Systematic Simulation of Fault Behavior by Analysis of Vehicle Dynamics

Artem Kolesnikov[1] Dzmitry Tretsiak[1] Morgan Cameron[2]
[1]ESI ITI GmbH, Dresden, Germany
[2]ESI Group, Rungis, France,
{Artem.Kolesnikov, Dzmitry.Tretsiak, Morgan.Cameron}@esi-group.com

Abstract

A new library for *System Reliability Analysis* for systematic modelling of fault effects in multi-physical systems is introduced. The motivation is outlined as well as a description of the library structure with two helper libraries and an Add-In for semi-automatic fault augmentation. The library is exemplified in the automotive domain with the fault effect simulation of a vehicle model by using a new library for *Driving Maneuvers*. The vehicle model is systematically augmented with connector and component faults for the analysis of different fault effects in vehicle dynamics.

Keywords: Fault, Reliability, Model-based Diagnosis, Driving Maneuvers, Vehicle Dynamics

1 Introduction

The decreasing development time and the rising competition in the automotive industry have led to the increased application of model-based approaches to system engineering. The applied models usually describe the designed ("nominal") behavior of a technical system and require a large amount of time and effort to be adopted because of system deviations. The reasons the physical system can deviate from its nominal behavior are manifold. First of all, the manufacturing of products is only exact to a certain point - there are variations and sometimes systematic deviations from the specified system design. Secondly, during the use of a product, its behavior can change because materials and material pairings are subject to wear, aging, abrupt failures etc. These effects include loss of lubrication, corrosion and hardening of materials. This has necessitated the development of specific models for simulation of fault effects.

Usually, models that represent fault effects are developed to describe the specific non-nominal behavior of a technical system and, hence, often require a large amount of effort to be reused for other failures or requirements. Moreover, the system simulation of multi-domain systems such as a vehicle is usually based on their nominal library components of object-oriented modelling languages. Most of them, such as Modelica®-based libraries, have seen limited attempts to extend models for model-based failure analysis. These include the *Fault Triggering* library, which allows for the insertion of faults into models of existing components (van der Linden, 2014); and the *Fault-Augmented Model Extension (FAME)* library with predefined fault augmentation (de Kleer *et al*, 2013). The latter was described in detail in (Saha *et al*, 2014) and has served as a basis for the systematic fault-augmentation approach presented in this paper.

In the following section the library developed for *System Reliability Analysis* (*SRA*) is described, whose components are the basis for the systematic fault augmentation of the vehicle model. The *SRA* library consists of a basic package containing elementary type and class definitions to parametrize faults, as described in (Gundermann *et al*, 2018). Beyond that, it is structured according to the structure for composable models in the Modelica® language (Minhas *et al*, 2014).

To demonstrate the capabilities of the *SRA* library, a vehicle model was created based on the components of the library *Driving Maneuvers* described in the section 3. As a sample, the preconfigured standard driving test defined by ISO "Severe double lane change" was chosen as described in (ISO 3888-1, 1999). Helper libraries supporting feature extraction and checking of requirements fulfilment are described in section 4. The augmentation of relevant faults in the investigated model and sample analysis of fault effects are shown in section 5.

The systematic fault augmentation methodology provides new capabilities to analyze fault behavior in multi-physical systems in *SimulationX* (www.simulationX.com), by exploiting the multi-domain SRA library and accompanying semi-automatic fault-insertion functionality. The paper motivates and demonstrates these capabilities in the context of model-based failure analyses in automotive applications. Specifically, the vehicle model was augmented with connector, component and signal fault structures of the

SRA library to simulate very different fault effects extracted and analyzed by the helper libraries.

2 The Library for System Reliability Analysis

Throughout this paper the term fault refers to any deviation from the nominal system's behavior. The Modelica® components developed for the *SRA* library in *SimulationX* simulate only the effect of a process on the system's behavior and focus on its dynamical evolution or interaction with other processes. For example, mechanical faults in joints, gears, shafts, springs or clutches because of breakage or slipping lead to the reduction of transmitted forces or torques. The current reduction because of bad or open connections in an electrical system, the mass flow decreases in a hydraulic system because of leakage or obstruction as well as changes of heat flows in a thermic system have comparable behavior and can be similarly structured, modeled and analyzed (Kolesnikov *et al*, 2018). As an example, the three oscillators from different physical domains shown in Figure 1 are similarly structured and augmented with faults in *SimulationX*. This follows from the analogies between the basic electrical, mechanical and hydraulic network elements.

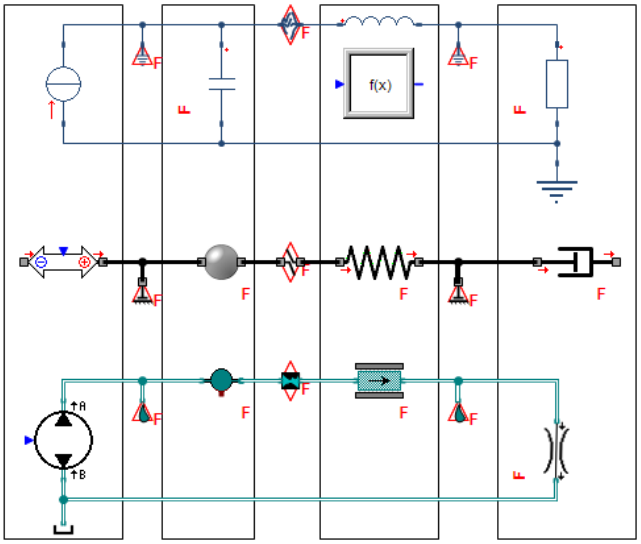


Figure 1. A model of oscillators in SimulationX with electrical, mechanical and hydraulic network elements after the augmentation with connector faults (red connectors) and parametric faults (components with the symbol F).

2.1 Fault Types

Based on the processes and effects outlined above the *SRA* library of *SimulationX* contains appropriate type definitions for two defined types of faults: *continuous* and *discrete*. Both these fault types are specialized classes – *records* containing an editable variable which is dynamic and can change its value during simulation (Fritzson, 2014). By using a type definition for the faults, it is easy to systematically read out and control the faults in a model.

