TEKNILLINEN KORKEAKOULU

Tietotekniikan osasto

Kimmo Varpaaniemi

UNIX-YMPÄRISTÖSSÄ TOIMIVAN OHJELMISTOLIITÄNNÄN SUUNNITTELU JA VERKKOTEOREETTINEN ANALYSOINTI

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 30.5.1991

Työn valvoja Leo Ojala

Työn ohjaaja Veli-Matti Taavitsainen

Alkusanat

Tämä työ on tehty Teknillisen korkeakoulun digitaalitekniikan laboratoriossa Kemira Oy:n toimeksiannosta.

Kiitän Kemira Oy:tä, digitaalitekniikan laboratoriota, säätötekniikan laboratoriota ja Control CAD Oy:tä saamastani avusta. Erityisesti kiitän professori Leo Ojalaa, joka ehdotti tätä työtä ja toimi työn valvojana, FL Veli-Matti Taavitsaista ja dosentti Heikki Haariota, jotka ohjasivat työtäni Kemira Oy:n taholta, sekä yliassistentti Ilkka Niemelää, jonka esittämistä kommenteista oli paljon apua työn kirjoittamisessa.

Kimmo Varpaaniemi

Sisällysluettelo

Liitteet

Kuvat

Lyhennysten ja merkintöjen luettelo

ASCII	American Standard Code for Information Interchange
C/E-verkko	ehto-tapahtuma-verkko
C_N	verkon N insidenssimatriisi
$\frac{dx}{dt}$	funktion x derivaatta muuttujan t suhteen
FIFO	first in first out
F_N	verkon N kaarien joukko
I/O	input-output
IP	Internet Protocol
IPC	Interprocess Communication
ISO	International Organization for Standardization
K_N	verkon N kapasiteettifunktio
M_N	verkon N alkumerkintäfunktio tai -vektori
$M[t > M'$	merkinnästä M päästään merkintään M' laukaisemalla transitio t
N	verkko
O	matriisi, jonka kaikki alkiot ovat nollia
OSI	Open Systems Interconnection
P/T-verkko	paikka-transitio-verkko

Pr/T-verkko	predikaatti-transitio-verkko
S_N	verkon N paikkojen joukko
TCP	Transmission Control Protocol
TLI	Transport Layer Interface
T_N	verkon N transitioiden joukko
UDP	User Datagram Protocol
W_N	verkon N kaaripainofunktio
X^T	matriisin X transpoosi
XY	matriisien X ja Y tulo
$\cdot t$	transition t etujoukko
$t\cdot$	transition t jälkijoukko
$-$	lukujen, matriisien, funktioiden tai joukkojen erotus
$\cap$	joukkojen leikkaus
$\cup$	joukkojen unioni
$\forall$	universaalikvanttori
$\exists$	eksistentiaalikvanttori

1 Johdanto

1.1 Ohjelmistoliitännöistä

Ohjelmistoliitäntä liittää ohjelmia tavalla tai toisella yhteen uudeksi kokonaisuudeksi. Usein jonkin kokonaisuuden suunnittelu ja toteutus ohjelmistoliitännän avulla on taloudellisesti kannattavampaa kuin vastaavan itseriittoisen ohjelman luominen.

Ohjelmistoliitäntä voidaan rakentaa yhteisten muistialueiden, muuttujien, aliohjelmien tai tiedostojen varaan. Moniajoympäristössä, kuten UNIX, prosessien välisen kommunikoinnin keinoin voidaan ohjelmistoliitäntöjä tehdä sujuvasti koskematta juuri lainkaan liitettävänä olevien ohjelmien lähdetiedostoihin tai tarvitsematta edes nähdä kaikkien liitettävänä olevien ohjelmien lähdetiedostoja.

1.2 Verkkoteoriasta

Verkkoteoria on rinnakkaisiin ja hajautettuihin järjestelmiin erikoistunut tietojenkäsittelyteorian osa-alue. Jotakin järjestelmää voidaan mallittaa verkolla, jossa on esitetty järjestelmän kannalta keskeiset synkronointi- ja kommunikointitapahtumat. Järjestelmän ominaisuuksia voidaan selvittää suorittamalla verkolle saavutettavuusanalyysi ja/tai invarianttianalyysi. Järjestelmän eri osia voidaan mallittaa erikseen omilla verkoillaan, mikä antaa mahdollisuuden tarkempaan analysointiin.

1.3 Työn tausta

1.3.1 Mallittaminen, simulointi, estimointi ja kokeiden suunnittelu

Kemiallista prosessia tutkittaessa pyritään muun muassa selvittämään eri suureiden välisiä riippuvuuksia mahdollisimman tarkasti ja taloudellisesti. Asioiden selvittäminen laskennallisin keinoin on yleensä huomattavasti taloudellisempaa kuin samojen asioiden selvittäminen puhtaan kokeellisesti.

Tehtyjen kokeiden sekä mahdollisesti alan teorian perusteella tutkittavalle ilmiölle rakennetaan malli. Malli antaa riippuvuuden säädettävien eli riippumattomien ja mitattavien eli riippuvien muuttujien välille. Näitä nimityksiä mallin eri muuttujille on käytetty lähteessä [3]. Tavallisia säädettäviä muuttujia ovat esimerkiksi

aika, lämpötila, paine ja reaktoriin syötettävien aineiden määrät. Tavallisia mitattavia muuttujia ovat puolestaan esimerkiksi lopputuotteen laatu ja saanto sekä eri aineiden konsentraatiot. Mallin parametreiksi kutsutaan jotakin valittua mallissa esiintyvien muuttumattomien suureiden joukkoa.

Mekanistisessa mallissa, lähteen [3] mukaan, säädettävien ja mitattavien muuttujien välinen yhteys pyritään perustamaan fysiikan ja kemian lakeihin. Empiirisessä mallissa säädettävien ja mitattavien muuttujien välinen riippuvuus on puhtaasti laskennallinen.

Käsitteellä simulointi tarkoitetaan malliin perustuvaa riippuvien muuttujien arvojen laskemista valituilla riippumattomien muuttujien arvoyhdistelmillä.

Parametrien estimointi on parametrien optimaalisten arvojen arviointia pyrittäessä hakemaan mahdollisimman hyvin havaintoihin sopiva malli. Yleisessä tapauksessa estimoinnissa lähdetään liikkeelle jostakin mallista ja luodaan parametrien arvoja muuttamalla iteratiivisesti uusia malleja, kunnes joko on saavutettu riittävän hyväksi katsottu malli tai on suoritettu sovittu maksimimäärä iteraatioita. Mallin vertaamisessa havaintoihin käytetään yleensä pienimmän neliösumman kriteeriä. Neliösummassa esiintyvät riippuvien muuttujien arvot saadaan simuloimalla. Estimoinnin tuloksena saadaan parametreille estimaatit sekä estimaateille virhearviot. Parametrin sanotaan identifioituvan hyvin, jos ko. parametrille saadun estimaatin suhteellinen virhe on pieni.

Kokeiden suunnittelussa etsitään optimaalisia koepisteitä. Koepisteet ovat optimaalisia, jos parametrit kokonaisuutena ottaen identifioituvat ko. koepisteitä käytettäessä mahdollisimman hyvin.

1.3.2 Modest ja Simnon

Yleiskäyttöisiä matemaattisia työkaluja on olemassa runsaasti. Puhtaita simulointiohjelmistojakin on tarjolla kohtalaisesti. Kansainvälisiä sekä simulointiin että estimointiin tarkoitettuja ohjelmistoja ei ole kovinkaan paljon. SimuSolv on yksi harvoista kansainvälisistä sekä simulointiin että estimointiin tarkoitetuista ohjelmistoista. Sen sijaan Simusolvia ei ilmeisestikään ole tarkoitettu kokeiden suunnitteluun, tai ainakaan SimuSolvia kuvaavassa lähteessä [12] ei mainita kokeiden suunnittelua.

Modest on Kemira Oy:n Espoon tutkimuskeskuksessa kehitetty ohjelmisto mekanistisen mallien parametrien estimointia ja kokeiden suunnittelua varten. Modestia

voi käyttää myös simulointiin.

Simnon on Lundin teknillisessä korkeakoulussa kehitetty simulointiohjelmisto.

Modestissa käyttäjä kirjoittaa mallinsa FORTRANilla. Minkäänlaista modulaarista mallitusta Modest ei tue. Simnon tarjoaa sen sijaan korkeatasoisen kuvauskielen sekä mahdollisuuden mallittaa systeemejä modulaarisesti. Simnonin mallivalikoima on laajempi kuin Modestin mallivalikoima.

Jos Modestin ja Simnonin hyvät puolet voitaisiin jotenkin yhdistää, saattaisi tuloksena olla kansainvälisiin ohjelmistoihin verrattuna riittävän kilpailukykyinen ohjelmisto.

Äskeiseen virkkeeseen on tiivistetty ne toiveet, joiden pohjalta tämä diplomityö lähti käyntiin.

1.4 Modest-Simnon-liitäntä

Työnäni toteutin Modest-Simnon-liitännän. Kyseessä on varsinaisesti Modest-versio, joka käyttää Simnonia hyväkseen. Käyttäjä kirjoittaa mallinsa Simnonin kuvauskielellä.

Modest-Simnon-liitäntä päätettiin toteuttaa UNIX-ympäristöön, koska UNIX-Simnonia oli Suomessa jo runsaasti käytetty, kun taas Simnonin mahdollisista versioista muissa moniajoympäristöissä ei ollut tietoja. Moniajoympäristö oli tarpeen, jotta liitännän toteuttaminen prosessien välisen kommunikoinnin avulla olisi mahdollista. Prosessien välinen kommunikointi valittiin ensisijaiseksi tavaksi tehdä liitäntä, koska suoremmat tavat kuten Simnonin globaalien muuttujien tai aliohjelmien käyttö olisivat edellyttäneet Simnoniin liittyvien lupien ostamista.

Toteutusympäristönä oli HP-UNIX, tietokoneina TKK:n säätötekniikan laboratorion HP 9000/300-sarjan tietokoneet tassu ja tossu. Varsinainen määränpää uudella Modestilla oli Kemira Oy:n Espoon tutkimuskeskus, jossa ei kuitenkaan vielä ollut UNIX-konetta.

Tehty liitäntä toimii yhden prosessorin sisällä, toisin sanoen kaikki Modest-ajossa mukana olevat prosessit sijaitsevat samassa prosessorissa. Hajautetun liitännän tekemiseen ei ainakaan toistaiseksi ole ilmennyt tarvetta.

1.5 Johdatus seuraaviin lukuihin

Luvussa 2 esittelen Simnonin ja luvussa 3 Modestin. Näiden kahden luvun ensisijaisena tarkoituksena on tarjota esitietoja lukuun 6.

Luvussa 4 kuvaan prosessien välistä kommunikointia yleisesti.Tietoliikenneasioita en ole katsonut voivani täysin sivuuttaa, koska prosessien välisessä kommunikoinnissa yleinen suuntaus on kohti sellaista käytäntöä, jossa prosessien sijainnista riippumatta asiat ohjelmoidaan yhdenmukaisella tavalla. Luku tarjoaa myös sopivassa määrin esitietoja lukuun 5.

Luvussa 5 kuvaan UNIXin prosessien välisen kommunikoinnin eri mekanismeja. Tavoitteena on koota asioita hieman yhteen. Yksinkertaistavaa vertailua olen välttänyt. (Sovellutuksen erityispiirteillä on usein niin ratkaiseva merkitys, että mekanismien todenmukainen paremmuusjärjestykseen asettaminen edellyttäisi kaikkien vertailtavina olevien mekanismien kokeilemista. Itse kokeilin ohjelmallisesti kolmea eri mekanismia, eivätkä kokeellisen vertailun tulokset olleet aivan ennakkokäsitysteni mukaisia.)

Luvussa 6 esitän Modest-Simnon-liitännän keskeiset piirteet ja tehtyjen ratkaisujen perustelut.

Luvussa 7 analysoin Modest-Simnon-liitännän kommunikointitapahtumia P/T-verkkojen avulla kahden keskeisen kommunikointivaiheen osalta.

2 Simnon

Simnon on ohjelma, joka on tarkoitettu tavallisten differentiaaliyhtälöiden ja differensiyhtälöiden systeemien numeeriseen ratkaisemiseen ja dynaamisten systeemien simulointiin. Simnon on kehitetty Ruotsissa Lundin teknillisessä korkeakoulussa. Ensimmäinen versio ilmestyi vuonna 1972.

Simnon on opetus- ja tutkimustyöväline. Sovellutusalueista mainittakoon säätötekniikka, biologia, kemian tekniikka, taloustiede, sähkötekniikka, matematiikka ja konetekniikka.

Dynaaminen systeemi voidaan yleisimmässä muodossaan kuvata useana alisysteeminä. Alisysteemi voi olla differentiaaliyhtälösysteemi tai differenssiyhtälösysteemi. Kullakin alisysteemillä voi olla sisääntulomuuttujia ja ulostulomuuttujia. Alisysteemit kytkee yhteen ns. yhdistävä systeemi. Dynaaminen systeemi voi olla myös yksittäinen differentiaaliyhtälösysteemi tai differenssiyhtälösysteemi.

Simnon on vuorovaikutteinen ohjelma. Käyttäjä antaa Simnonille komentoja. Komentojen avulla käyttäjä voi simuloida systeemejä, piirtää muuttujien arvoja kuvaavia käyriä kuvaruudulle, tallettaa simulointituloksia tiedostoihin, asettaa systeemien parametreja ja alkuarvoja sekä tehdä paljon muutakin. Käyttäjä voi koota komentoja makroiksi ja kutsua makroja, ja makrot voivat kutsua toisiaan.

Differentiaaliyhtälösysteemistä käytetään Simnonissa nimitystä jatkuva systeemi, mikä viittaa ajan jatkuvuuteen. Differenssiyhtälösysteemistä käytetään vastaavasti nimitystä diskreetti systeemi.

Jatkuvat, diskreetit ja yhdistävät systeemit kuvataan kukin omassa tiedostossaan. Kuvaukset kirjoitetaan erityisellä kuvauskielellä, josta käytän nimitystä Simnon-kieli.

2.1 Simnonin eri versiot

Simnon on saatavilla ainakin MS-DOS-koneisiin ja UNIX-koneisiin. MS-DOS-Simnonista saa tietoa lähteestä [11]. Ko. käyttöopas soveltuu pitkälti myös UNIX-Simnonin käyttäjälle. UNIX-Simnonin erityispiirteistä saa tietoa lähteestä [10].

MS-DOS-Simnonista on lähteen [11] mukaan olemassa kolme erityyppistä versiota:

1) Regular on yleisimmin käytetty ja soveltuu suuriinkin ongelmiin.

2) Classroom Kit on Regularin kaltainen halvempi versio, joka soveltuu vain kohtuullisen pieniin ongelmiin. Nimensä mukaisesti sitä myydään kouluihin ja yliopistoihin.

3) Real Time Capability on Regularin laajennus, joka tukee analogista ja digitaalista syöttöä ja tulostusta.

UNIX-Simnonista on lähteen [10] mukaan olemassa kaksi erityyppistä versiota:

1) Regular on samanlainen kuin MS-DOSin vastaava versio.

2) Plus Version on Regularin laajennus, johon voidaan liittää ns. ulkoisia systeemejä.

UNIX-Simnonin plus-version ulkoisia systeemejä käsittelevää osiota 2.6 lukuunottamatta tässä luvussa esitettävä Simnonin kuvaus koskee kaikkia edellä lueteltuja Simnonin versioita.

2.2 Jatkuvat systeemit

Sanoja piste ja aikahetki käytän synonyymeinä toisilleen. Simuloinnin aikahetkihän on piste aika-akselilla. Käytän nimitystä ratkaisupiste niistä pisteistä, joissa jonkin systeemin muuttujien arvot simuloitaessa lasketaan.

Jatkuva systeemi voidaan lähteen [11] mukaan esittää vektoriyhtälöparina

$$\begin{cases} \frac{dx}{dt} &= f(x,u,t) \\ y &= g(x,u,t) \ , \end{cases} \tag{1}$$

missä u on sisääntulomuuttujien vektori, y ulostulomuuttujien vektori, x tilamuuttujien vektori ja t jatkuva aikamuuttuja.

Tilamuuttujien arvot alkupistettä seuraavissa pisteissä Simnon laskee numeerisesti integroimalla. Käyttäjä määrää integroinnin maksimiaskelkoon ja sen, millaista integrointialgoritmia käytetään. Jotkut algoritmeista integroivat tasavälisesti, toiset säätelevät askelpituutta tarpeen mukaan.

Jatkuvan systeemin Simnon-kielinen kuvaus sisältää seuraavat osat, jotka on kuvattu lähteessä [11]:

- Otsikko. Otsikko sisältää systeemin nimen. Systeemin nimi on täysin riippumaton sen tiedoston nimestä, jossa ko. systeemin kuvaus on annettu.
- Muuttujien julistaminen. Muuttujat voivat olla tyyppiä TIME, INPUT, OUTPUT, STATE tai DER, ts. aika-, sisääntulo-, ulostulo-, tila- tai derivaattatyyppiä.
- Alkulaskenta. (Tämä osa ei ole pakollinen.) Alkulaskentalohkossa tapahtuva laskenta suoritetaan simuloinnin alussa. Alkulaskentalohkossa voidaan alustaa apumuuttujia ja tilamuuttujia. Nimitys apumuuttuja tarkoittaa tässä ja jatkossakin sellaista systeemin sisäistä muuttujaa, jota ei ole julistettu erikseen.
- Simulointiyhtälöt. Tässä osassa tapahtuva laskenta suoritetaan aina, kun Simnon on jossakin simulointivälin ratkaisupisteessä. Asettaa voidaan apumuuttujia, ulostulomuuttujia ja derivaattoja.
- Tilojen alkuarvojen asetus. Tilojen oletusalkuarvot annetaan tässä osassa. Jos jokin tila jätetään mainitsematta, ko. tilan oletusalkuarvo on 0. Alkulaskentalohkossa tapahtuva alustus kumoaa tässä osassa annetun asetuksen.
- Parametrien asetus. Parametrien oletusarvot annetaan tässä osassa. Parametri ei muuta arvoaan simuloinnin aikana. Kaikki parametrit on asetettava, sillä parametrit tunnistetaan tämän osan perusteella.

Edellä ollut jako on tarkoitettu vain kuvaamaan systeemissä esitettäviä asioita. Todellinen syntaksi on jossain määrin väljempi.

2.3 Diskreetit systeemit

Diskreetti systeemi voidaan lähteen [11] mukaan esittää vektoriyhtälökolmikkona

$$\begin{cases} t_{k+1} &= h(x(t_k), u(t_k), t_k) \\ x(t_{k+1}) &= f(x(t_k), u(t_k), t_k) \\ y(t_k) &= g(x(t_k), u(t_k), t_k) \ , \end{cases} \tag{2}$$

missä u on sisääntulomuuttujien vektori, y ulostulomuuttujien vektori, x tilamuuttujien vektori ja t_k k:s ratkaisupiste, $k = 1, 2, \ldots$. (Sekaannusten välttämiseksi

muistutettakoon, että kaikki muuttujat ovat aikamuuttujan funktioita. Jatkuvaa systeemiä kuvaavissa yhtälöissä vain ei tarvinnut täsmällisesti esittää ko. funktio-ominaisuutta.)

Diskreetin systeemin ratkaisupisteiden sijainnin määrää käyttäjä systeemikuvauksessa.

Diskreetin systeemin Simnon-kielinen kuvaus sisältää seuraavat osat, jotka on kuvattu lähteessä [11]:

- Otsikko.
- Muuttujien julistaminen. Muuttujat voivat olla tyyppiä TIME, TSAMP, INPUT, OUTPUT, STATE tai NEW. TSAMP-tyypin muuttuja kertoo seuraavan ratkaisupisteen ja NEW-tyypin muuttuja jonkin tilamuuttujan arvon seuraavassa ratkaisupisteessä.
- Alkulaskenta. Jatkuvan systeemin alkulaskentalohkosta poiketen diskreetin systeemin alkulaskentalohkossa voidaan alustaa myös ulostulo- ja TSAMP-muuttujia.
- Simulointiyhtälöt. Jatkuvan systeemin yhtälöihin nähden diskreetin systeemin yhtälöissä on se ero, että yhtälöiden vasemmalla puolella ei ole derivaattoja mutta sen sijaan kylläkin TSAMP- ja NEW-muuttujia.
- Tilojen alkuarvojen asetus.
- Parametrien asetus.

