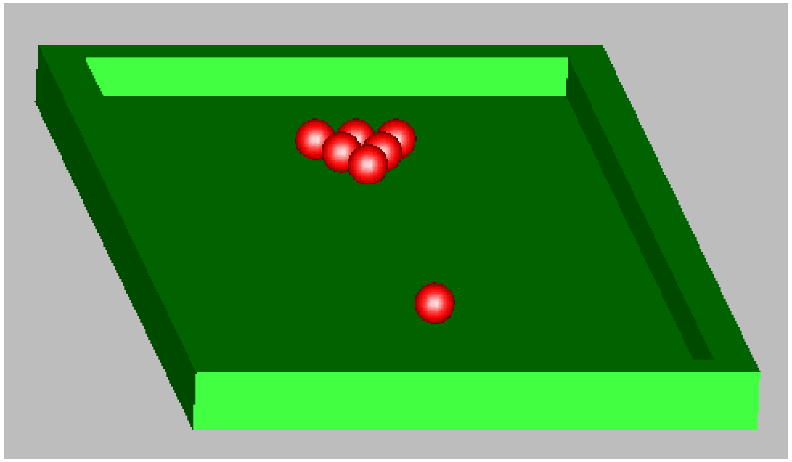
Discrete Events and Hybrid Systems

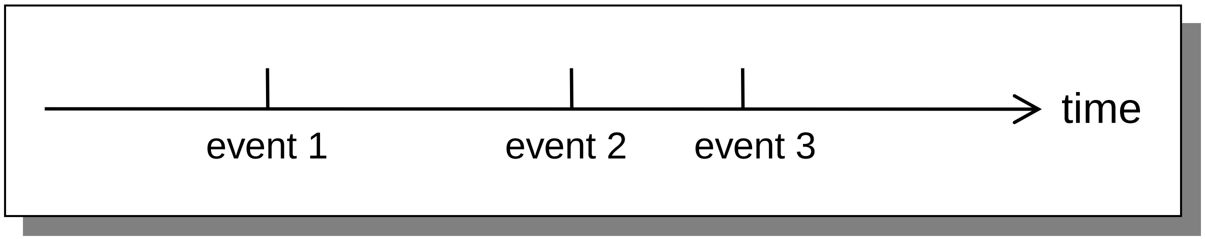


Picture: Courtesy Hilding Elmqvist

MODELICA

Events

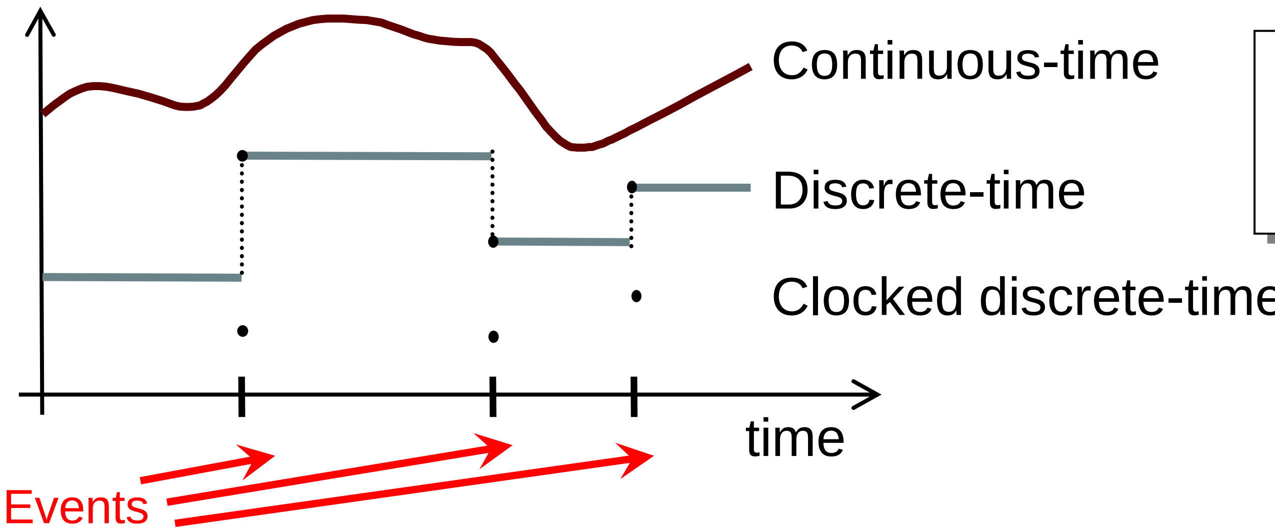
Events are ordered in time and form an event history



- A *point* in time that is instantaneous, i.e., has zero duration
- An *event condition* that switches from false to true in order for the event to take place
- A set of *variables* that are associated with the event, i.e. are referenced or explicitly changed by equations associated with the event
- Some *behavior* associated with the event, expressed as *conditional equations* that become active or are deactivated at the event. *Instantaneous equations* is a special case of conditional equations that are only active *at* events.

Modelica Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling




```
Real x;
Voltage v;
Current i;
```

```
discrete Real x;
Integer i;
Boolean b;
```

- A *point* in time that is instantaneous, i.e., has zero duration
- An *event condition* or *clock tick* so that the event can take place
- A set of *variables* that are associated with the event
- Some *behavior* associated with the event, e.g. *conditional equations* that become active or are deactivated at the event

Event Creation – if

if-equations, if-statements, and if-expressions

```
if <condition> then
  <equations>
elseif <condition> then
  <equations>
else
  <equations>
end if;
```

```
model Diode "Ideal diode"
  extends TwoPin;
  Real s;
  Boolean off;
equation
  off = s < 0;
  if off then
    v=s
  else
    v=0;
  end if;
  i = if off then 0 else s;
end Diode;
```

False if `s<0`

If-equation choosing equation for `v`

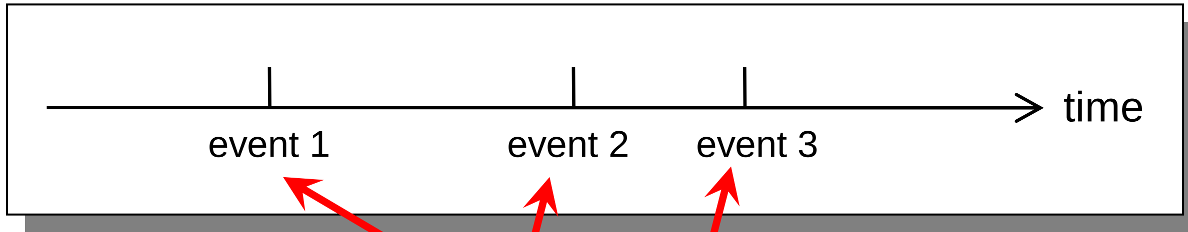
If-expression

Event Creation – when

when-equations *(two kinds: unclocked and clocked)*

when <*conditions*> **then**
<*equations*>
end when; // un-clocked version

when *clock* **then**
<*equations*>
end when; // clocked version



Equations only active at event times

Time event

```
when time >= 10.0 then
  ...
end when;
```

Only dependent on time, can be scheduled in advance

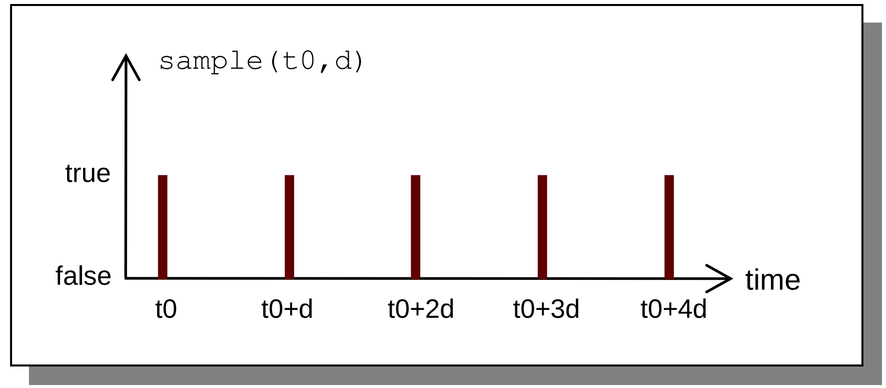
State event

```
when sin(x) > 0.5 then
  ...
end when;
```

Related to a state. Check for zero-crossing

Generating Repeated Events by unclocked sample

The call `sample(t0,d)` returns true and triggers events at times `t0+i*d`, where `i=0,1, …`

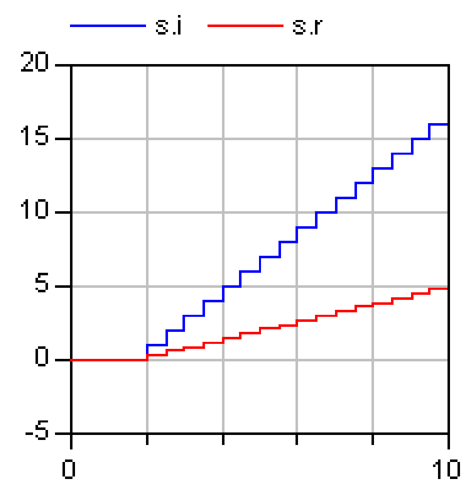


Variables need to be discrete

```
model SamplingClock
  Integer i;
  discrete Real r;
equation
  when sample(2,0.5) then
    i = pre(i)+1;
    r = pre(r)+0.3;
  end when;
end SamplingClock;
```

Creates an event after 2 s, then each 0.5 s

pre(...) takes the previous value before the event.

