ASAM OpenSCENARIO 2.0.0 Status and 2.0.1 Proposal

Gil Amid
Foretellix Ltd

10. November 2022

ASAM Association for Standardization of Automation and Measuring Systems

Outline

- ASAM OpenSCENARIO ® 2.0.0 overview
  - Difference between the two major versions (1.x, 2.0.0)
- Migration path – key points
- Feedback on open issues – status
- Direction statement toward 2.0.1

Quick Status

- ASAM OpenSCENARIO ® 2.0.0 released last July
- Feedback can be submitted directly from the on-line document
- ASAM initial roadmap is in place

ASAM OpenSCENARIO 1.x - Summary

- OpenSCENARIO describes the dynamic content, including the overall description and coordination of behavior of dynamic entities.
- OpenSCENARIO does not specify the behavior models themselves, nor their handling by the simulation engine, including initialization and setup, runtime interfaces, packaging, etc.
- OpenSCENARIO also does not define the road network or any geometric, visual or physical assets and characteristics used in a simulation. These are instead employed through references to other established formats. Hence, in certain contexts, OpenSCENARIO can be considered as a top-level container. It references other specifications for other relevant parts of the overall scenario.

ASAM OpenSCENARIO® 2.0.0 - High Level – Key improvements

- Improved readability – Better readable (vs XML)
- Programming language (vs XML) -Object Oriented DSL
- Consistent scenario description over levels of abstraction (supports Abstract, Logical and Concrete description)
- Easier and consistent binding to external code/functions/methods for every purpose
- Built in Constraints
- Built in KPI and Coverage measurements
- Composability - completely flexible modularization for reusability of scenarios, unlimited nesting and mixing
- Extendible - the language and domain entities
- Designed to support all test platforms: Sil, Hil, Vil, Mil, Proving Ground

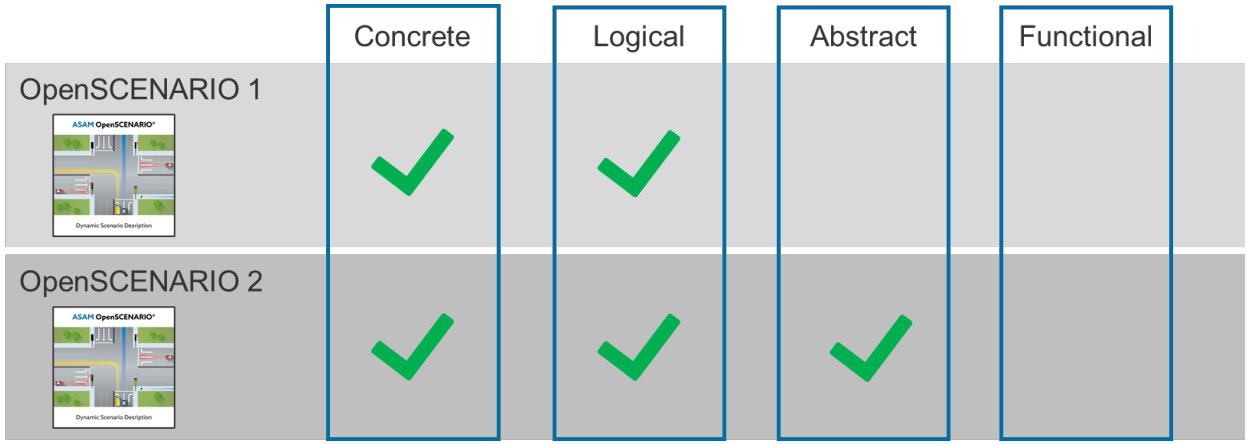
ASAM OpenSCENARIO 2.0 - RECAP

- ASAM OpenSCENARIO 2.0 is **the format and mechanism to supply dynamic content and functional behavior to all testing and execution platforms,** for all driving scenarios ranging from simple motor-way interactions to long-running, complex inner-city traffic scenarios. In addition, it supports static entities, scenario validity criteria, KPIs and measurement, coverage measurements.

- ASAM OpenSCENARIO 2.0 **is not** backward compatible with version 1.x. A migration guide with functionality and feature mapping is supplied as part of the documents for the standards.

ASAM OpenSCENARIO 2.0 – In slightly less plain language

- This is a declarative domain specific **programming language** combined with a domain model, specifying entities with their properties
- The language supports **abstract, logical** and **concrete** levels of abstraction



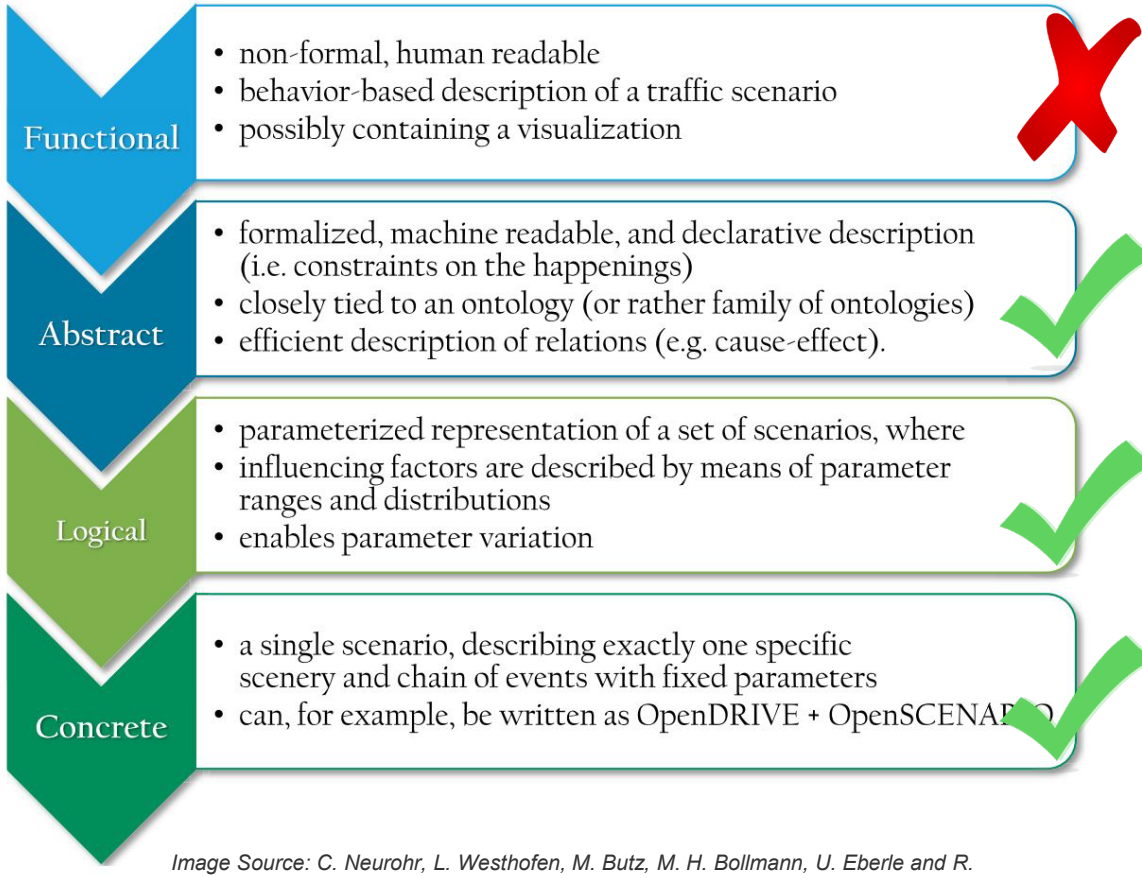


Image Source: C. Neurohr, L. Westhofen, M. Butz, M. H. Bollmann, U. Eberle and R. Galbas, "Criticality Analysis for the Verification and Validation of Automated Vehicles," in IEEE Access, vol. 9, pp. 18016-18041, 2021, doi: 10.1109/ACCESS.2021.3053159.

ASAM OpenSCENARIO 2.0 – In Plain language

- The language enables:

- •Specification of the temporal order of actions, through language mechanisms such as serial and parallel
- •Reuse or combine individual scenarios in order to create more complex scenarios.
- •Definition of events, e.g. for data capture or triggering of further actions
- •Use events to monitor ADS errors.
- •Specification of ranges or any other behavioral/numerical considerations, through constraints
- •Setting coverage goals and accumulating coverage
- •Abstract road network descriptions that remove dependency on specific maps, allowing one to match the specified constraints to multiple maps/ODDs
- •**Extending the domain model, adding objects, entities, types and more.**

ASAM OpenSCENARIO 2.0.0 – Enabling new capabilities

- The language features enable:
- **Developments and maintenance of test scenarios that are independent of specific map and geography** (abstract road networks). Can automatically be mapped to a geography.
- **Consistent reporting and documentation** of error checking and scenario test coverage results – using built in checking mechanism and coverage accumulation
- **Shared and consistent definition of ADS errors**, thresholds and pass/fail criteria – using constraints, events and built-in checking.
- **Consistent definition of required testing and coverage ranges** for every value/parameter (from colors of the vehicles to acceleration …..)
- **Reuse or combine individual scenarios** from libraries/catalogs in order to create more complex scenarios.

