A Task-Driven Framework for Driving Simulation: Scenario Orchestration with Autonomous simulated Vehicles (SOAV)

Zhita Xiong (Corresponding Author)

Research Centre for Integrated Transportation Innovation (rCITI)
School of Civil and Environmental Engineering
The University of New South Wales
UNSW Sydney NSW 2052 Australia
+61 (2) 9385 5721 (Centre Office)
z.xiong@unsw.edu.au

Institute for Transport Studies and School of Computing
38 University Road
University of Leeds, Leeds, LS2 9JT

Oliver Carsten, Hamish Jamson

Institute for Transport Studies
38 University Road
University of Leeds, Leeds, LS2 9JT
+44 (0)113 34 35348, +44 (0)113 34 35730
O.M.J.Carsten@its.leeds.ac.uk, A.H.Jamson@its.leeds.ac.uk

Anthony G. Cohn

School of Computing
E C Stoner Building 9.23
University of Leeds, Leeds, LS2 9JT
+44 (0)113 3435482
a.g.cohn@leeds.ac.uk

6242 words + 4 figures + 1 tables
Submitted: August 2, 2013 (August 1, 2013 HST)
Re-Submitted: November 16, 2013 (November 15, 2013 HST)
ABSTRACT

Scenarios in driving simulators cover what the human participants experience and what the researchers need: the physical scene, pre-defined traffic flow, simulated vehicles’ interactions with the participants and measurements to be collected.

Previous methodologies for orchestrating scenarios regarding the interactions have the following drawbacks: 1) action sequences that simulated vehicles should follow lack the contexts of each action; 2) programming languages always include platform-dependent details and are not suitable for context modelling and 3) scenarios cannot be generated dynamically to cope with failures that happen in trials.

To overcome the limitations above, an Ontology for Scenario Orchestration (OSO) was first developed to model concepts and their relationships in the domain of scenario orchestration, including a concept named Assignment, which represents the task(s) of virtual drivers and encodes the contextual information of proposed actions, e.g., simulated vehicles involved. It can also provide a file for machine processing.

An algorithm named NAUSEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAnning) was generated to utilise Assignments recorded using OSO.Encoded in the driver model SAIL (Scenario-Aware driver modeL), NAUSEA can be used by a virtual driver to control simulated vehicles dynamically. Failed Assignments, designed to generate specific interactions, can be re-tried if permitted. A framework SOAV (Scenario Orchestration with Autonomous simulated Vehicles) was formed to support SAIL/NAUSEA and orchestrate scenarios with autonomous vehicles.

Three verification experiments showed that SOAV worked properly by generating desired interactions and dealing with failures. OSO can also provide contextual information in a human-readable and machine processable manner.
INTRODUCTION

By imitating driving activities of the real world, driving simulators can:

- have reproducible and consistent scenarios;
- have a substantially risk-free environment;
- provide some hazardous situations not easily performed in the real world.

A scenario is then the key to provide a pre-defined environment that experimenters need a participant to experience. It includes the physical scene, pre-defined traffic flow, simulated vehicles’ interactions with the participant’s vehicle and measurements that need to be collected. Choreography of scenarios, therefore, plays an important role in driving simulation. In general, the interactions in scenarios should be reproducible and the human participants should be able to make similar decisions in driving simulators as they would in the real world, so there are two requirements when it comes to scenarios in driving simulation: realism and reproducibility.

Realism indicates the requirement of adopting sufficient driving behaviours. This term is motivated by the work from (1), which states that “In our experience with research studies in high-fidelity simulators, users generally focus their evaluation of the model realism towards the richness of the behaviors, not their fidelity.”

On the other hand, reproducibility means that the essential conditions in a scenario should be repeatable in each trial (2). However, the balance between realism and reproducibility can be hard to define because when scenarios are running, unexpected movement of vehicles, including the participant’s, may interrupt some pre-defined interactions. As a result, behaviours of each simulated vehicle are always limited in order to guarantee reproducibility.

This research, therefore, is trying to find a way of combining realism with reproducibility by orchestrating scenarios with autonomous simulated vehicles around the participant’s vehicle. By using an algorithm, it is possible to engage the participant’s vehicle actively based on rich behaviours and pre-defined tasks in order to create desired situations for interactions. During the process of developing the algorithm, a framework for scenario orchestration was developed.

This paper presents some major concepts and components in SOA V. “Related Work” Section introduces some background and is followed by a description of the framework SOA V (Scenario Orchestration with Autonomous simulated Vehicles). Verification and results will be presented in “Verification and Results” Section. Conclusion and future research will be discussed at the end.

