Cyber-Physical Components as building blocks of more dependable industrial CPS

Valeriy Vyatkin

vyatkin@ieee.org

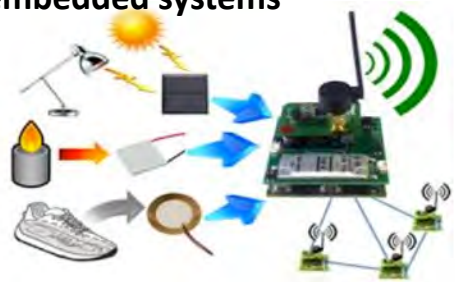
Aalto University
School of Electrical
Engineering

ITMO UNIVERSITY

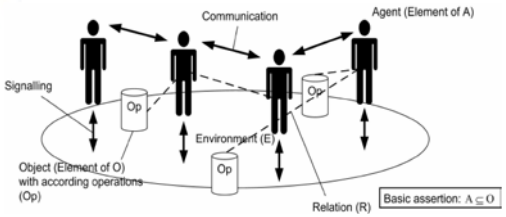
Industrial Cyber-Physical Systems Buzz

Supporting technologies

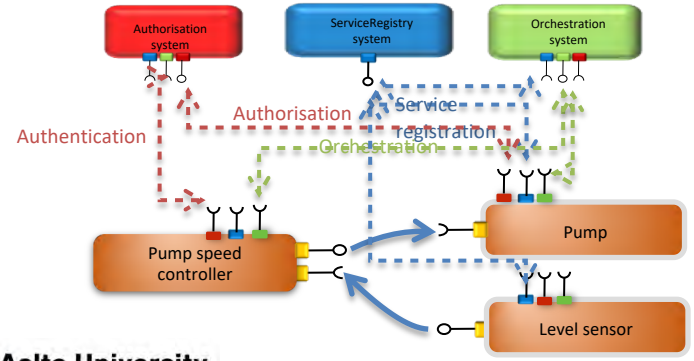
Communicating embedded systems



Agents



Service-oriented architecture



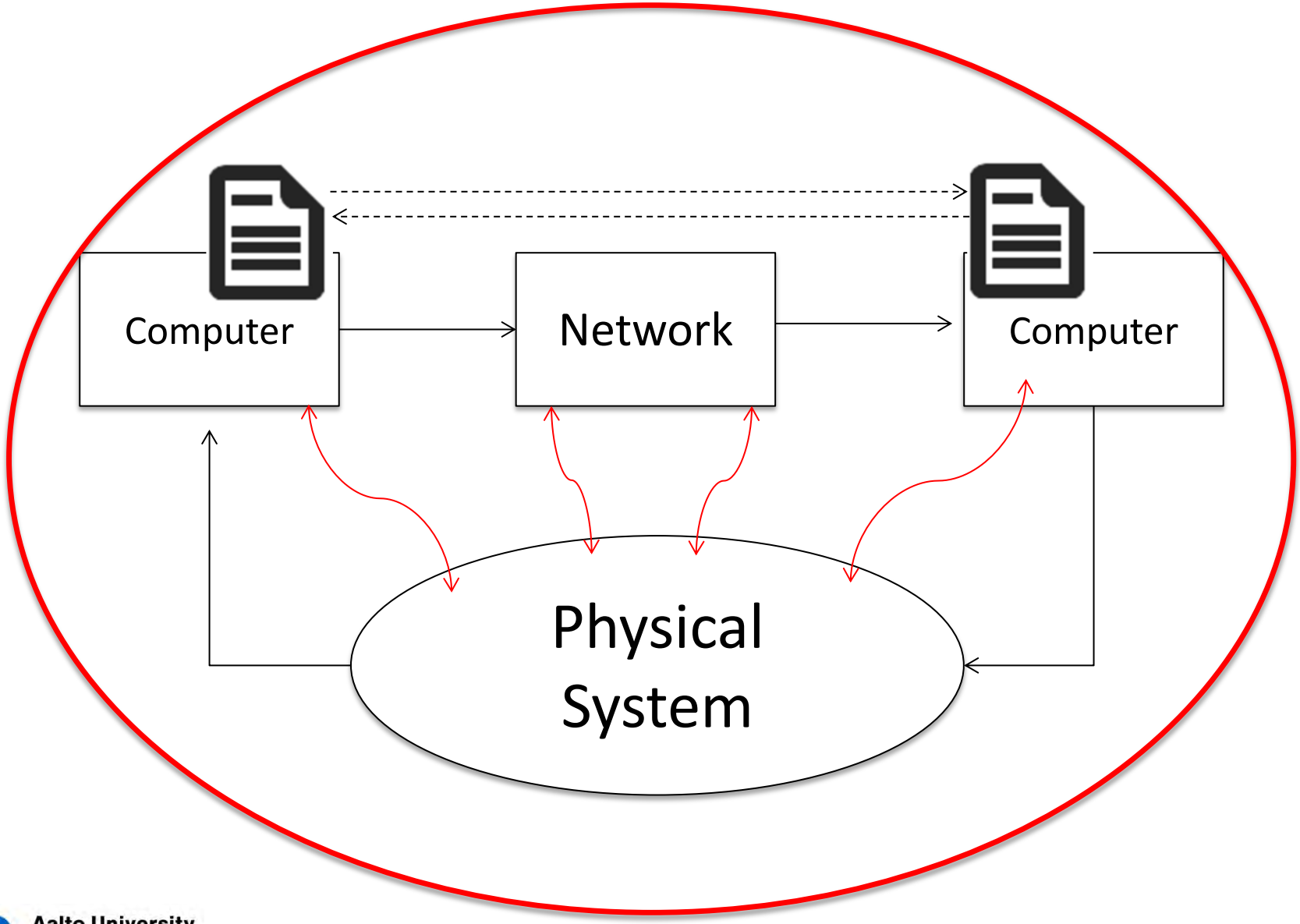
CPS science

How to make them working in a predictable and robust manner ?

Cyber-Physical Engineering !

Aalto University

Cyber-Physical phenomena: cross influences



CPS and Components

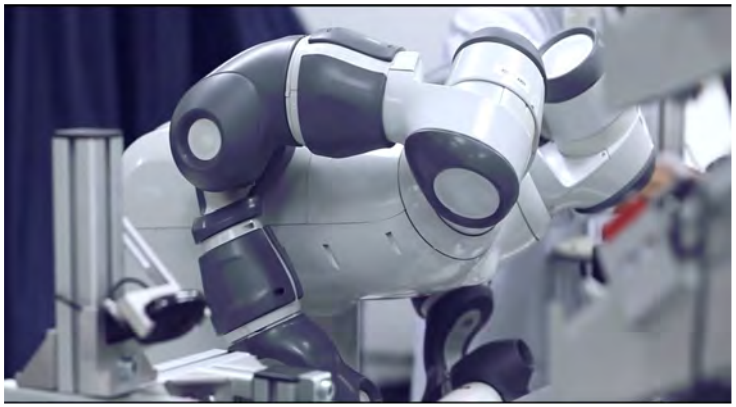
Components at any level of abstraction should be made predictable and reliable if this is technologically feasible. If it is not technologically feasible, then the next level of abstraction above these components must compensate with robustness.*

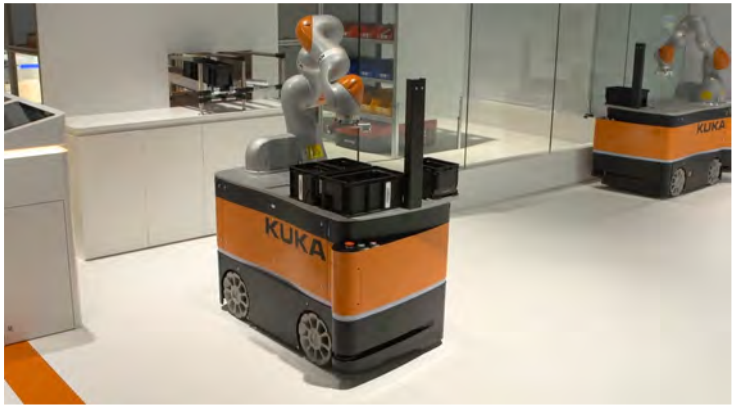
It is harder to make wireless links predictable and reliable. So we compensate one level up, using robust coding and adaptive protocols.

Lee, Edward A. "Cyber physical systems: Design challenges." In *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on*, pp. 363-369. IEEE, 2008.

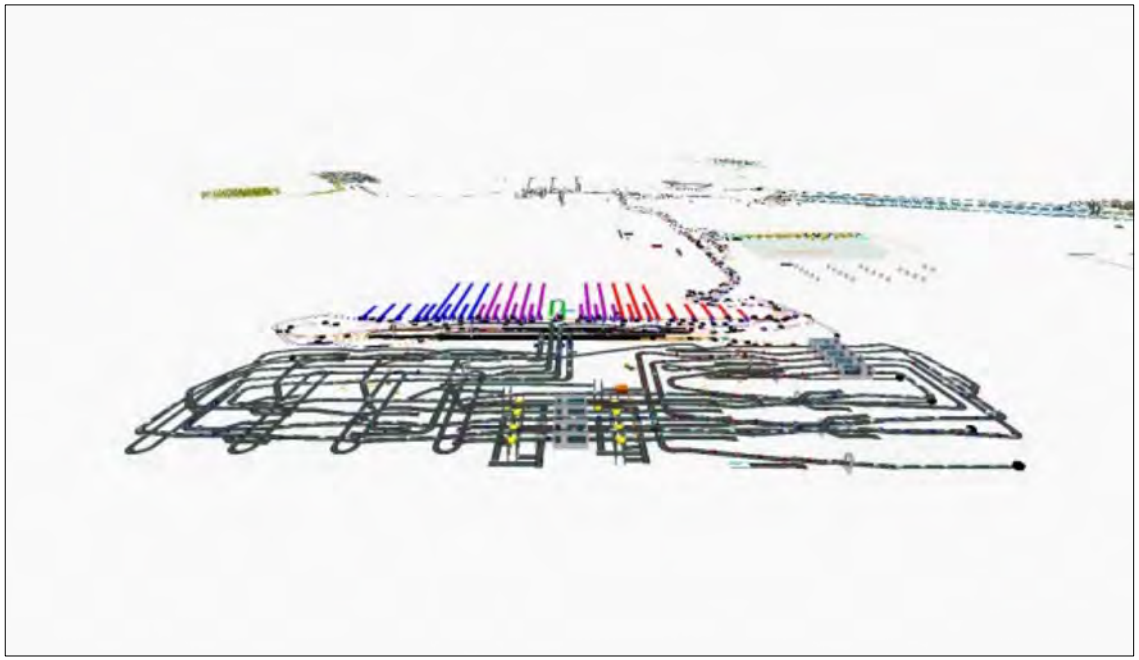
*Example: Network protocols, retransmissions…

Factory of the Future





Airport baggage handling



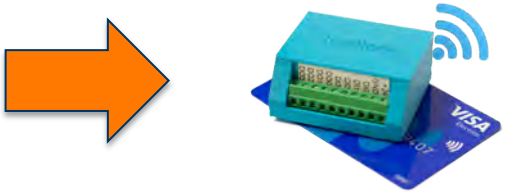
- 30 million bags were temporarily lost by airlines in 2005, and 200,000 of those bags were never reunited with their owners, due to baggage mismanagement
- Airlines and airports have lost between US$1.6 million to US$2.0 million every year in last 6 years. Rate of increase is 12%
- Due to inadequate V&V, there have been delays in delivery of the baggage handling systems to airports, **resulting in losses estimated at US$1 million a day – Denver Airport BHS**

What about nuclear power plants ?

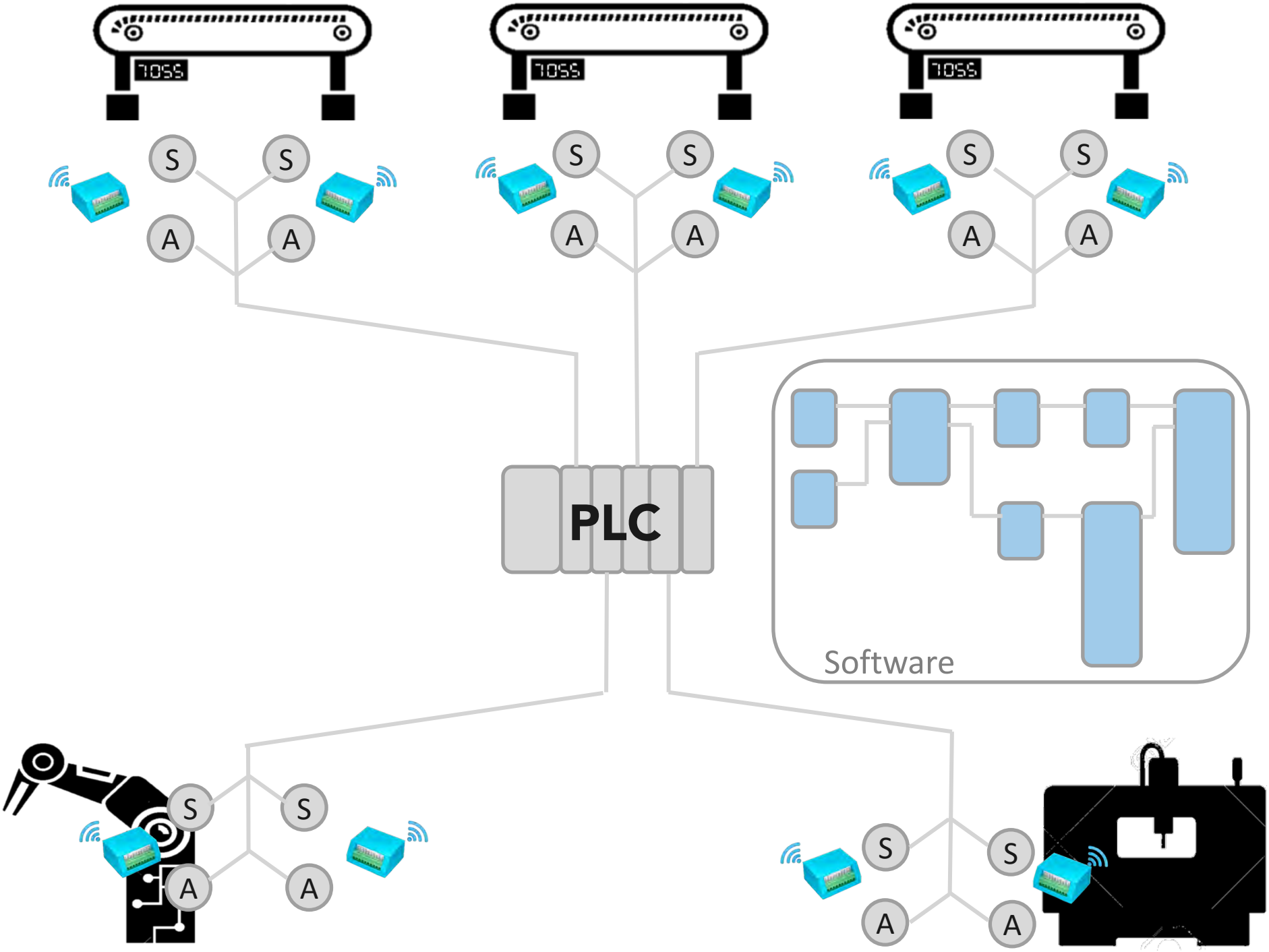
Olkiluoto 3

Future of the PLC Technology: merger with IoT

Current PLC



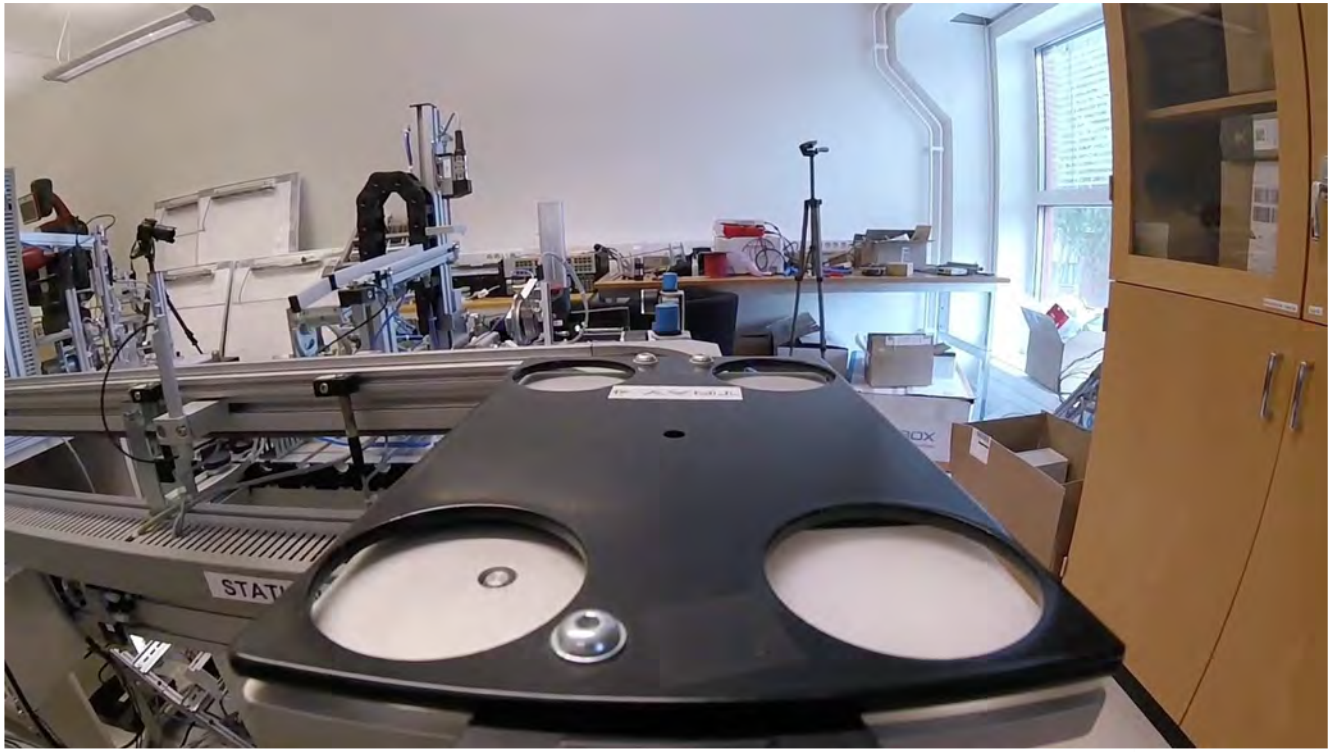
Future Wireless PLC


7055
7055
7055
S
A
PLC
Software

AIC^3 laboratory, Luleå

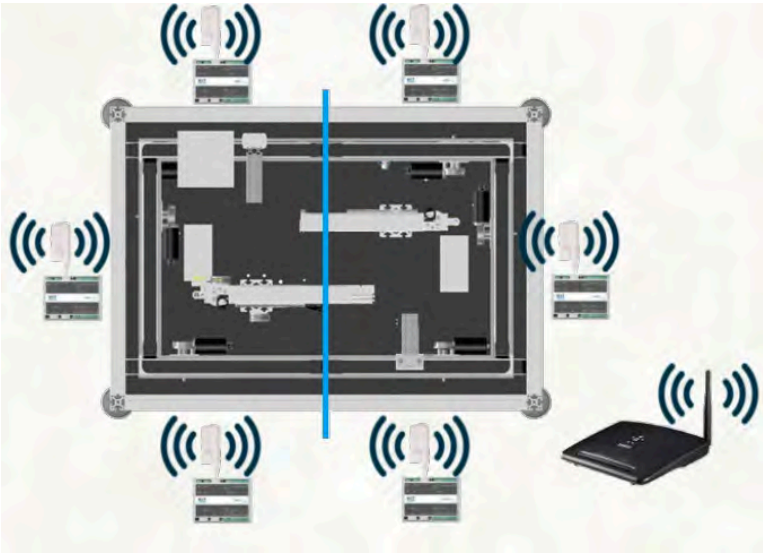
www.ltu.se/AICcube

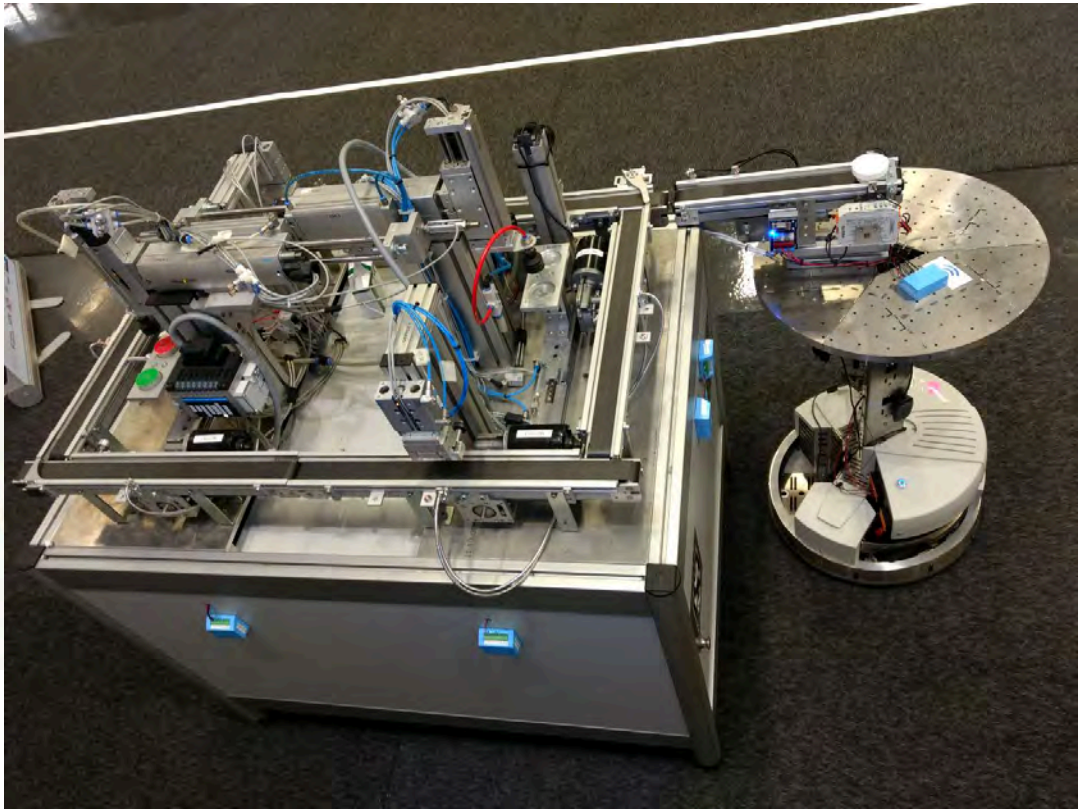
Automation, Industrial Computing, Communication and Control



Aalto Factory of the Future

- Assembly system with flexible physical layout
- Enabled by embedded wirelessly communicating controllers

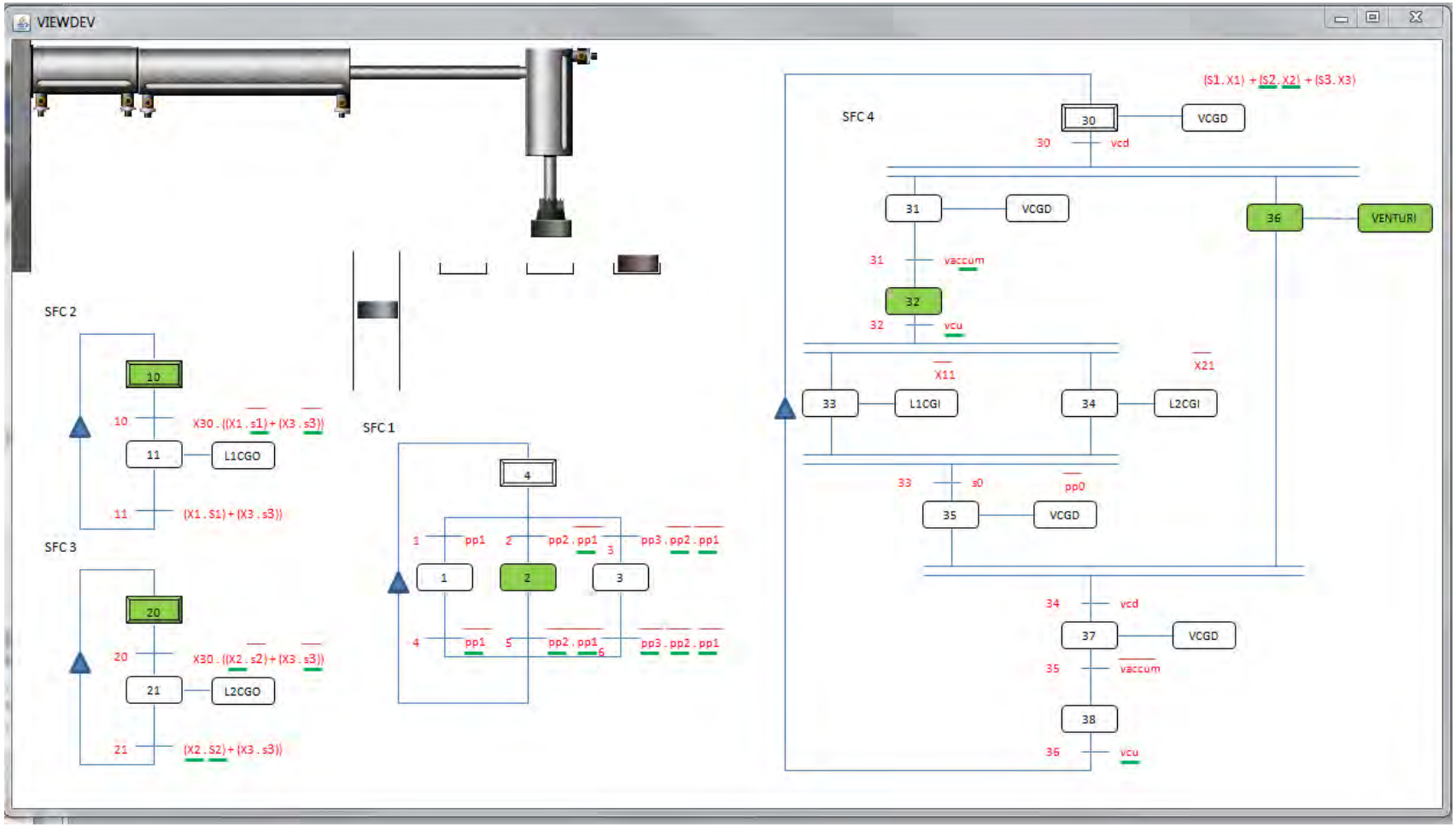




Changing Layout

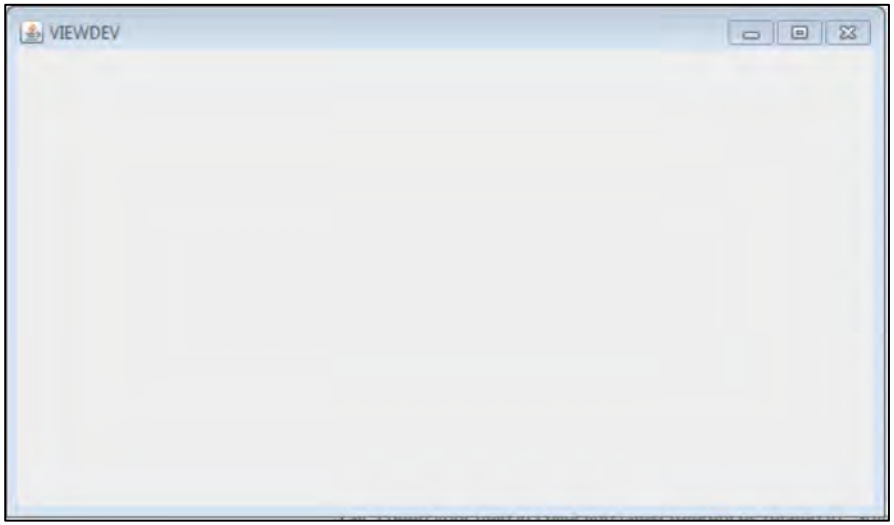
as a reaction on new product order

How do we test automation software?