ASAM OpenSCENARIO 1 – 2 levels of abstraction

Concrete:
scenario.xosc

```
<!-- scenario.xosc -->
<ParameterDeclarations>
  <ParameterDeclaration name="Ego_speed"
                        parameterType="double" value="10"/>
</ParameterDeclarations>
...
<Actions>
  <Private entityRef="Ego">
    <PrivateAction>
      <LongitudinalAction>
        <SpeedAction>
          <SpeedActionDynamics dynamicsShape="step"
                               dynamicsDimension="time" value="0"/>
          <SpeedActionTarget>
            <AbsoluteTargetSpeed value="$Ego_speed"/>
          </SpeedActionTarget>
        </SpeedAction>
      </LongitudinalAction>
    </PrivateAction>
  </Private>
</Actions>
```

Logical:
scenario.xosc +
distribution.xosc

```
<!-- distribution.xosc -->
<ParameterValueDistribution>
  <ScenarioFile filepath="scenario.xosc"/>
  <Deterministic>
    <DeterministicSingleParameterDistribution
                parameterName="Ego_speed">
      <DistributionRange stepWidth="10">
        <Range lowerLimit="10" upperLimit="60"/>
      </DistributionRange>
    </DeterministicSingleParameterDistribution>
  </Deterministic>
</ParameterValueDistribution>
```

ASAM OpenSCENARIO 2 – 3 levels of abstraction

```
# Concrete – specific test
ego.drive() with:
    speed(10kph)

# Logical (R-157 speed range)
ego.drive() with:
    speed(ël 0kph..60kph])

# Abstract
ego.drive() with:
    keep(it.speed <= speed_limit)
```

"The ADS shall not cross the legal speed limit "

ASAM OpenSCENARIO 2.0 – Examples

Example for use of the parallel operator

```
scenario parallel_phases:
    v1, v2: vehicle
    do parallel( ):
        phaseA: v1.drive( ) with:
            speed(speed: 0kph, at: start)
            speed(speed: 10kph, at: end)
        phaseB: v2.drive( ) with:
            speed(speed: [10..15]kph)
            position(distance: [5..100]meter, behind: v1, at: start)
```

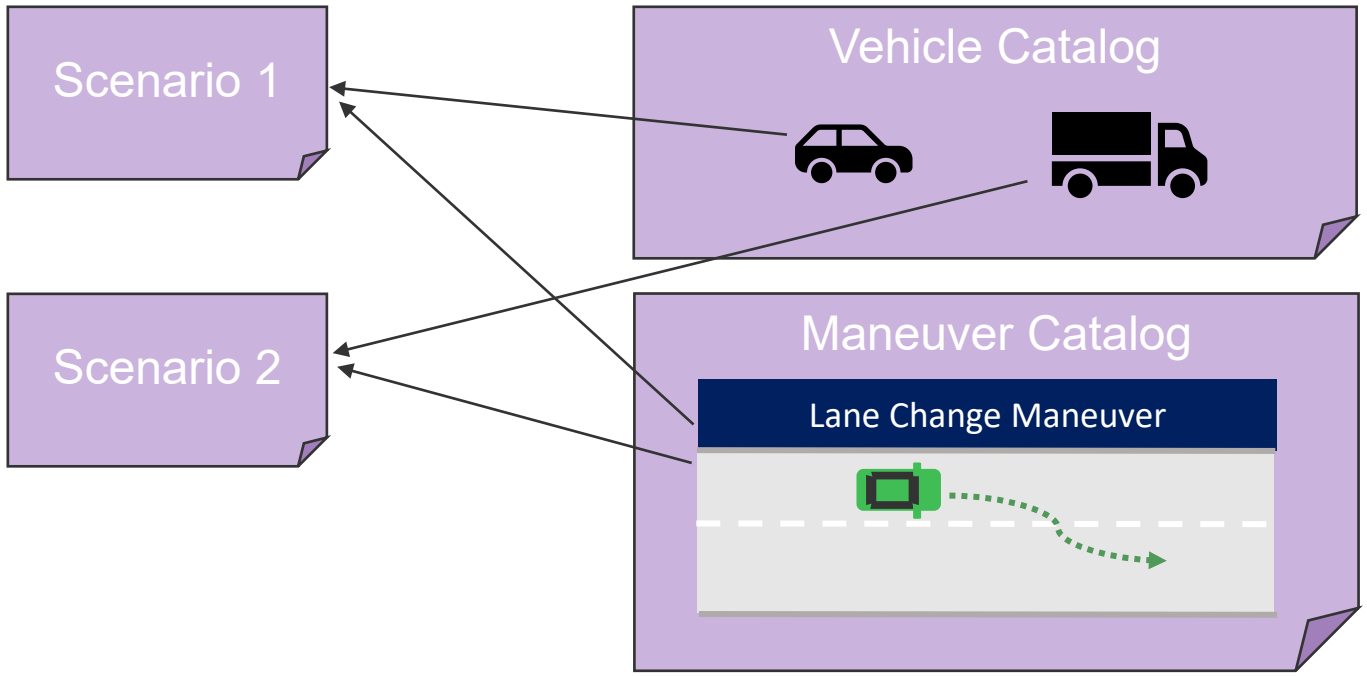
Example for emit event

```
event vehicle_reached_speed(vehicle : vehicle, vehicle_speed : speed) # event definition

...

scenario car.two_phases:
    do serial:
        phase1: drive( ) with:
            speed(speed: 0kph, at: start)
            speed(speed: [25..35]kph, at: end)
        emit vehicle_reached_speed(actor, actor.speed)
        phase2: drive( ) with:
            speed(speed: [30..50]kph)
```

ASAM OpenSCENARIO 1 – Scenario Composition

Use catalogs to reuse scenario sections:

- Only one hierarchy level
- Only specific predefined catalogs

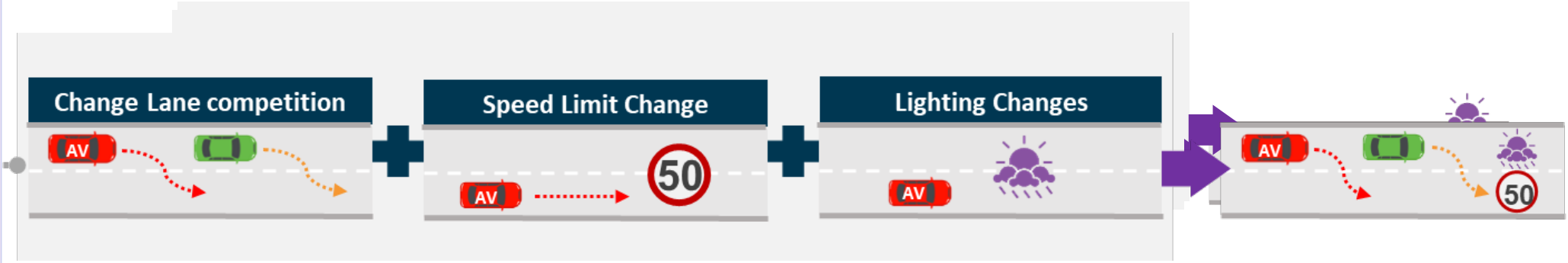


ASAM OpenSCENARIO 2 - Scenario Composition

Easily compose complex scenarios, by combining simple scenarios in serial or parallel

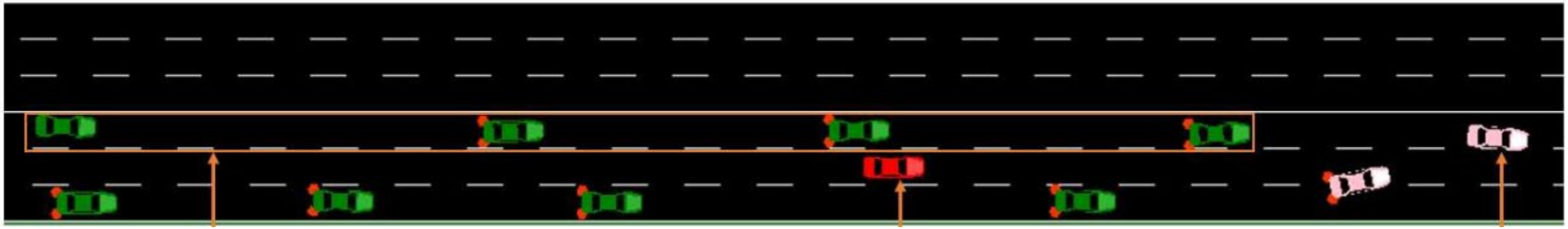
Use catalogs/libraries to create complex interactions and ODD conditions:

- Infinite hierarchy levels possible
- Reuse anything up to complete scenarios

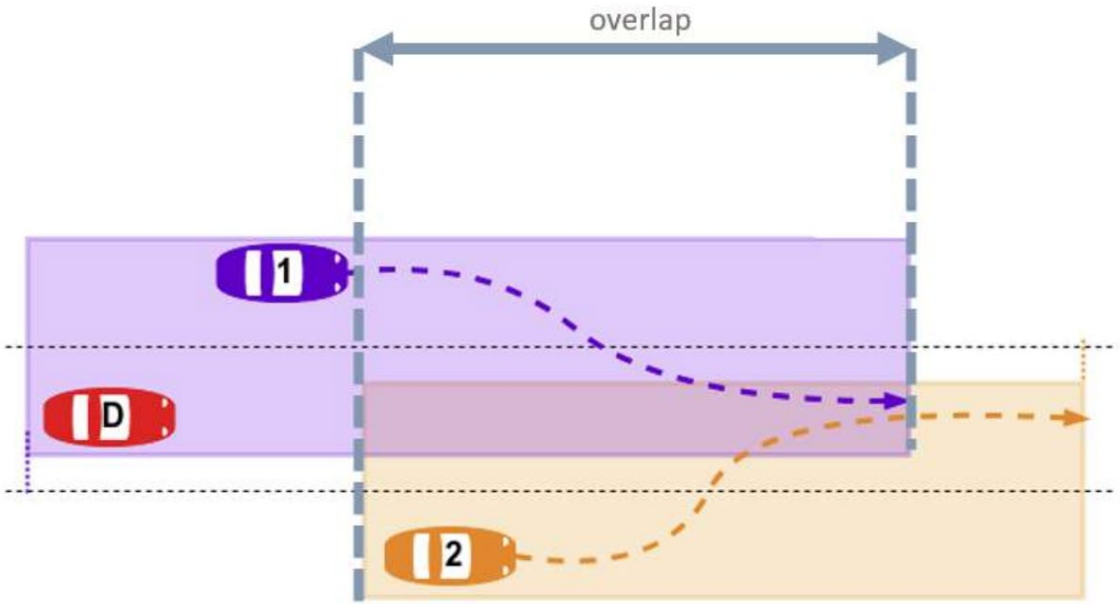


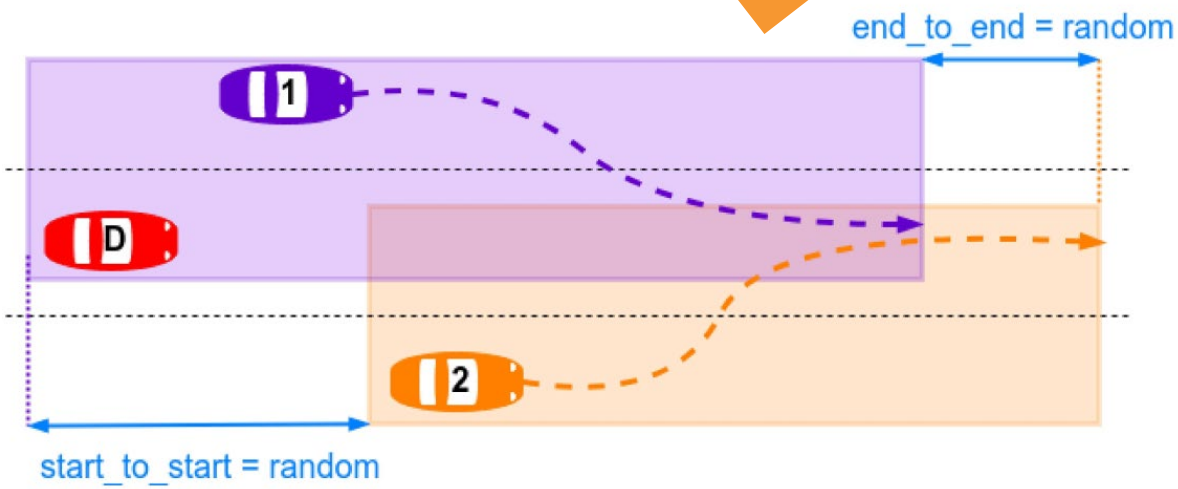
Scenarios are building blocks for more complex scenarios and tests.
Usage: ALKS multi vehicle interactions can be built upon single ALKS scenarios.

Various Mixing Options– Using “Parallel” operator

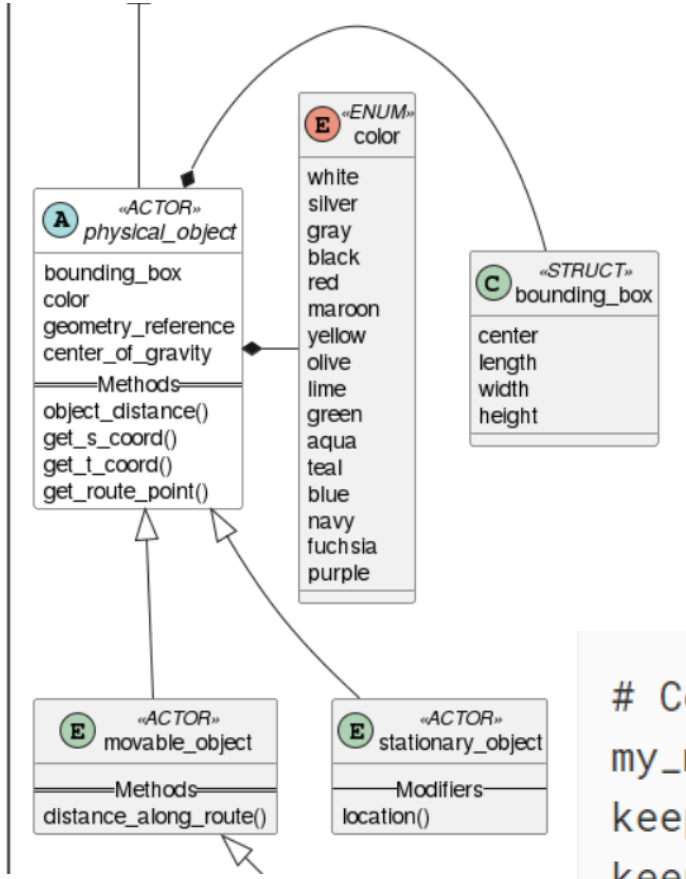


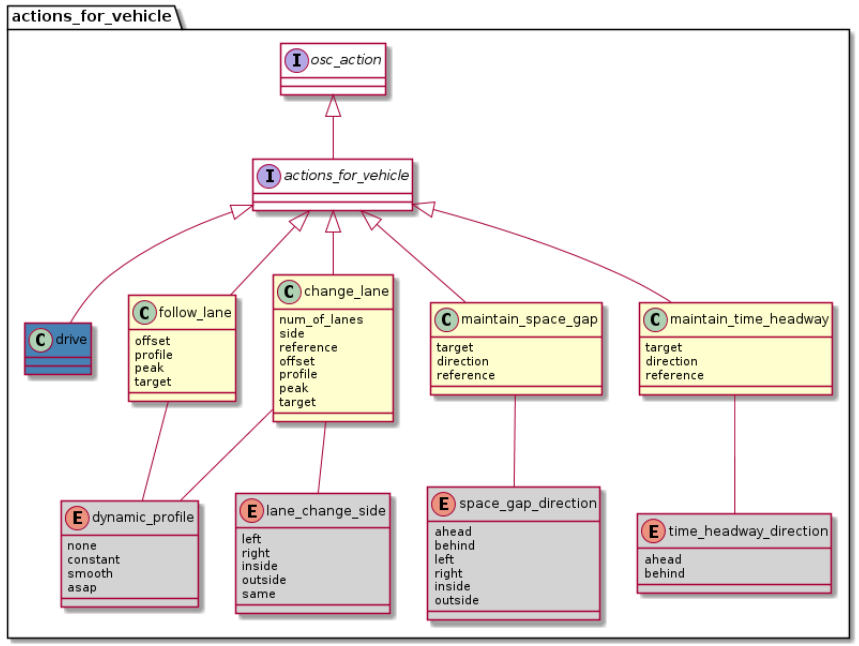
Ego





ASAM OpenSCENARIO 2.0 – Examples (Domain Model)