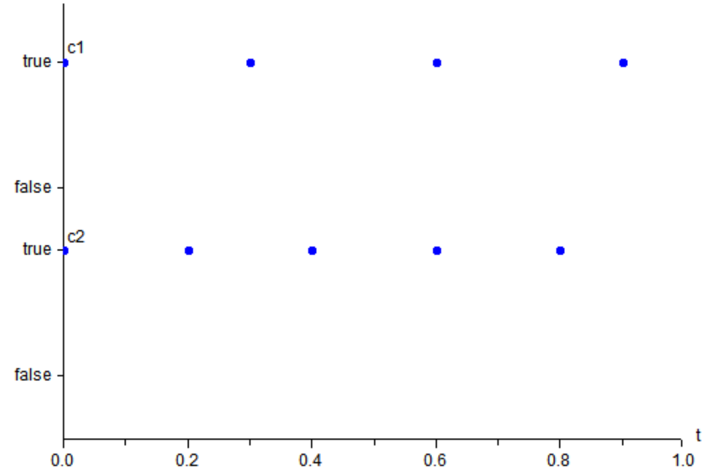


Generating Clock Tick Events using Clock()

(clocked models, Modelica 3.3 and later)

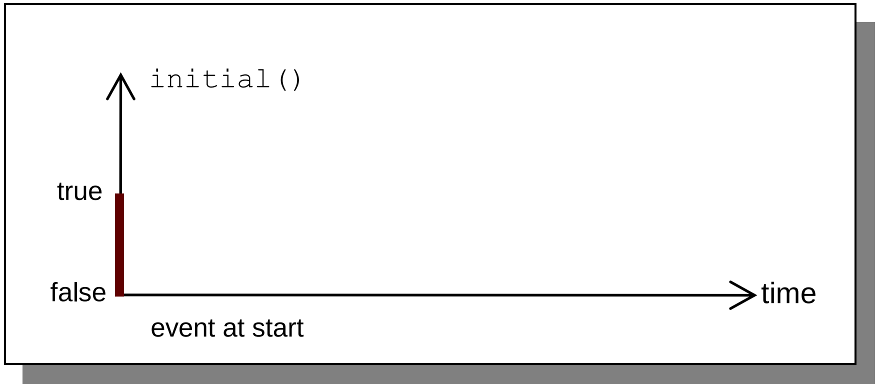
- Clock( ) – inferred clock
- Clock(intervalCounter, resolution) – clock with
  Integer quotient (rational number) interval
- Clock(interval) – clock with a Real value interval
- Clock(condition, startInterval)
- Clock – solver clock

```
class ClockTicks
  // Integer quotient rational number interval clock
  Clock c1 = Clock(3,10);        // ticks: 0, 3/10, 6/10, ..
  // Clock with real value interval between ticks
  Clock c2 = Clock(0.2);         // ticks: 0.0, 0.2, 0.4, ...
end ClockTicks;
```

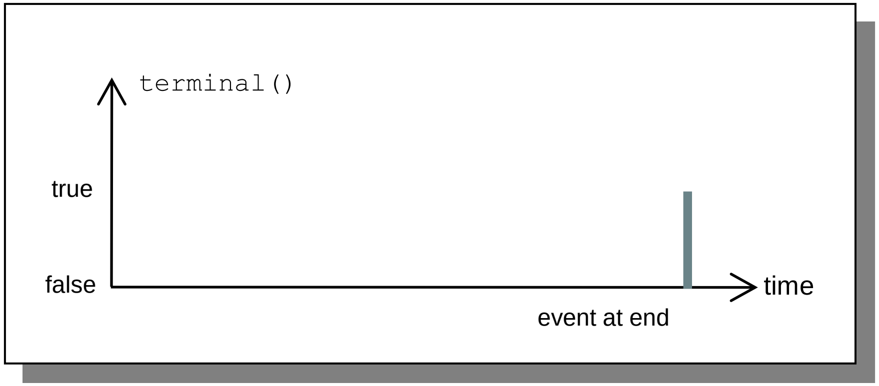


initial and terminal events

Initialization actions are triggered by `initial()`



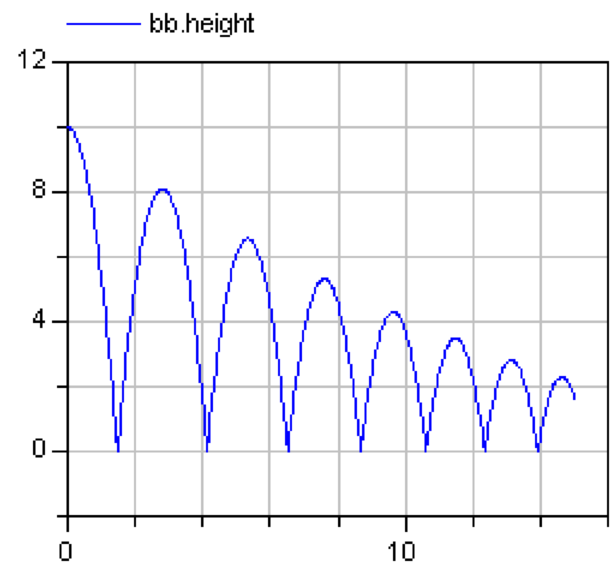
Actions at the end of a simulation are triggered by `terminal()`

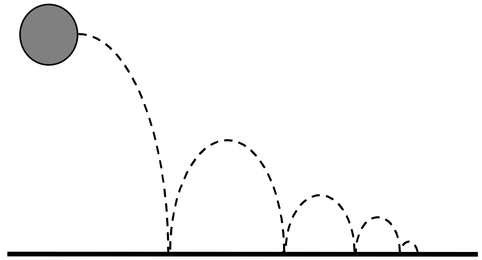


Reinit – Discontinuous Changes of Continuous

The value of a *continuous-time* state variable can be instantaneously changed by a `reinit`-equation within a `when`-equation

```
model BouncingBall "the bouncing ball model"
  parameter Real g=9.81;   //gravitational acc.
  parameter Real c=0.90;   //elasticity constant
  Real height(start=10),velocity(start=0);
equation
  der(height) = velocity;
  der(velocity)=-g;
  when height<0 then
    reinit(velocity, -c*velocity);
  end when;
end BouncingBall;
```



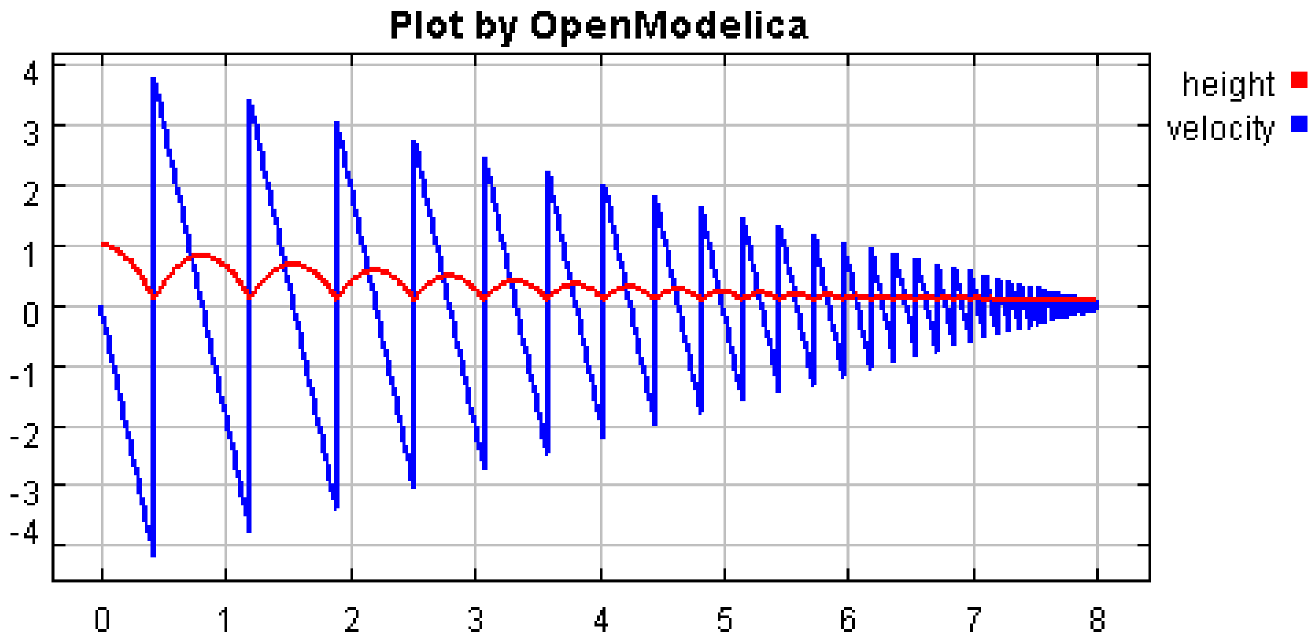


Reinit ”assigns” continuous-time variable `velocity` a new value

Initial conditions

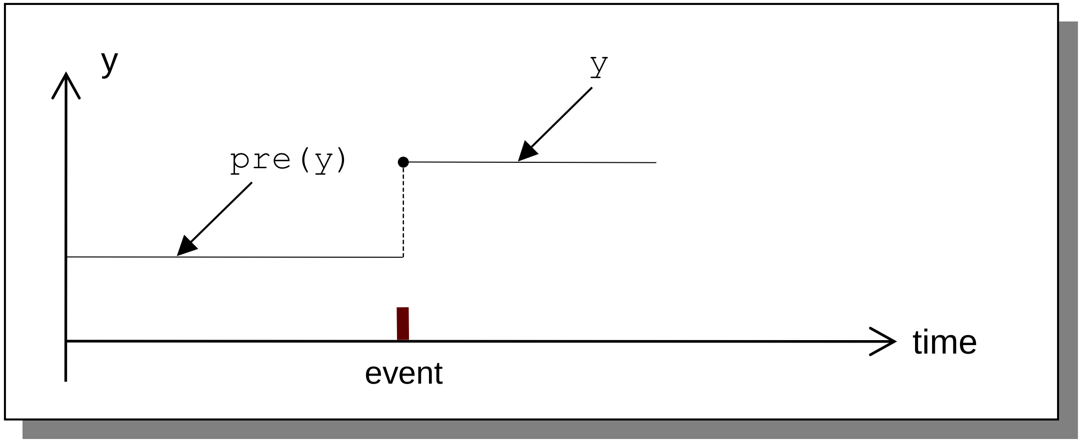
Exercise – BouncingBall

- Locate the BouncingBall model in one of the hybrid modeling sections of DrModelica (the When-Equations link in Section 2.9), run it, change it slightly, and re-run it.