2.4 Yhdistävät systeemit

Yhdistävän systeemin kuvaus sisältää seuraavat osat, jotka on kuvattu lähteessä [11]:

- Otsikko.
- Muuttujien julistaminen. Vain TIME-tyyppi kelpaa.
- Alisysteemien sisääntulomuuttujien asetus. Tämä osa suoritetaan aina, kun Simnon on jossakin simulointivälin ratkaisupisteessä. (Alisysteemeillä voi olla

erilaisia ratkaisupisteitä, jolloin tässä käytäneen ne kaikki läpi.) Alisysteemin muuttujaan, joka voi olla sisääntulo-, ulostulo- tai tilamuuttuja, viitataan loppuliitteellä, joka sisältää sen systeemin nimen, jonka muuttujasta on kyse. (Sisääntulomuuttuja voi esintyä vain yhtälön vasemmalla puolella.)

- Parametrien asetus.

2.5 Tärkeimpiä Simnonin komentoja

Seuraavassa on lyhyesti kuvattu tärkeimpiä Simnon-komentoja lähteen [11] mukaisesti. (Listassa ovat itse asiassa mukana lähinnä vain ne komennot, joita olen Modest-Simnon-liitännässä tarvinnut.)

- ALGOR-komennolla valitaan se integrointialgoritmi, jota jatkuvia systeemejä simuloitaessa käytetään. (Jos ko. komentoa ei ole Simnon-istunnon aikana annettu, Simnon käyttää jotakin oletusalgoritmia.) Kiinteän askelmitan algoritmeista mainittakoon EULER ja muuttuvan askelmitan algoritmeista DOPRI45R. Muuttuvan askelmitan algoritmit integroivat tarkemmin, joskin ne myös kuluttavat hieman enemmän aikaa kuin kiinteän askelmitan algoritmit.

- ASHOW-komennolla voidaan piirtää ns. store-tiedostoon talletettuja simulointituloksia tai muita aika-tila-sarjoja. Simnon valitsee tyhjän diagrammin ja skaalaa koordinaatiston siten, että kaikki käyrät mahtuvat kuvaan mutta myös täyttävät kuvan mahdollisimman hyvin.

- DISP-komennolla tulostetaan muuttujien arvoja kuvaruudulle, tarkemmin sanottuna Simnonin standarditulostusvirtaan. Kyseeseen tulevat vallitsevien, ts. SYST-komennolla viimeksi valittujen, systeemien muuttujat tai parametrit tai sitten Simnonin globaalit muuttujat. (Sivuutettakoon globaalit muuttujat.) Jonkun systeemin muuttujaan tai parametriin voi viitata loppuliitteettömällä tai loppuliitteellisellä nimellä. Loppuliite sisältää systeemin nimen, ja sitä tarvitaan lähinnä silloin, jos eri systeemien kaksi oliota ovat muuten samannimisiä.

- EXPORT-komennolla store-tiedosto muunnetaan ASCII-tiedostoksi. Store-tiedosto on binäärinen aika-tila-sarja-tiedosto, jonka formaatti on ainakin periaatteessa vain Simnonin tiedossa.

- IMPORT-komennolla ASCII-muotoisen aika-tila-sarja-tiedosto muunnetaan store-tiedostoksi.
- INIT-komennolla vallitsevan systeemin tilojen alkuarvoja muutetaan. (Argumenttisyntaksi sivuutettakoon.) Systeemin alkulaskentalohkossa tapahtuva alustus kuitenkin kumoaa INIT-komennon vaikutuksen. (Ks. edellä ollutta jatkuvan tai diskreetin systeemin kuvausta.)
- PAR-komennolla vallitsevan systeemin parametrien arvoja muutetaan. (Argumenttisyntaksi sivuutettakoon.)
- SHOW-komento muistuttaa paljolti ASHOW:ta. Ero on siinä, että SHOW piirtää käyrät vanhaan diagrammiin. Mikään koordinaatiston skaalaus ei tietenkään tällöin onnistu.
- SIMU-komennolla simuloidaan vallitsevia systeemejä jostakin aikapisteestä toiseen. Systeemien tilamuuttujat saavat välin alkupisteessä systeemikuvauksen ja esiintyneiden INIT-komentojen määräämät alkuarvot. Erinäisiä valitsimia on paljon. Esim. simuloinnin maksimiaskelkoko voidaan valita. Samoin voidaan tallettaa simulointituloksia argumenttina annettavaan store-tiedostoon. (Tallettaminen tapahtuu aina johonkin tiedostoon, oletusarvoisesti esim. store.d, jos STORE-komennolla on määrätty yksikin muuttuja talletettavaksi.)

  Valitsin -CONT on merkittävä. Kun -CONT-valitsin on annettu, simuloinnin alkuarvoiksi valitaan edellisen simuloinnin tilojen loppuarvot. Näin voidaan simuloida pätkittäin pisteestä pisteeseen.
- STORE-komennolla määrätään ne muuttujat, joiden arvot talletetaan simuloinnin aikana store-tiedostoon.
- SYST-komennolla valitaan vallitsevat systeemit. (Edelliset vallitsevat systeemit unohdetaan.) Jos halutaan kerrallaan olevan useampia kuin yksi vallitseva systeemi, pitää viimeisen argumentin olla yhdistävän systeemin sisältävän tiedoston nimi. (Ylipäätään argumentit ovat tiedostonimiä, koska varsinaiset systeeminimet lukevat vasta ko. tiedostoissa.)
- $-alkuisen komennon loppuosa tulkitaan käyttöjärjestelmäkomennoksi. Jos siis halutaan antaa Simnon-istunnon aikana käyttöjärjestelmäkomento, pistetään komennon eteen $.
- Makron nimi komentona aiheuttaa kyseisen makron suorittamisen.

2.6 UNIX-Simnonin plus-version ulkoiset systeemit

UNIX-Simnonin plus-versio tarjoaa mahdollisuuden liittää Simnoniin ns. ulkoisia systeemejä. Ulkoinen systeemi on FORTRAN- tai C-kielinen aliohjelma, joka esittää jatkuvaa tai diskreettiä systeemiä. Ulkoisen systeemin liittäminen Simnoniin tarkoittaa sitä, että ulkoisen systeemin lähdetiedosto käännetään ja syntyvä objektitiedosto linkitetään Simnonin varsinaisten objektitiedostojen kanssa yhdeksi ohjelmaksi, jota sitten käytetään perus-Simnon-ohjelman asemasta. (Tarkemmin ottaen, pelkät ulkoisten systeemien lähdetiedostot eivät ole ainoita, jotka käyttäjä kirjoittaa, vaan lisäksi tarvitaan yksi erityinen aliohjelma, jossa on mm. lueteltu kaikkien ulkoisten systeemien aliohjelmanimet.)

Ulkoisen systeemin kuvaavaa aliohjelmaa kutsutaan kaikkiaan kahdeksassa erityyppisessä tilanteessa. Tilanteen numero on eräässä globaalissa muuttujassa. Riippumatta siitä, tarvitaanko jokaista kahdeksaa tilannetta varten jokin varsinainen toiminto, jo pelkästään selkeyden vuoksi on parasta kirjoittaa aliohjelma muotoon, jossa kuhunkin kahdeksasta mahdollisesta tilanteesta on liitetty oma toiminto. Toiminto voi tarvittaessa olla ns. tyhjä toiminto, joka ei siis tee mitään eikä kuluta aikaa.

Seuraavassa on lueteltu em. kahdeksan tilannetta lähteen [10] mukaisesti:

1) SYST-komennon antamisen yhteydessä tarvitaan ko. ulkoisen systeemin Simnon-kielinen nimi.

2) SYST-komennon antamisen yhteydessä tarvitaan ko. ulkoisen systeemin kaikkien muuttujien ja parametrien Simnon-nimet ja vastaavat staattiset muistipaikat.

3) SYST-komennon antamisen yhteydessä tarvitaan ko. ulkoisen systeemin tilamuuttujien oletusalkuarvot sekä parametrien oletusarvot, sikäli kuin oletusarvot poikkeavat nollasta.

4) SIMU-komento on annettu ilman -CONT-valitsinta, jolloin on mahdollista laskea tilamuuttujille alkuarvoja samaan tapaan kuin alkulaskentalohkossa tehtäisiin.

5) Simulointivälin ratkaisupisteessä systeemin ulostulomuuttujien ja apumuuttujien asettaminen on ajankohtaista.

6) Simulointivälin ratkaisupisteessä systeemin derivaattojen tai seuraavan tilan laskeminen on ajankohtaista.

7) Uusi ratkaisupiste on löydetty, ja on otettu ajassa vastaava askel eteenpäin. (Tässä kohdassa kannattaa lähteen [10] mukaan laskea esim. kaikki puhtaasti tulostusta varten käytettävät muuttujat.)

8) Simulointi on päättynyt. (Tilanne esiintyy kerran simuloinnin päättyessä.)

Ulkoista systeemiä kuvaava aliohjelma voi kutsua mitä tahansa aliohjelmia. Simnonnimien ja muistipaikkojen vastaavuuden ilmoittaminen tapahtuu kutsumalla eräitä UNIX-Simnonin plus-version aliohjelmia.

3 Modest

Modest on ohjelma, joka on tarkoitettu mekanististen mallien parametrien estimointiin ja kokeiden suunnitteluun. Nimi Modest on lyhenne sanoista Model Estimation.

Modest on kehitetty Kemira Oy:n Espoon tutkimuskeskuksessa. Ohjelma on ollut käytössä ainakin vuodesta 1989 alkaen. Ympäristönä on toistaiseksi pääasiallisesti ollut MicroVax-tietokoneen VMS-käyttöjärjestelmä. Työnäni olen toteuttanut Modest-version, jossa on mukana Simnon-liitäntä ja joka toimii HP-tietokoneen UNIX-käyttöjärjestelmässä.

Modestin VAX/VMS-versiosta käytän nimityksiä vanha Modest ja vanha perus-Modest. (Jälkimmäinen nimitys korostaa Simnon-liitännän puuttumista.) UNIX-versiosta käytän vastaavasti nimitystä uusi Modest.

Seuraavissa tämän luvun osioissa kerron Modestista olennaisimmat asiat. Sikäli kuin vanhan Modestin ja oman tekemäni version välillä on eroja, mainitsen niistä tapauskohtaisesti. Modest-Simnon-liitännästä ja uudesta Modestista kerron tarkemmin luvussa 6.

3.1 Modestin keskeisiä piirteitä

3.1.1 Mallit

Modestin mallit kuvaavat yleensä jatkuvan aikamuuttujan systeemejä. Modestin mallit voivat olla algebrallisia, differentiaalisia tai implisiittisiä.

Algebrallisessa mallissa systeemin tilat on lausuttu suoraan säädettävienien muuttujien ja parametrien funktiona.

Differentiaalisessa mallissa systeemin tilojen muutokset on kuvattu lausumalla tilojen derivaatat säädettävien muuttujien, parametrien ja edellisten tilojen funktiona. Kyseessä on siis Simnonin jatkuvan systeemin tapainen differentiaaliyhtälösysteemi.

Implisiittinen malli on algebrallisen mallin yleistys siten, että tilamuuttujien arvot määräytyvät annetuista yhtälöistä mutta yleisesti ottaen implisiittisesti.

3.1.2 Tehtävät

Modestilla voi suorittaa estimointia, koesuunnittelua tai simulointia. Simuloinnissa simuloidaan kerran annetun mallin mukaista systeemiä tilojen arvojen selville saamiseksi. Simnonin ja Modestin simulointikäsite on siten kuta kuinkin sama.

3.1.3 Koesarjat

Havainnot voivat jakaantua useampaan kuin yhteen koesarjaan. Jos havainnot jakaantuvat useaan koesarjaan, systeemin tilat ovat koesarjakohtaisia.

3.1.4 Parametrit

Modestissa käsite parametri määritellään hieman osion 1.3.1 esityksestä poikkeavalla tavalla. Mallin muuttuja voi Modestissa olla mallin parametri. Parametreja on neljää eri tyyppiä:

1) Globaali parametri on kaikille koesarjoille yhteinen parametri.

2) Lokaali parametri on johonkin tiettyyn koesarjaan liittyvä parametri.

3) X-muuttuja on jokin säädettävistä muuttujista. X-muuttujia optimoidaan koesuunnittelussa.

4) Alkuarvo on systeemin jonkin tilan alkuarvo.

3.2 Modestin käyttöliittymä

3.2.1 Käyttöliittymän yleisluonne

Käyttäjä antaa Modestille kaiken informaation tiedostoissa ja saa Modestilta kaiken informaation tiedostoissa.

3.2.2 Havaintojen kuvaus

Havainnot käyttäjä kuvaa yhdessä tai useammassa koesarjatiedostossa. Painoarvoja varten voi lisäksi olla omia tiedostoja.

3.2.3 Mallin kuvaus

Mallin käyttäjä kuvaa FORTRAN-tiedostossa, jotka on siis käännettävä jokaisen muutoksen jälkeen. Mallin lisäksi FORTRAN-tiedostoissa joudutaan kuvaamaan mallin ja havaintojen välinen vastaavuus sekä annettujen ja todellisten tila-alkuarvojen vastaavuus.

Simnon-liitännällä varustetussa Modestissa malli sekä em. vastaavuudet voidaan kaikki kuvata Simnon-kielisillä systeemeillä, joita käyttäjä ei joudu kääntämään.

3.2.4 Ongelman ja toimintojen tarkka määrittely

Ongelman ja toimintojen määrittelyssä tarvitaan lukuisia eri valintoja ja asetuksia. Ne kaikki käyttäjä antaa ns. nimilistatiedostossa. (Nimilista on vapaa suomennos termille namelist.) Tiedostossa on useita nimilistoja. Nimilista on FORTRAN-ohjelmien syötössä ja tulostuksessa käytettävä lista, jossa on lueteltu muuttujien arvoja nimi-arvo-pareina. Nimilistatiedostoa ei tarvitse kääntää, koska se ei ole mikään lähdetiedosto.

Seuraavassa on lueteltu Modestin VAX/VMS-version nimilistat:

- files. Siinä luetellaan muut syötetiedostot sekä kaikki tulostetiedostot, paitsi jos tyydytään käyttämään oletusarvojen mukaisia nimiä.
- problem. Siinä valitaan estimointi, koesuunnittelu tai simulointi, algebrallinen, differentiaalinen tai implisiittinen malli, käytettävä numeerinen ratkaisija, estimoitavat ja optimoitavat parametrit sekä ko. parametrien suurimmat ja pienimmät sallitut arvot.
- modelpar. Siinä annetaan globaalien ja lokaalien parametrien lähtöarvot.
- dpar. Siinä asetetaan joukko havaintoihin liittyviä valitsimia.

- setpar. Siinä luetellaan koesarjojen alkuhetket ja tilamuuttujien alkuarvot, jos ko. arvot eivät sijaitse koesarjatiedostoissa.
- print. Siinä valitaan, mitä asioita kuvaruudulle tai standarditulostusvirtaan tulostetaan.
- design. Siinä määritellään koesuunnittelussa tarvittavia optimointikriteereitä.
- jacobian. Siinä määrätään numeerisen derivoinnin askelpituus.
- bcpol. Tämä nimilista sisältää samannimisen ratkaisijan eri parametreja.
- simflex. Vrt. bcpol.
- bclsf. Vrt. bcpol.
- nconf. Vrt. bcpol.
- neqnf. Vrt. bcpol.
- broyden. Vrt. bcpol.
- ivpag. Vrt bcpol.

Simnon-liitännällä varustetussa Modest-versiossa nimilistassa luetellaan lisäksi Simnon-olioiden nimiä. Modestin parametrien, tilojen ym. vastaavuus Simnonin muuttujien kanssa määritellään niin ikään. Tiedostojen ja numeeristen ratkaisijoiden suhteen on joitakin eroja.

3.2.5 Tulostustiedostot

Seuraavassa on lueteltu Modestin VAX/VMS-version tulostustiedostojen tyypit:

- statfile. Tähän tiedostoon talletetaan estimointiin liittyvät tilastolliset tunnusluvut. Estimaatit virhearvoineen löytyvät tästä tiedostosta.
- sfile. Tähän tiedostoon talletetaan estimoitujen parametrien mukaiset tilasuureiden arvot kustakin koesarjasta.
- resufile. Tämä tiedosto on hyödyllinen, jos tuloksia halutaan tarkastella Matlab-ohjelmistolla.

Simnon-liitännällä varustetussa versiossa tuotetaan lisäksi makro, jota käyttäjä voi myöhemmin Simnonia ajaessaan kutsua.

4 Prosessien välinen kommunikointi yleisesti tarkasteltuna

Tässä luvussa kuvaan prosessien välistä kommunikointia yleisesti, ts. sitoutumatta mihinkään käyttöjärjestelmään. Jotkut esitettävistä asioista koskevat yhden prosessorin sisällä tapahtuvaa kommunikointia, jotkut puolestaan eri prosessorien välillä tapahtuvaa kommunikointia ja jotkut molempia.

Prosessorien välillä tapahtuvan kommunikoinnin osalta pyrin esittämään luvun 5 kannalta välttämättömät esitiedot.

4.1 Synkronointi

Prosessien välisen kommunikoinnin monet keskeiset ongelmat koskevat prosessien välistä synkronointia. Prosessien A ja B välillä on synkronointia, jos prosessissa A on tapahtuma x ja prosessissa B tapahtuma y siten, että x ja y eivät voi tapahtua toisistaan riippumatta. Tällöin A:n ja B:n suorituksissa on niin sanottuja synkronointitapahtumia eli sellaisia tapahtumia, joita A ei voi suorittaa yksinään eikä B yksinään vaan jotka A ja B suorittavat yhdessä. Useamman kuin kahden prosessin välisessä synkronoinnissa synkronointitapahtumat koskevat useampaa kuin kahta prosessia.

Synkronointiongelmien ratkaisuilta usein, vaikkakaan ei välttämättä aina kaikkia yhdessä, vaadittavia ominaisuuksia ovat:

- Lukkiutumattomuus. Järjestelmä ei saa joutua sellaiseen tilaan, jossa mikään tapahtuma ei ole enää mahdollinen.
- Edistyksellisyys. Sellainen äärettömän pitkä tapahtumasekvenssi ei saa olla mahdollinen, jossa kukin prosessi odottaa jotakin tapahtuvaksi eikä yksikään näistä prosessien odottamista tapahtumista koskaan tapahdu. Kuvatunlaista täydellisesti edistyksetöntä tapahtumasekvenssiä kutsutaan elolukoksi. (Elolukko on vapaa suomennos termille livelock, jota on käytetty lähteessä [2]. Eo. määritelmän elolukolle olen muodostanut lähteestä [2] lukemieni asioiden perusteella.)
- Reiluus. Jos prosessi odottaa esim. jotakin resurssia, niin odottaminen palkitaan äärellisessä ajassa.

- Vankkuus. Jos yksi prosessi kaatuu, muiden prosessien tulisi voida jatkaa toimintaansa mahdollisimman normaalisti.

Synkronointiongelmien ratkaisemista varten on kehitetty erityisiä synkronointiprimitiivejä. Tällaisia primitiivejä ovat mm. lukitustiedostot, test-and-set-muuttujat, semaforit ja monitorit.

4.2 Muistin jakaminen ja sanomien välitys

Yhden prosessorin sisällä tapahtuva prosessien välinen kommunikointi voidaan toteuttaa siten, että käyttöjärjestelmä tarjoaa prosesseille mahdollisuuden käyttää jaettua muistia ja sovellusohjelmoija huolehtii itse tarvittavista synkronoinneista. Jos sen sijaan käyttöjärjestelmä huolehtii kommunikoinnista myös tapahtumien osalta, kommunikointia sanotaan sanomapohjaiseksi. Kommunikoinnin kahtiajako sanomapohjaiseen ja ei-sanomapohjaiseen kommunikointiin on esitetty lähteessä [7].