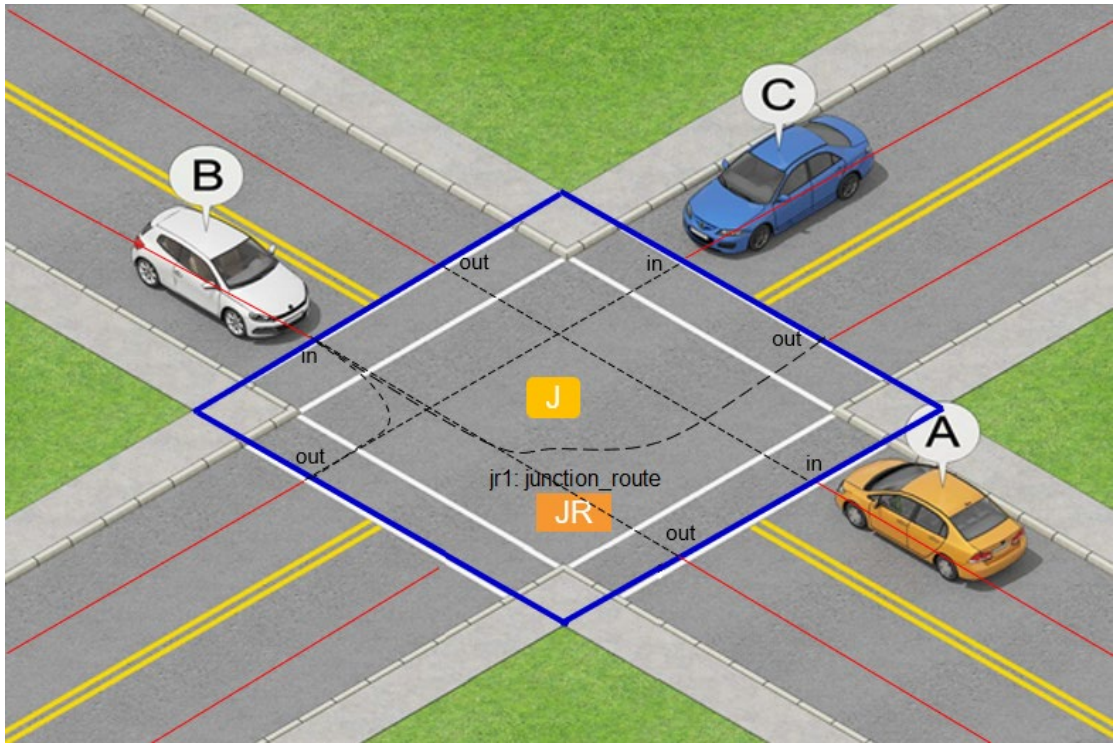


Key Difference:
OSC 2.0 has a domain model which is extendible.
OSC 1.x – has a data model. (defines actions, triggers)

```
# Constrain vehicle to be a motorcycle
my_motorcycle: vehicle
keep(my_motorcycle.vehicle_category == vru_vehicle)
keep(my_motorcycle.axles.size() == 2)
keep(my_motorcycle.axles[0].number_of_wheels == 1)
keep(my_motorcycle.axles[1].number_of_wheels == 1)
keep(my_motorcycle.intended_infrastructure[0] == driving)
```

ASAM OpenSCENARIO 2.0 – Road Abstraction

- An abstract description of the features of the road network that influence the behavior of the actors during the scenario. (mini-map)
- Not a replacement for openDRIVE/real map.
- Creates a search space to find suitable location in a map.
- Usage: Scenario can automatically be placed on any location within the map, where these features exist.
- Usage: Ability to describe actor behaviors on these features.




```
my_junction: junction
road_1, road_2, road_3, road_4: road
jr_12, jr_34: road

r1: map.roads_follow_in_junction(junction: my_junction, in_road: road_1, out_road: road_2, junction_route: jr_12)
r3: map.roads_follow_in_junction(junction: my_junction, in_road: road_3, out_road: road_4, junction_route: jr_34)
```

Built in risk mitigation:

- The standard is defining the language and domain model, while at the same time leaving room for implementation specific items.
- The standard offers significant extendibility capabilities, that can be used to bridge gaps.
- Examples:
  - Domain Model Extensions can be used to add entities and functionality.
  - External Functions calls and binding –can be used for various purposes – e.g. distribution functions.

ASAM OpenSCENARIO 2.0 - Out of Scope

System under test	The exact description of the system under test, e.g. detailed vehicle configuration, sensor placement, sensor models etc. is not part of OpenSCENARIO.
Behavioral model	The standard includes interface to behavioral models for actors. Those models are implementation specific.
Vehicle dynamics	Although the standard describes maneuvers in a kinematic way. The standard does not include all necessary elements to specify advanced motion dynamics.
Environmental models	The standard incorporates a domain actor to specify environment information and actions (e.g. rain, wind and more) but does not describe how this is to be interpreted by the testing platform.

Outline

- ASAM OpenSCENARIO ® 2.0.0 overview
  - Difference between the two major versions (1.x, 2.0.0)
- Migration path – key points
- Feedback on open issues – status
- Direction statement toward 2.0.1

MIGRATION OSC 1.x to 2.0.0 - Key Points

- Migration guide is non-normative
- Guide covers OpenSCENARIO 1.0 with some exceptions.

- OpenSCENARIO 1.1 – parameter expressions and constraints covered automatically
  - Parameter distributions covered in guide
- Catalogs – handled

- Storyboard, event priorities

- Controllers – proposed model for migration to cover OpenSCENARIO 1.0 controller concept, up to simulation environment to implement in this way.
- Routes, trajectories – functionality is there, some naming changes, slight semantic differences (duration of actions).

MIGRATION OSC 1.x to 2.0.0 –Not covered in this release, can be implemented with coding or domain model extensions:

- AddEntity/RemoveEntity – incompatible concepts. Scenarios with intended purpose for physically possible situation can be coded with regular OpenSCENARIO 2.0 concepts. (I.e. manual conversion)
- TrafficSource/Sink – traffic generation works differently. Similar concepts can be coded to implement original intent of traffic swarm creation. (i.e. manual conversion)
- SelectTriggeringEntities – cannot be translated as simply as a flag in OpenSCENARIO 1.x ManeuverGroup. However, recreating the same result is possible when the scenario is coded with this consideration from start. (i.e. manual conversion)
- OpenSCENARIO 1.1 RoutePosition, TrajectoryPosition – Coding using road abstraction.
- TrafficSignal/TrafficSignalController pairs of actions and conditions – road network abstraction is a new concept in OpenSCENARIO 2.0.0 and the signalization part was not yet developed. Equivalent features are planned for next releases of the standard (manual non normative extension is possible now).
- Conditions: EndOfRoad, OffRoad, StandStill, TravelledDistance ((manual non normative extension is possible now , Standstill may be questionable).

MIGRATION OSC 1.x to 2.0.0 - Key Points

- SimulationTime – handling absolute time references is non-normative in the standard today, as any scenario can be used as a sub-scenario of another one. If problem can be reformulated with relative time references, ‘wait elapsed’ can be used.
  - If absolute time is necessary – domain model extension can be used through a simulation environment specific extension (top level clock)

- For now, two minor releases of the OpenSCENARIO 2 are in the plan:
  - OpenSCENARIO 2.0.1 – no major new development, should consist of small fixes and limited additions.

  - OpenSCENARIO 2.1 – a lot of features were in advanced development stage but were not ready for inclusion in 2.0 version of the standard. Depending on maturity, interest in a feature and project schedule, this version should bring more interesting additions and solutions for some known issues in the current release.