Continuous faults parameterize the strength of a gradual effect. Typical examples are wear and aging phenomena which can lead to a gradually different value of a backlash parameter, or friction coefficient, etc. In the continuous record *fault* this gradual change is translated into a normalized variable named *intensity*, ranging between zero (nominal) and one (maximal effect). If the fault modifies a parameter or variable, the extended continuous record *FaultWithFunction* can be used, which allows for the definition of the functional dependency of parameter change, e.g. multiplicative

$$p = p_0(1 + c_0(scale \cdot intensity)^n) \quad (1)$$

or exponential

$$p = p_0 e^{c_0(scale \cdot intensity)^n} \quad (2)$$

Herein,
p_0 represents the nominal parameter value, c_0 , n represent parameters of the function. The scale parameter is needed to define a typical magnitude of the fault effect, which depends on the specific application, e.g. the mass of the surrounding components, acting forces, etc.

The *discrete* fault type can be only active or not, i.e. the fault is switched on or off. Typical examples are abrupt phenomena, such as failure of an electronic component, or breaking of a mechanical connection. The discrete record *fault* contains the *Boolean active*, which is false in the nominal case, and true if the fault is active.

If the fault modifies a parameter or variable, the extended discrete record *FaultWithFunction* can be used, which allows for the definition of the functional dependency of parameter change, e.g. proportional to the nominal parameter value including setting the prefactor d_0

$$p = p_0 \cdot d_0 \cdot scale \quad (3)$$

or as an alternative value including setting the value p_1

$$p = p_1 \cdot scale \quad (4)$$

2.2 Fault Classes

As shown in Figure 1 a model consists of components potentially taken from different libraries connected together. Fault modelling leads to changes in the behavior of components (*ComponentFaults*), alters the transport of quantities between components (*ConnectorFaults*) or adds new connections (*BridgeFaults*). A snapshot of a model which was augmented with *ComponentFaults*, *ConnectorFaults* and *BridgeFaults* from the SRA library, is shown in Figure 2. One of the resistors is replaced by its fault-augmented counterpart. A *ConnectorFault* modeling a short-to-ground has been added to the connection between the inductor and the capacitor. The connection between the two resistors has been cut by a *ConnectorFault* modeling a loose contact. An additional switchable connection (*BridgeFault*) models a short circuit between two junctions in circuits of consuming components.

DOI 10.3384/ecp19157451

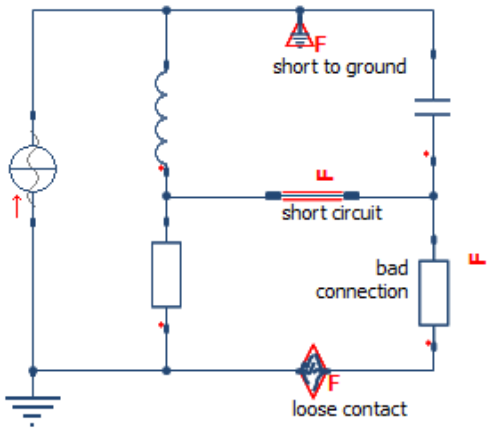


Figure 2. A model of an electrical circuit in SimulationX which has been augmented with faults.

ConnectorFaults can be inserted at (connected) connectors or into existing connections. The first type has two connectors, as shown at the bottom of Figure 2, and can be used to model, for example, the loose contact or the breaking of a mechanical component, or a mechanical obstruction. The second type has only one connection (in Figure 2, at the top), it is added to a connection, and can be used to model, for example, a contact-to-ground fault or a leakage in hydraulics. The models of these faults consist of a fault and a set of equations which depend on the intensity/active(-ity) of this fault, and, in general, on a scale.

BridgeFaults are elements with two connectors as shown in the middle of Figure 2. For each domain there exists one *BridgeFault*. They can be inserted between existing components, adding further connect-statements, if the fault is active. *BridgeFaults* can be used to model, for example, an interaction between two leaky hydraulic circuits.

Fault-augmented models with parametric *ComponentFaults* are fault-augmented counterparts to the nominal *SimulationX* library elements. As an example, Figure 2 shows a fault-augmented element named ComponentFaults.Electricity.Analog.Basic.Resistor, which is an extension of the Electricity.Analog.Basic.Resistor. It contains a fault, which changes the resistance parameter from its default value 1, as defined by intensity, scale, and function in the fault-record. The Fault triggering library models faults in a similar way to those contained within *ComponentFaults* (van der Linden, 2014).

2.3 The Fault Scale

When modeling with the connector or bridge faults, but also during the development of the fault-augmented counterparts of the library elements, the amplitude of the strongest effect (intensity=1, or active=true) differs depending on the surroundings of the system into which the fault is inserted. For example, the frictional torque in a rotational connection which prevents the latter from moving is several orders of magnitude higher in a ship's propeller than in a clockwork mechanism.

In the *SRA* library this phenomenon is captured with the definition of a positive real parameter named *scale* which can be included and defined in the definition of a fault if needed. It should be emphasized that the meaning of the *scale* differs between different faults. Furthermore, *scales* of the same faults in a model can differ in different places e.g. if the model contains both the ship's propeller and the clockwork mechanism.

For a meaningful effect of the inserted faults, it is crucial to find ways to estimate the *scale* for each fault. In the optimal case, one can calculate the *scale* from the amplitudes of variables in the surroundings of the fault as recorded during a single simulation of the nominal behavior. For example, the *scale* of the translational sticking fault is a prefactor to the friction force F_{stick} added to the connection. The latter is defined as $F_{stick} = scale$ for *intensity* 1.

The *scale* should be chosen to be high enough to ensure that the connection stops moving when the fault intensity is equal to one. On the other hand, it should be low enough that one is not faced with numerical issues, because the system is stiff or the solver has trouble integrating when the fault is switched on during a simulation. In the optimal case, the scale value should be set to ensure that the range of the friction force over increasing intensity is such that for low intensities it already has some (recognizable) effect on the connection, but does not already prevent it from motion at all (sensitivity to fault intensity). Estimating the *scale* can be challenging even in the seemingly simple case of the friction force described above. To this end, several algorithms are currently under investigation using base unit, force, energy or power values.

2.4 Semi-Automatic Fault Augmentation

Augmenting a model of the nominal system with various faults can be time-consuming and error-prone. Moreover, when replacing the nominal type of a component by its fault-augmented counterpart, one must ensure that the parameters are kept or modified correctly.

