Model-driven Quality and Resource Management for CPS

Marc Geilen, Twan Basten
in collaboration with Martijn Hendriks, Kees Goossens, and others from the FitOptiVis & TRANSACT teams

Electronic Systems, Dept. Electrical Engineering,
Eindhoven University of Technology

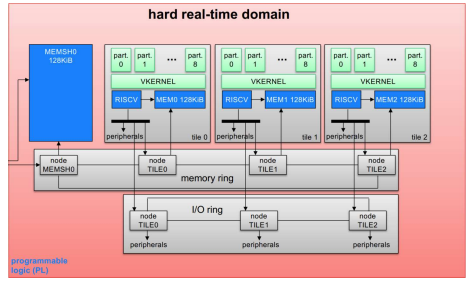
m.c.w.geilen@tue.nl

Overview

- Component Interface Model for QRM
  - a QRM use case
  - Budget models
- The Quality and Resource Modeling Language
  - language
  - Tools

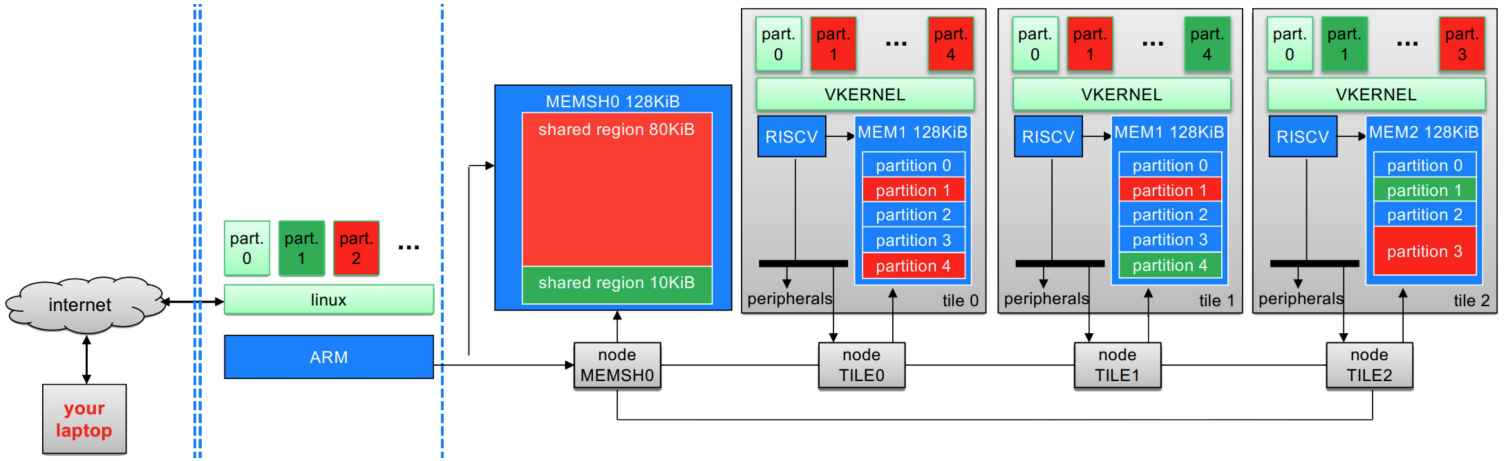
Use Case: Run-time Mapping on Predictable Platform

- consider a streaming application composed from a set of streaming components
- a compute platform supporting **virtual platforms**
  - a *set of reserved resources providing guaranteed resource budgets*
- find a suitable, minimal virtual platform that provides the resources required to satisfy the application performance requirements
- define appropriate **budget abstractions**