Proposal for the project – dedicated migration work group/work package

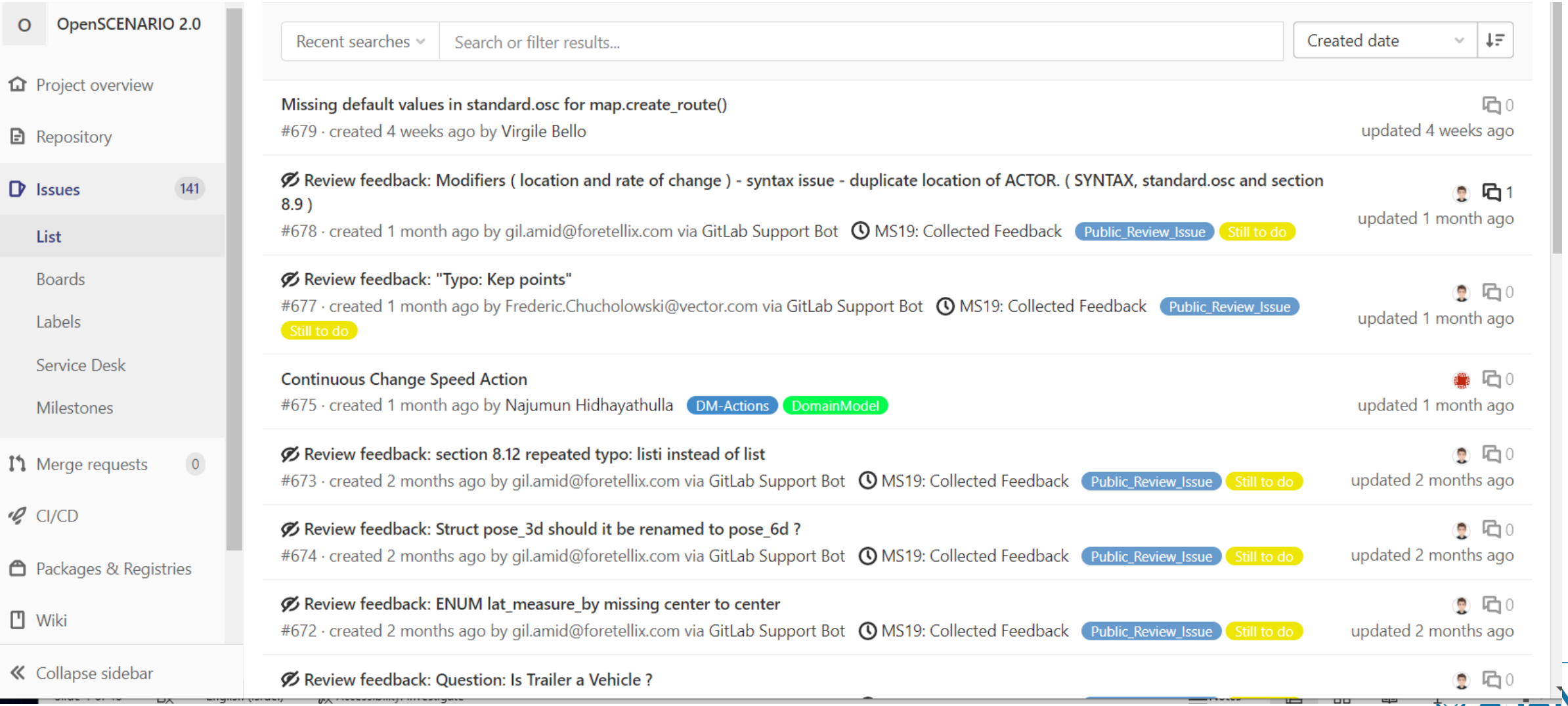
- As can be seen, some migrations topics have different implementation options.
- We need to complete the migration guide, agree on the best method, and potentially guidance for implementation,
- Potentially – consider an “Implementers Forum” for migration issues.

Outline

- ASAM OpenSCENARIO ® 2.0.0 overview
  - Difference between the two major versions (1.x, 2.0.0)
- Migration path – key points
- Feedback an open issues – status
- Direction statement toward 2.0.1

Feedback issues

https://code.asam.net/simulation/standard/openscenario-2.0/-/issues?sort=created_date



Language items pushed to OSC 2.1

Language features either dropped or reduced in scope:

- **Packaging and distribution of scenarios with their dependencies (models, maps, resources):**
  OSC 2.0 will only have file-based shipping of scenarios and import mechanism,
  i.e. a superset of OSC 1.x but not more intelligent packaging, module system, etc.
- **Meta-data mechanism:** A pre-defined mechanism to store meta-data along scenarios was postponed to 2.1.
- **Error-handling system:** OSC 2.0 will only specify the signaling of error condition, but no specified way of handling them, like a full-blown condition system, including handling, restarts, class hierarchies, etc.
- **Namespace support:** Support for creating new namespaces to segregate the global namespace have been designed up to the point that retrofit in 2.1 is feasible, but not included in 2.0.
- **Main entry point for execution:** OSC 2.0 does not specify how the main entry point to scenario execution is selected, this remains implementation dependent for now.
- **Soft constraints:** Postponed soft constraints that can in certain circumstances be broken when needed.
- **Built-in distribution functions:** All probabilistic functions were postponed to 2.1.
- **Full support for OSC 1.0 action dynamics specifications:** Postponed to 2.1.
- **Operational semantic specification:** OSC 2.0 does come with denotational semantics based on trace acceptance, but does not give a complete operational semantic definition.

Domain Model items pushed to OSC 2.1

- Domain Model approach – supply “breadth” , as depth can be added using domain model extension – so no show stopper for usage - Examples:
  - Human Actors (called Person) will only have a role as Pedestrians in 2.0. We envision Persons having roles like vehicle Passenger, mounting/dismounting vehicles, or others in 2.1+
  - Pedestrians (both human and animal) will have limited actions in 2.0. We think in the future you can have more detailed control of head/limb motion, gestures, etc. But only for 2.1+
  - Non-motion actions for vehicles should also fall to 2.1. Things like activating lights (turn, brake) and opening/closing vehicle doors. The 1.x project has an issue for these (see in GitLab). Others should be harmonized with OSI. Maybe better to harmonize 2.1 with 1.2 and OSI before releasing.
  - Traffic light actions: Postponed to 2.1+
  - Groups of vehicle and traffic: normative definitions were postponed to 2.1+

Outline

- ASAM OpenSCENARIO ® 2.0.0 overview
  - Difference between the two major versions (1.x, 2.0.0)
- Migration path – key points
- Feedback on open issues – status
- Direction statement toward 2.0.1

General Direction for the project – OSC 2.0.1 prespective

- Fix errors and bugs reported for ASAM OpenSCENARIO 2.0.0 and cover features being introduced in OSC 1.x Work Package
  Available Time frame is a critical factor for selecting corrections/improvements.
- Improve and close gaps in the migration guidelines ASAM OpenSCENARIO 1.x--->2.0, reach the goal that ASAM OpenSCENARIO 2.x is a full superset of ASAM OpenSCENARIO 1.x.
- Design and plan the roadmap for future ASAM OpenSCENARIO projects

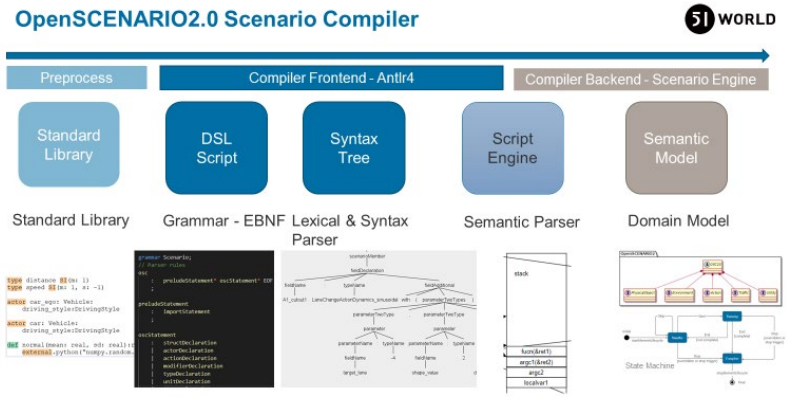
Backup slides

On going implementations (Asam technical seminar)

Foretellix OpenSCENARIO Execution

ASAM

OpenSCENARIO2.0 Scenario Compiler

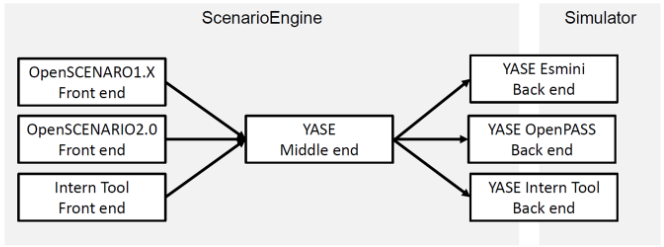


ASAM

YASE (Bosch)

