste kohta. (oktoobri kolmas nädal)
3. Edukalt tehtud kontrolltöö tarkvaraprojektide juhtimise kohta (novembri kolmas nädal).
2. Edukalt valminud kodutöö tarkvaratehnika keskkonnas ARTISAN Real-time Studio ja kodutöö esitlus (detsembri esimene-teine nädal)

Kui kodutöö ja kontrolltööd on sooritatud hindele "väga hea", on võimalik saada "väga hea" eksamihinne kodutööde ja kontrolltööde eest, ilma eksamit sooritamata.

Kes ei ole positiivse hindega läbinud vähemalt ühte kontrolltööd või kodutööd, ei pääse eksamile.

Loengusarja sisu ja üksikosade orienteeruv ajaline kestvus:

1. Süsteemitehnika ja tarkvaratehnika (sissejuhatus)
2. Reaalajasüsteemid (3 nädalat)
3. Tarkvaratehnika põhitõed (3 nädalat)
4. Spetsifitseerimine ja projekteerimine (5 nädalat)
    - struktuursel analüüsil baseeruv metoodika
    - objekt-orienteeritud metoodika
    - koduülesanded (lahendatakse kodus, avalik ettekanne ja arutelu laboris)
4. Arvutiteaduse ja tarkvaratehnika vahekorrast (1 nädal)
5. Probleemid, mille lahendust ei toeta traditsioonilised tarkvaratehnika tööriistad (1 nädal)

Sisukord

I osa
Süsteemi- ja tarkvaratehnika

1.1 Tarkvaratehnika üldine kontekst 6
1.2 Süsteemitehnika – süsteemide loomise meetodid ja vahendid 7
1.3 Süsteem 8
1.3.1 Süsteemile iseloomulikud omadused 9
1.4 Tarkvarale erinevaid nõudeid esitavad süsteemide klassid 10
1.5 Eelülevaade tarkvarasüsteemide loomise protsessist 11
1.6 Loengukursuse pragmaatiline eesmärk ja selle saavutamiseks kasutatud teed 13
1.7 Esimese osa kordamisküsimused 15
1.8 Esimeses osas kasutatud kirjanduse viited 15

II osa
Sissejuhatus reaalajasüsteemidesse

2.1 Reaalajasüsteemid 16
2.2 Reaalajas töötav arvutisüsteem 17
2.3 Funktsionaalsed nõuded reaalajasüsteemile 20
2.3.1 Andmehõive 21
2.3.2 Juhtimine 24
2.3.3 Inimliides 26
2.4 Alarmiseire ja eriolukordade töötlus 27
2.5 Ajakitsendused reaalajasüsteemis 29
2.5.1 Näide ajakitsenduste mõnedest allikatest 30
2.6 Mittefunktsionaalsed nõuded reaalajasüsteemidele 33
2.6.1 Töökindlus 34
2.6.2 Ohutus 34
2.6.3 Hooldatavus 35
2.6.4 Valmisolek 36
2.6.5 Turvalisus 36
2.7 Reaalajasüsteemide klassifikatsioon 37
2.7.1 Ranges või nõrgas reaalajas töötavad süsteemid 37
2.7.2 Ohutult riknevad või töövõimet säilitavalt riknevad süsteemid 40
2.7.3 Garanteeritud ajakitsendustega või parimate võimalike ajakitsendustega süsteemid 40
2.7.4 Piisavate ressurssidega või mittepiisavate ressurssidega süsteemid 41
2.7.5 Sündmuste poolt või aja poolt juhitud süsteemid 41
2.8 Reaalajasüsteemide levik ja maksumuse hindamine 42
2.8.1 Sardsüsteemid kitsamas mõttes 43
2.8.2 Protsessijuhtimissüsteemid 45
2.8.3 Multimeedia süsteemid 46

2.9 Teise osa kordamisküsimused 46
2.10 Teises osas kasutatud kirjanduse viited 47
2.11 Teise osa terminite selgitav sõnastik 48

III osa
Tarkvara projektide planeerimine ja haldamine

3.1 Meenutusi tarkvaratehnikast 50
3.2 Süsteemi (ja tarkvara) loomise üldine kirjeldus 52
3.2.1 Suurte süsteemide loomisel sageli esinevad probleemid 52
3.2.2 Süsteemi analüüs ja projekteerimine 55
3.2.3 Mudeli ja tegelikkuse vahekord tarkvaratootes 56
3.3 Süsteemi loomise planeerimine 57
3.3.1 Kommentaarid süsteemi loomise plaanile 59
3.3.2 Projekti jagamine töödeks 59
3.3.3 Ressursside vajadus 62
3.3.4 Projekti perioodilised ülevaatused 64
3.3.5 Projekti muudatuste haldamine 66
3.3.6 Tarkvara loomise plaan 66
3.4 Süsteemi loomise elutsükkel 67
3.4.1 Tarkvara elutsükkel 68
3.4.2 Tarkvara elutsükli mudelid 71
3.4.2.1 Üldine kaskaadmudel 72
3.4.2.2 DOD mudel 74
3.4.2.3 NASA mudel ja kokkuvõte kaskaadmudelitest 75
3.4.2.4 Spiraalne mudel 76
3.5 Tarkvara protsessi kirjeldamise mudelid 78
3.6 Tarkvaraprotsessi puudutavad kordamisküsimused 80
3.7 Tarkvaratehnikas kasutatavad tööriistad 81
3.7.1 Üldised abivahendid 83
3.7.1.1 Andmesõnastik 83
3.7.1.2 Otsustamine ja kompromisside tegemine 85
3.7.1.3 Stsenaariumid 86
3.7.2 Praktikas enamlevinud mudelid tarkvara töö ja/või struktuuri kirjeldamiseks 87
3.7.3 Kordamisküsimused 90
3.8 Tarkvara meetrika 90
3.8.1 Näiteid tarkvaratoote kvaliteedimeetrikas kasutatavatest mõistetest 90
3.8.2 Tarkvara toodet iseloomustavad parameetrid 91
3.8.3 Tarkvara projekteerimist ja realiseerimist kirjeldavad koefitsiendid 92
3.9 Tarkvaratootega seotud riski haldamine 93
3.9.1 Tüüpilised riskifaktorid tarkvaraprojektides 93
3.9.2 Riski mudel 96
3.9.3 Riski haldamine 98
3.10 Tarkvara meetrika ja riski haldamise kordamisküsimused 99
3.11 Kolmandas osas kasutatud kirjanduse loetelu 99

I osa

Süsteemi- ja tarkvaratehnika

Programmeerimine kui suvalise info töötlemine (ilma, et oleks tingimata vaja tungida töödeldava info semantikasse) ja tarkvara kui iseseisev ning omaette, muust maailmast sõltumatut väärtust omav objekt (toode) on traditsioonilise arvutiteaduse pooldajate ("true believers") maailmavaate alus. Selline maailmavaade on olnud pikka aega viljakas, ning paljud olulised tarkvara ja arvutiteaduse aluseks olevad teoreetilised tõed ja/või uskumused tulenevad otseselt taolisest maailmavaatest, või on saavutatud sellise maailmavaate sihikindla rakendamise tulemusena.

Näiteks väide, et programm on mudeli mudel (Meyer, 1996) on arvutiteaduse jaoks oluline lähtepunkt. On ju klassikaline programmeerimise tsükkel järgmine – võtame reaalse maailma nähtuse, teeme selle nähtuse kohta matemaatilise mudeli (või teooria), ja realiseerime mudeli (teooria) arvutil – s.t. teeme programmi, mis modelleerib arvutil mingit olemasolevat matemaatilist mudelit (formaalset teooriat). Vaikimisi tehtud eeldus on, et programm (mudeli mudel) *ei ole vahetult seotud* kogu tegevuse aluseks olnud reaalse maailma nähtusega. Teiste sõnadega, programm ei hangi ise oma lähteandmeid tegelikust maailmast, ei hakka omal initsiatiivil tööle ja tema tulemused ei pääse vahetult reaalse maailma nähtust mõjutama.

Esimene mõra ülalkirjeldatud elevandiluutorni löödi 1957 aastal, kui arvutiprogrammi sisendid ja väljundid ühendati reaalse maailmaga. See juhtus USA-s Monsanto keemiatehases, kus tehnoloogilise protsessi juhtimiseks kasutati arvutit. Filosoofiliselt võttes oli tegemist enneolematu sündmusega – mudeli mudel muutus äkki reaalse maailma osaks, mistõttu reaalse maailma omadused muutusid märgatavalt.

1.1 Tarkvaratehnika üldine kontekst

Tarkvaratehnika on süsteemitehnika konkretiseering tarkvarasüsteemide (s.t. tarkvara-toodete tegemise) valdkonda. Tarkvaratehnika on tugevalt seotud arvutiteadusega, tema üheks oluliseks erinevuseks arvutiteadusest võibki nimetada suhtumist reaalsesse maailma. Arvutiteadus tegeleb infotöötluse seaduspärasuste ja võimalustega, olenemata töödeldava info semantikast, päritolust, ja töötlusel saadud tulemuste kasutusmeetodist. Lähteinfo ja tulemuste õigsus on määratud matemaatilise (või mõne muu formaalse) mudeliga, mis täidab arvutiteaduses reaalse maailma rolli. Teise erinevusena võib nimetada tarkvaratehnika huvi tarkvara loomise protsessi administratiivsele ja tehnilisele korraldamisele ning juhtimisele.

Tarkvaratehnika on seega märksa pragmaatilisem kui arvutiteadus ja peab ideaalis oluliseks, et töötluseks kasutatav lähteinfo vastaks võimalikult täpselt tegelikkusele, ning töötluse tulemus oleks hästi vastav tegelikkuses vajaminevale – tarkvaratehnika

toode valideeritakse näidates tema vastavust reaalse maailma vajadustele, mitte reaalse maailma mudelile.

Tarkvara on enamuse kaasaegsete süsteemide peamine komponent. Seetõttu on oluline, et tarkvara arendamist vaadeldaks kui ühte osa süsteemi arendamisest ja/või loomisest. Tarkvaraprojekti lähteülesanne tuleneb vahetult süsteemile esitatavatest nõuetest, tarkvaraprojekti lahendus sõltub oluliselt kogu süsteemile ja süsteemi arendamisele seatud kitsendustest ning konkreetsetest nõuetest. Taoline lähenemine iseloomustab igasuguse tarkvaratoote tegemist ja muutub eriti oluliseks reaalajasüsteemide loomisel – tänu paljudele spetsiifilistele nõuetele, mis on eluliselt tähtsad reaalajasüsteemide normaalseks funktsioneerimiseks.

1.2 Süsteemitehnika – süsteemide loomise meetodid ja vahendid

Süsteemitehnika ingliskeelne vaste on *Systems Engineering*. Ingliskeelne selgitus sõnadele "engineering" ja "to engineer" on Webster's New World Dictionary kohaselt:

"Engineering" *-- "the art or science of making practical application of knowledge of pure science, such as physics, chemistry, biology, etc";*
"to engineer" *– " to arrange, manage, or carry through by skillful or artful contrivance";*
"contrivance" *– "ingenious plan, or mechanical device".*

Samaväärsed eestikeelsed vasted neile sõnadele puuduvad, mistõttu tuleks aeg-ajalt üle lugeda ja meenutada ingliskeelseid definitsioone. Kõige lähedasem vaste sõnale "engineering" on keelemeeste poolt soovitatud **"tehnokäsitlus"**, mis mõnes kontekstis annab rahuldava tulemuse.

Süsteemitehnika on tehnokäsitluse rakendamine süsteemide projekteerimiseks ja loomiseks. Võtmeisik süsteemi tegemisel on süsteemi-insener (ärirakendustes sageli nimetatud ka süsteemi-analüütikuks). Süsteemi-inseneri **tehniliste tegevuste** konspektiivne loetelu on (Behforooz & Hudson, 1996) tõlgenduses järgmine:

- Probleemi põhjalik analüüs, mis võimaldab selgitada süsteemile esitatavad nõuded;
- Süsteemi struktuuri (arhitektuuri) ja abstraktse projekti (conceptual design) väljatöötamine, lähtudes kindlalt fikseeritud nõuetest;
- Süsteemi piiride ja süsteemile seatud kitsenduste spetsifitseerimine, süsteemi sisendite, väljundite ja liideste detailsete kirjelduste koostamine;
- Süsteemi oodatava käitumise (eeldatavate funktsioonide) loetelu ja süsteemi käitumise (ning jõudluse)hindamise aluseks olevate parameetrite defineerimine;
- Süsteemi käitumise hinnangute vahemiku (optimaalsest kuni lubamatuni) fikseerimine; nende hinnangute alusel töötatakse hiljem välja spetsifikatsioonid ja jõudlusmõõdud alamsüsteemide projekteerijatele (ka tarkvara projektile); sageli ei ole optimaalne käitumine reaalselt saavutatav, kuid optimaalse käitumise teadmine aitab oluliselt kaasa mõistlike projekteerimisotsuste tegemisele;

- Süsteemi detailse sisemise struktuuri defineerimine; eraldi tuleb nimetada osad, mida tohib projekteerimise käigus muuta; defineeritakse muutujad, millede väärtused sõltuvad süsteemi tegevusest ja need, millede väärtused võivad muutuda sõltumatult süsteemi toimimisest;
- Matemaatiliste mudelite koostamine, mille abil on võimalik süsteemi kätumise kohta saada objektiivseid hinnanguid; neid mudeleid kasutatakse projekti alternatiivsete variantide võrdlemiseks, funktsionaalsete ja mittefunktsionaalsete nõuete rahuldamisele viiva projekteerimise strateegia leidmiseks;
- Kindlalt fikseeritud ja hindamiskriteeriumite valik ja nende alusel objektiivsete tehniliste otsuste tegemine, võrreldakse kõiki elujõulisi alternatiivseid variante;
- Süsteemi jagamine (dekomponeerimine) loogilisteks, hõlpsasti integreeritavateks komponentideks (alamsüsteemideks);
- Kõikide süsteemi arendamisega, ehitamisega, testimisega, integreerimisega, verifitseerimisega, tellijale üleandmisega ja installeerimisega seotud tegevuste kontroll, jälgimine ja hindamine;
- Matemaatiliste ja simulatsioonimudelite täiendamine, kasutades tekkivalt süsteemilt (tema komponentidelt) saadud tegelikke mõõtmistulemusi eesmärgiga parandada kogu töörühma arusaamist tehtavast süsteemist.

Lisaks tehnilisele tööle on paljudel süsteemi-inseneridel täita ka administratiivne roll seoses projekti planeerimise ja täitmise erinevate aspektidega. Nendest tegevusest tuleb lähemalt juttu tarkvara projekti juhtimise juures (vt. käesoleva teksti III osa).

Igasugune projektijuhtimise ja toote arendusega seotud tegevus peaks lähtuma järgmisest kolmest põhimõttest:

1. Liideste kirjelduse detailsus ja täpsus määravad süsteemi ja tema osade piirid, samuti kitsendused süsteemile ja/või tema osadele, ning ka vastuvõtu kriteeriumid. Võimaluse korral tuleks teadaolevad liideste kirjeldused, kitsendused ja vastuvõtu kriteeriumid fikseerida tellijaga sõlmitavas lepingus.
2. Süsteemi loomise protsessi edukus sõltub oluliselt projekteerijate ja realiseerijate otsuste objektiivsusest. Maksimaalselt tuleb kasutada objektiivseid variantide võrdlusi, lähtudes mudelitel ja prototüüpidel tehtud katsetuste tulemustest, ning kasutades varem kokkulepitud jõudlushinnangute ja otsuste tegemise kriteeriume.
3. Süsteemi tükeldamise kriteeriumid peavad tagama maksimaalselt sõltumatute, kuid koostöötavate komponentide tekkimise. Need komponendid peavad olema süsteemi integreerimisel, testimisel ja kasutamisel kerge vaevaga muudetavad ja asendatavad. (Osadeks jagamise kohta vaata käesoleva teksti IV osa – arvutiteaduse ja tarkvaratehnika vahelised suhted).

1.3 Süsteem

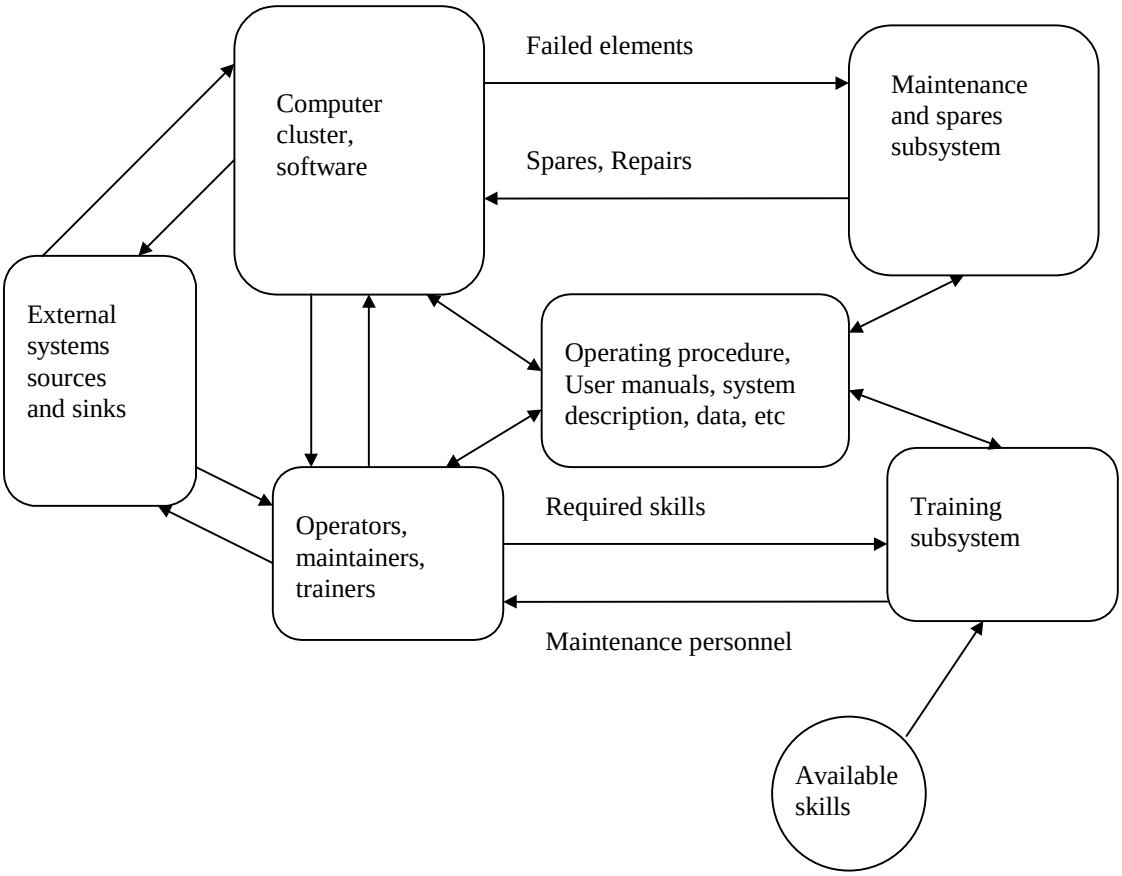
Süsteemi määratlus. Edaspidises käsitluses on süsteem kogum omavahel ühendatud, interaktsioonis olevaid alamsüsteeme. Süsteemi struktuur ja omadused peavad garanteerima funktsioneerimise ja reaktsiooni süsteemivälistele mõjudele, nii et realiseeruks süsteemile seatud eesmärkide täitmine, ja oleks rahuldatud kõik ümbritseva keskkonna poolt esitatud kitsendused.

Toodud definitsioon katab süsteemid, mis juba toimivad, või mis tulevad luua tööks bioloogilistes, meditsiinilistes, tööstuslikes, finants, juriidilistes, halduslikes ja muudes rakendustes. (vt. joonis 1.1)

1.3.1 Süsteemidele iseloomulikud omadused

Sünergism – üksikute alamsüsteemide omaduste ja funktsioonide kombineerimisel saadakse teenuse uus kvaliteet, mida alamsüsteemid eraldi töötades ei suuda tagada. Näiteks transpordisüsteem, mis viib isiku ja tema pagasi punktist A punkti B. Transpordisüsteemi alamsüsteemideks võivad olla maapinnal liikuvad sõidukid, veesõidukid, õhusõidukid, maanteede võrk, lennuväljad, lennuväljade juhtimissüsteemid, bensiinijaamad jne. Sünergism tekib alamsüsteemide sobivalt valitud seoste ja interaksioonide tõttu.

