Generation of Highway Sections for Automated Test Case Creation

Ulf Noyer [1], Michael Scholz [1], Andreas Richter [1]

(1) German Aerospace Center (DLR), Institute of Transportation Systems, Lilienthalplatz 7, 38108 Braunschweig, Germany, e-mail: ulf.noyer@dlr.de, michael.scholz@dlr.de, andreas.richter@dlr.de

Abstract *– Creating proper roads for driving simulation including logical representation is usually a demanding, time consuming, iterative and manual task using classical visual scenario editors. Furthermore, creating and managing multiple variants of tracks varying in details is needed to build virtual test cases, which become especially important with regard to automated driving and its enormous test requirements. From this background manual test scenario creation becomes quickly overwhelming. Consequently, approaches for automated creation of tracks for testing purposes are needed.*

This paper presents a solution to create various tracks automatically covering a huge variety of characteristics. It uses a lightweight road description to minimize definition effort done either manually or derived methodologically. This work is embedded in the PEGASUS project that is investigating generally accepted quality criteria, tools and methods as well as scenarios and situations for the release of highly-automated driving functions. PEGASUS is funded by the Federal Ministry for Economic Affairs and Energy (BMWi).

Keywords: *OpenDRIVE, testing, domain-specific language*

Introduction

In the PEGASUS project [Hun17] it was decided to use OpenDRIVE for the topological and topographical description of road networks. It is an open, vendor-independent mature XML-based file format, which contains all features to model real road networks [Dup15] and is supported by commonly used simulation software. Therefore, OpenDRIVE is a natural choice to model virtual tracks for test cases. In combination with OpenSCENARIO for describing dynamic behavior of objects and traffic participants (an open and XML-based file format as well), these two formats build a solid foundation to model environments for virtual test cases.

In the PEGASUS project each specific scenario results in a broad variety of different test cases for which individual road snippets are to be generated. These test cases define the parameter space of a scenario for the procedural road generation in OpenDRIVE.

It is a proven approach to use domain-specific languages (DSL) to reduce complexity and increase efficiency for testing purposes [Deu00]. This is also the chosen way for the presented solution, as OpenDRIVE is powerful but also quite complex. Modelling efforts can be reduced significantly by the introduction of such an abstraction layer into the prototyping toolchain of OpenDRIVE. It has to be expected that first scenarios of highly automated driving will occur on highways [Bar15]. Consequently, this fact allows in a first stage to focus on modelling highways and their characteristics in a DSL.

Highways are usually not built at will, but follow strict rules and guidelines. For example, in Germany the “Richtlinie für die Anlage von Autobahnen” (RAA, means guidelines to lay out motorways, [RAA08]) defines possible cross sections, allowed markings including their sizing etc. and combination of longitudinal courses and other properties of highways. The challenge is to include this rule set into a description with reduced complexity. With this domain-specific knowledge bundled in a DSL it should be possible to rapidly increase efficiency of modelling realistic virtual highways for test cases.

Related work and approach

Often manual and prebuilt test tracks are used for simulator studies. They can represent reality very well but are only flexible regarding weather conditions, traffic and dynamic content like that. Another approach uses tiles containing street sections and surrounding objects based on standardized edges such as applied in the ENABLE-S3 project (www.enable-s3.eu). These tiles can be easily and quickly combined by a user or algorithm to create larger areas. Obviously, this

approach drastically increases productivity. But as tiles must be combinable, the content near the edges has quite strict limitations, which result in rather unnatural street courses of the created maps. For limited scenarios (e.g. parking decks) this approach can be still valid though.

The presented background leads to the research question for this paper: Can a DSL be used to efficiently model various variants of highways to create (adequately realistic tracks) to describe test cases?

The methodology used for this examination covers the following steps based on an experimental approach: To investigate this question, first the needed elements of highways were identified and are used to create a first version of a DSL called SimplifiedRoad. As next step, a transformation engine for SimplifiedRoad into OpenDRIVE was developed. With these foundations the approach to use a DSL for describing highways for test cases is evaluated. Especially aspects of usability, efficiency and proper modelling are investigated.

