

OpenSCENARIO 1.x

Proposal to significantly simplifying runtime implications
by Andreas Hege, Stephan Meichle

2020/02/10

Runtime Model and implication

Why do we need a runtime model?

The OpenSCENARIO runtime model is the conceptual idea about what is happening with a crafted scenario when being executed.

The 1.0 Situation

Complicated runtime implications

- **Many rules**
Must be extensively documented. Hard to be kept consistent.
- **Context-sensitivity**
To get the rules consistent, many special cases must be defined. Many constraints, e.g. when an action is allowed to be used etc. => more rules.
- **High amount of unnecessary complication**
 - When does a simulator behaves the “right way”? => simulator provider
 - When is a scenario defined the “right way”? => author

The 1.0 Situation

Complicated conceptual modelling

- Complicated interactions between storyboard elements (e.g. actions), entities, controllers.
- Concept of never ending actions implies never ending events, never ending maneuvers, maneuver groups, acts, stories and scenarios.
- The same action type sometimes returns immediately and sometimes does “never” return, depending on the action properties.
- There is no simple answer on each of the following questions.
 - When does an action ends? See complex action table.
 - How does the actions interact with entities and other actions?
 - Is there any action allowed in the initialization phase that does not end “immediately”? See esmini examples.

The 1.0 Situation

Expert discussions

RE: Question about storyboard states - Nachricht (HTML)

RE: Question about storyboard states

Von: Stracabosko, Daniel AVU/HR <Daniel.Stracabosko@avl.com>
 Gesendet: Donnerstag, 6. Februar 2020 15:55
 An: Andreas Hege | RA Consulting <a.hege@rac.de>
 Cc: Ludwig.Friedmann@bmw.de; Benjamin Engel <benjamin.engel@asam.net>; Bernd Wenzel <wenzel@meskom.de>
 Betreff: RE: Question about storyboard states

Hi Andreas,

This is all ok for "End" transitions but I was talking about "Stop" transitions and maxExecutionCount is really important in this case because we have defined "Complete state" using the maxExecutionCount.

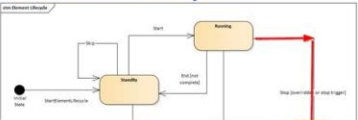
Let me guide you through the details.

If we have defined maxExecutionCount and that number hasn't been reached then we are not in "Completed" state. This is what the definition of "Complete" state says.

Complete State	<p>The completeState signals that the runtime instantiation of the StoryboardElement, cannot reach a running state without external interference. If the affected runtime instantiation of the StoryboardElement is defined with a maximumExecutionCount, to be Complete implies that there are no more executions left to run. Checking for completeness involves verifying if the given runtime instantiation of the StoryboardElement, still has executions left upon finishing the runningState. This check returns false if there are executions left. This check returns true if there are no executions left, or if the maximumExecutionCount is not defined in the StoryboardElement.</p> <p>Resetting the completeState can only be achieved externally by the parent StoryboardElement whose child is in the completeState. This may only occur if the parent initiates a new execution.</p>
-----------------------	---

Let's follow this execution:

- We have an Event that has maxExecutionCount=5
- This event has its state in "Running" and has been executed less than 5 times
- During Event's execution a stopTrigger becomes true on Act (to which Event belongs) which sends StopTransition to Act and all its children
- Event then follows marked transition on the image



The 1.x* Situation

Conceptual issues

- **Enhancing the conceptual runtime model**
Giving the authors a better and more simplified idea what happens during execution.
- **Sharpening the system boundaries**
Provide an clean idea what is in the responsibility of the standard and what is out of scope. Create abstractions and define interfaces to external systems (with OpenSCENARIO concept group).

*x>0

The 1.x Situation

What we need.

We need a **few** very **simple** rules that are **always true**. They should describe the concurrent interactions between entities, actions, controllers during runtime.

We can explore the interaction by inferring further rules from these few rules.

The 1.x Workgroup

This is a call to volunteer

A subgroup in a future OpenSCENARIO 1.x working group would be an excellent choice to work out the details of a proposal.

Concurrency

FLOW AND SIMULATION

Explicit Process Abstraction

Simulation and flow.

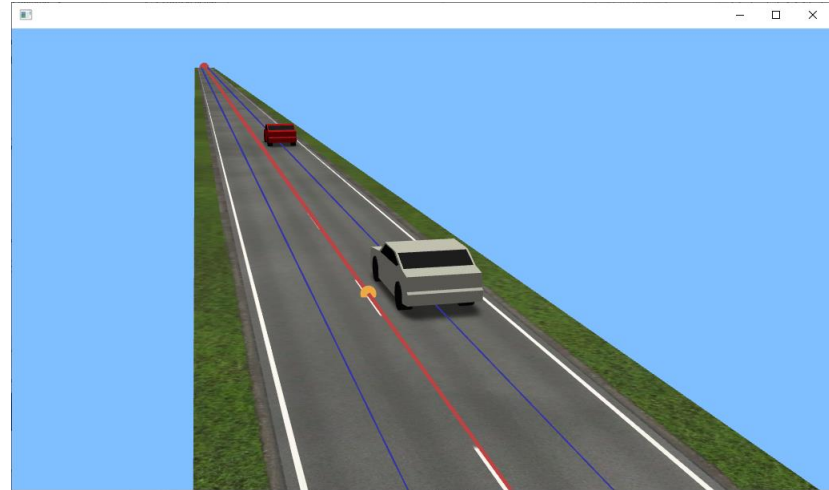
- **Two major processes**
From the view of the standard there are two major processes. A simulation process and a flow process.
- **Synchronization**
These processes are synchronized by the describing scenario.