Adaptiivsus -- süsteemide võime dünaamiliselt, ilma funktsioneerimist (s.t. teenuse osutamist) katkestamata, kohaneda muutunud nõudmistega ja/või väliskeskkonna seisundiga. Paljud süsteemid peavad funktsioneerima 20-30 aastat, mille jooksul võib oluliselt muutuda tehnoloogia, ümbritsev keskkond ja kasutaja poolt neile esitatud nõudmised. Mida suurem ja üldisem on süsteem, seda suurem osa tema ressursidest tegeleb adapteerumisega ja igapäevase hooldusega. Mida spetsialiseeritum on süsteem, seda halvemini ta adapteerub, kuid seda ökonoomsemalt ta töötab.



Joonis 1.1 Süsteemi definitsioon (© Beforooz & Hudson,1996)

Kompromiss – seisneb vastuvõetava taskaalu leidmises süsteemile esitatavate konfliktsete nõuete ja soovide vahel. Süsteem ei saa alati olla optimaalne iga üksiku alamsüsteemi (või kasutaja) mõttes, küll aga tuleb püüda teha süsteemi kõikide nõudmiste suhtes keskmiselt võimalikult hästi funktsioneerivaks.

1.4 Tarkvarale erinevaid nõudeid esitavad süsteemide klassid

Nagu igasugune liigitus, on ka järgnev pisut subjektiivne. Liigituse aluseks on võetud kasutajate poolt esitatavad kvalitatiivselt erinevad eeldused, soovid ja vajadused, mis mõjutavad tarkvara nõuete spetsifitseerimisel, tarkvara projekteerimisel, realiseerimisel, katsetustel ja modifitseerimisel kasutatavate metoodikate ja tööriistade valikut, verifitseeritavate omaduste valikut ja kasutatavaid verifitseerimismeetodeid, aga ka projekti realiseerimiseks vajalike tööde keerukust ja mahtu.

Andmetöötlus- ja infotöötlussüsteemid. Andmetöötlus tähendab tavaliselt numbrilise informatsiooni töötlust, infotöötlus võib sisaldada ka sümbolkujul ja/või piltkujul antud informatsiooni töötluse. Arvuti kasutaja on inimene, arvutil puudub vahetu ühendus andmeid ja muud infot tootva ja/või tarbiva allikaga. Süsteemi töö tulemusi tarbib samuti inimene – näiteks valemite järgi arvutused, differentsiaalvõrrandite lahendamine, ehituskonstruktsioonide arvutused, tabelarvutused, palgaarvestus-süsteemid, lihtsad laoarvestussüsteemid, raamatupidamisprogrammid. Andmetöötlus on ajalooliselt kõige vanem ja kõige laiem arvuti kasutuse valdkond. Andmetöötlussüsteemid on hästi dekomponeeritavad, iga osa on põhimõtteliselt ka teistest osadest eraldi lahendatav. Andmetöötlusvaldkond on hästi läbi uuritud, valdkonnasisesed põhjuslikud seosed on teada (algoritmiliselt määratud), juhuslikkust (välja arvatud riistvara ja progranmmi vigadest tingitud juhuslikkus) ei esine, eriolukordade töötlus on suhteliselt triviaalne. *Andmetöötlussüsteemid on pikka aega olnud arvutiteaduse (computer science) katsepolügoon* – põhjuseks on, tänu täielikult determineeritud katsetingimustele, lihtsalt korratavad eksperimendid. Ainsad nähtavad raskuste allikad on seoses lähteandmete sisestamise vigadega, või mitteasjakohaste lähteandmete kasutamisega. Väga harva tekitavad probleeme ka tegelikkusele mittevastavad (või osaliselt vastavad) algoritmid – aga see ei ole arvutiteaduse probleem. Tegelikkuse ja algoritmide (mudelite) vastavus on oluliselt suuremal määral tarkvaratehnika probleem – lähteandmete ning tulemuste valideerimine ja verifitseerimine.

Modelleerimise, eksperimendi juhtimise ja sellega seotud tulemuste töötlemise süsteemid. Selle süsteemide grupiga tegelevad nn. modelleerijad (computational scientists, arvutusteadlased), kes kasutavad arvutit füüsikalise või matemaatilise mudeli asemel valdkondades, kus eksperimendid tegelikul objektil ei ole eetilistel või majanduslikel põhjustel aktsepteeritavad. Näiteks universumi mudelite loomine, termotuumareaktsiooni kulgemise ja protsessi omaduste uurimine (ilma reaktsiooni – s.t. plahvatust tekitamata -- või ilma juhitavat reaktsiooni tekitava katseseadmeta), materjaliteaduses uue materjali struktuuri projekteerimine ja selle omaduste uurimine, geenitehnoloogias geenide interaktsiooni uurimiseks, jne.

Nende valdkondade esmane probleem on tagada olemasolevate matemaatiliste teooriate ja nende arvutirealisatsioonide võimalikult täpne vastavus. Seejärel püütakse hinnata arvutimudelil ja füüsikalisel mudelil (või tegelikul objektil) läbiviidud katsete kokkulangevuse täpsust. Arvutiteadus annab selle valdkonna probleemide lahendamiseks üsna ebamääraseid nõuandeid ja soovitusi – peamiselt arvutiteaduse ja teiste teaduste erinevate maailmakäsitluste ja paljude lahendamata teoreetiliste probleemide tõttu, näiteks reaalarv arvutis ja matemaatikas on sisuliselt täiesti erinevad mõisted, pideval ajal baseeruva teooria üleviimisel arvutisse (realiseerimisel arvutis) tekkivaid approksimatsioone (ja eriti nende approksimatsioonide mõju teooria kehtivuspiirkonnale) ei ole veel eriti tõsiselt uuritud. Seetõttu on sellesse klassi kuuluvate süsteemide realiseerimisel raskuseks oluliselt mittetäielikult kirjeldatud nõuete spetsifikatsioon, ja iga konkreetse süsteemi realiseerimisel tekkivad nähtused, mis vajavad spetsiaalset interpretatsiooni, kontrolli ja sageli ka täiendavaid uuringuid.

Reaalajasüsteemid erinevad põhimõtteliselt kahest eelnevalt kirjeldatud süsteemide klassist. Mõlemad eelmised klassid koosnevad süsteemidest, mis on mudelid või "mudeli mudelid" .Reaalajasüsteemid on täpsemal vaatlusel tegelikkuse uus komponent (s.o. mitte mudel, ega ka mitte mudeli mudel), mis toimib iseseisvalt reaalses maailmas, on interaktsioonis teda ümbritsevate reaalse maailma osadega ja võib aktiivselt mõjutada viimaste käitumist. Reaalajasüsteem koosneb tegelikku maailma sisseehitatud arvutist, andurite ja täiturite võrgust, ja programmidest. Näiteks, aerodünaamiliselt mittestabiilne lennuk ja mittestabiilne keemiline reaktsioon on realiseeritavad vaid tänu arvuti aktiivsele sekkumisele. Reaalajasüsteem toimib sageli üsna pika aja jooksul sõltumatult oma kasutajast (inim-operaatorist) ning teeb tema ette seatud ülesannete täitmiseks omi otsuseid. Küllalt sageli on need otsused siiski vaid taktikalised, strateegilisi otsuseid teeb inimene, ka uusi eesmärke formuleerida lubatakse enamasti vaid inimesel.

Paljudel juhtudel on sellise piirangu põhjuseks seadusandlikud kitsendused, mitte reaalajasüsteemide tehniline võimetus strateegilisi otsuseid vastu võtta, või dünaamiliselt uusi eesmärke formuleerida. On palju eksperimentaalseid ja/või mängulisi süsteeme, mis formuleerivad ise uusi eesmärke ning teevad ka strateegilisi otsuseid. Näiteks, tehiselu eksemplarid – tehissipelgad – mis/kes käituvad erinevalt tuttavate ja võõrastega, konkreetne käitumismall on üsna ettearvamatu. Praktikas on sageli ka odavam lasta strateegilisi küsimusi lahendada inimestel.

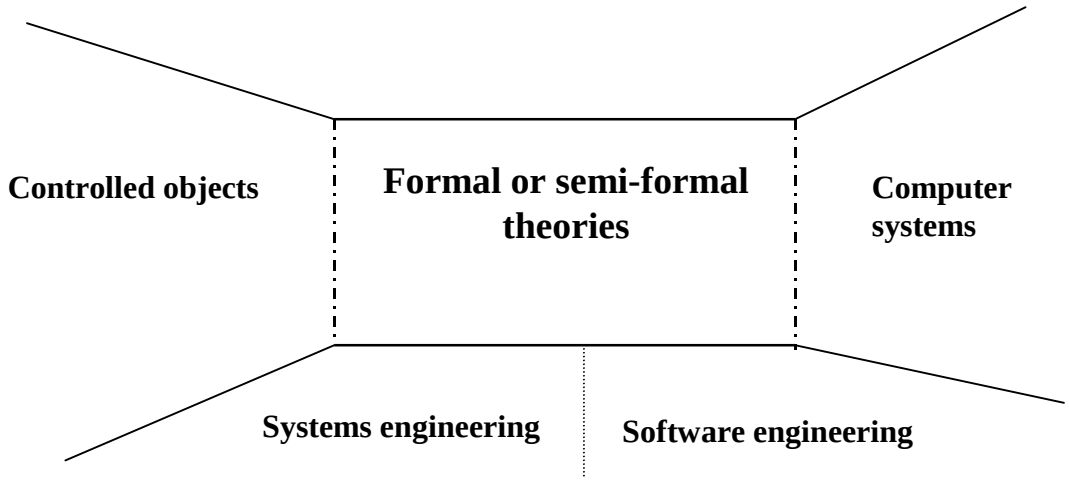
Reaalajasüsteemide tegijad on pikka aega olnud mitmesuguste erialade insenerid ja teadlased. Arvutiteadus on alles suhteliselt hiljuti (1980-ndate alguses) tunnistanud, et reaalajasüsteemid tekitavad mitmeid arvutiteaduses seni tähelepanuta (ja lahendamata) jäänud probleeme ja paradokse. Mõned nendest probleemidest on möödaminnes nimetatud käesolevas kursuses (seoses tarkvara ajakitsendustega) ja neist tuleb lähemalt juttu kursuses LAP 5712 "Tarkvara dünaamika".

1.5 Eelülevaade tarkvarasüsteemide loomise protsessist.

Tarkvarasüsteeme on kaua peetud kõige keerulisemaks inimese loodud tooteks. Alles üsna hiljuti on tekkinud konkurent – geneetiliselt modifitseeritud loom ja/või taim – kuigi konkurent on vaid osaliselt inimese loodud, on ta veelgi vähem haaratav ja

arusaadav kui tarkvara. Geenide interaktsioonid ja nende seos DNA-sse kodeeritud infoga on üsna tundmatud. Nõrga analoogina võiks ette kujutada isemodifitseeruva programmi kirjutamist, silumist ja täitmist mitme erineva inimese poolt – kusjuures iga inimese poolt sisseviidud dünaamiliselt muudetavad käsud ja konstandid on teistele tegijatele teadmata. Toote liigse keerukuse vähendamiseks on tarkvaratehnika sellise praktika lihtsalt ära keelanud, kuid veel 1960-ndate alguses kasutati sellist koodi üsna laialdaselt. Sellegi pärast on tarkvara kui toode ja tema loomise protsess oma suure keerukuse tõttu mitte kuigi hästi juhitav.

Tarkvara loomise protsessi edasiseks lihtsustamiseks on, nagu ka paljude teiste inimtegevuse aladel, süsteemide loomisprotsess jagatud etappideks. Neid etappe ja etappideks jagamise põhimõtteid selgitatakse lähemalt käesoleva teksti III osas. Läbitud etappide käigus liigutakse mitteformaalsest tegelikust maailmast – jättes kõrvale süsteemitegija arvates mitteolulised omadused – poolformaalsesse, mõnel juhul ka formaalsesse, maailma (vt. joonis.1.2). Poolformaalses ja formaalses maailmas on igal süsteemil, nähtusel, jne lõplik arv neid iseloomustavaid omadusi; mitteformaalses (ehk tegelikult eksisteerivas) maailmas usutakse igat süsteemi, nähtust, tegevust, jne olevat võimalik kirjeldada vaid lõpmatu arvu omadustega. Süsteemide spetsifitseerimine, projekteerimine, analüüs, verifitseerimine ja osaliselt ka testimine toimub (pool)formaalses maailmas. Lõpliku realiseerimise käigus viiakse süsteem jälle tegelikult eksisteerivasse (mitteformaalsesse) maailma, kus igal asjal, nähtusel, jne on lõpmata palju neid iseloomustavaid omadusi. Arvutiteadus toimib (pool)formaalses maailmas, süsteemitehnika ja tarkvaratehnika teisendavad mõisteid mitteformaalsest (pool)formaalsesse maailma ja tagasi.



Joonis 1.2. Süsteemi loomise käigus toimuv liikumine mitteformaalsest maailmast formaalsesse ja tagasi.(© Mõtus&Rodd)

Igal üksikul süsteemi arendamise sammul lahendatakse üks (kogum) probleem(e). See võimaldab abstraktsel tasemel igat süsteemi loomise sammu kirjeldada sama skeemiga -- fikseeritakse probleem, otsitakse täiendavat infot konkreetsest ülesandest, valitakse meetodid ja tööriistad, lahendatakse probleem, analüüsitakse saadud

kogemust ning modiftseeritakse meetodeid ja tööriistu. Süsteemi arendamise (loomise) samm on võrreldav teisendust kirjeldava protseduuriga, mida väljendab hulk omavahel seotud tegevusi. Täidetud tegevuste tulemusena areneb süsteem antud sammule eelnevast olekust (lähteolekust) järgmisse olekusse, mis on lähteolekuks järgmisele sammule süsteemi arendamisel ja/või loomisel.

Turul saadaolevate tarkvaratehnika tööriistade (CASE tools) enamus ei ole võimeline automaatselt koguma, salvestama, ja taaskasutama projekti käigus tekkinud uut kogemust. Peamine roll kogemuse kogumisel ja taaskasutamisel on ikka veel inimesel, kuigi tarkvaratehnika tööriistad pakuvad üha efektiivsemat tuge selle rolli täitmiseks.

Kõige esimene samm – teisendus mitteformaalsest maailmast (pool)formaalsesse maailma – on eriti raske ja sõltub oluliselt teisendaja (tarkvarainsener, süsteemi-insener, ingl.keeles *software engineer, systems engineer*) kogemusest, andekusest ja intuitsioonist. Pemine raskus on seotud lõpmata paljude nähtuste ja omaduste hulgast antud süsteemi eesmärgi realiseerimiseks (lõpliku arvu) kõige olulisemate nähtuste ja omaduste ning nendevaheliste seoste valikuga. Subjektiivseks teeb selle sammu ka põhimõtteliselt võimatu (pool)formaalse mudeli ja tegeliku maailma nähtuste kogumi vastavuse tõestamine. Vastavust saab näidata vaid katseliselt, üksikute omaduste suhtes, ja kindlalt fikseeritud olukordades – seda protsessi kutsutakse mudeli (spetsifikatsiooni, projekti) valideerimiseks.

Analoogiline raskus on seotud ka füüsilise projekteerimise etapiga, mis eelneb realiseerimisele. Sellel etapil teisendatakse süsteemi kirjeldus (pool)formaalsest maailmast tagasi reaalsesse maailma – s.t. valitakse väga paljudest võimalikest variantidest välja üks konkreetne füüsiline riistvara konfiguratsioon (arhitektuur) koos süsteemtarkvaraga, teisendatakse tarkvaraprojekt sellele konfiguratsioonile (mis on kombinatoorika ülesanne väga suure arvu võimalike kombinatsioonidega) ja kodeeritakse üksik-komponendid valitud programmeerimiskeeltes. Jällegi puudub reaalne võimalus tõestada, et süsteemi realisatsioon vastab algselt esitatud nõuetele – ainus võimalus on kasutada valideerimist. Praktikas tähendab see, et omaduste olemasolu, mille eksistentsi ei saa tõestada teoreetiliselt, tuleb katseliselt näidata. Saadud omaduste olemasolu kehtib kindlalt vaid olukordades, mis on katseliselt järele proovitud, ülejäänud olukordades tuleb loota heale õnnele ja intuitsioonile.

1.6 Loengukursuse pragmaatiline eesmärk ja selle saavutamiseks kasutatud teed

Käesoleva kursuse peamine eesmärk on anda üliõpilasele algteadmisi reaalajatarkvara tootmise omapärast ja kasutatavatest meetoditest, tuletades ühtlasi meelde üldise tarkvaratehnika põhitõdesid. Samas on see kursus esimene samm reaalajatarkvara, ja üldse reaalajasüsteemide, teooria ja praktika omandamisse. Kõrvaltulemusena peaks õppijal tekkima parem arusaamine mitmete ülikoolis õpetatavate kursuste omavahelistest seostest, ning vajadusest ühendada erinevates kursustes õpitu tervikuks – näiteks, süsteemiteooria, teoreetiline informaatika, praktilise programmeerimisega seotud kursused, arukad juhtimissüsteemid, jne.

Eelnevatel süsteemi elutsükli etappidel tehtud approksimatsioonidest põhjustatud vead kuhjuvad ja ilmnevad sageli alles tarkvara loomisel, kuna tarkvara loomise etapp on süsteemide elutsükli lõpupoole. Et taoliste vigade algpõhjuseid kiiresti avastada peaks tarkvarainseneridel olema hea interdistsiplinaarne haridus, head tööriistad, ja väga hea koostöö teiste erialade esindajatega. Reaalses elus jääb enamasti kõigist eeldustest puudu.

Tarkvara väljatöötajad ootavad tarkvaratehnika kursustelt soovitusi, kuidas:

- Valida ja kasutada paradigmasid ja arvutusmudeleid (*computational models*), ja kuidas nende alusel välja töötada antud rakendusele sobiv süsteemi arhitektuur
- Valida ja kasutada tarkvara loomise metoodikaid ja tarkvaratehnika tööriistu (*CASE tools*, *software engineering environments*)
- Juhtida ja hallata tarkvaraprojekte, valida projektijuhtimist toetavaid tööriistu, koostada projekti osade omaduste hindamiseks vajalikke teoreetilisi mudeleid
- Kasutada valideerimise ja formaalse verifitseerimise erinevaid tehnikaid siis kui neid on vaja
- Jne.

Peaaegu ühelegi loetletud ootustest ei ole võimalik anda ühest, ainuõiget vastust. Kursuse eesmärgiks on suunata kuulaja iseseisva mõtlemise abil konkreetsele rakendusele ja kuulaja iseloomule kõige sobivamate vastuste leidmisele.

Süsteemide loomise kontekstis oodatakse tarkvarainsenerilt (projektijuhilt) oskust:

- Esindada tarkvara rühma huve ja muresid diskussioonides süsteemi-inseneriga
- Objektiivselt, kompetentselt ja ausalt esitada kõrgema taseme juhtkonnale tarkvaraprojekti kulgu kirjeldavad realistlikud hinnangud ja arvamused
- Tagada tarkvara alaste teadmiste kasutamine süsteemi projekti plaani tegemisel, tarkvara mahu hindamisel, tarkvara hinna ja tema valmimise ajakava määramisel, funktsionaalsete ja jõudlushinnangute tegemisel ja ülesannete jagamisel meeskonna liikmete vahel
- Edastada ja filtreerida küsimusi ja ettepanekuid, mis tulevad tarkvara meeskonnalt ja on määratud süsteemi meeskonnale selgitusteks või seisukohavõttudeks
- Selgitada tarkvara meeskonnale ideid, põhimõtteid, kitsendusi ja otsuseid, millel baseeruvad projektile esitatavad tehnilised ja majanduslikud nõuded ning kitsendused.
- Olla, lisaks administreerimisele, tarkvara väljatöötamise tehniline juht.

Enamus ülalloetletud omadustest, oskustest ja võimetest tekib praktilise töö käigus. Paljud omadused ei ole vahetult seotud tarkvara alaste oskustega, vaid eeldavad administreerimis- ja psühholoogia alaseid kogemusi ja teadmisi. Käesolev loengukursus puudutab vaid ühte väikest osa vajalikest tarkvara-alastest teadmistest, kuid üritab anda vihjeid mida ja kuidas hiljem juurde õppida. Seega, käesolev tekst ja sellega kaasnevad loengud ei garanteeri projektijuhi oskusi, kuid soodustavad nende omandamist täiendava iseseisva töö korral.