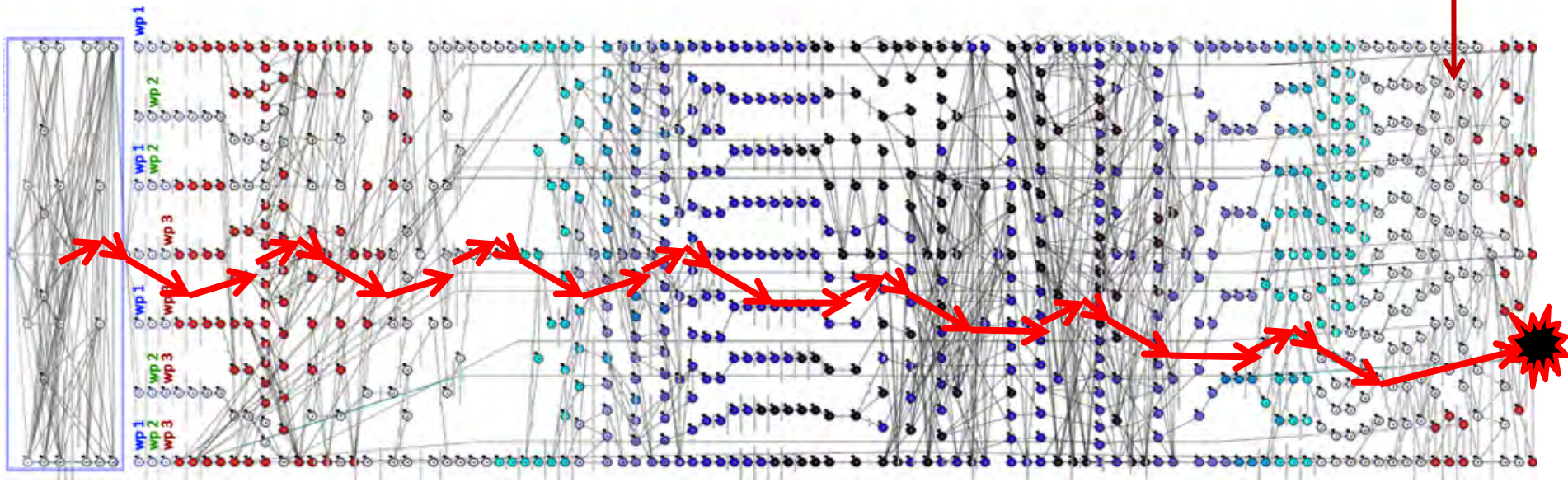
Validation of this code using simulation or in real plant is almost the same

Limits of Simulation



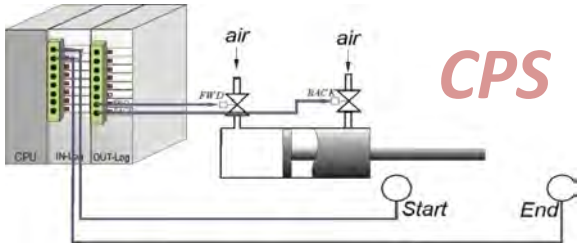
Every simulation run “plays” only one possible behaviour scenario out of the <u>many possible</u> ones.

It is time consuming or impossible to check all of them to ensure safe behaviour of the system

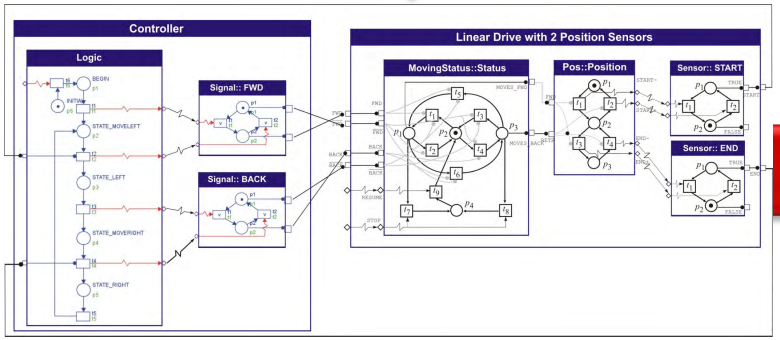


Formal verification tools can explore the complete state-space of the model

Formal Verification of CPS

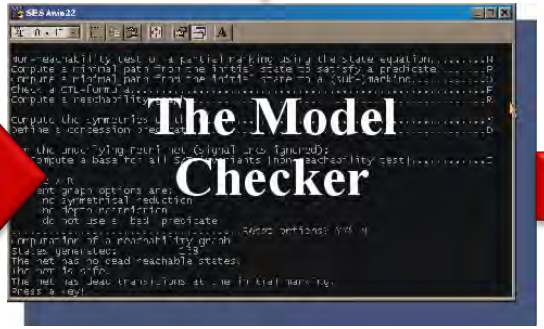


CPS

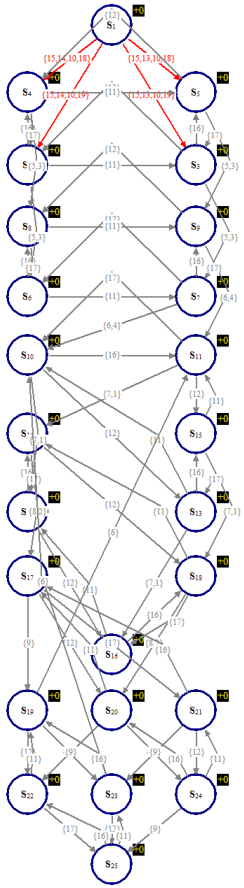


Closed-loop formal model of CPS

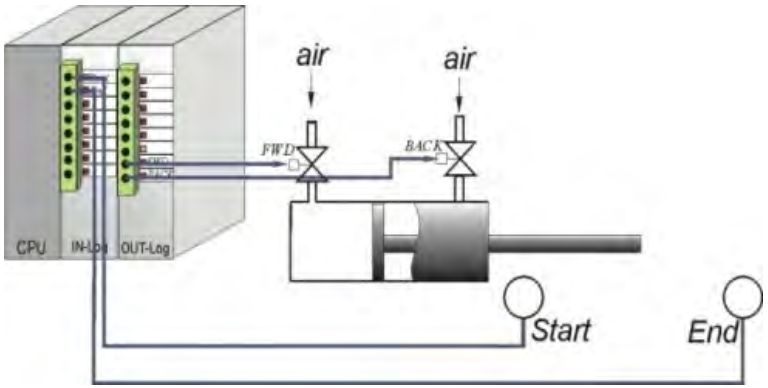
EF (BACK and FWD)

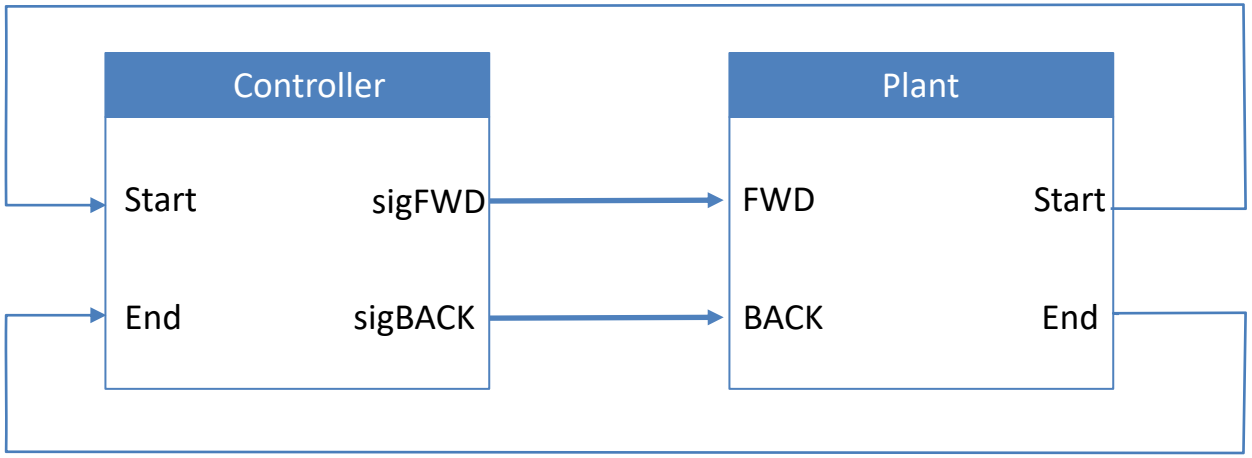


State space of the CPS

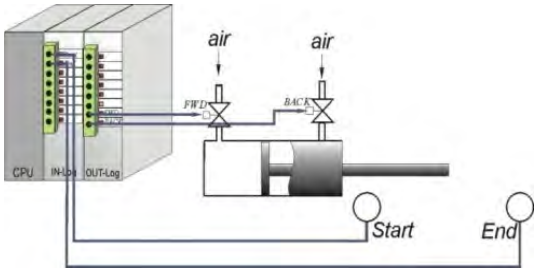


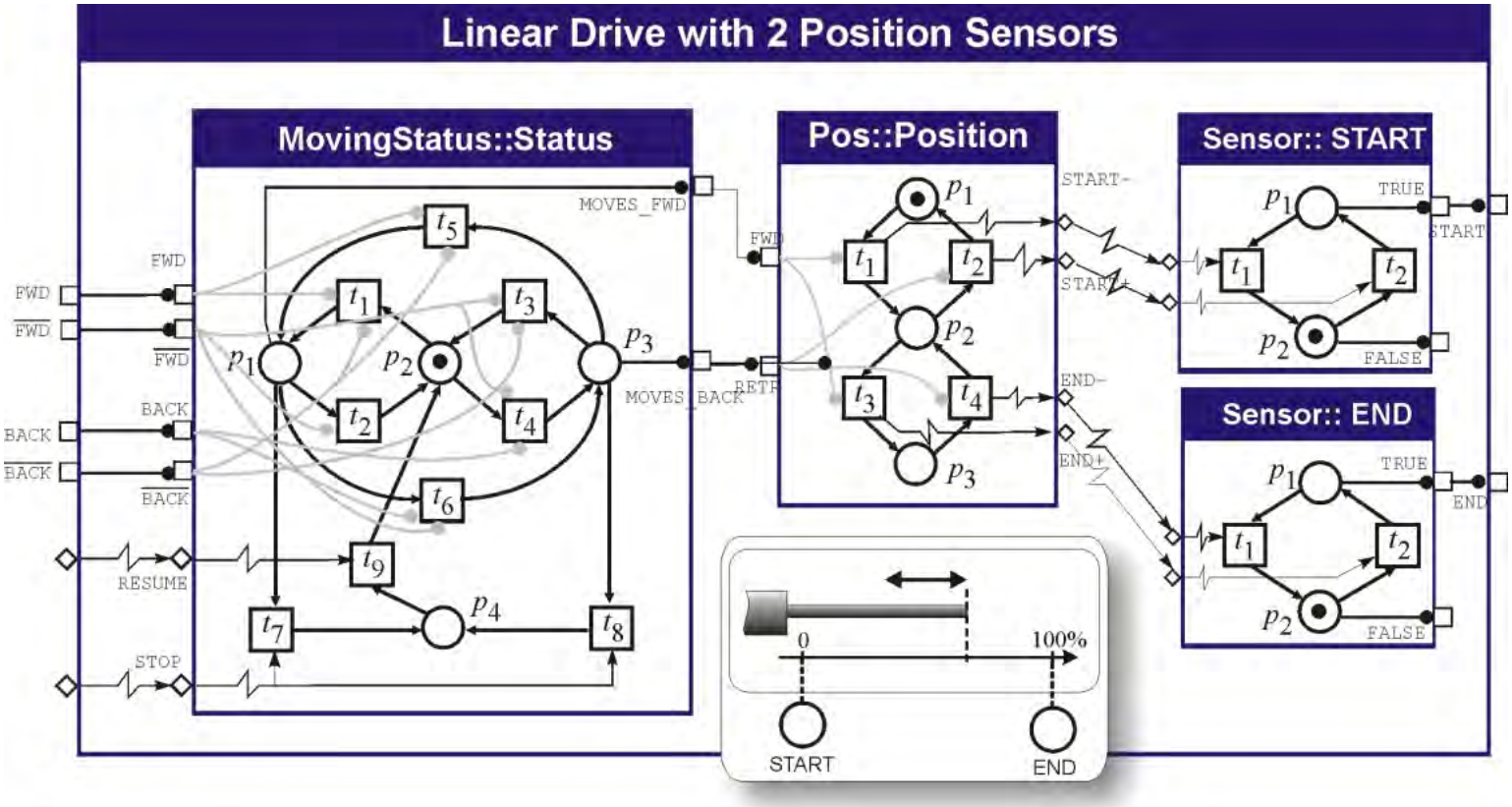
Closed – loop Modelling



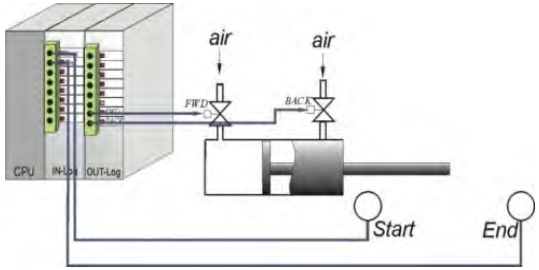


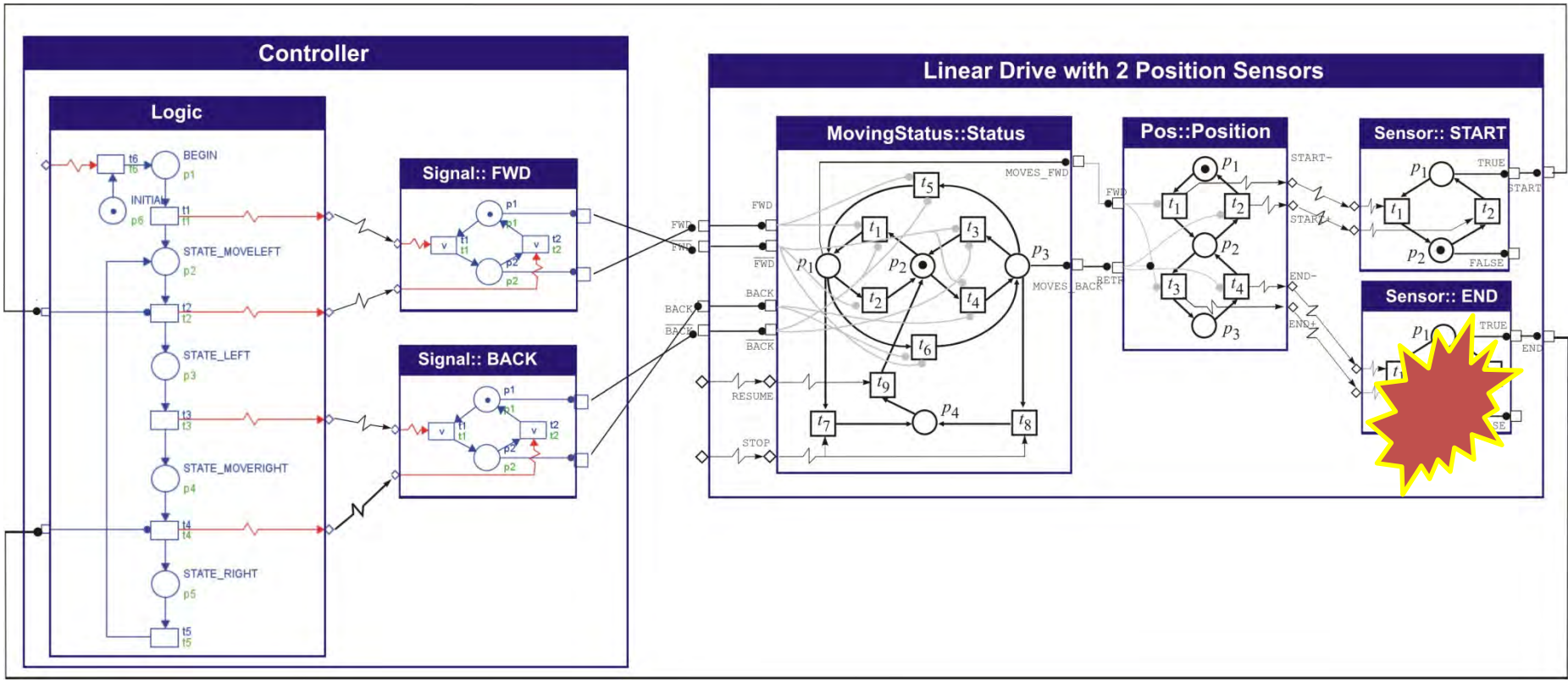
Model of Plant



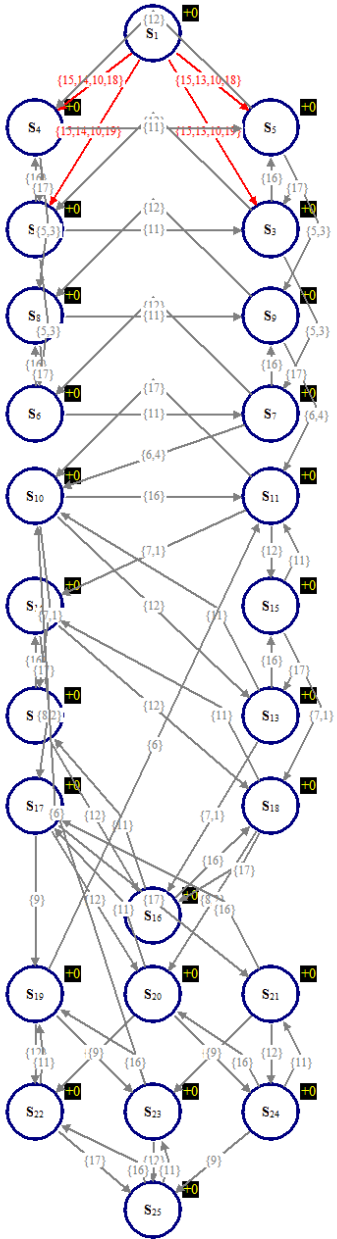


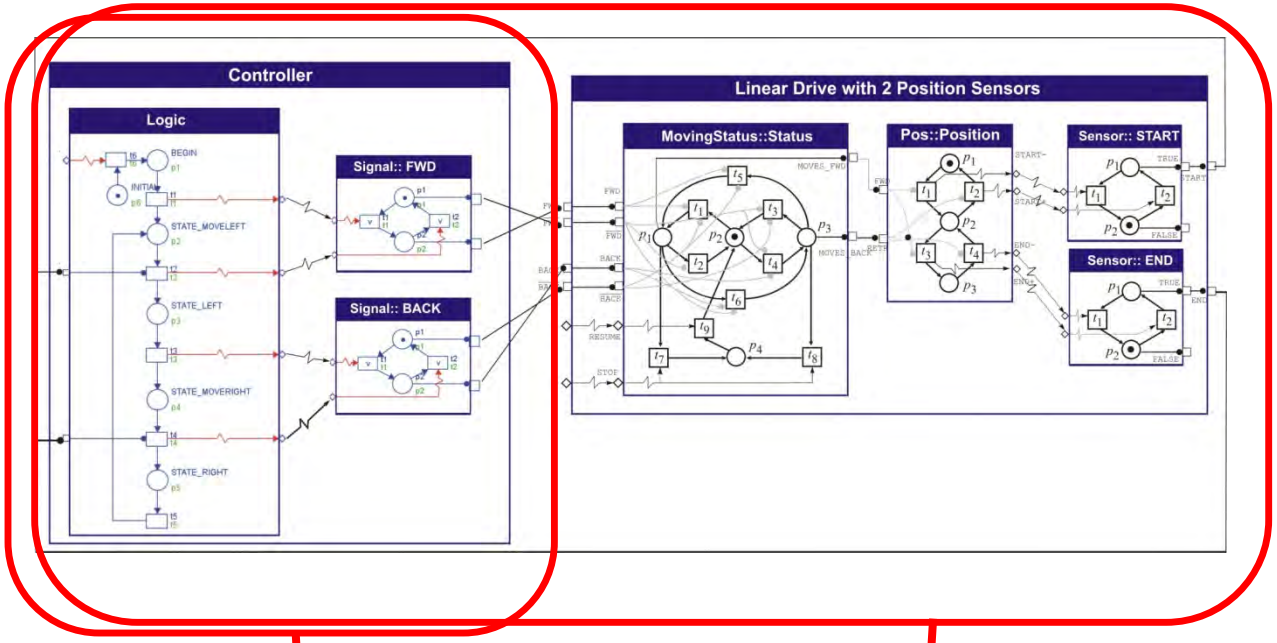
Closed-loop model in Net Condition-Event Systems

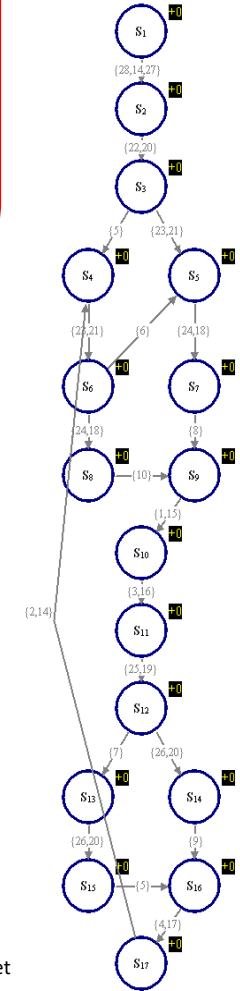




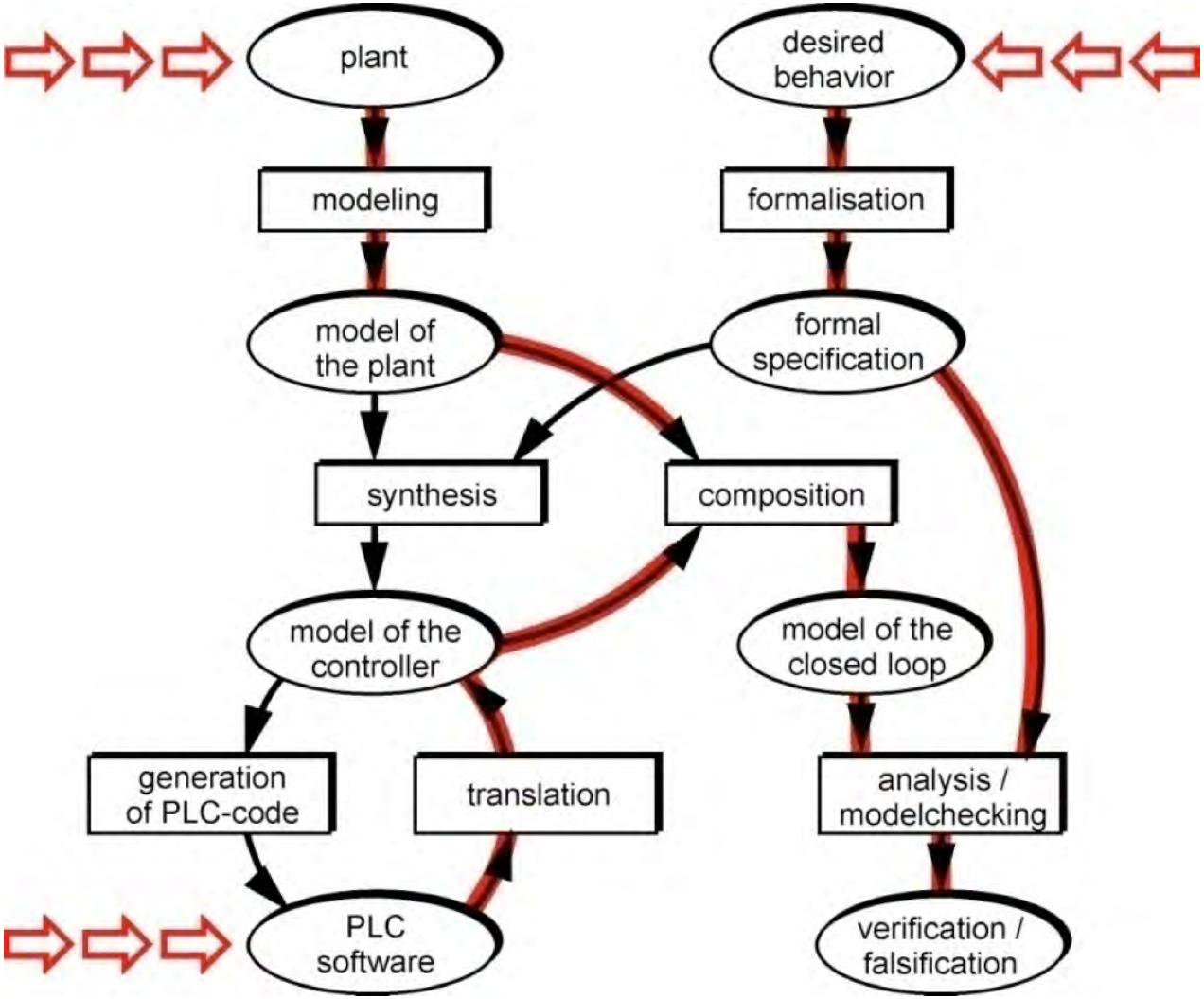
Complexity of Model vs. Complexity of Behaviour







