multiform

Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

Bridging between different modeling formalisms - results from the MULTIFORM project

Martin Hüfner, Christian Sonntag, Sebastian Engell

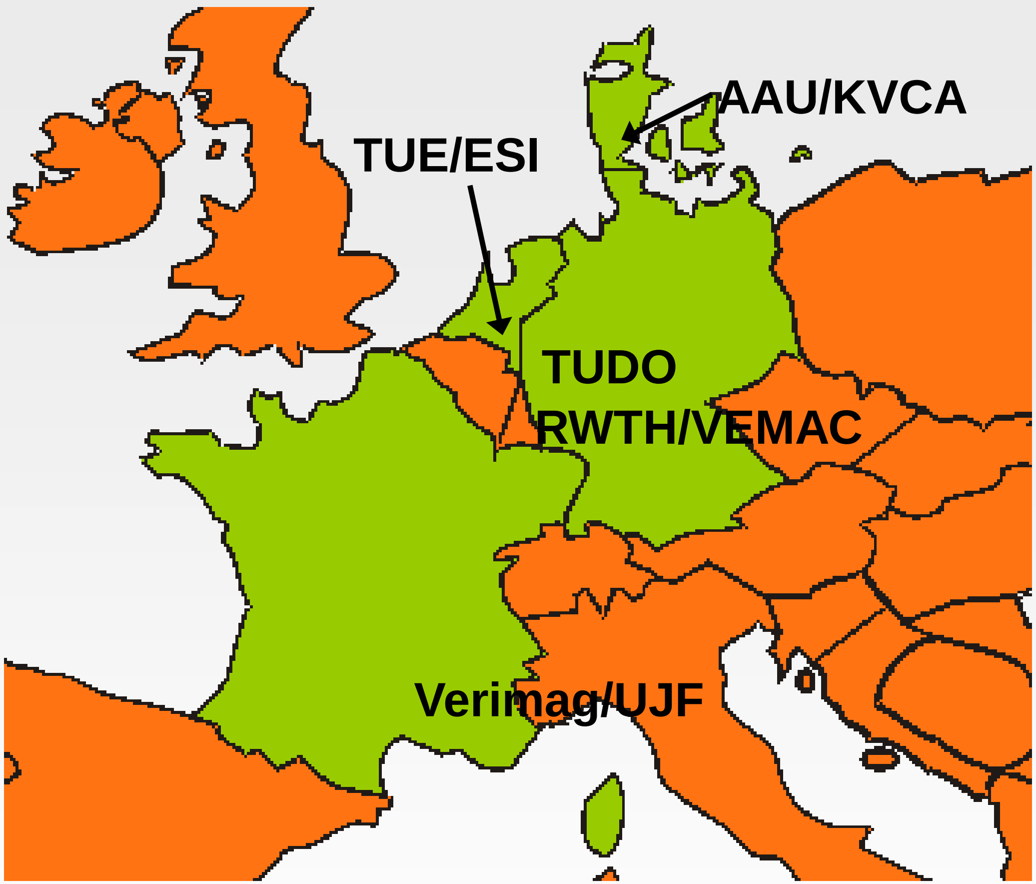
Process Dynamics and Operations Group
Department of Biochemical and Chemical Engineering
TU Dortmund
Germany

System Design meets Equation-based Languages, Sept. 19, 2012, Lund

Outline

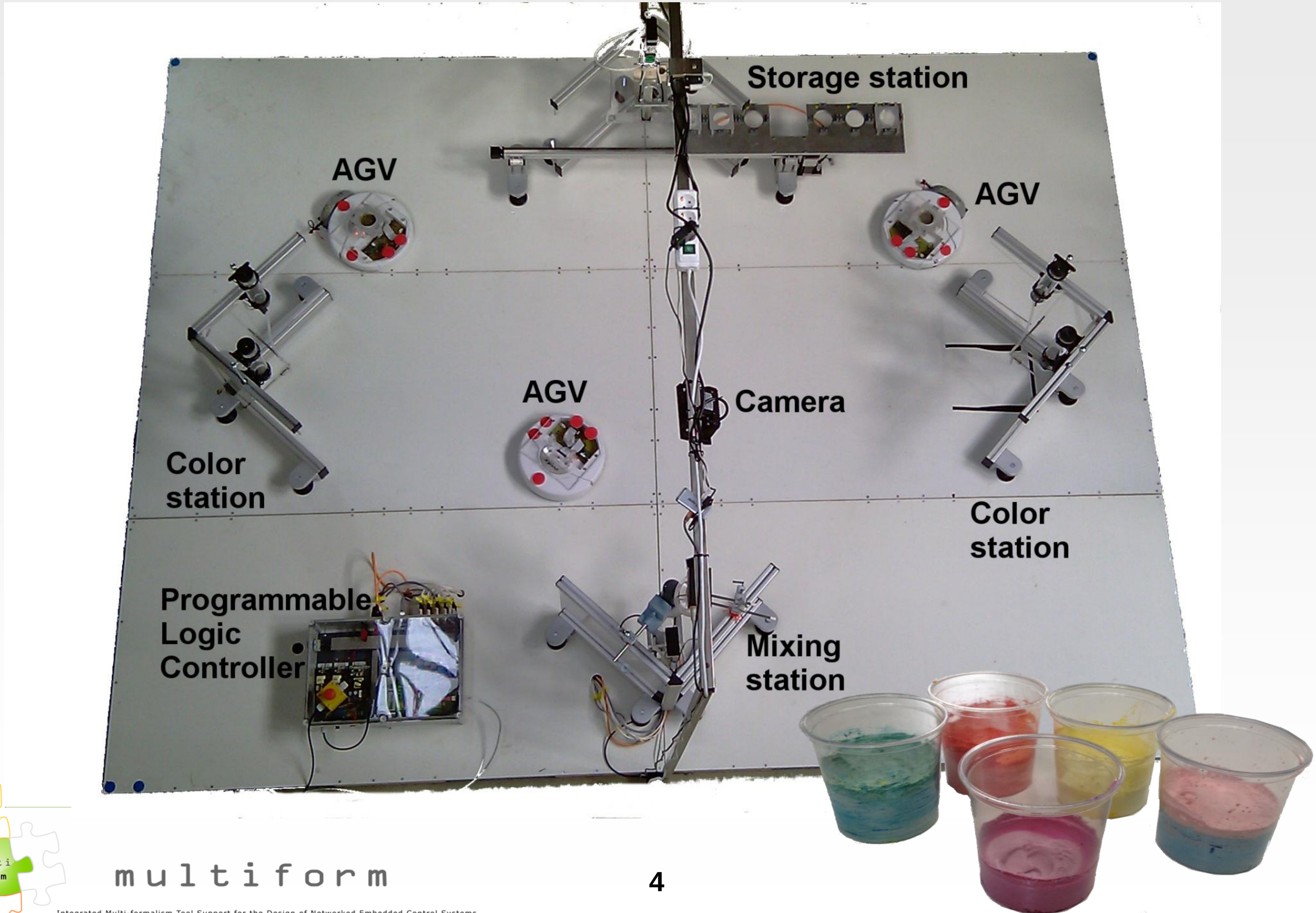
- The MULTIFORM project
- Design flow example
- Tool developments
- Model exchange and model transformations
- Lessons learned

MULTIFORM: EU ICT STREP 9/2008 – 5/2012

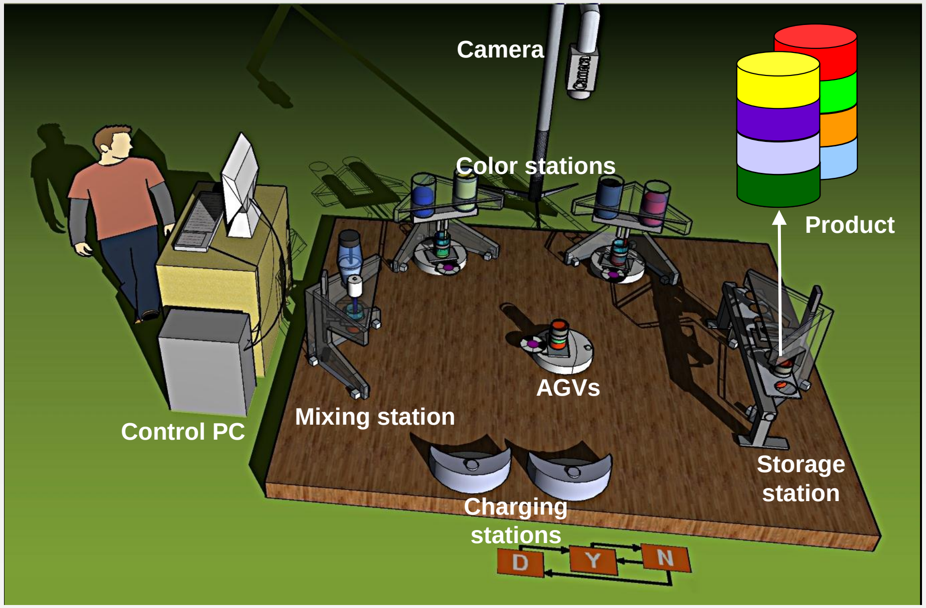


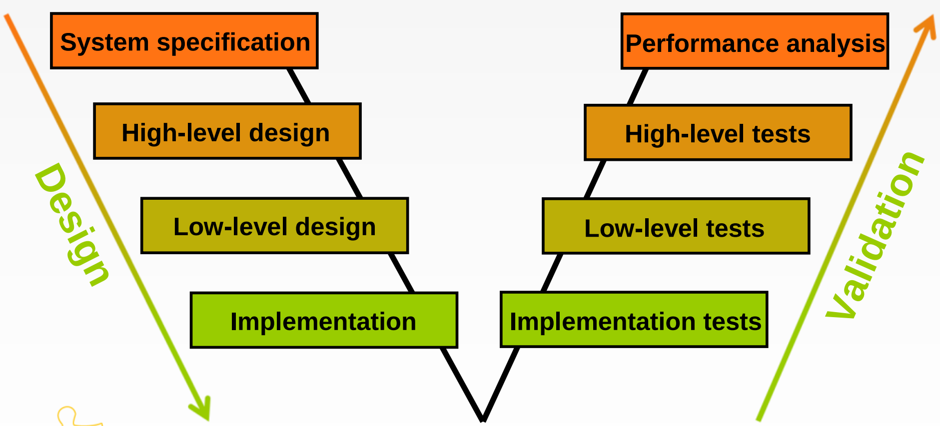
- **TUDO (Coordinator)**
  - TU Dortmund, Germany
    Sebastian Engell
- **TUE**
  - TU Eindhoven, Netherlands
    Koos Rooda, Bert van Beek, Jos Baeten
- **Verimag/ UJF**
  - Universite Joseph Fourier, Grenoble, France
    Goran Frehse, Oded Maler
- **RWTH**
  - RWTH Aachen, Germany
    Stefan Kowalewski
- **AAU**
  - Aalborg Universitet, Denmark
    Kim Larsen, Brian Nielsen
- **ESI**
  - Stichting Embedded Systems Institute
    Ed Brinksma, Boudewijn Haverkort
- **VEMAC**
  - Aachen, Germany
    Michael Reke
- **KVCA**
  - “Danish Cooling Cluster”
    Jens Andersen
  - Closely working with DANFOSS

