multiform

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

Using *OpenModelica* for the Translation of *Modelica* Models to the Compositional Interchange Format for Hybrid Systems

Martin Hüfner, Christian Sonntag, Adalat Jabrayilov
Process Dynamics and Operations Group (BCI-DYN)
Technische Universität Dortmund
Germany

technische universität dortmund

DYN

2nd OpenModelica Annual Workshop, Feb 8th, 2010, Linköping University

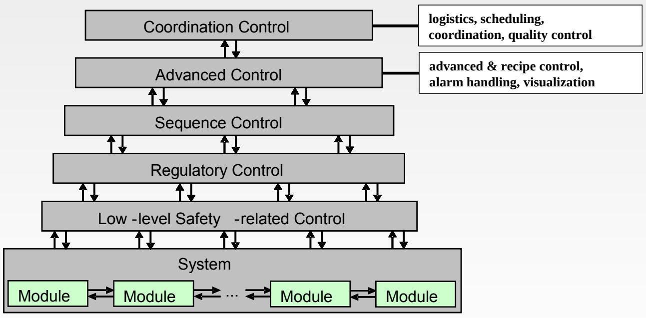
Outline

- Motivation: The goal of the MULTIFORM project
- The Compositional Interchange Format for Hybrid Systems (CIF)
- Translation of *Modelica* models to the CIF
  - Preprocessing using *OMC*
  - Variable sections
  - Equation sections
  - Algorithm sections
- Application examples
- Conclusions & Outlook

multiform

The Goal of the MULTIFORM Project

- Extend the model-based approach beyond the scope of classical feedback controller design to cover the complete control hierarchy.

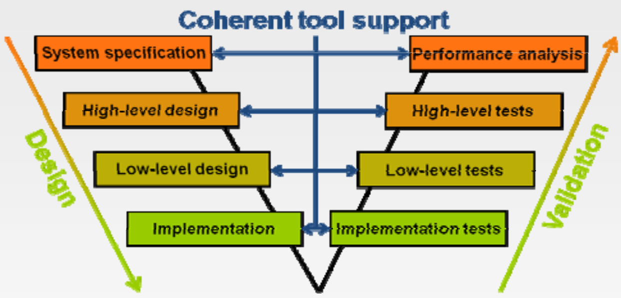


- The long-term goal: support a fully model-driven design process of a controlled system over its full life cycle.

multiform

3

Trans-level Tool Support



- Offering tool support over the complete design-cycle
- Re-use and extension of models rather than creating new ones
- Offering the right tool for the current task

➔ Shortening the design process while increasing the quality

- Model exchange via the *Compositional Interchange Format*

multiform

4

The *Compositional Interchange Format**

- Compact and powerful interchange format for general hybrid systems
- Based on hybrid automata in parallel composition
- Main features
  - Formal and compositional semantics allow property-preserving model transformations
  - Differential-algebraic equations (possibly discontinuous)
  - Hierarchy and modularity
    - Closed and open scopes
    - Automata instantiation
  - Support for different synchronization concepts
    - Synchronization by means of actions and channels
    - Shared variables
  - Support for different urgency concepts
  - Support for different representations
    - XML exchange format
    - Human-readable concrete format
    - Abstract format

multiform

* Developed by the Systems Engineering Group, TU Eindhoven

CIF Closed Scope

```
(<closed scope identifier> : )?
|[ //variable, clock, action label (act), and channel (chan) declarations optional:
   {extern|intern|input|output} var <identifier> : {disc|alg|cont} {real|int|bool|nat} (= <initial value>)?
   {extern|intern} clock <clock identifier>
   {extern|intern} act <act identifier>
   {extern|intern} chan <chan identifier> {send|recv}? : {real|int|bool|nat}

   //connection statements (optional)
   connect( <identifier>, <instantiated aut name>.<identifier> )
::
   //further inner closed scope, open scopes, or automata instantiations
   {   {openScope}*
     | { {closedScope}*
     | <instantiated aut name> : <aut identifier>(<optional parameter identifiers>?)
     }
   }+
]|
```