Loengukursus koosneb, lisaks juba loetud sissejuhatusele, veel kolmest osast:

- Teine osa, **“Sissejuhatus reaalajasüsteemidesse”,** selgitab reaalajasüsteemide valdkonna omapära võrreldes muude valdkondadega ja toob praktilisi näiteid reaalajasüsteemide kasutusaladest. Erilist tähelepanu on pööratud reaalajasüsteemide omadustele ja nõuetele, mis vahetult mõjutavad reaalajatarkvara loomise protsessi.
- Kolmas osa **“Tarkvara projektide planeerimine ja haldamine”** selgitab ja osaliselt põhjendab projektide haldamiseks vajalikke töid ning nende tegemise meetodeid. Erilist tähelepanu pööratakse riski haldamisele – see on asi, mis kipub igapäevases praktikas kahe silma vahele jääma, kuid on edukaks projektijuhtimiseks väga oluline.
- Neljas osa “**Andmevoo ja objekt-orienteeritud mudelitel baseeruv metoodika**”, mille kohta käesolevas tekstis puuduvad kommentaarid, annab ülevaate kahest enamlevinud programmide projekteerimismeetodite klassist. Mõlema nimetatud meetodite klassi kasutuspiirkonda on üritatud laiendada ka reaalajasüsteemidele – kahjuks ei ole seni saadud tulemused veel osutunud nii õnnestunud, et nad oleksid laialdaselt kasutusele võetud. Loengutes loetletakse kasutuspiirkonna laiendamise ebaõnnestumise põhjuseid, põhjalikum selgitus mõnede ebaõnnestumise põhjuste kohta antakse kursuses LAP 5712 “Tarkvara dünaamika”.
- Laboratoorsed tööd ja iseseisev töö UML-il baseeruval Artisan Real-time Studio keskkonnal peaks aitama süvendada loengutes õpitut ja ühtlasi andma mõningase iseseisva töö kogemuse UML-l baseeruvas projekteerimiskeskkonnas.

1.7. Esimese osa kordamisküsimused

1. Mille poolest erinevad süsteemitehnika ja tarkvaratehnika? Mis teeb neid sarnaseks?
2. Mida tuleb silmas pidada süsteemi tükeldamisel alamsüsteemideks?
3. Selgitage alamsüsteemide ühendamisel sageli ilmneva sünergismi tekkimise põhjuseid?
4. Nimetage mõned omadused, mis põhjustavad reaalajasüsteemide klassi ja modelleerimissüsteemide klassi erinevuse.
5. Miks on vaja süsteemi (tarkvara) elutsüklit jagada etappideks?
6. Selgitage, mis on peamine erinevus süsteemi spetsifikatsiooni (projekti) ja tegeliku süsteemi vahel.
7. Miks ei saa süsteemile esitatavad kasutajanõuded kunagi täielikult kirjeldada süsteemi soovitavat funktsioneerimist?
8. Millises proportsioonis, Teie arvates, peaks tarkvaraprojektijuhil olema erialaseid teadmisi võrreldes administratiivse töö alaste ja psühholooogia alaste teadmistega? Kuidas on Teie arvates olukord kõrgema tasemega juhtide puhul?
9. Kas lisaks funktsionaalsetele nõuetele on vaja tarkvarasüsteemile esitada ka muid nõudeid? Milliseid?
10. Palun andke oma õppejõule (kirjalikult) soovitusi Esimeses osas täiendavat selgitust vajavate teemad kohta.

1.8 Esimeses osas kasutatud kirjanduse viited

1. Behforooz A. and F.J.Hudson (1996) “Software Engineering Fundamentals” Oxford University Press, 661 pp., ISBN 0-202-56529-3
2. Meyer B. (1996) “Reality: A cousin twice removed”, IEEE Computer, vol.29, no.7, 96-97
3. Motus L. and M.G. Rodd (1994) “Timing analysis of real-time software”, Elsevier Science Publishing, Oxford, 212 pp, ISBN 0 08 0420257

II osa
Sissejuhatus reaalajasüsteemidesse

2.1 Reaalajasüsteemid

Reaalajasüsteemid on Eestis ikka veel (aastal 1999!) suhteliselt vähe levinud mõiste. Sageli tõlgendatakse seda mõistet valesti, veel sagedamini ei teata üldse mis on reaalajasüsteem. Heas (eestimaises haritlaste) seltskonnas mõjub termini "reaalajasüsteem" valjusti väljaütlemine ikka veel samuti kui 20-nda sajandi esimesel poolel tol ajal rõvedaks peetud väljendi rõhutatult kõvasti ütlemine -- tekib piinlik vaikus ja ütlejast hakatakse vaikselt eemale nihkuma. Mõnes teises Eestimaa seltskonnas (näiteks eesti arvutiteadlaste hulgas) leidub mitmeid inimesi kes on valmis vanduma, et nad teavad kõik reaalajasüsteemidest ja tegelevad nendega iga päev. Küllalt suure tõenäosusega on tegemist endast väga heal arvamusel olevate inimestega, kes on seda välismaal moesõnaks muutunud terminit kusagil arvutite kasutamise kontekstis kuulnud, võib olla üksikküsimustega isegi pisut tegelenud. Süstemaatilist reaalajasüsteemide alast haridust omavad Eestis praegult veel väga vähesed inimesed.

Selle peatüki eesmärgiks on kirjeldada reaalajasüsteeme mitmest erinevast vaatepunktist, selgitades nii nende erinevusi mittereaalajasüsteemidest. Loodetavasti ei jää lugejale reaalajasüsteemist sama ebamäärast ja eklektilist muljet nagu nelja pimeda (ja paigalseisva) eksperdi poolt koostatud elevandi kirjelduse põhjal võib elevandist jääda.

Reaalajasüsteemi mõiste mitteadekvaatne tajumine 1999 aasta Eesti igapäevases elus näitab kaudselt meie tsivilisatsiooni (eriti infotehnoloogia) äärmiselt õhukest kultuurikihti. Me oleme edukalt sundinud tiigrikutsika hüppama – populistlikus ja pikaajalises perspektiivis kahtlemata väga tugev esimene samm pikal teel – esialgu ei ole poliitikud ja ühiskond veel otsustanud kas tuleb ka järgmisi samme kõrgtehnoloogilise infoühiskonna loomisel. Igal juhul, valdkonda kuhu USA 1990-ndatel aastatel investeerib ligi 70% infotehnoloogia arendamiseks kuluvast rahast tunnevad paljud kõrgetel kohtadel olevad Eesti ametimehed vaid kuulujuttude järgi.

Reaalajasüsteem on tüüpiline tehismaailma teaduse uurimisobjekt ja ka produkt. Tehismaailma teadus on sünteesil baseeruvate teadusvaldkondade (paljud tehnikateadused, majandus, muusika, arhitektuur, kunst, jne) üldnimetus. Mõiste tõi sisse Nobeli (majandus)preemia laureaat H.A.Simon, kes on selle idee kallal töötanud 1960-ndate lõpust alates (vt. **Simon (1996)**). Tehismaailma teadust iseloomustab lähtumine eesmärgist ja vajadus sünteesida eesmärki rahuldav süsteem kahe või enama loodusseadusi rahuldava rakendusvaldkonna liidesesse (või liideseks). Reaalajasüsteem moodustab tavaliselt liidese kolme sellise valdkonna -- juhitav või jälgitav objekt, arvutisüsteem ja inimene-operaator -- vahel (vt.ka joonis 2.1). Reaalajasüsteemi võib vaadelda kui loodus- või tehiskeskkonda sisseehitatud arvutisüsteemi (*ingl. k. embedded computer system*, eesti keeles on soovitatud sardsüsteem), mis muudab oluliselt esialgse keskkonna funktsioneerimise kvaliteeti ja/või vormi.

Teistest, rohkem uuritud valdkondadest on taolist nähtus kirjeldatud automaatjuhtimis-teoorias – suletud juhtimisahelas töötav kontroller ja juhitav objekt moodustavad uue terviku, mille täpseid omadusi on üldjuhul (näiteks, mittelineaarsete objektide korral) üsna raske eelnevalt hinnata. Täpsemalt on selle nähtuse mõju tarkvaratehnikale kirjeldanud **Giddings (1984).**

Selles osas räägitakse:

- Mida endast kujutab reaalajasüsteem
- Kus kasutatakse reaalajasüsteeme, koos näidetega
- Tarkvara protsessi omapärast reaalajasüsteemides
- Arvutisüsteemi ja tema poolt juhitava või jälgitava reaalse maailma objekti käitumise kooskõlastamisest
- Miks on formaalsed meetodid reaalajatarkvara arendamisel olulisemad kui muudes tarkvaravaldkondades.

Kuna tegemist on reaalajatarkvaratehnika kursusega, pööratakse peamine tähelepanu arvutite ja tarkvaraga seotud aspektidele. Reaalse maailma objektid ja inimese-operaatori vajadused leiavad (loodetavasti) kajastamist teistes õppeainetes.

2.2 Reaalajas töötav arvutisüsteem.

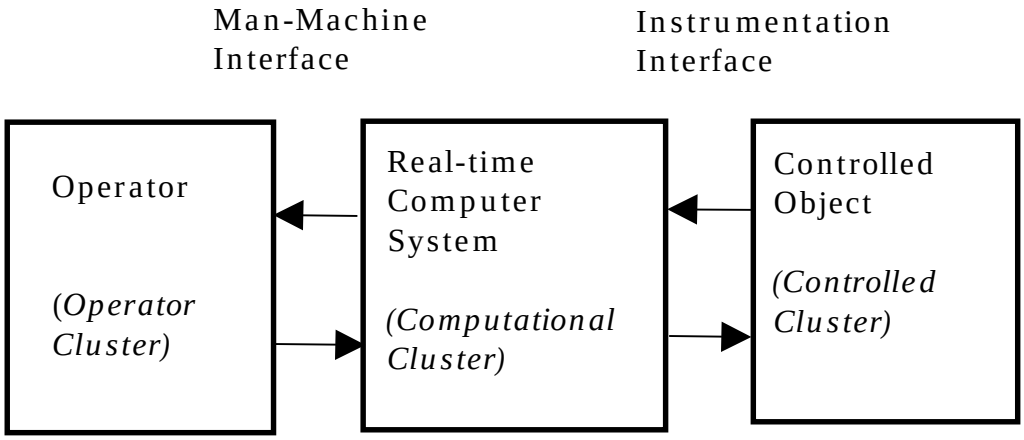
Arvutisüsteem töötab reaalajas (*is a real-time computer system*) kui süsteemi töö õigsus on defineeritud mitte ainult algoritmi täitmisel saadud arvutustulemuste alusel (*logical results*), vaid oluline on ka ajahetk, millal need tulemused saadi. **(Kopetz, 1997).** Sama kehtib ka algoritmi poolt kasutatud lähteandmete õigsuse kohta -- oluline on nii lähteandmete mõistlik väärtus, kui ka selle väärtuse tekkimise aeg. Tuletame meelde ka reaalajas töötava arvutisüsteemi teise olulise omaduse – reaalajas töötav arvutisüsteem on vahetult (s.t. inimesest sõltumatult) ühenduses lähteinformatsiooni allikaga, sageli ka arvutustulemusi kasutava loodusliku või tehisobjektiga.

Ülaltoodud, intuitiivselt loomulikuna tunduva definitsiooni esimesest avalikust kasutamisest kuni selle omaksvõtmiseni maailma juhtivate spetsialistide poolt, kulus peaaegu kakskümmend aastat. Vaatamata näilisele lihtsusele on selle definitsiooni taga sügav analüüs klassikalise matemaatika ja algoritmiteooria rakendusvõimalustest reaalajasüsteemide realiseerimisel. **(vt. näiteks, Maler 1992, Motus and Rodd 1994).** Matemaatiliste, nii nagu ka kõigi teiste formaalsete teooriate kasutamine baseerub vaikimisi tehtud eeldusel, et teooria aluseks olev aksiomaatiline baas ei muutu teooria kasutamise ajal -- selle vaikimisi tehtud eelduse täitmise vajadus on üks peamine reaalajas töötavatesse arvutisüsteemidesse ajakitsenduste sissetoomise põhjustest. Paljud ajakitsendused fikseerivad ajaintervallid, kus kasutatava teooria aksiomaatilise baasi võib lugeda muutumatuks (s.t. formaalsed teooriad (ja järelikult ka algoritmiteooria) kehtivad kindlasti selles ajaintervallis). Suhteliselt harvem kirjeldavad ajakitsendused nn. sünkroniseerimisnõudeid, mis tagavad reaalajasüsteemi erinevate komponentide probleemideta koostöö.

Vaikimisi eeldatud aksiomaatilise baasi staatilisuse nõude täitmine tundub matemaatikutele sedavõrd loomulik, et sageli unustatakse sellest eeldusest isegi rääkida. Reaalajasüsteem on üks esimesi rakendusi, kus samaaegselt on kasutusel mitmed, erinevate objektide kohta käivad teooriad (kusjuures erinevad objektid ja nende kohta käivad teooriad (ja teooriate aksiomaatilised baasid) muutuvad/evolutsioneeruvad erineva kiirusega). On üsna loomulik, et kõikide kasutusel olevate teooriate kehtivuspiirkonnad ei pruugi kokku langeda. Erinevate teooriate koos kasutamine otsuste tegemiseks on aga õigustatud ainult eeldusel, et kõik teooriad kehtivad samaaegselt otsuse tegemisele vahetult eelnevas piisavalt pikas ajaintervallis.

Näide aksiomaatilise baasi staatilisuse rikkumisest algoritmiteoorias. Kujutage ühe või mitme lugemispeaga Turingi masinat, mis täidab teie poolt antud algoritmi. Kokkuleppe kohaselt täidetakse algoritmi kuni on saavutatud algoritmi lõppolek. Olukord, kus mõne aja möödudes, aga enne kui algoritm jõuab lõppolekusse, hakkab loodus (või teie kolleeg) lindil olevat informatsiooni (algoritmi, lähteandmeid ja vahetulemusi) oma tahte kohaselt muutma, on käsitletav Turingi masina aksiomaatika dünaamiline muutumisena – samal ajal on see äärmiselt tüüpiline olukord reaalajasüsteemides.

Reaalajas töötav arvutisüsteem ei oma mõtet kui ta töötab muust maailmast isoleeritult -- ta on alati suurema süsteemi osa. Meid huvitavas kontekstis kutsutakse seda suuremat süsteemi **reaalajasüsteemiks** *(real-time system).* Reaalajasüsteem on enamikel juhtudel vaadeldav mitme omavahel interaktsioonis oleva dünaamilise süsteemi kogumina. Üks nimetatud kogumis olevatest dünaamilistest süsteemidest peab olema reaalajas töötav arvutisüsteem, mis jälgib ja/või juhib teisi kogumisse kuuluvaid dünaamilisi süsteeme.



Joonis 2.1 Reaalajasüsteemi üldistatud struktuur (© H. Kopetz, 1997)

Pragmaatilistel kaalutlustel jagatakse (käesolevas loengusarjas) reaalajasüsteem kolme ossa (vt. joon. 2.1), millest käesoleva loengusarja kontekstis suurimat huvi pakub **Arvutikobar (Computational cluster)**, selle loomine ning selle funktsioneerimise analüüs ja verifitseerimine. **Operaatori kobar (Operator cluster**) ja

Juhitav/jälgitav kobar (Controlled cluster) ühendatakse sageli mõiste "ümbritsev keskkond" alla. Ümbritsevast keskkonnast tulenevad inimese ning loodusliku ja/või tehnolooogilise protsessi poolt esitatud nõuded ja kitsendused, mis suures osas määravad arvutikobara funktsioneerimise.

Arvutikobar ja ümbritsev keskkond on ühendatud kahe liidesega:

- inimliidese (*man-machine interface*) kaudu suhtleb inimene-operaator arvutikobaraga; paljudel juhtudel on inimliides selline, et kasutaja ei aimagi selle taga arvutit (näiteks, gaasipedaal kaasaegses autos); inimliides koosneb seadmetest (näiteks, sõrmistik, hiir, auto gaasipedaal, interaktiivne mnemoskeem, kuvar, valgustabloo, hääleväljund/-sisend) mis võimaldavad arvutil ja inimesel (kasutajal) teineteisest aru saada; keerulisemad inimliidesed suudavad kohaneda inimese tervisliku seisundiga (näiteks stressi tase) ja vastavalt muuta inimesele väljastatava informatsiooni kogust ning valikut;
- protsessiliidese (*instrumentation interface*, vahel ka *process interface*) kaudu suhtlevad arvutikobar ja juhitav/jälgitav kobar, see suhtlemine toimub enamasti ilma inimesepoolse initsiatiivi ja vahelesekkumiseta; protsessiliides koosneb juhitava/jälgitava kobaraga ühenduse pidamiseks vajalikest seadmetest, näiteks andurid ja täiturid (*sensors and actuators*); andurite ülesandeks on arvutis lähteandmetena kasutatavatele muutujatele hetkel aktuaalsete väärtuste hankimine juhitavalt kobaralt (s.t. mõõtmine ja sobivale kujule teisendamine), täiturite ülesandeks on arvutikobaras juhitava kobara käitumise mõjutamiseks arvutatud muutujate väärtuste edasiandmine juhitavasse ja/või jälgitavasse kobarasse (teisendamine sobivale kujule, sobivate osade kaupa sobival hetkel väljastamine). Näiteks termopaari pinge teisendamine temperatuuriks ja selle väärtuse edastamine arvutisse digitaalsel kujul, sobiva kontsentratsiooniga bensiini/õhu segu pritsimine mootori põlemiskambrisse.

Reaalajatarkvaratehnika, eriti selle aluseks olevate arvutusmudelite, toimimise seisukohalt on oluline mõista reaalajatarkvara ja konventsionaalse tarkvara töökeskkonna erinevust termodünaamika mõttes. ***Andme-ja infotöötlussüsteemid on termodünaamilises mõttes suletud*** – lähteinfo liigub arvutisse sisse ja tulemused arvutist välja läbi väga rangelt tsenseeritud kanalite, hästi defineeritud kujul (näiteks, ASCII kood, *bitmap* kuju, inimkõne (mis teisendatakse lõppkokkuvõttes ASCII koodi), andmevahetus protokollid on fikseeritud). Võrrelge olukorda tuntud näitega, kus Maxwelli deemon kontrollib aatomite/molekulide energiat suletud kastist sisse ja välja liikumisel. Sellisesse kasti võib, vastavalt deemoni tahtele, tekitada kas kõrgema energianivoo (või madalama energianivoo) kui kasti ümbritsevas keskonnas. Selline nähtus on tuntud elevandiluutorni paradigma nime all – arvuti (elevandiluutorni) sees saab eeldada ja/või teha seda, mille eeldamist/tegemist ümbritsevas keskkonnas takistab liiga suur entroopia (s.t. mõjuvate faktorite suur hulk, suutmatus neid jälgida, ja mõnede faktorite oluliselt juhuslik iseloom).

Reaalajasüsteemid on termodünaamilises mõttes avatud. Reaalajasüsteemil on kaks liidest – läbi inimliidese liigub enamus informatsiooni analoogiliselt ülalkirjeldatuga (kuid mitte kogu inimliidest läbiv informatsioon ei ole nii rangelt tsenseeritud). Läbi protsessiliidese (ja osaliselt ka läbi inimliidese – näiteks gaasipedaal autos) liigub väga erinevatesse esinemisvormidesse kodeeritud informatsioon – analoogkujul (mehhaaniliselt või elektriliselt kodeeritud) informatsioon, impulsijadadesse

kodeeritud informatsioon, jne, mida on üsna lootusetu rangelt tsenseerida. Lisaks puudub arvutil võimalus valida info vastuvõtu hetke (erinevalt andme- ja infotöötlussüsteemidest) – infovahetus toimub tihti keskkonna initsiatiivil ja keskonnale sobival hetkel ning paljud andmevahetust toimetavad osad on pidevalt (s.t. lõpmata sageli) aktiviseeritud. Termodünaamilise avatuse tõttu toimivad arvuti, jälgitav/juhitav objekt ja inimene operaator tugevas koostöös ja arvutis ei ole võimalik kasutada eeldusi/tegevusi, mis ei oleks võimalikud ümbritsevas keskkonnas. See seab olulised piirangud reaalajatarkvara projekteerimisele ja toimimisele.

Ülalöeldust selgub ka, et *paralleelsus (concurrency)* on reaalajasüsteemide normaalne töövorm – arvuti peab samaaegselt tegelema andmevahetusega, täitma operaatori soove, ja arvutama välja juhttoimeid (tegema otsuseid). Lisaks neile tegevustele tuleb olla valmis eriolukordade töötluseks (*exception handling*), jne. Rangelt võttes on reaalajatarkvara normaalne töörezhiim nn. ***sundparalleelsus*** – ***ingl. k. forced concurrency*** (vt. täpsemalt **Motus & Rodd, 1994**) – see on paralleelsus, mis on tarkvara insenerile peale sunnitud ümbritseva keskonna poolt. Meenutame, et tavaline paralleelsus (maatriks- ja konveierarhitektuuriga arvutites, või ka lihtsalt multiprotsessorites) on põhimõtteliselt vabatahtlik töörezhiim, mille kasutamine sõltub projekteerija tahtest (ning algoritmi võimalustest) ja vabade protsessorite olemasolust. Sundparalleelsuse nõude mittetäitmine tähendab reeglina loobumist reaalajasüsteemi nõuete täitmisest ja viib garanteeritult projektiülesannet mitterahuldava tulemuseni..

