Abstract
This module addresses Modeling and Analysis: Inputs and Uncertainties. The input for models comes from different sources: facts obtained from market and technology research, data from measurements, and assumptions. All these sources have uncertainties and may hide unknowns, or may even be wrong. We zoom in on commonly used technology.
### Goal of this Module

Provide foundation and figures of merit for technology modeling

Provide insight in the inputs of models

Provide measurement fundamentals

### Content of this Module

Problem statement

generic layering and block diagrams

measuring HW and SW

### Exercise

Measurement of loop and file open performance

Participants may choose their own programming environment or Python
Where are we in the Course?

- facts from research
- measurements
- assumptions
- uncertainties
- unknowns
- errors
- modeling
- results
- analysis
- project
  - specification
  - verification
  - decisions
  - risk
  - customer satisfaction
  - time, cost, effort
  - profit margin

version: 0.6
July 31, 2014
MMAFTposition
Abstract
What is System Performance? Why should a software engineer have knowledge of the other parts of the system, such as the Hardware, the Operating System and the Middleware? The applications that he/she writes are self-contained, so how can other parts have any influence? This introduction sketches the problem and shows that at least a high level understanding of the system is very useful in order to get optimal performance.

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.

July 31, 2014
status: preliminary draft
version: 0.5
content of this presentation

Example of problem

Problem statements
Sample application code:

```java
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

alternative application code:

```
event 3*3 -> show screen 3*3
<screen 3*3>
    <row 1>
        <col 1><image 1,1></col 1>
        <col 2><image 1,2></col 2>
        <col 3><image 1,3></col 3>
    </row 1>
    <row 2>
        <col 1><image 1,1></col 1>
        <col 2><image 1,2></col 2>
        <col 3><image 1,3></col 3>
    </row 2>
    <row 3>
        <col 1><image 1,1></col 1>
        <col 2><image 1,2></col 2>
        <col 3><image 1,3></col 3>
    </row 3>
</screen 3*3>
```
Sample application code:

```java
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```
More Process Communication

What If....

Sample application code:

```c
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

UI process

screen server

database

screen
What If....

Sample application code:

```java
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

Attribute = 1 COM object
100 attributes / image
9 images = 900 COM objects
1 COM object = 80µs
9 images = 72 ms
I/O overhead

What If....

Sample application code:

```c
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

- I/O on line basis (512² image)

\[
9 \times 512 \times t_{I/O}
\]

\[
t_{I/O} \sim= 1\text{ms}
\]
Sample application code:

```c
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

can be:
- fast, but very local
- slow, but very generic
- slow, but very robust
- fast and robust
- ...

The emerging properties (behavior, performance) cannot be seen from the code itself!

Underlying platform and neighbouring functions determine emerging properties mostly.
Function in System Context

The performance and behavior of a function depend on realizations of used layers, functions in the same context, and the usage context.
Performance = Function (F&S, other F&S, MW, OS, HW)
MW, OS, HW >> 100 Manyear : very complex

Challenge: How to understand MW, OS, HW
with only a few parameters
Summary of Introduction to Problem

Resulting System Characteristics cannot be deduced from local code.

Underlying platform, neighboring applications and user context:

- have a big impact on system characteristics
- are big and complex

Models require decomposition, relations and representations to analyse.
Why do we model?

- what are indicators that modeling and analysis beyond "business as usual" architecture is needed.

- What questions trigger Modeling and Analysis.

The answer to the question from business side is not evident

The answer is business critical (e.g. poor performance -> unusable service). We did not discuss business value for this case.

Past experience shows that design choices have big impact on the outcome, in other words this part of the design is critical
Abstract
This presentation shows fundamental elements for models that are ICT-technology related. Basic hardware functions are discussed: storage, communication and computing with fundamental characteristics, such as throughput, latency, and capacity. A system is build by layers of software on top of hardware. The problem statement is how to reason about system properties, when the system consists of many layers of hardware and software.

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.

July 31, 2014
status: preliminary
draft
version: 0.5
content of this presentation

generic layering and block diagrams

typical characteristics and concerns

figures of merit

example of picture caching in web shop application
What do We Need to Analyze?

