IAF0542

Sardsüsteemid
II Loeng

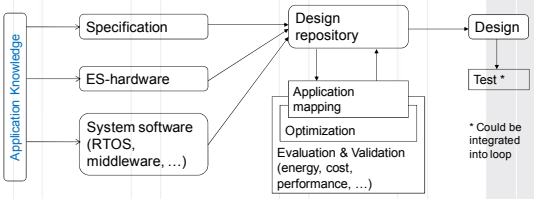
Gert Jervan
Arvutitehnika instituut

ati.ttu.ee/~gerje

www.pld.ttu.ee/IAF0542

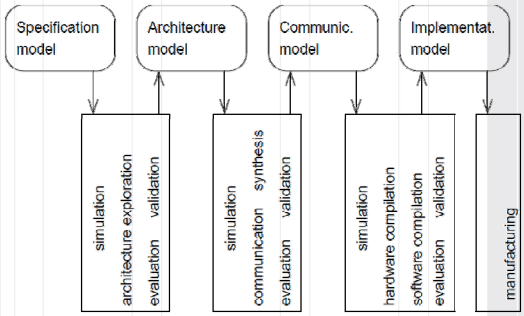
Disainivoog

Hüpoteetiline disainivoog



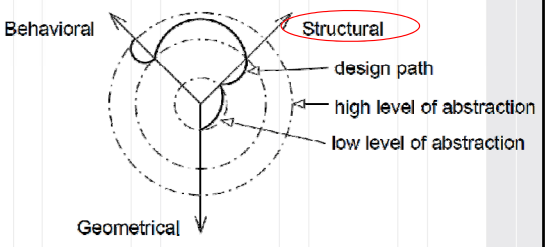
Generic loop: tool chains differ in the number and type of iterations

Iterative design (1)
- After unrolling loop -



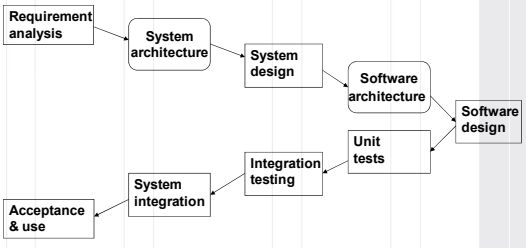
SpecC tools

Iterative design (2)
- Gajski's Y-chart -



Iterative design (3)
- After unrolling loop -

Example: V-model



Skipping some explicit repository updates ..

Arvutusmudelid

Spetsifitseerimine

Sissejuhatus

- Sardsüsteemide puhul on märksõnadeks aeg ja samaaegsus (*concurrency*)
- Lõimed (*threads*)!
  - Probleemid:
    - Absoluutselt mittedeterministlikud
    - Täitmisjärjekord tuleb jõuga paika panna (näiteks mutex-itega)
  - "*... **threads as a concurrency model are a poor match for embedded systems**. ... they work well only ... where best-effort scheduling policies are sufficient.*"

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

Von Neuman'i mudel on surnud!

- ***"The lack of timing in the core abstraction is a flaw, from the perspective of embedded software, ..."***

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

- ***"Timing is everything"***

Frank Vahid, WESE 2008

- ***What is needed is nearly a reinvention of computer science.***

Ed Lee: Absolutely Positively on Time, *IEEE Computer*, July, 2005

Spetsifitseerimine

- Sardsüsteemide jaoks on vaja arvutusmudeleid, mis ei põhineks threadidel ja mis ei põhineks von Neumanni arvutusmudelil

- *Nontrivial software written with threads, semaphores, and mutexes is incomprehensible to humans.*

Spetsifitseerimine

- Aga mis on mudel?
  - **Definition:** A model is a simplification of another entity, which can be a physical thing or another model. The model contains exactly those characteristics and properties of the modeled entity that are relevant for a given task. A model is minimal with respect to a task if it does not contain any other characteristics than those relevant for the task.

[Jantsch, 2004]

- Millised on nõuded sardsüsteemide spetsifitseerimistehnikatele ja mudelitele?

Nõudmised spetsifitseerimistehnikatele

Hierarhia

- Inimesed ei suuda aru saada süsteemidest, milles on rohkem kui ca 5 objekti.
  Tegelikud süsteemid nõuavad palju enamat

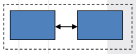
  - Käitumuslik hierarhia
    Näited: olekud, protsessid, protseduurid.

proc
proc
proc

  - Struktuurne hierarhia
    Näited: Protsessorid, räkid, trükkplaadid

Nõudmised spetsifitseerimistehnikatele

Komponentide-põhine disain



- Süsteemid peavad olema loodud komponentidest
- Struktuurne käitumine
  - Alamsüsteemide käitumisest peaks olema "kerge" tuletada süsteemi, kui terviku käitumine
- Samaaegsus
  - Reaalsetes süsteemides toimub palju tegevusi samaaegselt
- Sünkroniseerimine ja kommunikatsioon
  - Komponendid peavad suhtlema!

