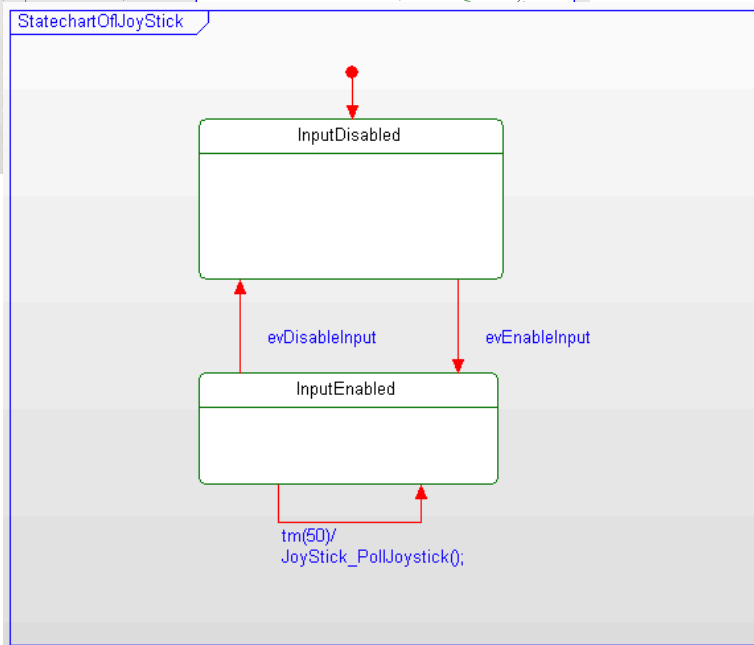
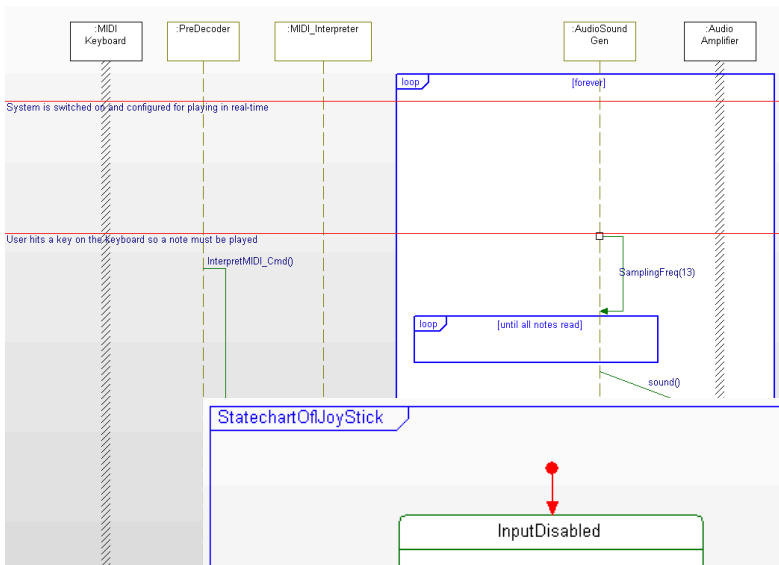


# Statecharts & Sequence Diagrams

IBM® Rational® Rhapsody® StartUp Training

WILLERT.