Virtual Execution Platform

- CompSoC Verintec platform
- composable / predictable
- strong virtualization

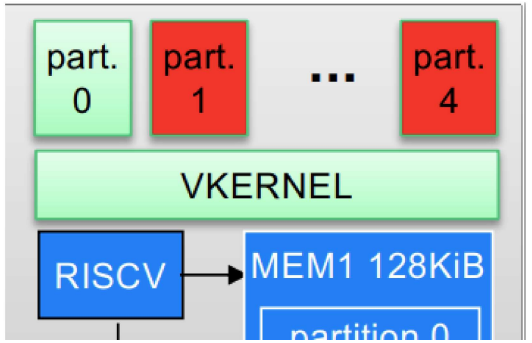


A Processing Budget

A **processing budget**

$$(C, I) \in \mathbb{N} \times \mathbb{R}$$

a task receives at least C processor MCycles in every interval of I milliseconds

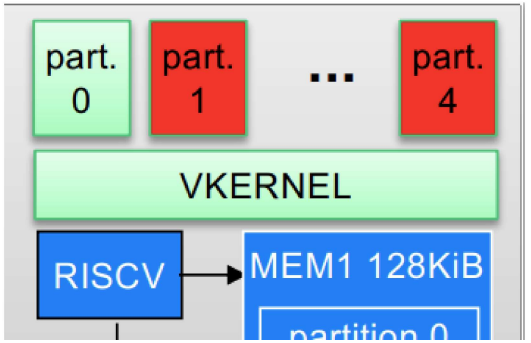


A Processing Budget

A **processing budget**

$$(C, I) \in \mathbb{N} \times \mathbb{R}$$

a task receives at least C processor MCycles in every interval of I milliseconds



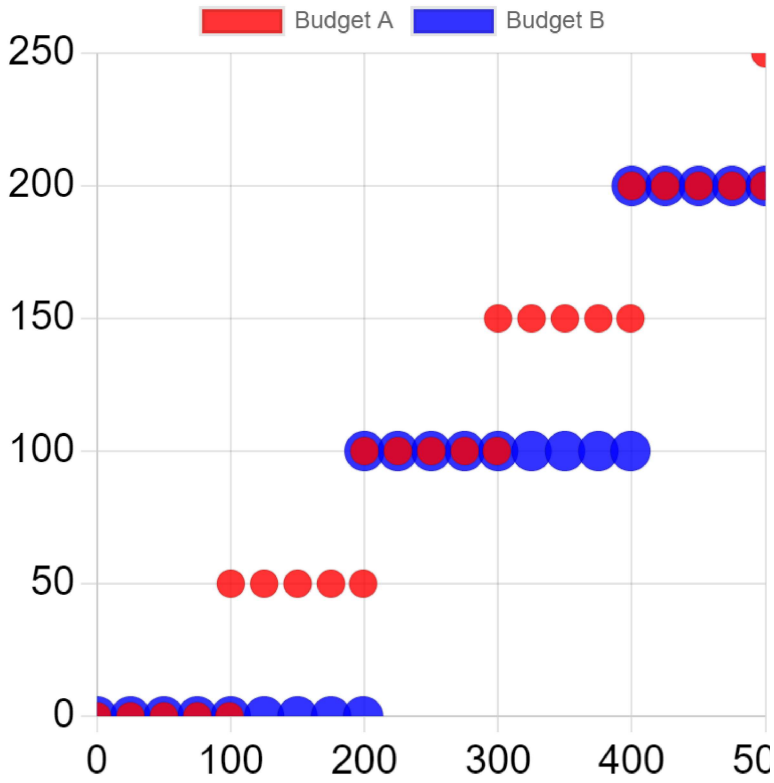
Is the following true? ($\preceq$ means "is at least as good")

$$(50, 100) \preceq (100, 200)?$$

A Processing Budget

$$A = (50, 100)$$

$$B = (100, 200)$$



A Processing Budget

$$A = (50, 100)$$

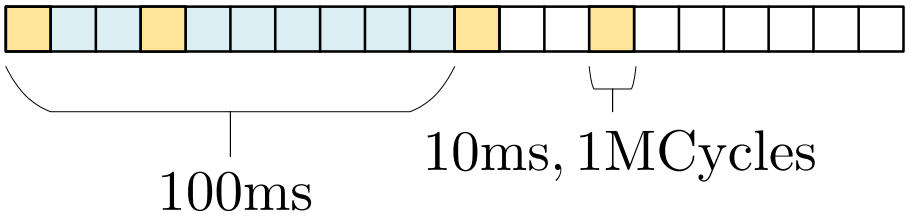
$$B = (100, 200)$$

$(C_1, I_1) \preceq (C_2, I_2)$ if and only if
$I_1 \leq I_2$ and $\left\lfloor \frac{I_2}{I_1} \right\rfloor \cdot C_1 \geq C_2$

