Towards a Component Architecture for Hard Real Time Control Applications

Wolfgang Pree

Josef Templ

CS, University of Salzburg, Austria

http://www.modecs.cc/

Motivation

- ECUs are becoming more powerful
- may execute multiple control applications in parallel
- automotive industry needs to save money
- high-end cars have up to 70 ECUs

ECU Consolidation

add control application(s) to an existing ECU *if possible*

possible if:

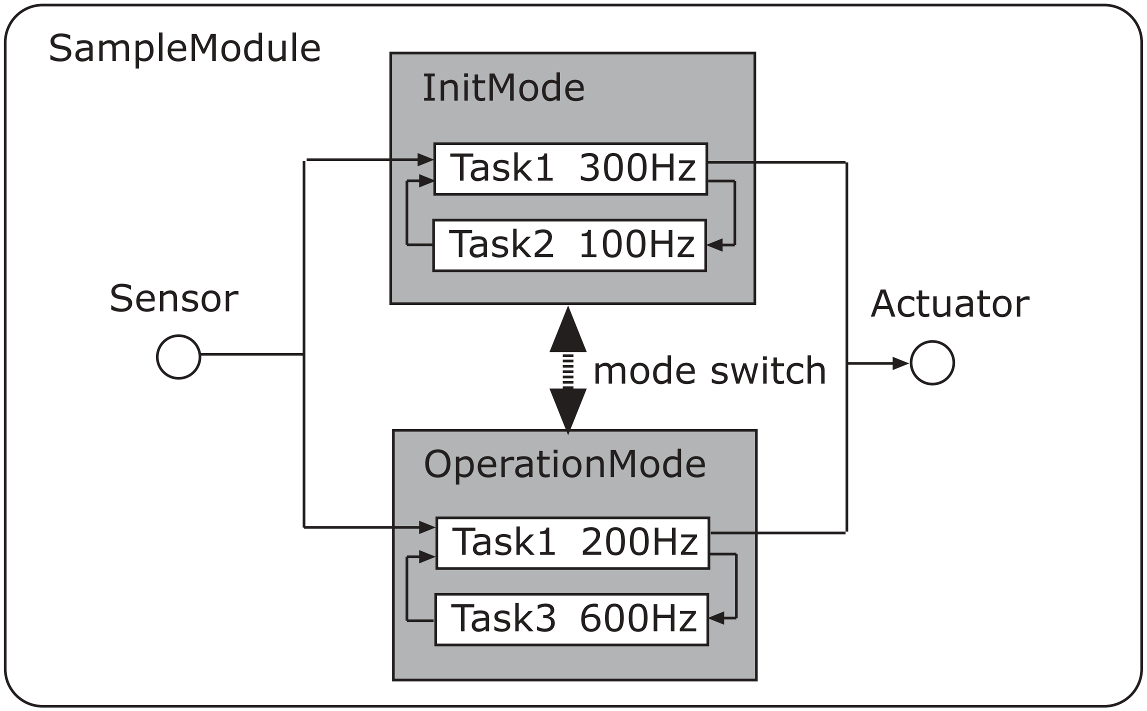
- **time safety** is guaranteed
- fault behavior is preserved (not dealt with here)

