SUMO User Conference 2022
https://doi.org/10.52825/scp.v3i.114

Published: 29 Sept. 2022

High fidelity modelling of traffic light control with XML logic representation

Maik Halbach (1) and Jakob Erdmann (2),
(1) German Aerospace Center (DLR), Braunschweig, Germany
maik.halbach@dlr.de
(2) German Aerospace Center (DLR), Berlin, Germany
jakob.Erdmann@dlr.de

Abstract

The paper describes a novel approach to modelling traffic light control in SUMO with control logic configured in xml inputs. It shows also, that this approach allows for high fidelity replication of a real-world traffic controller by comparing original and simulated switching behavior when confronted with the same traffic situations. The approach could process from a traffic engineer with limited programming knowledge.

Contents

1 Motivation

Traffic lights are an important element of traffic simulation and their accurate representation is necessary to achieve good agreements between real world traffic and simulation traffic. Even when new control concepts are to be tested, it is necessary to represent existing systems for a fair assessment of gains.

The traffic simulation SUMO supports default traffic algorithms for fixed-timing control as well as adaptive traffic control based on detected time gaps. It also permits coupling with an external process for fine-grained control of traffic light switching via the TraCI API.[1] When using TraCI, it is possible to model the traffic light logic within different programming Languages such as Python, C++, Matlab, and Java[2]. Both native and coupled approaches have their advantages and disadvantages. In this paper we introduce a new configuration approach that has advantages over the previous approaches:

- High fidelity representation of control algorithms which differ from the default algorithms
- No need for process coupling (simpler and faster)
- Simplified and standardized algorithm description mirroring existing standards (compared to unconstrained TraCI code)
- Graphical analysis of operation and internal controller state with sumo-gui

Keywords: Simulation, traffic lights, adaptive control

2 Introduction

The paper describes a novel approach to modelling traffic light control in SUMO with control logic configured in xml inputs. We show that this approach allows for high fidelity replication of a real-world traffic controller by comparing original and simulated switching behavior when confronted with the same traffic situations.

In the following, we explain the syntax and semantics of the new configuration langue. For this purpose, an exemplary control algorithm is introduced and presented in the typical format used by traffic engineers in Germany. We then show how to translate this algorithm into an xml configuration usable by SUMO.

The figures C, 4, 7, 9, 10, 12, 15 represent a traffic light document (TLD). The translation of the TLD elements into a SUMO XML configuration are shown in figures

A, B, 1, 2, 3, 5, 6, 8, 11, 13, 14, 16, 18. The process of translating the TLD into the corresponding XML configuration is described in the following chapters.

2.1 Enhancements of the SUMO Software

To enable a high-fidelity modelling of traffic light control with xml logic representation, we implemented the new features in SUMO listed below. Further information about these features beyond the information in this paper, can be found in the SUMO documentation.[3]

- Coordination of actuated traffic lights
- Type 'actuated' with custom switching rules
- Overriding phase attributes with expressions
- Storing and modifying custom controller variables at runtime
- Extended signal plan visualization

In the researching project SAVeNoW[4] we used all listed functions to model the logic of a real traffic light controller in a SUMO xml-file. This paper uses an exemplary TLD which shows the most important controller design elements for a simplified intersection. The example controller is not completely efficient and realistic but it includes most of the functions typically needed for traffic light control modelling (phase logic and public transport prioritization).

2.2 Structure of this paper

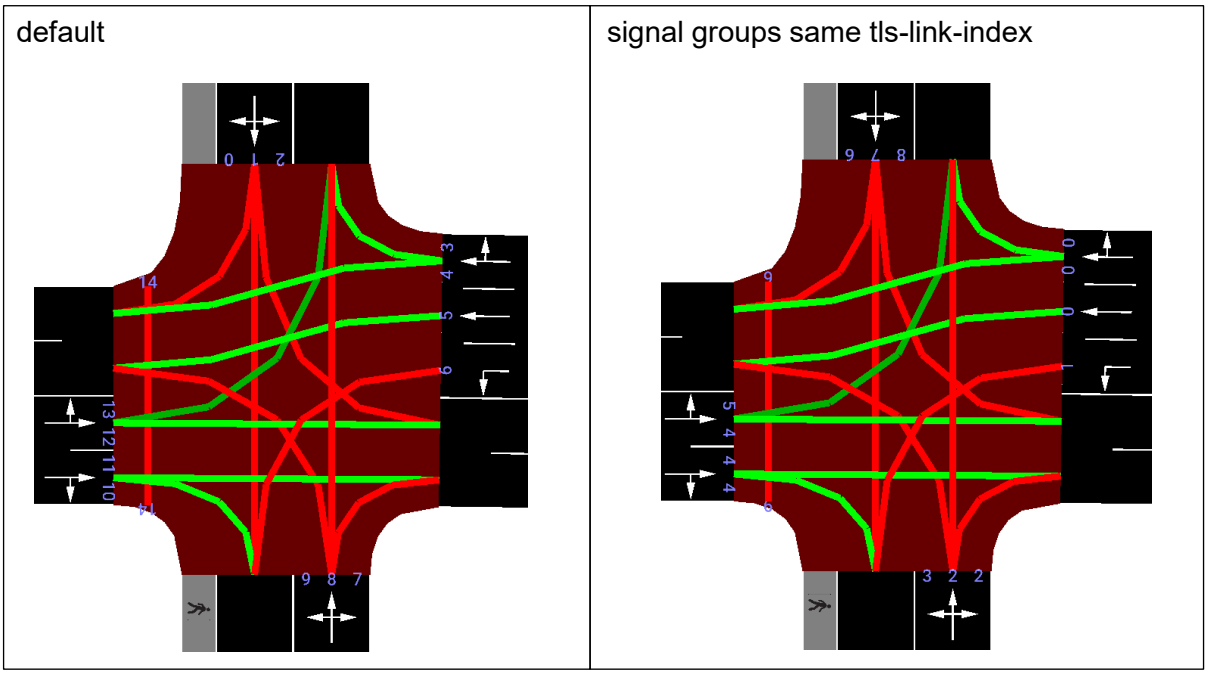
First of all, we describe the SUMO Net preparation and the preparation of the additional xml files for the detectors and the traffic light. Than we describe basic traffic light configurations. We follow by describing the modelling of traffic light logics in the traffic light xml file. First the definition of constants/parameters, then how to identify important logic blocs for the basic timing attributes of sumo and also the basic function of every expressions. Afterwards we describe the definition in detail and also discuss alternative definition styles.

3 XML modelling description

3.1 SUMO Net preparation

First of all, the SUMO Net and traffic light of the examined intersection has to be created. By default, every lane-to-lane connection is assigned its own “tls link index” to define its position within the signal state description string. To remove redundancy in the phase description and visualization, we define signal groups by assigning the same tls-link-index to all connections with the same signal behavior (in this, permissive green and protected green are counted as different signal behavior). (Figure 1, example link0, link2, link4; right frame), besides/apart from conditionally compatible the signal state description changes, so the link number has to be count up. By default, SUMO counts the link number clockwise. It is also possible to change the order of the link indices. An example can be seen in Figure 1.

Figure 1: SUMO intersection tls-link-index



The traffic light program in SUMO is defined as a list of phases numbered from 0 to n. We distinguish these phases by calling them “stages” and “interstages” depending on their role during the operation of the traffic light. Stages permit vehicle movements and their length may be adjusted in response to traffic conditions. The state of every signal stays constant while a stage is active. Interstages form the transition between stages and must ensure that traffic flowing in one stage has left the intersection space before a new stage begins. So, the states of signals could change from one interstage phase to the next.

In the following program we assign to each sumo phase a name according to their stage number or the interstage (stage-to-stage transition) it belongs to (i.e. 1,2,3, 1.2, 2.3, 3.1). (TLD: Figure C, SUMO-xml: Figure A, Figure B) These names serve as a visual aid when following the sequence of phases in the tracking diagram (chapter: 3.6 Tracking Phases).