To support the fault-augmentation process, a dedicated user interface to *SimulationX* has been developed. The tabbed interface guides the user through the whole augmentation and analysis process in an intuitive way. Once the augmentation is completed, faults of interest can be selected for subsequent analysis. The augmentation and analysis workflow can be summarized by the steps described below.

Definition of the candidate: First of all, the user must define the candidate components, i.e. all those components and connections in the model, which are to be fault-augmented. Additionally, it can be decided whether to exclude or include a certain class of faults (connector or component faults), or whether to insert faults only from

specific domains (mechanical, electrical, etc.) as shown in Figure 3.

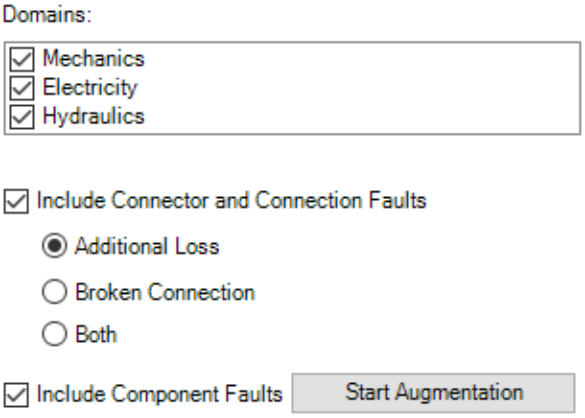


Figure 3. Selection of fault classes and domains supported by the SRA Add-In.

Augmentation: The actual fault-augmentation can be executed by clicking the "Start augmentation" button. The process changes the model structure as shown in the right-hand side of Figure 4. "Nominal" components are replaced by their fault-augmented counterparts. In this example, a fault with two pins (broken) is added for each connected connector was added, and another fault with one pin (sticking) was added for the whole connection.

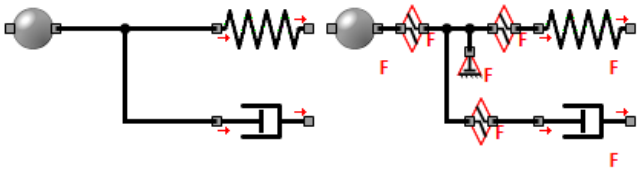


Figure 4. SimulationX model before (left) and after (right) the automatic fault augmentation for mechanical components.

3 Driving Maneuvers Library for Analysis of Vehicle Dynamics

A vehicle model for the fault effect simulation was built using a new library for *Driving Maneuvers* in *SimulationX*. The modular library contains various chassis model elements, wheel and axle suspensions, wheel elements with tire model, driver models, track and environment components and complete vehicle models. The library elements are parameterized and validated using real experimental data (Tretsiak *et al*, 2018).

The library structure has an open, user-extensible, object-oriented model architecture thanks to its compatibility with the Modelica® modelling language. The library structure, shown in Figure 5, includes following subsections described in detail below:

- Environment, Drivers and Maneuvers
- Bodies and Wheels
- Suspensions and Axles with Suspensions
- Vehicles

The real-time capable curve-based vehicle models can be used in combination with elements such as powertrains or brakes from other libraries of *SimulationX* (*Mechanics*, *Power Transmission (1D/MBS)*), and with their fault-augmented counterparts from the *SRA* library.

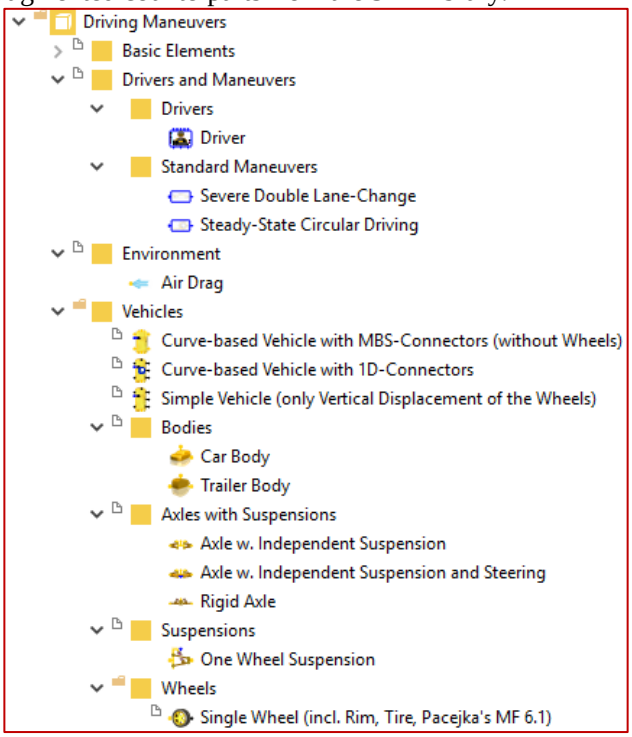


Figure 5. Structure of *Driving Maneuvers* library in SimulationX

3.1 Environment, Drivers and Maneuvers

The sub-library *Driver* contains the two-level driver model with anticipatory (open-loop) and compensatory (closed-loop) control for vehicle longitudinal and lateral dynamics, considering a curvature difference, steering angle difference and lateral displacement (Figure 6), which is generally based on Donges' model (Donges, 1978).

Driver models can be configured individually, with the option of externally-defined speed and steering profiles. The output signals are steering, load and brake demands (steer, gas, brake). Maneuver element types allow setting of preconfigured standard driving tests defined by International Standard Organization, for instance:

- (Severe) Double Lane Change (ISO 3888-1, 1999; ISO 3888-2, 2002)
- Steady-State Circular Driving (ISO 4138, 2012)

The driving track and designation of maneuver sections are depicted in Figure 7. There is the possibility to set customer-specific test tracks.

