

# **Modeling Methodology**

## **MODEL-BASED DESIGN & VERIFICATION OF EMBEDDED SYSTEMS (MODEVES)**

**Version: 2.0.0.0**



## Table of Contents

|  |           |
|--|-----------|
| <b>1. Introduction.....</b>  | <b>7</b>  |
| <b>1.1 Document Structure.....</b>                                 | <b>7</b>  |
| <b>2. Modeling Guidelines for Structural Aspects .....</b>         | <b>9</b>  |
| <b>2.1 Flow Ports .....</b>  | <b>9</b>  |
| <b>2.2 Properties.....</b>   | <b>9</b>  |
| <b>2.3 Enumeration .....</b>                                       | <b>10</b> |
| <b>2.4 Clock and Timer .....</b>                                   | <b>11</b> |
| <b>2.5 Summary.....</b>  | <b>12</b> |
| <b>3. Modeling Guidelines for Behavioral Aspects .....</b>         | <b>13</b> |
| <b>3.1 Region.....</b>   | <b>13</b> |
| <b>3.2 State.....</b>  | <b>14</b> |
| <b>3.2.1 How to Specify Entry / Exit conditions .....</b>          | <b>14</b> |
| <b>3.2.2 How to call Do Activity .....</b>                         | <b>17</b> |
| <b>3.3 Transition.....</b>   | <b>19</b> |
| <b>3.3.1 How to Specify Effect.....</b>                            | <b>19</b> |
| <b>3.3.2 How to Specify Guard.....</b>                             | <b>20</b> |
| <b>3.3.3 How to specify trigger.....</b>                           | <b>22</b> |
| <b>3.4 Initial and Final State .....</b>                           | <b>23</b> |
| <b>3.5 Fork .....</b>  | <b>23</b> |
| <b>3.6 Join .....</b>  | <b>24</b> |
| <b>3.7 Choice.....</b>   | <b>24</b> |
| <b>3.8 Summary.....</b>  | <b>25</b> |
| <b>4. Properties / Assertions Specification through SVOCL.....</b> | <b>26</b> |
| <b>4.1 SVSeq (s, d, p).....</b>                                    | <b>26</b> |
| <b>4.2 SVRep (p,r) .....</b>                                       | <b>27</b> |
| <b>4.3 SVImplication (ant,con,t) .....</b>                         | <b>27</b> |
| <b>4.4 SVStable (expr) .....</b>                                   | <b>28</b> |
| <b>4.5 SVChanged (expr).....</b>                                   | <b>28</b> |
| <b>4.6 SVPast (expr, ct).....</b>                                  | <b>29</b> |
| <b>4.7 SVRose (expr).....</b>                                      | <b>29</b> |
| <b>4.8 SVFell (expr).....</b>                                      | <b>30</b> |
| <b>4.9 Disif expression.....</b>                                   | <b>30</b> |

|       |   |    |
|-------|---|----|
| 4.10  | Property Name .....                                       | 31 |
| 4.11  | Rules for Start and Close Brackets.....                   | 32 |
| 5.    | Properties / Assertions Specification through NLCTL.....  | 33 |
| 6.    | Examples of traffic lights and elevator case studies..... | 33 |
| 6.1   | Traffic Lights Case Study Description .....               | 34 |
| 6.2   | Modeling Structural Aspects.....                          | 34 |
| 6.3   | Modeling Behavioral aspects.....                          | 35 |
| 6.3.1 | Transition.....   | 37 |
| 6.3.2 | State.....  | 39 |
| 6.3.3 | Other state machine nodes .....                           | 42 |
| 6.4   | Modeling SystemVerilog Assertions .....                   | 42 |
| 6.5   | Elevator Case Study Description .....                     | 44 |
| 6.6   | Demo of MODEVES Transformation Engine.....                | 45 |

## List of Figures

|   |    |
|---|----|
| <b>Figure 1:</b> Flow Port Example .....  | 9  |
| <b>Figure 2:</b> Example of properties and flow ports in block .....  | 10 |
| <b>Figure 3:</b> Enumeration example .....  | 10 |
| <b>Figure 4:</b> Block diagram of clock .....   | 11 |
| <b>Figure 5:</b> Example of pre-condition for calling clock / timer .....   | 11 |
| <b>Figure 6:</b> Example of clock / time declaration .....  | 11 |
| <b>Figure 7:</b> State machine Regions example .....  | 13 |
| <b>Figure 8:</b> Attributes of state node .....   | 14 |
| <b>Figure 9:</b> Example of specifying Entry condition for state node (OpaqueBehavior) .....                          | 15 |
| <b>Figure 10:</b> Example of specifying Entry condition for state node (Language Selection) .....                     | 15 |
| <b>Figure 11:</b> Example of specifying Entry condition for state node (condition specification) .....                | 16 |
| <b>Figure 12:</b> Defining activity in state for “Do Activity” .....  | 17 |
| <b>Figure 13:</b> Defining CallBehaviorAction to call activity in state for “Do Activity” .....                       | 18 |
| <b>Figure 14:</b> Configuring CallBehaviorAction to set behavior and ports .....                                      | 18 |
| <b>Figure 15:</b> Transition attributes (Name, Effect, Guard and Trigger) .....                                       | 19 |
| <b>Figure 16:</b> Specifying effect of the transition through OpaqueBehavior .....                                    | 20 |
| <b>Figure 17:</b> Specifying guard condition of transition .....  | 20 |
| <b>Figure 18:</b> Specifying guard condition of transition (constraint element and specification) .....               | 21 |
| <b>Figure 19:</b> Another way of specifying Guard of Transition .....   | 21 |
| <b>Figure 20:</b> Accessing guard condition which is specified through constraint block .....                         | 22 |
| <b>Figure 21:</b> Specifying trigger ports of transition .....  | 22 |
| <b>Figure 22:</b> Example of initial and final state of state machine diagram .....                                   | 23 |
| <b>Figure 23:</b> Example of state machine fork node .....  | 24 |
| <b>Figure 24:</b> Example of state machine join and choice nodes .....  | 24 |
| <b>Figure 25:</b> Extend OclExpression for SVOCL DisifExp .....   | 30 |
| <b>Figure 26:</b> Syntax of SVOCL DisifExp .....  | 31 |
| <b>Figure 27:</b> Example of SVOCL assertion in papyrus .....   | 32 |
| <b>Figure 28:</b> Assertion Example in NLCTL .....  | 33 |
| <b>Figure 29:</b> Modeling structural aspects of traffic lights controller in papyrus .....                           | 34 |
| <b>Figure 30:</b> Activity diagram of timer implemented in clock block .....  | 35 |
| <b>Figure 31:</b> State machine diagram of traffic lights controller .....  | 36 |
| <b>Figure 32:</b> Example of specifying transition effect for traffic lights controller .....                         | 37 |
| <b>Figure 33:</b> Example of specifying effect condition for traffic lights controller .....                          | 38 |
| <b>Figure 34:</b> Example of specifying transition guard condition for traffic lights controller .....                | 38 |
| <b>Figure 35:</b> Specifying guard condition through constraint block for traffic lights controller .....             | 38 |
| <b>Figure 36:</b> specifying trigger ports for traffic lights controller .....  | 39 |
| <b>Figure 37:</b> Example of state attributes (Name, Do Activity, Entry and Exit) for traffic lights controller ..... | 40 |
| <b>Figure 38:</b> Specifying activity name in state (Do Activity) for traffic lights controller .....                 | 40 |
| <b>Figure 39:</b> Adding CallBehaviorAction to call activity in state for traffic lights controller .....             | 41 |
| <b>Figure 40:</b> Setting CallBehaviorAction behavior and port for traffic lights controller .....                    | 41 |
| <b>Figure 41:</b> Representing traffic lights controller assertions (1 to 3) in SVOCL .....                           | 43 |
| <b>Figure 42:</b> Representing traffic lights controller assertions (4 to 7) in SVOCL .....                           | 43 |

|  |    |
|--|----|
| <b>Figure 43:</b> Representing traffic lights controller assertions (8 to 11) in SVOCL ..... | 44 |
| <b>Figure 44:</b> Design of Elevator .....   | 44 |
| <b>Figure 45:</b> Verification aspects of elevator in NLCTL and SVOCL .....                  | 45 |
| <b>Figure 46:</b> MODEVES transformation engine main interface.....                          | 45 |

# 1. Introduction

## 1.1 Document Structure

This article has following chapters:

**Chapter 2:** This chapter define the rules to model structural aspects of embedded systems, such that, SystemVerilog structure code can be generated through MODEVES transformation engine. The concepts of SYSML block definition diagrams have been described to model structural aspects in the context of equivalent SystemVerilog concepts like input / output registers etc.

**Chapter 3:** This chapter define the rules to model behavioral aspects of embedded systems, such that, SystemVerilog behavior code and Timed Automata model can be generated through MODEVES transformation engine. In this chapter, various rules have been described to capture behavioral aspects of embedded systems by utilizing state machine diagram.

**Chapter 4:** This chapter provide the details of SVOCL (SystemVerilog in Object Constraints Language) to specify embedded system properties / constraints at higher abstraction level (model). SVOCL is solely developed and implemented as a part of MODEVES project to specify SystemVerilog Assertions (SVA's) in model. The assertions specified in SVOCL are automatically transform to equivalent SystemVerilog assertions code through MODEVES transformation engine.

**Chapter 5:** This chapter provide the details of NLCTL (Natural Language for Computation Tree Logic) to specify embedded system properties / constraints at higher abstraction level (model) by means of CTL. NLCTL is developed and implemented as a part of MODEVES project to specify CTL assertions in model. The assertions specified in NLCTL are automatically transform to equivalent CTL assertions code through MODEVES transformation engine.

**Chapter 6:** In this chapter, we demonstrate the application of modeling methodology through traffic lights and elevator case studies. This facilitates end users to understand the application of our proposed modeling methodology in order to represent the structure, behavior and assertions at higher abstraction level. The demonstration of MODEVES transformation engine is also presented in order to generate SystemVerilog RTL, Timed Automata model, SVA's and CTL assertion code from the model.





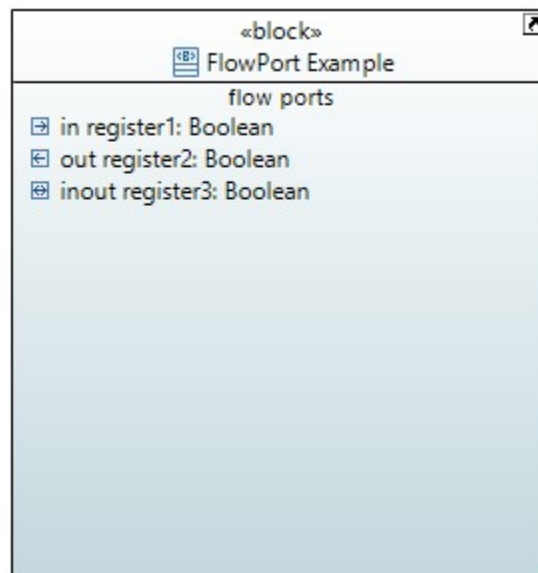
## 2. Modeling Guidelines for Structural Aspects

SysML Block Definition Diagram (BDD) has been proposed to specify structural aspects of embedded systems. Any number of blocks can be used to represent system structure. Following components of BDD have been considered:

### 2.1 Flow Ports

BDD flow ports have been used to represent SystemVerilog input, output and inout registers. Particularly, input flow ports are equivalent to SystemVerilog input logic and output flow ports are equivalent to output logic. Further, both inout flow ports are equivalent to SystemVerilog wire.

**Example:**

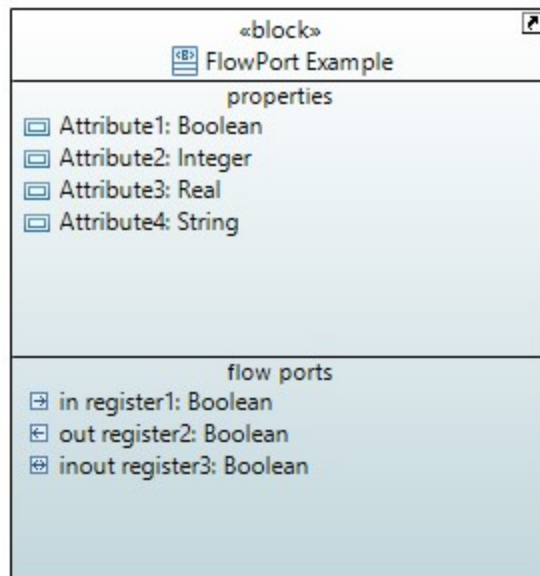


**Figure 1:** Flow Port Example

### 2.2 Properties

Properties can be used to represent SystemVerilog integer, string, real and Boolean data types. It is important to mention that properties compartment of BDD can be defined along with flow ports (see example below) or separately depending upon particular modeling requirements. MODEVES transformation engine is capable of transforming any number of BDD's, flow ports and properties as far as given modeling guidelines have been followed.

**Example:**

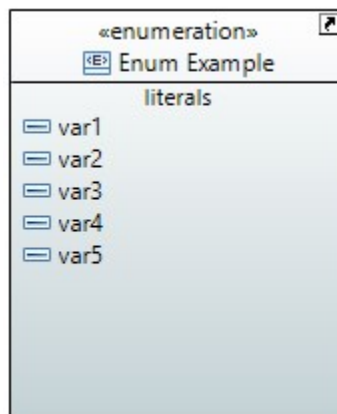


**Figure 2:** Example of properties and flow ports in block

### 2.3 Enumeration

UML enumeration can be used to represent corresponding SystemVerilog enumeration. Enumeration type can be used as a flow ports to model system structure.

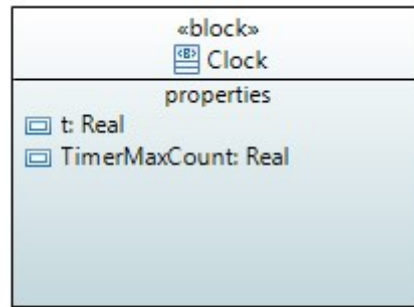
**Example:**



**Figure 3:** Enumeration example

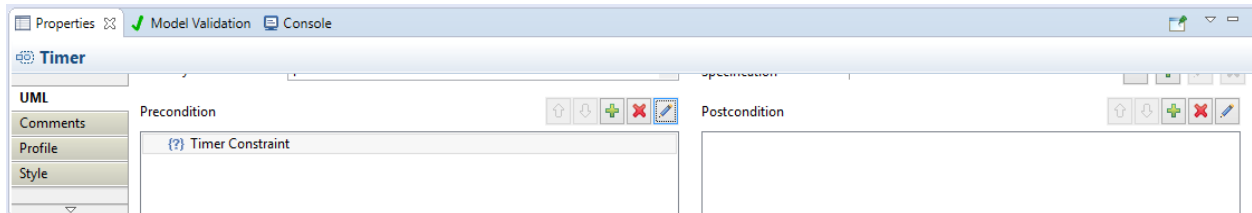
## 2.4 Clock and Timer

In MODEVES modeling framework, clock and timers are implemented as a Signal type. Separate block (clock) has been proposed where clock and timer are implemented as two activity diagrams. The user should utilize clock block to model clock and timer of system (if required). The clock block is shown in **Figure 4** below



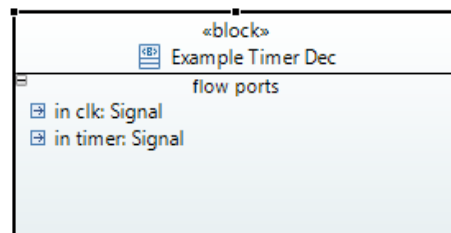
**Figure 4:** Block diagram of clock

The user should set the default values of t and TimerMaxCount variables in order to adjust clock and timer values respectively. For example, if it is required to send the clock signal on some particular port after 5 seconds then TimerMaxCount should be assign 5 as a default value (timer will be reset to 0 after 5 seconds). It is also possible to set the precondition to call timer signal on particular port as shown **Figure 5**



**Figure 5:** Example of pre-condition for calling clock / timer

**Example:**



**Figure 6:** Example of clock / time declaration

## **2.5 Summary**

The guidelines to model system structure, as given from Section **2.1** to **2.4**, are sufficient to model simple as well as complex embedded system structure. MODEVES transformation engine is able to generate SystemVerilog structure code as far as model is developed through given guidelines (Section **2.1** to Section **2.4**). However, SystemVerilog has various other concepts and data types (e.g. typedef, time etc.) that cannot be currently represented through given modeling guidelines. For such unsupported SystemVerilog constructs, user can perform minor alterations on generated source code to meet particular objective.

