Interpreting Distributed System Architectures using VDM++
A Case Study

ir. Marcel Verhoef, Chess and
Radboud University Nijmegen, The Netherlands

prof. dr. Peter Gorm Larsen,
Engineering College of Aarhus, Denmark

Agenda

- ➢ **Why is System Architecting hard?**
- **Is VDM++ a candidate for improving this?**
- **A Case Study: A Counter Measures System**
- **Simulation of Abstract Model**
- **Concluding Remarks**

Why Is System Architecting Hard?

- Early phases system life-cycle are extremely volatile
- Many unknowns (not just technical), but nevertheless:
- The ***key decisions*** need to be made ***early*** on
- Often ***out-of-phase*** system development occurs
- Design is typically ***mono-disciplinary*** organized
- "Shooting at a moving target"

iha.dk

Why Is System Architecting Hard?

- Task of the System Architect is to
  - Increase confidence in the system design
  - To reduce project and product risks
  - While dealing with uncertainty
  - While working under high time pressure

- The System architect needs to ***bridge the gap*** between the disciplines and ***deal with the design complexity*** in a very ***cost-effective way***

Agenda

- ✓ **Why is System Architecting hard?**
- ➢ **Is VDM++ a candidate for improving this?**
- **A Case Study: A Counter Measures System**
- **Simulation of Abstract Model**
- **Concluding Remarks**

What is VDM++?

- Vienna Development Method (we use an object oriented extension of the ISO standard language)
- Can be used for creating abstract and precise models
- VDM++ has excellent industrial track record
- Industry strength tools are available

  VDMTools → `http://www.vdmtools.jp/en`
- Very rich tool set with an interpreter, UML coupling, code generation, round-trip engineering
- Extensions for distributed embedded real-time systems

Real-Time Distributed Systems Modelling

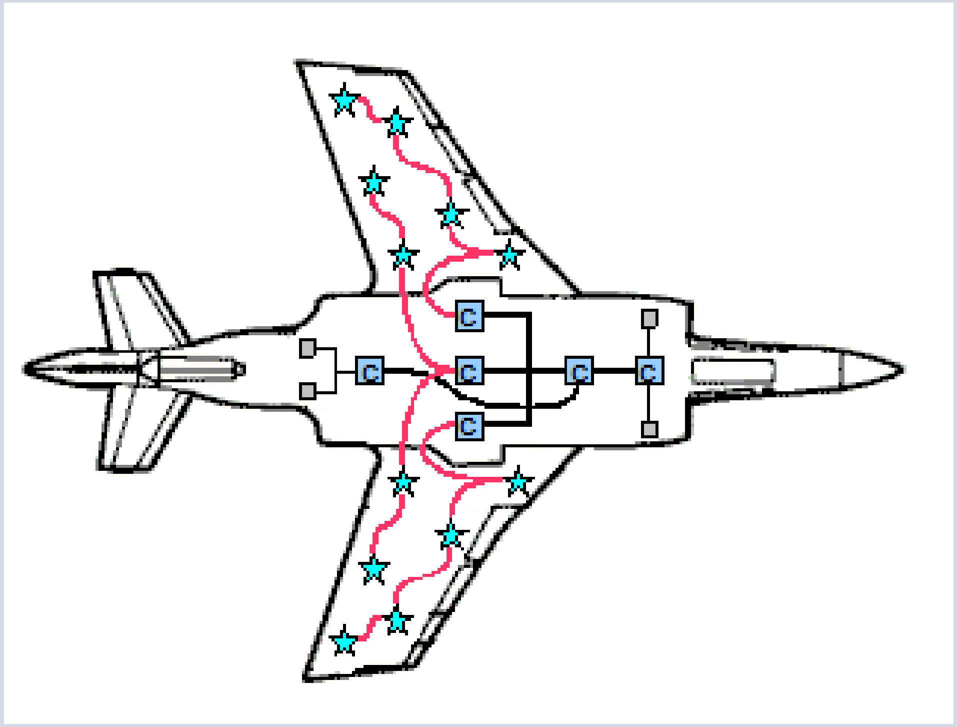
- Already available: **periodic** and **duration**
- MoC: communicating multi-processor multi-threading
- Introduce asynchronous operations ("**async**")
- Introduce context aware time penalties ("**cycles**")
- Introduce **BUS** and **CPU** as first class citizens
- Class instances can be deployed on a specific **CPU**
- Introduce the **system** class to capture the architecture
- Allow non-strict periodic behavior (*p*, *j*, *d*, *o*)