Key Ingredients of an Embedded Control Software Model

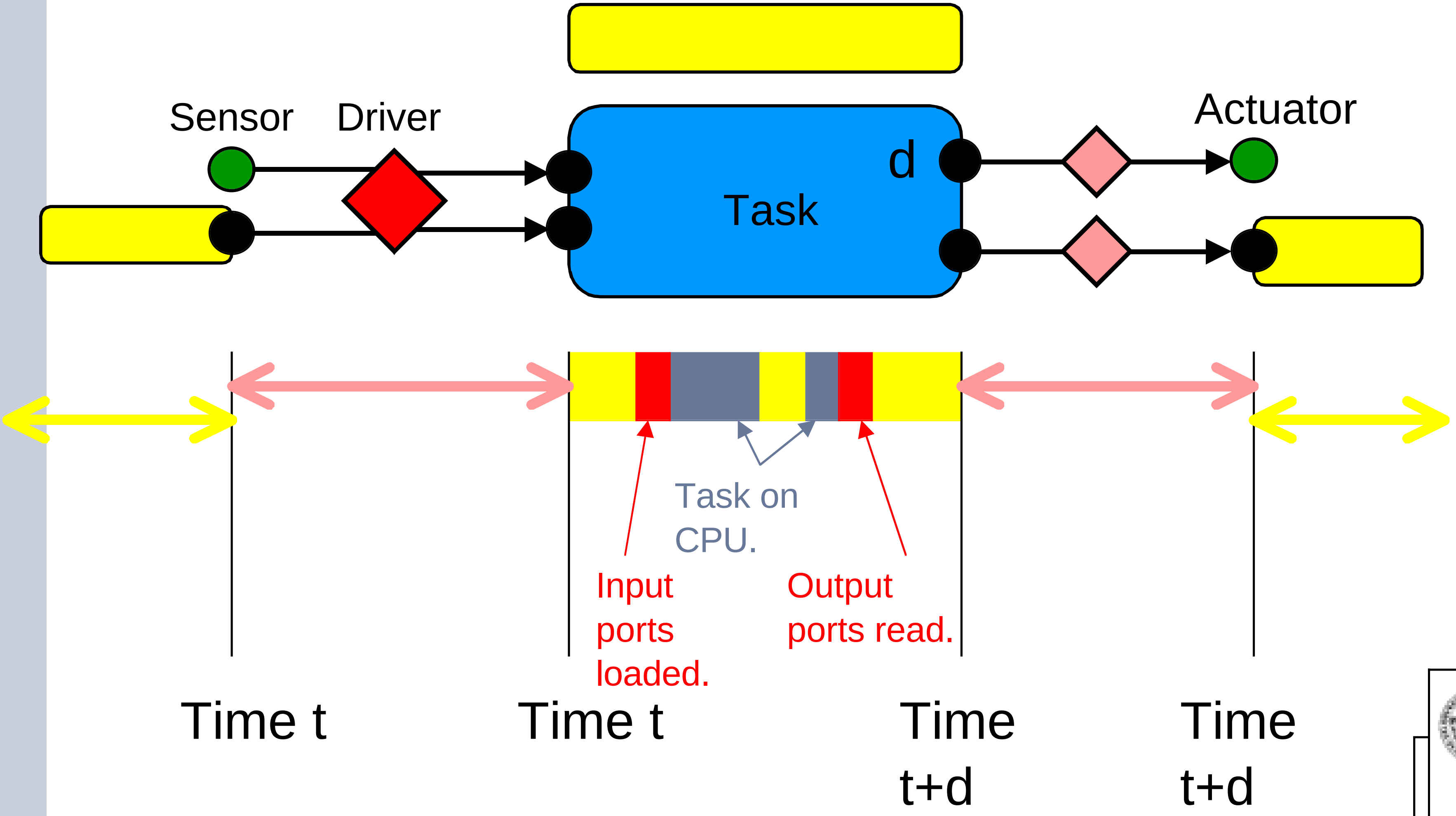
- platform-independence
- FLET assumption

based on *Giotto*

Platform-Independent Specification of Computation and Communication Activities



The Fixed Logical Execution Time (FLET) assumption: a precondition for RT composition



Introducing Modules

```
module EngineControl {
  //Giotto/TDL code consisting of sensor, actuator,
  //task and mode declarations
}
```

- named Giotto program
- provides name space
- loaded into E-machine
- may have a ‘start’ mode
- module = component

CPU Partitioning

- start mode is executed after loading of a module
- module needs CPU time
- executing module = CPU partition
- dynamic loading of (independent) modules = dynamic partitioning of an ECU
- in principle unloading is also possible upon request by the user

Module Import

```
module AdvancedCar{
  import EngineControl;
  import BrakeByWire;
  import ...;
  //Giotto/TDL code consisting of sensor, … declarations
  //May access public elements of imported modules
}
```

- import specifies static dependencies between modules
- allows to decompose large applications
- = static partitioning of an ECU

Information Hiding

```
module EngineController {
  public const maxRpm = 6500;
  //... more code
}
```

- sensors may be read by multiple modules
- actuator updates by multiple modules must be prevented
- TDL-rule: actuator update only in declaring module

=> modules partition the set of actuators

Mode Extension

```
module ExtendedEngineControl {
  import EngineControl;
  actuator int newActuator uses setNewActuator;
  task newTask ...; //provides output variable res
  mode normal extends EngineControl.normal {
    task [1] newTask(...);
    actuator [1] newActuator := newTask.res;
  }
}
```