Sanomapohjaisessa kommunikoinnissa prosessit lähettävät ainakin loogisessa mielessä toisilleen sanomia. Sanoman välitykseen käyttöjärjestelmä voi käyttää tiedonsiirtokanavaa, tiedostoa tai vaikkapa jaettua muistia. Viimeksi mainitussa tapauksessa sanoma ei välttämättä fyysisesti liiku mihinkään.

Jatkossa tarkastelen pääasiassa sanomapohjaista kommunikointia. Osion 4.8 loppuun saakka käytän sanaa yhteys synonyyminä sanalle link. Tällainen yhteys on kommunikoivien prosessien välillä välttämättä ainakin jossakin vaiheessa olemassa, kuten lähteessä [7] määritellään. (Osiossa 4.10 määrittelen käsitteen yhteys merkitsemään samaa kuin englanninkielinen käsite connection. Kaikki kommunikointi ei ole yhteydellistä sanan tässä merkityksessä. Osiosta 4.10 alkaen sekä kaikissa lukua 4 seuraavissa luvuissa käytän nimitystä yhteys osion 4.10 määritelmän mukaisessa merkityksessä.)

4.3 Kommunikaatioyhteyden looginen toteutus

Yhteyden loogiseen toteutukseen liittyy lähteen [7] mukaan seuraavia kysymyksiä:

- Kommunikoivatko prosessit toistensa kanssa suoraan vai postilaatikkoa käyttäen ?

- Onko kommunikointi symmetristä ?
- Miten yhteys perustetaan ?
- Kuinka monta prosessia voi samanaikaisesti olla kytkettynä samaan yhteyteen ?
- Kuinka monta yhteyttä voi samanaikaisesti olla kahden prosessin välillä ?
- Voivatko sanomat jonottaa yhteydessä, ja jos voivat, niin kuinka monta sanomaa voi jonottaa kerrallaan ?
- Sisältävätkö sanomat varsinaista tietoa vai vain osoitteita varsinaiseen tietoon ?
- Voivatko sanomat olla vaihtelevan mittaisia ?
- Voiko yhteyteen kytketty prosessi sekä lähettää että vastaanottaa sanomia, ja voiko yhteyteen olla kytkettynä samanaikaisesti useita mahdollisia vastaanottajia ?

4.4 Suora kommunikointi

Jos prosessit kommunikoivat toistensa kanssa suoraan ja symmetrisesti, jokaisen prosessin, joka haluaa lähettää tai vastaanottaa sanoman, täytyy täsmällisesti nimetä sanoman vastaanottaja tai lähettäjä. Symmetrisellä suoralla kommunikoinnilla on lähteen [7] mukaan seuraavat ominaisuudet:

- Yhteys syntyy, kun kaksi prosessia, A ja B, tuntevat toistensa nimet, A lähettää sanoman B:lle ja B vastaanottaa sanoman A:lta. Yhteys katoaa, kun sanoma on välitetty.
- Yhteys on aina täsmälleen kahden prosessin välinen.
- Kahden prosessin välillä voi kerrallaan olla vain yksi yhteys.

Epäsymmetrinen suora kommunikointi eroaa lähteen [7] mukaan symmetrisestä suorasta kommunikoinnista siten, että vastaanotettaessa ei tarvitse nimetä lähettäjää. Prosessi vain pyytää sanoman, ja lähettäjän nimi selviää sanoman saannin yhteydessä.

4.5 Postilaatikkokommunikointi

Postilaatikkokommunikoinnissa lähettävä prosessi jättää sanoman postilaatikkoon ja vastaanottava prosessi noutaa sanoman postilaatikosta. Kaksi prosessia voi kommunikoida keskenään vain, jos niillä on yhteinen postilaatikko. Postilaatikkokommunikoinnilla on lähteen [7] mukaan seuraavat ominaisuudet:

- Yhteisen postilaatikon omistavien prosessien välillä on aina yhteys.
- Yhteyteen voi samanaikaisesti olla kytkettyinä enemmän kuin kaksi prosessia.
- Kahden prosessin välillä on samanaikaisesti yhtä monta yhteyttä kuin ko. prosesseilla on yhteisiä postilaatikoita.

Periaatteessa voisi olla mahdollista, että kolmella eri prosessilla, P_1, P_2 ja P_3, olisi yhteinen postilaatikko, P_1 laittaisi sanoman ko. postilaatikkoon ja P_2 ja P_3 yrittäisivät samanaikaisesti ottaa ko. sanoman.

Tällaisen mahdollisuuden välttämiseksi tai syntyvän kilpatilanteen purkamiseksi on lähteen [7] mukaan olemassa seuraavanlaisia tapoja:

1) Postilaatikon ei koskaan anneta olla yhteinen useammalle kuin kahdelle prosessille.
2) Vain yksi prosessi kerrallaan voi yrittää ottaa sanoman postilaatikosta.
3) Käyttöjärjestelmä esim. valitsee mielivaltaisesti joko P_2:n tai P_3:n, muttei molempia, saamaan sanoman ja ilmoittaa vastaanottajan tunnuksen P_1:lle.

Postilaatikon voi lähteen [7] mukaan omistaa joko prosessi tai käyttöjärjestelmä. Jos prosessi omistaa postilaatikon, omistaja voi vain ottaa ko. laatikosta postia, ja muut prosessit voivat vain laittaa laatikkoon postia. Postilaatikko voi esim. syntyä automaattisesti omistajansa syntyessä ja hävitä automaattisesti omistajansa kuoltua.

Käyttöjärjestelmän omistavan postilaatikon luo joku prosessi, laatikolle voi tulla uusia käyttäjäprosesseja, eikä ko. laatikon luonut prosessi ole välttämättä missään erityisasemassa. Käyttöjärjestelmä hävittää ne omistamansa postilaatikot, joilla ei ole enää yhtään käyttäjää.

4.6 Yhteyden jonotusvara

Sanomien jonotus yhteydessä voidaan lähteen [7] mukaan toteuttaa kolmella eri perustavalla:

1) Jonotusmahdollisuutta ei ole, vaan lähettäjän on odotettava sitä, että vastaanottaja saa sanoman.

2) Yhteydessä voi jonottaa kerrallaan korkeintaan tietty määrä sanomia. Jos jono on täynnä, lähettäjän on odotettava paikan vapautumista jonossa.

3) Yhteydessä voi jonottaa kerrallaan rajoittamattoman monta sanomaa, joten lähettäjä ei joudu koskaan odottamaan.

Mikäli jonotus on käytössä, sanoman perillemeno voidaan varmistaa siten, että vastaanottaja lähettää takaisin kuittauksen. Ohjelmoija huolehtii kuittaustoimintojen toteutuksesta.

4.7 Sanomien koko tai tyyppi

Sanomat voivat lähteen [7] mukaan olla:

1) kiinteämittaisia, jolloin niiden fyysinen toteutus on helppoa.

2) vaihtelevan mittaisia, jolloin ohjelmoijan työ on helpompaa kuin kiinteämittaisten sanomien tapauksessa.

3) tyypitettyjä, mikä tukee vahvasti tyypitettyä ohjelmointia. Tämä vaihtoehto on mahdollinen lähinnä vain postilaatikkokommunikoinnissa.

4.8 Poikkeustilanteet

Jos prosessi P odottaa sanomaa prosessilta Q, joka on kuollut, P jää normaalikäytännön mukaan ikuisesti odottamaan. Tämän pattitilanteen purkamiseksi käyttöjärjestelmän on lähteen [7] mukaan joko tuhottava P tai ilmoitettava P:lle, että Q on kuollut.

Jos prosessi P lähettää sanoman prosessille Q, joka on kuollut, P jää normaalikäytännön mukaisesti ikuisesti odottamaan, ellei sanomien jonotusmahdollisuutta yhteydessä ole. Tämän pattitilanteen purkamiseksi käyttöjärjestelmän on lähteen [7] mukaan niin ikään joko tuhottava P tai ilmoitettava P:lle, että Q on kuollut. Mikäli yhteydessä sen sijaan on jonotusmahdollisuus, P ei joudu ainakaan lähettämisen takia loputtomasti odottamaan.

Muut tässä osiossa kuvattavat poikkeustilanteet koskevat lähinnä eri prosessorien välillä tapahtuvaa kommunikointia.

Sanoman katoamiseen matkan varrella voidaan lähteen [7] mukaan varautua kolmella eri perustavalla:

1) Käyttöjärjestelmä on vastuussa sanoman katoamisen havaitsemisesta ja sanoman uudelleenlähettämisestä.

2) Lähettäjä on vastuussa sanoman katoamisen havaitsemisesta ja menettelee katoamisen havaittuaan parhaaksi katsomallaan tavalla.

3) Käyttöjärjestelmä on vastuussa sanoman katoamisen havaitsemisesta ja ilmoittaa katoamisesta lähettäjälle, joka sitten menettelee parhaaksi katsomallaan tavalla.

Sanoman katoamisen havaitseminen on mahdollista vain, jos sanoman perillemeno aina varmistetaan esim. kuittauksin. Sanoma julistetaan kadonneeksi, mikäli tietoa sen perillemenosta ei ole saatu tietyn ajan kuluessa. Uudelleenlähetykseen liittyy aina se vaara, että sanoma meneekin perille kahteen kertaan. Monistumisen havaitseminen edellyttää lisätoimia, joihin en tässä puutu.

Sanoma saattaa vääristyä matkalla. Vääristymien havaitsemiseksi voidaan käyttää mm. tarkistussummia. Sanoman vääristymisen havaitsemisen ja sanoman uudelleenlähettämisen vastuu voidaan määrätä samaan tapaan kuin sanoman katoamisen tapauksessa.

4.9 OSI-malli ja Internet-malli

ISOn (International Organization for Standardization) OSI-malli (Open Systems Interconnection) kuvaa avointen järjestelmien välillä tapahtuvaa kommunikointia

seitsemän eri kerroksen avulla. Kerrokset matalimmasta korkeimpaan lukien ja kerrosten tehtävät ovat lähteen [14] mukaan:

1) Fyysinen kerros siirtää bittejä siirtotietä pitkin verkon solmulta toiselle.

2) Siirtoyhteyskerros siirtää tietoa verkon kahden vierekkäisen solmun välillä muodostaen biteistä kehyksiä, havaiten virheitä ja toipuen niistä sekä suorittaen vuon valvontaa.

3) Verkkokerros huolehtii sanomien reitityksestä verkon läpi.

4) Kuljetuskerros tarjoaa luotettavaa kaksisuuntaista kuljetuspalvelua yhteyden päästä päähän.

5) Yhteysjaksokerros tarjoaa organisoitua ja synkronoitua tiedonsiirtoa.

6) Esitystapakerros neuvottelee käytettävän siirtoesitysmuodon ja suorittaa esitystapamuunnoksia siirrettäville tiedoille.

7) Sovelluskerros toimii liitäntänä sovellusprosessin ja tietoliikennemaailman välillä.

Internet-malli kuvaa lähteen [14] mukaan järjestelmien välillä tapahtuvaa kommunikointia neljän eri kerroksen avulla siten, että alin kerros vastaa pintapuolisesti tarkasteltuna OSI-mallin kahta alinta kerrosta ja ylin kerros puolestaan OSI-mallin kolmea ylintä kerrosta. Verkkokerros ja kuljetuskerros ovat samantapaiset kuin OSI-mallissa. Internet-mallin verkkokerroksessa käytetään protokollaa IP (Internet Protocol) ja kuljetuskerroksessa protokollia TCP (Transmission Control Protocol) sekä UDP (User Datagram Protocol). Usein protokollista käytetään yhdistelmänimiä kuten TCP/IP.

4.10 Internet-mallin kuljetuskerroksen sovellusohjelmalle tarjoamien palvelujen tyyppi

Internet-mallin kuljetuskerroksen sovellusohjelmalle tarjoamien palvelujen tyyppiä kuvaavat lähteen [13] mukaan seuraavat parametrit:

- Onko palvelu yhteydellistä vai yhteydetöntä ?

- Voidaanko samanaikaisesti lähettää tietoa molempiin suuntiin kahden osapuolen välillä ?
- Missä järjestyksessä tieto saapuu määränpäähänsä ?
- Millainen on virhevalvonta ?
- Millainen on vuovalvonta ?
- Lähetetäänkö tieto tavuvirtana, vai käytetäänkö tietuerajoja ?

Yhteydellisyys merkitsee lähteen [13] mukaan sitä, että seuraavat kolme vaihetta on oltava:

1) Yhteyden muodostaminen.
2) Tiedon siirto.
3) Yhteyden purkaminen.

Yksisuuntaisessa yhteydessä on vain yksi kanava, ja tieto voi kulkea kanavaa pitkin vain yhteen suuntaan. Kaksisuuntaisessa yhteydessä on kaksi kanavaa, jolloin voidaan samanaikaisesti lähettää toiseen kanavaan tietoa ja vastaanottaa toisesta kanavasta tietoa. Vuorosuuntaisessa yhteydessä on vain yksi kanava, mutta tietoa voidaan siirtää kanavaa pitkin eri suuntiin. Yhteydettömässä palvelussa tietosähkeiksi kutsuttuja sanomia lähetetään paikasta toiseen. Tietosähkeet ovat toisistaan riippumattomia, ja jokaisen tietosähkeen on sisällettävä kaikki kuljetuksessa tarvittava informaatio.

Virhevalvonnalla tarkoitetaan tiedon katoamisen, monistumisen tai vääristymisen havaitsemista sekä tarpeen mukaan suoritettavaa tiedon uudelleenlähettämistä. Vuovalvonnalla pyritään siihen, että lähettäjä ei lähetä tietoa nopeammin kuin vastaanottaja ehtii sitä prosessoida. Tavuvirtapalvelussa tieto siirtyy sovellusohjelman kannalta tavuvirtana eikä mahdollinen paketointi kuulu sovellusohjelman tehtäviin.

Protokolla TCP tarjoaa lähteen [13] mukaan kaksisuuntaisyhteydellistä tiedon järjestyksen säilyttävää tavuvirtapalvelua täydellä virhe- ja vuovalvonnalla varustettuna. (Vrt. OSI-mallin kuljetuskerroksen kuvaus.) Protokolla UDP tarjoaa lähteen [13] mukaan yhteydetöntä palvelua ilman virhe- tai vuovalvontaa. Tietosähkeiden perillemenojärjestys on myöskin sattumanvaraista.

4.11 Asiakas-palvelija-malli

Verkkosovellutusten standardimalliksi on muodostunut ns. asiakas-palvelija-malli. Ko. malli on nykyään melko suosittu myös yhden prosessorin sovellutuksissa. Mallin perusidea on se, että eri palveluja varten on olemassa palvelijaprosesseja, joilta asiakasprosessit pyytävät palveluja. Palvelijaprosessia ei luoda vain yhtä palvelusta varten, vaan palvelijaprosessi elää omaa elämäänsä odottaen ja täyttäen palvelupyyntöjä.

Palvelijaprosesseja on lähteen [13] mukaan kahta eri perustyyppiä:

1) Iteratiivinen palvelija suorittaa koko palveluksen itse.

2) Rinnakkainen palvelija ottaa palvelupyynnön vastaan mutta ei suorita itse varsinaista palvelusta vaan luo prosessin, joka suorittaa varsinaisen palveluksen. Kun näin menetellään, palvelupyyntöjen käsittely on nopeaa eikä riipu palvelusten suorittamiseen kuluvasta ajasta. Nimitys rinnakkainen palvelija tulee ilmeisestikin siitä, että palvelija tavallaan palvelee useita asiakkaita rinnakkain.

Yhteydellisissä protokollissa käytetään lähteen [13] mukaan tavallisesti rinnakkaisia palvelijoita ja yhteydettömissä protokollissa puolestaan tavallisesti iteratiivisia palvelijoita.

5 Prosessien välinen kommunikointi UNIXissa

Tässä luvussa esittelen UNIXissa toteutettuja prosessien välisen kommunikoinnin mekanismeja. Mukana ovat tärkeimmät yhden prosessorin sisällä tapahtuvaan kommunikointiin tarjolla olevat systeemikutsutason mekanismit. Eri prosessorien välillä tapahtuvan kommunikoinnin mahdollisuuksistakin kerron jonkin verran, mutta tältä osin esitys ei ole millään tavoin kattava.

Systeemikutsutason yläpuolisia kommunikaatiomekanismeja en käsittele kuin lähinnä eri prosessorien välillä tapahtuvan kommunikoinnin kohdalla ja silloinkin ensisijaisesti vain siksi, että joissakin UNIX-järjestelmissä verkko-ohjelmointi suoraan systeemikutsuja käyttäen on erittäin hankalaa eikä nykypäivänä enää kovin mielekästä. Esim. ikkunointijärjestelmien tarjoamiin mahdollisuuksiin en sen sijaan puutu lainkaan.

5.1 Signaalit

Signaalit informoivat prosesseja lähteen [1] mukaan mm. seuraavantyyppisistä tapahtumista:

- Prosessi on tavalla tai toisella yrittänyt ylittää valtuutensa.
- Käyttöjärjestelmä ei syystä tai toisesta pysty suorittamaan jotakin toimintoa.
- Jokin prosessi on tarkoituksella lähettänyt signaalin.
- Jokin fyysinen poikkeustapahtuma, kuten keskeytysnäppäimen painallus, on tapahtunut.
- Prosessin odottama jokin tapahtuma, esim. lapsiprosessin kuolema, on tapahtunut. (Signaali herättää nukkuvan prosessin.)

Kun jotain pitää saada tapahtumaan todella nopeasti ja ilman epävarmuutta aiheuttavia välikäsiä, signalointi on paras ellei ainoa vaihtoehto. Prosessit voivat esim. tuhota toisiaan signaalien avulla.

Kun prosessi saa signaalin, se normaalisti suorittaa jonkin käyttöjärjestelmän määräämän toiminnon. Virhetilanteiden signaalit esim. tyypillisesti johtavat exit-toimintoon, jolla signaalin saanut prosessi lopettaa toimintansa. Prosessi voi kuitenkin

halutessaan myös ottaa signaaleja kiinni ja määritellä kiinniottamiensa signaalien saapumista seuraavat toiminnot. Tällä tavoin prosessi voi vaikkapa olla tyystin välittämättä jostakin signaalista. Kaikki signaalit eivät ole kiinniotettavissa. Esim. ns. KILL-signaali, jolla prosessit voivat tuhota toisiaan, ei ole kiinniotettavissa. Eräs hyödyllinen signaloinnin mahdollistama asia on se, että prosessi voi odottaa lapsiprosessiensa kuolemaa ja siivota lastensa jälkiä.

Prosessijäljityksessä, lähteen [1] mukaan, prosessi seuraa ja mahdollisesti myös ohjaa askel askeleelta toisen prosessin toimintaa. Prosessijäljitystä tekevät tyypillisesti esim. debuggerit. Prosessijäljitys on toteutettu signaalinkäsittelytoimintojen avulla.

5.2 Tavanomaiset tiedostot

Tavanomaisilla tiedostoilla tarkoitan tiedostoja, jotka luodaan tavallisilla tiedostonluomiskutsuilla ja joista luetaan ja joihin kirjoitetaan tavallisin tiedostonkäsittelykutsuin.

Yhteisten tiedostojen käyttö on vanhin prosessien välisen kommunikoinnin muoto. (Ei-moniajoympäristöissäkin prosessit tavallaan kommunikoivat yhteisiä tiedostoja käyttäen, joskin kommunikointi on tällöin perin yksisuuntaista.)

Tavanomaisten tiedostojen käytössä on se ongelma, että kirjoittaminen ja lukeminen eivät voi tapahtua samanaikaisesti. Nykyisin on sentään jo mahdollista lukita erillisiä tiedoston osia, jolloin yksi prosessi voi kirjoittaa lukittuun osaan ja muut prosessit voivat lukea lukitsemattomia osia.

