## Method and Tools Specifications

<table>
<thead>
<tr>
<th>Deliverable n.</th>
<th>D1.3.2 – Method and Tools Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sub Project</td>
<td>SP1</td>
</tr>
<tr>
<td>Workpackage</td>
<td>WP1.2</td>
</tr>
<tr>
<td>Task n.</td>
<td>T1.3.2</td>
</tr>
<tr>
<td>Authors</td>
<td>N. Pallaro</td>
</tr>
<tr>
<td></td>
<td>A. Ghiro</td>
</tr>
<tr>
<td></td>
<td>CRF</td>
</tr>
<tr>
<td></td>
<td>CRF</td>
</tr>
<tr>
<td>File name</td>
<td>DESERVE_D13.2_MethodandToolsSpecifications_v22.doc</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Distribution</td>
<td>Public (PU)</td>
</tr>
<tr>
<td>Issue date</td>
<td>02.08.2013</td>
</tr>
<tr>
<td>Creation date</td>
<td>08.05.2013</td>
</tr>
<tr>
<td>Project start and duration</td>
<td>1st of September, 2012 – 36 months</td>
</tr>
</tbody>
</table>
## REVISION AND HISTORY CHART

<table>
<thead>
<tr>
<th>VER.</th>
<th>DATE</th>
<th>AUTHOR</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>08.05.2013</td>
<td>N. Pallaro (CRF)</td>
<td>Template, first draft</td>
</tr>
<tr>
<td>0.2</td>
<td>05.06.2013</td>
<td>N. Pallaro (CRF)</td>
<td>Review of template</td>
</tr>
<tr>
<td>0.3</td>
<td>07.06.2013</td>
<td>A. Ghiro (CRF)</td>
<td>Paragraphs 2.1, 2.2 (flow chart and introduction)</td>
</tr>
<tr>
<td>0.4</td>
<td>12.06.2013</td>
<td>M. Kunert (Bosch)</td>
<td>Paragraphs 1.1, 2.2.3, 3.3.2.</td>
</tr>
<tr>
<td>0.5</td>
<td>13.06.2013</td>
<td>G. Payá Vayá (IMS)</td>
<td>Chapter 4, Cost models on hardware implementations</td>
</tr>
<tr>
<td>0.6</td>
<td>28.06.2013</td>
<td>A. Rolfsmeier (dSPACE)</td>
<td>Paragraphs 2.2.7, 2.2.8, 2.2.9 and 3.2.2</td>
</tr>
<tr>
<td>0.7</td>
<td>05.07/2013</td>
<td>A. Ghiro (CRF)</td>
<td>Paragraphs 2.1 (review), 2.2 (review), 2.2.10</td>
</tr>
<tr>
<td>0.8</td>
<td>09.07.2013</td>
<td>I. Camuffo (CRF)</td>
<td>Paragraph 3.2.1</td>
</tr>
<tr>
<td>0.9</td>
<td>09.07.2013</td>
<td>J. Perez (INRIA)</td>
<td>Paragraphs 2.2.6, 3.1.3</td>
</tr>
<tr>
<td>1.0</td>
<td>09.07.2013</td>
<td>N. Pallaro (CRF)</td>
<td>Paragraphs 2.2.1, 2.2.2</td>
</tr>
<tr>
<td>1.1</td>
<td>12.07.2013</td>
<td>G. Dunand (Intempora)</td>
<td>Paragraphs 2.2.4, 2.2.5, 3.1.1, 3.1.3, 3.3.1, 3.3.3, 3.4.1 (contribution), 3.4.2.</td>
</tr>
<tr>
<td>1.2</td>
<td>15.07.2013</td>
<td>N. Pallaro (CRF)</td>
<td>Chapter 1 and 5. Paragraph 3.1.2 Review of complete report.</td>
</tr>
<tr>
<td>1.3</td>
<td>16.07.2013</td>
<td>A. Ghiro (CRF)</td>
<td>Paragraph 3.4.1</td>
</tr>
<tr>
<td>1.4</td>
<td>17.07.2013</td>
<td>M. Kunert (Bosch)</td>
<td>Review of paragraph 2.2.3</td>
</tr>
<tr>
<td>1.5</td>
<td>19.07.2013</td>
<td>A. Ghiro (CRF)</td>
<td>Review of table no. 3 and 4</td>
</tr>
<tr>
<td>1.6</td>
<td>19.07.2013</td>
<td>N. Pallaro (CRF)</td>
<td>Executive summary, final version for peer review</td>
</tr>
<tr>
<td>1.7</td>
<td>22.07.2013</td>
<td>A. Ghiro (CRF)</td>
<td>Review of paragraphs 2.2.3, 2.2.4.1, 2.2.4.2, 2.2.5</td>
</tr>
<tr>
<td>1.8</td>
<td>24.07.2013</td>
<td>A. Zlocki (IKA)</td>
<td>Peer review</td>
</tr>
<tr>
<td></td>
<td>Date</td>
<td>Name (Laboratory)</td>
<td>Version</td>
</tr>
<tr>
<td>----</td>
<td>------------</td>
<td>-------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>1.9</td>
<td>26.07.2013</td>
<td>L. Gatti (ReLAB)</td>
<td>Peer review</td>
</tr>
<tr>
<td>2.0</td>
<td>02.08.2013</td>
<td>M. Bertozzi (VISLAB)</td>
<td>GOLD framework</td>
</tr>
<tr>
<td>2.1</td>
<td>02.08.2013</td>
<td>J. Klimke (IKA)</td>
<td>PELOPS simulation tool</td>
</tr>
<tr>
<td>2.2</td>
<td>02.08.2013</td>
<td>N. Pallaro (CRF)</td>
<td>Final version for ARTEMIS submission</td>
</tr>
</tbody>
</table>
# Table of Contents

## Revision and History Chart ................................................................. 2

## List of Figures ................................................................. 6

## List of Tables ................................................................. 6

## List of Acronyms ........................................................................... 7

## Executive Summary ........................................................................ 9

1. **Introduction** .................................................................................. 10
   1.1 Objectives and Scope of the Document .......................................................... 10
   1.2 Structure of the Deliverable ........................................................................... 11