Nõudmised spetsifitseerimistehnikatele

Ajaline käitumine

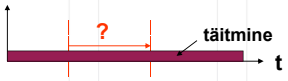
- Esmaoluline sidumaks reaalse maailmaga
  - Igasugune lisainformatsioon (perioodid, sõltuvused, stsenaariumid) on teretulnud
  - Ka kasutatava platvormi ajaline käitumine (kiirus) peaks olema teada
  - Väga suur mõju disainiprotsessile!

"The lack of timing in the core abstraction (of computer science) *is a flaw, from the perspective of embedded software"*
[Lee, 2005]

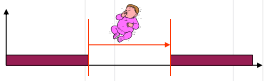
Ajaline käitumine

Neli nõuet spetsifikatsioonidele:

1. Ligipääs timerile aja mõõtmiseks

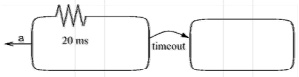


2. Protsesside viivitamise võimaldamine

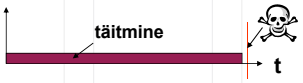


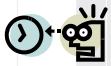
Ajaline käitumine

3. Timeoutide kirjeldamine



4. Meetodid deadline'ide ja planeeringute (*schedule*) kirjeldamiseks

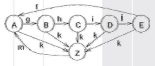




Nõudmised spetsifitseerimistehnikatele

Tugi reageerivate süsteemide loomiseks

- Olekutele suunatud käitumine
  Vajalik reageerivatele süsteemidele;
  Tavaline automaadimudel ei ole piisav.



- Sündmuste käsitsemine
  (sisemised või välised sündmused)

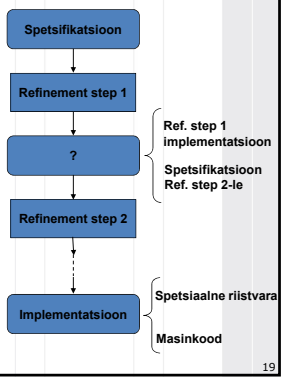
- Eranditele (*exception*) suunatud käitumine

Nõudmised spetsifitseerimistehnikatele

- Programmeerimiselementide olemasolu
  - Näiteks peaks olemas olema: aritmeetilised operatsioonid, tsüklid ja funktsioonide väljakutsed
- Täidetav (ei ole algebralisi spetsifikatsioone)
- Tugi suurte süsteemide kirjeldamiseks (☞ OO)
- Valdkonna-spetsiifilisus
- Loetavus
- Porditavus ja paindlikkus
- Lõpetamine
  - Peaks olema selge, milliseks ajahetkeks on kõik arvutused lõppenud
- Tugi mitte-standartsetele I/O seadmetele
  Otsene ligipääs lülititele, displeidele, ...
- Mitte-funktsionaalsed omadused:
  - Veakindlus, kättesaadavus, EMC-omadused, kaal, suurus, kasutajasõbralikkus, laiendatavus, eluiga, võimsustarve, ...
- Sobiv arvutusmudel
  - Aga mida tähendab "arvutamine"?

Spetsifikatsioonid ja implementatsioonid

- Spetsifikatsioon: Süsteemi käitumuse ja muude omaduste kirjeldus
  - Projekteerija saab oma tööks (sisendiks) spetsifiaktsiooni ja loob selle põhjal implementatsiooni (toote). Toodet luuakse paljude täiustavate sammude jooksul (refinement steps)



Spetsifikatsioon

- Spetsifikatsioonid võivad olla:
  - Mitteformaalsed (loomulikus keeles)
  - Detailsemad ja ühetähenduslikumad, kasutades spetsifikatsioonikeeli
- Spetsifikatsioonikeeled peavad:
  - Olema võimelised hästi väljendama peamisi süsteem omadusi ja erinevaid aspekte sisutihedal ja selgel kujul
  - Sobima hästi nõuete täitmise kontrolliks ja implementatsioonide sünteesiks (soovitavalt automaatselt)
- Alati tuleb valida see keel, mis antud süsteemi jaoks sobiks kõige paremini!

Spetsifikatsioonikeeled

- Spetsifikatsioonikeeled võivad olla
  - Graafilised
  - Tekstilised
- Spetsifikatsioonikeeled võivad olla
  - Tavalised programmeerimiskeeled (C, C++)
  - Riistvara kirjelduskeeled (VHDL, Verilog)
  - Spetsiaalsed keeled, mida kasutatakse erinevates valdkondades süsteemide spetsifitseerimiseks. Tihti põhinevad need mingil *arvutusmudelil* (model of computation)

Süsteemide spetsifitseerimine

- Mida me tahame sardsüsteemi spetsifikatsiooniga peale hakata?

