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

- Functional model
  - receive
  - demux
  - display
  - process
  - store

- Hardware
  - CPU
  - DSP
  - RAM
  - tuner
  - drive

- Repository structure
  - Applications
    - play
    - zap
    - list
    - UI toolkit
    - menu
  - Processing
    - DCT
  - Foundation classes
    - queue
    - list
  - Hardware abstraction
    - tuner
    - DVD drive

- Execution architecture issues:
  - concurrency
  - scheduling
  - synchronisation
  - mutual exclusion
  - priorities
  - granularity

Other architecture views

Dead lines
Timing, throughput requirements

Map

Process
Task
Thread
Interrupt handlers
Fuzzy customer view on real time

hard real time ———— soft real time

disastrous failure dissatisfaction
human device safety safety
loss of functionality limited throughput waiting time
loss of information loss of eye hand coordination

Execution architecture concepts
3 Gerrit Muller
version: 1.1 September 9, 2018 EAChardVsSoft
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

connection latency

perceived delay

connection latency

speak
listen

speak
listen

reaction

reaction

bla bla bla

bla bla bla

long distance connection

Execution architecture concepts
5 Gerrit Muller

version: 1.1
September 9, 2018
EAClatency
Response Time

**Diagram:**
- New Channel: Redirection from remote control to a new channel.
- Zap: Instantaneous switch between channels.
- Visual Feedback Time: Time taken for visual feedback after a zap.
- Total Response Time: Cumulative time including zap repetition and visual feedback.

**Concepts:**
- Zap Repetition: Repeated actions of switching channels.
- Visual Feedback: Immediate visual confirmation of channel change.
- Time: Duration of response time.

**Execution Architecture Concepts:**
- Execution Architecture (EAC) discusses the response time in the context of remote control operations.
throughput:
+ processing steps/frame
+ frames/second
+ concurrent streams
bus bandwidth, processor load [memory usage]
useful macroscopic views, be aware of microscopic behavior

application overhead is still in this "nett" number

- function 1
- function 2
- function 3
- function 4

overhead
bus, OS, scheduling

loss = not schedulable

margin

depends on design

depends strongly on granularity
Design recommendations separation of concerns

- Minimize influence
- Decoupling
- Queues or buffers
- Clear single demarcation between hard and soft
- Process as unit of execution
- Separation
- Minimal shared resources
- Manage tension explicit
- Cost
- Performance
Design recommendations understandability

hard real time systems should be explainable with a few A4 diagrams

overview is based on understanding many (critical) details

complex reality; many details, many relations

limited use of tasks, threads, priorities

reasoning must be possible
to combine or not to combine?
simulation: additional means if declared indispensable this is often a symptom of poor models

simulation

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

Execution architecture concepts

version: 1.1
September 9, 2018
EAC granularity
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;
- Low "tunable" overhead
- Understandable
- Fast rhythms significant overhead

**Interrupt based**
- I/O and HW events
- Separation of timing concerns
- Definition of interrupts determines: overhead, understandability
Synchronous design

double buffer:
full decoupling of calculation and execution
Actual timing on logarithmic scale

- FO4 inverter delay
- Cycle 2 GHz CPU
- Pure context switch
- Zero message transfer
- Application level message exchange
- Application level function response
- Disk seek
- Application level network message exchange
- Application level function response
- 1 byte transfer (fast Ethernet)
- DRAM cycle time
- DRAM latency
- 1 package transfer (fast Ethernet)
- 100 Hz video pixel time
- 100 Hz video line
- 100 Hz TV frame
- Human eye
- Eye-hand co-ordination
- Human reaction time
- Human 1st irritation threshold
- Human 2nd irritation threshold
- Execution architecture concepts

- Light travels 1 cm
- Execution from low level to high level processing times

- Processing times from low to high level storage/network application needs

**Version:** 1.1

Gerrit Muller

September 9, 2018

RVTimeAxis
### Typical micro benchmarks for timing aspects

<table>
<thead>
<tr>
<th>Category</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 construction</strong></td>
<td>component creation, component destruction</td>
<td>method invocation</td>
</tr>
<tr>
<td><strong>Low level construction</strong></td>
<td>object creation, object destruction</td>
<td>method invocation same scope other context</td>
</tr>
<tr>
<td><strong>Basic programming</strong></td>
<td>memory allocation, memory free</td>
<td>function call loop overhead basic operations (add, mul, load, store)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>task, thread creation</td>
<td>task switch interrupt response</td>
</tr>
<tr>
<td><strong>HW</strong></td>
<td>power up, power down boot</td>
<td>cache flush low level data transfer</td>
</tr>
</tbody>
</table>
The transfer time as function of blocksize

- **time**
- **block size**
- **worst case**
- **optimal block-size**
- **t_overhead**
- **rate⁻¹**

Execution architecture concepts

Gerrit Muller

version: 1.1
September 9, 2018
RVparametrizedTransferRate
## 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>
<tr>
<td>application SW total</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>10.0</td>
</tr>
<tr>
<td>file cache</td>
<td></td>
<td></td>
<td></td>
<td>3.0</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
<td></td>
<td>74.0</td>
</tr>
</tbody>
</table>
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>