Obtaining Predecessor Values of a Variable Using `pre()`

At an event, `pre(y)` gives the previous value of y immediately before the event, except for event iteration of multiple events at the same point in time when the value is from the previous iteration



- The variable *y* has one of the basic types `Boolean`, `Integer`, `Real`, `String`, or `enumeration`, a subtype of those, or an array type of one of those basic types or subtypes
- The variable y is a discrete-time variable
- The `pre` operator can *not* be used within a function

Event Priority

Erroneous multiple definitions, single assignment rule violated

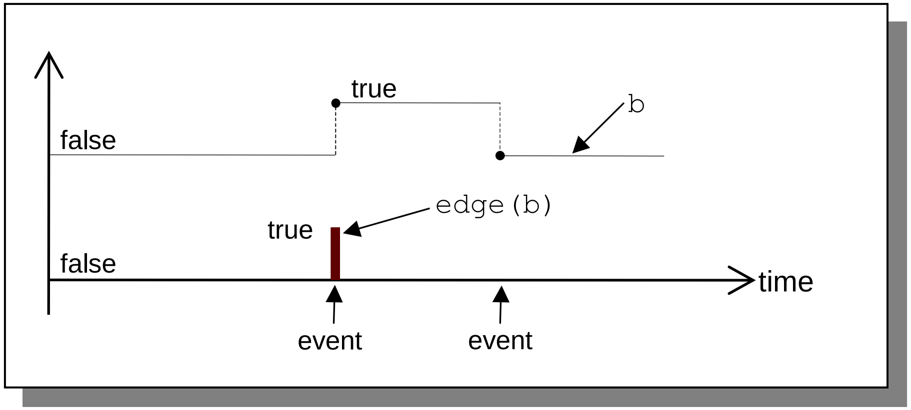
```
model WhenConflictX    // Erroneous model: two equations define x
  discrete Real x;
  equation
  when time>=2 then    // When A: Increase x by 1.5 at  time=2
   x = pre(x)+1.5;
  end when;
  when time>=1 then    // When B: Increase x by 1 at   time=1
   x = pre(x)+1;
  end when;
end WhenConflictX;
```

Using event priority to avoid erroneous multiple definitions

```
model WhenPriorityX
  discrete Real x;
equation
  when time>=2 then          // Higher priority
    x = pre(x)+1.5;
  elsewhen time>=1 then      // Lower priority
    x = pre(x)+1;
  end when;
end WhenPriorityX;
```

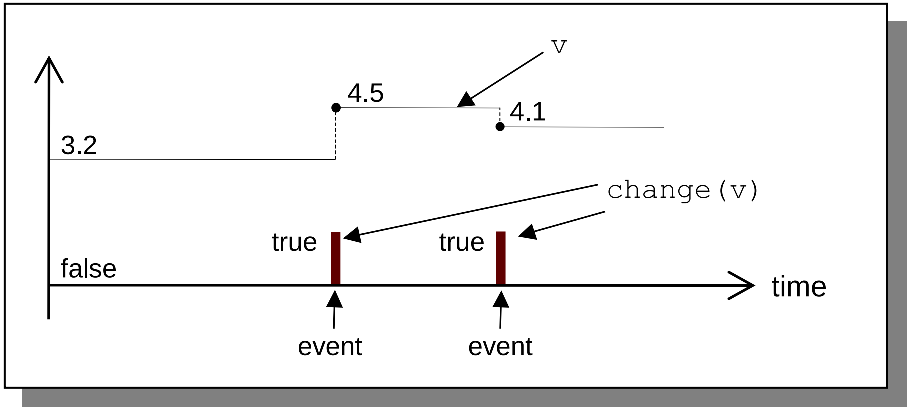

Detecting Changes of Boolean Variables Using edge() and change()

Detecting changes of boolean variables using `edge()`



The expression `edge(b)` is true at events when b switches from false to true

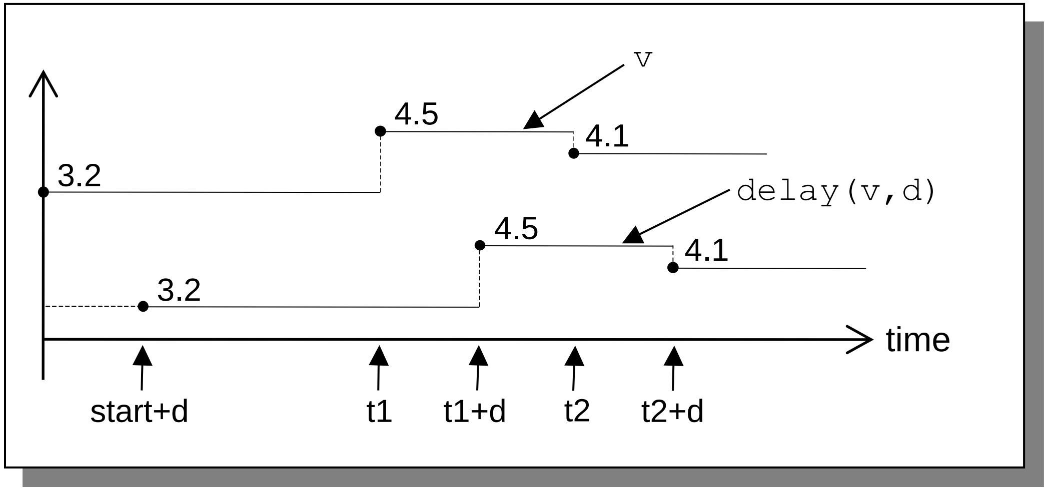
Detecting changes of discrete-time variables using `change()`



The expression `change(v)` is true at instants when v changes value

Creating Time-Delayed Expressions

Creating time-delayed expressions using `delay()`



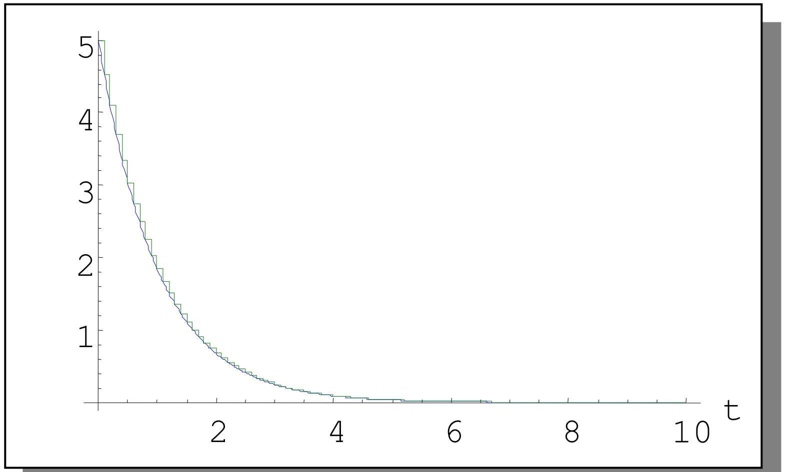
In the expression `delay(v,d)` *v* is delayed by a delay time *d*

A Sampler Model

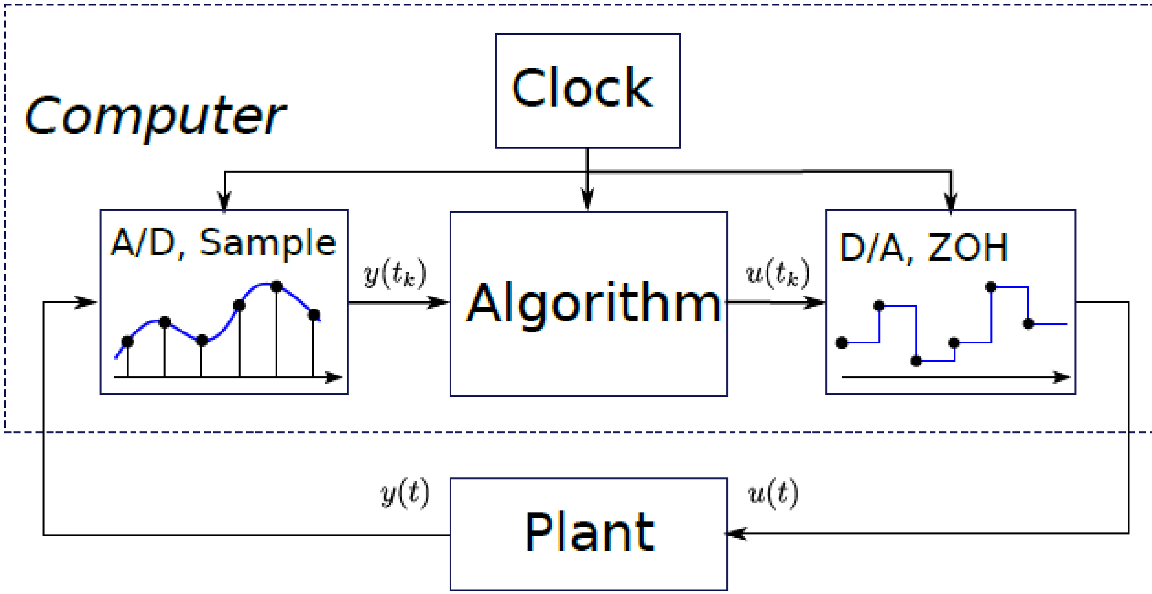
```
model Sampler
  parameter Real sample_interval = 0.1;
  Real x(start=5);
  Real y;
equation
  der(x) = -x;
  when sample(0, sample_interval) then
    y = x;
  end when;
end Sampler;
```

```
simulate(Sampler, startTime = 0, stopTime = 10)

plot({x,y})
```



Application: Digital Control Systems



- Discrete-time controller + continuous-time plant = hybrid system or sampled-data system
- Typically periodic sampling, can be modeled with "**when sample**(t0,td) **then** …"

Sampled Data-Systems in Modelica

```
// time-discrete controller
when {initial(),sample(3,3)} then
    E*xd = A*pre(xd)+ B*y;
    ud = C*pre(xd) + D*y;
end when;
```

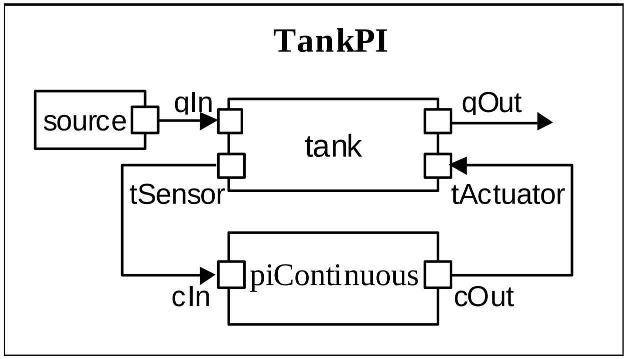
y

ud

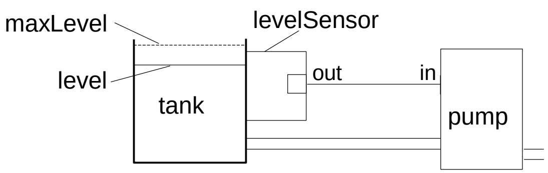
```
// plant (continuous-time process)
0 = f(der(x), x, ud);
y = g(x);
```

