A Design Framework for Developing a Reconfigurable Driving Simulator

Bassem Hassan
Project Group Mechatronic Systems Design
Fraunhofer Institute for Production Technology IPT
33102 Paderborn, Germany
Bassem.Hassan@ipt.fraunhofer.de

Jürgen Gausemeier
Heinz Nixdorf Institute
University of Paderborn
33102 Paderborn, Germany
Juergsen.Gausemeier@hni.uni-paderborn.de

Abstract - **Driving simulators have been used successfully in various application fields for decades. They vary widely in their structure, fidelity, complexity and cost. Nowadays, driving simulators are usually custom-developed for a specific task and they typically have a fixed structure. Nevertheless, using the driving simulator in an application field, such as the development of the Advanced Driver Assistance Systems, requires several variants of the driving simulator. Therefore, there is a need to develop a reconfigurable driving simulator, which allows its operator to easily create different variants without in-depth expertise in the system structure. In order to solve this challenge, a design framework for developing a Task–Specific reconfigurable driving simulator has been developed. The design framework consists of a procedure model and a configuration tool. The procedure model describes the required development phases, the entire tasks of each phase and the used methods in the development. The configuration tool organizes the driving simulator's solution elements and allows its operator to create different variants of the driving simulator by selecting a combination of the solution elements, which are like building blocks. The design framework is validated by developing three variants of a reconfigurable driving simulator. This paper includes a modified procedure model, more detailed analysis of the state of the art and new results comparing with the previous published paper "Concept for a Task–Specific Reconfigurable Driving Simulator".**

Keywords - Advanced Driver Assistance Aystems (ADAS); reconfigurable driving simulator; confiuration mechanis; solution elements; bulding blocks; variants

I. INTRODUCTION

The development and testing of the in-vehicle systems, such as Advanced Driver Assistance Systems (ADAS), is a challenge due to their complexity and dependency on the other vehicle systems, initial conditions, and the surrounding environment [1] [2]. The testing of ADAS in reality leads to significant efforts and cost. Therefore, virtual prototyping and simulation are widely used instruments in the development of such complex systems [3].

Virtual prototyping is well-established in facilitating the development of new vehicle systems and components [4]. It is the process of building, simulating, and analyzing virtual prototypes. Virtual prototypes are the digital representations (models) of the real prototypes. It allows the verification of the properties and the functions of the product in the early development phases without having to build a real prototype. This saves time and costs [5]. One of the most useful virtual prototyping tools in the automotive field are driving simulators.

Driving simulators allow the ADAS developer to investigate the interaction between the human driver, the Electronic Control Unit "ECU" virtual prototype and the vehicle, while the human driver steers a virtual vehicle in a virtual environment. Driving Simulators rank among the most complex testing facilities used by automotive manufacturers during the development process. They are based on close collaboration of different simulation models at runtime [6]. These partial models represent dedicated aspects of the different vehicle components, as well as the vehicle environment [7].

Driving simulators vary in their structural complexity, fidelity and their cost. They range from simple low-fidelity, low-cost driving simulators such as computer-based driving simulators to complex high-fidelity, high-cost driving simulators such as high-end driving simulators with complex motion platforms [8].

Nowadays, existing driving simulators are usually task-specific devices, which are individually custom-developed by suppliers for a specific usage during the ADAS development. For example, a task-specific driving simulator is typically used for testing the ADAS main functionality without considering the human-machine-interfaces and another task-specific driving simulator is used for investigating different variant of human-machine-interfaces. These driving simulators can only be configured by a driving simulator expert. This is done by exchanging one or more of their entire components. Existing driving simulators do not allow their operator to change the system architecture or to exchange simulation models without in-depth knowledge of the driving simulator's components and structure.

The development of a driving simulator is a costly and complex task; the testing and training of ADAS often requires more than one configuration of a driving simulator. That is why there is a need for developing a reconfigurable driving simulator that allows the system operator to reconfigure it in a simple way without in-depth expertise in the system.

This work is based on a previous paper of the authors "Concept for a task–specific reconfigurable driving Simulator" [1]. However, this paper describes a modified procedure model, more detailed analysis of the state of the art, and presents the new reached results in more details.

II. Reconfigurable Driving Simulators Definition

In most of existing driving simulators' descriptions or brochures, they are defined as a "reconfigurable driving simulator". Therefore, the term "reconfigurable driving simulator" has to be clearly-defined with the help of three questions: "Which driving simulator components could be reconfigured?", "Who can reconfigure the driving simulator?" and "What is the difference between a configurable and a reconfigurable driving simulator?" Based on the answers of the questions, the term "Reconfigurable Driving Simulator" will then be defined.

Which driving simulator components could be reconfigured? The term "reconfigurable driving simulator" is sometimes misused instead of using the term "driving simulator with exchangeable components" or the term "driving simulator with parameterized models". Driving simulators consist of various components. These components are classified into three categories: hardware, software, and resources. There are many driving simulators which have exchangeable hardware components, e.g., vehicle mock-up, motion platform, and visualization system. Other driving simulators have exchangeable software components, e.g., vehicle model, traffic model, etc. Most driving simulators have parameterized simulation models, e.g., a parameterized vehicle model to simulate different vehicle types, parameterized traffic models to simulate different traffic scenarios, etc.

Who can reconfigure the driving simulator? The term "reconfigurable driving simulator" is sometimes misused instead of using the term "modular driving simulator" or "configurable driving simulator". Many driving simulators could be customized individually by their manufacturer according to the customer requirements. These are "modular driving simulators". Some driving simulator components could be exchangeable or some components could be added or removed. These are configurable driving simulators, which can be reconfigured or upgraded only by their manufacturer or developer.

What is the difference between a configurable and a reconfigurable driving simulator? A configurable driving simulator means that a variant of a driving simulator could be created by selecting its entire components during the development, but its structure and/or its entire components cannot be changed after the development. However, a reconfigurable driving simulator structure and entire components can be changed after the development. In this paper, we describe a reconfigurable driving simulator development approach in means of, adding, removing, modifying, and resampling the components of the driving simulator is granted after the development.

Reconfigurable driving simulator definition: A driving simulator is reconfigurable when different configurations can be used optimally in different tasks at different times. The reconfiguration should be feasible by the operator without in-depth expertise in the system structure. The operator can create different configurations by changing the system structure (adding or removing some of its entire components) and by exchanging the entire system components with other suitable components.

III. Related Work

There are thousands of driving simulators spread all around the globe. They are complex mechatronic systems and include different technologies, which widely range from computer graphics to controlling a complex motion platform. The publications about driving simulators usually take one technology into consideration or just a partial aspect of developing a specific driving simulator. The state of the art in this section will only consider the publications that are related to the development methods of driving simulators and the previous approaches towards developing a reconfigurable driving simulator.

This section surveys an existing driving simulator selection method and previous approaches towards developing a reconfigurable driving simulator.

A. The Driving Simulators Selection Method according to Negele[6]

Negele developed a method called the "Application Oriented Conception of Driving Simulators for the Automotive Development". He considered driving simulators as one of the most complex test rigs used in the automotive development. The development of a driving simulator requires a wide expertise in different technologies and disciplines, which widely range from the visualization techniques to platform motion control. This essential know-how is not in the core competence of the automotive manufacturer. Therefore, driving simulators, which are used as automotive test rigs, are usually developed by driving simulator suppliers. Nevertheless, it is tough for automotive engineers, who do not have a basic knowledge of driving simulator technologies to select and specify a driving simulator that fits with a specific-task [6].

Therefore, Negele developed a method, which allows automotive engineers to formulate the requirements and specifications of a driving simulator for a specific application. The main objective of the method is to define the relationships between the automotive applications and driving simulators' specification [6].

Automotive engineers could select a driving simulator type based on two main criteria: a driving task category and a driver stimulus-response mechanism, according to the application of the required driving simulator.

The driving tasks are categorized into primary tasks, secondary tasks, and tertiary tasks. The primary tasks consist of vehicle navigation, vehicle guidance and vehicle stabilization. The driver stimulus-response mechanisms are categorized into the following: skills-based responses, which are senso-motoric responses (e.g., acceleration or steering), rule-based responses (e.g., driving slower in a curve) and knowledge-based responses (e.g., route planning with the help of paper maps) [6].

The driving simulator application should be defined by means of the following: a driving task category (Which driving tasks should be investigated?) and a driver stimulus-response mechanism (Which driver stimulus-response

mechanism is relevant?). For example, if the driving simulator application is the testing of vehicle dynamics, then the application is focusing on a primary driving task (vehicle stabilization) and investigating a skills-based response of the vehicle driver [6].

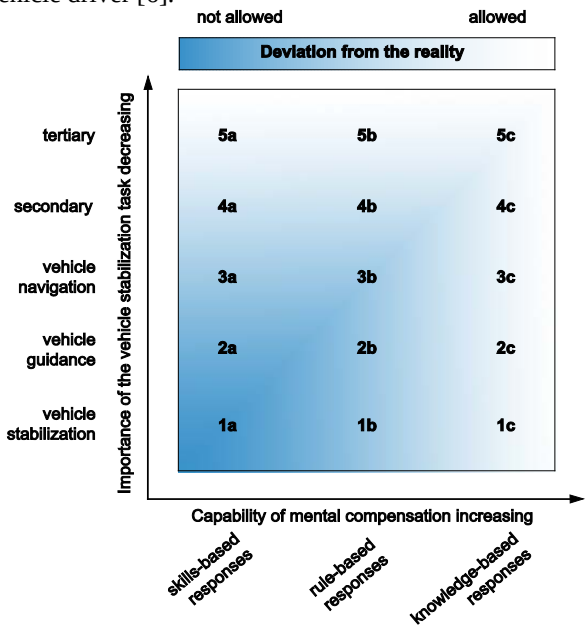

Figure 1. Scheme for classifying driving simulator applications [6].

