Abstract
The course Execution Architecture is a joint effort of Ton Kostelijk and Gerrit Muller. Only limited theory is given, most time is spent hands-on. Not all the material is in this bundle, the material made by Ton Kostelijk is in a separate handout.

Distribution
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.
Abstract
The course execution architecture is a joint effort of Ton Kostelijk and Gerrit Muller. The intention of the course is to help the participants in the practical aspects of designing an execution architecture. Most time during the course is spent in the normal development environment in exploring, measuring and modifying the current design. In the course setting the results are evaluated and next steps are planned. The amount of theory in the course itself is very limited, plenty of theoretical courses exist already.
Course Execution Architecture

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

Abstract

The course Execution Architecture (EA) is described. The program existing of
2 modules and 3 feedback and plan sessions is described. The course format,
based mostly on hands on work in real products being created, is explained.

The course execution architecture is a joint effort of Ton Kostelijck and Gerrit Muller.
Course Program

context theory plan

presentation discussion next step

presentation discussion next step

presentation discussion evaluation follow up

requirements design

design exploration micro measurements

analysis verification

design improvements measurements

Course Execution Architecture
Gerrit Muller
Rules of the Broadcast Part

• Please write your questions/remarks/statements on yellow stickers and attach them at the end on the P-flip.

  *These will be used in the interactive section for discussion and to increase insight.*

• Short clarification questions are welcome,

  *Discussion will take place in the interactive part.*

• Stupid questions don’t exist. Learning is based on **safe** and **open** interaction.

  *Very individual-oriented questions can be referred to a break or after the session.*
Rules of the Interactive and the Practice Part

- Your contribution is essential.

- Don’t monopolize the time. Everyone, also the quiet people, should have the opportunity to contribute.

  The facilitator will intervene if the contribution is limited to a small group of participants.

- Respect the contribution of others.

  Opinions can’t be wrong, difference of opinion is normal and called pluri-formity.

- The course format is highly experimental and based on improvisation, constructive proposals are welcome.

  It is your course! Regular evaluations will give the opportunity to influence the rest of the course.
Evaluation of the Expectations

Please write your name and expectations with a marker on one A4 page.

Describe your expectations as one-liner or in a few keywords.

These pages will be displayed on the wall of the room.

At the end of the course we will look back on these expectations, with the purpose of two-way learning.
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.
An incremental execution architecture design approach

What does Customer need in Product and Why?

Customer
What
C
ustomer objectives

Customer
How
A
pplication

Product
What
F
unctional

Product
How
C
ceptual
R
ealization

SMART
+ timing requirements
+ external interfaces

execution architecture design
threads
interrupts
timers
queues
allocation
scheduling
synchronization
decoupling

models analysis
models analysis

simulations measurements
simulations measurements
An incremental execution architecture design approach

11    Gerrit Muller

version: 1.0
July 31, 2014
EAAspiral

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
Quantification steps

- Order of magnitude
- Guestimates
- Calibrated estimates

- Feasibility measure, analyze, simulate
- Back of the envelope benchmark, spreadsheet calculation
- Cycle accurate

10  30  100  300

99.999 100.001

70  140

90  115

50  200
An incremental execution architecture design approach

measurements existing system

micro benchmarks
aggregated functions
applications

can be more complex than additions

t_{proc}

+ 

+ 

t_{over}

+ 

+ 

t_{disp}

model

V4aa

spec

SRS

t_{boot} 0.5s

t_{zap} 0.2s

design estimates; simulations

feedback

budget

New (proto) system

tuning

measurements

micro benchmarks
aggregated functions
applications
profiles
traces

existing system

Can be more complex than additions
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

Execution architecture concepts
16    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

- **soft real time**
  - dissatisfaction

- **soft real time**
  - waiting time
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
Response Time

- **P+**
- **P-**
- **remote control**
- **new channel**
- **zap**

**total response time**
- **zap repetition**
- **visual feedback**
- **open for next response**
- **new channel**

**visual feedback time**
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></th>
<th>margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>loss</td>
<td>not schedulable</td>
</tr>
<tr>
<td>overhead</td>
<td>bus, OS, scheduling</td>
</tr>
<tr>
<td>function 4</td>
<td></td>
</tr>
<tr>
<td>function 3</td>
<td></td>
</tr>
<tr>
<td>function 2</td>
<td></td>
</tr>
<tr>
<td>function 1</td>
<td></td>
</tr>
</tbody>
</table>

application overhead is still in this "nett" number

depends on design

depends strongly on granularity

version: 1.1
July 31, 2014
EACbrutoVsNetto
Design recommendations separation of concerns

- Soft Real Time
- Hard Real Time

**HW**

- Minimize influence
- Decoupling
- Minimal shared resources

- Queues or buffers
- Clear single demarcation between hard and soft
- Process as unit of execution