## Index:

Statecharts

Events

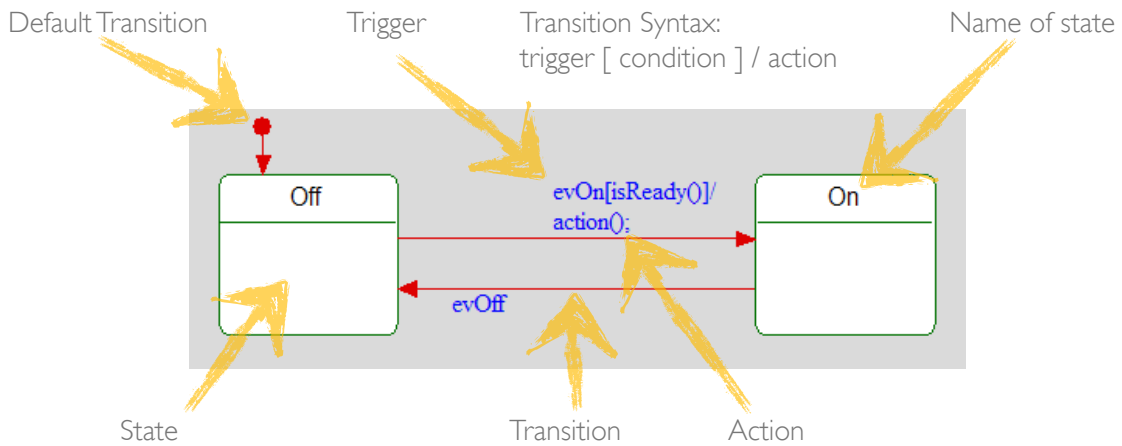
Actions

Timers

Triggered Operations

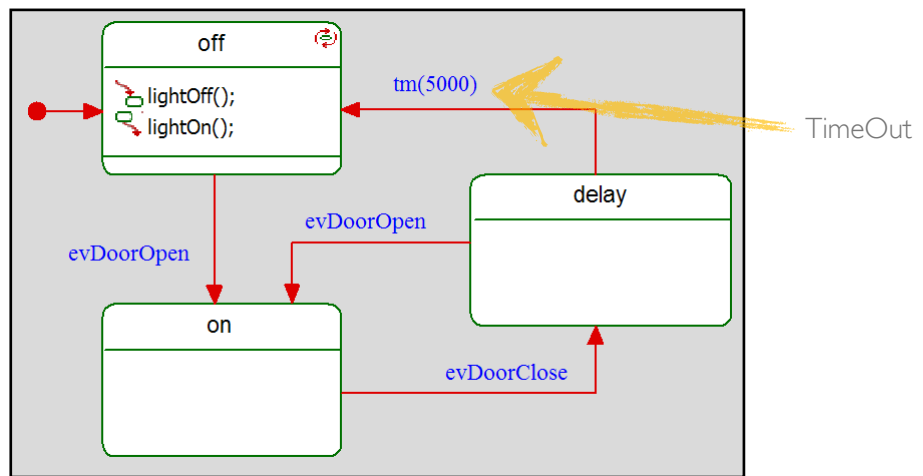
NULL Transitions

# Statecharts



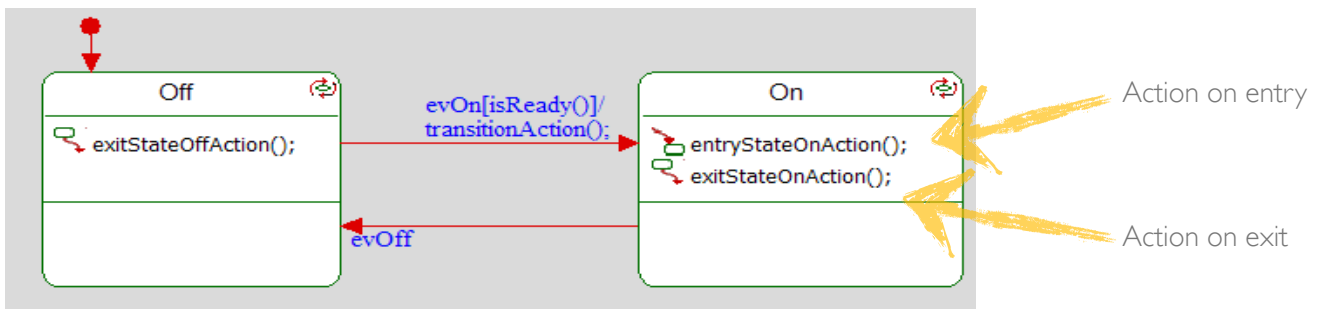
## Triggered Events

If an object enters a state, the timeout is started immediately. Of course, this will only happen in the states with a timeout. If the timeout expires, the state-chart receives this as an event and a transition is passed. If a state is left, each timeout stops. There is only one timeout per state possible.



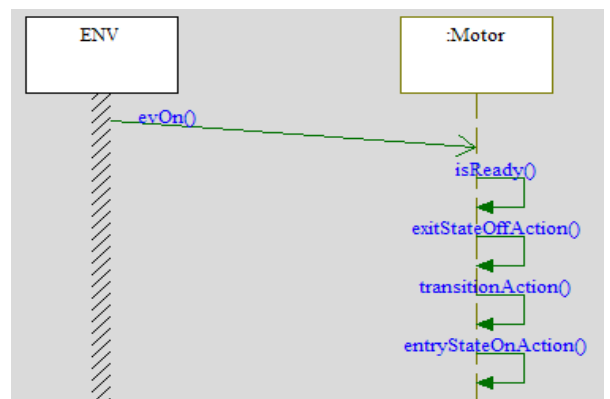
## Action on entry / Action on exit

Such a state-chart needs something to do. We can insert our code inside the "action" fields. But we should note the order of execution.