Variable declarations

Connect sets

Parallel open or closed scopes

multiform

CIF Open Scope

```
|(
   //optional initial equations
   (init (<equation>)+,)?

   //modes
   {mode <mode identifier> =
        // equations within modes
        ({inv|tcp|flow} <equations>)*
        // transitions
           ((when <expression>)? now? (act <transition label>)?
           (do <variable identifier> (:)?= <expressions>)?
           goto <mode identifier>)*
   }+
   :: <mode identifier> // starting mode
)|
```

Optional variable initialization

Discrete modes / locations

Differential-algebraic equations

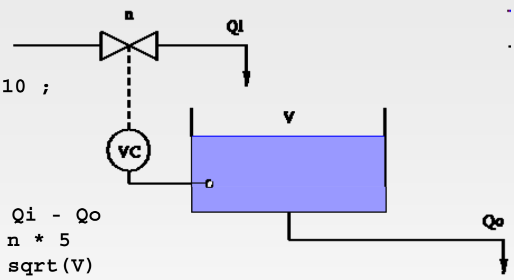
Discrete transitions

Initial discrete mode

7

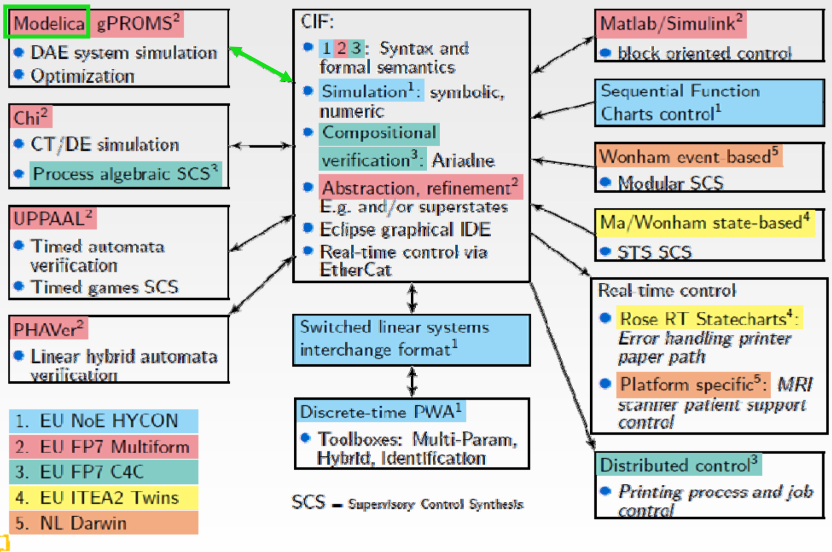
CIF Example: A Tank Controller

```
model TankController() =
|[ // variable declaration:
   extern var V: cont real = 10 ;
         Qi, Qo: alg real ;
              n: disc nat = 0
  ::
   |( // model invariants:
   mode physics = inv dot V = Qi - Qo
                        & Qi = n * 5
                        & Qo = sqrt(V)
   :: physics
   )|
 || // parallel composition
   |( // discrete controller switchings:
   mode closed = when V <= 2 now do n := 1 goto opened
      , opened = when V >= 10 now do n := 0 goto closed
   :: closed
   )|
]|
```



8

CIF Tool Connections*



* Courtesy of Bert van Beek, Systems Engineering Group, TU Eindhoven

multiform

Translation from *Modelica* to the CIF

1. Preprocessing using the *OpenModelica Compiler (OMC)*
   - Removal of complex syntactical elements (e.g. instantiation etc.)
   - Inclusion of referenced model libraries
   - Assurance that the model is syntactically correct
   - The translator is „robust“ to smaller syntactical changes in the languages
2. Parsing of preprocessed model using *ANTLR*
   - Adapted grammar from Modelica 3.0 specifications
   - Automatically generated lexer and parser
   - Decoupling of parsing and translation
3. Translation
   - Recursive top-down translation of a hierarchical model starting from highest Modelica *model*
   - Recursive translation of algorithmic model parts into CIF open scopes

