VALI-CPPS

Verification And Large-scale Integration for Cyber-Physical Production Systems

Dr.-Ing. Jörg Walter
OFFIS e.V., Oldenburg, Germany

The Challenges of Cyber-Physical Production Systems

1. **Heterogeneity & Interdisciplinarity**
   > Production systems use lots of different physical effects at the same time (electromagnetism, heat, pressure, mechanics, …)

   > Equipment manufacturers force you to use certain tools/languages

2. **Customisation**
   > Most production lines are one-of-a-kind, no economy of scales

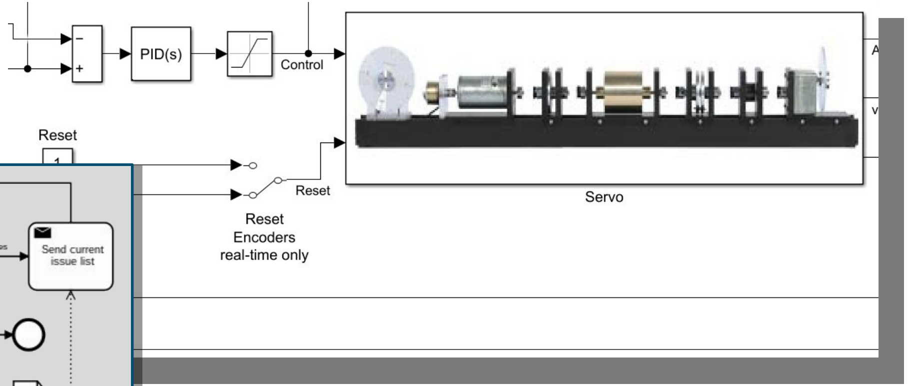
3. **Evolution and Legacy Components**
   > Production systems change over their regular lifetime

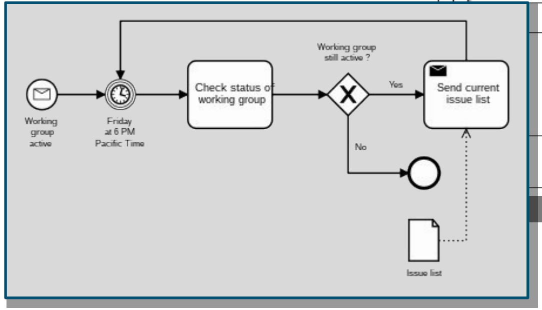
   > Systems run for decades, parts may exist for more than a century

4. **Safety (and increasingly, Security)**
   > Failures can be expensive, cause injury, and cascade to other systems

5. **Task Complexity**
   > Industry 4.0 creates computation and interaction requirements like never before

Desired Design Flow





The Problem with the State of the Art

Control systems are developed with tools and languages from 30 years ago

- IEC 61131 defines programming languages that incorporate the latest trends from the 80s
- Examples (in decreasing order of usability): Structu
- No syst

Market fo methodol

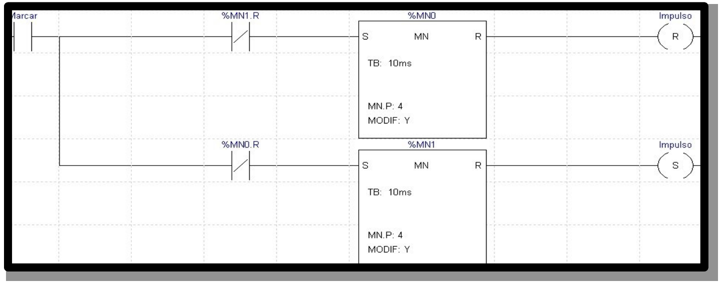
- Individ
- PLC pro change
  - Protection of business models through closed ecosystems
  - Chicken-and-egg problem with new tools/languages

How can we improve the design process of control systems for emerging cyber-physical production systems under these constraints?

```
(* simple state machine *)
TxtState := STATES[StateMachine];

CASE StateMachine OF
   1: ClosingValve();
      StateMachine := 2;
   2: OpeningValve();
ELSE
    BadCase();
END_CASE;
```

```
           LD      Speed
           GT      2000
           JMPCN   VOLTS_OK
           LD      Volts
VOLTS_OK   LD      1
           ST      %Q75
```