esmini

Representation of the processes

```

Eingabeaufforderung - VerySimpleExample.bat
OSAction.hpp / 65 / scenarioengine::OSAction::Stop(): Action Init OverTaker Position (Private) stop
ped
ScenarioGateway.cpp / 136 / scenarioengine::ScenarioGateway::reportObject(): Adding Ego state: (0, 0,
0.00)
ScenarioGateway.cpp / 136 / scenarioengine::ScenarioGateway::reportObject(): Adding OverTaker state:
(1, 0.00)
void StateSet::setGlobalDefaults() ShaderPipeline disabled.
void StateSet::setGlobalDefaults() ShaderPipeline disabled.
viewer.cpp / 64 / FindNamedNode::apply(): Found wheel_fl
viewer.cpp / 64 / FindNamedNode::apply(): Found wheel_fr
viewer.cpp / 64 / FindNamedNode::apply(): Found wheel_rr
viewer.cpp / 64 / FindNamedNode::apply(): Found wheel_rl
NodePath 0
Camera
MatrixTransform
PositionAttitudeTransform
LOD
viewer.cpp / 113 / viewer::CarModel::AddWheel(): Found no wheel node wheel_fl in vehicle model car_re
d.osgb, ignoring
viewer.cpp / 113 / viewer::CarModel::AddWheel(): Found no wheel node wheel_fr in vehicle model car_re
d.osgb, ignoring
viewer.cpp / 113 / viewer::CarModel::AddWheel(): Found no wheel node wheel_rr in vehicle model car_re
d.osgb, ignoring
viewer.cpp / 113 / viewer::CarModel::AddWheel(): Found no wheel node wheel_rl in vehicle model car_re
d.osgb, ignoring
OSCondition.cpp / 479 / scenarioengine::TrigBySimulationTime::Evaluate(): Trigged CutInActStart (sim
Time: 0.10 >= condition: 0.00 result: 1 last_result: 0 edge: Rising
story.hpp / 72 / scenarioengine::Act::Trig(): Act CutInAndBrakeAct trigged
OSCondition.cpp / 543 / scenarioengine::TrigByTimeHeadway::Evaluate(): Trigged CutInStartCondition h
wt: 1.00 > 1.00. Rising
OSAction.hpp / 58 / scenarioengine::OSAction::Trig(): Action CutInAction (Private) trigged
OSManeuver.cpp / 24 / scenarioengine::Event::Trig(): Event CutInEvent trigged
OSAction.hpp / 65 / scenarioengine::OSAction::Stop(): Action CutInAction (Private) stopped
OSManeuver.cpp / 30 / scenarioengine::Event::Stop(): Event CutInEvent stopped
    
```



FLOW PROCESS

SIMULATION PROCESS

Explicit Process Abstraction

Making it explicit

It becomes very obvious that these two processes really exist when the flow process is suspended (e.g. waits for a start trigger) while the simulation process is still running (vehicles are moving).

Explicit Process Abstraction

Example

1. Two vehicles are moving along their lane with a constant speed.
2. Ego is moving on the right lane. Red vehicle is moving on a left lane.
3. Red vehicle runs faster and passes Ego.
4. As soon as a headway condition becomes true, the red vehicle cuts in.
5. The simulation ends after lane change.

