## Opetusmateriaali

Marko Mäkelä ja
Teemu Tynjälä: luentokalvot (opintojakson kotisivulta tai opetusmonisteina)
Mikko Tiusanen: *Petri-verkkojen perusteet* (opetusmonisteina?)
Marko Mäkelä: MARIA-käyttöohje: http://www.tcs.hut.fi/Software/maria/
David Walker: *Introduction to a Calculus of Communicating Systems*

## Muuta lukemista

Tadao Murata: *Petri Nets: Properties, Analysis and Applications*
Robin Milner: *Communication and Concurrency*





## Tiedottaminen

- Kotisivu http://www.tcs.hut.fi/Teaching/T-79.179/
  - harjoitustehtävät
  - kotitehtävät
- Keskusteluryhmä nntp://news.tky.hut.fi/opinnot.tik.rhj
  - pikaiset ilmoitusasiat
  - keskustelua harjoitus- ja kotitehtävistä



# T-79.179 Rinnakkaiset ja hajautetut digitaaliset järjestelmät

## Johdanto

Marko Mäkelä

15. tammikuuta 2002



## Kevään 2002 opintojakso

**Luennot:** TkL Marko Mäkelä, 451 3365, Marko.Makela@hut.fi
DI Teemu Tynjälä, 451 5138, Teemu.Tynjala@hut.fi
**Harjoitukset:** Jukka Honkola, 451 5244, Jukka.Honkola@hut.fi
**Suorittaminen:** Tentti ja pakolliset kotitehtävät
(vähimmäispisteraja; mahdollisuus parantaa arvosanaa)
**Esitiedot:** T-79.144 Logiikka tietotekniikassa, perusteet
T-79.148 Tietojenkäsittelyteorian perusteet

*(This course corresponds to T-79.231, which is lectured in English.)*

## Rinnakkaiset ja hajautetut järjestelmät: mitä ne ovat?

- rinnakkaisuus: tapahtumien samanaikaisuus (concurrency) ja suoritusjärjestysten vaihtoehtoisuus (epädeterminismi)
- hajautus: järjestelmää ei hallita täysin keskitetysti

  > Hajautettu järjestelmä on sellainen, jossa entuudestaan tuntemattoman koneen hajoaminen saa koneesi käyttökelvottomaksi. *L. Lamport*

- asynkronisuus: järjestelmän osia ei ole tahdistettu muuten kuin viestiliikenteellä
- reaktiivisuus: jatkuvatoimisen järjestelmän toimintaa säätelevät ulkoiset ärsykkeet





## Rinnakkaiset ja hajautetut järjestelmät: suunnitteleminen

Rinnakkaisuutta sisältävät järjestelmät ovat usein niin mutkikkaita, ettei niitä ole mahdollista tehdä huonotapaisille ohjelmoijille tutulla yrityksen ja erehdyksen menetelmällä. Vaikka perustapaus toimisikin, erikoistapaukset voivat aiheuttaa ongelmia.

- epädeterminismi (vaihtoehtojen suuri määrä):
  - paikallinen haarautuminen: mielivaltainen syöte ja laaja `switch`-lohko
  - ajassa haarautuminen: jokin ärsyke voi saapua hyvin monella eri hetkellä
- järjestelmän nykyisten ja tulevien osien keskinäiset riippuvuudet:
  - hajautettuja toimintoja ohjaava koodi tuppaa pirstoutumaan
  - laajennukset on helpointa ottaa huomioon abstraktissa mallissa





## Opintojakson tavoite

Kurssin tavoitteena on antaa valmiuksia rinnakkaisten ja hajautettujen järjestelmien mallintamiseen sekä niitä koskevien vaatimusten esittämiseen ja tarkistamiseen.