A signal plan can be generated for SUMO with the aid of an OCIT-file and the tool ocit2SUMO[5]. A signal plan may also be created visually in netedit (i.e. when the traffic light specification is only available as a pdf document).

An additional preparation for the SUMO simulation is the definition of the detectors (induction loops). They are defined with the same name and also the same position (Figure 2, Figure 3) at each lane according to the TLD signal location plan (Figure 4).

Figure 2: detector definition – SUMO det.add xml

```
1   <additional>
2       <inductionLoop id="DA1" lane="-E3_0" pos="-30" freq="30" file="cross.out"/>
3       <inductionLoop id="DA2" lane="-E3_1" pos="-30" freq="30" file="cross.out"/>
4       <inductionLoop id="DA3L" lane="-E2_0" pos="-10" freq="30" file="cross.out"/>
5       <inductionLoop id="RPD" lane="-E3_0" pos="-50"      […]       vTypes="bus"/>
6       <inductionLoop id="RPE" lane="E4_0"  pos="10"       […]       vTypes="bus"/>
7   </additional>
```

Figure 3: net - SUMO GUI

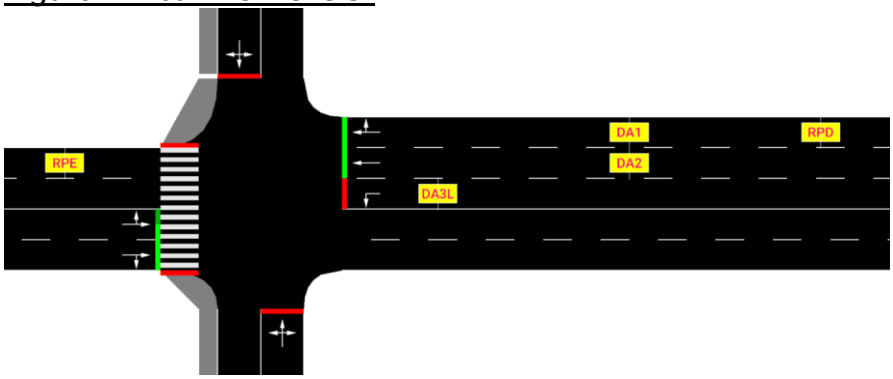
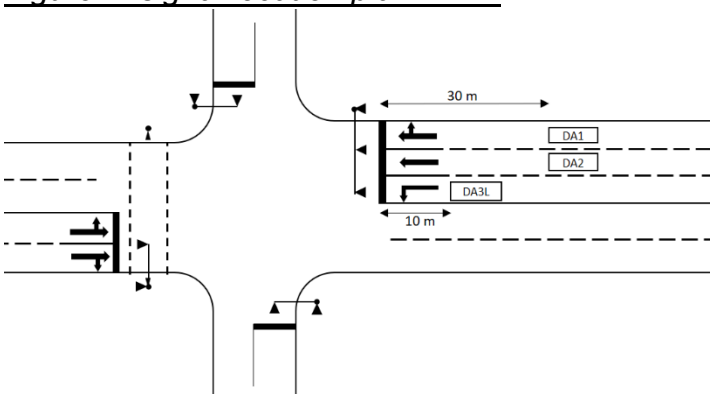


Figure 4: signal location plan - TLD



For the public transport prioritization, mostly “telegrams” (sent upon passing a specific location) are used. Here it is possible to use induction loops in sumo that are placed at the corresponding location and which are only triggered by specific vehicle types (i.e. bus) (Figure 2, line 5,6). All detectors are defined in an additional file. (Figure 2)

3.2 Basic traffic light configurations

The next step is to declare the traffic light type of the example intersection as “actuated” to activate the core functionality of switching in response to traffic detectors. (Figure 5, line 2; type = “actuated”). The parameter coordinated has to be set to “true” in order to align the cycle second counter of the traffic light with the simulation time (Figure 5, line 3). In coordinated mode, the cycle-second time range configured by the attributes *earliestEnd* and *latestEnd* will be aligned with all other traffic lights of the same *cycleTime*. The *cycleTime* attribute (Figure 5, line 4) denotes the duration of one switching cycle. The *offset* attribute (Figure 5, line 2) defines where within the cycle, the signal program will start, effectively shifting the initial cycle second.

An optional next step is to assign the detectors related to the signal plan and the intersection (Figure 5, line 6-8). This permits using custom detector placement instead of detectors that would otherwise be generated in default locations on each incoming lane during initialization. To define a custom detector, a lane that is incoming to the traffic light is used as the key and the id of a custom detector is used as the value. Both custom and automatic detectors can later be observed when tracking operations visually (chapter: 3.6 Tracking Phases)

Figure 5: basic traffic light configurations – SUMO xml

```
2 <tlLogic id="J3" type="actuated" programID="1" offset="-5">
3      <param key="coordinated" value="true"/>
4      <param key="cycleTime" value="90"/>
5
6      <param value="DA1" key="-E3_0" />
7      <param value="DA2" key="-E3_1" />
8      <param value="DA3L" key="-E3_2" />
```

(abstract of Figure A)

3.3 Define constants/parameters out of the TLD

The parameter setting out of the TLD are defined in conditions with fixed values in the SUMO XML (Figure 6). This corresponds directly to a table of values as in TLD (Figure 7). This simplifies re-use of an existing algorithm by only modifying its parameters to create another signal plan with different parameters for a different time of the day.

Figure 6: Table of algorithm configuration constants – SUMO xml

```
14     minal/maximal times
15     <condition id="min_Stage_1"      value="10"/>
16     <condition id="max_Stage_1"      value="60"/>
17     <condition id="tgrmin_FVA"       value="35"/>
18     <condition id="tgrmax_FVA"       value="32"/>
19     <condition id="max_pedestrian"   value="75"/>
20
21     constants
22     <condition id="k1" value="1"/>
23     <condition id="k2" value="2"/>
24
25     time conditions
26     <condition id="t01"  value="30"/>
27     <condition id="t02"  value="65"/>
28     <condition id="tb01" value="45"/>
29     <condition id="tb02" value="80"/>
```

(abstract of Figure A)

Figure 7: Table of algorithm configuration constants – TLD

minal/maximal times

variable	P1
min_Stage_1	10
max_Stage_1	60
tgrmin_FVA	35
tgrmax_FVA	32
max_pedestrian	75

constants

constants	P1
k1	1
k2	2

time conditions

time conditions	P1
t01	30
t02	65
tb01	45
tb02	80

3.4 Define logical condition

The logical condition out of the TLD (Figure 9) are defined in conditions (Figure 8).

Figure 8: logical conditions – SUMO xml

```
59      <condition id="l1" value="(z:DA1 >= 3) or (z:DA2 >= 3)"/>
60      <condition id="An" value="z:RPD + k1"/>
61      <condition id="Ab" value="z:RPE + k2"/>
62      <condition id="l2" value="a:DA3L"/>
```

(abstract of Figure A)

Figure 9: logical conditions – TLD

name	Logical condition	comment
l1	(ZD(DA1) >= 3) or (ZD(DA2) >= 3)	FVA
An	ZD(RPD) + k1	Bus
Ab	ZD(RPE) + k2	Bus
L2	Demand(DA3L)	FVAL

All logic conditions here used functions to retrieve information from detectors previously assigned to the traffic light controller, but it is also possible to retrieve information from every detector loaded into the simulation. The function z:DETID retrieves the time (in seconds) since the last vehicle detection at the detector with id DETID. The function a:DETID returns a value of 1 if the given detector is occupied and 0 otherwise.

3.4 Phase logic modelling

We show two possible ways to model a phase logic.

	TLD	SUMO-XML modelling
Constant time bounds	Figure10	Figure A
Variable time bounds	Figure13	Figure B

The approach of the first example can be used when all attributes that describe the time boundaries of a phase *minDur*, *maxDur*, *earliestEnd* and *latestEnd* have fixed values.