**relevant non functional requirements**
- latency: time from start to finish
- throughput: amount of information per time transferred or processed
- footprint (size): amount of data/code stored

**parameters in design space**
- network medium: (e.g. ethernet, ISDN)
- communication protocol: (e.g. HTTPS, TCP)
- message format: (e.g. XML)

**system design**
- working range
- dependencies
- realization variability
- scalability

**required analysis:**
How do parameters result in NFR's?
Typical Block Diagram and Typical Resources

- data base server
- web server
- screen
- client
- network
- presentation
- computation
- communication
- storage
# Hierarchy of Storage Technology

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
<th>Latency</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td>L1 cache, L2 cache, L3 cache</td>
<td>sub ns</td>
<td>n kB</td>
</tr>
<tr>
<td>Volatile</td>
<td>Main memory</td>
<td>tens ns</td>
<td>n GB</td>
</tr>
<tr>
<td>Persistent</td>
<td>Disks, disk arrays, disk farms</td>
<td>ms</td>
<td>n*100 GB</td>
</tr>
<tr>
<td>Archival</td>
<td>Robotized optical media, tape</td>
<td>&gt;s</td>
<td>n PB</td>
</tr>
</tbody>
</table>
Performance as Function of Data Set Size

![Graph showing performance as a function of data set size.

Legend:
- L1 cache
- L3 cache
- Main memory
- Hard disk
- Disk farm
- Robotized media

Y-axis: Random data processing performance in ops/s
X-axis: Data set size in bytes

Graph indicates the performance degradation across different data set sizes and storage locations.
### Communication Technology: Figures of Merit

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Frequency</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>on chip</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>connection</td>
<td>sub ns</td>
<td>n GHz</td>
<td>n mm</td>
</tr>
<tr>
<td>network</td>
<td>n ns</td>
<td>n GHz</td>
<td>n mm</td>
</tr>
<tr>
<td><strong>PCB level</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial I/O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tens ns</td>
<td>n 100MHz</td>
<td>n cm</td>
</tr>
<tr>
<td><strong>network</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAN</td>
<td>n ms</td>
<td>100MHz</td>
<td>n km</td>
</tr>
<tr>
<td>WAN</td>
<td>n 10ms</td>
<td>n GHz</td>
<td>global</td>
</tr>
</tbody>
</table>
Multiple Layers of Caching

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>Cache Miss Penalty</th>
<th>Cache Hit Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>application cache</td>
<td>1 s</td>
<td>10 ms</td>
</tr>
<tr>
<td>network layer cache</td>
<td>100 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>file cache</td>
<td>10 ms</td>
<td>10 us</td>
</tr>
<tr>
<td>virtual memory</td>
<td>1 ms</td>
<td>100 ns</td>
</tr>
<tr>
<td>memory caches L1, L2, L3</td>
<td>100 ns</td>
<td>1 ns</td>
</tr>
</tbody>
</table>
Why Caching?

- **project risk**
- **response time**
- **life cycle**
- **cost**

**Long latency**
- Mass storage
- Communication
- Overhead
- Resource intensive processing

**Low latency**
- Fast storage
- Local storage
- Larger chunks
- In (pre)processed format

**Limit storage needs to fit in fast local storage**

**Frequently used subset**

**Design parameters**
- Caching algorithm
- Storage location
- Cache size
- Chunk size
- Format

**Modeling and Analysis Fundamentals of Technology**
©2006, Embedded Systems Institute
©2006, Gerrit Muller
Example Web Shop

- **Client**
  - Browsing products
  - Ordering products
  - Paying
  - Tracking orders

- **Network**
  - Exhbiting products
  - Sales and order intake
  - Order handling
  - Stock handling
  - Financial bookkeeping

- **Enterprise**
  - Logistics
  - Finance
  - Product management
  - Customer management

- **Data Base Server**
  - Product descriptions
  - Logistics ERP
  - Financial
  - Customer relations

---

Modeling and Analysis Fundamentals of Technology
©2006, Embedded Systems Institute
©2006, Gerrit Muller

version: 0.5
July 31, 2014
MAFExampleWebShop
Impact of Picture Cache

- Fast response
- Less load
- Less server costs

Diagram:
- Client
- Screen
- Network
- Mid Office Server
- Back Office Server
- Product Descriptions
- Logistics ERP
- Financial
- Customer Relations
- Picture Cache
Risks of Caching