multiform

Preprocessing using the *OpenModelica Compiler*

- *OMC* removes most of the advanced syntactical content of the model:
  - Classes, Predefined Types, and Declarations
  - Scoping, Name Lookup, and Flattening
  - Interface or Type Relationships
  - Inheritance, Modification, and Redeclaration
  - Connectors and Connections
  - Arrays
  - The *Modelica Standard Library* (i.e. resolving references)
- Returns a flattened representation of the original model

```
fclass IDENT
[element_list]       // public elements
                     // (variables, parameters, constants, etc.)
[equation]           // equation section
[initial equation]   // section with initial equations
[algorithm]          // algorithm section
[initial algorithm]  // section with initial algorithms
end IDENT;
```

multiform

Translation of *Modelica* Variable Sections

- Replacement of dots "." in non-top-level variables with "_DOT_"
- Real, Integer, and Boolean types are present in the CIF
- Enumerations have to be modeled using integer variables
- Discrete-time variables, constants and parameters are translated to discrete CIF variables (keyword *disc*)

Modelica: `parameter Real Tanks.t_upper = 0.5;`

CIF: `Tanks_DOT_t_upper : disc real = 0.5`

multiform

Translation of *Modelica* Variable Sections

- Replacement of dots "." in non-top-level variables with "_DOT_"
- Real, Integer, and Boolean types are present in the CIF
- Enumerations have to be modeled using integer variables
- Discrete-time variables, constants and parameters are translated to discrete CIF variables (keyword *disc*)
- Continuous-time variables are translated to
  - algebraic variables (keyword *alg*) if they do not occur differentiated
  - continuous variables (keyword *cont*) if they occur differentiated.

Modelica:

```
parameter Real Tanks.t_upper = 0.5;
Real Tanks.Tank1.flowTop.h0;
Real Tanks.Tank1.h;
```

CIF:

```
  Tanks_DOT_t_upper : disc real = 0.5
; Tanks_DOT_Tank1_DOT_flowTop_DOT_h0 : alg real
; Tanks_DOT_Tank1_DOT_h : cont real
```

multiform

Translation of *Modelica* Equation Sections (I)

- Expressions and operators can be translated by adapting them to the CIF syntax
- A *Modelica model* is translated into a single CIF model that is composed of open CIF scopes in parallel composition

Modelica:

```
fclass TwoTanks.TwoTanks
   …
end TwoTanks.TwoTanks;
```

CIF:

```
model TwoTanks_DOT_TwoTanks () =
|[
  //declarations
  …
  //inner open scopes in parallel
::  |(  …
    )|
   ||
    |(  …
    )|
   …
]|
```

multiform

Translation of *Modelica* Equation Sections (I)

- Expressions and operators can be translated by adapting them to the CIF syntax
- A *Modelica model* is translated into a single CIF model that is composed of open CIF scopes in parallel composition
- Continuous (i.e. unconditional) equations are directly translated into an open CIF scope, which contains a single mode

Modelica:

```
der(Tanks.Tank2.h) =
 ( Tanks.Tank2.flowTop.vol_flow
 + Tanks.Tank2.flowBottom.vol_flow)
 / Tanks.Tank2.A;
```

CIF:

```
dot Tanks_DOT_Tank2_DOT_h =
 ( Tanks_DOT_Tank2_DOT_flowTop_DOT_vol_flow
 + Tanks_DOT_Tank2_DOT_flowBottom_DOT_vol_flow)
 / Tanks_DOT_Tank2_DOT_A
```

multiform

15

Translation of *Modelica* Equation Sections (I)

- Expressions and operators can be translated by adapting them to the CIF syntax
- A *Modelica model* is translated into a single CIF model that is composed of open CIF scopes in parallel composition
- Continuous (i.e. unconditional) equations are directly translated into an open CIF scope, which contains a single mode
- *initial equation* sections are transferred to the *init* section of the open CIF scope that hold all continuous (unconditional) equations

Modelica:

```
initial equation
        Tanks.Tank1.h = 0.25;
        Tanks.Tank2.h = 0.45;
```

CIF:

```
|(
 init Tanks_DOT_Tank1_DOT_h = 0.25
    & Tanks_DOT_Tank2_DOT_h = 0.45,
 mode equation ...
|)
```

multiform

16

Translation of *Modelica* Equation Sections (II)