2. **Architecture of the DESERVE Development Platform** .................... 11

3. **Methodology for Application Development** .................................... 13
   3.1 Beyond V-model Development Process .......................................................... 13
   3.2 Development Phases of DAS Applications .................................................... 13
      3.2.1 Use-cases definition.................................................................................. 14
      3.2.2 Requirements definition ........................................................................ 15
      3.2.3 Software architecture definition .............................................................. 15
      3.2.4 Configuration of simulators environments .............................................. 16
      3.2.5 Development of perception and fusion algorithms .................................. 18
      3.2.6 Development of specific control algorithms .......................................... 20
      3.2.7 MIL Verification & Validation ................................................................. 20
      3.2.8 SIL Verification & Validation .................................................................. 21
      3.2.9 HIL Verification & Validation .................................................................. 22
      3.2.10 Integration on vehicle ............................................................................ 23

4. **Development Tools Specifications** ............................................... 24
   4.1 Environment and Sensors Simulation Tool .................................................. 24
      4.1.1 Main tool specifications ........................................................................... 24
      4.1.2 PreScan (TASS) ..................................................................................... 25
      4.1.3 ProSiVIC (CIVITEC) ............................................................................... 26
      4.1.4 ASM (dSPACE) ..................................................................................... 27
      4.1.5 PELOPS (IKA) ....................................................................................... 27
   4.2 Vehicle Dynamics Simulation Tool .......................................................... 28
      4.2.1 Main tool specifications ........................................................................... 28
      4.2.2 ASM (dSPACE) ..................................................................................... 29
      4.2.3 PELOPS (IKA) ....................................................................................... 31
   4.3 Perception and Fusion Development Tool ............................................. 31
      4.3.1 Main tool specifications ........................................................................... 31
      4.3.2 ADTF (Elektrobit) .................................................................................. 32
      4.3.3 GOLD (University of Parma) ................................................................. 34
      4.3.4 RTMaps (Intempora) ............................................................................... 34
   4.4 Control Algorithms Development Tool ................................................. 36
      4.4.1 Main tool specifications ........................................................................... 36
      4.4.2 Matlab/Simulink/Stateflow (MathWorks) .............................................. 38

5. **Cost Models on Hardware Implementations** ................................... 40
5.1 DESIGN SPACE EXPLORATION FRAMEWORK ................................................................. 40
5.2 COST MODELS ................................................................................................................. 42

6. CONCLUSIONS .................................................................................................................. 43

REFERENCES ......................................................................................................................... 44
LIST OF FIGURES

FIGURE 1 - DESERVE DEVELOPMENT PLATFORM CONCEPT  12
FIGURE 2 - DEVELOPMENT PHASES OF DAS APPLICATIONS  14
FIGURE 3 - EXAMPLE OF DESERVE HW ARCHITECTURE  15
FIGURE 4 - INTERACTION BETWEEN RTMAPS AND THE SIMULATOR  16
FIGURE 5 - VEHICLE DYNAMIC SIMULATOR  17
FIGURE 6 - PROTOTYPING PHASE  17
FIGURE 7 - REAL-DATA PLAYBACK  18
FIGURE 8 - RTMAPS SDK  19
FIGURE 9 - RTMAPS ARCHITECTURE  19
FIGURE 10 - EXAMPLE OF AN GENERAL ARCHITECTURE FOR ARBITRATION AND CONTROL ON AUTONOMOUS VEHICLES  20
FIGURE 11 - TYPICAL PROCESS WITH MODEL-BASED DEVELOPMENT  21
FIGURE 12 - MODEL-IN-THE-LOOP SIMULATION  21
FIGURE 13 - SOFTWARE-IN-THE-LOOP SIMULATION  22
FIGURE 14 - HARDWARE-IN-THE-LOOP SIMULATION FOR ECU VALIDATION TESTS  23
FIGURE 15 - HOW DOES PRESCAN WORK?  26
FIGURE 16 - DIFFERENT VEHICLES AND SCENARIOS IN PRO-SIVIC.  27
FIGURE 17 - AUTOMOTIVE SIMULATION MODELS (ASM) AND PACKAGES FOR THE DEVELOPMENT OF DRIVER ASSISTANCE SYSTEMS  30
FIGURE 18 - EB ASSIST ADTF SCREENSHOT OF GUI  33
FIGURE 19 - SCREENSHOTS OF THE MAIN GOLD CONSOLE  34
FIGURE 20 - GLOBAL SCHEMA: INPUTS, PROCESSING AND OUTPUTS  35
FIGURE 21 - LOCAL MAP CONSTRUCTION WITH LANE DETECTION AND VEHICLES DETECTION,  35
FIGURE 22 - MODELLING AN AUTOMATIC TRANSMISSION CONTROLLER BY SIMULINK  38
FIGURE 23 - DESIGN SPACE EXPLORATION FRAMEWORK IN THE DESIGN FLOW FOR HETEROGENEOUS HARDWARE ARCHITECTURES  40
FIGURE 24 - OBJECTIVES IN THE DESIGN SPACE  41

LIST OF TABLES

TABLE 1 – SPECIFICATIONS OF ENVIRONMENT AND SENSORS SIMULATION TOOL  24
TABLE 2 – SPECIFICATIONS OF VEHICLE DYNAMICS SIMULATION TOOL  29
TABLE 3 – SPECIFICATIONS OF PERCEPTION AND FUSION DEVELOPMENT TOOL  31
TABLE 4 – SPECIFICATIONS OF CONTROL ALGORITHMS DEVELOPMENT TOOL  37
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>ABBREVIATION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>Three dimensional</td>
</tr>
<tr>
<td>ADAS</td>
<td>Advanced Driver Assists Systems</td>
</tr>
<tr>
<td>ADTF</td>
<td>Automotive Data and Time-Triggered Framework</td>
</tr>
<tr>
<td>ADTF</td>
<td>Automotive Data and Time triggered Framework</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASM</td>
<td>Automotive Simulation Model</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>AUTomotive Open System ARchitecture</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>DAS</td>
<td>Driver Assistance System</td>
</tr>
<tr>
<td>EB</td>
<td>Elektrobit</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic control unit</td>
</tr>
<tr>
<td>ESP</td>
<td>Electronic Stability Program</td>
</tr>
<tr>
<td>GPS</td>
<td>Global positioning system</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HIL</td>
<td>Hardware-in-the-loop</td>
</tr>
<tr>
<td>HMI</td>
<td>Human Machine Interface</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communication Technology</td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial measurement unit</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>LIN</td>
<td>Local Interconnect Network</td>
</tr>
<tr>
<td>Matlab</td>
<td>Matrix Laboratory (Mathworks, Inc.)</td>
</tr>
<tr>
<td>MIL</td>
<td>Model-in-the-loop</td>
</tr>
<tr>
<td>MOST</td>
<td>Media Oriented Systems Transport</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PIL</td>
<td>Processor In the Loop</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SIL</td>
<td>Software-in-the-loop</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification &amp; Validation</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

The **purpose of D132 deliverable** (work package 1.3 dealing with the specification of the DESERVE development platform) is to define the methodology for the ADAS application development based on DESERVE platform.