DOI 10.3384/ecp19157451

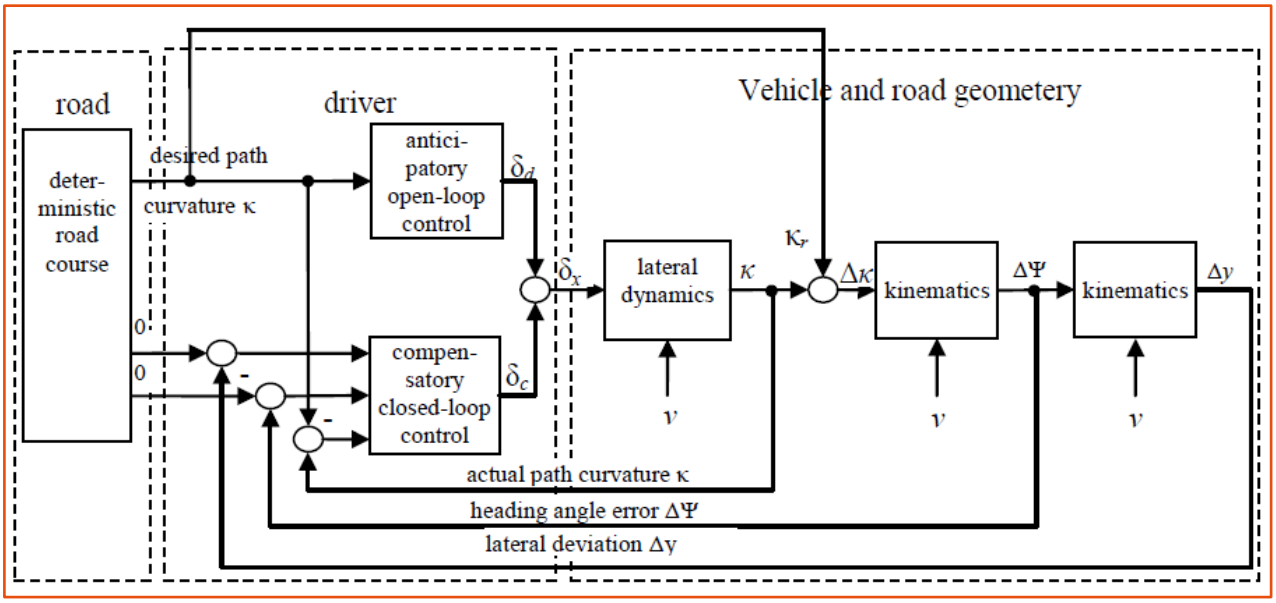
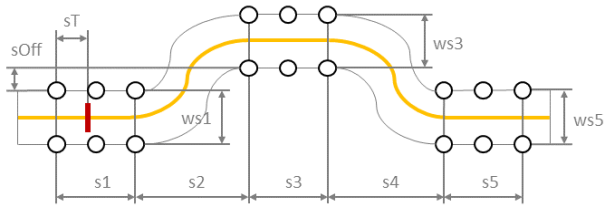


Figure 6. A two-level model of driver steering behavior

a



b

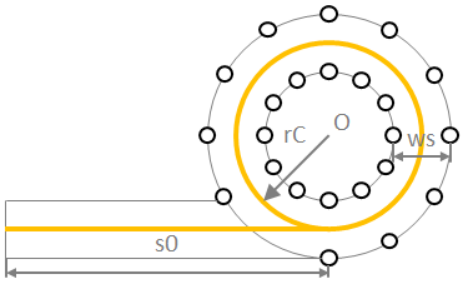


Figure 7. Maneuver tracks and designation of sections: a – (*Severe) Double Lane Change*; b – *Steady-State Circular Driving*

The model element *Air Drag* provides air drag forces and torques in all directions and gives the possibility to define corresponding air drag coefficients:

- Only Cw-coefficient
- All air drag coefficients (cx,cy, ...) scaled on the Cw-value
- All air drag coefficients
  - Air drag force coefficients
  - Air drag torque coefficients

3.2 Bodies and Wheels

Vehicle bodies can be visualized by default shapes or user-defined imported CAD geometry with corresponding scaling in all directions.

By default, three car types are available for visualization of a vehicle body:

- Compact car
- Station wagon
- Sport utility vehicle

A single CAD geometry is used for the representation, with corresponding scaling coefficients for each vehicle type. The car body model element can be used to represent, for instance, a truck cabin or a bus saloon, requiring only corresponding CAD data. The trailer body component has the same functionality as the car body. Figure 8 presents the 3D model visualization with the above-described library elements.

Figure 8. Visualization of *Car Body* and *Trailer Body* model elements

The tire is modelled according to Pacejka's MF 6.1 (Bakker *et al*, 1987) and is used in a single wheel element, which includes a corresponding visualization body.

3.3 Suspensions and Axles with Suspensions

These parts contain predefined curve-based models of wheel and axle suspensions with anti-roll bars.

Suspension kinematics are defined by zero-order kinematics parameters (e.g. value) or wheel lift trajectories (e.g. camber angle over vertical displacement of a wheel center).

Different steering types are available:

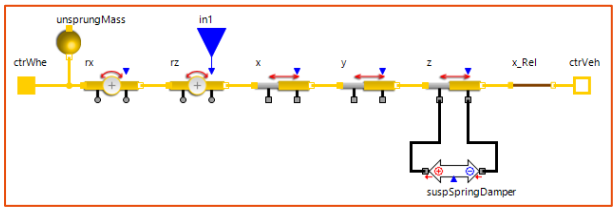
- With constant or variable steering gear ratio

- With direct input of wheel rotation angles over a steering wheel angle

Five degrees of freedom of one wheel are constrained in the case of an independent suspension.

Figure 9 shows the model structure of a one-wheel suspension and an axle with independent suspension and steering.

a



b

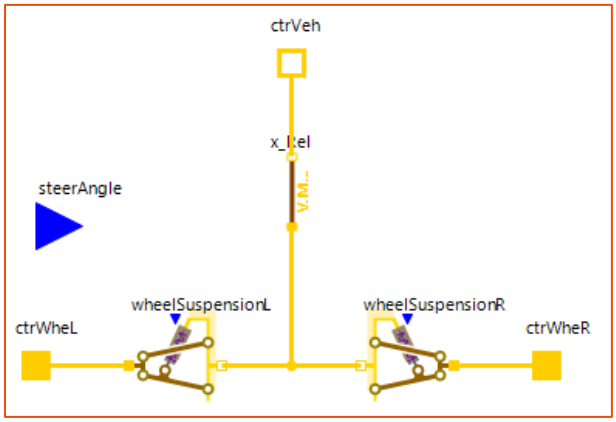


Figure 9. Structures of suspension models: a – one-wheel suspension; b – axle with independent suspension and steering.