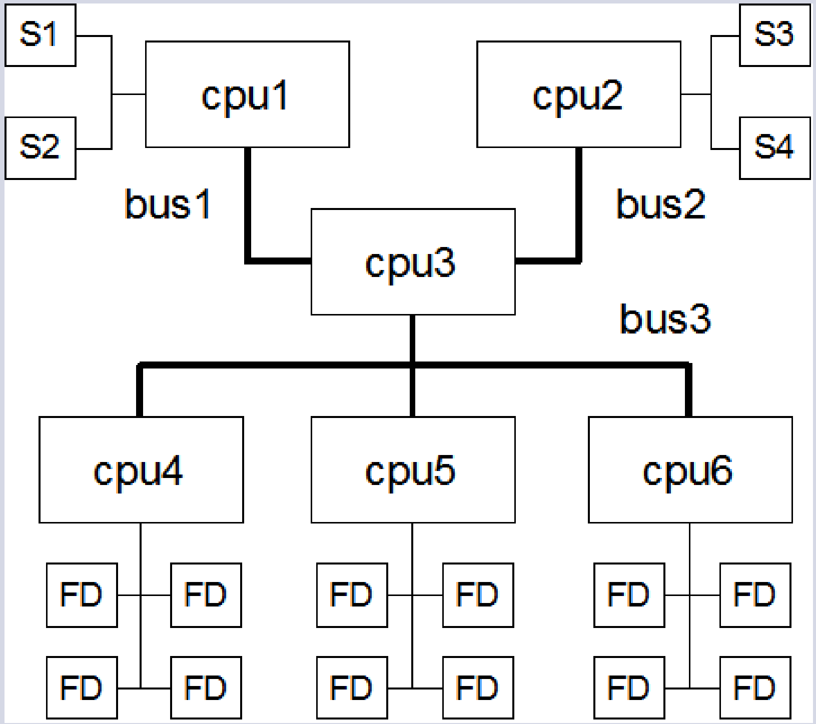
Agenda

- ✓ **Why is System Architecting hard?**
- ✓ **Is VDM++ a candidate for improving this?**
- ➢ **A Case Study: A Counter Measures System**
- **Simulation of Abstract Model**
- **Concluding Remarks**