### 3. Modeling Guidelines for Behavioral Aspects

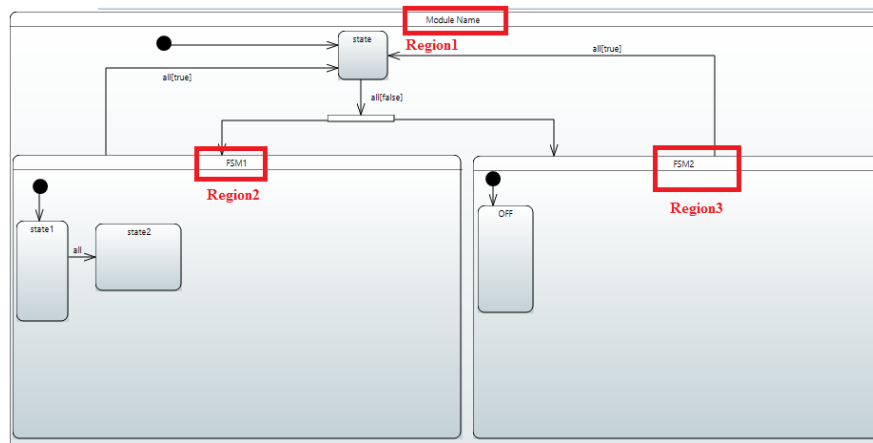
Modeling behavioral aspects of embedded systems is a crucial part of requirement specification. In MODEVES framework, we propose state machine diagram to specify the behavioral aspects. However, it is not possible to consider all state machine diagram concepts / constructs in transformation engine. Therefore, we select significant state machine constructs with certain restrictions to model system behavior, so that, corresponding SystemVerilog RTL code and Timed Automata model can be generated through MODEVES transformation engine. The guidelines for modeling system behavior in the context of state machine diagram concepts / constructs are given below:

#### 3.1 Region

State Machine diagram can have various regions. We define following rules for regions:

- The default region of state machine (always named as Region1) is logically equivalent to SystemVerilog module and Timed Automata. The name of the state machine is logically equivalent to SystemVerilog module name and Timed Automata name.
- It is possible to have various state machines within main state machine. Each sub state machine should be represented through separate region (within Region1). For example, if there are three state machines within main state machine then these state machines should be represented through Region2, Region3 and Region4 (where Region1 is used to represent main state machine).

**Example:**



**Figure 7:** State machine Regions example

## 3.2 State

State node of state machine diagram has been used to represent different system states. Following important properties have been considered for behavioral modeling:

**Name:** it is used to represent state name.

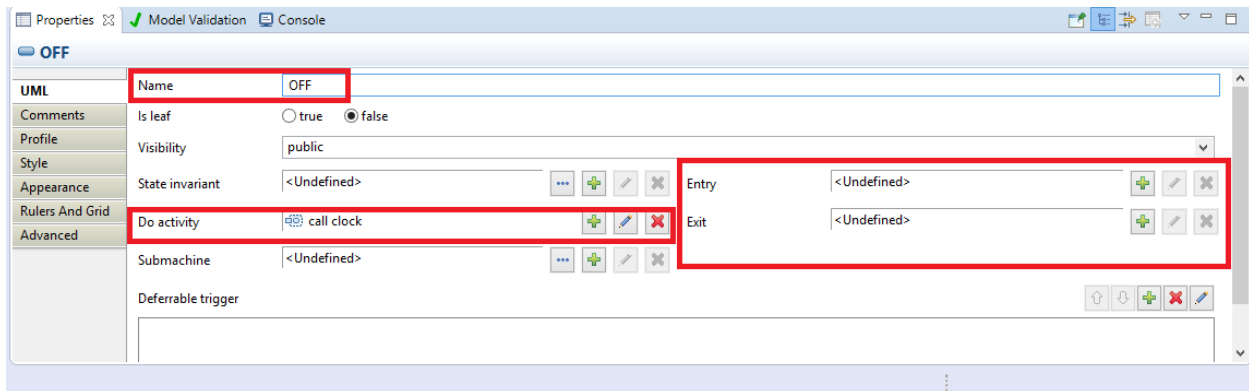
**Entry:** It is used to specify particular condition when system is entered into state.

**Exit:** It is used to specify particular condition when system exist from the state.

**Do Activity:** It is used to specify particular activity that need to be executed in state. This property is mostly used to call clock and timer activities in given state. However, it is also possible to call any other user-defined activity.

These four attributes of state node has been used to specify behavior. MODEVES transformation engine is capable of generating corresponding SystemVerilog code and timed automata model for these attributes.

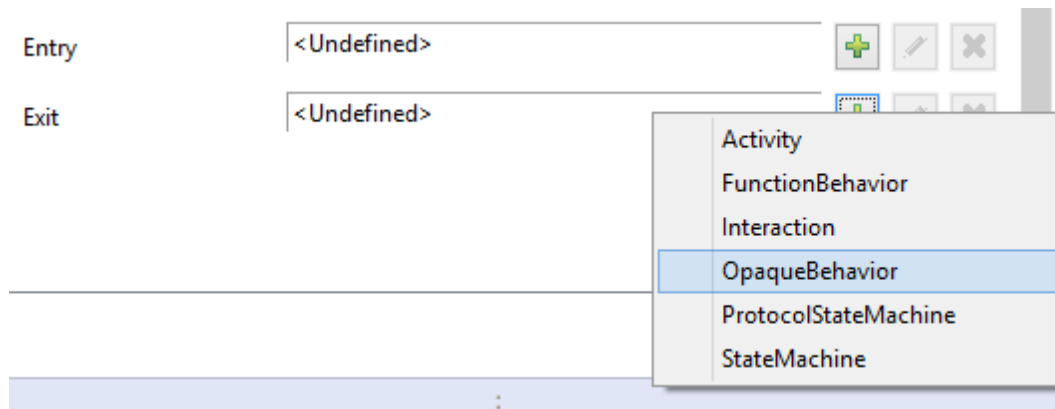
**Examples:**



**Figure 8:** Attributes of state node

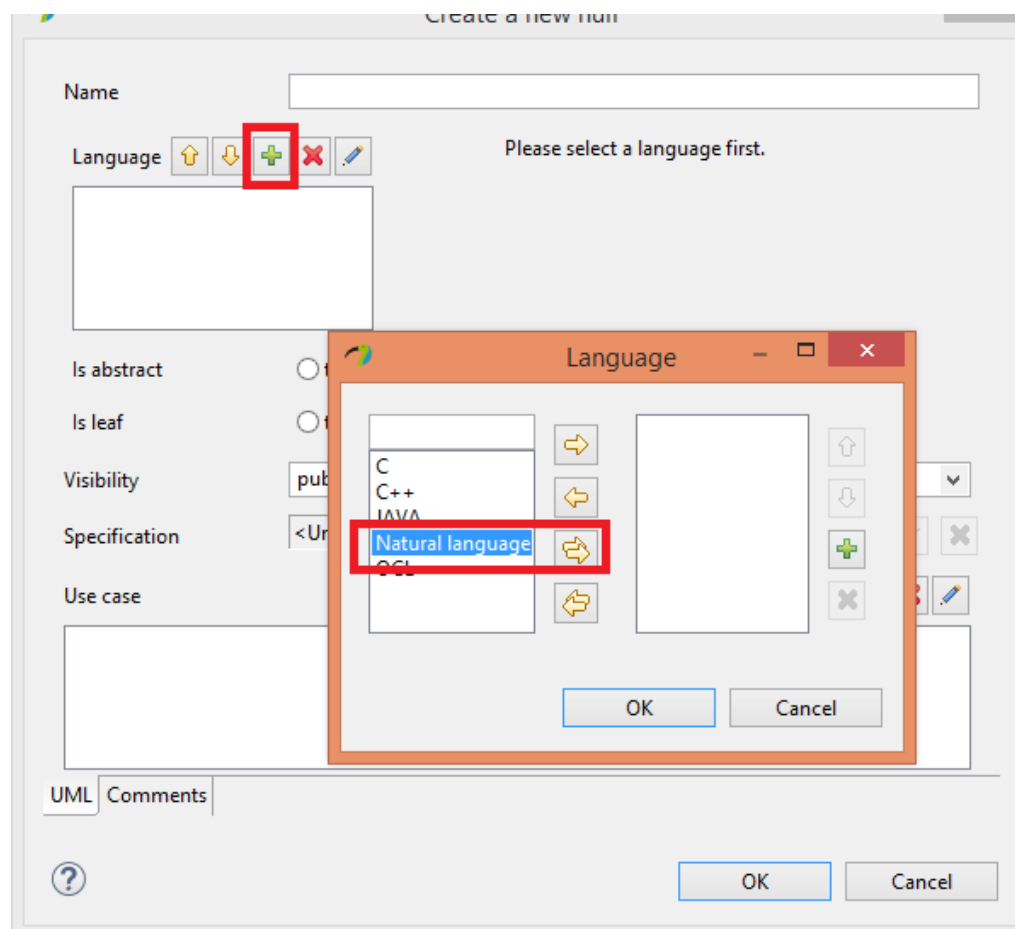
### 3.2.1 How to Specify Entry / Exit conditions

Firstly, click on add button (+) and select OpaqueBehavior as shown in **Figure 9**



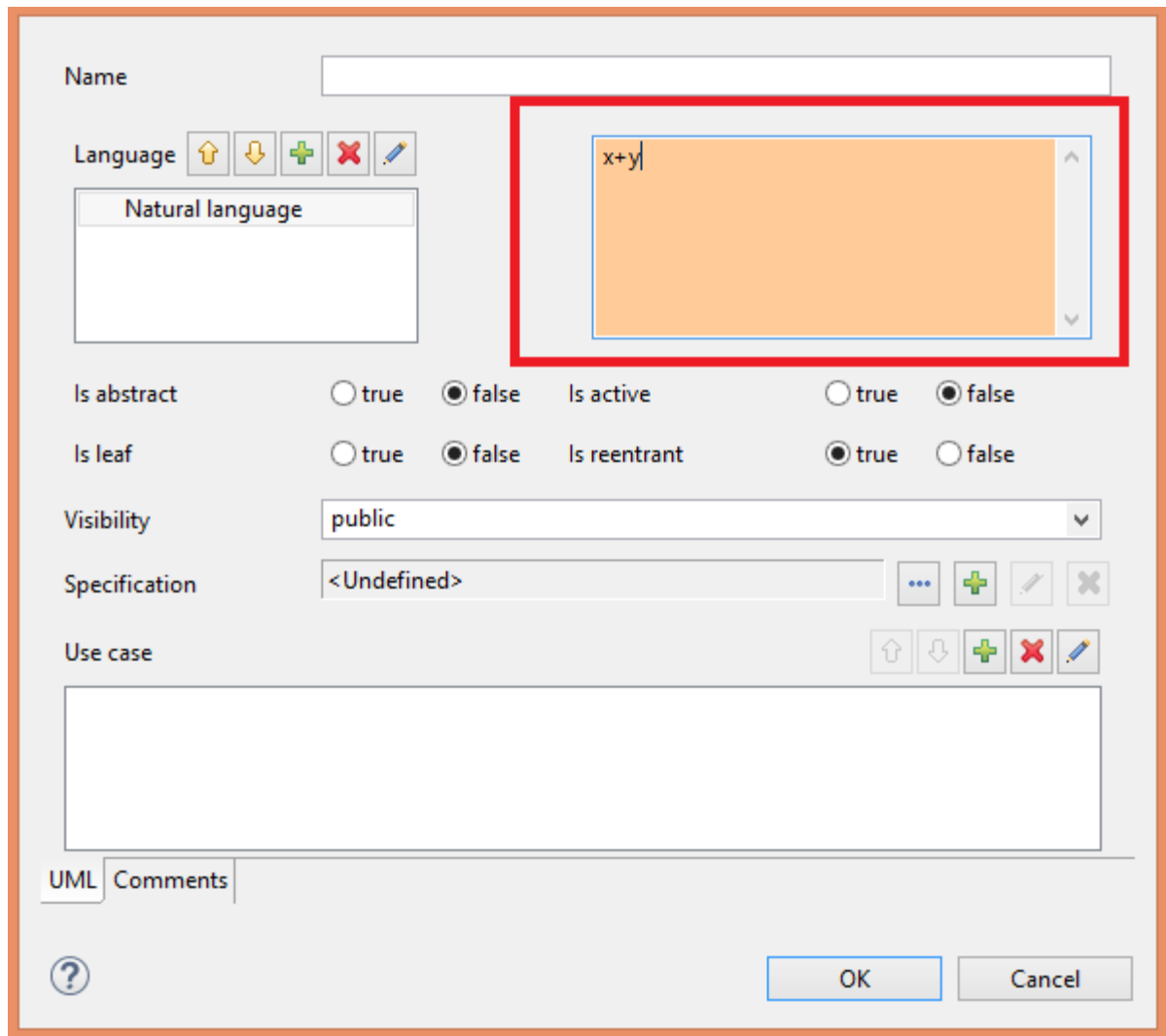
**Figure 9:** Example of specifying Entry condition for state node (OpaqueBehavior)

Secondly, Select Natural Language and click OK as shown **Figure 10**



**Figure 10:** Example of specifying Entry condition for state node (Language Selection)

Finally, write specification as shown in **Figure 11**



The image shows a dialog box for specifying a state node. It has a light gray background and an orange border. At the top, there is a 'Name' field. Below it, a 'Language' section contains icons for up, down, add, delete, and edit, followed by a list box showing 'Natural language'. To the right of the 'Language' section is a large orange text area with a red border, containing the text 'x+y'. Below the 'Language' section are four groups of radio buttons: 'Is abstract' (true, false), 'Is active' (true, false), 'Is leaf' (true, false), and 'Is reentrant' (true, false). Below these is a 'Visibility' dropdown menu set to 'public'. The 'Specification' field contains '<Undefined>' and has icons for adding, deleting, and editing. Below this is a 'Use case' section with icons for up, down, add, delete, and edit, followed by a large empty text area. At the bottom, there are two tabs: 'UML' and 'Comments'. A help icon (?) is on the bottom left, and 'OK' and 'Cancel' buttons are on the bottom right.