- Conditional equations are translated to **if-then-else** constructs
- **If-then-else** constructs are translated to separate open CIF scopes
  - Simple **if-then-else** constructs
    - Each branch is represented by a single mode in the open scope containing the equations of that branch

Modelica:

```
if Tanks.V2.q == 1 then
  Tanks.V2.vol_flow = 3.0;
else
  Tanks.V2.vol_flow = 0.0;
end if;
```

CIF:

```
|(
 mode IF_0 = tcp false inv true
         when (Tanks_DOT_V2_DOT_q = 1)
                   now goto IF_1
         when (not(Tanks_DOT_V2_DOT_q = 1))
                   now goto IF_2
 , IF_1 = inv Tanks_DOT_V2_DOT_vol_flow = 3.0
         when (not(Tanks_DOT_V2_DOT_q = 1))
                   now goto IF_2
 , IF_2 = inv Tanks_DOT_V2_DOT_vol_flow = 0.0
         when (((Tanks_DOT_V2_DOT_q = 1 )))
                   now goto IF_1
 :: IF_0
)|
```

17

Translation of *Modelica* Equation Sections (II)

- Conditional equations are translated to **if-then-else** constructs
- **If-then-else** constructs are translated to separate open CIF scopes
  - Simple **if-then-else** constructs
    - Each branch is represented by a single mode in the open scope containing the equations of that branch
    - Transitions between the modes ensure immediate switching if the valuation of the Boolean predicates changes

Modelica:

```
if Tanks.V2.q == 1 then
  Tanks.V2.vol_flow = 3.0;
else
  Tanks.V2.vol_flow = 0.0;
end if;
```

CIF:

```
|(
 mode IF_0 = tcp false inv true
         when (Tanks_DOT_V2_DOT_q = 1)
                   now goto IF_1
         when (not(Tanks_DOT_V2_DOT_q = 1))
                   now goto IF_2
 , IF_1 = inv Tanks_DOT_V2_DOT_vol_flow = 3.0
         when (not(Tanks_DOT_V2_DOT_q = 1))
                   now goto IF_2
 , IF_2 = inv Tanks_DOT_V2_DOT_vol_flow = 0.0
         when (((Tanks_DOT_V2_DOT_q = 1 )))
                   now goto IF_1
 :: IF_0
)|
```

18

Translation of *Modelica* Equation Sections (II)

- Conditional equations are translated to **if-then-else** constructs
- **If-then-else** constructs are translated to separate open CIF scopes
  - Simple **if-then-else** constructs
    - Each branch is represented by a single mode in the open scope containing the equations of that branch
    - Transitions between the modes ensure immediate switching if the valuation of the Boolean predicates changes
  - Nested **if-then-else** constructs
    - A tree of modes is constructed to switch according to the conditions of the **if-then-else** constructs
    - All higher-level equations are shifted to the leafs of the tree
    - One leaf is always active
- **When-elsewhen** constructs
  - Modeled as CIF open scopes
  - Boolean variables represent the states of the Boolean predicates

multiform

19

Translation of *Modelica* Algorithm Sections (I)

- Algorithms are modeled in a single open CIF scope
- The translation of each statement has a single CIF starting mode and a single CIF end mode

Modelica:

```
when Tanks.Tank1.h >= Tanks.t_upper then
   Tanks.V1L_u := 1.0;
end when;
```

The CIF:

```
|(
mode ALGORITHM_0 = when
      (Tanks_DOT_Tank1_DOT_h >= Tanks_DOT_t_upper)
         now goto ALGORITHM_1
  , ALGORITHM_1 = tcp false now do
       Tanks_DOT_V1L_u := 1.0
         goto ALGORITHM_2
  , ALGORITHM_2 = when
     (not(Tanks_DOT_Tank1_DOT_h >= Tanks_DOT_t_upper))
         now goto ALGORITHM_0
 :: ALGORITHM_0
)|
```