RELATED WORK

In driving simulation, orchestration of scenarios has been a focus of research since the mid 1990s regarding how scenarios can be described and how to use those descriptions (e.g., (3)). Less effort has been put into this area in recent years since existing methodologies seem adequate enough for applications. Generally speaking, these methodologies share the same idea: have humans describe every aspect of the scenario, and then author the scenario to relevant simulated vehicles. This process can define the rules or sequence of actions that the simulated vehicles should follow before or when the scenario is exposed to human participants.

In general, three questions need to be answered in order to orchestrate a particular scenario:

---

1In this research, only one participant or participant’s vehicle (simulator) is included in a particular scenario.
which simulated vehicles will be involved in the scenario?

how will those simulated vehicles be prepared?

how can the scenario be produced?

Most researchers put a focus on answering individual question with some specific algorithms as listed below.

Question one can be termed “actor management problem”. Simulated vehicles that will interact with human participant’s vehicle can be pre-defined and fully described beforehand (e.g., (4)). Alternatively, an actor management algorithm was developed in (5) by considering the average speeds of simulated vehicles needed to reach the proposed location for interactions. A simulated vehicle with an inconspicuous speed trajectory, which is not noticeable to human participants, can be assigned to interact with the participant.

Question two can be termed “actor preparation problem” and is related to the “realism” part of the scenario, as in this phase simulated vehicles can have some autonomy to prepare themselves. In (5), some work was done regarding how to prepare simulated vehicles in an inconspicuous manner by adopting a specific speed trajectory towards the required location for interactions.

Question three can be termed “scenario execution problem” and is related to three aspects: what are available actions, how to trigger an action and how to schedule those actions. In previous work (e.g., (6)), the event-driven mechanism was widely used and can be regarded as pre-conditions for particular actions: conditions must be true before these actions are applied. Moreover, the scheduling of such actions is often crafted by humans in order to specify the execution order of actions or sub-scenarios (e.g., (7)).

None of these questions have covered scenario failures and some other scenario development issues as discussed in (8), such as the following examples:

Pre-Run Issues:

- the experimenter describes interaction outcomes without corresponding context and,
- the experimenter fails to correctly predict human participants’ reactions.

Run-Time Issues:

- participants do not want to be engaged in some interactions and,
- some interactions never happen due to design or system issues.

Pre-run issues suggest that an efficient communication mechanism is lacking. Experimenters may fail to foresee the extra actions from a participant as they focus on outcomes only; the experimenters do not have a general picture of the capacities of the scenario orchestration mechanism and the simulation software. This communication problem can make the scenario orchestration process time-consuming and make the scenario liable to run-time issues, so there needs to be some ways to share knowledge such that both experimenters and programmers can have some pre-knowledge, which includes, but is not limited to, the capacities of the driving simulator software, the potential pre-conditions and the effects of particular interactions.

Run-time issues invariably cause failures, e.g., a trigger does not fire or participants do not want to be engaged. Those failures are hard to avoid, but there is a possibility that they can be
fixed dynamically without help from humans by automatically re-orchestrating the scenarios, if
permitted.

In order to deal with the four issues above, both run-time and pre-run ones, a framework
SOAV (Scenario Orchestration with Autonomous simulated Vehicles) was developed and its two
major components: the Ontology for Scenario Orchestration (OSO) and an algorithm named NAU-
SEA (autoNomous locAl manoeUvre and Scenario orchEstration based on automated action plAn-
ning) encoded in a driver model named SAIL (Scenario-Aware drIver modeL) have been elaborated
in (9, 10, 11).

OSO is used to describe scenarios and relevant driving context to a virtual driver in a formal,
context-oriented, programming-independent, logic-based, human-understandable and machine-
processable manner. SAIL/NAUSEA is utilised by virtual drivers to carry out Assignments, which
are tasks that virtual drivers need to accomplish in order to generate the required interactions with
the human participant. Assignments can provide relevant contextual information to virtual drivers:
the Formation Position, Monitor(s), Success Condition(s), Failure Condition(s), Assignment-action(s)
and the measurement from the interaction generated by this Assignment.

In general, SOAV can provide the following features: 1) the simulated vehicles that are
controlled by virtual drivers can actively engage the participant to avoid failure and 2) if failures
happen, the simulated vehicles can be re-driven to generate the proposed interactions by virtual
drivers. However, SOAV still needs further development. In the rest of this paper, the framework
of SOAV will be described with more details compared to (9, 10, 11). All three experiments that
have been carried out to verify SOAV will be introduced and future research will also be discussed
at the end. In the next section, some domain concepts will be introduced first, which have been
encoded in OSO and formed the basis for this research.