- frequently used subset
- fast storage
- local storage
- larger chunks
- in (pre)processed format

- robustness for application changes
- ability to benefit from technology improvements
- robustness for changing context (e.g. scalability)
- robustness for concurrent applications
- failure modes in exceptional user space

life cycle
- cost
- effort

project risk
- cost
- effort
- performance

Modeling and Analysis Fundamentals of Technology
©2006, Embedded Systems Institute
version: 0.5
July 31, 2014
MAFTrisksOfCaching
### Conclusions

Technology characteristics can be discontinuous

Caches are an example to work around discontinuities

Caches introduce complexity and decrease transparency

### Techniques, Models, Heuristics of this module

Generic block diagram: Presentation, Computation, Communication and Storage

Figures of merit

Local reasoning (e.g. cache example)
Abstract
This presentation addresses the fundamentals of measuring: What and how to measure, impact of context and experiment on measurement, measurement errors, validation of the result against expectations, and analysis of variation and credibility.

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.

July 31, 2014
status: preliminary
draft
version: 1.2
content

What and How to measure

Impact of experiment and context on measurement

Validation of results, a.o. by comparing with expectation

Consolidation of measurement data

Analysis of variation and analysis of credibility
### what

1. What do we need to know?
2. Define quantity to be measured.
3. Define required accuracy
   - initial model
4A. Define the measurement circumstances
   - fe.g. by use cases
4B. Determine expectation
   - historic data or estimation
4C. Define measurement set-up
5. Determine actual accuracy
   - uncertainties, measurement error
6. Start measuring
7. Perform sanity check
   - expectation versus actual outcome

### how
1. What do We Need? Example Context Switching

- **guidance** of concurrency design and task granularity

- **estimation** of total lost CPU time due to context switching

What:
context switch time of VxWorks running on ARM9

- VxWorks
  - operating system

- ARM 9
  - 200 MHz CPU
  - 100 MHz bus

(test program)
2. Define Quantity by Initial Model

What (original):
context switch time of VxWorks running on ARM9

What (more explicit):
The amount of lost CPU time, due to context switching on VxWorks running on ARM9 on a heavy loaded CPU

\[ t_{\text{context switch}} = t_{\text{scheduler}} + t_{p1, \text{loss}} \]

\( t_{p1, \text{no switching}} \)

Legend:
- Scheduler
- Process 1
- Process 2

\( t_{p1, \text{before}} \)
\( t_{\text{scheduler}} \)
\( t_{p2, \text{loss}} \)
\( t_{p2} \)
\( t_{\text{scheduler}} \)
\( t_{p1, \text{loss}} \)
\( t_{p1, \text{after}} \)

\emph{p2 pre-empts p1}
\emph{p1 resumes}
\emph{= lost CPU time}
3. Define Required Accuracy

- *guidance of concurrency design and task granularity*
- *estimation of total lost CPU time due to context switching*
- *number of context switches depends on application*
- *cost of context switch depends on OS and HW*

\[ \text{~10\%} \]

purpose drives required accuracy
Intermezzo: How to Measure CPU Time?

Low resolution ( ~ μs - ms)
Easy access
Lot of instrumentation

High resolution ( ~ 10 ns)
requires
HW instrumentation

Cope with limitations:
- Duration (16 / 32 bit counter)
- Requires Timer Access

OS

CPU

HW Timer

I/O

Logic analyzer / Oscilloscope

OS-Timer

Version: 1.2
July 31, 2014
PHRTmeasuringTime
4A. Define the Measurement Set-up

*Mimick relevant real world characteristics*

**real world**

- many concurrent processes, with
  - # instructions >> I-cache
  - # data >> D-cache

**experimental set-up**

- P1
- P2
- pre-empts
- cache flush
- no other CPU activities

- \( t_{p1, \text{before}} \)
- \( t_{\text{scheduler}} \)
- \( t_{p2, \text{loss}} \)
- \( t_{p2} \)
- \( t_{\text{scheduler}} \)
- \( t_{p1, \text{loss}} \)
- \( t_{p1, \text{after}} \)

