# A Library for Synchronous Control Systems in Modelica

Martin Otter[1], Berhard Thiele[1], Hilding Elmqvist[2]
[1]DLR Institute of System Dynamics and Control, D-82234 Wessling, Germany
[2]Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden
Martin.Otter@dlr.de, Bernhard.Thiele@dlr.de, Hilding.Elmqvist@3ds.com

## Abstract

Based on the synchronous language elements introduced in Modelica 3.3, a library is described to utilize the new features in a convenient way for graphical model definition of sampled data systems. The library has elements to define periodic clocks and event clocks that trigger elements to sample, sub-sample or super-sample partitions synchronously. Optionally, quantization effects, computational delay or noise can be simulated. Continuous-time equations can be automatically discretized and utilized in a sampled data system. This is demonstrated by using the inverse of a nonlinear plant model in the feed forward path of a discrete controller of a mixing unit.

*Keywords: Synchronous models, sampled data systems, periodic systems, clock, inverse systems*

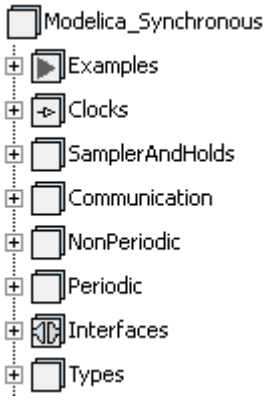
## 1 Introduction

In the Modelica language version 3.3 (*Modelica Association 2012*) synchronous language features have been introduced to precisely define and synchronize sampled data systems with different sampling rates. This paper is a companion paper to (*Elmqvist et.al. 2012*) which should be first inspected to understand why new language elements have been introduced, as well as the syntax and semantics of them.

The new language elements follow the synchronous approach (*Benveniste et. al. 2002*). They are based on the clock calculus and inference system proposed by (*Colaco and Pouzet 2003*) and implemented in Lucid Synchrone version 2 and 3 (*Pouzet 2006*). However, the Modelica approach also uses multi-rate periodic clocks based on rational arithmetic introduced by (*Forget et. al. 2008*), as an extension of the Lucid Synchrone semantics. Additionally, the built-in operators of Modelica 3.3 also support non-periodic and event based clocks[1].

In order to utilize these elements in an actual model in a *convenient way*, a free library "Modelica_Synchronous" has been developed using a prototype of Dymola (*Dassault Systèmes 2012*) for the new language elements. This library is in a prototype status. After an evaluation period it is planned to include this library into the Modelica Standard Library. Note, all Modelica libraries designed so far for sampled systems, such as Modelica.Blocks.Discrete, Modelica_LinearSystems2.Controller (*Baur et. al. 2009*) and Modelica_EmbeddedSystems (*Elmqvist et.al. 2009*) are becoming obsolete and should be replaced by this new library.

In the figure to the right a screenshot of the library is shown with the first sub library level. The most important sub libraries are:




- **Clocks**:
  Library of blocks that generate clocks.
- **SamplerAndHolds**:
  Library of blocks that sample, sub-sample, super-sample and hold signals.
- **NonPeriodic**:
  Library of blocks that operate on periodically and non-periodically clocked signals (the blocks depend explicitly on the *actual* sample interval).
- **Periodic**:
  Library of blocks that are designed to operate only on periodically clocked signals, mainly described by z transforms (the blocks do not *explicitly* depend on the sample period, but implicitely, since the block parameters need to be designed for one specific sample period).

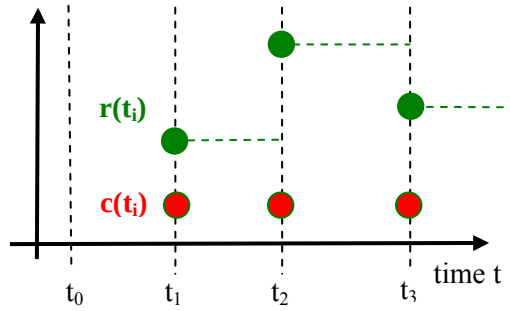
In the following subsections, the most important blocks are discussed and their usage demonstrated in examples.

## 2 Clocks

A "Clock" is a new base data type introduced in Modelica 3.3 (additionally to Real, Integer, Boolean, String) that defines when a particular partition of equations of a model is active. Every variable and every equation is either continuous-time or is associ-

[1] A non-periodic clock is defined by a varying interval and an event clock by a Boolean condition.

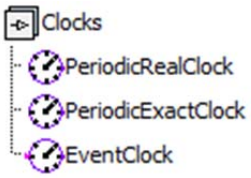
ated exactly to one clock (*Elmqvist et.al. 2012*). This feature is visualized in the figure below where $c(t_i)$ is a clock that is active at particular time instants and $r(t_i)$ is a variable that is associated to this clock. A clocked variable has only a value when the corresponding clock is active:




Similarly to RealInput, RealOutput etc., clock input and output connectors are defined in sub library "Interfaces" in order to propagate clocks via connections:

| *Icon* | *Modelica Definition* |
|---|---|
|  | `connector ClockInput  = input Clock;` |
|  | `connector ClockOutput = output Clock;` |

Sub library "Clocks", see screenshot to the right, defines the following components that generate clocks, and provide the respective clock via its ClockOutput connector to other components:

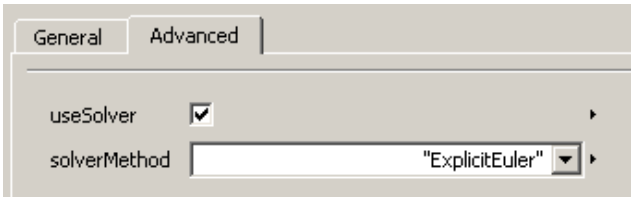



- **PeriodicRealClock** defines a periodic clock where the period is defined with a Real number (e.g. "period = 0.1" for 0.1 s). If clocks are related relatively to each other (see section 4), then only one of them can be a PeriodicRealClock.
- **PeriodicExactClock** defines a periodic clock with a resolution defined by enumeration "Types.Resolution" (with values "y, d, h, min, s, ms, us, ns") and an integer multiple "factor" of this resolution. For example "factor = 3" and "resolution = Types.Resolution.ms" defines a periodic clock with sample period 3 ms.
- **EventClock** defines a clock that is active when the Boolean input to this component changes from false to true.