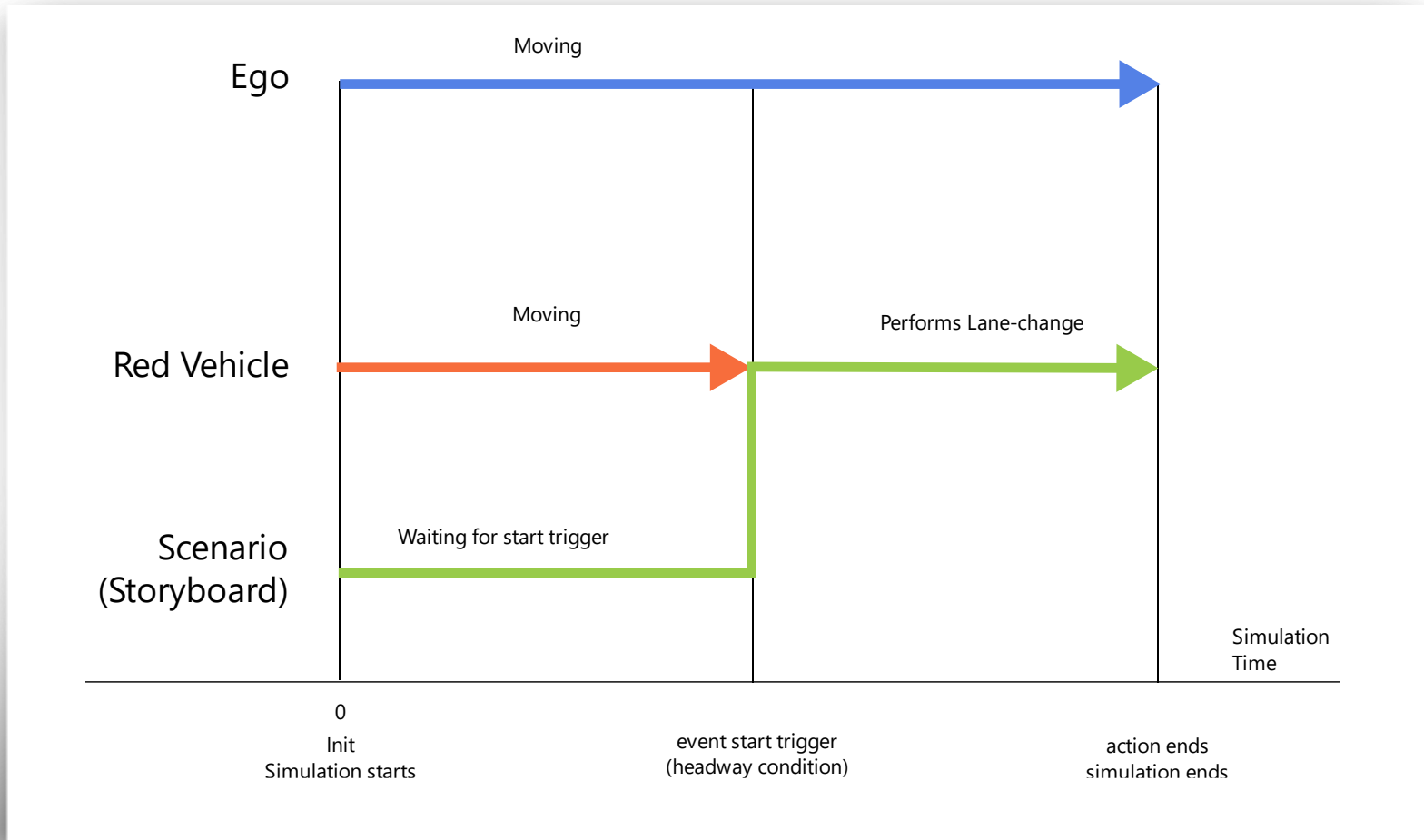
Example

Implementation

1. Ego and vehicle are initialized with their position and their speed.
2. One event is implemented that waits for the headway condition (start trigger).
3. The lane change action is performed within the event.