The complete tool-chain may become heterogeneous and manifold, depending on the scope and possible field of application the DESERVE platform is intended to be used in the demonstrator vehicles. The DESERVE platform system may work at least on **three different development stages**:

- Fully PC-based HW platform;
- Mixed PC and embedded controller platform;
- Embedded and custom ASIC HW-framework.

The big challenge is to close the gap between these three development axes and make the whole development process as seamless and integrated as possible.

The DESERVE development process has to **adapt the actual V-model cycle** in order to:

- Provide a common environment for design, development and testing of ADAS functions;
- Provide a common environment for coexistence of ADAS functions;
- Allow reuse of pre-validated software components.

The **main phases of the development process** of DAS applications are described:

- Concept (use cases and requirements definition, coexistence between ADAS functionalities);
- Implementation (configuration of environment and sensors simulators, software architecture definition, development of perception / fusion and control algorithms);
- Verification & Validation (Model In the Loop, Software in the Loop and Hardware in the Loop);
- Integration on vehicle.

The report also defines the **specifications of the development tools** to be used in the DESERVE platform:

- Environment and sensor simulation;
- Vehicle dynamics simulation;
- Perception and fusion development;
- Control algorithms development.

A benchmarking of the currently available tools was conducted by the partners and the tools were already selected by many of the partners in their respective demonstrators.

The last chapter describes **cost models on hardware implementations** based on standard components, FPGA, and System On Chip.
1. INTRODUCTION

1.1 Objectives and scope of the document

The selection of an appropriate and well-suited tool-chain to operate the DESERVE platform is at least as important as the specification and selection of the necessary hardware and software components.

During the first discussions among the DESERVE partners it became obvious that the realization of the DESERVE platform would have been practiced in different ways regarding the physical implementation and architectural concepts that go through the different demonstrators build up by the partners in the different European countries.

A scanning and benchmarking of the currently available tools was conducted by the partners and the tools selection was already done by many of the partners in their respective demonstrators.

The complete tool-chain may become heterogeneous and manifold, depending on the scope and possible field of application the DESERVE platform is intended to be used in the demonstrator vehicles. The DESERVE platform system may work at least on three different development stages:

- Fully PC-based HW platform;
- Mixed PC and embedded controller platform;
- Embedded and custom ASIC HW-framework.

The big challenge is to close the gap between these three development axes and make the whole development process as seamless and integrated as possible.

The purpose of D132 deliverable (output of work package 1.3, dealing with the complete specification of the DESERVE development platform) is to define the methodology to be followed during the development of the safety applications, together with the specifications of the development tools to be used in the DESERVE platform. In the course of this document the different tool-chain variants are described in the light of their requirements, functionalities and needs.

D132 starts from the outcome of “D121 Development Platform Requirements” [1][2] in terms of definition of general requirements of DESERVE platform and ADAS rapid prototyping platform; the requirements and user needs (e.g. user friendly graphical user interface) are described in the report for each specific tool.

This report is also strictly linked with “D213 Development method (first release)” [3] (output of work package 2.1 focusing on the identification, development and integration of tools and development systems of the overall platform), where more detailed guidelines will be given on how to use the DESERVE platform. D132 report introduces the methodology from a more general point of view with an overview of possible tools, D213 is focused on the specific tools associated with the DESERVE Development Platform and rapid prototyping.
1.2 Structure of the deliverable

The report is structured with the following chapters:

- Chapter 1 (current one) provides an overview with the scope of the document and the structure in chapters;
- Chapter 2 introduces the methodology for the application development, focusing on the different development phases of DAS applications;
- Chapter 3 describes the development tools specifications adopted in DESERVE project, defining the main target specifications and comparing them with respect to some specific tools;
- Chapter 4 addresses cost models on hardware implementations based on standard components, FPGA, and System On Chip.

2. ARCHITECTURE OF THE DESERVE DEVELOPMENT PLATFORM

The architecture of the DESERVE development platform shall follow both the principle of standard DAS development cycles (see next paragraph 3.2, Figure 2) and the mappings of application building blocks to final, often heterogeneous hardware implementations (see chapter 5.1).

To date there is no tool or framework available that covers both requirements at the same time on the same platform.

In the early concept and implementation phase the basic development, specification and validation (e.g. with MIL, SIL or HIL) is often done with another development framework (both for SW and HW) than the one applied for the final target platform. Little is known or taken into account from the final embedded system characteristics when first application algorithms are programmed and very often the SW modules written in this first development environment have to be reprogrammed from the scratch when porting it to the embedded system on chip. If the software, mostly written in a high-level programming language, finally fits the target system one has selected for series production, is a game of pure chance and not rarely during the series product development cycle a larger target system or some “add-ons” have to be chosen. With the new design space exploration methodology the certainty to select the suitable embedded target system at first time is significantly increased.

The DESERVE development platform architecture has to comply with the following basic needs:

1) Enough flexibility to encompass different development environments in a common, seamless framework for both the high-level algorithm development and the easy porting of these SW modules to the embedded target platform.

2) Real time recording and playback capabilities for both the high-level and embedded system implementations.

3) A communication architecture that is capable to shift SW portions from the high-level development side to the embedded target system as required (i.e. bypassing with HW accelerators).

4) A seamless interoperability and replacement between the high-level (i.e. PC-based) and embedded target systems both for development and validation purposes.
How the above mentioned architectural requirements can be implemented in a dedicated DESERVE platform is shown in the following for the Daimler Demonstrator “Inter Urban Assist” that was elaborated and finally adopted by the partners involved in WP46 (Figure 1).