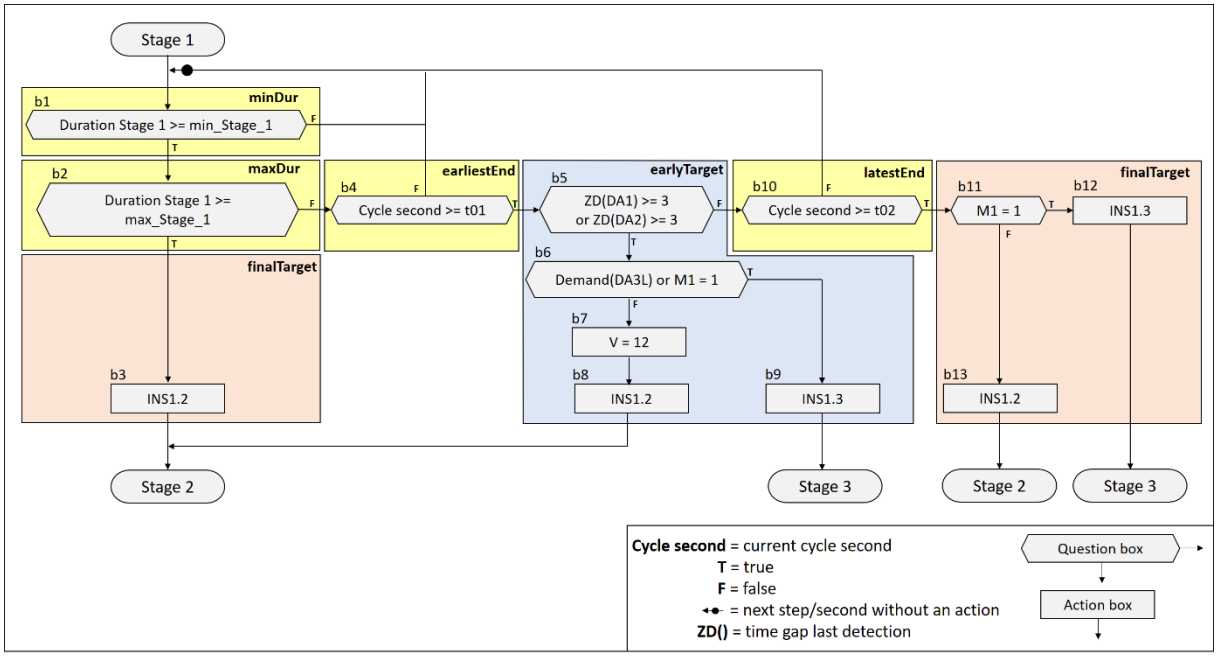
```
52 <phase […] minDur="0" maxDur="40" earliestEnd="70" latestEnd="84"/>
(abstract of Figure A)
```

The advantage of this approach is the brevity of the XML definition. However, some TLD descriptions may be too complex to use constant values to describe time boundaries or it may be too hard to restructure the flow diagrams of the TLD and extract these constants.
In this case it can be simpler to "blindly" transcribe all logic elements from the TLD and forgo the "simpler" XML definition. Hence, we also describe a second approach where the attributes *minDur*, *maxDur*, *earliestEnd* and *latestEnd* are replaced by complex conditions. This more general approach can be used to transcribe any TLD logic but the resulting XML configuration is lengthier and somewhat harder to understand. Good names for the employed conditions help to keep the descriptions readable and aids in debugging with the phase tracking dialog.

3.4.1 Example with constant time bounds

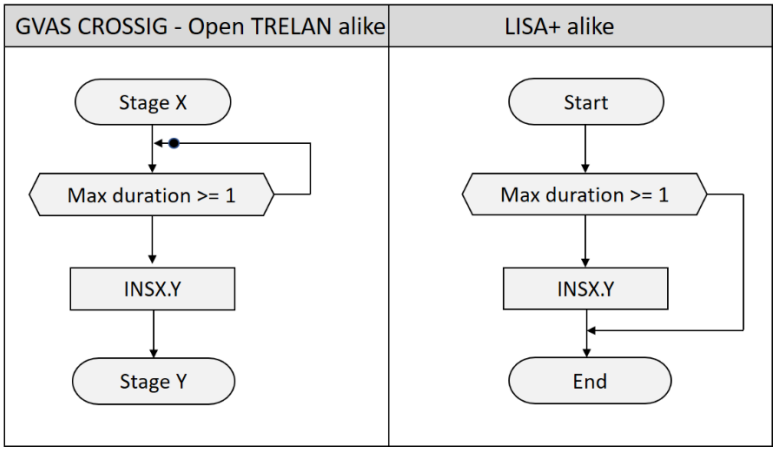
Figure 10 shows the phase logic of the first example as expressed in the TLD.

Figure 10: Example 1 - TLD logic



All shown TLD phase logics in this paper are similar to the CROSSIC Software notation (Open TRELAN). For readers familiar with the LISA+ notation, we show the main difference in flow diagram style in Figure 11.

Figure 11: Phase x logic example



3.4.1.1 Sumo phase definition

Each phase (stage or interstage) can either be of fixed or variable duration. Whereas fixed phases may be fully described by the *duration* attribute, the *attributes minDur, maxDur, earliestEnd, latestEnd* are used to describe the time bounds for variable-length phase. The attributes *next, earlyTarget, finalTarget* are used to define successor relationships among phase and the conditions for switching between them. All these attributes will be described in the following.

3.4.1.2 Phase logic modelling

The switching time for a phase of variable length may be restrained by upper and lower bounds with regard to its running duration and also with regard to a time window for coordination. For this purpose, the 'phase' element provides the following attributes:

- **minDur**: minimum running duration (mandatory)
- **maxDur**: maximum running duration (optional)
- **earliestEnd**: earliest time within a cycle for ending phase (optional)
- **latestEnd**: latest end within a cycle for ending phase (optional)

These attributes correspond to standard control parameters in a typical TLD and in our example they take on the values of b1, b2, b4 and b10 (Figure 10).

The possible reachable interstages are here in the blocs b3, b8, b9, b12 and b13 (Figure 10). These blocs represent which interstages are used to skip in another phase and also define in which following phase the phase0/stage1 is able to switch. Attribute *next* defines in which possible following phases the phase0/stage1 could switch. *EarlyTarget* defines conditions which are checked upon entering the lower time bounds of *minDur* and *earliestEnd*. *FinalTarget* defines conditions which are checked when reaching the upper time bounds of *maxDur* or *latestEnd*. In Figure 10, the yellow blocks (b1, b2, b4, b10) define these time bounds. The remaining blocks from the *earlyTarget*

area *(blue)* and *finalTarget* area *(red)* in Figure 10 we have to define for possible paths through the logic. So, every question bloc in the *earliestTarget* area (Figure 10, blue) and *finalTarget* area (Figure 10, red) are defined as a condition with the same name like the bloc named in TLD Logic. (Figure 10) These conditions are necessary for later modelling. For later defining of b10 (in chapter 3.4.1.6) we have also to model the blocs b1, b2, b4, b10. (Figure 12) The function c: access the current cycle second of the operating signal plan.

Figure 12: Example 1 bloc conditions – SUMO xml

```
70    <condition id="b5" value="11"/>
71    <condition id="b6" value="12"/>
72    <condition id="b11" value="M1 = 1"/>
73
74     <condition id="b1" value="min_Stage_1 >= c:"/>
75     <condition id="b2" value="max_Stage_1 >= c:"/>
76     <condition id="b4" value="t01 >= c:"/>
77     <condition id="b10" value="t02 >= c:"/>
```

(abstract of Figure A)

For *earlyTarget* the starting point is in general *minDur* and/or *earliestEnd***.** In our example it is *earliestEnd***,** so b4. In the example *earlyTarget* is checked when *minDur* and *earliestEnd* is reached. For *finalTarget* the starting point is in general *maxDur* or *latestEnd***.** In our example it is *maxDur* or *latestEnd* which has to be reached**,** so b2 or b10. Now every frame condition is described in general. In the following the detailed modelling of each frame condition will be described.