- experimental feature
- allows *hot deployment* of new functionality

Scheduling Issues

- **global hyper period ‘hp’** = GCD of all activity periods of all modes of all partitions

  => activity periods should not be relative primes
- preemptive EDF scheduling per mode
- every partition gets a slot in hp
- slot size allocated for the most CPU intensive mode
- if all partitions execute most CPU intensive mode, CPU may be utilized up to 100%
- dynamic loading => dynamic scheduling + rescheduling if hp changes (background task)
- we experiment also with RM scheduling (OSEK)

Implementation Status

- TDL Compiler implemented in Java using Coco/R
- Java based E-machine with loading and executing multiple modules is running
- uses Java threads with suspend()/resume()
- not strictly real time
- alternative considered is ‘realtime’ Java
- in parallel work on C-based E-machine on top of OSEK, OSEK/Time etc.

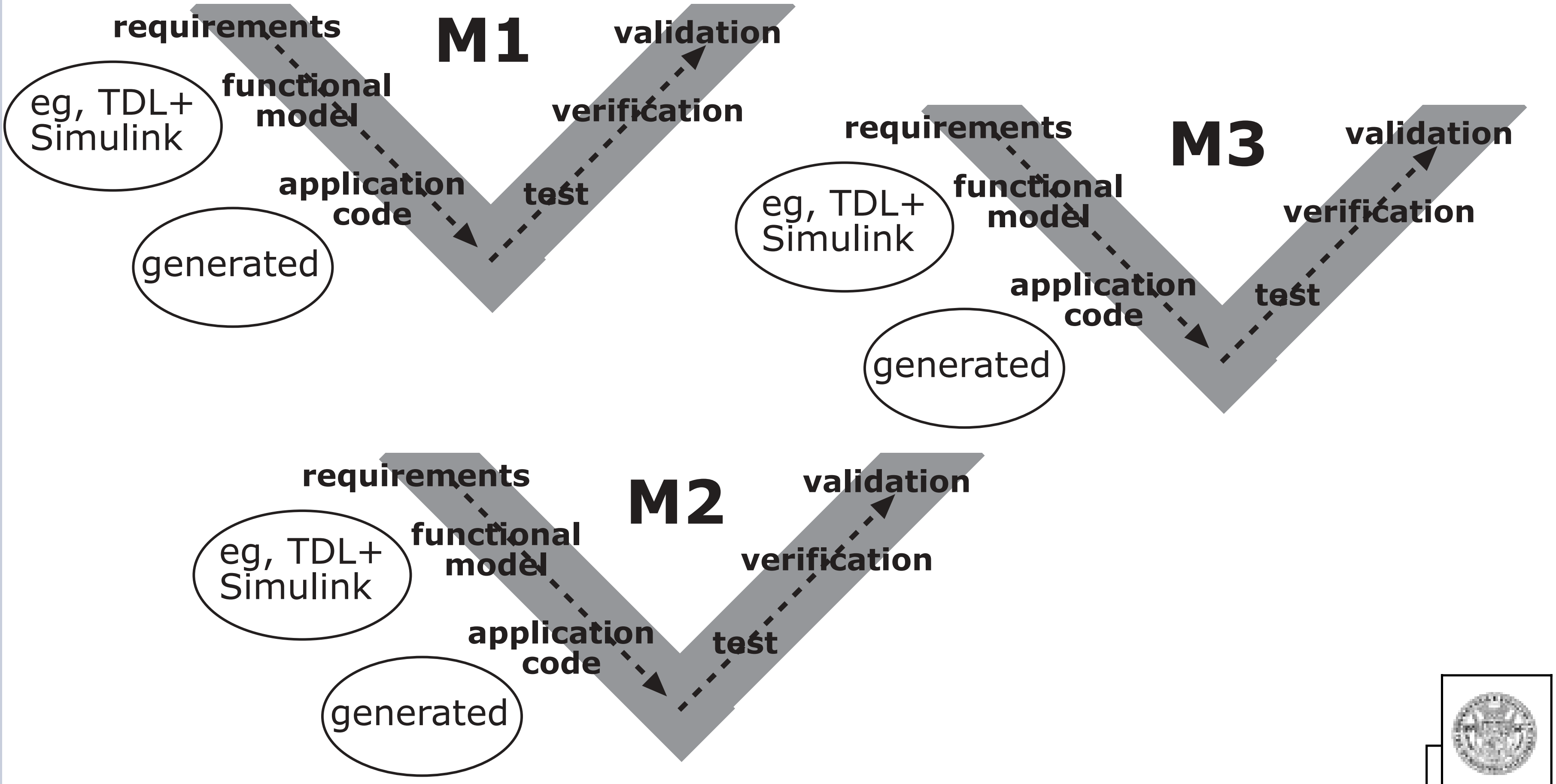
Outlook – Distribution + FT

- modules that work across a network of ECUs

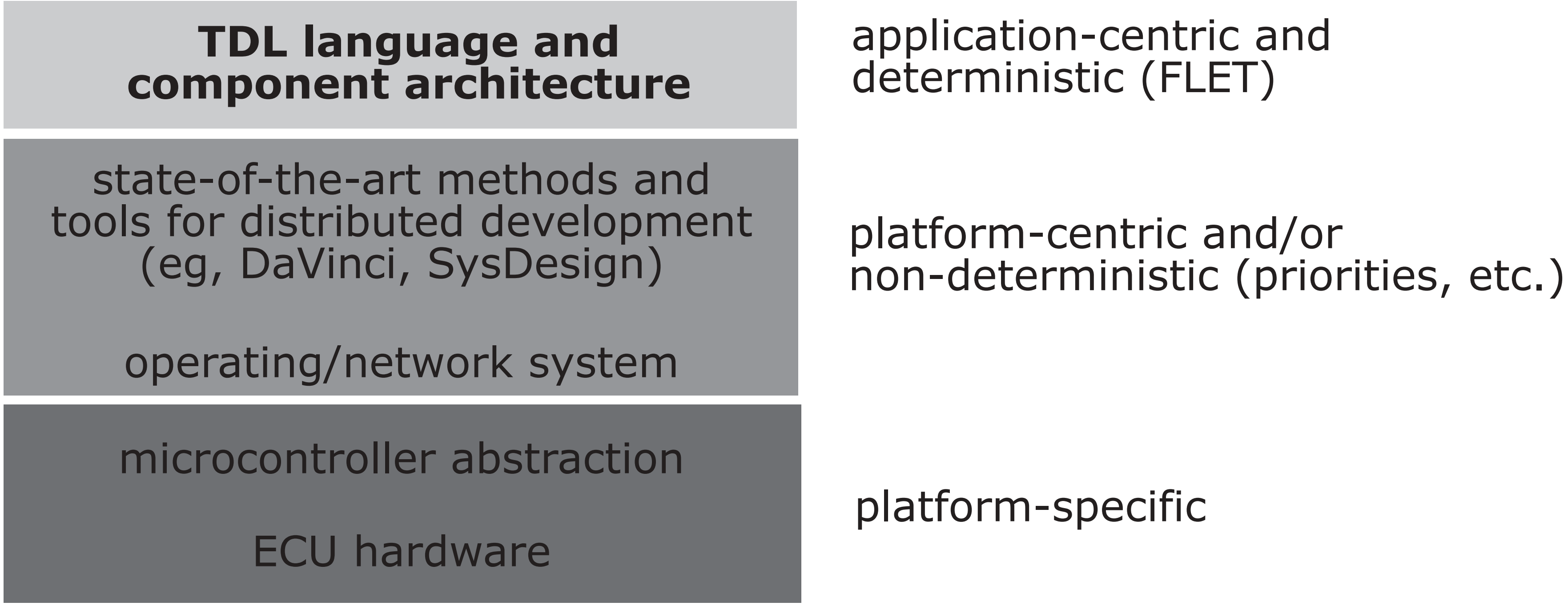
module	@
M1	ECU1
M2	ECU2
M3	ECU1

- communication mechanism for sensors/actuators (Software Bus)
- implementations on (TT)CAN, TTP/C, RT-Linux with TT-Ethernet
- steps towards platform independent fault tolerance

V-Cluster-Life-Cycle: independently developed TDL components



Abstraction levels for control system development



TDL language and component architecture

application-centric and deterministic (FLET)

state-of-the-art methods and tools for distributed development (eg, DaVinci, SysDesign)

operating/network system

platform-centric and/or non-deterministic (priorities, etc.)

microcontroller abstraction

ECU hardware

platform-specific