Fig. 1 shows the intersections matrix between the five driving tasks categories: (vehicle stabilization, vehicle guidance, vehicle navigation, secondary tasks, and tertiary tasks) and the three driver stimulus-response mechanisms: (skills-based responses, rule-based responses, and knowledge-based responses). These result in 15 types of driving simulators, which are marked from 1a to 5c [6].

Each driving simulator type is described by a profile table. The profile table specifies the entire components of the driving simulator variant. Negele divided the simulator into 26 components grouped into 6 groups.

The method of Negele allows automotive engineers to formulate the requirements and the specifications of a task-specific driving simulator. The focus was on how to specify the requirements of a driving simulator to fit with a specific task. He did not consider the reconfigurability of driving simulators and he did not mention a driving simulator's development method.

Nevertheless, the method is useful as a preliminary work for driving simulator operators. They can use Negele's method to specify the preferred driving simulator's requirements and its entire components, then they can use the design framework described in this work in order to create a specific driving simulator variant.

B. *Existing Low-Level Driving Simulators*

Low-level driving simulators have restricted fidelity, high usability and they are usually low-cost driving simulators. Typically, they have a single display that provides a narrow horizontal field of view and a gaming steering wheel as a Human-Machine-Interface (HMI) [9].

The following sections describe one previous approach towards developing low-level reconfigurable driving simulator.

A Modular Architecture based on the FDMU Approach: Filippo et al. had developed "a modular architecture for a driving simulator based on the FDMU approach". This approach describes a modular and easily configurable simulation platform for ground vehicles based on the Functional Digital Mock-Up approach (FDMU). FDMU is a framework developed by the Fraunhofer Institute. The framework consists of a central component called "Master Simulator", which connects different components through an application called "Wrapper". Each module communicates with the master simulator through its own wrapper application and a standardized Functional Building Block (FBB) interface. Fig. 2 shows the basic scheme of the FDMU architecture [10].

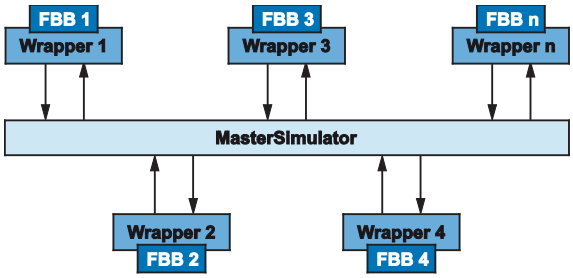

Figure 2. Basic scheme of FDMU architecture [10].

Filippo et al. [10] had developed a driving simulator based on the FMDU architecture. This driving simulator consists of two hardware components and two software components. The hardware components are a motion platform, which is an off-the-shelf Steward platform, and an input device, which is an off-the-shelf Universal Serial Bus "USB" steering wheel and pedals. The software components are the master simulator simulation core and a simple vehicle model implemented with the help of OpenModelica, which is an open-source modeling and simulation environment [10].

The developed approach: "A Modular Architecture for a driving simulator based on the FDMU Approach" focusses on the interfacing of the different components of the driving simulator with the help of an FMDU modular structure. The problem with this approach is that in order to add or exchange any component, a wrapper application has to be reprogrammed or adjusted for the new component. The approach does not describe how to add, remove or exchange any of the four pre-programmed components. Indeed, the approach is promising for simulation core components, which interface the driving simulator components with each other. But it could not be used in a reconfigurable driving simulator without some enhancements, e.g., the master simulation has to be dynamically adjustable depending on the connected modules without being pre-programmed by the user.

C. *Existing Mid-Level Driving Simulators*

Mid-level driving simulators have a greater fidelity than the low-level driving simulators, as well as high usability.

Typically, they have multi-displays, which provide a wide horizontal field of view, a real vehicle dashboard as an HMI, and they are sometimes equipped with a simple motion platform [9].

The following section describes one previous approach towards developing reconfigurable mid-level driving simulator.

The University of Central Florida Driving Simulator: The University of Central Florida (UCF) driving simulator is operated in the Centre of Advanced Transportation Systems Simulations (CATSS). It has evolved since the late 1990's into a mid-level driving simulator with the aim of conducting research in transportation, human factors and real-time simulation. The UCF driving simulator is equipped with a hexapod motion platform with 6 DoF. It has a passenger vehicle cabin as an input device. The vehicle cabin is mounted over the motion platform. The UCF has a visualization system that consists of 5 displays: one for the front view, two for side views and two for the left and middle rear mirrors. The simulator is also equipped with an audio system, force feedback steering wheel and the main operator console [11]. The simulator was designed with an exchangeable vehicle cabin. The user can choose from a commercial truck cabin and a passenger vehicle cabin according to the test requirements. The vehicle model could also be changed according to the used vehicle cabin [11].

The UCF driving simulator has exchangeable driving cabins and exchangeable vehicle models. It could be configured according to the customer requirements by choosing from the passenger car cabin with its respective vehicle model or the commercial truck cabin with its respective vehicle model. The UCF driving simulator is not a reconfigurable driving simulator because only the driving cabin and vehicle model are exchangeable. Moreover, the driving simulator user cannot exchange the entire components or add a new component to the system without the help of the manufacturer.

D. Existing High-Level Driving Simulators

High-Level driving simulators have great fidelity, high usability and they are high-cost driving simulators. Typically, they almost have a 360 degrees horizontal field of view and a complete real vehicle as an HMI, which is mounted on a high-end motion platform with at least 6 degrees of freedom [9].

The following section describes one previous approach towards developing reconfigurable high-level driving simulator.

Daimler Full-Scale Driving Simulator: Daimler AG inaugurated the Daimler full-scale driving simulator in October 2010 in Sindelfingen, Germany. The Daimler full-scale driving simulator is used mainly in developing new ADAS and the evaluation of different vehicle dynamics concepts. It is equipped with a 7 DoF motion platform that consists of the following two parts: the lateral 12 m long rail system, which provides linear motion in Y-direction and a hexapod which provides 6 DoF. The dome of Daimler full-scale driving simulator has a diameter of 7.5 m, which can be moved by a rail system for 12 m (in X or Y directions) and by the hexapod as follows: +1.4 to -1.3 m in X-direction, ±1.3 m in Y-direction, and ±1 m in Z-direction, ±20 degrees roll-rotation, -19 degrees to +24 degrees pitch-rotation and ±38 degrees yaw-rotation.

The Daimler full-scale driving simulator has a cylindrical visualization system powered by 8 projectors and gives 360 degrees horizontal field of view and three rear mirrors displays. It has several exchangeable driving cabins, e.g., S-Class, A-Class, Actros-Truck, etc. It is operated by a Daimler in-house developed software. The used software can also operate Daimler internal fixed-base driving simulator variants [12].

The Daimler full-scale driving simulator has exchangeable driving cabins and a parameterized vehicle model. It could be configured according to the test experiment requirements by choosing from different driving cabins and their respective vehicle model parameter set. The Daimler full-scale driving simulator is not a reconfigurable driving simulator because the driving simulator components are only compatible with Daimler internal components. The driving simulator user cannot exchange the entire components or add a new component to the system without the help of the manufacturer.

E. The National Advanced Multi-Level Driving Simulators

The multi-level driving simulators are different variants of a driving simulator as they have different levels of fidelity, usability and cost. But they are developed based on the same structure using the same software, hardware, and resources components. An example of the multi-level driving simulator is the NADS driving simulator, which is described in this section.

The National Advanced Driving Simulator (NADS) is a driving simulator centre located at the University of Iowa. The NADS centre has three driving simulators: the high-level driving simulator “NADS-1”, the mid-level driving simulator “NADS-2”, and the low-level driving simulator “NADS miniSim”. The NADS driving simulators are based on the same system architecture, software, and resources [13].

The NADS-1 and NADS miniSim driving simulators are modular driving simulators, which have been developed based on the same software components. They could be configured for different applications according to the customer specifications. The NADS minSim is a low-level configurable driving simulator. It is a promising approach towards developing a reconfigurable driving simulator. However, it is not a reconfigurable driving simulator, because as well-developed as it is, the user cannot exchange the entire components or add a new component to the system without the help of the manufacturer.

The analysis of the existing methods and approaches towards a reconfigurable driving simulator has shown that there is no method, approach or developed driving simulator to date which describes any systematics or approaches for the development of a reconfigurable driving simulator and none of them allows the operator of the driving simulator to reconfigure the system without in-depth expertise in the system structure.

IV. The Solution Approach

The main aim of this work is to simplify a driving simulator structure during the development. This simple structure allows the operator to create different task-specific variants by selecting the desired solution elements of the driving simulator.

The development of reconfigurable mechatronic systems, which consist almost of standardized modular components, can follow the "Building Blocks Concept". The benefits of using the building blocks concept are speeding up the learning curve of the system structure based on the many years of experiences in the development of their entire components [14].

The typical virtual prototyping cycle consists of three phases: modelling, simulation and analysis. The modelling process is the developing of simplified formal models of the system under development. The system models represent the system properties. The simulation process represents the calculations of the system models with the help of numerical algorithms in order to simulate the system behaviour. The analysis process represents the interpretation of the simulation results that are usually done by extracting, preparing and visualizing the relevant information [5] [15]. The usage of driving simulators allows ADAS developers to analyse the system under test functionality, the system behaviour in different simulation scenarios as well as the investigation of the interaction between the system, driver, and environment.