3.4.1.3 MinDur maxDur

The first values are the *minDur* and *maxDur*. In SUMO we could implement this in two ways. With a fixed value

```
<phase […] minDur="10"[…] maxDur="60"[…]/>
```

or we override the attributes with an expression

```
32    <phase […] minDur="-1" maxDur="-1"[…]/>

64    <condition id="minDur:0" value="min_Stage_1"/>
65    <condition id="maxDur:0" value="max_Stage_1"/>
```

(abstract of Figure A)

Even if the minimum duration of a phase is constant, it may be advantageous to override the phase attribute in order to collect all configuration parameters in a single place within the XML configuration. This makes it easier to copy and re-use a configuration for another controller program.

3.4.1.4 EarliestEnd latestEnd

For the *earliestEnd* and *latestEnd* the same override approach as described for *minDur* and *maxDur* may be used. But also, a fixed value defined directly in the phase definition works.

```
32      <phase […] earliestEnd="-1" latestEnd="-1" […]/>

67      <condition id="earliestEnd:0"     value="t01"/>
68      <condition id="latestEnd:0"       value="t02"/>
```
(abstract of Figure A)

EarliestEnd and *latestEnd* will be always compare with currently cycle second.

3.4.1.5 Next

Attribute *Next* defines possible successor phases for a given phase. It is typically used in stages to declare the first phase for each possible interstage sequences that target a successor stage. However, in some controllers it is also possible to branch of into different interstages from within an interstage. It is possible to switch when any of the basic conditions ((*minDur* and/or *earliestEnd*) or *maxDur* or *latestEnd*) is reached. In the example we defined in *next* with phase indices 1 and 6 because these are the interstages/phases which transition after a switch to the stage 2 and stage 3.

```
32      <phase name="stage1" […] next="1 6"/>
```
(abstract of Figure A)

These transitions are shown in the stage-sequence-diagram (Figure C). If a phase does not use attribute *next*, the signal plan switches to the subsequent phase in the phase list after reaching max duration (with a wraparound to 0 at the end). To define a signal plan with a single phase that is permanent green phase one could write the index of the phase itself in the *next* attribute or specify only a *minDur* but no *maxDur*.

3.4.1.6 EarlyTarget and finalTarget

In our controller there are two ways to switch out of the phase0/stage1. The first way is by adapting to traffic measurement (elapsed time since detection). Effectively, the phase is ended when a defined condition evaluates to ‘true’ (or non-0). In the example we could exit the phase earlier as soon as *minDur* and *erliestEnd* are reached, then Sumo will check by order of the values of *next* the listed phases. The attributes *earliestEnd* check the including conditions of the mentioned phases. If a condition is true the signal plan switches in this phase. If none of the conditions are true, the signal plan remain in the phase 0/stage1. For *earliestEnd* here are two possible paths.

path b5-b6-b9
```
34      <phase […] name="INS1.2" earlyTarget="b5 and !b6 " […]/>
```
(abstract of Figure A)

and path b5-b6(f)-b7-b8
```
42      <phase […] name="INS1.3" earlyTarget="b5 and b6" […]/>
```
(abstract of Figure A)

We have only to model the questions blocks from the paths because they will change something on the operating decision. Here the logic conditions (Figure 8) and the conditions about the blocs (Figure 12) are used to define the paths. The possible paths must be written in the attribute *earliestEnd* for each phase which is possible to reach for the signal program (INS1.2/phase 1 and INS1.3/phase 6), as shown above.
The path b5-b6(f)-b7-b8 has a special bloc b7 (Figure 10), in this bloc a variable is be written. It is not possible do it directly in the path. But when the path b5-b6(f)-b7-b8 is reached it is a unique path within the how logic, so we define an assignment to modelling that. If all blocs of the path get the right state, V will be assigned with the integer 12 and the path b5-b6(f)-b7-b8 is modelled completely.

```
79      <condition id="V" value="0"/>
80
81      <assignment id="V" check="b1 and b2 and b4 and b5 and !b6" value="12"/>
```
(abstract of Figure A)

It is also possible to model it when there are more than two paths for the *earliestTarget*.

When defining *latestEnd = cycle time - duration_interstage from the last phase*, then each cycle will last for the give cycle duration.

The second diagram path for leaving the state is via the *finalTarget* bloc. It is used when *maxDur* or *latestEnd* are reached. Here we have two possible pathways. The path begins at the starting element *maxDur* b2 and *latestEnd* b10. Sumo will check by order of the values of *next* Phase the attributes *finalTarget* and check the including conditions of the mentioned phases. If a condition is true the signal program switch to this phase. Here we have to define two paths to the interstage1.2:

path 1: b2-b3 path 2: b10-b11(f)-b13

```
34      <phase […] name="INS1.2" finalTarget="b2 or !b11"/>
```
(abstract of Figure A)

and one path to the Interstage1.3:

path: b10-b11-b12

```
42      <phase […] name="INS1.3" finalTarget="b10 and b11"/>
```
(abstract of Figure A)

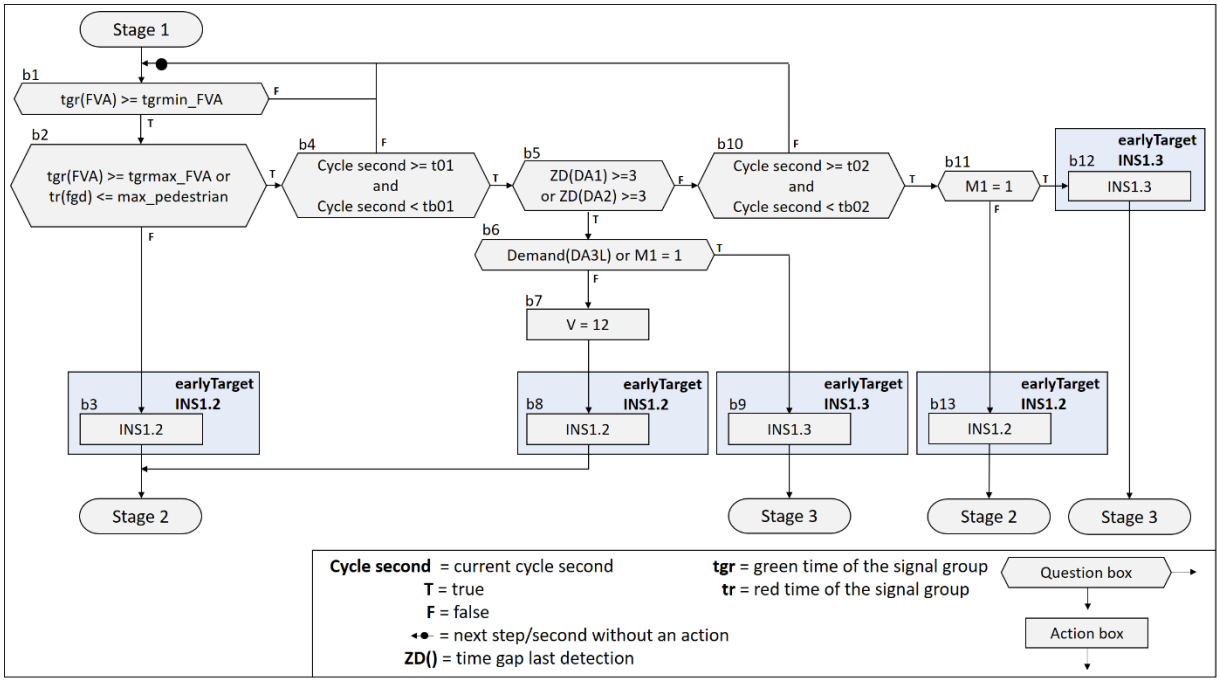
Here we also only model the question blocs for the same reason as mentioned. If none of the condition evaluates to true, SUMO switches to the last value of the next attribute as a fallback. If next is not defined, it switches to the next phase in definition order. In a typical TLD this should not occur and at least one of the path conditions should evaluate to true.