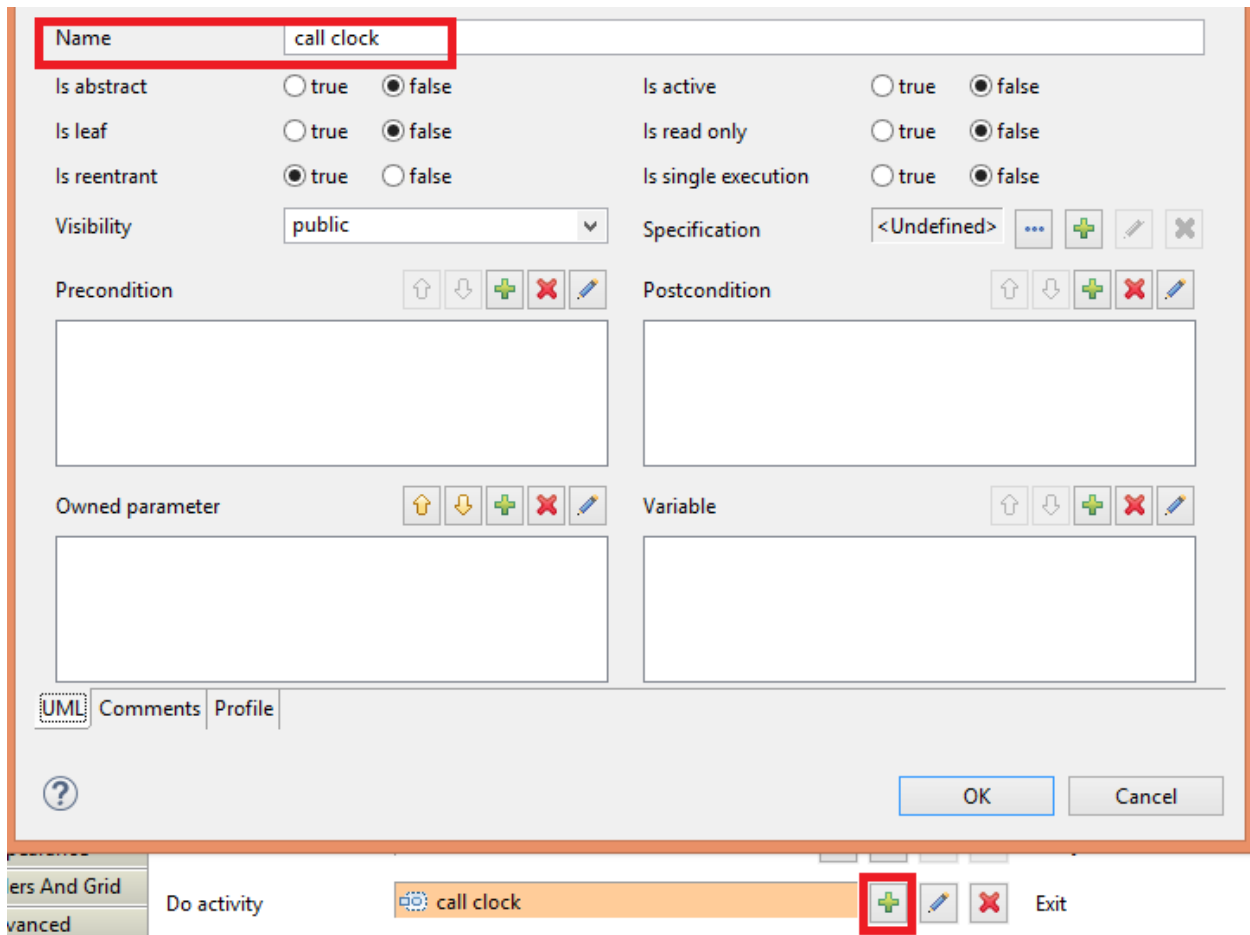
**Figure 11:** Example of specifying Entry condition for state node (condition specification)

Same Process can be used to specify Exit condition.



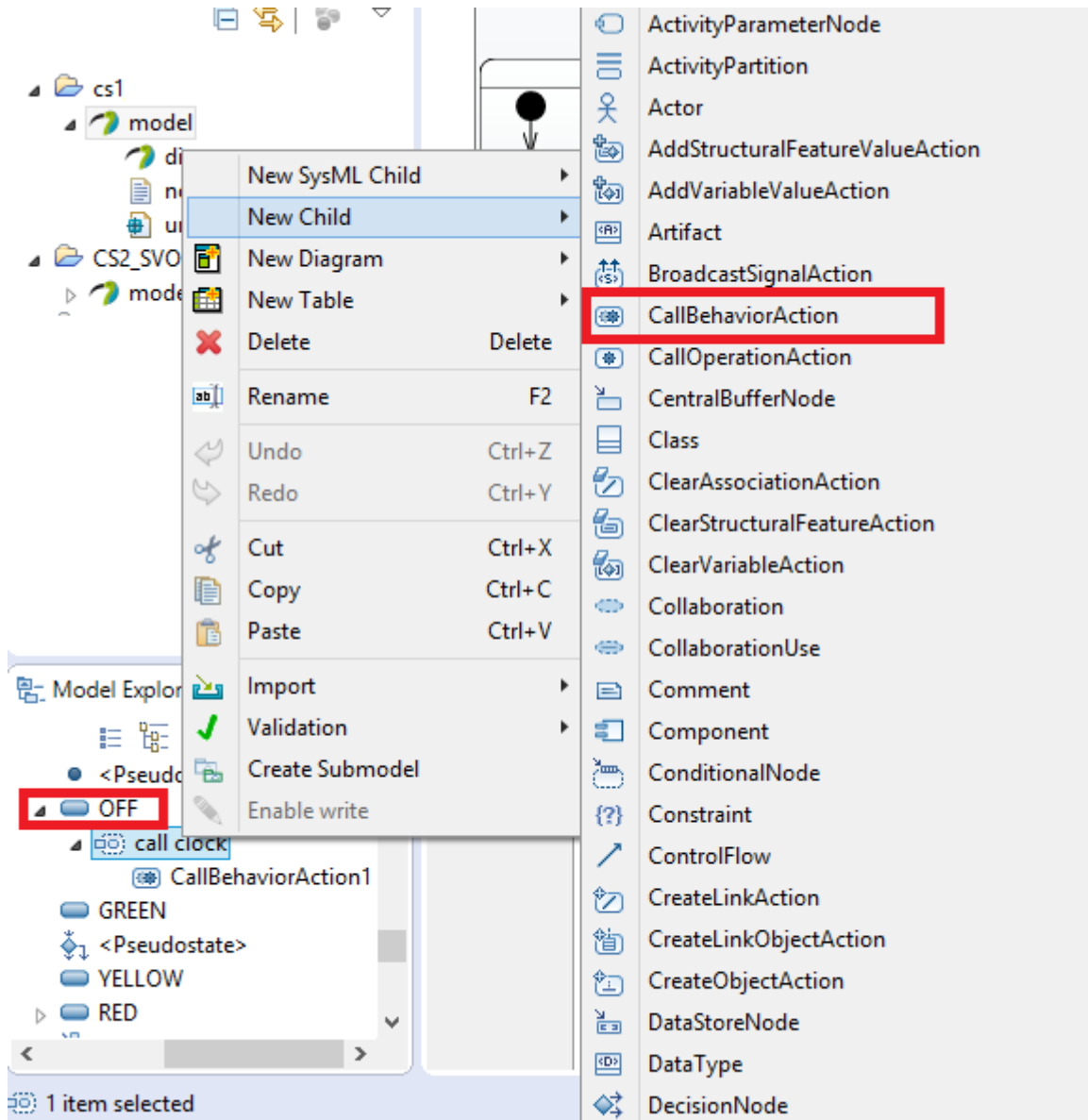
### 3.2.2 How to call Do Activity

Firstly, click (+) button on “Do Activity” and specify activity name that need to be executed as in **Figure 12**



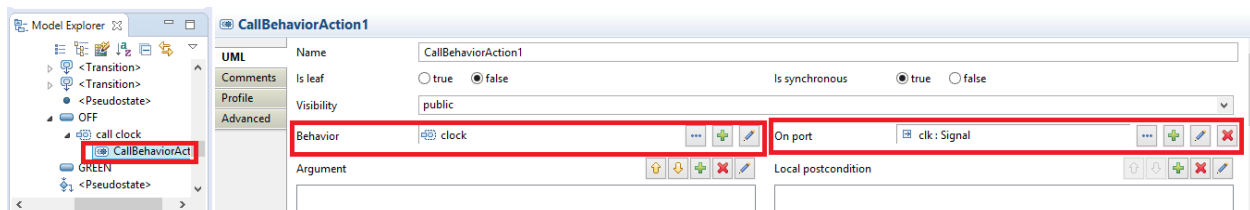
**Figure 12:** Defining activity in state for “Do Activity”

Secondly, select the state (on which activity need to be called) through model explorer. Right click, add child and add CallBehaviorAction. The process is shown in **Figure 13**



**Figure 13:** Defining CallBehaviorAction to call activity in state for “Do Activity”

Finally, click on CallBehaviorAction to view its properties. Specify behavior (activity that need to be executed) and port as shown in *Figure 14*



**Figure 14:** Configuring CallBehaviorAction to set behavior and ports

### 3.3 Transition

Transition has been used to specify particular condition to move from current state to next state. Following properties of transition has been considered:

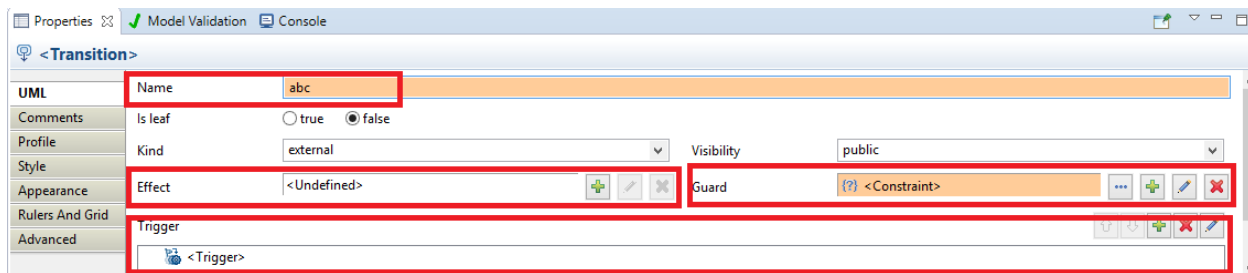
**Name:** Specify the name of transition.

**Effect:** Specify the condition that need to be executed after successful completion of transition.

**Guard:** It is used to specify guard condition that must be true for successful execution of transition (move from current to next state accordingly). Two attributes of guard have been used i.e. Constraint Elements and Specification.

**Trigger:** It is used to specify ports / variables that will be considered (trigger) for the execution of transition.

#### Examples:



**Figure 15:** Transition attributes (Name, Effect, Guard and Trigger)

#### 3.3.1 How to Specify Effect

Click on add (+) button and select OpaqueBehavior. New interface will be open. Thereafter, select Natural Language to specify particular condition as shown in **Figure 16**

The dialog box is titled 'UML Element Properties' and is used for specifying the effect of a transition through OpaqueBehavior. It contains the following fields and controls:

- Name:** A text input field.
- Language:** A dropdown menu with icons for up, down, add, remove, and edit. The current selection is 'Natural language'.
- Condition:** A large text area for specifying the condition.
- Is abstract:** Radio buttons for 'true' and 'false' (selected).
- Is active:** Radio buttons for 'true' and 'false' (selected).
- Is leaf:** Radio buttons for 'true' and 'false' (selected).
- Is reentrant:** Radio buttons for 'true' (selected) and 'false'.
- Visibility:** A dropdown menu with 'public' selected.
- Specification:** A text input field with '<Undefined>' and icons for add, remove, and edit.
- Use case:** A large text area with icons for up, down, add, remove, and edit.
- UML Comments:** A tab at the bottom left.
- Buttons:** '?' (help), 'OK', and 'Cancel' at the bottom right.

**Figure 16:** Specifying effect of the transition through OpaqueBehavior

### 3.3.2 How to Specify Guard

Click on add (+) button and Select constraint as shown in **Figure 17**

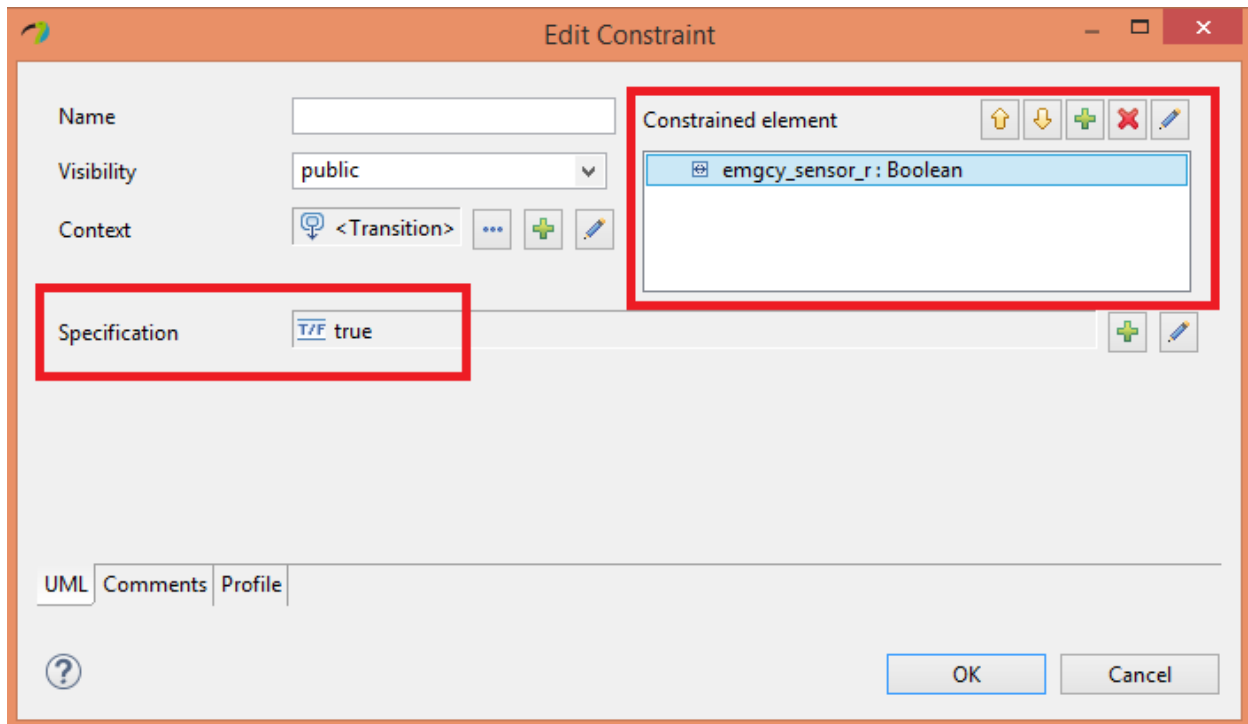
The dialog box shows the 'UML Element Properties' for a transition. The 'Guard' field is highlighted, and a dropdown menu is open, showing the following options:

- Constraint
- DurationConstraint
- InteractionConstraint
- IntervalConstraint
- TimeConstraint

The 'Visibility' field is set to 'public' and the 'Guard' field is set to '<Undefined>'.