Source: Wikipedia

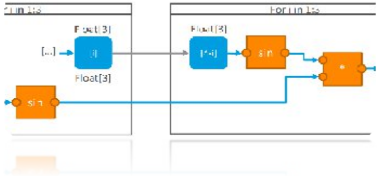
The Idea

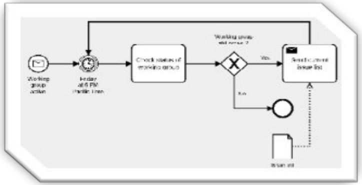
Proposal: IEC 61499 as intermediate implementation/integration model

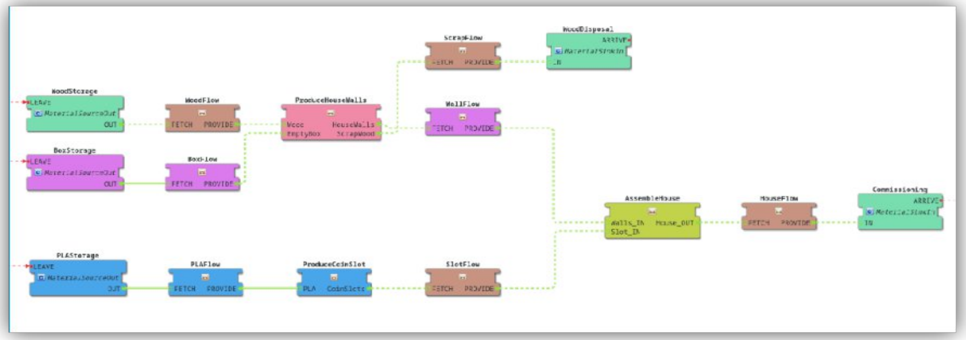
- Unified view on control software
- Model-based design flow from the top down to this model
- Iterative refinement
- Complexity management

Advantages:

- Traceability!
- Support for heterogeneous CPPS
- Reduced developer effort (low code?)
- Extension to DevOps possible







https://www.eclipse.org/4diac/

https://www.ros.org/

What the Integration Model Buys Us

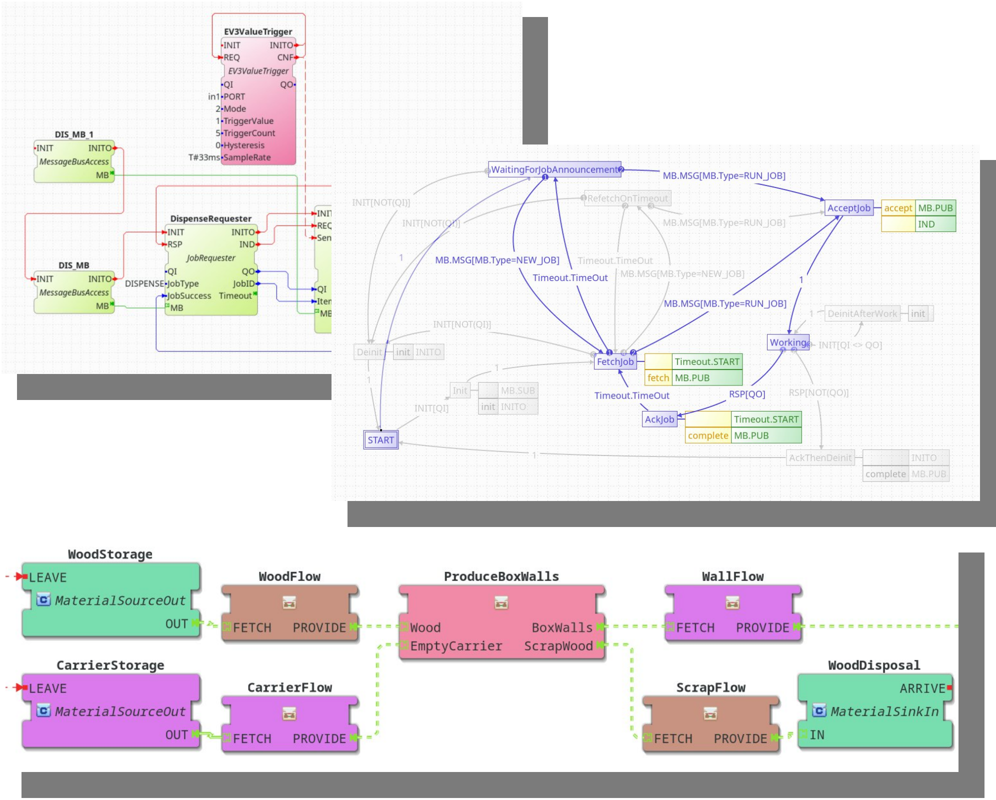
IEC 61499 as Integration Model

Model-based

- Multiple modelling styles
- Semantics compatible with popular source model languages
- Component model accomodates wide range of targets
- Suitable abstraction level range