Framework for Formal Methods in Automation (H.-M. Hanisch Diagram)



Challenges for Formal Verification

Who needs it?
- Only nuclear industry in Finland firmly requires it

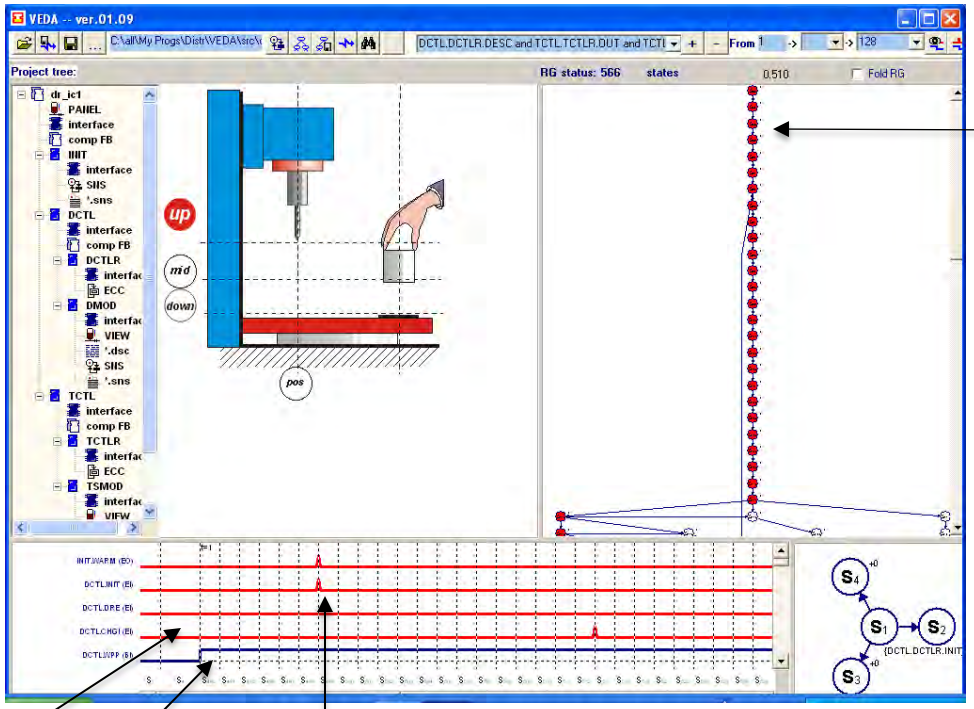
Complexity
- Of model-checking
- Of model-creation
- Symbolic model-checking of large models

User-friendliness
- Model-generation
- Requirements
- Interpretation of counter-examples
- Integration to the routines

Trust to models

…

Prototype of IDE with Formal Verification

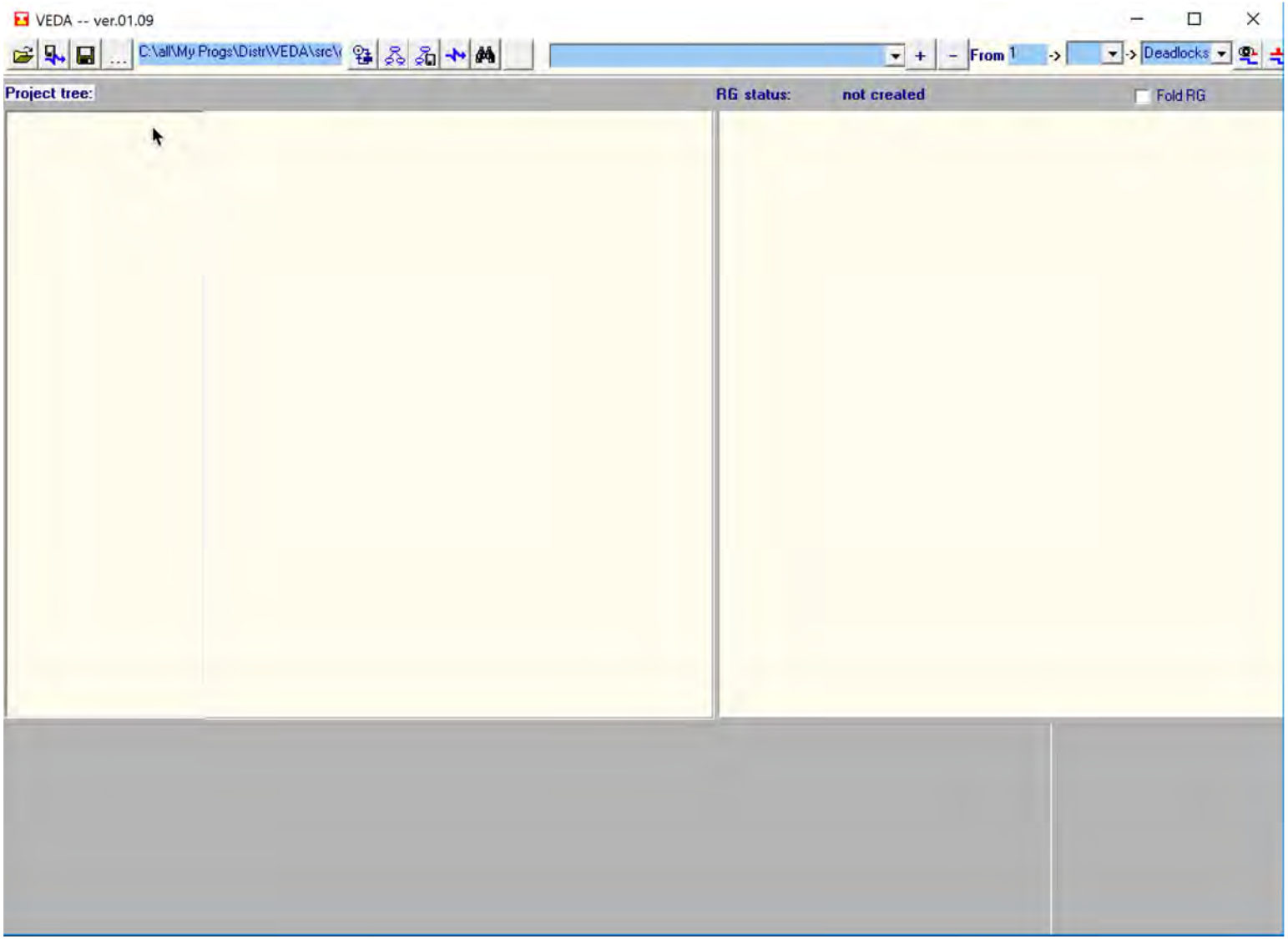


Reachability graph with a path to a state (counterexample) highlighted

Some state transitions have a non-zero time duration (plant), while others have no duration (controller).

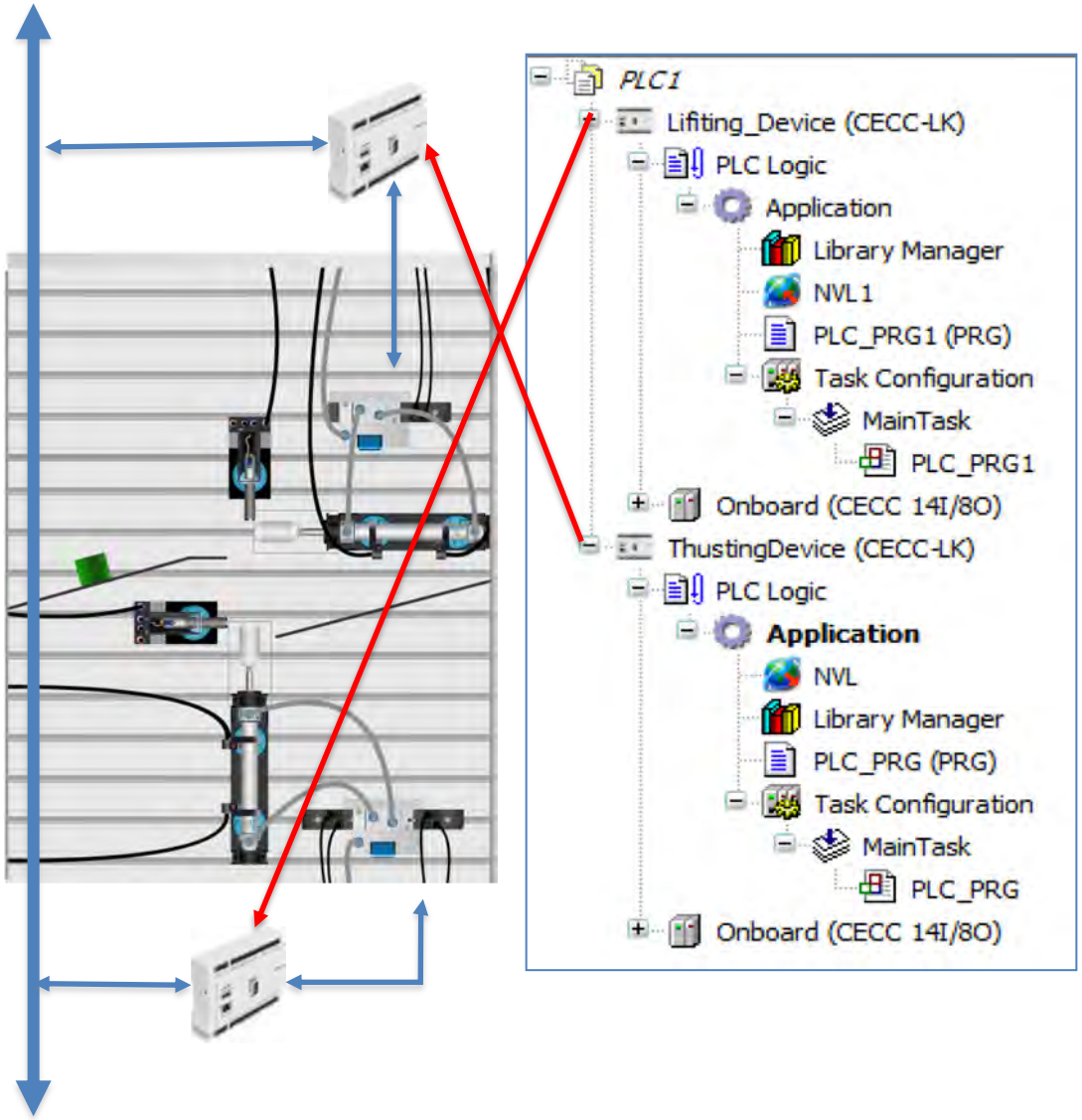
Each operation in controller corresponds to one state transition

VEDA in work


VEDA -- ver.01.09
C:\all\My Progs\Distr\VEDA\src\
From 1
Deadlocks
Project tree:
RG status:
not created
Fold RG

New Challenges: Distributed Systems of PLCs

Ethernet




```
PROGRAM PLC_PRG1
VAR
    LCExtended : BOOL;
    LCRetracted: BOOL;
    LuggageArrived: BOOL;
    ExtendLC: BOOL;
    RetractLC: BOOL;
END_VAR
IF LuggageArrived AND  NOT LCExtended THEN
    ExtendLC := TRUE;
    RetractLC := FALSE;
ELSIF LCExtended AND SharedVariable THEN
    ExtendLC := FALSE;
    RetractLC := TRUE;
END_IF
```

Sensors

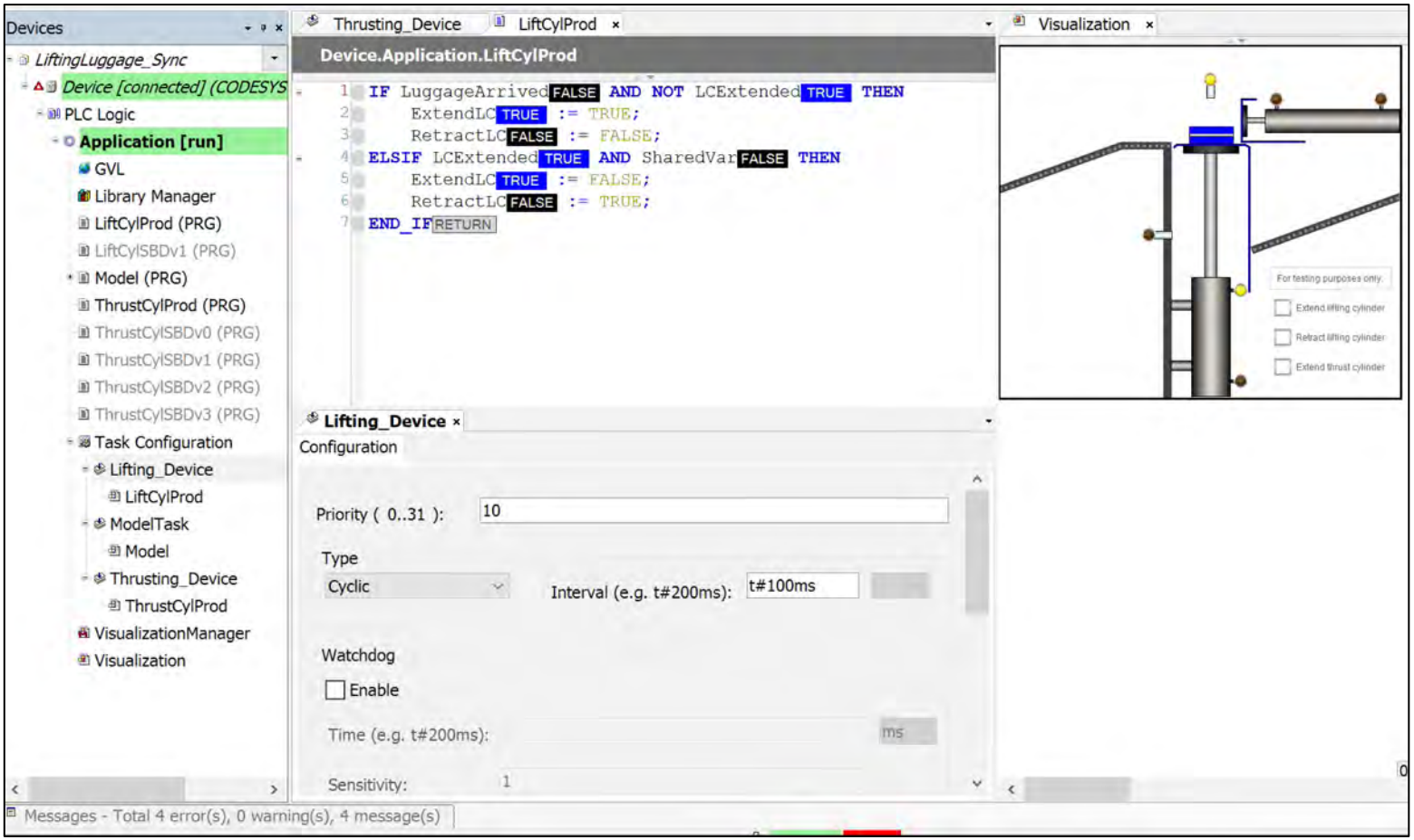
Actuator

```
PROGRAM PLC_PRG
VAR
    TCRetracted: BOOL;
    TCExtended: BOOL;
    LuggageRaised: BOOL;
    ExtendTC: BOOL;
    LuggageAway: BOOL;
END_VAR
IF LuggageRaised AND TCRetracted THEN
    ExtendTC:= TRUE;
    LuggageAway := FALSE;
    SharedVariable := FALSE;
ELSIF TCExtended AND NOT LuggageRaised THEN
    ExtendTC:= FALSE;
    SharedVariable := TRUE;
END_IF
```

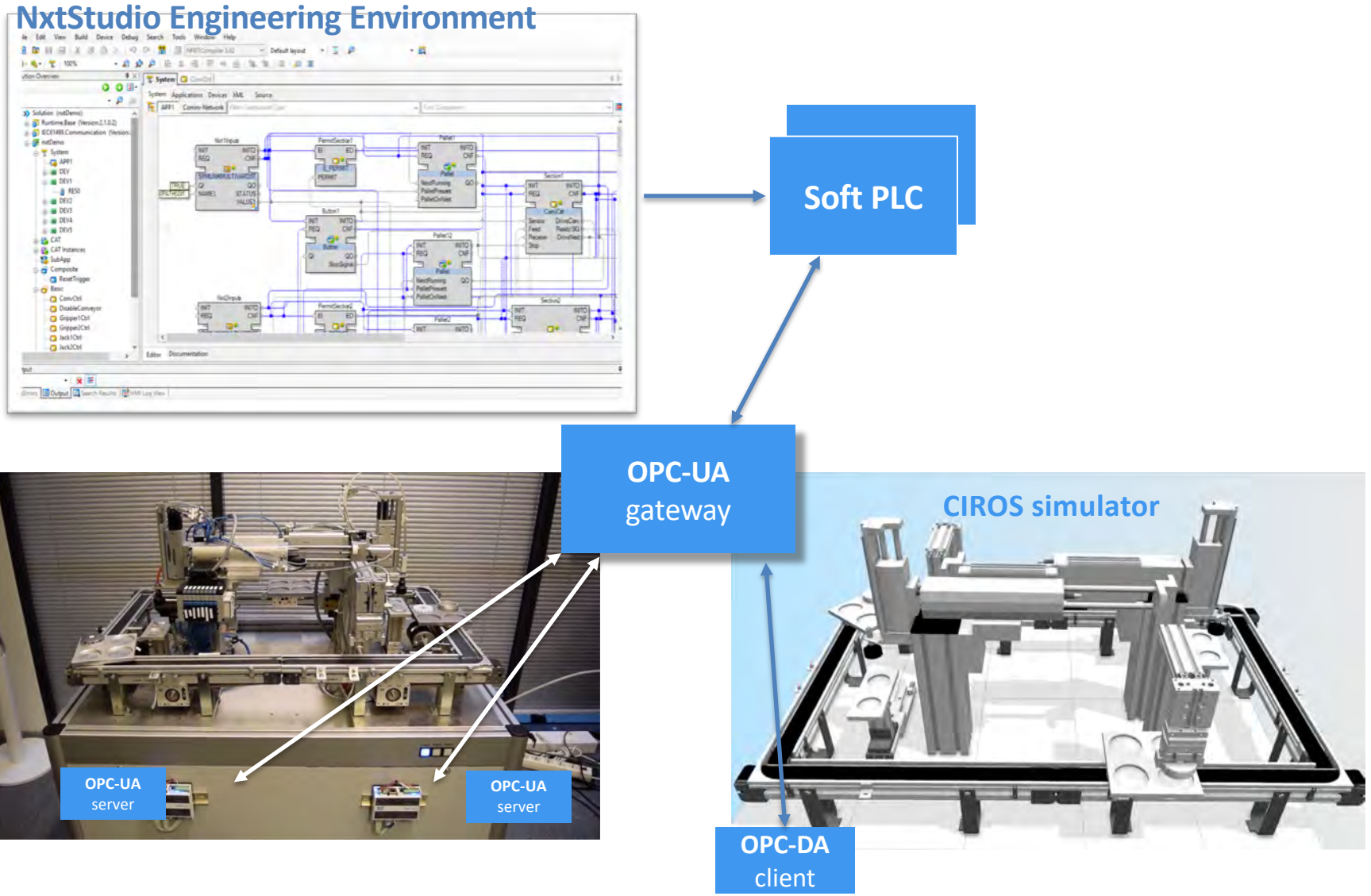
Sensors

Actuator

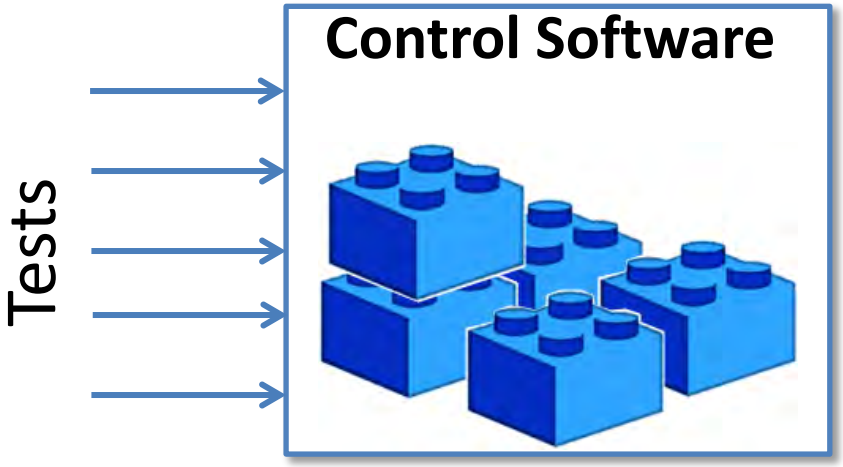
Distributed Systems of PLCs



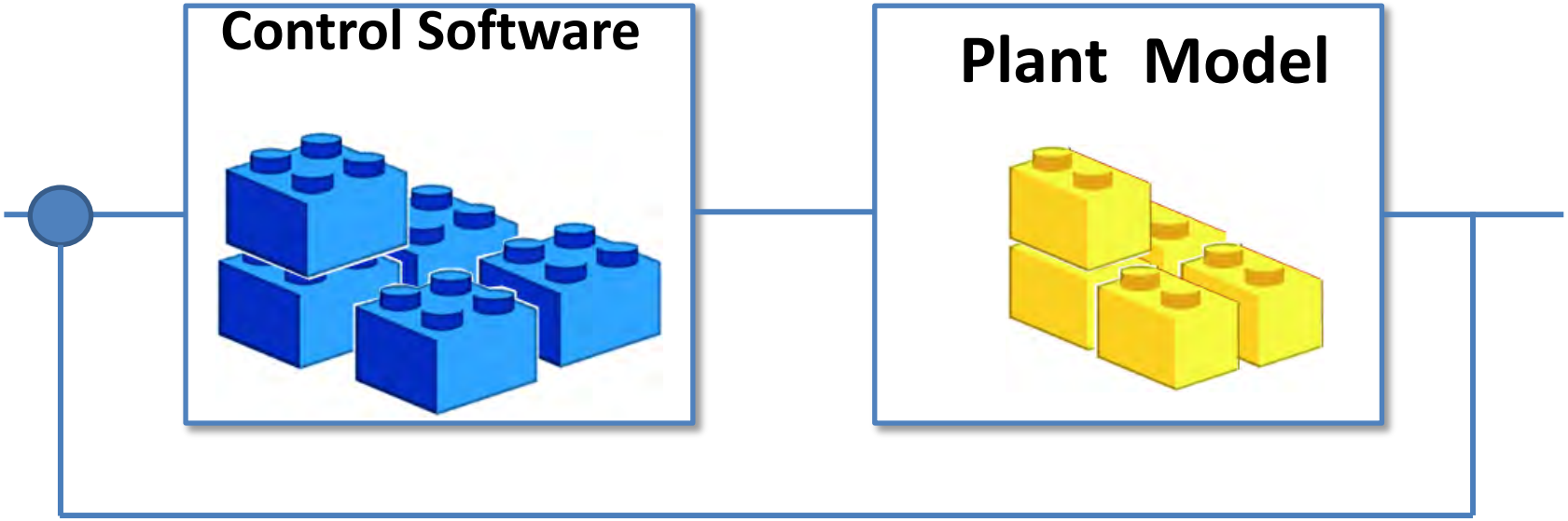
New Opportunities: Use of Digital Twin for Testing