**Figure 17:** Specifying guard condition of transition

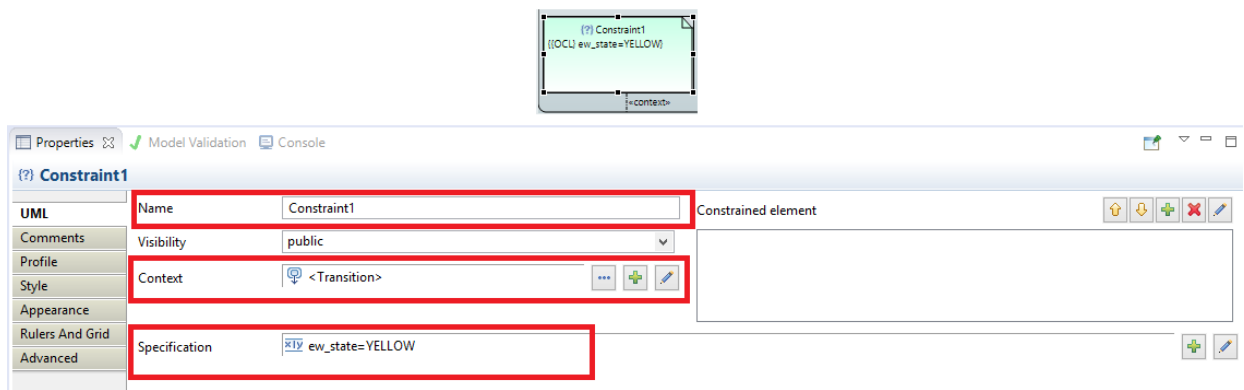
This will open new interface. Specify constraint elements and specification as shown in **Figure 18**



**Figure 18:** Specifying guard condition of transition (constraint element and specification)

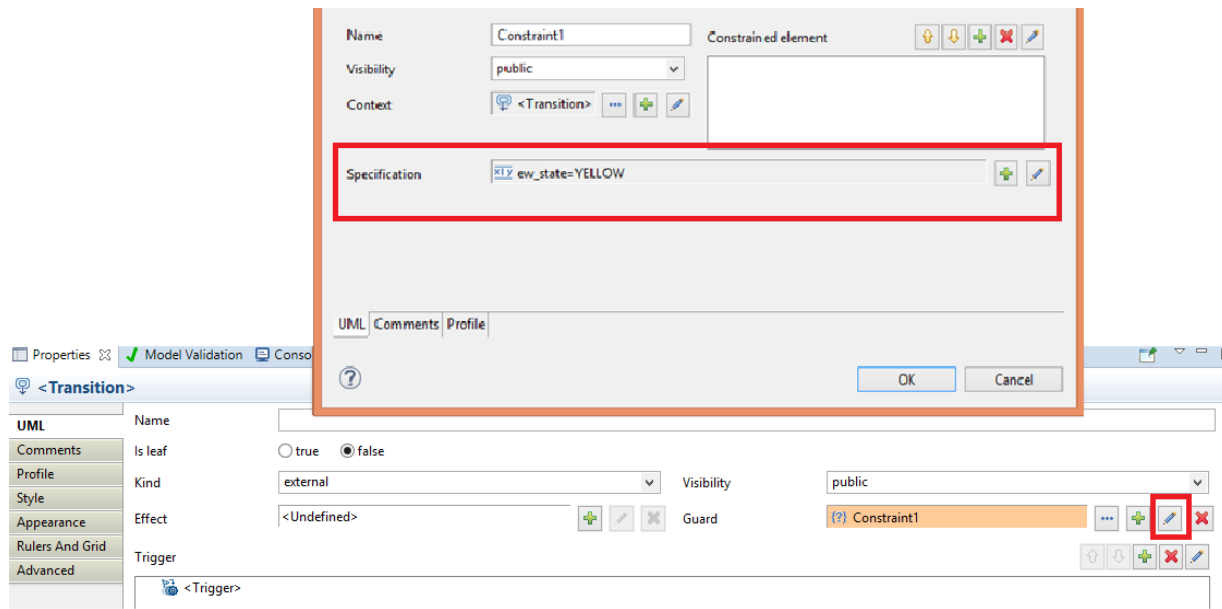
### Another Way for Guard Condition Specification:

It is also possible to assign guard condition through constraint block. Drop constraint block, set its context and write constraint in specification as shown in **Figure 19**



**Figure 19:** Another way of specifying Guard of Transition

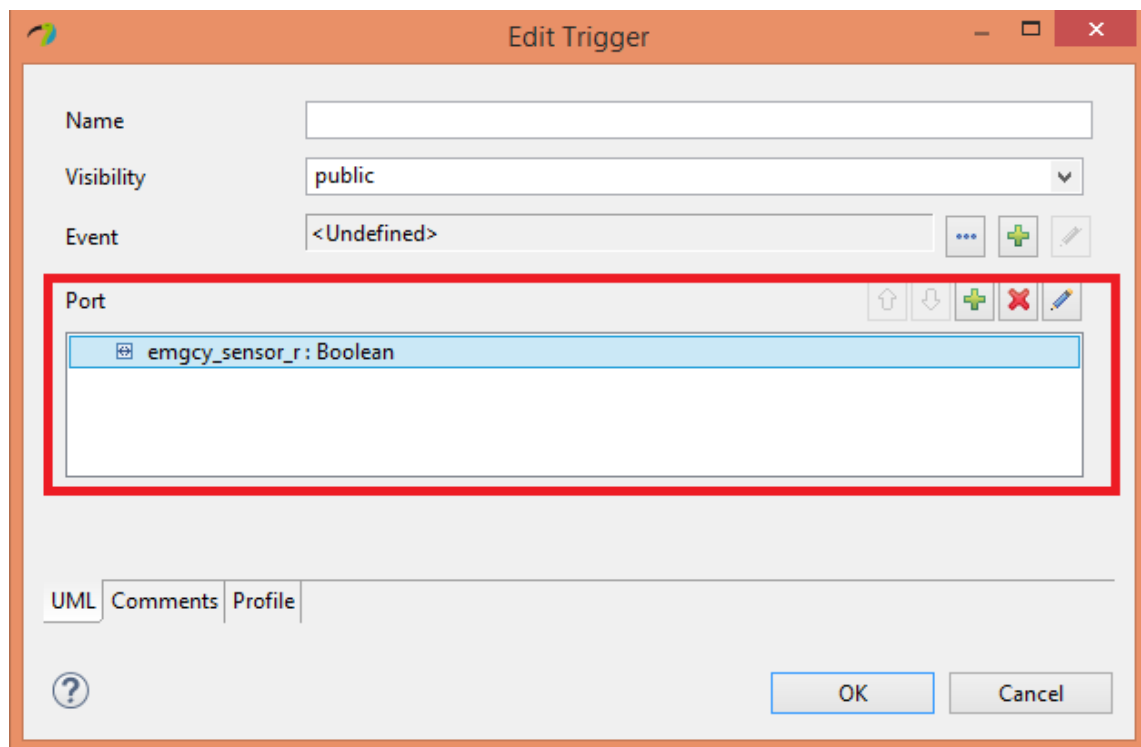
Now, same constraint is accessible in Guard of particular transition because transition is specified as a context of this constraint block. In **Figure 20** , you can see that constraint written through block (**Figure 19**) is accessible in corresponding transition guard attribute.



**Figure 20:** Accessing guard condition which is specified through constraint block

### 3.3.3 How to specify trigger

Click on add (+) button to add trigger. New interface will open. Enter the ports that need to be used for transition as shown in **Figure 21**

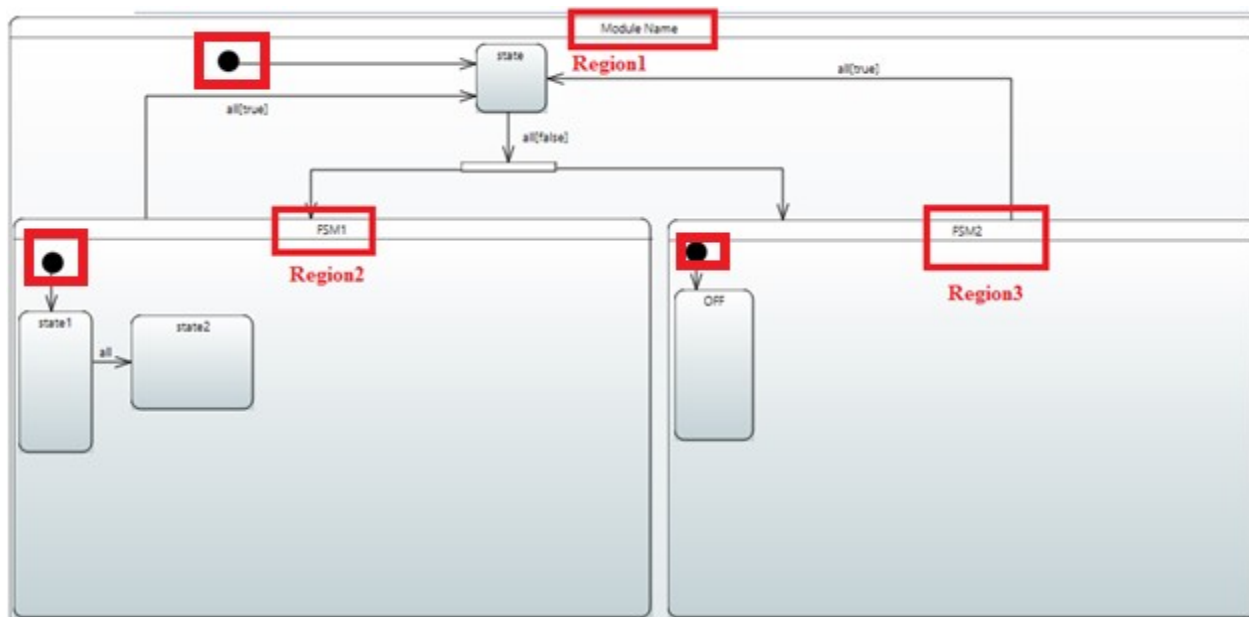


**Figure 21:** Specifying trigger ports of transition

### 3.4 Initial and Final State

State machine diagram initial and final state nodes can be used to represent start and end execution points. It is possible to use more than one start and end execution points within main state machine. For example, each Region1, Region2 and Region3 can have initial and final state.

**Example:**

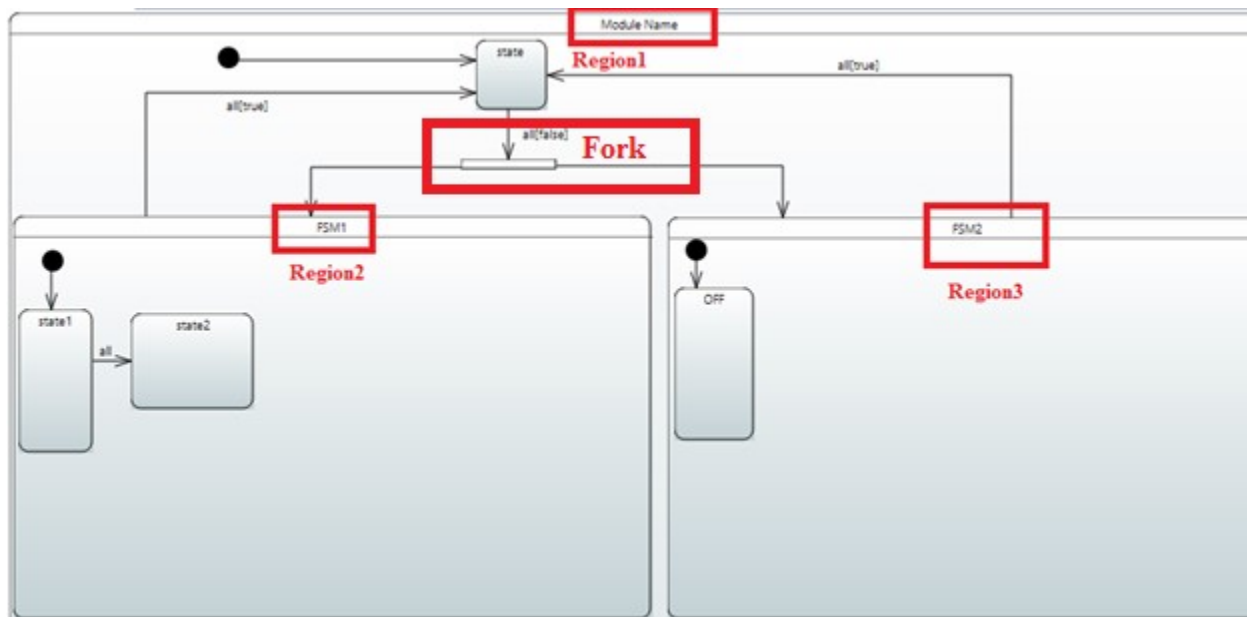


**Figure 22:** Example of initial and final state of state machine diagram

### 3.5 Fork

State machine fork node has been used to split single input transition to two or more output transitions for concurrent execution. Basically, it simply shift the control to two or more output transition so it is not possible to specify any guard condition on output transitions.

**Example:**



**Figure 23:** Example of state machine fork node

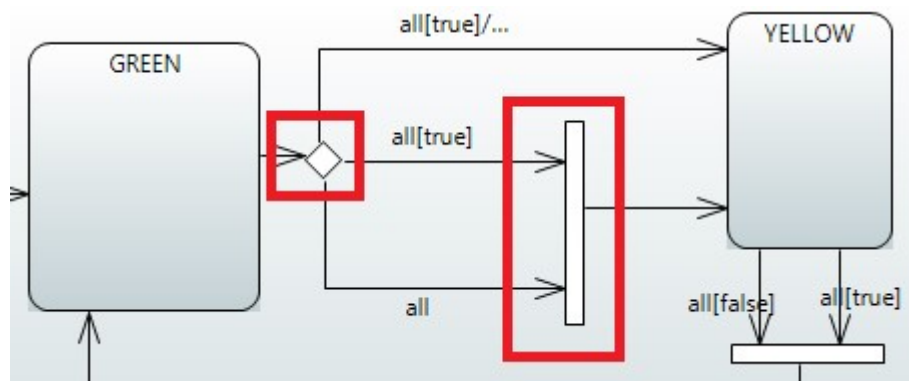
### 3.6 Join

State machine join node is used to join two or more input transition into single output transition. Logically, it checks combination of several conditions (e.g. through && operator etc.) to produce single output path when all input conditions becomes true. Example is shown in **Figure 24**

### 3.7 Choice

State machine choice node is used for conditional branching. It is normally utilized to represent IF ELSE / CASE statements.

**Example:**



**Figure 24:** Example of state machine join and choice nodes



### **3.8 Summary**

The concepts / constructs of state machine diagram, presented in Section **3.1** to **3.7**, have been fully supported in MODEVES transformation engine. Consequently, it is possible to generate SystemVerilog RTL code and Timed Automata model through MODEVES transformation engine as far as system design (model) is developed through proposed guidelines (Section **3.1** to **3.7**). The traffic lights and elevator case studies, given in Chapter **6**, further elaborate the behavioral modeling concepts to generate SystemVerilog RTL code and Timed Automata model through MODEVES transformation engine.