The rigid axles library allows the modelling of specific axles with suspensions also for heavy-duty trucks and buses:

- Rigid axle with conventional suspension
- Complete rigid axle
- Swing axle (without suspension)

A mirror function for parameterization of symmetric suspensions facilitates this routine process.

3.4 Vehicles

The real-time capable curve-based vehicle models with 1D and MBS connectors are based on the following library elements:

- Car body
- Axle with independent suspension
- Axle with independent suspension and steering
- Air drag and motion sensor

Up to six load masses for passengers and luggage can be defined.

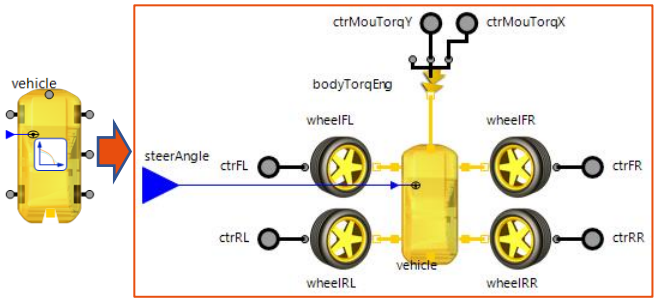
Engine and powertrain mount torques can be considered, depending on the engine position (longitudinal or transverse).

There is a signal connector for steering input and different variants are available to set wheel parameters:

- 4 equal wheels
- 2 equal front wheels, 2 equal rear wheels
- Free definition (individual settings)

The corresponding diagram views of characteristic curve-based vehicle models with 1D and MBS connectors are presented in Figure 10.

a



b

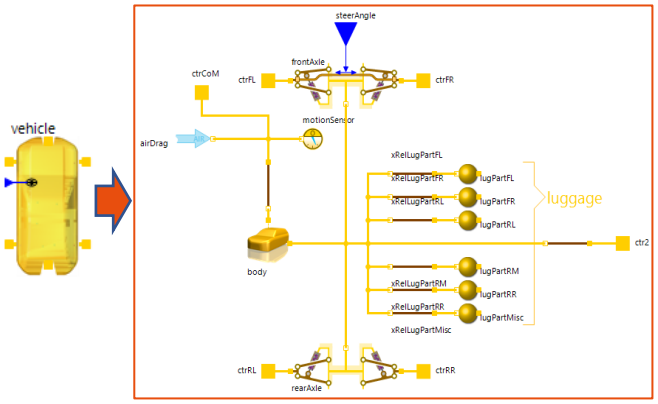


Figure 10. Structures of curve-based vehicle models: a – curve-based vehicle with 1D-connectors; b – curve-based vehicle with MBS-connectors (without wheels).

4 Analyzing Faults

As well as structuring the modeling and insertion of faults, it is also helpful to structure and systematize all output quantities of the model that constitute related information about fault effects. Moreover, the checking of requirements fulfillment in the model-based applications must consider not only pre-defined criteria, but also scenarios executed for them. The additional libraries for special signal analysis in *SimulationX* presented in this section serve as helpers for this task.

4.1 Signal Feature Extraction

For model validation, the model output is compared to real systems' data. Such data is, in general, sampled, can sometimes be averaged, noisy, or even in frequency-domain representation. To prepare the model output in the same way, one needs ways to extract features from the variables.

To avoid dealing with (different) sampling rates and the quality of time series, or to use single-valued data from

DOI 10.3384/ecp19157451

output time series, meaningful features must be extracted. Furthermore, requirements fulfillment is categorized by blocks returning a single value: successful, failed, or undecided/not clear. The determination of these categories (for more details see subsection 4.2) is based on calculations of output variables, which are again features of the latter.

The library for signal feature extraction contains helper blocks that support feature extraction. Helper blocks shown in Figure 11 are provided for extracting features such as time interval between two pulses or two events and band check. The list is extended based on the examples studied. One important requirement is that all features are insensitive to numerical side effects. For example, the extraction of the time between two pulses should not pick a "numerical" peak, the height of which is dependent on tolerance or other numerical artefacts. The extraction of the time between two mean values should be possible over restricted time spans to avoid a dependency on the overall simulation time (e.g. the average velocity of the vehicle decreases to zero, because the model driving scenario depicts an unnecessary amount of time span after the vehicle has come to rest).

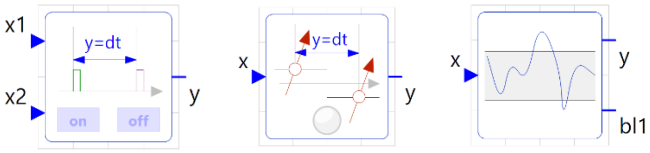


Figure 11. Time Interval bw. 2 pulses or 2 events and band check elements

The additional sub-packages ChecksInFixedWindow, ChecksInSlidingWindow, SignalAnalysis from (Otter et al., 2015) - although motivated by a completely different application - serve similar purposes.

4.2 Requirements Fulfillment Analysis

The sample vehicle model analyzed in the next section addresses the question of the requirements fulfillment when it is subject to faults. Stated differently: which fault or combination of faults leads to the violation of pre-defined criteria. An example of such a criterion, taken from the drive train example, is: The vehicle shall be able to brake from 100 km/h to stop in 7 Seconds. To assess this in the model, output variables (velocity, time) are read, features are extracted (velocity at 7 seconds after start of braking) and tested against the criterion “stopping”. The formalization of the last step is supported by the elements modeled in the library *Requirements Fulfillment.*

The basic definition of a criteria contains an array of assertions as an input. In the example of the velocity test this array has one entry: $v(t_{StartBraking} +7)$[km/h]. If this entry is lower than zero, the criterion is fulfilled, if not, it is violated. If the array of the assertions contains more than one entry, it has to be defined whether they are connected by an AND (i.e. all must be fulfilled to fulfill the whole criterion), an OR (i.e. only one must be fulfilled), or NOT (i.e. all must be violated to fulfill the whole criterion). Sometimes it does not make sense to test a criterion at all - for example, if there was no ignition, there is no startup time. To address this scenario, the requirements fulfillment elements contain a second input named *validity indicator* defined in a similar way as the *assertion* - i.e. if this variable is positive, the validity is given and the assertion can be tested, if not, the assertion does not need to be tested. The integer output *requirements fulfillment* is based on the validity indicator and the assertions, and is restricted to three values, which represent the categories as listed in Table 1.

