OpenSCENARIO 2.0 Concept Project

P2019-02 – Status Update

Gil Amid (with Pierre Mai)
Foretellix Ltd

28. Oktober 2019
München

ASAM Association for Standardization of Automation and Measuring Systems

Agenda

- Background
- Current Status
- Deliverables

Motivation

- AV development and certification requires massive usage of scenario driven simulation.
  Exhaustive simulation is a MUST HAVE for development and qualification of AD and Autonomous driving systems

- OpenSCENARIO 0.9/1.x is in its stabilization phase,
  - during various workshop it became clear there are additional needs, which may not be met by evolution.

- Overall Goal: A standard with all the required features to enable testing and validation of ADAS systems and autonomous vehicles.

- OpenSCENARIO 2.0 should serve as 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.

Technical Content

- OpenSCENARIO 2.0 needs to support:
  - Definition of tests and scenarios for the full development process of autonomous vehicles
  - the full complexity of real-world scenarios, including complex inner-city traff
- Required use cases: span from pure software-based simulation, through SIL, HIL, VIL hybrid testing models, up to test tracks and street driving.
- Concept project focus:
  - Focus on the set of 12 features as defined in the proposal work shop.
  - Define architecture for the main scenario models, and interface to other required models (e.g. Environment, Driver, Traffic)
  - Address varying levels of requirements for parametrization, accuracy
  - Address different use cases of scenarios.

Feature	Type
F001: Maneuver model	Change
F008: High level maneuver descriptions	New
F003: Traffic Model	New
F007: Parameter stochastics	New
F002: Driver Model	New
F004: Environmental Condition Model	New
F009: Replay of Recorded Scenarios	New
F010: Automatic parameter calculation	New
F005: Infrastructure Event Model	New
F006: Vehicle dynamics model	Change
F011: Additional metadata for parameters	New
F012: Language Constructs for Localization	New

General Requirements

- The requirements span over many use cases, and many needs.

TABLE: ISSUE DESCRIPTIONS

ID	Title/Description
R001	Avoid Different Ways to Model
R002	Define Elements as 'Mandatory' Only When Absolutely Needed
R003	Maintain Independence and Open Linking Between Standards.
R004	Define Three Levels of Control for Ego Vehicles.
R005	Allow Tool-Vendor Specific Extensions.
R006	Allow Definition of Feature Subsets
R007	Define Semantics to Enable Reproducibility and Single Interpretation. (Workshop phrasing was: Well Defined Semantics Requirement)
R008	Allow both Open-loop and Closed-loop Simulation by the Same Maneuver Descriptions. (Workshop phrasing: Maneuver Description Shall be Suitable for Open-loop and Closed-loop Simulation)
R009	Define Parameter Boundaries
R010	Synchronize Maneuvers and Events
R011a	Allow Definition of Success Criteria for Individual Maneuvers, and for Full Scenarios and Tests – DUT criteria
R011b	Allow Definition of Success Criteria for Individual Maneuvers, and for Full Scenarios and Tests – non-DUT criteria
R012	Allow Textual Editing of the Format. (Workshop phrasing was: Suitability for textual editing)

Current Status

Key Messages

(Details in next slides)

- Projects includes ~100 engineers from ~50 companies. (about 50% active – attending f2fs)
- Key concepts identified and agreed on by project team
- Project restructured in order to accelerate progress toward meeting schedule.
- New structure launched early October.
- Expecting to meet schedule +/- a month.