Processing Budget Abstractions

A TDMA scheduler has

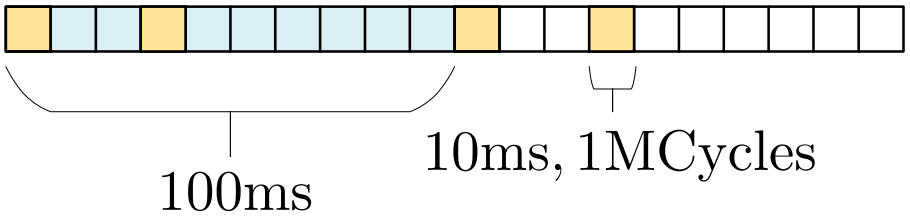
- a period of 100ms
- 10 slots of 1Mcycles each



Processing Budget Abstractions

A TDMA scheduler has

- a period of 100ms
- 10 slots of 1Mcycles each

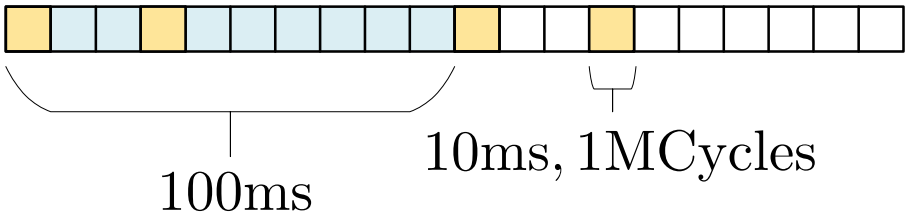


*What is the **strongest** budget that an allocation of slots 1 and 4 offers?*

Processing Budget Abstractions

A TDMA scheduler has

- a period of 100ms
- 10 slots of 1Mcycles each



*What is the **strongest** budget that an allocation of slots 1 and 4 offers?*

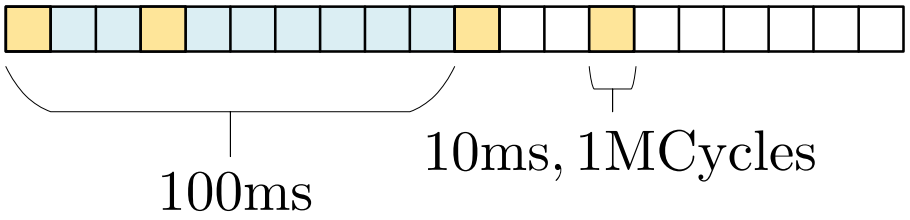
- **There is no single strongest!**

$$(1, 70), (2, 100), (0.001, 60.01), \ldots$$

Processing Budget Abstractions

A TDMA scheduler has

- a period of 100ms
- 10 slots of 1Mcycles each

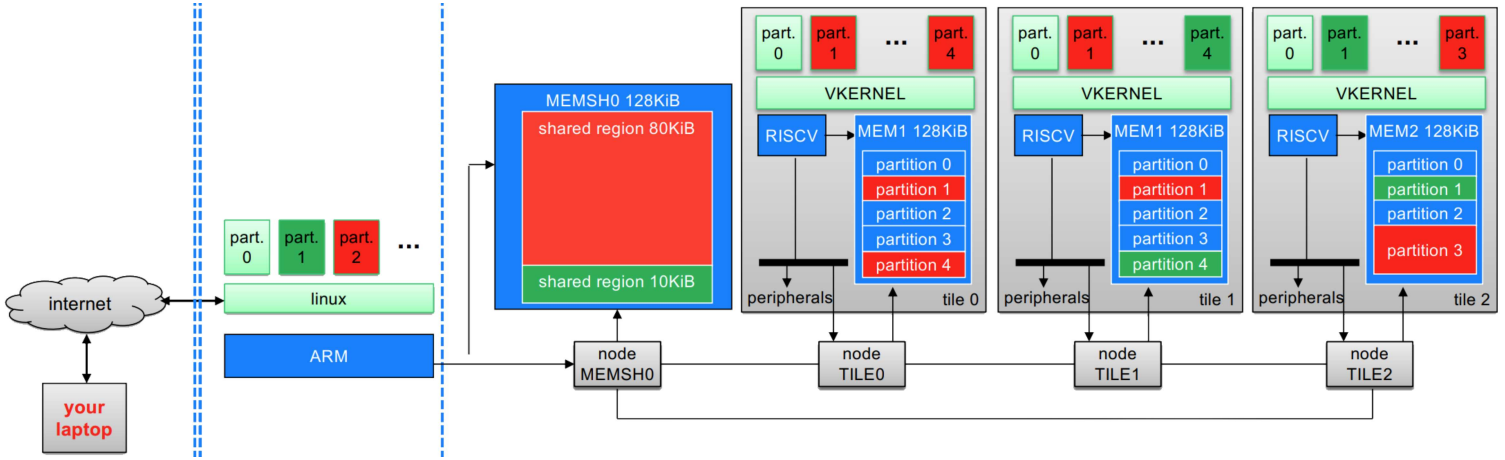


- **There is no single strongest!**

$$(1, 70), (2, 100), (0.001, 60.01), \ldots$$

- Pareto optimal **trade-offs** (between *throughput* and *latency*)
- budget abstractions may be defined and/or compared at different **levels of abstraction**

Virtual Execution Platform




```
"components":
[
  {
    "id" : "Task1",
    "configurations":
    [{
      "inputs":[{"raw_frames" : "30Hz"}, ...],
      "outputs":{"processed_frames" : "30Hz"}, ...],
      "parameters":[{"resolution" : "720p"}, ...],
      "qualities":[{"framerate" : 30}, ...],
      "required_budget":
       {
        "TILE": { "RISCV":
                    [{
                      "unit" : "cycles",
                      "type" : "average_rate",
                      "value" : 100K
                    },... // other services from RISCV
                   ], ... // other resources from TILE