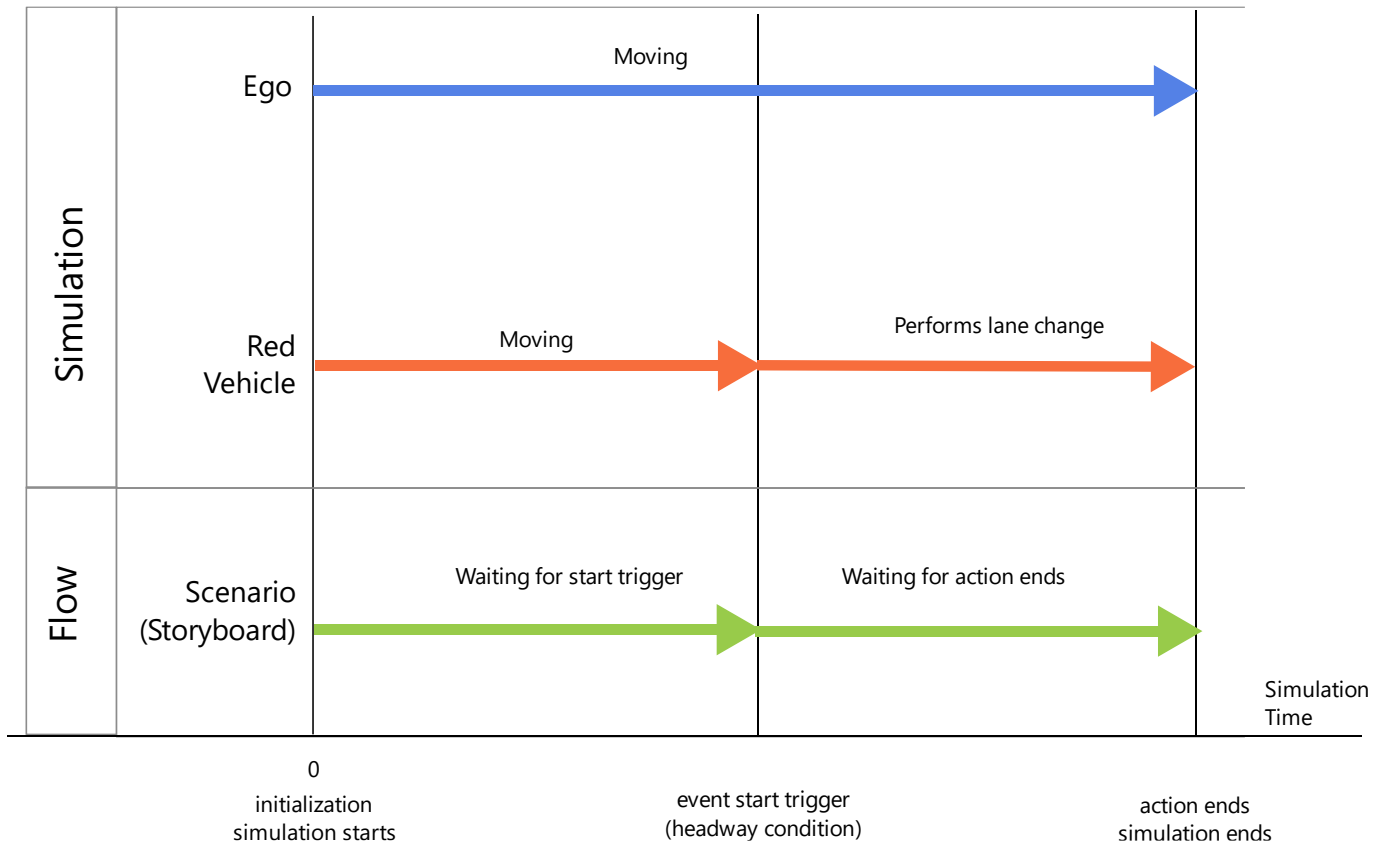
Runtime Interpretation

Current interpretation



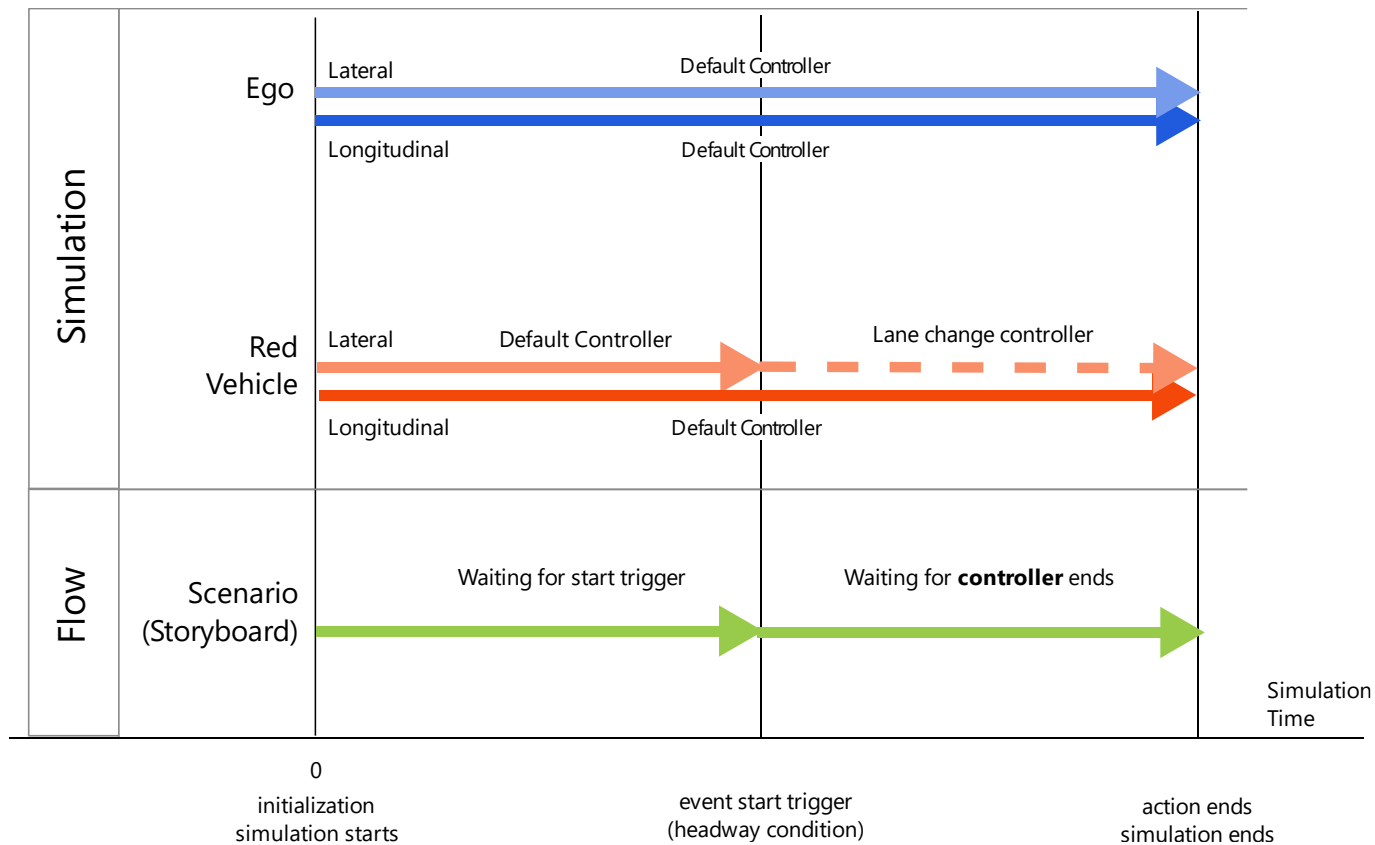
Runtime Interpretation

Abstract process interpretation



Runtime Interpretation

Abstract controller interpretation



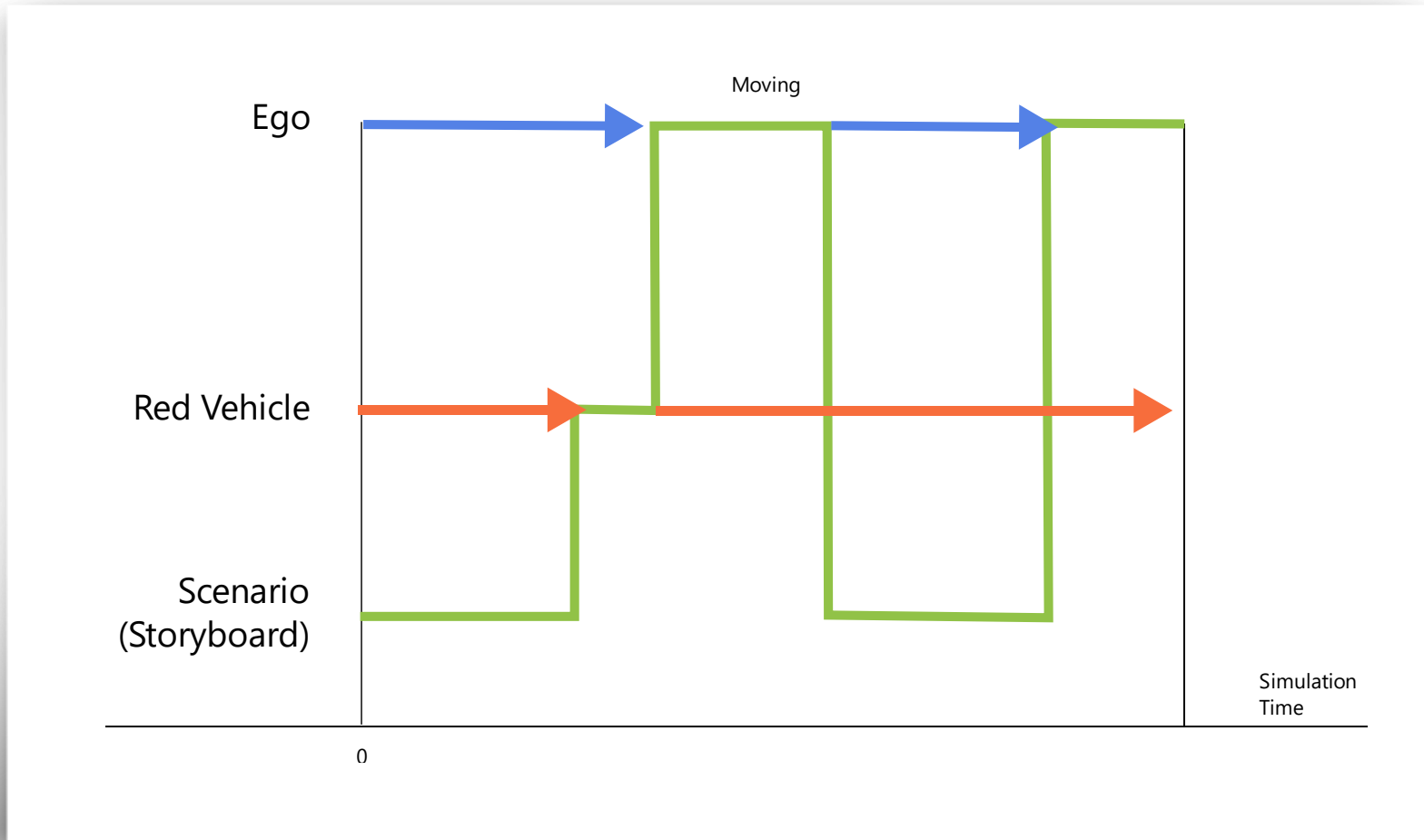
First step

Sketch of a simplified system

1. A set of orthogonal subsystems.
 - Flow process and simulation process are orthogonal.
 - Entity instances are orthogonal (independent). Like in real traffic.
 - Lateral and longitudinal control dimensions are orthogonal.
 - Controllers are orthogonal.
2. Clear system boundaries
 - Runtime instances like entities, controllers, traffic signal controllers exclusively exist in the simulation process.
 - Runtime instances of storyboard elements exclusively exist in the flow process.
 - Runtime instances in both processes are communicating with messages.