With this, every possible path in the logic is modeled and the SUMO controller can determine the switching conditions as described in the TLD.

3.4.2 Example with variable time bounds

For the second example we transcribe a TLD that does not express the time boundaries *minDur* (b1), *maxDur* (b2), *earliestEnd* (b4) and *latestEnd* (b10) with numerical constants (Figure 13). This means the XML definition cannot use attributes *minDur*, *maxDur*, *earliestEnd* and *latestEnd* as shown in the first example. In the following text we describe how to create the appropriate XML definitions.

Figure 13: Example 2 - TLD logic



First of all, we define a condition for every question bloc with the same bloc name as in the TLD logic. (Figure 14)
Here we used two SUMO function which we want to describe shortly. Function **g:** accesses the current running green time of a link. function **r:** accesses the current running red time of a link. This could be used for example to define that a maximum waiting time for pedestrians is not exceeded.

Figure 14: Example 2 bloc conditions – SUMO xml

```
62 <condition id="b1" value="(g:0 >= tgrmin_FVA)"/>
63 <condition id="b2" value="(g:0 >= tgrmax_FVA) or (max_pedestrian >= r:9)"/>
64 <condition id="b4" value="(c: >= t01 or tb01 > c:)"/>
65 <condition id="b5" value="11"/>
66 <condition id="b6" value="(12 and b11)"/>
67 <condition id="b10" value="(c: >= t02 or tb02 > c:)"/>
68 <condition id="b11" value="M1 = 1"/>
```

(abstract of Figure B)

In our example in bloc b2 (Figure 13) expresses that max_pedestrian has to be smaller or equal than the red time of fgd/link 10, here we have to swap both expressions (from: tr(fgd) <= max_pedestrian to max_pedestrian >= tr(fgd)), because the traffic light definition is processed in a xml file, for that reason we have to pay attention with the possible syntax of xml. The XML standard prohibits use of the '<' character within an attribute value. The simplest solution is to reverse the inequality and use the permitted

'>' character. A less readable alternative would be to use the xml code '<' to encode the '<' character. These two options are possible ways to implement the specific condition:

```
63    <condition id="b2" value="(g:0 >= tgrmax_FVA) or (max_pedestrian >= r:9)"/>
```
(abstract of Figure B)

or

```
      <condition id="b2" value="(g:0 >= tgrmax_FVA) or (r:9 &lt;= max_pedestrian)"/>
```

Then we define the pathways for reachable interstage.

```
70      <condition id="b3" value="(b1 and !b2)"/>
71      <condition id="b8" value="(b1 and b2 and b4 and b5 and !b6)"/>
72      <condition id="b9" value="(b1 and b2 and b4 and b5 and b6)"/>
73      <condition id="b13" value="(b1 and b2 and b4 and !b5 and b10 and b11)"/>
74      <condition id="b12" value="(b1 and b2 and b4 and !b5 and b10 and !b11)"/>
```
(abstract of Figure B)

The conditions were named like the interstage action bloc. Then we define conditions which will be used for the attribute *earlyTarget* of every reachable interstage

```
76      <condition id="earlyTarget_INS_1_2" value="b3 or b8 or b13"/>
77      <condition id="earlyTarget_INS_1_3" value="b9 or b12"/>
```
(abstract of Figure B)

Then every *earlyTarget* attribute is assigned the condition (defined above) which collects all paths by which this transition may be reached.

```
32      <phase […] name="INS1.2" earlyTarget="earlyTarget_INS_1_2"/>

40      <phase […] name="INS1.3" earlyTarget="earlyTarget_INS_1_3"/>
```
(abstract of Figure B)

While in stage 1, the conditions for each path are checked in every simulation step. If any of them evaluates to true (non-zero) the phase will switch. It works because every path is unique.

Effectively, all checks that were modelled by attributes *minDur, maxDur, earliestEnd, latestEnd* and *finalTarget* in the first example, were moved into the *earlyTarget* check. for that reason, minDur was set "0" and maxDur to a very high value (maxDur may also be omitted for the same effect).

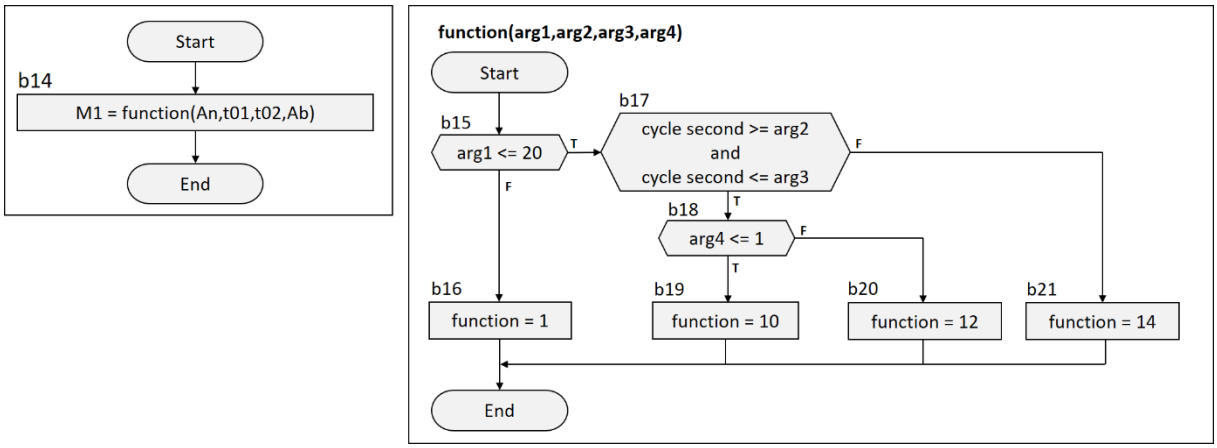
```
31      <phase duration="99"[…] name="stage1" minDur="0" maxDur="1000000" […]/>
```
(abstract of Figure B)

In the example 1, the logic uses a special function which we describe below.

3.5 Function definition

In the algorithm description within the TLD logic, function definitions are often use to structure repeating code. In our example we use a function for data preparation of the public transport prioritization.

Figure 15: function – TLD logic



A function definition requires a name (*id*) and the number of input arguments (*nArgs*). (Figure 16, line 83) This is followed by assignments which model all possible paths through the logic diagram of the TLD function. In our example we named the assignments/path like the last action bloc in each path (Figure b16, b19, b20, b21). In the value description of every assignments the input arguments are defined with a **$** number argument. Then assignments for the output argument of the function are defined (Figure 16, line 88-91). The function now checks if a unique path is reached, depending on this a corresponding value is assigned to the function output **$0**. To call the function the expression *ID:arg1,arg2,…,argN* is used in a condition expression. (Figure 16, line 94) (Figure 15, b14). Note that there may be no spaces after a comma.

Figure 16: function – SUMO-xml

```
83    <function id="function" nArgs="4">
84      <assignment id="function_b16" check="1" value="!(20 >= $1)"/>
85      <assignment id="function_b20" check="1"
              value="(20 >= $1) and ((c: >= $2) and ($3 >= c:)) and !(1 >= $4)"/>
86      <assignment id="function_b21" check="1"
              value="(20 >= $1) and !((c: >= $2) and ($3 >= c:))"/>
87      <assignment id="function_b19" check="1"
             value="(20 >= $1) and ((c: >= $2) and ($3 >= c:)) and (1 >= $4)"/>
88      <assignment id="$0" check="function_b16" value="1"/>
89      <assignment id="$0" check="function_b20" value="10"/>
90      <assignment id="$0" check="function_b21" value="12"/>
91      <assignment id="$0" check="function_b19" value="14"/>
92    </function>
93
94    <condition id="M1" value="function:An,t01,t02,Ab"/>
```

(abstract of Figure B)

Assignment definitions placed outside a function definition are evaluated and executed in every simulation step where switching is possible. They are executed in the order in which they are defined. Likewise, in a function all assignments are executed in definition order every time the function is evaluated.