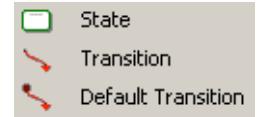


Every transition has an action field in his general tab. If the transition is passed, the action is executed. There is no delay, when a transition is passed.

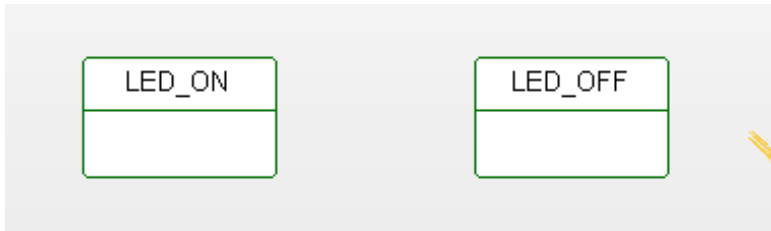
A State can be active for a longer time. Therefore, we have "action on entry" and "action on exit" fields.



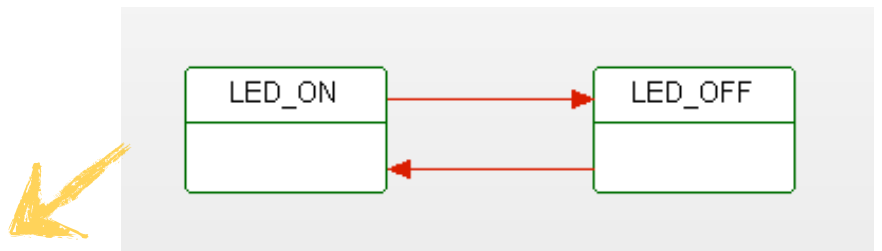
We will start with an simple example. Create a C\_LED class. Create now a statechart in this class. Right click the C\_LED class. Select Add New / Diagrams / Statechart. We need only three different elements from the drawing toolbar. We will start with two states.



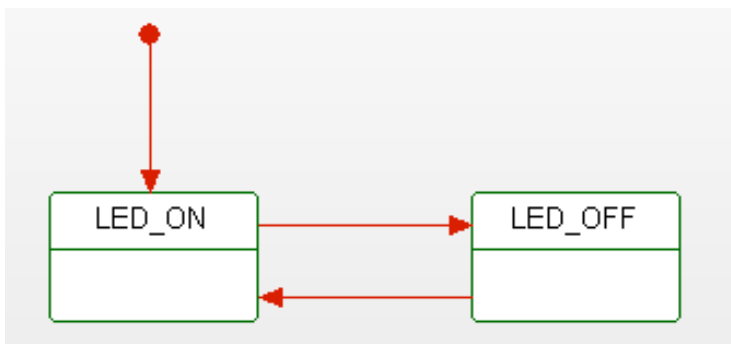
- draw two states on the diagram surface
- name it "LED\_ON" and "LED\_OFF"



draw a transition from LED\_ON to LED\_OFF and backwards



draw a default transition to the LED\_ON state

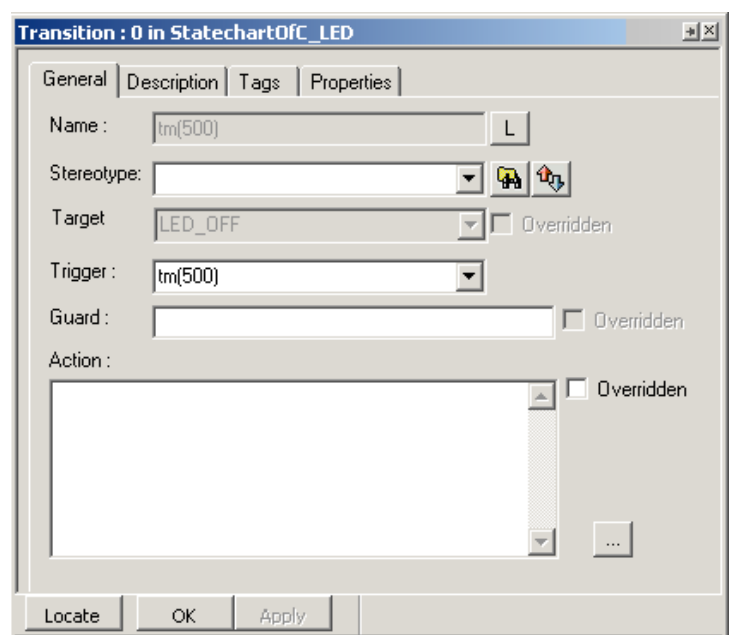
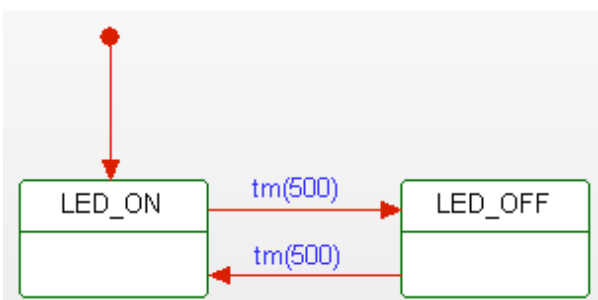