                   }, ... // other resources besides TILE
       },
      "initial_state":[{"IDMEM" : ".../task1.hex"}, ...]
    }, ... // other application configurations
    ]}, ... // other components
],
"compositions":
[
  "App1 = Task1 => Task2",
  ...
]
```

Partially Ordered Qualities

All interface components (budgets, inputs/outputs, qualities) should, mathematically speaking, be **partial orders**

- can be arranged in terms of better / worse
- including trade-offs
- redundant solutions eliminated at design-time

A **partial order** relation is

- **reflexive**, equal properties always match
- **transitive**, if x is better than y, and y is better than z, then x is better than z too
- **anti-symmetric**, if different values are comparable, then one is the better one and the other is worse
- but different values need not be comparable

Types of Qualities

- non-numerical qualities
  - E.g., Boolean types, “provides redundancy”: $\{true, false\}$

$$true \preceq false$$

Types of Qualities

- non-numerical qualities
  - E.g., Boolean types, “provides redundancy”: $\{true, false\}$

$$true \preceq false$$

- Enumeration types (sets):
  - E.g., modulation scheme

$$\{16\text{QAM}, 64\text{QAM}, 256\text{QAM}\}$$

Types of Qualities

- non-numerical qualities
  - E.g., Boolean types, “provides redundancy”: $\{true, false\}$

$$true \preceq false$$

- Enumeration types (sets):
  - E.g., modulation scheme

$$\{16\text{QAM}, 64\text{QAM}, 256\text{QAM}\}$$

- Partially ordered qualities
  - allows $x \not\preceq y$ and $y \not\preceq x$ for distinct, ‘incomparable’ values x and y
  - allows one to require equality in producer-consumer constraints: $\preceq$ is equality