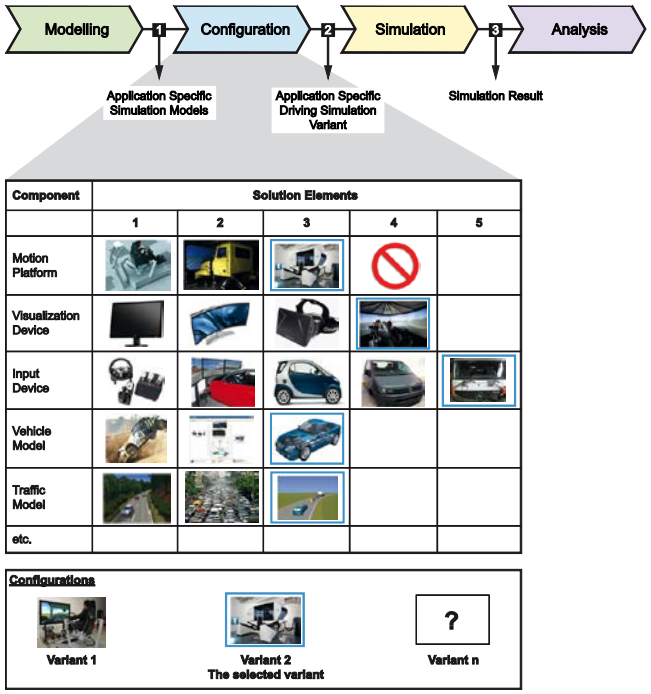

Figure 3. The solution approach of the reconfigurable driving simulator, according to the building blocks concept.

In order to reconfigure a driving simulator, there is a need to add a phase between the modelling and simulation phases. The new phase is the configuration phase shown in Fig. 3. In the configuration phase, the driving simulator operator can select the desired solution elements to create a task-specific variant of the driving simulator.

The models that have been developed during the modelling phase will be available for the selection in addition to other existing components. The operator selects a solution element for each component. These selected solution elements, acting as building blocks, build together a driving simulator variant. Fig. 3 shows a simplified example of the configuration process; the selected solution elements and the created variant are marked with a blue frame. As soon as a variant has been created, the driving simulator will be ready for the simulation and the analysis phases.

V. The Design Framework

This section is the core of the present work. It describes a design framework for developing a reconfigurable driving simulator. This design framework supports driving simulator developers and operators to develop and operate a reconfigurable driving simulator. The design framework consists mainly of the procedure model and the configuration tool. They are specifically described as follows:

- **The procedure model**, which defines the required phases in a hierarchy, in order to develop a reconfigurable driving simulator. Each phase contains entire tasks; these tasks have to be carried out in order to achieve the phase objectives. The procedure model organizes the required tasks in each phase and describes which method or algorithm should be used to fulfill each task. The used methods and algorithms contain existing approaches, as well as new approaches, which were developed during this work. Moreover, the procedure model defines the result of each phase. This is needed as an input for the following phases.
- **The configuration tool**, which supports the driving simulator operators in creating a driving simulator variant or in reconfiguring an existing variant. The configuration tool organizes the existing driving simulator software and hardware components and their corresponding solution elements in a solution elements database. As soon as the solution elements database is filled, the software guides the driving simulator operator in order to create the desired driving simulator variant. The variant creation will be done by selecting a combination of solution elements, which are available in the database. Moreover, the configuration tool can deal with guidelines for testing and/or for training approaches. They can be added to the tool, and the configuration tool can check whether the created variant guideline conforms or not.

Fig. 4 describes a design framework for developing a Rreconfigurable driving simulator. This design framework supports driving simulator developers and operators to develop and operate a reconfigurable driving simulator.

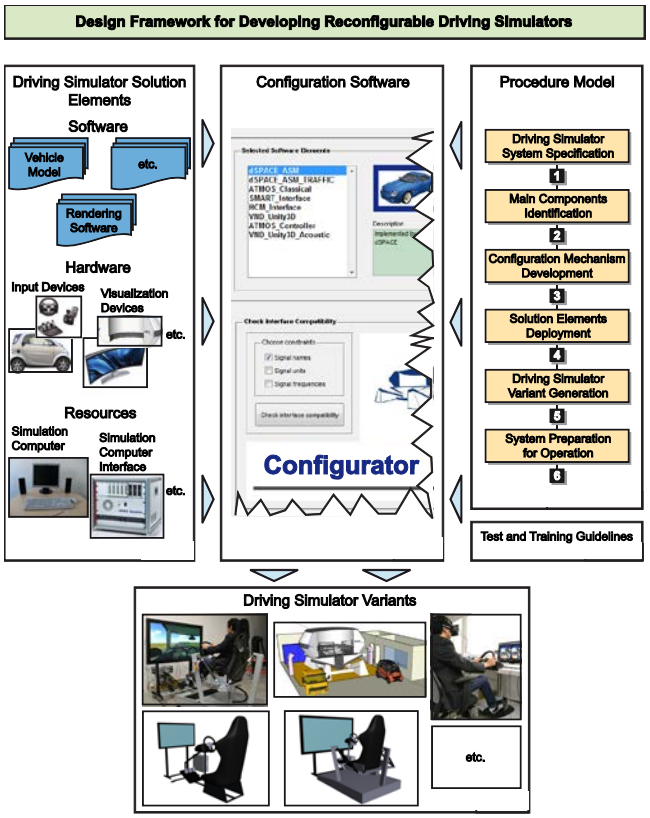

Figure 4. A design framework for developing a reconfigurable driving simulator structure and components.

Procedure Model Overview: the procedure model is the most essential part of the design framework; it describes the theoretical fundamentals of the design framework. The procedure model supports driving simulator developers in the development of a reconfigurable driving simulator. The procedure model is kept general and could be used for different driving simulator areas of use, as well as other mechatronic systems. It consists of six consequent phases divided into two stages. Fig. 5 shows the procedure model in the form of a phases/milestones diagram that shows each phase. It also shows the tasks that have to be carried out, as well as the results from each phase.

The six phases of the procedure model are generally divided into two stages: The system development stage and the variants creation stage. Each stage consists of three phases. The first three development phases have to be performed once by the driving simulator developer. As soon as the developer finishes the development phases, the driving simulator operator should carry out the variant creation phases each time he/she creates a driving simulator variant.

In the following sections, a detailed description of all needed tasks and operations during each phase, as well as the results of each phase, will be presented.

A. *Phase 1 – Driving Simulator System Specification*

The objective of the first phase is to specify a reconfigurable driving simulator, which is a complex multidisciplinary mechatronics system. Therefore, there is a need to specify the system under a multidisciplinary development with the help of a specification technique.

The CONSENS – "*Conceptual Design Specification Technique for the Engineering of Complex Systems*" will be used during this work. CONSENS is developed in order to specify complex mechatronic systems. The specifications are multidisciplinary and they simplify the complexity of the developed mechatronic system by describing it using a coherent system of partial models [16].

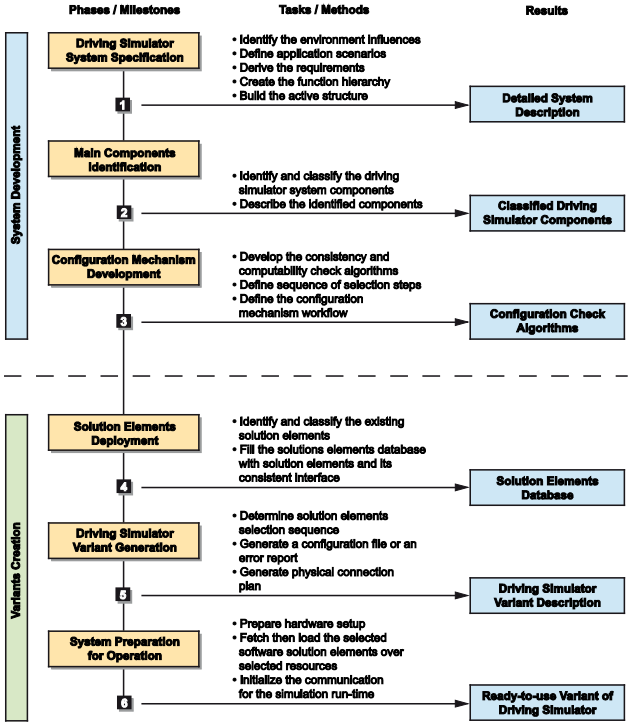

Figure 5. Procedure model for developing a reconfigurable driving simulator.

CONSENS Work Flow for a Reconfigurable Driving Simulator: the specification technique "CONSENS" divides the principle solution specification into coherent partial models. The CONSENS partial models are: requirements, environment, application scenarios, functions, active structure, shape, and behaviour. Each partial model specifies a precise aspect of the system under development [16].

The partial models' weights of importance are not equal within the development of reconfigurable driving simulators. During this work, the focus will be on five of seven CONSENS partial models. The relevant partial models are environment, application scenarios, requirements, functions, and active structure. The shape and behaviour partial models will be neglected within the scope of this work because they are not relevant to design a driving simulator. The both neglected partial models are important to design a new product.

The CONSENS work flow is divided into three steps: firstly, the environment, the application scenarios and the requirements have to be specified simultaneously. Secondly, based on the result of the first step, the function hierarchy

has to be derived. The third step is to build up the active structure based on the result of the previous steps. Fig. 6 shows the CONSENS work flow towards specifying a reconfigurable driving simulator.

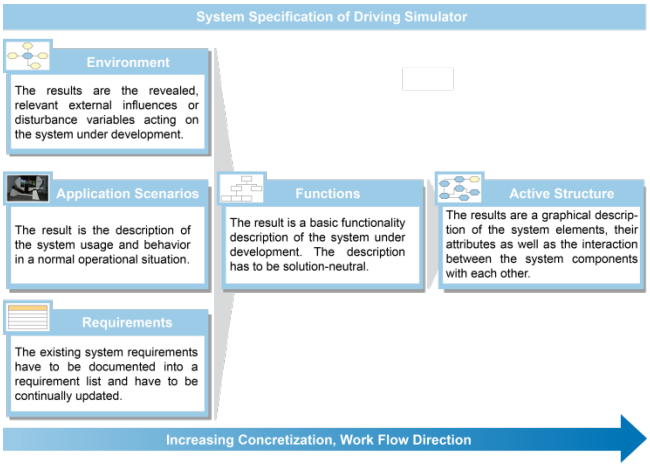

Figure 6. CONSENS work flow for reconfigurable driving simulator according to Gausemeier [17].

The specification of the system is typically carried out in the context of expert workshops with the help of a workshop cards set. The workshops' participants are usually experts in several disciplines such as mechanical engineering, software engineering, control engineering, and electrical engineering. The definition of each partial model is presented in the next sections.