The implementation of these clocks is a direct mapping to the new clock generators. Example:

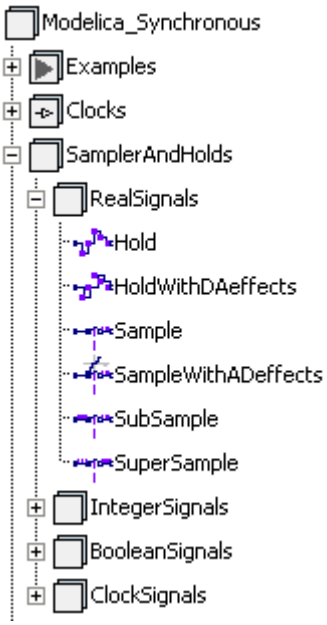
```
block PeriodicRealClock
  parameter Modelica.SIunits.Time period;
  extends Modelica_Synchronous.Interfaces.PartialClock;
equation
  y = Clock(period);
end PeriodicRealClock;

partial block PartialClock
  parameter Boolean useSolver = true
      annotation(Dialog(tab="Advanced"));
  parameter Modelica_Synchronous.Types.SolverMethod
      solverMethod="External"
      annotation(Dialog(tab="Advanced",enable=useSolver));
  Modelica_Synchronous.Interfaces.ClockOutput y;
end PartialClock;
```

All these clocks have an "Advanced" menu in which an optional integration method (such as "explicit Euler method") can be associated to the clock, see next figure. The effect of such a definition will be explained below.




## 3 Sample and Hold

Within the sub library "SamplerAndHolds" various blocks are defined to sample, sub-sample, super-sample and hold signals. Since Modelica does not have generic types, for every base type a separate sub-library is present, such as SamplerAndHolds.RealSignals, see screenshot to the right. All these components define boundaries between different partitions, especially:




- **Sample** requires that the input signal is continuous-time. The block samples the input and provides it as clocked output signal. The equations that have a dependency to that output, are collected/grouped into the same clocked partition.
- **Hold** requires that the input signal is clocked and provides it as continuous-time signal to the output with a zero order hold. Before the first tick of the clock that is associated to the input, the output is set to parameter `y_start` (this value is also displayed in the icon, see Figure 1).


DOI 10.3384/ecp1207627

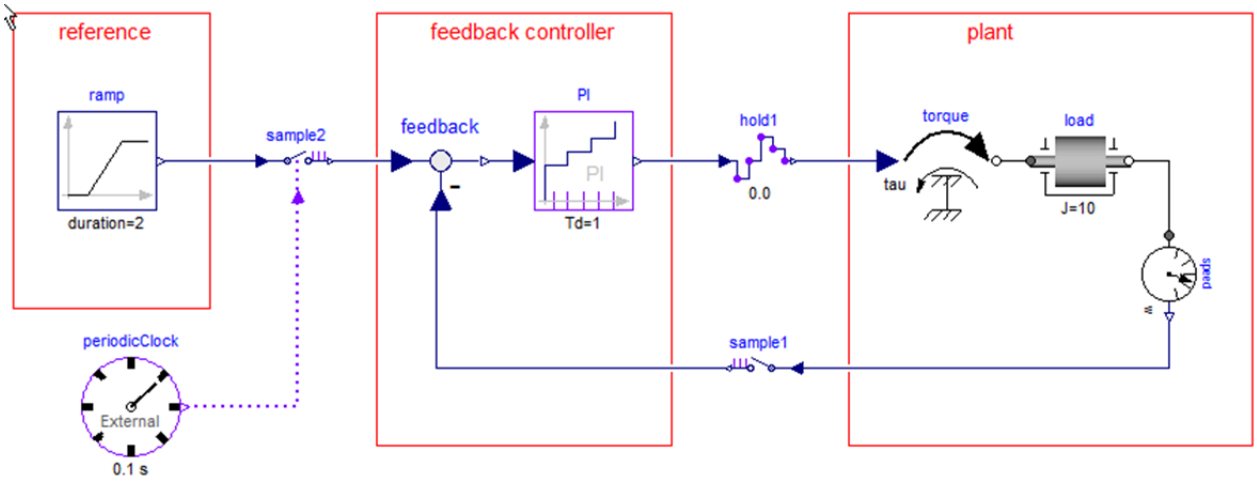



*Figure 1: Simple drive train with clocked PI controller, samplers, hold and periodic clock.*

- **SampleWithADeffects, HoldWithDAeffects** are similar to Sample and Hold, but provide additionally the options to simulate particular effects, such as noise, signal limitations and quantization effects, as well as computational delays.

The Sample and Hold blocks have again a direct mapping to the corresponding new language elements. For example, the RealSignals.Sample block is implemented as:

```
block Sample
  parameter Boolean useClock=false;
  Modelica.Blocks.Interfaces.RealInput    u;
  Modelica.Blocks.Interfaces.RealOutput y;
  Modelica_Synchronous.Interfaces.ClockInput
                                          clock if useClock;
protected
  Modelica_Synchronous.Interfaces.ClockInput c_internal;
equation
  connect(clock, c_internal);
  if useClock then
    y = sample(u,c_internal);
  else
    y = sample(u);
  end if;
end Sample;
```

With the default option `useClock=false`, just the input `u` is sampled, `y = sample(u)`, and the clock of the output `y` is deduced by clock inference due to the clock definition somewhere else (*Elmqvist et.al. 2012*).

If `useClock=true`, the input clock connector `clock` is enabled and the clock propagated to this connector is used as clock for the output: `y=sample(u,clock)`, see block `sample2` in Figure 1.