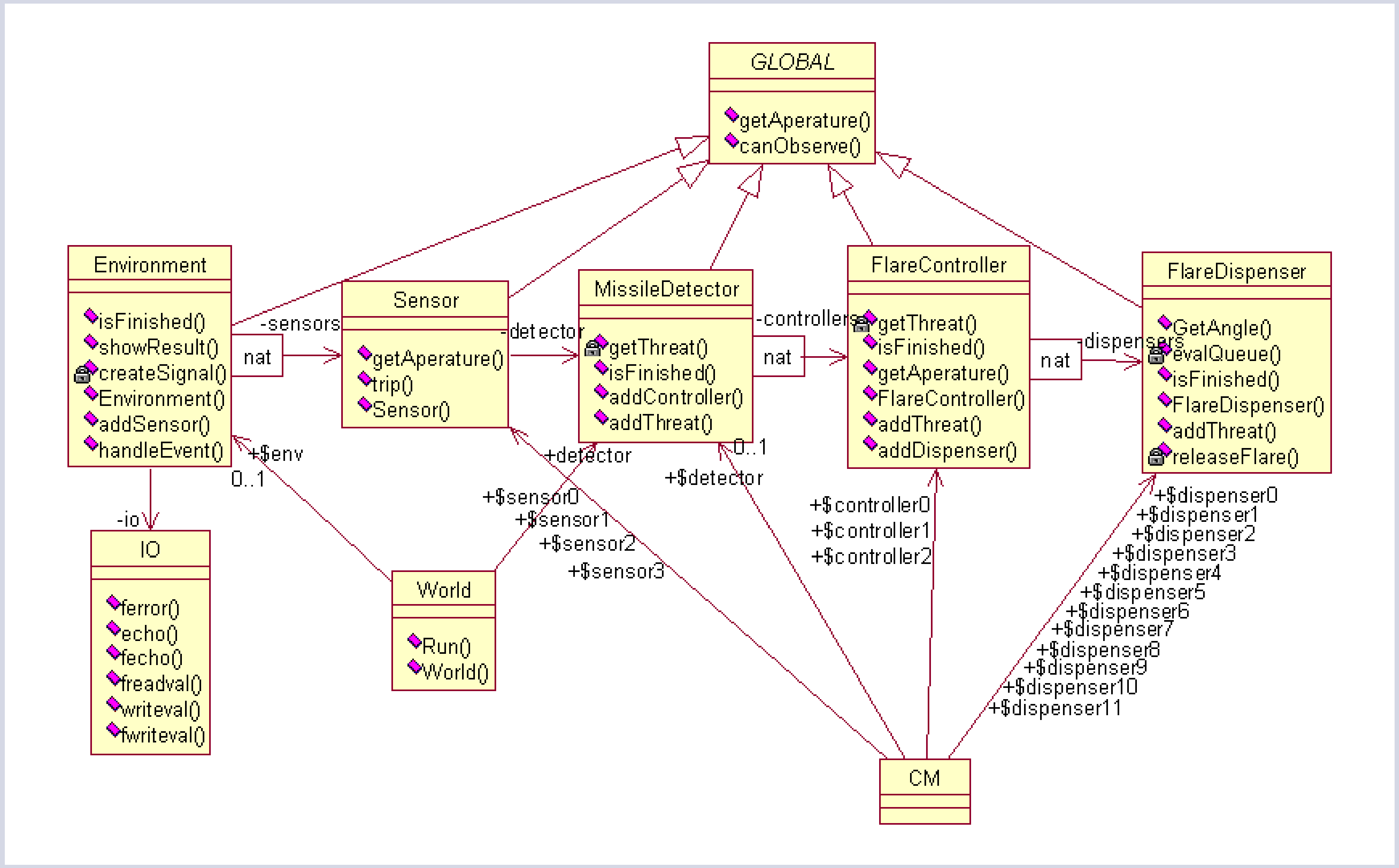
An Aircraft Counter Measures System

Potential Physical System Architecture



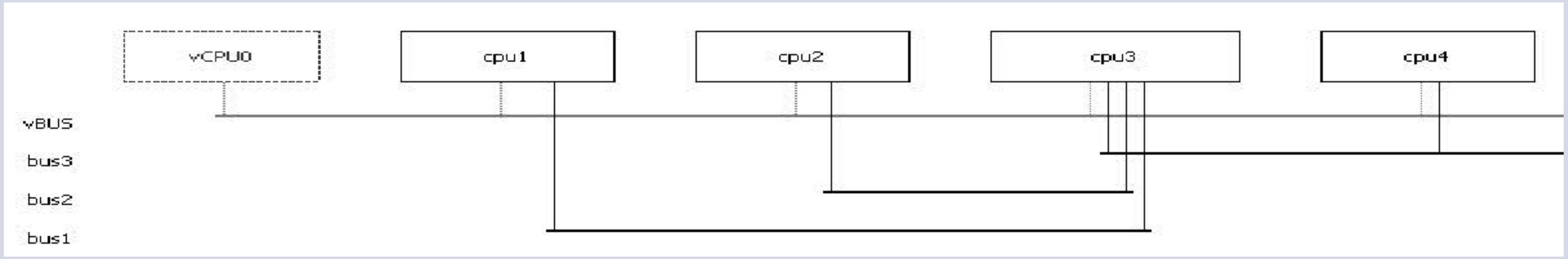


Software Architecture Overview



Declaring CPUs and BUSes

```
instance variables
cpu1 : CPU := new CPU (<FCFS>,1E6);
cpu2 : CPU := new CPU (<FCFS>,1E6);
cpu3 : CPU := new CPU (<FP>,1E9);
cpu4 : CPU := new CPU (<FCFS>,1E6);
cpu5 : CPU := new CPU (<FCFS>,1E6);
cpu6 : CPU := new CPU (<FCFS>,1E6);
bus1 : BUS := new BUS (<FCFS>,1E6,{cpu1,cpu3});
bus2 : BUS := new BUS (<FCFS>,1E6,{cpu2,cpu3});
bus3 : BUS := new BUS (<FCFS>,1E6,{cpu3,cpu4,cpu5,cpu6});
```