Partially ordered qualities

*can you define a partial order relation that **prioritizes** latency over power consumption?*

$$(l_1, p_1) \preceq (l_2, p_2) \quad \text{if and only if} \quad \ldots$$

Partially ordered qualities

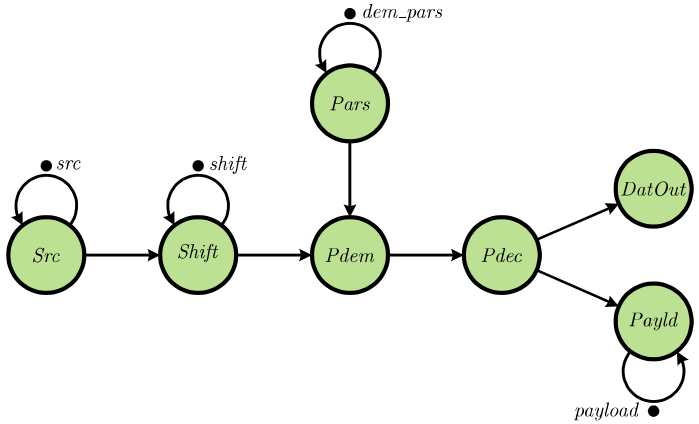
*can you define a partial order relation that **prioritizes** latency over power consumption?*

$$(l_1, p_1) \preceq (l_2, p_2) \text{ if and only if } \ldots$$

$$(l_1, p_1) \preceq (l_2, p_2) \text{ if and only if } l_1 < l_2 \text{ or } l_1 = l_2 \text{ and } p_1 \leq p_2$$

Component Abstractions

- **streaming applications** represented by **dataflow models**
- (worst-case) performance (throughput, latency) can be determined from a **compositional** max-plus algebraic model
- components can relate performance to processing budgets
- allows application components to be combined into an application



Set Points

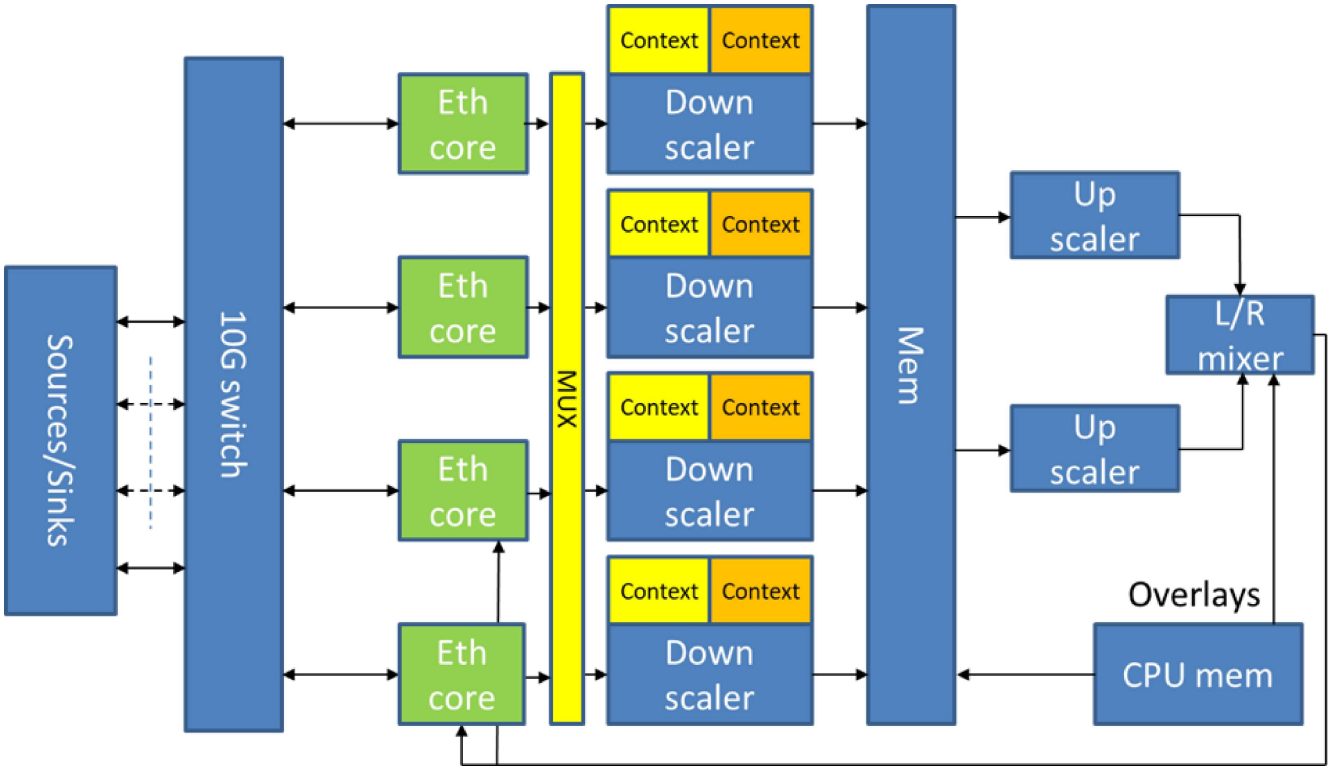
- set points of a **dataflow application component** consist of
  - binding of tasks to virtual processors (processing budgets)
  - static-order schedules of the tasks on the virtual processors
    - to make the performance predictable
- Use **monotonic optimization** techniques [1] to determine set points with **trade-off** between performance and required budget
- design-time component model
- run-time creation of a suitable virtual platform
- may be refined with DMAs, memory allocations, …