1. Valideerida süsteemi kirjeldust, et kontrollida, kas funktsionaalsus vastab soovitule ja et vajadused on kirjeldatud korrektselt. Selleks kasutatakse:
   - Formaalset verifitseerimist
   - Simuleerimist
2. Et sünteesida efektiivseid rakendusi

Formaalsed mudelid

- Me sooviksime, et spetsifitseerimiskeeled oleks hästi defineeritud semantikaga ➔ spetsifikatsioonid peaksid olema ühetähenduslikud
  - Semantika on reeglite kogu, mis seob tähenduse (interpretatsiooni) süntaktiliste keelekonstruktsioonidega (sümbolite kombinatsiooniga)
  - Semantika põhineb aluseks oleval arvutusmudelil

  Nimetatud mudel määrab ära, milliseid süsteeme saab selle keelega kirjeldada
  Arvutusmudel määrab ära keele väljendusvõime

Formaalsed mudelid (2)

- Kas me sooviksime kasutada suure väljendusvõimega keeli (et saaksime kirjeldada mida iganes)?
  Vahest mitte!
  - Suur väljendusvõime: imperatiivne mudel (näiteks C või Java piiranguteta kasutamine):
    - Võime kirjeldada "kõike"
    - Puudub võimalus formaalseks analüüsiks (või see on väga keerukas)
  - Piiratud väljendusvõime, mis põhineb hästi valitud arvutusmudelil:
    - Spetsifitseerida saab ainult valitud süsteeme
    - Formaalne analüüs on võimalik
    - Efektiivse (võib-olla isegi automaatse) sünteesi võimalus

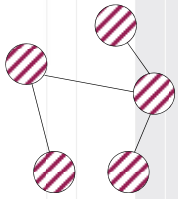
Arvutusmudelid (Models of Computation)

Arvutusmudelid

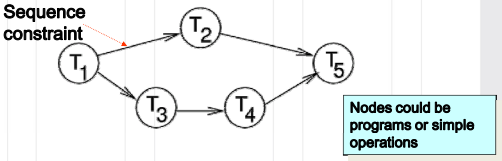
- Arvutusmudelid (*models of computation*) käsitlevad keele täitmismudeli (*execution model*) loomiseks vajalikke teoreetiliste valikute kogumeid
  - Disain on esitatud kui komponentide kogum, mida võib vaadelda kui isoleeritud monoliitseid mooduleid (tihti kutsutakse neid protsessideks – *processes* või ülesanneteks – *tasks*), mis suhtlevad omavahel ja ümbruskonnaga
  - Arvutusmudel defineerib nende moodulite käitumise ning omavahelise suhtlemise

Arvutusmudelid (2)

- Arvutusmudelid esitavad:
  - Süsteemi loomiseks vajalikke komponente
  - Kuidas iga komponent (protsess või ülesanne) teostab oma sisemisi arvutusi
  - Kuidas komponendid vahetavad omavahel informatsiooni
  - Kuidas nad seostuvad omavahel kattuvuse (concurrency) mõistes
- Mõningad arvutusmudelid ei kajasta moodulite sisemust, vaid üksnes nende suhtlemist ja kattuvust



Dependence graph

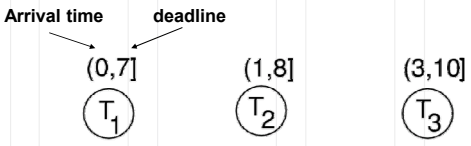


- **Def.:** A **dependence graph** is a directed graph $G=(V,E)$ in which $E \subseteq V \times V$ is a partial order.
- If $(v1, v2) \in E$, then $v1$ is called an **immediate predecessor** of $v2$ and $v2$ is called an **immediate successor** of $v1$.
- Suppose E^* is the transitive closure of E. If $(v1, v2) \in E^*$, then $v1$ is called a **predecessor** of $v2$ and $v2$ is called a **successor** of $v1$.

Dependence graph

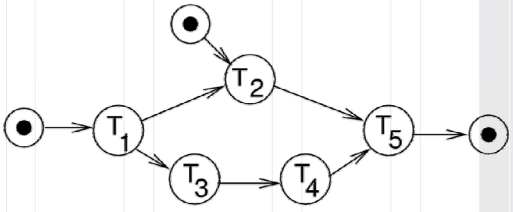
Aja mõiste

- Sõltuvusgraafid võivad kanda täiendavat informatsiooni, nagu näiteks ajalist informatsiooni



