Abstract
The module Execution architecture approach and concepts addresses an incremental approach to design an execution architecture. A set of concepts is introduced and illustrated, which is useful in the hands on phase of the course.
An incremental execution architecture design approach

by Gerrit Muller    Buskerud University College
e-mail: gaudisite@gmail.com
www.gaudisite.nl

Abstract
An incremental design approach for the execution architecture is described. The method is based on identification of the most critical requirement from both user as well as technical point of view. The implementation itself is based on quantified budgets. The creation, modification and verification of the budget is discussed.
Positioning in CAFCR

What does Customer need in Product and Why?

Customer What
- Customer objectives

Customer How
- Application

Product What
- Functional

Product How
- Conceptual
- Realization

SMART
- timing requirements
- external interfaces

execution architecture design
- threads allocation
- interrupts scheduling
- timers synchronization
- queues decoupling

models analysis

simulations measurements

An incremental execution architecture design approach
3 Gerrit Muller

version: 1.0
July 31, 2014
EAAandCAFCR
Incremental approach

- measure
- evaluate
- analyse

- determine most important and critical requirements

- simulate
- build proto

- model
- analyse constraints and design options
Decomposition of system TR in HW and SW

- most and hardest TR handled by HW
- new control TRs

An incremental execution architecture design approach

Gerrit Muller

version: 1.0
July 31, 2014
EAAhwswRequirements
Quantification steps

- back of the envelope
- benchmark, spreadsheet calculation
- measure, analyze, simulate
- cycle accurate

order of magnitude

guestimates

calibrated estimates

feasibility

99.999 to 100.001 cycle accurate
Budget based design

Can be more complex than additions

model

measurements
existing system

design
estimates;
simulations

budget

spec

measurements
new (proto)
system

SRS
\[ t_{\text{boot}} = 0.5\text{s} \]
\[ t_{\text{tap}} = 0.2\text{s} \]

feedback

form

V4aa

micro benchmarks
aggregated functions
applications
profiles
traces

can be more complex than additions

An incremental execution architecture design approach

version: 1.0
July 31, 2014
EAAbudget
Abstract
The execution architecture determines largely the realtime and performance behavior of a system. Hard real time is characterized as ”missing a deadline” will result in system failure, while soft real time will result ”only” in dissatisfaction.

An incremental design approach is described. Concepts such as latency, response time and throughput are illustrated. Design considerations and recommendations are given such as separation of concerns, understandability and granularity. The use of budgets for design and feedback is discussed.
Execution architecture concepts

9 Gerrit Muller

version: 1.1
July 31, 2014
CVexecutionArchitecture
Fuzzy customer view on real time

- **hard real time**
  - disastrous failure
  - human safety
  - device safety
  - loss of information

- **soft real time**
  - dissatisfaction
  - limited throughput
  - waiting time
  - loss of functionality
  - loss of eye hand coordination
Smartening requirements

Limited set of hard real time cases

Precise form of the distribution is not important.

Be aware of systematic effects

No exception allowed
Worst case must fit

Well defined set of performance critical cases

Typical within desired time, limited exceptions allowed.

Exceptions may not result in functional failure
Latency

Execution architecture concepts
12  Gerrit Muller
Response Time

Response Time

- Total response time
- Zap repetition
- Visual feedback
- Open for next response
- New channel

Visual feedback time

Execution architecture concepts
13  Gerrit Muller
throughput:
+ processing steps/frame
+ frames/second
+ concurrent streams
Gross versus Nett

bus bandwidth, processor load [memory usage]
useful macroscopic views, be aware of microscopic behavior

<table>
<thead>
<tr>
<th>margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>loss = not schedulable</td>
</tr>
<tr>
<td>overhead</td>
</tr>
<tr>
<td>bus, OS, scheduling</td>
</tr>
</tbody>
</table>

depends on design
depends strongly on granularity

application overhead is still in this "nett" number

function 1
function 2
function 3
function 4
Design recommendations separation of concerns

- decoupling
- queues or buffers
- minimal shared resources
- process as unit of execution
- clear single demarcation between hard and soft
- minimize influence
- performance
- separation
- cost
- manage tension explicit
- HW
- HW
- HW
- HW
- soft Real Time
- hard Real Time

Execution architecture concepts
Gerrit Muller

version: 1.1
July 31, 2014
Design recommendations understandability

- Complex reality; many details, many relations
- Limited use of tasks, threads, priorities
- To combine or not to combine?
- Reasoning must be possible
- Overview is based on understanding many (critical) details
- Hard real time systems should be explainable with a few A4 diagrams
- Simulation: additional means if declared indispensable this is often a symptom of poor models

Simple is better
Granularity considerations

<table>
<thead>
<tr>
<th>unit of buffering</th>
<th>== or &lt; &gt;</th>
<th>unit of synchronization</th>
<th>== or &lt; &gt;</th>
<th>unit of processing</th>
<th>== or &lt; &gt;</th>
<th>unit of I/O</th>
</tr>
</thead>
</table>