multiform

20

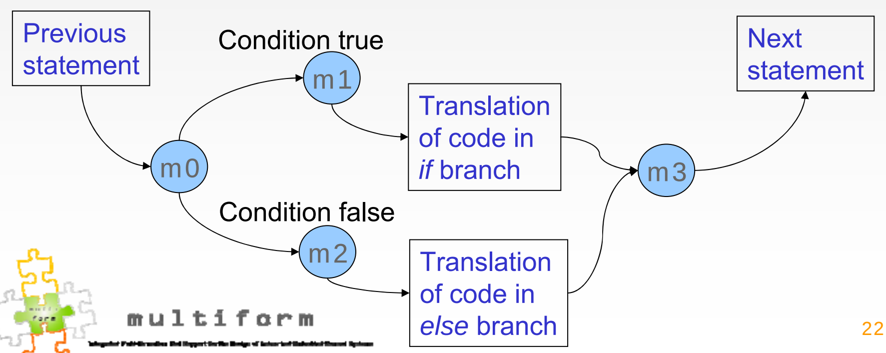
Translation of *Modelica* Algorithm Sections (I)

- Algorithms are modeled in a single open CIF scope
- The translation of each statement has a single CIF starting mode and a single CIF end mode
- A sequence of algorithmic statements is translated into a chain of modes (loops are possible)
- A depth-first recursive algorithm is employed to translate nested statements
  - Operates on a tree data structure that represents the hierarchy of the (nested) algorithmic constructs
  - Each node of the tree has an unique ID that is used to generate unique mode names in the CIF

multiform

Translation of *Modelica* Algorithm Sections (II)

- **If-then-else** construct
  - Starting mode (m0) is given from previous algorithmic statement
  - Modes for *if* branch (m1) and *else* branch (m2)
  - Recursive translation algorithm is invoked for the statements of the if- and else-body
  - End mode (m3) is returned to the invoking instance of the recursive algorithm



multiform

Translation of *Modelica* Algorithm Sections (II)

- **If-then-else** construct
  - Starting mode (m0) is given from previous algorithmic statement
  - Modes for *if* branch (m1) and *else* branch (m2)
  - Recursive translation algorithm is invoked for the statements of the if- and else-body
  - End mode (m3) is returned to the invoking instance of the recursive algorithm
- **While-do** construct
  - Is similarly translated as the **if-then-else** construct
  - A simple transition from *else* branch to end mode (with guard *true*) is added
  - Returned end mode is connected from the last statement to the starting mode
- **For** construct
  - Similar to the translation of the **while-do** construct
  - Equipped with additional counting variables

multiform

23

Translation of *Modelica* Algorithm Sections (III)

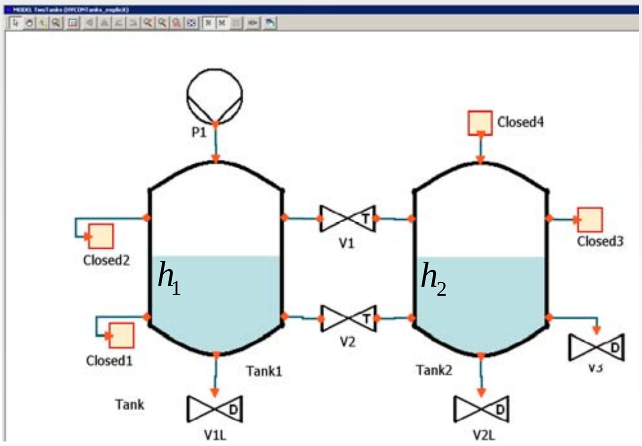
- **Assignments**
  - A new mode *m1* is added to the open CIF scope
  - An urgent transition *m0* → *m1* (with guard *true* and variable resets according to the statement*)* resets the variables
  - *m1* is returned as the end mode of the translation
- **terminate()**
  - CIF does not provide facilities to terminate the simulation → an artificial deadlock is created
  - A new mode *m1* is added to the open CIF scope in which time cannot progress (*tcp false)*
  - No transition from *m1* is added
- **reinit()**
  - Translated like an assignment because the CIF does not differentiate between state variables and algebraic variables in reset/reinitialization operations