The basic idea and intention of this hardware architecture is to standardize the interfaces between the three different development concept levels as good as possible:

- Level 1: PC-based development framework
- Level 2: Embedded controller development framework
- Level 3: Dedicated ASIC development framework

Inputs from proprietary ADAS sensor systems and information sources are analysed via a generic interface no.1 to the PC based development environment. Here the ADTF tool with its filter programming concept is used to develop or improve SW modules on a high-level programming language. The partitioning and optimization of parts of the SW modules is consecutively done by shifting such portions over the generic interface no.2 to the embedded controller framework that is already much nearer to the final commercial product. Via this bidirectional interface bypassing techniques like PIL (embedded Processor In the Loop) can be realized. In a final step, dedicated HW accelerators can be linked in via the generic interface no.3 by applying the same bypassing concept. Especially computationally intensive tasks can so be “outsourced”, so that even the PC-based platform is capable to keep the stringent real-time constraints.

Depending on the performance of the PC either all or only specific parts of the SW modules can be executed there. During the development process more and more SW parts are transferred to the HW-Accelerator level, which, in the final development stage, results in the next generation embedded ADAS target system. At this last development step, the level 1 (PC) and level 2 (embedded controller) platform will only serve as a shell to keep up the overall development framework.

Reuse of already existing components from former ADAS generations may be used in the early development phase as HW accelerators for computational intensive calculations. Mainly standard algorithms that are fixed and receive no further modifications are preferred candidates for such specific HW accelerators.
3. METHODOLOGY FOR APPLICATION DEVELOPMENT

3.1 Beyond V-model development process

DESERVE development process has to adapt the actual V-model cycle \([4][5]\) in order to achieve the following main results:

- Provide a common environment for design, development and testing of ADAS functions;
- Provide a common environment for coexistence of ADAS functions;
- Allow reuse of pre-validated software components.

Consolidated methodologies exist to guide the development process and the validation of new safety systems. With integration of ADAS it is needed to step beyond the traditional V-Model-based system engineering. It is needed to establish a consolidated design and development validation environment where new components can be embedded and functions can be developed and tested.

Differently from today development phases, individual functions should be designed from the beginning in such a way that they operate within a common environment, with shared resources, where the different ADAS functions will not simply “live together”, but coexist and deeply cooperate by providing their assistance to the drivers simultaneously and in an interrelated way.

Development of new ADAS functions will be done using pre-validated software components. This software components reuse, in particular about the interpretation of the vehicle’s surroundings and of the driver behaviour, will allow to rapid qualification or certification of compositionally designed systems and especially rapid re-qualification or re-certification after change.

Without the need to re-qualify all systems, but only the aspects related to the specific newly integrated application of ADAS functions, DESERVE platform will enable the evolution of ADAS functions, managing the system complexity, reducing overall costs (fixed and variable) and improving safety and robustness.

3.2 Development phases of DAS applications

During the development process of DAS applications several phases have to be completed. The below chart (Figure 2) shows the idea behind DESERVE round-trip.

The process starts from the Concept phase. Here use-cases of interest and requirements of the developing DAS function are defined and collected. Needs for standards compliance (i.e. AUTOSAR \([6]\) and for Functional Safety (i.e. ISO 26262 \([7][8][9][10][11]\)) have to be considered in this early phase. Coexistence between ADAS functionalities have to be considered.

Starting from requirements defined in the Concept phase two parallel activities of the Implementation phase can start. The first one refers to the configuration of simulators environments (Vehicle, Sensors, Scenarios) with the goal to implement the defined use-cases. The second activity is the development of perception, fusion and specific control algorithms based on defined HW and SW architectures. In order to realize the coexistence between ADAS functionalities, this phase have to be developed taking in account modular architectures and software (pre-validated) components reuse.
The outputs provided by the implementation activities are used for following **Verification and Validation** phase. Here different V&V steps (Model In the Loop, Software in the Loop and Hardware in the Loop) allow to check requirements compliancy and to provide robust and safe code. In case of unexpected behaviour or tests failure, the process allows to update the requests with the objective to modify the output of the implementation phase or, if necessary, of the concept phase.

The last phase of DESERVE round-trip is the **Integration** on vehicle. DAS functionality is integrated into the vehicle. Final tests are performed. In case of residual problems, Implementation and/or Concept updates are performed.

### 3.2.1 Use-cases definition

In the Concept phase the scope of **Use Cases** is to define how the addressed safety functions should prevent/mitigate the undesired outcomes (road accidents, traffic rule violations) related to the target scenarios. The use cases definition starts from the flow of events based on the target scenarios and describes how the safety function, by means of interaction with the driver and/or direct intervention with vehicle control, prevents/mitigates the undesired outcome defined by the target scenario.

The use case-based methodology is today a standard practice in industrial system development and various models for defining use cases exist. However, these models are generally not optimal for use with active safety systems. In particular, there is usually no
explicit link between use cases and the target accidents that they address. The key role of the use cases is to provide a fairly general description of the intended functionality of the envisioned systems as a basis for the more detailed specification of functional requirements.

### 3.2.2 Requirements definition

There are also numerous tools (e.g. POLARION) available for **functional requirements** definition used for product development support and control. Typically, requirements constitute a hierarchical structure in a process that starts from a defined problem. The requirements can be hierarchically organised starting from a general need-type of requirement describing what the function has to perform in order to provide the desired outcome. This can be further specified by defining the operating conditions under which the application needs to be functional and then specifying in more detail aspects related e.g. to the performance, operation or usability. These definitions will then lead to specification phase where the actual system parts and components are specified to fulfil the requirements. So, to sum up, requirements form an intermediate process between the problem definition and the specification phase enabling the actual development work.

### 3.2.3 Software architecture definition

The goal of the task is the **definition of the software architecture** and the mapping of each software component into the hardware architecture defined in the DESERVE platform and mainly constituted of an embedded HW or PC running perception and fusion algorithms and a rapid prototyping ECU running control algorithms (Figure 3). HW architecture could be extended with the introduction of additional components like sensors and/or actuators. This activity has to satisfy the requirements defined into the concept phase and to provide the basis for the development of control, perception and fusion algorithms. Standardisation of interfaces, communication networks, etc. and the definition of a modular architecture is the way to improve the re-usability and the robustness of the DESERVE developed functionalities without continuously re-qualify all systems, but only the aspects related to the specific integrated application of ADAS functions. Tools based on UML language have to be integrated into the DESERVE tool chain in order to achieve these results. Another example is System Desk, a tool which allows defining SW architectures based on the AUTOSAR standard.