Three phases of scenario compiler/ ScenarioEngine

SET Level



ASAM

Open Source Tools

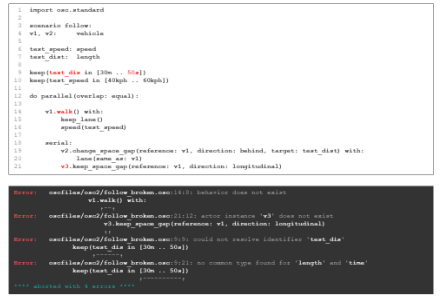
51WORLD OpenSCENARIO2 Grammar Checker (osc2checker)

https://github.com/51WORLD/osc2checker

PMSF py-osc2 Framework

https://github.com/PMSFIT/py-osc2

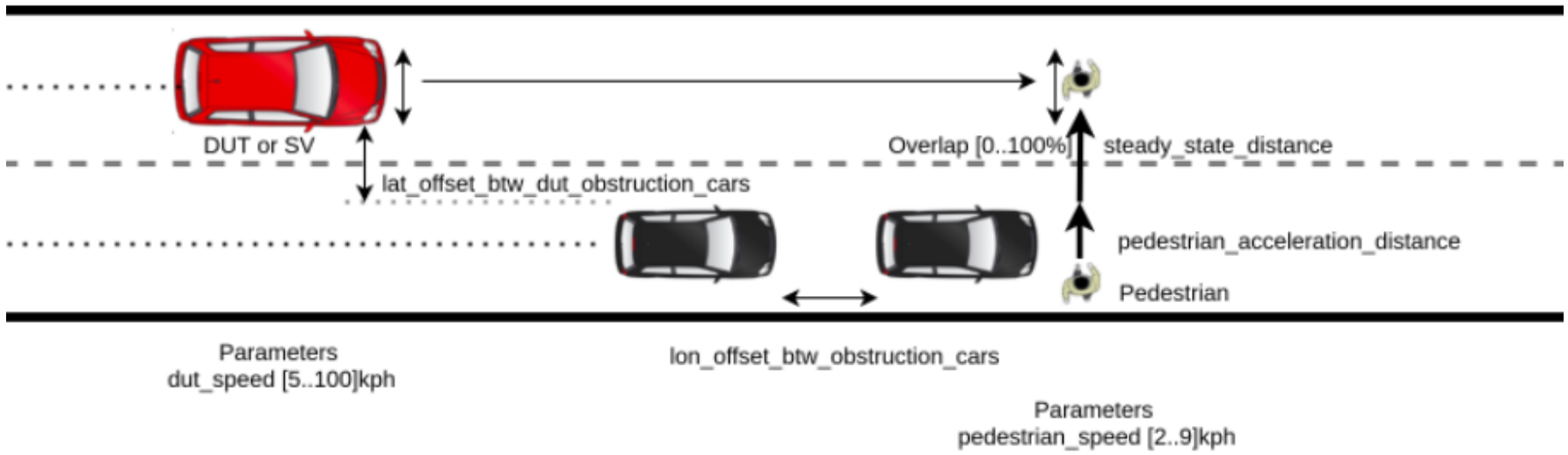
Syntactic and Semantic Checking Example



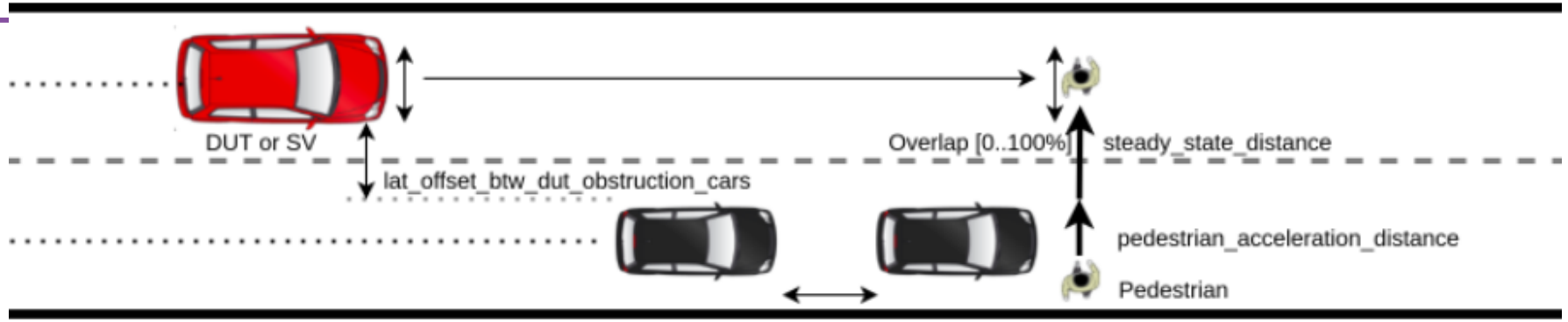
ASAM

Example: NCAP AEB scenario

- ASAM OpenSCENARIO 2.0.0 EXAMPLE – NCAP AEB/VRU



NCAP – AEB : Car to Pedestrian nearside child



Configuration	Type	Range	Description
pedestrian_speed	speed	[2..9]kph	Speed of Pedestrian
sut_speed	speed	[5..90]kph	Speed of SUT or SV
steady_state_distance	distance	[0..30]m	Pedestrian travel distance to 50% overlap
overlap	int	[0..100]	The percentage of the pedestrian path that overlaps with the Ego's path
ncap_test_ttc	time	[2..10]s	The start of the test
lane_num	int		DUT's lane referenced from the curb

Easily define various ranges

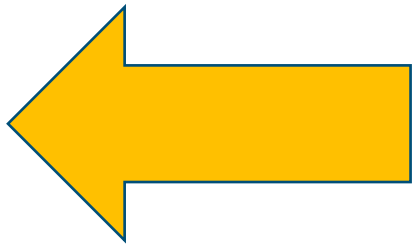
Define Desired testing space
Using constraints

```
scenario sut.aeb_c2p:
    pedestrian: person
    pedestrian_scenario_type: pedestrian_scenario_list
    pedestrian_speed: speed with:
        keep(it in [2..9]kph)
    sut_speed: speed with:
        keep(it in [5..90]kph)
    steady_state_distance: length with: #To 50% overlap
        keep(it in [-30..30]m)
    pedestrian_acceleration_distance: length with:
        keep(it in [-10..10]m)
    overlap: int with:
        keep(it in [0..100])
    lane_num: int with:
        keep(it in [1,2]) #SUT's lane referencing from the curb
    ncap_test_ttc: time with: #As per NCAP's document
        keep(it in [2..10]s)
```

```
# Copyright (c) 2019-2020 Foretellix Ltd. All Rights Reserved.
# ------------------------------------------------------------
# sut::aeb_c2p
# In this scenario, the dut starts driving on a road where it reaches a
# pedestrian vicinty at different speeds and overlaps.
# Document - Euro NCAP AEB c2p test protocol - v 3.0.3
# Scenario - All pedestrians scenarios from the document

# In this scenario you may constrain specifically:
#     ped_speed - Pedestrian speed range.
#     sut_speed - Specific SUT speed range.
#     ped_move_distance - Pedestrian total move distance.
#     overlap - SUT and pedestrian overlap percentage (0-100%).
#     ped_direction - Pedestrian placement on either side of the SUT or SV.
#     lane_num - SUT's lane on the road referenced from curb.
```

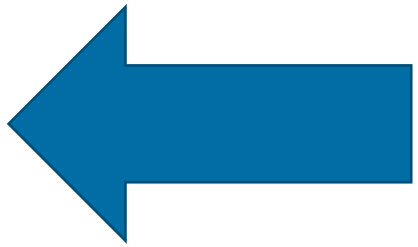
The basic scenario – expressed using parameters.

```
do parallel(start_to_start:0s, overlap:any):
   sut_drive: serial:
      sut_driver.request_AEB_mode(state: active)
      sut_test_drive: sut.car.drive() with:
         speed(sut_speed, at: all,gen_mode:gen_only)
         lateral(distance: 0m, line: center, at: all)
         lane(lane_num, from: curb, at: all)
         avoid_collisions(false)
         distance((sut_to_pedestrian_start_lon_distance+10m),gen_mode:gen_only)
   pedestrian_drive: serial():
      wait @test_start
      wait elapsed(time_to_trigger)
      pedestrian_acceleration: pedestrian.move(pedestrian_acceleration_pos,
                                 start_at_speed: false,
                                 duration: pedestrian_acceleration_time,
                                 start_position: pedestrian_start_pos)
      pedestrian_final: pedestrian.move(pedestrian_final_pos, duration:pedestrian_actual_total_move_time)
```



SUT behavior

Pedestrian behavior



Varying the obstacles !

```
scenario sut.aeb_c2p_nearside_child:
  keep(nc.pedestrian.age_group==child)
```