Implementation

Overview

The next sections discuss the outcome of our work. As relevant content from the road construction guidelines RAA several attributes for describing the appearance of the road as well as the road infrastructure “on top” were identified: number of lanes and their width, line markings and sizing, course and elevation of the road, lateral profile, traffic signs and environmental objects. A fundamental design goal of SimplifiedRoad is to be as similar to OpenDRIVE as possible without applying the complexity of OpenDRIVE to SimplifiedRoad.

Especially some optional enhancements are introduced in addition to OpenDRIVE: SimplifiedRoad allows mirroring sides of the roads with a simple attribute and furthermore pre-defined cross sections according to the RAA can be applied. With these additions simple road segments can be described already with very few lines of code. Furthermore, meaningful default values are used for all elements where possible. For example, default coordinates of street segments are generated at the end of previous segments. Environmental objects like guide posts, guard rails or noise barriers are typically used repeatedly and consist of several individual elements, for simplification all elements are identical.

An important upcoming feature is the possibility to allow parametrization of attributes in a SimplifiedRoad file. That is especially helpful to manage several variants of a track with slight differences (e.g. different lane widths, distances of road signs etc.).

SimplifiedRoad definition

The SimplifiedRoad format is specified by a XML schema. Figure 1 shows the structure of it. To improve clarity in the figure, some node details are collapsed. Especially, the children for *course* are the same for *lane*, *elevation* and *lateral profile*. Furthermore, children of *center* are the same for *left* and *right*.

With these elements, it is currently only possible to define one long street without junctions. With the focus on test scenarios, this limitation is perfectly