- video frame
- video line
- pixel

**fine grain:**
- flexible
- high overhead

**coarse grain:**
- rigid
- low overhead
Design patterns

synchronous
- safety critical, reliable, subsystems
- very low overhead
- predictable
- understandable
- works best in total separation
- does not work for multiple rhythms

thread based
- Asynchronous applications and services
- separation of timing concerns
- sharing of resources (no wait)
- poor understanding of concurrency
- danger of high overhead

timer based
- regular rhythm;
  - e.g. monitor HW status, update time, status display
- low "tunable" overhead
- understandable
- fast rhythms significant overhead

interrupt based
- I/O and HW events
  - data available, display frame sync
- separation of timing concerns
- definition of interrupts determines:
  - overhead, understandability
Synchronous design

Execution architecture concepts

version: 1.1
July 31, 2014
EACsynchronousDesign
Actual timing on logarithmic scale

- Application needs
- Light travels 1 cm: $10^{-12}$ ps
- $10^{-9}$ ns
- $10^{-6}$ s
- $10^{-3}$ ms
- 1 s

- 2 GHz CPU cycle
- DRAM cycle time
- DRAM latency
- 1 byte transfer fast ethernet
- 1 package transfer fast ethernet
- Disk seek
- Appl level network message exchange
- Appl level function response
- 100 Hz TV frame
- Human eye
- Human reaction time
- Human 1st irritation threshold
- Human 2nd irritation threshold
- From low level to high level processing times
Typical micro benchmarks for timing aspects

<table>
<thead>
<tr>
<th></th>
<th>Infrequent operations, often time-intensive</th>
<th>Often repeated operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>database</strong></td>
<td>start session, finish session</td>
<td>perform transaction query</td>
</tr>
<tr>
<td><strong>network, I/O</strong></td>
<td>open connection, close connection</td>
<td>transfer data</td>
</tr>
<tr>
<td><strong>high level</strong></td>
<td>component creation, component destruction</td>
<td>method invocation</td>
</tr>
<tr>
<td><strong>construction</strong></td>
<td></td>
<td>same scope</td>
</tr>
<tr>
<td><strong>low level</strong></td>
<td>object creation, object destruction</td>
<td>method invocation</td>
</tr>
<tr>
<td><strong>construction</strong></td>
<td></td>
<td>other context</td>
</tr>
<tr>
<td><strong>basic</strong></td>
<td>memory allocation, memory free</td>
<td>function call</td>
</tr>
<tr>
<td><strong>programming</strong></td>
<td></td>
<td>loop overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic operations (add, mul, load, store)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>task, thread creation</td>
<td>task switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupt response</td>
</tr>
<tr>
<td><strong>HW</strong></td>
<td>power up, power down boot</td>
<td>cache flush</td>
</tr>
<tr>
<td></td>
<td></td>
<td>low level data transfer</td>
</tr>
</tbody>
</table>

Execution architecture concepts

version: 1.1
July 31, 2014
RVuTimingBenchmarks
The transfer time as function of blocksize

- worst case
- optimal block-size
- $t_{\text{overhead}}$
- $\text{rate}^{-1}$
## Example of a memory budget

<table>
<thead>
<tr>
<th></th>
<th>memory budget in Mbytes</th>
<th>code</th>
<th>obj data</th>
<th>bulk data</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared code</td>
<td></td>
<td>11.0</td>
<td></td>
<td></td>
<td>11.0</td>
</tr>
<tr>
<td>User Interface process</td>
<td></td>
<td>0.3</td>
<td>3.0</td>
<td>12.0</td>
<td>15.3</td>
</tr>
<tr>
<td>database server</td>
<td></td>
<td>0.3</td>
<td>3.2</td>
<td>3.0</td>
<td>6.5</td>
</tr>
<tr>
<td>print server</td>
<td></td>
<td>0.3</td>
<td>1.2</td>
<td>9.0</td>
<td>10.5</td>
</tr>
<tr>
<td>optical storage server</td>
<td></td>
<td>0.3</td>
<td>2.0</td>
<td>1.0</td>
<td>3.3</td>
</tr>
<tr>
<td>communication server</td>
<td></td>
<td>0.3</td>
<td>2.0</td>
<td>4.0</td>
<td>6.3</td>
</tr>
<tr>
<td>UNIX commands</td>
<td></td>
<td>0.3</td>
<td>0.2</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td>compute server</td>
<td></td>
<td>0.3</td>
<td>0.5</td>
<td>6.0</td>
<td>6.8</td>
</tr>
<tr>
<td>system monitor</td>
<td></td>
<td>0.3</td>
<td>0.5</td>
<td>0.0</td>
<td>0.8</td>
</tr>
<tr>
<td>application SW total</td>
<td></td>
<td>13.4</td>
<td>12.6</td>
<td>35.0</td>
<td>61.0</td>
</tr>
<tr>
<td>UNIX Solaris 2.x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10.0</td>
</tr>
<tr>
<td>file cache</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.0</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>74.0</td>
</tr>
</tbody>
</table>
## Complicating factors and measures

### Complications
- cache
- bus allocation
- memory management
- garbage collection
- memory (buffer, storage) fragmentation
- non preemptable OS activities
- "hidden" dependencies (i.e., deadlocks)
- systematic "coincidences", avalanche triggers
- instable response, performance

### Measures
- considered margin
- explicit behavior
- architecture rules
- monitoring, logging
- pool management
- feedback to architect
- flipover simulation