Figure 1 demonstrates all blocks that have been discussed so far within an illustrative example model. This model consists of a load inertia that is driven by a torque. The goal is to control the speed of the inertia. For this, a feedback controller is provided in form of a periodic sampled data system described with clocked equations. The reference part is again a continuous-time model and provides the desired speed of the inertia.

The boundaries of the feedback controller are defined with components `sample1`, `sample2` and `hold1` that are instances of blocks `Sample` and `Hold` respectively. All equations inside this partition ("feedback controller") need to be associated to a clock. For this, the `Sample` block has an optional `ClockInput` connector that can be enabled. In the figure, a periodic clock with period 0.1 s is connected to `sample2` and therefore the "feedback controller" partition is active every 0.1 s. Note, it would also be fine to connect the clock additionally to `sample1`, since associating the same clock definition several times to a partition is allowed.

The PI component is a clocked block from Modelica_Synchronous.NonPeriodic. It is implemented as (note, `previous(x)` defines that `x` is clocked and that the value from the previous clock tick is used; `interval(u)` is the time duration from the previous to the actual clock tick as Real number):

```
block PI  "From Modelica_Synchronous.NonPeriodic"
  extends Modelica_Synchronous.Interfaces.PartialClockedSISO;
  parameter Real k "Gain of continuous PI controller";
  parameter Real T "Time constant of continuous PI controller";
  output Real x(start=0) "Discrete PI state";
protected
  Real Ts = interval(u) "Sample period";
equation
  x = previous(x) + u*Ts/T;
  y = k*(x + u);
end PI;
```

This PI controller is parameterized with the coefficients of a continuous-time PI controller and with the actual sample period the coefficients of the discretized (clocked) PI controller are computed. Changing

the sample period will therefore result in a similar controller behavior.

It would also be possible to utilize the PI controller from the Modelica_Synchronous.Periodic sub-library. In this sub-library it is assumed that the blocks are utilized only with periodic clocks and the block parameters have been designed for a particular sample period. The corresponding PI controller is implemented as:

```
block PI "From Modelica_Synchronous.Periodic"
  extends  Modelica_Synchronous.Interfaces.
                              PartialPeriodicallyClockedSISO;
  parameter Real kd "Gain of discrete PI controller";
  parameter Real Td "Time constant of discrete PI controller";
  output Real x(start=0) "Discrete PI state";
equation
  x = previous(x) + u/Td;
  y = kd*(x + u);
end PI;
```

The PI coefficients `kd` and `Td` are designed for a particular sample period. Changing this sample period, without changing `kd` and `Td`, will significantly change the controller behavior.

It would also be possible to use a *continuous-time* block, in particular the continuous-time PI controller from Modelica.Blocks.Continuous.PI that is basically implemented as:

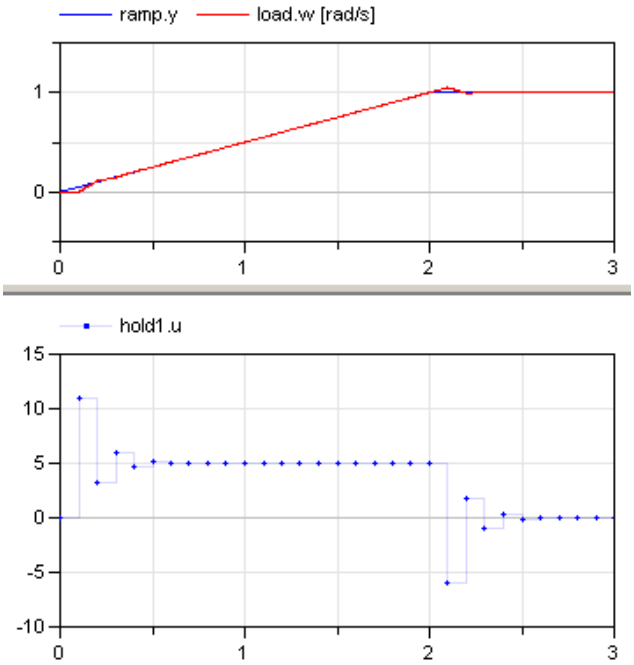
```
block PI "From Modelica.Blocks.Continuous "
  parameter Real k=1 "Gain";
  parameter Modelica.SIunits.Time T "Time Constant";
  extends Modelica.Blocks.Interfaces.SISO;
  output Real x "State of block";
equation
  der(x) = u/T;
  y = k*(x + u);
end PI;
```

In this case the PI controller is described by a differential equation. Since the input signal to this block is a clocked signal when present in the block diagram of Figure 1, the differential equation is automatically discretized by integrating from the previous to the actual clock tick with the integration method defined in component "periodicClock". In Figure 1, solver "External" is defined (see icon of the clock). This means that the solver defined in the simulation environment is used to integrate the continuous-time block: This might be a variable step-solver with error control where the step size is selected such that it hits the clock tick always exactly.

On the other hand, if solverMethod = "ImplicitEuler" is selected, then the differential equation of the PI component will be discretized with a fixed step implicit Euler method. This approach is also called "inline integration". For details, see (*Elmqvist et.al. 1995*). In this case exactly the same result will be obtained as with the previous two PI components.

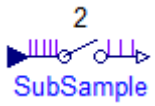
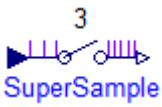
This approach is very powerful, since *every linear* or *non-linear continuous-time block* can be utilized in the clocked partition. It is therefore in many cases is is no longer necessary to derive discretized blocks manually as, e.g., done in the Modelica_LinearSystems2.Controller library (*Baur et.al. 2009*).

Typical simulation results are shown in the next figure. Note, here it is clearly visualized by Dymola, that the input to `hold1` (= `hold1.u`) is a clocked signal.