5.3 Jaettu muisti

Jaettu muisti, semaforit ja viestijonot ovat lähteen [1] mukaan UNIX System V:n ns. IPC-pakkauksen osat. System V:stä ne on kuitenkin omaksuttu melko moneen muuhunkin UNIX-järjestelmään. Semaforit ja viestijonot on toteutettu jaetun muistin avulla, joten IPC:n kommunikointimekanismeilla on ainakin periaatteessa nopeuden suhteen etulyöntiasema tiedostopohjaisiin mekanismeihin nähden. (System V:n IPC:tä lukuunottamatta kaikki tässä luvussa esitettävät kommunikointimekanismit ovat tiedosto- tai tiedonsiirtopohjaisia. Tiedostopohjaisten mekanismien aseena nopeustaistelussa on puskurointi. Tietoa pidetään puskurissa, eikä esim. levyllä, niin paljon kuin mahdollista.)

Jaettuun muistiin kirjoittamista ja jaetusta muistista lukemista varten ei tarvita mitään systeemikutsuja, vaan jaetussa muistissa olevia muuttujia asetetaan ja luetaan kuten mitä tahansa muuttujia.

Jaettu muisti täytyy kuitenkin varata ja vapauttaa systeemikutsuin. Jaettuun muistiin liittyvät systeemikutsut lähteen [1] mukaan:

- shmget-kutsulla joko luodaan uusi jaetun muistin alue tai haetaan tunnusnumero olemassaolevaan jaettuun muistialueeseen. Jaetulla muistialueella on oma avain vähän samaan tapaan kuin tiedostolla on nimi.
- shmat-kutsulla kiinnitetään jaettu muistialue prosessin virtuaaliseen osoiteavaruuteen.
- shmdt-kutsulla irrotetaan jaettu muistialue prosessin virtuaalisesta osoiteavaruudesta.
- shmctl-kutsulla voidaan tutkia ja asettaa jaettuun muistialueeseen liittyviä parametreja. Jaetun muistialueen poistaminen tapahtuu tällä kutsulla.

Sovellusohjelmoija joutuu itse huolehtimaan jaetun muistin käytössä mahdollisesti tarvittavista synkronoinneista. Synkronointi on yleisessä tapauksessa helpointa toteuttaa semaforien avulla.

5.4 Semaforit

UNIX System V:n semafori koostuu lähteen [1] mukaan seuraavista komponenteista:

- Semaforin arvo. (Klassisen määritelmän mukaanhan semafori ei muuta olekaan kuin kokonaisluku.) Arvo voi olla mikä tahansa positiivinen kokonaisluku.
- Semaforia viimeksi käsitelleen prosessin numero.
- Semaforin arvon nousua odottavien prosessien lukumäärä. Prosessi voi odottaa semaforin arvon nousua jonkun muunkin luvun kuin nollan yläpuolelle.
- Semaforin arvon nollautumista odottavien prosessien lukumäärä. (Arvoa nolla ei tietenkään voi enää pienentää, vaan odottamisella on jokin muu tarkoitus.)

System V:ssä usean semaforin joukkoon voidaan kohdistaa yhtäaikaisesti eri operaatioita. Näin on mahdollista välttää esim. sellainen tilanne, jossa prosessi A on lukinnut semaforin x ja yrittää lukita semaforia y ja prosessi B on lukinnut semaforin y ja yrittää lukita semaforin x.

Semaforeihin liittyvät systeemikutsut lähteen [1] mukaan:

- semget-kutsulla luodaan uusi semaforijoukko tai haetaan tunnusunumero olemassaolevaan semaforijoukkoon. Semaforijoukolla on oma avain.
- semop-kutsulla semaforijoukkoon kohdistetaan joukko yht'aikaisia operaatioita.
- semctl-kutsulla voidaan tehdä semaforien alkuasetuksia, lukea semaforien arvoja, poistaa semaforijoukko ja suorittaa erinäisiä semaforijoukkoon kohdistuvia ohjaustoimia.

5.5 Viestijonot

Viestijono on jaetun muistin avulla toteutettu synkroninen viestintämekanismi. Monta prosessia voi hallitusti kommunikoida keskenään yhtä viestijonoa käyttäen.

Viestijonoihin liittyvät systeemikutsut lähteen [1] mukaan:

- msgget-kutsulla luodaan uusi viestijono tai haetaan tunnusnumero olemassaolevaan viestijonoon. Viestijonolla on oma avain.
- msgsnd-kutsulla laitetaan viesti viestijonoon. Viestille annetaan tyyppi.
- msgrcv-kutsulla otetaan viestijonosta se halutuntyyppisistä viesteistä, joka on ensiksi lähetetty. (Muitakin valintamahdollisuuksia on, mutta edellä mainittu on ehkä tärkein.)
- msgctl-kutsulla voidaan tutkia ja asettaa viestijonon parametreja. Viestijonon poistaminen tapahtuu tällä kutsulla.

Sopiva viestin tyyppi on esimerkiksi sen prosessin numero, jolle viesti on tarkoitus lähettää.

5.6 Nimeämättömät putket

Nimeämätön putki on tiedostopohjainen yksikanavainen vuorosuuntaisyhteydellinen kommunikointimekanismi. Nimeämättömät putket ovat UNIXin peruskalustoa ja ne on toteutettu kaikissa UNIX-järjestelmissä. Putki on luonteeltaan FIFO.

Putkeen voi samanaikaisesti olla kiinnittyneenä useita lukijoita ja kirjoittajia. Prosessi voi lukea putkesta, jos ko. prosessilla on ko. putken avoin lukupää. Vastaavasti avoin kirjoituspää antaa oikeuden kirjoittaa putkeen. Yksinkertaisin ja toisaalta eniten käytetty tapa käyttää putkea on se, että putkea hyödyntävät kaksi ja vain kaksi prosessia, joista toinen vain kirjoittaa putkeen ja toinen vain lukee putkesta. Putken käyttö useamman kuin kahden prosessin väliseen kommunikointiin vaatii erillistä ja täsmällistä synkronointia prosessien välillä.

Nimeämättömän putken välityksellä voivat kommunikoida vain keskenään esi-isä-jälkeläis-suhteessa olevat prosessit. Jos putken luonut prosessi myöhemmin luo itselleen lapsiprosesseja, lapsiprosessit perivät avoinna olevat putken päät. Lapsiprosessi voi luonnollisestikin luoda omia lapsiprosessejaan, ja niin edelleen. Periaatteessa jopa kaikilla putken luoneen prosessin jälkeläisillä on mahdollisuus kommunikointiin kyseisen putken välityksellä.

Nimeämättömiin putkiin liittyviä systeemikutsuja kuvattuna lähteen [13] mukaisesti:

- pipe-kutsulla luodaan putki. Tuloksena saadaan kaksi tiedostoviitenumeroa (file descriptor), toinen lukemista ja toinen kirjoittamista varten.
- dup-kutsulla jokin tietovirta voidaan suunnata uudelleen joko menemään putkeen tai tulemaan putkesta.
- fcntl-kutsulla voidaan tehdä ja lukea hyvin monenlaisia avoimeen putkeen liittyviä asetuksia. Esim. putken pää voidaan laittaa odottamisen kieltävään tilaan, kun putken pää oletusarvoisesti muuten on odottamisen sallivassa tilassa.
- read-kutsulla luetaan putkesta kuten mistä tahansa tiedostosta. Jos yhdelläkään prosessilla ei ole putken kirjoituspäätä auki, kutsusta palataan välittömästi. Jos putki on tyhjä, putken lukupää on odottamisen sallivassa tilassa eikä käytetä viivytykset kieltävää valitsinta, niin jäädään odottamaan sitä, että joku kirjoittaa putkeen. Odottaminen katkeaa, jos viimeinen avoinna

ollut kirjoituspää suljetaan. Jos putki on tyhjä ja joko putken lukupää on odottamisen kieltävässä tilassa tai käytetään viivytykset kieltävää valitsinta, readista palataan välittömästi ja nolla tavua todetaan luetuksi. Jos putki ei ole tyhjä ja lukija pyytää enemmän tavuja kuin putkessa on, niin ei jäädä odottamaan lisää tavuja, vaan luetaan ne tavut, mikä putkessa sillä hetkellä on.

- write-kutsulla kirjoitetaan putkeen kuten mihin tahansa tiedostoon. Jos yhdelläkään prosessilla ei ole putken lukupäätä auki, kutsusta palataan välittömästi. Jos kaikki kirjoitettavaksi tarkoitetut tavut eivät kutsuhetkellä mahdu putkeen, putken kirjoituspää on odottamisen sallivassa tilassa eikä käytetä viivytykset kieltävää valitsinta, odotetaan niin kauan, että tilaa tulee riittävästi. Odottaminen katkeaa kuitenkin, jos viimeinen avoinna ollut lukupää suljetaan. Jos kaikki kirjoitettavaksi tarkoitettut tavut eivät kutsuhetkellä mahdu putkeen ja joko putken kirjoituspää on odottamisen kieltävässä tilassa tai käytetään viivytykset kieltävää valitsinta, writesta palataan välittömästi ja nolla tavua todetaan kirjoitetuksi.

  Odottamisen sallivan tilan vallitessa ei koskaan tyydytä kirjoittamaan esim. vain osaa tavuista. Sen sijaan kirjoitustapahtuman atomisuus on taattu vain tiettyä arvoa pienemmälle tavumäärälle. Jos samanaikaisesti on useita kirjoittajia, voivat rinnakkaisten write-kutsujen suoritukset pahimmassa tapauksessa limittyä.

- close-kutsulla suljetaan putken lukupää tai kirjoituspää.

Tässä yhteydessä en voi olla mainitsematta kolmea lähteessä [13] esiteltyä standardi-I/O-pakkauksen funktiota:

- fdopen-funktio liittää tiedostoviitenumeroon tiedosto-osoittimen. Tiedosto-osoittimen hankkimisen jälkeen voidaan siirtyä käyttämään standardi-I/O-pakkauksen funktioita aivan kuin kyseessä olisi tavanomainen tiedosto.

- popen-funktio muistuttaa fopen-funktiota (tavanomaisen tiedoston avaus, ei siis fdopen), mutta tiedostonimiargumentin sijasta annetaankin shell-komennon nimi. Kutsuvalle prosessille luodaan lapsiprosessi, jota vastaava ohjelma on shell-komennon mukainen. Kutsuva prosessi saa pyyntönsä mukaisen pään

putkesta ja lapsiprosessi toisen pään. Lapsiprosessin standardisyötevirta suunnataan uudelleen tulemaan putkesta, tai lapsiprosessin standarditulostusvirta suunnataan uudelleen menemään putkeen.

- pclose-funktiolla suljetaan putki. Funktio palauttaa lapsiprosessin exit-statuksen, joten lapsiprosessi siis häviää viimeistään silloin kuin putkikin.

Käyttäjän ei tarvitse huolehtia nimeämättömän putken hävittämisestä, vaan käyttöjärjestelmä hävittää nimeämättömän putken, kun ko. putkeen ei ole joko enää yhtään avointa lukupäätä tai yhtään kirjoituspäätä. Ennen putken hävittämistä mahdollisten avointen turhiksi käyneiden putken päiden omistajat saavat käyttöjärjestelmältä asianmukaiset signaalit.

5.7 Nimetyt putket

Nimetty putki on UNIX System V:n tiedostopohjainen kommunikointimekanismi, joka on tosin omaksuttu System V:stä moniin muihinkin UNIX-järjestelmiin. (Nimitystä nimetty putki on käytetty lähteessä [1]. Lähteessä [13] käytetään vastaavasta oliosta nimitystä FIFO.)

Nimetty putki ei juurikaan eroa nimeämättömästä putkesta. Ainoa merkittävä ero on nimen olemassaolo. Kun putkella on nimi, mielivaltaiset prosessit voivat kommunikoida putken välityksellä.

Nimettyihin putkiin liittyviä systeemikutsuja kuvattuna lähteen [13] mukaisesti:

- mknod-kutsulla luodaan nimetty putki, minkä jälkeen putki esim. näkyy hakemistolistauksessa.

- open-kutsulla putki avataan kuten mikä tahansa tiedosto joko lukemista tai kirjoittamista varten. Avattava pää saa tilakseen odottamisen sallivan tilan, ellei käytetä viivytykset kieltävää valitsinta, jolloin tilaksi tulee odottamisen kieltävää tila. Kirjoituspään (vastaavasti lukupään) avaamista yrittävä prosessi odottaa, jos putken yksikään lukupää (vastaavasti kirjoituspää) ei ole auki eikä putken yhtään lukupäätä (vastaavasti kirjoituspäätä) ole vielä yritetty avata. Jos kuitenkin käytetään viivytykset kieltävää valitsinta, niin openista palataan välittömästi, mikäli pyyntöä ei voida heti täyttää.

- dup-kutsu toimii kuten nimeämättömän putken tapuksessa.

- fcntl-kutsu toimii kuten nimeämättömän putken tapauksessa.
- write-kutsu toimii kuten nimeämättömän putken tapuksessa.
- read-kutsu toimii kuten nimeämättömän putken tapuksessa.
- close-kutsu toimii kuten nimeämättömän putken tapuksessa.
- unlink-kutsulla nimetty putki pyyhitään pois käyttäjän hakemistosta samalla tavoin kuin mikä tahansa tiedosto. Käyttöjärjestelmä hävittää nimetyn putken, kun unlink-kutsu on suoritettu ja ko. putkeen ei ole joko enää yhtään avointa lukupäätä tai yhtään kirjoituspäätä. Ennen putken hävittämistä mahdollisten avointen turhiksi käyneiden putken päiden omistajat saavat käyttöjärjestelmältä asianmukaiset signaalit.

Systeemikutsujen sijasta standardi-I/O-pakkauksen funktioilla voidaan tehdä kaikki asiat putken luomista lukuunottamatta.

5.8 Vastakkeet

Vastake on sekä yhden prosessorin sisäiseen kommunikointiin että eri prosessorien välillä tapahtuvaan kommunikointiin soveltuva BSD UNIXin monikanavainen kommunikointimekanismi, joka on BSD UNIXista omaksuttu moniin muihin UNIX-järjestelmiin. (Vastake on vapaa suomennos termille socket.)

Vastakkeet ryhmitellään kommunikaatio-ominaisuuksiensa perusteella eri kohdealueiden vastakkeisiin. Kohdealueita ovat lähteen [13] mukaan muun muassa:

- UNIX-kohdealue. Sen vastakkeiden avulla yhden prosessorin sisällä prosessit voivat kommunikoida keskenään. UNIX-kohdealueen vastakkeet ovat tiedostopohjaisia.
- Internet-kohdealue. Sen vastakkeiden avulla eri koneissa sijaitsevat prosessit voivat kommunikoida keskenään.
- Xerox NS -kohdealue. (Vrt. Internet-kohdealue.)

UNIX-kohdealueen ja Internet-kohdealueen vastakkeet on toteutettu jokaisessa sellaisessa UNIX-järjestelmässä, jossa vastakkeet on toteutettu. Muut kohdealueet ovat sen sijaan ainakin vielä toistaiseksi harvinaisuuksia.

Vastakkeita on mm. seuraavanlaisia tyyppejä, jotka on kuvattu lähteessä [13]:

- Virtavastakkeet tarjoavat kaksisuuntaisyhteydellistä tiedon järjestyksen säilyttävää tavuvirtapalvelua täydellä virhe- ja vuovalvonnalla varustettuna. (Virtavastake on vapaa suomennos termille stream socket.) Internet-kohdealueen virtavastakkeet käyttävät protokollaa TCP. UNIX-kohdealueen virtavastakkeille eo. kuvaus pätee kuta kuinkin, joskin esim. virhevalvontaa ei tarvita.

- Tietosähkevastakkeet tarjoavat yhteydetöntä palvelua ilman virhe- tai vuovalvontaa. (Tietosähkevastake on vapaa suomennos termille datagram socket.) Tietosähkeiden perillemenojärjestys on sattumanvaraista. Internet-kohdealueen tietosähkevastakkeet käyttävät protokollaa UDP. UNIX-kohdealueen vastakkeiden suhteen eo. kuvaus pätee kuta kuinkin. (UNIX-kohdealueen tietosähkevastakkeilla esim. vuovalvonnan puute merkitsee sitä, että sovellusohjelmoijan on varauduttava lähettämään tieto uudelleen, sillä vastaanottaja ei välttämättä ehdi tyhjentää puskuria siinä tahdissa kuin lähettäjä sitä täyttää.)

- Peräkkäispakettivastakkeet ovat muuten kuin virtavastakkeet, paitsi että tietuerajoja käytetään. (Peräkkäispakettivastake on vapaa suomennos termille sequenced packet socket.) UNIX- ja Internet-kohdealueisiin tätä vastaketyyppiä ei ole toteutettu.

- Luotettavan sanomanvälityksen vastakkeet ovat tietosähkevastakkeiden paranneltu versio siinä mielessä, että sanomien perillemeno on luotettavaa. (Luotettavan sanomanvälityksen vastake on vapaa suomennos termille reliably delivered message socket.) UNIX- ja Internet-kohdealueisiin tätä vastaketyyppiä ei ole toteutettu.

- Raa'at vastakkeet tarjoavat mahdollisuuden esim. kehittää omia protokollia, sillä raakojen vastakkeiden protokollat ovat alempitasoisia kuin muiden vastakkeiden. (Raaka vastake on vapaa suomennos termille raw socket.) Internet-kohdealueen raa'at vastakkeet käyttävät protokollaa IP. UNIX-kohdealueeseen tätä vastaketyyppiä ei ole toteutettu.

UNIX-kohdealueen virtavastakkeet muistuttavat hyvin paljon putkia. Yksi putki tarjoaa kuitenkin vain yhden yhteyden, ja ko. yhteys on vuorosuuntainen. Yhden virtavastakkeen kautta voidaan ylläpitää samanaikaisesti useita kaksisuuntaisia yhteyksiä. Jokaista yhteyttä varten on kaksi puskuria. Yhteydessä on kaksi vastapuolta, mutta prosesseja voi olla useita kuten putken tapauksessa. Tiedon kulun nopeuden suhteen UNIX-kohdealueen virtavastakkeet ovat samanlaisia kuin putket. BSD UNIXissa nimeämättömät putket ovat itse asiassa vain naamioituja UNIX-kohdealueen virtavastakkeita.

Vastakkeisiin liittyviä systeemikutsuja kuvattuna lähteen [13] mukaisesti:

- socket-kutsulla haetaan vastakeviitenumero (socket descriptor) valitsemalla kohdealue, tyyppi ja protokolla. (Esim. Internet-kohdealueen vastakkeen tyyppi implikoi protokollan. Usein valitaankin protokollaksi 0, eli annetaan käyttöjärjestelmän päättää protokollasta.)

- dup-kutsulla jokin tietovirta voidaan suunnata uudelleen joko menemään vastakkeeseen tai tulemaan vastakkeesta.

- setsockopt-kutsulla voidaan tehdä hyvin monenlaisia vastakkeeseen liittyviä asetuksia. Vastake voidaan esim. laittaa odottamisen kieltävään tilaan, kun vastake oletusarvoisesti muuten on odottamisen sallivassa tilassa.

- getsockopt-kutsulla voidaan lukea vastakkeeseen liittyviä asetuksia.

- fcntl-kutsulla voidaan tehdä ja lukea vastakkeeseen liittyviä asetuksia.

- ioctl-kutsulla voidaan niin ikään tehdä ja lukea vastakkeeseen liittyviä asetuksia.

- socketpair-kutsulla hankitaan vastakeviitenumeropari hieman samaan tapaan kuin pipe-kutsulla tiedostoviitenumeropari. Vastakeviitenumeroa voidaan kuitenkin käyttää sekä lukemisessa että kirjoittamisessa. Socketpair-kutsu huolehtii siitä, että voidaan ryhtyä suoraan lukemaan tai kirjoittamaan. Esim. virtavastakkeen kyseessä ollessa yhteys on siis valmis. Vastakkeen kohdealue, tyyppi ja protokolla valitaan socketpair-kutsussa muodollisesti samaan tapaan kuin socket-kutsussa. Toistaiseksi socketpair on toteutettu vain UNIX-kohdealueeseen. Huomattakoon, että mitään nimeä ei socketpair-kutsu vastakkeelle anna niin kuin ei socket-kutsukaan anna.