1) Environment:

The environment partial model defines the external influences, which affect the system under development. The driving simulator has to be considered as a black box which means that the investigation is not of the system itself, but of the relevant external influences. These external influences are environment elements or disturbance variables [16].

Fig. 7 shows an environment model example of a driving simulator variant.

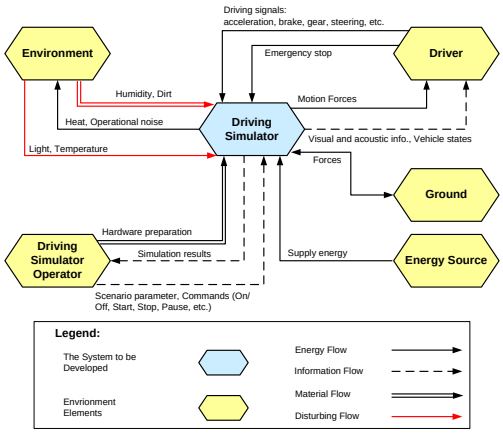

Figure 7. Environment model of a driving simulator variant.

2) Application Scenarios:

The application scenarios partial model is an essential partial model of the system specification. In this specification step, some operational application scenarios are defined. Each application scenario describes the system under development in terms of way of use, operation modes, system manner and main components. By using CONSENS, each application scenario will be described in a profile page, which contains the scenario title, scenario numbering, the scenario description and a simple sketch of the needed hardware components [16].

3) Requirements:

This partial model collects and organizes the system requirements of the system under development which need to be covered and implemented during the development process. The requirement list contains functional and non-functional requirements [16]. Additionally, the organized requirements distinguish between demands and wishes (D/W) [18].

4) Functions:

The functions partial model is built based on the previous partial models: environment, application scenarios and requirements. It describes the system and its entire components' functionality in a top-down hierarchy [16]. Each block describes a sub-function of the system. Function catalogues, according to Birkhoffer [19] or Langlotz [20], support the creation of the functional hierarchy.

Due to the variation of the main function, structure, and required components of the stated application scenarios, the functions specification also varies in its complexity and number of its entire sub functions. Therefore, there is a need to merge the identified functions of the stated application scenarios. Fig. 8 shows a function model example of a driving simulator variant.

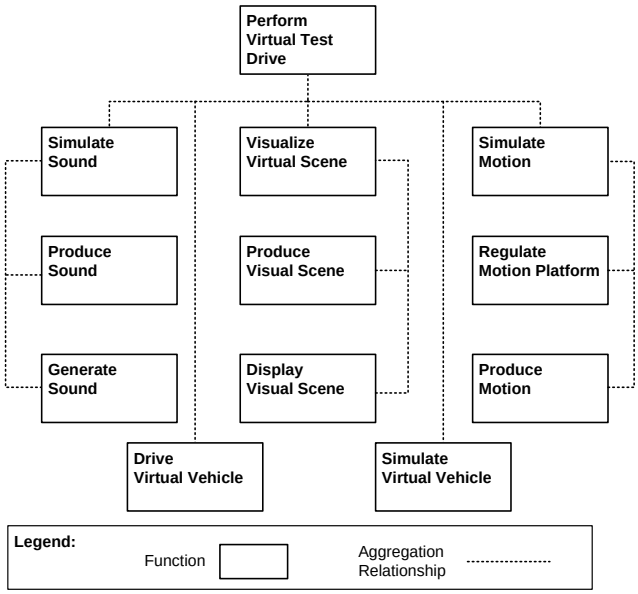

Figure 8. Function model of a driving simulator variant.

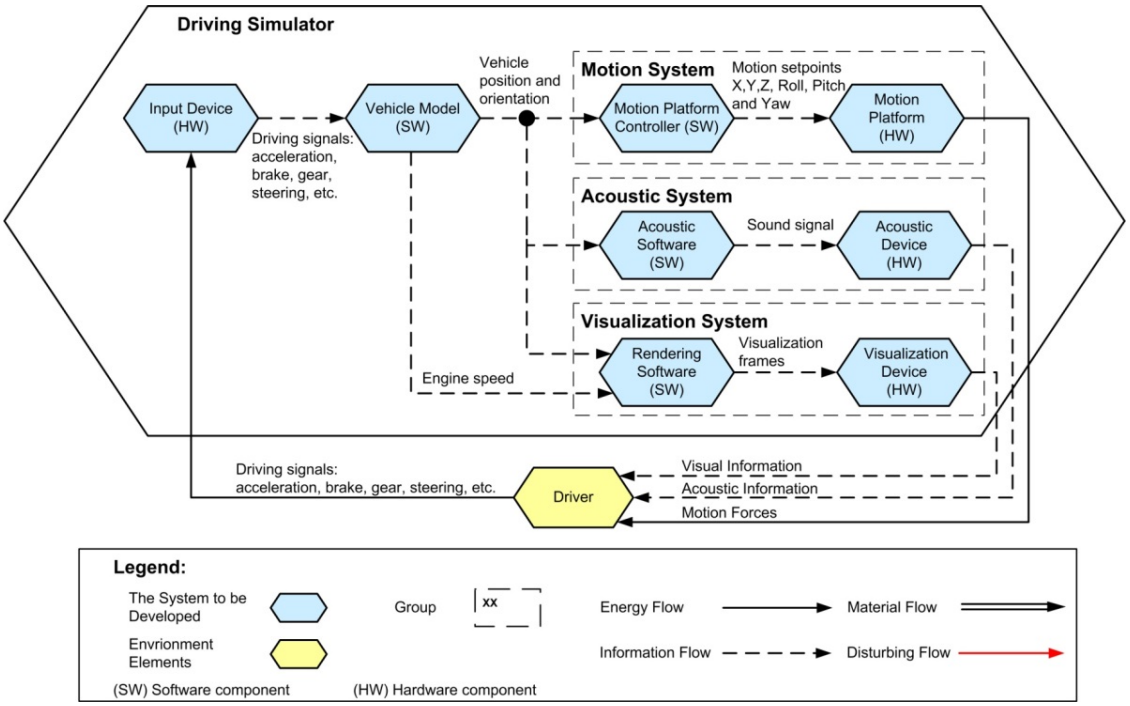

Figure 9. Active structure model of a driving simulator variant.

5) Active Structure

The active structure partial model is built based on the previous partial models results, specifically the functions partial model. The active structure describes the entire system in more details in the form of system component active principles. It describes the system components, their attributes, the entire interfaces and how the components interact with each other. Depending on the modeling level of details, each system element could be described abstractly as an active principle or a software pattern. Additionally, material, energy, and information flows, as well as logical relationships, describe the interactions between the system elements [16]. Fig. 9 shows an active structure model example of a driving simulator variant.

The first phase results, which are the driving simulator system specification describes in the form of five partial models, are: environment, application scenarios, requirements, functions, and active structure. This result is the input for the second phase.

B. *Phase 2 – System Components Identification*

The second phase objectives are the identification, classification and definition of the driving simulator components based on the results of the first phase. Towards the identification of the driving simulator system components, a distinction between optional components, key components and solution elements must be defined.

As the driving simulator structure could also be changed during the reconfiguration process, the key components have to be identified. The key components are the obligatory system components that always have to exist in the simulator structure. For example, each driving simulator has to have a visualization rendering software but a motion platform is an optional component and not a key component, because a driving simulator does not need to have a motion platform.

1) Identification of Driving Simulator Components

Based on the active structure partial model, the system components, as well as the system key components can be identified with the help of the following three operations:

1. Identify all components:

The reconfigurable driving simulator components are the union of the different variants components as follows:

$$Sim\_Cp = Var\_1\_cp \cup Var\_2\_cp \cup \dots Var\_n\_cp \quad (1)$$

Where: Sim_Cp is the reconfigurable driving simulator components, Var_1_cp is variant 1 components, Var_2_cp is variant 2 components, and n is the number of modelled variants.

For example, if variant 1 components are {A,B,C} and variant 2 components are {A,B,D,E}, and the common system components will be {A,B}.

2. Identify common components:

The common components of the reconfigurable driving simulator are defined based on the intersection between the different variants components as follows:

$$Sim\_Cp = Var\_1\_cp \cap Var\_2\_cp \cap \dots Var\_n\_cp \quad (2)$$

3. Identify key components:

In order to identify the system's key components, the selection will be done based on the common components set. Each component has to be investigated individually in a

logical way by eliminating the component from the set. If the driving simulator can be operated without this component, this means that it is an optional component. But if the driving simulator cannot be operated, then this means that it is a key component.

2) Classification of the Identified Components

In addition to the modelled software and hardware components, the reconfigurable driving simulator resources have to be taken into consideration. Each software or model needs a computing unit (e.g., a computer) to be executed on. Moreover, each hardware component needs a physical interface to communicate with its corresponding software interface.

In order to organize the identified components easily, these have to be classified under the following three categories: hardware, software, and resources. The software category contains two subcategories: the applications/models and the hardware interfaces. The resources category contains two subcategories: the computing units and the signal processing interfaces. Fig. 10 shows an example of the classification of the identified components.

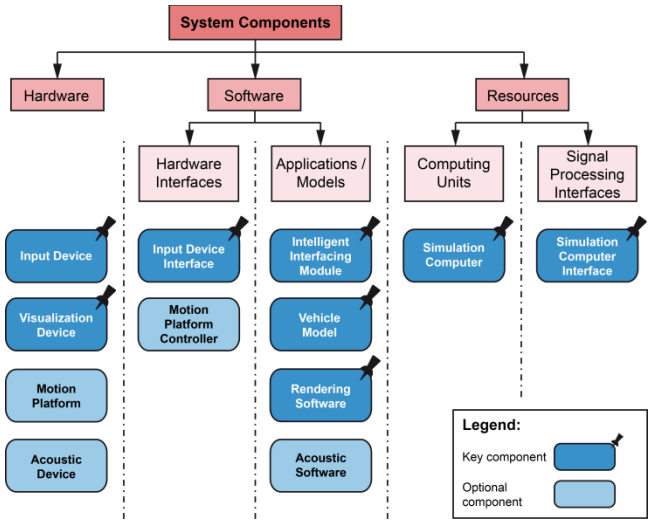

Figure 10. Classification of the identified components example.