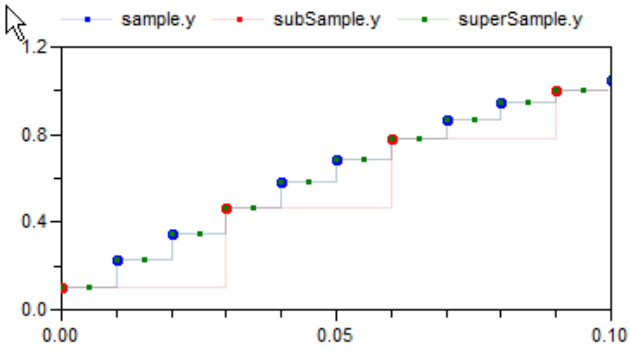

## 4 SubSample and SuperSample

With blocks "SubSample" and "SuperSample" it can be defined that a partition is sub- or super-sampled with respect to another clocked partition:

| | |
|---|---|
|  | At every "factor" ticks of the input (here: factor = 2), the output ticks and is set to the input. |
|  | At every "factor" ticks of the output (here: factor = 3), the input ticks. The output is set to the last available value of the input. |

The factor of a sub- or super-sampled partition can either be explicitly defined with the block, or it can be inferred, since either the factor is defined at another element or exact periods are associated with the partitions (see below). In the next figure an example is shown, where the signal `sample.y` is sub-sampled by a factor of 3 (= `subSample.y`) and super-sampled by a factor of 2 (= `superSample.y`).


DOI 10.3384/ecp1207627




There are now many possible ways to define the clocks of time-synchronized partitions. In Figures 2-Figure 4 on the next page some useful variants are demonstrated at hand of a cascade control system for a very simple drive system. The goal is that the load inertia travels according to the desired reference angle. This angle is defined with block KinematicPTP2 from the Modelica Standard Library (the reference signal is constructed so that it moves from a start to an end angle as fast as possible for given maximal speed and maximum acceleration). The "slow" controller part is a simple P-controller to control the position, whereas the "fast" controller part is a PI controller to control the speed.

In *Figure 2* one real periodic clock with a sample period of 0.02 s is defined. This clock is then subsampled with a factor of 5 which defines a second clock with a sample period of 0.1 s. The "slow" and the "fast" controller partitions are separated by the `super1` block (an instance of `SuperSample`) and therefore it is defined that the output of `super1` is faster than the input of `super1` (the input clock is an integer multiple of the output clock). The two defined clocks are associated with `sample3` and `super1` and therefore the clocks are associated with the partitions "slow controller" and "fast controller". Note, the factor at `super1` is inferred to be 5.

In Figure 3 only one real clock with a sample period of 0.02 s is defined. This clock is associated to the "fast controller" partition via component `super1`. Now, in component `super1` a factor of "5" is defined. This means that the fast partition is 5-times faster as the slow partition, and therefore the clock of the "slow controller" partition is implicitly defined.

In Figure 4 two "exact" clocks are defined: One clock with a period of 20 ms and one clock with a period of 100 ms. These "absolute" clocks are associated with the "slow" and "fast" partition respectively. Since component `super1` defines that the "fast" partition must be an integer factor faster as the "slow" partition, an implicit constraint is present, that the clocks of the two partitions must have periods that are an integer multiple of each other. Therefore, defining 20 ms and 100 ms is fine. However, defining periods of 30 ms and 100 ms would result in an error, since this constraint is violated.

The preferred modeling style is a matter of taste. Note, the relative definitions of Figure 2 and Figure 3 have the advantage that parameter `factor` can still be changed after the model is translated (provided a tool supports this feature). Instead, in the definition of Figure 4 it would be typically no longer possible to change the (absolute) periods after translation, since there is a constraint between the two definitions (one period must be an integer multiple of the other period).

# 5 Nonlinear Inverse Models

Since a long time, Modelica is used to model advanced nonlinear control systems. Especially, Modelica allows a *semi-automatic* treatment of *inverse nonlinear plant models*. In the fundamental article (*Looye et.al. 2005*) this approach is described and several controller structures are presented to utilize an inverse plant model in the controller. This approach is attractive because it results in a systematic procedure to design a controller for the whole operating range of a plant. This is in contrast to standard controller design techniques that usually design a linear controller for a plant model that is linearized at a specific operating point. Therefore the operating range of such controllers is inherently limited. Up to Modelica 3.2, controllers with inverse plant models can only be defined as continuous-time systems. Via the export mechanism of Dymola they could be exported with solvers embedded in the code and then used as sampled data system in other environments. However, it is not possible to re-import the sampled data system to Modelica.

The synchronous features of Modelica 3.3 together with the Modelica_Synchronous library offer now completely new possibilities, so that the inverse model can be designed and evaluated as sampled data system within Modelica and a Modelica simulation environment such as Dymola. The approach is sketched at hand of a simple nonlinear plant model of a mixing unit (*Föllinger 1998, page 279*) and the design of a nonlinear feed-forward controller according to (*Looye et.al. 2005*):

A substance A is flowing continuously into a mixing reactor. Due to a catalyst, the substance reacts and splits into several base substances that are continuously removed. The reaction generates energy and therefore the reactor is cooled with a cooling medium. The cooling temperature $T_c(t)$ in [K] is the primary actuation signal. Substance A is described by its concentration $c(t)$ in [mol/l] and its temperature $T(t)$ in [K] according to the following