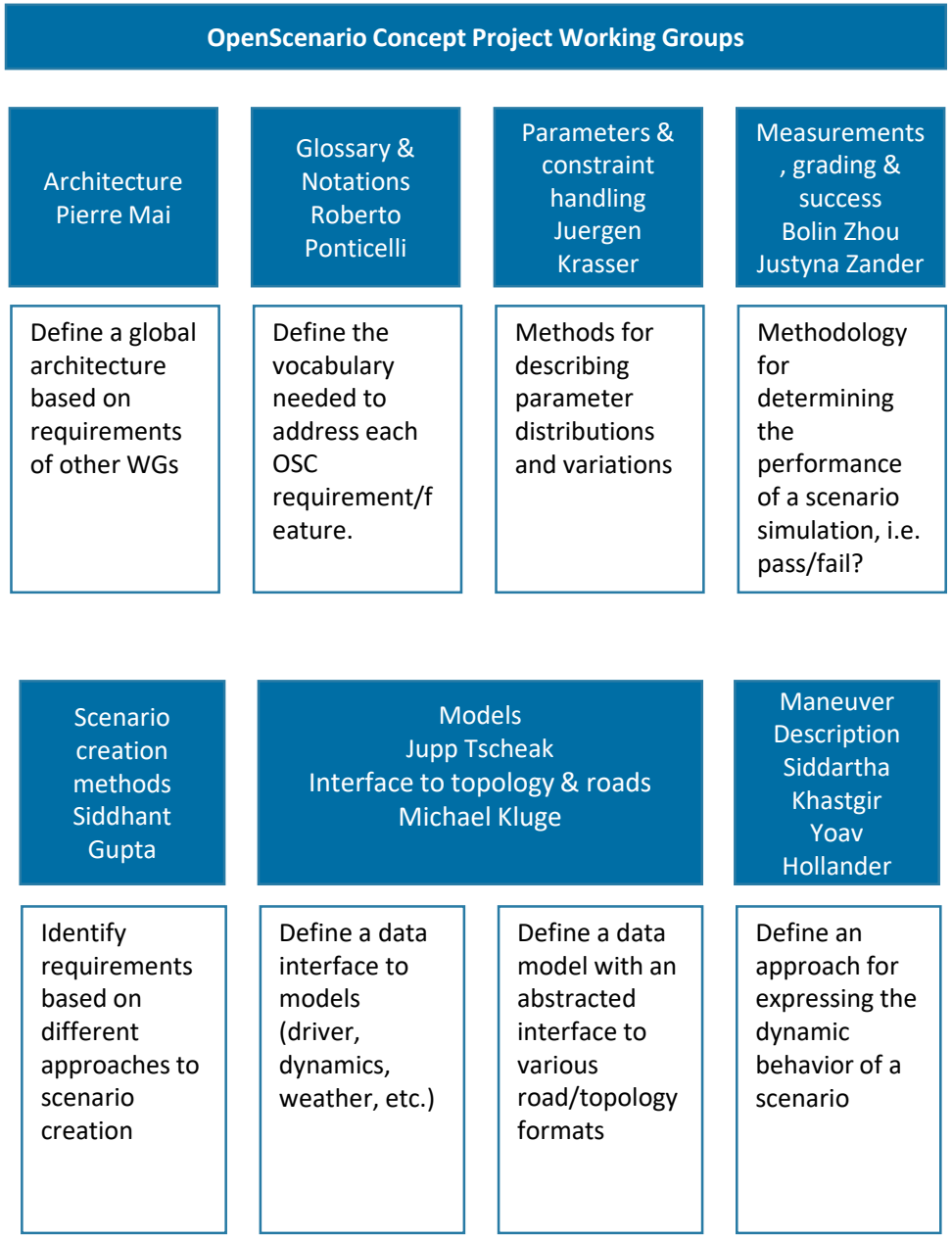
Sep-19/20 F2F Key Decisions

~40 participants in the f2f.

- Project approved a DSL (Domain Specific Language) direction
- Project approved key concepts recommended by the maneuvers work group for the DSL content (Composability of scenarios, constraints, scenario modifiers for abstract scenarios)
- Concepts for parameters, measurements and grading were ratified
- Project decided to use Foretellix's M-SDL as an example language, whenever syntax and examples are required
- Project created a revised outline for the concept document
- Approved restructure to 3 main work groups, with smaller teams/tasks forces to deliver different sections of the concept document.