- y is automatically sampled at t = 3, 6, 9,…;
- xd, u are piecewise-constant variables that change values at sampling events (implicit zero-order hold)
- initial() triggers event at initialization (t=0)

Water Tank System with PI Controller




```
model TankPI
  LiquidSource             source(flowLevel=0.02);
  Tank                     tank(area=1);
  PIcontinuousController piContinuous(ref=0.25);
equation
  connect(source.qOut, tank.qIn);
  connect(tank.tActuator, piContinuous.cOut);
  connect(tank.tSensor, piContinuous.cIn);
end TankPI;
```




```
model Tank
  ReadSignal tOut;  // Connector, reading tank level
  ActSignal  tInp;  // Connector, actuator controlling input flow
  parameter Real flowVout = 0.01;       // [m3/s]
  parameter Real area = 0.5;            // [m2]
  parameter Real flowGain = 10;         // [m2/s]
  Real h(start=0);                      // tank level [m]
  Real qIn;                   // flow through input valve[m3/s]
  Real qOut;                  // flow through output valve[m3/s]
equation
  der(h)=(qIn-qOut)/area;               // mass balance equation
  qOut=if time>100 then flowVout else 0;
  qIn = flowGain*tInp.act;
  tOut.val = h;
end Tank;
```


Water Tank System with PI Controller – cont'

```
partial model BaseController
  parameter Real Ts(unit = "s") = 0.1   "Time period between discrete samples";
  parameter Real K = 2                  "Gain";
  parameter Real T(unit = "s") = 10     "Time constant";
  ReadSignal cIn                        "Input sensor level, connector";
  ActSignal  cOut                       "Control to actuator, connector";
  parameter Real ref                    "Reference level";
  Real error                            "Deviation from reference level";
  Real outCtr                           "Output control signal";
equation
  error = ref - cIn.val;
  cOut.act = outCtr;
end BaseController;
```

```
model PIDcontinuousController
  extends BaseController(K = 2, T = 10);
  Real  x;
  Real  y;
equation
  der(x) = error/T;
  y      = T*der(error);
  outCtr = K*(error + x + y);
end PIDcontinuousController;
```

```
model PIdiscreteController
  extends BaseController(K = 2, T = 10);
  discrete Real x;
equation

end PIdiscreteController;
```

Water Tank System with PI Controller – cont'

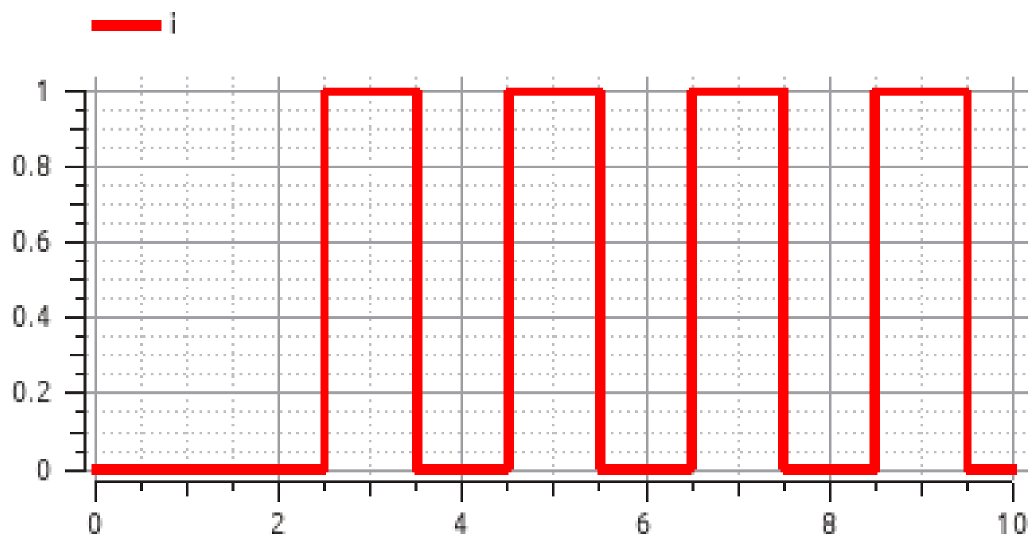
```
partial model BaseController
  parameter Real Ts(unit = "s") = 0.1   "Time period between discrete samples";
  parameter Real K = 2                  "Gain";
  parameter Real T(unit = "s") = 10     "Time constant";
  ReadSignal cIn                        "Input sensor level, connector";
  ActSignal  cOut                       "Control to actuator, connector";
  parameter Real ref                    "Reference level";
  Real error                            "Deviation from reference level";
  Real outCtr                           "Output control signal";
equation
  error = ref - cIn.val;
  cOut.act = outCtr;
end BaseController;
```

```
model PIDcontinuousController
  extends BaseController(K = 2, T = 10);
  Real  x;
  Real  y;
equation
  der(x) = error/T;
  y      = T*der(error);
  outCtr = K*(error + x + y);
end PIDcontinuousController;
```

```
model PIdiscreteController
  extends BaseController(K = 2, T = 10);
  discrete Real x;
equation
  when sample(0, Ts) then
    x = pre(x) + error * Ts / T;
    outCtr = K * (x+error);
  end when;
end PIdiscreteController;
```

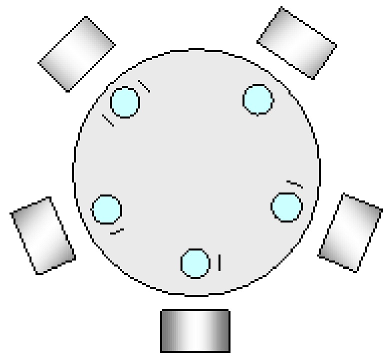
Exercise – Square Signal

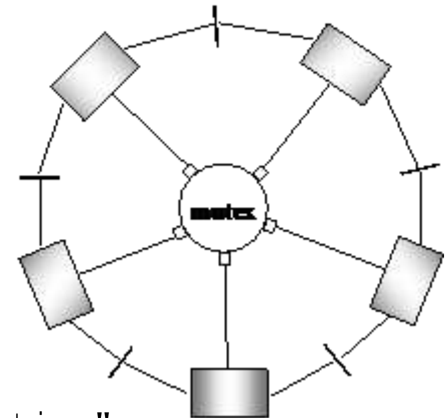
- Make a square signal with a period of 1s and that starts at t = 2.5s.
- Hint: an easy way is to use sample(...) to generate events, and define a variable that switches sign at each event.



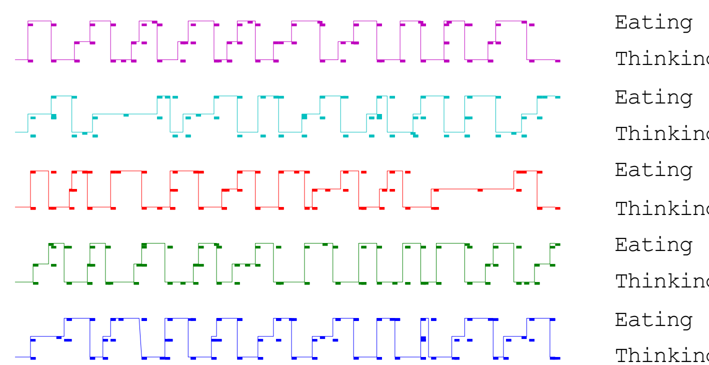
Concurrency and Resource Sharing

Dining Philosophers Example



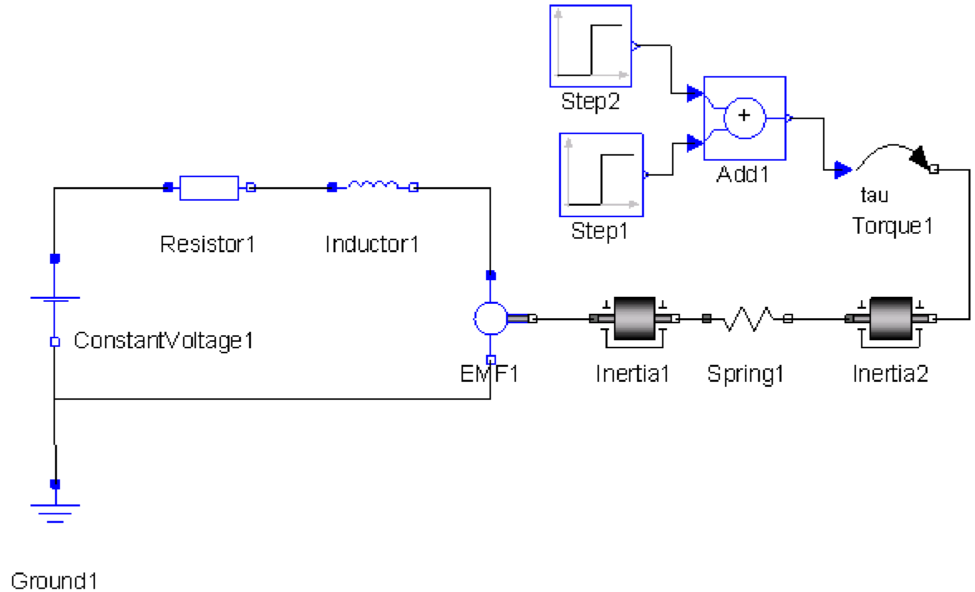


```
model DiningTable
  parameter Integer n = 5  "Number of philosophers and forks";
  parameter Real sigma = 5 " Standard deviation for the random function";
  // Give each philosopher a different random start seed
  // Comment out the initializer to make them all hungry simultaneously.
  Philosopher phil[n](startSeed=[1:n,1:n,1:n], sigma=fill(sigma,n));
  Mutex       mutex(n=n);
  Fork        fork[n];
equation
  for i in 1:n loop
    connect(phil[i].mutexPort, mutex.port[i]);
    connect(phil[i].right, fork[i].left);
    connect(fork[i].right, phil[mod(i, n) + 1].left);
  end for;
end DiningTable;
```



Exercise – Hybrid DC Motor - Generator

- What is needed if you want to make a hybrid DC motor, i.e. a DC motor that also can act like a generator for a limited time?
- Make it work like a DC motor for the first 20s
- Apply a counteracting torque on the outgoing axis for the next 20s, and then turn off the counteracting torque, i.e. you would like to have a torque pulse starting at 20s and lasting 20s.