Example: Design of a Pipeless Plant



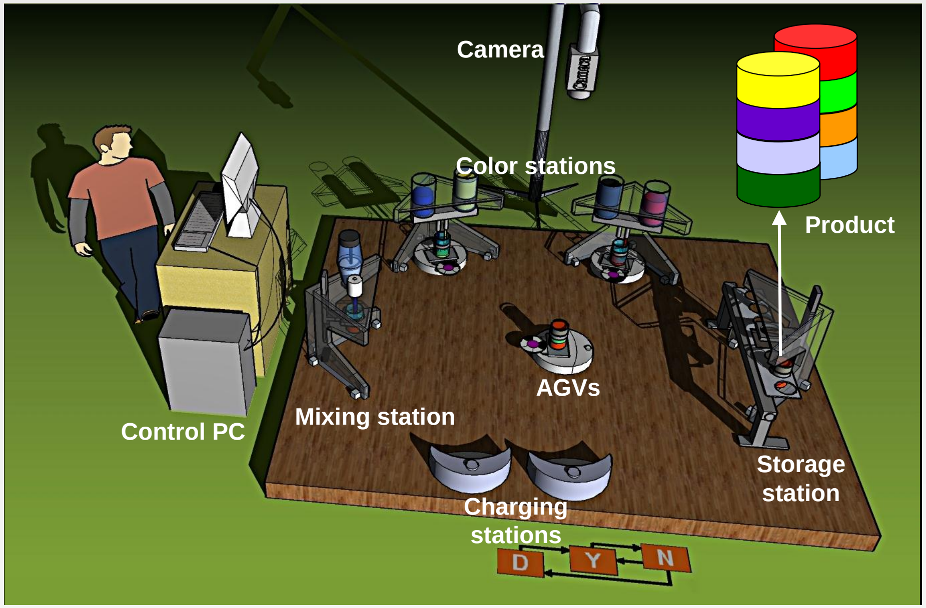
Challenges for Model-based Design (1)

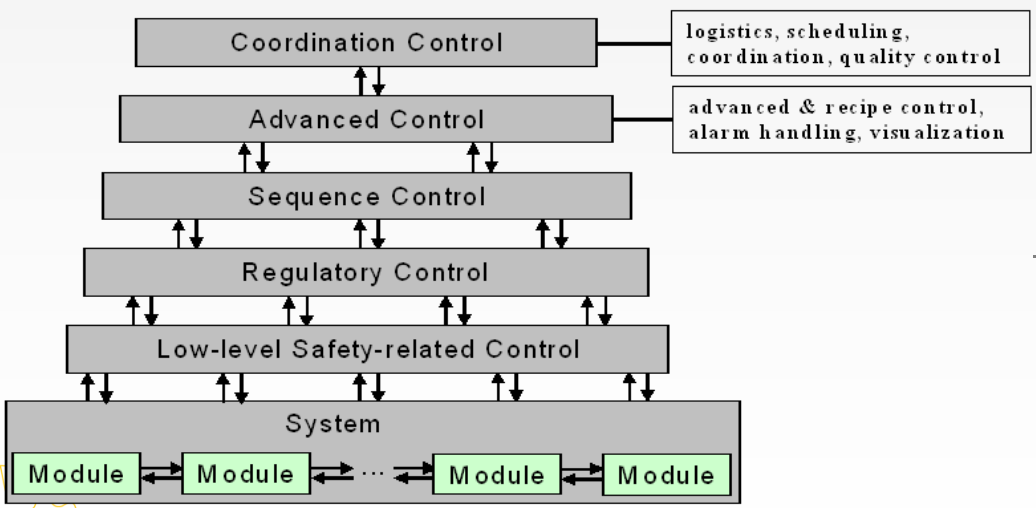




- Design and validation on different levels of abstraction
  - Specification
    - Specification of the tasks and of the performance of the system
  - High-level design
    - Choice of the equipment, feasibility and bottleneck analysis, throughput maximization, plant layout optimization
  - Low-level design
    - Optimization and control of processing steps and motion dynamics, logic control
    - Choice of sensors and actuators, communication system
  - Implementation
    - PLCs, embedded controllers, communication system

Challenges for Model-based Design (2)



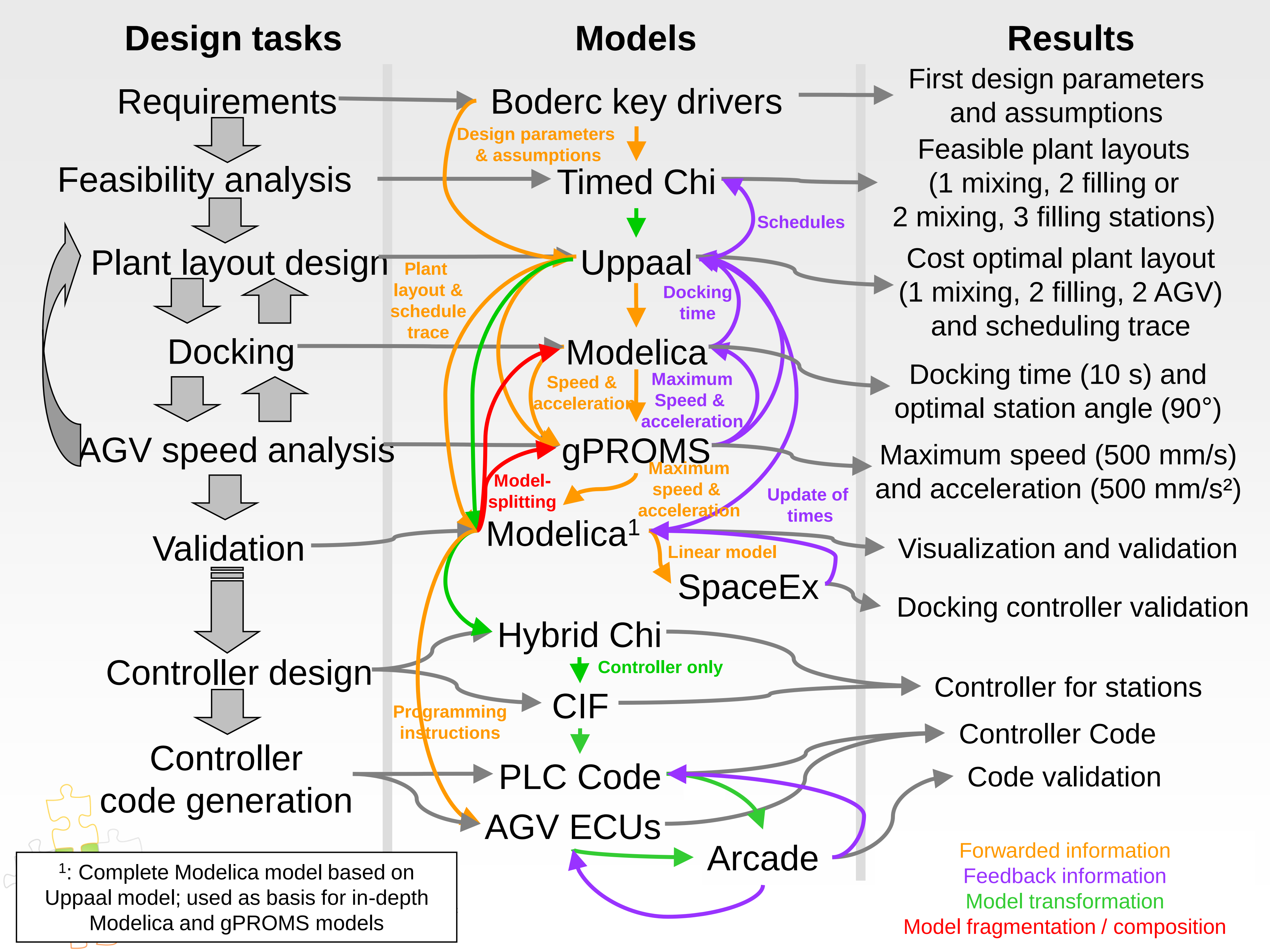


- The control system spans the complete control hierarchy
  - Coordination control — **Timed or hybrid models**
    - Scheduling and performance optimization
  - Advanced control
    - Control of batch processes
    - AGV path planning — **Continuous models**
  - Regulatory control — **Discrete-event, hybrid, and continuous models**
    - AGV motion control
    - Docking control
    - Sequence control in the processing stations
    - Low-level continuous control
  - Low-level safety-related control

Discrete-event, timed, and hybrid models

multiform

Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems



Design tasks
Models
Results
Requirements
Feasibility analysis
Plant layout design
Docking
AGV speed analysis
Validation
Controller design
Controller
code generation
Boderc key drivers
Design parameters
& assumptions
Timed Chi
Uppaal
Plant
layout &
schedule
trace
Schedules
Docking
time
Modelica
Speed &
acceleration
Maximum
Speed &
acceleration
gPROMS
Model-
splitting
Maximum
speed &
acceleration
Update of
times
Modelica¹
Linear model
SpaceEx
Hybrid Chi
Controller only
CIF
Programming
instructions
PLC Code
AGV ECUs
Arcade
First design parameters
and assumptions
Feasible plant layouts
(1 mixing, 2 filling or
2 mixing, 3 filling stations)
Cost optimal plant layout
(1 mixing, 2 filling, 2 AGV)
and scheduling trace
Docking time (10 s) and
optimal station angle (90°)
Maximum speed (500 mm/s)
and acceleration (500 mm/s²)
Visualization and validation
Docking controller validation
Controller for stations
Controller Code
Code validation
Forwarded information
Feedback information
Model transformation
Model fragmentation / composition
¹: Complete Modelica model based on
Uppaal model; used as basis for in-depth
Modelica and gPROMS models