Executable

- Early Simulation
  - Virtual integration testing
  - Extrafunctional Properties



Implementation Refinement

Use hierarchy to change viewpoint

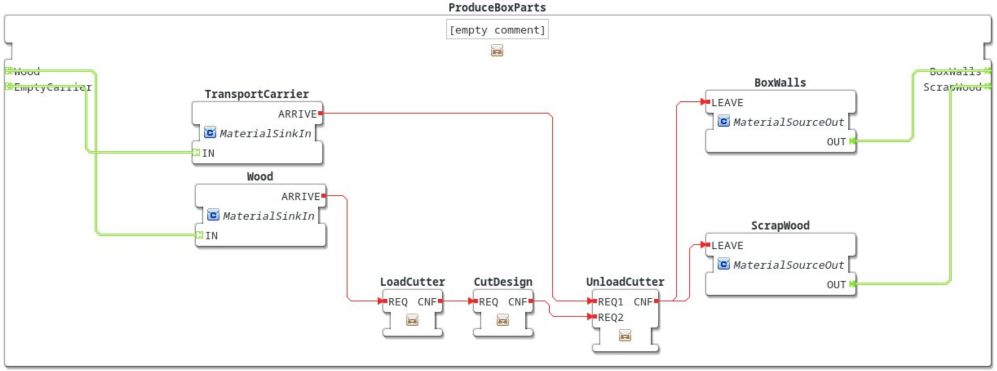
- Goal: Control program
- Basic program unit: skill

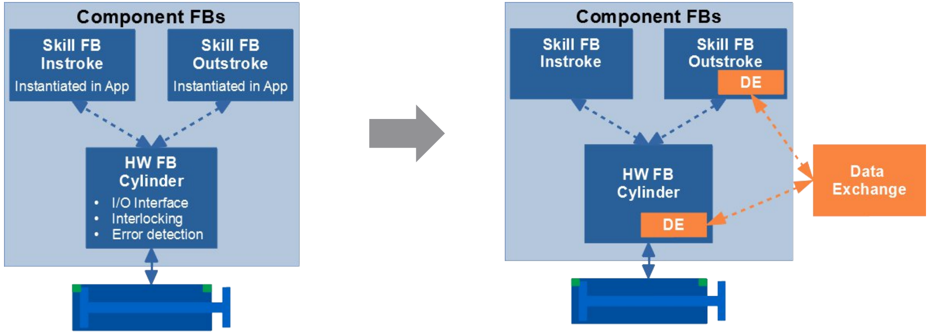
Skills orthogonalize secondary aspects

- Monitoring, HMI, error recovery, …
- Even scheduling/MES is changeable
  - Self-organized? Central control?

Skills allow black-box specialisation

- Manually optimised implementations
- Custom hardware w/o code generation
- Generated code from other tools
- Skills from other run-time environments (e.g. ROS2)





Architectural Concepts for IEC 61499-based Machine Controls: Beyond Normal Operation Handling, Sonnleithner et al., ETFA 2022