3) Description of the Identified Components

In order to understand the function of each component, each component has to be defined from a solution-neutral point of view. The following are the description of two identified components as an example:

Input Device: This is a hardware MMI (Man-Machine Interface) between the driver and the driving simulator. It provides driving signals, e.g., acceleration pedal position, brake pedal position, etc. The input device provides the driving simulator with these signals in energy flow, which represents a physical signal.

Input Device Interface: This is a software component, which converts the energy flows of the input device to its computer representative information flows (digital signals).

C. *Phase 3 – Configuration Mechanism Development*

This is the third and last phase of the development stage. The objective of the third phase is to develop a configuration mechanism, which ensures that the selected solution elements could operate together. This check is done after selecting the preferred structure and the desired solution elements. The configuration mechanism has to ensure the consistency and the compatibility of the selected structure and its entire solution elements. After the configuration mechanism ensures the selected solution element consistency and compatibility of the solution elements, it generates a configuration file. The configuration file contains a list of the selected solution elements, the interfaces' topology and the selected resources.

The configuration mechanism checks the selected solution elements. However, the solution elements will be deployed in the next phase, but it is the preferred order of the procedure. Developing the configuration mechanism before deploying the solution elements allows the mechanism to also deal with unknown solution elements, which can be added in the future.

There are two types of relationships between the selected solution elements and each other. These relationships have to be checked and confirmed by the configuration mechanism. The first relationship is the logic consistency between the selected solution elements with each other. The second relationship is the compatibility between the interfaces of the selected solution elements.

1) Consistency Check Algorithm

The consistency relationship can be determined by two levels. The first level is the **logic dependency** between components, which determines if there is a logic correlation between two components or not. The second level is the **logic consistency** between two solution elements.

a) Logic dependency between two components:

It is a logic relationship between two components, which describes if they depend on each other logically or not. For example, the motion platform and the input device are a dependent pair of components. They depend on each other, i.e., an input device has to be mounted on a motion platform. Therefore, the motion platform dimensions and payload have to match with the selected input device.

Dependency matrix: the dependency matrix is a two-dimensional matrix that describes the logic dependency between the identified components. The components are stated in both the first row and the first column; the matrix is mirrored along its diagonal. Therefore, only the lower half of the matrix has to be filled with 0 or 1 by the **driving simulator developer**.

0: means the components pair is logically independent of each other, thus the inherited solution elements belonging to these components will also be logically independent of each other.

1: means the components pair is logically dependent on each other, thus the inherited solution elements belonging to these components will also be logically dependent on each other. Fig. 11 shows the dependency matrix based on the identified components.

Dependency Matrix 0 = Independent pair 1 = Dependent pair		Hardware Components				Software Components					Resources	
		Input Device	Visualization Device	Motion Platform	Acoustic Device	Vehicle Model	Rendering Software	Acoustic Software	Input Device Interface	Motion Platform Controller	Simulation Computer	Simulation Computer Interface
		A	B	C	D	E	F	G	H	I	J	K
Hardware	A. Input Device											
	B. Visualization Device	0										
	C. Motion Platform	1	1									
	D. Acoustic Device	0	0	0								
Software	E. Vehicle Model	0	0	0	0							
	F. Rendering Software	0	1	0	0	0						
	G. Acoustic Software	0	0	0	1	0	0					
	H. Input Device Interface	1	0	0	0	0	0	0				
	I. Motion Platform Controller	0	0	1	0	0	0	0	0			
Resources	J. Simulation Computer	0	0	0	0	1	1	1	1	1		
	K. Simulation Computer Interface	0	0	0	0	0	1	1	1	1	1	

Figure 11. Dependency matrix of the identified components.

b) Logic consistency between two solution elements

It is a logic relationship between two solution elements, which describes if they are logically consistent with each other or not. The first relationship depends on whether the solution elements' parent components are independent. This means that the two solution elements inherited the independence and there is no need to check their consistency. Otherwise, if the solution elements' parent components are dependent, this means that the two solution elements inherited the dependency and have to be checked if they are consistent or not.

Consistency matrix: the Consistency matrix is a two-dimensional matrix that describes the logic consistency between the available solution elements. The solution elements are stated in both the first row and the first column. The matrix is mirrored along its diagonal. Therefore, only the lower half of the matrix has to be filled with 0, 1 or 2 by the reconfigurable driving simulator operator.

0: means the solution elements pair is logically inconsistent with each other. This means that they could not be selected together in a driving simulator variant.

1: means the parent components pair was originally logically independent of each other, thus the inherited solution elements under those components will also be logically independent of each other. This means that the solution elements do not have to be checked for consistency.

2: means the solution elements pair is logically consistent with each other. This means that they could be selected together in a driving simulator variant.

Fig. 12 shows a part of a consistency matrix based on the result with the assumption that each component has two solution elements. Dealing with the solution elements in this section will be illustrated in an abstract form, e.g., the solution elements will be called (A1, A2, B1, etc.); where A and B are components and A1 is the first solution element for the component A, etc.

The consistency matrix is filled out based on the dependency matrix. If a pair of components is independent (0 value in the dependency matrix), e.g., A and B, their solution elements will inherit this relation (1 value in the consistency matrix). Otherwise, if a pair of components is dependent (1 value in the dependency matrix), e.g., A and C, their solution elements will inherit the dependency relationship and they are either consistent or not (respectively 2 or 0 value in the consistency matrix).

Consistency matrix 0 = Logically Inconsistent 1 = Logically Neutral 2 = Logically Consistent			Hardware Components									
			A. Input Device		B. Visualization Device		C. Motion Platform		D. Acoustic Device		E. Vehicle Model	
			A1	A2	B1	B2	C1	C2	D1	D2	E1	E2
Hardware	A. Input Device	A1										
		A2										
	B. Visualization Device	B1	1	1								
		B2	1	1								
	C. Motion Platform	C1	2	0	2	0						
		C2	0	2	0	2						
	D. Acoustic Device	D1	1	1	1	1	1	1				
		D2	1	1	1	1	1	1				

Figure 12. The consistency matrix – example of some solution elements.

Consistency check sequence: considering the consistency relationship, which is determined by two-level matrices, the consistency check will also be performed by two level checks.

Fig. 13 shows a flowchart of the consistency check. For example, the consistency between solution elements A1 and B2 has to be checked. The first check will be based on the dependency matrix between the two parent components A and B. The second level will be based on the consistency matrix between the solution elements A1 and B2.

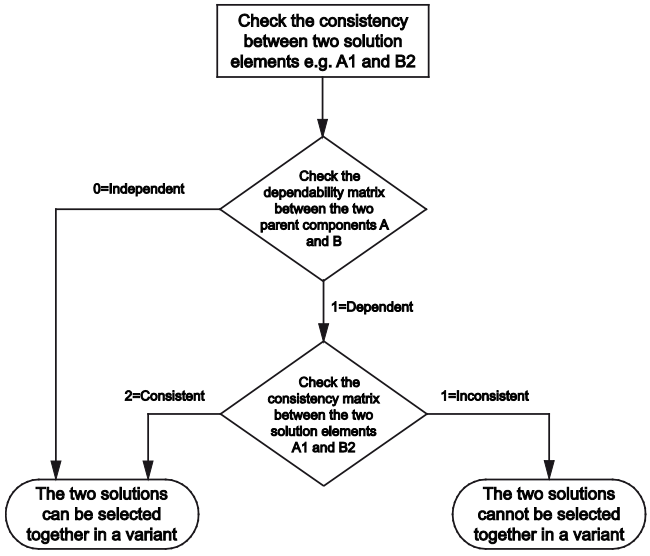

Figure 13. Consistency check flowchart.

2) Compatibility Check Algorithm

One of the main approaches to building a reconfigurable driving simulator is the ability of adding, removing or exchanging one or more solution elements. In order to build such a reconfigurable system, the applications/models interfaces have to be carried out automatically. Therefore, there is a need for an algorithm to check if all selected solution elements are compatible with each other or not. The compatibility here means whether the interfaces of the selected solution elements match together or not. Hence, each software component has its programming language and naming system of the input and output signals. Additionally, there is a need to extend the reconfigurable system

continuously by adding new unknown solution elements. Therefore, a generic solution elements' interface concept has been developed to manage and check different existing solution elements, as well as unknown solution elements that could be added in the future.

Generic solution elements' interface concept: in order to interface the entire solution elements, each solution element has to be considered as a black box. Mainly, only the input and output interfaces have to be considered. To keep the configuration process flexible and extendable, any solution element can be added as soon as its input and output interfaces are defined. The only required task for integrating any solution element is to map its inputs and outputs to the reconfigurable driving simulator's unique signal names there, this task is called signal multiplexing.

Fig. 14 shows an example of the signal multiplexing. A vehicle model has to be integrated as a solution element. The model will be considered as a black box, but all its input and output signals have to be mapped to the reconfigurable driving simulator's unique signal names. The output signal called "Otutput_ID563[m/s]" is the vehicle under test velocity in m/s, but this signal's unique name and unit predefined in the reconfigurable driving simulator has the name "Chassis_Velocity" and its unit is km/h. Also in this case, a simple unit conversion will be used.

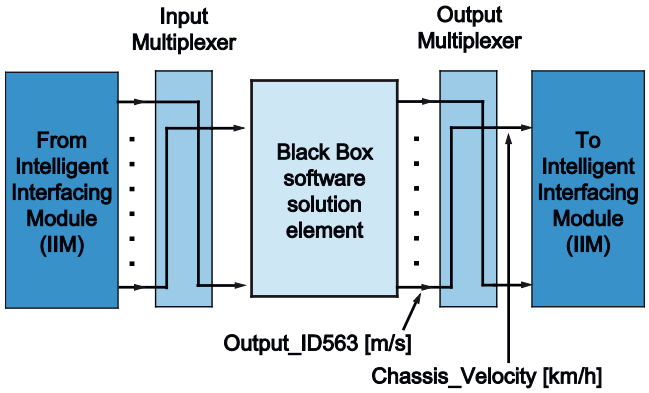

Figure 14. Generic solution elements interface concept.