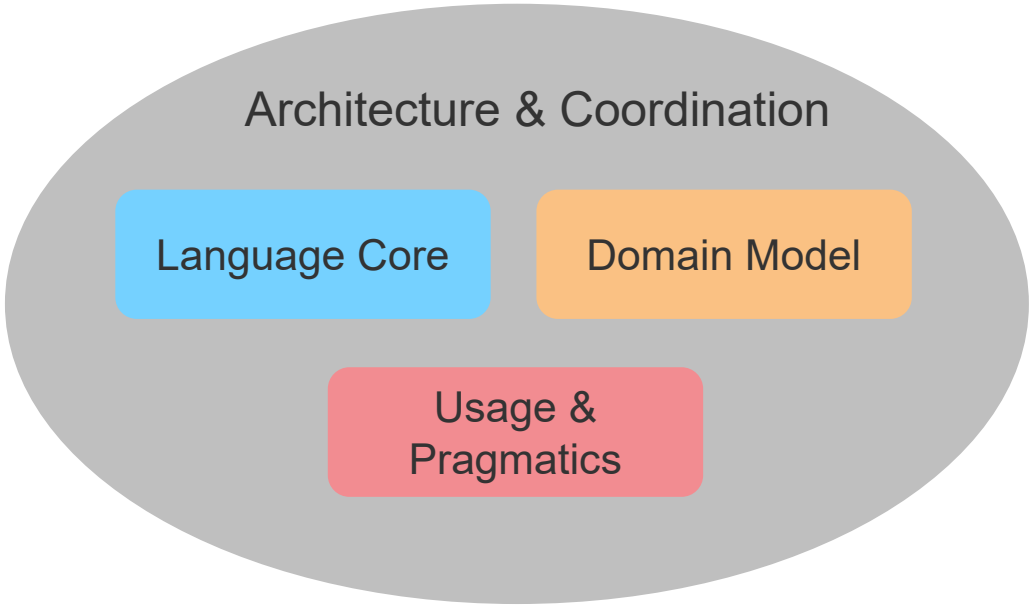
Original Project structure

- 7 Work groups worked in parallel.
- Architecture groups owns overall architecture concept and interface.
- WG leaders meeting serves as synchronization body.



New Project Structure

- 3 main clusters, each coordinating smaller task forces
- Each task force is responsible for a section of the concept document