Table 1. Categories of the output of the *Requirements Fulfillment* elements based on the incoming validity and insertion.

requirements fulfillment	*category*	*conditions*
1	fulfilled	validity>0, assertion>0
0	violated	validity>0, assertion<0
-1	undecided	valididy<0

For convenience, the assertions are fed to an output variable assertionsOut. The output *requirements fulfillment* can only tell if the simulation fulfilled or violated the criterion, but not how far it was from violating or fulfilling it. To have a measure for this, the relevant continuous variables should be analyzed. Sometimes it is important to have information about how close to violating/fulfilling the specification, e.g. since due to noises/variations which are not in the model the outcome might not be robust. Furthermore, it proves helpful to have some information about whether an increasing fault intensity influences an assertion. This information cannot be obtained from the integer output.

Figure 12 contains elements of the *Requirements Fulfillment* library. Currently, the connection of assertions via AND, OR and NOT is possible, since any logical expression can be brought into either of these forms (disjunctive/conjunctive normal form, see e.g. (Hazewinkel, 1994)). More flexible definitions of *Requirements Fulfillment* components will be developed as applications demand.

The *Modelica_Requirements* library presented in (Otter et al., 2015) contains a similar 3-valued logic to those presented here. Its motivation comes from a connection between system simulation and the formal definition of requirements. For the *Modelica_Requirements* library an extension of the Modelica language is proposed to handle the 3-valued temporal logic (satisfied, violated, undecided). The formalization of proving the logical expressions built up from validity and assertions (as described above) could be improved if the handling of the 3-valued logic becomes

part of the language standard. However, it is not necessary in our case.

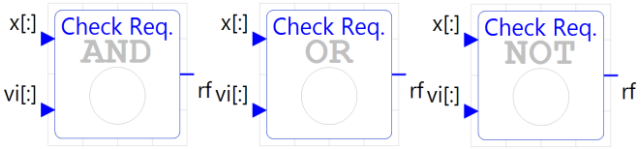


Figure 12. Checking requirement criteria – ALL, ONE or No Requirements must be met.

5 Simulation and Analysis of Vehicle Dynamics with Fault Effects

To demonstrate the application of the *SRA* library in *SimulationX* for considering fault effects by the analysis of vehicle dynamics, the model for a preconfigured standard driving test double lane change is used. It is a well-known test method which is generally used for subjective evaluation of vehicle dynamics (ISO 3888-1, 1999). The driver can use a vehicle brake system passing through maneuver track. Consequently, there is the possibility to simulate fault effects in the brake system and analyze its influence on a vehicle dynamics and road-holding ability.

5.1 Fault-Augmented Vehicle Model

The vehicle model is built using the model elements of *Driving Maneuvers* library, described in the section 3. Figure 13 shows the corresponding diagram view of the vehicle model augmented with fault elements from the SRA library and is described below.

The model structure includes the maneuver element described in the subsection 3.1, whose input signal is the driven route of the vehicle with a transverse engine. The simplified vehicle model includes the torque source for front drive wheels, whose value is defined by a corresponding curve, and the differential element. The corresponding mount torque of the car with transverse engine is also considered.

The output from the maneuver component is connected to the driver element and provides the closed-loop driver model with the information about the desired route curvature and desired vehicle speed. The instantaneous curvature and car speed for the driver model are received from the fault-augmented vehicle element. The output signals from the driver element are: steering wheel angle demand and load and brake requirements. Wheel brakes provide corresponding brake torques based on the brake signal from the driver model element.

For the fault augmentation by the SRA Add-In described in the section 2.4, the elements of the brake system and the vehicle were selected as candidate components, as shown in Figure 14. The brake system was augmented with ConnectorFaults (sticking 1-4) to analyze the following failures:

- Drop of a brake pad friction coefficient, reducing the braking torque acting on a wheel. It can be caused by uneven wear of brake pads or their surface contamination with oil film, dirt, etc.
- Drop of an actuation force of brake drive, reducing the braking torque acting on a wheel because of a drop of a hydraulic pressure caused by pipe rupture or leakage in a brake drive.
- Jamming the wheel brake, decreasing vehicle stability and control because of brake lining wear or wear of sealing ring of a wheel hydraulic cylinder.

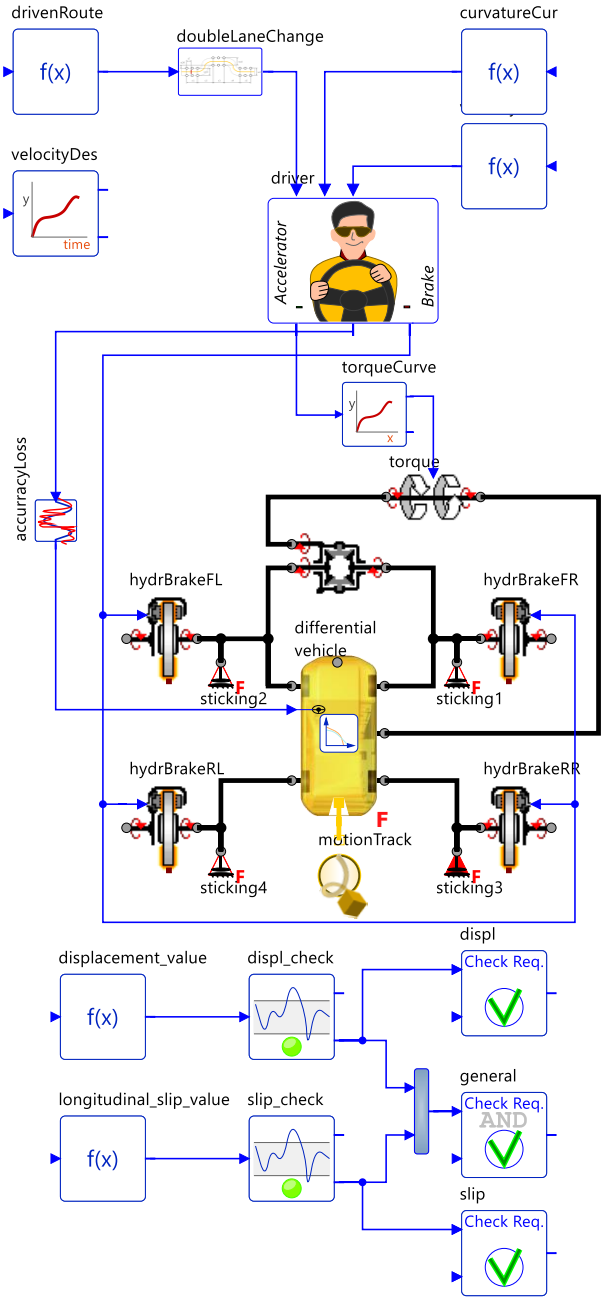


Figure 13. Vehicle model with augmented connector, signal and component faults

The vehicle component was replaced with its fault-augmented counterpart (*ComponentFault*) to investigate