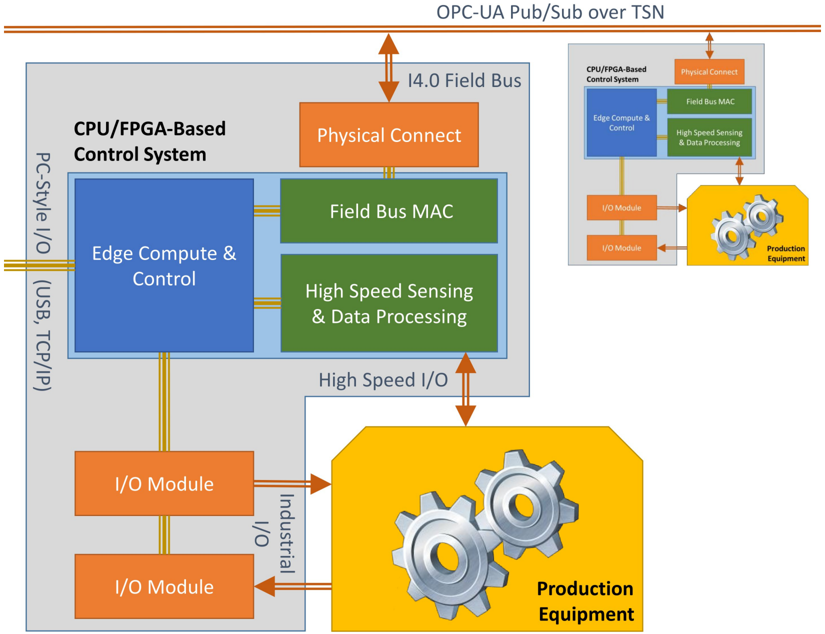
Target Variability

Platform-based approach: meet-in-the-middle

- Local distribution
  - AI accelerators, GPGPU, FPGA, DSP, ASIP
  - Run-time environments (4diac, ROS, plain C++, …)
- Non-local distribution
  - Legacy devices
  - Device sharing
  - Virtual PLC

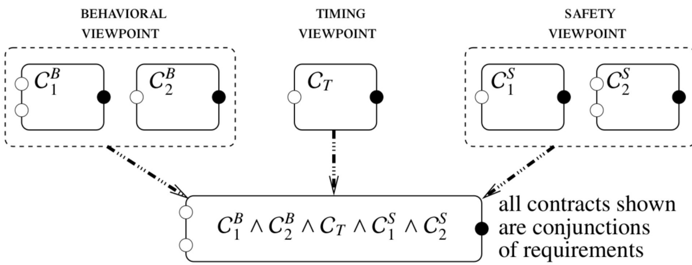
Multi-objective design space exploration

- Latency
- Throughput
- Energy



Safety for Model-Based CPPS

Assume-Guarantee Contracts in a Nutshell

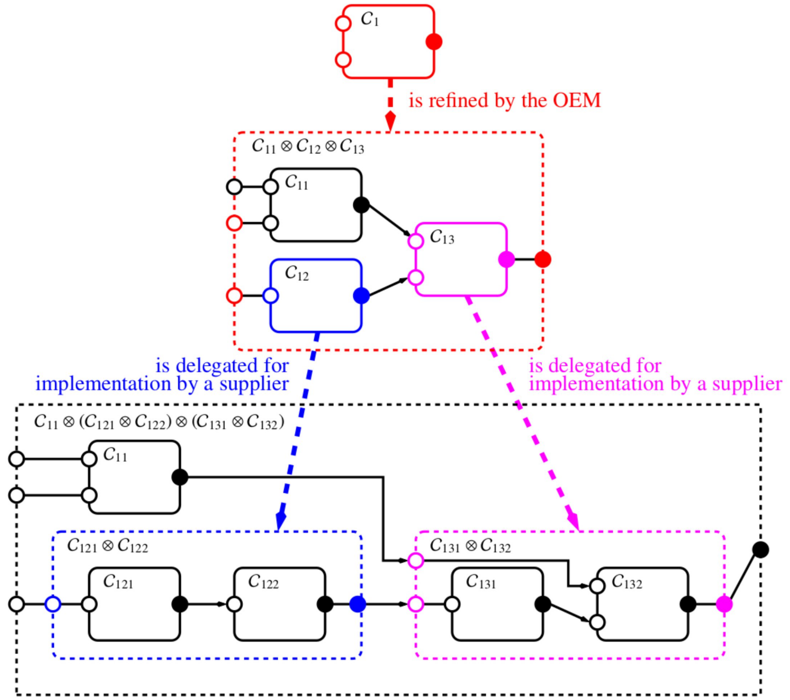


Contract: pair C=(A, G)

- Assumptions on environment
- Gurantees of the system under those assumptions

Operations for hierarchical design

- Refinement (vertical)
- Composition (horizontal)



Benveniste et al.: Contracts for Systems Design: Theory (2015)