3.6 Tracking Phases

A useful mode for observing and analyzing the switching behavior of a traffic light is a diagram that shows signal and detector states over time. Such functionality is often part of the software suite used by commercial traffic light design software. The sumo-gui application which is part of the SUMO software package, provides the 'Phase Tracker' (Figure 17) dialog to display such a diagram. It was recently extended to optionally show internal controller variables along with detector states to aid in debugging controller operation. The dialog is accessed by right-clicking on a traffic signal and selecting the 'track phases' menu entry.

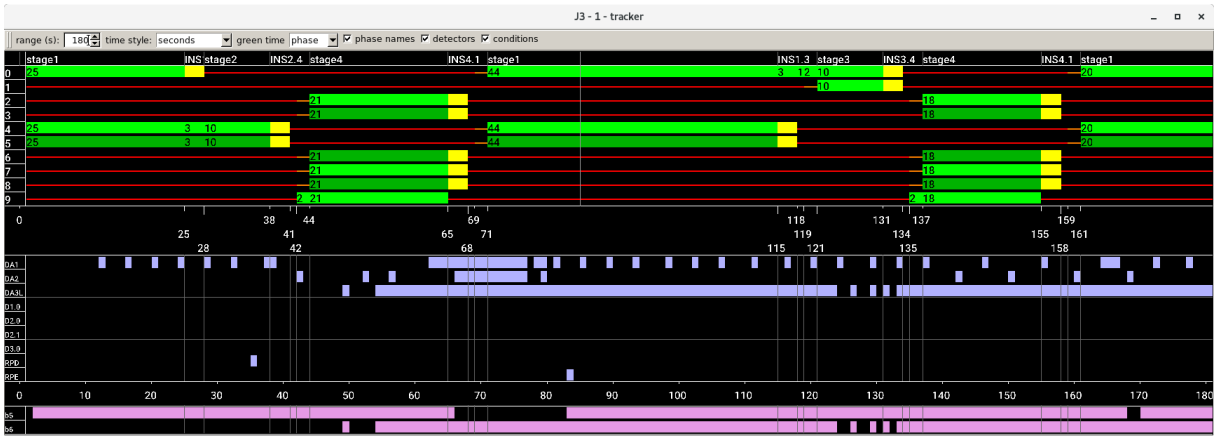
With the parameter key show-conditions, the list of observed expression can be customized.

```
11      <param key="show-conditions" value="b5 b6"/>
```
(abstract of Figure A / Figure B)

With the parameter key extra-detectors, it is possible to visualize in the tracking mode any additional detectors within the sumo simulation. In our example, this is used to track the state of additional induction loops used for the bus prioritization. It works for induction loops as well as laneAreaDetecors.

```
12      <param key="extra-detectors" value="RPD RPE"/>
```
(abstract of Figure A / Figure B)

Figure 17: Tracking Phases:



4 Replication study

4.1 Traffic light protocol

Real-world traffic lights may supply a second-by-second record of their signal states, detector occupation, and public transport telegrams.

Figure 18: Traffic light record:

```
2021-07-31-11-19-57 secOfDay:40796 lza:15 konr:156 diff:1836 values: 303 0 0 189 145 273 0 0 0 0 0 729 175 0 0 0 40 3630 3970 40
2021-07-31-11-19-58 secOfDay:40797 lza:15 konr:156 diff:1857 values: 403 0 0 190 145 273 0 0 0 0 0 729 175 0 0 0 40 3620 3960 40
2021-07-31-11-19-59 secOfDay:40798 lza:15 konr:156 diff:1775 values: 503 0 0 191 145 273 0 0 0 0 0 729 175 0 0 0 40 3610 3950 40
```

This protocol is specific to a particular controller software version and also to the control algorithm itself. With the aid of controller software documentation and the TLD it is possible to interpret all elements of the record and to write a semantically identical record based on the outputs of a SUMO simulation.

By preparing a simulation that replays the detector-states from a real-world controller record, and writing a corresponding simulation record, we can compare whether the simulated controller has the same switching behavior as the real-world controller for the recorded traffic situation.

4.2 TraCI Simulation

To simplify the replay-simulation, we have used the TraCI client functionality for setting artificial detector states. This avoids the need for creating vehicles which would trigger the detectors at the recorded times.

The TraCI function `inductionloop.overrideTimeSinceDetection` was used to replicate the exact detection times from the real-world recording.

Figure 19 example trigger a SUMO detector with TraCI (python)

```
traci.inductionloop.overrideTimeSinceDetection("DA1",0)   #Demand
traci.inductionloop.overrideTimeSinceDetection("DA1",-1)  #no Demand
```

In our project, the real-world intersection (located in Ingolstadt) also participates in a network wide control scheme. This means it is supplied externally with integer values that may change over time and which are used in the controller logic.

To emulate access to these variables, we have added virtual laneAreaDetectors in SUMO and supplied the values (x) from the real-world record via the function `traci.lanearea.overrideVehicleNumber`.

Figure 20: example trigger a SUMO laneAreaDetectors with TraCI (python)

```
traci.lanearea.overrideVehicleNumber("T6", X) #Value X
traci.lanearea.overrideVehicleNumber("T6",-1) #No Value
```

This way we can use the expression “a:DETECTOR_ID” to retrieve the recorded numerical values.

Figure 21: implementation to retrieve the recorded numerical values in the traffic light control

```
<condition id="T6" value="a:T6"/>
<assignment id="t06" check="a:T6 != 0 " value="T6"/>
```

Since the operating of the network wide control scheme has its own control logic based on parameters and detector states, it would have also been possible to replicate this logic within the custom switching rules. This, would make it necessary to include all detectors that participate in network control within the simulation.

4.3 Results

We simulated 60 minutes with the detector record data and compared the second-by-second sequence of stages and interstages from the record with the corresponding sequence from the simulation. A matching sequence indicates that the simulation traffic light in SUMO has the same behavior as the recorded real-world traffic light.

In our experiment the stages and interstages were reproduced with an accuracy of 96%. While analyzing the real-world record we observed errors such as gaps and duplicate time steps and this is likely a source of the disagreement between both records. And some synchronizations situations from outside/external are not completely reproduced.

Due to time constraints, we did not translate the logic modules for initializing the program at daybreak or for switching it off at nightfall. We also excluded the code for emergency vehicle prioritization. In our tests we therefore used a record sequence that did not feature these events.

Unfortunately, we also could not test the code for bus prioritization because some of the functions referenced in the TLD were not made available by the vendor of the controller software in time for this publication.[i] For this reason, our tests also excluded bus approaches.

Nevertheless, we are convinced that the omitted logic modules can be reproduced in SUMO as long as their behavioral description is available in full.

5 Conclusion

In prior versions of SUMO, it was only possible to model detailed adaptive traffic light control with an external TraCI client process and this route was only open to users with considerable experience in computer programming. With the configuration language presented in this work, it is possible to achieve a high-fidelity simulation given familiarity with traffic signal design documents and very limited programming knowledge.

Nevertheless, replicating the described toolchain for replaying a controller record and generating a corresponding simulation record still requires programming experience.

[i] GEVAS Software GmbH has graciously provided control flow diagrams for some of its functions already and we expect to replicate all features of bus prioritization eventually.