Define the obstacles' characteristics

```
  obs_car_1: stationary_vehicle with:
    keep(it.color == black)
    keep(it.category == sedan)
  obs_car_2: stationary_vehicle with:
    keep(it.color == black)
    keep(it.category == sedan)
  obs_car_1_pos: msp_position
  obs_car_2_pos: msp_position
```

Vary the obstacles

```
  do parallel(overlap:equal):
    obs_car_1_positioning: obs_car_1.exists_at(obs_car_1_pos)
    obs_car_2_positioning: obs_car_2.exists_at(obs_car_2_pos)
    nc: aeb_c2p(pedestrian_scenario_type: nearside)
```

Error Checking and Coverage

Error Checking and reporting

```
on @top.clk if sut.car.vehicle_indicators.aeb_state != engaged and sut.car.get_ttc() < 1.2second:
   call sut_warning(AEB_checks, "AEB failed to engage!")

 on @sut_drive.end if top.sut.car.vehicle_indicators.aeb_state != active
                                  and top.sut.car.vehicle_indicators.aeb_state != engaged:
    call sut_error(AEB_checks, "Requested AEB is not activated")
```

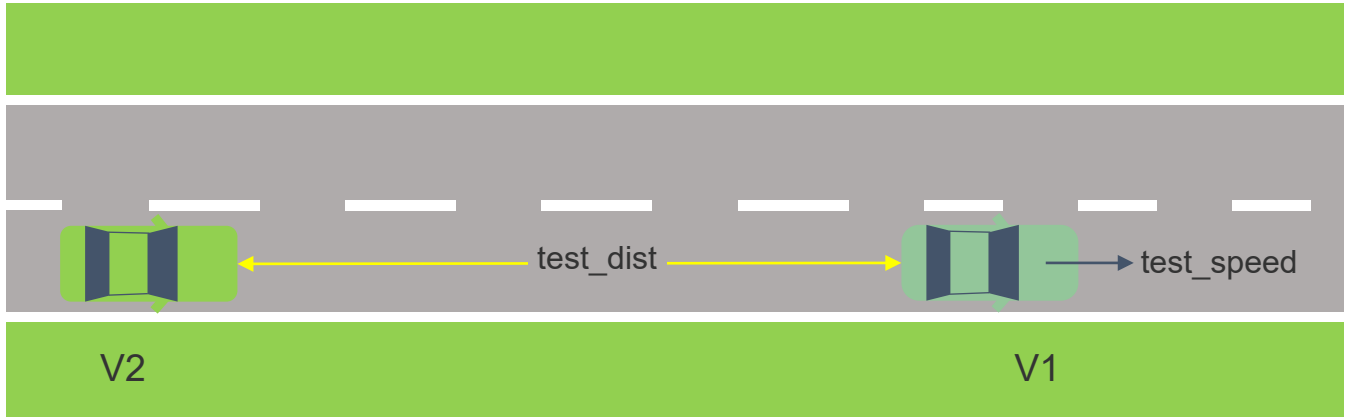
Define and Accumulate Coverage

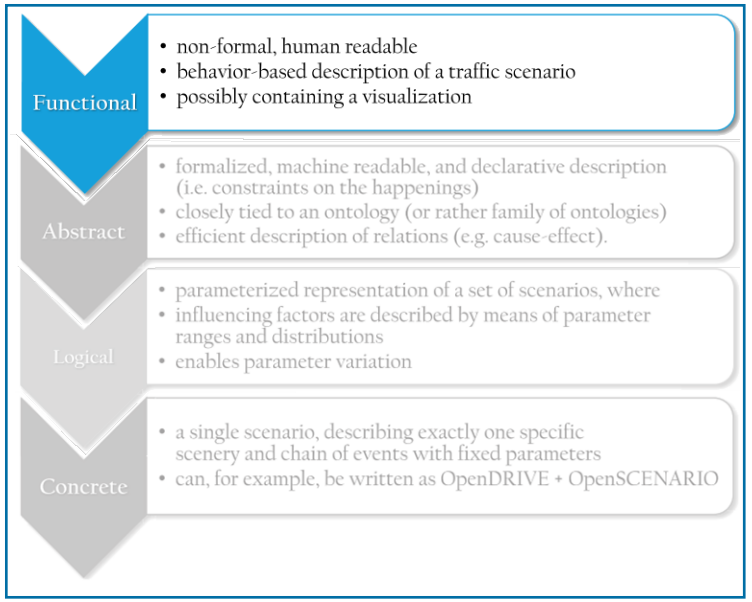
```
var actual_sut_speed := sample(sut.car.state.speed,@sut_test_drive.start)
cover(actual_sut_speed,text:"Actual speed of SUT car"
      ,unit: kph,range: [10..80],every:5)
var sut_min_ttc := sample(get_min_ttc(),@end)
cover(sut_min_ttc, text:"SUT's minimum TTC"
      ,unit: ms,range:[0..3000],every:300)
```

Elements of OpenSCENARIO and Abstraction Levels

Abstraction Levels in OpenSCENARIO V2.0.0

- The scenario describes two vehicles, v1 and v2
- v1 is driving at a constant speed (test_speed)
- v2 is following v1 at a certain distance (test_dist)





Abstraction Levels in OpenSCENARIO V2.0.0

```
# Simple scenario where the Ego vehicle follows
# another vehicle at a constant distance
import osc.standard

scenario follow:
    v1, v2:     vehicle

    test_speed: speed
    test_dist:  length

    do parallel(overlap: equal):

        v1.drive() with:
            keep_lane()
            speed(speed: test_speed)

        serial:
            v2.change_space_gap(reference: v1,
                                direction: behind
                                target: test_dist) with:
                lane(same_as: v1)
            v2.keep_space_gap(reference: v1,
                              direction: longitudinal)
```

standard library import

scenario declaration

actor parameters

physical type parameters

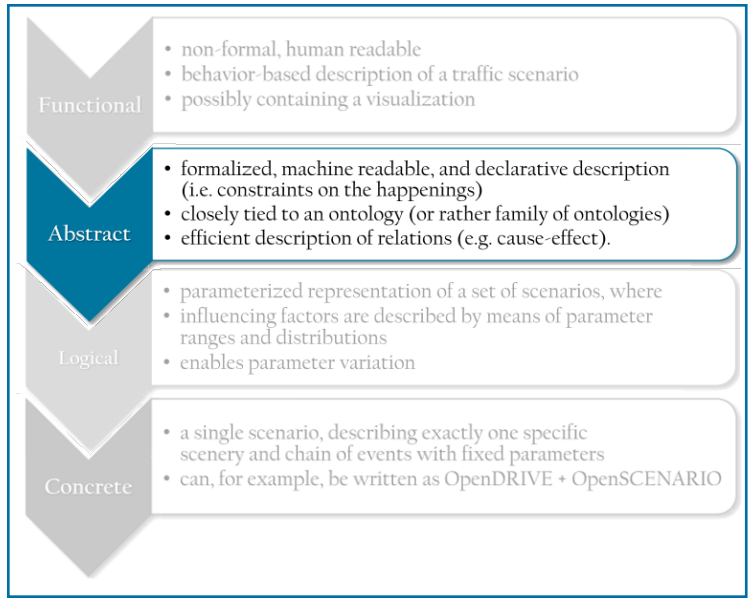
behavior

scenario composition operator

generic action

modifiers

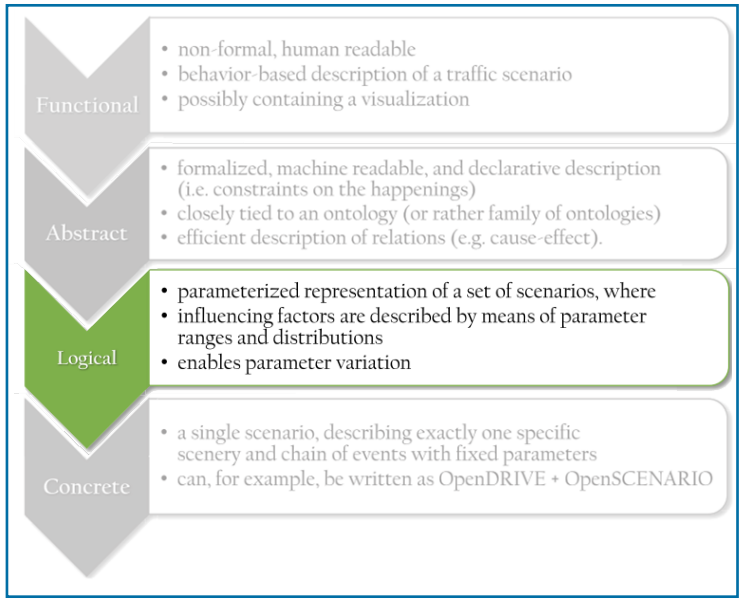
specialized action



Abstraction Levels in OpenSCENARIO V2.0.0

```
# Simple scenario where the Ego vehicle follows
# another vehicle at a constant distance
import osc.standard

scenario follow:
    v1, v2:     vehicle

    test_speed: speed
    test_dist:  length

    keep(test_dist in [30m .. 50m])
    keep(test_speed in [40kph .. 60kph])

    do parallel(overlap: equal):

        v1.drive() with:
            keep_lane()
            speed(speed: test_speed)

        serial:
            v2.change_space_gap(reference: v1,
                                direction: behind
                                target: test_dist) with:
                lane(same_as: v1)
            v2.keep_space_gap(reference: v1,
                              direction: longitudinal)
```