- bind-kutsulla nimeämätön vastake nimetään. UNIX-kohdealueen vastakkeen nimi on tiedoston nimi ja Internet-kohdealueen vastakkeen nimi puolestaan porttinumeron ja Internet-osoitteen yhdistelmä. (Esim. rinnakkainen TCP-palvelija käyttää porttinumeroita voidakseen palvella samanaikaisesti useita asiakkaita.) UNIX-kohdealueen vastake mm. näkyy hakemistolistauksessa.

- connect-kutsulla asiakasprosessi muodostaa yhteyden palvelijaan. (Asiakkaan ja palvelijan kiinnittäminen tällä tavalla yksinkertaistaa asioita. Todellisuudessa mikään ei esimerkiksi estä connectia kutsunutta prosessia toimimasta yhteyden myöntäneen prosessin yksityisenä palvelijana yhteyden muodostamisen jälkeen.) connect-kutsulla vastakeviitenumero pyritään liittämään nimeltä mainittuun vastakkeeseen. bind-kutsua asiakkaan ei tarvitse käyttää, vaan riittää, että palvelija on tehnyt bind-kutsun.

  connect-kutsua voi käyttää myös yhteydettömän palvelun asiakas. Vaikutus on tällöin se, että kaikki asiakkaan kyseiseen vastakkeseen myöhemmin lähettämä tieto menee connect-kutsun mukaiseen osoitteeseen ja kaikki asiakkaan kyseisestä vastakkeesta lukema tieto tulee connect-kutsun mukaisesta osoitteesta.

- listen-kutsulla yhteydellinen palvelija ilmoittaa valmiudestaan ottaa vastaan yhteyspyyntöjä annettuun vastakkeeseen. Kutsussa määritellään, kuinka monta yhteyspyyntöä voidaan laittaa jonoon sillä aikaa, kun palvelija on suorittamassa palvelusta.

- accept-kutsulla palvelija odottaa yhteyspyyntöjä. accept-kutsu palauttaa uuden vastakeviitenumeron, joka viittaa samaan vastakkeeseen kuin alkuperäinenkin. Palvelija voi antaa accept-kutsun palauttaman viitenumeron lapsiprosessin käyttöön ja vastaanottaa seuraavan yhteyspyynnön vanhaa viitenumeroa käyttäen.

- read-kutsulla vastakkeesta voi lukea kuin putkesta, ainakin virtavastakkeen tapauksessa.

- write-kutsulla vastakkeeseen voi kirjoittaa kuin putkeen, ainakin virtavastakkeen tapauksessa. Mikäli kohdealue on jokin muu kuin UNIX-kohdealue, on kuitenkin mahdollista, että tavuista kirjoitetaan vain sen verran kuin kerralla puskuriin mahtuu. Write-kutsua on siis varauduttava toistamaan riittävän monta kertaa. Tässä mielessä write-kutsu on itse asiassa varsin symmet-

rinen read-kutsun kanssa. (UNIX-kohdealueen tapauksessakin on hyvä totutella käyttämään write-toistoa, jotta vastakkeita käyttävät ohjelmat eivät olisi liian riippuvaisia vastakkeiden kohdealueista.)

- recv-kutsu on read-kutsua monipuolisempi vastaanottokutsu. (Yksityiskohtiin en puutu.)
- send-kutsu on write-kutsua monipuolisempi lähetyskutsu.
- sendto-kutsulla voidaan kirjoittaa nimettyyn vastakkeeseen ilman, että yhteyttä olisi muodostettu.
- recvfrom-kutsullla voidaan lukea nimetystä vastakkeesta ilman, että yhteyttä olisi muodostettu.
- close-kutsulla yhteys puretaan niin, että kanavaan ei jää tavuja.
- shutdown-kutsulla voidaan yhteys purkaa osittain siten, että joko lukeminen tai kirjoittaminen kielletään. Kutsulla voi myös purkaa yhteyden kokonaan.
- select-kutsulla palvelija voi odottaa samanaikaisesti useihin eri vastakkeisiin kohdistuvia yhteyspyyntöjä. Käyttöjärjestelmä herättää palvelijan, kun johonkin vastakkeeseen kohdistuu yhteyspyyntö.
- unlink-kutsulla UNIX-kohdealueen vastake pyyhitään pois käyttäjän hakemistosta samalla tavoin kuin mikä tahansa tiedosto. unlink-kutsu ei automaattisesti aiheuta vastakkeen hävittämistä, vaan hävittäminen tapahtuu vasta, kun yhtään ko. vastakkeeseen liittyvää yhteyttä ei enää ole. (Vrt. unlink nimetyn putken tapuksessa.)

5.9 Kaksoisvirrat

Kaksoisvirta on mekanismi, joka on kehitetty parantamaan laiteohjainten ja protokollien ohjelmoinnin modulaarisuutta. (Kaksoisvirta on vapaa suomennos termin streams yksikkömuodolle.) Kaksoisvirrat on toteutettu ainakin UNIX System V:ssä.

Kaksoisvirta on lähteen [13] mukaan kaksisuuntainen yhteys käyttäjän prosessin ja joko laiteohjaimen tai näennäislaiteohjaimen välillä.

Kaksoisvirta koostuu lähteen [13] mukaan kahdesta lineaarisesti linkitetystä listasta. Toinen lista on syöttölista ja toinen tulostuslista. Kummassakin listassa on saman verran jäseniä. Syöttölistan alusta lukien k:s jäsen ja tulostuslistan lopusta lukien k:s jäsen ovat samalla hierarkiatasolla ja muodostavat moduulin. Käyttäjän prosessi on välittömässä yhteydessä kaksoisvirran ylimpään moduuliin. Alin moduuli vastaa aitoa tai näennäistä laiteohjainta.

Oletusarvoisesti kaksoisvirta sisältää em. kaksi moduulia. Käyttäjä voi kuitenkin lisätä väliin moduuleja, ottaa lisäämiään moduuleja pois kaksoisvirrasta, lisätä uusia moduuleja jne.

Näennäislaiteohjain voi olla esimerkiksi Ethernet-ajuri. Pohjalta lukien seuraava moduuli voi vastaavasti olla vaikkapa IP-moduuli, sitä seuraava TCP-moduuli jne. Kaksoisvirrat tarjoavat joustavan tavan ohjelmoida protokollia kerroksittain.

Kaksoisvirtoihin liittyviä systeemikutsuja kuvattuna lähteen [13] mukaisesti:

- open-kutsulla avataan kaksoisvirta. Kaksoisvirta sijaitsee UNIXin tiedostojärjestelmän hakemistossa "/dev".
- dup-kutsulla jokin tietovirta voidaan suunnata uudelleen joko menemään kaksoisvirtaan tai tulemaan kaksoisvirrasta.
- ioctl-kutsulla luetaan ja tehdään kaksoisvirtaan liittyviä asetuksia. Eritoten moduulien lisääminen ja poistaminen tapahtuu tällä kutsulla.
- read-kutsulla luetaan niin kuin fyysisiltä laitteilta on tapana lukea. (ioctl:llä huolehditaan tarvittavista asetuksista.)
- write-kutsulla kirjoitetaan niin kuin fyysisille laitteille on tapana kirjoittaa.
- close-kutsulla kaksoisvirta suljetaan.
- getmsg-kutsulla luetaan sanomia. (Vrt. viestijonot.)
- putmsg-kutsulla kirjoitetaan sanomia.
- poll-kutsulla odotetaan annettuihin kaksoisvirtoihin kohdistuvia I/O-signaaleja. (Vrt. select vastakkeilla.)

5.10 TLI

TLI eli Transport Layer Interface on UNIX System V:hen kuuluva eri prosessorien väliseen kommunikointiin tarkoitettu mekanismi. TLI tarjoaa liitännän OSI-mallin tai Internet-mallin mukaiseen kuljetuskerrokseen. TLI:n funktiot eivät ole systeemifunktioita. TLI on lähteen [13] mukaan toteutettu kaksoisvirtojen avulla. TLI piilottaa kaksoisvirrat ja kaksoisvirtoihin kohdistuvat operaatiot käyttäjältä. TLI:n funktiot muistuttavat lähteessä [13] annetun kuvauksen perusteella sekä nimiltään että ominaisuuksiltaan vastakkeisiin liittyviä systeemifunktioita. Funktioiden kuvaus sivuutettakoon.

5.11 HP:n NetIPC

Hewlett Packardin NetIPC on HP:n UNIX-koneiden väliseen kommunikointiin kehitetty mekanismi. HP:n NetIPC:n funktiot eivät ole systeemifunktioita. HP:n NetIPC on lähteen [6] mukaan toteutettu vastakkeiden avulla. Vastakkeisiin viittaaminen tapahtuu NetIPC:ssä tavanomaisten vastakeviitenumeroiden avulla. Käyttäjä voi lähteen [6] mukaan tietyssä mitassa käyttää NetIPC:n funktioita ja vastakkeisiin liittyviä systeemifunktioita sekaisin. NetIPC:tä käytettäessä verkko-ohjelmointi on jossain määrin yksinkertaisempaa kuin pelkästään systeemikutsuja käytettäessä. NetIPC:n funktioiden kuvaus sivuutettakoon.

6 Modest-Simnon-liitännän keskeiset piirteet

Tässä luvussa esitän Modest-Simnon-liitännän ja tekemäni uuden Modest-version keskeiset piirteet sekä tehtyjen ratkaisujen perustelut. Yritän keskittyä asioihin nimien sijasta. Esimerkiksi aliohjelmien tai muuttujien nimiä vältän käyttämästä.

6.1 Prosessit

Uusi Modest koostuu käytännöllisesti katsottuna kolmesta ohjelmasta: varsinainen Modest-ohjelma, Simnon-ohjelma sekä valvova ohjelma. Jokainen näistä ohjelmista on Modest-ajon aikana olemassa omana prosessinaan. Käytän näistä prosesseista nimitystä Modest-prosessi, Simnon-prosessi ja valvova prosessi.

Valvova prosessi luo itselleen kaksi lapsiprosessia. Toinen lapsiprosesseista muuttaa itsensä Modest-prosessiksi ja toinen Simnon-prosessiksi. (Tällainen itsensä muuttaminen on tavanomaista UNIXissa.)

Valvovan prosessin tehtävänä on huolehtia siitä, ettei Modest-ajosta jää jäljelle roskaa. Roskalla tarkoitan muun muassa sellaisia tiedostoja, jotka on luotu ajon aikana vain kyseistä ajoa varten. Myös ajoon osallistunut prosessi on ajon päätyttyä roskaa. Modest-ajon päättymisen syynä voi olla Modest-prosessin pääsy lopputilaan mutta yhtä hyvin myös ajonaikainen virhe tai vaikkapa käyttäjän tekemä keskeytys. Valvova prosessi on varautunut mahdollisimman hyvin niihin erilaisiin tapoihin, joilla Modest-ajo voi päättyä.

Modest-prosessin ja Simnon-prosessin välisten yhteyksien muodostaminen tapahtuu ilman erillistä palvelijaprosessia. Valvova prosessi ei osallistu yhteyksien muodostamiseen eikä myöskään kommunikointiin. Yhteyksiä muodostettaessa tavallaan Modest-prosessi ja Simnon-prosessi palvelevat toinen toistaan. Yhteyksien muodostamisen jälkeen asiakas-palvelija-suhde on selvä: Simnon-prosessi on Modest-prosessin palvelija.

6.2 Modest-ajon kulku Modest-prosessin osalta

Modest-ajon kulku valvovan prosessin osalta tuli kuta kuinkin selvitettyä jo osiossa 6.1. Simnon-prosessi on puolestaan Modest-prosessin palvelija heti yhteyksien muodostamisen jälkeen. Näin ollen keskityn kuvaamaan Modest-ajon kulkua

Modest-prosessin osalta. Laskentaa en varsinaisesti kuvaa lainkaan.

Modest-prosessin ensimmäinen tehtävä on lukea koesarjat, ongelman määrittelyt ja käyttäjän tekemät asetukset syötetiedostoista. Käyttäjä kertoo määrittelytiedostossa (nimilistat, ks. osio 3.2) muun muassa Modest-ohjelman tilamuuttujien ja estimoitavien parametrien Simnon-kieliset nimet. Tämä käytäntö on kaikessa redundanttisuudessaankin parempi kuin se, että vastaavuus määrättäisiin jonkin Simnonin noudattaman luettelointijärjestyksen perusteella. Käyttäjä luettelee määrittelytiedostossa myös niiden tiedostojen nimet, joissa ongelmaa kuvaavat Simnon-kieliset systeemit sijaitsevat. (Yksinkertaisessa tapauksessa systeemejä on yksi, mutta systeemejä voi siis aivan hyvin olla useampiakin.)

Seuraavaksi Modest-prosessi rakentaa pelkästään kyseistä Modest-ajoa varten käytettävän Simnon-kielisen yhdistävän systeemin. (Yleisessä tapauksessa kyseessä on käyttäjän antaman yhdistävän systeemin laajennettu versio.) Periaatteessa ko. systeemin kirjoittaminen voitaisiin tehdä vaikka ennen Modest-ajoa. Tehtävään kuluva aika on kuitenkin äärimmäisen lyhyt. Jos tehtävää varten olisi erillinen ohjelma, saattaisi käyttäjältä jopa kulua tarpeettomasti aikaa sen miettimiseen, milloin ko. ohjelma pitäisi ajaa.

Sitten Modest-prosessi avaa komentojen välittämistä varten yhteyden Simnon-prosessiin sekä toisen yhteyden päinvastaiseen suuntaan. Se valvovan prosessin lapsiprosesseista, joka muuttaa itsensä Simnon-prosessiksi (ks. osio 6.1), on näiden yhteyksien muodostamisessa tarvittava toinen osapuoli. Ennen Simnon-prosessiksi muuntautumistaan ko. prosessi myös suuntaa standardisyötevirtansa ja standarditulostusvirtansa uudelleen niin, että Modest-prosessi voi yhtä kanavaa pitkin lähettää komentoja Simnon-prosessille ja toisesta kanavasta lukea Simnon-prosessin standarditulostusta.

Kun edellä mainitut yhteydet on saatu avatuiksi, Modest-prosessi antaa Simnon-prosesille SYST-komennon, ja valitsee näin vallitseviksi systeemeiksi käyttäjän valitsemat systeemit. Systeemien joukossa on lisäksi erityinen geneerinen ulkoinen systeemi. (Em. yhdistävän systeemin rakentaminen on tarpeen, jotta ko. geneeristä ulkoista systeemiä voitaisiin käyttää.)

Välittömästi SYST-komennon annettuaan Modest-prosessi avaa kaksi uutta yhteyttä Simnon-prosessiin, yhden molempiin suuntiin. Yhteyksien muodostamisen toisena osapuolena on tällä kertaa todellakin Simnon-prosessi. Em. ulkoiseen systeemiin on ohjelmoitu yhteyksien muodostamiseen Simnon-puolella tarvittavat toiminnot.

Seuraavaksi Modest-prosessi lähettää ulkoiseen systeemiin päin vievää kanavaa pitkin eräiden muuttujien arvoja ulkoiselle systeemille. Kyseisiä arvoja ulkoinen systeemi käyttää omissa alustustoimenpiteissään. Ulkoisen systeemin on ehdottomasti oltava täydessä valmiustilassa ennen kuin Modest-prosessin kannattaa lähettää Simnon-prosessille simulointikomentoja. Tehtyjen alustusten jälkeen ulkoinen systeemi on täydessä valmiustilassa.

Jotta Modest-prosessi varmistuisi käyttäjän antamien Simnon-määrittelyjen virheettömyydestä, se antaa tarkistusmielessä kattavan joukon komentoja Simnon-prosessille ja päättelee Simnon-prosessin standarditulostuksen perusteella, onko kaikki niin kuin pitää. Komennot on myös valittu siten, että Simnon-prosessin standarditulostuksesta saadaan vähällä vaivalla luettua esimerkiksi tilamuuttujien ja estimoitavien parametrien alkuarvot. Simnon-systeemi-tiedostoja ei näin ollen ainakaan alkuarvojen selvittämistä varten tarvitse lukea.

Tästä eteenpäin Modest-prosessi lähettää Simnon-prosessille SIMU-, PAR- ja INIT-komentoja. SIMU-komennon seurauksena ulkoinen systeemi lähettää simulointitulokset Modest-prosessille sitä kanavaa pitkin, joka tätä tarkoitusta varten avattiin SYST-komennon jälkeen. Tarkalleen ottaen ulkoinen systeemi lähettää Modest-prosessille joukon pisteitä eli aikamuuttujan arvoja ja tulosmuuttujien arvot kyseisissä pisteissä. Tulosmuuttujat ovat käyttäjän määrittelytiedostossa valitsemia STATE-muuttujia ja/tai OUTPUT-muuttujia. PAR- ja INIT-komentojen avulla Modest-prosessi muuttaa systeemien parametreja ja tila-alkuarvoja, mikä on tarpeen esimerkiksi estimoitaessa tai siirrytäessä koesarjasta toiseen.

Ajon lopussa Modest-prosessi tuottaa Modestin tavanomaisten tulostustiedostojen (ks. osio 3.2) lisäksi Simnon-makron, jonka avulla käyttäjä voi myöhemmin Simnonia ajaessaan verrata estimoinnin tms. tulosten mukaisia käyriä koetulosten graafisiin esityksiin.

Kun Modest-prosessi lopettaa toimintansa, valvova prosessi huolehtii kaikesta siivoamisesta. Yhteyksiä ei koskaan varsinaisesti pureta, vaan ne purkautuvat tällä tavoin ikään kuin luonnollista tietä.

6.3 Perustelut ulkoisen systeemin käytölle

Simulointitulosten välittämisen Simnon-prosessilta Modest-prosessille on tapahduttava nopeasti. Ulkoista systeemiä käytetään tästä syystä.

Ulkoiseen systeemiin voidaan ohjelmoida systeemikutsuja tavalliseen tapaan, mikä antaa mahdollisuuden simulointitulosten välittämiseen binäärimuodossa mitä tahansa UNIXin prosessien välisen kommunikoinnin mekanismia käyttäen. Ulkoinen systeemi lukee simulointitulokset muistista, suoraan sieltä minne ne on laskettu. Ulkoinen systeemi voidaan ohjelmoida valikoimaan Modest-prosessille lähetettävät simulointitulokset, ja näin saadaan prosessilta prosessille kulkevan tiedon määrää optimoitua. Ulkoinen systeemi on helposti ohjelmoitavissa niin geneeriseksi, ettei sitä tarvitse muutella ainakaan periaatteessa koskaan. Esimerkiksi taulukkojen teholliset koot voidaan ilmoittaa ulkoiselle systemille ajon aikana. Jos Modest-ajon kesto vanhaa perus-Modestia käytettäessä oli minuutti, oli vastaavan ajon kesto vanhinta ulkoiseen systeemiin nojaavaa Modest-versiota käytettäessä noin kolme minuuttia. Nykyistä ulkoiseen systeemiin nojaavaa Modest-versiota käytettäessä vastaavan ajon kesto on korkeintaan kaksi minuuttia.

Tarkasteltakoon lyhyesti eri tapoja välittää simulointituloksia Simnon-prosessilta Modest-prosessille käyttämättä ulkoista systeemiä. Tekemällä pieniä muutoksia Simnonin lähdetiedostoihin voitaisiin varmastikin saada asiat sujumaan tehokkaasti, mutta osioon 1.4 viitaten sivuutan tämän vaihtoehdon. Simnon-prosessi voitaisiin laittaa kirjoittamaan simulointitulostukset standarditulostusvirtaansa, mutta moinen järjestely vaatisi runsaasti komentoja ja binäärimuodosta ASCII-muotoon ja takaisin suoritettuja muunnoksia. Simnon-prosessi voitaisiin laittaa kirjoittamaan simulointitulostukset store-tiedostoihin. Valitettavasti store-tiedostojen formaatti on salainen, joten store-tiedostot olisi muunnettava ASCII-muotoon. Muunnokset olisivat siis rasitteena. Tiedostojen avaamiseen, asiaan kuuluvat odottelutuokiot mukaan luettuina, kuluva kokonaisaikakin on melkoinen. Kokemukseni store-tiedostojen käytöstä simulointitulosten välittämisessä eivät olleet erityisen rohkaisevia. Jos Modest-ajon kesto vanhaa perus-Modestia käytettäessä oli minuutti, oli vastaavan ajon kesto store-tiedostoihin nojaavaa Modest-versiota käytettäessä aina vähintään kymmenen minuuttia.

