

TDDD07 Real-time Systems

Lecture 9: Dependability & Design

Simin Nadjm-Tehrani

Real-time Systems Laboratory
Department of Computer and Information Science
Linköping University



Two lectures

This lecture and part of lecture 9:

- Basic notions of dependability and redundancy in fault-tolerant systems
- Fault tolerance:
 - Relating faults/redundancy to distributed systems from lectures 4-6
 - Relating timing and fault tolerance

Lecture 9:

- Fault prevention and design aspects

Treatment of faults

- Last lecture: We mentioned four approaches for treating faults in dependable systems

- This lecture:

1. Fault prevention
2. Fault removal
3. Fault tolerance
4. Fault forecasting

Reading: Section 5.1 & 5.3 of article by Avezienis et al.

Also article on platform-independent design
Huang et al.

System requirements

- Functional requirements
 - Describe the main objectives of the system, referred to as “correct service” earlier
- Extra-functional requirements
 - Also called non-functional properties (NFP)
 - Cover other requirements than those relating to main function, in particular dependability attributes: the frequency and severity of acceptable service failures
- Example NFP
 - Timeliness, availability, energy efficiency

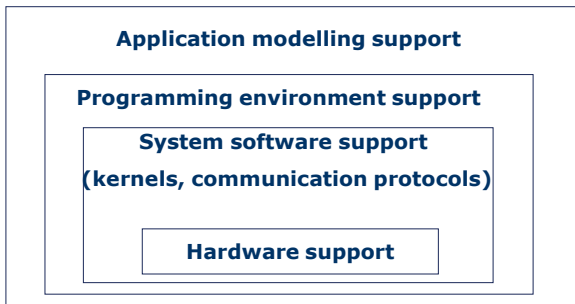
Basic approach

Design for timeliness:

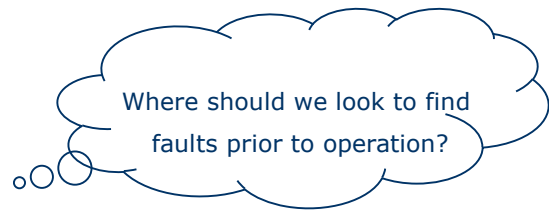
- define end-to-end deadlines
- define deadlines for individual tasks
- ascertain (worst case) execution/communication time for each task/message
- document assumptions/restrictions
- Prove/show that implementation satisfies requirements

So, what is so hard about this?

Layers of design



Fault prevention/removal



Historical snapshots

- Hardware design
 - 1970 's Dedicated hardware
 - 1980 's Micro computers & ASICS
 - 1990 's High performance Micro computers, FPGAs, MEMs
 - 2000 's SoCware
- Earlier predictable hardware is replaced with components that are complex to analyse (including cache, pipeline)

Layers of design



Historical snapshots

- Scheduling principles
 - 1970 's Fixed priority scheduling
 - 1980 's Multiprocessor, Dynamic
 - 1990 's Incorporating shared resources
 - 2000 's Load variations, adaptation
Multicore scheduling for real
- OS interfaces to optimise memory management, prefetching instructions to boost performance

Layers of design



Historical snapshots

- Programming environments
 - 1970 's "High" level programming
 - 1980 's Real-time specific: Ada
 - 1990 's OO languages, languages with formal semantics
 - 2000 's Software libraries, reuse
- Industry lecture: Reactive/actor-based!

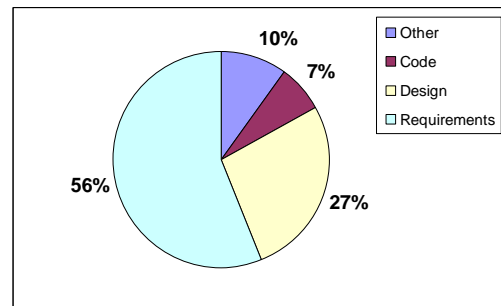
Engineers: Fool me once, shame on you – fool me twice, shame on me



Software developers: Fool me N times, who cares, this is complex and anyway no one expects software to work...



Frequency of faults



[Jim Cooling 2003, cited from DeMarco78]

Testing does not do ...

If a test fails, what was the cause?

- Undocumented assumptions on operational conditions, external impact?
- Wrong program code?
- Unexpected impact of OS functionality?
- Hardware timing dependencies?
- Embedded test code affecting timing?