![Figure 3 - Example of DESERVE HW architecture](image-url)
3.2.4 Configuration of simulators environments

Use of simulators in the toolchain for the development of embedded functions has many advantages:

- capability to work offline in a reproducible context;
- capability to test situations which are difficult to run in the real-world (dangerous or rare);
- capability to provide a ground truth for perception algorithms validation;
- capability to test many different configurations in terms of sensor models, positions, combinations, etc.

There are various kinds of simulators which can be used in the frame of the DESERVE project. Mainly three different kinds can be distinguished:

- Sensors and environment
- Vehicle dynamics
- Real-data playback systems

3.2.4.1 Sensors and environment simulators

Such simulators have capability to place a virtual vehicle in a 3D environment and equip it with virtual sensors such as virtual cameras, laser-scanners, IMUs, GPS, radars, ultrasound etc. They can run given scenarios either in real-time or in simulated time, and can allow to log virtual sensors data but also most of the time to establish inter-process communication with other software which are often dedicated to embedded software development (Matlab/Simulink, RTMaps, ADTF, etc.).

Such architectures will be mainly used in DESERVE to develop, benchmark and validate the perception functions for the ADAS systems.

Figure 4 - Interaction between RTMaps and the simulator
Tools dedicated to process environment and sensors data have to be interfaced with simulators via dedicated components. These components represent inputs for the diagram and can be replaced later by components reading real data. The advantage is that the main part of the diagram (algorithms and controls) remains unchanged. The same mechanism for writing.

From RTMaps side, values reading from the simulator is done via a dedicated component (Figure 4).

### 3.2.4.2 Vehicle dynamics simulators

The capability for a simulator to compute the dynamic vehicle state allows to close the loop and to develop, test, validate and benchmark control algorithms (Figure 5).

**Figure 5 - Vehicle dynamic simulator**

Such tools exploit various kinds of solvers and models based on ordinary or partial differential equations.

Matlab/Simulink for instance can be used as the base framework for running such models, as well as control algorithms in close loop.

During prototyping phases perception and decision algorithms streams data from and to Simulink which runs the vehicle dynamics models and control laws in order to close the loop between vehicle and environment (Figure 6). After verification and validation phases it would be replaced by the real vehicle.

**Figure 6 - Prototyping phase**
3.2.4.3 Real-data playback

The drawback of synthetic data computed by simulators is that it can never be perfectly realistic. Using tools like RTMaps or ADTF will allow to record sensors data in real-time from prototype vehicles, then exploit the datasets in playback mode for offline developments and validation works (Figure 7).

This allows going one step further in the validation of the perception and decision algorithms but does not allow to close the loop for control algorithms as it is not possible to apply feedback to the pre-recorded datasets.

3.2.5 Development of perception and fusion algorithms

Tools suited for development of perception and fusion algorithms have to guarantee a way to prototype efficiently on new algorithms and new systems by providing a modular environment to easily test and evaluate functions based on different sets of sensors, in different configurations, and different sets of processing and data fusion strategies.

The development or integration of perception algorithms have to be done via an SDK (Software Development Kit) (Figure 8).

The "Software Development Kit" allows the development of additional components (as plugins) and their addition in the Standard Library. The programming has to be done in C or C++; it should be facilitated by code skeletons generated automatically by the SDK Wizard. Moreover, a complete cross-platform API (Application Programming Interface) has to provide access to all the engine functions and to remain independent from the operating system (file system or real time programming for example) (Errore. L'origine riferimento non è stata trovata.).
For data fusion algorithms, it is often a complex issue to ensure synchronization of the data to be fused and which originates from multiple asynchronous sources (cameras, CAN bus, etc.). Perception tools has to provide various methods to ensure that the developer can implement easily the best-suited synchronization policy for his/her functions taking advantage of the timestamps associated to each data sample (periodic re-sampling of multiple inputs, event-based callbacks upon samples arrival on any of the inputs, triggering by one of the inputs, synchronized reading allowing to re-synch the streams at any point using samples time stamp independently from the samples latency, sorting of the samples by increasing order of timestamps...).
3.2.6 Development of specific control algorithms

The Figure 10 shows the different level of the control architecture for autonomous vehicles. Based on the information from sensors and driver stage, different control algorithms will be defined. The on board control is divided in two different types: for the arbitration and autonomous control of the vehicle.

![Diagram of control architecture for autonomous vehicles](image)

**Figure 10 - Example of an general architecture for arbitration and control on autonomous vehicles**

The arbitration part is in charge of to manage the driver requirements (through a HMI), driver stage (from different sensors) and data fusion information (environment information) to decide which control mode (autonomous or manual) is usable. In the context of the DESERVE project, the arbitration will be in charge to manage the different ADAS used in the demonstrators.

The autonomous control is divided in two forms: lateral and longitudinal (steering wheel and pedals, respectively). The lateral control will use the information from the sensors to keep a predefined trajectory, whereas that the longitudinal control will keep the reference speed in order to reduce the speed error.

3.2.7 MIL Verification & Validation

The model-based design of ECU software as shown in Figure 11 is increasingly being used in the automotive industry. Especially with driver assistance systems, this approach allows engineers to evaluate and verify functional concepts on a PC early in the development process by means of the so-called model-in-the-loop (MIL) simulation and to reuse plant models and test libraries in subsequent development stages comprising software-in-the-loop (SIL) and hardware-in-the-loop (HIL) simulation [12].
Applying the MIL approach, controller algorithms are developed and implemented by means of dedicated models that can be simulated in a block diagram environment providing graphical editors, block libraries and solvers for modeling and simulating dynamic systems. For closing the control loop suitable plant models are required which are mathematical representations of the associated system under control (Figure 12). This way MIL serves as a convenient and cost-efficient method to verify and validate both controller and plant models in an early development stage on a PC by means of simulation. Development environments which are commonly used in this context are Matlab®/Simulink® and ASCET®.