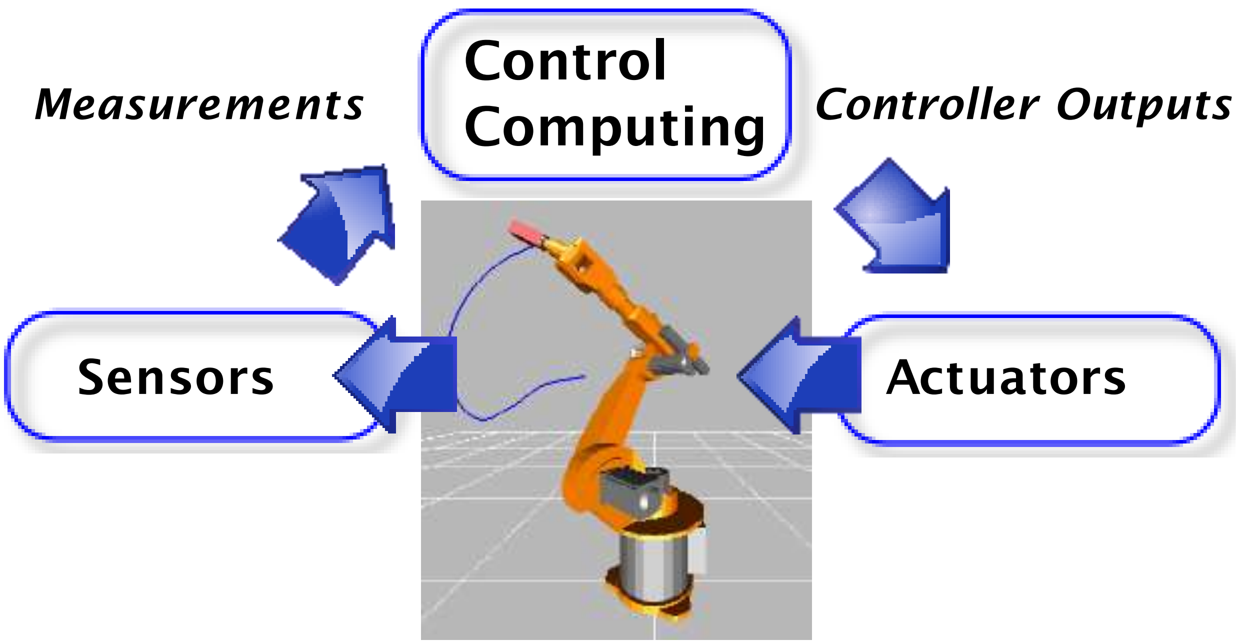
Clocked Synchronous Models and State Machines

and Applications for Digital Controllers

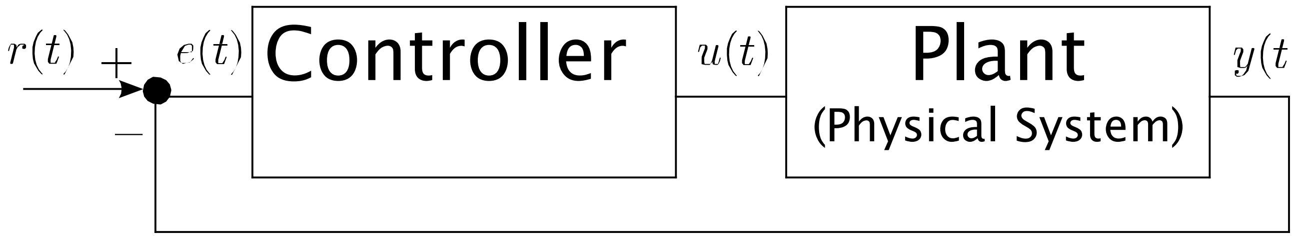
Control System Applications

Control System

A control system is a device, or set of devices, that manages, commands, directs or regulates the behavior of other devices or systems (wikipedia).



Control Theory Perspective
Feedback Control System

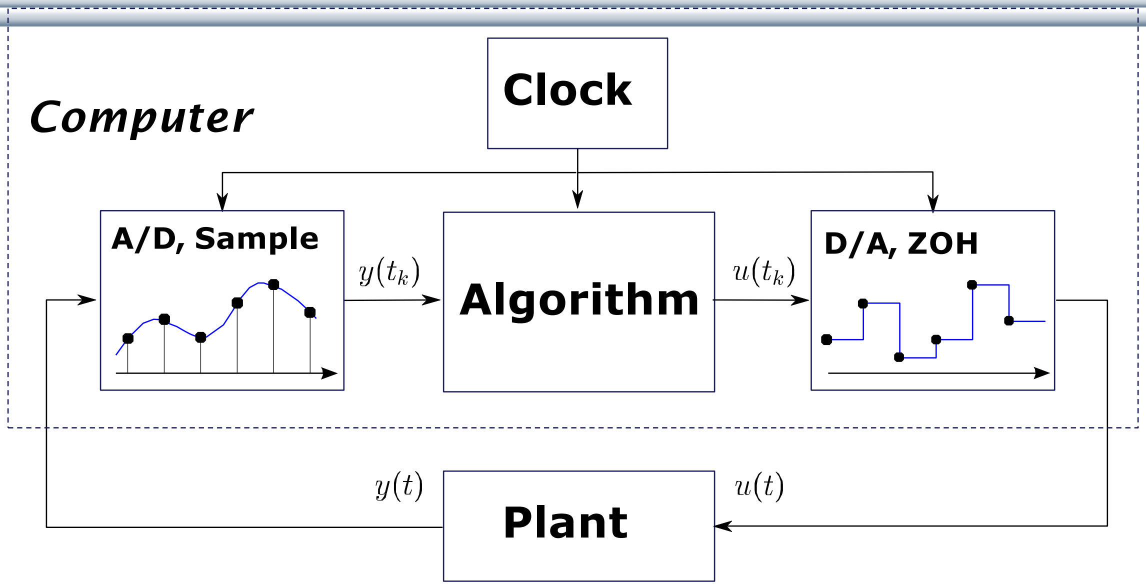


$\boldsymbol{r(t)}$	**reference (setpoint) error**
$\boldsymbol{e(t)}$	**measured process variable (plant output)**
$\boldsymbol{y(t)}$	**control output variable (plant input)**
$\boldsymbol{u(t)}$	

Usual Objective

Plant output should follow the reference signal.

Embedded Real-Time Control System



1. ***Discrete-time*** controller + ***continuous-time*** plant ≡ *hybrid system* or *sampled-data system*

2. Interface between digital and analog world: Analog to Digital and Digital to Analog Converters (ADC and DAC).

3. ADC→Algorithm→DAC is synchronous (zero-delay model!)

4. A *clock* controls the *sampling instants*. Usually *periodic sampling*.

Controller with Sampled Data-Systems

(unclocked models, using pre() and unclocked sample())

```
// time-discrete controller
when {initial(),sample(3,3)} then
    E*xd = A*pre(xd)+ B*y;
    ud = C*pre(xd) + D*y;
end when;
```

y

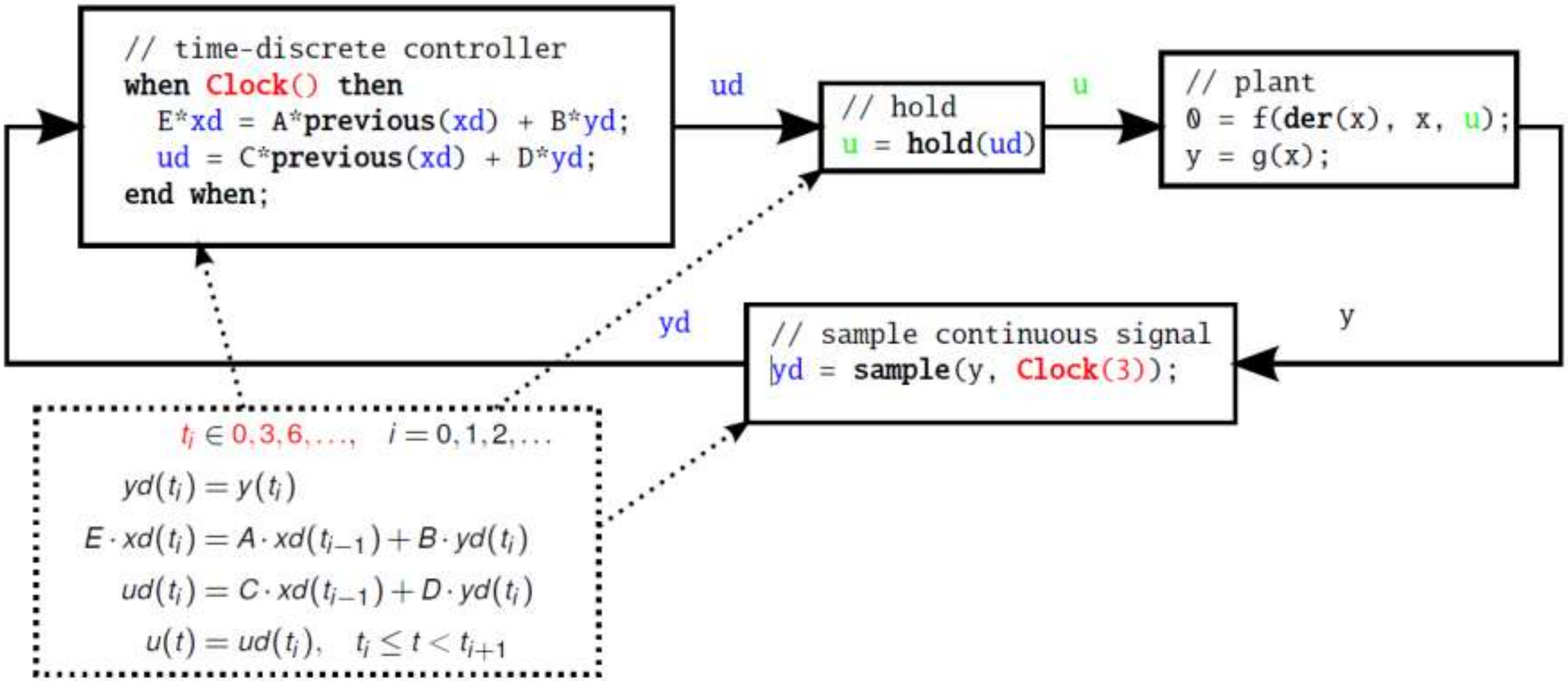
ud

```
// plant (continuous-time process)
0 = f(der(x), x, ud);
y = g(x);
```