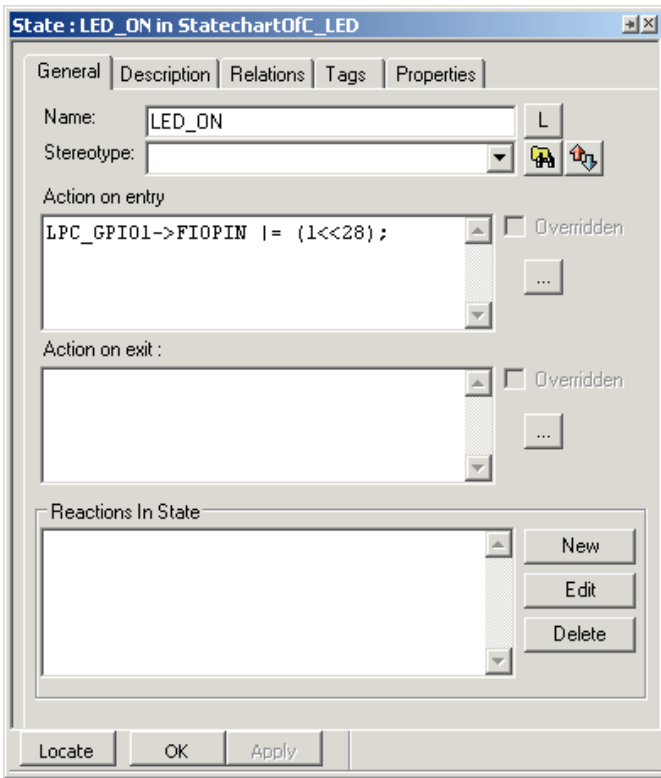
**Info**  
A default transition is an entry point to our statechart.

Our statechart is now full functional, but senseless. Immediately after object startup, the default transition will be passed. After that, the active state will toggle between LED\_ON and LED\_OFF as fast as possible. We can change this, by using a trigger:

### Timer events

Open the features dialog of both transitions. Select the general tab and insert a "tm(500)" in the Trigger-field. The tm() uses the systick from our CortexM3. 500 is the time for the timeout in milliseconds. After expiration of the selected time, the transition will be passed and the state-chart switches to the next state.

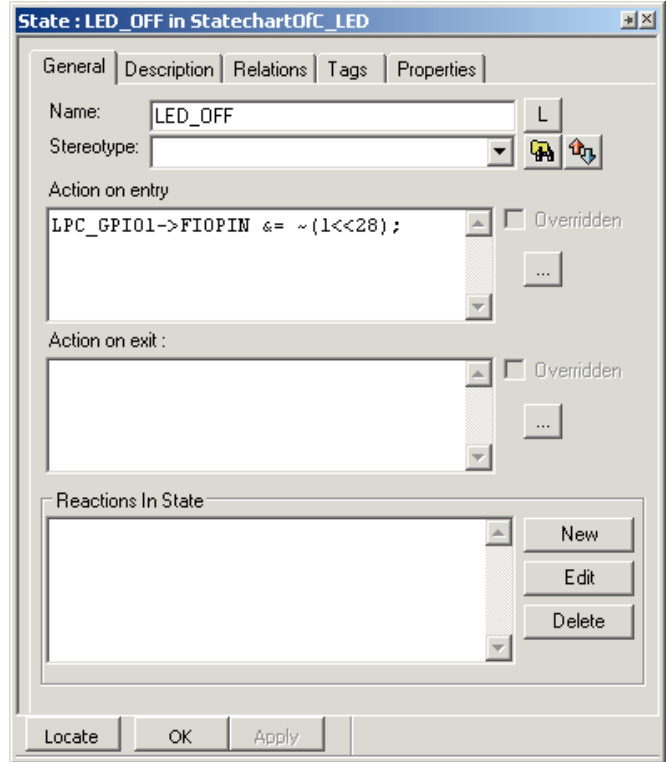




Such a statechart needs something to do. So we will add actions to the states.

The actions needed here are "on" and "off".

If you put the right actions for your CPU in the LED\_ON and LED\_OFF state, it will be executed every 500 milliseconds.