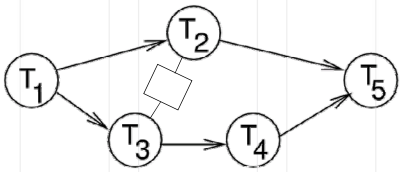
Dependence graph

I/O-informatsioon



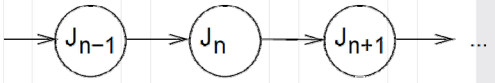
Dependence graph

Jagatud ressursid



Dependence graph

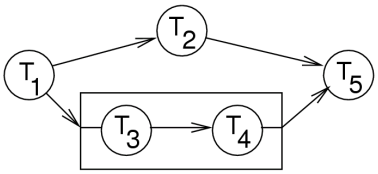
Perioodilised planeeringud



- Üks töö (*job*) on sõltuvusgraafi ühekordne läbimine
- Perioodilised sõltuvusgraafid on lõpmatud

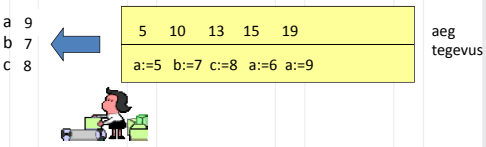
Dependence graph

Hierarhilised ülesandegraafid



Komponendid

- Von Neumann'i mudel
  - Järjestikuline täitmine
- Diskreetsed sündmused
  - järjekord



Komponendid (2)

- Automaadid

- Differentsiaalvõrrandid

$$\frac{\partial^2 x}{\partial t^2} = b$$

Kattuvus (*Concurrency*)

- Süsteemid koosnevad tegevuste (protsessid või ülesanded) kogumist. Neid tegevusi võib potensiaalselt täita paralleelselt, ehk teisisõnu: nad on kattuvad (*concurrent*).
- Kuidas väljendada kattuvust?
  See on üks aspekt, milles arvutusmudelid erinevad
  - Andmete-põhine kattuvus
  - Kontrolli-põhine kattuvus

Andmete-põhine kattuvus

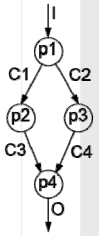
- Süsteem on kirjeldatud kui protsesside kogum ilma määratlemata täitmisjärjekorraga

- Protsesside täitmise järjekord (ja selle põhjal võib kaudselt teha järeldusi parallelismi kohta) on fikseeritud ainult andmete sõltuvuse põhjal
  - Väga tüüpiline paljudes DSP rakendustes

Andmete-põhine kattuvus (2)

```
Process p1( in int a, out int x, out int y) {
.....
}
Process p2( in int a, out int x) {
.....
}
Process p3( in int a, out int x) {
.....
}
Process p4( in int a, in int b, out int x) {
.....
}
channel int I, O, C1, C2, C3, C4;
p1(I, C1, C2);
p2(C1, C3);
p3(C2, C4);
p4(C3, C4, O);
```

Ei ole oluline, mis järjekorras me need kirjutame



Kontrolli-põhine kattuvus

- Protsesside täitmise järjekord on üheselt kirjeldatud süsteemi spetsifikatsioonis
- Kasutatakse spetsiaalseid konstruktsioone et määrata ära täitmise järjekord ja kattuvus

Kontrolli-põhine kattuvus (2)

```
module p1:
.....
end module

module p2:
.....
end module

module p3:
.....
end module

module p4:
.....
end module

run p1;
[run p2 || run 3];
run p4
```

Siin on kirjutamise järjekord esmaoluline

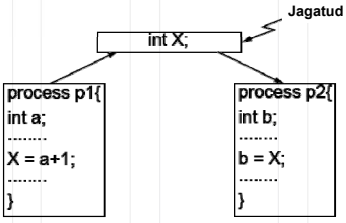
- See näide on kirjutatud ESTERELis
- Protsess p1 algab esimesena ja peab lõppema enne kui p2 ja p3 algavad
- p2 ja p3 algavad paralleelselt
- p2 ja p3 peavad mõlemad lõppema, enne kui p4 saab alustada

Kommunikatsioon

- Protsessid peavad info vahetuseks kommunikeeruma
- Erinevad arvutusmudelid kasutavad erinevaid arvutusmudeleid
- Jagatud mälu (*shared memory*)
- Sõnumite edastamine (*message passing*)
  - Blokeeriv
  - Mitte-blokeeriv

Jagatud mälu

- Iga saatev protsess kirjutab jagatud muutujatesse, mida saavad omakorda vastuvõtvad protsessid lugeda



Lokaalsed muutujad:
- a: p1
- b: p2

Jagatud muutujad:
- X

Jagatud mälu (2)

- Võivad tekkida võidujooksud (*race conditions*) ☞ tagajärjeks vastuolulised andmed
  ☞ Kriitilised sektsioonid = sektsioonid, kus ressursile (näiteks jagatud mälu) tuleb garanteerida ainuõiguslik ligipääs

```
process a {
  ..
  P(S)  //lukustamine
  ..    //kriitiline sektsioon
  V(S)  //luku vabastamine
}
```