**Simple Drive with cascade controller for position and speed control**

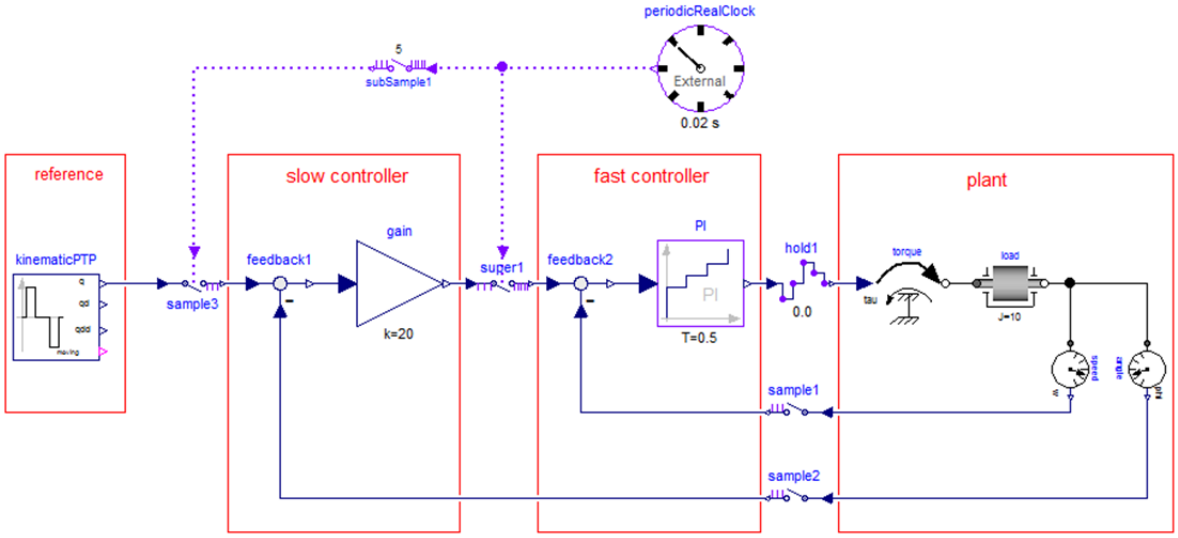



*Figure 2: Two clocks are defined with sub-sampling and partitions with super-sampling.*

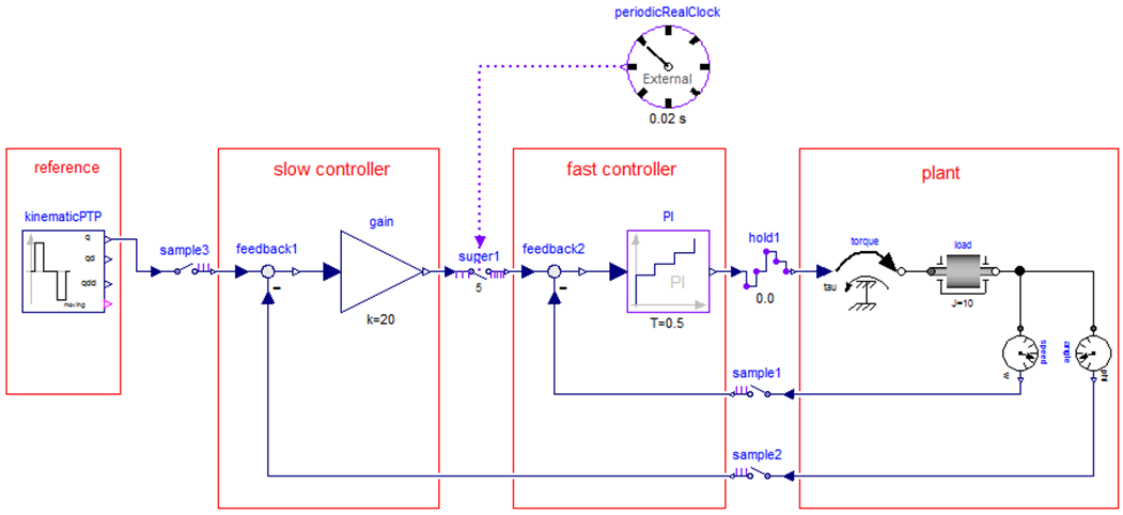



*Figure 3: One clock is defined and the second clock is inferred by the factor of the super-sample block.*

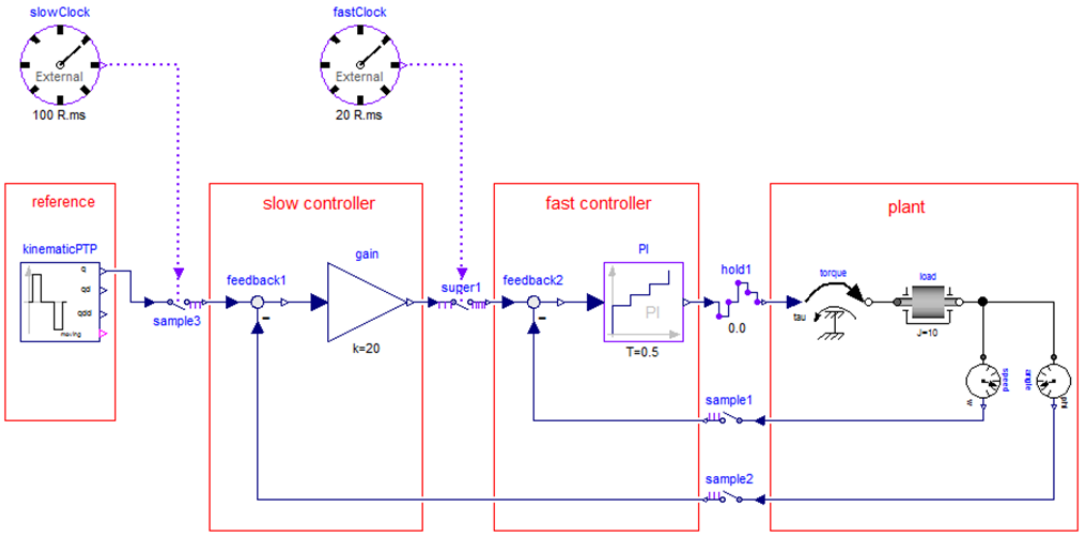



*Figure 4: Partitions are defined with exact (integer) clocks that need to be compatible to each other.*


DOI 10.3384/ecp1207627

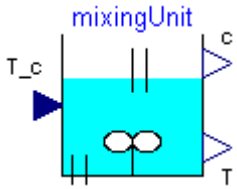
nonlinear differential algebraic equation system:

$$\begin{aligned} \gamma &= c \cdot k_0 \cdot e^{-\varepsilon/T} \\ \dot{c} &= -a_{11} \cdot c - a_{12} \cdot \gamma + a_{13} \\ \dot{T} &= -a_{21} \cdot T + a_{22} \cdot \gamma + a_{23} + b \cdot T_c \end{aligned} \quad (1)$$

with

$$\begin{aligned} k_0 &= 1.24 \cdot 10^{14} & a_{11} &= 0.00446 & a_{21} &= 0.0303 \\ \varepsilon &= 10578 & a_{12} &= 0.0141 & a_{22} &= 2.41 \\ b &= 0.0258 & a_{13} &= 0.00378 & a_{23} &= 1.37 \end{aligned}$$