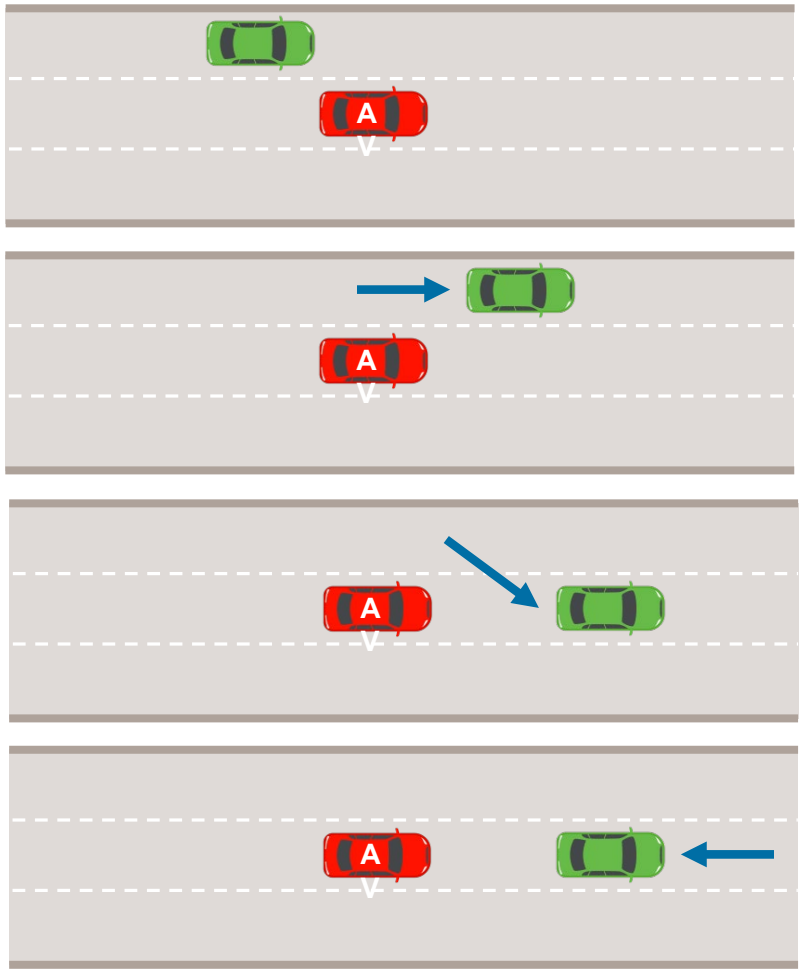


			Overview Documentation
			Prepare the general chapters in the Concept Document
Data Type Definition Language	Basic Parameters and Constraints	(Atomic) Operations and Modifiers	Pragmatics of parametrization
Finalize and document the data types of the language	Discuss and finalize basic semantics of parameters and constraints.	Clarify and tighten semantics of Modifiers	Abstraction levels of scenarios, translation between levels
Entity Definition	Ontology Definition	Essential Terminology	Use case examples
Define entities (actors) via their basic properties, relationships and actions	Describe the basic framework of the ontology	Collect and collate the terminology and definitions used across the WGs	Define multiple real-world use cases using multiple abstraction levels.

Some Examples

M-SDL example: cut_in_and_slow

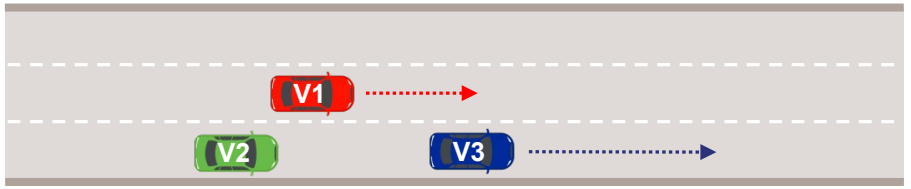
```
scenario dut.cut_in_and_slow:

    car1: car                          # The other car
    side: av_left_right                # A side: left or right
    path:  path                        # A path in the map
    path_min_lanes(path, 2)            # Path should have at least two lanes

    do serial:
        get_ahead: parallel(duration: in [1..5]s):
            dut.car.drive(path)  with:
                speed([30..70]kph)
            car1.drive(path, adjust: TRUE) with:
                position([5..100]m, behind: dut.car,at: start)
                position([5..15]m, ahead_of: dut.car, at: end)

        change_lane: parallel(duration: in [2..5]s):
            dut.car.drive(path)
            car1.drive(path) with:
                lane(side_of: dut.car, side: side, at: start)
                lane(same_as: dut.car, at: end)

        slow: parallel(duration: in [1..5]s):
            dut.car.drive(path)
            car1.drive(path) with:
                speed_change(-[10..15]kph)
```



Using modifiers to control scenario dynamics

- Modifiers are like constraints but more general. Examples:
- Control movements via *movement* modifiers
  - v1 drives 10..20 kph faster than v2:

    v1.drive() with: speed([10..20]kph, faster_than: v2)



- Control the scenario's location via *path* modifiers
  - Path (road) p should have at least 2 lanes:

    path_min_lanes(p, 2)

- Control *synchronization* between events
  - Sync these two events to within -1..1 second of each other:

    synchronize(phase_a.end, phase_b.start, [-1..1]s)

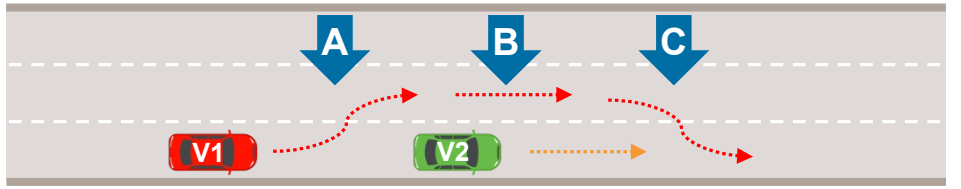
- You can use *any number* of modifiers in the same invocation
  - E.g. to express the complex situation on the right