In order to integrate this vehicle model, the user has to connect all the input and output signals with different names and units to the unique names and the units of the parent reconfigurable system. The input and output signals multiplexers should be programmed before registering the solution elements in the solution element database.

Compatibility check steps: after selecting the preferred solution elements, the compatibility check algorithm proofs the solution elements one by one to ensure that the input signals could be satisfied from the outputs from other solution elements. The compatibility check algorithm does not only check the signals' name but also other signal attributes such as frequency and unit to ensure the compatibility.

Fig. 15 shows a flowchart of the compatibility check. The compatibility check algorithm checks the compatibility of each signal through the following steps:

a) The algorithm checks each input signal of each selected solution element.

b) Each input signal has a unique name and must be delivered as an output from another selected solution element output. Therefore, the algorithm searches by the signal unique name in all output signals of the other selected solution element.

c) If the search engine finds the input signal as an output signal of the other selected solution elements that means this input signal could be satisfied.

d) Additionally, the search algorithm can check the compatibility of the signal unit and frequency. The output signal must have a greater frequency than the input signal or a sample rate converter will be required.

e) Then, the algorithm confirms the compatibility of this signal or stores an error in the error log.

These five steps have to be repeated for each input signal of each selected solution element.

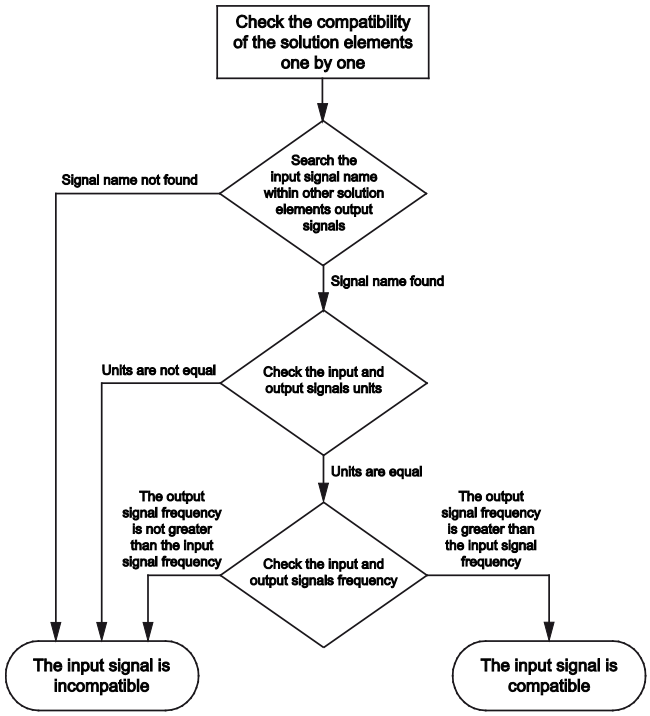

Figure 15. Compatibility check flowchart.

D. *Phase 4 – Solution Elements Deployment*

The first stage of the development procedure "System Development" was described, as well as its entire three phases. The first stage has to be carried out only once by the driving simulator developer. The result of the first stage is a reconfigurable driving simulator outline, which should be extended in the **variants creation stage** by the driving simulator operator. The first stage describes the system's entire components from a solution-neutral point of view. The second stage is the concretisation stage, which deals with solution elements instead of the solution-neutral components.

The second stage "variants creation" consists of three phases, starting with phase 4 "solution elements deployment". The main objective of this phase is to build a solution elements database, which contains the existing solution elements, their interfaces and attributes. This phase is an iterative process that has to be carried out each time to add or modify a solution element to the solution elements database.

The solution elements deployment is carried out in two steps. The first step is the identification and classification of the solution elements and the second step is the filling out of the solution elements database with the required attributes of each solution element.

1) Identify and Classify Solution Elements

The solution elements' identification and classification will be carried out based on the results of the first and second phases. The preferred solution elements will be carried out based on the morphological box concept according to Zwicky [21].

2) Filling the Solution Elements Database

In order to make the configuration tool deal with the component and solution elements, there is a need to register the identified components and solution elements in a database. This database stores and organizes the components and solution elements. It also has to be readable by the driving simulator operator and accessible by the configuration tool.

The main database operations are based on CRDU classes [22]: create, read, update, and delete. These operations must be covered by the database.

Create: This operation could be performed for both components and solution elements. The database is always extendable by adding a new component or by adding a new solution element for an existing component. This operation will be described in detail in this section.

Read: This operation can be executed for both components and solution elements. The database internal entries are accessible for the driving simulator operator, as well as for any software that would be used during the configuration process. All stored component and solution elements as well as their attributes can be accessed.

Update: This operation can be executed for both components and solution elements. Each stored component or solution element can be changed and restored.

Delete: This operation can be executed for both components and solution elements. Each stored component or solution element can be deleted from the database.

In this section, the create operation is described in detail in order to fill the solution elements database. The filling process is done in two steps: create component then create solution element.

Create a component entry: In order to create a component, the following attributes must be registered and stored in the database: **Component name** "which is the unique name of each component", **Component type "**a key component or an optional component", **Component classification** "hardware, software or resources", **Component description**, C**omponent symbol**, **Component logic dependency row** "which is a row contains the logic dependency between the components and the previously added components", and **Component guideline entry** " that is an optional attribute, which defines a preferred parameter value and condition regarding the component". For example, a guideline defines that the visualization device must have a minimum horizontal viewing angle of 100 degrees. This attribute can be added to the component in the form of the condition greater than (>) and parameter value (100 degrees).

Create a solution element entry: In order to create a solution element, the following attributes must be registered and stored in the database:

Solution Element Name: This attribute is the unique name for each solution element.

Solution Element Path: This attribute is the storage path on the file storage system. This is applicable only for an application/model.

Solution Element – Parent Component: This attribute is the name of the corresponding parent components. Therefore, it represents the relationship between this solution element and a component.

Solution Element Description: This attribute is a brief description of the solution element.

Solution Element Symbol: This attribute contains a symbol (logo) associated with the solution element.

Solution Element Author: This attribute is the solution element developer name, if known.

Solution Element Company: This attribute is the solution element producer company name if known.

Solution Element Release Date: This attribute is the date of when the solution element was released.

Solution Element Interface: This attribute is a table containing all the input and output signals of the solution element. Each signal has the following attributes:

Signal Name: It contains the names of the input and output signals of the corresponding solution element.

Input/Output: It indicates the direction of the signal, i.e., whether it is an input or an output signal.

From: It contains the component name from which this signal is to be fulfilled. This is applicable only for input signals.

Unit: It contains the measuring unit of the corresponding signal.

Frequency: It contains the sampling frequency of the corresponding signal.

Resolution: It contains the resolution of the corresponding signal.

Protocol: It contains the transmission protocol of the corresponding signal, e.g., Controller Area Network "CAN" or Transmission Control Protocol / Internet Protocol (TCP/IP) TCP/IP.

Physical Port: It contains the physical port used to transmit the corresponding signal.

Mandatory/Optional: It indicates whether the signal is mandatory or optional.

Description: It contains a brief description of the corresponding signal.

Solution Element Consistency Row: This attribute is a row, which contains the logic consistency between the

solution element and the previous added solution elements. This row is part of the solution elements consistency matrix.

Solution Element Guideline Entry: If the parent component has a guideline entry, the solution element inherits this entry and should define a parameter value for the entry to check the solution element confirmation with the guideline.

After registering all identified components and all preferred solution elements, which result from the metrological box in the database, the solution elements database is filled and ready to be used in the variant generation phase.

E. Phase 5 – Driving Simulator Variant Generation

The main objective of this phase is to define the configuration selection sequence, as well as define the configuration file structure, error reports structure and the physical connection plan.

1) Configuration Selection Sequence

In order to make a reasonable selection sequence for the solution elements, the identified components and their relationships have to be investigated. The selection sequence can be changed based on the area of use. During this phase, an example of the use case study shows how it can be determined.

The driving simulator components have been previously classified as three main classes: Hardware, software, and resources. A driving simulator structure is respectively based on hardware components, software, and finally, the used resources.

In order to make the selection sequence reasonable, it is not sufficient to make the selection sequence based on the classification, because of the tight correlation between some hardware and software components. Therefore, the identified components will be divided into groups of software and/or hardware based on the groups identified during the active structure specification step.

2) Configuration Files and Error Reports Structure

After the compilation of the solution elements' selection process, the configuration mechanism checks the selected components in terms of consistency and compatibility.

Based on the configuration mechanism check results, if the selected solution elements are consistent and compatible with each other, the configuration tool confirms that the selected solution elements can build a driving simulator variant and generates a configuration file. However, if the configuration tool finds any inconsistency or incompatibility between the selected solution elements, the configuration tool generates an error report. In the next section, the structures of the configuration file as well as the error report will be described.

Configuration File Structure: the configuration file is considered to be the result of the configuration process. It is a readable text file containing all the relative data about the selected variant. It consists of four parts: configuration data, hardware, software, and resources. The configuration data is the part that describes general information about the configuration itself, e.g., configuration name, author, etc. The hardware part contains all selected hardware solution elements attributes, parent component name and detailed input/output signal descriptions. The software part contains all selected software solution elements attributes, parent component name and detailed input/output signal descriptions. The resources part contains the selected resources.

Error Report Structure: the error report is a readable text file containing warnings and errors, which are detected by the configuration mechanism. It contains five parts: configuration data, hardware, software, resources and, errors/warning. The first four parts are the same as in the configuration file. The error and warning part lists all detected inconsistent solution elements, as well as all incompatible signals.