Integrated Model-based Design

- Integrated modeling and design of the system itself and of the multi-layered and networked control system
  - Including a structured approach to the management of specifications, design decisions, models, and results
- Coverage of all layers of the automation and design hierarchy
  - Integrated tool support on all layers of the automation and design hierarchies
    - Current state: Islands of support for specific design and analysis tasks
- Trans-level integration of model-based design approaches
- Support of iterations in the design process
    - Propagation of faults and unexpected behaviors
    - Modifications over the life cycle without top-down redesign

→ Improvement of the ***tool support*** for the design steps

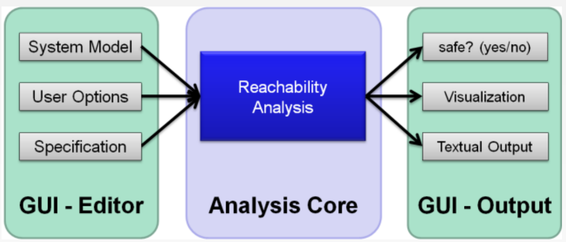
→ Tool integration and ***Design Framework***

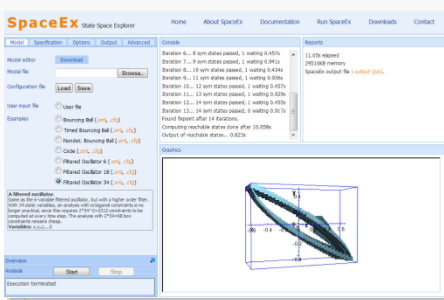
→ ***Exchange of models*** between tools via the CIF (Compositional Interchange Format)

MULTIFORM Tools and Tool Chains

Verification

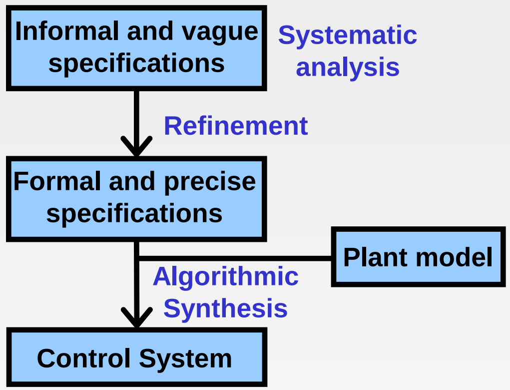
- Verification tool *SpaceEx* (successor of *PHAVer*)
- Consistency checking methods using *UPPAAL*
- Step-wise refinement based on *HCIF*



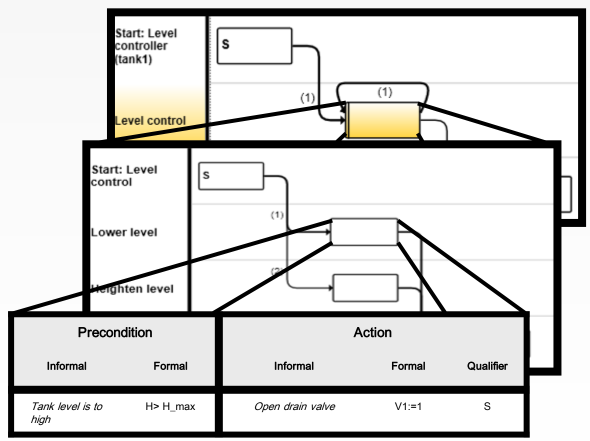


Logic Controller Design

Integrated controller design and analysis

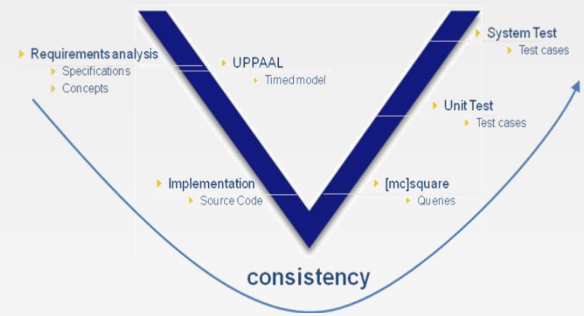


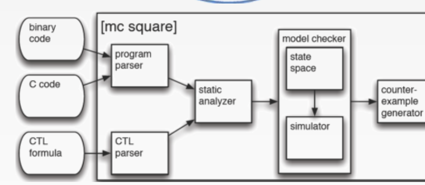
Specification using *DC/FT*

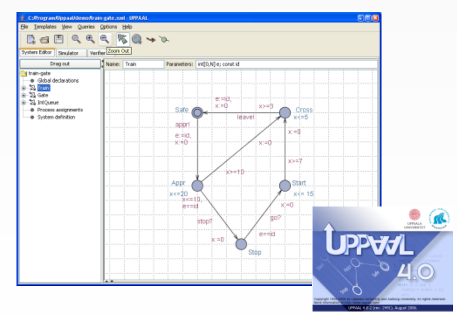


Code Analysis

Code and requirements analysis for ECUs using *Arcade* and *UPPAAL*





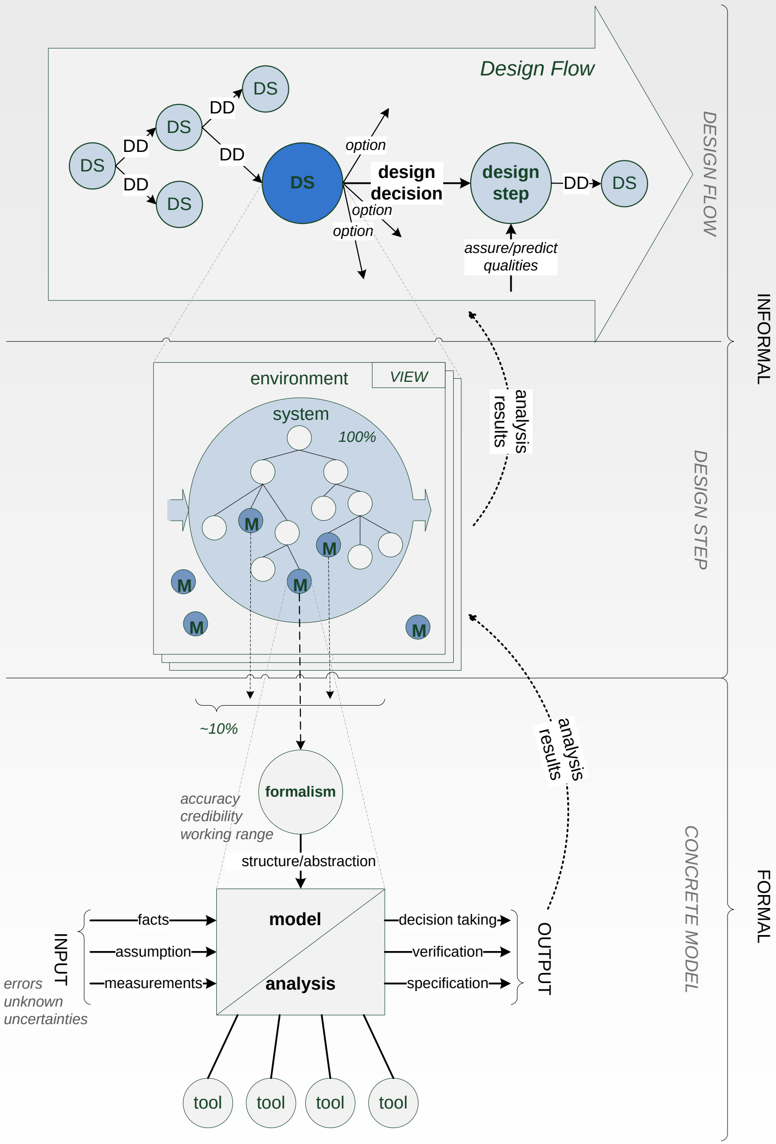


multiform

Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

The MULTIFORM *Design Framework* [ESI]

- Consistent integration of design models into a common software framework
- Support of a generic design flow model
  - Design decisions
  - System design
- Consistency management
  - Communication of design parameters
  - Conflict detection
  - Models and results management