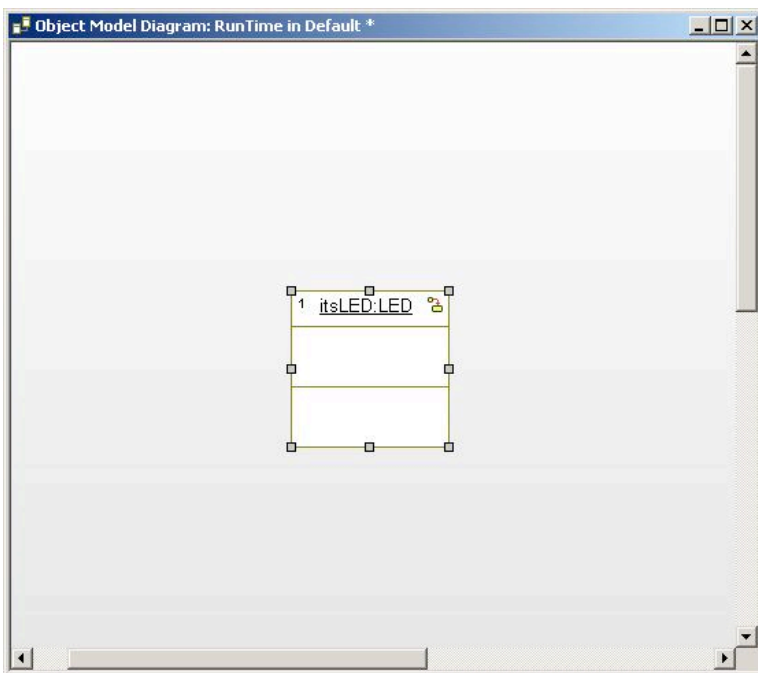


Create an initializer (constructor) for the C\_LED class. We do not need an action in the default transition, this should be done by the initializer.

We must add the statement for setting the port to output into the initializers implementation tab.

```
LPC_GPIO1->FIODIR |= (1<<28);
```

Don't forget to include the lpc17xx.h for the target specific code.



We only need to create an instance now to test our statechart.

We do this by drawing another Object Model Diagram (OMD)

We drag the class LED from the browser to the diagram, right click, choose "Make an Object"

This creates an object.

When we now do a Generate Make Run, the program will let the LED blink.