- y is automatically sampled at t = 3, 6, 9,…;
- xd, u are piecewise-constant variables that change values at sampling events (implicit zero-order hold)
- initial() triggers event at initialization (t=0)

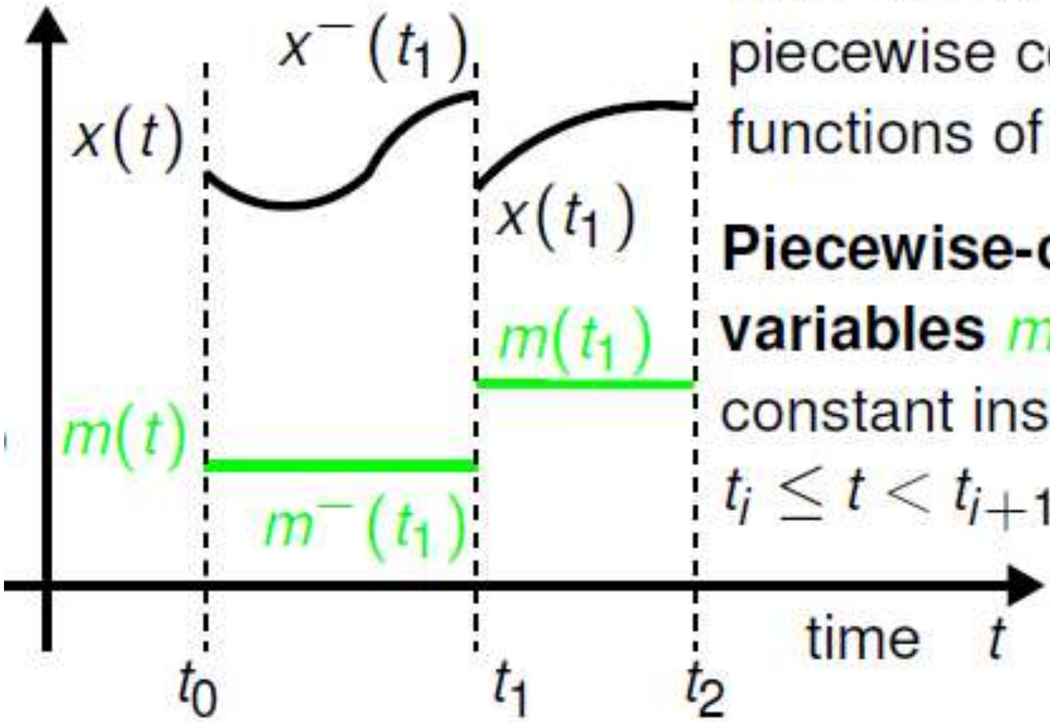
Controller with Clocked Synchronous Constructs

clocked models using Clock(), previous(), hold() in Modelica 3.3



Using **previous**() instead of **pre**(), **hold**() to get values between ticks, and *clocked* **sample**() to sample at clock ticks.

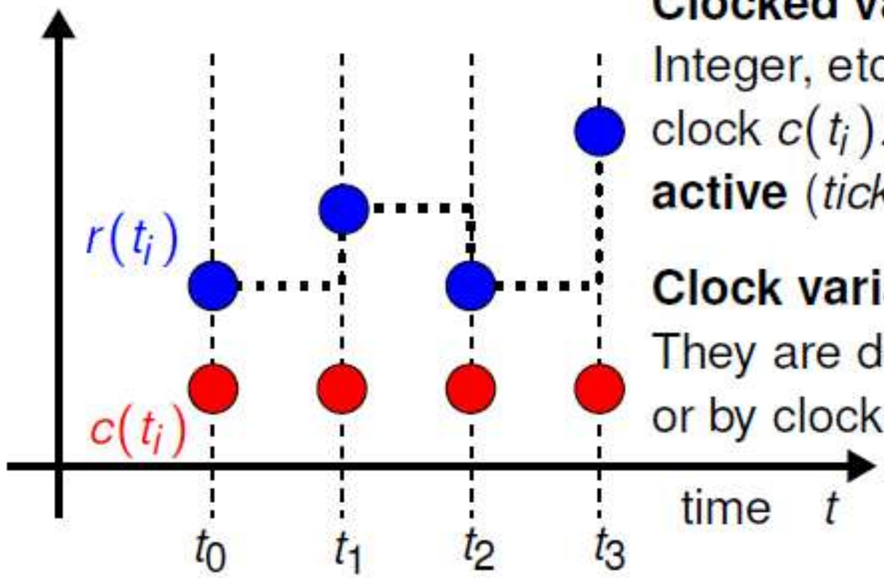
Unclocked Variables in Modelica 3.2



Continuous variables are Real numbers defined as piecewise continuous functions of time.

Piecewise-constant variables $m(t)$ are constant inside each $t_i \leq t < t_{i+1}$.

Clock variables (Clock) and Clocked Variables (Real) (in Modelica 3.3)

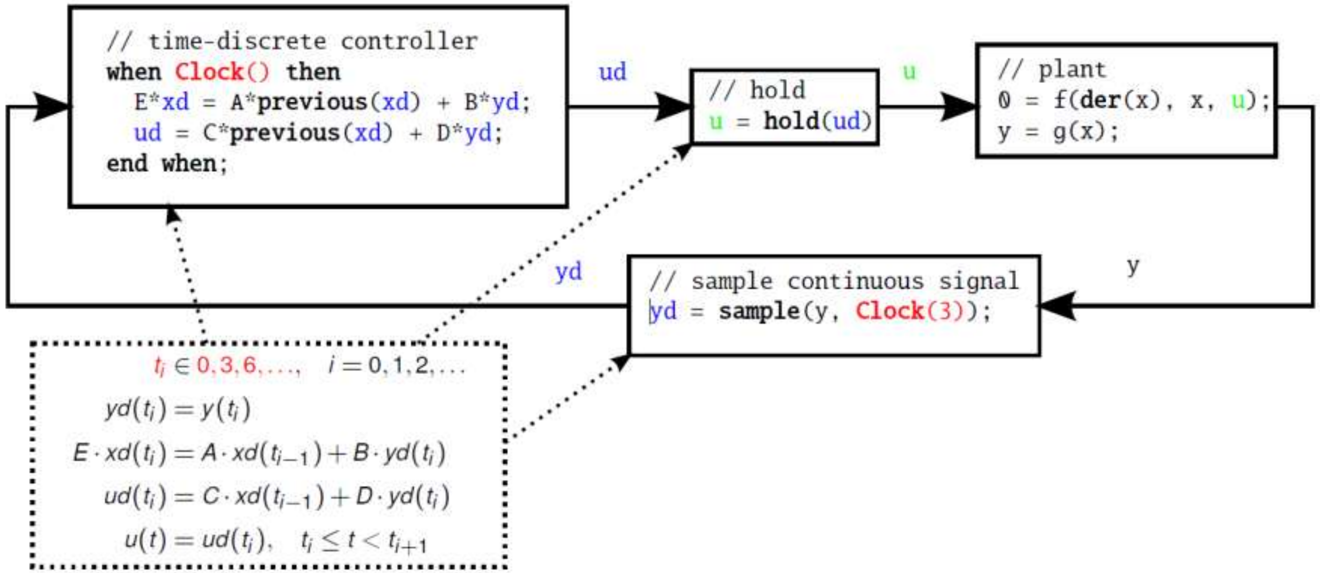


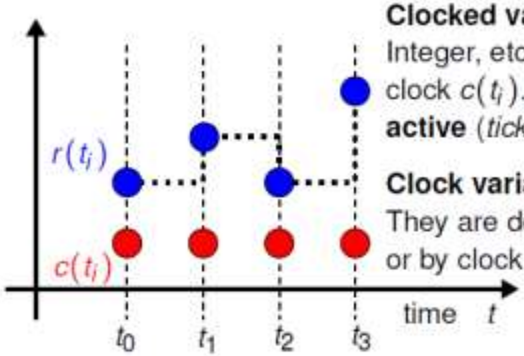
Clocked variables $r(t_i)$ are of base type Real, Integer, etc. They are uniquely associated with a clock $c(t_i)$. Can only be accessed when its clock is **active** (*ticks*).

Clock variables $c(t_i)$ are of base type **`Clock`**. They are defined by constructors such as **`Clock`**`(3)` or by clock operators relatively to other clocks.

–

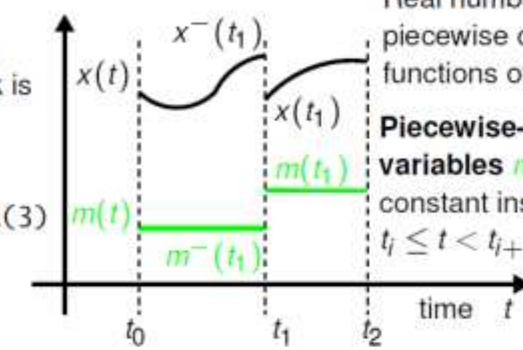
Clocked Synchronous Extension in Modelica 3.3





Clocked variables $r(t_i)$ are of base type Real, Integer, etc. They are uniquely associated with a clock $c(t_i)$. Can only be accessed when its clock is **active** (*ticks*).

Clock variables $c(t_i)$ are of base type `Clock`. They are defined by constructors such as `Clock(3)` or by clock operators relatively to other clocks.

