Abstract Environment API

An Approach To Have Environmental Simulator Agnostic Scenario Engines
jupp.tscheak@daimler.com, Sindelfingen, 2020-10-26
Motivation (Use Case 1)

EBTB Scenario Database

*Highway*

EBTB Scenario Database

*Parking*

MB

Environment and Scenario Simulator

Potential for reuse

No possibility to separate scenario and environment simulation.
Approaches To Couple A Scenario System

1. Coupling via data interface (FMI, OSI). Environment 1 is running as Co-simulation.

2. Coupling using proprietary API. Scenario System is running as part of Environment Simulator 1.

3. Scenario does not know the underlying Environment Simulator.

Environment Simulator 2 implements the Abstract Environment API.
Where Are We Coming From?

1) Extending the Scope of Future OpenSCENARIO Releases
Kick-Off Workshop ASAM OpenSCENARIO, Kaiserslautern 2018-11-13
Architectural Overview

Abstract Environment API

Common
- IIdentifiable

Environment
- IEntity
- IVehicle
- IStaticObject
- IPedestrian
- IEntityRepository
- IEnvironmentController

Driver
- IVehicleController
- IVehicleControllerFactor

Map
- IRoute
- ITrajectory
- IMAP
- IQueryService

Execution
- IEnvironmentSimulator
- IScenarioState
- ISimulationState

Scenario Language Implementations
- EBTB
  - EBTBSenarioState
- OSC 1.0
  - OSC10ScenarioState
- OSC 2.0
  - OSC20ScenarioState

Environment Simulators
- CARLA
  - CarlaEnvironmentSimulator
- OpenMobility
  - SumoEnvironmentSimulator
- "Another" Provider
  - AnotherEnvironmentSimulator

Simulation Frameworks
- CARLA
  - Unreal
- Open Source Robotics Foundation
  - ROS
- "Another" Provider
  - Another Simulator

Mercedes-Benz
Exemplary Configuration

```cpp
// This class is a typical representative of a node/plugin/module of a simulation framework that is frequently updated.
// This is the blue component.
class ScenarioPlugin : public simvendor::BasePlugin, public scenario::abstract::ISimulationState {
public:
    bool Initialize() {
        // This is the red component.
        environment_simulator_ = std::make_unique<DriveSimEnvironmentSimulator>();
        // This is the green component.
        scenario_state_ = std::make_unique<scenario::ebtb::domain::EbtbScenarioState>(*this, *environment_simulator_);
    }

    void Update() {
        scenario_state_->Tick();
        environment_simulator_->Tick();
    }

private:
    std::unique_ptr<scenario::abstract::IEnvironmentSimulator> environment_simulator_{nullptr};
    std::unique_ptr<scenario::abstract::IScenarioState> scenario_state_{nullptr};
};
```
Using Different Scenario Language Databases

- Reusable and simulator agnostic.
- Portable to several test benches.
- There’s potentially still the need to have different DSLs for different users.

**Scenario Databases**
- C++/Python
- OSC 1.0
- EBTB
- YAML
- OSC 2.0

**Configuration**

**Simulator**
- DriveSim

**Environment/Traffic Simulators**
- libNVDriveSimEnvSim.so
Another Solution

Different languages to describe scenarios.

A standardized, Turing-complete format that represents a superset to all other languages. This approach is comparable to the LLVM concept.

- C++/Python
- EBTB
- YAML
- Compiler
- Compiler
- Compiler
- OSC 2.0
- OSC 2.0 Interpreter
- Environmental Simulator
Use Case 2: Rapid Development Platform

friendlysimuser@thinclient:~$ testpgm --script cut_in_testcase.cc --simulator libenv_simulator_binding.so --realtime

ISimulationState -> testpgm

IScenarioState -> CppScenarioState

IEnvironmentSimulator

OSI Data Provider

Mercedes-Benz
Use Case 2: Rapid Development Platform

friendlysimuser@thinclient:$ testpgm --script cut_in_testcase.ebtb --simulator libenv_simulator_binding.so --realtime
Use Case 3: Vehicle Controllers
Use Case 3: Vehicle Controllers

**Algorithm**

1. Get next junction ahead (IMap, IJunction).
2. Determine the path through junction that correlates with the desired turn direction (Query of possible paths with turn angles, IJunction).
3. Check whether vehicle is already on correct incoming lane for queried junction paths.
4. Navigational lane change if not on correct incoming lane.
5. Determine the corresponding traffic light signal (ILaneProperty, OSI traffic signal types?).
7. Get potential vehicles from opposite direction that have right of way (IEntityRelationGraph).
8. If there are vehicles that have right of way, stop accordingly, if not, leave junction.
Use Case 4: Parameter Variation

The “Parameter Variation” component needs to have access to the environmental simulator as well to e.g. query the road. This component especially reflects parts of the V&V methodology and describes the way how corner cases are found.

Abstract Scenario API

Concrete Scenario Set

\[ S_1 = \{ \text{Cut}_1, \text{Cut}_2, ..., \text{Cut}_n \} \]
Summary

- The Abstract Environment API enables the reuse of the EBTB databases.
- Any scenario language could be implemented in a generic, simulator agnostic way using such an API. This would allow the use of scenario databases in different languages without much integration effort.
- The scenario databases can be transferred not only from test bench to test bench but also reused in upcoming generations of ADAS.
- The environmental simulator is decoupled. New features like support of custom maps, internationalization or driver models need to be implemented there.
- Vehicle Controller models could potentially be developed using the Abstract Environment API as a world data provider. They inherently produce different behavior for different models and should therefore be part of the scenario deployment.
- The V&V methodology is reflected by the tool chain (parameter variation).
Follow Ups

- ???