- Mallintaminen: Petri-verkot, siirtymäjärjestelmät ja prosessialgebra
- Vaatimusten esittäminen: aikalogiikka
- Vaatimusten tarkistaminen: saavutettavuusanalyysi, työkalujen käyttö





## Muita aihepiiriin liittyviä opintojaksoja

| | |
|---|---|
| T-79.186 | Reaktiiviset järjestelmät |
| T-79.190 | Rinnakkaisohjelmistojen testaus |
| T-79.193 | Rinnakkaisohjelmistojen määrittelymenetelmät |
| T-76.164 | Sulautetut järjestelmät |
| T-76.166 | Rinnakkaisohjelmointi |
| T-79.146 | Logiikka tietotekniikassa: Erityiskysymyksiä I |
| T-79.185 | Verifiointi |

## Menetelmiä

- Petri-verkot: paikka–siirtymä-verkot ja korkean tason verkot
- Aikalogiikka
- MARIA ja algebralliset järjestelmäverkot
- Siirtymäjärjestelmät ja prosessialgebra
- Tila-avaruuden kutistusmenetelmien perusperiaatteet: osittaisjärjestyskutistukset ja vastaavuudet (ekvivalenssit)





## Menetelmien käyttökohteita

- tietoliikenne
  - yhteyskäytäntöjen (protokollien) tarkistaminen (verifiointi) ja testaaminen
  - suorituskyvyn arviointi
- turvallisuuskriittiset sulautetut järjestelmät
  - rautateiden kulunvalvonta (JKV)
  - lentokoneiden ja lennonjohdon järjestelmät
- laitteistosuunnittelu: *locally synchronous, globally asynchronous*
  - suorittimet, liitäntäpiirit, välimuistit ja muistiväylät
  - ohjelmoitavan logiikan ja mikro-ohjainten työnjako
- mitkä tahansa reaktiiviset järjestelmät





## Rinnakkaiset ja hajautetut järjestelmät: mihin tarvitaan?

- tehostaminen: monta pienitehoista suoritinta saa edullisemmin kuin yhden huipputehokkaan, eikä yhden tehokkaimman suorittimen teho välttämättä riitä sovellukseen
- redundanssi: järjestelmä pysyy toiminnassa huoltojaksojen ja joidenkin laitteiden toimintahäiriöiden aikana
- modulaarisuus: voi olla helpompi hallita monta erikoistunutta prosessia kuin yhtä monimutkaista, joka huolehtii kaikesta
- maantieteellinen hajautus: varautuminen suuronnettomuuksiin ja sotatoimiin





## Järjestelmien ominaisuuksia

- turvallisuus *(safety)* : "järjestelmä ei joudu pahaan tilaan"; kussakin tilassa pätee $P$
  - lukkiutumattomuus *(deadlock freedom)*
  - keskinäinen poissulkevuus *(mutual exclusion)* jne.
- elävyys *(liveness)* : "järjestelmä etenee"; $X$ tapahtuu äärettömän usein
- reiluus *(fairness)* ; $X$:n jälkeen $Y$ tapahtuu $n$ askeleen kuluessa
  - lähetetyt viestit saapuvat perille
  - kukin palvelupyyntö suoritetaan
- vakautuvuus *(self-stabilisation)* : "järjestelmä toipuu häiriöstä äärellisessä ajassa"

## Mitä määrittäminen (spesifiointi) on?

- Määrittäminen on järjestelmän toimintojen oletusten ja vaatimusten kuvaamista.
- Määritysten tulisi olla yksikäsitteisiä mutta ei välttämättä täydellisiä.
- Määrityksissä kuvataan sallitut laskennat (tai suoritukset).
- Määritystä voidaan pitää asiakkaan ja toimittajan välisenä *sopimuksena* siitä, mitä toimitettava ohjelma tekee.
- Spesifioinnilla tarkoitetaan yleensä järjestelmän formaalia kuvaamista.





## Formaalien kuvausmenetelmien etuja

- oikeellisuus: automaattinen tarkistus, että järjestelmä toteuttaa vaatimusmääritykset
- täydellisyys: määritykset toimivat muistilistana
- yhdenmukaisuus: määrityksistä voidaan varhaisessa vaiheessa havaita ristiriitaisia tai kohtuuttomia vaatimuksia
- uudelleenkäytettävyys: määritykset ovat varsin riippumattomia ohjelmisto- ja laitteistoympäristöstä, ja samat ratkaisut sopivat erilaisiin hankkeisiin *(“design patterns”)*

Ohjelmistotuotannon kuvausmenetelmät, kuten UML, eivät usein ole riittävän formaaleja.