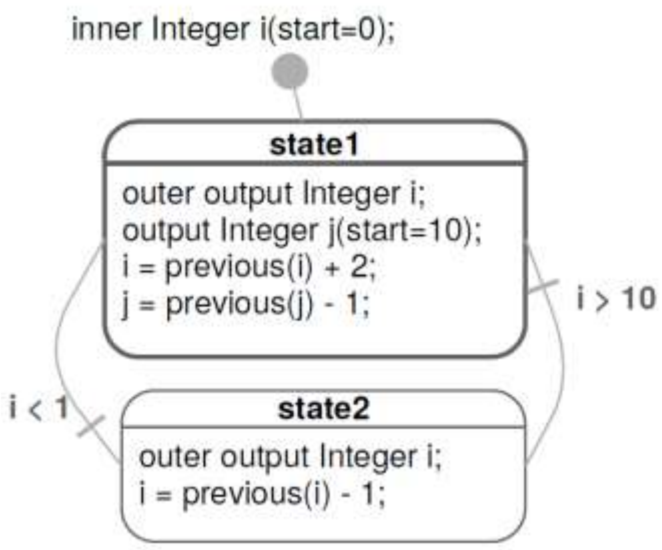


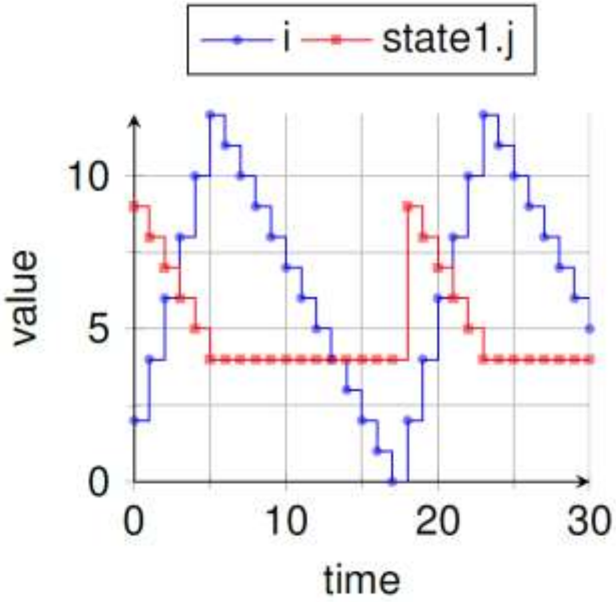
Continuous variables are Real numbers defined as piecewise continuous functions of time.

Piecewise-constant variables $m(t)$ are constant inside each $t_i \le t < t_{i+1}$.

MODELICA

State Machines in Modelica 3.3: Simple Example



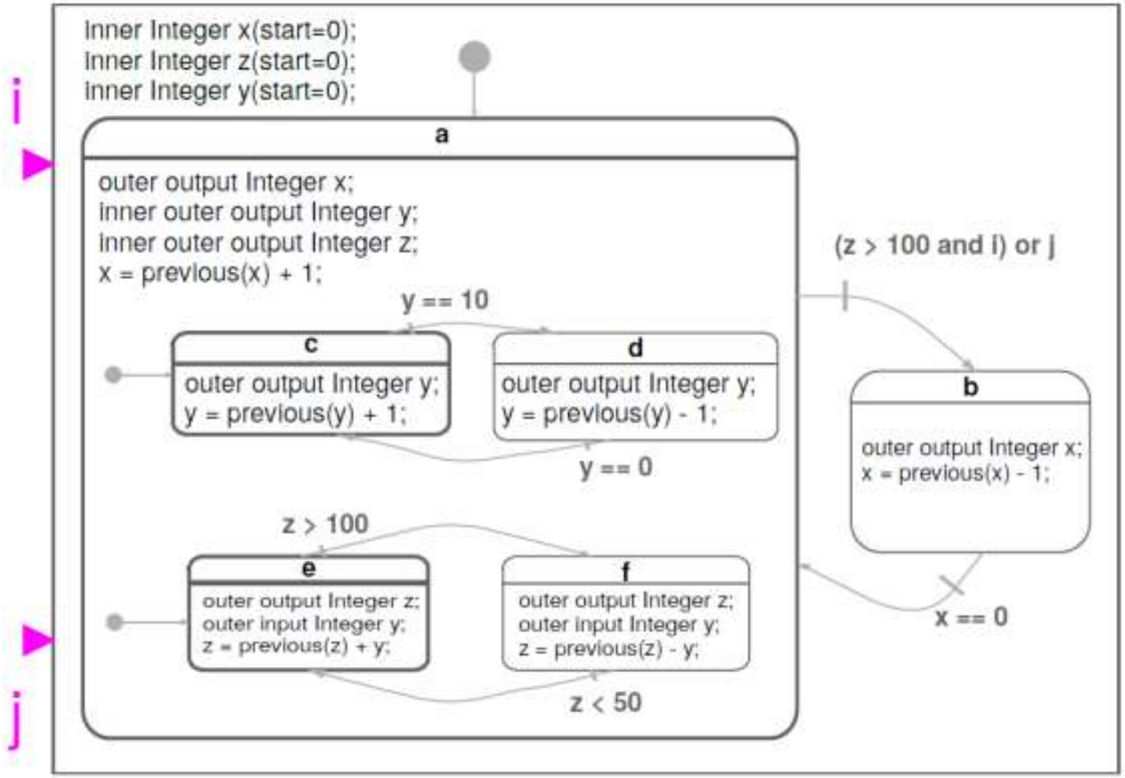


- Equations are active if corresponding *clock* ticks. Defaults to periodic clock with 1.0 s sampling period
- "i" is a shared variable, "j" is a local variable. Transitions are "*delayed*" and enter states by "*reset*"

Simple Example: Modelica Code

```
model Simple_NoAnnotations "Simple state machine"
  inner Integer i(start=0);
  block State1
    outer output Integer i;
    output Integer j(start=10);
  equation
    i =  previous(i) + 2;
    j =  previous(j) - 1;
  end State1;
  State1 state1;
  block State2
    outer output Integer i;
  equation
    i =  previous(i) - 1;
  end State2;
  State2 state2;
equation
  transition(state1,state2,i > 10,immediate=false);
  transition(state2,state1,i < 1,immediate=false);
  initialState(state1);
end Simple_NoAnnotations;
```

Hierarchical and Parallel Composition of Modelica State Machine Models



Semantics of Modelica state machines (and example above) inspired by Florence Maraninchi & Yann Rémond's "Mode-Automata" and by Marc Pouzet's Lucid Synchrone 3.0.

Synchronous Language Elements in Modelica 3.3

- Recent language extension of the Modelica standard
- Introduced new language elements suited for modelling digital control systems
- Motivated by synchronous languages for reactive system (real-time constraints!)
- Synchronous languages have been particularly successful for safety-relevant applications

Benveniste, A., Edwards, S. A., Halbwachs, N., Le Guernic, P., and de Simone, R. The synchronous languages 12 years later. In *Proceedings of the IEEE*, volume 91 (1), pages 64–83, 2003. doi:10.1109/JPROC.2002.805826.

Improvements in Modelica 3.3 Sampled Control

- Possible to **detect modelling errors** if accidentally connecting control blocks with different sampling period
- All variables assigned in an old when-clause need to be treated as “discrete-time states” even if only a subset are real discrete-time states.
- **General equations are allowed in clocked partitions** and in particular also in clocked when-clauses.
- Equations in different (old) when clauses are only synchronous if triggered by same event (i.e., when sample(0,3) then . . . ≠ when sample(0,3) then . . .). In contrast, **clocking information is propagated by clock inference**.
- **Improved support for inverse-models** for advanced controllers

Comparison to Sampled Data-systems in Modelica 3.2

No *Clocks* in Modelica 3.2. Periodically sampled data systems are defined by a sample operator. Note: The 'old' **sample** is completely different from the 'new' **clocked sample**!

```
event = sample(0,3);
when event then
  xd = A*pre(xd) + B*y;
  u = C*pre(xd) + D*y;
end when;
```

Example with old unclocked sample

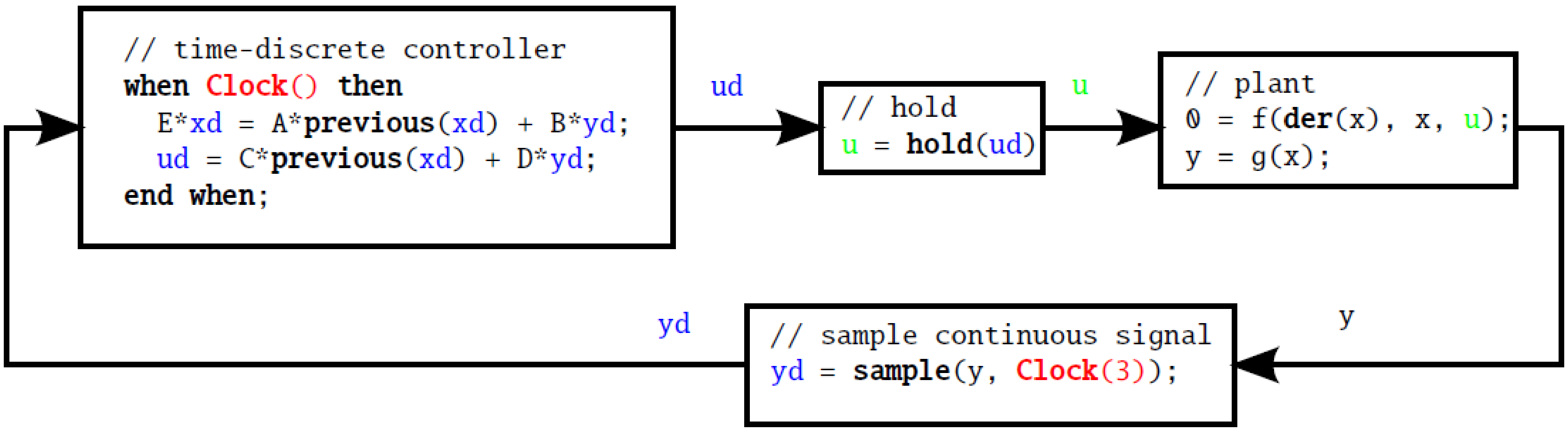
y (continuous input) is automatically sampled at t = 0, 3, 6, . . .; variables xd, u are **piecewise-constant variables** that change values at the sampling events.