3) Physical Connections Plan

The configuration tool generates configuration files that contain the interfaces between the selected solution elements and the software side, but the configuration file does not contain the physical connections between the selected hardware solution elements and the selected resources. A physical connection plan is very useful for the driving simulator operator in order to prepare the driving simulator for operation. It shows in a simple way how the diverse hardware solution elements should be connected with the resource interfaces. It could be considered as a simple wiring plan.

Fig. 16 shows an example of the physical connection plan regarding. This variant consists of four hardware solution elements, which have to be connected to the simulation computer interfaces. With the help of the information stored in the solution elements database, the physical plan for the components can be generated. In this case, there were 4 connections, each hardware solution element is connected through one connection.

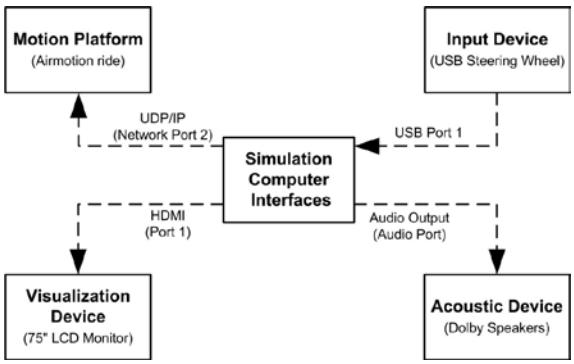

Figure 16. Example of a physical connection plan.

F. Phase 6 – System Preparation for Operation

The result of the fifth phase is the configuration file and a physical connection plan. The configuration file contains the selected solution elements, interface topology, and selected resources. Additionally, the physical connection plan contains the physical interfaces between the selected hardware solution elements.

There are two preparation steps required in order to build up the selected driving simulator variant and to prepare it for

the simulation. The first step is the preparation of the hardware connections and the second step is the software preparation.

1) Hardware Setup Preparation

Assuming that the selection process finished successfully and the configuration tool generated the physical connection plan, and then the driving simulator operator has to plug the different hardware solution elements together. The physical connection plan makes this step easy and understandable.

For the example, in Fig. 16, the driving simulator operator has to plug in 4 cables: a USB cable between the steering wheel and the simulation computer, an High-Definition Multimedia Interface "HDMI" cable between the 75" Liquid Crystal Display "LCD" monitor and the simulation computer, a network cable between the motion platform and the simulation computer, and an audio cable between the dolby speakers and the simulation computer. The example shows that the hardware preparation step can be easily done manually.

2) Simulation Software Preparation

To prepare the selected software solution elements for the operation, which is a complicated process (unlike the hardware preparation step) there is a need to develop software to assist this step. The software is called "Assistant". The assistant software is responsible for preparing the software solution elements for the simulation by the following three steps:

Read the configuration file: The assistant software can load and phrase the configuration file. It identifies the selected applications/models and their different attributes.

Fetch the applications/models: The assistant software retrieves the storage path for each application/model. It accesses the storage file system where the applications/models are stored.

Distribute the applications/models over resources: The assistant software loads each application/model on its corresponding source selected during the selection process.

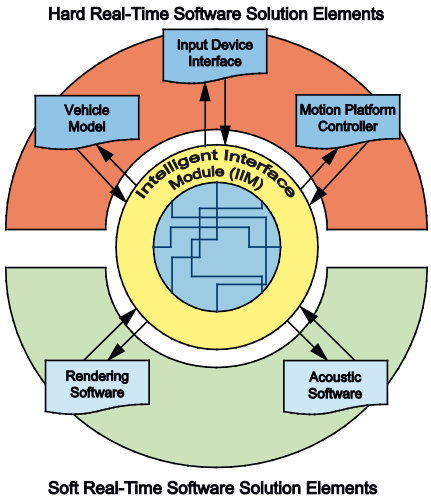

Figure 17. IIM function during simulation run-time.

The Intelligent Interfacing Module (IIM) initializes the communication between the selected software solution elements based on the interface topology, which is described in the configuration file. As soon as the user starts the simulation, the IIM ensures the communication between the simulation-related software solution elements during simulation run-time.

Fig. 17 shows the IIM function. The IIM exchanges the required input and output from and to the simulation related software solution elements during run-time. Moreover, IIM can connect the software solution elements together although a part of them runs under hard real-time conditions and the other part runs under soft real-time conditions.

The result of this phase is a ready-to-use driving simulator that consists of the selected software and hardware solution elements, as well as the selected resources.

VI. Implementation Prototype of the Configuration Tool

A prototype of the described concept has to be implemented as a part of this work. The implemented configuration tool consists of more than 150 embedded functions. This section describes the essential components of the configuration tool, the graphical user interface and the important tasks/functions covered by the tool.

The software was implemented using two software tools: Microsoft Office Excel and Matlab. The reconfigurable driving simulator database is implemented simply in MySQL. Further, the functions and algorithms are implemented with the help of Matlab M-Functions and the graphical user interface is implemented with the help of Matlab-GUI utility.

The development of the reconfigurable driving simulator database was done based on the relational database model approach. This approach is efficient and overcomes the complexity of the relationships between the entire different database tables. The implemented database mainly contains three types of tables: the components' table, the solution elements' table and the interfaces' table. These three types of tables are connected together based on a relational model of the database.

The dealing with the developed configuration tool is carried out mainly via a graphical user interface. Fig. 18 shows the start screen, which contains the main operations of the configuration tool and their correlation to the various phases of the development procedure model.

The start screen operations of the configuration tool are described as follows:

Configure New System: this operation is the essential task of the configuration tool. It is responsible for creating a new driving simulator variant by selecting solution elements for hardware, software, and resources in a predefined sequence; so that the user is prevented from dealing with complex algorithms such as consistency and compatibility check algorithms. Firstly, the consistency check algorithm runs in the background parallel to the selection steps. The configuration tool shows only the consistent solution elements that match with the previously selected solution element. Secondly, after the selection steps end, the

configuration tool executes the compatibility check algorithm to check the compatibility of the selected solution elements. After the compatibility check has finished, the configuration tool generates a configuration file if the selected solution elements are compatible with each other or it generates an error file if the selected solution elements are not compatible with each other.

Load Configuration File: this function allows the user to view and modify a previously generated configuration file. Moreover, it allows the operator to modify the previously generated configuration file by exchanging one or more of the previously selected solution elements.

View Components and Solution Elements: this function allows the user to deal with the stored components and the solution elements in the database. The user can view, modify or delete one or more component or solution element.

Add New Component: this function allows the user to add one new driving simulator component per execution. This function will guide the user through predefined schemes in order to register the different attributes of the new component.

Add New Solution Element: this function allows the user to add one new driving simulator solution element under a selected component per execution. This function will guide the user through predefined schemas in order to register the different attributes of the new solution elements.

Behind each operation in the main screen, a set of panels/schemas exists to accompany the user until he accomplishes the selected function.

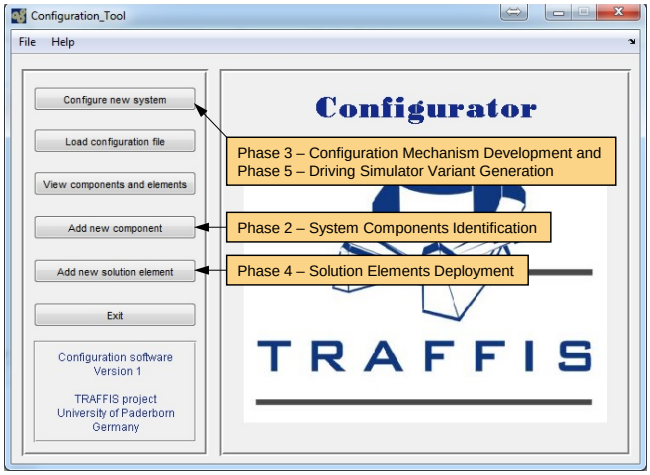

Figure 18. The graphical user interface of the configuration tool's implementation prototype – start screen.

VII. The Design Framework Validation

In order to validate the design framework, three ADAS driving simulator variants have been generated with the help of the described procedure model and the implementation prototype of the configuration tool. The three generated ADAS driving simulator variants were generated simply by selecting their desired components and their preferred solution elements.

A. *Configuration 1 – TRAFFIS-Full*

The name of the first generated variant is "TRAFFIS-Full". This variant has the most complex structure and it contains most of the ADAS reconfigurable driving simulator components. This variant is based on an application scenario. The main objective of the TRAFFIS-Full variant is testing the real Head-Lamp Control Module "HCM" control unit in HiL environment [23]. Additionally, the driving simulator motion platform and the real vehicle cabin allow the investigating of the inter-action between the driver and the HCM control unit in a Human-in-the-Loop environment. Fig. 19 shows the TRAFFIS-Full variant.

Figure 19. The TRAFFIS-Full variant.

The motion platform, which is used in this variant is the ATMOS motion platform. It consists of two dynamical parts with 5 DoF. The first dynamical part is the moving platform. It has 2 DOF and is used to simulate the lateral and longitudinal accelerations of the vehicle. It can move in the lateral plane and at the same time, it has the ability to tilt around its lateral axis with a maximum angle of 13.5 degrees and around the longitudinal axis with a maximum angle of 10 degrees. Four linear actuators are used to control the movements in both directions. The second dynamical part is the shaker system, which has 3 DOF to simulate the roll and pitch angular velocities and the vertical acceleration of the vehicle. It is driven by a three drive crank mechanism (three actuators).

B. *Configuration 2 – TRAFFIS-Portable*

The name of the second generated variant is "TRAFFIS-Portable". This driving simulator variant is a stripped-down version of the TRAFFIS-Full variant, which is based on an application scenario. The main objectives of the TRAFFIS-Portable variant are traffic safety training, as well as illustrating the bene-fits of ADAS functions. The traffic safety trainings typically take place on site at logistic agencies. Therefore, a portable driving simulator variant with a simple motion platform was needed. Fig. 20 shows the TRAFFIS- Portable variant.

Figure 20. The TRAFFIS-Portable variant.