DOI 10.3384/ecp19157451

the friction change in the contact between wheel and road because of:

- Change of wheel pressure
- Tire tread wear

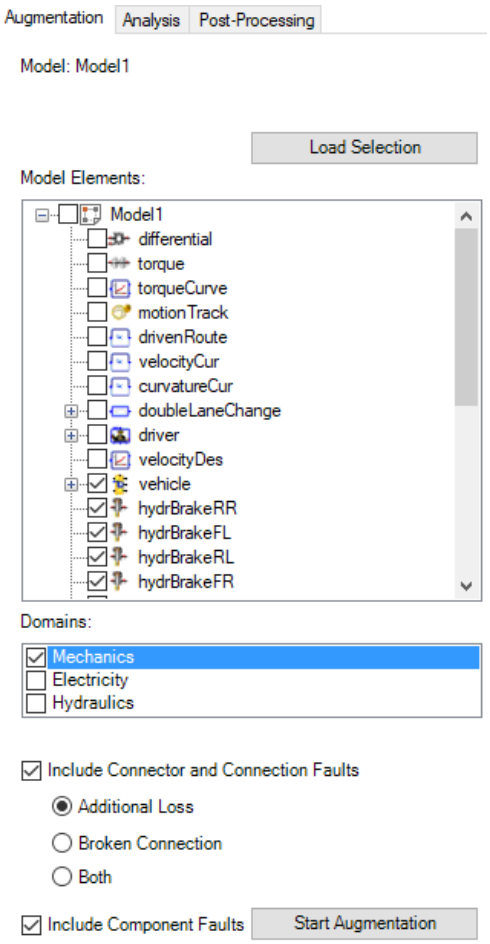


Figure 14. Selection of model components supported by the SRA Add-In.

The component *accuracyLoss* from the *SRA*-library was placed in the model between the *driver* and *vehicle* components to consider signal fault (control signal distortion) and to investigate its influence on the vehicle's behavior during the standard driving test with steering and braking failures.

5.2 Sample Analysis of Vehicle Dynamics with Fault Effects

As was mentioned above, the double-lane change maneuver is used for subjective determination of a vehicle obstacle performance, which is a part of vehicle dynamics and road-holding ability. If a vehicle is in the track frames during the test, then it is assumed that it has a sufficient obstacle performance and road-holding ability. The model allows visual analysis based on a vehicle trajectory through the maneuver track (Figure 15). However, this kind of analysis inefficient for assessment of multitude fault combinations and cannot provide sufficient information for the post-processing.

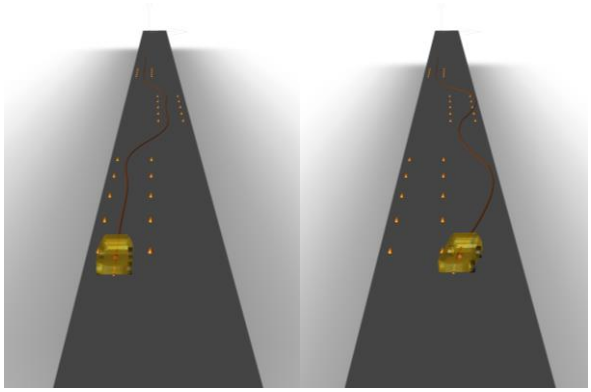


Figure 15. Vehicle trajectories by successful (left) and unsuccessful (right) passing through maneuver track

To systematically analyse fault effects in the brake system, components from the helper library *Signal Feature Extraction* were used as shown in the lower-part of Figure 13. These components (*displ_check* and *slip_check* in the model) check information about passing of a vehicle through maneuver based on the displacement of a vehicle in XY-plane and the longitudinal slip of vehicle front right wheel (Figure 16).

Figure 16 shows the deviation of the selected output values of the fault-augmented model from its nominal behaviour (without fault) because of the activation of the connector faults. The connector faults (components *sticking 1-3* in Figure 13) decreases the braking torque in the right rear wheel (fault in RR) or front left wheel (fault in FL) up to 50% or jam the front right wheel brake (fault in FR).

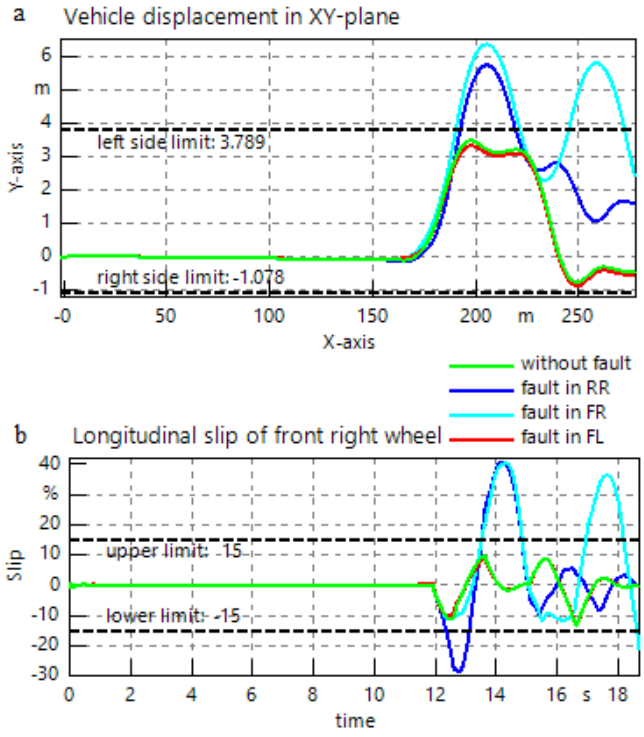


Figure 16. Selected output values of the fault-augmented model with different faults in the brake system

The limits depicted in Figure 16 define the parameters of the components *displ_check* and *slip_check* that serve to automatically extract the important features for the *Requirements Fulfillment* components (*displ*, *slip* and *general*) in the vehicle model. In the "Post-processing" tab of the SRA Add-In output quantities in these components provide a means to analyse requirements fulfillment under different faults. Figure 17 shows the results of the fault effect analysis with different fault intensities (between 0 and 1) and checked requirements (successful – 1, unsuccessful - 0). The analysis results can be saved in CSV format or uploaded to the further investigation in the data analytics tool ESI MINESET (Mineset).