## Ohjelmistotuotannon ongelmia

- Miten suunnitellaan ja toteutetaan ohjelmisto siten, että
  - se kyetään toimittamaan ajallaan tilauksen mukaisena,
  - sen kustannukset pysyvät ennalta määrätyissä rajoissa,
  - asiakas saa odotuksiaan vastaavan ratkaisun ja
  - ohjelmistoa on helppo laajentaa?
- Ohjelmien tekeminen on pitkälti käsityötä.
- Formaalit menetelmät *(formal methods)* tarjoavat ratkaisuja joihinkin ongelmiin.





## Ohjelmistokriisi

Ohjelmien koon kasvaessa niiden tuotanto- ja ylläpitokustannukset kasvavat. Mitä myöhemmin virhe havaitaan, sen kalliimpaa se on korjata (enemmän koodimuutoksia tai jopa asiakastoimituksia). Sopimussakot ja maineen menettäminenkin maksavat.

- puhelinverkko: vaihteiden kaatumisia, *feature interaction* , laskutusongelmia
- avaruusluotainten virheitä: Mariner I, Ariane 5, ...
- Pentium-suoritinten virheitä: `fpdiv`, `f0 0f`, ...
- huonosti ohjelmoitujen WWW-palvelinten ylläpito sitoo tolkuttomasti resursseja
- muita esimerkkejä virheistä: `nntp:comp.risks`

# Esimerkki: Vaihtuvan bitin yhteyskäytäntö (1/6)

Liukuvan ikkunan yhteyskäytäntö on vuonna 1969 julkaistun vaihtuvan bitin yhteyskäytännön yleistys. Vaihtuvan bitin yhteyskäytännössä sanomanumeroita on kaksi: vaihtuvan bitin tilat 0 ja 1. Sekä lähettäjällä että vastaanottajalla on oma käsitys vaihtuvan bitin kulloisestakin tilasta. Lähettäjällä se on muuttujan $l$ arvo, vastaanottajalla muuttujan $v$.

- lähettäjä: lähetä sanoma $l$:llä täydennettynä
  - jos vastaanottaja ei kuittaa $l$:llä määräajassa, toista sanoma
  - jos vastaanottaja kuittaa $l$:llä, vaihda $l$:n tila: $l \leftarrow 1 - l$.
- vastaanottaja: ota vastaan sanoma ja lähetyshetken $l$:n arvo $b$, ja kuittaa $b$:llä.
  - välitä sanoma ja vaihda $v \leftarrow 1 - v$, jos $v = b$
  - muutoin unohda sanoma

Sanomat ja kuittaukset vuorottelevat linjalla. Väitämme, että yhteyskäytäntö toipuu siitä, että kanava hävittää sanomia tai kuittauksia äärellisen monta kertaa.





# Esimerkki: Vaihtuvan bitin yhteyskäytäntö (2/6): C-koodi

Yhteyskäytäntö voidaan toteuttaa kahdella silmukassa kutsuttavalla aliohjelmalla:

```
void
abp_send (char c)
{
  static char b;
  do
    send (data, b, c);
  while (receive_timeout (ack) != b);
  b = !b;
}
```

```
char
abp_receive (void)
{
  static char b_expect;
  char b, c;
  do
    b = receive (data, &c),
    send (ack, b);
  while (b != b_expect);
  b_expect = !b_expect;
  return c;
}
```





# Spesifiointi ohjelmistotuotannossa

- vaatimusten esittäminen: asiakkaan tarpeiden selventäminen
- suunnittelu: ositus, osien liittäminen *(composition)* ja tarkentaminen *(refinement)*
- verifiointi: toteuttaako järjestelmän formaali malli halutut ominaisuudet
- validointi: vastaako järjestelmän toteutus mallista johdettuja testitapauksia
- toteutuksen dokumentointi
- analysointi ja arviointi: olemassaolevan järjestelmän mallintaminen sen ymmärtämistä ja jatkokehitystä varten *(reverse engineering)*