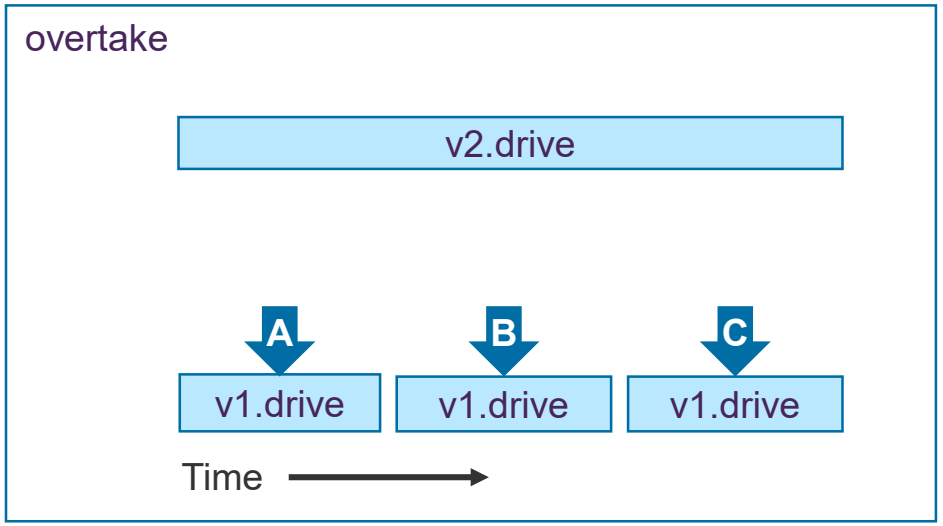
```
v3.drive(p) with:
    lane(right_of: v1)
    speed([7..15]kph, faster_than: v1)
    position([20..70]m, ahead_of: v1)
    position([10..30]m, ahead_of: v2)
    lane(same_as: v2)
    lateral([10..25]cm, left_of: v2)
```

Composition: Writing a full scenario

- Here is the full *overtake* scenario

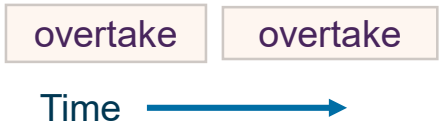
```
scenario traffic.overtake:
    v1: car # The first car
    v2: car # The second car
    p: path

    do parallel(duration: [3..20]s):
        v2.drive(p)
        serial:
            A: v1.drive(p) with:
                position([10..20]m, behind: v2, at: start)
                lane(same_as: v2, at: start)
                lane(left_of: v2, at: end)
            B: v1.drive(p) with:
                position([1..10]m, ahead_of: v2, at: end)
            C: v1.drive(p) with:
                lane(same_as: v2, at: end)
                position([5..10]m, ahead_of: v2, at: end)
```





- You can then compose this scenario using e.g. *serial*

```
scenario overtake_serial
    car_a: car
    car_b: car
    do serial:
       overtake(v1: car_a, v2: dut.car)
       overtake(v1: car_b, v2: dut.car)
```

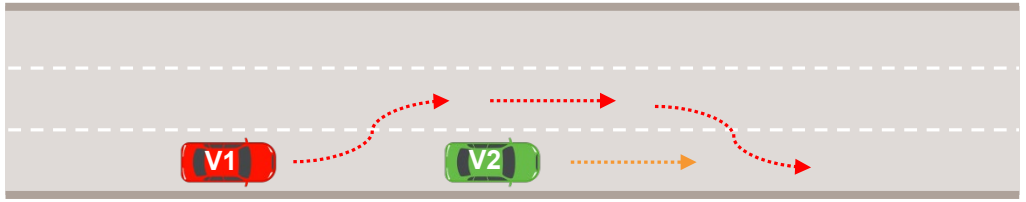


Scenario invocation syntax

- Scenario name
  - scenario operators
    serial: … parallel: … first_of: … one_of: … mix: … repeat: …
  - atomic scenarios (actions)
    drive() … walk() … wait …
  - user-defined scenarios
    overtake() … cut_in() …
- Scenario invocation