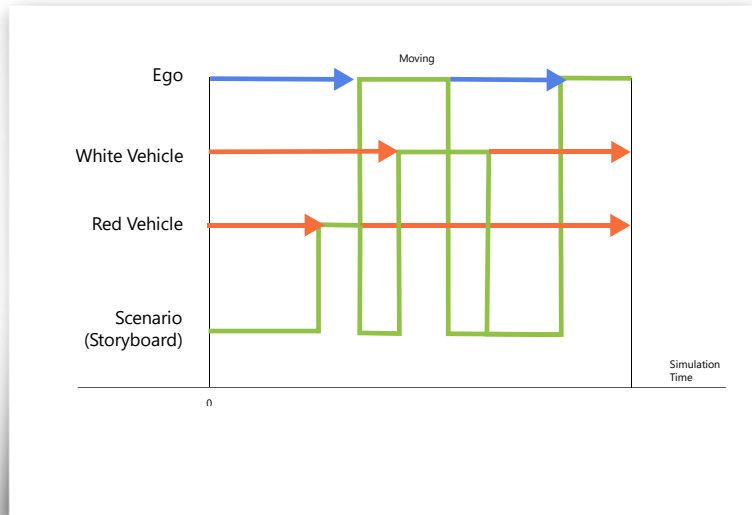
Runtime Interpretation

Complexity increases fast



Runtime Interpretation

Really Complicated



- Increasing numbers of entities
- Spawning parallel storyboard elements
- Stopping storyboard elements
- Applying default behavior

The complexity does not scale

Proposal

ABSTRACT CONTROLLER MODELLING

Overview

Three simple steps to simplify runtime implications

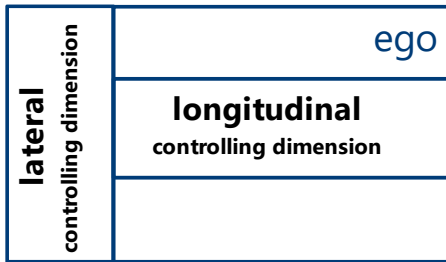
1. Separate actions and controllers more precisely.
2. Describe the relationships and the interactions between controllers, entities and actions with a few simple rules that are always true.
3. Make some small semantic adjustments and provide full downward compatibility to version 1.0.

Simple rules for

CONTROLLERS AND ENTITIES

Schematic Illustration

Let's start with controllers end entities.



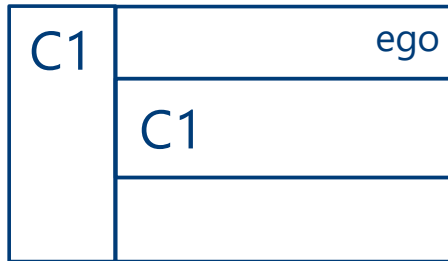
An entity provides a lateral controlling dimension and a longitudinal controlling dimension.
 (Orthogonal controlling dimensions).



Feels natural: The controlling dimensions of a vehicle are simplified by regarding its steering wheel (lateral dimension) and its throttle/brake system (longitudinal dimension)

Schematic Illustration

Controllers and entities.



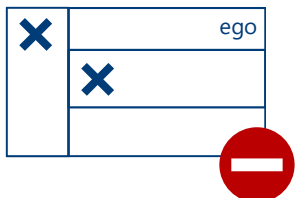
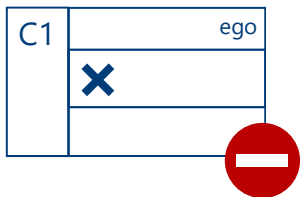
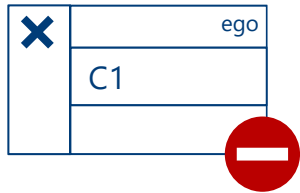
Example: A controller instance (C1) accesses both controlling dimensions of an entity.



Feels natural: Control must be applied to steering and throttle/brake by a driver or a driver assistant during movement.

Schematic Illustration

Controllers and entities.



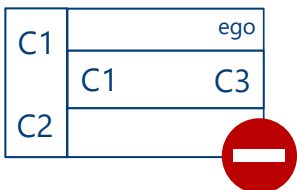
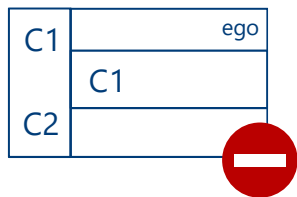
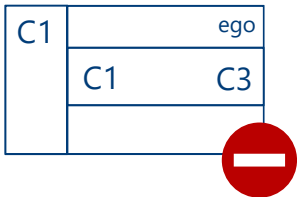
An active controller is defined for both controlling dimensions at *any time during the lifecycle of an entity*. If no explicit controller is assigned, the *default controller* will automatically step in.



Feels natural: A driver or a driver assistant must control both dimensions, while moving.

Schematic Illustration

Controllers and entities.



At any given time, only one controller can access a controlling dimension of an entity.

Or: a controlling dimension cannot be accessed by multiple controller instances at the same time.



Feels natural: Either a driver or a driver assistant accesses a controlling dimension exclusively (See ADAS levels).

Schematic Illustration

Controllers and entities.



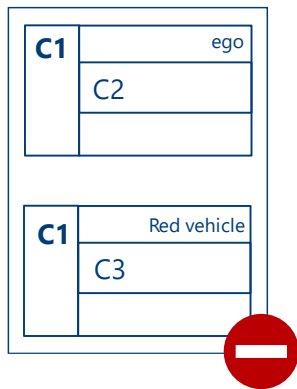
At any given time, a controller instance can only access **one or two** orthogonal controlling dimensions of exactly one entity.