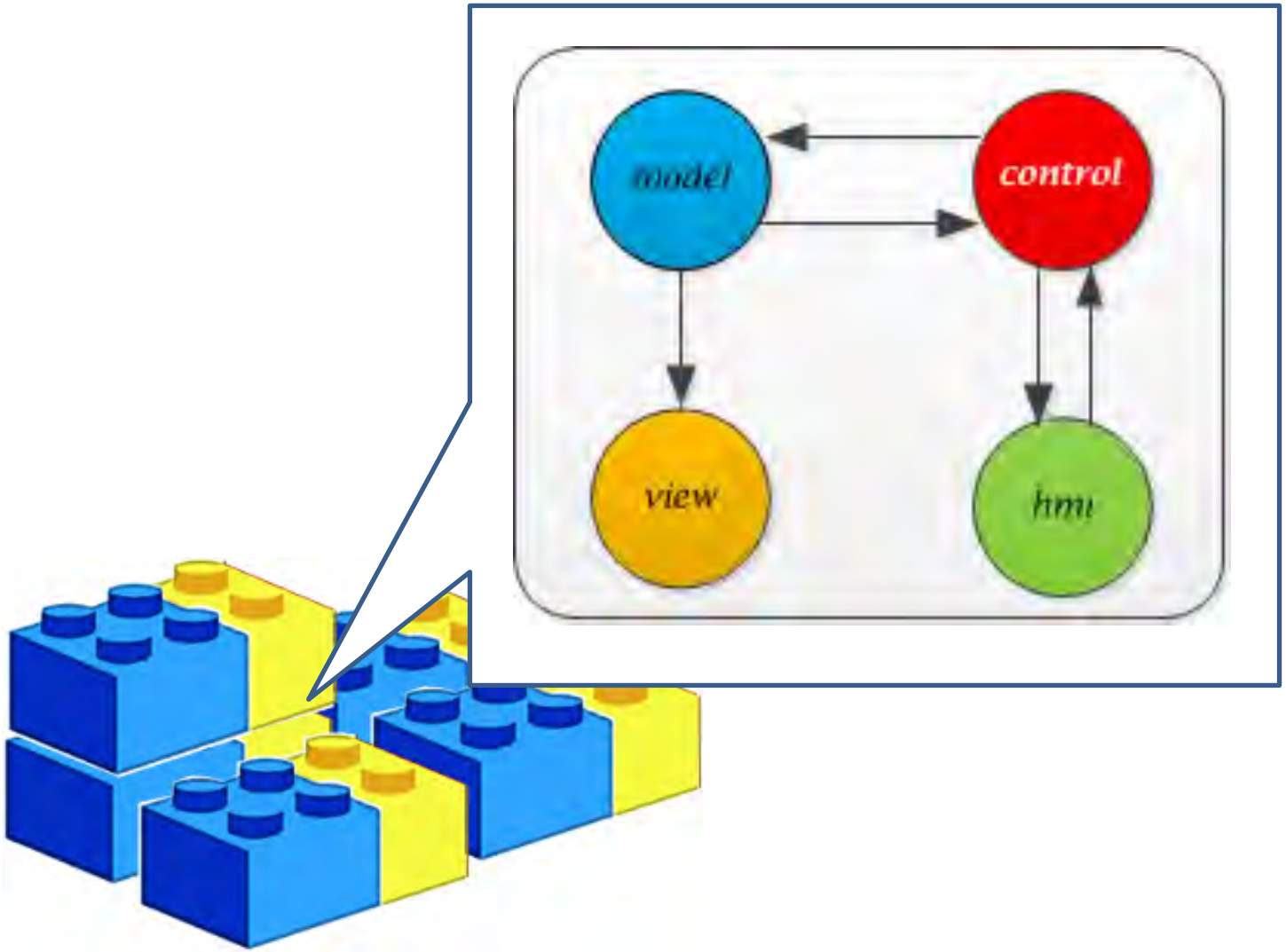
Testing of Component-based Systems



Traditional Control Engineering Approach: Closed-Loop

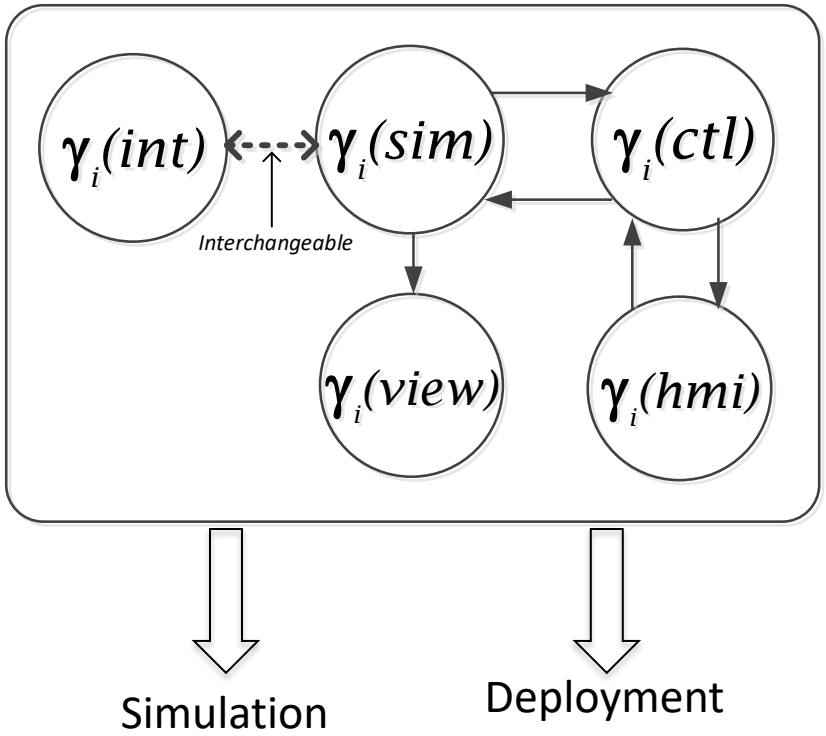


Cyber-Physical Component Architecture

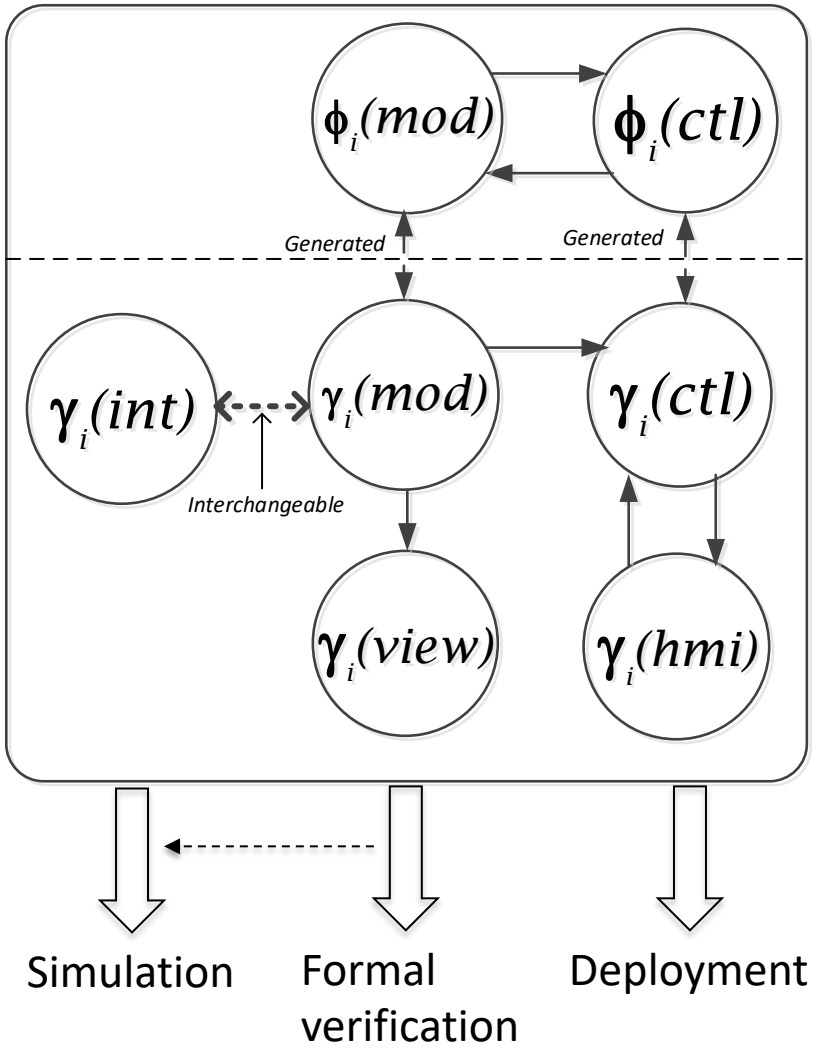


From simulation to formal verification and back

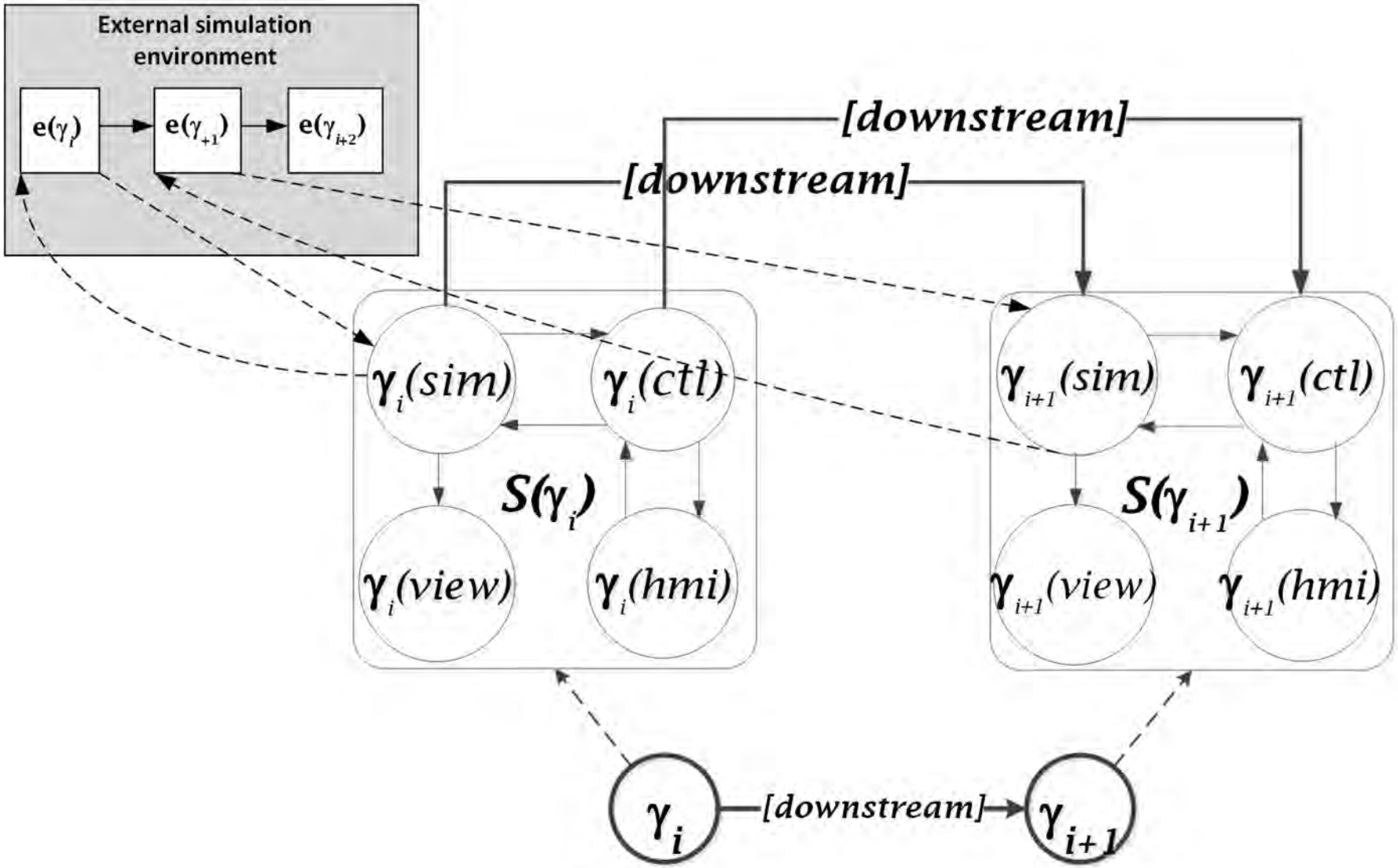
CPCA



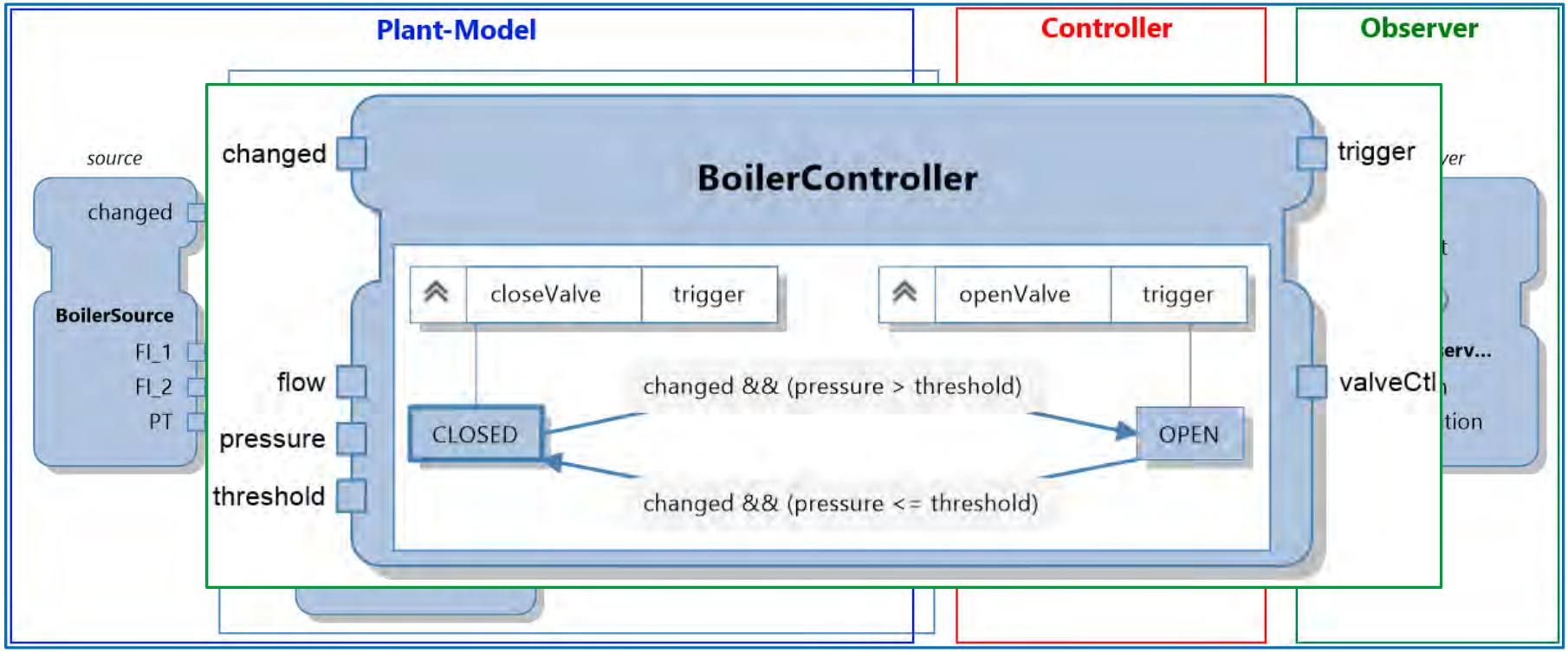
CPCA extended with formal model



Integration of CPCs

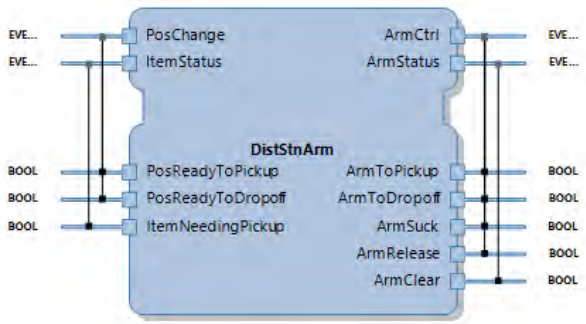


Distributed Component Architecture of IEC 61499



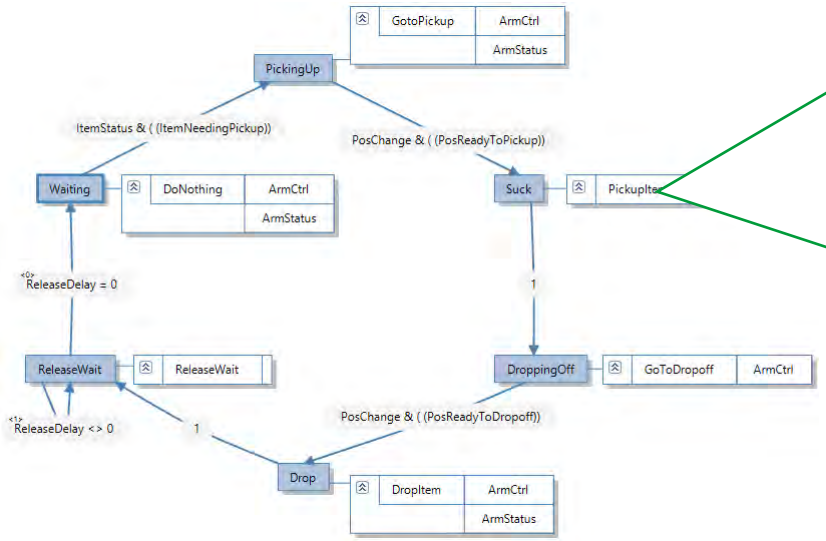
Function block

Function Block Interface



Function Block Interface explicitly declares input/output events and variables of a function block.

State machine implementation

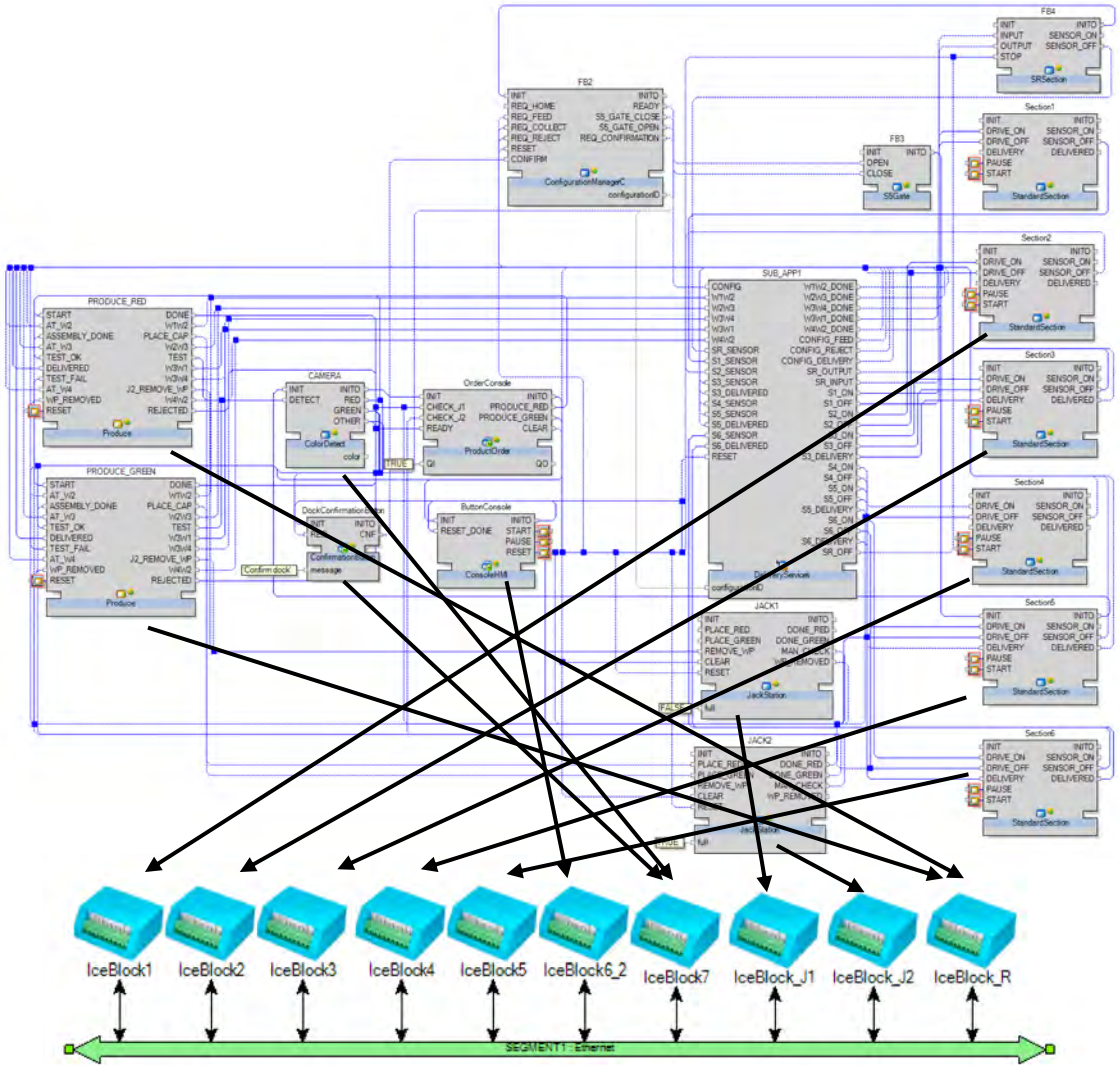


Behavior of a Basic Function Block is implemented by an execution control chart. Textual algorithms can be invoked upon entering a state.

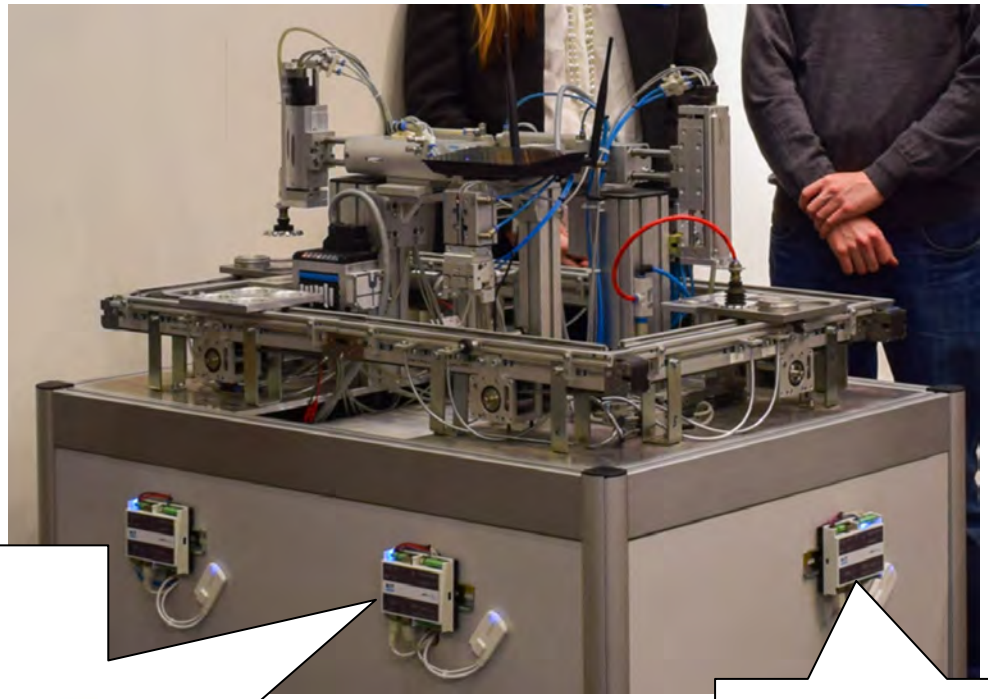
Legacy code

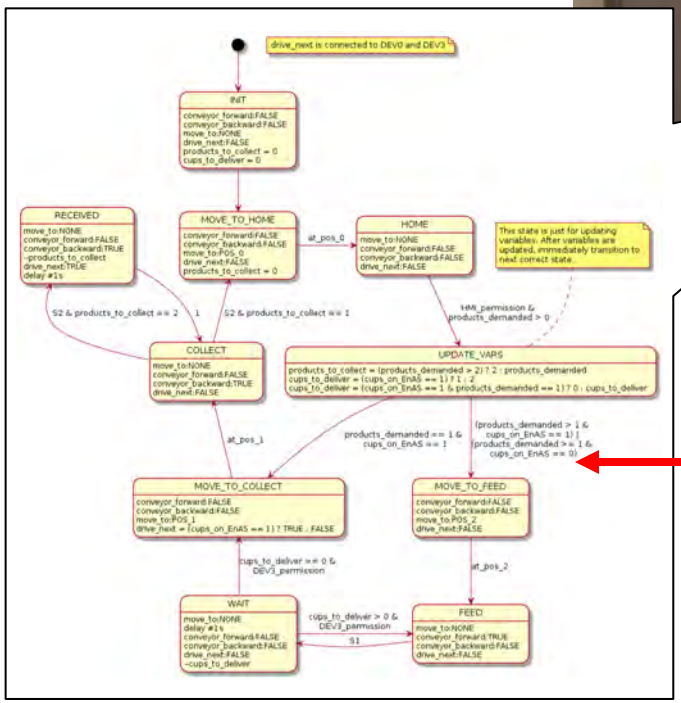
```
IF state_door = state_closed THEN
    IF doOpen0 THEN
        state_door := state_opening_0;
    ELSIF doOpen1 THEN
        state_door := state_opening_1;
    ELSIF doOpen2 THEN
        state_door := state_opening_2;
    END_IF;
ELSIF state_door = state_opening_0 THEN
    open0:=TRUE;
    IF NOT doorClosed0 THEN
        state_door:=state_open;
    END_IF;
ELSIF state_door = state_opening_1 THEN
    open1:=TRUE;
    IF NOT doorClosed1 THEN
        state_door:=state_open;
    END_IF;
ELSIF state_door = state_opening_2 THEN
    open2:=TRUE;
    IF NOT doorClosed2 THEN
        state_door:=state_open;
    END_IF;
ELSIF state_door = state_open THEN
    t1.IN:=TRUE;
    t1.PT:=T#3S;
    t1();
    IF t1.Q THEN
        t1.IN:=FALSE;
        t1();
        state_door:=state_closing;
    END_IF;
ELSIF state_door = state_closing THEN
    open0:=FALSE;
    open1:=FALSE;
    open2:=FALSE;
    IF allClosed THEN
        state_door:=state_closed;
    END_IF;
END_IF;
```