**3.2.8 SIL Verification & Validation**

The SIL approach allows the direct integration of control algorithms in terms of target code in a simulation environment (Figure 13). Typically, the target code is C-code which was automatically generated, for example, from the controller models designed during MIL.
Simulation. The target code is connected with plant models and simulated in a closed loop on
the developers PC.
The benefit of SIL is that the target code can be simulated and verified without having the
final electronic control unit (ECU) available. Even if the ECU hardware is not defined yet,
developers are able to test the target code in an early development stage.
Software-in-the-loop can thus be viewed as a PC-based method to verify and validate the
actual controller software which is the same code that runs on the final hardware controller.
Therefore, SIL offers the possibility to execute tests before the hardware is available.

![Software-in-the-Loop simulation on host PC](image)

3.2.9 **HIL Verification & Validation**

Today, ECU software is typically tested for production use with real-time hardware-in-the-
loop simulators. Here, the plant models are calculated in real-time and they are connected
to the ECU(s), the device(s) under test, via the vehicle bus and dedicated I/O interfaces.
With HIL (Figure 14), the simulation of these components have to be as accurate as it is
required to run the ECU(s) without generating diagnostic trouble codes in the on-board error
memory. Typically, the plant models have to provide mathematical representations of the
related dynamic systems. For example, an HIL simulator for the validation of an automotive
anti-lock braking system may have mathematical representations in the plant model for the
vehicle dynamics such as suspension, wheels, tires, road characteristics and dynamics of the
brake system's hydraulic components.

HIL simulation is a technique that can be applied to the verification of a single ECU (called
component verification, for example for an engine or anti-lock braking ECU) and for
networked ECUs related to a complete system (called system verification or integration
tests). In particular in the ADAS context the associated algorithms are distributed across
several ECUs and the final validation for production use is done by HIL integration tests.
Often also real-vehicle components (real parts, such as the throttle or injection valves) are
connected via their electrical interfaces to the simulator, especially when the associated
component models are not accurate enough for a certain verification task.
An HIL simulation often also includes electrical emulation of sensors and actuators. These electrical emulations act as the interface between the plant model and the ECU(s). The value of each electrically emulated sensor is controlled by the plant model and is read by the ECU(s). Likewise, the ECU(s) calculate the control algorithms and output actuator control signals. Changes in the control signals result in changes to variable values in the plant simulation.

![Diagram of Hardware-in-the-Loop simulation](image)

**Figure 14 - Hardware-in-the-Loop simulation for ECU validation tests**

### 3.2.10 Integration on vehicle

The integration on vehicle phase allows verifying and validating the complete HW and SW system developed. A wide range of tests is performed in order to ensure the ADAS functionality meets its design requirements. Testing program, for example, has to evaluate:

- Performance, efficiency, and durability
- Structures and components
- Environmental capabilities
- Electromagnetic compatibility

This task require the normal production ECU, sensors and actuators programmed with the final software release, then involve OEM and supplier(s). *These aspects are not of interest for DESERVE project and therefore this phase will not be approached.* Anyway the DESERVE development process has to consider also the update requests that can arise also in this final step.
4. DEVELOPMENT TOOLS SPECIFICATIONS

The purpose of this chapter is to define the main features of development and simulations tools to be used in DESERVE. It provides also a general description of the basic functionalities of the interesting tools for DESERVE. A deeper analysis and comparison of the available tools and the relative development methods will be done in D213 “Development method (first release)” deliverable.

4.1 Environment and sensors simulation tool

4.1.1 Main tool specifications

The development, testing and validation of multifunctional Advanced Driver Assists Systems (ADAS) are overwhelming tasks. It requires testing for a wide variety of driving manoeuvres and critical situations that the system should recognise and handle. Moreover, changes in environmental conditions will ever trim down the detection performance.

For this reason environment / sensors modelling and simulation software platforms [13] offer engineers the opportunity to perform functional design up to design validation of their driving assistance system from the early stages of the development cycle.

The above mentioned simulation tools allow to:

- Reproduce test scenarios for a wide range of environment and traffic conditions
- Emulate multiple perception sensors with realistic distortion effects
- Add system control with Matlab/Simulink interface
- Run tests with single or batch of scenarios

In the table below the required main features of environment and sensors simulation tools are summarised.

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large library of sensor models</td>
<td>The simulators needs to provide a large library of configurable sensor models (cameras, lidars, radars, ultrasonic sensors, GPS, etc..)</td>
</tr>
<tr>
<td>Large library of environments</td>
<td>The simulators have to provide a large library of pre-existing environment for various situations (highways, urban and inter-urban roads, crossings, etc.).</td>
</tr>
<tr>
<td>Customizability</td>
<td>The sensors models have to be configurable. The environments have to be editable. The scenarios have to be easily designed (via high level scripting or graphical tools).</td>
</tr>
</tbody>
</table>
# 4.1.2 PreScan (TASS)

PreScan by TASS International is a physics-based simulation platform that is used in the automotive industry for development of Advanced Driver Assistance Systems (ADAS) that are based on different sensor technologies. PreScan is also used for designing and evaluating vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication applications. PreScan can be used from model-based controller design (MIL) to real-time tests with software-in-the-loop (SIL) and hardware-in-the-loop (HIL) systems.

The program works using four easy steps (Figure 15):

1. **Build scenario**
   A dedicated pre-processor (GUI) allows users to build and modify traffic scenarios within minutes using a database of road sections, infrastructure components (trees, buildings, traffic signs), actors (cars, trucks, bikes and pedestrians), weather conditions (such as rain, snow and fog) and light sources (such as the sun, headlights and lampposts). Representations of real roads can be quickly made by reading in information from OpenStreetMap, Google Earth, Google 3D Warehouse and/or a GPS navigation device.

2. **Model sensors**
   Vehicle models can be equipped with different sensor types, including radar, laser, camera, ultrasound, infrared, GPS and antennas for vehicle-to-X (V2X) communication. Sensor design and benchmarking is facilitated by easy exchange and modification of sensor type and sensor characteristics.

3. **Add control system**
   A Matlab/Simulink interface enables users to design and verify algorithms for data processing, sensor fusion, decision making and control as well as the re-use of existing Simulink models such as vehicle dynamics models from CarSim, Dyna4 or ASM.

4. **Run experiment**
   A 3D visualisation viewer allows users to analyse the results of the experiment. It provides multiple viewpoints, intuitive navigation controls, and picture and movie generation...
capabilities. Also, interfaces with ControlDesk and LabView can be used to automatically run an experiment batch of scenarios as well as to run hardware-in-the-loop (HIL) simulations.