Feels natural: A driver and driver assistants can access orthogonal controlling dimensions. (Driver, driver and cruise control, driver and lane keeping assistant, cruise control and lane keeping assistant)

Schematic Illustration

Controllers and entities



At any given time a controller instance can only access one or two orthogonal controlling dimensions of **exactly one entity**.



Feels natural: Though drivers are able to communicate with each other, a driver cannot control different vehicles
 => Principle of connected cars.

Controllers

Controller abstraction.

Abstracting a controller as a (closed loop) controller (“Regler”).

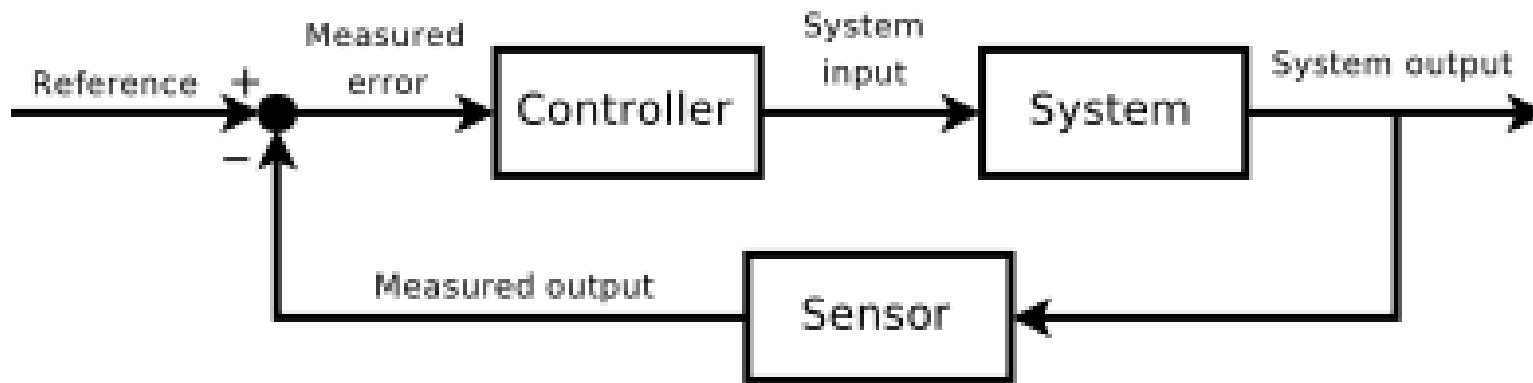


Image: [Wikipedia](#)

💡 Feels natural: Even a driver can be regarded as a closed loop controller with eyes, sense of touch as sensors, legs and hands as actuators and the brain as the “computer unit”.

Entities and Controllers

Summarized rules.

1. Controllers and entities are instances that interact during runtime. Controllers are accessing logical controlling dimensions of an entity.
2. An entity provides a lateral controlling dimension and a longitudinal controlling dimension. (Orthogonal controlling dimensions).
3. The simulation manages the concurrent access from the controllers to the controlling dimensions (lateral, longitudinal) of an entity.
4. An active controller is defined for both controlling dimensions at any time during simulation. If no explicit controller is assigned, the default controller will automatically step in.
5. At any given time, only one controller can access a controlling dimension of an entity.
6. At any given time a controller can only access one or two orthogonal controlling dimensions of exactly one entity.

Controllers

Some simple rules for controllers

1. A control (active controller) takes simulation time ≥ 0 .
2. An active controller always overrides other active controllers on longitudinal and/or lateral controlling units of an entity.
3. Controllers might additionally interrupt the orthogonal controlling unit. E.g. if an active FollowTrajectory controller is interrupted by a lane change controller on the lateral controlling unit, it does not make sense to keep the longitudinal control of the FollowTrajectory active. \Rightarrow Default Controller steps in for longitudinal control.
4. An active controller stops when
 - the controller accomplished it's final state (e.g. designated speed)
 - a controller is overridden by another.
 - constraints for a running controller are no longer valid (e.g. the reference entity disappears).
 - explicitly requested (Future).

Proposal

CONTROLLERS AND ACTIONS

Controllers and Actions

Current understanding of controllers and actions

1. Sometimes an entity is controlled by an action as part of the flow sometimes it is controlled by a controller as part of the simulation process (e.g. default controller, explicit assigned controller).
2. Even worse: Sometimes a controller controls one dimension and an action controls the other dimension.
3. Even worse: A controller (e.g. explicit assigned controller) stops an action and an action can stop a controller (e.g. speed action stops the default controller).

Controllers and Actions

From the view of an action

1. An action sometimes acts a function that takes zero simulation time. E.g. EnvironmentAction
2. Sometimes an action behaves like a controller that takes simulation time. E.g. SpeedAction.
3. In other cases an action behaves like a function that takes zero simulation time and sets a controller to be executed in the simulation. E.g. AssignControllerAction.
4. Even worse: An action of the same type sometimes acts different in terms of simulation times, depending on their settings (See complicated action table).