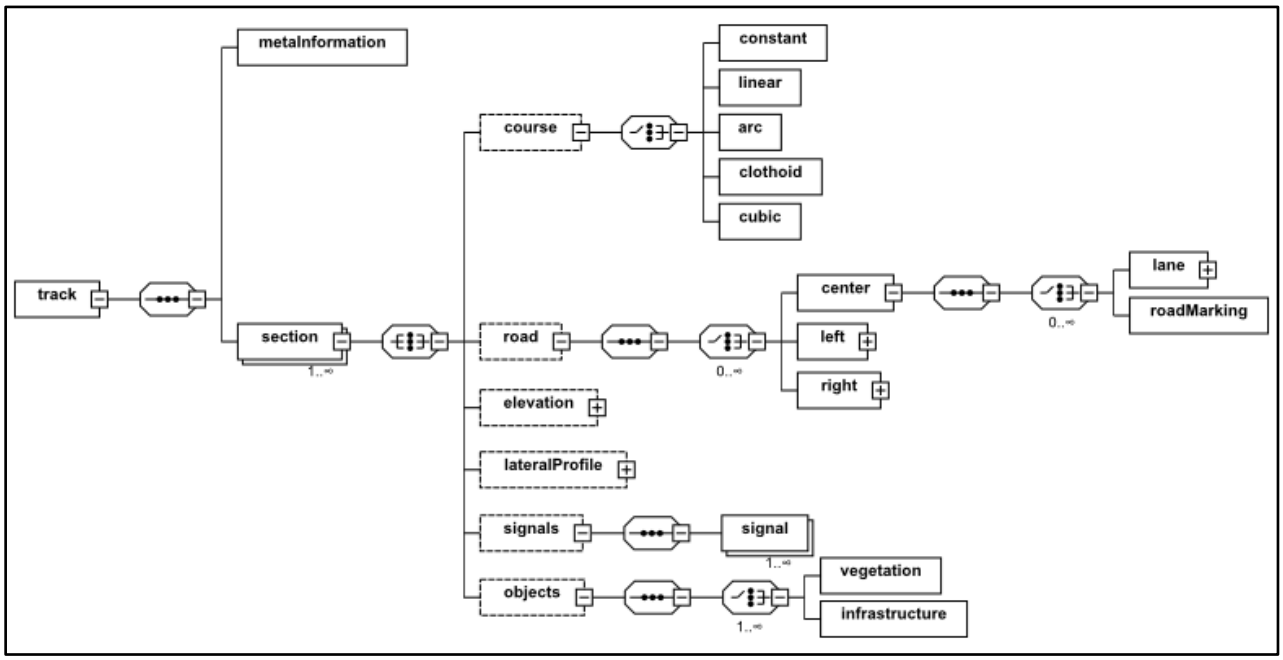

Figure 1. Structure of SimplifiedRoad

admissible. In the future, it should be possible to define highway entries and exits based on guidelines and a possibility of modification, too. From the perspective of the test scenario it is just needed to provide the road currently under test. Crossing streets, including entries and exits, need to exist for some distance, as sensors can just detect some limited area. That means, that dead ends of crossing streets are perfectly valid and these streets must not be defined behind their first sections.

Figure 1 does not contain any attributes for the XML elements. As example, Table 1 shows the attributes of the element *infrastructure*.

Table 1. Infrastructure attributes

Name	Type	Default	Use
type	enum_infrastructure	guide post	optional
left	boolean	false	optional
center	boolean	false	optional
right	boolean	true	optional
s	double	*variable*	optional
t	double	*variable*	optional
height	double	*variable*	optional
repeated	boolean	false	optional
length	double	*variable*	optional
distance	double	*variable*	optional

From the table it is visible, that there are several simplifications compared to OpenDRIVE, as less XML elements are needed to specify same content. Furthermore, attributes have meaningful default values as often as possible.

Besides *guide posts* there are currently further supported infrastructure elements, such as *noise barrier*, *railing* and *guard rail*.

In OpenDRIVE e.g. noise barriers are built of single elements as poles and individual wall sections, which are individually positioned. In SimplifiedRoad noise barriers are basically just turned on or off for a certain section. This simplification as design choice implies again, that SimplifiedRoad cannot specify as many details as OpenDRIVE. However, it does not need as it is primarily meant for rapid design of roads for test use cases.

Another approach to reduce needed definitions in SimplifiedRoad uses variable default values. These default values are not specified on XML schema level anymore, but they are evaluated by the transformation engine. Table 1 lists the default value *variable*, where this pattern applies. E.g. depending of the infrastructure type, the value *t* for the position of the infrastructure is derived. When guide posts, guard rails and noise barriers are used together for a road, they are placed sided by side by default, without necessarily specifying a position. However, adjustments are always possible. It might be also an option for the future, to allow attribute values as relative values to be interpreted as difference from the default.

The introduced simplifications limit the possibilities compared to OpenDRIVE, but make SimplifiedRoad also much more intuitive. In this way SimplifiedRoad behaves like a classical DSL.

The implementation of the transformation tool is straightforward from this background. The transformation engine is written in Java. As a command line tool, it can be easily integrated in other workflows and (server based) batch processes, while parameter sets are stored in dedicated files.

Example

Figure 2 shows a part of an example for a road section in SimplifiedRoad. The road has a cross section called RQ36 defined in the RAA. Its definition includes the width of the individual lanes and their markings resulting in the total width of the highway of 36 meters. Furthermore, the section has course of an arc and an additional noise barrier.

```
<section length="1000">
  <road crossSection="RQ36" mirror="true"/>
  <course>
    <arc radius="250"/>
  </course>
  <objects>
    <infrastructure
      type="noise barrier" t="0.5"
      repeated="true" right="true"
      center="false" left="true"/>
  </objects>
</section>
```

Figure 2. Section in SimplifiedRoad

The following Figure 3 shows the *partial* translation of the SimplifiedRoad section into OpenDRIVE. Where a horizontal divider is visible, content is hidden for a XML element. Furthermore, only the first lanes are included and the infrastructure is missing at all. With a little different formatting (one element in each line) the shortened example in Figure 2 has nine lines or 293 chars, which are translated into 165 lines or 7,330 chars in OpenDRIVE. The complete test file with surrounding content has 22 lines or 7,852 chars in SimplifiedRoad and 319 lines or 13,588 chars in OpenDRIVE.