Arvutikobar peab reageerima ümbritsevast keskkonnast tulevatele stiimulitele (muutujate väärtused, info sündmuste toimumise kohta) kindla aja -- nn. piir-aja (*deadline*) jooksul. Tavaliselt kujutab piir-aeg endast ajaintervalli kahe sündmuse vahel – stiimuli (mõõtmine, sündmus) toimumisest juhitavas/jälgitavas kobaras kuni arvutikobara vastava reaktsiooni (uus klapi asend, seadme väljalülitamine) jõudmiseni tagasi juhitavasse/jälgitavasse kobarasse. Piir-aja kestus sõltub ümbritseva keskkonna dünaamilistest omadustest ja ei ole programmisti suva järgi muudetav. Kui arvutuste tulemus on kasutatav ka pärast piir-aja möödumist (väheneb vaid saadav tulu), on tegemist nõrga piir-ajaga (*soft deadline*); kui arvutustulemuse kasutamine pärast piir-aja möödumist ei ole eriti efektiivne (tulu asemel võib tekkida mitte eriti suur kulu), on tegemist kindla piir-ajaga (*firm deadline*); kui arvutustulemuse kasutamisel pärast piir-aega võib tekkida katastroof, on tegemist range piir-ajaga (*hard deadline*).

Reaalajas töötavat arvutisüsteemi, mis peab rahuldama vähemalt ühte ranget piiraega, nimetatakse ranges reaalajas töötavaks arvutisüsteemiks (*hard real-time computer system*), vahel ka ohutuskriitiliseks reaalaja-arvutisüsteemiks (*safety-critical real-time computer system*).

Kui kogu arvutisüsteemis on vaid nõrgad ja/või kindlad piir-ajad ning ei eksisteeri ühtegi ranget piiraega, on tegemist nõrgas reaalajas töötava arvutisüsteemiga (*soft real-time computer system*). Nõrgas reaalajas töötavate arvutisüsteemide projekteerimine on märksa erinev ranges reaalajas töötavate arvutisüsteemide projekteerimisest (reeglina lihtsam, kuid mitte alati).

1.3 Funktsionaalsed nõuded reaalajasüsteemidele.

Kuna reaalajasüsteemi kaks osapoolt (operaator ja juhitav/jälgitav kobar) on looduses tihti juba olemas, mõistetakse reaalajasüsteemi funktsionaalsete nõuete all tavaliselt inim-operaatori ja juhitava/jälgitava kobara poolt arvutikobarale esitatavaid nõudeid. Funktsionaalsed nõuded iseloomustavad kasutaja ja/või tellija ootusi süsteemi poolt täidetavatele funktsioonidele – tegevustele, mis on suunatud süsteemi eesmärgist tulenevate põhitegevuste sooritamisele. Oma iseloomule vastavalt grupeeritakse funktsionaalsed nõudmised kolme suurde gruppi:

- nõudmised andmehõivele (*data collection, data acquisition, data elicitation*)
- nõudmised juhtimisele (*direct digital control and supervisory control*)
- nõudmised inimliidesele (*man-machine interface*).

Lisaks funktsionaalsetele nõuetele esinevad ka nn. mittefunktsionaalseid nõudeid, mis kirjeldavad funktsionaalsete nõuete rahuldamise kvaliteedile esitatavaid nõudeid. Kvaliteedi alla kuuluvaks loetakse ajakitsenduste täitmine, töökindlus, ohutus, turvalisus, jne. Mittefunktsionaalsetest nõuetest tuleb juttu alajaotustes 2.4 ja 2.5.

2.3.1 Andmehõive

Juhitav ja/või jälgitav objekt (näiteks auto, tehnoloogiline protsess, keemiline reaktsioon, looduskeskkonna seisund) muudab oma olekut ajas -- kui olek ajas ei muutuks, ei oleks ka vastava objekti jälgimisel ja/või juhtimisel mingit mõtet. Teoreetiliselt on igal valitud ajahetkel võimalik mõõta *kõigi objekti olekut iseloomustavate muutujate väärtuseid* -- saadud väärtuste komplekt määrab objekti oleku antud ajahetkel. Pragmaatilistel põhjustel ei ole me kunagi huvitatud kõigi olekut määravate muutujate väärtustest -- neid on tavaliselt liiga palju (mitteformaalses maailmas lõpmatu palju). Me huvitume vaid muutujatest, millede väärtuste teadmine on oluline objekti jälgimise ja/või juhtimise eesmärgi saavutamiseks. Seega iga **olekumuutuja** (nagu neid tihti lihtsustatult nimetatakse) on tegelikult mingi konkreetse **eesmärgi saavutamiseks oluline olekumuutuja.**

Tegelikus elus, erinevalt teooriast, ei suuda me ühel ja samal ajahetkel mõõta isegi mitte kõiki väljavalitud olulisi olekumuutujaid – piirangud tekivad kellade sünkroniseerimise täpsusest, mõõtmisprotsessi alustamise hetke sünkroniseerimise täpsusest, mõõtmisprotsessi kestusest, analoog/digitaal muundurite töö kestusest, jne. Ka ei suuda arvuti kuigi paljude muutujate väärtuseid samaaegselt sisestada. Siit tekib lisaküsimus – kui lähestikku peavad olema mõõtmishetked, et neid saaks (selle objekti puhul) lugeda samaaegseteks?

Iga olekumuutuja on vähemalt ühe alamsüsteemi juhtimisalas (*sphere of control*). Olekumuutuja väärtust võivad muuta ainult need alamsüsteemid, mille juhtimisalas antud olekumuutuja on. Teised alamsüsteemid võivad (reeglina) ainult lugeda tema väärtust, kuid ei tohi seda muuta. Näiteks, kolvi asendi muutmine mootori silindris on auto mootori juhtimisalas -- ükski alamsüsteem väljaspool mootorit ei saa oma tahtmist mööda kolvi asendit muuta, teatud kitsendustega võib seda asendit ainult jälgida.

Esimene funktsionaalne nõue andmehõivele on -- jälgida kõigi oluliste olekumuutujate väärtuseid ja moodustada neist muutujatest juhitava objekti (või tema osa) olekuväärtus (*real-time image*). Iga olekumuutuja väärtus on aja funktsioon (muutub ajas), seetõttu suvalisel hetkel mõõdetud väärtus saab tegelikkusele vastata ainult mingi lõpliku ajaintervalli jooksul -- igal olekumuutujal on oma väärtuse kehtivusintervall (widely used term is *validity interval* (e.g. Motus and Rodd, 1994), sometimes also named as *accuracy interval* (Kopetz, 1997)). Kehtivusintervalli pikkus sõltub juhitava objekti dünaamikast. Kui meid huvitab süsteemi või tema osa olekuväärtuse (*real-time image*) kehtivusintervall, saab selle arvutada olekusse kuuluvate muutujate väärtuste kehtivusintervallide kaudu. Küllalt tihti võivad mõned olekumuutuja väärtuse kasutajad kehtestada konkreetsele olekumuutujale oma isiklikud kehtivusintervallid. Seega on kehtivusintervall ühelt poolt määratud objektiivselt jälgitava ja/või juhitava objekti dünaamikaga, kuid teiselt poolt ka olekumuutuja väärtuse kasutamiseesmärgiga – selge lisaraskus tarkvara projekteerijatele ja täiendav programmi täitmisaegse töö kontrolli vajadus.

Kehtivusintervalli pikkuse määramise näide. Muutuja väärtuse kehtivusintervalli teadmise ja arvestamise tähtsust illustreerib järgmine näide. Vaatame näiteks autot, mis on jõudnud valgusfoori poolt kontrollitavale ristteele. Kui pikk on vaatluse “foori tuli on roheline” kehtivusintervall? On päevaselge, et kui auto sõidab ristteele foori punase tule ajal on katastroof üpris tõenäoline. Kehtivusintervalli pikkuse mõistlikuks määramiseks on võimalik kaks interpretatsiooni:

- need, kellele meeldib range piiraeg (*hard deadline*) eeldavad, et vaatlus “foori tuli on roheline” kehtib rohelise tule süttimisest kuni kollase tule kustumiseni
- need, kellele meeldib nõrk piiraeg (*soft deadline*), või lepivad ka kindla piirajaga (*firm deadline*), loevad nimetatud vaatluse kehtivusajaks vahemiku rohelise tule süttimisest kuni rohelise tule kustumiseni; nii jääb kollase tule põlemise aeg halliks tsooniks, mille ulatuses võib piiraega ületada ilma eriti suure riskita.

Eestis – erinevalt tsiviliseeritud maadest -- on laialt levinud kolmas (mittesoovitatav) foori rohelise tule kehtivusaja interpretatsioon, mille kohaselt roheline tuli hakkab kehtima punasele tulele järgneva kollase tule süttimise hetkest ja lõpeb rohelisele tulele järgneva kollase tule kustumise hetkega. Mingil kombel peegeldub taoline käitumine ka tarkvaraprojektide juhtimise ja tarkvaratoodete projekteerimise ning valmistamise praktikat Eestis.

Teine funktsionaalne nõue andmehõivele on -- uuendada olekumuutujate väärtuseid enne kui lõpeb eelmise väärtuse kehtivusintervall. Sõltuvalt muutuja olemusest on nende väärtuste uuendamiseks kaks teed:

- perioodiline väärtuse mõõtmine/uuendamine vastavalt kella poolt mõõdetud kindlale perioodile (*time-triggered observations*)
- väärtuse mõõtmine/uuendamine iga kord kui muutuja väärtus muutub (*event-triggered observations*).

Viimane variant on praktikas sageli mugavam (vältimatu), kuid arvutis töötlemisel toob kaasa ebamugavusi ja potentsiaalseid riske. Sellisel juhul ei ole muutuja väärtuse kehtivusintervalli pikkus täpselt teada (on juhuslik). Juhusliku pikkusega mõõtmisintervall võib tekitada tõsiseid konflikte seoses tarkvara projekteerimisele eelnevas süsteemi loomise tegevuses kasutatud teooriate kehtivuspiirkonna

ennustamatu (raskesti ennustatava) muutumisega. Enamus diskreetse ajaga teooriaid eeldab, et mõõtmisperiood on regulaarne. Mitteregulaarse mõõtmisperioodiga saadud andmete kasutamine võib juhtimisalgoritmi töö kvaliteedi alla viia, või isegi teoreetiliselt stabiilse süsteemi muuta mittestabiilseks.

Kolmas funktsionaalne nõue andmehõivele on anduritelt kogutud toorandmete "normaliseerimine", s.t. arvutile arusaadavaks tegemine. Normaliseerimist nimetatakse vahel ka eeltöötluseks (*signal conditioning*). Andurilt tulev signaal on tavaliselt mingi füüsikaline suurus -- pinge, vool, mehhaaniline nihe, vms. -- mis sellisel kujul on arvutile arusaamatu ja võib olla küllalt keerulises vahekorras tegelikult mõõdetud muutuja olemusega (näiteks ristlõikest ajaühikus läbivoolava vedeliku kogus). Seetõttu nimetatakse andurilt tulevat signaali väärtust toorandmeteks (*raw data*). Toorandmed tulevad kalibreerida, teisendada vajalikkudesse mõõtühikutesse, alles siis saab rääkida mõõdetud andmetest (*measured data*). Viimasel ajal levinud rakendustes (näiteks iseliikuvad seadmed = autonomous intelligent vehicles) tuleb üha sagedamini usaldatavate mõõdetud andmete saamiseks kombineerida mitmelt erinevalt andurilt saadud toorandmeid -- nn. erinevatelt anduritelt saadud mõõtetulemuste ühendamine (*sensor fusion*). Sama olukord on kaudsete mõõtmiste kasutamise korral – selle asemel, et mõõta vahetult väga raskelt kättesaadava (mõõdetava) muutuja väärtust, mõõdame mitme lihtsamini kättesaadava muutuja väärtuseid ja arvutame meid huvitava muutuja väärtuse kaudselt (valemite, mudelite, jne kaudu), kasutades nn. mõõtetulemuste ühendamise ideed.

Lisaks ülalkirjeldatule tuleb kontrollida, kas saadud mõõtmistulemused on füüsikaliselt mõistlikud (s.t. kas andur ise oli töökorras) ning lisada muutuja väärtusele usaldatavuse lipik ja ka mõõtmishetke fikseeriv ajalipik. Kui kõik mõõtmisega seotud etapid on edukalt läbitud, saame toorandmetest arvutitöötluseks sobiva andmeelemendi (*agreed data element*).

Neljas funktsionaalne nõue andmehõivele on -- alarmiseire (*alarm monitoring*), s.t. mõõdetud andmete ja vastuvõetud sündmuste pidev seire, selleks et õigeaegselt avastada, ja teatada operaatorile, objekti (normaalse käitumisega võrreldes) ebatavaline käitumine. Näiteks, toru lõhkemine keemiatehases põhjustab paljude arvutitöötluseks sobivate andmeelementide kõrvalekaldumise normist -- erinevates kohtades mõõdetud rõhud, temperatuurid, vedeliku nivood, vahel ka atmosfääri koostis. Tekib nn. alarmilaviin (*alarm shower*). Arvutikobar peab avastama ja kuvama operaatorile kõik need alarmiteated ja peab aitama operaatoril välja selgitada laviini põhjustanud primaarse põhjuse (*primary event*).

Selleks registreeritakse kõik avastatud alarmid spetsiaalses alarmi logis, märkides ära alarmi tekkimise täpse aja. Alarmisõnumite täpne järjestus ajas aitab elimineerida sekundaarseid alarmisõnumeid ja kontsentreeruda algpõhjusele. Keerukates süsteemides ei piisa ainult ajalisest järjestusest, kasutusele tulevad teadmusbaasidel põhinevad veapuud (*fault trees*), või nendega analoogilised põhjuslikke seoseid kirjeldavad vahendid. Lisaks operaatori abistamisele vigade avastamisel ja kõrvaldamisel on taolistel abivahenditel oluline roll alarmilaviini korral sidevõrgus hüppeliselt kasvava andmevahetuse mahu hindamiseks (*peak-load alarm traffic*).

Sidevõrgu seisukohalt on hüppeline alarmisõnumitest tingitud andmevahetuse mahu kasv väga harv sündmus, mis tähendab, et suur osa võrgu läbilaskevõimest on enamus aega kasutamata. Näiteks, Etherneti tüüpi võrkude korral algab ülekandeaegade eksponentsiaalne suurenemine umbes 36% saavutamisel võrgule projekteeritud maksimaalse liikluse mahust. Üldine soovitus (rusikareegel) reaalajasüsteemides, ilma et arvestatakse rakenduse omapära, on hoida võrgu liikluskoormus normaalses töörezhiimis tasemel 2-3% maksimaalsest projekteeritud liikluskoormusest. Alarmilaviini korral on eeldatav liikluskoormuse kasv tavaliselt umbes üks suurusjärk. Selline ettevaatus tagab sõnumite edastamise enam-vähem normaalse (arvutusliku) kiiruse ka alarmilaviini korral.

Näide: Tuumajaama seire ja automaatse sulgemise süsteemi ainus ülesanne on tagada usaldusväärne töö alarmilaviini korral. Kõik loodavad, et seda olukorda kunagi ei tule, kuid ressursid peavad olema selleks olemas. *Three-mile islandi* tuumakatastroof on hea näide, mis võib juhtuda kui seiresüsteem ei toimi alarmilaviini korral hästi. Teine näide on 1994 aastal toimunud õnnetus keemiatehases (vt. **Bransby (1998)**)

Märkus: Alarmiseire, mis on sisuliselt andmehõive alamsüsteemi funktsioon, moodustab vaid väikese osa reaalajasüsteemi eriolukordade alamsüsteemist (exception handling subsystem). Eriolukordade alamsüsteemi osi võib leida praktiliselt kõigist reaalajatarvara osadest, kaasa arvatud operatsioonisüsteem ja andmebaasi haldussüsteem. Eriolukordade alamsüsteemist on pisut rohkem kirjutatud alajaotuses 2.4 Alarmiseire ja eriolukordade töötlus.

2.3.2 Juhtimine

Reaalajasüsteemides esineb kahte liiki juhtimist:

- superviisorjuhtimine tehnolooogiliste protesside juhtimises (*supervisory control*) tegeleb kaugema ja lähema perspektiiviga strateegiate väljatöötamisega ning realiseerimisega teiste süsteemi osade kaudu; näiteks, lähema perspektiiviga strateegia on kontrolleri seadesuuruste arvutamine, kaugema perspektiiviga strateegia on tehnoloogilise protsessi toimimise optimeerimisest tulenevad kitsendused ja/või soovitused üksikute kontrollerite seadesuuruste väärtustele;
- juhtimine (*direct digital control*), mis tegeleb strateegilistest eesmärkidest ja otsustest lähtuvate konkreetsemate tegevuste valiku ja realiseerimisega nii, et täiturite (actuators) kaudu sobivalt mõjutada juhitavat kobarat, rahuldades samas ka kõiki kitsendusi.

Muudes rakendusvaldkondades on superviisorjuhtimise asemel kasutusel terminid strateegiline otsustamine, strateegia väljatöötamine, eesmärkide ja/või alameesmärkide valik, jne. Autonoomse iseliikuva seadme puhul strateegia võib olla lõppeesmärgi ja vahe-eesmärkide valik; lähema perspektiiviga strateegia määrab ära navigeerimismeetodi lähima vahe-eesmärgini (kaardi järgi kasutades GPS-i (Geographical Positioning System), maastiku järgi kasutades aerofotosid jne).

Arvuti vaatepunktist on kõik juhtimisrakendused väga regulaarsed struktuurid – arvuti abil juhtimine koosneb (teoreetiliselt lõputult korduvast) juhtimisperioodide jadast. Iga jada element koosneb neljast, vahel üsna keerulisest, tegevusest – objekti

tegelikku seisundit kirjeldavate olekumuutujate väärtuste mõõtmisest, strateegilise otsuse tegemiseks vajaliku algoritmi täitmisest, taktikalise otsuse tegemiseks vajaliku algoritmi täitmisest, taktikalise otsuse rakendamine juhitavale objektile.

Juhtimisel võivad tekkida probleemid erinevate teooriate, nende arvutirealisatsioonide, ja tegelikkuse mittepiisava ühildumisega, mis viib tegeliku oleku väära hindamiseni, algoritmi tulemuste ebapiisava täpsuseni ja/või juhitava objekti, täiturite või muude süsteemi osade kitsenduste rikkumiseni.

Näiteks, sageli on juhitava objekti olekumuutujad käsitletavad ajas pidevalt muutuvatena; arvuti mõõdab neid väärtuseid diskreetsetel ajahetkedel. Mitmesugustel põhjustel ei pruugi Shannon/Nyquisti tingimused (vt allpool toodud kommentaari) täidetud olla ja sellisel juhul ei ole arvutisse jõudnud olekumuutujate väärtuste põhjal võimalik rekonstrueerida mõõdetud pidevat signaali – mis loomulikult põhjustab juhtimise kvaliteedi languse. Analoogiline probleem võib tekkida seadesuuruse väljastamisel. Lisaks võivad tekkida konfliktid erinevates teooriates kasutatud aja diskretiseerimise ühikute vahel, pidevas ajas toimiva teooria tulemuste ülekandmisel diskreetsesse aega, jne. Kõige viimases järjekorras nimetame arvutis realiseerimisel tekkivat diskretiseerimisühiku juhuslikku pikkust – lisaks muudele pahandustele võib juhusliku pikkusega ajaühik muuta teooria järgi stabiilse juhtimissüsteemi mittestabiilseks.

Kommentaar.
***Nyquist criterion**: To sample a bandlimited signal (with maximum frequency f_{max}) the sampling frequency f_s should be at least $2f_{max}$*

***Sampling theorem (Shannon).** A bandlimited function x with maximum frequency f_{max} sampled at **equidistantly spaced** points with frequency $\geq 2\, f_{max}$ is retrieved from its samples using the cardinal series.*

*Whittaker (1915) studied the problem of finding an analytic expression for a function, whose values at $(t_0 + n\Delta)$, $n \in Z$, are known. It is obvious that the sequence of points does not uniquely define a function. Whittaker therefore called all the functions through these points co-tabular, and the functions with the lowest harmonic constituents **cardinal functions**. This function can be defined as the sum of the cardinal series:*

$$\Sigma_{-\infty}^{+\infty} (x(t_0+n\Delta) \sin \pi/\Delta(t-t_0-n\Delta)) / \pi/\Delta(t-t_0-n\Delta)$$

*If the Nyquist criterion is not met, either by sampling a non-bandlimited signal, or by choosing ther sampling frequency too low (undersampling), the reconstruction process may yield a special kind of error, which is called **aliasing*** **-- s.o. diskreetsetel hetkedel tehtud mõõtmistest pideva funktsiooni rekonstrueerimisel saadakse nn. varifunktsioon.**

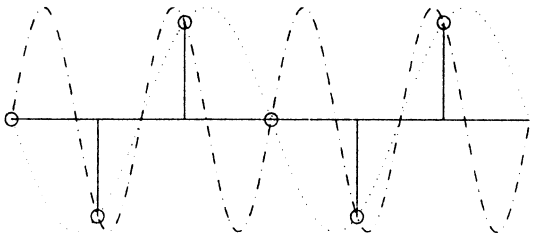


Figure 2.2 Undersampling a signal causes aliasing (©M.Van der Laan, 1995)

Marten D. van der Laan “Signal sampling techniques for data acquisition in process control” Ph.D. thesis University of Groningen, the Netherlands, 1995,186 pp.