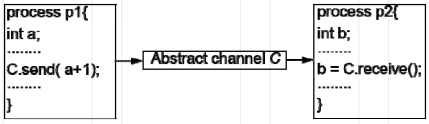
```
process b {
  ..
  P(S)  //lukustamine
  ..    //kriitiline sektsioon
  V(S)  //luku vabastamine
}
```

Ilma võidujooksuta ligipääs jagatud mälule, mida kaitseb lukk S

- Seda mudelit kasutavad:
  - Kriitiliste sektsioonide vastastikune välistamine
  - Cache coherency (jagatud vahemälu) protokollid

Sõnumite edastamine

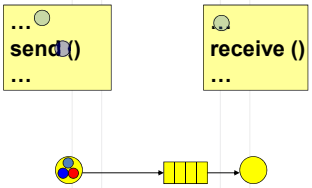
- Andmed edastatakse üle abstraktse kommunikatsioonikeskkonna, mida nimetatakse kanaliks



- See kommunikatsioonimudel on piisav kirjeldamaks hajussüsteeme (*distributed systems*)

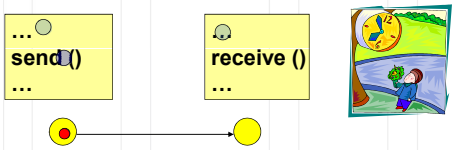
Mitteblokeeriv (asünkroonne)

- Saatja ei pea ootama kuni sõnum on kohale jõudnud; Potensiaalne probleem: Puhvri täitumine



Blokeeriv sõnumite edastamine – rendez-vous

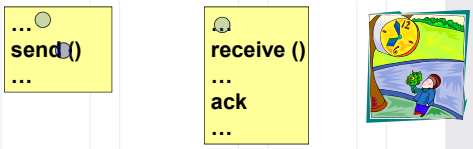
- Saatja ootab kuni vastuvõtja on sõnumi kätte saanud



- Protsess, mis suhtleb, blokeerib end (suspends), kuni teine protsess on valmis andmete ülekandeks
- Need kaks protsess peavad ennast enne andmete ülekannet sünkroniseerima

Laiendatud rendez-vous

- Vastuvõttev pool peab saatma spetsiaalse teate kättesaamise kohta. Vastu võttev pool võib teostada enne teate saatmist andmete kontrolli.



Sünkroniseerimine

- Sünkroniseerimist ei saa eraldada kommunikatsioonist
  - Iga protsesside vaheline suhtlemine eeldab mõningast kommunikatsiooni ja sünkroniseerimist
- Sünkroniseerimine: Üks protsess on seisatud (*suspended*) kuni teise täitmine jõuab mingi punktini
  - Kontrolli-põhine sünkroniseerimine
  - Andmete põhine sünkroniseerimine

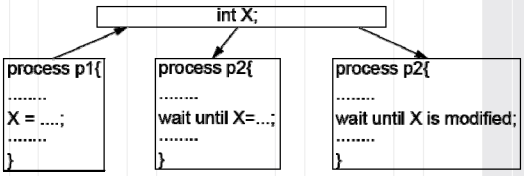
Kontrolli-põhine sünkroniseerimine

```
module p1:
.....
end module

module p2:
.....
end module

module p3:
.....
end module

module p4:
.....
end module

run p1;
[run p2 || run 3];
run p4
```

- Kontrolli-põhise sünkroniseerimise puhul tegeleb sünkroniseerimisega kontrollistruktuur
- Siin on mitmeid sünkroniseerimise punkte:
  - peale p1 lõpetamist ja enne p2, p3 algust
  - Peale p2 ja p3 lõppemist ning enne p4 algust

Andmete põhine sünkroniseerimine

- Kommunikatsioonimehhanismid väljendavad kaudselt ka sünkroniseerimist
- Jagatud mälu põhine sünkroniseerimine



Andmete põhine sünkroniseerimine (2)

- Sõnumite edastamise põhine sünkroniseerimine
  - Kommunikatsiooni blokeerimine sõnumitega tähendab automaatselt saatja ja vastuvõtja vahelist sünkroniseerimist

Protsesside omadused

- Protsesside arv
  - Staatiline;
  - Dünaamiline (Dünaamiliselt muutuv riistvaraplatvorm?)
- Käitumuslik hierarhia:
  - Protsesside rekursiivne deklareerimine (ADA, VHDL)
    ```
    process {
      process {
        process {
    }}}
    ```
  - või kõik deklareeritud samal tasemel (samm struktuurse hierahia suunas)
    ```
    process { … }
    process { … }
    process { … }
    ```


Protsesside omadused (2)