FRAMEWORK DESCRIPTION

Generally speaking, SOAV is a framework used to orchestrate scenarios with autonomous simu-
lated vehicles in driving simulation. It was formed in order to support and test a decision making
algorithm, which needed a driver model to provide it with data for decision making and module for
decision execution. This driver model was used to develop a Virtual Driver. Some other facilities
were also implemented in order to control the simulated vehicles in the simulation according to the
decisions sent from the Virtual Driver.

Main Concepts

Virtual Driver

A Virtual Driver in this research indicates an intelligent controller that can be used to make driving
decisions or carry out pre-scheduled actions based on scenario requirements.

Driving Decisions indicate actions that should be undertaken to drive safely and satisfy
scenario requirements. The actions that this research will use are: overtaking, speed adap-
tation and lane-changing;

Pre-scheduled Actions indicate some actions required to change the states of objects in
driving simulation in order to generate specific interactions, e.g., request to “set the desired
speed of the simulated vehicle no.1 as 30 mph” or request to “place cones in a specific
road segment whose id is ‘r3.2’ ”;
That is, this Virtual Driver should be able to interfere with the simulated vehicles’ autonomous decision-making process and change the states of some objects in driving simulation, rather than the simulated vehicles themselves.

Monitor System
Monitors are used to supervise the state variables in simulation, e.g., vehicles’ speeds and positions. A Monitor System can be developed based on the work from (12), which identified four kinds of Monitors:

- **When** The Monitor will be activated once if its condition is satisfied. The condition is a state transition.
- **Every** The Monitor will be activated repeatedly if its condition is satisfied. The condition is a state transition.
- **As long as** The Monitor will be activated during the time when its condition is satisfied;
- **Whenever** The Monitor will be activated repeatedly if its conditions is satisfied.

As a result, the Monitors can supervise state transitions happening at any Instant or state status during any time intervals. The four keywords representing the four types of Monitors will be called Monitor operators in this research.

Flock and Ego-Vehicle/Flock
A Flock refers to a platoon of simulated vehicles. Its leader’s states, e.g., position, speed, etc., will be used to represent the Flock’s states in this research. A scenario can have several Flocks. The simulated vehicle or Flock the Virtual Driver is controlling is termed the “Ego-vehicle” or the “Ego-flock” respectively. The leader of the Ego-flock is driven by a Virtual Driver, who will manage that Flock with appropriate orders. Other members of the Ego-flock will simply adopt the same orders from the Virtual Driver.

Action
An Action is what a Virtual Driver can do to change the state of the simulated world. Actions are divided into two categories: High-Level and Low-Level.

A Low-Level Action is defined as an Action that can only be performed in one way, one sequence and by one entity. Examples of Low-Level Actions are: “Set Steering Angle”, “Set Speed”, “Set Acceleration Rate”, “Set Action Lane”, “Create an Object” (e.g., a vehicle or a road segment), “Set Action Status”, etc.

High-Level Actions need Recipes (13), each of which is a set of Low-Level Actions that specify how to perform a High-Level Action. For instance, “Block” is a High-Level Action that can be designed as: three simulated vehicles pass the participant’s vehicle to prevent it from overtaking its leader vehicle.

Each Action contains some information that can be used by a Virtual Driver: 1) the name/id of the Ego-vehicle/flock; 2) the proposed deadline of the Action; 3) the proposed Duration of the Action; and 4) the proposed release time of the Action. Moreover, each Action has a type and an Action profile, e.g., “change desired speed to 10 mph” is an Action whose type is “Low-Level” and “Adapt-Speed”. It has an Action profile of setting “Desired Speed” with a value of “10.0” (mph).
1

2 Trigger

3 Monitors mentioned in Section “Monitor System” can be used to establish Triggers: Pre-conditions, 
4 Success Conditions and Failure Conditions.
5
6 Because of the importance of Pre-conditions that can make a Virtual Driver perform par-
7 ticular Actions when some conditions are satisfied, the word Monitor has been used to specifically 
8 refer to Pre-conditions. Success Conditions and Failure Conditions, on the other hand, specify 
9 Triggers that can indicate whether or not an Action has succeeded and failed respectively. All 
10 three kinds of Triggers should include the following information:

11 • if the Trigger should dictate a state or an event (state transition).
12 • which object the Virtual Driver should supervise, e.g., a simulated vehicle or a Flock;
13 • which state variable of the object should be dictated and compared to some threshold 
14 value, e.g., speed, position;
15 • what threshold value should be compared to and,
16 • how to dictate the identified state variable: when, whenever, aslongas, every, see Figure 
17 ?? on Page ?? for more details.