## 4. Properties / Assertions Specification through SVOCL

In MODEVES project, we propose OCL temporal extension named SVOCL (SystemVerilog in Object Constraint Language) to represent SystemVerilog Assertions (SVA's) at higher abstraction level. We first identify leading property specification techniques (e.g. CCSL, OCL, SysML Parametric diagram) and provide the platform of SVOCL. Subsequently, we extend OCL to manage past and future temporal dimensions which are mandatory to represent SVA's. We are not including technical details of SVOCL and such details can be found at MODEVES website. We propose 9 SVOCL functions to represent corresponding SystemVerilog functions as shown in **Table I** below:

**Table I:** Summary of SVOCL functions to represent SystemVerilog Assertions at higher abstraction level

| Sr. # | SVOCL Functions                  | Equivalent SystemVerilog Constructs                 |
|-------|----------------------------------|---|
| 1.    | <i>SVSeq (s, d, p)</i>           | ## operator (Sequential delay)                      |
| 2.    | SVRep (p,r)                      | * operator (Consecutive repetition)                 |
| 3.    | <i>SVImplication (ant,con,t)</i> | overlapping  -> and non-overlapping  => implication |
| 4.    | SVStable (expr)                  | \$stable  |
| 5.    | SVChanged (expr)                 | \$changed   |
| 6.    | SVPast (expr, ct)                | \$past  |
| 7.    | SVRose (expr)                    | \$rose  |
| 8.    | SVFell (expr)                    | \$fell  |
| 9.    | Disif expression                 | disable iff   |

### 4.1 SVSeq (s, d, p)

In SystemVerilog, ## operator is frequently used to represent sequential delay. It can be represented as prec\_term delay\_seq seq\_term in its simplest form. For example, SystemVerilog statement X ##5 Y represents sequential delay on variable Y where X is prec\_term, ##5 represents delay\_seq and Y represents seq\_term. This statement shows that Y must hold 5 clock ticks later after X. We propose SVSeq function to represent the sequential delay as follows:

SVSeq (s, d, p)

Where s is seq\_term, d is delay\_seq and p is preceding term (prec\_term). The d and p are optional parameters. The default value of d is 1 (one clock delay). This function returns True if sequence is matched otherwise returns False.

**Examples:**

- SystemVerilog expression  $(a=b) \## 2 (c=d)$  can be represented as  $SVSeq ((c=d), 2, (a=b))$  in SVOCL
- SystemVerilog expression  $(a=b) \## [1:3] (c=d)$  can be represented as  $SVSeq ((c=d), 0103, (a=b))$  in SVOCL

**4.2 SVRep (p,r)**

SystemVerilog also provides consecutive repetition through  $*$  operator. For example, SystemVerilog statement  $a \## 2 b [*3]$  represents that  $b$  consecutively holds three times after two clock delays of  $a$ . Formally, it can be represented as  $a \## 2 b \## 1 b \## 1 b$ . In SVOCL, we provide the very basic semantics to represent the SystemVerilog consecutive repetition. We include SVRep function as follows:

SVRep (p,r)

Where  $p$  is the expression and  $r$  is the numeric value that represents number of repetitions

**Examples:**

- SystemVerilog expression  $(x+y) [*5]$  can be represented in SVOCL as  $SVRep ((x+y), 5)$

It is important to note that SVRep function represents very basic semantics which are not meaningful alone. Therefore, SVRep function is supposed to use along with SVSeq function to logically represent more complex and complete SystemVerilog repetition semantics

- For example, SystemVerilog statement  $a \## 2 b [*3]$  can be represented in SVOCL by using both SVRep and SVSeq functions as  $SVSeq (SVRep (b, 3), 2, a)$

**4.3 SVImplication (ant,con,t)**

There are two types of implication in SystemVerilog i.e. overlapping  $|->$  and non-overlapping  $|\Rightarrow$ . Formally, it is defined as:

$seq\_expr (antecedent) |-> property\_expr (consequent)$

$seq\_expr (antecedent) |\Rightarrow property\_expr (consequent)$

Sequence expression is given at the left side which is named as antecedent. Property expression is given at the right side which is named as consequent. Implication results in vacuous success if

antecedent fails. In case of each antecedent success, consequent is evaluated separately. Implication is successful (true) if consequent is evaluated as a true otherwise unsuccessful (false). In overlapping implication, consequent is evaluated on the same clock tick of successful antecedent. In non-overlapping implication, consequent is evaluated on next clock tick after successful antecedent. The result of implication can be true or false.

In SVOCL, we develop single function to represent the semantics of SystemVerilog both overlapping and non-overlapping implications. It takes three parameters as follows:

SVImplication (ant,con,t)

Where ant is antecedent expression, con is consequent expression and t is the type of implication. Antecedent and consequent expression can be developed by using different SVOCL functions like SVSeq, SVReq etc. The parameter t represents implication type i.e. true for overlapping and false for non-overlapping.

#### **Examples:**

- SystemVerilog implication  $(a=b) \rightarrow (c=d)$  can be represented in SVOCL as SVImplication ( (a=b), (c=d), 1)
- Similarly, more complex implication like  $x \#4 y \rightarrow z$  can be represented through SVOCL as SVImplication (SVSeq (y,4,x), z, 1)

#### **4.4 SVStable (expr)**

The SystemVerilog \$stable function is used to check whether the value of expression is changed between last time instance to current time instance. Formally, it is defined in SystemVerilog as \$stable (Expression, [clocking\_event]) where clocking event is an optional parameter. In SVOCL, we propose SVStable (expr) function to logically represent SystemVerilog \$stable function.

#### **Examples:**

- SystemVerilog \$stable(Y) (where Y is Boolean variable) can be represented in SVOCL as SVStable (Y)

#### **4.5 SVChanged (expr)**

SystemVerilog \$Changed function is contrary to \$stable function and given in SVOCL as:

SVChanged (expr)

**Examples:**

- SystemVerilog \$changed(X) (where X is Boolean variable) can be represented in SVOCL as SVChanged (X)

#### **4.6 SVPast (expr, ct)**

SystemVerilog \$past function is used to evaluate the value of expression at any past time instance. Formally, it is defined as \$past (expression, [number of ticks], [expression 2], [clockingevent]) where the last three parameters are optional. Number of ticks define the value of past time instance. If it is not given then 1 is used as a default past time instance. Expression 2 can be used as a gating expression. In SVOCL, we propose SVPast(expr,ct) function to logically represent \$past function. The function takes two parameters where expr represents expression and ct represents value of past time instance.

**Examples:**

- SystemVerilog \$past(X,2) (where X is Boolean type) can be represented in SVOCL as SVPast(X,2)
- Similarly, \$past(Y) can be represented in SVOCL as SVPast(Y) because default ct value is 1

#### **4.7 SVRose (expr)**

SystemVerilog \$rose and \$fell functions are used to evaluate the status of least significant bit with respect to last time occurrence. Logically, \$rose returns true if the value of expression is changed to true which was false in last time occurrence. In SVOCL, we propose SVRose(expr) to logically represent SystemVerilog \$rose function

**Examples:**

- SystemVerilog \$rose(gnt) (where gnt is Boolean type) can be represented in SVOCL as SVRose (gnt).

#### 4.8 SVFell (expr)

SystemVerilog \$fell returns true if the value of expression is changed to false which was true in last time occurrence. In SVOCL, we include SVFell(expr) to logically represent \$fell function.

#### Examples:

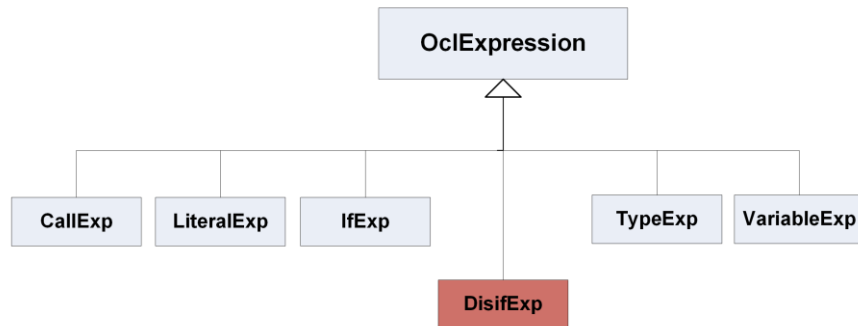
- SystemVerilog \$fell(gnt) can be represented in SVOCL as SVFell(gnt).

#### 4.9 Disif expression

SystemVerilog disable iff clause is frequently used to check a given condition like reset while executing a particular expression. Formally, it is defined as disable iff (conditional statement) property\_expr. If conditional statement is true then property expression (property\_expr) is not evaluated at all that results in a vacuous success. Further, if conditional statement becomes true while evaluating the property expression, then results of property expression are disabled with vacuous success.

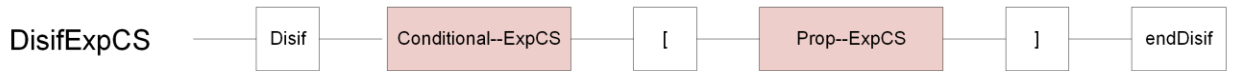
There is no direct mechanism, available in OCL, to represent the semantics of SystemVerilog disable iff clause. For example, OCL IfExp cannot be used to represent semantics of SystemVerilog disable iff. If conditional statement of disable iff clause is given as OCL if condition and property expression is given in OCL then clause then else clause is empty which is invalid in OCL. Furthermore, the results of property expression cannot be disabled in case the conditional statement is true during the evaluation of property expression. To summarize, OCL cannot directly represent the semantics of SystemVerilog disable iff clause.

To represent SystemVerilog disable iff clause, in SVOCL, we add new Disif expression by extending OclExpression as shown in **Figure 25**



**Figure 25:** Extend OclExpression for SVOCL DisifExp

The syntax of *Disif* expression is shown in **Figure 26**



**Figure 26:** Syntax of SVOCL DisifExp

### Examples:

SystemVerilog expression *disable iff (reset) (a=b) ##3 (c=d)* can be represented through SVOCL *Disif* expression as:

```

Disif (reset)
[
  SVSeq ((c=d), 3, (a=b))
]
endDisif

```

### 4.10 Property Name

To this point, we have discuss various SVOCL functions in order to model SVA's at higher abstraction level. These functions can be used to represent simple as well as complex SVA's. However, it is still required to include property / assertion name in model. To manage this, constraint name can be used to represent the corresponding property name.

### Examples:

Consider SystemVerilog Assertion code as follows:

```

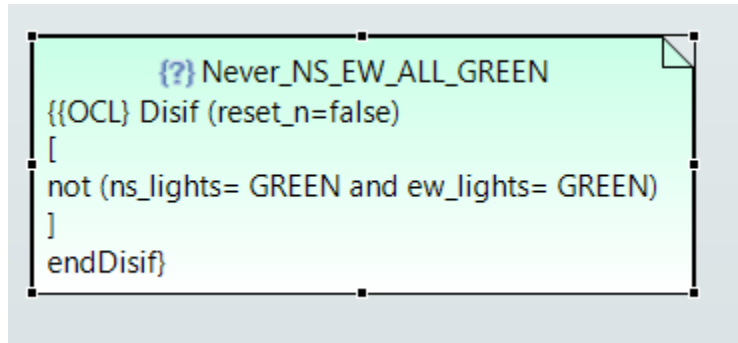
property Never_NS_EW_ALL_GREEN;

disable iff(reset_n=false)
not (ns_lights= GREEN && ew_lights= GREEN)

endproperty : Never_NS_EW_ALL_GREEN;
Never_NS_EW_ALL_GREEN_1 : assert property(@ (posedge clk)
Never_NS_EW_ALL_GREEN);

```

Here, assertion / property name is **Never\_NS\_EW\_ALL\_GREEN**. The logic of the property is marked as a **bold** text. This SystemVerilog assertion / property can be represented in SVOCL through Papyrus as shown in **Figure 27**



**Figure 27:** Example of SVOCL assertion in papyrus

It can be seen from figure 1 that property name is represented as a constraint name and logic is represented through SVOCL. The complete code can be generated through MODEVES transformation engine.

#### 4.11 Rules for Start and Close Brackets

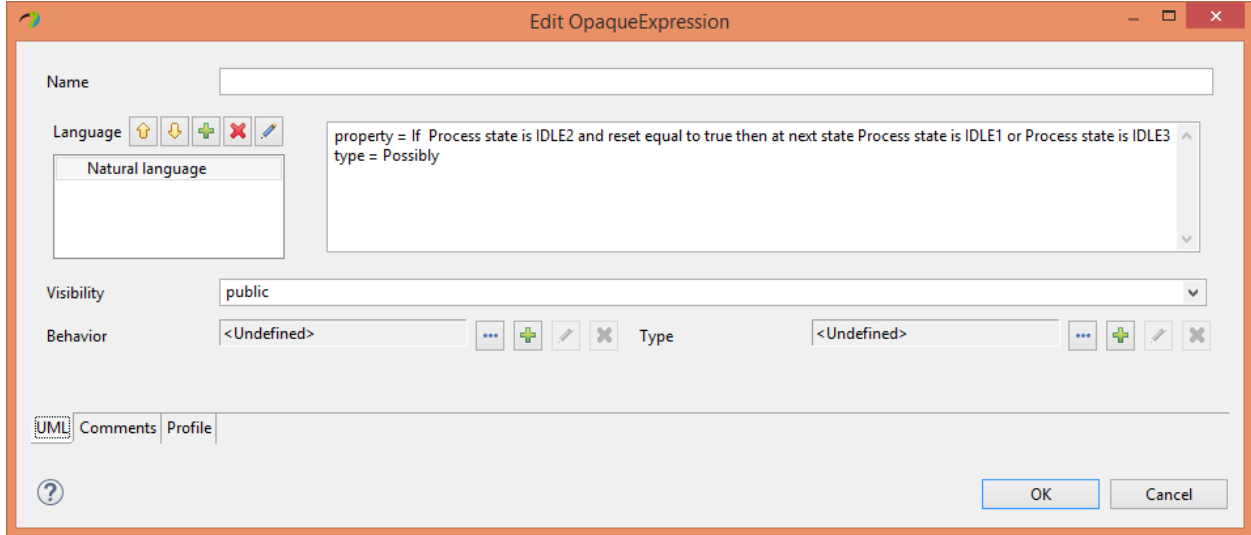
It is mandatory to start and close brackets properly otherwise SystemVerilog code will be generated with errors. Following rules are defined:

- Number of start brackets must be equal to number of close brackets. For example, statement  $(a+b)) + d$  is not allowed and it should be written as  $(a+b) + d$ .
- Brackets should be used as per requirements. For example, condition  $(a+b) + (c=d) * 3$  and condition  $((a+b) + (c+d))*3$  have different meanings. Consequently, MODEVES transformation engine will generate the code accordingly.
- An additional brackets are required when using **not** operator with different SVOCL functions. For example, consider SVOCL assertion as **not** (SVSeq ( ew\_lights=RED, 1 , ew\_lights=GREEN)). Here, you can see that additional brackets have been used because SVSeq function is called right after **not** operator.



## 5. Properties / Assertions Specification through NLCTL

NLCTL grammar comprises fourteen rules is developed to write both simple as well as complex assertions in model. Modeling editor like papyrus fully support the specification constraints in natural language. Therefore, it is straightforward to include assertions in models through NLCTL as shown in Figures below:



**Figure 28:** Assertion Example in NLCTL

Similarly, other assertions can be included in the model through NLCTL. The complete details of NLCTL rules with example can be found in the referring article.