  *[label:] [path.]name(parameter, …) [*with: *modifier …]*

  - label is optional
    d: drive(…) … or drive(…) …
  - *path* is optional
    dut.car.drive(…) … or drive(…) …
  - *parameter* can be by name or by position
    drive(path) or drive(path)
  - *modifier* is similar to scenario invocation
    speed(5 kmh, faster_than: car1)




```
scenario traffic.overtake:
    v1: car # The first car
    v2: car # The second car
    p: path
    keep(v1.color != green)

    do parallel(duration: [3..20]s):
        v2.drive(p)
        serial:
            A: v1.drive(p) with:
                lane(same_as: v2, at: start)
                lane(left_of: v2, at: end)
                position([10..20]m, behind: v2, at: start)
            B: v1.drive(p)
            C: v1.drive(p) with:
                lane(same_as: v2, at: end)
                position([5..10]m, ahead_of: v2, at: end)
```

```
import sumo_config.sdl # Execution platform
import lane_change_scenarios.sdl # Library

extend top.main:  # Extend the predefined main
    set_map("some_map.xodr") # Map to use in test
    do overtake(v2: dut.car)
```

Example: Writing a concrete scenario

- So far, we wrote an abstract scenario, then constrained it “from above”

```
scenario traffic.overtake:
    v1: car
    …
    do parallel(duration: [3..20]s):
        …   position([10..20]m, behind: v2, at: start)
```

Some lines from the original abstract scenario: Note the ranges

- We can write a concrete scenario “from scratch”

```
scenario traffic.concrete_overtake:
    v1: car:
    keep(v1.color == green)
    keep(v1.category == truck)
    …
    do parallel(duration: 7second):
       … position(10.5m, behind: v2, at: start)
       … speed(18.7kph)  # Note that speed was not
mentioned
```

Same lines if we want to write a concrete scenario from the start

Example: Driver-in-the-loop

Time

```
scenario dut.near_hit:
    do one_of():
        turn_right_plus(v2: dut)
        overtake(v2: dut)
        car_ignoring_red_light()
        …
```

This scenario will cause a random near hit situation

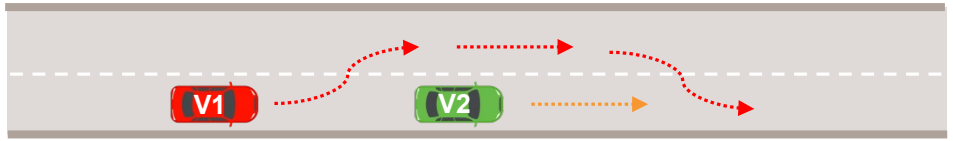
```
scenario dut.DIL_multi_near_hit:
    how_long: time # How long to run it

    do run_time(duration: how_long):
        dut.car.drive(duration: how_long) # Drive the dut
        repeat():
            wait_time([5..20]s) # Let him relax a bit
            near_hit() # Plan the next near-hit
```

This scenario will repeatedly wait some seconds and then plan and execute another random near-hit

Concrete to abstract

```
scenario traffic.overtake:
    v1: car # The first car
    v2: car # The second car
    p: path

    do parallel(duration: [3..20]s):
        v2.drive(p)
        serial:
            A: v1.drive(p) with:
                lane(same_as: v2, at: start)
                lane(left_of: v2, at: end)
                position([10..20]m, behind: v2, at: start)
            B: v1.drive(p)
            C: v1.drive(p) with:
                lane(same_as: v2, at: end)
                position([5..10]m, ahead_of: v2, at: end)
```

This is a more abstract version of overtake



v1	A (to left)	B (straight)	C (to right)
v2	straight		

Time

This is a very concrete version of overtake

```
scenario traffic.overtake_concrete:
    v1: car with(category: sedan, color: black)
    p: path
    path_explicit(p,[point("15",30m), point("95",1.5m), …])

    do parallel(duration: 10s):
        dut.car.drive(p) with:
            lane(2)
            speed(50kph)
        serial:
            A: v1.drive(p, duration: 3s) with:
                speed(70kph)
                lane(2, at: start)
                lane(1, at: end)
                position(15m, behind: dut.car, at: start)
                position(1m, ahead_of: dut.car, at: end)
            B: v1.drive(p, duration: 4s) with:
                position(5m, ahead_of: dut.car, at: end)
            C: v1.drive(p, duration: 3s) with:
                speed(80kph)
                lane(2, at: end)
                position(10m, ahead_of: dut.car, at: end)
```

```
scenario traffic.overtake_dut
    do overtake(v2: dut.car) with:
        keep(it.A.duration == 3s)
```