- Erinevad tehnikad protsesside loomiseks
  - Ilmutatud kujul koodis (vt. ADA)
    ```
    declare
      process P1 …
    ```
  - fork ja join (vt. Unix)
    ```
    id = fork();
    ```
  - spetsiaalsed protsessi loomise funktsioonid
    ```
    id = create_process(P1);
    ```


Sünkroonsed v. asünkroonsed keeled

- Mitmete protsesside kirjeldamine on paljudes keeltes mittedeterministlik:
  Ülesannete täitmise järjekord ei ole kindlaks määratud (võib mõjutada tulemust).
- Sünkroonsed keeled: põhinevad automaatide teoorial.
- „Sünkroonsete keelte eesmärgiks on pakkuda kõrgtaseme, modulaarseid konstruktsioone, et selliseid automaate oleks kergem luua” [Halbwachs].
- Sünkroonsed keeled kirjeldavad samaaegselt töötavaid automaate.

Sünkroonsed v. asünkroonsed keeled

- Sünkroonsed keeled eeldavad (globaalset) taktsignaali. Igal taktil arvestatakse kõikide sisenditega, arvutatakse uued olekud ja väljundid ning alles siis tehakse siire.
- See eeldab levitamismehhanisme kõikidesse süsteemi osadesse.
- Ideaalne vaade üheaegsele toimimisele.
- Eeliseks on deterministliku käitumise tagamine.

Tüüpilised arvutusmudelid

- Olekudiagrammid (StateCharts)
- Andmevoo (dataflow) mudelid
- Petri võrgud (Petri Nets)
- Diskreetsed sündmused (Discerete events)
- (Sünkroonsed) lõplikud olekumasinad (Finite State Machines)
- Sünkroonsed/reaktiivsed keeled
- Koosdisaini lõplikud olekumasinad
- Timed Automata

Klassifikatsioon

Communication/ local computations	**Shared memory**	**Message passing** **Synchronous | Asynchronous**	
Undefined components	Plain text, use cases | (Message) sequence charts		
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL, Verilog, SystemC, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

Kokkuvõtteks

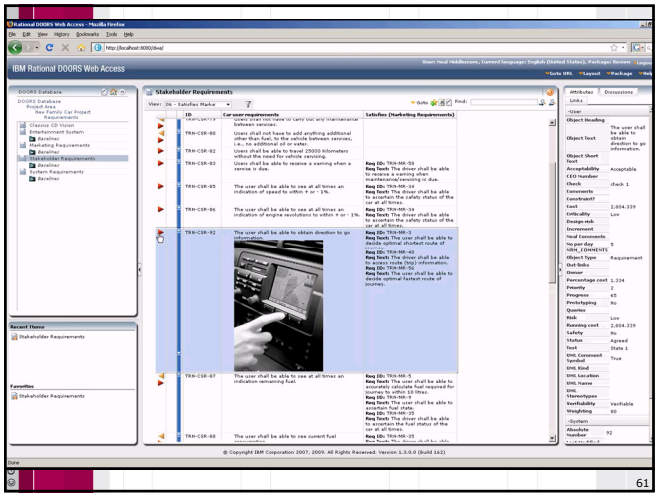
- Googelda: Edward A. Lee, UCB
- Axel Jantsch: Modeling Embedded Systems and Soc's: Concurrency and Time in Models of Computation, Morgan-Kaufman, 2004
- Võimalus ise katsetada: Ptolemy
  - Ptolemy (UC Berkeley) is an environment for simulating multiple models of computation
  - http://ptolemy.berkeley.edu/

Küsimusi?

Gert Jervan
ati.ttu.ee/~gerje

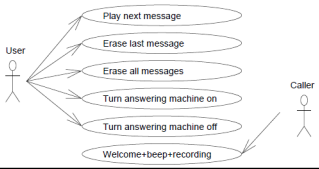
Nõudmiste hõlmamine tekstilisel kujul

- Disainiprojekti väga varastes faasides on loodav süsteem (*system under design - SUD)* üldiselt kirjeldatud vaid tavalises keeles (Eesti, rootsi, inglise...)
- Nõudmised töövahenditele:
  - Masin-loetav
  - Versioonihaldus
  - Sõltuvuste analüüs
  - Näide: IBM Rational DOORS®



Kasutusmallid (*Use cases*)

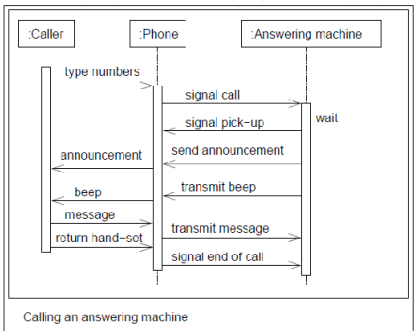
- Kasutusmallid kirjeldavad SUD võimalikku rakendamist
- Kasutusel UMLis (Unified Modeling Language)
- *Neither a precisely specified model of the computations nor a precisely specified model of the communication*
- Näide: automaatvastaja