Controllers and Actions

From the view of an action (ongoing)

5. Sometimes the flow and the simulation are synchronized (if an action acts on an entity), sometimes they run in parallel. This is not explicitly modeled but it is implicitly depending on whether an action or a controller is acting on an entity and/or whether an action returns immediately or not.
6. This gets extremely complicated and incomprehensible when the flow spawns parallel branches (e.g. parallel events, executing actions).
7. Even more complicated when actions operate on different dimensions.
8. Even more complicated when regarding nested storyboard elements in the flow. (Stories own nested acts, acts own nested maneuver groups, maneuver groups own nested maneuvers, maneuvers own nested events, events own nested actions)

It is almost impossible for an author to scale from simple scenarios to more complex scenarios.

Controllers in 1.0

Just a statement

Controllers have been left as semantically empty. A controller owns only properties as attributes and parameter declarations to parameterize the properties. The semantics of the properties are out of scope of the standard.

```
<xsd:complexType name="Controller">
  <xsd:all>
    <xsd:element name="ParameterDeclarations" type="ParameterDeclarations" minOccurs="0"/>
    <xsd:element name="Properties" type="Properties"/>
  </xsd:all>
  <xsd:attribute name="name" type="String" use="required"/>
</xsd:complexType>
```

Proposal

A SIMPLIFIED CONCEPTUAL MODEL

A Simplified Model

A few rules solve many problems

1. An entity is *always* controlled by controllers (see previous rules). It is never controlled by an action.
2. An action sends *messages* to runtime instances like entities and their controllers, traffic signal controllers, the global instance etc. . Actions *do not act* on runtime instances and *return always immediately*.
3. The “flow” and the “simulation” are explicitly synchronized not implicitly.

Synchronizing Actions and Controllers

Default behavior 1.x

The action immediately returns after setting the controller:

```
<SpeedAction synchronize="false">  
  <SpeedActionDynamics dynamicsShape="step" value="2" dynamicsDimension="time"/>  
  <SpeedActionTarget>  
    <AbsoluteTargetSpeed value="20"/>  
  </SpeedActionTarget>  
</SpeedAction>
```


Synchronizing Actions and Controllers

A fully backward compatible solution to 1.0

An action that sets a controller is able to stop the flow until the controller is stopped.

```
<SpeedAction synchronize=„true“>  
  <SpeedActionDynamics dynamicsShape="step" value="2" dynamicsDimension="time"/>  
  <SpeedActionTarget>  
    <AbsoluteTargetSpeed value="20"/>  
  </SpeedActionTarget>  
</SpeedAction>
```

With `synchronized=true` as the default, this is the behavior from 1.0

```
<SpeedAction>  
  <SpeedActionDynamics dynamicsShape="step" value="2" dynamicsDimension="time"/>  
  <SpeedActionTarget>  
    <AbsoluteTargetSpeed value="20"/>  
  </SpeedActionTarget>  
</SpeedAction>
```

Problems solved

Answers given by the conceptual model

1. The initialization phase does not take simulation time by definition.
2. There will be no complicated action table any more. Only some small and generic rules for controllers (whether they take lateral, longitudinal or both controlling dimensions. Whether they give up control on the opposite control dimension when stopped).
3. Controller/Action concept can easily and immediately be extended to other controllers like traffic light controllers and traffic concepts (traffic source as an independent controller that creates vehicle, swarm as a controller for a swarm of vehicles).
4. Controller concept is also valid for pedestrians.
5. Controller concept is valid for driver in the loop. Take over control of lateral/longitudinal controlling dimension. Give up the control.
6. There are clear answers when a storyboard element ends.

Problems solved

Answers given by the conceptual model

7. The concepts are cleanly separated. On one hand the concept of storyboard elements flow (including actions), that is considered very stable over OpenSCENARIO versions. And on the other hand the expandable concept of controllers that is expected to be very volatile during the next years of ADAS development. (many different controllers, new controllers).
8. There is a one to one relation between an entities controlling dimension and a controller. An entity has an active controller at any time. There is no comparable strong relation between an entity and an action for 1.0.
9. An entity is always controlled by one or two controllers. It is never controlled by an action. In 1.0 an entity is sometimes controlled by action and sometimes controlled by a controller (e.g. by the default controller, or explicit controller).
10. An action can never be overridden by other actions. This solves a lot of conflicts by definition.

Problems solved

Answers given by the conceptual model

11. The flow of storyboard elements is in line with the complex but well known characteristics of concurrent systems. Splitting one process in many parallel processes. Synchronizing concurrent access to resources. Joining parallel processes into a single one.
12. An action does not necessarily effect a controller or a lateral or longitudinal controlling dimension. There is no one to one relation between action and controller.
13. An action does never take simulation time. A controller does always take simulation time. An action does not behave like a controller at one time, and like an immediate returning action the other time.
14. An active controller can exist beyond the limits of a storyboard element by definition. Not implicitly by setting a controller.
15. No special cases that some actions activate a control and some others applying control. „Simulation“ and the „flow“ run independently.

ADDITIONAL SLIDES

Conflicts

Potential conflicts still exist

1. Conflicts exist in 1.0: If two contradicting actions that return immediately are applied at the same time (e.g. Two contradicting EnvironmentActions).
2. But: In 1.x, conflict resolution is exclusively part of the flow and not part of the simulation.
3. For 1.x: In the simulation there are clear rules by definition: Last action wins. E.g. Last EnvironmentAction wins. Latest SpeedAction provides the controller.