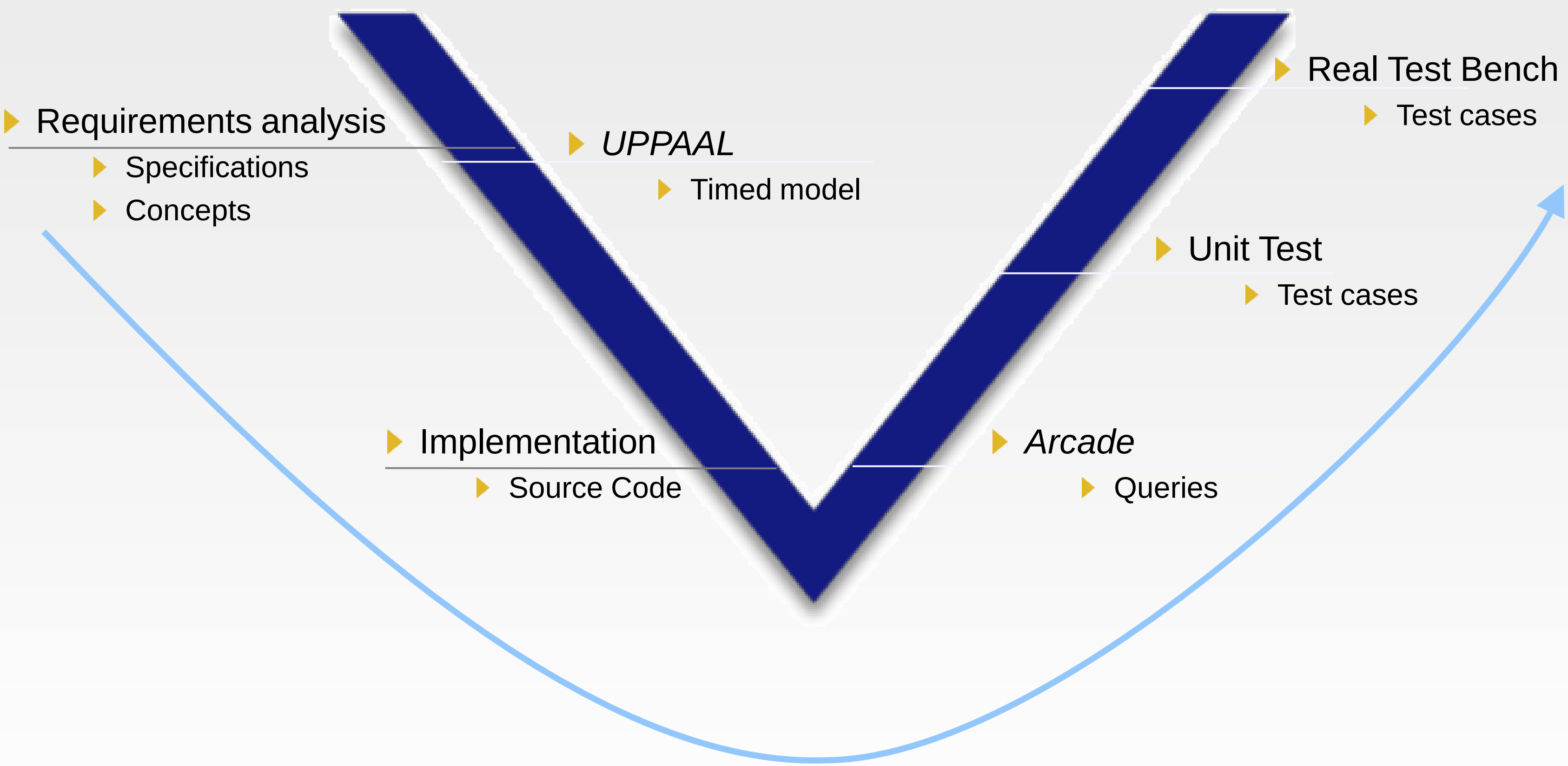


multiform

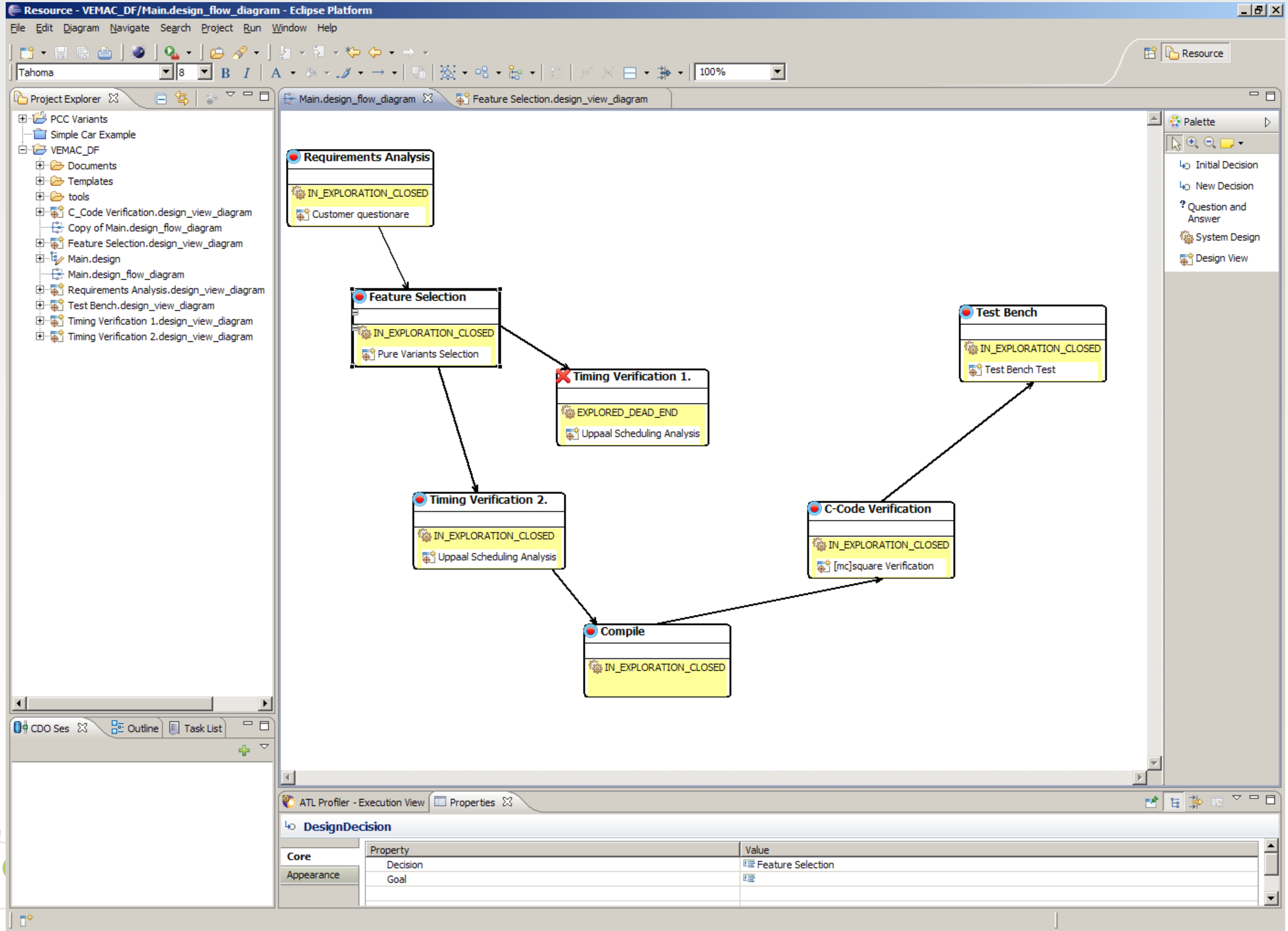
Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

VEMAC Development Process

V-Model



Customized Design Framework Prototype

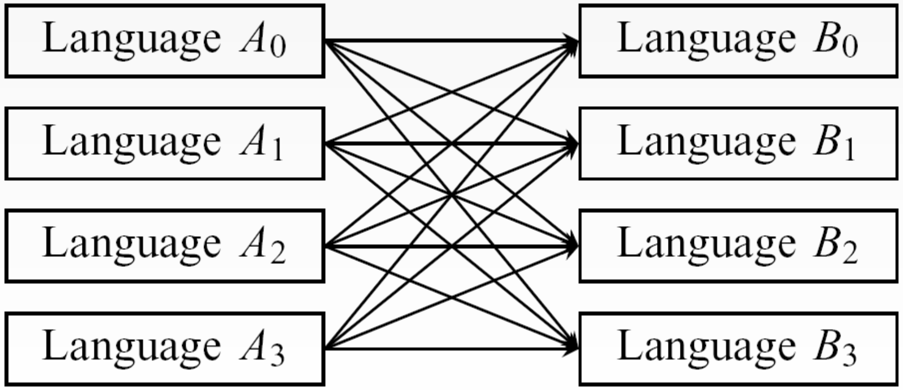


Model Exchange Using the Compositional Interchange Format (CIF)

- Incompatibility of tools is one of the major obstacles for a broader acceptance of model-based design in industry

→ Achieve inter-operability by (algorithmic) model transformations

- One possibility: Bi-lateral transformations
  - Problems
    - Many transformations may be needed
    - The developer of a transformation must be familiar with many different formalisms



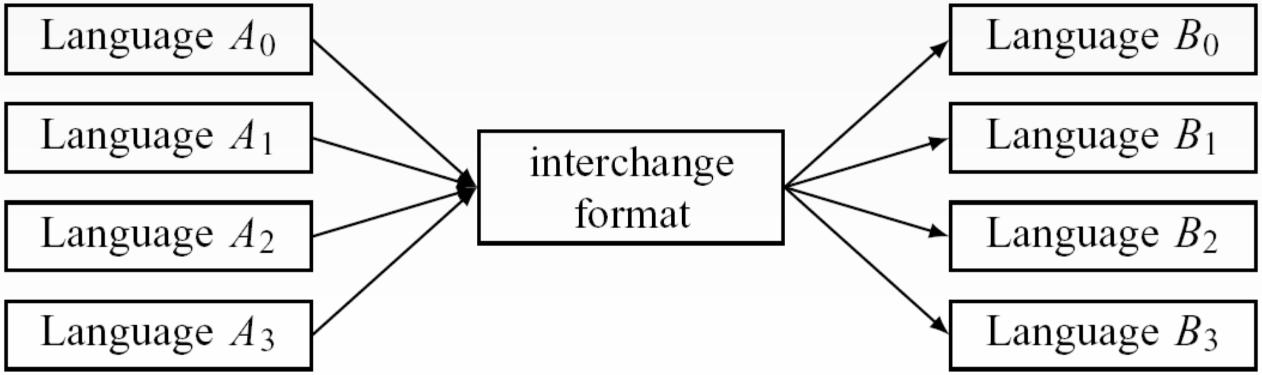
Model Exchange with the Compositional Interchange Format (CIF)

- Incompatibility of tools is one of the major obstacles for a broader acceptance of model-based design in industry

→ Achieve inter-operability by (algorithmic) model transformations

- One possibility: Bi-lateral transformations
- Interchange format
  - Generic and sufficiently rich modelling formalism
  - Only transformations from/to the interchange format are necessary