**Performance**

- Manage tension explicit
- Separation
- Cost

Execution architecture concepts

Gerrit Muller

Version: 1.1

July 31, 2014

EAC separation
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

Execution architecture concepts
24 Gerrit Muller

version: 1.1
July 31, 2014
EACunderstandability
### Granularity considerations

<table>
<thead>
<tr>
<th>Unit of Buffering</th>
<th>==</th>
<th>Unit of Synchronization</th>
<th>==</th>
<th>Unit of Processing</th>
<th>==</th>
<th>Unit of I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>or</td>
<td></td>
<td>or</td>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;</td>
<td></td>
<td>&lt;&gt;</td>
<td></td>
<td>&lt;&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- **Video frame**
- **Video line**
- **Pixel**

**Fine grain:**
- Flexible
- High overhead

**Coarse grain:**
- Rigid
- Low overhead
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

- Human 1 first irritation threshold
- Human 2 second irritation threshold
- Human reaction time
- Eye-hand co-ordination

- 100 Hz TV frame
- 100 Hz video line
- Human eye

- FO4 inverter delay
- Cycle 2 GHz CPU
- Light travels 1 cm
- Pure context switch
- Zero message transfer

- Package transfer fast ethernet
- Disk seek
- Message exchange
- Network exchange

- Application level function response
- Application level message exchange

- From low level to high level processing times

- DRAM cycle time
- DRAM latency
- 1 byte transfer fast ethernet

- From low to high level storage/network needs

- 1 cm light travels

<table>
<thead>
<tr>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps</td>
<td>$10^{-12}$</td>
</tr>
<tr>
<td>ns</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>s</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>ms</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
</tr>
</tbody>
</table>
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</td>
<td>perform transaction query</td>
</tr>
<tr>
<td></td>
<td>finish session</td>
<td></td>
</tr>
<tr>
<td><strong>network, I/O</strong></td>
<td>open connection</td>
<td>transfer data</td>
</tr>
<tr>
<td></td>
<td>close connection</td>
<td></td>
</tr>
<tr>
<td><strong>high level construction</strong></td>
<td>component creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>component destruction</td>
<td>same scope, other context</td>
</tr>
<tr>
<td><strong>low level construction</strong></td>
<td>object creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>object destruction</td>
<td></td>
</tr>
<tr>
<td><strong>basic programming</strong></td>
<td>memory allocation</td>
<td>function call</td>
</tr>
<tr>
<td></td>
<td>memory free</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>
The transfer time as function of blocksize

Execution architecture concepts
30    Gerrit Muller
**Example of a memory budget**

<table>
<thead>
<tr>
<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>11.0</td>
<td></td>
<td></td>
<td>11.0</td>
</tr>
<tr>
<td>User Interface process</td>
<td>0.3</td>
<td>3.0</td>
<td>12.0</td>
<td>15.3</td>
</tr>
<tr>
<td>database server</td>
<td>0.3</td>
<td>3.2</td>
<td>3.0</td>
<td>6.5</td>
</tr>
<tr>
<td>print server</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>0.3</td>
<td>2.0</td>
<td>1.0</td>
<td>3.3</td>
</tr>
<tr>
<td>communication server</td>
<td>0.3</td>
<td>2.0</td>
<td>4.0</td>
<td>6.3</td>
</tr>
<tr>
<td>UNIX commands</td>
<td>0.3</td>
<td>0.2</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>compute server</td>
<td>0.3</td>
<td>0.5</td>
<td>6.0</td>
<td>6.8</td>
</tr>
<tr>
<td>system monitor</td>
<td>0.3</td>
<td>0.5</td>
<td>0</td>
<td>0.8</td>
</tr>
</tbody>
</table>

| application SW total    | 13.4 | 12.6     | 35.0      | 61.0  |

| UNIX Solaris 2.x        |      |          |           | 10.0  |
| file cache              |      |          |           | 3.0   |

| total                   |      |          |           | 74.0  |
Complicating factors and measures

<table>
<thead>
<tr>
<th>complications</th>
<th>measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>considered margin</td>
</tr>
<tr>
<td>bus allocation</td>
<td>explicit behavior</td>
</tr>
<tr>
<td>memory management</td>
<td>architecture rules</td>
</tr>
<tr>
<td>garbage collection</td>
<td>monitoring, logging</td>
</tr>
<tr>
<td>memory (buffer, storage) fragmentation</td>
<td>pool management</td>
</tr>
<tr>
<td>non preemptable OS activities</td>
<td>feedback to architect</td>
</tr>
<tr>
<td>&quot;hidden&quot; dependencies (ie [dead]locks)</td>
<td>flipover simulation</td>
</tr>
<tr>
<td>systematic &quot;coincidences&quot;, avalanche triggers</td>
<td></td>
</tr>
<tr>
<td>instable response, performance</td>
<td></td>
</tr>
</tbody>
</table>

Execution architecture concepts
32    Gerrit Muller