Järgnevusdiagrammid *(sequence charts)*

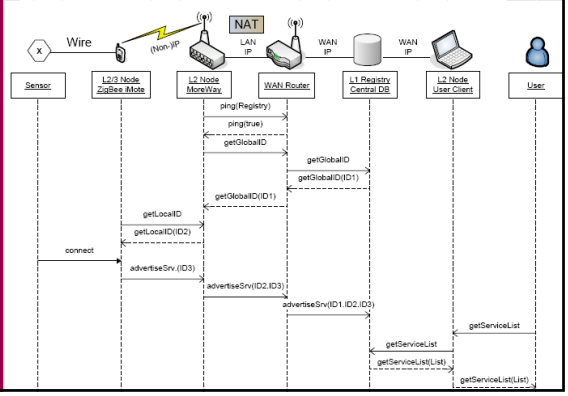
- Kirjeldavalt üheselt informatsiooni liikumist
- Üks dimensioon (tavaliselt vertikaalne) on aeg
- Teine kirjeldab ruumilist jaotust
- Varem kutsuti ka *Message Sequence Charts*
- Sisalduvad UMLis

Näide



Automaatvastaja kirjeldus UMLis

Näide

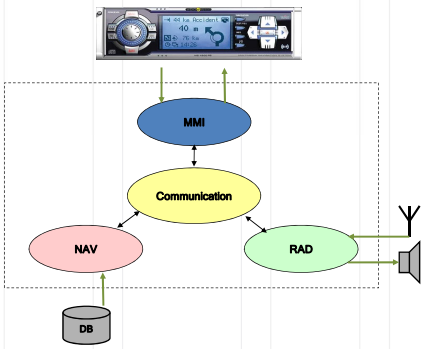


Application: In-Car Navigation System

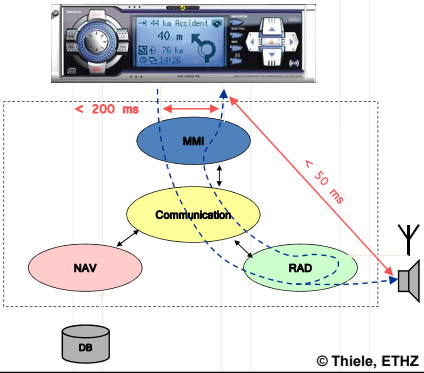
- Car radio with navigation system
- User interface needs to be responsive
- Traffic messages (TMC) must be processed in a timely way
- Several applications may execute concurrently