18 As there are some common events in simulation, e.g., a vehicle enters or leaves a zone, 
19 Triggers can also include some Trigger type, which can be: “Cross (a pre-defined line)”, “Enter (a 
20 pre-defined zone)”, “Exit (a pre-defined zone)” or “Timer” (12).

21 In addition, this research has put a focus on dealing with failures caused by triggers that 
22 do not fire and participants that do not want to be engaged, so failure conditions represent some 
23 special states of objects in The Sim when an Assignment is not fired.

24 Formation Position

25 Inspired by (14), the concept of Formation Position (Figure 1) has been proposed to specify the 
26 spatial goal of a Virtual Driver. For instance, if the Virtual Driver’s spatial goal is “Follower”, then 
27 he/she will try to drive the Ego-vehicle/flock to the “Follower” position as specified in Figure 1.
28 Because Formation Position is a set of pre-defined relative local positions around the simulator 
29 driver/participant’s vehicle, no assumption has been made regarding the absolute spatial relations-
30 ships between them. However, according to specific applications, values can be assigned in order 
31 to indicate how far those positions should be. For instance, 200 metres can be used to indicate the 
32 optimal distance between the “Leader” position to the Participant’s vehicle’s position.

33 Role Matching

34 This is a process used to decide which simulated vehicle a Virtual Driver should control. It is 
35 determined by required vehicle type, Formation Position and the required speed. A successful 
36 Role Matching should identify any simulated vehicles that 1) are the required vehicle type, e.g., 
37 Volvo S40; 2) are at or near the required Formation Position and 3) have sufficient maximum speed 
38 that can reach the required Formation Position in time. A description of this process can be found 
39 in (9).
Assignment
An Assignment is the task that a Virtual Driver needs to carry out. Every scenario contains at least one Assignment. An Assignment includes three kinds of information: general information, Triggers and Assignment-actions.

Firstly, a Virtual Driver should know the following facts:

- the state of an Assignment, which can be:
  - **Initial** indicates that the Triggers in the Assignment should not be monitored at present;
  - **Pending** indicates that the Triggers in the Assignment are being dictated or should be dictated from the next decision loop;
  - **Failure** indicates that this Assignment is failed because of the failure of its Assignment-action(s) or the Failure Conditions are true;
  - **Success** indicates that this Assignment is now finished and has succeeded;
- any requirements regarding the Ego-vehicle/flock, e.g., required vehicle type;
- the spatial goal of the Virtual Driver where the Ego-vehicle/flock should be driven to before the execution of some Assignment. This spatial goal is based on the Formation Position in Figure 1;
- how many times the Virtual Driver can try to finish this Assignment;
- the pre-defined Ego-vehicle/flock for the Assignment. If it is not specified, the Virtual Driver will look for candidates dynamically.

Secondly, a Virtual Driver should know any Triggers that include 1) the Monitor that is used to activate Assignment-actions and 2) the Success and Failure Conditions that indicate the accomplishment or abortion of its parent Assignment.
Finally, a Virtual Driver should know what Actions to undertake in an Assignment, these are detailed in the Assignment-actions. Assignment-actions can be driving actions, e.g., change-lane, or non-driving actions, e.g., “create new vehicle”, “collect driver’s reaction time (braking)”, “create a road segment”, “activate traffic flow one”, “activate Assignment two” or “activate Situation one”. Assignment-actions can be triggered and executed once unless a failure has been identified. Repetition of Assignment-actions is possible by adopting an indicator stating how many times the Assignment-action can be repeated, but in this research, this is ignored.

Assignments can be used not only by a Virtual Driver to generate any required interactions with the capability of re-generating those interactions, but also by experimenters to describe interaction context along with outcomes (measurements or Actions needed). They can also predict the reactions from participants by providing Success/Failure Conditions, so the pre-run issues can be dealt with by utilising Assignments to encode and notify 1) the interaction outcomes along with corresponding context and 2) the potential reactions from participants.

**General Plan**

A Virtual Driver should have some plan to guide its behaviours so a concept of the General Plan was introduced. It is a temporal constraint graph $G_{\alpha}$ (see (13) and (9) for more details) and is built from several pre-defined Actions and Assignments. It contains metric and precedence constraints. For instance, “Action $\alpha$ before Action $\beta$” is a precedence constraint; “start time of $\beta$ - finish time of $\alpha \leq 100$ (seconds)” is a metric constraint. There are two types of metric constraints: duration and delay. Delay is the period between two Instants that belong to different time intervals, while duration is the period between two Instants that are within the same time interval.

In every scenario, each Virtual Driver needs to finish a top High-Level Action $\alpha$ that can be either Perform-scenario or Free. Perform-scenario has only one Recipe that contains four sub-Actions, namely, $\beta_0$, $\beta_1$, $\beta_2$ and $\beta_3$ as shown in Figure 2. Free makes the Virtual Driver ignore any Assignments and autonomously navigate the world, in which case the route or the destination will be based on a pre-defined route. Recipes are not needed for Free.