- **\( p2 \) pre-empts \( p1 \)**
- **\( p1 \) resumes**
- **= lost CPU time**
4B. Case: ARM9 Hardware Block Diagram

- **CPU**
- **Instruction cache**
- **Data cache**
- **Memory**

**PCB**

**200 MHz**

- **CPU**
- **on-chip bus**
- **Instruction cache**
- **Data cache**
- **memory bus**

**100 MHz**

- **memory**

**cache line size:**

- 8 32-bit words
Key Hardware Performance Aspect

memory request

22 cycles

memory response

38 cycles

memory access time in case of a cache miss

200 Mhz, 5 ns cycle: 190 ns

©2006, Embedded Systems Institute

version: 1.2

July 31, 2014

©2006, Gerrit Muller

EBMImemoryTimingARM
Determine Expectation

simple SW model of context switch:
- save state P1
- determine next runnable task
- update scheduler administration
- load state P2
- run P2

Estimate how many instructions and memory accesses are needed per context switch

Calculate the estimated time needed per context switch

input data HW:
- $t_{\text{ARM instruction}} = 5 \text{ ns}$
- $t_{\text{memory access}} = 190 \text{ ns}$
Determine Expectation Quantified

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Memory Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td><strong>100</strong></td>
<td><strong>6</strong></td>
</tr>
</tbody>
</table>

input data HW:
- $t_{ARM\ instruction} = 5\ ns$
- $t_{memory\ access} = 190\ ns$

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Memory Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td><strong>6</strong></td>
</tr>
</tbody>
</table>

- simple SW model of context switch:
  - save state P1
  - determine next runnable task
  - update scheduler administration
  - load state P2
  - run P2

Estimate how many instructions and memory accesses are needed per context switch

Calculate the estimated time needed per context switch

round up (as margin) gives expected $t_{context\ switch} = 2\ \mu s$
4C. Code to Measure Context Switch

**Task 1**
- Time Stamp End
- Cache Flush
- Time Stamp Begin
- Context Switch

**Task 2**
- Time Stamp End
- Cache Flush
- Time Stamp Begin
- Context Switch

This code is used to measure the time spent on context switches between two tasks.
Measuring Task Switch Time
Understanding: Impact of Context Switch

Clock cycles Per Instruction (CPI)

Task 1

Task 2

Based on figure diagram by Ton Kostelijk

Modeling and Analysis: Measuring
©2006, Embedded Systems Institute

version: 1.2
July 31, 2014
PHRTarmCacheTaskSwitch

©2006, Gerrit Muller
5. Accuracy: Measurement Error

Measurements have stochastic variations and systematic deviations resulting in a range rather than a single value.
Accuracy 2: Be Aware of Error Propagation

\[ t_{\text{duration}} = t_{\text{end}} - t_{\text{start}} \]

\[ t_{\text{start}} = 10 \pm 2 \mu s \]

\[ t_{\text{end}} = 14 \pm 2 \mu s \]

\[ t_{\text{duration}} = 4 \pm \? \mu s \]

- **systematic errors**: add linear
- **stochastic errors**: add quadratic
Measurements have stochastic variations and systematic deviations resulting in a range rather than a single value.

The inputs of modeling, "facts", assumptions, and measurement results, also have stochastic variations and systematic deviations.

Stochastic variations and systematic deviations propagate (add, amplify or cancel) through the model resulting in an output range.
### 6. Actual ARM Figures

**ARM9 200 MHz**

$t_{\text{context switch}}$ as function of cache use

<table>
<thead>
<tr>
<th>cache setting</th>
<th>$t_{\text{context switch}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>From cache</td>
<td>2 µs</td>
</tr>
<tr>
<td>After cache flush</td>
<td>10 µs</td>
</tr>
<tr>
<td>Cache disabled</td>
<td>50 µs</td>
</tr>
</tbody>
</table>
### 7. Expectation versus Measurement

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Memory Accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td><strong>100</strong></td>
<td><strong>6</strong></td>
</tr>
</tbody>
</table>

**Input Data HW:**

- $t_{\text{ARM instruction}} = 5 \text{ ns}$
- $t_{\text{memory access}} = 190 \text{ ns}$