Deploying Objects to Processors

```
public CM: () ==> CM
CM () ==
  ( cpu1.deploy(sensor0);
    cpu1.deploy(sensor1);
    cpu2.deploy(sensor2);
    cpu2.deploy(sensor3);
    cpu3.deploy(detector);
    cpu3.deploy(controller0);
    cpu3.deploy(controller1);
    cpu3.deploy(controller2);
    cpu4.deploy(dispenser0);
    cpu4.deploy(dispenser1);
    cpu4.deploy(dispenser2);
    cpu4.deploy(dispenser3);
    cpu5.deploy(dispenser4);
    cpu5.deploy(dispenser5);
    cpu5.deploy(dispenser6);
    cpu5.deploy(dispenser7);
    cpu6.deploy(dispenser8);
    cpu6.deploy(dispenser9);
    cpu6.deploy(dispenser10);
    cpu6.deploy(dispenser11) )
```

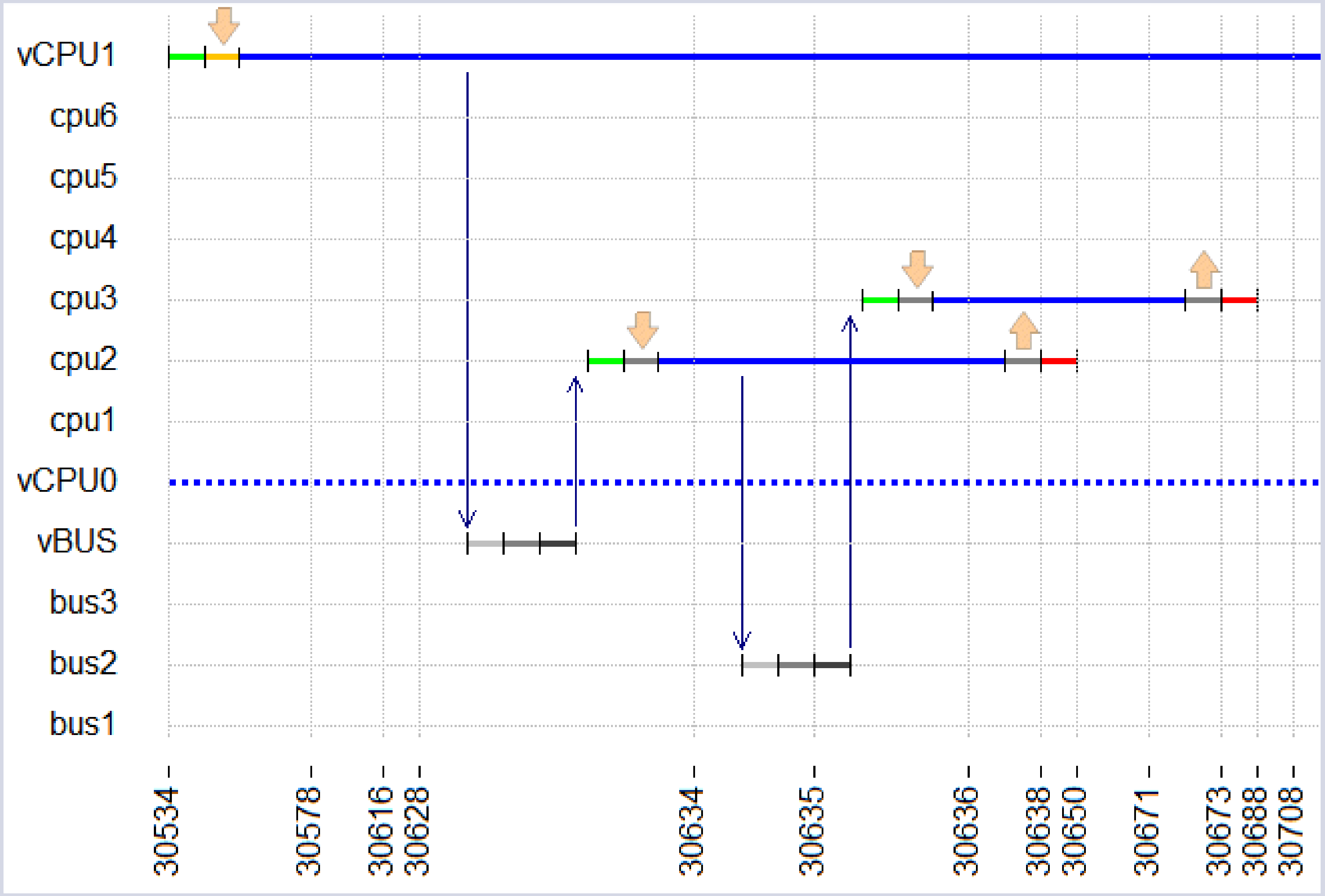
Agenda

- ✓ Why is System Architecting hard?
- ✓ Is VDM++ a candidate for improving this?
- ✓ A Case Study: A Counter Measures System
- ➢ Simulation of Abstract Model
- Concluding Remarks

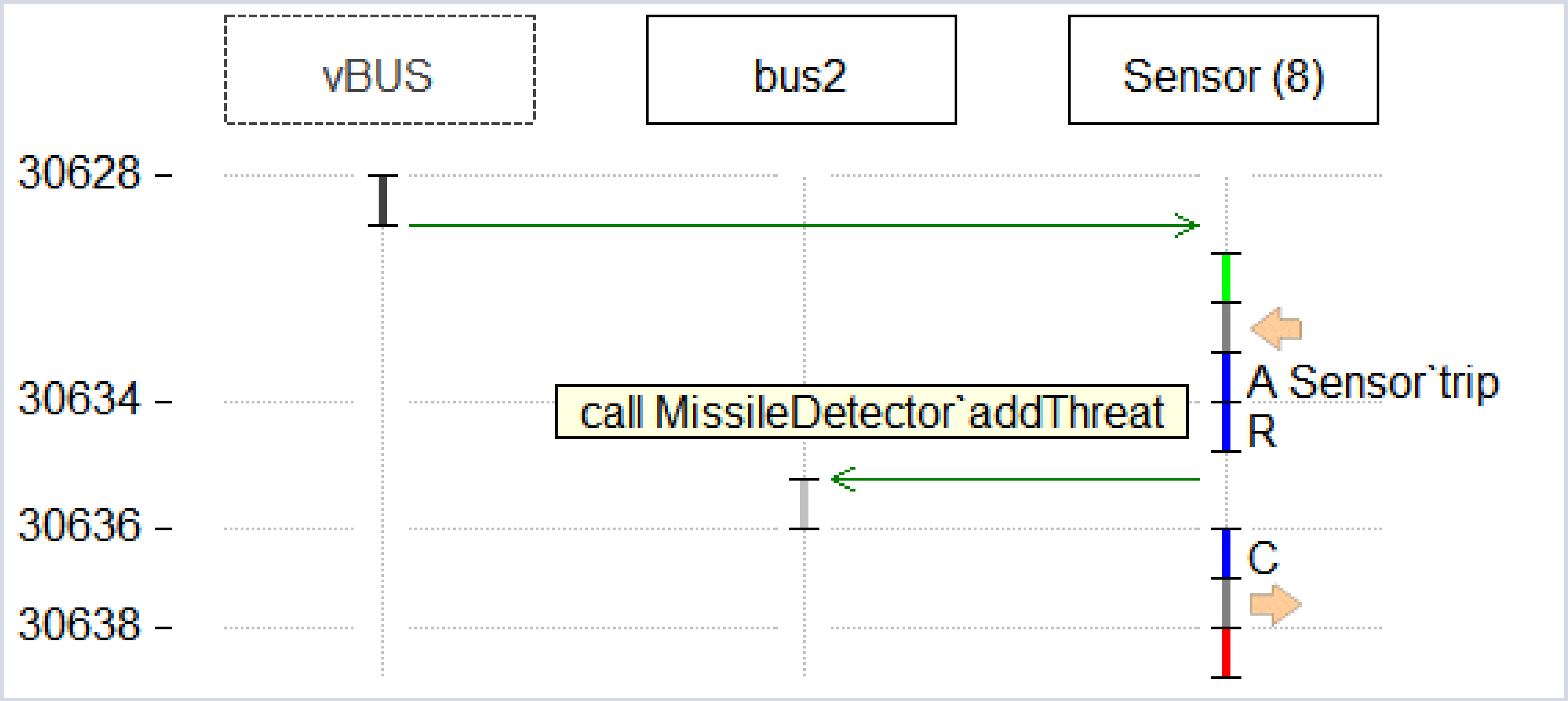
Simulation of System Functionality

- The VDM++ enables simulation of model functionality
- Observe behavior distributed hardware architecture
- A log file is produced during simulation
- The log file can be analyzed for timing behaviour