## 6. Examples of traffic lights and elevator case studies

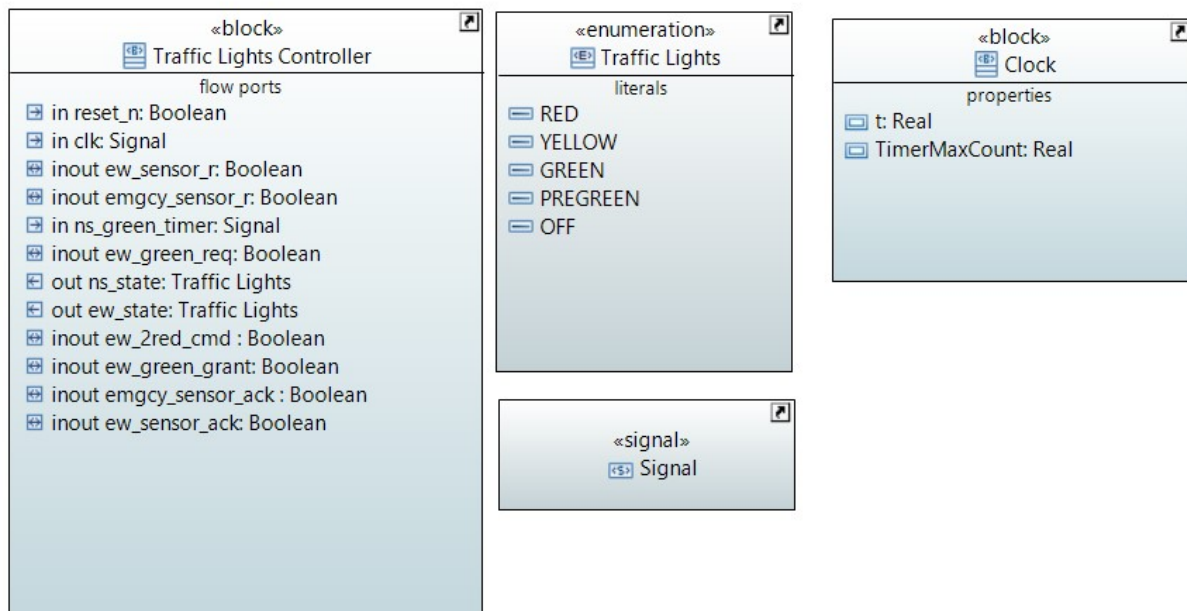
So far, we have presented the modeling methodology to capture structural aspects (Chapter 2), behavioral aspects (Chapter 3) and properties / assertions through SVOCL (Chapter 4) and NLCTL (Chapter 5). In this section, we demonstrate the application of modeling methodology through traffic lights and elevator case study. This facilitates end users to understand the application of our proposed modeling methodology in order to represent the structure, behavior and assertions at higher abstraction level. The demonstration of MODEVES transformation engine is also presented in order to generate SystemVerilog RTL, Timed Automata model, SVA's and CTL assertions code from the model.

## 6.1 Traffic Lights Case Study Description

This case study represents the design of traffic lights controller in order to manage North-South (NS) and East-West (EW) roads traffic as shown in fig. The NS is main road and potentially allowed green light most of the time. The sensor is attached at EW road to check the presence of vehicle. When EW sensor is activated and NS light is green for enough time then EW light is turned green to pass the traffic on EW road. Emergency sensor is also attached at the NS and EW intersection to manage the passage of emergency vehicles. Upon its activation, the lights of NS and EW roads become red for minimum three cycles to pass the emergency vehicle.

## 6.2 Modeling Structural Aspects

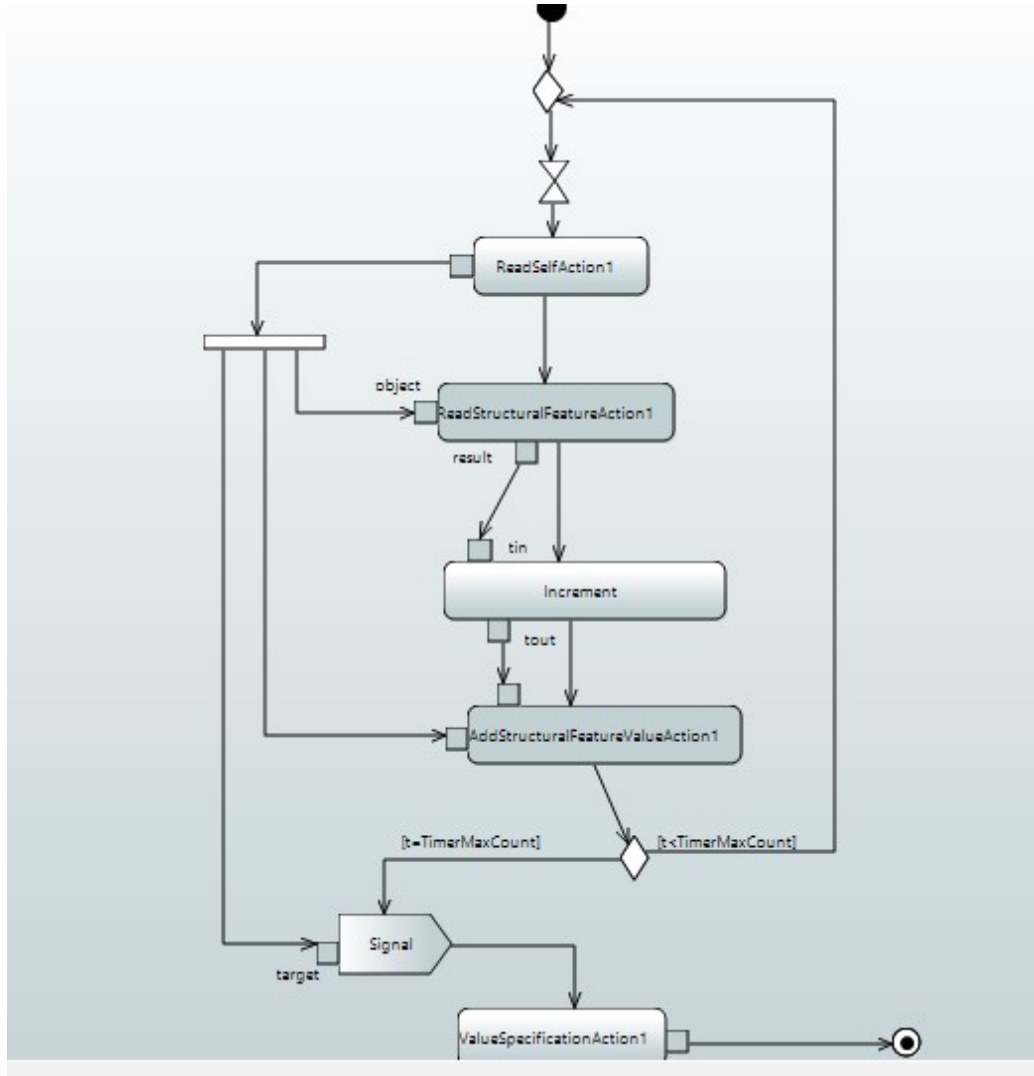
We represent structural aspects of traffic lights controller in SYSML Block Definition Diagram (BDD) as per defined guidelines (Chapter 2) as shown in **Figure 29**.



**Figure 29:** Modeling structural aspects of traffic lights controller in papyrus

The states of traffic lights have been represented through enumeration (Traffic Lights). Input and / or output registers have been represented through flow ports. Signal type has been used to manage clock and timer signals. A separate block has been developed to manage clock and timer. It contains timer activity that implements timer logic. It is required to use existing clock block to set the value timer. User only require to set the value of TimerMaxCount variable in order to define

the duration of timer. Here, our requirement is to reset timer after 3 seconds. Consequently, we set value of TimerMaxCount = 3. The activity diagram of timer, included in clock block, has been shown in **Figure 30**



**Figure 30:** Activity diagram of timer implemented in clock block

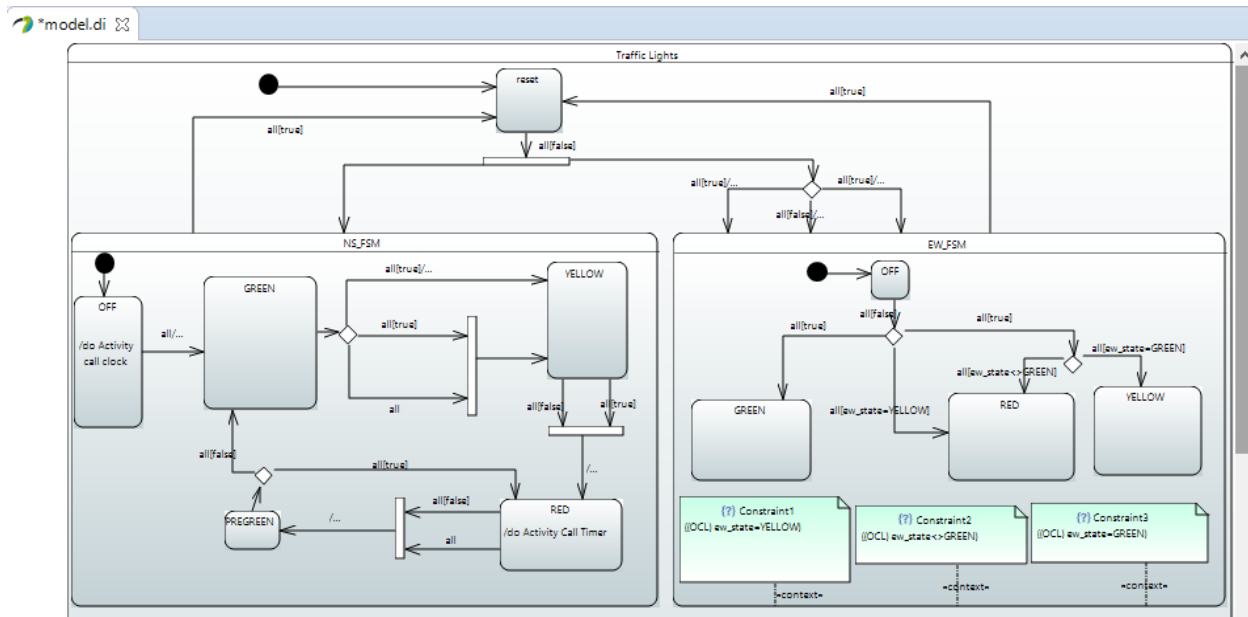
The timer and clock signals will be called in different states as per modeling requirements.

### 6.3 Modeling Behavioral aspects

State machine diagram has been used model behavioral aspects of traffic lights controller as per defined rules (Chapter 3). Following behavioral requirements of controller are represented at higher abstraction level (SMD):-

- NS light should stay green until one of the EW sensor is activated.
- If NS green timer is three and NS light is red, then NS light should turn to green.
- NS light should turn to yellow from green upon the activation of emergency sensor even if the value of NS timer is three.
- NS green timer should increment on every clock for max. count of three. However, NS green timer should reset to zero on controller reset, or upon NS yellow light. Similar is the case with EW green timer.
- EW light turn from red to pre green (to enable NS light to turn yellow) in case NS timer is three and EW sensor is activated.
- EW light should turn to green at next clock if it is currently pre green. Similarly, EW light should turn to red if it is currently yellow.
- EW light should turn to yellow from green if emergency sensor is activated or EW timer is reached to three.

The state machine diagram of traffic lights controller is shown in **Figure 31**



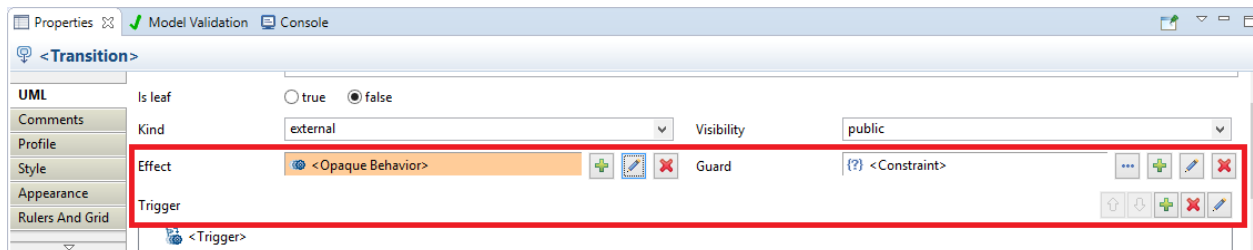
**Figure 31:** State machine diagram of traffic lights controller

It can be seen from the figure that state machine name is "Traffic Lights" which will be transform to module name of SystemVerilog RTL code and Timed Automata model. There are two FSM in

traffic lights controller design which are define on Region2 and Region3 respectively. Following state machine nodes have been used to capture behavioral aspects:

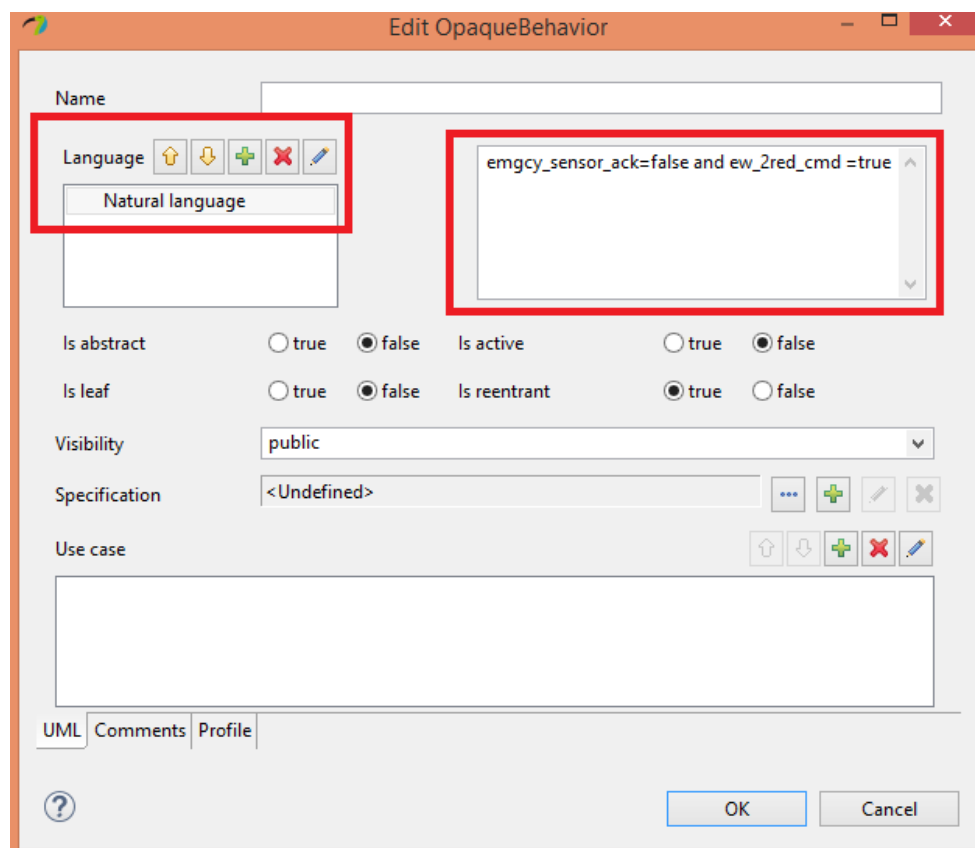
### 6.3.1 Transition

Three attributes of transition have been used to represent state navigation conditions i.e. Guard, Trigger and effect. The description and modeling guidelines of these attributes are already defined in section. Here, we present how we use them to model traffic lights controller as shown in **Figure 32**