E.T.Whittaker “On the functions which are represented by the expansions of the interpolation theory”, Proc. of the Royal Society of Edinburgh, 1915, 181-194

2.3.3 Inimliides

Reaalajasüsteem peab informeerima operaatorit juhitava/jälgitava kobara hetkeolekust ja olekute muutumise ajaloost ning aitama operaatoril juhtida vajalikku objekti selles kobaras. Selleks on arvutisüsteemile tavaliselt ehitatud inimliides, mis võimaldab arvutiga suhelda inimesele sobilike meetodiga. Lisaks juhitava/jälgitava kobara seisundi kohta info esitamisele, peab inimliides tagama selle info kooskõlalisuse (loogilises ja ajalises mõttes), võimaldama esitatavat infot vastavalt vajadusele filtreerida, hõlpsat juurdepääsu otsuste tegemise abivahenditele (juhitava/jälgitava kobara mudelid, hüpoteeside kontrolli algoritmid, otsustusalgoritmid, jne). Inimene-operaator on suhteliselt konservatiivne, selletõttu paljud inimliidese dialoogivahendid erinevad oluliselt traditsiooniliselt arvutitehnikas kasutatavast (hiir, kuvar, sõrmistik). Ei ole harvad juhused, kus kombineeritakse traditsioonilisi arvutitehnika vahendeid traditsiooniliste rakendusvaldkonna vahenditega. Näiteks arvutiga lennukijuhtimine (fly-by-wire plane) näeb välja nagu traditsioonilise lennuki juhtimine, mõõteriistade paneel koosneb aga paljudest arvutikuvari tüüpi näiduga mõõteriistadest, relvade juhtimiseks kasutatakse multimeedia tüüpi liideseid.

Paljud ohutuskriitilistes reaalajasüsteemides arvutite kasutamisel tekkinud katasroofid on tekkinud mitte eriti läbimõeldult projekteeritud, hinna alandamiseks lihtsustatud, lihtsalt vigase, või ebaõnnestunud konstruktsiooniga inimliidese tõttu (detailsed näited ja palju muud huvitavat infot raamatust Leveson N.G. (1995) “Safeware: system safety and computers” Addison Wesley Company, Reading, Massachusetts)

Enamus protsessijuhtimise süsteeme sisaldavad oma inimliidese koosseisus juhitavat/jälgitavat kobarat iseloomustavate andmete ajaloo salvestamise ja aruandluse (*data logging and reporting*) alamsüsteemi, mis on projekteeritud vastava tööstuse nõuete kohaselt. Näiteks, farmaatsiatööstuses on paljudes maades kohustuslik iga üksiku toodetud ravimipartii eluloo säilitamine arhiivis (väga pika aja jooksul) – kui

kunagi peaks ilmnema probleeme selle ravimiga ja selle partiiga, saab kogu info arhiivist kätte.

Paljudes rakendustes on inimliides selline, et tavaline kasutaja ei oska selle taga arvutisüsteemi aimatagi. Näiteks, arvutid autos, ja lennukites (fly-by-wire airplane, drive-by wire car). Inimliidese projekteerimine ja realiseerimine on kujunenud omaette erialaks, koos oma käsiraamatutega. Näiteks Ebert R.E. (1994) "User Interface Design" Prentice Hall Inc., Englewood Cliffs, NJ.

Pisut fantastikat. Üks uuemaid suundi inimese ja arvuti suhtlemise parandamiseks on nn. bioloogiliste signaalide kasutamine arvuti töö juhtimiseks (bioliides – ingl.k. *neural interface*). Juba pikka aega kasutatakse inimkõnet arvuti juhtimiseks ja kehahoiakut ning liigutusi inimese identifitseerimiseks. Esimesed õnnestunud laborikatsed inimese närvisüsteemis ja lihasesüsteemis (human nervous and muscular systems) toimuvate protsesside abil arvuti juhtimiseks on toimunud USA-s. Rohkem detaile selle suuna kohta leidub artiklis J.Charles (1999) "Neural Interfaces Link the Mind and the Machine", IEEE Computer, vol.32, no.1, 16-18.

2.4 Alarmiseire ja eriolukordade töötlus

Eriolukordade töötluse alamsüsteem on näide, kuidas mittefunktsionaalsed nõuded võivad segi lüüa ainult funktsionaalsete nõuete alusel projekteeritud süsteemi. Vähegi nõudlikemates rakendusvaldkondades algab eriolukordade töötlus suuremast või väiksemast süsteemtarkvara spetsialiseerimisest – näiteks, sisend/väljund draiverite automaatne (s.t. universaalne) veatöötlus suunatakse operatsioonisüsteemi kaudu kasutaja programmi. Vastavalt rakenduse veatundlikkusele projekteeritakse ja realiseeritakse vigade kompenseerimise, või elimineerimise eest vastutavad rakendustarkvara (sageli koos spetsialiseeritud riistvaraga) osad.

Peetakse üsna tavaliseks, et reaalajatarkvara mittefunktsionaalsete nõuete rahuldamisega tegeleva koodiosa maht moodustab 60-90% kogu tarkvara mahust. Andmehõivega ja inimliidesega seoses kirjeldatud alarmiseire alamsüsteem moodustab eriolukordade töötluse süsteemist ühe osa – kõik, mis on seotud vigade automaatse kompenseerimisega, valesti tehtud arvutuste korrigeerimisega, tarkvara ja riistvara perioodilise diagnostikaga, tarkvara ja/või riistavara võimalik automaatne ümberkonfigureerimine, riknenud tark- või riistvara puhul süsteemi funktsionaalsuse muutmisega, ja muu sellise tegevusega jääb andmehõive alamsüsteemist ja inimliidesest välja.

Süsteemi projekteerimise seisukohalt on kasulik vaadata eriolukordade töötlust kui süsteemi struktureerimise tehnikat, mis võimaldab kapseldada vea avastamisega ja vea tõttu tekkinud väära käitumise kompenseerimisega seotud tegevusi. On päris loomulik, et selline struktureerimine peab algama kohe süsteemi loomise algusest, s.o. kasutajanõuete spetsifitseerimise etapist.

Andmetöötlussüsteemide tegemise pikaajalises praktikas on tihti (ja valdavalt) eriolukordade analüüs ja vastavad tegevused lisatud süsteemile testimise käigus,

pärast funktsionaalsete nõuete täitmist tagavate tegevuste kodeerimist. Tarkvaratehnika võitleb igati sellise praktika vastu. Selline praktika on kuidagi mõeldav andmetöötlus süsteemide tarkvara loomisel tänu selle suhtelisele triviaalsusele ja keerukamate kitsenduste puudumisele. Reaalajatarkvaras on selline praktika täiesti hukatuslik – kujutage ette, et te olete realiseerinud funktsionaalsed nõuded ja suutnud rahuldada ka etteantud ajakitsendused. Mis saab ajakitsendustest kui samale riistvaraplatvormile lisatakse 60-90% täiendavat koodi?

Igal süsteemi loomise elutsükli etapil keskendutakse konkreetsetele, sellel etapil tekkivatele ja/või lahendatavatele eriolukordadele. Kui tarkvara areneb edasi, lisanduvad uued detailid ja ilmnevad uued eriolukorrad, mida polnud võimalik ette näha eelnevates elutsükli etappides. Paljudel juhtudel saab uued eriolukorrad loogiliselt ja/või põhjuslikult ühendada varem kirjeldatutega – objekt-orienteeritud projekteerimis-metoodika korral tekivad nn. eriolukordade klassid, ning kogu projekti saab analüüsida vastuolude, täielikkuse ja potentsiaalsete konfliktide avastamiseks – nii nagu tavalist objekt-orienteeritud projekti.

Eriolukordade näiteid.

Kasutaja nõuete kirjeldamise etapil fikseeritakse potentsiaalset huvi pakkuvad vead juhitavas/jälgitavas kobaras, ühtlasi kirjeldatakse vigade kõrvaldamise teed või nende mõju kompenseerimise/vähendamise võimalused.

Projekteerimise etapil fikseeritakse potentsiaalset huvi pakkuvad vead rakendus-tarkvaras (abstraktse projekteerimise etapp), mis detailse projekteerimise etapil tükeldatakse edasi arhitektuuriga seotud või detailse loogilise projektiga seotud eriolukordadeks (tarkvara veakindluse poliitika väljatöötamine, robustsete (vigade suhtes tundetute) andmestruktuuride kasutamine, algoritmide veakindlust tagavad meetmed, jne) ja vigade kõrvaldamise või (osalise) kompenseerimise võimalused.

Realiseerimise etapp (mõnedes mudelites saab samasuguseid vigu käsitleda juba füüsilise projekteerimise etapil) -- fikseeritakse huvi pakkuvad vead konkreetses arvutisüsteemis ja nende kõrvaldamise või (osalise) kompenseerimise võimalused; näiteks, tarkvara realiseerimisvead, tarkvara täitmisel tekkida võivad vead, rakendustarkvara interaktsioonil operatsioonisüsteemiga või riistvaraga (kas vahetul suhtlemisel või läbi süsteemtarkvara) tekkida võivad vead.

Oletades, et tarkvara projekteerimine toimub objekt-orienteeritud keskkonnas võivad eriolukorra lahendamisel tekkida järgnevad võimalused:

- eriolukorra tekkimine, avastamine ja töötlemine on võimalik kapseldada ühte objekti;

- üks objekt ei ole võimeline lokaalselt (s.o. sama objekti sees) töötlema eriolukorda, eriolukorra töötlemine tuleb jagada mitme objekti vahel;

- mitut objekti haarava ühistegevuse käigus tekib mitu samaaegset eriolukorda; sellisel juhul tuleb kasutada nn. eriolukordade lahendajat (*exception resolution*), et leida “peamine” eriolukord millest alustada.

Täiendavaid detaile ja kasulikke ideid eriolukordade töötluse kohta võib leida

Cristian F. (1995) “Exception handling and tolerance of software faults”, in Software Fault Tolerance, ed. M.Lyu, J.Wiley&Sons, 81-107

Miller R. and A.Tripathi (1997) “Issues with Exception Handling in Object-Oriented Systems”, Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP’97), Lecture Notes in Computer Science # 1241, eds. M.Aksit, S.Matsuoka, Springer-Verlag, Berlin, Germany, 85 - 103

Kaasaegse inimliidese, alarmiseire ja eriolukordade töötlust on populaarselt selgitatud

Dicken C.R. (1999) “”Soft” control desks and alarm display” , IEE Computing and Control Engineering Journal, vol, 10, no. 1, 11-16.

2.5 Ajakitsendused reaalajasüsteemis

Ajalooliselt viidi ajakitsendused reaalajasüsteemidesse sisse vigade ja katsetuste meetodil, selleks, et tagada nende toimimine kooskõlas juhitava/jälgitava kobara vajadustega. Alles palju hiljem (peaaegu 30 aastat hiljem) tekkisid ka teoreetilised põhjendused paljudele, empiiriliselt sisseviidud ajakitsendustele. Tuletame meelde, et reaalajasüsteem koosneb mitmest koostoimivast (s.t. interaktsioonis olevast) dünaamilisest süsteemist ja arvutisüsteemi roll reaalajasüsteemis on, lisaks juhtimis- ja otsustustegevuse toetamisele, garanteerida erinevate dünaamiliste süsteemide vahelise liidese nõutav funktsioneerimine. Kui eelöeldut veidi lahti mõtestada, võib ajakitsenduste sisseviimise peamiste põhjustena loetleda:

- Algoritmiteooria kehtivuse tagamine tarkavara töö käigus – s.t. algoritmi aksiomaatilise baasi staatilisuse tagamine algoritmi täitmise ajal
- Algoritmis kasutatavate lähteandmete ajalise korrektsuse tagamine (kasutatavad muutujate väärtused peavad olema oma kehtivuspiirkonnas)
- Algoritmi tulemuste õigeaegne (kehtivuspiirkonna sees) kasutamine (inimesele-operaatorile antavates sõnumites ja juhitava/jälgitava kobara funktsioneerimise mõjutamiseks)
- Erinevates dünaamilistes süsteemides kasutatavate teooriate koostoimimise eelduste kontroll ja vastastikuse sobivuse tagamine
- Sundparalleelsusest tulenevate ümbritseva keskkonna nõuete täitmise vajaduse (jõudlushinnangud ja õigeaegsed protsesside interaktsioonid) rahuldamine
- Mittetäielikult teadaolevate põhjuslike seoste aproksimeerimine

Kõik need erinevad ajakitsendused tekivad juhitava/jälgitava kobara ja tema osade toimimise analüüsist, reaalajasüsteemi ehitamise eesmärgi lahtimõtestamisest, analüüsil ja modelleerimisel kasutatud teooriate omadustest ning eeldustest (eriti ilmutamata kujul tehtud eeldustest), inim-operaatori füsioloogilistest ja

psühholoogilistest omadustes ja kasutatavate algoritmide ning arvutisüsteemi omadustest.

Asi tundub olema üsna keeruline – ja teooria poole pealt ongi – kuid mõningase vaevaga õnnestub kogu ülaltoodud loetelu esitada üsna väikese arvu ajaparameetritega. Seda teemat käsitletakse lähemalt kursuses LAP 5712 “Tarkvara dünaamika” (TTÜ-s loetakse kevadsemestril). Reaalajasüsteemides vajalikest ajalistest nõuetest, nende allikatest ja ajalise korrektsuse formaalsest tõestamisest saab põhjaliku ülevaate raamatust Motus and Rodd “Timing analysis of real-time software”, Elsevier Science Publishing/Pergamon, Oxford, 1994.

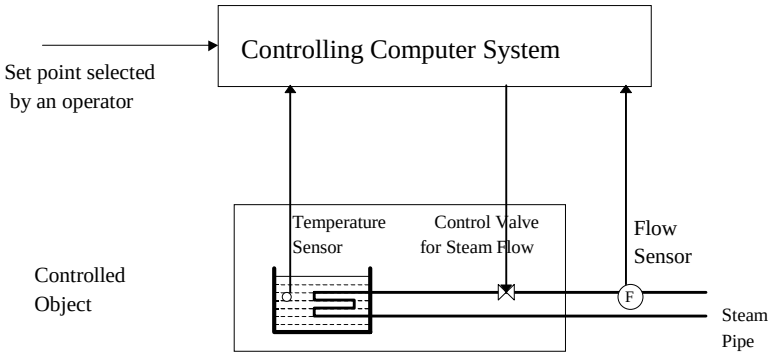
Käeolevas alajaotuses esitatakse näiteid väliskeskkonnast tulenevatest ajakitsendustest, mida tuleb arvestada tarkvarasüsteemide projekteerimisel. Toodud näited pärinevad automaatjuhtimisteooriast ja illustreerivad juhtimisteooria ning tarkvara projekteerimise ja realiseerimise sarnasust. Vastupidiselt klassikalises arvutiteaduses levinud seisukohale, et pärast seda kui algoritmid on fikseeritud ja programmeerimisülesanne antud, on programmisti tegevus sõltumatu teiste alade spetsialistide tööst, eeldab reaalajasüsteemide tarkvara tegemine kogu süsteemiprojekti meeskonna tihedat koostööd. Põhjuseks on jällegi, et me ei ehita mudeli mudelit, vaid reaalse maailma osa.

Ajakitsendused esinevad kõikides reaalajasüsteemide rakendustes ja ei ole tingimata seotud automaatjuhtimissüsteemidega. See valdkond on lihtsalt hästi läbiuuritud ning sealt on lihtne tuua näiteid. On loomulik oletada, et kõige rangemad ajakitsendused saavad alguse tagasidestatud juhtimisahelate nõuetest – näiteks kiire mehhaaniline protsess (automootori juhtimine), oma loomulikus seisundis (s.t. ilma välise juhtimiseta) mittestabiilne keemiline reaktsioon, aerodünaamiliselt mittestabiilse lennuki õhus hoidmine. Ajakitsendused tulenevad ka inimliidesest, kuid need on suhteliselt lõdvad, tingituna inimese intelligentsist johtuvast tolerantsusest ja kohanemisvõimest. Paljudes inimliidestes tuleb siiski arvestada üsna rangeid ajakitsendusi – seoses operaatori võimaliku stressiga ei ole lubatud eeldada tema intelligentsil baseeruvat kohanemisvõimet.

Käesolevast kursusest, nagu ka toodud näidetest, jäävad välja ajakitsendused, mis on tingitud erinevate teooriate koostöö tagamise nõuetest, diskreetses ja pidevas ajas toimivate teooriate koostöö vajadusest, erinevate algoritmide interaktsiooni ajalistest nõuetest arvutis jms.

2.5.1 Näide ajakitsenduste mõnedest allikatest.

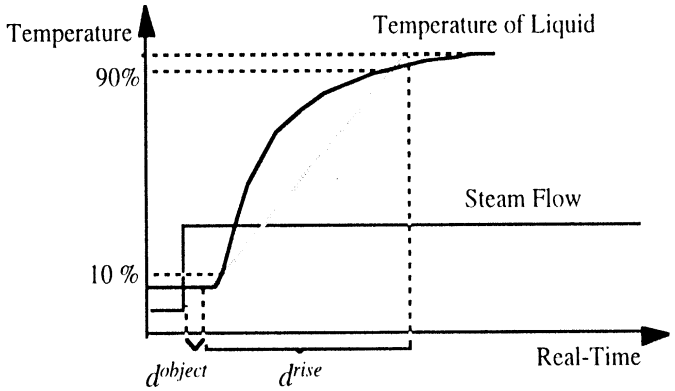
Vaatame lihtsat soojavahetit (joonis 2.3), kus auruventiili juhtimisega püütakse soojavahetis oleva vedeliku temperatuur hoida operaatori poolt etteantud tasemel. Vaikimisi eeldatakse, et auru temperatuur ja rõhk on konstantsed.



Joonis 2.3 Lihtne kontroller temperatuuri hoidmiseks soojavahetis.
(© Kopetz (1997))

Juhitava objekti (soojavaheti) ajaparameetrid.

Oletame, et soojavaheti on tasakaaluasendis. Kui auruvoogu suurendada hüppeliselt, siis vedeliku temperatuur muutub vastavalt joonisel 2.4 toodud graafikule kuni saavutatakse uus tasakaaluasend. Sellist graafikut nimetatakse juhtimisteoorias siirdefunktsiooniks (vahel ka kajafunktsiooniks, inglise k. *response function*, *step response function*). Juhime tähelepanu tüüpiliselt idealistlikele eeldustele teoorias – auruvoogu ei saa (ja ei tohi) praktikas hüppeliselt suurendada -- sellised meetodid ja eeldused on kasutatavad kui tegelikkuses toimuvate protsesside aproksimatsioonid ja täiesti omal kohal nii kaua kui me jääme mudelite maailma. Üleminekul tegelikku maailma tuleb hoolega jälgida taoliste eelduste paikapidavust ja võimalikku mõju tegelikus maailmas.



Joonis 2.4 Soojavaheti kontrolleri siirdefunktsiooni (hüppekaja) viited ja tõusuaeg.
(© Kopetz (1997))

Temperatuuri siirdefunktsioon boileris (soojavahetis) sõltub vedeliku kogusest, auruvoo intensiivsusest (ja ka auru temperatuurist). Siirdefunktsiooni iseloomustavad kaks ajaparameetrit:

- objekti viide (*object delay, d^{object}*), aeg mis kulub auruvoo hüppelisest suurenemisest kuni temperatuuri kasvu registreeritud alguseni. Objekti viide on omakorda seotud juhitava protsessi inertsiga (protsessi hilistumine, *process lag*)
- tõusu aeg (*rise time, d^{rise}*), mis kulub soojavahetis oleva vee temperatuuri uue tasakaaluasendini jõudmiseks.