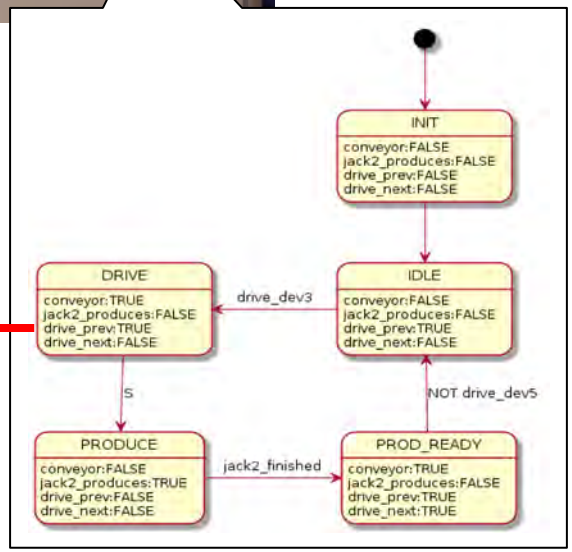
IEC61499: seamless distribution



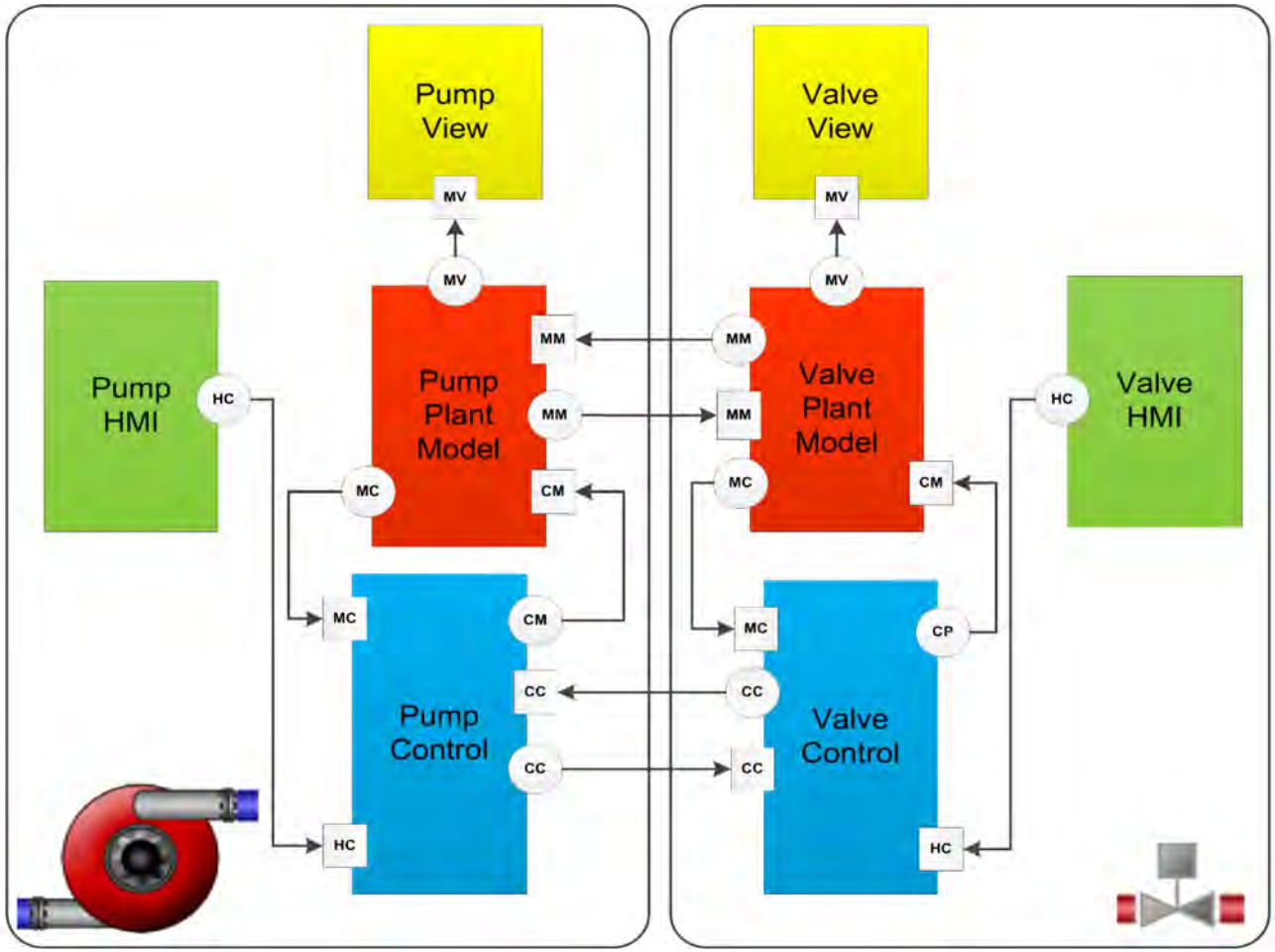
Communicating state-machines







Composition of CPCA



Why not the UML/SysML process?

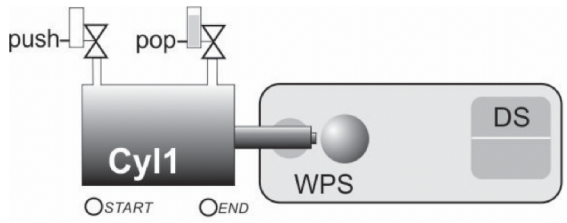
UML diagrams?
What ?

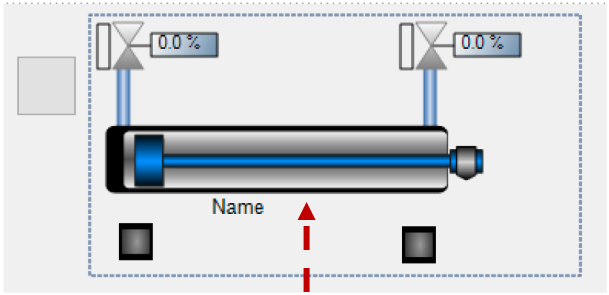
VDMA demonstrator

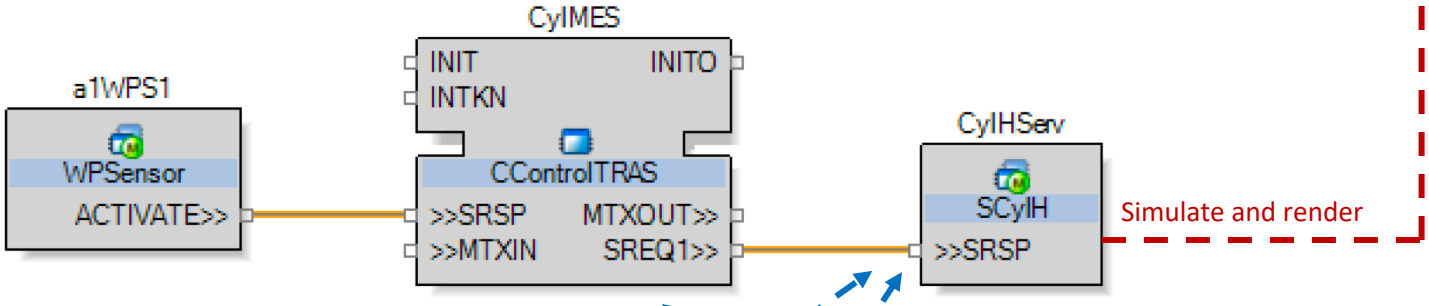
Integrates components of 25 vendors
High-level control: Communicating state machines connected via message passing
Transport: OPC-UA

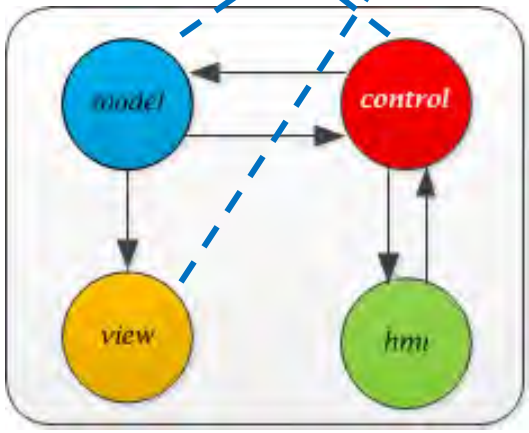
- https://www.youtube.com/watch?v=kT_3IHimNyc
- Same, but live: https://www.youtube.com/watch?v=QQwcIcrONMc