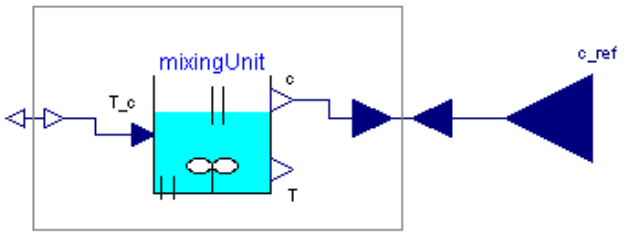
For the given input $T_c(t)$ these are 1 algebraic equation for the reaction speed $\gamma(t)$ and two differential equations for $c(t)$ and $T(t)$. The concentration $c(t)$ is the signal to be primarily *controlled* and the temperature $T(t)$ is the signal that is *measured*. These equations are collected together in an input/output block:



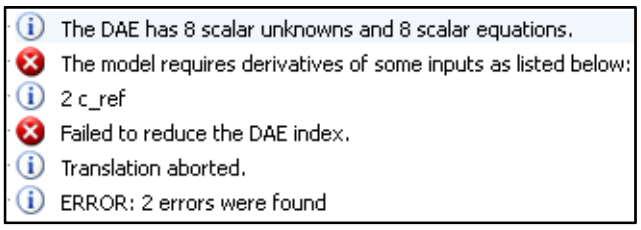

The design of the control system proceeds now in the following steps:

### 5.1 Design of Pre-Filter

Inverting a model usually means that equations need to be symbolically differentiated and that higher derivatives of the inputs are needed (that are usually not available). One approach is to filter the inputs, so that a Modelica tool can determine the derivatives of the filtered input from the filter states. The minimum needed filter order is determined by first inverting the continuous-time plant model from the variable to be primarily controlled (here: "$c$") to the actuator input (here: "$T_c$"). This is performed with the help of block "Modelica.Blocks.Math.InverseBlockConstraints" that allows connecting an external input (`c_ref` below) to an output (`c` below):




Translating this model will generate the continuous-time inverse plant model. However, Dymola gives (correctly) an error message:




This message states, that Dymola has to differentiate the model, but this requires the second derivative of the external input `c_ref` and this derivative is not available. The conclusion is that a low pass filter of at least second order has to be connected between `c_ref` and `c`, for example Modelica.Blocks.Continuous. Filter. Only filter types should be used that do not have "vibrations" in the time domain for a step input. Therefore, parameter `analogFilter` of the component should be selected as "CriticalDamping" (= only real poles), or "Bessel" (= nearly no vibrations, but steeper frequency response as "CriticalDamping"). The cut-off frequency `f_cut` is manually selected by simulations of the closed loop system. In the example, we use a CriticalDamping filter of third order (the third order is selected to get smoother signals) and a cut-off frequency of 1/300 Hz.

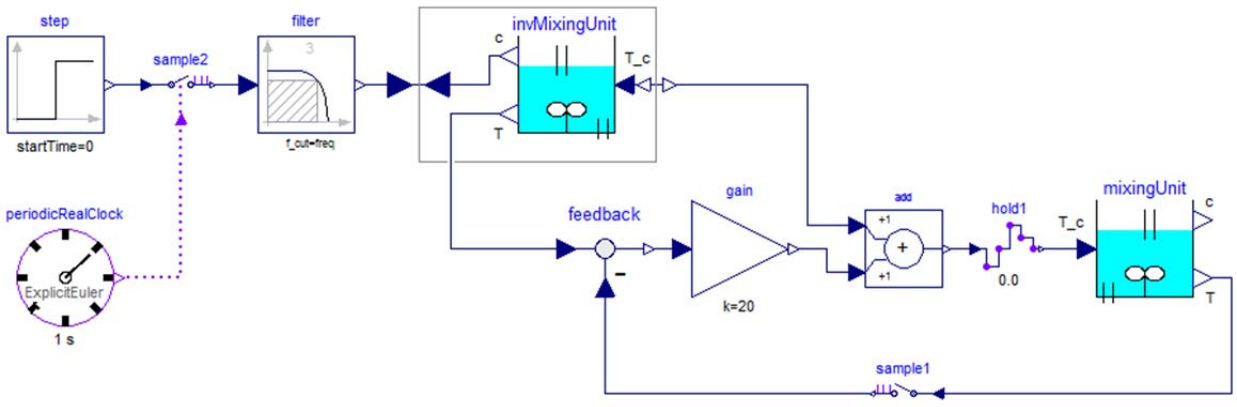



*Figure 5: Sampled data controller for mixing unit including the inverse plant model.*

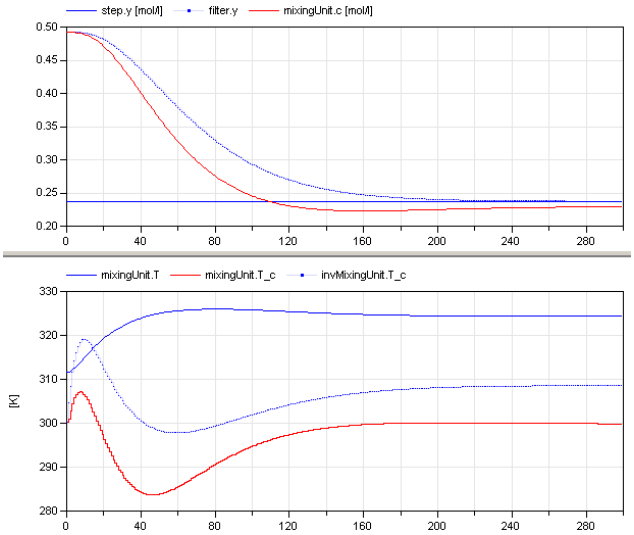
### 5.2 Design of Controller

The controller for the mixing unit is shown in Figure 5. It consists of the filter discussed in the previous section. The input to the filter is the reference concentration which is filtered by the low pass filter. The output of the filter is used as input to the concentration `c` in the inverse plant model. This model computes the desired cooling temperature `T_c` (which is used as desired cooling temperature at the output of the controller) and the desired temperature `T` (which is used as desired value for the feedback controller). This part of the control system is the "feed forward" part that computes the desired actuator signal. As feedback controller a simple P-Controller with one gain is used.