Selleks, et objekti viidet ja tõusu aega määrata siirdefunktsiooni graafikult (vt. joon. 2.4), on välja töötatud järgmine metoodika:

- leitakse graafikul kaks punkti, kus siirdefunktsioon on saavutanud vastavalt 10% ja 90% vana ja uue tasakaaluasendi erinevusest (vt. joonis 2.4)
- ühendatakse need punktid sirglõiguga, pikendatakse sirglõiku kuni lõikumiseni tasakaaluasendeid tähistavate sirgetega
- lõikepunktid tasakaaluasendit tähistavate sirgetega määravad d^{object} ja d^{rise}.

Arvutisüsteemi (soojavaheti kontrolleri) kohta käivad ajanõuded

Arvutisüsteem peab perioodiliselt mõõtma vedeliku temperatuuri boileris ja võrdlema temperatuuri mõõdetud väärtust soovitud väärtusega. Kahe mõõtmise vahelist ajaintervalli nimetatakse mõõtmisperioodiks (d^{sample}) ja tema pöördväärtus on mõõtmise sagedus (f^{sample}). Diskreetses ajas toimivas süsteemis (digitaalne süsteem) kehtib empiiriline reegel – selleks, et digitaalne süsteem käituks sarnaselt pidevas ajas töötava süsteemiga, peab mõõtmisperiood olema väiksem kui üks kümnendik siirdefunktsiooni tõusuaega (d^{rise}) (vt. ka kommentaar alajaotuses 2.3.2).

Märkus: Empiiriline reegel põhineb Nyquisti kriteeriumil ja Shannoni teoreemil pideva signaali mõõtmistest saadud diskreetse ajajada põhjal mõõdetud signaali kuju täpse taastamise võimalusest. Viimasel ajal on aproksimeerimisteooriat oluliselt arendatud, millega seoses on ka tema kasutusvaldkond laienenud. Näiteks tihti mustkunstina esitatavad närvivõrgud on teoreetiliselt korralikult põhjendatavad aproksimatsiooni-teooriat kasutades, oluliselt on paranenud arusaamine mittelineaarsete funktsioonide lähendamisest ortogonaalsete baasfunktsioonidega.

Mõõtmise tulemusena saadud väärtuse ja soovitud väärtuse vahe alusel arvutab juhtimisalgoritm uue juhttoime väärtuse. Intervall, mis kulub mõõtmise hetkest (*sampling point*) kuni uue juhttoime väärtuse väljastamiseni auruvoogu reguleerivale klapile, on arvuti viide (*computer delay*, $d^{computer}$). Intuitiivselt loomulik nõue on, et $d^{computer}$ peab olema väiksem kui mõõtmisperiood d^{sample}.

Erinevust minimaalse ja maksimaalse arvuti viite väärtuse vahel nimetatakse fluktuatsiooniks (*jitter*). Fluktuatsiooni suurusel on oluline mõju juhtimise kvaliteedile. Juhtimisteoorias arvestatakse tavaliselt ainult arvuti viite fluktuatsiooni, praktiliselt on sageli isegi tõsisem probleem seotud mõõtmisperioodi fluktuatsiooniga (ja üldse arvutis kasutusel oleva ajaühiku pikkuse fluktuatsiooniga). Enamus algoritme toimib diskreetses ajas, kus eelduseks on ajaühiku konstantne suurus. Ajaühiku konstantsust on arvutis realiseerimisel väga raske tagada, mistõttu ka paljud teooriad käituvad arvutirealisatsioonis teisiti kui paberil.

Avatud juhtimisahela tundetuse aeg (dead time) on intervall mõõtmise hetkest kuni juhitava objekti esimese reaktsioonini, mis on põhjustatud arvuti poolt mõõtmistulemuse alusel välja arvutatud, täiturile edastatud ja täideviidud tegevusest.

Tähis	Nimetus	Kuuluvus	Suhted teistega
d^{object}	controlled object delay,	controlled object,	physical process,

	juhitava objekti viide	juhitav objekt	füüsikaline protsess
d^{rise}	rise time of step response siirdefunktsiooni tõusuaeg	controlled object, juhitav objekt	physical process füüsikaline protsess
d^{sample}	sampling period mõõteperiood	computer arvuti	$d^{sample} << d^{rise}$
$d^{computer}$	computer delay arvuti viide	computer arvuti + algoritm	$d^{computer} < d^{sample}$
$\Delta d^{computer}$	jitter of the computer delay arvuti viite fluktuatsioon	computer arvuti	$\Delta d^{computer} << d^{computer}$
$d^{deadtime}$	dead time tundetusaeg	computer and controlled object arvuti ja juhitav objekt	$d^{computer} + d^{object}$

Kuna tegemist on äärmiselt lihtsa näitega, on ka selles kirjeldatud ajaparameetrid (suhteliselt) lihtsalt määratavad. Sellegipoolest on ülaltoodud näites tehtud olulisi lihtsustusi – mõõtmisperiood on eeldatud konstantseks ja veel mitmetele parameetritele on eeldatud determineeritud väärtust. Näiteks tundetusaeg (*dead time*), arvutiviide (computer delay). Konstantne mõõtmisperiood on teoreetiliselt vajalik eeldus, mida arvutis (eriti hajussüsteemis) on praktiliselt lootusetu realiseerida. Praegune juhtimisteooria on võimeline kompenseerima ajaühikute fluktuatsioone tingimusel, et nad on kaduvväikesed (s.t. neid võib lugeda konstantseks).

Praktikas nii ideaalset juhtu ei esine. Näiliselt arusaamatu juhtimise kvaliteedi langus pärast algoritmi realiseerimist arvutis – mille peamiseks põhjustajaks on enamusel juhtudest arvutis kasutatav mitteregulaarne ajaühik -- on sagedane lahkhelide allikas tarkvarainseneri ja juhtimisspetsialistide vahel. Juhtimisteooria ei suuda veel arvestada reaalselt võimalikku pideva maailma kirjeldamise täpsust arvutis (kuna see probleem tekkis alles suhteliselt hiljuti (1970-ndad)) ja arvutispetsid ei oska kuigi põhjalikult ennustada ja/või analüüsida nende poolt projekteeritud programmisüsteemi ajalist käitumist; nõutud ajalise käitumise garanteerimiseks tarvilikud praktilised vahendid on ikka veel tegemata, või üldsuse poolt omaks võtmata. Viimase kümne aasta (1987-1997) peamiseks saavutuseks maailmas sellel alal tuleb lugeda ajakitsenduste mõju tunnistamist süsteemi käitumisele ja ajakitsendustest tingitud vigadega seotud probleemi olemasolu laialdast tunnustamist ning üksikute uurimisgruppide suhteliselt põhjalikke uuringuid selles valdkonnas.

Muude parameetrite fluktuatsioonid (jitter) avaldavad mõju nende parameetrite toime hindamise meetodite valikule. On valida kas kasutada süsteemi ajalise käitumise korrektsuse hinnangu leidmiseks intervallarvutust, tõenäosusarvutust, või mõnda muud meetodit.

Kaasajal ei tohiks enam kasutada sisuvabu mõisteid “reaalaja nõuded” või “reaalaja kitsendused”. Nii abstraktsel tasemel mõisted pärinevad reaalajasüsteemide varasest lapsepõlvest ja nende kasutamine näitab asjatundmatust või asjast mittehoolimist. Tuleks kasutada märksa konkreetsemaid väljendeid. Näiteks, on suhteliselt laialt aktsepteeritud kolme erinevat tüüpi ajakitsenduste olemasolu ja oleks kena teatada, millisega neist (või kõigiga) tegeletakse:

- jõudluskitsendused – näiteks, käivitusperiood, töö kestus, piir-ajad, kitsendused sündmuste järjekorrale; need kitsendused on kasutusel ka planeerimisteoorias (*scheduling theory*) programmide täitmise planeerimiseks arvutis
- kitsendused protsesside vahelisele interaktsioonile – näiteks, interaktsiooni alustamise hetk, protsesside töö sünkroniseerimise rezhiim (sünkroonne, poolsünkroonne, asünkroonne), protsesside, sündmuste, tegevuste, sünkroniseerimise täpsus
- andmete ja sündmuste kehtivuse intervallid – näiteks, andmete kehtivuse intervall, sündmuste ekvivalentsuse intervall, sündmuste samaaegsuse intervall

Erinevad arvutusmudelid ja verifitseerimise meetodid suudavad tõestada erinevate ajakitsenduste rahuldamist (või rikkumist) süsteemis. Seetõttu tuleb erilise umbusuga suhtuda meetoditesse/tööriistadesse, mis tegelevad lihtsalt reaalajanõuete täitmise kontrollimisega ilma nõuete tüüpe täpsustamata.

2.6 Mittefunktsionaalsed nõuded reaalajasüsteemidele

Mittefunktsionaalsed nõuded kirjeldavad kasutaja soove loodava reaalajasüsteemi töö kvaliteedi kohta süsteemi kasutusfaasis (ingl. k. *quality of service*). Kasutaja võib olla inimene, või mõni tehniline süsteem. Inglisekeelses kirjanduses on kasutusel termin **non-functional requirements**, ohutuskriitilistes rakendustes kasutatakse sageli terminit **dependability requirements** samade nõuete tähistamiseks.

Vaatamata sellele, et mittefunktsionaalsetest nõuetest hakati rääkima alles hiljuti ja, et arvutiteadus on traditsiooniliselt tegelenud ainult funktsionaalsete nõuete rahuldamisega, on tegelikus elus mittefunktsionaalsed nõuded mitte vähem tähtsad. Ohutuskriitilistes rakendustes võib olla isegi pisut tähtsamad.

Tarkvaratehnika soovitab tungivalt, et mittefunktsionaalsed nõuded saaksid fikseeritud juba kasutaja nõuete dokumendis. Ka on oluline neid arvestada alates süsteemi elutsükli esimestest etappidest. Võib tunduda uskumatu, aga protsessijuhtimis-süsteemides võib mittefunktsionaalsete nõuete realiseerimiseks kuluda kuni 90% kogu tarkvara koodimahust.

Ohutuskriitilistes rakendustes kasutatud terminitega tutvumiseks sobiv raamat on Laprie (1992):
Laprie J.C. (Editor) “Dependability: Basic Concepts and Terminology – in English, French, German, and Japanese” Springer-Verlag, Vienna, Austria, 1992**.**

2.6.1 Töökindlus.

Süsteemi töökindlus (*reliability*) R(t) on tõenäosus, et süsteem garanteerib ettenähtud teenused ajahetkest t_0 kuni ajahetkeni t -- eeldusel, et süsteem oli töökorras hetkel t_0. Kui eeldada süsteemil konstantset rikete tekkimise sagedust (*failure rate*), näiteks λ riket tunnis, siis on töökindlus arvutatav valemiga

$R(t) = \exp(-\lambda(t-t_0))$, kus $t-t_0$ on antud tundides.

Rikete tekkimise sageduse pöördväärtus $1/\lambda$ = MTTF on keskmine aeg järgmise rikkeni (*mean-time-to-failure*), antud tundides. Kui rikete tekkimise sagedus on suurusjärgus 10^{-9} siis on tegemist ülitöökindla süsteemiga (*ultrahigh reliability*).

2.6.2 Ohutus

Ohutus (*safety*) on töökindluse eriliik, kus arvestatakse ainult raskete tagajärgedega riketega (*critical failure modes*). Raskete tagajärgedega rike (kriitiline rike) on pahaloomuline (*malign failure*) erinevalt tavalistest riketest (mittekriitilistest riketest), mis on healoomulised (*benign failure*). Pahaloomuliste rikete maksumus on tavaliselt mitu suurusjärku suurem kui süsteemi normaalse töö ajal toodetav kasum.

Pahaloomuliste rikete näiteid – lennuki hävimine juhtimissüsteemi vea tõttu, autoavarii intelligentse pidurisüsteemi vea tõttu. Ohutuskriitilistes reaalajasüsteemides (rangete ajakitsendustega reaalajasüsteemid, hard real-time systems) on rikete tekkimise sageduse nõuded samad, mis ülitöökindlates süsteemides (*systems with ultrahigh reliability*) -- s.t. vähem kui 10^{-9} viga tunnis.

Näiteks – eeldusel, et autot kasutatakse keskmiselt üks tund päevas vastab üks pahaloomuline rike miljoni auto kohta aastas rikete tekkimise kiirusele 10^{-9} riket tunnis (Kopetz, 1997).

Enne ohutuskriitiliste süsteemide normaalsesse töösse lubamist on paljudes maades kohustuslik **Litsenseerimine/Sertifitseerimine**. Litsenseerimist/Sertifitseerimist teevad süsteemi tootjatest ja/või väljatöötajatest sõltumatud ettevõtted – näiteks Saksamaal kuuluvad need ettevõtted riigile (TüV), USA-s teevad seda tööd eraettevõtted. Erinevad on ka süsteemi tööpõhimõtted – Saksamaal on sertifitseerimine kohustuslik (enne ei tohi süsteemi kasutusele võtta), USA-s on küsimus taandatud sertifitseerimata süsteemi poolt potentsiaalselt tekitatud kahjude tasuja määramisele. Kui süsteem on sertifitseerimata katab kahjud tootja firma, kui sertifitseeritud siis sertifitseeriv firma.

Ranges reaalajas töötava süsteemi **sertifitseerimisel kontrollitakse** järgmisi süsteemi omadusi:

- kas ohutu töö seisukohalt kriitilised alamsüsteemid on muust süsteemist isoleeritud liidestega, mis takistavad vigade sissetungi muust süsteemist ja rikkis alamsüsteemist vigade levimist ülejäänud süsteemi (ei tohi esineda tarkvara mürgituse võimalust, ingl.k. *software poisoning* või *system poisoning*)
- kas kõigile koormus- ja veahüpoteesidele vastavad stsenaariumid on läbimängitud/läbimängitavad ja kas tulemused vastavad süsteemi spetsifikatsioonile; stsenaariumite läbimängimisel ei tohi kasutada tõenäosuslikke argumente; selle nõude täitmiseks on vaja staatiliselt jagatud ressursse (ohutus-kriitilistes alamsüsteemides on reeglina keelatud dünaamiline ressursside jagamine);

- süsteemi arhitektuur toetab sertifitseerimisprotsessi – s.t. alamsüsteemide sertifitseerimine saab toimuda üksteisest sõltumata; see eeldab, et iga alamsüsteem on piisavalt autonoomne, omab isiklikku sihifunktsiooni ja on võimeline ennustama oma käitumist tulevikus.

Süsteem, mis on sobiv sertifitseerimiseks, peab olema projekteeritud nii, et:
- tema kohta on võimalik koostada täielik ja täpne töökindluse mudel; kõik mudeli parameetrid, mida ei saa tuletada analüütiliselt peavad mõistliku aja ja vähese vaevaga olema mõõdetavad süsteemil tehtud katsetest;
- töökindluse mudel ei tohi sisaldada projekteerimise vigu kirjeldavat olekumudelit (s.t. lõplik automaat, Petri võrk, jne); töökindluse mudel peab võimaldama analüütiliste argumentidega demonstreerida, et võimalikud projekteerimisel tehtud vead ei takista süsteemil funktsionaalsete nõuete täitmist;
- projekteerimisel tehtud otsused peavad eelistama lahendusi, mis minimiseerivad töökindluse mudeli uurimiseks vajalike mõõdetavate parameetrite arvu ja lihtsustavad analüütilist arutelu.

2.6.3 Hooldatavus

Hooldatavust (*maintainability*) iseloomustab aega, mis on vajalik süsteemi töökorda viimiseks pärast pahaloomulist viga (*malign failure*). Hooldatavust mõõdetakse tõenäosusega M(d), et süsteem taastatakse ajaintervalli d jooksul pärast pahaloomulist viga. Vastavalt töökindluse teooria traditsioonidele tuuakse sisse parandamise sagedus μ (parandusi tunnis) ja keskmine parandusaeg (MTTR) – mean-time-to repair – kui hooldatavuse kvantitatiivsed karakteristikud.

Töökindluse ja hooldatavuse vahel on põhimõtteline vastuolu. Hooldatav süsteem eeldab tema jagamist vähimateks asendatavateks ühikuteks (*smallest replaceable units*), mis on omavahel ühendatud hooldatavate liidestega (*serviceable interfaces*). Hooldatav liides on näiteks pistikühendus jootekoha asemel. On selge, et pistikühenduse töökindlus on tunduvalt väiksem kui jootekohal või keerutatud ühendusel. Lisaks on hooldatav liides kallim. Need argumendid on põhjuseks, miks paljudes masstoodetes on eelistatud töökindlust hooldatavusele. Tulemuseks võib olla olukord, et odavam on seade ära visata kui teda parandada.

2.6.4 Valmisolek

Valmisolek (*availability*) on ettenähtud teenuse osutamise valmiduse mõõt. Valmisolek näitab suhtelistes ühikutes ajaosa mille jooksul süsteem on võimeline tagama spetsifitseeritud teenuse täies mahus. Näiteks telefoni keskjaam peab piisavalt kõrge tõenäosusega olema valmis korrektselt valitud ühenduse loomiseks ükskõik millal kasutaja selleks soovi avaldab. Keskjaama lubatud mitte-valmisoleku aeg aasta jooksul ei tohiks olla mitte üle mõne minuti.

Süsteemis, kus rikete tekkimise sagedus ja nende parandamise sagedus on konstantne, saab valmisoleku mõõdu (A) arvutada järgmise valemi järgi:

$$A = MTTF/(MTTF + MTTR).$$

Summat MTTF + MTTR nimetatakse ka keskmiseks vigadevaheliseks ajaks (*mean-time-between-failures, MTBF*)

2.6.5 Turvalisus

Turvalisus (*security*) iseloomustab süsteemi omadust takistada mittevajalike isikute juurdepääsu süsteemis salvestatud ja/või liikuvale informatsioonile ning süsteemi poolt osutatavatele teenustele. Seni puudub standardse sissemurdja (*standard burglar*) definitsioon, mille abil saaks arvutada süsteemi sissemurdmise keskmise aja. Sellest tingituna puuduvad ka üldtunnustatud kvantitatiivsed mõõdud süsteemide turvalisuse kohta.

Turvaprobleemide näiteid reaalajasüsteemides – krüptograafiline vargakaitse süsteem autos, mis blokeerib mootori käivitamise kui autojuht ei suuda esitada varem kokku lepitud juurdepääsu koodi. Analoogilisi näiteid võib tuua pangasüsteemidest. Turvalisus on üks käesoleva aja kuumi teemasid. Valdkonna stabiliseerumine võtab veel aega, mida omakorda pikendab kogu temaatika tundlikkus – paljud selle teema alased uuringud ja tulemused on salastatud või kättesaadavad kitsale ringile.

2.7 Reaalajasüsteemide klassifikatsioon

Klassifitseerida on võimalik lõpmata paljude kriteeriumite järgi, selles alljaotuses tuuakse mõned näited reaalajasüsteemide puhul pragmaatilist huvi pakkuvatest klassifitseerimisprintsiipidest.

Arvutikobarat ümbritseva keskkonna poolt vaadates on esile tõusnud kaks võimalikku printsiipi:

- ranges reaalajas või nõrgas reaalajas töötavad süsteemid (*hard real-time versus soft real-time systems*)
- ohutult riknevad või töövõimet säilitavalt riknevad süsteemid (*fail-safe versus fail-operational real-time systems*)

Arvutikobara sisemistest omadustest ja projekteerimisel tehtavatest otsustest lähtudes on huvipakkuvad kolm klassifitseerimisprintsiipi:

- garanteeritud ajakitsendustega või parimate võimalike ajakitsendustega süsteemid (*guaranteed-timeliness versus best-effort real-time systems*)
- piisavate ressurssidega või mittepiisavate ressurssidega süsteemid (*resource-adequate versus resource-inadequate real-time systems*)
- sündmuste poolt juhitud või aja poolt juhitud süsteemid (*event-triggered versus time-triggered real-time systems*)

Ülaltoodud klassifitseerimisprintsiipe on kasutatud, ja osaliselt ka selgitatud raamatus Kopetz (1997).

2.7.1 Ranges reaalajas või nõrgas reaalajas töötavad süsteemid

Süsteemi, mis peab tulemused väljastama täpselt ettenähtud ajaks (s.t. ranges reaalajas töötava süsteemi, *hard-real-time system*), projekteerimine erineb oluliselt süsteemi projekteerimisest, millel puuduvad tulemuse väljastamiseks ranged piirangud (nõrgas reaalajas töötav süsteem, *soft real-time system, on-line system, transaction pocessing system*). Erinevused tulenevad:

- kasutajanõuete detailsusest,
- paljudest kvantitatiivsetest ja kvalitatiivsetest kitsendustest, mida tulevane süsteem peab täitma,
- valmis süsteemi käitumise kohustuslikust determineeritusest, vaatamata lähteinformatsiooni mittedetermineeritusele ja ainult osaliselt teadaolevatele põhjuslikele seostele
- testimise absoluutsest mittepiisavusest; enamus omadusi peab olema formaalselt tõestatud, mis omakorda eeldab suhteliselt formaalseid projekteerimis- ja analüüsimeetodeid
- suhteliselt suure riskiga rakendusvaldkonnad, mis õigustavad kallimate süsteemiloomisvahendite kasutamist

Järgnevas tabelis 2.1 on võrreldud ranges ja nõrgas reaalajas töötavate süsteemide mõningaid omadusi. Tabelis kasutatud mõistete selgitus on antud alltoodud kommentaarides.