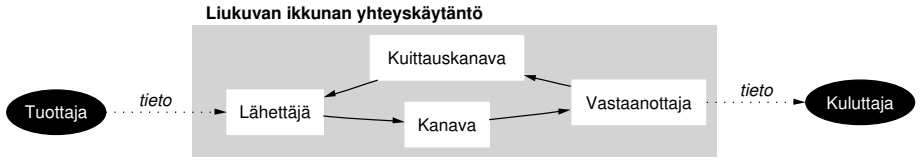




# Esimerkki: luotettava yhteys häviöllisellä kanavalla

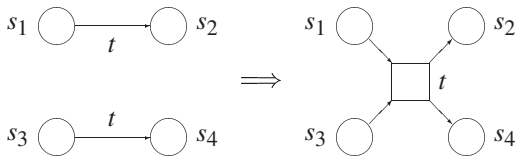
Monet tietoliikennejärjestelmät perustuvat epäluotettavaan yhteyteen, joka voi vääristää, hävittää tai toistaa siirrettäviä sanomia. Vääristymiset voidaan muuttaa häviämisiksi tarkistussummia käyttämällä, ja häviämiset ja toistot havaitaan numeroimalla sanomat.

Perusratkaisu, liukuvan ikkunan yhteyskäytäntö *(sliding window protocol)* , perustuu matkalla olevien sanomien numerointiin siten, että vastaanottaja voi havaita sanoman katoamisen ja pyytää lähettäjää toistamaan aiempia sanomia. Aina, kun vastaanottaja on saanut yhtenäisen rykelmän peräkkäin numeroituja sanomia, se välittää ne kuluttajalle.



# Petri-verkot

Siirtymäkaavioiden luettavuutta haittaa se, että yksittäiset automaatit ovat täysin sekventiaalisia ja että automaattien väliset tahdistukset on esitetty nimien avulla. Petri-verkot ovat eräänlainen siirtymäkaavioiden yleistys, jossa tahdistukset kuvataan graafisesti:



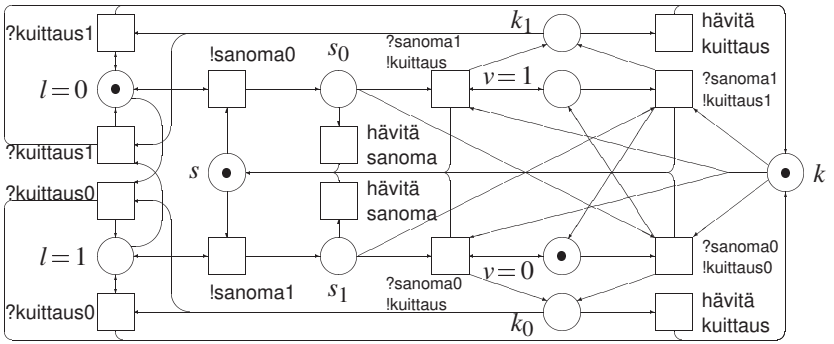

Petri-verkoissa on paikkoja ◯ (Stelle; *place* ) ja siirtymiä (Transition; *transition* ) □, joiden välillä on suunnattuja kaaria.

Petri-järjestelmän tila koostuu paikallisista tiloista (paikkojen merkinnöistä).





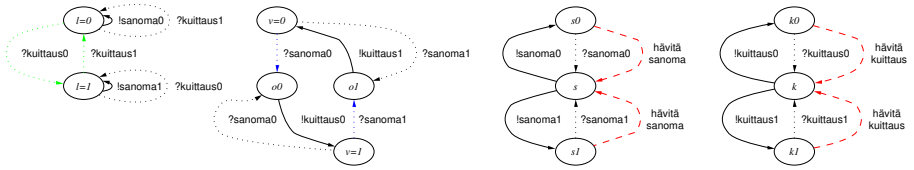
# Esimerkki: Vaihtuvan bitin yhteyskäytäntö (5/6): Petri-verkko








# Esimerkki: Vaihtuvan bitin yhteyskäytäntö (3/6)




Kuvassa lähettäjän, vastaanottajan ja kanavien toiminta on esitetty siirtymäkaavioina *(labelled transition systems)* eli kommunikoivina tilakoneina. Huutomerkki kuvaa viestin lähettämistä ja kysymysmerkki vastaanottamista. Samannimiset tapahtumat suoritetaan samanaikaisesti eri tilakoneissa. Kanaviin mahtuu enintään yksi sanoma kerrallaan.