Platform-independent design

Eliminating "butterfly effect" means trying to isolate the impacts of different layers

Back to basics

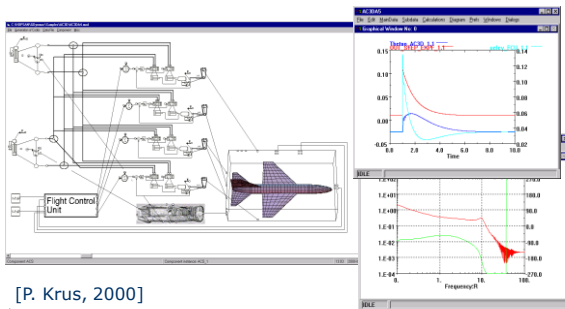
- define end-to-end deadlines
 - Model the environment!
- define deadlines for individual tasks
 - Specify system decomposition!
- ascertain (worst case) execution/communication time for each task/message
 - Assume hardware/bus characteristics!
- document assumptions/restrictions
 - Model, model, model!
- Prove/show that implementation satisfies requirements
 - Analyse models, then test implementation!

An engineering discipline

Using mathematics can never be wrong!

Non-digital hardware

Extensive simulations of coupled aircraft flight dynamics and actuator dynamics



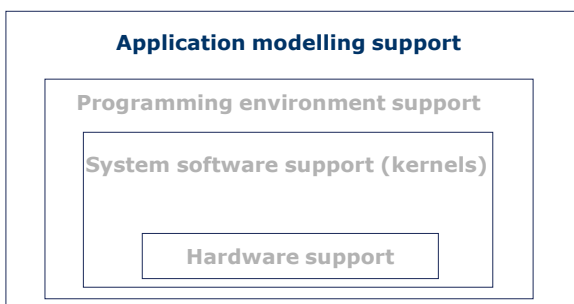
[P. Krus, 2000]

Model-based development

In software-intensive systems:

- Models as "higher level" programs
- Idea: use models to analyse the design, automatically generate code from the model!
- Adequate support for modularisation: Well-tested libraries with well-defined interfaces

Layers of design



Historical snapshots

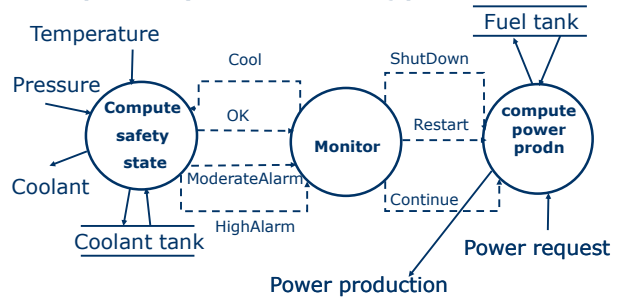
- Application modelling & analysis tools
 - 1970's Sequential systems
 - 1980's Concurrent/Distributed systems
 - 1990's Timed models, Combining discrete & continuous, UML
 - 2000's Incorporation in CASE tools
- 2012 crossroad: Domain-specific or Unified?

UML standard

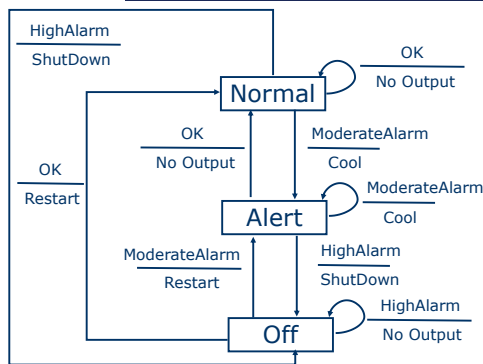
- UML 2.0 models components with required and provided interfaces
- Family of modelling techniques that are a further development of languages in early 80's, for example: Ward & Mellor Diagrams
- Next two slides from an example [Heitmeyer and Mandrioli, Wiley, 1996]

Power plant

- Transformation schemata for functional part & dynamic monitoring part



Monitor state machine



Advances in 2000's

- CASE tool design models for digital hardware and software components, and *functional analysis* by
 - Simulations
- and sometimes...
 - Formal verification of functional properties
 - Semi-automatic code generation

Simulations of a model

Need a unique interpretation:

- The language should be platform-independent
- The language should have an operational semantics to enable "execution" of the model

Simulations

What do they show?



Formal techniques (proofs)

- **Remove** (design) faults that lead to demonstrated bad things
 - debugging the design
- But also **Prove** that *specific* bad things *never* happen
- Can be automated, but suffer from combinatorial explosion

Advanced techniques

- Smart data structures for efficient representation of state space
- Smart deduction engines (satisfiability checkers) that find proofs fast
- Smart abstractions of the design to capture the essential properties
 - Synchronous languages (e.g. Esterel, Lustre), used for Airbus 320 software

Historical snapshots

- Application modelling & analysis tools
 - 1970's Sequential systems
 - 1980's Concurrent/Distributed systems
 - 1990's Timed models, Combining discrete & continuous, UML
 - 2000's Incorporation in CASE tools
- 2012 crossroad: Domain-specific or Unified?

Adding time to UML

- Has not been easy
- No industry-wide tool support
- Recent development: UML profile for Real-time and Embedded Systems (MARTE)
- Meta-models for a class of systems with timing and performance parameters

Reading: handout on
MARTE!