![Diagram showing PreScan workflow](image)

**Figure 15 - How does PreScan work?**

The main **PreScan software features** can be summarized as follows:

- Large library of sensor models including radar, laser, camera, ultrasound, infrared, GPS and antennas for vehicle-to-X (V2X) communication
- Ground truth sensors models (SELF sensor, depth camera, lane marker sensor)
- Generation of a wide range of virtual traffic and road environments.
- Manoeuvre control (Open-loop manoeuvres with prescribed motion, Closed-loop manoeuvres with PreScan vehicle dynamics, Closed-loop manoeuvres with 3rd party vehicle dynamics, Driver-in-the-Loop using steering console)
- Interfacing with Matlab/Simulink and different vehicle dynamics models (CarSim, Dyna4 or ASM)

### 4.1.3 **ProSiVIC (CIVITEC)**

Pro-SiVIC by CIVITEC is another powerful software environment for the assessment of sensor robustness and reliability, and for the rapid and proven completion of perception and detection systems validation. It comes easy to generate numerous variations of environmental conditions while re-playing the same identical scenario to assess (Figure 16).
Pro-SiVIC simulator offers a multi-sensorial environment, and takes into account several parameters of a real car such as the inertia, steering wheel response, lateral acceleration with yaw angles, damping suspension, simple weather conditions, friction parameters and more. Pro-SiVIC has been successfully used in path planning, vehicle control (lane following, road departure avoidance, collision mitigation and collision avoidance, speed regulation, etc.), perception application prototyping, ADAS test and validation.

Moreover, synchronized time, acceleration (in wheel torque), steering, odometer information, lidar information and camera viewports are some of the components supporting the connection between the control architecture in RTMaps and the simulation.

4.1.4 **ASM (dSPACE)**

ASM by dSPACE is a further environment tool that will be described more in detail in D213 “Development method (first release)” report.

4.1.5 **PELOPS (IKA)**

The traffic simulation tool PELOPS consists of three basic modules. The tool focuses on the interactions of driver, vehicle and environment. Each module called driver model, vehicle model and environmental model is designed independently but with well-defined interfaces. Thus it is possible to choose different vehicles and driver types.

The influences of the traffic environment can be adequately represented by the environment model. The course of the road is described not only by radii and transitions in horizontal and vertical direction but also the number and width of lanes, etc. In addition to this geometric information, the traffic signs and environmental parameters can also be simulated.

This information is collected in an interface structure called “driving view” and is submitted to the driver module which can react on these data. In the next calculation time step the vehicle data (steering angle, acceleration, etc.) is taken to calculate the updated environment status.

PELOPS also contains sensor models implementing the sensor characteristics (accuracy and resolution of different signals, detection area, sample time, mounting position, etc.).

A detailed description of PELOPS will be released in D311 “Standard driver model definition”.

---

**Figure 16 - Different vehicles and scenarios in Pro-SiVIC.**
4.2 Vehicle dynamics simulation tool

4.2.1 Main tool specifications

For driver assistance systems which interact with the steering, braking or throttle control, a detailed model of the vehicle and its dynamic behaviour is essential. The simulation tool used to predict the vehicle dynamics behaviour of the vehicle shall include a mathematical model capable of calculating variables of interest for the test procedures being simulated.

For this purpose the vehicle dynamics simulation tool should be based at least on a 14 degrees of freedom vehicle model, having the following subsystems:
- Vehicle body
- Wheels and tires
- Primary suspensions
- Steering system
- Powertrain
- Brake system (basic)

The “body” subsystem should include a six degrees of freedom rigid part representing the vehicle overall sprung mass. Essential properties of it are the value of the mass, the location of the centre of gravity and moments and products of inertia.

The “wheels and tires” subsystem should include two degrees of freedom rigid part for each corner, representing the total unsprung mass related to the corner. A tire model is necessary to describe vertical, lateral, and longitudinal contact forces and moments between tire and road. Steady-state behaviour as well as transient dynamics shall be included in the tire model.

The “primary suspensions” subsystem should include the characteristic curves that determine how the wheel is located and oriented with respect to the vehicle body (under the action of suspension jounce and contact forces and moments) as well as how forces and moments from the tires are transferred to the sprung mass.

The steering system interacts with the suspensions to determine how the tire is oriented on the ground: hence the “steering” subsystem should include kinematical and compliance relationships needed to calculate the road wheel angles from the steering wheel angle.

The “powertrain” subsystem should include the description of how the engine torque is transferred to the drive wheels through clutch, transmission and differentials.

The “brake” subsystem should include at least the functional model of the actuators (brake torques as a function of corner pressures) and a generic function to define rear pressures as a function of the front ones.

Other generic requirements of the simulation tool are:
- the possibility to easily integrate external subsystem models (brakes, driveline, ICT-based safety systems) developed in Matlab/Simulink;
- the possibility to easily supply vehicle model input data;
- the possibility to easily define driver inputs on primary controls (steering wheel, gas and brake pedals, clutch, gear) in order to simulate generic vehicle dynamics manoeuvres;
- the possibility to easily customize simulation output channels and to import the simulation output files in Matlab.

In the table below the **required main features** of vehicle dynamics simulation tools are summarised.

### Table 2 – Specifications of vehicle dynamics simulation tool

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle mathematical model</td>
<td>14 degrees of freedom vehicle model, including vehicle body, wheels and tires, primary suspensions, steering system, powertrain and basic brakes.</td>
</tr>
<tr>
<td>Matlab/Simulink interfacing</td>
<td>Easy integration of external subsystem models developed in Matlab/Simulink.</td>
</tr>
<tr>
<td>Graphical user interface</td>
<td>User-friendly graphical interface in order to easily supply vehicle input data and easily define dynamic manoeuvres.</td>
</tr>
<tr>
<td>Post-processing</td>
<td>Easy customization of output channels. Compatibility of output formats with Matlab.</td>
</tr>
<tr>
<td>Simulation time</td>
<td>Excellent performances for SiL applications. Real time capabilities for HiL applications.</td>
</tr>
</tbody>
</table>

In the following paragraph, an example of vehicle dynamics simulation tool is provided.

### 4.2.2 ASM (dSPACE)

The Automotive Simulation Models (ASM) is a tool suite offering dedicated packages for the simulation of driver assistance systems [14]. The packages are open Simulink® models and are especially designed as plant models for PC offline (MIL, SIL) and real-time simulation (HIL). For the latter use case the models support real-time code generation via MathWorks’ Simulink® Coder™ and dSPACE’s Real Time Interface.