System-level Execution Overview



Detailed Execution Overview per CPU



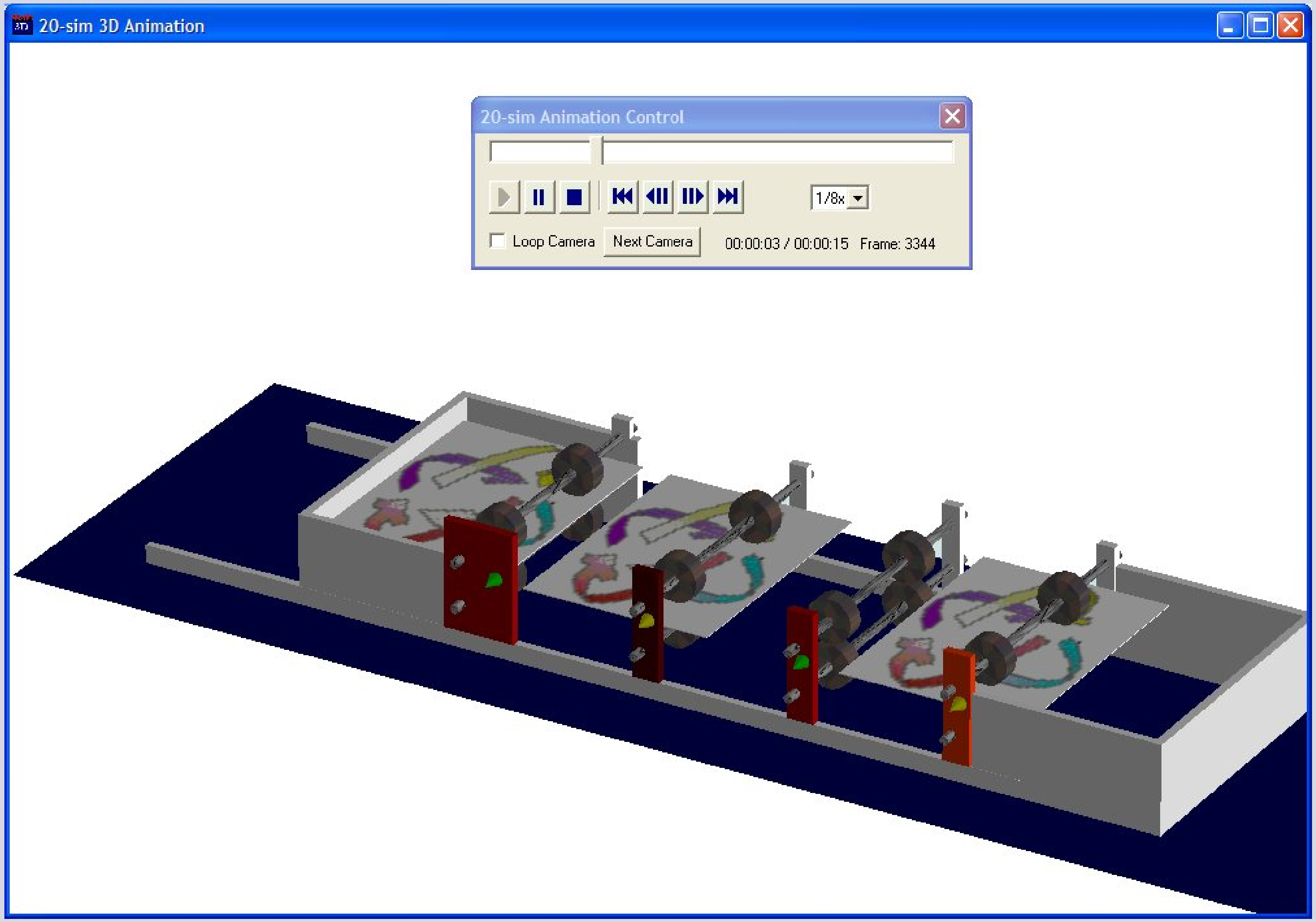
Agenda

- ✓ Why is System Architecting hard?
- ✓ Is VDM++ a candidate for improving this?
- ✓ A Case Study: A Counter Measures System
- ✓ Simulation of Abstract Model
- ➢ Concluding Remarks