CPSC implemented with FBs

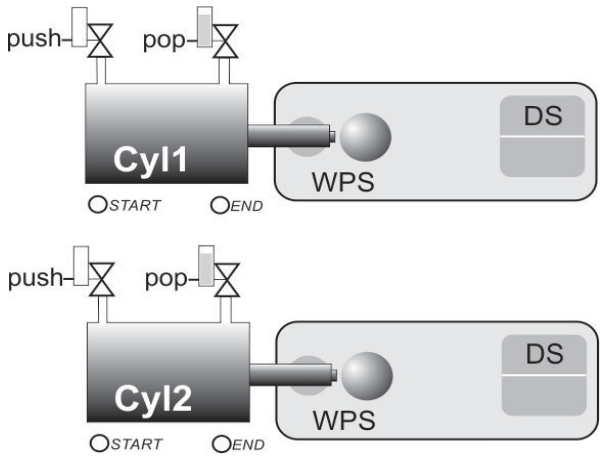


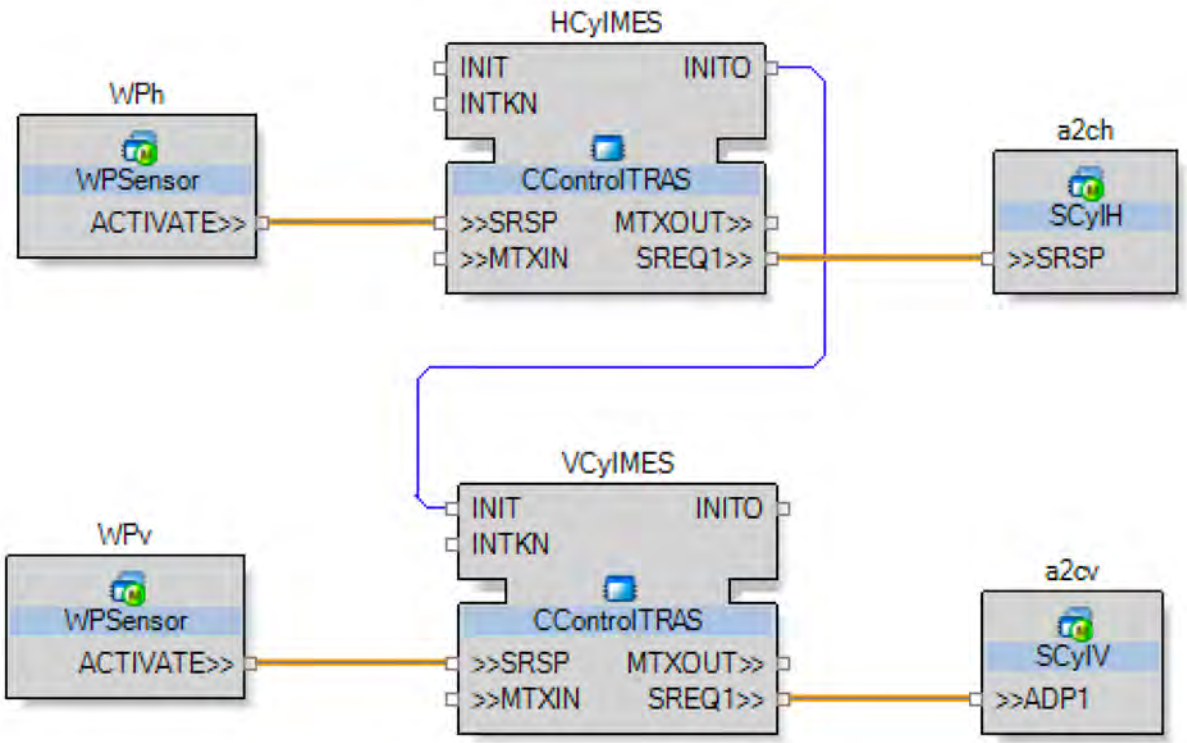




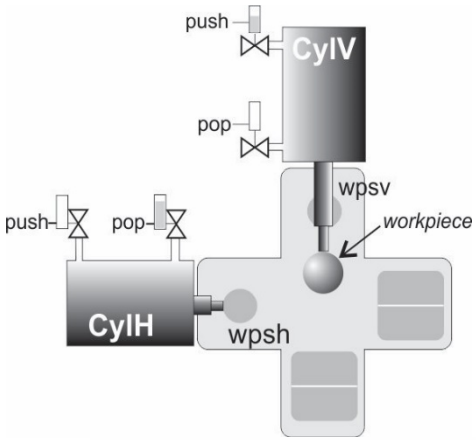


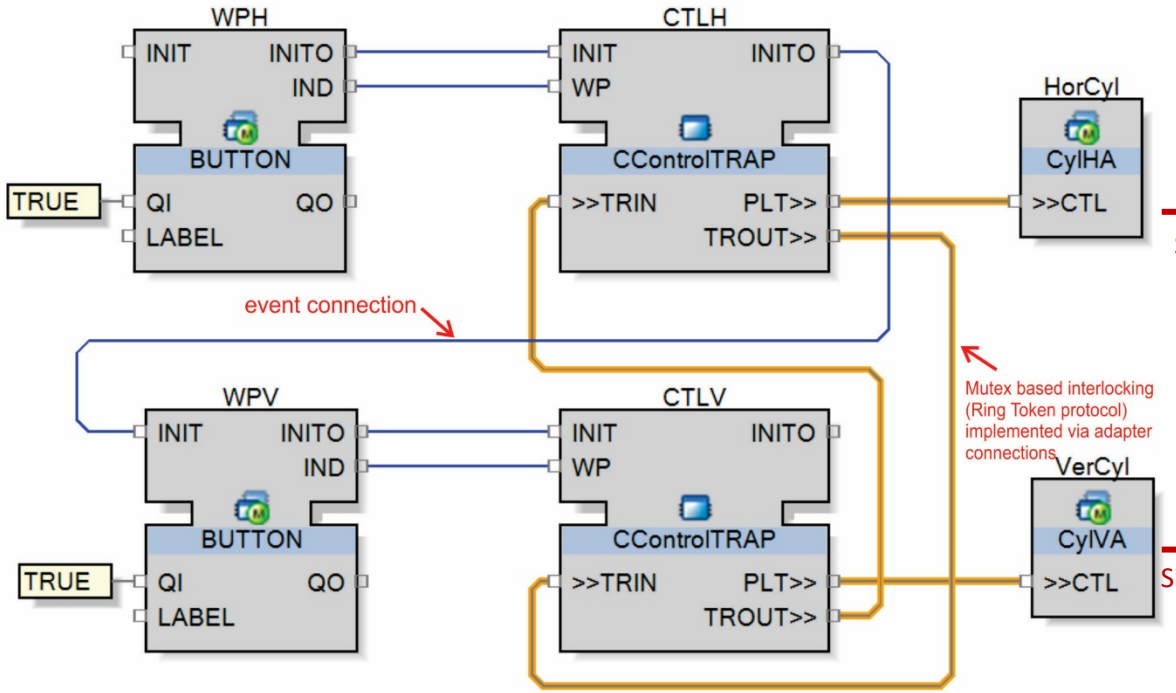
Two independent processes

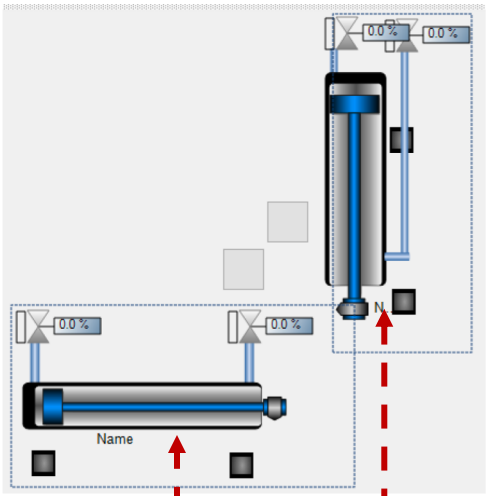




Intersecting cylinders: Mutual Exclusion



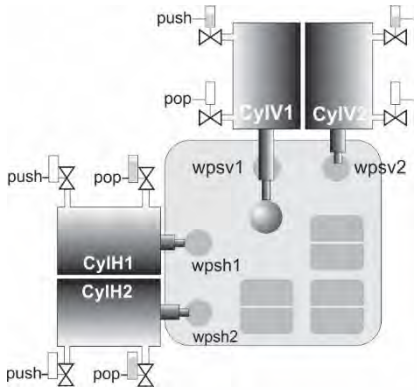


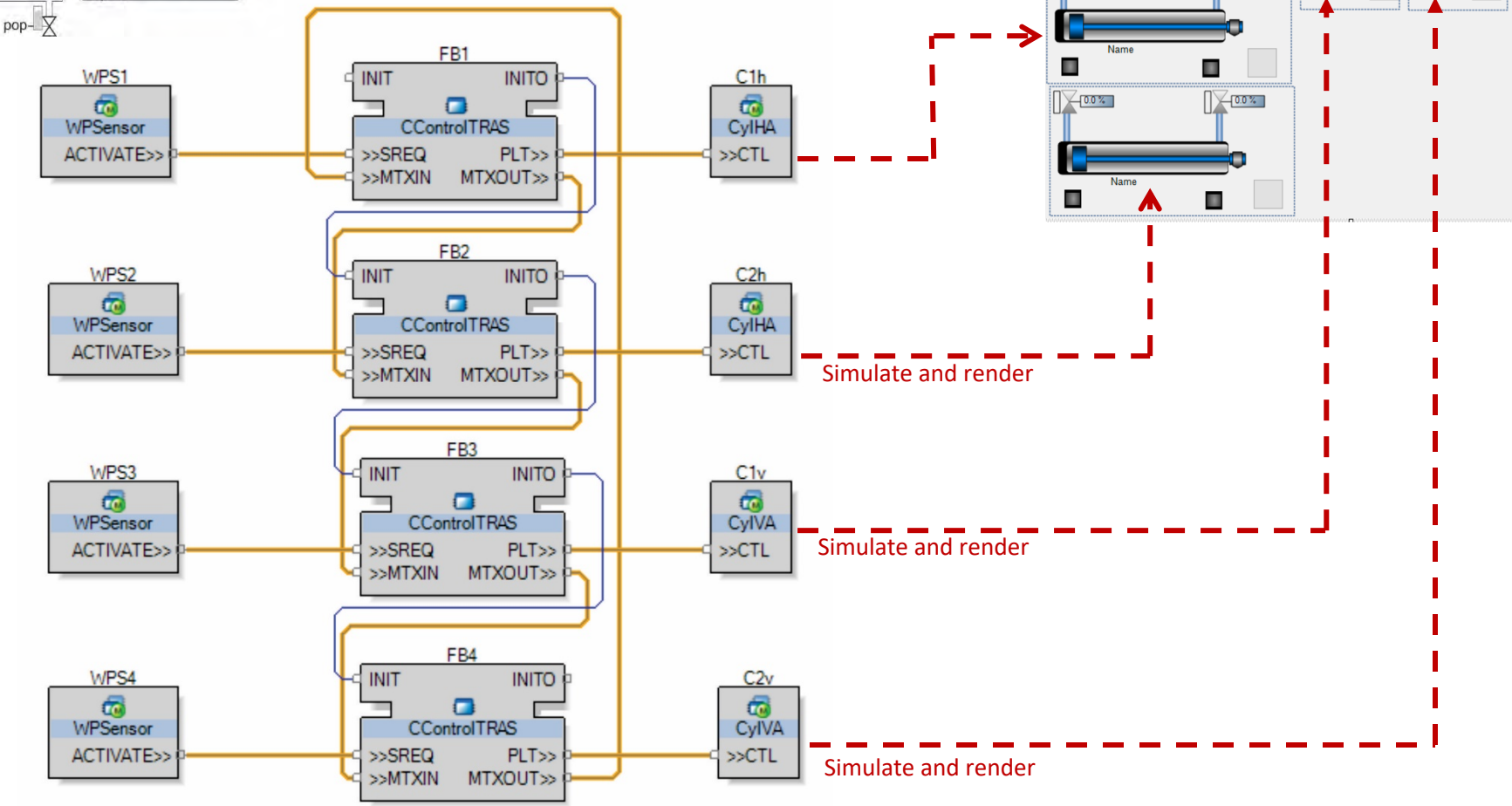


Simulate and render

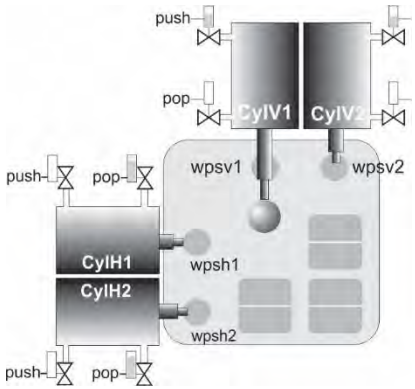
Simulate and render

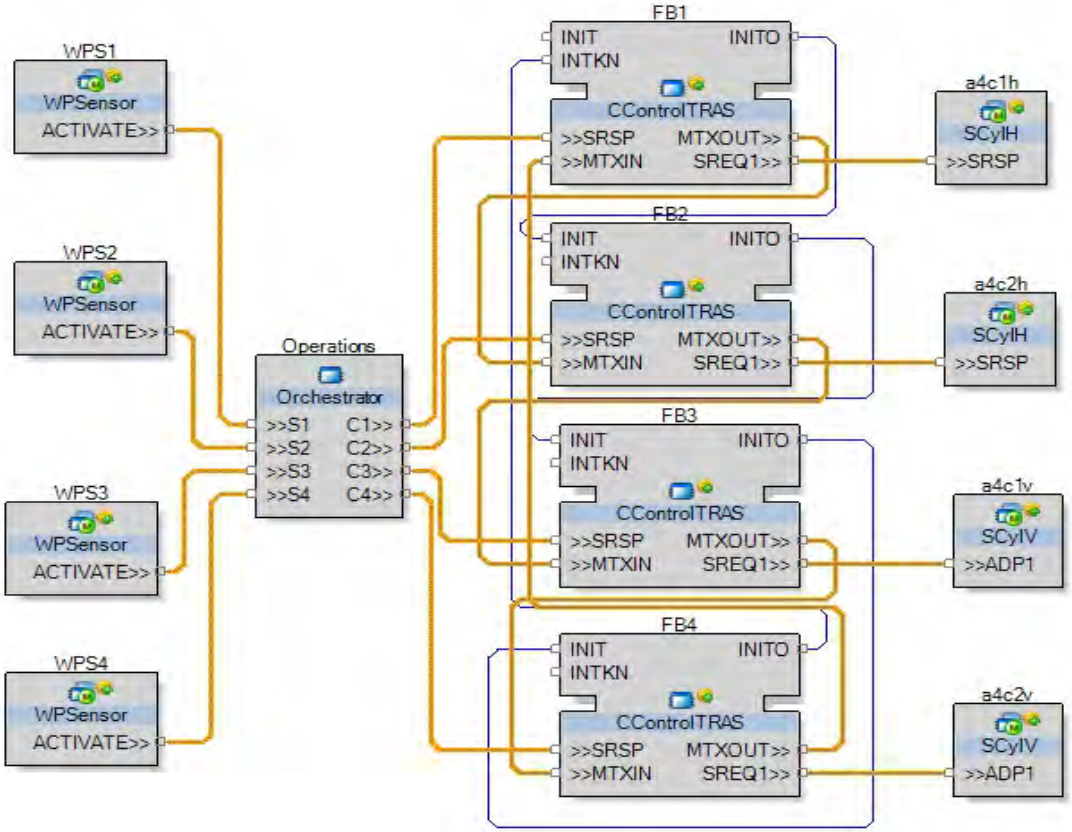
Four cylinders



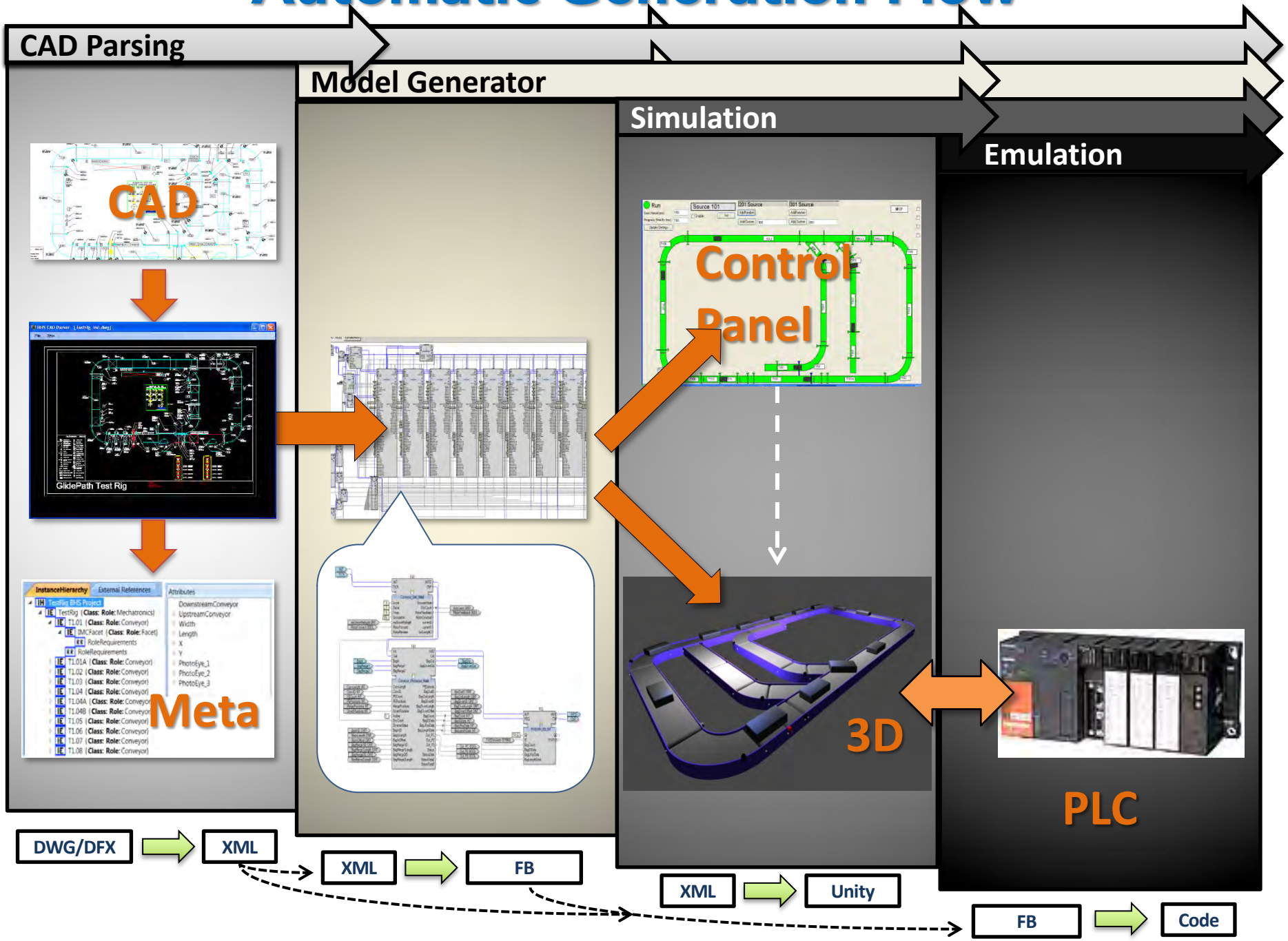


Central Orchestration

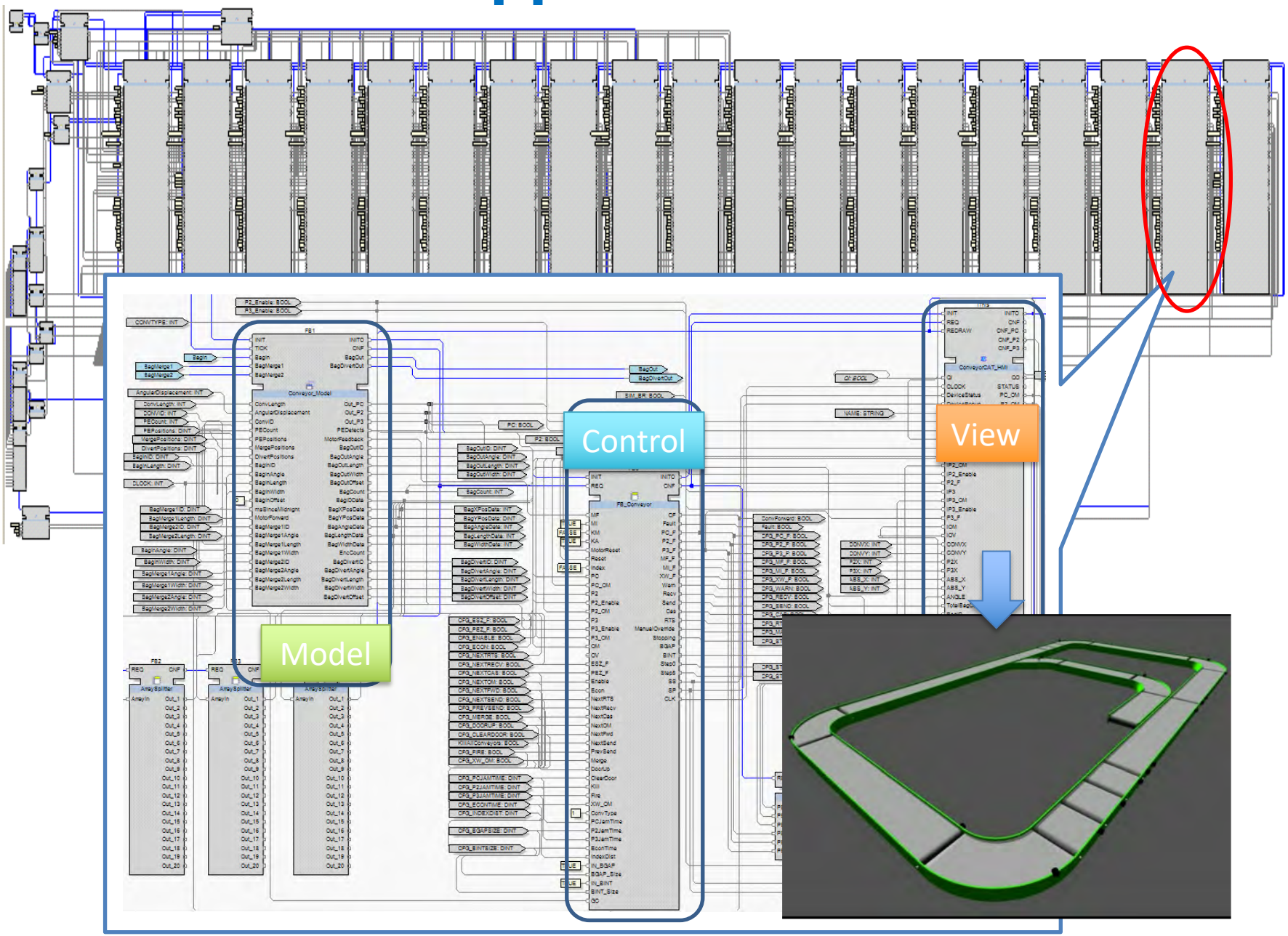




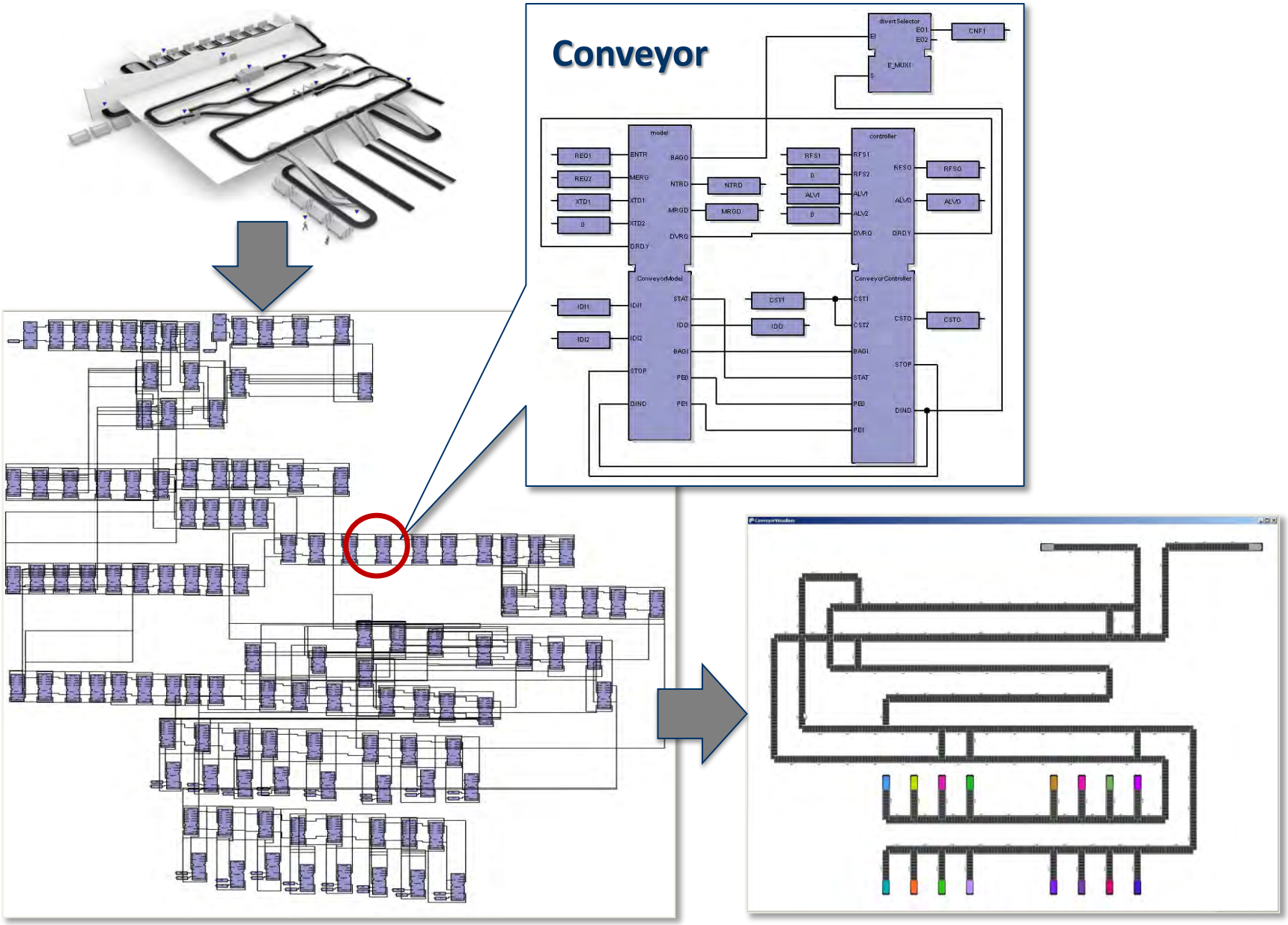
Automatic Generation Flow



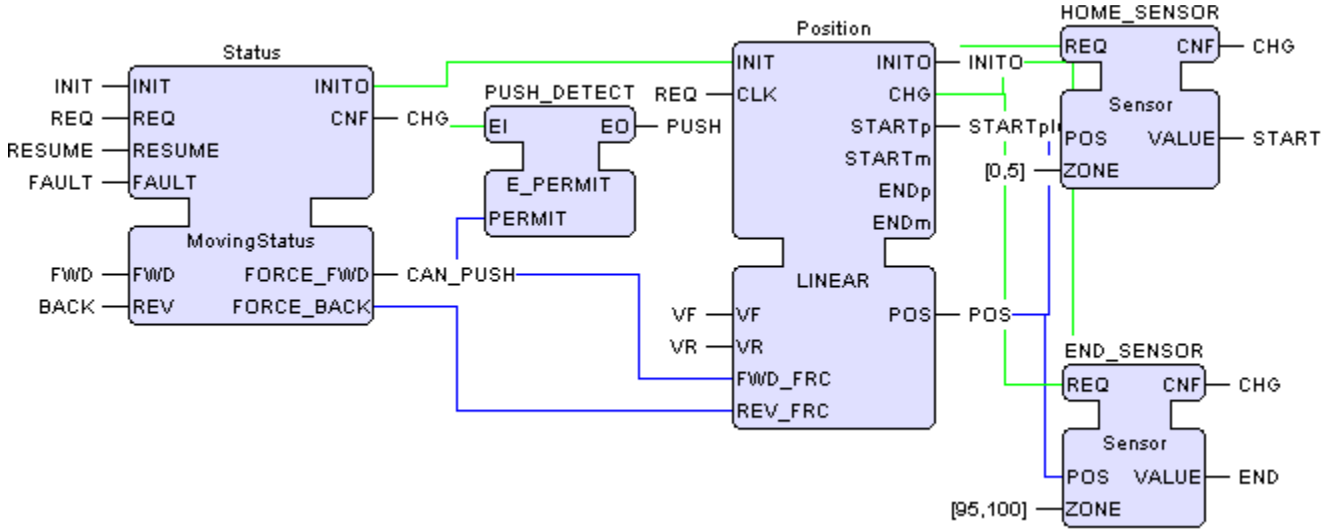
Generated FB Application

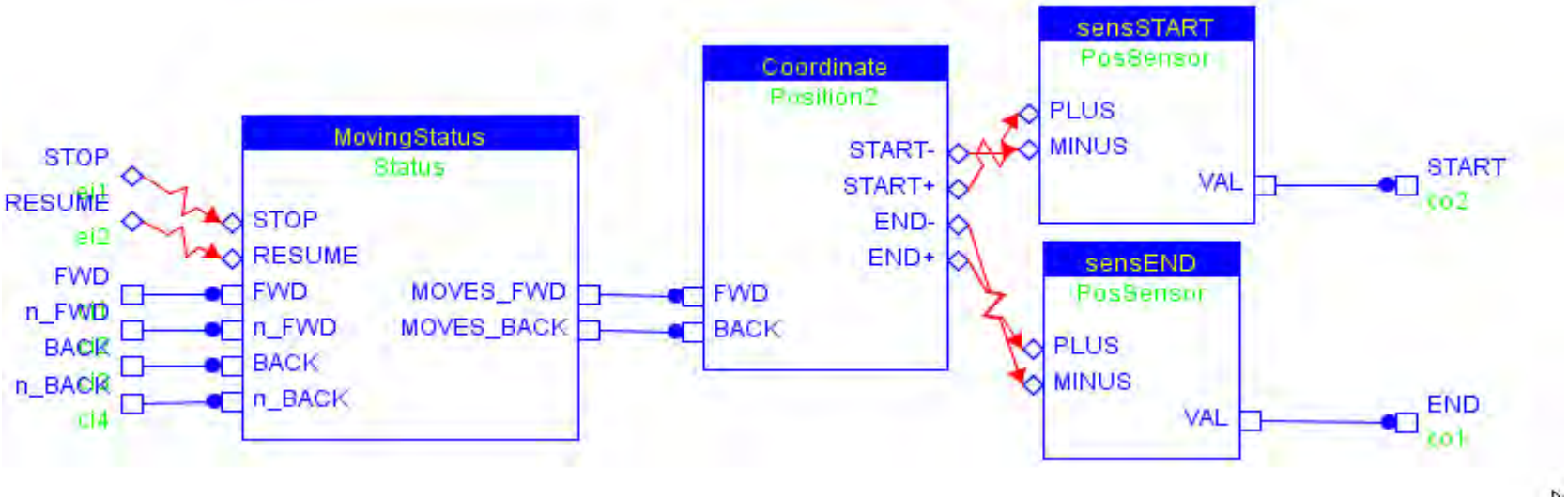


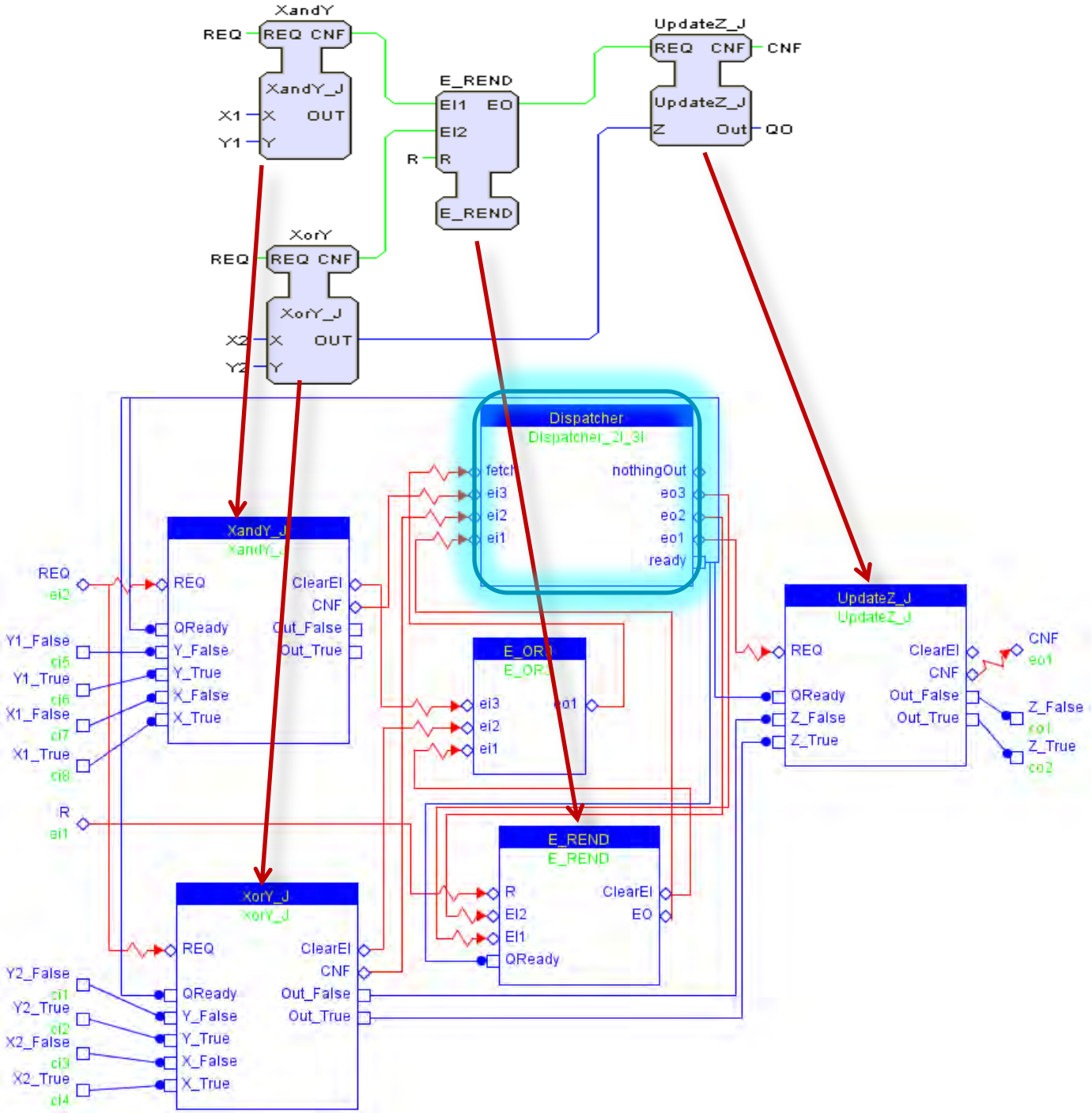
Scaling


Conveyor
divertSelector
CNF1
E_MUX1
model
controller
REQ1
REQ2
XTD1
0
ENTR
MERG
XTD1
XTD2
DRDY
BAGO
NTRD
MRGD
DVRQ
RFS1
0
ALVI
0
RFS1
RFS2
ALVI
ALVO
DVRQ
RFSO
ALVO
DRDY
ConveyorModel
ConveyorController
IDI1
IDI2
STOP
DIND
STAT
IDO
BAGI
PEO
PEI
CST1
IDO
CST1
CST2
BAGI
STAT
PEO
PEI
CSTO
STOP
DIND
CSTO

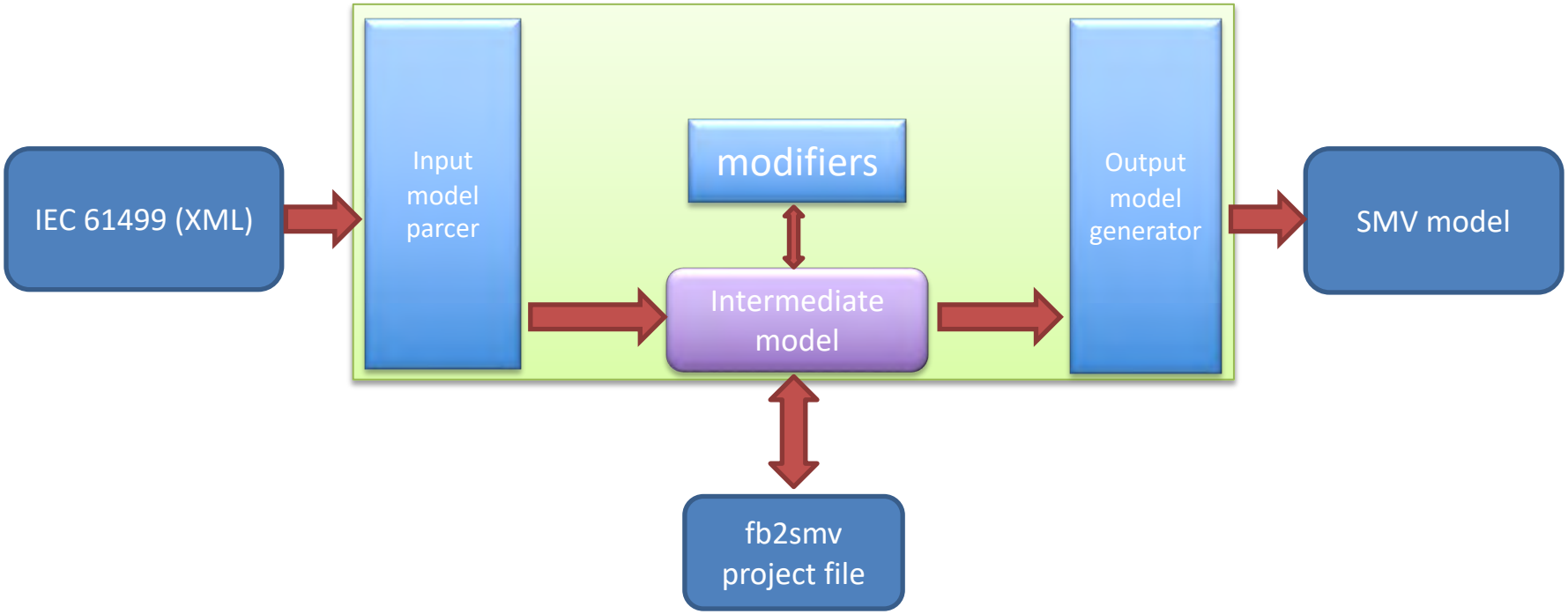
Executable model Formal model





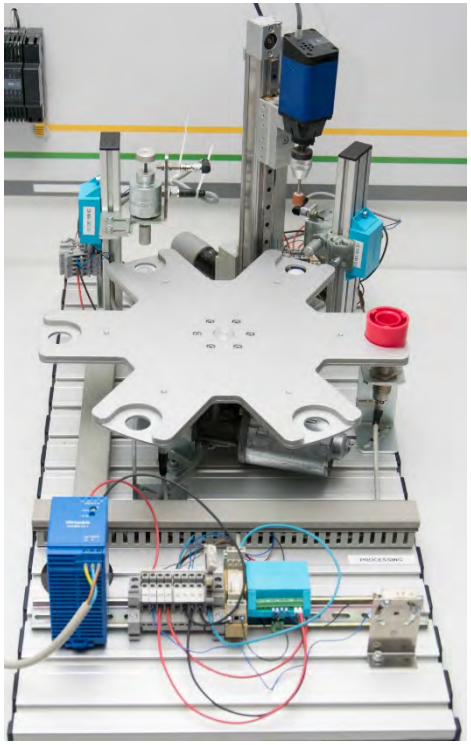

XandY
REQ
REQ CNF
XandY_J
X1
X
OUT
Y1
Y
E_REND
EI1
EO
EI2
R
R
E_REND
UpdateZ_J
REQ
CNF
CNF
UpdateZ_J
Z
Out
QO
XorY
REQ
REQ CNF
XorY_J
X2
X
OUT
Y2
Y
Dispatcher
Dispatcher_2I_3I
fetch
ei3
ei2
ei1
nothingOut
eo3
eo2
eo1
ready
XandY_J
REQ
ei2
REQ
ClearEI
CNF
QReady
Out_False
Y1_False
ci5
Y_False
Out_True
Y1_True
ci6
Y_True
X1_False
ci7
X_False
X1_True
ci8
X_True
R
ei1
E_OR1
E_OR
ei3
ei2
ei1
eo1
UpdateZ_J
UpdateZ_J
REQ
ClearEI
CNF
CNF
eo1
QReady
Out_False
Z_False
Z_False
co1
Z_True
Out_True
Z_True
co2
E_REND
E_REND
R
ClearEI
EI2
EO
EI1
QReady
XorY_J
XorY_J
REQ
ClearEI
CNF
Y2_False
ci1
QReady
Out_False
Y2_True
ci2
Y_False
Out_True
X2_False
ci3
Y_True
X2_True
ci4
X_False
X_True