  → Reduction of the implementation effort



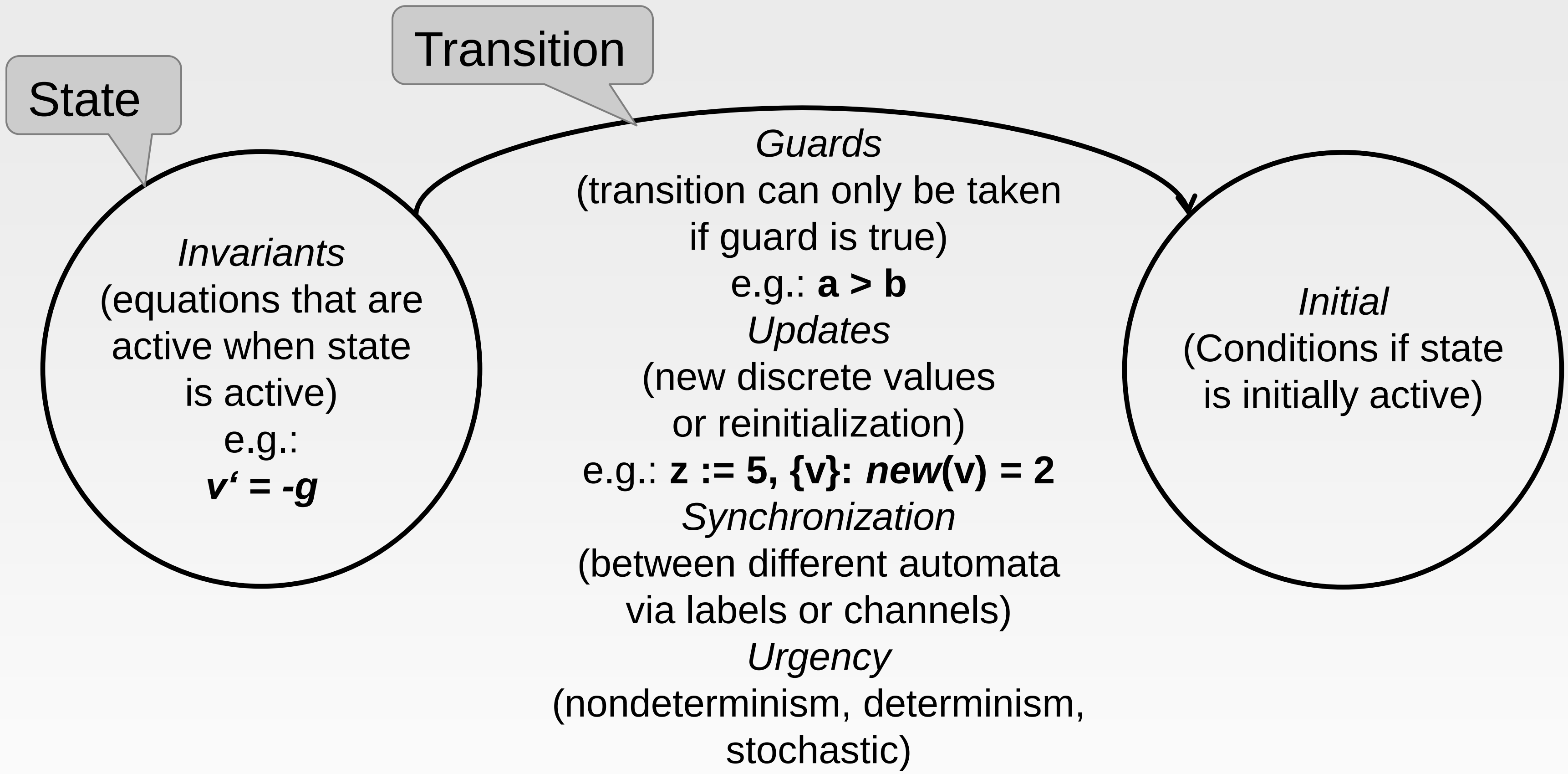
The Compositional Interchange Format (CIF) [Bert van Beek et al., TUE]

- Purposes
  - Establish inter-operability of a wide range of tools
  - Provide a generic formalism for general hybrid systems
- Major features
  - Formal and compositional semantics
    - Independent of implementation aspects
    - Mathematical correctness proofs of translations

    → Property-preserving model transformations possible
  - Fully implicit DAE dynamics (possibly discontinuous)
  - Hierarchy and re-usability
    - Parallel composition with different communication concepts
  - Model component interaction
    - Point to point communication, multi-component synchronization, broadcast communication, shared variables
  - Different urgency concepts

http://devel.se.wtb.tue.nl/trac/cif/

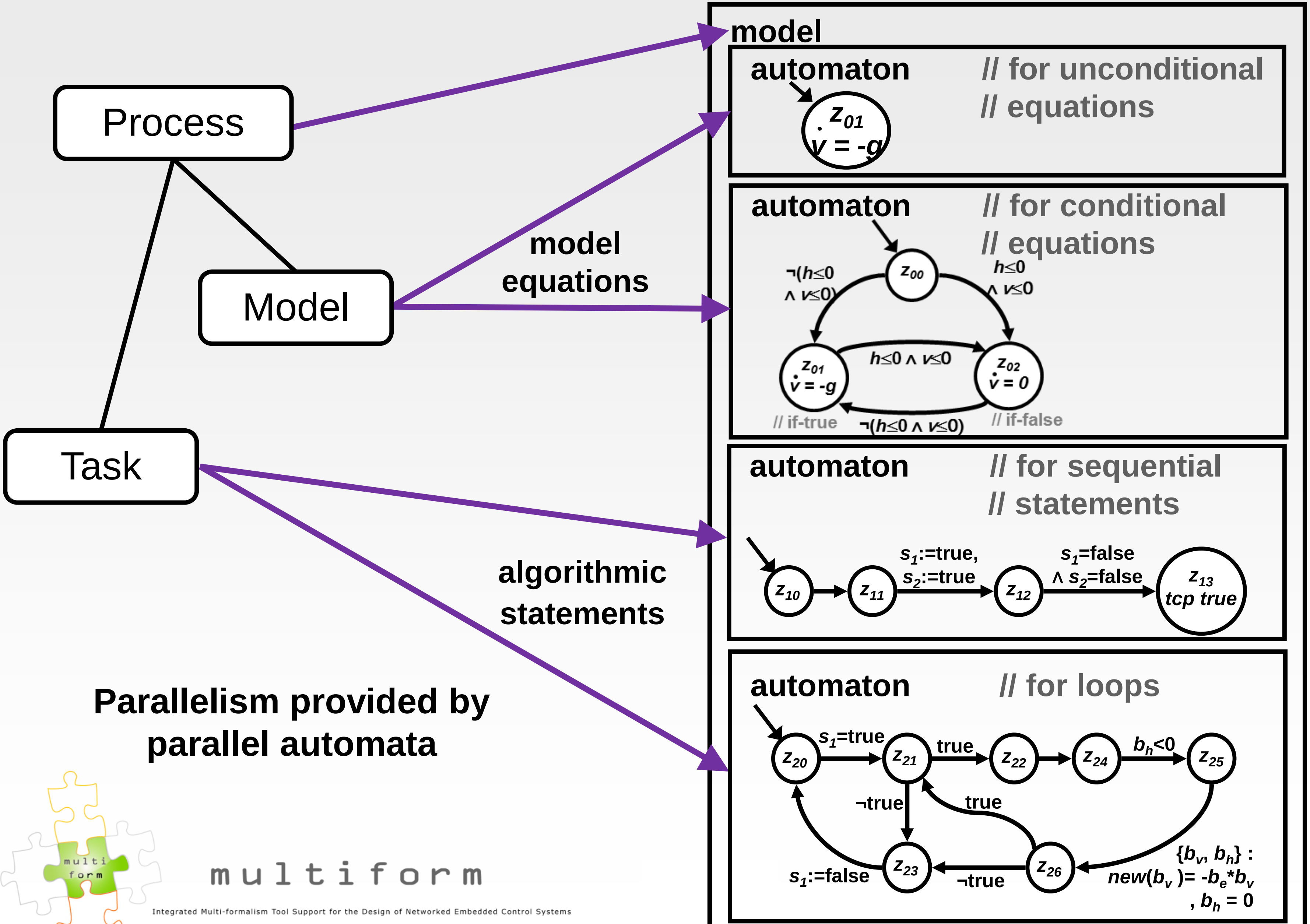
The Compositional Interchange Format (CIF) [Bert van Beek et al., TUE]



Formal definition by *Structural Operational Semantics* (SOS) rules, e.g.:

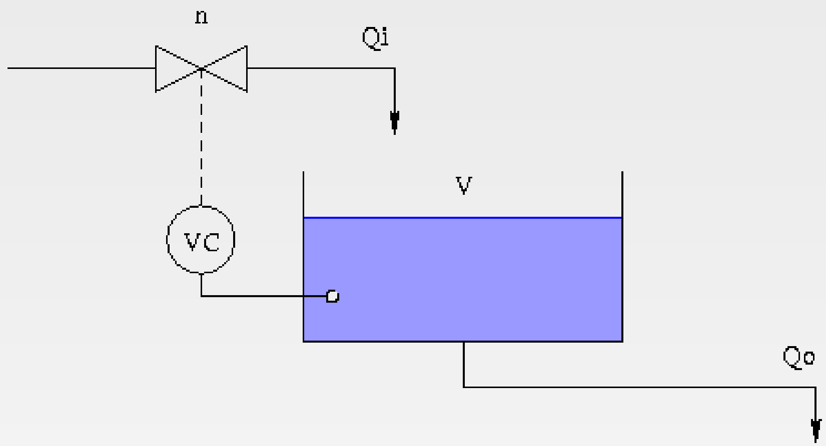
$$\frac{(\alpha_0, \sigma) \xrightarrow{a,\mathrm{true},X} (\alpha_0', \sigma'),\ (\alpha_1, \sigma) \xrightarrow{a,\mathrm{true},X} (\alpha_1', \sigma')}{(\alpha_0 \parallel \alpha_1, \sigma) \xrightarrow{a,\mathrm{true},X} (\alpha_0' \parallel \alpha_1', \sigma')}$$

Transformations – gPROMS ➔ CIF (Excerpt)