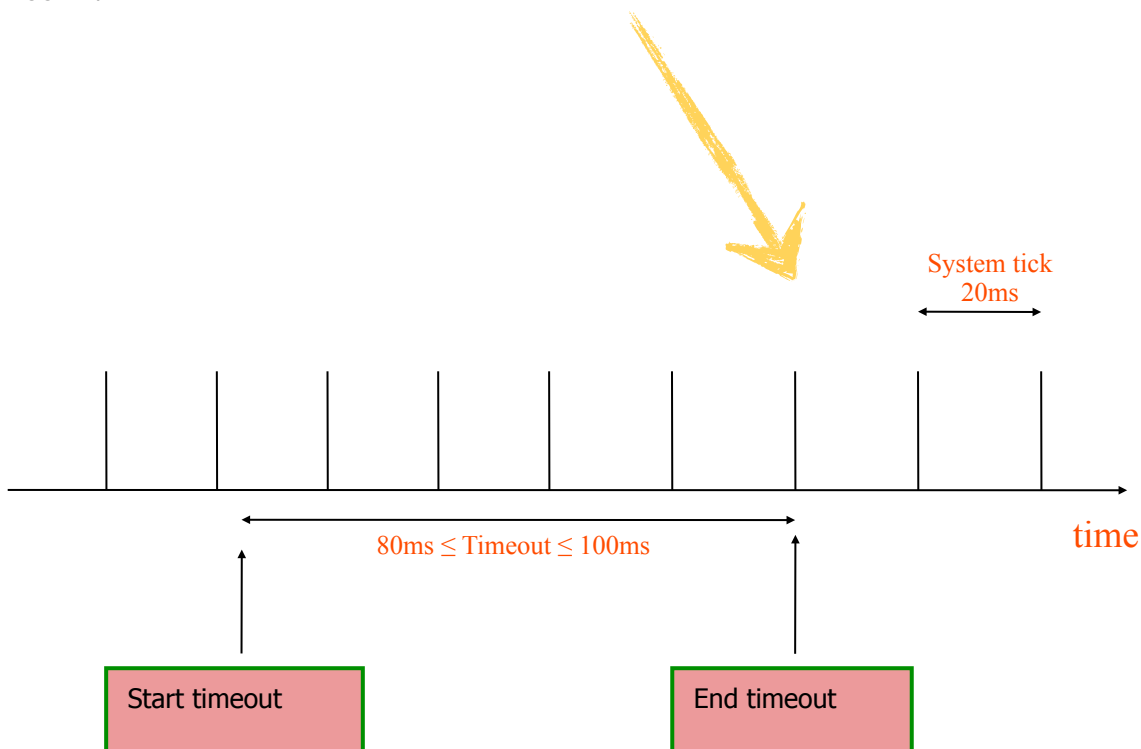
## Timer accuracy

Timers in UML have, as we have already seen, a granularity of 1 millisecond. The timers are handled by the framework, which is triggered by the underlying RTOS. The tick from the RTOS is used as "pace-maker" for the UML tm driver:

tm(xx) acts as an event that is taken xx mS after the state is entered where the tm(xx) trigger is used as outgoing transition. So the timer is started when the state is entered, when the state is exited (for any reason) the timer is stopped.

This means that the minimum time for a tm() trigger is the time that an RTOS tick takes. Since this is done as slow as possible but as fast as needed (Due to the fact that servicing a timer interrupt causes a system load) a tick-time of more than 1 mS is well possible.

If your tick-time is 20mS, you will have to realize that a tm(65) in fact will give you a tm(40) at least. Since the framework can either be just before or just after a tick it is possible that you have to wait 20mS before the first tick is started. So the actual timer delay of tm(65) is between 80mS and 100mS.



### Info

If you want timers that are either more accurate or faster then you can use your CPU's hardware timer(s)  
Just program the timer to a value that you need and let the timer's interrupt routine call a triggered operation that causes a state-chart transition.

### Attention

The Windows Framework uses a tick-time of 100mS due to the bad timer response of Windows, which still uses the 18.5 mS timer, derived from the 4.77MHz clock-frequency of the first IBM PC in 1981

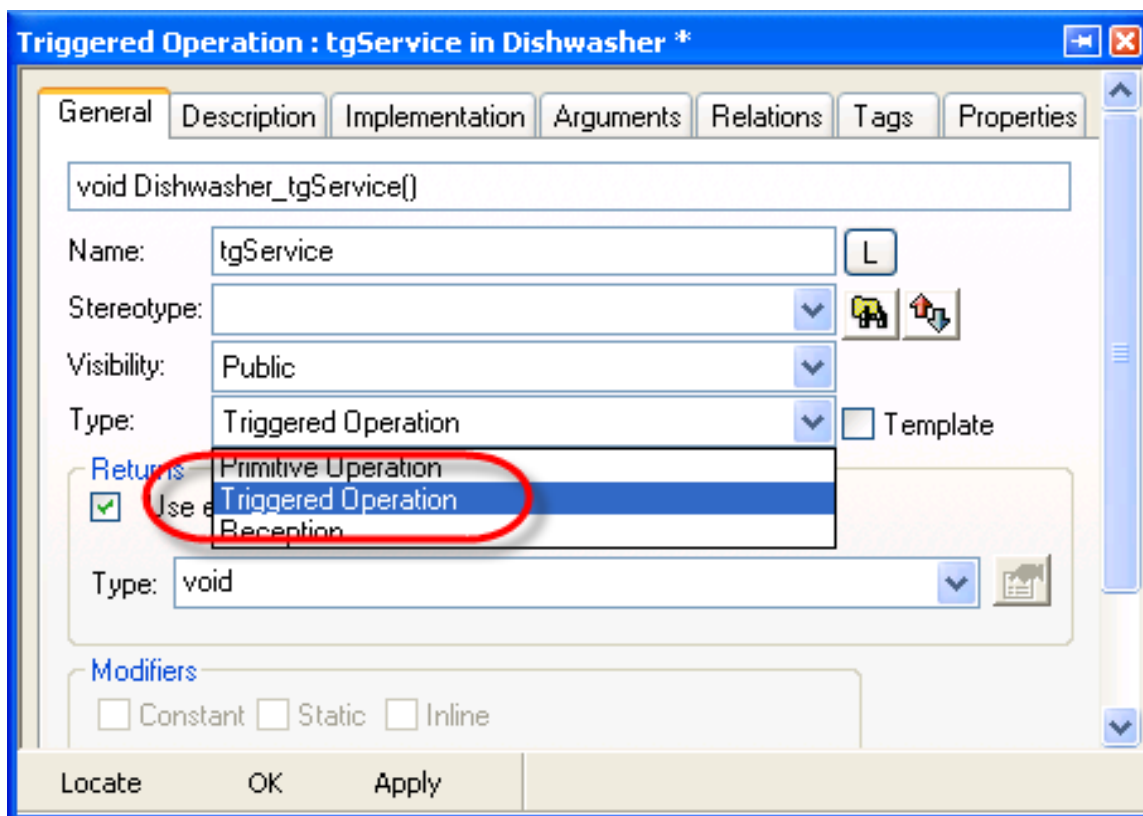
## Triggered Operations

Beside events and timers there is another possibility to trigger a transition, so-called triggered operations. Events and timers both have a disadvantage namely that they are both executed via the event queue.