This scenario invokes overtake with some parameters

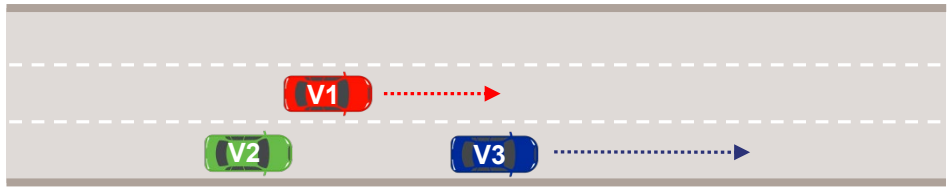
```
scenario traffic.overtake_serial
    car_a: car
    car_b: car
    do serial:
        overtake(v1: car_a, v2: dut.car)
        overtake(v1: car_b, v2: dut.car)
```

This scenario does two overtakes serially

```
scenario traffic.overtake_repeat
    do repeat(count: 10):
        wait_time([10..20]s)
        overtake_serial
```

This scenario repeats overake_serial 10 times

Multiple, independent movement constraints



This is phase A
Note the relations (speed, position, lateral offset etc.)
between v3 and the other cars

Phase A | Phase B

```
scenario traffic.multi_car:
    v1: car # The first car
    v2: car # The second car
    v3: car # The third car
    p: path

    do serial:
        A: parallel(duration: [3..20]s):
            v1.drive(p) with: …
            v2.drive(p) with: …
            v3.drive(p) with:
                lane(right_of: v1)
                speed([7..15]kph, faster_than: v1)
                position([20..70]m, ahead_of: v1)
                position([10..30]m, ahead_of: v2)
                lane(same_as: v2)
                lateral([10..25]cm, left_of: v2, measured by: center to center)
        B: parallel(duration: [3..20]s):
            v1.drive(p) with: …
            v2.drive(p) with: …
            v3.drive(p) with: …
```

Here is how you say that. Note that we need here six movement constraints (modifiers), each with its own set of parameters.

Using event-based synchronization

```
scenario traffic.multi_car:
    v1: car # The first car
    v2: car # The second car
    p: path

    event e1 is map.reach_position(v2, point1)
    event e2 is map.reach_speed(v2, 40kph)
    …

    do parallel:
        v1_part: serial:
            wait @e1
            v1.drive(p) with:
                speed(…)
                position(…)
                on @e2: end()
            v1.drive(p) with:
                speed(…)
                position(…)
                on @e3: end()
         v2_part: serial:
            v2.drive(p) with:
                speed(…)
                position(…)
                on @e4: end()
            v2.drive(p) with:
                speed(…)
                position(…)
                on (v1.distance_to(point2) < 7): end()
            v2.drive(p) with:
                speed(…)
                position(…)
            …
```

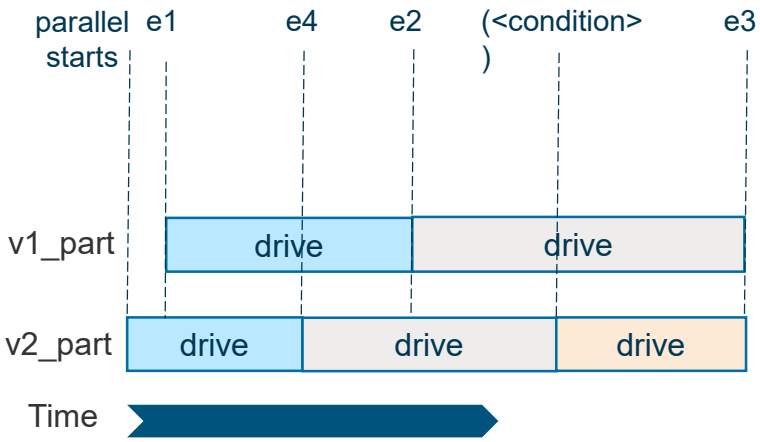
Events represent moments in time. Can be defined using any condition(or other events)

Define the whole scenario as “parallel of serials”