New Language Elements in Clocked Modelica 3.3

Clock Constructors	**Clock**(); **Clock**(intervalCounter, resolution); **Clock**(interval); **Clock**(condition, startInterval); **Clock**(c, solverMethod);
Base-clock conversion operators	**sample**(u,c); **hold**(u)
Sub-clock conversion operators	**subSample**(u,factor); **superSample**(u,factor); **shiftSample**(u,shiftCounter,resolution); **backSample**(u,backCounter,resolution); **noClock**(u)
Other operators	**previous**(u); **interval**(u)

The Modelica_Synchronous library

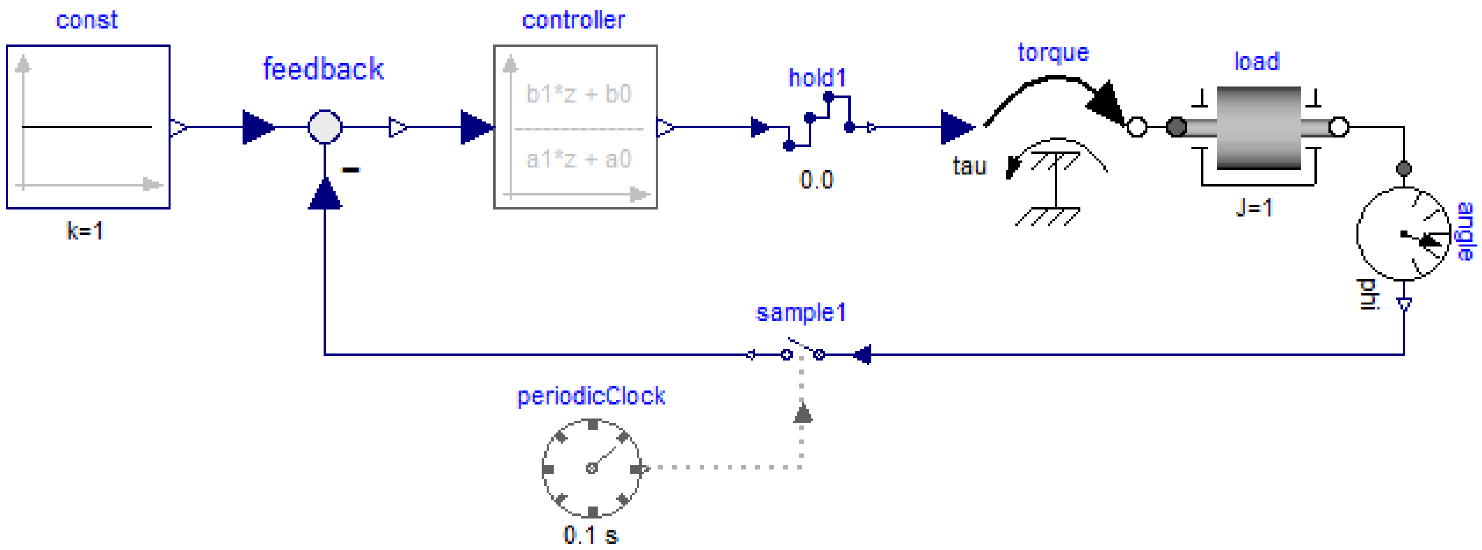
The *Modelica_Synchronous* library was developed for precise and convenient definition and synchronization of multi-rate data systems.



Martin Otter, Bernhard Thiele, Hilding Elmqvist. A Library for Synchronous Control Systems in Modelica. In *9th Int. Modelica Conference*, Munich, Germany, September 2012.

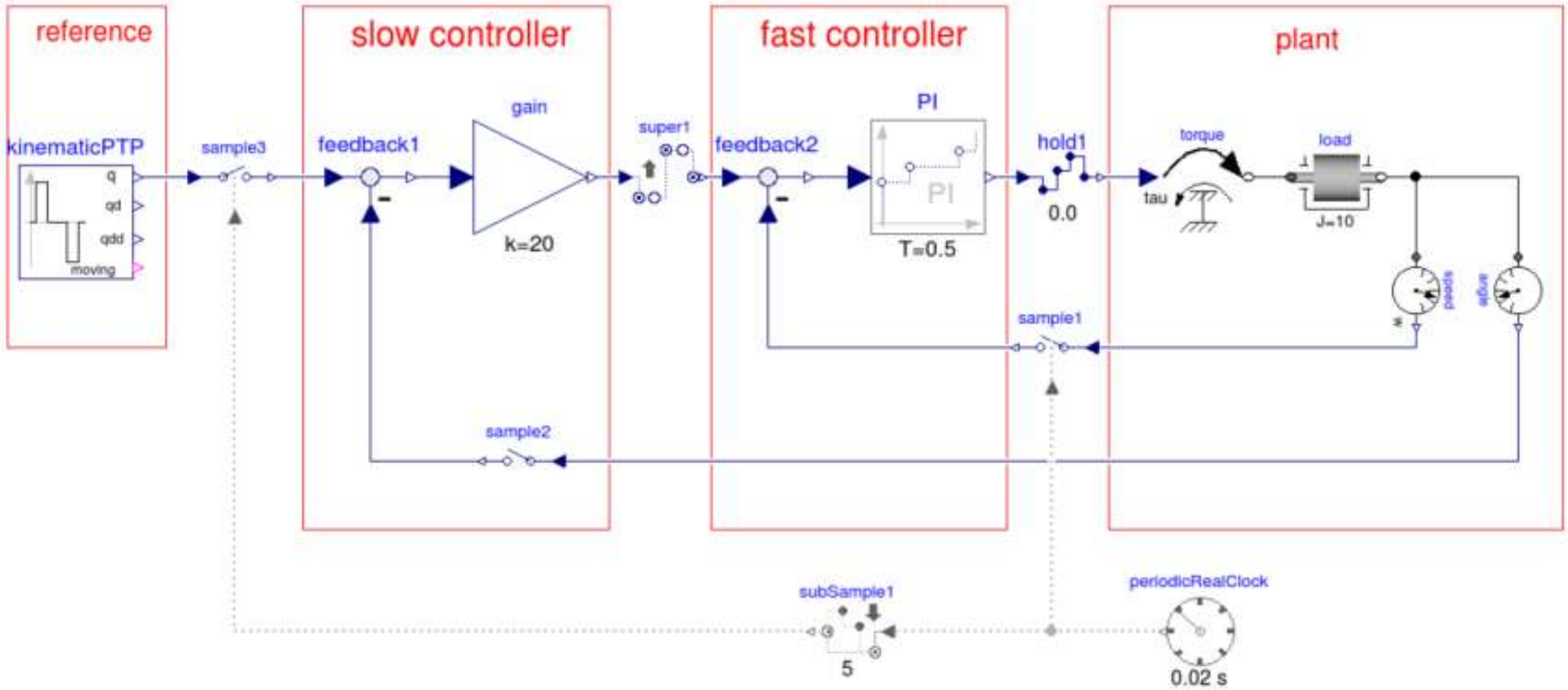
The *Modelica_Synchronous* library cont'

The *Modelica_Synchronous* library was developed for precise and convenient definition and synchronization of multi-rate data systems.

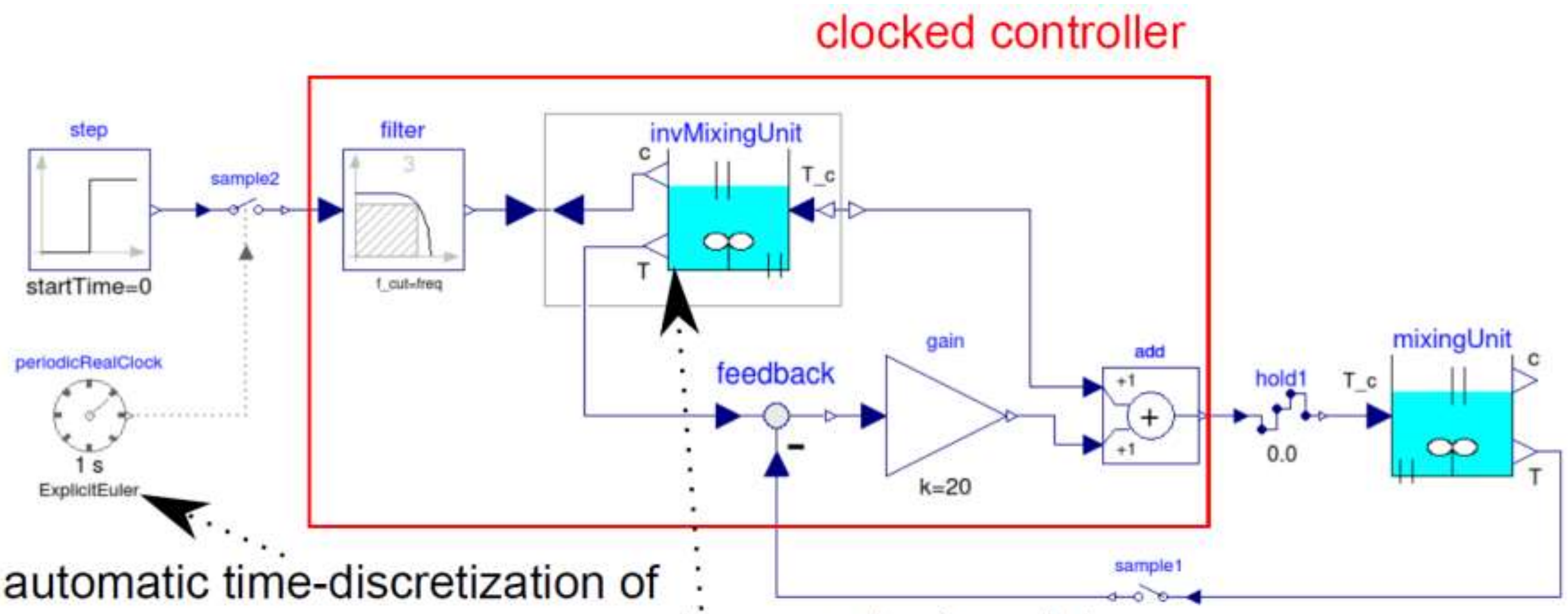


Martin Otter, Bernhard Thiele, Hilding Elmqvist. A Library for Synchronous Control Systems in Modelica. In *9th Int. Modelica Conference*, Munich, Germany, September 2012.

Modelica_Synchronous: Cascaded Control



Modelica_Synchronous: Inverse Models



automatic time-discretization of continuous-time model using the Explicit Euler method and a sample period of 1s

inverse plant model (input T_c becomes output output c becomes input)

Synchronous Language Elements in OM

- OpenModelica 1.11.0 and later has full support for the clocked synchronous extension.
- Basic graphical editing support for state machines available in OMEdit
- For digital control, try examples from the Modelica_Synchronous library
- For state machines try the two examples in file StateMachines.onb