**Simple SW Model of Context Switch:**

1. Save state P1
2. Determine next runnable task
3. Update scheduler administration
4. Load state P2
5. Run P2

**Expected**

$t_{\text{context switch}} = 2 \mu s$

**Measured**

$t_{\text{context switch}} = 10 \mu s$

**How to explain?**

- Potentially missing in expectation:
  - Memory accesses due to instructions: ~10 instruction memory accesses ~ 2 µs
  - Memory management (MMU context)
  - Complex process model (parents, permissions)
  - Bookkeeping, e.g., performance data
  - Layering (function calls, stack handling)
  - The combination of above issues

**Potentially missing in expectation:**

- Memory accesses due to instructions
  - ~10 instruction memory accesses ~ 2 µs
- Memory management (MMU context)
- Complex process model (parents, permissions)
- Bookkeeping, e.g., performance data
- Layering (function calls, stack handling)
- The combination of above issues

**However, measurement seems to make sense**
## Conclusion Context Switch Overhead

The overhead time for context switches can be calculated using the formula:

\[
\text{t}_{\text{overhead}} = n_{\text{context switch}} \times t_{\text{context switch}}
\]

where:
- \( t_{\text{context switch}} \) = 10\( \mu \text{s} \)
- \( t_{\text{context switch}} \) = 2\( \mu \text{s} \)

<table>
<thead>
<tr>
<th>( n_{\text{context switch}} ) (s(^{-1}))</th>
<th>( t_{\text{overhead}} )</th>
<th>CPU load overhead</th>
<th>( t_{\text{overhead}} )</th>
<th>CPU load overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>5ms</td>
<td>0.5%</td>
<td>1ms</td>
<td>0.1%</td>
</tr>
<tr>
<td>5000</td>
<td>50ms</td>
<td>5%</td>
<td>10ms</td>
<td>1%</td>
</tr>
<tr>
<td>50000</td>
<td>500ms</td>
<td>50%</td>
<td>100ms</td>
<td>10%</td>
</tr>
</tbody>
</table>
goal of measurement

Guidance of concurrency design and task granularity

Estimation of context switching overhead

Cost of context switch on given platform

examples of measurement

Needed: context switch overhead ~10% accurate

Measurement instrumentation: HW pin and small SW test program

Simple models of HW and SW layers

Measurement results for context switching on ARM9
**Conclusions**

Measurements are an important source of factual data.

A measurement requires a well-designed experiment.

Measurement error, validation of the result determine the credibility.

Lots of consolidated data must be reduced to essential understanding.

**Techniques, Models, Heuristics of this module**

experimentation

error analysis

estimating expectations
This work is derived from the EXARCH course at CTT developed by Ton Kostelijk (Philips) and Gerrit Muller.

The Boderc project contributed to the measurement approach. Especially the work of Peter van den Bosch (Océ), Oana Florescu (TU/e), and Marcel Verhoef (Chess) has been valuable.
Abstract
A simple measurement exercise is described. Purpose of this exercise is to build up experience in measuring and its many pitfalls. The programming language Python is used as platform, because of its availability and low threshold for use.

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.
Select a programming environment, where loop overhead and file open can be measured in 30 minutes.

If this environment is not available, then use Python.
Active State Python (Freeware distribution, runs directly)
http://www.activestate.com/Products/ActivePython/

Python Language Website
http://www.python.org/

Python Reference Card
http://admin.oreillynet.com/python/excerpt/PythonPocketRef/examples/python.pdf
import time

for n in (1,10,100,1000,10000,100000,1000000):
    a = 0
    tstart = time.time()
    for i in xrange(n):
        a = a+1
    tend=time.time()

    print n, tend-tstart, (tend-tstart)/n

def example_filehandling():
    f = open("c:\temp\test.txt")
    for line in f.readlines():
        print line
    f.close()

tstart = time.time()
example_filehandling()
tend=time.time()
print "file open, read & print, close: ",tend-tstart,"s"
Exercise

• Perform the following measurements
  1. loop overhead
  2. file open

• Determine for every measurement:
  What is the expected result?
  What is the measurement error?
  What is the result?
  What is the credibility of the result?
  Explain the result.
  (optional) What is the variation? Explain the variation.
+ measuring is easy
+ measuring provides data and understanding

~ result and expectation often don't match

- sensible measuring is more difficult