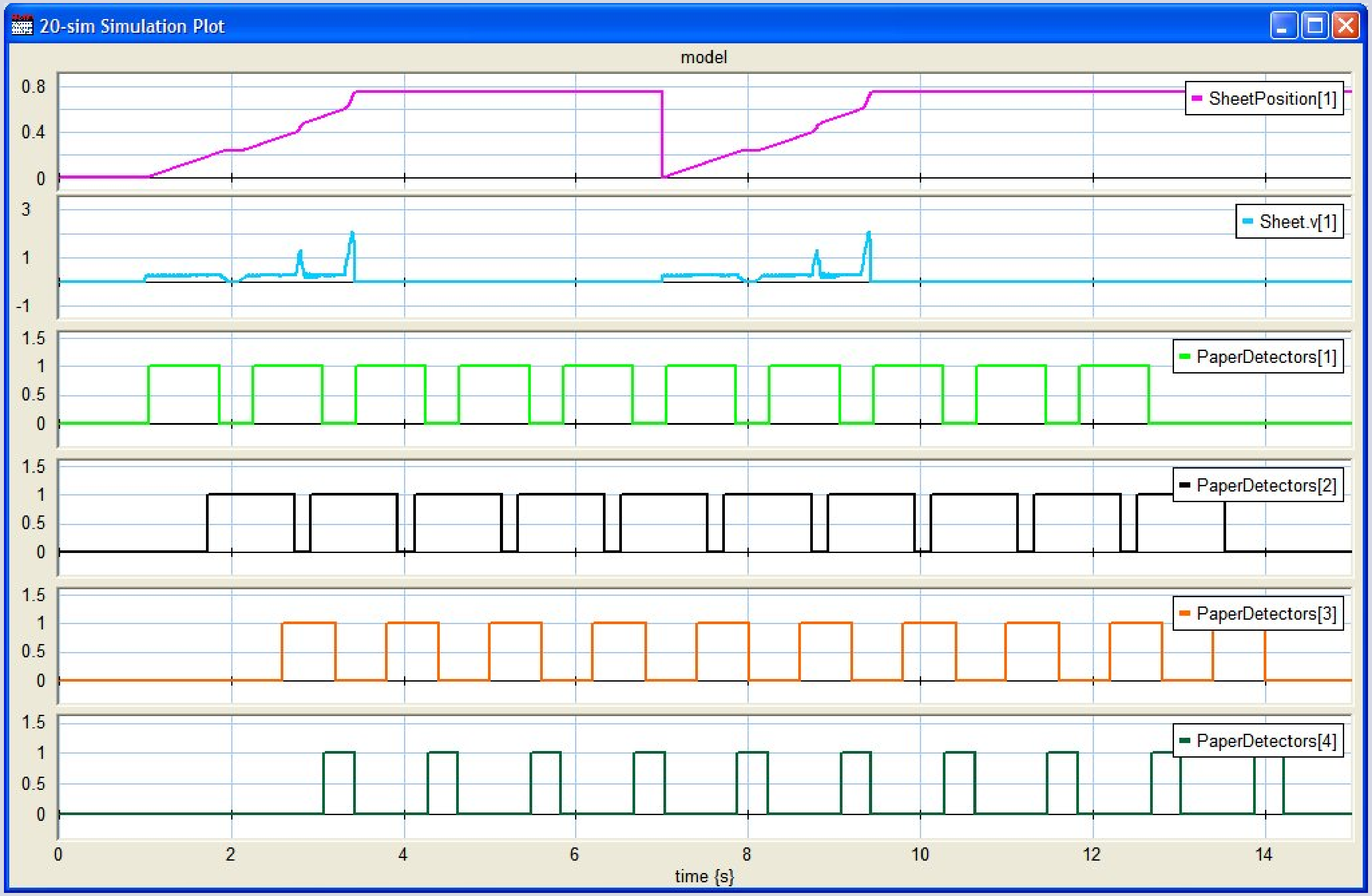
Concluding remarks

- VDM++ can be used to reduce design complexity using powerful abstraction mechanisms
- Potential design bottlenecks can be discovered cost-effectively very early in the life cycle
- A potential enabler for bridging the discipline gap
- System level timing requirements can be formally stated and tested explicitly
- The discrete event simulation can be combined with continuous time simulation (current work)