The QRML Language

[kar-uh-muh l] a kind of chewy candy, commonly in small blocks, made from sugar, butter, milk, etc.

The Quality and Resource Modelling Language

Examples from the Multi-Source Streaming Case



Defining Types

Examples

```
budget Bw integer

budget FrameRate integer

budget Computation integer

channel Video {
    hres: integer ordered by =
    vres: integer ordered by =
    rate: integer ordered by =
} ordered by a<=b if a.hres<=b.hres & a.vres<=b.vres & a.rate<=b.rate

budget Scaling {
    segs: integer
    comp: Computation
} ordered by element-wise

budget Scalers {
    streams: integer
    scaling: Scaling
}
```

Defining Types

- Non quantitative properties:

```
budget Services  subset of { S1, S2, S3 } ordered by subset
typedef Encryption boolean
channel Transport enumeration { ts, mkv, mp4 } unordered
```

- For composition budgets are added or subtracted.

```
budget Storage integer with addition a+b = a max b
```

Defining Partially Ordered Sets

How can we define a memory budget the requires a capacity and a type: EC / non-ECC?

Defining Partially Ordered Sets

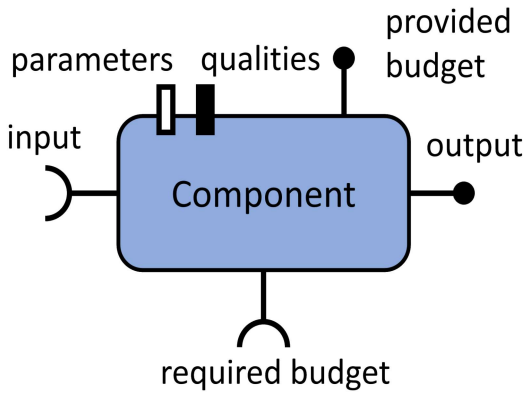
How can we define a memory budget the requires a capacity and a type: EC / non-ECC?

```
budget Memory {
    cap: integer
    ecc: boolean
}
```

Defining Components

```
component Fiber {
    provides Bw p_bw { p_bw = 10*1000 }
}

component FaceRecognition {
    input Image inp
    output Id out
    requires FaceIdentification faceId
  ...
}
```



- interfaces: **provides**, **requires**, **input**, **output**, **quality**, **parameter**