multiform

24

Example 1: Hybrid Controlled Two-Tank System

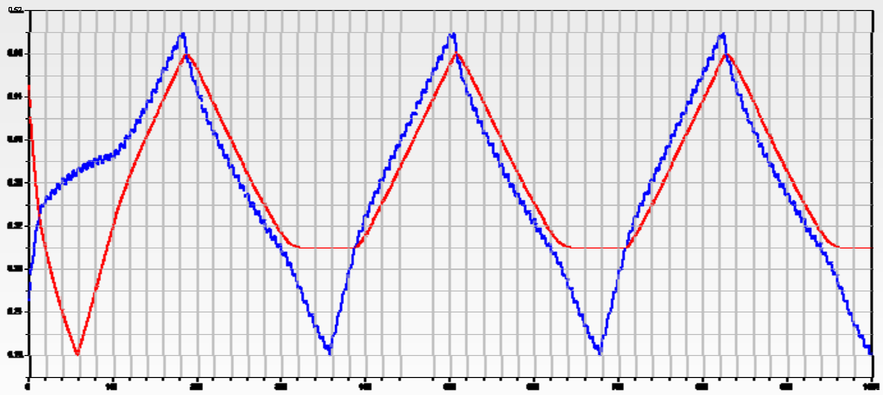
- Discretely controlled hybrid two-tank system[2]
  - Designed to contain many constructs of equation-based languages



Discrete controllers
Parallel algorithms that switch V1L and V3 depending on h_1 and h_2

[2] C. Sonntag: Modeling, Simulation, and Optimization Environments. *Handbook of Hybrid Systems Control - Theory, Tools, Applications*, 2009.