Kosteaeg (*response time*) – aeg sündmuse tekkimisest (tavaliselt ümbritsevas keskkonnas, aga võib olla ka arvutis) kuni sellele reaktsiooniks valitud tegevuse täideviimiseni (tavaliselt arvuti, aga vahel ka operaatori poolt). Ranges reaalajas süsteemide korral tuleb ettenähtud kosteajast kindlasti kinni pidada, nõrgas reaalajas on oluline keskmise kosteaja garanteerimine.

Tabel 2.1 Ranges ja nõrgas reaalajas töötavad süsteemid (© Kopetz, 1997)

Näitaja **Characteristics**	Range reaalaja süsteem Hard real-time	Nõrga reaalaja süsteem soft real-time, on-line
kosteaeg (response time)	rangelt nõutud (hard - required)	soovituslik (soft-desired)
jõudlus tippkoormusel (peak-load performance)	ennustatav (predictable)	väheneb määramatult (degraded)
töökiiruse juhtimine (control of pace)	ümbritsevast keskkonnast (environment)	arvutist (computer)
ohutus (safety)	sageli kriitilise tähtsusega (often critical)	enamasti mittekriitiline (non-critical)
andmemahud (size of data files)	väikesed või keskmised (small/medium)	üldiselt suured (usually large)
liiasuse tüüp (redundancy type)	kuum reserv vajalik (active)	taastamine kontrollpunkti-ni (check-point recovery)
andemete terviklikkus (data integrity)	lühiajaliselt nõutav (short term)	pikaajaliselt nõutav (long term)

vigade avastamine (error detection)	autonoomne, automaatne (autonomous)	kasutaja abiga (user assisted)

Jõudlus tippkoormusel (*peak-load performance*) – tippkoormus tekib tavaliselt alarmilaviini korral (palju eriolukorra töötlemise programme töös, liigub palju alarmi sõnumeid, on vaja teha ja täide viia palju juhtimisotsuseid). Niisugused olukorrad tuleb vastavate stsenaariumitega ette näha ja hinnata tekkivat tippkoormust kvantitatiivselt. Ranges reaalajas töötavate süsteemide jõudlus tuleb valida selline, et ka tippkoormuse puhul kõik piir-ajad on garanteeritult rahuldatud. Nõrgas reaalajas on oluline keskmine jõudlus, mis tähendab, et tippkoormuse ajal võib vähemtähtsaid töid edasi lükata ja tähtsate tööde tegemise aeg võib veidi pikemaks venida – enamasti on keskmise jõudluse tagamine majanduslikult palju odavam. Keskmise jõudluse tagamisel võib kannatada süsteemi funktsionaalsete nõuete täitmise kvaliteet.

Töökiiruse juhtimine (*control of pace*) – ranges reaalajas on kogu süsteemi töökiirus määratud ümbritseva keskkonna dünaamiliste omadustega (arvuti töö peab olema "sünkroniseeritud" väliskeskkonna evolutsiooni ja käitumisega); nõrgas reaalajas töötavate süsteemide korral eeldatakse, et arvutil on ümbritseva keskkonna mõjutamiseks suhteliselt tugevad võimalused (näiteks, lennupiletite reserveerimise süsteem, või pangasüsteem – kui tellimusi tuleb liiga sageli, venib süsteemi kosteaeg pikemaks ja kliendid peavad kauem ootama), või ümbritsev keskkond on vähenõudlik.

Ohutus (*safety*) – range reaalaja süsteem seab palju kitsendusi süsteemi projekteerijale; näiteks peab vigade avastamine töötama sõltumatult muudest süsteemi osadest, vea avastamise korral tuleb viga kompenseeriv tegevus käivitada kindlasti ümbritseva keskkonna poolt määratud ajaintervalli sees. Nõrgas reaalajas töötavate süsteemide korral seda laadi ohutuse probleemi enamasti ei esine (tänu venivale kosteajale). Tõsise ohu korral peab püüdma süsteemi viia ohutusse olekusse (kui see on olemas), või püüdma süsteemi funktsionaalsust vähendada samm-haaval (et vältida suuremat õnnetust).

Andmemahud (*size of data files*) – andmed ranges reaalaja süsteemis kirjeldavad ümbritseva keskkonna olekumuutujate väärtuseid ajas, alarmisõnumeid ja reageerimist neile, ning soovitatud ja realiseeritud juhttoimeid. Tegemist on suhteliselt lühikeste andmekogumitega, mis peavad aga tingimata olema varustatud ajalipikutega. Kesksel kohal on kaks probleemi – primaarne on andmete ajaline kokkusobivus ja sekundaarne on geograafiliselt hajutatud (ja sageli dubleeritud) andmekogumite ajalise ja loogilise kooskõla säilitamine. Need probleemid on sageli koondatud reaalaja hajusandmebaaside *(distributed real-time data bases)* uurimistemaatika alla. Enamasti on andmete ajalise terviklikkuse säilitamine oluline suhteliselt lühikese aja jooksul (minuti, tunnid, päevad, nädalad, kuud).

Nõrgas reaalajas töötavate süsteemide puhul on võtmeprobleemiks suurte andmekogumite kooskõlalisuse säilitamine pika aja jooksul (aastad) – näiteks elanike registrid, pankade andmebaasid, farmaatsiatehaste toodangu partiide andmebaasid.

Liiasuse tüüp (*redundancy type*) – pärast nõrgas reaalajas töötava süsteemi vea avastamist algavad arvutused uuesti kontrollpunktist (milles on salvestatud täpne ja garanteeritult õige süsteemi olek), s.t. käivitatakse spetsiaalne taastamisprotseduur. Ranges reaalajasüsteemis on sellise tagasivõtmise võimalus väga harva esinev juhus, kuna primaarne tööde põhjustaja/algataja on ümbritsev keskkond. Füüsikalistel ja bioloogilistel põhjustel on enamus keskkondades toimuvatest protsessidest mittepööratavad:

- keskkonna protsessid on (vähemalt reaalajasüsteemi arvuti poolt) mittepööratavad
- isegi kui mõned nendest protsessidest oleksid pööratavad, on sageli võimatu garanteerida kosteajast kinnipidamist – soovitud tulemus jääks kindlasti saavutamata, halvemal juhul võib juhtuda avarii, või katastroof, või õnnetus
- kosteaja (või muude ajakitsenduste) rikkumisel on karta, et kasutatavate andmete ajaline kehtivus on lõppenud, nende kasutamine (poolelijäänud arvutuste taasalustamiseks) võib kasu asemel kahju (tihti väga suurt kahju) tuua.

Sellest tingituna on ranges reaalajas töötavates süsteemides kriitiliste tööde tegemiseks kasutusel kuum reserv. See tähendab, et kõik (kriitilised) arvutused tehakse paralleelselt kahel või enamal arvutil/protsessoril ja need protsessorid kontrollivad perioodiliselt üksteise korrasolekut. Teine võimalik alternatiiv on hääletamisel baseeruva otsuse tegemine – mitu programmi (protsessorit) teevad sama tööd ja töö lõppedes võrdlevad omavahel tulemusi, õige tulemus valitakse hääletuse teel. Kogu liiasuse teema on siiski omaette uurimisvaldkond, mida selles loengukursuses ei vaadelda (nn. Byzantine Generals' Problem).

2.7.2 Ohutult riknevad või töövõimet säilitavalt riknevad süsteemid.

Mõnedes ranges reaalajas töötavates süsteemides eksisteerib üks (või rohkem) ohutut olekut, millesse on ohtlike vigade korral võimalik süsteem viia. Näiteks raudtee signalisatsioon süsteem – tõsise vea avastamise korral on võimalik katastroofi vältimiseks kogu rongiliiklus peatada. Kui sellised ohutud olekud eksisteerivad ja neisse on võimalik kiiresti (soovitavalt automaatselt) jõuda, nimetatakse vastavat reaalajasüsteemi ohutult riknevaks süsteemiks (*fail-safe real-time system*). Ohutu riknemine on ümbritseva keskkonna, mitte arvutisüsteemi omadus. Arvutisüsteemile esitatakse ohutult riknevas süsteemis kõrge vea-avastusvõime nõue (*high error-detection coverage*), s.t. ohtlike vigade õigeaegse avastamise tõenäosus peab olema lähedane ühele.

Sageli kontrollitakse arvutite, nende osade, ja tähtsamate programmide tööd spetsiaalsete (sageli arvutiväliste) valvetaimeritega (*watchdog*). Kontrollitav objekt peab valvetaimerile saatma perioodilise kinnituse oma elusolemisest ja töövõimest. Kui selline sõnum ei saabu ettenähtud ajaintervallis, eeldab valvetaimer, et objekt on töövõimetu ja sunnib objekti üle minema ohutusse olekusse (kus ta ei saa ülejäänud süsteemile kahju teha). Samuti võib valvetaimer vigase objekti asemele süsteemi lülitada tema reservis oleva koopia (kui selline on olemas).

Paljudes rakendustes ei ole ohutut olekut võimalik leida, näiteks lennukit juhtivas reaalaajasüsteemis. Sellises rakenduses peab arvutikobar rikete korral võimalikult

kaua säilitama eluliste funktsioonide täitmise võime (võimalik, et väiksemas mahus kui normaalselt). Selliseid reaalajasüsteeme nimetatakse töövõimet säilitavalt riknevateks (*fail-operational, or fail-soft real-time systems*).

2.7.3 Garanteeritud ajakitsendustega või parimate võimalike ajakitsendustega süsteemid.

Garanteeritud ajakitsendustega süsteemide loomine nõuab hoolikat planeerimist ja ulatuslikku objekti uurimist. Projekteerimine algab halvima juhu jaoks spetsifitseeritud vea- ja koormushüpoteesidest, ning tagab halvimate juhtude puhul tekkiva koormuse korral ajakitsendustest kinnipidamise (garanteeritud, mitte tõenäosusliku). Ajakitsendustest kinnipidamist saab garanteerida vaid tingimusel, et projekteerimise aluseks olnud vea- ja koormushüpoteesid on tõesed.

Kui projekteerija ei suuda garanteerida ajakitsendustest kinnipidamist ning lubab vajaduse korral neid rikkuda, on tegemist parimate võimalike ajakitsendustega süsteemiga (*best-effort design*). Teiste sõnadega, süsteem täidab esitatud nõudeid sedavõrd kuivõrd ta konkreetses olukorras suudab. Ajakitsendustest kinnipidamiseks ei vähendata vähemoluliste funktsionaalsete nõuete täitmist. Sellised süsteemid ei vaja väga ranget vea- ja koormushüpoteeside spetsifikatsiooni. Kõik projektotsused tehakse “nii head kui võimalikud”, nende piisavust kontrollitakse katsetustega. On üliraske (praktiliselt võimatu) demonstreerida, et selline süsteem töötab korrektselt alarmilaviini korral (*rare-event scenarios*). Paljud mitte-ohutuskriitilised reaalajasüsteemid on projekteeritud kui parimate võimalike ajakitsendustega süsteemid. Ka inimene kui süsteem töötab tavaliselt parimate võimalike ajakitsendustega süsteemina.

2.7.4. Piisavate ressurssidega või mittepiisavate ressurssidega süsteemid.

Garanteeritud ajakitsendustega süsteemid baseeruvad eeldusel, et ressursse on ka tippkoormuste rahuldamiseks piisavalt. Majanduslikus mõttes on see kallis lahendus kuna suur osa ressursse seisab enamik aega kasutult. Sellepärast on paljud mitte-ohutuskriitilised süsteemid projekteeritud teadlikult eeldades mittepiisavate ressursside olemasolu. Kuigi mittepiisavate ressurssidega projekt baseerub ressursside dünaamilisel jaotamisel, on ressursside koguhulk valitud nii, et piisavalt suur hulk normaalseid olukordi on olemasolevate ressurssidega lahendatavad ilma ajakitsenduste rikkumiseta.

Arvestades riistvara hinna langust tundub tuleviku suund olema piisavate ressurssidega süsteemide osakaalu suurenemisel. Arvutite üha suurenev kasutamine lennukite, autode, tehnoloogiliste protsesside juhtimisel ja muude ohutuskriitiliste funktsioonide realiseerimisel suurendab poliitikute ja laiade rahvamasside huvi süsteemide töökindluse ning ohutuse vastu. Seega on oodata, et seadusandlikult sunnitakse projekteerijad argumenteeritult põhjendama oma projekti toimimist **igas** olukorras (vt. ka sertifitseerimise kohta käivaid märkuseid). Psühholoogiliselt on huvitav, et inimeselt seda ei nõuta – inimesele esitatavad töö- ja veakindluse nõuded on oluliselt madalamad kui arvuti baasil töötavatele seadmetele esitatavad nõuded.

Ranges reaalajas töötavad reaalajasüsteemid peavad olema projekteeritud lähtudes piisavate ressursside olemasolu eeldusest.

2.7.5 Sündmuste poolt juhitud või aja poolt juhitud süsteemid

Pealkirja ingliskeelne variant on *Time-triggered versus event triggered systems,* mis viitab võimalusele ja vajadusele aeg-ajalt asendada aeg sündmustega või vastupidi. Aja asendamise võimalus sündmustega ja vastupidi on üks ammuse filosoofilise vaidluse allikas. Astronoomiline aeg on kujutatav rangelt suunatud ajateljega, mida mööda saab normaalolukorras liikuda vaid ühes suunas (aja suurenemise suunas). Filosoofiast lähtudes modelleeritakse ajatelge reaalarvudega. Iga sündmus on punkt ajateljel. Jättes ajutiselt kõrvale pragmaatilised inimese ja arvuti maailmatajumise võimega seotud probleemid, taandub pealkirjas toodud alternatiiv probleemile millal kasutada reaalajasüsteemi töö juhtimiseks sündmuseid, millal lähtuda ajast (ja millise dünaamilise süsteemi ajast) – koosneb ju reaalajasüsteem mitmetest koostoimivatest dünaamilistest süsteemidest.

Pealkirjas toodud alternatiivid viitavad sügavatele filosoofilistele tunnetusteooria ja modelleerimise alastele probleemidele, millede lahenduses ei ole veel üksmeelele jõutud. Elementaarse sissejuhatusena ja täpsemate viidete allikana aja modelleerimisse tarkvara rakendusi silmas pidades, võiks vaadata artiklit Motus (1993) “Time concepts in real-time software” Control Engineering Practice, vol.1, no.1, pp 21-33.

Sündmuste poolt juhitud süsteemis saab igasugune interaktsioon, tegevuse käivitamine, või lõpetamine algatatud olulise muudatuse (s.t. sündmuse tekkimise) tõttu ümbritseva keskkonna (vahel ka arvutisüsteemi) olekus. Tavaliselt puudub projekteerijal ümbritseva keskkonna kohta täielik informatsioon, seega toimuvad sündmused projekteerija jaoks näiliselt juhuslikel ajahetkedel. Arvutisüsteem peab olema igal hetkel võimeline vastu võtma tundmatu arvu sündmuseid ja neile reageerima ettenähtud aja jooksul. Tüüpiline näide on katkestuste töötlemise mehhanism arvutis, mis teatud tingimustel reageerib mõnede sündmuste tekkimisele. Sündmuste poolt juhitud süsteemide tarkvara tööd planeeritakse dünaamiliselt. Dünaamilise planeerimise korral on väga raske hinnata vajaminevat ressursside hulka, mille abil oleks võimalik garanteerida ajakitsenduste rahuldamist. Ei tohi unustada, et dünaamiline planeerimine ise on üsna töömahukas ja raskelt läbiviidav protsess.

Lisaraskused võivad tekkida seoses ebasobivalt valitud aja mudeliga. Näiteks nn. pideva ajamudeli korral (aega modelleeritakse reaalarvudega) võib kuitahes lühikeses ajaintervallis tekkida lõpmata palju sündmuseid. Iseküsimus on kas kõik need sündmused on olulised, ja kas reaalarvuline aeg on üldse tegelikkuses võimalik. Kõigile peaks olema teada, et reaalarvuline (s.t. pidev) aeg on teooria jaoks mugav abstraktsioon, ei enamat. Olukorda aitaks parandada kui projekteerija teaks (suudaks hinnata) ümbritseva keskkonna sundparalleelsuse astet. Nii või teisiti on sündmustega juhtimise korral vaja tegelikkust kirjeldada ligikaudselt, tulemuse kvaliteet sõltub aproksimatsiooni täpsusest. Sündmustega juhtimise kasuks räägib klassikalise arvutiteaduse sügav ja pikaajaline armastus nn. loogilise aja vastu (millel baseerub

arvuti töö) – traditsiooniline programmeerimine on alati oluliselt baseerunud sündmuste abil juhtimise ideel.

Aja poolt juhitud süsteemides on kõik interaktsioonid, tegevuste käivitamised ja/või lõpetamised algatatud aja kulgemise järgi. Sisuliselt asendatakse aja punkt-semantika intervall-semantikaga – arvutisüsteem kontrollib vahetult möödunud ajaintervallis toimunud sündmuseid ja valib reageerimiseks nende hulgast välja olulised. Tegevuste planeerimine lihtsustub oluliselt – suurem osa selliselt esitatud sündmusi on esitatavad perioodilistena (või lähendatavad perioodiliste sündmuste jadaga), vaid mõned üksikud sündmused jääavad oluliselt juhuslikuks. On võimalik läbi ajada peamiselt staatiliste plaanuritega kuna suuremat osa interaktsioone ja käivitusi saab algatada ettemääratud ajahetkedel. Lisaraskus tekib hajutatud arvutisüsteemi sisese kellade sünkroniseeri-misega, aga ka arvutisüsteemi kellade ja ümbritseva keskkonna kellade (ajaarvestus süsteemide) omavahelise sünkroniseerimisega.

Hajutatud arvutisüsteemis (tüüpiline reaalajas töötava arvutikobara esinemise vorm) on kõikide võrgu sõlmede kellad sünkroniseeritud nn. globaalse ajaga. Iga süsteemis liikuv andmeelement peab olema varustatud globaalsele ajale viitava ajalipikuga. Globaalse aja granulaarsus peab olema valitud selline, et kõigi oluliste juhitava/jälgitava kobara mõõtmis- ja vaatlustulemuste ajaline järjestus oleks üheselt määratud.

Lisadetaile aja modelleerimise ja ajaliste omaduste uurimise kohta antakse kursustes LAP 5712 “Tarkvara dünaamika” ja LAP 8780 “Reaalaeg ja tehisintellekt”.

2.8 Reaalajasüsteemide levik ja maksumuse hindamine

Reaalajasüsteemide leviku, või mitteleviku, määrab peamiselt nende poolt pakutava lisateenuse ja nende hinna suhe. Kui on odavam kasutada mittereaalajasüsteemi (s.t. ajalistest vigadest põhjustatud kahjud on keskmiselt väiksemad kui lisakulutused, mis on vajalikud reaalajasüsteemi loomiseks), siis seda ka tehakse. Viimase aja tendents tundub siiski olema üleminek reaalajasüsteemidele – seoses arvutite kasutamise levikuga inimeludega, ja/või väga suurte materiaalsete väärtustega seotud valdkondadesse.

Kogu süsteemi maksumuse võib ligikaudu jagada kolme ossa – süsteemi loomise kulud (*development cost*), süsteemi tootmise kulud (*production cost*), ja süsteemi tööshoidmise (kasutamise) kulud (*maintenance cost*). Erinevates rakendustes võivad erinevate etappide kulude omavahelised suhted tunduvalt erineda.

2.8.1 Sardsüsteemid kitsamas mõttes (embedded systems, in narrow sense))

Terminit *embedded systems* kasutatakse inglise keeles kahes tähenduses. Selles alajaotuses toodud näidetes on kasutatud terminit kitsamas mõttes. Sellegipoolest kehtivad paljud järeldused kõikide reaalajasüsteemide kohta.

Kitsamas mõttes tähendab sardsüsteem (*embedded system)* reaalaja süsteemi, mis on ehitatud seadme sisse ja ei ole tavakasutajale üldse märgatav (näiteks,

videomakk, automootori juhtimine, autopidurite juhtimine, ratastool, jne); sellistes rakendustes on enamasti tegemist mikroprotsessoritega ja või spetsiaalselt tehtud kiipidega (ASIC – application specific integrated circuit), mis on nii sisendi kui ka väljundi poolt vahetult ühendatud juhitava ja/või jälgitava objektiga.