This controller could be defined as continuous-time system in previous Modelica versions. However, with Modelica 3.3 it is now also possible to define the controller as sampled data system. For this, the two inputs are sampled (`sample1` and `sample2`) and the actuator output is hold (`hold1`).

The controller partition is then associated with a periodic clock (via `sample2`) that has a sample period of 1 s and a solverMethod = "ExplicitEuler". Since the controller partition is a continuous-time system, it is discretized and solved with an explicit Euler method at every clock tick (by integrating from the previous to the actual time instant of the clock).

The controller works perfectly if the same parameters for the plant and the inverse plant model are used (follows perfectly the filtered reference concentration). Changing all parameters of the inverse plant model by 50 % (with exception of ε since the plant is very sensitive to it) still results in a reasonable control behavior as shown by the following simulation results (the desired concentration jumps from 0.492 to 0.237):




The piecewise constant (blue) curve in the upper window is the output of the filter (that is, it is the desired concentration). The red curve in the upper window is the concentration of model mixingUnit, which is the concentration in the plant. Obviously, the concentration follows reasonably well the desired one. By using a more involved feedback controller, the control error could be substantially reduced.

## 6 Event Clocks –Engine Control

All previous sections concentrated on periodic clocks. However, also non-periodic synchronous sampled data systems can be defined with Modelica 3.3. This is demonstrated at hand of a closed-loop throttle control synchronized to the crankshaft angle of an internal combustion engine. This system has the following properties:

- Engine speed is regulated with a throttle actuator.
- Controller execution is synchronized with the engine crankshaft angle.
- The influence of disturbances, such as a change in load torque, is reduced.

The complete system is shown in Figure 6. Block

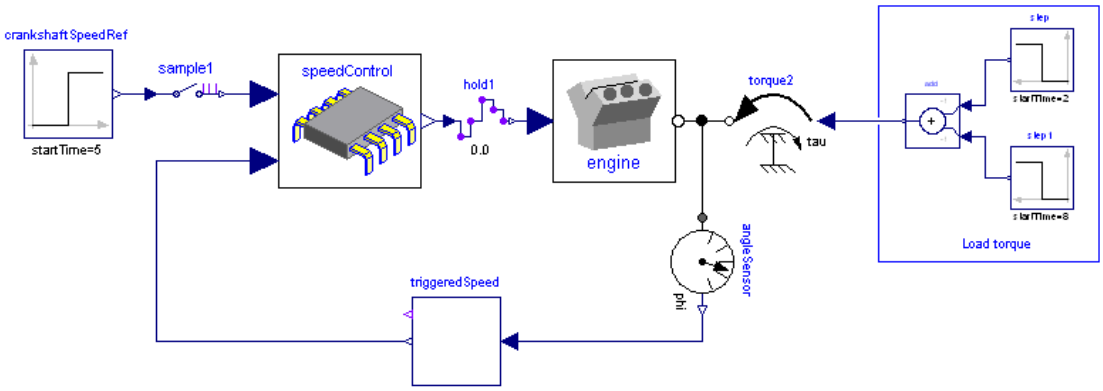



*Figure 6: Sampled data engine controller that is synchronized with the crankshaft angle.*


DOI 10.3384/ecp1207627

`speedControl` is the discrete control system. The boundaries of this controller are defined by `sample1` and `hold1`. A special element `triggeredSpeed` has the crankshaft angle as input and provides the sampled crankshaft speed as output. Additionally, the clock associated with the output (and therefore also to component `speedControl`) ticks, at every 180 degree rotation of the crankshaft angle. This special application is implemented in the text layer of component `triggeredSpeed` as:

```
N = der(angle);
when Clock(angle >= hold(offset)+Modelica.Constants.pi) then
    offset = sample(angle);
   N_clocked = sample(N);
end when;
```

Here, `N` is the derivative of the crankshaft angle. Whenever this angle becomes larger as 180 degree an event clock is activated due to `Clock(..)`. In such a case the when-clause becomes active, and the speed `N` is sampled, and the new offset for the next event is computed.

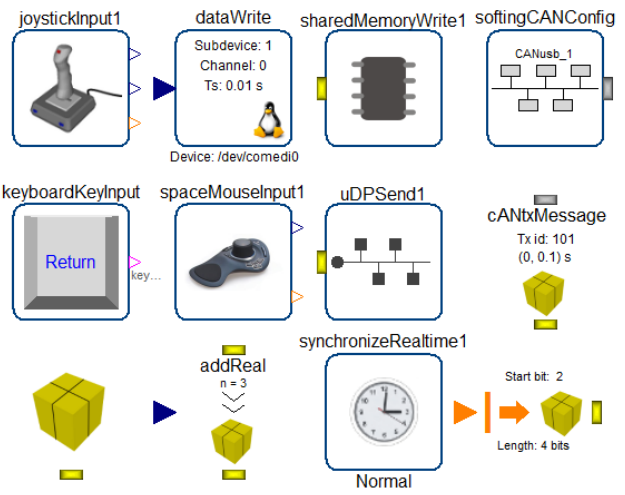
## 7 Interfaces to External Devices

Bellmann presented in (*Bellmann 2009*) a Modelica library with capabilities for creating interactive simulation models with advanced (3D-) visualization[2]. It included support for standard input devices such as keyboard and joysticks, as well as communication mechanisms like UDP or shared memory. These device interfaces have been adapted to work with the Modelica synchronous extensions, and have been extended to also support the Linux OS. Furthermore additional functionality such as support for Softing CAN interface cards[3] and the (Linux specific) Comedi[4] control and measurement device interface have been added. In the next figure some of the blocks are shown that are currently available in the external devices library.

