Timed Automata

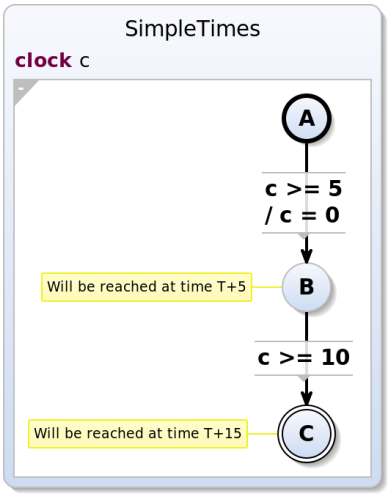
> **Related Publications**
>
> **Time in SCCharts**. Alexander Schulz-Rosengarten and Reinhard von Hanxleden and Frédéric Mallet and Robert de Simone and Julien Deantoni. In Proc. Forum on Specification and Design Languages (FDL '18), Munich, Germany, September 2018.

Clocks

Clocks can be defined to gain access to real time and specify timed transitions.

This feature requires a *dynamic tick* environment to work. Then the time in all clocks will be automatically updated and the SCChart will request a sleep time such that it will wake up when a timed transition will be enabled.

```
scchart SimpleTimes {
  clock c

  initial state A
  if c >= 5 do c = 0 go to B

  /** Will be reached at time
T+5 */
  state B
  if c >= 10 go to C

  /** Will be reached at time
T+15 */
  final state C
}
```



Various annotations (for the root SCChart) can be used to modify the generated behavior.

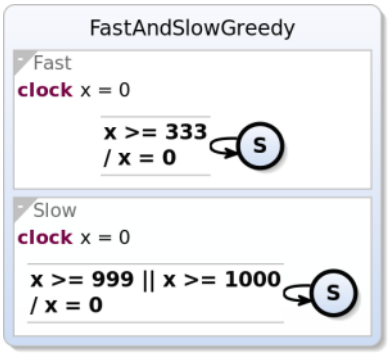
Annotation	Effect
@NoSleep	Prevents declaration and calculation of *sleepT*.
@DefaultSleep	Sets the default sleep time, requested if there is no timed transition for the active state.
@SimulateSleep	Generates code that simulates the dynamic tick environment by assuming that between ticks always the requested sleep time passes.
@IntegerClockType	Switches from type float to int for the *sleepT* and *deltaT* variables and related calculations.
@ClocksUseSD	Enables support for **concurrent** and **hierarchical** usage of the clock variable. (experimental)

Greedy Bounds

With timed SCCharts there might occur situations where timing events can fall very closely to each other. Usually a system can not handle such events precisely. For example one timed transition triggers a reaction that takes some milliseconds to compute, but there is another transition that is enable one millisecond after the first one. The handling of the second event will be delayed by the computation time of the reaction. If such delay should not happen but it is acceptable to handle the second event a bit earlier (at time of the first event), greedy bounds can be used.

If a disjunction of multiple timing constraints with the same clock is specified, the dynamic tick environment will only trigger timing events for the latest of these bounds but will also handle the transition if earlier bounds are satisfied.

```
scchart FastAndSlowGreedy {
  region Fast {
    clock x = 0

    initial state S
    if x >= 333 do x = 0 go to S

  }
  region Slow {
    clock x = 0

    initial state S
    if x >= 999 || x >= 1000 do
x = 0 go to S
  }
}
```

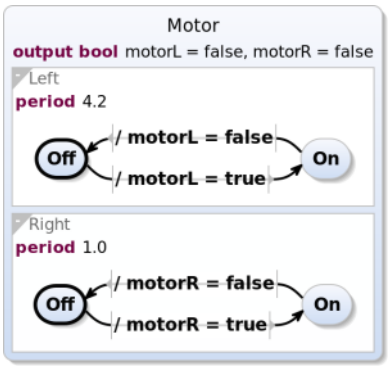


Periodic Regions

With the **period x** keyword a region can be guarded such that it will only be active every **x** seconds.

Annotation	Effect
@SoftReset	Prevents accumulation of clock deviations (if the tick function is not invoked at the exact time w.r.t. the sleep time) by leaving the additional time on the clock. Effectively, the clock is not reset to 0 but to x - sleepT.
@HardReset	(**default**) Resets the periods timer to 0 when triggered.

```
@DefaultSleep 1000
scchart Motor {
  output bool motorL = false,
motorR = false

  region Left {
    @SoftReset
    period 4.2

    initial state Off
      do motorL = true go to On

    state On
      do motorL = false go to
Off
  }

  region Right {
    @SoftReset
    period 1.0

    initial state Off
      do motorR = true go to On

    state On
      do motorR = false go to
Off
  }
}
```



Dynamic Ticks

Dynamic Ticks are based on an extended tick environment that provides *deltaT* and receives *sleepT*.

The simulation in KIELER supports a dynamic ticks mode to test models using this feature.
In the simulation view switch mode to *Dynamic*, see screenshot below.