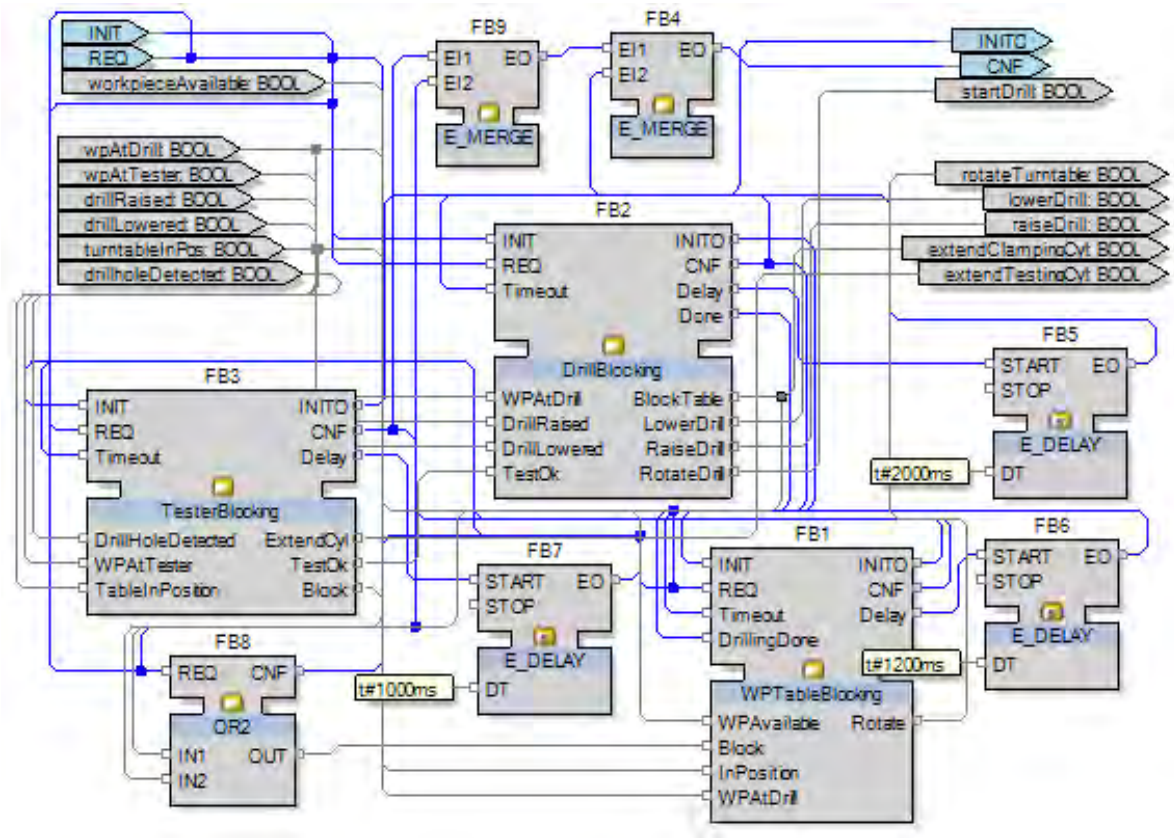
Tool: fb2smv



C# .NET 4.0 / MSVS 2015

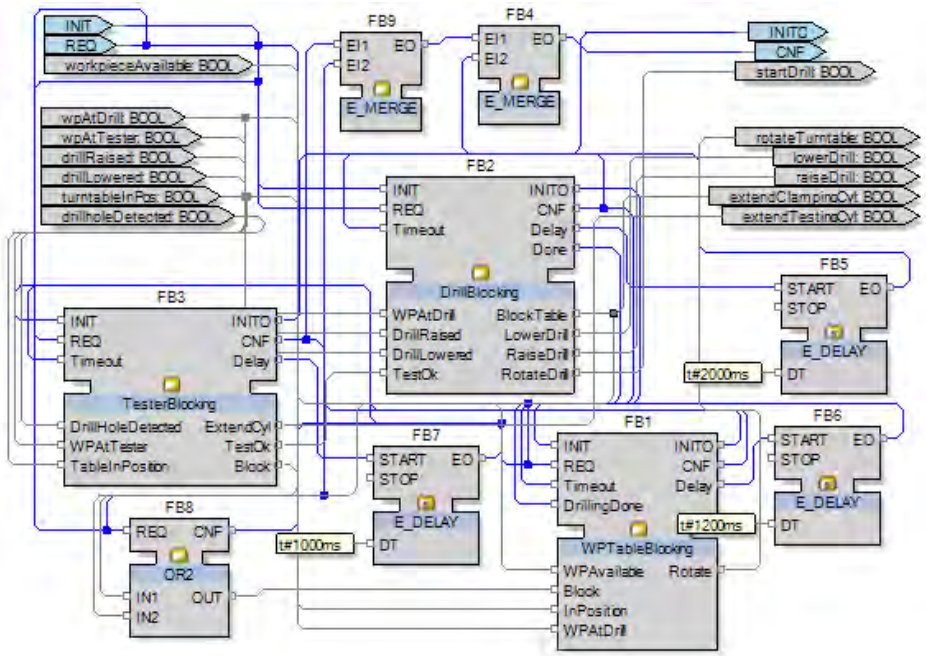
Case study: FESTO MPS 500 processing station



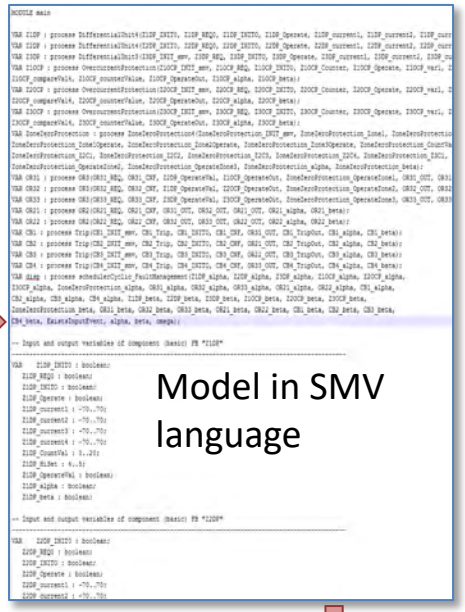


State	-1.336	-1.337	-1.338	-1.350	-1.351	-1.352
AddWorkpiece	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
DGmin	500	500	200	200	200	0
Drill.State	AtTop	AtTop	AtTop	AtTop	AtTop	AtTop
Drill_Di	500	0	500	500	300	300
Drill_Do	500	500	500	500	500	300
rotateTurntable	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
testerExtend	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
TurnTable_Di	500	0	500	500	300	300
TurnTable_Do	500	500	500	500	500	300

Model-checking with fb2smv



FB2SMV Generator



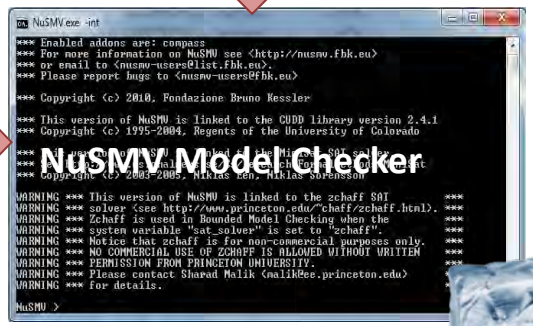
Model in SMV language

Property of the system (IFM) to safely cope with unpredicted changes, e.g.

- Equipment failure (IFM devices, AMU, communication, ROB3 ROB4, ROB5)
- Invalid/unknown inputs (sampling data, ROB3)
- Unexpected disturbances in the system (ROB1, ROB2)
- Intentional attacks (ROB3, ROB4)

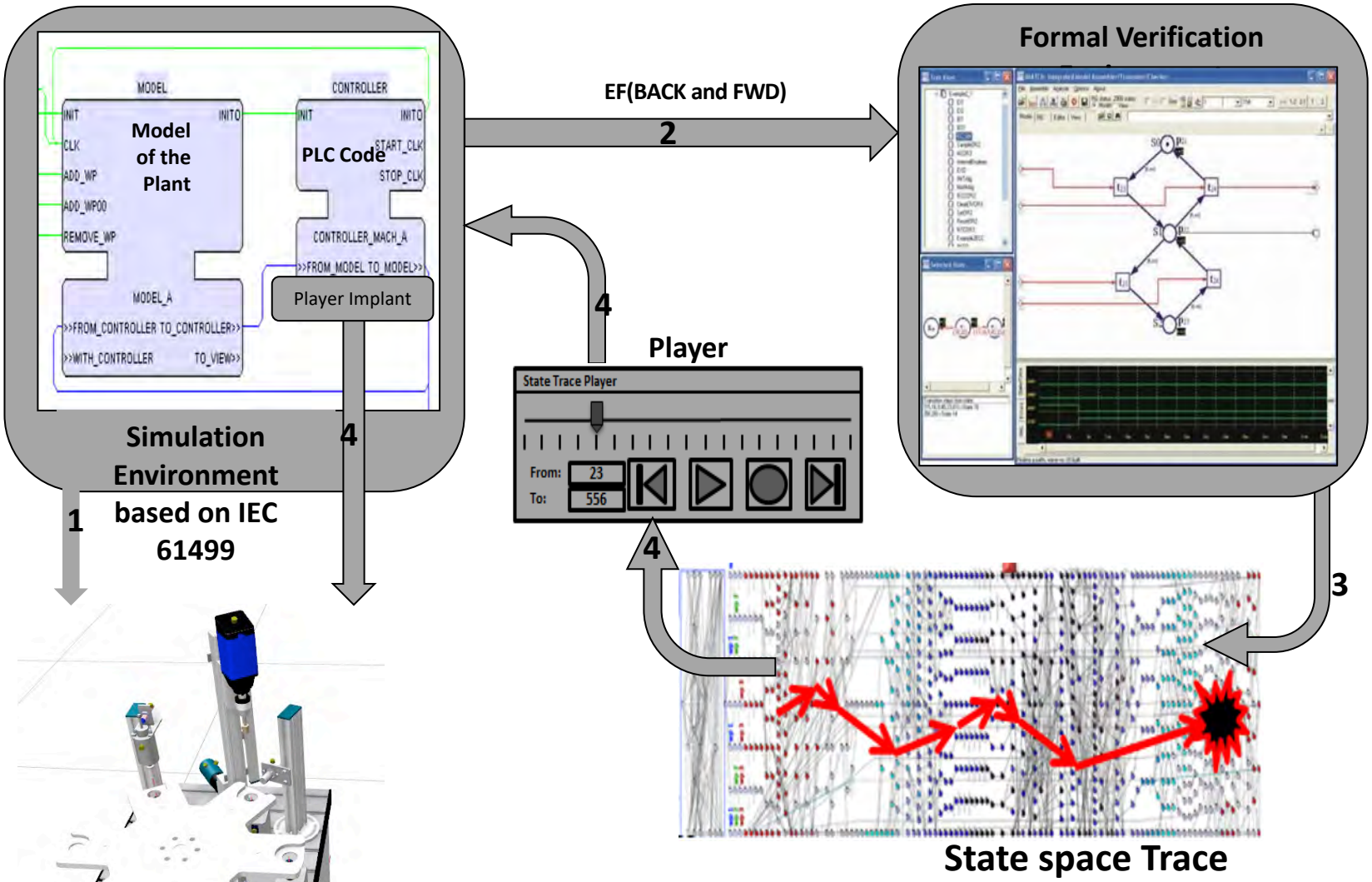
Specifications

SPEC EF alpha
SPEC EF beta
SPEC AG (alpha -> AF (beta))
SPEC EF Z1DP_alpha
SPEC EF Z1DP_beta
SPEC EF Z2DP_alpha
SPEC EF Z2DP_beta

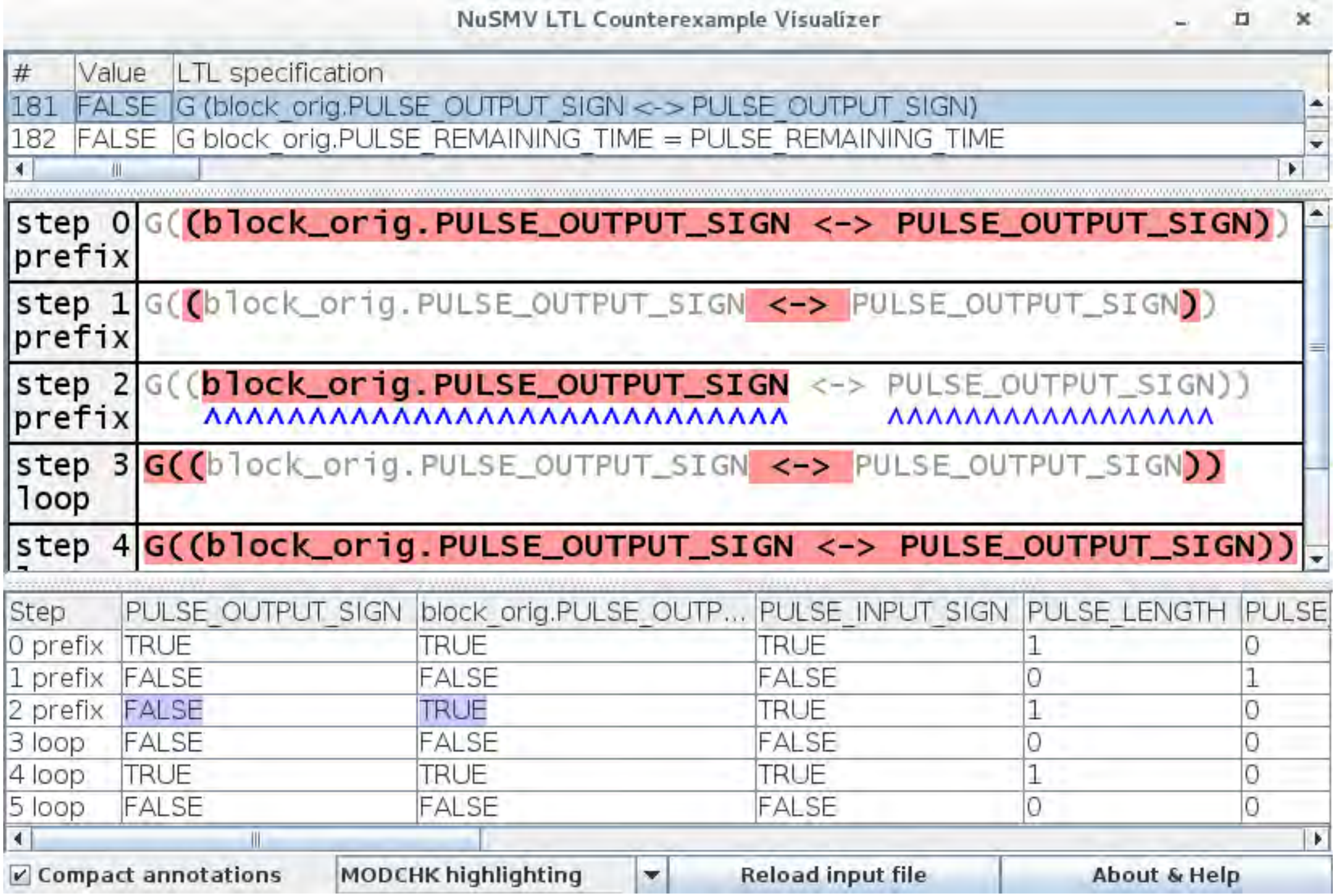


NuSMV Model Checker

Counter-example interpretation



Tool for visualizing requirements along counterexamples



#	Value	LTL specification
181	FALSE	G (block_orig.PULSE_OUTPUT_SIGN <-> PULSE_OUTPUT_SIGN)
182	FALSE	G block_orig.PULSE_REMAINING_TIME = PULSE_REMAINING_TIME

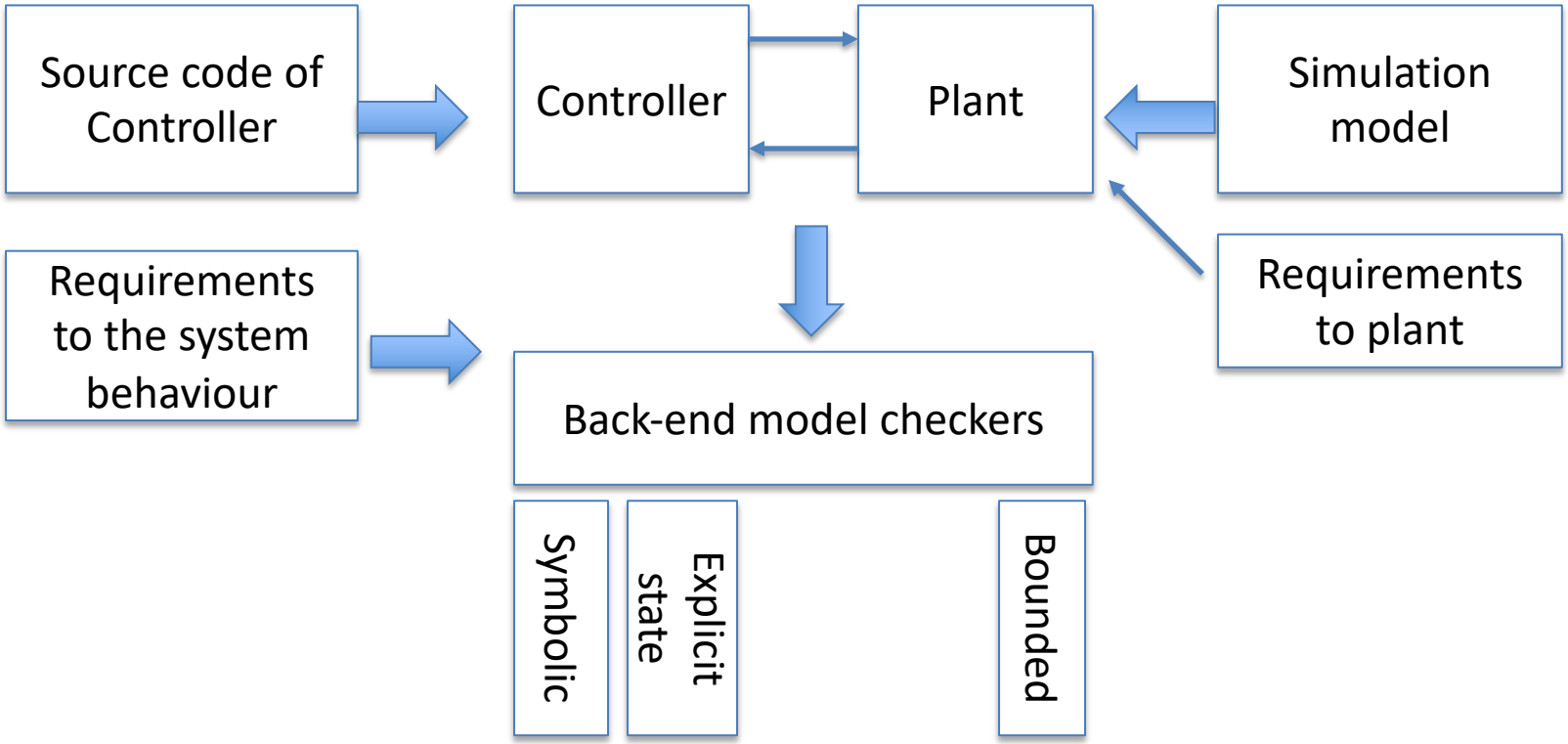
```
step 0 prefix | G((block_orig.PULSE_OUTPUT_SIGN <-> PULSE_OUTPUT_SIGN))
step 1 prefix | G((block_orig.PULSE_OUTPUT_SIGN <-> PULSE_OUTPUT_SIGN))
step 2 prefix | G((block_orig.PULSE_OUTPUT_SIGN <-> PULSE_OUTPUT_SIGN))
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^       ^^^^^^^^^^^^^^^^^
step 3 loop   | G((block_orig.PULSE_OUTPUT_SIGN <-> PULSE_OUTPUT_SIGN))
step 4        | G((block_orig.PULSE_OUTPUT_SIGN <-> PULSE_OUTPUT_SIGN))
```

Step	PULSE_OUTPUT_SIGN	block_orig.PULSE_OUTP...	PULSE_INPUT_SIGN	PULSE_LENGTH	PULSE
0 prefix	TRUE	TRUE	TRUE	1	0
1 prefix	FALSE	FALSE	FALSE	0	1
2 prefix	FALSE	TRUE	TRUE	1	0
3 loop	FALSE	FALSE	FALSE	0	0
4 loop	TRUE	TRUE	TRUE	1	0
5 loop	FALSE	FALSE	FALSE	0	0

☑ Compact annotations

MODCHK highlighting

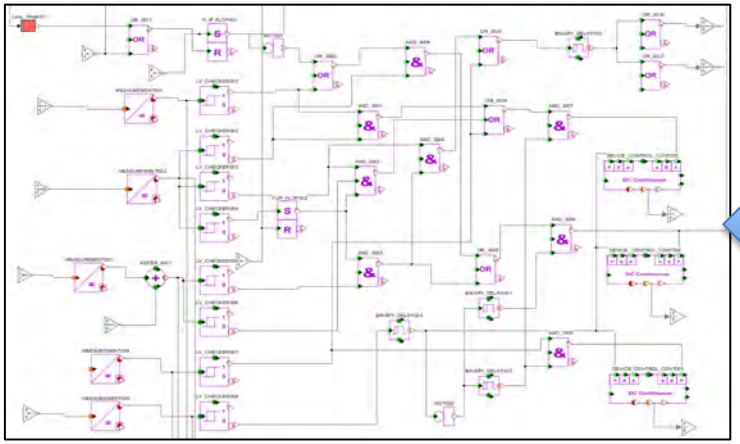
Opportunistic verification framework



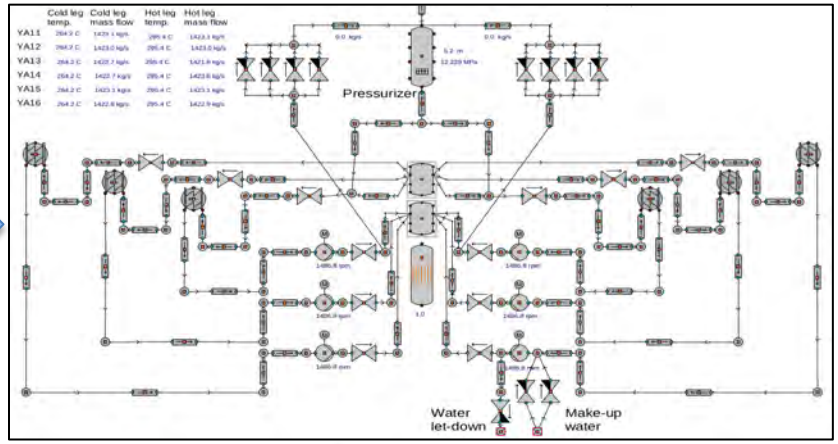
Nuclear systems verification

Case study: generic PWR model

Control Logic



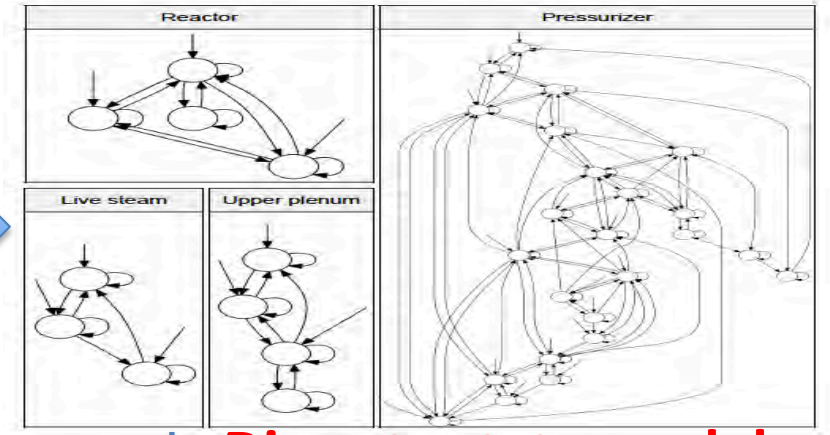
Simulation Model



Generated using meta-heuristic algorithms

```
MODULE FLIP_FLOP(FF_RESET_SIGN, FF_SET_SIGN, FF_OUTPUT_VALUE,
FF_RESET_DOMINATES)
VAR
memory: boolean;
DEFINE
FF_OUTPUT_SIGN := case
FF_RESET_SIGN & FF_SET_SIGN : !FF_RESET_DOMINATES;
FF_SET_SIGN : TRUE;
FF_RESET_SIGN : FALSE;
TRUE : memory;
esac;
FF_OUTINV_SIGN := !FF_OUTPUT_SIGN;
FF_OUTPUT_SIGN_FAULT := FALSE;
FF_OUTINV_SIGN_FAULT := FALSE;
ASSIGN
init(memory) := FF_OUTPUT_VALUE;
next(memory) := FF_OUTPUT_SIGN;
```