![FIGURE 2 Action Recipe for the Virtual Driver (Perform-scenario)](image)

$\beta_0$ (Get-to-the-initial-state) adopts the initial state (e.g., initial speed, initial target speed etc.). $\beta_1$ (Generate-formation) means that a Virtual Driver should navigate the Ego-vehicle/flock to the proposed Formation Position in order to perform the corresponding Assignment-actions. Because the recipe of $\beta_1$ will change according to the dynamic environment, this Action will not be further divided into sub-Actions, but will be monitored throughout the scenario in order to make sure that the Ego-vehicle/flock can get to the Formation Position in time. $\beta_2$ means Perform-assignment Action, and can be further divided into several Assignment-actions, which...
are represented as $\gamma_0$ through $\gamma_n$ ($n$ is the number of Assignment-actions a Virtual Driver needs to carry out). Each Assignment-action ($\gamma_0$ through $\gamma_n$) is contained in a corresponding Assignment. $\beta_3$ (Clean-up) should be specified by experimenters as an Assignment-action in most circumstances; however, it can be an autonomous Action by changing it to the top-Action of Free. In this research, some High-Level Actions in the Recipe of $\alpha$ will not be further refined, i.e., $\beta_0$, $\beta_1$ and $\beta_3$, as they change according to dynamic environment. Moreover, all Assignment-actions have been regarded as Low-Level Actions in his research.

Scenario
A scenario is a pre-defined environment and situation(s) that experimenters need a participant to experience. It includes the physical scene, pre-defined traffic flow, simulated vehicles’ interactions with the only one participant’s vehicle and measurements that need to be collected.

The interactions in a scenario are generated by the Virtual Driver, who changes the state of the environment by controlling simulated vehicles or modifying the physical scene, e.g., place cones on the road. In the former case, the Virtual Driver controls the Ego-vehicle/flock to produce interactions with the participant’s vehicle and in the latter case, the Virtual Driver will request corresponding facilities to modify the physical scene.

A scenario can have several Assignments performed by the Virtual Driver, so a scenario can have different interactions, aiming at providing individual measurements in one trial or run. For instance, a particular scenario can have two interactions, the participant should first be blocked by a lorry at some position for 10 seconds, after which, the participant should experience a five minutes’ free drive. The second interaction should happen right after the free drive: a lead car should break down and force the participant to brake.

In the following section, components of SOA V will be introduced.

Components of SOA V
SOA V is generally divided into two parts: Offline and Online (Figure 3). Descriptions of each component are as follows:

Offline Component
The Offline Component is in charge of describing scenarios using OSO (Ontology for Scenario Orchestration) based on Protégé (15), which outputs an SDF (Scenario Definition File) with the RDF/OWL syntax recorded in an XML file. As how OSO is recorded does not affect the work in this research, RDF, OWL and XML will not be introduced. Details of those languages can be found in (16).

Protégé is an ontology editor and knowledge acquisition system developed by Stanford University and the University of Manchester. It is free, Open Source, widely used and well supported, so Protégé has been chosen as the editor for OSO and thus the scenario orchestration tool. Moreover, Protégé also provides reasoners, e.g., HermiT, to check a particular ontology’s subsumption: if one class can be a subclass of another class or its consistency: if there are any classes that cannot have any individuals.

Footnotes:
2RDF is short for Resource Description Framework; OWL stands for Web Ontology Language; XML stands for Extensible Markup Language.
3http://protegewiki.stanford.edu/wiki/HermiT
OSO covers concepts in driving simulation ranging from physical objects such as roads or junctions to virtual ones such as Assignments or Actions. OSO can provide:

- driving context representation, which is the description of a logical road network and other knowledge regarding physical objects and their states, e.g., simulated vehicles and their vehicle model;
- task representation, which specifies Assignments that Virtual Drivers should do. An Assignment contains Assignment-action, Trigger(s) and other related information, e.g., Ego-vehicle requirement;
- Action library, indicating what High-Level and Low-Level Actions are available. Recipes are provided to guarantee reproducibility by forcing Virtual Drivers follow the same Action trajectory when facing the same context. They are always used to defined how to perform the “Perform-assignment”, which is $\beta_2$ as specified in Section “General Plan”;
- Monitor System representation, indicating what should be dictated so that particular Assignment-actions can be triggered, finished or failed when conditions in Triggers are satisfied and,
- temporal representation, specifying temporal relationships between some entities;

More details regarding OSO can be found in (11).
Online Component

The Online component contains three major modules and one optional module. The four modules are described as below:

Virtual Driver (Smith)

Smith is the Virtual Driver in SOAV. He is equipped with a driver model named SAIL (Scenario-Aware driver model), which uses an algorithm named NAUSEA (autonomous manoeuvre and Scenario orchestration based on automated action planning) to make decisions according to Assignments. NAUSEA contains several procedures, namely Targeting, Situation Assessment and Regulating. Targeting is used to plan the route for Smith and set goals for the Situation Assessment. The Situation Assessment is used to supervise the Assignments and route. Regulating uses any relevant tactical driving behaviours to drive safely and satisfy the goals of the Situation Assessor, which are mainly the goals of reaching a specific Formation Position. This three procedures uses the General Plan to guide their execution.

A running Smith is paired to a simulated vehicle in the scenario and committed to Assignments throughout that scenario. Smith’s name has been borrowed from “The Matrix” along with the related philosophy, so Smith is a male and he/his will be used when necessary. More details regarding ”The Matrix” metaphor, Smith, SAIL/NAUSEA can be found in (9) and (10).

The Sim

The Sim contains the vehicle dynamics and rendering facility for the simulation. It updates simulated vehicles’ states every frame according to each one’s autonomous driving behaviours and related parameters, e.g., desired speed. It also updates the participant’s vehicle’s states according to the driving instructions from the participant. It then updates the graphics accordingly to display the driving conditions. Moreover, The Sim needs to broadcast information of all vehicles every frame and has been enhanced with a module named SMM described below to receive and execute orders from Smith.

SMM

SMM is short for Scenario Management Module, which is used to interpret orders from Smith and execute them in the Sim. It has been developed as a module within the Sim. Hence, when SMM receives orders from Smith, which are termed Smith Orders, the Sim/SMM will activate the relevant process(es) to handle those orders. The simulated vehicles’ behaviours or road conditions (traffic flow or objects on the roads such as cones) will then be updated along with the graphics. A Smith Order can include goal lane to change to, target speed, etc.

Scenario Observer

This module is a “listening” module that receives all the data transferred within SOAV. It can visualize and indicate what is happening within SOAV. However, as a GUI is not one of the research goals, this module simply records the data with a visualization of the frequency and number of data packages.
As illustrated in Figure 4, the scenario description is first generated with instantiated individuals based on classes and properties in OSO. Protégé will then export those individuals along with classes and properties to an SDF (Scenario Definition File). Smith will then parse the SDF and store the information, which is the initial World Model for the whole simulation. Role Matching will take place if scenario Assignments are found in the World Model. After the Ego-vehicle/flock is found and ready, Smith will then try to finish the scenario according to Assignments. In SOA V, entity creation/destruction is handled by SMM and can be requested by Smith. Smith monitors and triggers any Assignments to generate corresponding interactions on his own. This is achieved by sending “Smith Orders” containing relevant Assignment-actions, e.g., the reference lane to track or target speed to achieve, to the Sim. Actions sent in “Smith Orders” are regarded as Low-Level Actions in this research.

In general, SAIL/NAUSEA should be able to help Smith plan his Actions based on Assignments so that Smith can 1) find Ego-vehicle/flock, 2) “drive” the Ego-vehicle/flock to the proposed Formation Position and 3) execute Assignment-actions as required. He should also be able to replan if necessary. Moreover, OSO should be evaluated to test its expressiveness and demonstrate its usage in describing relevant context for interactions along with potential reactions from partici-
pants. Finally, SOAV should also be evaluated to see if its implementation requires improvement for future application. The verification process has four steps: framework verification, framework application, framework migration & enhancement and OSO evaluation. During evaluation, some assumptions have been made:

- the simulated vehicles in The Sim are already realistic enough for the verification of the framework, i.e., they have already been equipped with rich enough tactical behaviours such as overtaking;
- Assignments are independent, so Role Matching can be performed by considering only the successive Assignment;

Experiment one was used for Framework Verification. It involved five human participants and five pre-defined Assignments (“Coherence”, “ACC-BL”, “Layby”, “Free Traffic Flow” and “Gap Acceptance”). The simulation software from UoLDS\(^4\) was used as The Sim. Automatic software test by utilizing the data collected with human participants was also performed. Smith controlled simulated vehicles in The Sim according to the three Assignments sequentially (“Acc-BL”, “Coherence” and then “Layby”) or requested traffic flows for the other two Assignments (“Free Traffic Flow” during “Coherence” and “Gap Acceptance” after “Layby”). Another Assignment “CL-BL” was created dynamically by Smith to compensate for the failure of Assignment “Acc-BL”.