This means that there is a delay in execution which can be quite long on a system with a high load. It is possible that there are already many events in the queue that wait for processing. This will give events and timers an extra delay.

Another problem might be that events can be sent faster than the framework can handle them. The event queue is static and the size is not unlimited, after the queue is full the framework will run into the error handler function.

The solution to this is use a triggered operation which is basically no more than a synchronous event, also known as a blocking call.



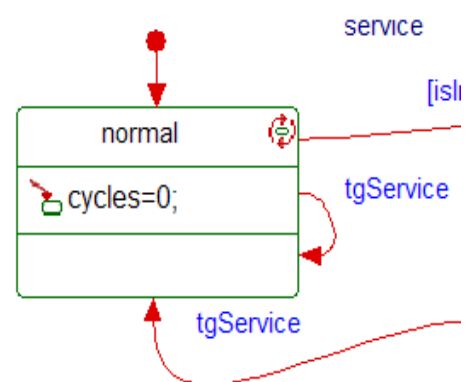
A triggered operation can be used on a transition in a state-chart in exactly the same way as an event. For example, you could change all the events in the dishwasher to be triggered operations. (It is often a good idea to change also the prefix `ev` to `tg`)

Note that the syntax for calling a triggered operation is the same as for an operation.

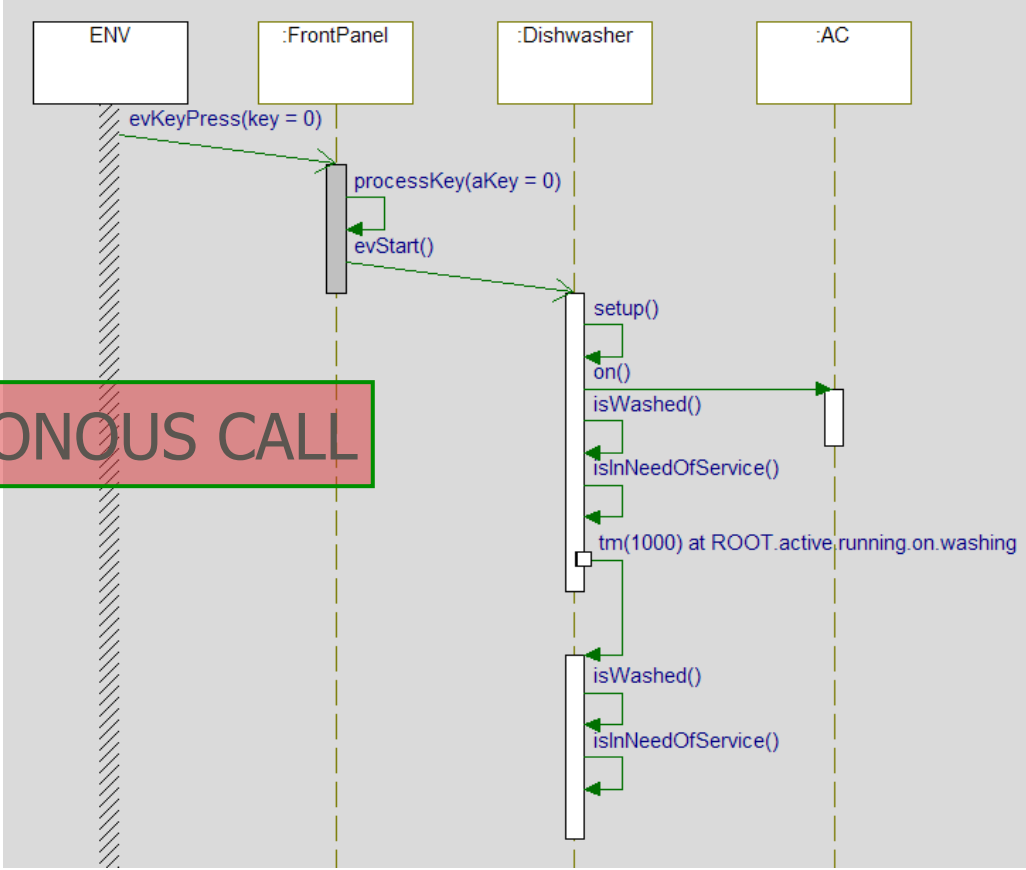
Event: `CGEN (me->itsDishWasher, evStart() );`