© Thiele, ETHZ

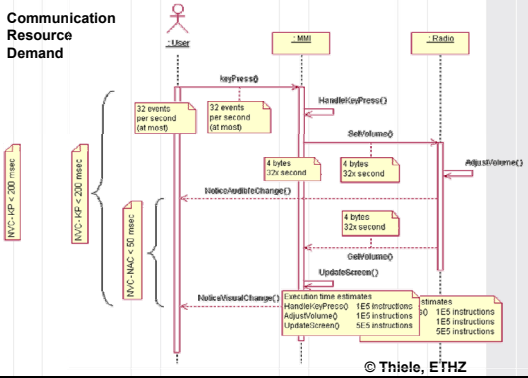
System Overview



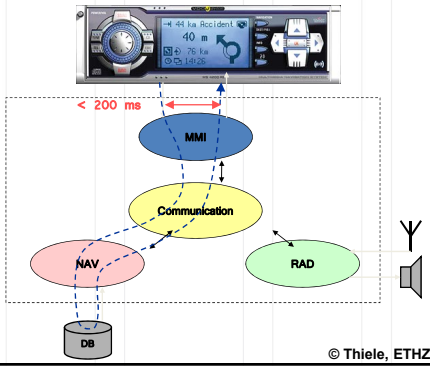
Use case 1: Change Audio Volume



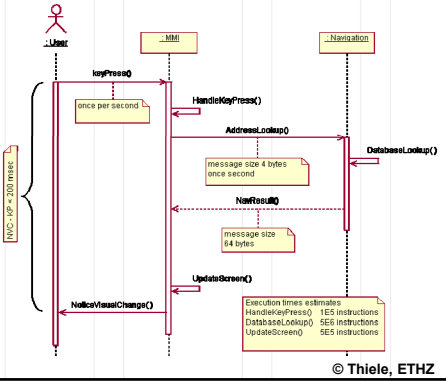
Use case 1: Change Audio Volume



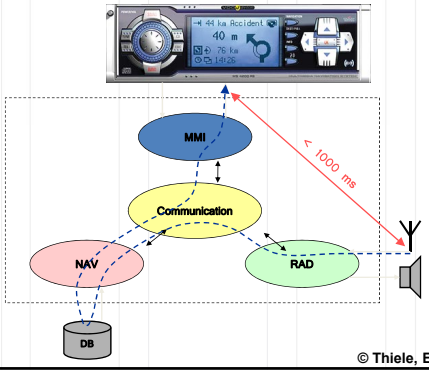
Use case 2: Lookup Destination Address



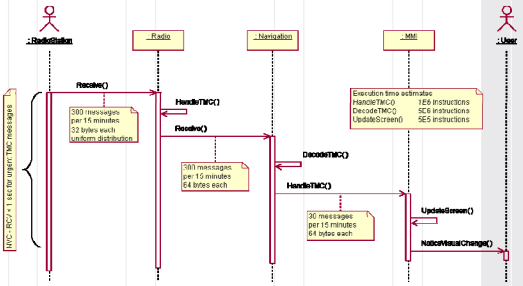
Use case 2: Lookup Destination Address



Use case 3: Receive TMC Messages



Use case 3: Receive TMC Messages



© Thiele, ETHZ

(Message) Sequence Charts

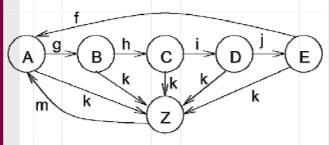
- PROs:
  - Appropriate for visualizing schedules,
  - Proven method for representing schedules in transportation.
  - Standard defined: ITU-TS Recommendation Z.120: Message Sequence Chart (MSC), ITU-TS, Geneva, 1996.
  - Semantics also defined: ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)—Annex B: Algebraic Semantics of Message Sequence Charts, ITU-TS, Geneva.
- CONS:
  - describes just one case, no timing tolerances: "What does an MSC specification mean: does it describe all behaviors of a system, or does it describe a set of sample behaviors of a system?"

Olekudiagrammid (StateCharts)

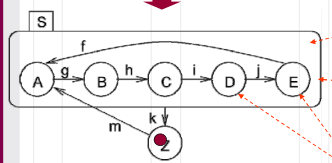
Olekudiagrammid

- Arvutusmudel, mis põhineb jagatud mäluga kommunikatsioonil
- Sobib ainult kohtrakendustele (mitte hajussüsteemidele)
- Klassikaline automaat ei ole sobiv keerukate süsteemide kirjeldamiseks (keerukaid graafe ei ole võimalik inimestel mõista)
- Hierarhia sisse toomine ☞ StateCharts [Harel, 1987]

Hierarhia



FSM on täpselt ühes S'i oleks kui S on aktiivne (kas A või B või ...)

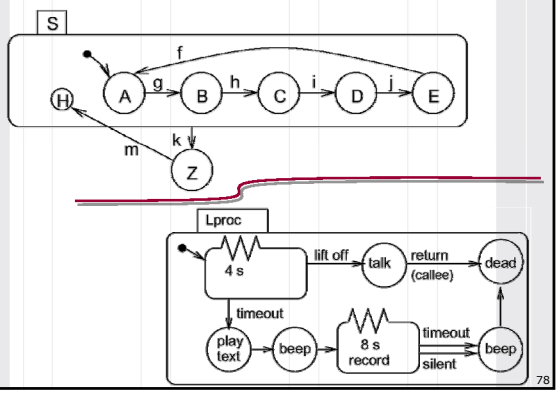


Superstate

Oleku E eellane (ancestor)

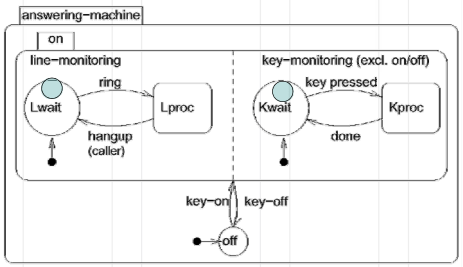
Alamolekud (substates)

Ajalugu, vaikimisi olek, timerid



Kattuvus

– AND-superstate



Hinnang StateChart'ile

– Pros:
  - Hierarhia lubab suvalist komplekti AND- ja OR-superstate'e.
  - Mitmed kommertstarkvarapaketid (StateMate, StateFlow, BetterState, ...)
  - On olemas „back-end" tarkvara StateChart'ide transleerimiseks C-sse või VHDLi, võimaldades sedasi tarkvaralisi ja riistvaralisi lahendusi.

Hinnang StateChart'ile (2)

– Cons:
  - Genereeritud C programmid ei ole alati efektiivsed
  - Ei sobi hajusrakendustele
  - Ei ole programmilisi konstruktsioone
  - Ei võimalda kirjeldada mitte-funktsionaalset käitumist
  - Ei ole objekt orienteeritud
  - Ei võimalda haarata struktuurset hierarhiat

Laiendused:
- **Module charts struktuurse hierarhia kirjelduseks**

Küsimusi?

Gert Jervan
ati.ttu.ee/~gerje