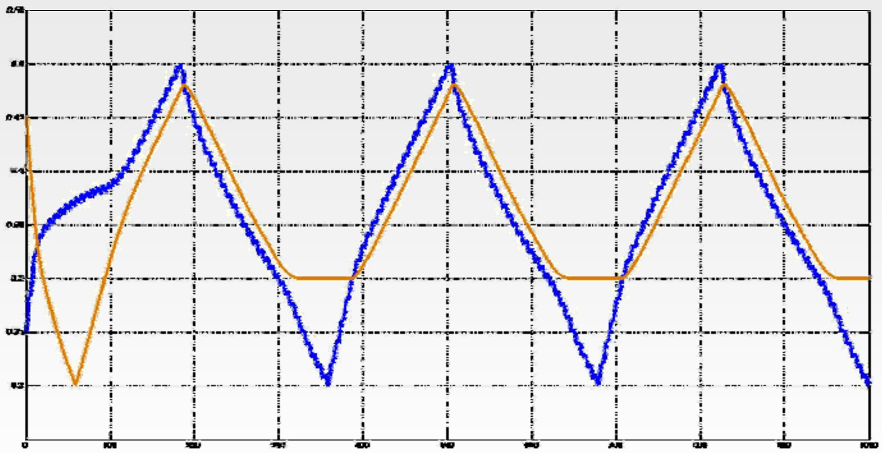
Example 1: Simulation Results



Modelica

multiform

Example 1: Simulation Results

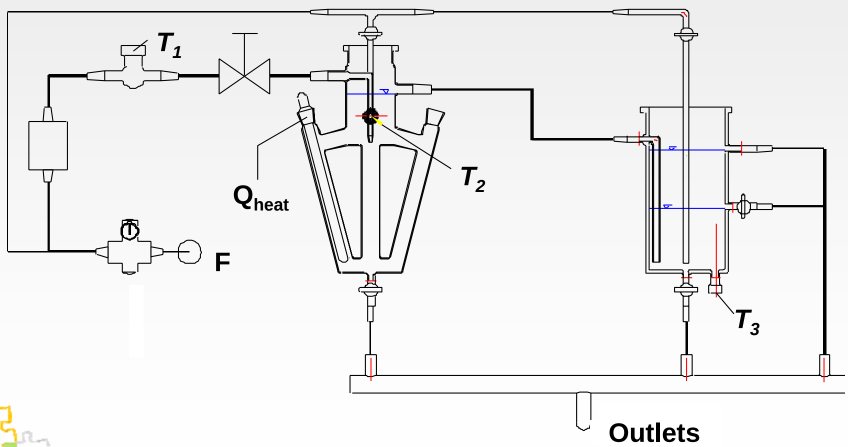


CIF

27

Example 2: Lab Plant at TU Dortmund

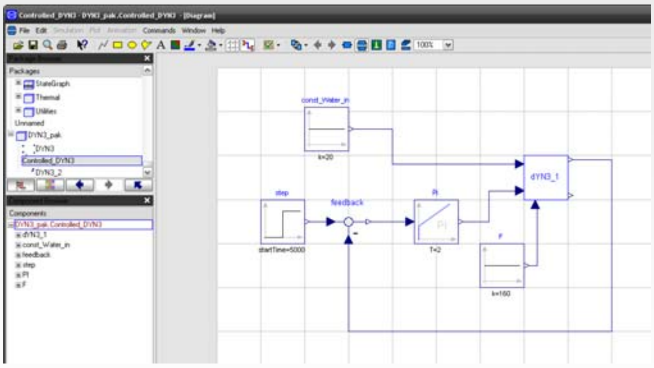
- A simple control example from the students lab at TU Dortmund
- A PI controller regulates temperature T_3 using load Q_{heat}



28

Example 2: *Modelica* Model

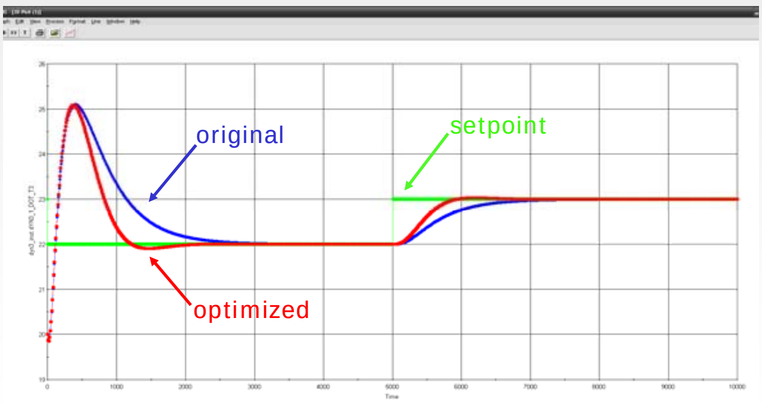
- Lab plant model created from *Modelica* standard library blocks in *Dymola*



- Automatically translated to the CIF using *OMC*
- CIF representation translated to *gPROMS*

Example 2: Dynamic Optimization with *gPROMS/gOPT*

- Dynamic optimization of controller parameters *k* and *T*
- Minimization of the integrated square error (ISE) between the set point and the temperature T_3



Conclusions

- Goals of the MULTIFORM project
  - Development of a model-based design flow framework
  - Integration of model-based design and analysis tools
    - Model exchange via the Compositional Interchange Format (CIF)
  - Propagation of design parameters and decisions between all levels of the design hierarchy
- The Compositional Interchange Format (CIF)
  - Compact and powerful interchange language for general hybrid systems
  - Main features: formal compositional semantics, hierarchy and modularity, different urgency and synchronization concepts
- Algorithmic translation from Modelica to the CIF
  - Preprocessing using the *OpenModelica Compiler* (*OMC*)
    - Flattening, syntactical simplification, inclusion of library components
  - Recursive top-down translation to the CIF

multiform

Outlook

- Planned extensions
  - Improvement of the CIF language (simplified formal semantics etc.)
    - CIF core language (almost) finalized
  - Extension of the CIF with support for co-simulation
    - Inclusion of (external) function calls in equation/algorithm sections
  - Extension of the CIF with pure time delays
  - Translation of meta information
    - Annotations, units etc.
- MATLAB-based CIF simulator (➔ Simulink integration)
- Goal: Direct support for the translation in *OpenModelica*
  - Adaptation of the preprocessor to retain more structural information that can be translated to the CIF
    - Hierarchical and modular models
  - Cooperation with Open-Source Modelica Consortium (OSMC), Linköping University
- MULTIFORM: Cooperation with ITEA2 project OPENPROD

multiform

Acknowledgements

This work has been performed as part of the project
Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems (MULTIFORM)
that is funded within the Seventh Research Framework Programme of the European Commission.

Grant agreement number: INFSO-ICT-224249

http://www.ict-multiform.eu/

Information on the **CIF**
Toolset, syntax, examples and publications are available at:
http://se.wtb.tue.nl/sewiki/cif/general