Parallelism provided by parallel automata

Simple Example: tank.cif



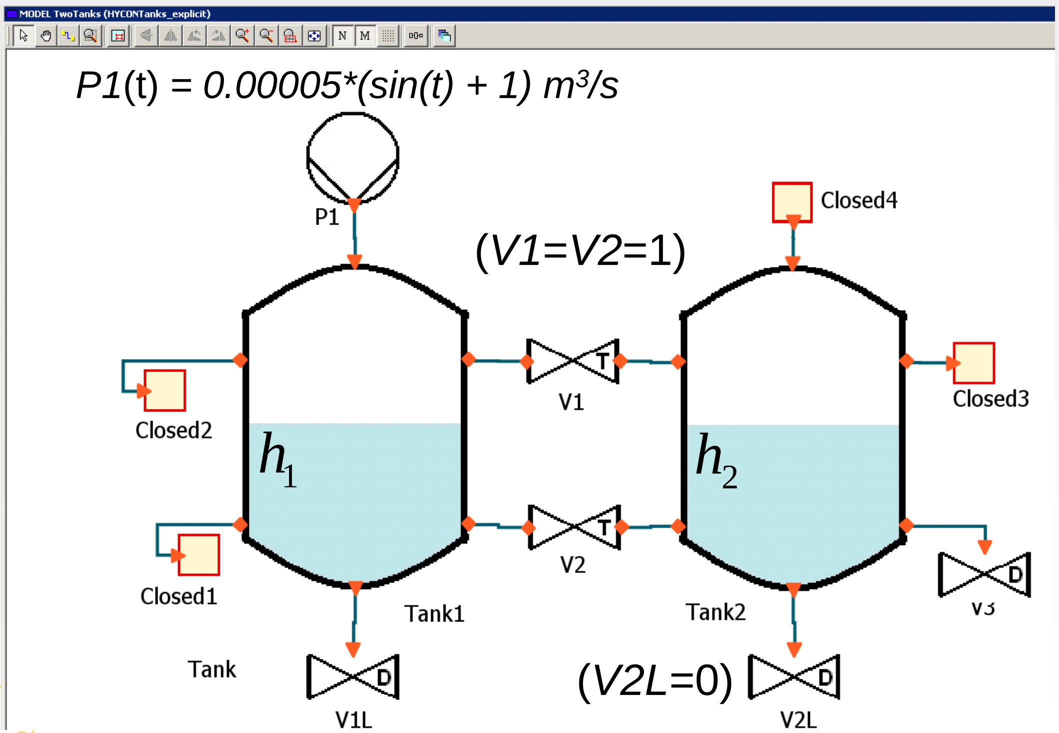

```
model TankController()=
|[ cont control real V = 10.0
 ; var          real Qi, Qo
 ; disc control nat n = 0
:: Tank : |( mode physics = initial
                  inv V' = Qi - Qo
                    , Qi = n * 5.0
                    , Qo = sqrt(V)
        )|
||
   Controller : |( mode closed = initial
                            (when V <= 2  now do n := 1) goto opened
                   , opened = (when V >= 10 now do n := 0) goto closed
             )|
]|
```

Flattened Example: tank_flat.cif

```
model TankController() =
|[ var real V = 10.0  ; var real Qi  ; var real Qo  ; var nat n = 0
:: Tank_Controller:
 |(
     var string Controller_LP ; var string Tank_LP
   ; controlset Controller_LP, Tank_LP, V, n
   ; dyntypemap disc Controller_LP; disc Tank_LP; disc n; cont V;
    mode X =
      initial  (((Tank_LP) = ("physics")) and (true))
                    and (((Controller_LP) = ("closed")) and (true))
      inv ((Tank_LP) = ("physics")) => ((((V)') = ((Qi) - (Qo)))
                    and (((Qi) = ((n) * (5.0))) and ((Qo) = (sqrt(V)))))
      tcp ((Controller_LP) = ("closed")) => (not ((V) <= (2)))
      tcp ((Controller_LP) = ("opened")) => (not ((V) >= (10)))
       ( when (V) <= (2), (Controller_LP) = ("closed") do
           {Controller_LP, n} : (new(n)) = (1), (new(Controller_LP)) = ("opened") )
       ( when (V) >= (10), (Controller_LP) = ("opened") do
           {Controller_LP, n} : (new(n)) = (0), (new(Controller_LP)) = ("closed") ) goto
      X
 )|
]|
```

A Two-tank System under Discrete Control

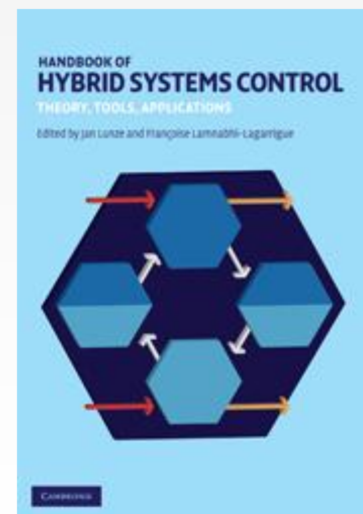
- Hybrid non-linear model of a two-tank system, modeled in *gPROMS*
  - Designed to contain many constructs of the *gPROMS* language



Two-tank system in the graphical *gPROMS* model editor

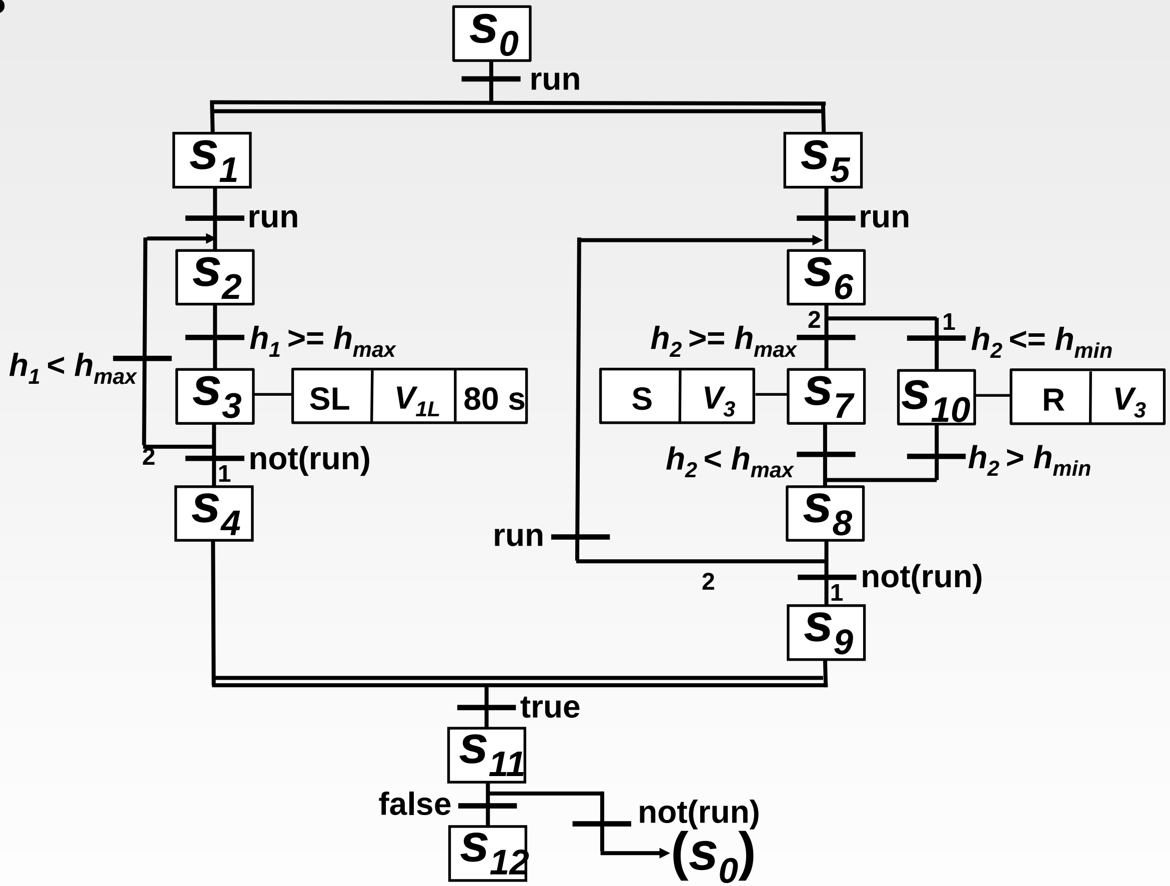
- Controlled variables: h_1, h_2
- Manipulated (discrete) variables: *V1L, V3*

Taken from:

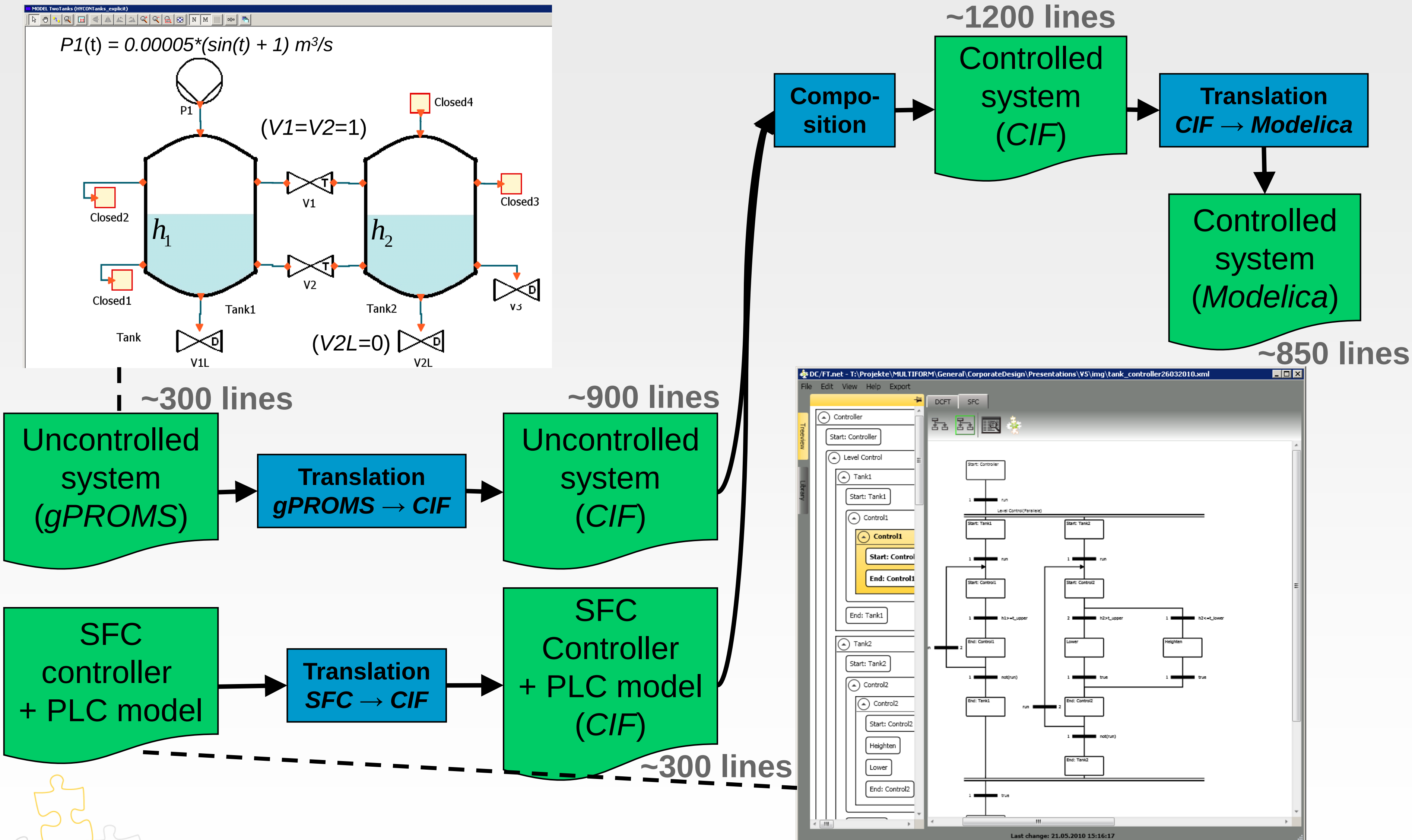


Two-tank Example: *SFC* Controller

- The SFC controller keeps the filling levels h_1 and h_2 between h_{min}=0.2 m and h_{max}=0.5 m
- If h_1 exceeds h_{max}, valve *V1L* is opened for 80s
- If h_2 exceeds h_{max}, valve *V3* is opened until h_2 falls below h_{min}



Two-tank Example: Transformation Tool Chain



DC/FT*: Software tool for the systematic refinement of informal specifications into *SFCs

```
model TwoTanks_SFC () =
|[
extern var Tanks_DOT_Tank1_DOT_h: cont real
```

```
::
|(
 mode v_tr1 = when l_u now do (c_c,t_rem) := (0, t_c - (time mod t_c)) goto v_tr2
, v_tr2 = when c_c >= t_rem now do (opt_V1,opt_V2,R_V1,R_V2,l_R_V2,l_u,c_c) := (0,0,false,false,true,false,0) goto v_tr3
```

```
|( //action "SL V1 End1"
```

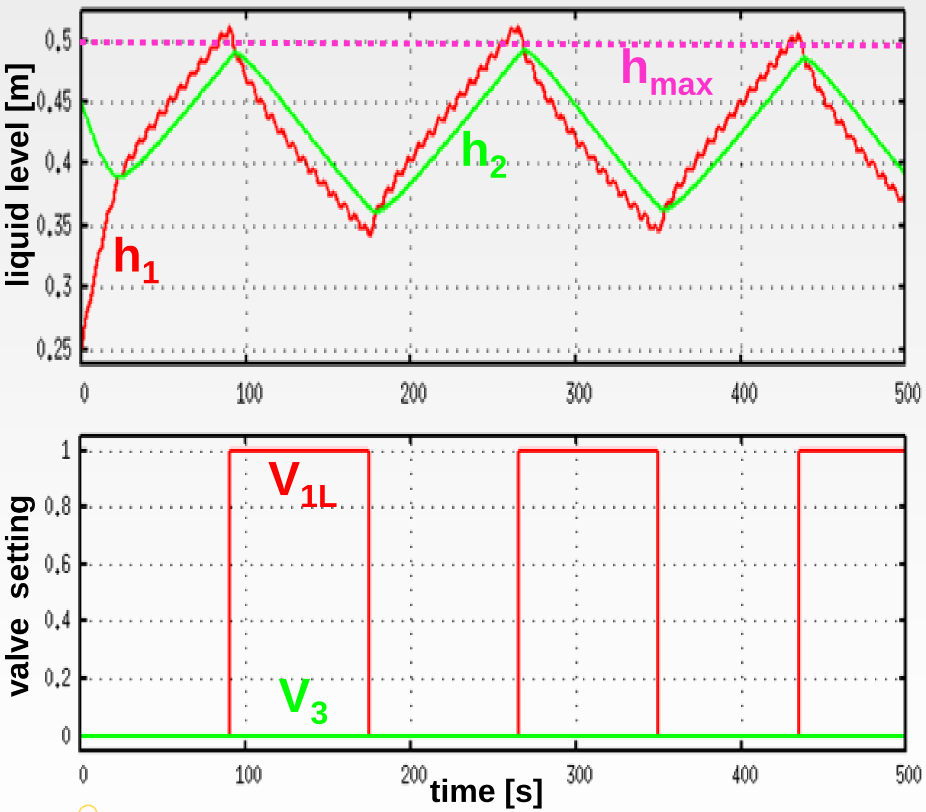
```
// Main structure automaton a_str,m
|(
```

```
||
 |(
 mode
vi_2= when l_par2 and l_str2 and not(l_strm) now do (s_S2,l_str2,not_finished2):=(true,false,true) goto v_s_S2
 when not(l_par2) and l_str2 and not(l_strm) now do (l_str2,not_finished2):=(false,false) goto vi_2
, v_s_S2= when (run) and l_str2 now do (s_Start2,s_S2,l_str2,not_finished2):=(true,false,false,true) goto v_s_Start2
 when not(run) and l_str2 now do (l_str2,not_finished2):=(false,false) goto v_s_S2
, v_s_Start2= when (Tanks_DOT_Tank2_DOT_h<=t_lower) and l_str2 now do (s_Hei2,s_Start2,l_str2,not_finished2):=(true,false,false,true)
goto v_s_Hei2
 when (Tanks_DOT_Tank2_DOT_h>t_upper) and not(Tanks_DOT_Tank2_DOT_h<=t_lower)  and l_str2 now do
(s_Low2,s_Start2,l_str2,not_finished2):=(true,false,false,true) goto v_s_Low2
 when not(Tanks_DOT_Tank2_DOT_h<=t_lower or Tanks_DOT_Tank2_DOT_h>t_upper) and l_str2 now do
(not_finished2,l_str2):=(false,false) goto v_s_Start2
, v_s_Hei2= when (Tanks_DOT_Tank2_DOT_h>t_lower) and l_str2 now do (s_End2,s_Hei2,l_str2,not_finished2):=(true,false,false,true)
goto v_s_End2
 when not(Tanks_DOT_Tank2_DOT_h>t_lower) and l_str2 now do (not_finished2,l_str2):=(false,false) goto v_s_Hei2
, v_s_Low2= when (Tanks_DOT_Tank2_DOT_h<=t_upper) and l_str2 now do (s_End2,s_Low2,l_str2,not_finished2):=(true,false,false,true)
goto v_s_End2
 when not(Tanks_DOT_Tank2_DOT_h<=t_upper) and l_str2 now do (not_finished2,l_str2):=(false,false) goto v_s_Low2
, v_s_End2 = when (not(run)) and l_str2 now do (s_FEnd_1,s_End2,l_par2,l_str2,not_finished2):=(true,false,false,false,false) goto vi_2 //
last step
 when (run) and not(not(run))  and l_str2 now do (s_Start2,s_End2,l_str2,not_finished2):=(true,false,false,true) goto v_s_Start2
 when not(not(run) or run) and l_str2 now do (not_finished2,l_str2):=(false,false) goto v_s_End2
// last step, return to start (step variable is deactivated by main automaton)
:: vi_2
)|
//SFC end
```

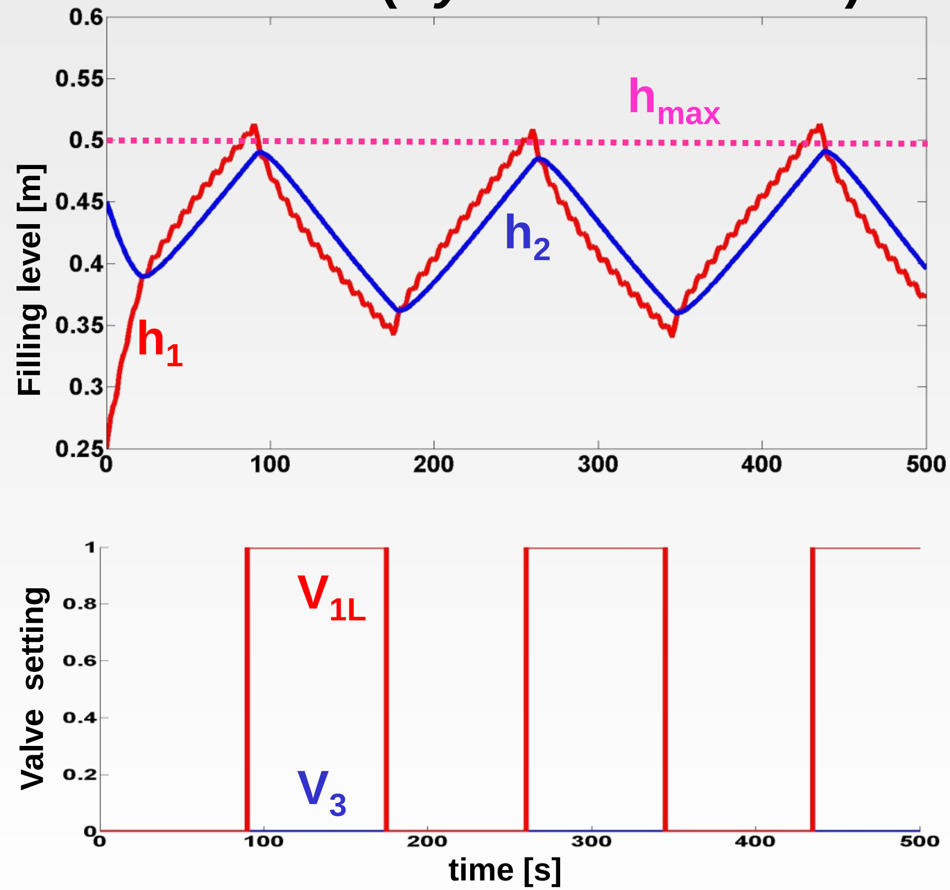
```
:: vi_1
)| // structure automaton
```

Two-tank Example: Simulation Results

CIF simulator



Modelica (*Dymola simulator*)