Triggered Operation: `DishWasher_tgStart( me->itsDishwasher);`

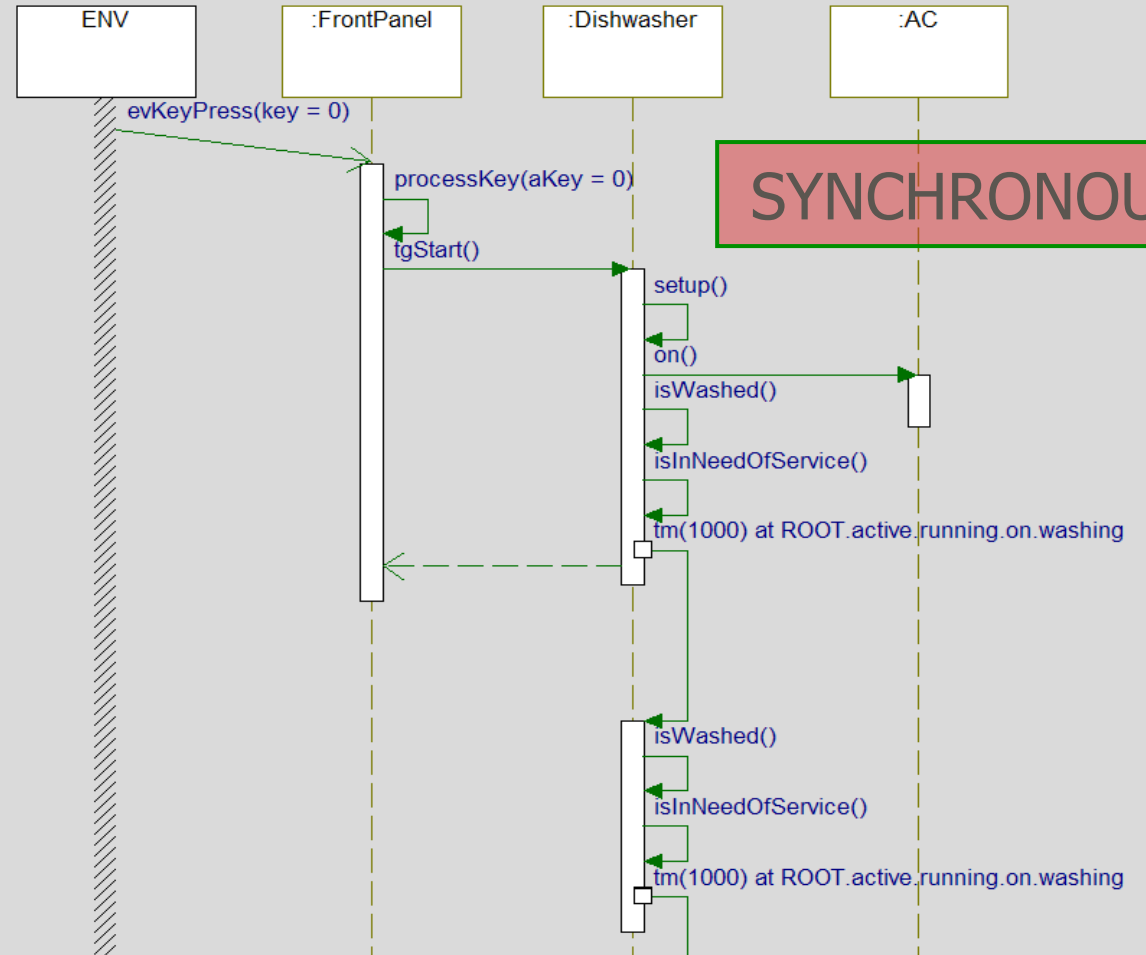
The only difference is that the latter will be executed immediately as we can see on the next two sequence diagrams.



# ASYNCHRONOUS CALL



# SYNCHRONOUS CALL



Product:

# **Embedded UML Start-Up Training**

Author:

**Marco Matuschek**

**Walter van der Heiden**

Editor:

**Willert Software Tools GmbH**

Hannoversche Strasse 21

DE - 31675 Bückeburg

[www.willert.de](http://www.willert.de)



IBM® is a registered trademark of International Machines Corporation  
Rational® is a registered trademark owned by IBM  
DOORS® is a registered trademark owned by IBM  
Rhapsody® is a registered trademark owned by IBM  
MS Word® is a registered trademark of Microsoft Corporation