6 Abbreviations

b	logic bloc
bx(f)	bx is false
bx-by-bz	it defines a path of logic blocs
by	by is true
cycle second	the time within the current cycle (0 <= cycle second < cycle time)
cycle time	the duration of a full cycle
F	false
Fg	Pedestrian
FV	Individual motorized Traffic
INS	interstage
P1	Signal program 1 / Signal plan 1
T	true
tgr	green time of the signal group
TLD	traffic light document
tr	red time of the signal group
TraCI	Traffic Control Interface (a SUMO API)

7 Appendix

Figure A: Example 1 – SUMO-xml

```
<additional>
  <tlLogic id="J3" type="actuated" programID="1" offset="-5">
     <param key="coordinated" value="true"/>
     <param key="cycleTime" value="90"/>

     <param value="DA1" key="-E3_0" />
     <param value="DA2" key="-E3_1" />
     <param value="DA3L" key="-E3_2" />


     <param key="show-conditions" value="b5 b6"/>
     <param key="extra-detectors" value="RPD RPE"/>

     <!-- minal/maximal times-->
     <condition id="min_Stage_1"          value="10"/>
     <condition id="max_Stage_1"          value="60"/>
     <condition id="tgrmin_FVA"           value="35"/>
     <condition id="tgrmax_FVA"           value="32"/>
     <condition id="max_pedestrian"       value="75"/>

     <!--constants-->
     <condition id="k1" value="1"/>
     <condition id="k2" value="2"/>

     <!--time conditions-->
     <condition id="t01"  value="30"/>
     <condition id="t02"  value="65"/>
     <condition id="tb01" value="45"/>
     <condition id="tb02" value="80"/>

    <!—link index:              0123456789      -->
     <phase duration="99" state="GrrrGgrrrr" name="stage1" minDur="-1" maxDur="-1"
            earliestEnd="-1" latestEnd="-1" next="1 6"/>                          <!--0-->
     <phase duration="3" state="yrrrGgrrrr" name="INS1.2"
            earlyTarget="b5 and !b6" finalTarget="b2 or !b11"/>                    <!--1-->

     <phase duration="10"state="rrrrGgrrrr" name="stage2"/>                        <!--2-->

     <phase duration="3" state="rrrryyrrrr" name="INS2.4"/>                        <!--3-->
     <phase duration="1" state="rrrrrrrrrr" name="INS2.4"/>                        <!--4-->
     <phase duration="2" state="rruurruuuG" name="INS2.4" next = "13"/>            <!--5-->

     <phase duration="3" state="Grrryyrrrr" name="INS1.3"
            earlyTarget="b5 and b6" finalTarget="(b10 and b11)"/>                  <!--6-->
     <phase duration="1" state="Grrrrrrrrr" name="INS1.3"/>                        <!--7-->
     <phase duration="2" state="Gurrrrrrrr" name="INS1.3"/>                        <!--8-->

     <phase duration="10"state="GGrrrrrrrr" name="stage3"/>                        <!--9-->

     <phase duration="3" state="yyrrrrrrrr" name="INS3.4"/>                        <!--10-->
     <phase duration="1" state="rrrrrrrrrr" name="INS3.4"/>                        <!--11-->
     <phase duration="2" state="rrurrruuuG" name="INS3.4"/>                        <!--12-->

     <phase duration="40" state="rrGgrrgGgG" name="stage4" minDur="0" maxDur="40"
            earliestEnd="70" latestEnd="84"/>                                      <!--13-->

     <phase duration="3" state="rryyrryyyr" name="INS4.1"/>                        <!--14-->
     <phase duration="1" state="rrrrrrrrrr" name="INS4.1"/>                       <!--15-->
     <phase duration="2" state="urrruurrrr" name="INS4.1"/>                       <!--16-->

     <!--logical condtion definition-->
     <condition id="l1" value="(z:DA1 >= 3) or (z:DA2 >= 3)"/>
     <condition id="An" value="z:RPD + k1"/>
     <condition id="Ab" value="z:RPE + k2"/>
     <condition id="l2" value="a:DA3L"/>

     <condition id="minDur:0" value="min_Stage_1"/>
     <condition id="maxDur:0" value="max_Stage_1"/>

     <condition id="earliestEnd:0"        value="t01"/>
     <condition id="latestEnd:0"          value="t02"/>
```

```
69
70      <condition id="b5" value="11"/>
71      <condition id="b6" value="12"/>
72      <condition id="b11" value="M1 = 1"/>
73
74      <condition id="b1" value="min_Stage_1 >= c:"/>
75      <condition id="b2" value="max_Stage_1 >= c:"/>
76      <condition id="b4" value="t01 >= c:"/>
77      <condition id="b10" value="t02 >= c:"/>
78
79      <condition id="V" value="0"/>
80
81      <assignment id="V" check="b1 and b2 and b4 and b5 and !b6" value="12"/>
82
83      <function id="function" nArgs="4">
84              <assignment id="function_b16" check="1" value="!(20 >= $1)" />
85              <assignment id="function_b20" check="1"
                        value="(20 >= $1) and ((c: >= $2) and ($3 >= c:)) and !(1 >= $4)"/>
86              <assignment id="function_b21" check="1"
                        value="(20 >= $1) and !((c: >= $2) and ($3 >= c:))"/>
87              <assignment id="function_b19" check="1"
                        value="(20 >= $1) and ((c: >= $2) and ($3 >= c:)) and (1 >= $4)"/>
88              <assignment id="$0" check="function_b16" value="1"/>
89              <assignment id="$0" check="function_b20" value="10"/>
90              <assignment id="$0" check="function_b21" value="12"/>
91              <assignment id="$0" check="function_b19" value="14"/>
92      </function>
93
94      <condition id="M1" value="function:An,t01,t02,Ab"/>
95
96    </tlLogic>
97</additional>
```