C. Configuration 3 – TRAFFIS-Light

The name of the third generated variant is "TRAFFIS-Light". This variant has the simplest structure and contains the smallest number of ADAS reconfigurable driving simulator components. This variant is based on an application scenario. The main objective of the TRAFFIS-Light variant is testing the main HCM algorithms in the laboratory in a SiL simulation environment. The generated setup is a PC-based simulator with a simple vehicle model and a visualization system. Fig. 21 shows the TRAFFIS-Light variant.

Figure 21. The TRAFFIS-Light variant.

VIII. Conclusion and Outlook

Driving simulators have been used successfully for decades in different application fields. They vary in their structure, fidelity, complexity and cost from low-level driving simulators to high-level driving simulators. Nowadays, driving simulators are usually developed individually by suppliers and they are developed with a fixed structure to fulfil a specific task. Nevertheless, using a driving simulator in an application field, such as ADAS development, requires several variants of a driving simulator. These variants differ in their structure, in the used solution elements and in the level of detail of the entire models. Therefore, there is a need to develop a reconfigurable driving simulator, which allows its operator to easily create different variants without in-depth expertise in the system structure and without the help of the driving simulator's manufacturer.

Driving simulators are complex, interdisciplinary mechatronic systems. Therefore, the development of a reconfigurable driving simulator is a challenge. During the problem analysis, this challenge was analysed, the reconfigurable driving simulator term was de-fined and the essential requirements of the design framework were identified.

The extensive analysis of the state of the art has shown an existing method for the selection of the driving simulator and previous approaches towards developing reconfigurable driving simulators. The method named "Application Oriented Conception of Driving Simulators for the Automotive Development", developed by Negele, allows automotive engineers to formulate the requirements and specifications of a driving simulator for a specific application. Further to this, many driving simulators were investigated, but only seven of them could be identified as possible previous approaches towards developing a reconfigurable driving simulator. The seven identified driving simulators were classified into four categories: low-level, mid-level driving simulators, high-level, and multi-level driving simulators. The investigation of the existing methods and driving simulators has shown that there is no existing method or a developed driving simulator to date which covers all the design framework requirements. Therefore, a need for action was identified.

In order to solve the challenge of developing a reconfigurable driving simulator, a design framework for developing a reconfigurable driving simulator was developed to meet the defined requirements and to fulfil the need for action. The design framework consists mainly of the procedure model and the configuration tool.

The design framework has been validated with the help of a validation example. The validation example was the development of ADAS reconfigurable driving simulators. They are task-specific driving simulators, which are used for the testing and training of ADAS. During the validation, three variants of the reconfigurable driving simulator were successfully developed.

This paper described a modified procedure model comparing with [1]. Moreover, it showed a more detailed analysis of the state of the art, and it presented three validation examples of different driving simulators variants.

In summary, the developed design framework for developing a task-specific reconfigurable driving simulator is a comprehensive framework, which supports the driving simulator developers in their development of reconfigurable driving simulators. Moreover, it allows the driving simulator operators to easily create task-specific driving simulator variants.

Added value: In order to show the added value of using the design framework, two driving simulators variants: TRAFFIS-Portable and TRAFFIS-Light were developed individually. Each one of them has its fixed structure, certain software and hardware components. Furthermore, the interfaces between the different components were done manually. The development duration of the TRAFFIS-Portable variant was about four work months and of the TRAFFIS-Light was about three work months. By using the design framework the development duration of each was only two work weeks. That shows the benefits of using the design framework from the effort and cost points of view.

Outlook: The developed design framework for developing a reconfigurable driving simulator has considered the driving simulator as a mechatronic system. The procedure model and the configuration tool have been kept general, in order to be applicable for other mechatronic systems. The usage of the developed design framework for other mechatronic systems still has to be investigated. For example, in the plant engineering and construction field, most of the components are standard, e.g., conveyers, actuators, sensors, etc., as well as a customised components, e.g., controllers, robots, etc. This design framework can be easily adapted in order to configure customer-oriented plant solutions. These plant solutions are variants consisting of standard and customised components in a desired engineering design.

ACKNOWLEDGMENT

This work, as part of the project TRAFFIS (German acronym for "Test and Training Environment for Advanced Driver Assistance Systems"), which is funded by European Union "ERDF: European Regional Development Fund" and the Ministry of Economy, Energy, Industry, Trade and Craft of North Rhine Westphalia – Germany, within the "Ziel2" program.

We thank our project partner dSPACE for providing detailed vehicle and traffic models, as well as specific HiL-simulation hardware. We thank our project partner Varroc Lighting Systems GmbH for providing a head light control module for adaptive bending lights.

REFERENCES

[1] B. Hassan and J. Gausemeier, "Concept for a task–specific reconfigurable driving simulator," in Proc. International Conference on Advances in System Simulation (SIMUL 2013), IARIA, pp. 40-46, 2013.

[2] T. Hummel, M. Kühn, J. Bende, and A. Lang "Advanced Driver Assistance Systems – An investigation of their potential safety benefits based on an analysis of insurance claims in Germany," German Insurance Association – Insurers Accident Research, Research Report FS 03, Berlin, 2011.

[3] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen, "Development of Advanced Driver Assistance Systems with Vehicle Hardware-in-the-Loop Simulations," The vehicle system dynamics, July 2006, volume 44, issue 7, pp. 569–590, 2006.

[4] M. Meywerk, "CAE-Methoden in der Fahrzeugtechnik," Springer-Verlag, Berlin, 2007.

[5] J. Gausemeier, P. Ebbesmeyer, and F. Kallmeyer, "Produktinnovation – Strategische Planung und Entwicklung der Produkte von Morgen," Carl Hanser Verlag München, 2011.

[6] J. Negele, "Anwendungsgerechte Konzipierung von Fahrsimulatoren für die Fahrzeugentwicklung," Ph.D. thesis, Faculty of Mechanical Engineering, 2007, Technische Universität München, Germany.

[7] S. Espié, E. Follin, G. Gallée, and D. Ganieux, "Automatic Road Networks Generation Dedicated to Night-Time Driving Simulation," in Proc. Driving Simulation Conference North America, 8.-10. October 2003, Dearborn, Michigan – ISSN 1546-5071.

[8] G. Weinberg and B. Harsham, "Developing a Low-Cost Driving Simulator for the Evaluation of In-Vehicle Technologies," in Proc. the First International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2009), September 21-22 2009, Essen, Germany.

[9] H. Jamson, "Cross-Platform Validation Issues," In: D. Fisher, J. Caird, M. Rizzo, J. Lee (Eds.): Handbook of Driving Simulation for Engineering, Medicine, and Psychology. CRC Press Taylor & Francis Group, USA, 2011, pp. 12.1-12.13 – ISBN 978-1-4200-6100-0.

[10] F. Filippo, A. Stork, H. Schmedt, and F. Bruno,"A modular architecture for a driving simulator based on the FDMU approach," International Journal on Interactive Design and Manufacturing (IJIDeM), Springer-Verlag, March 09 2013, Paris, France, 2013, ISSN 1955-2513.

[11] D. Gue, H. Klee, and E. Radwan, "Comparison of Lateral Control in a Reconfigurable Driving Simulator," in Proc. Driving Simulation Conference North America, 2003, Dearborn, Michigan, USA.

[12] E. Zeeb, "Daimler's new full-scale, high-dynamic driving simulator – A technical overview," in Proc. Driving Simulation Conference Europe 2010, September 9-10 2010, Paris, France, pp. 157-165 – ISBN 978-2-85782-685-9.

[13] National Advanced Driving Simulator, "Overview 2010," The University of Iowa – National Advanced Driving Simulator, Iowa City, USA, 2010.

[14] I. Gräßler, "Kundenindividuelle Massenproduktion". Springer-Verlag Berlin, 2004 – ISBN 978-3-642-18681-3

[15] S. Kreft, J. Gausemeier, M. Grafe, and B. Hassan "Automated generation of roadways based on geographic information systems," ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, Washington DC, USA, 28-31 Aug. 2011.

[16] J. Gausemeier, U. Frank, J. Donoth, and S. Kahl,"Specification technique for the description of self-optimizing mechatronic systems," Research In: Research in Engineering Design, November 2009, Volume 20, Issue 4, Springer, London, 2009, pp. 201-223 – ISSN 0934-9839.

[17] M. Vaßholz, and J. Gausemeier, "Cost-Benefit Analysis – Requirements for the Evaluation of Self-Optimizing Systems," in Proc. 1st Joint International Symposium on System Integrated Intelligence 2012 – New Challenges for Product and Production Engineering, June 27-29 2012, Hannover, Germany, 2012, pp. 14-16.

[18] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote, "Konstruktionslehre – Grundlagen erfolgreicher Produktentwicklung – Methoden und Anwendung," Springer-Verlag, Berlin, 7. Auflage, 2007.

[19] H. Birkhofer, "Analyse und Synthese der Funktionen technischer Produkte," VDI-Verlag Fortschritts-Bericht VDI-Z, Reihe 1, Nr. 70, Düsseldorf, Germany, 1980.

[20] G. Langlotz, "Ein Beitrag zur Funktionsstrukturentwicklung innovativer Produkte. Forschungsberichte aus dem Institut für Rechneranwendung in Planung und Konstruktion," RPK der Universität Karlsruhe, Shaker Verlag, 2000.

[21] F. Zwicky, "Morphologische Forschung – Wesen und Wandel materieller und geistiger struktureller Zusammenhänge," Schriftenreihe der Fritz-Zwicky-Stiftung, Band 4, Verlag Baeschlin, Glarus, 1989 – ISBN 978-3-8135-0314-2.

[22] M. Brown, "Developing with Couchbase Server," O'Reilly Media, Sebastopol, California, USA, February 2013 – ISBN 978-1-4493-3116-0.

[23] C. Schmidt, "How to Make an AFS System Predictive: ADASIS Interface Implementation," in Proc. 7th International Symposium on Automotive Lighting, September 25-26, 2007, Darmstadt, Germany.