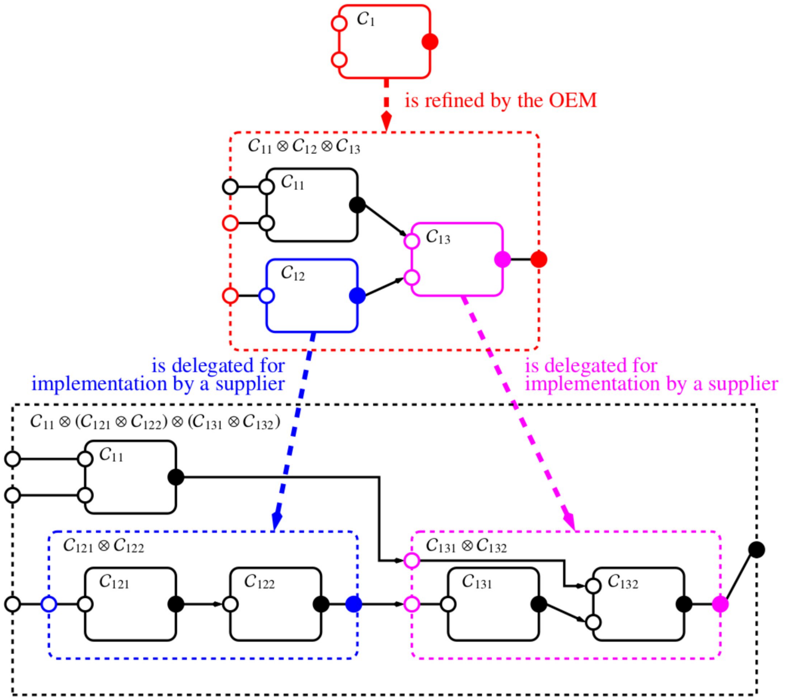
Verification and Validation

Composition & Refinement formally defined

- Model-checking for small systems (e.g. Unit Testing)
- Simulation for large systems (e.g. Integration Testing)
- Virtual Integration Testing

Advantages for complex systems

- Fusing multiple viewpoints
- Traceability of contracts to origin specification
- Independent component development (and updating!)



Contract Language Examples

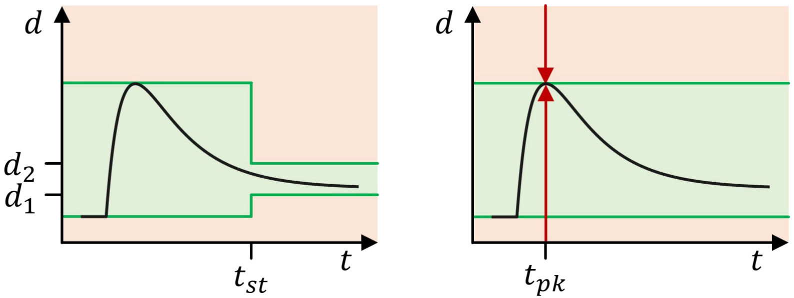
TSBC – Time-Sensitive Behavioural Contracts

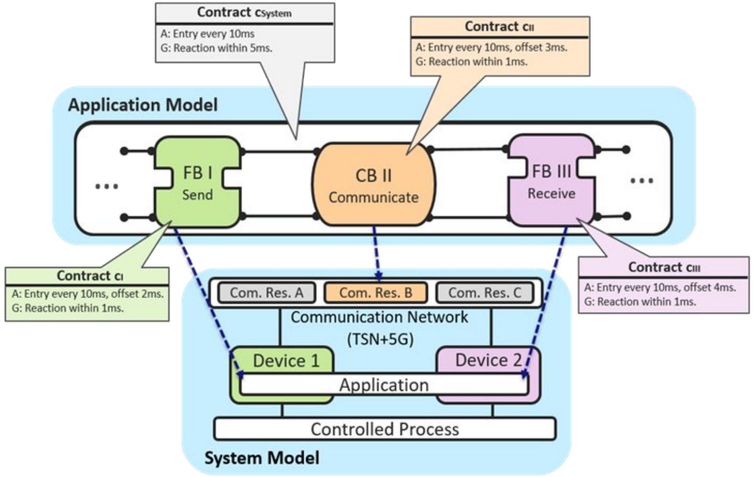
- Functional (value) constraints
- Restriction to time intervals

MTSL – MULTIC Time Specification Language

- Huge amount of of timing properties
- Latency, Jitter, Causality, Duplication, Exclusion, …
- Well researched

$t \in [0; t_{st}] \mathrel{?} d \in [0; d_{pk}] : d \in [d_1; d_2]$ $\quad$ $t = t_{pk} \mathrel{?} d = d_{pk} : d \in [0; d_{pk}]$