Results

Table 2 shows the comparison of several files with test cases in SimplifiedRoad and OpenDRIVE. With SimplifiedRoad these test cases show a reduction of required lines of code and characters to just about 10 percent of the generated OpenDRIVE. Especially parametrization of these files has the potential to boost productivity further because variants of the road section can be generated instantly.

```
<road length="1000.0" id="0">
  <link>
    <successor elementType="road"
      elementId="1" contactPoint="start"/>
  </link>
  <type s="0.0" type="motorway"/>
  <planView>
    <geometry x="0.0" y="0.0" hdg="0.0"
      length="1000.0">
      <arc curvature="0.004"/>
  <elevationProfile>
  <lateralProfile/>
  <lanes>
    <laneSection>
      <left>
        <lane id="7" type="border">
        <lane id="6" type="shoulder">
        <lane id="5" type="stop">
        <lane id="4" type="driving">
          <link>
            <successor id="4"/>
          </link>
          <width a="3.75"/>
          <roadMark type="solid" weight="bold"
            color="white" width="0.3">
            <type>
              <line length="1.0" space="0.0"
                width="0.3"/>
            </type>
          </roadMark>
```

Figure 3. *Partial* transformation into OpenDrive

Figure 2 and Figure 3 show extracts from the file of test case 6 listed in the following Table 2. This table shows different reduction results with diverse examples varying different elements of the motorway. A more complex test case is contained in file 1 that includes different road sections, traffic signs and infrastructure elements. The resulting OpenDRIVE representation of this file is finally rendered as 3D model and shown in Figure 4.

Table 2. Comparison of OpenDRIVE and SimplifiedRoad

File	OpenDRIVE		SimplifiedRoad	
	Lines	Chars	Lines	Chars
1	1,456	85,322	7.55%	5.12%
2	80	3,039	15.00%	14.58%
3	321	12,910	21.18%	10.60%
4	125	4,976	9.60%	9.16%
5	187	7,183	10.16%	8.27%
6	319	13,588	6.90%	5.53%
7	187	7,124	8.56%	7.62%
8	201	8,377	9.95%	8.63%

Conclusion and outlook

The final conclusion is that using a DSL to efficiently model highway sections for test cases is a good choice. It must be admitted, that a format like OpenDRIVE was likely never designed to be edited manually. Compared with a visual track editor using a DSL editor has different challenges for the user, but also using the visual editor is not always intuitive to model correct logic. Using a DSL does not replace classical scenario editors, but complements certain use case creation. It can be expected that in future DSLs will gain importance for creation of virtual tracks for test cases. Planned features for near future include support for highway entries, exits and construction sites as well as country roads.

The procedural generation of OpenDRIVE through the presented DSL will be triggered via a web interface and included into batch processing of scenario test cases. Thus, it can easily be integrated in automated test environments. The PEGAUS project will make use of this approach. PEGASUS is also building a ontology based knowledgebase to fill DSL attributes with realistic values and combinations [Bag18].

Of further interest is a matching between the parameter space of virtual tracks and real-world road data acquired through mobile mapping. This would result in the possibility to include “similar” real-world snippets in the testing of specific scenarios in addition to the synthetically generated data.

Figure 4. 3D model of OpenDRIVE based on SimplifiedRoad

References

[Bag18] Bagschik G., Menzel T., Maurer M.: **Ontology based Scene Creation for the Development of Automated Vehicles**, IEEE Intelligent Vehicles Symposium, 2018

[Bar15] Bartels A., Ruchatz T.: **Einführungsstrategie des Automatischen Fahrens**, *at – Automatisierungstechnik*; 2015

[Deu00] Deursen A., Klint P., Visser J.: **Domain-Specific Languages: An Annotated Bibliography**; *ACM SIGPLAN Notices*; June 2000

[Dup15] Dupuis M., **OpenDRIVE Format Specification, Rev. 1.4,** *VIRES Simulationstechnologie GmbH*, 2015.

[Hun17] Hungar H., Köster F., Mazzega, J., **Test Specifications for Highly Automated Driving Functions: Highway Pilot,** *Autonomous Vehicle Test & Development Symposium*, 2017, Stuttgart, Germany.

[RAA08] Forschungsgesellschaft für Straßen- und Verkehrswesen: **Richtlinie für Anlage von Autobahnen**; 2008; ISBN 978-3-939715-51-1