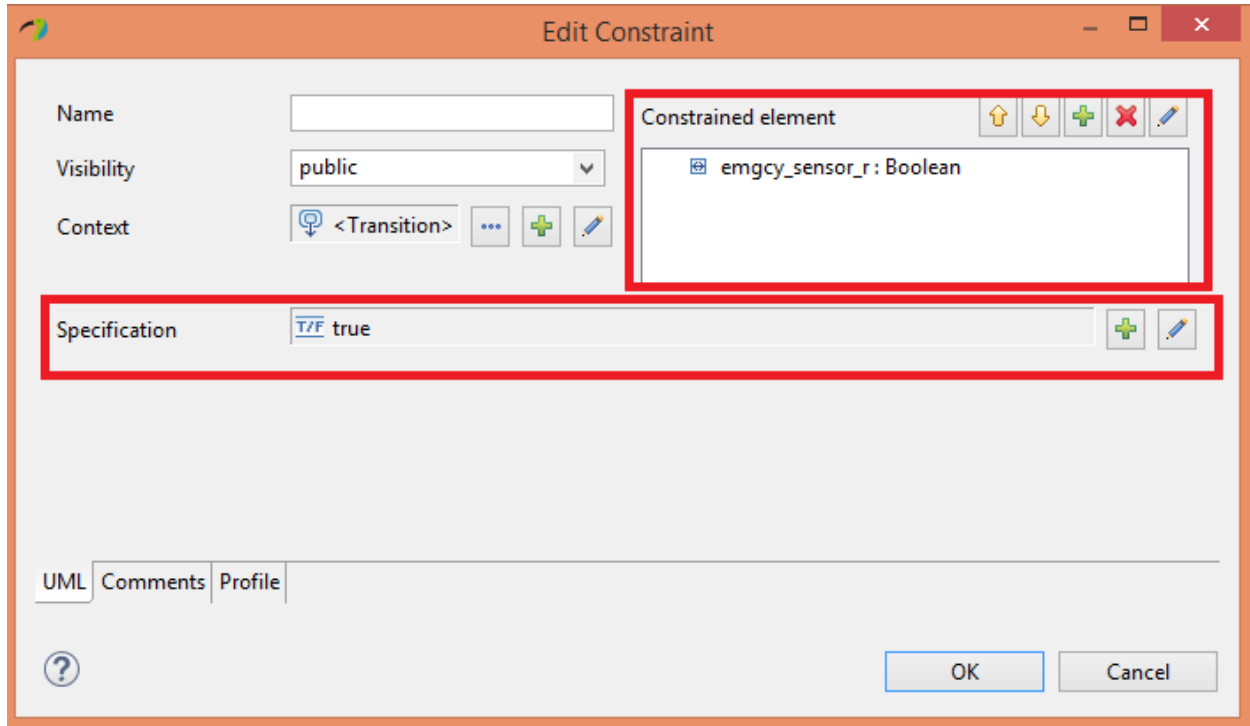
**Figure 32:** Example of specifying transition effect for traffic lights controller

To specify effect, new interface is opened to specify condition as shown **Figure 33**



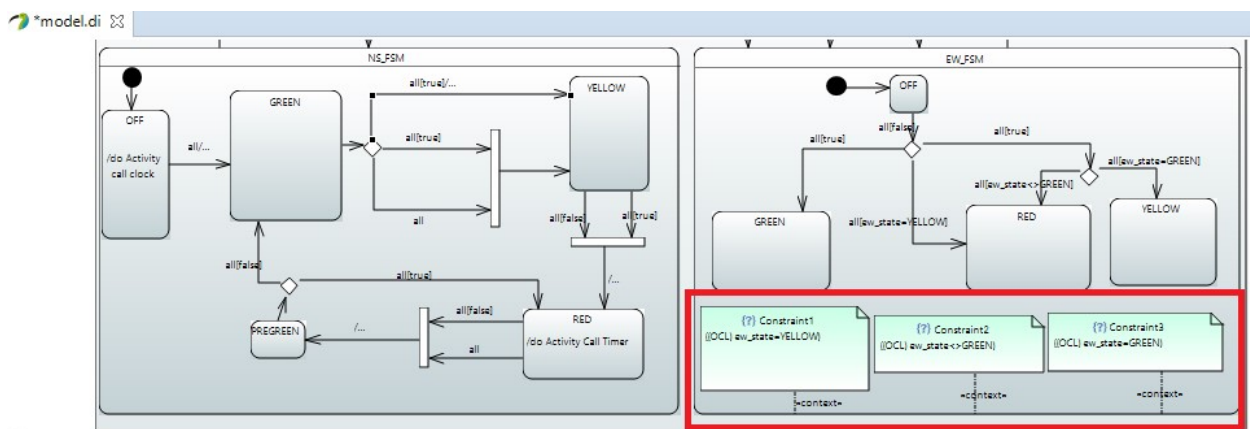
**Figure 33:** Example of specifying effect condition for traffic lights controller

The guard condition can be specified as shown in **Figure 34**



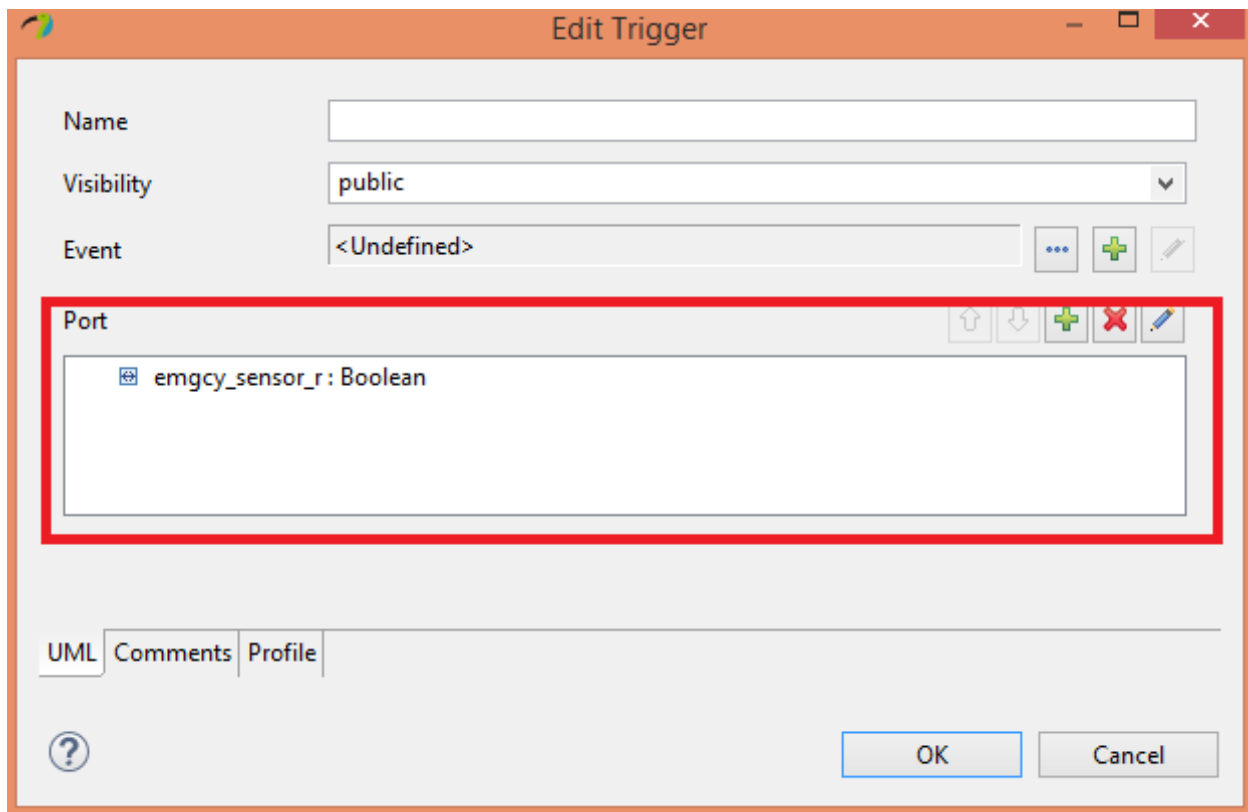
**Figure 34:** Example of specifying transition guard condition for traffic lights controller

It is also possible to specify Transition guard condition as constraint block. We use three constraint blocks to specify guard conditions of corresponding transitions as shown in **Figure 35**



**Figure 35:** Specifying guard condition through constraint block for traffic lights controller

Trigger port / ports are set as shown in **Figure 36**

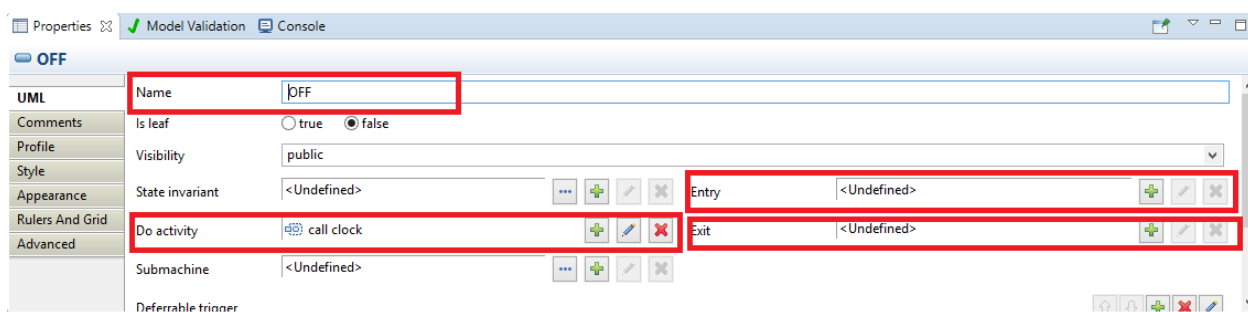


**Figure 36:** specifying trigger ports for traffic lights controller

The process presented here to specify the transition conditions (guard, trigger and effect) has been used for all transitions of state machine.

### 6.3.2 State

State machine state node has been used to specify different states of traffic lights controller. Four attributes (i.e. name, Do Activity, Entry and Exit) have been considered while modeling system behavior as shown in **Figure 37**



**Figure 37:** Example of state attributes (Name, Do Activity, Entry and Exit) for traffic lights controller

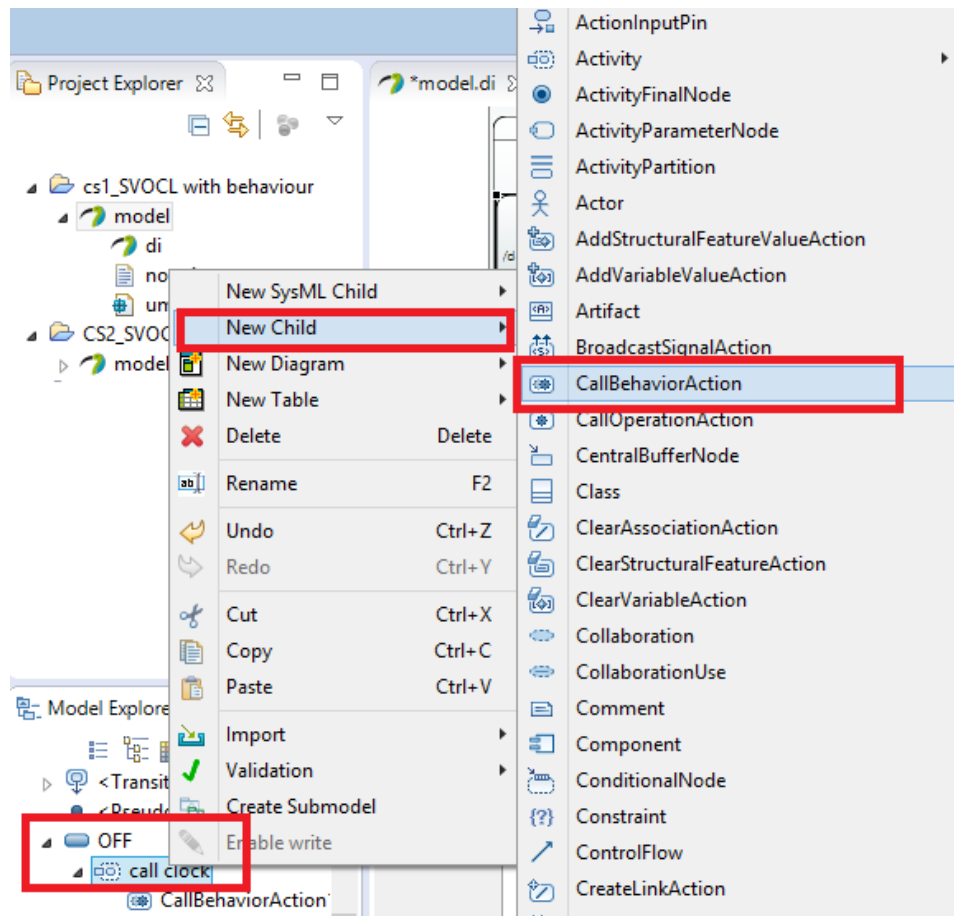
“Do activity” has been used to call clock / timer activity in particular state. Firstly, call clock is defined in Do activity as shown in **Figure 38**

The screenshot shows the 'Edit Activity' dialog box. The 'Name' field is set to 'call clock'. The 'Is abstract' and 'Is leaf' options are set to 'false', while 'Is reentrant' is set to 'true'. The 'Visibility' is set to 'public'. The 'Is active', 'Is read only', and 'Is single execution' options are all set to 'false'. The 'Specification' field is empty, showing '<Undefined>'. The 'Precondition', 'Postcondition', 'Owned parameter', and 'Variable' fields are also empty. The 'UML' tab is selected at the bottom.

**Figure 38:** Specifying activity name in state (Do Activity) for traffic lights controller

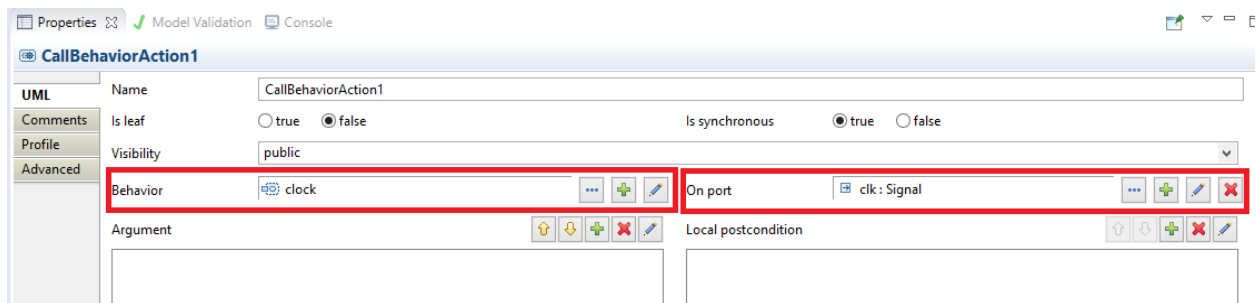
Secondly, CallBehaviorAction has been added in particular state against defined do activity (**Figure 38**) by using model explorer as shown in **Figure 39**





**Figure 39:** Adding CallBehaviorAction to call activity in state for traffic lights controller

Finally, behavior (timer or clock activity) and port (for signal) are defined as shown in figure



**Figure 40:** Setting CallBehaviorAction behavior and port for traffic lights controller

**This process will always be used whenever it is required to call clock / timer activity on particular state.**

### 6.3.3 Other state machine nodes

The important nodes like transition and state are already described above with screenshots. The other nodes like join, fork and choice have also been used as per requirements. However, usage and semantics of such nodes is very straightforward. Therefore, we are not including further details and screenshots of these nodes.