Laiemas mõttes tähendab sardsüsteem (*embedded system)* suvalist reaalajasüsteemi, s.o. igasugust süsteemi kus vähemalt osa sisendeid ja väljundeid on vahetult ühendatud objektiga. Sellises interpretatsioonis on ka liikluspolitsei poolt tänavalt kasutatav autoregister sardsüsteem (*embedded system*), mis on ehitatud politsei infovahetusvõrku ja on vajalik igapäevase töö tegemiseks.

Paljud varasema perioodi mehaanilised ja jäiga struktuuriga elektroonilised seadmed on viimasel ajal asendatud programmeeritavate seadmetega, mis sisuliselt on (sardsüsteemid) reaalajasüsteemid. Näideteks on automootori sissepritsesüsteem, südame stimulaator (*cardiac pacemaker*), faxiaparaat, mobiiltelefon, televiisor, pesumasin, jne. Arvuti ei ole taolistes rakenduses kasutajale nähtav, kuna enamasti on inimliides tahtlikult jäetud traditsiooniliseks.

Kitsamas mõttes sardsüsteemidel on selgelt eristatavad omadused ja nõuded, mis mõjutavad nende loomisprotsessi (*development process*):

- **masstoodang** – neid toodetakse automatiseeritud liinidel väga suurtes kogustes; iga üksiku eksemplari hind peab olema väike, s.t. muutub jälle oluliseks mälu ja protsessori efektiivne kasutamine (nagu see oli enne 1980-ndaid); hind on üks põhjus, miks autotööstus püüab siiani 8 ja/või 16 bitise protsessoriga läbi saada;
- **staatiline struktuur** – arvuti on jäigalt ehitatud toote sisse, ei arvuti ega ka toote struktuuri ole peale toote valmimist võimalik muuta. Aprioorselt teada oleva keskkonna omadusi saab eelnevalt analüüsida (projekteerimise ajal) – nende põhjal tuleb juba projekteerimise ajal tarkvara lihtsustada, teha tundetumaks vigade jms sellise suhtes (robustsemaks) ja suurendada töö efektiivsust. Teiste sõnadega, püütakse vältida dünaamilisi tarkvara töö juhtimise mehhanisme (dünaamiline mälujaotus, tegude (*tasks*) dünaamiline loomine ja likvideerimine, jne), mis suurendavad toote vajadust ressursside järele, ning sellega suurendavad tema keerukust – järelikult üldjuhul vähendavad ka veakindlust;
- **inimliides** – sardsüsteemide kasutajateks on mitte-arvutustehnika spetsialistid, järelikult tuleb inimliides projekteerida lähtudes kasutusvaldkonna traditsioonidest, kasutaja vajadustest ja suhteliselt vähe arvestades programmistide ja teiste arvutispetside traditsioonidega;
- **mehhaanilise osa minimiseerimine** – tootmishinna vähendamiseks ja töökindluse tõstmiseks tuleb minimiseerida mehhaaniliselt liikuvate osade arvu tootes.
- **funktsioonid defineerib tarkvara** – toote funktsionaalsus määratakse püsimälus asuva tarkvara poolt; kuna pärast toote valmimist ei ole võimalik seda tarkvara enam muuta, on tarkvara kvaliteedi nõuded väga kõrged;
- **kasutamise (*maintenance*) strateegia** – paljudel juhtudel ei ole mingi hooldus võimalik (arvutisüsteemi, või toote jagamine hooldatavateks osadeks on liiga kallis), vea korral asendatakse kogu toode; kui hooldus on siiski võimalik, peab toode olema võimeline ennast normaalse töö ajal diagnoosima, või omama liidest välise diagnostika süsteemiga
- **võime suhelda teiste toodetega** – enamus sardsüsteeme (kitsamas mõttes) töötavad alguses omaette, kuid hiljem võib tekkida vajadus neid omavahel

ühendada; selletõttu on enamuses arvutit sisaldavates toodetes liides teiste arvutisüsteemidega suhtlemiseks (isegi kui seda kasutatakse harva, on ikkagi odavam teha suur seeria ühesuguseid tooteid kui kaks väikest seeriat veidi erinevaid tooteid). Andmesidet juhtiv protokoll peab olema lihtne ja robustne; andmeside kiiruse probleemid tekivad taolistes rakendustes väga harva. Hetkel on selles valdkonnas kõige levinum sideprotokoll CAN (Controller Area Network) – praktiliselt kõik autorakendused baseeruvad CAN protokollil.

Praktikas on aja jooksul välja kujunenud neljafaasiline sardsüsteemide arendustsükkel:

- **esimene faas** – iseseisev arvutisüsteem mikroprotsessoril (tavaliselt ilma operatsioonisüsteemita), mis põhimõtteliselt realiseerib tulevase toote funktsioonid; see faas realiseeritakse rakendust tundvate spetsialistide poolt, kes ei pruugi olla hiilgavad arvutispetsialistid; vaadeldav tulevase toote prototüübina;
- **teine faas** – lisab süsteemile kasutamise mugavuse tõstmiseks vajalikud funktsioonid; tulemuseks on esialgne (s.t. mitte spetsiaalselt disainitud) tarkvara arhitektuur, mis on üsna juhuslikult modifitseeritud hulgaliselt lisatud täiendavate osadega;
- **kolmas faas** – teise faasi täienduste tulemusena on tarkvara struktuur läinud ülikeeruliseks, tekivad töökindluse probleemid, silumine ja katsetamine muutuvad äärmiselt töömahukaks ja väheusaldatavaks; projekti kaasatakse professionaalsed programmistid, kogu tarkvara struktuur projekteeritakse uus, enamasti tuuakse sisse ka operatsioonisüsteem; see faas ei lisa olulist süsteemi kasutamismugavusele ja funktsionaalsusele, kuid on hädavajalik töökindluse tagamiseks – selles faasis kulub palju raha näiliselt mitte millegi peale ja tihti jäetakse töö selles faasis katki;
- **neljas faas** – tulevast toodet vaadatakse kui suurema süsteemi osa, mis peab olema interaktsioonis end ümbritseva keskkonnaga; lisatakse andmevahetusliides, mis esialgu rahuldab ettevõttesisest standardit, seejärel viiakse vastavusse mingi rahvusvahelise standardiga; otsitakse võimalusi realiseerida süsteemi osi ASIC'u abil, tarkavara ressursimahukus minimiseeritakse, kood pannakse püsimällu, jne.

On selgelt näha, et tegemist ei ole mitte teooria poolt soovitatud arengufaasidega, vaid tegelikkuses väljakujunenud praktikaga. Lähitulevikus (10 aastat) on oodata massilist sardsüsteemide kasutamist autotööstuses – pidurid, käigukastid, vedrustus, mootor, diagnostika. Siia lisanduvad seadmed, mis on vajalikud automaatsetel kiirteedel sõitmiseks. Ennustuse kohaselt ületavad arvutisüsteemide loomiseks kulutatavad summad autotööstuses peatselt kontorite arvutisüsteemide tegemiseks kulutatavad summad.

2.8.2. Protsessijuhtimissüsteemid

Tehnoloogilise protsessi juhtimiseks loodi maailma esimene reaalajasüsteem – Monzanto keemiatehases kasutati raalregulaatoreid (*direct digital controller, DDC*) tehnoloogiliste protsesside vahetuks juhtimiseks juba 1957 aastal.

Protsessijuhtimissüsteemid on enamasti unikaalsed. Kulub väga palju aega ja raha selleks, et kohandada juba valmis arvutisüsteem ja tarkvara konkreetse tehnoloogilise protsessi andurite ja täiturite füüsiliste omaduste ja paigutusega, protsessi juhtimise

konkreetse strateegiaga, kohalike reeglite ja määrustega, aruandlussüsteemiga, kasutajate soovidega.

Vaatamata hästi väljaarendatud modulaarsele struktuurile, kus iga moodul on omaette häälestatav, on taoliste süsteemide loomise hind väga palju kõrgem, kui nende tootmise hind (aparatuuri ja arvuti riistvara hind pluss tarkvara ümberhäälestus ja võimalik modifitseerimine). Tarkvara kopeerimise hinnast ei maksa üldse rääkida, see on praktiliselt füüsilise kandja hind.

Näide: Jaapanis sama firma kahele eri linnas asuvale praktiliselt identsele tehnoloogilisele protsessile otsustati ehitada ekspertsüsteemil baseeruv superviisorjuhtimissüsteem. Kasutades standardset ekspertsüsteemi tuuma (expert system shell) ehitati juhtimissüsteem ühele protsessile valmis 18 kuuga. Selle tööga jäädi rahule ning otsustati sama süsteem üle kanda identsele tehnoloogilisele protsessile teises linnas. Ülekandmiseks ja tööle rakendamiseks kulus veidi üle 12 kuu – põhiaeg kulus inimliidese muutmiseks ja otsuste põhjendussüsteemi modifitseerimiseks. Firma kõrgem juhtkond lootis juhtimissüsteemi teisele protsessile rakendada kolme kuuga.

Tehnoloogiliste protsesside juhtimissüsteemid on kallid ja neid tehakse maailmas suhteliselt harva ning väikestes kogustes -- tavaliselt siis kui tekivad uued tehnoloogilised protsessid.

Protsessijuhtimissüsteemide koguarv on liiga väike, et majanduslikult õigustada spetsialiseeritud võrkude, võrguprotokollide ja riistvara väljatöötamist. Seetõttu kasutatakse enamasti teistes arvutirakendus-valdkondades loodud riistvara ja süsteemtarkvara tooteid (näiteks, robotite, autode jms jaoks loodud tooteid – CAN protokoll, kellade sünkroniseerimiskiibid, jne).

Eelmine lõik on tõene enamusel juhtudest, kuid leidub alati sedavõrd olulisi rakendusi, kus ei pöörata hinnale tähelepanu (tuumaelektrijaamad, sõjatööstus, keemiatööstus, metallurgia). Selletõttu on paljud olulised reaalajasüsteemide kontseptsioonid ja komponentide põhiideed välja kasvanud just nimelt protsessijuhtimisrakendustest – selles valdkonnas on reaalajasüsteemide kasutamise kogemus kõige pikaajalisem ja ka rakendusprobleemid on hästi läbiuuritud (ning teoreetiliselt üldistatud).

2.8.3 Multimeedia süsteemid

Multimeedia rakendused on suhteliselt hiljuti tekkinud nõrgas reaalajas töötavate arvutisüsteemide kasutamise valdkond. Mõned piir-ajad – näiteks audio- ja videosignaali sünkroniseerimiseks -- on kindlad piir-ajad (*firm deadlines*), meelelahutustööstuse alastes rakendustes ei ole tegu rangete piir-aegadega (*hard deadline*). Ajutised piir-aegade rikkumised põhjustavad teenuse kvaliteedi (*quality of service, QOS*) languse, kuid reeglina ei vii katastroofini. Viimane väide kehtib praeguste multimeedia rakenduste puhul, juba praegu uurimisfaasis olevad multimeedia rakendused võivad kvaliteedi langemise korral põhjustada tõelisi katastroofe – seega on peatselt oodata ka rangete piir-aegade sissetoomist. Näiteks,

multimeedia süsteemi kasutamine protsessijuhtimisoperaatori inimene-masin liidesena (Xia, 1999)

Pideva audio-videosignaali edastamiseks nõutav arvutisüsteemi jõudlus on üldjuhul suur ja seda on raske mõistlikult piirata – suurema jõudluse korral on alati võimalik saada paremat kvaliteeti. Ressursi jagamise strateegia multimeedia rakendustes baseerub olemasolevate ressursside dünaamilisel jagamisel, mitte staatiliselt eraldatud ressursside kasutamisel (nagu on tavaline ranges reaalajas töötavates süsteemides). Oluliseks argumendiks dünaamilise ressursijagamise toetuseks on äärmiselt dünaamiliselt muutuvad multi-meedia süsteemi koormuskarakteristikud.

Kasutaja otsused ja nõudmised mõjutavad ressursi jagamise poliitikat märgatavalt. Näiteks, kui kasutaja vähendab ühe videoakna suurust ja suurendab teise akna suurust, siis vähendatud aknas saab süsteem kasutada märksa väiksemat ülekande kiirust sama kvaliteediga pildi säilitamiseks ning suunata vabanenud ressursid suurendatud aknas oleva pildi kvaliteedi parandamiseks.

Sisuliselt samasse klassi kuuluvad ka viimasel ajal levima hakanud Interneti vahendusel realiseeritavad reaalajarakendused – C^3 süsteemid (Command, Control and Communication Systems), mis on peamiselt (kuid mitte ainult) sõjalisteks rakendusteks, CORBA (Common Object Request Broker Architecture), ORB (Object Request Broker) ja muudel hajusarhitektuuridel baseeruvad rakendused.

2.9 Teise osa kordamisküsimused

1. Mis on reaalajasüsteem? Mille poolest ta erineb teistest süsteemidest?
2. Mis teeb arvutisüsteemist reaalajas töötava arvutisüsteemi?
3. Iseloomustage termodünaamilises mõttes avatud ja suletud arvutisüsteeme.
4. Mille poolest erinevad programmeerimises kasutatavad paralleelsed ja sundparalleelsed programmid (concurrent and forced concurrent programs)?
5. Mille poolest erinevad funktsionaalsed ja mittefunktsionaalsed nõuded tarkvarale?
6. Millest lähtudes määratakse olekumuutuja väärtustele kehtivusintervall? Kas ühel olekumuutuja väärtusel võib olla rohkem kui üks kehtivusintervall? Miks?
7. Mida tähendab mõõtmise eeltöötlus? Millistest tegevustest see koosneb?
8. Kas inimliides ja alarmiseire alamsüsteem on kuidagi omavahel seotud? Miks ja kuidas?
9. Mida Te teate alarmilaviinist (alarm shower)? Miks võib alarmilaviin olla tarkavara-süsteemile ohuallikaks?
10. Milliseid juhtimise liike eristatakse reaalajasüsteemides?
11. Kas juhtimisalgoritmi realiseerimine arvutis võib muuta algoritmi omadusi? Põhjendage miks, või miks mitte.
12. Pideva signaali mõõtmisel diskreetsetel ajahetkedel saadud väärtuste jadast ei õnnestu mitte alati taastada esialgne signaal. Selgitage miks?
13. Iseloomustage lühidalt reaalajasüsteemides kasutatavat inimliidest. Mille poolest see võib/võiks erineda traditsioonilisest arvutiliidesest?
14. Mida kujutab endast eriolukordade töötlus? Miks eriolukordade töötlusel on eriline tähtsus reaalajasüsteemides?
15. Kuidas on alarmiseire seotud eriolukordade töötlusega?

16. Millisest elutsükli etapist peaks hakkama tõsiselt tegelema eriolukordade spetsifitseerimisega ja nende lahendusalgoritmidega?
17. Tooge eriolukordade näiteid erinevatest elutsükli etappidest.
18. Miks on reaalajasüsteemis vaja ajakitsendusi?
19. Loetlege ajakitsenduste peamised allikad?
20. Miks on vaja andmeelementide kehtivusaegu? Millised faktorid määravad kehtivusintervalli pikkuse?
21. Selgitage kuidas (miks) teevad alumise ja ülemise piirina antud ajakitsendused (nn. fluktuatsiooniga ajakitsenduste väärtused) kogu tarkvara analüüsi raskemaks? Kas ajakitsenduste väärtuse fluktuatsioon võib põhjustada raskuseid arvutis realiseeritud teoreetiliste tulemuste kasutamisel?
22. Millised iseloomulikud ajakitsenduste grupid esinevad reaalajasüsteemis? Tooge näiteid erinevatesse gruppidesse kuuluvatest ajakitsendustest.
23. Loetlege tüüpilisi mittefunktsionaalseid nõudeid.
24. Mille poolest erinevad töökindlus (*reliability*) ja valmisolek (*availability*)?
25. Milliseid nõudeid peab täitma ranges reaalajas töötava süsteemi tarkvara, et teda saaks sertifitseerida?
26. Mille poolest erinevad sardsüsteemid (embedded systems) kitsas mõttes ja sardsüsteemid laias mõttes?
27. Mis on ranges reaalajas töötava süsteemi peamine erinevus nõrgas reaalajas töötavast süsteemist?
28. Mille poolest erineb aja poolt juhitud süsteem sündmuste poolt juhitud süsteemist? Kas ja miks saab ühte eelistada teisele?
29. Milliseid ranges reaalajas töötavaid rakendusi Te näete multimeedia süsteemidele?

2.10 Teises osas kasutatud kirjanduse viiteid

Bransby M. (1998) “Explosive lessons”, IEE Computing and Control Engineering Journal, vol. 9, no.2, 57- 60

B.W.Boehm (1988) “A spiral model of software development and enhancement”, IEEE Computer, vol.21, no.5, 61-72

Charles J. (1999) “Neural Interfaces Link the Mind and the Machine”, IEEE Computer, vol.32, no.1, 16-18.

Cristian F. (1995) “Exception handling and tolerance of software faults”, in Software Fault Tolerance, ed. M.Lyu, J.Wiley&Sons, 81-107

de Lemos R. and A.Romanovsky “Exception handling in a Co-operative Object-Oriented Approach”, University of Newcastle, (private communication)

Dicken C.R. (1999) “”Soft” control desks and alarm display” , IEE Computing and Control Engineering Journal, vol, 10, no. 1, 11-16.

Ebert R.E. (1994) “User Interface Design” Prentice Hall Inc., Englewood Cliffs, NJ.

Giddings R.V. (1984) “Accommodating uncertainty in software design”, Communications of the ACM, vol. 27, no.5, 428-434

Kopetz H. (1997) “Real-time systems. Design principles for distributed embedded applications”, Kluwer Academic Publishers, 338 pp

Laprie J.C. (Editor) “Dependability: Basic Concepts and Terminology – in English, French, German, and Japanese” Springer-Verlag, Vienna, Austria, 1992

Leveson N.G. (1995) “Safeware: system safety and computers”, Addison Wesley Company, Reading, Massachusetts

O.Maler (1992) “Hybrid systems and real world computations” - Workshop on Theory of Hybrid Systems, Technical University of Denmark

Marten D. van der Laan “Signal sampling techniques for data acquisition in process control” Ph.D. thesis University of Groningen, the Netherlands, 1995,186 pp.

Miller R. and A.Tripathi (1997) “Issues with Exception Handling in Object-Oriented Systems”, Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP’97), Lecture Notes in Computer Science # 1241, eds. M.Aksit, S.Matsuoka, Springer-Verlag, Berlin, Germany, 85 - 103

Motus L. (1993) “Time concepts in real-time software” Control Engineering Practice, vol.1, no.1, pp 21-33.

Motus L. and M.G.Rodd (1994) “Timing analysis of real-time software”, Elsevier Science Publishing/Pergamon, Oxford, 212 pp

Simon H.A. (1996) “The science of the Artificial”, MIT Press, 3-rd edition

E.T.Whittaker “On the functions which are represented by the expansions of the interpolation theory”, Proc. of the Royal Society of Edinburgh, 1915, 181-194

Xia Q., M.Rao (1999) “Dynamic case-based reasoning for process operation support systems”, Engineering Applications of Artificial Intelligence, vol. 12, no.3, 343-362

2.11 Teise osa terminite selgitav sõnastik

Arvutikobar (Computational cluster): *Reaalajasüsteemi* alamsüsteem mis koosneb ühest või mitmest iseseisvast arvutist, kõik arvutikobaras olevad arvutid suhtlevad omavahel sihtotstarbelise andmeside võrgu kaudu.

Juhitav/jälgitav kobar (Controlled cluster): on füüsikaline või tehnoloogiline protsess, seade, nähtus või nende interaktiivne kogum, mille funktsioneerimise mõjutamiseks ja/või jälgimiseks kasutatakse *arvutikobarat.* Näiteks keemiaprotsess, lennuk, auto, videomagnetofon, elementaarosakeste kiirendi, gaaskromatograaf, ratastool.

Reaalajasüsteem (Real-time system): Omavahel interaktsioonis olevate (dünaamiliste) süsteemide kompleks, mille üheks kohustuslikuks komponendiks on arvutisüsteem. Üldjuhul on tegemist geograafiliselt (s.o. ruumis) hajutatud süsteemiga, mis jagatakse tinglikult kolmeks kobaraks -- Inim-operaatorite kobar (Human-Operator cluster), *Arvutikobar* (Computational cluster), ja *Juhitav/jälgitav kobar* (Controlled cluster).

Reaalajas töötav arvutisüsteem (Real-time Computer System): Arvutisüsteem, mille töö õigsus ei sõltu mitte ainult realiseeritud algoritmi alusel saadud

arvutustulemusest (logical results), vaid ka nende tulemuste saamise hetkest. USA admirali definitsioon -- arvutisüsteem kus õige tulemus valel ajal on vale tulemus.

Sõnastik jätkub hiljem, kui aega tekib.