## 8 Cyber-Physical Models

Modelica is designed for modeling of systems containing both physical parts and control systems. It is possible to hierarchically assemble a system out of smart subsystems, i.e. which includes their local control systems.

In (*Elmqvist et.al. 2009*) it is described how parts of the model which is used for evaluating the system




architecture and performance can be used for different studies and for generation of embedded code. The solution in the Modelica_EmbeddedSystems library is to introduce generic "communication blocks" between the partitions. Such communication blocks can then be configured in different ways, for example to just contain an ideal Sample block or a block with A/D effects. It can also contain a device driver for a A/D converter for the input to the discrete-time partition. It is then possible to use the code of this partition for embedding to control hardware.

If instead, the communication block contains a D/A converter, for the output of the continuous-time partition, the code for the continuous-time partition can be used for hardware-in-the-loop simulation.

The point is that this configuration can be done without changing the original model. It is done by using redeclarations of the content of the communication blocks by using a hierarchical modifier in a model extending the original model. This approach is beneficial with regards of maintaining the original model since only one version is needed.

It is planned that this technique, already evaluated in the Modelica_EmbeddedSystems library, is included in the Modelica_Synchronous library.

## 9 Summary

A new, free Modelica library is presented that provides a convenient graphical user interface for the synchronous language elements introduced in Modelica 3.3. This library is planned to replace all previous Modelica libraries designed for sampled data systems, since

- the clocking for a partition needs to be defined only at one block (and not at every block of a controller),

[2] Today the *visualization* part of that library has evolved into the commercially available product "*Visualization Library*", which is distributed by BAUSCH-GALL GmbH, http://www.bausch-gall.de/.
[3] Softing AG (2012), http://www.softing.com.
[4] The Comedi project (2012), http://www.comedi.org/.

- every continuous-time block (including inverse models) can be directly utilized in the clocked partition, thereby making it unnecessary in most cases to provide a manually implemented discrete-time version,
- errors to define the sample periods can be detected by the translator (because all variables and equations of a clocked partition must be associated exactly to one clock),
- more efficient simulation of an overall model consisting of plant (= continuous-time) and controller (= clocked partitions),
- providing the possibility to easily identifying the controller part that shall be downloaded to actual hardware (because all equations and variables of a clocked partition are associated exactly to one clock).

## 10 Acknowledgement

Sven Erik Mattsson developed the Dymola prototype supporting the synchronous features of Modelica 3.3. Without this prototype, it would not have been possible to develop the Modelica_Synchronous library.

## References

Baur M., Otter M., and Thiele B. (2009): **Modelica Libraries for Linear Control Systems**. Proceedings of 7th International Modelica Conference, Como, Italy, September 20-22. www.ep.liu.se/ecp/043/068/ecp09430068.pdf

Benveniste A., Caspi P., Edwards S.A., Halbwachs N., Le Guernic P., and Simone R. (2003): **The Synchronous Languages Twelve Years Later**. Proc. of the IEEE, Vol., 91, No. 1. www.irisa.fr/distribcom/-benveniste/pub/synch_ProcIEEE_2002.pdf

Bellmann T. (2009): **Interactive Simulations and advanced Visualization with Modelica**. Proceedings of 7th International Modelica Conference, Como, Italy, September 20-22. www.ep.liu.se/ecp/043/062/ecp09430056.pdf

Colaco J.-L., and Pouzet M. (2003): **Clocks as First Class Abstract Types**. In Third International Conference on Embedded Software (EMSOFT'03), Philadelphia, Pennsylvania, USA, October 2003. http://www.di.ens.fr/~pouzet/lucid-synchrone/papers/emsoft03.ps.gz

Dassault Systèmes (2012): Dymola. http://www.Dymola.com

Elmqvist H., Otter M. and Cellier F.E. (1995): **Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems**. Keynote Address, Proceedings ESM'95, European Simulation Multiconference, Prague, Czech Republic, June 5-8, pp. xxiii-xxxiv. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6E666F4221CFED902DCA7BDF8DC51AB6?doi=10.1.1.127.3787&rep=rep1&type=pdf

Elmqvist H., Otter M., Henriksson D., Thiele B., Mattsson S.E. (2009): **Modelica for Embedded Systems**, Proceedings 7th Modelica Conference, Como, Italy, Sep. 20-22. http://www.ep.liu.se/ecp/043/040/ecp09430096.pdf

Elmqvist H., Otter M., and Mattsson S.E. (2012): **Fundamentals of Synchronous Control in Modelica**. Proceedings of 9th International Modelica Conference, Munich, Germany, Sep. 3-5.

Föllinger O. (1998): **Nichtlineare Regelungen I**, Oldenbourg Verlag, 8. Auflage.

Forget J., F. Boniol, D. Lesens, C. Pagetti (2008): **A Multi-Periodic Synchronous Data-Flow Language**. In 11th IEEE High Assurance Systems Engineering Symposium (HASE'08), Dec. 3-5 2008, Nanjing, China, pp. 251-260. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reoad=true&arnumber=4708883&contentType=Conference+Publications

Looye G., Thümmel M., Kurze M., Otter M., and Bals J. (2005): **Nonlinear Inverse Models for Control**. Proceedings of 4th International Modelica Conference, ed. G. Schmitz, Hamburg, March 7-8. https://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3c3.pdf

Modelica Association (2012): **Modelica Language Specification Version 3.3**. https://www.modelica.org/documents/ModelicaSpec33.pdf.

Pouzet M. (2006): **Lucid Synchrone, Version 3.0, Tutorial and Reference Manual**. http://www.di.ens.fr/~pouzet/lucid-synchrone/

 DOI 10.3384/ecp1207627