## 6.4 Modeling SystemVerilog Assertions

So far, we describe the structure and behavior of traffic lights controller. Now, we describe the utilization of SVOCL functions to represent SystemVerilog Assertions at higher abstraction level. From the behavioral aspects, we identify eleven assertions / constraints for design verification. The description of each assertion is given below:

### **Assertion 1:** Never\_NS\_EW\_ALL\_GREEN

**Description:** This property ensure that the status of NS and EW lights shouldn't be green concurrently.

### **Assertion 2:** nsLightAtReset

**Description:** This property ensure that the NS lights should be OFF at reset.

### **Assertion 3:** ewLightAtReset

**Description:** This property ensure that the EW lights should be OFF at reset.

### **Assertion 4:** NsNeverFromGreenToRed

**Description:** This is Safety property to ensure that NS lights status should never switch directly from green to red. There should be a Yellow light between this.

### **Assertion 5:** EwNeverFromGreenToRed

**Description:** This is Safety property to ensure that EW lights status should never switch directly from green to red. There should be a Yellow light between this.

### **Assertion 6:** NsGreenNext

**Description:** This property ensure that if NS light status is green and there is no emergency or ew sensor, then next sequence is also GREEN. This ensures that NS light get maximum green status as NS is the main road.

### **Assertion 7:** NsLightsWhenEmergency

**Description:** This property ensure the status change sequence of NS lights during emergency. Lights should switch from GREEN to YELLOW to RED.

### Assertion 8: EwLightsWhenEmergency

**Description:** This property ensure the status change sequence of EW lights during emergency. Lights should switch from GREEN to YELLOW to RED.

### Assertion 9: NeverGreenYellow

**Description:** This property ensure that NS/EW green and yellow lights should not be on simultaneously.

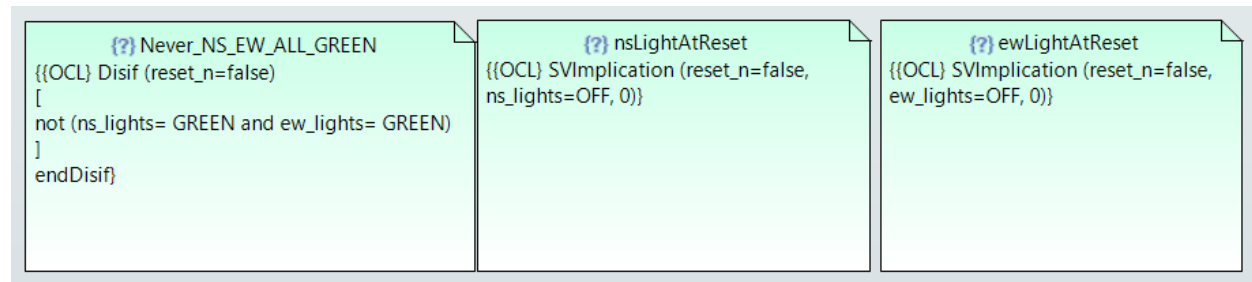
### Assertion 10: NsGreenForMin3Cyles

**Description:** The NS is main road. Its lights should remain GREEN for ns\_green\_timer == 3 before it can switch. The timer ns\_green\_timer should count to 3, and remain at 3 until light changes.

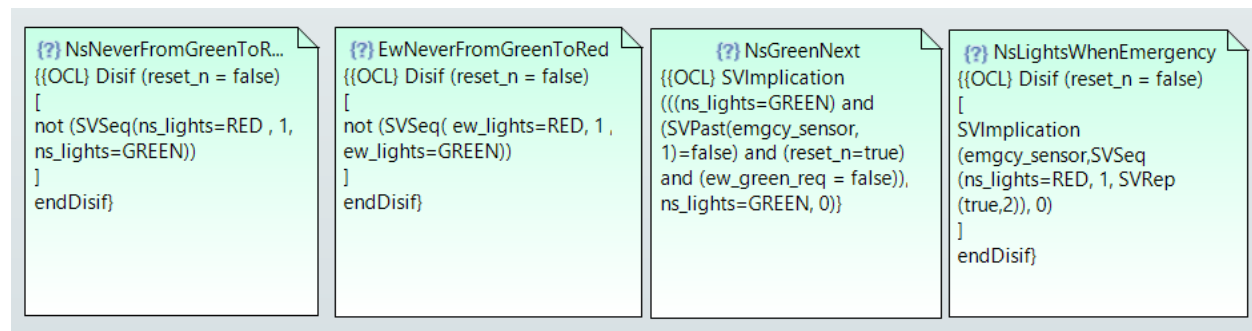
### Assertion 11: EwNewSensorActivation

**Description:** This property ensure NS and EW lights sequence on activation of EW sensor.

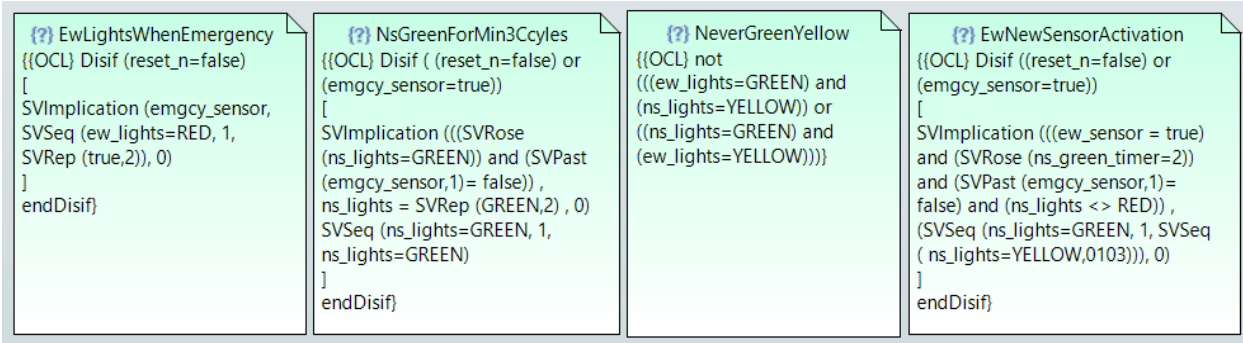
The above-mentioned SystemVerilog Assertions have been presented through SVOCL constructs as shown in **Figure 41**, **Figure 42** and **Figure 43** below:



**Figure 41:** Representing traffic lights controller assertions (1 to 3) in SVOCL



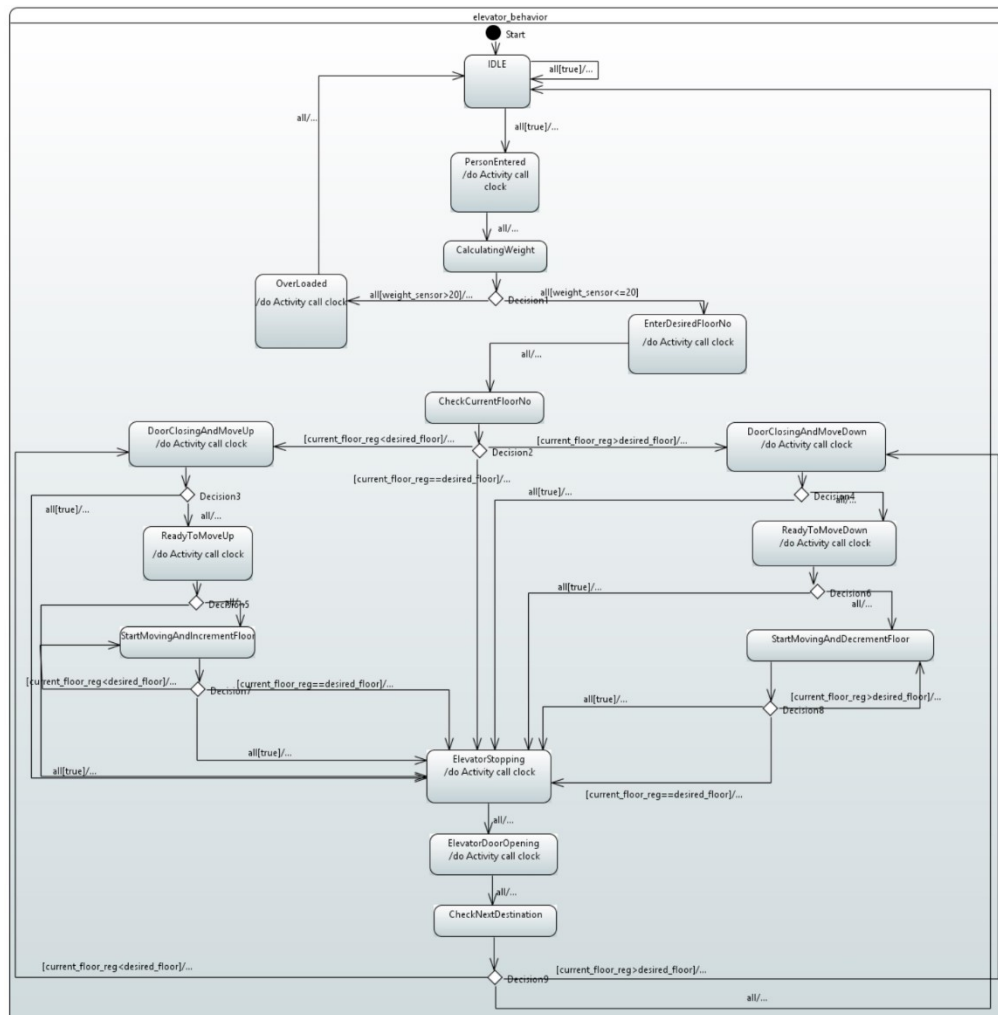
**Figure 42:** Representing traffic lights controller assertions (4 to 7) in SVOCL



**Figure 43:** Representing traffic lights controller assertions (8 to 11) in SVOCL

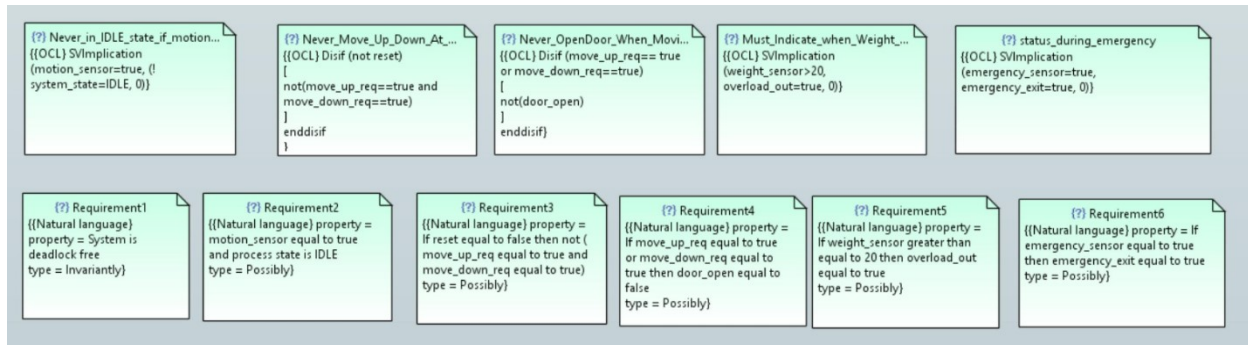
## 6.5 Elevator Case Study Description

This case study represents the design of elevator. Here, we represent verification aspects through both NLCTL and SVOCL. The design of elevator is shown in **Figure 44**



**Figure 44:** Design of Elevator

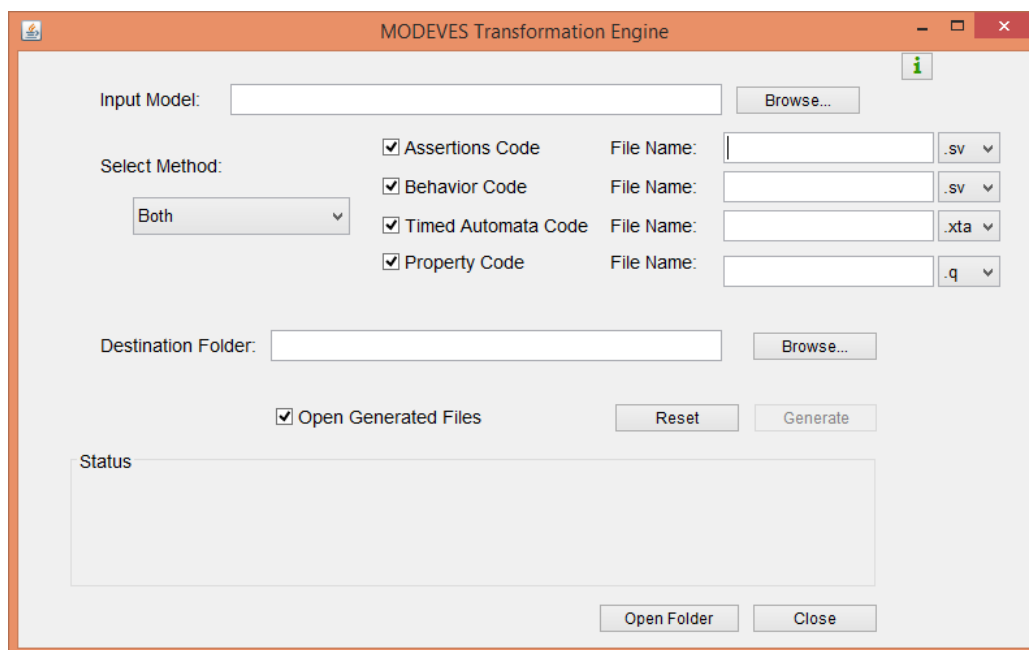
The verification aspects of elevator are included in the model through both NLCTL and SVOCL as shown in **Figure 45**



**Figure 45:** Verification aspects of elevator in NLCTL and SVOCL

## 6.6 Demo of MODEVES Transformation Engine

In this section, we transform the elevator design to SystemVerilog RTL, Timed Automata model, SVA's and CTL assertions code through MODEVES transformation engine. The main interface of MODEVES transformation engine is given below:



**Figure 46:** MODEVES transformation engine main interface

- **Input Model:** Browse button can be used to select the model that need to be transformed.
- **Output File Name:** It is used to specify the name of generated file. User can also select the type of generated file i.e. .sv, .c etc.

- **Destination Folder:** Browse button can be used to select the destination folder in which generated source code file is placed.
- **Open Generated File:** This checkbox can be selected if user want to open generated file.
- **Reset:** This button clears all current selections to define new configurations.
- **Generate:** This button will transform the selected model into SystemVerilog RTL, Timed Automata model, SVA's and CTL assertions code.
- **Status:** This displays the status of transformation i.e. Target files Generated Successfully or target files Generated with Errors (in case of transformation errors)
- **Open folder:** This button can be used to open the folder where output source code file has been generated.
- **Close:** This button will close the MODEVES transformation engine.