If a Test Case represents a specific set of Assignments that a participant could experience, i.e., a Test Case is a set of Assignments that a participant would experience or sabotage and the corresponding information Smith received, there were in total nine Test Cases in this experiment. For instance, in Test Case no. 1, “Coherence” was not failed by the participant, so it was performed by vehicle no.1 and thus, Assignment “Coherence1” was executed; “Layby” was not failed by the participant, so it was performed by vehicle no.3 and thus, “Layby-V3” was executed. As a contrary, Assignment “Coherence” and “Layby” were both failed by the participant in Test Case 8, so they were performed by vehicle no. 2 and vehicle no. 4 respectively, which were represented by “Coherence2” and “Layby-V4”. In general, every participant experienced Test Case no.1 without failed Assignments and one or two Test Cases with failed Assignments (9).

With human participants, all Assignments have been triggered as desired. The statistics of release times in automatic software test can be found in Figure 1. The time points are the time elapsed since the beginning of the simulation in The Sim. In general, SAIL/NAUSEA was working properly with a 100% success rate in triggering desired Assignments. Therefore, it can be used to 1) find Ego-vehicle/flock, 2) navigate the Ego-vehicle/flock to a Formation Position and 3) execute required Assignment-actions. It can be also used to retry Assignment after failure. However, the implementation needs improvement as the time needed by Smith to make decisions vary a lot from less than 1 frame (1/60 second) to more than 10 frames compared to the mean release time, which can be problematic when considering reproducibility. OSO also worked as desired by providing sufficient contexts for interaction generation. Details of the experiment can be found in (9) and how to represent the experiment with OSO can be found in (11).

\(^4\)University of Leeds Driving Simulator
TABLE 1 Statistics of the Release Times in Automatic Software Test (s) (9)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Cohesion1</th>
<th>Cohesion2</th>
<th>Layout-V3</th>
<th>Layout-V4</th>
<th>Gap Acceptance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Mean</td>
<td>Std Deviation</td>
<td>Min</td>
</tr>
<tr>
<td>1</td>
<td>247</td>
<td>247</td>
<td>183</td>
<td>108</td>
<td>0.037</td>
</tr>
<tr>
<td>2</td>
<td>206</td>
<td>206</td>
<td>206</td>
<td>564</td>
<td>177</td>
</tr>
<tr>
<td>3</td>
<td>244</td>
<td>244</td>
<td>244</td>
<td>0.093</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>238</td>
<td>238</td>
<td>238</td>
<td>0.093</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>238</td>
<td>238</td>
<td>238</td>
<td>0.093</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>238</td>
<td>238</td>
<td>238</td>
<td>0.093</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>181</td>
<td>181</td>
<td>181</td>
<td>0.093</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>215</td>
<td>215</td>
<td>215</td>
<td>0.093</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>196</td>
<td>196</td>
<td>196</td>
<td>0.093</td>
<td></td>
</tr>
</tbody>
</table>

Even so, SOAV should be tested with other projects’ scenarios and some components in
NAUSEA, especially the Regulating procedure, needed improvement. NAUSEA had no special
algorithm to handle simulated vehicles’ speeds when approaching any proposed Formation Posi-
tions. Therefore, two further experiments have been conducted.

Experiment two was used for framework application. It was used to ascertain the function-
ality of SOAV in a “real” application from a previous research project - ITERATE, because it had
failed several times as the participant was too far behind the proposed vehicle for interactions (17).
It was carried out with ten human participants and utilised the simulation software from UoLDS as
well. The results show that SOAV successfully generated required interactions but how to navigate
the Ego-vehicle, which relates to Regulating procedure, needs improvement, especially when a
specific vehicle model is needed for Assignments.

Experiment three was used for framework migration & enhancement. The simulation soft-
ware from VTI5 was used as The Sim in order to adopt the actor preparation algorithm from
(5), which can enhance the Regulating procedure in NAUSEA, i.e., how those simulated vehicles
should be driven to a specific Formation Position. This final experiment utilised ten autonomous
simulator drivers that shared the same behaviour model as the autonomous vehicles in (5). The
results show that SOAV successfully prepared not only the Ego-vehicle but also the Ego-flock
beforehand to prevent participants from overtaking in leader car-braking interactions. This was
achieved not only by committing to Assignment based on corresponding contextual information,
but also by utilising an enhanced Regulating procedure derived from (5) to get the optimized speed
trajectory when approaching human participant’s vehicle and get to the desired Formation Posi-
tions.