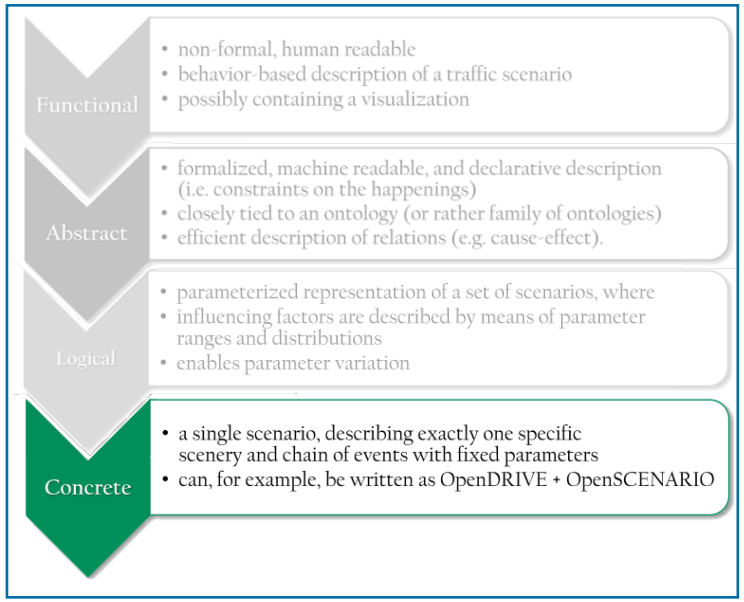
constraints



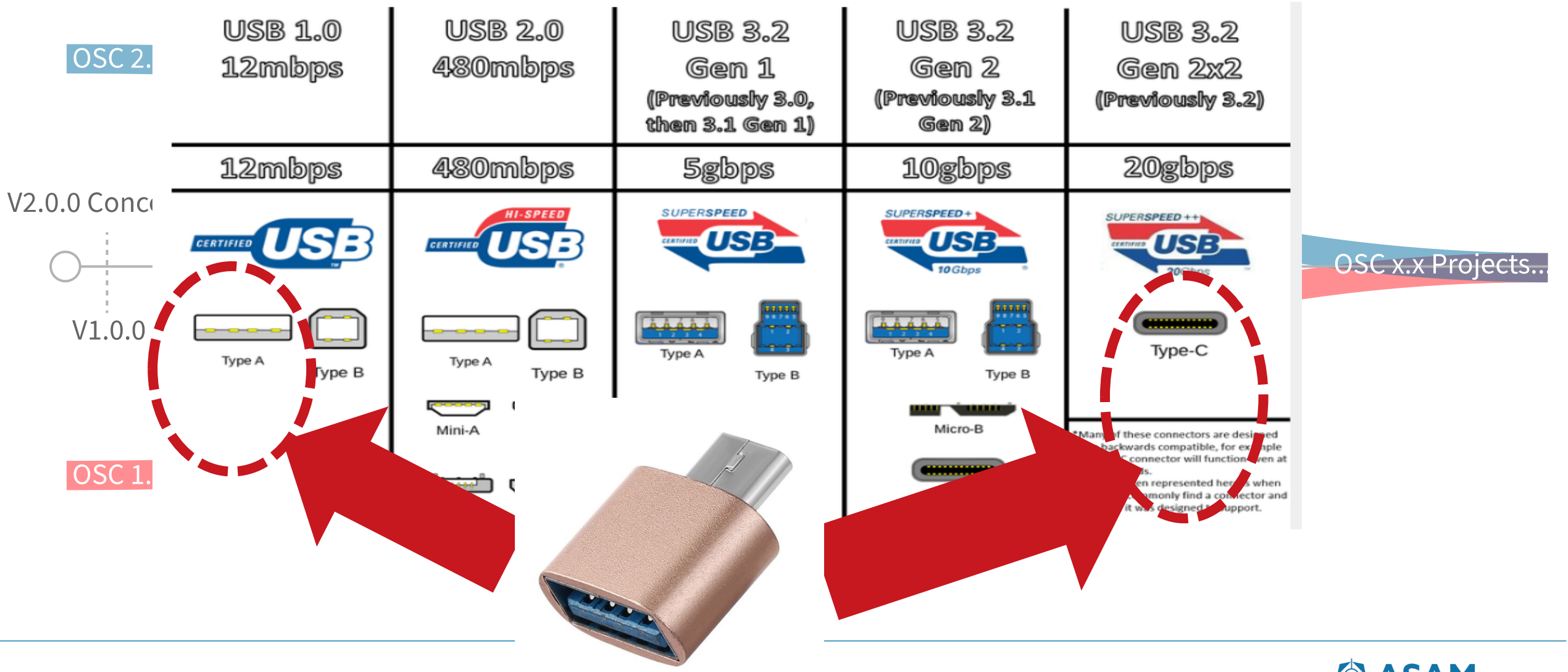
Abstraction Levels in OpenSCENARIO V2.0.0

```
# Simple scenario where the Ego vehicle follows
# another vehicle at a constant distance
import osc.standard

scenario follow:
    v1, v2:     vehicle

    test_speed: speed
    test_dist:  length

    keep(test_dist == 34m)
    keep(test_speed == 55kph)

    do parallel(overlap: equal):

        v1.drive() with:
            keep_lane()
            speed(speed: test_speed)

        serial:
            v2.change_space_gap(reference: v1,
                                direction: behind
                                target: test_dist) with:
                lane(same_as: v1)
            v2.keep_space_gap(reference: v1,
                              direction: longitudinal)
```

constraints



My analogy of the roadmap from USB 1.0 to type-C



ASAM OpenSCENARIO 1.x

What is OSC1 ?

- OpenSCENARIO 1.x comprises the specification and file schema for the description of the dynamic content in driving simulation applications. The primary use of OpenSCENARIO is the description of complex maneuvers that involve multiple vehicles. OpenSCENARIO 1.x is implemented using XML format and a schema.

- OpenSCENARIO 1.x defines the dynamic content of the (virtual) world (e.g. behavior of traffic participants). Static components (such as the road network) are not part of OpenSCENARIO but can be referenced by the format.

- OpenSCENARIO 1.x defines a data model and a derived file format for the description of scenarios used in driving and traffic simulators, as well as in automotive virtual development, testing and validation. The primary use-case of OpenSCENARIO 1.x is to describe complex, synchronized Maneuvers that involve multiple instances of Entity, like Vehicles, Pedestrians and other traffic participants. The description of a scenario may be based on driver Actions (e.g. performing a lane change) or on instances of Trajectory actions (e.g. derived from a recorded driving Maneuver).

ASAM OpenSCENARIO 1.x - Out of Scope

Test configuration description	The standard neither describes the actual test instance nor its structure.
System under test	The exact description of the system under test, e.g. detailed vehicle configuration, sensor placement, sensor models etc. is not part of OpenSCENARIO. [Sensors are candidates for OpenSCENARIO 1.3]
Test case language	Although including a set of driver input, the standard does not attempt to specify all possible user or system interactions with a vehicle.
Test evaluation	Even though the standard includes the evaluation of conditions for triggering actions, there is no concept for creating test verdicts. A scenario end condition can be defined, but not a verdict.
Driver model	As of version 1.0 The standard does not include Driver or Behavioral Driver Model. Controllers were introduces and can be used.
Vehicle dynamics	Although the standard describes maneuvers in a kinematic way, it also defines a very basic vehicle model, which can be used for more realistic vehicle dynamics simulation. The standard does not include all necessary elements to specify advanced motion dynamics.
Road network	The standard does not include elements to describe roads, other than references to an external road network description. The OpenDRIVE standard can be used for this purpose .
3D environment models	The standard only specifies how to refer to external 3D environment models. Further details, like file format or model structure, are not specified.
Environmental models	The standard incorporates elements to specify the current time and weather information but does not describe how this is to be interpreted by the simulator.