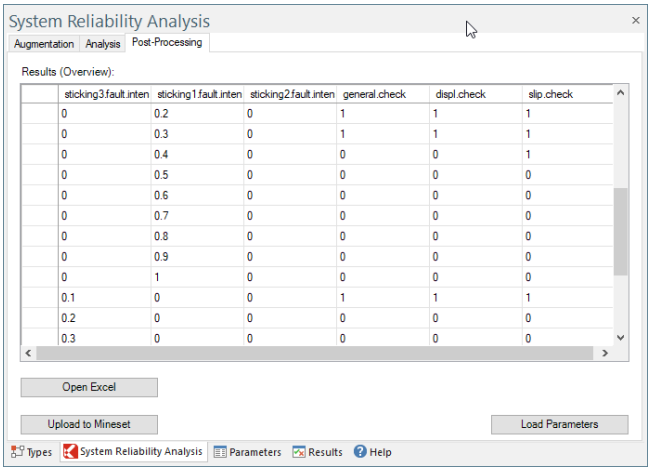


Figure 17. Results of requirement fulfillment analysis for different fault combinations.

Conclusion

In this publication, we introduced the *System Reliability Analysis* library, which enables the user to model and simulate physical systems outside their nominal behavior in a systematic way. We motivated the utility of the *System Reliability Analysis* library with a vehicle model from the *Driving Maneuvers* library having different failures.

Based on the model, the broad range of applicability of the fault effect analysis in multi-physics domains was outlined. Illustrative simulation results - based on the given sample model - were presented and show the possibility of systematic simulation of fault behavior by analysis of vehicle dynamics using new *System Reliability Analysis* and *Driving Maneuvers* libraries.

In addition, the motivation for and implementation of helper libraries (*Signal Feature Extraction*, *Requirements Fulfillment*) and tools (*SRA* Add-In) was described.

References

Edmund Donges. A two-level model of driver steering behaviour. *Human factors*, 20(6), 1978, pp. 691-707.

E. Bakker, L. Nyborg, and H. B. Pacejka. Tyre Modelling for Use in Vehicle Dynamics Studies. *Society of Automotive Engineers*, January 1987.

P. Fritzson. Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach," Nov 2014, Wiley-IEEE Press, ISBN: 978-1-118-85912-4.

J. Gundermann, A. Kolesnikov, M. Cameron, and T. Blochwitz. The Fault library - A new Modelica library allows for the systematic simulation of non-nominal system behavior. *Proceedings of the 2nd Japanese Modelica Conference*, Japan, Tokyo, May 17-18, 2018, Industrial Paper, pages 161-168, 2018, doi: 10.3384/ecp18148161.

Michiel Hazewinkel. Encyclopaedia of Mathematics (set). Encyclopaedia of Mathematics. Springer Netherlands, 1994. ISBN 9781556080104. URL https://books.google.de/books?id=uxUBQwAACAAJ.

International Organization for Standardization, 1999. ISO 3888-1 Passenger cars — Test track for a severe lane-change manoeuvre — Part 1: Double lane-change. Geneve: ISO.

International Organization for Standardization, 2002. ISO 3888-2 Passenger cars — Test track for a severe lane-change manoeuvre — Part 2: Obstacle avoidance. Geneve: ISO.

International Organization for Standardization, 2012. ISO 4138 Passenger cars - Steady-state circular driving behaviour - Open-llop test methods. Geneve: ISO.

J. de Kleer, B. Janssen, D. G. Bobrow, T. Kurtoglu, et al. Fault augmented modelica models. *24th International Workshop on Priciples of Diagnosis*, Jerusalem, Israel, pages 71-78, 2013.

A. Kolesnikov, M. Andreev, and A. Abel. The Fault-Augmented Approach for the Systematic Simulation of Fault Behavior in Multi-Domain Systems in Aerospace. *SAE Technical Paper* 2018-01-1917, 2018, doi: 10.4271/2018-01-1917.

F. L. J. van der Linden. General fault triggering architecture to trigger model faults in modelica using a standardized blockset. *10th International Modelica conference*, number 96 in Linköping Electronic Conference Proceedings, LiU Electronic Press, pages 427-436, 2014, URL http://elib.dlr.de/90576/

R. Minhas, J. de Kleer, I. Matei, B. Saha, at al. Using fault augmented modelica models for diagnostics. *Proceedings of the 10th International Modelica Conference*, March 10-12; 2014; Lund; Sweden, number 96, Linköping University Electronic Press; Linköpings universitet, pages 437-445, 2014.

Martin Otter, Nguyen Thuy, Daniel Bouskela, Lena Buffoni, Hilding Elmqvist, Peter Fritzson, Alfredo Garro, Audrey Jardin, Hans Olsson, Maxime Payelleville, et al. Formal requirements modeling for simulation-based verification. In *Proceedings of the 11th International Modelica Conference*, Versailles, France, September 21-23, 2015, number 118, pages 625–635. Linköping University Electronic Press, 2015.

Mineset: https://cloud.esi-group.com/analytics

B. Saha, T. Honda, I. Matei, E. Saund, at al. A model-based approach for an optimal maintenance strategy. *Second European conference of the prognostics and health management society*, pages 521-531, 2014.

D. Tretsiak, T. Wiedemann, C. Bellanger, and F. Kocksch. Modeling of vehicles with varying level of detail for system simulation - Development of a modular chassis model kit including a consistent parameterization process. *Proc. of ESI SimulationX Conference*, Dresden, Germany, 2018.