As tested in the three verification experiments, SOAV, as a task-driven framework based on
a concept Assignment and corresponding algorithm NAUSEA that utilised this concept, worked
as desired by having a virtual driver who can actively commit to and plan Actions towards As-
signments with the possibility of restoring interactions from failures, i.e., SOAV can be used to
orchestrate scenarios with replanning capability by not only generating required scenarios but
also constructing desired context before corresponding interactions. Moreover, as demonstrated
in experiment three, SOAV can be enhanced with other algorithms and adopted in another driving
simulation platform.

5The Swedish National Road and Transport Research Institute
OSO was also evaluated, as described in (11). In general, as an early attempt of 1) mod- 
eLLing scenarios in driving simulation, 2) modelling driving context along with human-directed 
tasks and 3) standardising scenario description and orchestration mechanism, OSO has achieved 
its original purposes. Moreover, the context modelling and ontology engineering criteria from 
(18) are largely satisfied, e.g., 1) Applicability: OSO can be also used in areas that need to model 
driving context, so it is not limited to scenario orchestration in driving simulation only; and 2) 
Inference: in order to be descriptive, OSO has been designed with the purpose of expressiveness, 
so the capability of inference has not been covered.

The framework still needs further developments, especially regarding OSO and some com- 
ponents in NAUSEA:

• failures should be categorised and represented in OSO in order to provide relevant in- 
formation to driving simulator users, e.g., which type of failure can happen if a specific 
Assignment is adopted; this category can also be used to test SOAV;
• multiple Triggers, especially multiple Monitors, were assumed to be satisfied in the mean 
time, so further research should be done regarding disjunctive Triggers when needed. 
OSO should be updated to reflect Monitors’ new relationships;
• NAUSEA has been designed to handle local manoeuvres that cover less than 200 metres 
around the participant’s vehicle or six seconds time-headway. By providing some suitable 
Regulating algorithm, not only can the Formation Positions be defined with absolute 
locations, but Regulating (Action \( \beta_1 \)) and replanning after failure can be enhanced also.
• based on personal communication with relevant experts or documents authored by them, 
OSO is designed to be comprehensible to human. However, whether or not it is user- 
friendly has not been examined and how to present OSO to different audiences, e.g., 
scenario designers, simulator developers, needs further investigation.

CONCLUSION AND FUTURE RESEARCH

Compared to previous work (9, 10, 11), this paper presented the full framework of SOAV with 
a summary of results from another two experiments. It put a focus on the overall mechanism of 
SOAV.

Different from the existing methodologies in scenario orchestration, SOAV used 1) a pro- 
gramming language-independent and logic-based ontology to describe scenarios, including poten- 
tial Actions and relevant contextual information encoded in Assignments; 2) a Monitor system 
derived from (12) to model a standardized event-drive mechanism in OSO to form Triggers; 3) a 
hierarchy of Actions in order to standardize the Actions available for scenario orchestration and 
describe scenarios in different details, e.g., scenarios can be orchestrated with detailed Action such 
as “decelerate” or with abstract Action such as “Block” and 4) an algorithm NAUSEA along with 
its user SAIL for scenario orchestration, which can provide a scenario-driven way of generating 
interactions with the capability of replanning. In general, the contributions of this work are:

• A medium for scenario and driving context sharing/understanding/reusing was developed 
based on the Ontology for Scenario Orchestration (OSO);
• A methodology of adopting human driver’s task into an autonomous vehicle was proposed. It can make the simulated vehicles carry out the tasks as required in a controllable manner. In driving simulation, this controllable manner can be used to guarantee reproducibility.

This research is promising and has provided some insights into the solutions to some scenario orchestration issues. However, due to the research focus, SOAV, especially NAUSEA, still has some drawbacks that need to be taken care of in the future. Collaborations within the driving simulation community and relevant researchers from other areas, e.g., autonomous vehicles, are anticipated.

To sum up, the computing environment SOAV, which was created to orchestrate scenarios with autonomous simulated vehicles in driving simulation, worked as desired. Its major components, namely OSO and Smith (SAIL/NAUSEA), can be the keys to providing scenario orchestration with a new future, when knowledge is more widely shared in different simulator platforms and failure-free scenarios can be orchestrated. Moreover, SOAV can also benefit some other areas:

• a standardized framework for designing the controller of autonomous vehicles can be proposed based on SOAV. Autonomous vehicles can interact with human drivers and focus on cooperation. Also, The Sim will be replaced with a physical vehicle and the SMM will be replaced with some physical interface, e.g., wheel or pedal controllers. Data collected in autonomous vehicles can be directly integrated into a simulation platform with SOAV;

• similarly, in-vehicle devices that can understand human drivers’ tasks can be also designed based on SAIL/NAUSEA;

• OSO can be used to standardize information exchanged in vehicular communication systems, e.g., vehicle to vehicle or vehicle to infrastructure communications.

REFERENCES