Ulkoisen systeemin käytön ainoa merkittävä haitta lienee sitoutuminen UNIX-Simnoniin ja sen plus-versioon. Aikanaan nimittäin harkittiin MS-DOS-Simnoniin tyytyvän Modest-version toteuttamista SCO-UNIX-ympäristöön. Lähteen [9] mukaan nimittäin SCO-UNIX-ympäristössä voidaan ajaa MS-DOS-ohjelmia varsinaisten UNIX-ohjelmien rinnalla.

6.4 Kommunikaatiomekanismi

Osiossa 6.2 kätkin kommunikaatiomekanismit nimityksen kanava taakse. Yhteyksiä Modest-prosessin ja Simnon-prosessin välillä mainitsin olevan neljä siten, että kaikki ovat yksisuuntaisia. Näin asia itse asiassa onkin nykyisessä Modest-versiossa, mutta jos tarkastellaan erilaisia kommunikaatiomekanismivaihtoehtoja, niin Modest-prosessin ja ulkoisen systeemien välillä voisi olla myös pelkästään yksi kaksisuuntainen yhteys.

Nykyisessä Modest-versiossa kommunikointiin käytetään yksinomaan vastakkeita (socket), tarkemmin sanottuna UNIX-kohdealueen virtavastakkeita. Modest-prosessin ja Simnon-prosessin välillä on Modest-ajon aikana neljä eri vastakeyhteyttä, joita jokaista käytetään yksisuuntaisesti ja joista jokaista varten on oma erillinen vastake. Vastakkeiden tällainen käyttö ei ole välttämättä tyylikästä, mutta tarkoituksena on säilyttää mahdollisuus korvata vastakkeet tarvittaessa helposti nimetyillä putkilla. Kunkin vastakkeen nimessä on vakioalkuosa ja lopussa Modest-prosessin ja Simnon-prosessin isäprosessin eli valvovan prosessin numero. Tunnisteen tarkoituksena on suojata kommunikaatiokanava ulkopuolisilta. Modest-ajon kommunikoivat osapuolet osaavat alusta alkaen käyttää kustakin neljästä vastakkeesta oikeaa nimeä, koska isäprosessin numeron saa selville systeemikutsulla.

Käytän siis UNIX-kohdealueen virtavastakkeita kuin nimettyjä putkia, ja voisin helposti korvata vastakkeet nimetyillä putkilla. Olen itse asiassa tehnytkin sellaisen Modest-version, jossa vastakkeet on korvattu nimetyillä putkilla. Kun olen ajanut Modestin putkiversiota samoilla syöteaineistoilla kuin Modestin vastakeversiota, en ole havainnut ekvivalenttien Modest-ajojen kestoissa eroja. Tulos on sopusoinnussa sen osiossa 5.8 esittämäni yleisluontoisen väitteen kanssa, että putket ja UNIX-kohdealueen virtavastakkeet ovat kommunikaationopeuden kannalta yhdenveroisia.

En ole keksinyt tapaa, jolla nykyisen Modest-versioni kaikki vastakkeet korvattaisiin nimeämättömmillä putkilla. Ne vastakkeet, jotka liittyvät Simnon-prosessin standardisyötevirtaan ja standarditulostusvirtaan, olisivat luonnollisestikin helposti korvattavissa nimeämättömillä putkilla. Sen sijaan ulkoiseen systeemiin liittyvien vastakkeiden korvaaminen nimeämättömillä putkilla ei liene aivan yksinkertaista.

Simnon-prosessin standardisyötevirran ja standarditulostusvirran suuntauksessa putket tai vastakkeet ovat käytännössä ainoa vaihtoehto. Jos esimerkiksi standardisyötevirta suunnattaisiin tulemaan tavanomaisesta tiedostosta, Simnon-prosessi ei pystyisi tiedoston lopun kohdattuaan jatkamaan toimintaansa.

Modest-prosessin ja ulkoisen systeemin välisessä kommunikoinnissa olen kokeillut myös viestijonoja. Viestijonojen arvelin olevan kommunikaationopeuden kannalta vastakkeita ja putkia parempia. Kahden kuukauden aikana pidin yllä kahta eri versiota Modestista. Toisessa käytin pelkästään vastakkeita ja toisessa viestijonoja ja vastakkeita. (Puhdas viestijonoversio ei ollut mahdollinen sen perusteella, mitä edellisessä kappaleessa totesin.) Ekvivalenttien Modest-ajojen kestoissa en kuitenkaan koskaan havainnut paria sekuntia suurempia eroja. Tulos ei tarkoita sitä, etteivätkö viestijonot missään sovellutuksessa olisi vastakkeita tehokkaampia. Omassa sovellutuksessani kommunikointimekanismi ei ilmeisesti kuitenkaan ole pullonkaulan asemassa. Muitakin selityksiä on haettavissa. Simulointituloksia välitettiin kenties kerralla sen verran vähän, että käyttöjärjestelmä saattoi pitää vastakkeen kautta välitetyn tiedon yksinomaan puskureissa. Kummassakin versiossa käytettiin vastakkeita, ja ehkäpä teoreettinen puhdas viestijonoversio olisi ollut vastakeversiota tehokkaampi.

Jaetun muistin käyttöä simulointitulosten välittämisessä en ole kokeillut. Jaetun muistin käyttö ei ole sinänsä mitenkään vaikeaa, koska semaforeilla voidaan tarvittava synkronointi aikaansaada melko vaivattomasti. Koska kuitenkin viestijonojen ja vastakkeiden keskinäisen vertailun tulokset olivat sellaisia kuin olivat, en ole mitenkään vakuuttunut siitä, että jaettua muistia käyttämälläkään saataisiin Modest-ajon kestoa olennaisesti lyhenemään.

Mikäli joutuisin hajauttamaan Modest-Simnon-liitännän, valitsisin kommunikointimekanismiksi vastakkeet. Vastakkeet on toteutettu varsin monessa UNIX-järjestelmässä, eikä toista yhtä monessa UNIX-järjestelmässä toteutettua eri prosessorien välillä tapahtuvaan kommunikointiin soveltuvaa systeemikutsutason mekanismia taida olla ainakaan vielä olemassa. Nykyiseen Modest-versioon olisin aiemmin esittämieni tarkastelujen nojalla voinut valita vastakkeiden sijasta nimetyt putket. Hajauttamiskaavailut kallistivat vaa'an vastakkeiden hyväksi. Kokemukset UNIX-kohdealueen virtavastakkeista ovat varmasti hyödyksi, ryhdyttiinpä käyttämään mitä tahansa muita vastakkeita.

6.5 Interpolointi

Selvitettäköön ensin se ongelma, joka ratkaistiin ottamalla käyttöön interpolointi. Ongelmaksi muodostui se, miten Modest saisi juuri ne tulokset, joita se tarvitsee. Ideaalitapauksessa Simnon laskisi tulokset aina vähintään niissä pisteissä,

joista Modest on kiinnostunut, eikä käyttäisi silti laskentaan paljoa aikaa. Käytännössä kuitenkin Simnon valitsee SIMU-komennossa annettuja välin päätepisteitä lukuunottamatta kaikki välipisteet itse.

Ensi alkuun ratkaisin ongelman käyttämällä SIMU -CONT -komentoa erikseen jokaisen koepistevälin yli. Ulkoinen systeemi lähetti kustakin simuloinnista vain välin päätepisteen tulokset Modest-prosessille. Asettamalla simuloinnin maksimiaskelpituus kulloinkin sopivasti saatiin Modest-ajon kesto pysymään kohtuullisena. Jos Modest-ajon kesto vanhaa perus-Modestia käytettäessä oli minuutti, oli vastaavan ajon kesto tätä ratkaisua käytettäessä noin kolme minuuttia.

Koska tiedossa oli kuitenkin se, kuinka nopeasti laskenta tapahtui vanhassa perus-Modestissa, päätettiin Simnonin käyttöä vielä tehostaa. Otettiin käyttöön interpolointi. Interpolointia käyttäen päästään yleisesti ottaen hyviin tuloksiin esim. estimoinnissa. Interpoloinnin aiheuttaman virheen vaikutus on aika olematon, kun tavoitekäyrät ovat sileitä ja interpolointialgoritmi interpoloi lokaalisti. Interpolointiin oli tarjolla sopivia julkisia aliohjelmia. Käyttöön otettiin lähteessä [4] esitetyt splini-interpolointi-aliohjelmat.

Interpolointia käytettäessä simuloidaan koesarjan pisteistön yli yhdellä SIMU-komennolla. Modest-prosessi suorittaa interpoloinnin ulkoiselta systeemiltä saamiensa simulointitulosten perusteella. Ulkoinen systeemi valitsee solmupisteet mahdollisimman tasavälisesti. Suhteellisen pieni solmupistemäärä, esim. 20, on sopiva. Käyttäjä ilmoittaa määrittelytiedostossa, kuinka monta solmupistettä kutakin koesarjaa kohti käytetään. Solmupisteiden määrää ei kannata valita liian suureksi, sillä interpoloinnissa tarvittavien laskutoimitusten määrä on vähintään neliöllinen solmupisteiden määrään nähden. Solmupisteiden määrää lisättäessä tietyn rajan, tyypillisesti noin 20 pistettä, jälkeen interpoloinnin tarkkuus ei enää mainittavasti parane. Syynä ilmiöön on interpoloitavien funktioiden sileys.

Jos interpoloitava funktio on niin hankalasti käyttäytyvä, että interpolointi antaa huonoja tuloksia, on paras ratkaisu korvata hankalasti käyttäytyvä funktio paremmin käyttäytyvällä apufunktiolla, sillä sellainen funktio, jota on vaikea interpoloida, aiheuttaa melko varmasti myös suurta epästabiiliutta estimoinnissa ja kokeiden suunnittelussa.

Jos Modest-ajon kesto vanhaa perus-Modestia käytettäessä on minuutti, on vastaavan ajon kesto interpolointia käytettäessä kaksi minuuttia.

6.6 Mallien kirjo

Uudessa Modestissa voidaan määritellä mitä tahansa Simnon-kielisillä systeemeillä kuvattavissa olevia malleja. Differentiaaliyhtälösysteemimalli, algebrallinen malli tai differentiaalialgebrallinen malli (yhdistelmä differentiaaliyhtälösysteemimallista ja algebrallista mallista) on helposti kuvattavissa Simnon-kielisillä jatkuvilla systeemeillä. Usean systeemin, joukossa mahdollisesti sekä jatkuvia että diskreettejä systeemejä, muodostama malli on jotakin sellaista, mitä vanhassa perus-Modestissa ei pystytä lainkaan kuvaamaan. Vastapainoksi todettakoon, että vanhassa perus-Modestissa kuvattavissa olevaa implisiittistä mallia ei voida ainakaan kovin yksinkertaisesti kuvata Simnon-kielisillä systeemeillä.

6.7 Siirrettävyys

Uusi Modest, jossa on siis Modest-Simnon-liitäntä, on tarkoitus asentaa Kemira Oy:n Espoon tutkimuskeskuksen HP-UNIX-koneeseen, jota ollaan parhaillaan hankkimassa. Uuden Modestin pitäisi olla varsin helposti siirrettävissä melkein mihin tahansa UNIX-ympäristöön. UNIX-Simnonin sijainnin vuoksi en ole vain päässyt kokeilemaan Modest-Simnon-liitäntää muissa kuin HP-UNIX-koneissa.

Vanha perus-Modest käyttää IMSL-aliohjelmakirjastoa. Uudessa Modestissa IMSL on siirrettävyyssyistä korvattu julkisilla aliohjelmakirjastoilla kuten Linpack, Minpack ja Eispack.

Tarkasteltakoon esimerkkinä sitä, miten uusi Modest siirettäisiin lähteen [9] mukaiseen SCO UNIX System V -ympäristöön. Heti alkuun on todettava, että MS-DOS-Simnon, joka SCO-UNIX- ympäristössä muuten olisi käytettävissä, on poissa laskuista, koska uusi Modest tarvitsee UNIX-Simnonin plus-version. UNIX-Simnonin plus-versio pitäisi siis siirtää lähdetiedostomuodossa SCO-UNIXiin ja lähdetiedostot kääntää SCO-UNIXissa. Simnonin asentamisen jälkeen siirrettäisiin Modestin lähdetiedostot SCO-UNIXiin ja suoritettaisiin käännös. SCO UNIX System V:ssä ei ainakaan lähteen [9] mukaan ole toteutettu vastakkeita. Sen sijaan puhtaan System V:n mukaiset prosessien välisen kommunikoinnin mekanismit, kuten nimetyt ja nimeämättömät putket, viestijonot, jaettu muisti ja semaforit, on toteutettu. (Eri prosessorien välillä tapahtuvaan kommunikointiin SCO-UNIXissa on ikioma ohjelmistonsa, johon en ole tutustunut.) Modest-Simnon-liitännän vastakkeiden korvaaminen nimetyillä putkilla ei vaatisi kuin kosmeettisia muutoksia lähdetiedostoissa.

7 Modest-Simnon-liitännän kommunikointitapahtumien analysointia P/T-verkkojen avulla

Liitännän toimivuuden osoittamiseksi analysoin kahta keskeistä kommunikointivaihetta, vastakeyhteyden muodostaminen ja simulointitulosten välittäminen, P/T-verkkojen avulla. Kumpaakin em. vaihetta mallitan omalla P/T-verkollaan. Jos liitäntä ajatellaan järjestelmäksi, eri kommunikointivaiheet edustavat ko. järjestelmän osia.

7.1 P/T-verkot

Petri-verkko on rakenne, jolla voidaan mallittaa rinnakkaisia ja hajautettuja järjestelmiä. P/T-verkko eli paikka-transitio-verkko on eräs Petri-verkko-laji. Muista Petri-verkko-lajeista mainittakoon C/E-verkko eli ehto-tapahtuma-verkko sekä Pr/T-verkko eli predikaatti-transitio-verkko.

7.1.1 P/T-verkon määritelmä

Lähteessä [8] annetun määritelmän mukaan kuusikko

$$N = (S_N, T_N; F_N, K_N, M_N, W_N) \tag{1}$$

on P/T-verkko, jos ja vain jos

(i) $(S_N, T_N; F_N)$ on äärellinen suunnattu graafi, jonka solmujen joukko on $S_N \cup T_N$ ja kaarien joukko F_N siten, että S_N:n alkiosta ei ole koskaan kaarta S_N:n alkioon eikä T_N:n alkiosta koskaan kaarta T_N:n alkioon. Joukon S_N alkioita kutsutaan paikoiksi ja joukon T_N alkioita transitioiksi.

(ii) K_N on funktio, joka kuvaa S_N:n alkiot äärettömällä täydennetylle ei-negatiivisten kokonaislukujen joukolle. Jos s on paikka, niin $K_N(s)$:ää sanotaan paikan s kapasiteetiksi.

(iii) M_N on funktio, joka kuvaa S_N:n alkiot äärettömällä täydennetylle ei-negatiivisten kokonaislukujen joukolle siten, että

$$\forall\ s \in S_N\ M_N(s) \leq K_N(s). \tag{2}$$

M_N:ää kutsutaan verkon alkumerkinnäksi.

(iv) W_N on funktio, joka kuvaa F_N:n alkiot positiivisille kokonaisluvuille. Jos f on kaari, niin $W_N(f)$:ää kutsutaan kaaren f painoksi.

7.1.2 P/T-verkon merkinnät. Saavutettavuus

Lähteessä [8] annetun määritelmän mukaan funktio M joukolta S_N äärettömällä täydennetylle ei-negatiivisten kokonaislukujen joukolle on P/T-verkon N merkintä, jos ja vain jos

$$\forall\ s \in S_N\ M(s) \leq K_N(s). \tag{3}$$

P/T-verkon merkintöjen yhteydessä on tapana puhua merkeistä. Jos M on merkintä ja s paikka, niin sanotaan että verkon merkinnän M vallitessa on paikassa s $M(s)$ merkkiä.

Jos t on transitio, niin lähteessä [8] annetun määritelmän mukaan joukko $\cdot t$ on niiden paikkojen joukko, joista on kaaria t:hen. Vastaavasti $t\cdot$ on niiden paikkojen joukko, joihin t:stä on kaaria.

Olkoon M P/T-verkon N merkintä. Verkon N transitio t on lähteessä [8] annetun määritelmän mukaan M-vireessä, jos ja vain jos

$$\forall\ s \in \cdot t\ M(s) \geq W_N(s,t) \text{ ja } \forall\ s' \in t\cdot\ \ M(s') \leq K_N(s') - W_N(t,s'). \tag{4}$$

P/T-verkon N merkinnässä M on lähteessä [8] annetun määritelmän mukaan verkon transitioon t liittyvä kontakti, jos ja vain jos

$$\forall\ s \in \cdot t\ M(s) \geq W_N(s,t) \text{ ja } \exists\ s' \in t\cdot\ \ M(s') > K_N(s') - W_N(t,s'). \tag{5}$$

Jos M on P/T-verkon N merkintä ja verkon N transitio t on M-vireessä, niin M:n seuraajamerkintä transitioon t liittyen on M', joka määritellään lähteen [8] mukaisesti:

$$M'(s) = \begin{cases} M(s) + W_N(t,s) - W_N(s,t) & , \text{ jos ja vain jos } s \in \cdot t \cap t\cdot \\ M(s) + W_N(t,s) & , \text{ jos ja vain jos } s \in t\cdot - \cdot t \\ M(s) - W_N(s,t) & , \text{ jos ja vain jos } s \in \cdot t - t\cdot \\ M(s) & , \text{ muulloin.} \end{cases} \tag{6}$$

Tällöin on tapana sanoa, että merkinnästä M päästään merkintään M' laukaisemalla transitio t. Lyhennysmerkintä asialle on

$$M[t > M'. \tag{7}$$

Merkintä M' on saavutettavissa merkinnästä M, jos ja vain jos

$$\begin{array}{l} M' = M \text{ tai} \\ \exists\ t_1, \ldots, t_n\ \in T_N\ \exists\ M_1, \ldots, M_{n+1} \\ (M_1 = M \text{ ja } M_{n+1} = M' \text{ ja} \\ \forall\ k \in \{1, \ldots, n\}\ M_k[t_k > M_{k+1}). \end{array} \tag{8}$$

P/T-verkon N saavutettavuusgraafi on suunnattu graafi, jonka solmuina ovat kaikki N:n alkumerkinnästä M_N saavutettavissa olevat merkinnät. Solmusta M on solmuun M' kaari, jos ja vain jos on olemassa t siten, että $M[t > M'$. Jos $M[t > M'$ useammalle kuin yhdelle t, niin jokaista tällaista t kohti on oma kaari.

P/T-verkko on rajoitettu, jos ja vain jos minkään paikan merkkimäärä ei voi kasvaa rajatta. Selvästikin saavutettavuusgraafi on äärellinen, jos ja vain jos verkko on rajoitettu.

Suunnatun graafin G aligraafi g on vahvasti kytketty, jos ja vain jos g:n jokaisesta solmusta on g:ssä polku g:n jokaiseen muuhun solmuun eikä ole olemassa g:stä eroavaa G:n aligraafia g' siten, että g olisi g':n aligraafi ja g':n jokaisesta solmusta olisi g':ssa polku g':n jokaiseen muuhun solmuun. Graafin G vahvasti kytkettyä aligraafia kutsutaan G:n vahvasti kytketyksi komponentiksi. Selvästikin graafin G solmu kuuluu aina yhteen ja vain yhteen G:n vahvasti kytkettyyn komponenttiin.

P/T-verkko on syklinen, jos ja vain jos sen saavutettavuusgraafi on vahvasti kytketty ja graafin jokaisella solmulla on seuraaja. (Jälkimmäinen ehto on ei-redundantti, jos ja vain jos saavutettavuusgraafissa on yksi ja vain yksi solmu.)

P/T-verkko on lähteessä [8] annetun määritelmän mukaan elävä, jos ja vain jos jokaiselle transitiolle t ja jokaiselle alkumerkinnästä saavutettavissa olevalle merkinnälle M on olemassa merkintä M' siten, että M' on saavutettavissa M:stä ja t on M'-vireessä.