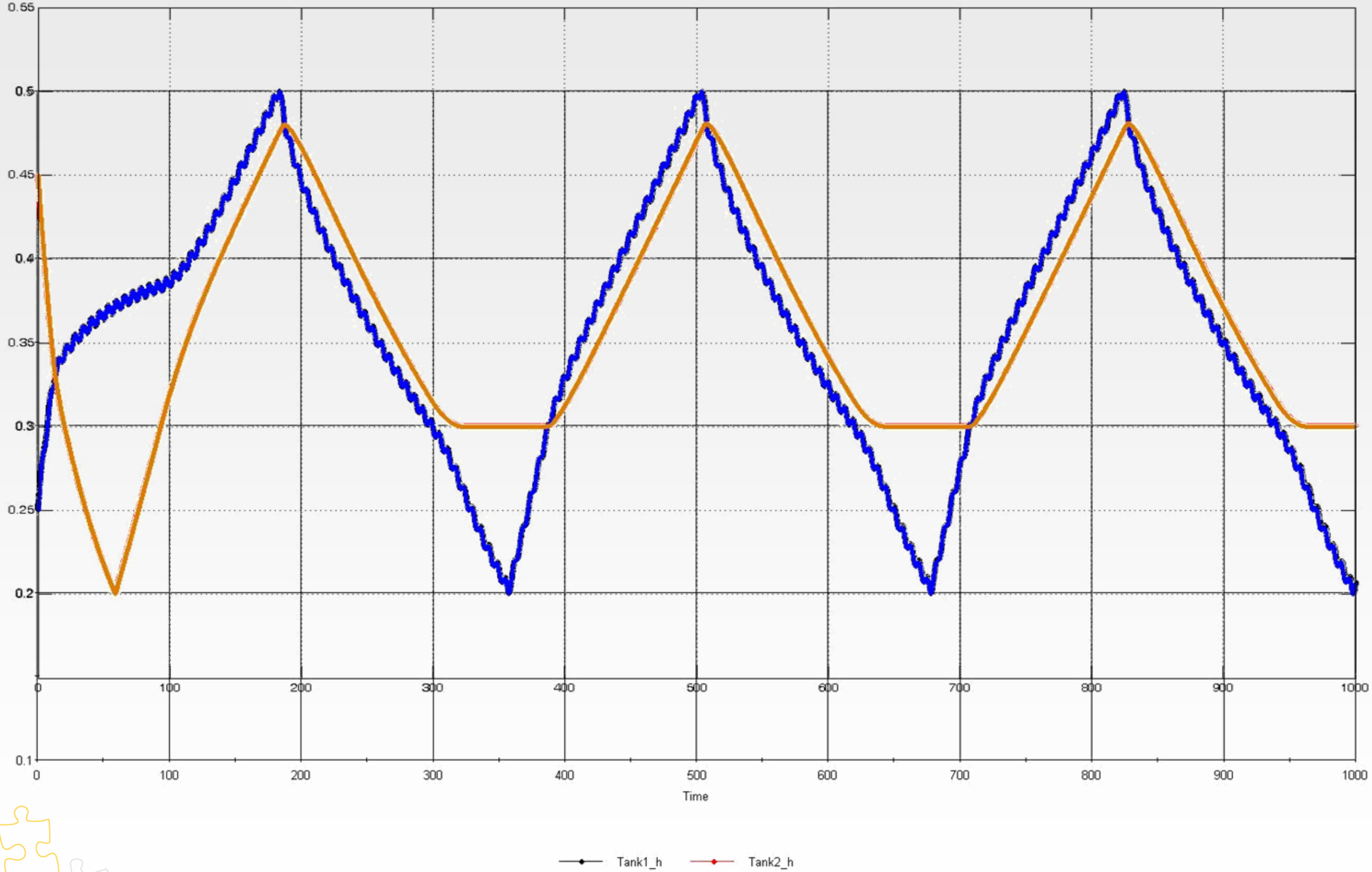


multiform

Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

24

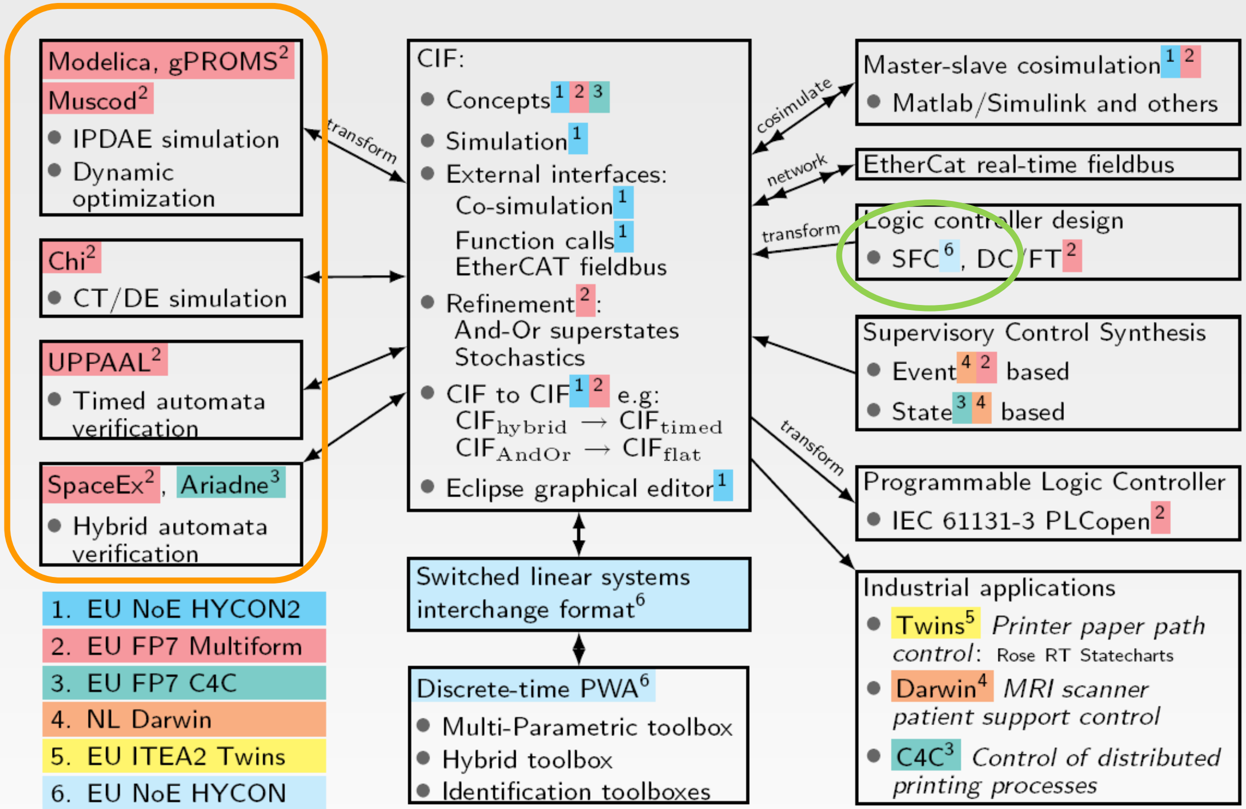
Output Identical



multiform

Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems

Compositional Interchange Format (CIF)



http://se.wtb.tue.nl/sewiki/cif/start

References:

Fischer, S.; Hüfner, M.; Sonntag, C.; Engell, S.: Systematic Generation of Logic Controllers in a Model-based Multi-formalism Design Environment. Proc. 18th IFAC World Congress, 28.08.-02.09.2011, 12490-12495.

Hendriks, D.; Schiffelers, R.; Hüfner, M.; Sonntag, C.: A Transformation Framework for the Compositional Interchange Format for Hybrid Systems. Proc. 18th IFAC World Congress, 28.08.-02.09.2011, 12509-12514.

Sonntag, C.; Hüfner, M.: On the Connection of Equation- and Automata-based Languages: Transforming the Compositional Interchange Format to Modelica. Proc. 18th IFAC World Congress, 28.08.-02.09.2011, 12515-12520.

Equation-based vs. Automaton-based Formalisms

- Simulation/Solver/Tool options encoded in model code (e.g. *EcosimPro*, *gPROMS*)
  - Tool specific options cannot be transformed
    ➔ Other tools might not find a solution for a difficult initialization problem

- Formal semantics not available ➔ Transformation not provably correct

- Equation-based models can be more restrictive than automata models
  - E.g. *Modelica* enforces globally and locally balanced models
    ➔ Automata models need to be preprocessed
    - Either by flattening of the model
    - Or by rebuilding the automata structure in an equation-based formalism

Summary

- There is a need for efficient model-based support of the design of complex automated systems with trans-level propagation and iteration, and re-use of models
- An all-encompassing mega-tool for the design of complex automated systems is not realistic, so several tools and modeling formalisms must be used in the design process.
- Three different routes to tool and model integration and design support were pursued in MULTIFORM:
  - Model exchange and tool chains via the *CIF*
  - Direct coupling of tools for testing of specifications
  - Propagation of parameters via the Design Framework

Lessons and Challenges from MULTIFORM

- The *CIF* and its tool set are stable and relatively mature
- Available under open source licence
- The effort for developing model transformations is high
- Transformations from the CIF in most cases can only be performed for subsets of the models which can be represented in the formalism.
  - A **formal** specification of the the supported *CIF* subset of a tool is needed
- It should be possible to trace elements of a model after the transformation
- Model blow-up is not as bad as could be expected

Lessons and Challenges from MULTIFORM (2)

- The *CIF* is very expressive and well suited for model exchange between automata-based formalisms, but conceptually different from equation-oriented languages (e.g. *Modelica*, *gPROMS*, *EcosimPro*)
  - **Possible solution:** Use a *Modelica* subset as an exchange formalism for equation-oriented languages, bridge equation- and automata-oriented formalisms via the *CIF* ↔ *Modelica* transformation
- Often only some elements of a system are modeled precisely, and these models are formulated in different formalisms (*fragmented modeling*)
  - How can the interdependencies between model fragments be **formally** described and exploited?
- **Model ontology needed**
  - Specification of model formalism expressivity using a common formal vocabulary
  - Equipping model artifacts with meta data on their origin(s) → traceability
  - Description of relations of partial models to an overall model