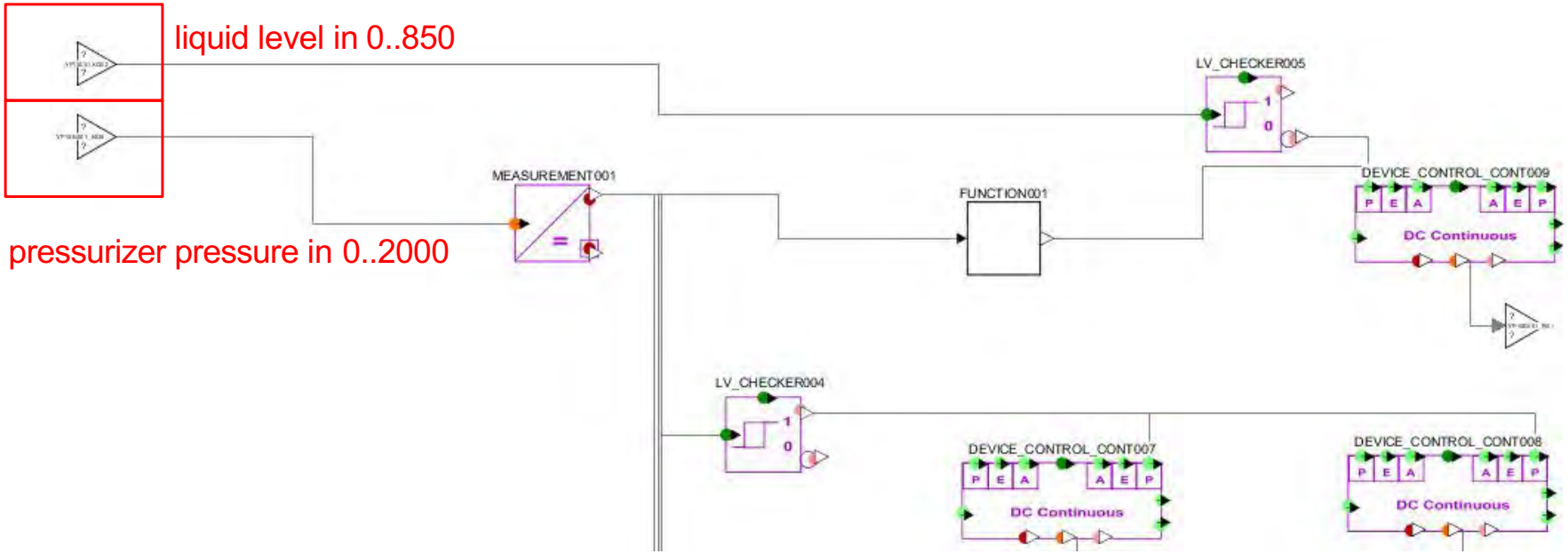
SMV model



Discrete state model

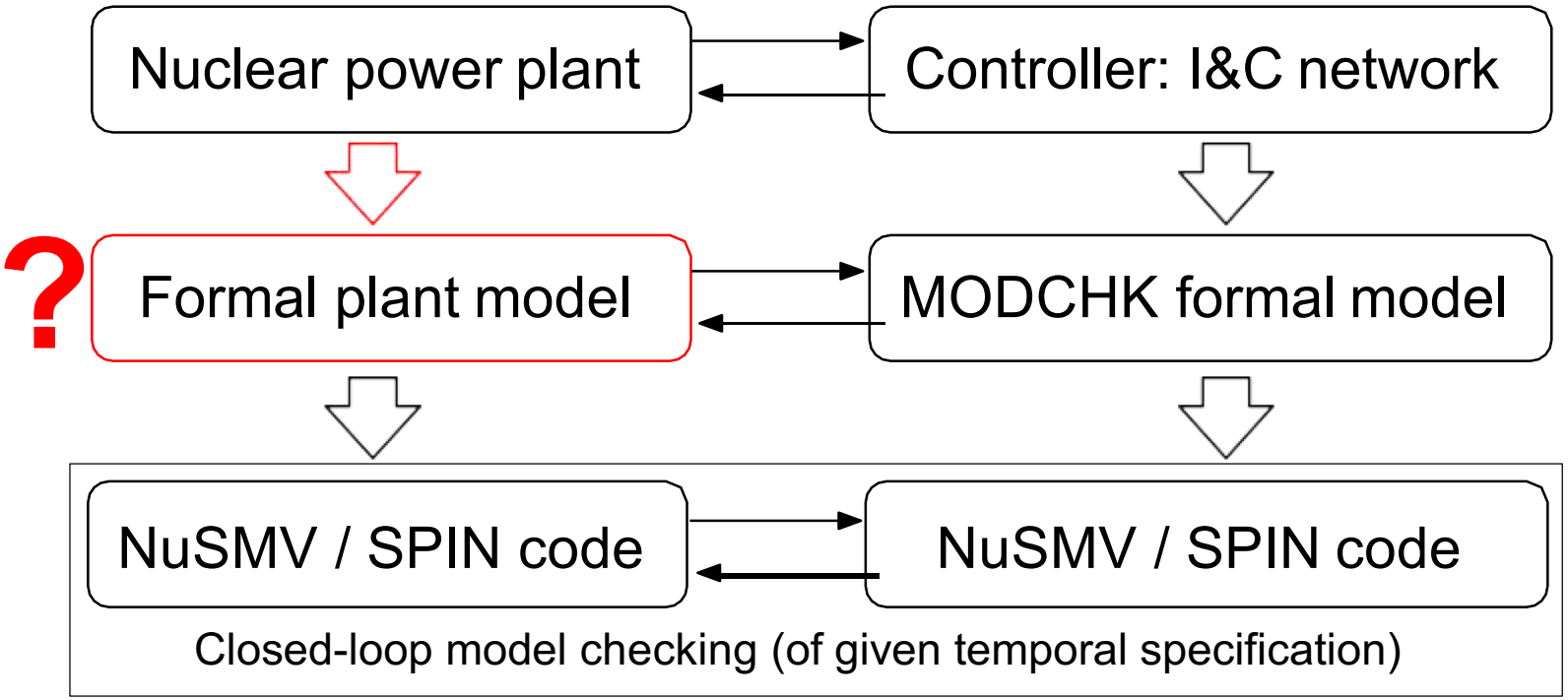
Model-Checker Tool

Default approach: open-loop model checking in MODCHK



- Inputs are assigned arbitrary values from defined ranges
- Request-response properties, spurious actuation properties and invariants can be checked
- What if some input value combinations/sequences are impossible, and the controller does not behave well on them?
- What if we wish to check a property over inputs?

Benefits of having the Plant Model



- Pros 1: can check a wider class of temporal specifications
- Pros 2: state space reduction (can be used in explicit-state verification, but not in symbolic)
- Cons: if the model is incomplete, some behaviors can be missed

Where to get the plant models from?

Model Mining Method

Simulation model of the closed-loop automation system

Initial simulation settings:
- plant initial condition
- controller mutations

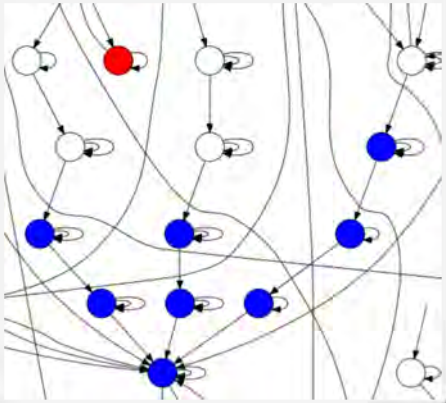
Data measured from the real process

Set of execution traces (around several hours or days of execution for large systems)

Symbolic model

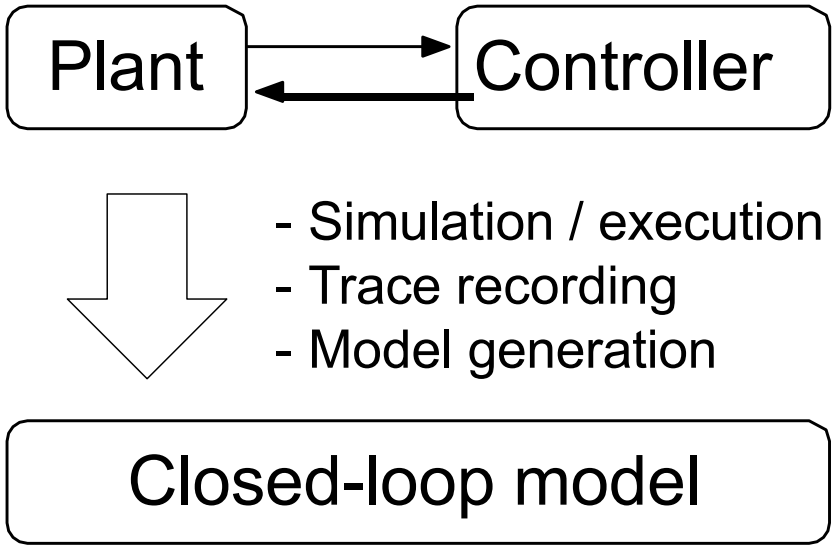
```
TRANS
    (next(output_pressure1) in
  & (next(output_voltage) in {
  & (next(output_pressure1) =
  & (next(output_pressure2) =
  & (next(output_pressure2) =
  & (next(output_voltage) = 1
  & (next(output_voltage) = 1
  & (next(output_voltage) = 1
  & (CONT_INPUT_1 in 0..50 & T
  & (CONT_INPUT_1 in 0..50 & n
  & (CONT_INPUT_1 in 0..50 & T
```

Explicit-state model



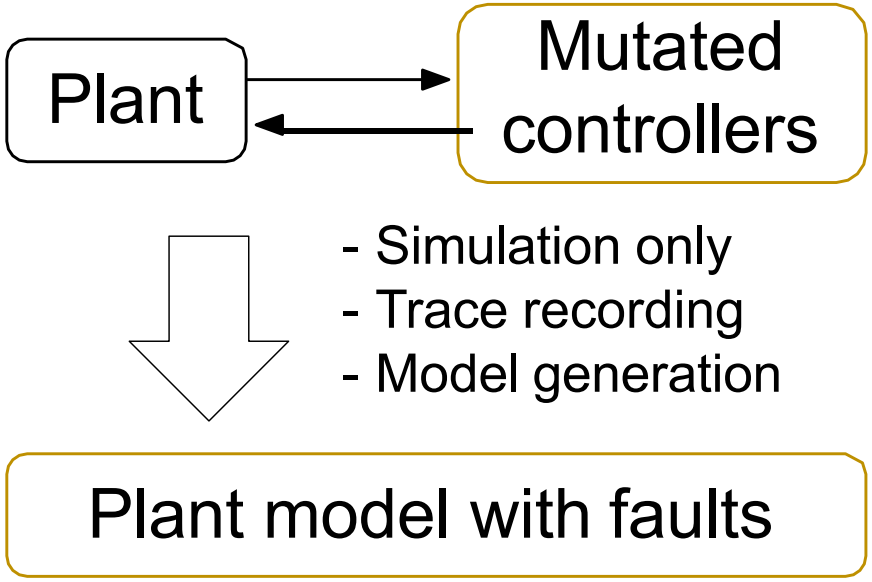
Why do we synthesise plant models and not closed-loop models?

Closed-loop model identification



- Only behaviors which are likely to appear in simulations, but we'd like to have some faults as well

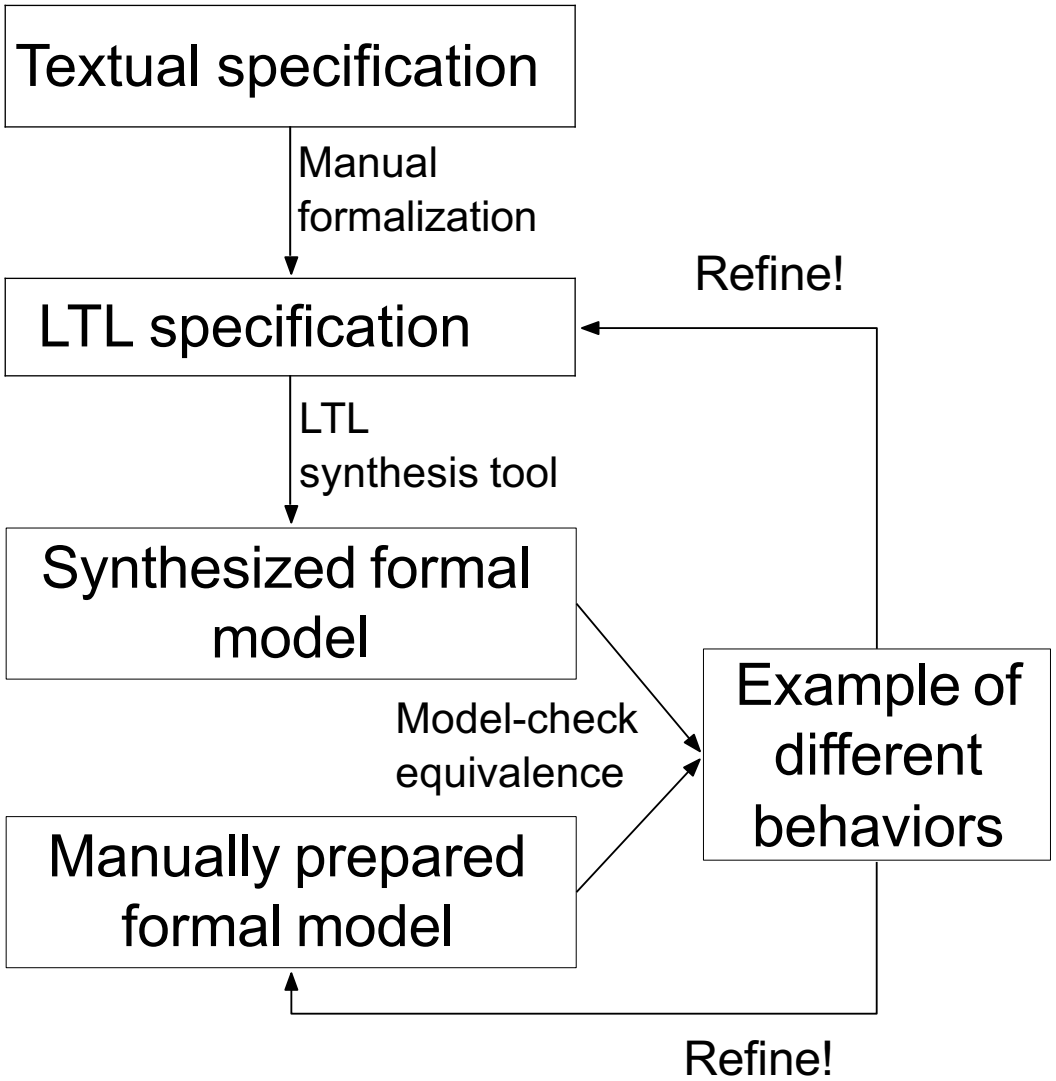
Plant model generation



- Can compose with the controller model (assume that it is available) and model-check

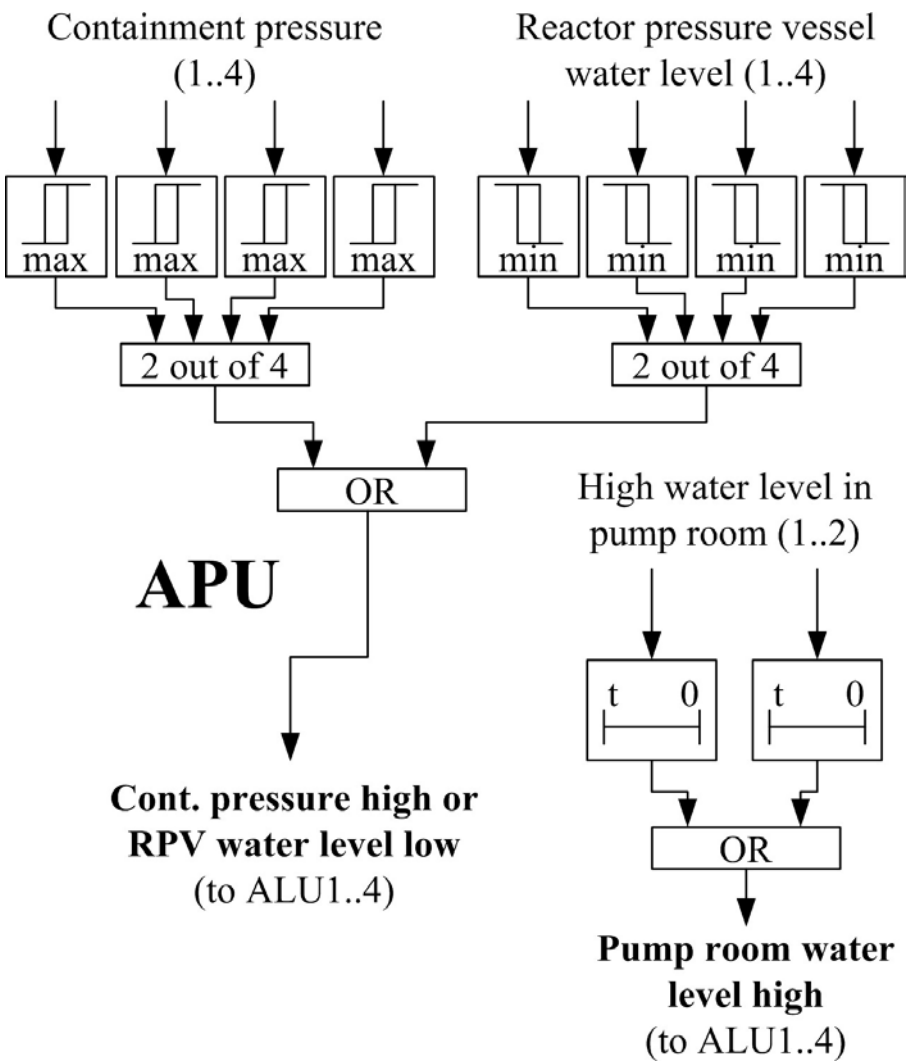
- Thus, we construct "semi-controlled" plant models
- Simulating without the controller would lead to the lack of non-faulty behaviors

Model construction from behavior examples and specifications



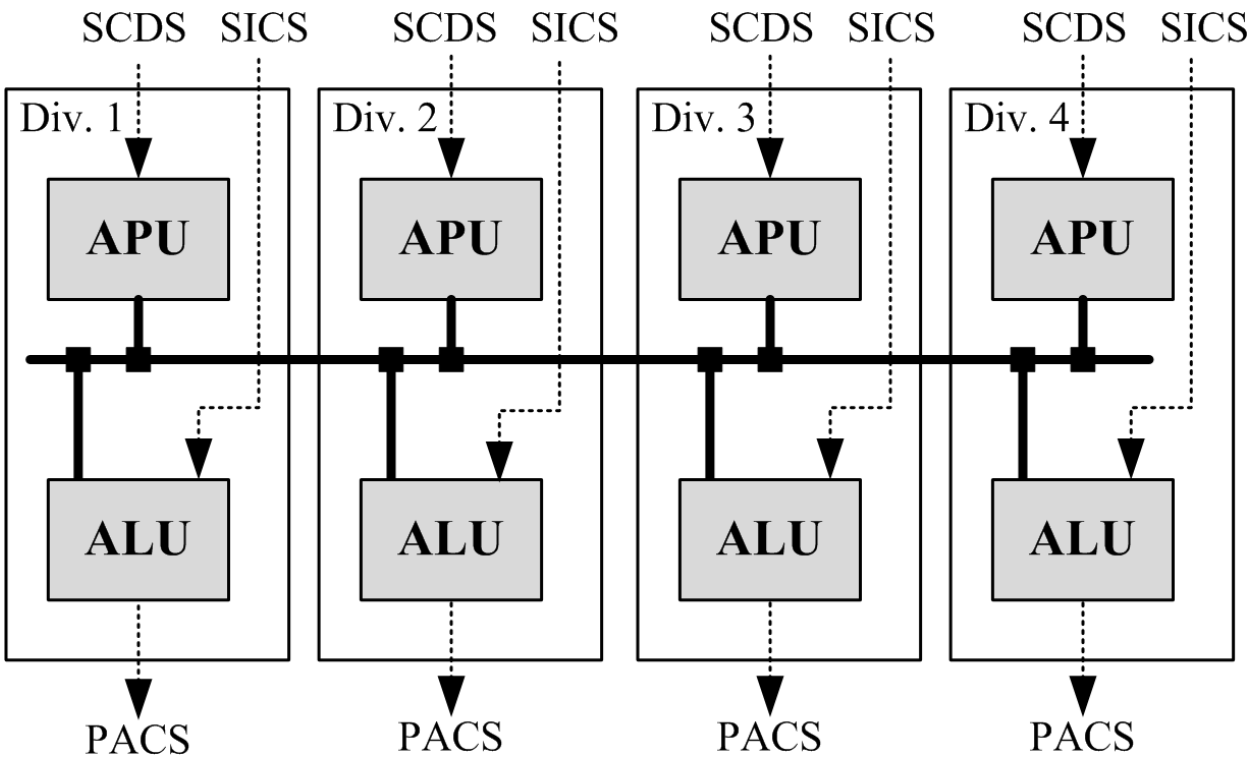
- LTL synthesis may fail. In this case, the specification is incorrect and needs to be refined
- If it does not fail and there is an example of different behaviors, refine **either** the model or the specification
- If there is no such example, **relax IO value boundaries**

Case study 2: Redundancy in nuclear systems



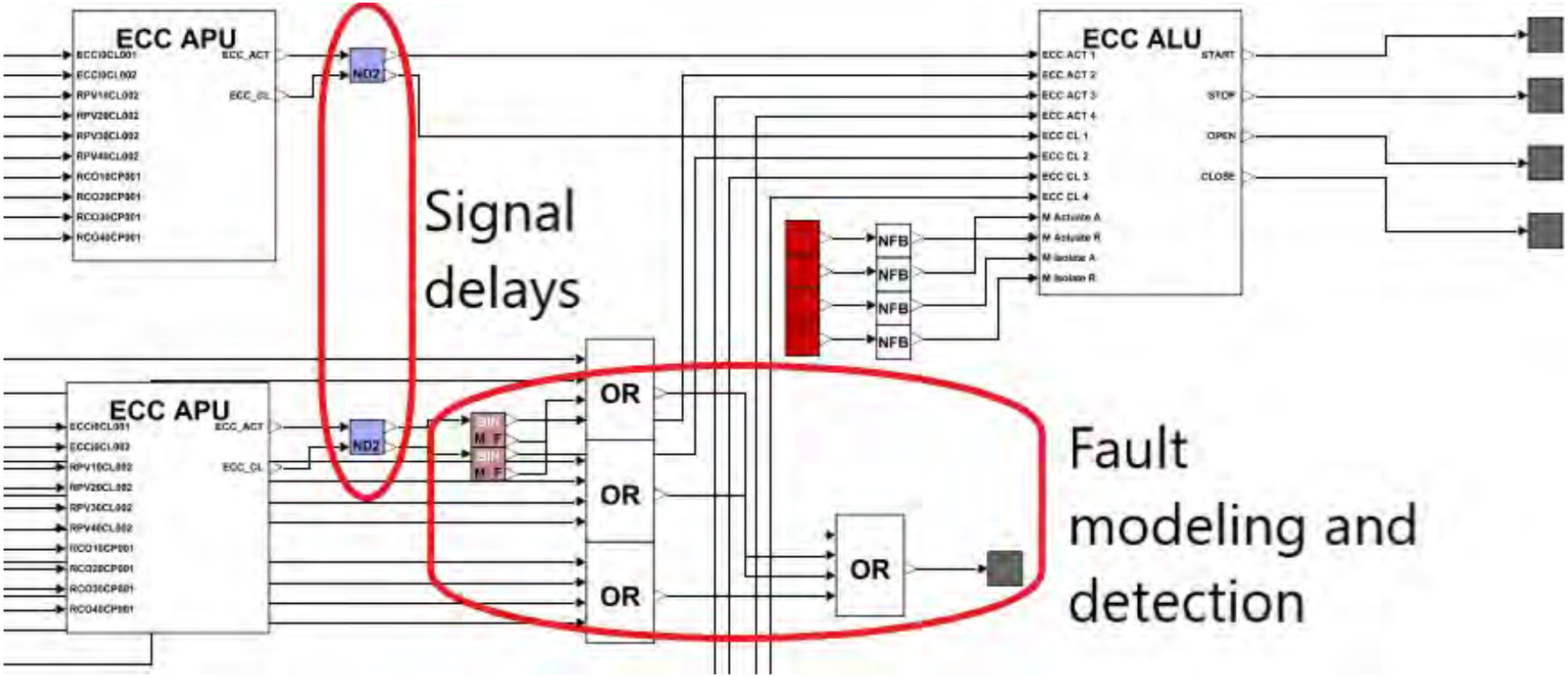
- Basic network: interconnected basic blocks
- Graphical modeling in a verification tool
- A very typical scenario is to detect certain conditions on the input and generate output accordingly
- This may involve delays, storing information in flip-flops, etc.

Hardware view: 4 physically isolated redundant divisions



- APU (acquisition and **processing unit**) and ALU (actuation logic unit = **voting unit**) are two I&C networks
- There are four pairs of APU/ALU, corresponding to four isolated divisions
- Goal: verify the entire system, accounting for possible failures

Modeling approach: implementation in MODCHK / NuSMV



- MODCHK is a graphical frontend to the model checker NuSMV
- Faults are modeled as replacement of signals with nondeterministic values
- Asynchrony is modeled as communication delays

Results of model checking

Case	Model checking time (s)	
	BMC LTL	BDD-based LTL
No faults, no delays	98	3
Only delays	117	TL
Only faults	25125	38
Both faults and delays	**138**	TL

- TL (time limit) = Neither of the properties were checked within the time limit of 8 hours
- BDD-based LTL model checking does not scale to the case of delays, even if they are bounded by 1 cycle
- Bounded model checking (BMC) works better

Cyber-physical engineering

Cyber-physical engineering is a new approach that is aiming at addressing *essentially new cyber-physical phenomena* of automation systems.

It relies on the newly developed cyber-physical science that is aiming at predicting and guaranteeing dependability properties of complex interconnected systems.

Cyber-physical engineering

- Wide use of the physical system models at all stages of system design and life-cycle
- Use of new cyber-physical formal languages that are capable of capturing efficiently properties of both physical world and cyber world
- Computations in the cyber part are enhanced with awareness of time, error, energy and other important physical parameters
- Use of design documents of both physical world and cyber world with ability to interconnect them and derive properties of one from another.
- Use of cyber power to create and validate models

Credits

Hans-Michael Hanisch

Igor Buzhinsky

Dmitrii Drozdov

Sandeep Patil

Cheng Pang

Edmund Clarke