Vaikka P/T-verkko olisi syklinen ja rajoitettu, se ei välttämättä ole elävä, sillä on mahdollista, että jokin transitio ei ole minkään saavutettavissa olevan merkinnän kohdalla vireessä, ja silti jokaisesta saavutettavissa olevasta merkinnästä päästään

jokaiseen saavutettavissa olevaan merkintään. Jos jokaiselle transitiolle on olemassa alkumerkinnästä saavutettavissa oleva merkintä, jossa ko. transitio on vireessä, niin syklisyydestä seuraa elävyys, sillä syklisen P/T-verkon kyseessä ollessa elävyyden määritelmässä oleville t ja M löydetään ko. määritelmän mukainen M' valitsemalla M':ksi mikä tahansa sellainen alkumerkinnästä saavutettavissa oleva merkintä, jossa t on vireessä. Lähde [8] tarjoaa esimerkin P/T-verkosta, joka on elävä ja rajoitettu muttei syklinen.

P/T-verkko N on kontaktillinen, jos ja vain jos jossakin N:n alkumerkinnästä saavutettavissa olevassa merkinnässä on johonkin transitioon liittyvä kontakti. Muussa tapauksessa N on kontaktiton. Jos verkko N on kontaktiton ja verkko N' eroaa N:stä korkeintaan siten, että N':ssa joillakin tai vaikkapa kaikilla paikoilla on suuremmat kapasiteetit kuin N:ssä, niin N:n ja N':n saavutettavuusgraafit ovat samat.

7.1.3 P/T-verkkojen lineaarialgebrallinen esittäminen

Olkoon N P/T-verkko, $S_N = \{s_1, \ldots, s_n\}$, ja M jokin N:n merkintä. M voidaan tulkita vektoriksi $(M(s_1) \ldots M(s_n))^T$. Vektorit ovat matriiseja, ja matriisit määritellään matematiikassa usein funktioiksi.

P/T-verkon N insidenssimatriisi C_N on lähteessä [8] annetun määritelmän mukaan funktio S_N:n ja T_N:n karteesiselta tulolta kokonaisluvuille siten, että vaakarivit vastaavat paikkoja ja pystyrivit transitioita ja kaikille paikoille s ja kaikille transitioille t pätee

$$C_N(s,t) = \begin{cases} W_N(t,s) - W_N(s,t) & , \text{ jos ja vain jos } s \in {\cdot t} \cap t{\cdot} \\ W_N(t,s) & , \text{ jos ja vain jos } s \in t \cdot - \cdot t \\ -W_N(s,t) & , \text{ jos ja vain jos } s \in {\cdot t} - t{\cdot} \\ 0 & , \text{ muulloin.} \end{cases} \tag{9}$$

7.1.4 S-invariantit

Olkoon N P/T-verkko ja i funktio, joka kuvaa S_N:n alkiot kokonaisluvuille. Lähteessä [8] annetun määritelmän mukaan i on N:n S-invariantti, jos ja vain jos

$$C_N^T i = O, \tag{10}$$

missä i:lle on annettu samanlainen vektoritulkinta kuin verkon merkinnöille edellä.

Selvästikin S-invarianttien kokonaislukukertoiminen lineaarikombinaatio on aina myös S-invariantti. Lähteessä [8] on osoitettu, että jos i on N:n S-invariantti, niin

$$M^T i = M_N^T i, \tag{11}$$

oli M mikä tahansa alkumerkinnästä M_N saavutettavissa oleva merkintä. Kutsun tätä ominaisuutta sisätuloinvarianssiominaisuudeksi.

S-invarianttia i sanotaan positiiviseksi, jos ja vain jos i:n mikään komponentti ei ole negatiivinen ja vähintään yksi i:n komponentti on positiivinen.

P/T-verkkoa sanotaan lähteessä [8] annetun määritelmän mukaan S-invarianttien peittämäksi, jos ja vain jos jokaista paikkaa s kohti on olemassa positiivinen S-invariantti i siten, että $i(s) > 0$.

Koska S-invarianttien kokonaislukukertoiminen lineaarikombinaatio on S-invariantti, niin helposti todetaan, että verkko on S-invarianttien peittämä, jos ja vain jos on olemassa sellainen S-invariantti, jonka kaikki komponentit ovat positiivisia. S-invarianttien sisätuloinvarianssiominaisuuden perusteella on selvää, että S-invarianttien peittämä P/T-verkko on rajoitettu.

7.1.5 T-invariantit

Olkoon N P/T-verkko ja i funktio, joka kuvaa T_N:n alkiot kokonaisluvuille. Lähteessä [8] annetun määritelmän mukaan i on N:n T-invariantti, jos ja vain jos

$$C_N i = O, \tag{12}$$

missä i:lle on annettu luontainen vektoritulkinta.

Selvästikin T-invarianttien kokonaislukukertoiminen lineaarikombinaatio on aina myös T-invariantti.

T-invarianttia i sanotaan positiiviseksi, jos ja vain jos i:n mikään komponentti ei ole negatiivinen ja vähintään yksi i:n komponentti on positiivinen.

P/T-verkkoa sanotaan lähteessä [8] annetun määritelmän mukaan T-invarianttien peittämäksi, jos ja vain jos jokaista transitiota t kohti on olemassa positiivinen T-invariantti i siten, että $i(t) > 0$.

Koska T-invarianttien kokonaislukukertoiminen lineaarikombinaatio on T-invariantti, niin helposti todetaan, että verkko on T-invarianttien peittämä, jos ja vain jos on olemassa sellainen T-invariantti, jonka kaikki komponentit ovat positiivisia.

N:n positiivinen T-invariantti i on lähteessä [8] annetun määritelmän mukaan toteutuva, jos ja vain jos

$$\begin{array}{l} \exists\ t_1, \ldots, t_n\ \in T_N\ \exists\ M_1, \ldots, M_{n+1} \\ (M_1 \text{ on saavutettavissa } M_N\text{: stä ja} \\ \forall\ k \in \{1, \ldots, n\}\ M_k[t_k > M_{k+1} \text{ ja} \\ \forall\ t \in T_N\ i(t) = |\{j\ :\ 1 \leq j \leq n \text{ ja } t_j = t\}|). \end{array} \tag{13}$$

Lähteessä [8] on osoitettu, että P/T-verkon saavutettavuusgraafin jokaista silmukkaa x vastaa jokin T-invariantti, joka on toteutuva siten, että x on toteutuvuuden määritelmässä vaadittu polku. Lähteessä [8] on osoitettu myös, että jokainen elävä ja rajoitettu P/T-verkko on T-invarianttien peittämä.

7.1.6 P/T-verkkojen graafinen esittäminen

P/T-verkkoja on yleensä helpointa tarkastella johonkin merkintään liittyvinä graafisina esityksinä. Kuvaan seuraavassa sitä esitystapaa, jota omissa esimerkeissäni käytän.

Paikat piirretään ympyröinä, transitiot suorakaiteina ja kaaret nuolina. Ykköstä suurempi tai symbolisella vakiolla ilmaistu kaaripaino kirjoitetaan näkyviin kaaren viereen. Kapasiteetteja ei laiteta mukaan piirrokseen, vaan ne kerrotaan verkkoa käsittelevässä tekstissä. Verkon merkintä ilmaistaan piirtämällä merkit mustina täplinä paikkoja kuvaavien ympyröiden sisään.

7.2 Prena

Prena on TKK:n digitaalitekniikan laboratoriossa kehitetty predikaatti-transitioverkkojen saavutettavuusanalysaattori. Prenalla voi tuottaa määrittelemälleen Pr/T-verkolle saavutettavuusgraafin sekä kysellä monia saavutettavuusgraafiin liittyviä asioita.

Käytännöllisesti katsoen P/T-verkot eivät ole mitään muuta kuin tietyllä tavalla rajoittuneita Pr/T-verkkoja. Näin ollen Prenalla voi aivan hyvin analysoida puhtaita P/T-verkkoja.

Prenaa on kuvattu tarkemmin lähteessä [5].

7.3 Modestin ja Simnonin välisen vastakeyhteyden muodostamisen analysointi

7.3.1 Modestin ja Simnonin välisen vastakeyhteyden muodostamista mallittava P/T-verkko

Tarkastellaan kuvan 1 P/T-verkkoa. Kuvaan on piirretty alkumerkintä. Kaikkien paikkojen kapasiteetit ovat äärettömiä.

Kutakin transitiota vastaa tapahtuma. Transitioita vastaavat tapahtumat ovat:

t_1 : Modest kutsuu funktioita socket(), bind() ja listen(), tässä järjestyksessä.

t_2 : Simnon kutsuu funktiota socket().

t_3 : Modest kutsuu funktiota accept().

t_4 : Simnon kutsuu funktiota connect().

t_5 : Modestin accept-kutsu osoittautuu tuloksettomaksi.

t_6 : Simnonin connect-kutsu osoittautuu tuloksettomaksi.

t_7 : Modest ei luovuta, vaan yrittää uudelleen yhteyden muodostamista.

t_8 : Simnon ei luovuta, vaan yrittää uudelleen yhteyden muodostamista.

t_9 : Yhteys Modestin ja Simnonin välille muodostuu.

t_{10} : Modestin tekemästä socket-kutsusta on kulunut tasan 30 sekuntia.

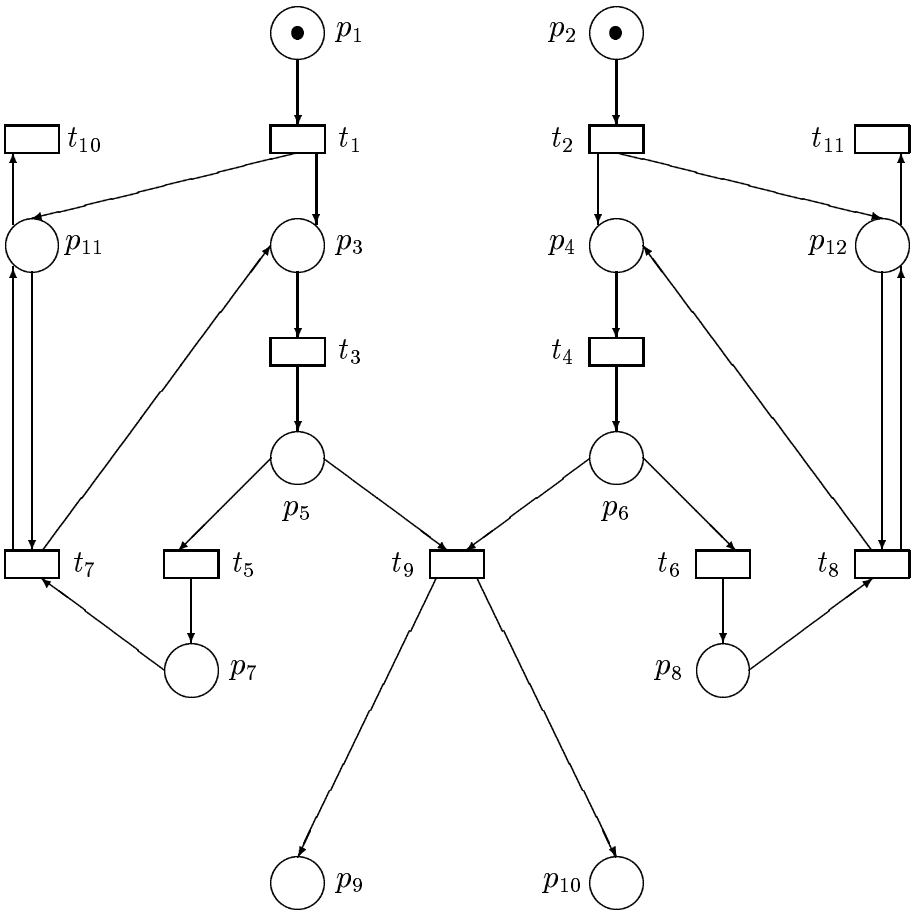
t_{11} : Simnonin tekemästä socket-kutsusta on kulunut tasan 30 sekuntia.

Kutakin paikkaa vastaa ehto, joka on voimassa, jos ja vain jos ko. paikassa on merkki. Paikkoja vastaavat ehdot ovat:

p_1 : Modest on yhteyden muodostamisen kannalta alkutilassa.

p_2 : Simnon on yhteyden muodostamisen kannalta alkutilassa.

p_3 : Modest on valmis kutsumaan funktiota accept().



Kuva 1: Modestin ja Simnonin välisen vastakeyhteyden muodostamista mallittava P/T-verkko

p_4 : Simnon on valmis kutsumaan funktiota connect().

p_5 : Modest on juuri kutsunut funktiota accept() ja odottaa jonkin muun prosessin connect-kutsua.

p_6 : Simnon on juuri kutsunut funktiota connect() ja odottaa jonkin muun prosessin accept-kutsua.

p_7 : Modest on juuri epäonnistunut yhteyden muodostamisessa.

p_8 : Simnon on juuri epäonnistunut yhteyden muodostamisessa.

p_9 : Modest voi lähettää Simnonille tietoa tai vastaanottaa Simnonilta tietoa.

p_{10} : Simnon voi lähettää Modestille tietoa tai vastaanottaa Modestilta tietoa.

p_{11} : Modestin tekemästä socket-kutsusta on kulunut korkeintaan 30 sekuntia. (Tässä kohtaa on syytä muistaa, että transition vireessäoloehdot ovat vain välttämättömiä ehtoja transition laukeamiselle. P/T-verkossa ei voida antaa riittäviä ehtoja transition laukeamiselle, ts. transitiota ei voida pakottaa laukeamaan.)

p_{12} : Simnonin tekemästä socket-kutsusta on kulunut korkeintaan 30 sekuntia.

Olkoon esimerkin P/T-verkko N. Insidenssimatriisi C_N, alkumerkintävektori M_N, S-invarianttien kantavektorit ja T-invarianttien kantavektorit on esitetty kuvassa 2. i_1, i_2 ja i_3 ovat S-invarianttien kantavektorit, j_1 ja j_2 puolestaan T-invarianttien kantavektorit.

7.3.2 S-invariantti-analyysi

Verkon N S-invariantit olen ratkaissut kynää ja paperia käyttäen yhtälöstä $C_N^T y = O$. Ko. yhtälön ratkaisu on $y = \sigma_1 i_1 + \sigma_2 i_2 + \sigma_3 i_3$, missä σ_1, σ_2 ja σ_3 ovat mitä tahansa kokonaislukuja ja i_1, i_2 ja i_3 ovat kuvassa 2 esitetyt kantavektorit.

Verkko N ei ole S-invarianttien peittämä, koska kaikissa S-invarianteissa paikkoihin p_{11} ja p_{12} liityvät komponentit ovat nollia.

C_N	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	M_N	i_1	i_2	i_3
p_1	-1	0	0	0	0	0	0	0	0	0	0	1	1	0	0
p_2	0	-1	0	0	0	0	0	0	0	0	0	1	0	1	0
p_3	1	0	-1	0	0	0	1	0	0	0	0	0	1	0	0
p_4	0	1	0	-1	0	0	0	1	0	0	0	0	0	1	0
p_5	0	0	1	0	-1	0	0	0	-1	0	0	0	1	0	0
p_6	0	0	0	1	0	-1	0	0	-1	0	0	0	0	1	0
p_7	0	0	0	0	1	0	-1	0	0	0	0	0	1	0	0
p_8	0	0	0	0	0	1	0	-1	0	0	0	0	0	1	0
p_9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
p_{10}	0	0	0	0	0	0	0	0	1	0	0	0	1	1	-1
p_{11}	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0
p_{12}	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	0
j_1	0	0	1	0	1	0	1	0	0	0	0				
j_2	0	0	0	1	0	1	0	1	0	0	0				

Kuva 2: Modestin ja Simnonin välisen vastakeyhteyden muodostamista mallittavan P/T-verkon insidenssimatriisi, alkumerkintävektori, S-invarianttien kantavektorit ja T-invarianttien kantavektorit

Olkoon M mikä tahansa N:n alkumerkinnästä M_N saavutettavissa oleva merkintä. Invariantin i_1 sisätuloinvarianssiominaisuuden perusteella

$$\begin{aligned} & M(p_1) + M(p_3) + M(p_5) + M(p_7) + M(p_{10}) \\ = \; & M_N(p_1) + M_N(p_3) + M_N(p_5) + M_N(p_7) + M_N(p_{10}) = 1. \end{aligned} \tag{14}$$

Invariantin i_2 sisätuloinvarianssiominaisuuden perusteella

$$\begin{aligned} & M(p_2) + M(p_4) + M(p_6) + M(p_8) + M(p_{10}) \\ = \; & M_N(p_2) + M_N(p_4) + M_N(p_6) + M_N(p_8) + M_N(p_{10}) = 1. \end{aligned} \tag{15}$$

Invariantin i_3 sisätuloinvarianssiominaisuuden perusteella

$$M(p_9) - M(p_{10}) = M_N(p_9) - M_N(p_{10}) = 0. \tag{16}$$

Näistä tuloksista on suoraan nähtävissä, että missään paikoista $p_1, \ldots, p_{10}$ ei voi olla kerralla enempää kuin yksi merkki. Toisaalta nähdään, että paikoissa $p_1, \ldots, p_{10}$ on yhteensä aina täsmälleen kaksi merkkiä. Jos p_9 tai p_{10} ei ole tyhjä, niin ko. merkit ovat p_9:ssä ja p_{10}:ssä. Jos p_9 ja p_{10} ovat tyhjiä, niin toinen merkeistä on jossakin paikoista p_1, p_3, p_5 ja p_7 ja toinen jossakin paikoista p_2, p_4, p_6 ja p_8.

Saadut tulokset osoittavat, että paikoille $p_1, \ldots, p_{10}$ annetut tulkinnat ovat keskenään konsistentteja. Paikoista p_{11} ja p_{12} S-invariantti-analyysi ei kerro mitään.

7.3.3 T-invariantti-analyysi

Verkon N T-invariantit olen ratkaissut kynää ja paperia käyttäen yhtälöstä $C_N y = O$. Ko. yhtälön ratkaisu on $y = \tau_1 j_1 + \tau_2 j_2$, missä τ_1 ja τ_2 ovat mitä tahansa kokonaislukuja ja j_1 ja j_2 ovat kuvassa 2 esitetyt kantavektorit.

Verkko N ei ole T-invarianttien peittämä, koska esimerkiksi transitiota t_1 vastaava komponentti on nolla jokaisessa T-invariantissa. N ei ole elävä, sillä jos N olisi elävä, niin N olisi T-invarianttien peittämä, koska N on rajoitettu. (Sen, että N on rajoitettu, osoittaa saavutettavuusanalyysi, ks. osio 7.3.4.)

Selvästikin N:n T-invariantti $y = \tau_1 j_1 + \tau_2 j_2$ on positiivinen, jos ja vain jos τ_1 ja τ_2 ovat ei-negatiivisia ja ainakin toinen τ_1:stä ja τ_2:sta on positiivinen.

Jokainen N:n positiivinen T-invariantti on toteutuva, koska alkumerkinnästä päästään transitiosekvenssillä $t_1 t_2$ merkintään, jossa p_3:ssa, p_4:ssä, p_{11}:ssä ja p_{12}:ssa on

merkki ja joka on saavutettavissa itsestään muun muassa sellaisella transitiosekvenssillä, jossa esiintyy τ_1 kertaa alisekvenssi $t_3t_5t_7$ ja τ_2 kertaa alisekvenssi $t_4t_6t_8$, kun τ_1 ja τ_2 ovat ei-negatiisia ja ainakin toinen τ_1:stä ja τ_2:sta on positiivinen.

7.3.4 Saavutettavuusanalyysi

Invarianttianalyysi on eräänlaista saavutettavuusanalyysiä, jossa savvutettavuusgraafia ei muodosteta. Varsinaisessa saavutettavuusanalyysissä muodostetaan saavutettavuusgraafi. Liitteessä 1 on Prenalla tuotettu saavutettavuusgraafi verkolle N.