Selkeyden vuoksi tuottajan ja kuluttajan toiminta sekä sanomien sisältö on jätetty pois.





# Esimerkki: Vaihtuvan bitin yhteyskäytäntö (4/6): tila-avaruus

Sanomien sisältö kiinnostaa sovelluksen käyttäjää, mutta yhteyskäytännön toiminnan kannalta sillä ei ole merkitystä. Mitä vähemmän muistia malli sisältää, sen helpompi sitä on tarkistaa, sillä $b$ bittiä muistavalla järjestelmällä on enintään $2^b$ tilaa.

Sanomien lähettämistä ohjaavat ajastimet on yksinkertaistettu pois. Täysin asynkronisessa järjestelmässä ajastimen ajatellaan voivan laueta milloin tahansa. Laukeamisia voitaisiin rajoittaa mallintamalla ajan kulumista, mutta se hankaloittaisi tarkistamista.

Hajautetun järjestelmän tila koostuu sen kaikkien osajärjestelmien tiloista, esimerkiksi $\{l=0, l=1\} \times \{v=0, o0, v=1, o1\} \times \{s, s0, s1\} \times \{k, k0, k1\}$. Alussa kukin osajärjestelmä on alkutilassaan, jolloin koko järjestelmän alkutila on esimerkiksi $\langle l=0, v=0, s, k \rangle$.

# Vaihtuvan bitin yhteyskäytäntö (6/6): saavutettavuusgraafi

Osoittautuu, että tilasta $\langle l=0, v=0, s, k\rangle$ saavutettavia tiloja on 18 ja siirtymiä 40:

# Merkkipeli (1/2)

- Petri-järjestelmän tila muodostuu verkon paikkojen sisältämien merkkien määristä.
- Järjestelmän tila muuttuu virittyneiden siirtymien laukeamisten välityksellä.
- Siirtymä on virittynyt, jos kussakin siihen kytketyssä esipaikassa on merkki.
- Kun virittynyt siirtymä laukeaa, sen kustakin esipaikasta poistetaan merkki ja sen kaikkiin jälkipaikkoihin lisätään merkki.





# Sama järjestelmä, monta esitystapaa

Rinnakkaisia ja hajautettuja järjestelmiä voidaan kuvata hyvin monella eri tavalla:

- siirtymäjärjestelmillä tai prosessialgebralla,
- Petri-järjestelmillä (paikka–siirtymä-järjestelmillä tai korkean tason järjestelmillä),
- jollakin puoliformaalilla määrittelykielellä (kuten UML) tai
- jollakin ohjelmointikielellä.

Esitystapa on järkevää valita kuvattavan kohteen ja halutun tarkkuuden mukaan. Formaalejakin kuvauksia voi esittää monella tavalla: graafisesti, taulukkoina tai tekstinä.





# Merkkipeli (2/2)

- Järjestelmän käyttäytymistä voidaan nyt kuvata siirtymäketjuilla. Esimerkiksi viestin onnistunut lähetys: !sanoma0 ?sanoma0 !kuittaus0 ?kuittaus0.
- Siirtymäketjuista voidaan muodostaa suunnattu graafi, joka kuvaa järjestelmän kaikkia mahdollisia siirtymiä:
  - tila-avaruus: järjestelmän alkutilasta saavutettavissa olevat tilat
  - saavutettavuusgraafi: solmut ovat tiloja ja kaaret tilojen välisiä siirtymiä
- Merkkipeliä pelaamalla on “melko” helppoa varmistua vaihtuvan bitin yhteyskäytännön oikeellisuudesta. Tiloja on enintään $2 \cdot 4 \cdot 3 \cdot 3 = 72$.