Hierarchical Components

```
component ExecutionPlatform {
    contains Fiber fiber
    contains HWscaler hwScaler

    provides Bw p_bw from fiber.p_bw
    provides Scalers p_sc from hwScaler.p_sc
}
```

- free composition

Alternative Choices

```
component HWScaler {
    provides Scalers p_sc { streams = 4; scaling.comp = 300; scaling.segs =
        32 }
}

component SWScaler {
    provides Computation p_cmp { p_cmp = 100 }
}

component HWorSWscaler {
    contains HWscaler hs or SWscaler ss

    provides Scalers p_sc

    constraint p_sc = hs.p_sc
    constraint p_sc = [top, ss.p_cmp]
}
```

- constraints are enforced if the component is selected
- **`top`** denotes a value that is better than any other value
- **`bottom`** denotes a value that is worse than any other value

Constraints

```
AC1.o output to AC2.i
A.r runs on R.p
...
```

- **horizontal composition** introduces a constraint between output o and input i using **Pareto dominance**
  - "*output o is 'at least as good' as input i*"

$$o \preceq i$$

- **vertical composition** constrains provided budget p and required budget r
  - "*provided budget p is 'at least as good' as required budget r*"

$$p \preceq r$$

Parameters

```
parameter Mode mode
requires Power power

constraint mode = m1 => power = 1 // W
constraint mode = m2 => power = 5 // W
```

- parameters are the controllable ‘knobs’ of components to select
  - feasible set points
  - optimal set points

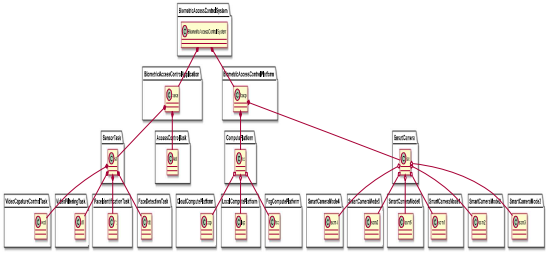
Objectives

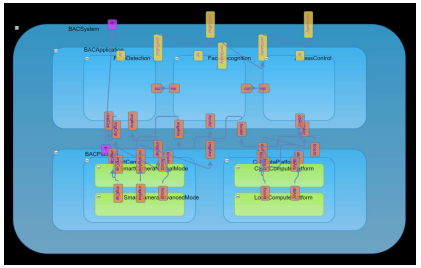
```
quality PowerConsumption pc
quality Torque t
```

- qualities identify the optimization objectives in a system
- determine optimal, feasible set points

Tools

- Xtext Domain-Specific Language tools
  - domain model, syntax checking, validation, code generation and transformations
- Visualization
  - PlantUML component diagrams
  - qrmlvis
- Web-based modeling environment
  - qrml.org
- Optimization
  - Conversion to constraint problem for the Z3 constraint solver





Transformation Engine

- **rewrite QRML model** with a rich, user-friendly syntax into QRML model using only small core of syntax and a simple structure
  - replace component classes by instances
  - unfold hierarchical types
  - rewrite complex ordering relations
  - …
- final model is essentially a set of variables and constraints
- convert to a **mathematical constraint problem**
- translate the result back into terms of the original model

Constraint Solving

- model represented as a constraint program
- SMT (*Satisfiability Modulo Theories*) solver can flexibly apply a set of ‘theories’ and a set of strategies to solve the problem
  - e.g., **Z3**
- no native support for Pareto fronts, but exploration can often be done by customized iteration of constraint programs

Conclusion

- Component Interface Model of QRM
  - builds on partial order relations for composition
  - expresses feasibility and multi-objective optimality
- QRML is a domain-specific language to build such models
  - some visualization options
  - translation to constraint program to check feasibility and find optimal set points
- we will build some models in the tutorial session after the coffee break
- please go to **`https://qrml.org`** → **`login`** → **`Sign Up`**
- use an email address that you have access to for verification
- use the registration code: **`mwX5e9#x`**