Saavutettavuusgraafin tilastotiedot kertovat, että graafissa ei ole kontakteja (overflow). Tämä tulos kertoo, että Prenalle annetussa verkkokuvauksessa tehty kapasiteettien valinta on täysin sopiva, vaikka N:n määritelmän mukaan kaikki kapasiteetit ovat äärettömiä. Tulos vahvistaa myös sen, että N on rajoitettu.

Kuolleita paikkoja ei ole, toisin sanoen ei ole sellaisia paikkoja, joiden merkkimäärä olisi nolla kaikissa saavutettavuusgraafin solmuissa.

Kuolleita transitioita ei ole, toisin sanoen ei ole sellaista transitiota, joka ei olisi vireessä missään saavutettavuusgraafin solmussa.

Graafissa on kaksi lukkiutumaa. Toinen lukkiutumista on merkintä, jossa paikoissa p_9 ja p_{10} on kummassakin yksi merkki eikä muissa paikoissa ole merkkejä. Tämä lukkiutuma on toivottu lopputila, sillä kyseisessä tilassa yhteys on valmis. Toinen lukkiutumista on puolestaan merkintä, jossa paikoissa p_7 ja p_8 on kummassakin yksi merkki eikä muissa paikoissa ole merkkejä. Tämä lukkiutuma on se lopputila, johon pitääkin päätyä silloin, jos yhteyttä ei saada aikaiseksi 30 sekunnin kuluessa siitä, kun ensimmäinen prosesseista tekee socket-kutsun.

Graafissa on peräti 29 vahvasti kytkettyä komponentteja. Mukana on paitsi yksisolmuisia myös monisolmuisia vahvasti kytkettyjä komponentteja. Monisolmuiset vahvasti kytketyt komponentit ovat ongelmallisia, sillä ne antavat periaatteessa mahdollisuuden siihen, että kumpaankaan edellä mainituista lopputiloista ei päästä koskaan.

Elolukkoja ei ole. (Elolukko on vapaa suomennos termille livelock.) Prenassa elolukolla tarkoitetaan sellaista saavutettavuusgraafin vahvasti kytkettyä komponenttia, jonka solmuista ei ole mahdollista päästä muihin kuin komponentin omiin solmuihin ja joka ei kuitenkaan ole lukkiutuma eikä koko saavutettavuusgraafi. Tässä luvussa

käytän sanaa elolukko tässä merkityksessä.

Olettakaamme, että mikään transitio ei voi olla pysyvästi vireessä koskaan laukeamatta. Tällöin kumpikin transitioista t_1 ja t_2 laukeavat joskus. Transition t_1 laukeaminen virittää transition t_{10}. Transition t_2 laukeaminen virittää transition t_{11}. Kun transitio t_{10} on kerran virittynyt, virityneisyyttä ei poista mikään muu tapahtuma kuin t_{10}:n laukeaminen. Vastaava tulos pätee t_{11}:lle. Transitiot t_{10} ja t_{11} laukeavat siis joskus. Saavutettavuusgraafista nähdään, että paikassa p_{11} ei ole koskaan enempää kuin yksi merkki. Sama tulos pätee paikalle p_{12}. Välttämättä päädytään siis tilaan, jossa yhdessäkään paikoista p_1, p_2, p_{11} ja p_{12} ei ole merkkiä. S-invariantti-analyysin tulosten perusteella kyseisessä tilassa verkossa on täsmälleen kaksi merkkiä, joista toinen on jossakin paikoista p_3, p_5, p_7 ja p_9 ja toinen jossakin paikoista p_4, p_6, p_8 ja p_{10}. Paikkoihin p_{11} ja p_{12} ei mikään transitoiden laukeamissekvenssi tuo enää merkkejä, joten transitoita t_7 ja t_8 ei saada enää vireeseen. Saadut tulokset yhdessä sen olettamuksen kanssa, että mikään transitio ei voi olla pysyvästi vireessä koskaan laukematta, merkitsevät sitä, että välttämättä päädytään jompaankumpaan järjestelmän kahdesta lukkiutumasta, jotka siis aiemmin todettiin korrekteiksi lopputiloiksi.

7.4 Simulointitulosten välittämisen analysointi

7.4.1 Simulointitulosten välittämistä mallittava P/T-verkko

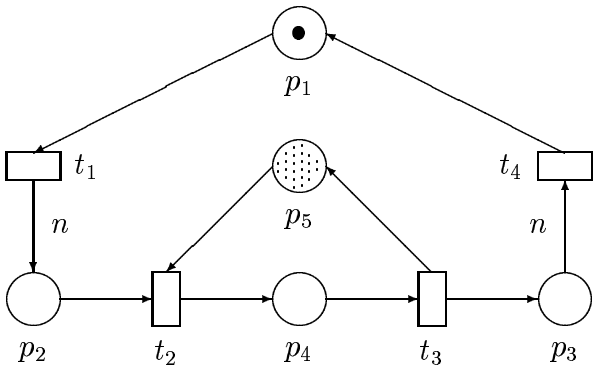
Tarkastellaan kuvan 3 P/T-verkkoa. Kuvaan on piirretty alkumerkintä. Kaikkien paikkojen kapasiteetit ovat äärettömiä.

n on yhden simuloinnin aikana Simnonilta Modestille kulkevan tiedon määrä tavuina. k on se määrä tavuja, mikä vastakkeeseen kerrallaan enimmillään mahtuu. UNIX-kohdealueen vastakkeen tapauksessa yleensä $k > n$, mutta Internet-kohdealueen vastakkeen tapauksessa voi olla myös mahdollista $k < n$. (Olen toteuttanut Modest-Simnon- liitännän niin, että tapaus $k < n$ ei aiheuta vaikeuksia.)

Kutakin transitiota vastaa tapahtuma. Transitioita vastaavat tapahtumat:

t_1 : Modest lähettää Simnonille SIMU-komennon.

t_2 : Simnon kirjoittaa yhden tavun vastakkeeseen. Jokaista Simnonin suorittamaa atomista kirjoitusoperaatiota esittää sarja peräkkäisiä transition t_2 laukeamisia.



Kuva 3: Simulointitulosten välittämistä mallittava P/T-verkko. Paikassa p_5 on k merkkiä.

t_3 : Modest lukee yhden tavun vastakkeesta. Jokaista Modestin suorittamaa atomista lukuoperaatiota esittää sarja peräkkäisiä transition t_3 laukeamisia.

t_4 : Modest jatkaa laskentaansa saatuaan kaikki simulointitulokset.

Transitiot t_2 ja t_3 mallittavat tavuvirtaa. Write-kutsulla kirjoitettujen ja read-kutsulla luettujen palasten ei tarvitse olla yksitellen samoja, vaan riittää, että vastaanotettujen tavujen määrä on sama kuin lähetettyjen tavujen määrä. Tiedon järjestyksen säilyminen on selvää, koska käytössä on virtavastake. Kuvan 3 verkossa tiedon järjestyksen säilymistä ei edes malliteta.

Paikkojen tulkinnat:

p_1 : Modest on valmis antamaan SIMU-komennon, jos ja vain jos p_1:ssä on merkki.

p_2 : Simnonilla on yhtä monta tavua kirjoittamatta kuin p_2:ssa on merkkejä.

p_3 : Modest on lukenut yhtä monta tavua kuin p_3:ssa on merkkejä.

p_4 : Vastakkeessa on yhtä monta tavua tietoa kuin p_4:ssä on merkkejä.

C_N	t_1	t_2	t_3	t_4	M_N	i_1	i_2
p_1	-1	0	0	1	1	n	0
p_2	n	-1	0	0	0	1	0
p_3	0	0	1	$-n$	0	1	0
p_4	0	1	-1	0	0	1	1
p_5	0	-1	1	0	k	0	1
j	1	n	n	1			

Kuva 4: Simulointitulosten välittämistä mallittavan P/T-verkon insidenssimatriisi, alkumerkintävektori, S-invarianttien kantavektorit ja T-invarianttien kantavektorit

p_5 : Vastakkeessa on vapaata tilaa yhtä monta tavua kuin p_5:ssä on merkkejä.

Olkoon esimerkin P/T-verkko N. Insidenssimatriisi C_N, alkumerkintävektori M_N, S-invarianttien kantavektorit ja T-invarianttien kantavektorit on esitetty kuvassa 4. i_1 ja i_2 ovat S-invarianttien kantavektorit, j puolestaan T-invarianttien kantavektori.

7.4.2 S-invariantti-analyysi

Verkon N S-invariantit olen ratkaissut kynää ja paperia käyttäen yhtälöstä $C_N^T y = O$. Ko. yhtälön ratkaisu on $y = \sigma_1 i_1 + \sigma_2 i_2$, missä σ_1 ja σ_2 ovat mitä tahansa kokonaislukuja ja i_1 ja i_2 ovat kuvassa 4 esitetyt kantavektorit.

Verkko N on S-invarianttien peittämä, sillä S-invariantin $i_1 + i_2$ kaikki komponentit ovat positiivisia. N on siis rajoitettu.

Olkoon M mikä tahansa N:n alkumerkinnästä M_N saavutettavissa oleva merkintä. Invariantin i_1 sisätuloinvarianssiominaisuuden perusteella

$$\begin{aligned} & nM(p_1) + M(p_2) + M(p_3) + M(p_4) \\ = \; & nM_N(p_1) + M_N(p_2) + M_N(p_3) + M_N(p_4) = n. \end{aligned} \tag{17}$$

p_1:lle, p_2:lle, p_3:lle, p_4:lle ja n:lle annetut tulkinnat ovat siis keskenään konsistentteja. Lisäksi todetaan, että $M(p_1) \leq 1$, $M(p_2) \leq n$, $M(p_3) \leq n$ ja $M(p_4) \leq n$.

Invariantin i_2 sisätuloinvarianssiominaisuuden perusteella

$$M(p_4) + M(p_5) = M_N(p_4) + M_N(p_5) = k. \tag{18}$$

p_4:lle, p_5:lle ja k:lle annetut tulkinnat ovat siis keskenään konsistentteja. Lisäksi todetaan, että $M(p_4) \leq k$ ja $M(p_5) \leq k$.

7.4.3 T-invariantti-analyysi

Verkon N T-invariantit olen ratkaissut kynää ja paperia käyttäen yhtälöstä $C_N y = O$. Ko. yhtälön ratkaisu on $y = \tau j$, missä τ on mikä tahansa kokonaisluku ja j on kuvassa 4 esitetty kantavektori.

Verkko N on T-invarianttien peittämä, sillä T-invariantin j kaikki komponentit ovat positiivisia. N:n pitääkin olla T-invarianttien peittämä, koska N selvästikin on elävä ja N on edellä todettu rajoitetuksi.

j on toteutuva, sillä mikä tahansa alkumerkinnästä saavutettavissa oleva merkintä on saavutettavissa itsestään sellaisella transitiosekvenssillä, jossa t_1 esiintyy kerran, t_2 n kertaa, t_3 n kertaa ja t_4 kerran. Koska kaikki N:n T-invariantit ovat j:n kerrannaisia, niin todetaan, että kaikki N:n positiiviset T-invariantit ovat toteutuvia.

Koska verkon N saavutettavuusgraafin jokaista silmukkaa välttämättä vastaa jokin toteutuva T-invariantti, niin todetaan, että mikään tila ei ole saavutettavissa itsestään muuten kuin sellaisella transitiosekvenssillä, jossa t_1 esiintyy kerran, t_2 n kertaa, t_3 n kertaa ja t_4 kerran. Minkäänlaisia järjestelmän toiminnan tuloksellisuuden kannalta hyödyttömiä silmukoita saavutettavuusgraafissa ei siis ole.

7.4.4 Saavutettavuusanalyysi

Liitteessä 2 on Prenalla tuotettu saavutettavuusgraafi verkolle N,kun n ja k on kiinnitetty. Luvut ovat tietenkin aivan väärää suuruusluokkaa mutta analyysin kannalta täysin edustavia. Valitsin tarkoituksellisesti n:n ja k:n siten, että $k > 1$ ja $n > 2k$.

Saavutettavuusgraafin tilastotiedot kertovat, että graafissa ei ole kontakteja. Prenalle annetussa verkkokuvauksessa tehty kapasiteettien valinta on siten täysin sopiva, vaikka N:n määritelmän mukaan kaikki kapasiteetit ovat äärettömiä.

Lukkiutumia, elolukkoja, kuolleita paikkoja tai kuolleita transitioita ei ole.

Vahvasti kytkettyjä komponentteja on vain yksi. Saavutettavuusgraafi on siis vahvasti kytketty, mistä seuraa verkon N syklisyys, koska saavutettavuusgraafissa solmuja on enemmän kuin yksi. N:n elävyys seuraa siitä, että N on syklinen ja N:ssä ei ole kuolleita transitioita.

Syklisyys, elävyys ja se T-invariantti-analyysin osoittama tulos, että kaikki esiintyvät silmukat ovat järjestelmän toiminnan tuloksellisuuden kannalta toisiinsa verrat-

tuna yhtä hyödyllisiä, merkitsevät yhdessä sitä, että kyseessä oleva järjestelmän osa toimii aina ehdottoman tarkoituksenmukaisesti.

8 Yhteenveto

Tämän työn tavoitteena oli luoda estimointiohjelma Modestin ja simulointiohjelma Simnonin välille sellainen yhteys, että Modestin käsittelemät ongelmat voitaisiin kuvata Simnonin korkeatasoisella kuvauskielellä ja että Modest voisi laskennassaan käyttää Simnonia mahdollisimman täysimittaisesti hyväkseen. Yhden estimointitehtävän suorittaminen sisältää yleensä runsaasti simulointia, joten simulointiohjelman käyttäminen estimointiohjelman palvelijana on luontevaa. Simnon tarjoaa mahdollisuuden määritellä malleja modulaarisesti. Vastaavaa mahdollisuutta ei ole vanhassa FORTRAN-malleja käyttävässä Modestissa. Simnon-kielisistä jatkuvista ja diskreeteistä systeemeistä voidaan koota monia sellaisia malleja, jotka eivät ole kuvattavissa vanhassa perus-Modestissa.

Työn tuloksena syntyi UNIX-ympäristössä toimiva Modest-versio, jonka oleellisena osana on Modest-Simnon-liitäntä.

Modest-Simnon-liitännän toteutin UNIXissa prosessien välisen kommunikoinnin keinoin. Simnonin lähdetiedostot olivat kaupallisesti suojattuja, eikä Simnoniin liittyvien lupien ostamista katsottu aiheelliseksi ainakaan ennen kuin mahdollisimman tehokas prosessien väliseen kommunikointiin perustuva liitäntä olisi valmis.

Lukuisista tarjolla olleista UNIXIN prosessien välisen kommunikoinnin mekanismeista valitsin UNIX-kohdealueen virtavastakkeet. (Vastake on vapaa suomennos termille socket.) Ratkaisevia tekijöitä valinnassa oli kolme:

1) Komentojen välittäminen Simnonin standardisyötevirtaan vaati käytännössä joko putkien tai vastakkeiden käyttöä.

2) Suorittamani vertailu osoitti Modest-ajon keston olevan, syöteaineistosta riippumatta, UNIX-kohdealueen virtavastakkeita käytettäessä saman kuin nimettyjä putkia käytettäessä tai viestijonoja yhdessä UNIX-kohdealueen virtavastakkeiden kanssa käytettäessä.

3) Jos Modest-Simnon-liitäntä joskus hajautettaisiin, silloin mitä todennäköisimmin tultaisiin käyttämään vastakkeita.

Työn suurimmat ongelmat koskivat kokonaislaskennan nopeutta. Simnoniin liitetyn ulkoisen systeemin avulla sain ratkaistua nopeusongelman karkealla tasolla. Ulkoinen systeemi kirjoittaa Simnonin laskemat simulointitulokset vastakkeeseen suo-

raan binäärimuodossa. Interpoloinnin käyttöönotto paransi vielä jonkin verran kokonaislaskennan nopeutta. Simnon laskee simulointitulokset valitsemissaan pisteissä. Ulkoinen systeemi välittää Modestille tietyn määrän simulointituloksia, joiden perusteella Modest laskee tulokset niissä pisteissä, joiden tuloksia Modest varsinaisesti tarvitsee. Simnonia käyttävä Modest suoriutuu tehtävästään keskimäärin kaksi kertaa niin pitkässä ajassa kuin samaa tehtävää suorittava Simnonia käyttämätön Modest. Käytännössä tämä merkitsee tyypillisesti esimerkiksi kahta minuuttia yhden minuutin sijasta.

Liitännän toimivuuden osoittamiseksi analysoin kahta keskeistä kommunikointivaihetta, vastakeyhteyden muodostaminen ja simulointitulosten välittäminen, P/T-verkkojen avulla. Kumpaakin em. vaihetta mallitin omalla P/T-verkollaan. Jos liitäntä ajatellaan järjestelmäksi, eri kommunikointivaiheet edustavat ko. järjestelmän osia. S-invariantti-analyysi osoitti, ettei P/T-verkoilla mallittamissani järjestelmän osissa esiinny kiellettyjä tiloja. T-invariantti-analyysi näytti, millaisilla tapahtumasekvensseillä mallittamieni järjestelmän osien jokin tila on saavutettavissa itsestään. Saavutettavuusanalyysi osoitti, ettei mallittamissani järjestelmän osissa ole lukkiutumia, jos korrekteja lopputiloja ei lasketa mukaan. Vastakeyhteyden muodostamisessa korrekteja lopputiloja oli olemassa ja saavutettavuusanalyysi S-invarianttianalyysin tukemana osoitti, että johonkin korrekteista lopputiloista aina lopulta päästään, jos oletetaan, että mikään transitio ei voi olla pysyvästi vireessä koskaan laukeamatta. Simulointitulosten välittämistä mallittaneen P/T-verkon saavutettavuusanalyysi ja T-invariantti-analyysi puolestaan yhdessä osoittivat, että kaikki saavutettavissa olevat tilat ovat saavutettavissa toisistaan ja että kyseessä oleva järjestelmän osa toimii aina ehdottoman tarkoituksenmukaisesti.

Uusi Modest otettaneen Kemira Oy:n Espoon tutkimuskeskuksessa käyttöön heti, kun tarvittava UNIX-kone on sinne saatu.

Lähdeluettelo

[1] Bach, M. J., *The Design of the UNIX Operating System.* Prentice Hall, Englewood Cliffs, New Jersey, 1986, 471 s.

[2] Ben-Ari, M., *Principles of Concurrent and Distributed Programming.* Prentice Hall, New York, 1990, 225 s.

[3] Haario, H., *Epälineaarinen regressio.* INSKO 40-91 VIII, Helsinki, 1991, 11 s.

[4] Kahaner, D., Moler, C. & Nash, S., *Numerical Methods and Software.* Prentice Hall, Englewood Cliffs, New Jersey, 1988, 495 s.

[5] Leszak, M. & Eggert, H., *Petri-Netz-Methoden und -Werkzeuge. Hilfsmittel zur Entwerkungsspezifikation und -validation von Rechensystemen.* Springer-Verlag, Berlin, 1989, 254 s.

[6] *NetIPC Programmer's Guide, HP 9000 Series 300 and 800.* Hewlett-Packard Company, Fort Collins, Connecticut, January 1989.

[7] Peterson, J. L. & Silberschatz, A., *Operating System Concepts.* Addison-Wesley, Reading, Massachusetts, 1985, 625 s.

[8] Reisig, W., *Petri Nets.* Springer-Verlag, Berlin, 1985, 161 s.

[9] *SCO UNIX System V, Development System, Programmer's Reference.* The Santa Cruz Operation, Inc., Santa Cruz, California, 1989.

[10] *Simnon Reference Manual for UNIX Systems, Version 3.0.* SSPA Systems, Göteborg, March 1990, 112 s.

[11] *Simnon User's Guide for MS-DOS Computers, Version 3.0.* SSPA Systems, Göteborg, January 1990, 228 s.

[12] *SimuSolv, Modeling and Simulation Software.* The Dow Chemical Company, Midland, Michigan, 1990, 10 s.

[13] Stevens, W. R., *UNIX Network Programming.* Prentice Hall, Englewood Cliffs, New Jersey, 1990, 772 s.

[14] TKK, *Kurssin Tik-76.152, Tietokoneverkot, opetusmonisteet.* Otapaino, Espoo, 1989, 215 s.