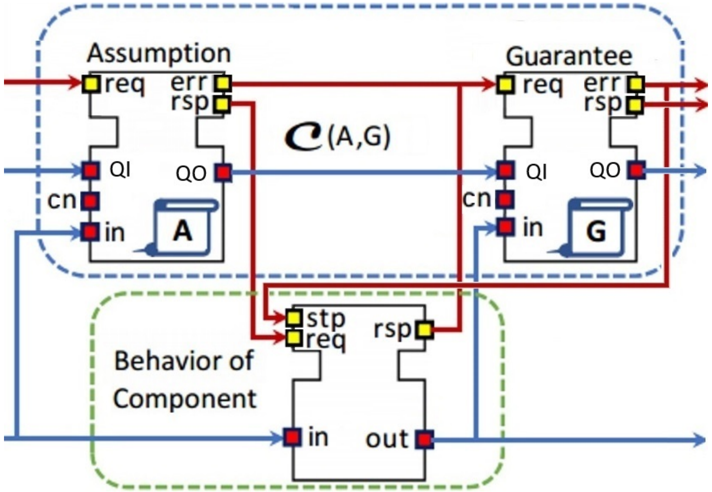




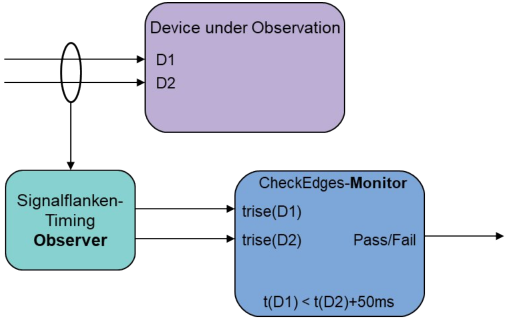
Safety at Run Time and Beyond

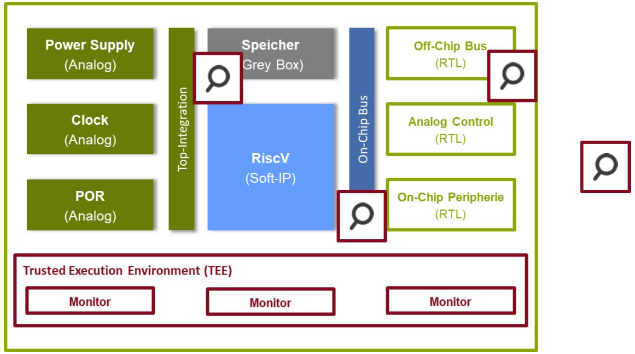
Contract-Based Run-Time Monitoring

End-to-end safety checks in Software…



…and Hardware





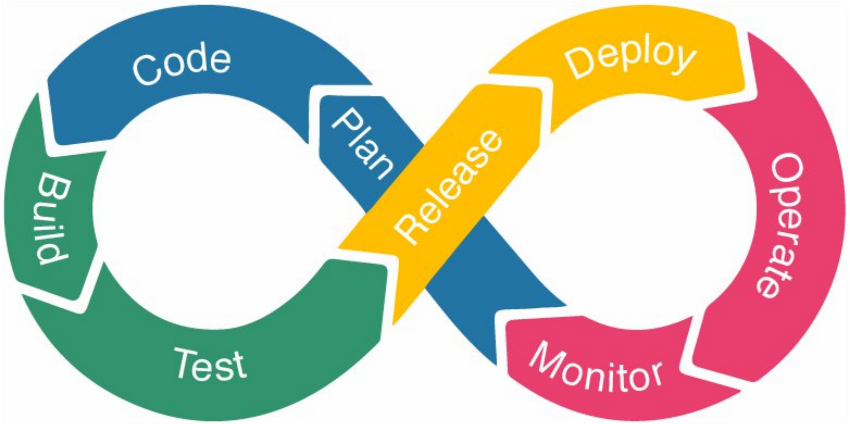
DevOps for CPPS

Revision possible at any model level

- Contracts define and limit scope of re-testing
- Evolution of contracts possible

Monitoring gives required insight

- Auto-generation reduces effort
- Traceability closes the loop



Conclusion

Model-based engineering is the way to go

- Formal contracts are a perfect extension

Unified implementation model decouples models from targets/capabilities

- Implementation details can be changed
- Impact of changes can be contained
- IEC 61499 & 4diac give flexibility

Ultimately allows end-to-end safety checks

- Run-time monitoring
- DevOps