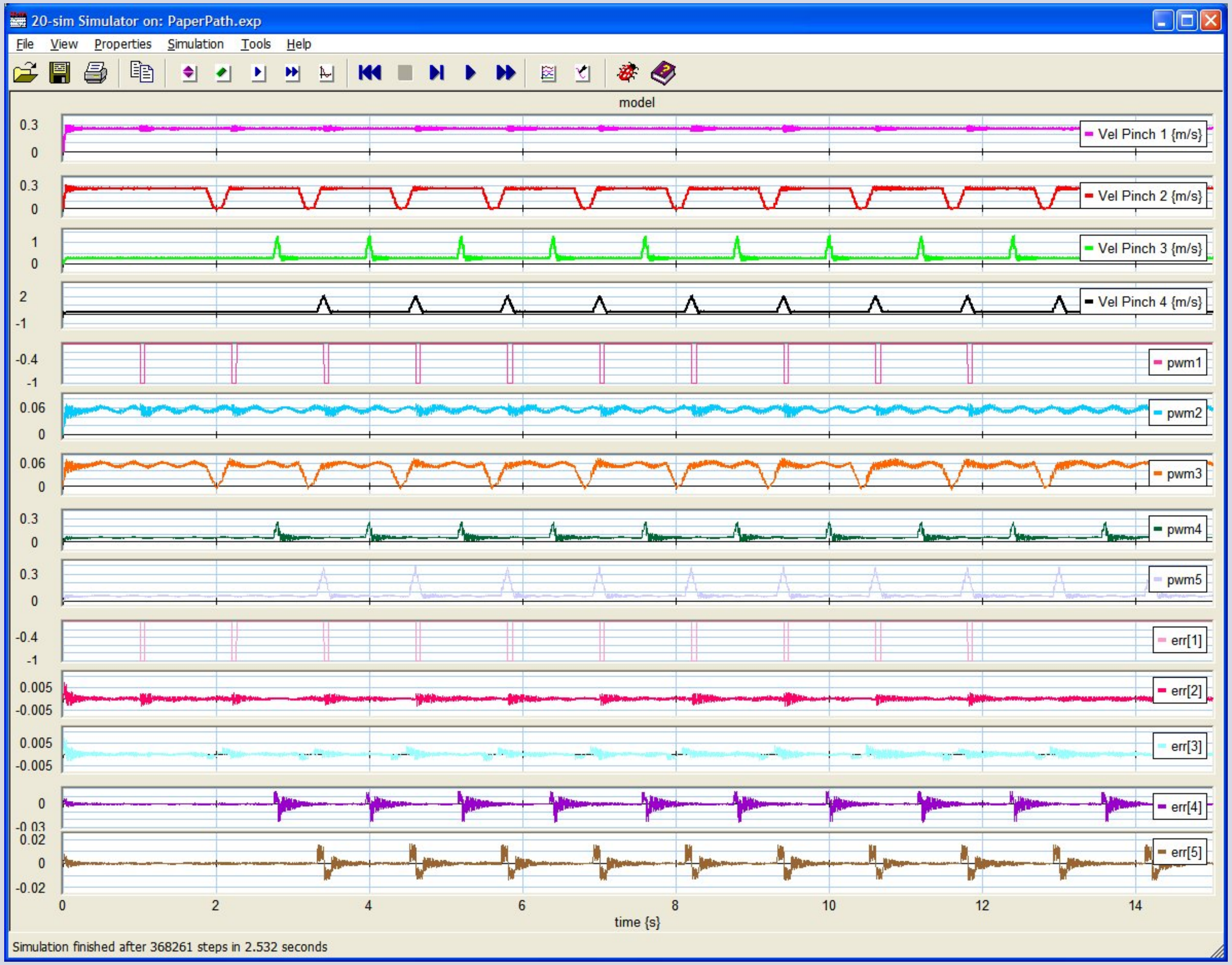
Current and future work (1)



Current and future work (2)



Current and future work (3)



Simulation finished after 368261 steps in 2.532 seconds