End each step of the serial upon some event

Can also specify the condition inline

parallel of serials

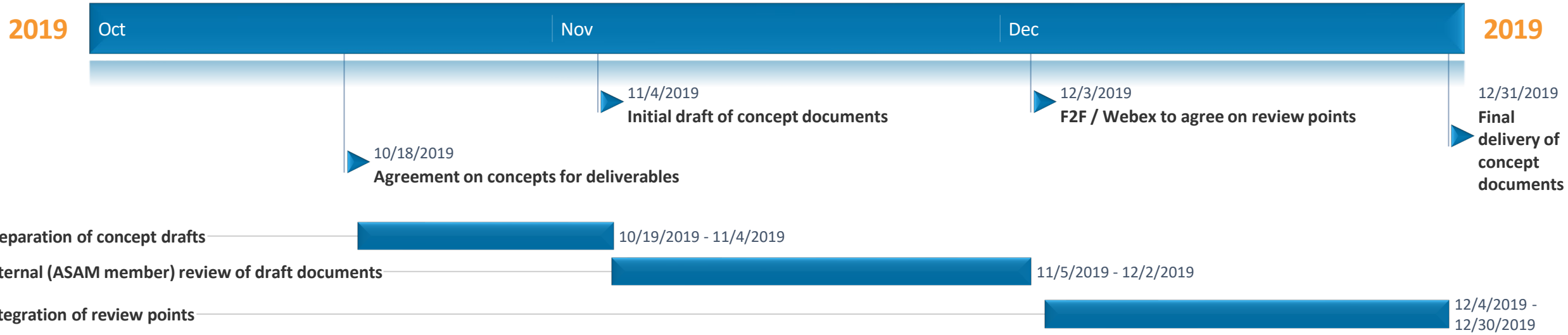


Note that this whole parallel-of-serials can still be a lego brick in something bigger

```
scenario traffic.multi_car_plus
    do serial:
       multi_car(v2: dut.car)
       if (dut.car.distance_to(point3) < 10):
           repeat(count: 3):
              overtake_serial
```

Deliverables

Current schedule



Deliverables

- The overall expected outcome of the project is a concept document

I	Foreword
1	Introduction
1.1	Overview
1.2	Problem Statement & Motivation
1.3	Intended Audience
1.3.1	Standard Developers
1.3.2	ASAM Members
1.4	Relations to Other Standards
1.4.1	Backward Compatibility
1.4.2	References to Other Standards
2	Scope
2.1	Goals
2.2	Non-Goals
2.3	Methodology – Declarative Language
3	Key Terminology
4	Use Cases

5	Architecture
5.1	Domain Model
5.2	Interface Description
5.3	Dependencies and Interrelations
5.4	Ontology
5.5	Relationship Diagram / Ontology Nodes
5.6	Class Reference
6	Language Concepts
6.1.1	Concepts / Value Proposition
6.1.2	Parameters
6.1.3	Maneuvers
7	Usage and Pragmatics
7.1.1	Translation of Intent into Implementation
7.1.2	Library Concepts and Packaging
7.1.3	Scenario Creation
7.1.4	Measurement and Success Criteria
7.1.5	Usage Restrictions
7.2	Roadmap
A	Glossary of Terms
B	Language Constructs (Reference Manual)
C	Syntax

Progress to date

- Initial assignment of task force members expected to complete in the week before the TSC
- Set up first meetings for each task force ASAP after this to ensure a running start
- First/partial drafts of each section expected by next F2F in December

Relation to ISO and other standartization activities

- Interact and align with ISO TC 22/SC 33/WG 9
  “Test Scenario of autonomous driving vehicles”
- Carlo Van-Driesten, Gil Amid, Siddartha Khastgir, Siddhant Gupta are members of this WG.
- A communication and synchronization mechanism is being discussed these days (-> Category C Liaison).
  Approved by the WG, need to be approved by the SC
- Focus of WG 9 on architecture and framework, potential to specify ASAM OpenScenario as relevant format standard.

- X-membership with ISO’s SOTIF (WG08) WG.
- X-membership with UL-4600 Stake holders review cycle.
- X-Membership with BSI PAS-1881
- X-membership with UNECE/GRVA
- X-membership with SAE’s ORAD committee and task forces.

Backup slides

Thank you for your attention!

Gil Amid
Foretellix Ltd

Phone: +972-58-4347475
Email:gil.amid@Foretellix.com

ASAM