Figure B: Example 2 – SUMO-xml

```
1<additional>
2   <tlLogic id="J3" type="actuated" programID="1" offset="-5">
3       <param key="coordinated" value="true"/>
4       <param key="cycleTime" value="90"/>
5
6       <param value="DA1" key="-E3_0" />
7       <param value="DA2" key="-E3_1" />
8       <param value="DA3L" key="-E3_2" />
9
10
11      <param key="show-conditions" value="b5 b6"/>
12      <param key="extra-detectors" value="RPD RPE"/>
13
14      <!-- minal/maximal times-->
15      <condition id="tgrmin_FVA"          value="35"/>
16      <condition id="tgrmax_FVA"          value="32"/>
17      <condition id="max_pedestrian"      value="75"/>
18
19      <!--constants-->
20      <condition id="k1" value="1"/>
21      <condition id="k2" value="2"/>
22
23      <!--time conditions-->
24      <condition id="t01"  value="30"/>
25      <condition id="t02"  value="65"/>
26      <condition id="tb01" value="45"/>
27      <condition id="tb02" value="80"/>
28
29     <!-- link index:           0123456789 -->
30      <phase duration="99" state="GrrrGgrrrr" name="stage1" minDur="0" maxDur="1000000"
               next="1 6" />                                                    <!--0-->
31
32      <phase duration="3" state="yrrrGgrrrr" name="INS1.2"
               earlyTarget="earlyTarget_INS_1_2"/>                              <!--1-->
33
34      <phase duration="10"state="rrrrGgrrrr" name="stage2"/>                   <!--2-->
35
36      <phase duration="3" state="rrrryyrrrr" name="INS2.4"/>                   <!--3-->
37      <phase duration="1" state="rrrrrrrrrr" name="INS2.4"/>                    <!--4-->
```

```
38      <phase duration="2" state="rruurruuuG" name="INS2.4" next= "13"/>              <!--5-->
39
40      <phase duration="3" state="Grrryyrrrr" name="INS1.3"
               earlyTarget="earlyTarget_INS_1_3"/>                                    <!--6-->
41      <phase duration="1" state="Grrrrrrrrr" name="INS1.3"/>                         <!--7-->
42      <phase duration="2" state="Gurrrrrrrr" name="INS1.3"/>                         <!--8-->
43
44      <phase duration="10"state="GGrrrrrrrr" name="stage3"/>                         <!--9-->
45
46      <phase duration="3" state="yyrrrrrrrr" name="INS3.4"/>                        <!--10-->
47      <phase duration="1" state="rrrrrrrrrr" name="INS3.4"/>                        <!--11-->
48      <phase duration="2" state="rrurrruuuG" name="INS3.4"/>                        <!--12-->
49
50      <phase duration="40" state="rrGgrrgGgG" name="stage4" minDur="0" maxDur="40"
               earliestEnd="70" latestEnd="84"/>                                     <!--13-->
51
52      <phase duration="3" state="rryyrryyyr" name="INS4.1"/>                        <!--14-->
53      <phase duration="1" state="rrrrrrrrrr" name="INS4.1"/>                        <!--15-->
54      <phase duration="2" state="urrruurrrr" name="INS4.1"/>                        <!--16-->
55
56      <!--logical condition definition-->
57      <condition id="l1" value="(z:DA1 >= 3) or (z:DA2 >= 3)"/>
58      <condition id="An" value="z:RPD + k1"/>
59      <condition id="Ab" value="z:RPE + k2"/>
60      <condition id="l2" value="a:DA3L"/>
61
62      <condition id="b1" value="(g:0  >= tgrmin_FVA)"/>
63      <condition id="b2" value="(g:0 >= tgrmax_FVA) or (max_pedestrian >= r:9)"/>
64      <condition id="b4" value="(c: >= t01 and tb01 > c:)"/>
65      <condition id="b5" value="l1"/>
66      <condition id="b6" value="(l2 and b11)"/>
67      <condition id="b10" value="(c: >= t02 and tb02 > c:)"/>
68      <condition id="b11" value="M1 = 1"/>
69
70      <condition id="b3" value="(b1 and !b2)"/>
71      <condition id="b8" value="(b1 and b2 and b4 and b5 and !b6)"/>
72      <condition id="b9" value="(b1 and b2 and b4 and b5 and b6)"/>
73      <condition id="b13" value="(b1 and b2 and b4 and !b5 and b10 and b11)"/>
74      <condition id="b12" value="(b1 and b2 and b4 and !b5 and b10 and !b11)"/>
75
76      <condition id="earlyTarget_INS_1_2" value="b3 or b8 or b13"/>
77      <condition id="earlyTarget_INS_1_3" value="b9 or b12"/>
78
79      <condition id="V" value="0"/>
80
81      <assignment id="V" check="b1 and b2 and b4 and b5 and !b6" value="12"/>
82
83      <function id="function" nArgs="4">
84              <assignment id="function_b16" check="1" value="!(20 >= $1)" />
85              <assignment id="function_b20" check="1"
                        value="(20 >= $1) and ((c: >= $2) and ($3 >= c:))and !(1 >= $4)/>
86              <assignment id="function_b21" check="1"
                        value="(20 >= $1) and !((c: >= $2) and ($3 >= c:))"/>
87              <assignment id="function_b19" check="1"
                        value="(20 >= $1) and ((c: >= $2) and ($3 >= c:)) and (1 >= $4)"/>
88              <assignment id="$0" check="function_b16" value="1"/>
89              <assignment id="$0" check="function_b20" value="10"/>
90              <assignment id="$0" check="function_b21" value="12"/>
91              <assignment id="$0" check="function_b19" value="14"/>
92      </function>
93
94      <condition id="M1" value="function:An,t01,t02,Ab"/>
95
96    </tlLogic>
97</additional>
```

Figure C: Stage following diagram – TLD

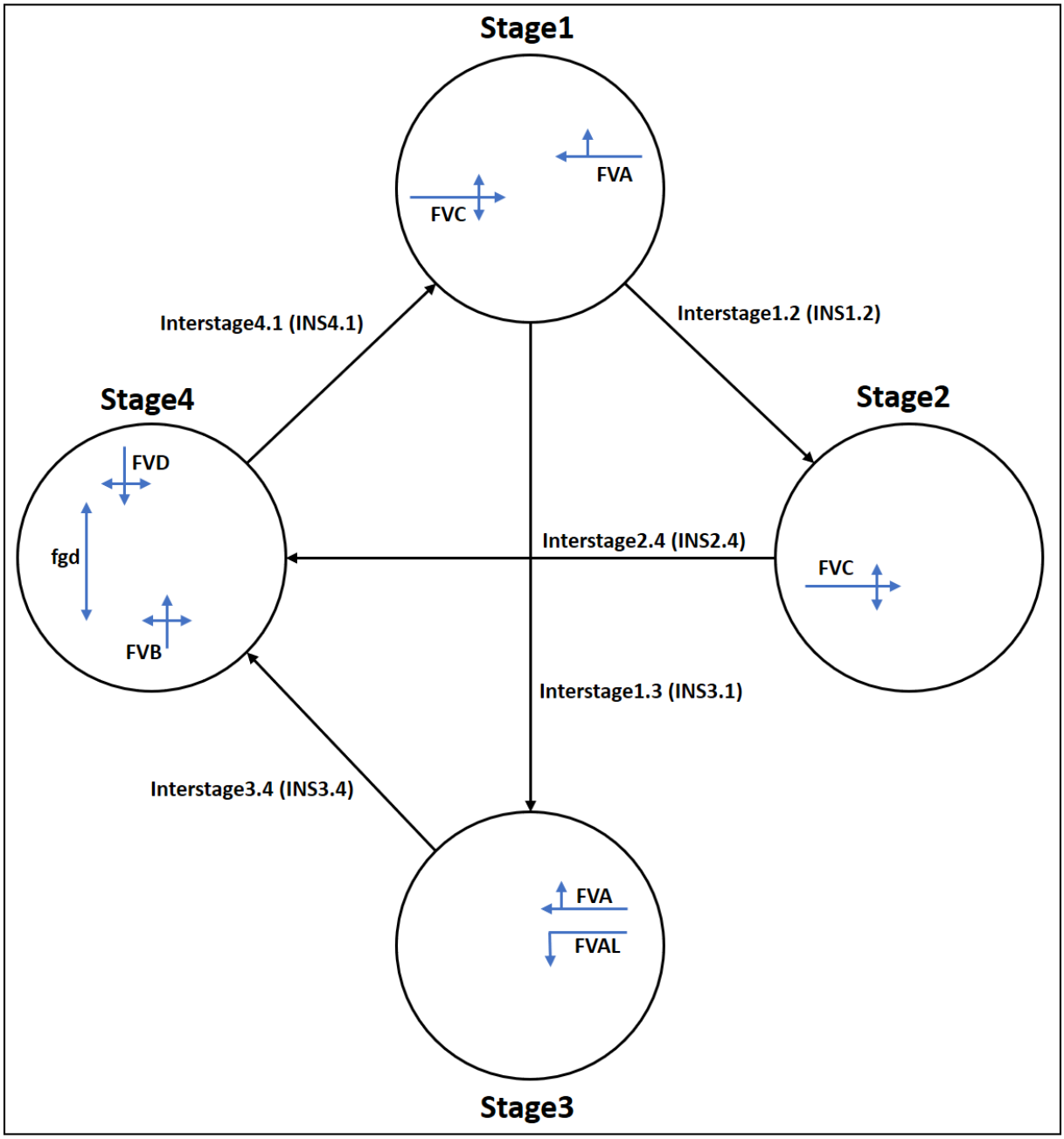


Figure D: confic.xml – SUMO

Note, that the detectors det.xml has always to be loaded bevor the traffic light configuration which references the detectors is loaded.

```
<configuration>

 <input>
   <net-file value="paper_net.net.xml"/>
   <route-files value="paper_routes.rou.xml"/>
   <additional-files value="paper_inductionloop.det.xml,paper_traffic_light_control.add.xml"/>
 </input>

</configuration>
```

8 References

[1] https://sumo.dlr.de/docs/TraCI.html#using_traci
Web page of German Aerospace Center (DLR), accessed 11.04.2022
[2] https://sumo.dlr.de/docs/TraCI.html#resources
Web page of German Aerospace Center (DLR), accessed 11.04.2022
[3] https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html
Web page of German Aerospace Center (DLR), accessed 11.04.2022
[4] https://savenow.de/de/
Web page of SAVeNoW, accessed 11.04.2022
[5] https://github.com/DLR-TS/sumo-ocit
Source code repository for import of OCIT data files, accessed 11.04.2022