The ASM Vehicle Dynamics Simulation Package is a comprehensive Simulink model for the vehicle dynamics simulation in all phases of the model-based development process. All the **Simulink blocks in the model are visible**, so it is easy to add or replace components with custom models and to adapt the vehicle’s properties perfectly to individual needs. ASM’s standardized interfaces allow the vehicle dynamics model to be expanded to meet specific requirements or even create a virtual vehicle. Roads and driving manoeuvres can be easily created using graphical tools with preview and clear visualization. The actual physical vehicle characteristics are represented by a **multi-body system with 24 degrees of freedom**. It consists of a drivetrain with elastic shafts, a table-based engine, two semi-empirical tire models, a nonlinear or table-based vehicle multi-body system with geometrical suspension kinematics and aerodynamics, and a steering model. An environment with a road, manoeuvres, and an open- and closed-loop driver is included as well. All parameters can be altered during run time. The included brake hydraulics model consists of a dual-circuit hydraulics system.
The vehicle multi-body system is modelled as a **nonlinear system with geometrical or table-based suspension kinematics and table-based compliances**. It supports the simulation of vertical, longitudinal, and lateral dynamics. The kinematic behaviours of common suspension types are implemented as precise analytical equations which are solved during each simulation step. User-definable geometrical linkage points connect the suspension with the wheel carrier and the chassis. There is no pre-processing required, so the linkage points can be changed during PC offline and HIL simulation. In addition, the vehicle model includes two tire models based on the published model descriptions Magic Formula and TMEasy, which are both fully implemented.

The ASM vehicle dynamics models provide an excellent basis for developing and testing vehicle dynamics ECUs, such as ESP, steering and active damping. They are ideal for vehicle dynamics investigations in early development phases. Models for passenger vehicles, trucks and trailers are available and they can be extended by other model packages or custom models as shown in Figure 17.

![Image of the Automotive Simulation Models (ASM) and packages](image)

**Figure 17 - Automotive Simulation Models (ASM) and packages for the development of driver assistance systems**
4.2.3 PELOPS (IKA)

PELOPS consists of three modules, as already described in chapter 4.1.5. The vehicle model is able to receive information from the environmental model for the current area around the related vehicle and the current vehicle status to calculate a resulting force on the vehicle body. Also the yaw rate is calculated by PELOPS and can be used by the environmental model to update the vehicle position in each calculation time step. However the lateral and longitudinal dynamic of different vehicles can be simulated.

In this module the vehicle dynamic characteristics are calculated based on the actuating variables, such as pedal position, steering wheel angle and gear selection. In addition, environmental data that influences the motion (gradient, inclination, etc.) are also taken into account.

However, the focus of PELOPS is not on the vehicle dynamics and therefore uses only simple dynamic models. In fact, the focus is on the interaction of the three mentioned models (driver, vehicle, and environment) in order to realise a realistic traffic simulation.

A detailed description of PELOPS will be released in D311 "Standard driver model definition".

4.3 Perception and fusion development tool

4.3.1 Main tool specifications

In the framework of model-based development methods for the efficient development of embedded systems, perception and fusion development tools represent a main building block. They support the vehicle application designer in easily creating new driver assistance and active safety functionalities with a multitude of ready-to-use modules and examples for optimized software components.

In the table below the required main features of perception and fusion development tools are summarised.

Table 3 – Specifications of perception and fusion development tool

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large sensor models library</td>
<td>Large library of configurable sensor models like cameras, lidars, radars, ultrasonic sensors, GPS, etc.</td>
</tr>
<tr>
<td>Multiple interfaces</td>
<td>Availability of several interfaces: CAN, LIN, USB, Ethernet, Firewire, Analog, Digital, etc.</td>
</tr>
<tr>
<td>Real-time capability</td>
<td>Real-time data recording, streaming and playback</td>
</tr>
<tr>
<td>Asynchronous data acquisition</td>
<td>To capture asynchronous data from different sensor sources</td>
</tr>
<tr>
<td>Online and offline processing</td>
<td>Real-time data playback, data handling, processing and visualization in the lab as well as online in the car</td>
</tr>
</tbody>
</table>
### Features

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab/Simulink interfacing</td>
<td>Easy integration of external modules developed in Matlab/Simulink.</td>
</tr>
<tr>
<td>Sensors and environment simulation tools</td>
<td>Easy integration of external information provided by environment and sensors simulators</td>
</tr>
<tr>
<td>Integration of external code</td>
<td>Component development e.g. in C/C++</td>
</tr>
<tr>
<td>Graphical user interface</td>
<td>User-friendly graphical interface for configuration and control. Signal data flow between software components is defined by drag and drop.</td>
</tr>
<tr>
<td>Operating System</td>
<td>Windows, Linux</td>
</tr>
</tbody>
</table>

In the next paragraphs two tools, ADTF and RTMaps are shortly described.

#### 4.3.2 **ADTF (Elektrobit)**

This chapter describes the Driver Assistance Application and Safety System Development with the tool developed by Audi AEV and Elektrobit. The EB Assist Automotive Data and Time-Triggered Framework (ADTF) support the software developer in creating new functionalities with an extensive software development kit for driver assistance solution.

#### 4.3.2.1 **The Automotive Data and Time-Triggered Framework (ADTF)**

Elektrobit (EB) Assist ADTF, the Automotive Data and Time-Triggered Framework, is a flexible tool for the development of new functions in the car. The modular system provides together with standard components a solid basis with open interfaces. With the platform independent software development kit new functions can be efficiently implemented. To protect the intellectual property the ADTF framework additionally offers the possibility to exchange software components also in binary form.

EB Assist ADTF is able to capture asynchronous data from different sensor sources and provides standard components for data recording and interpretation of LIN, MOST, CAN and FlexRay bus systems. Besides data recording, the framework offers tools for real-time data playback, data handling, processing and visualization in the lab as well as online in the car. To support data exchange with proprietary tools, a so called “Streaming Library” is available.

EB Assist ADTF simplifies the development process, especially the cooperation between OEMs and suppliers. The initial development of the framework has been driven by a major German OEM. Currently the product is in use by several renowned OEMs and Tier 1 suppliers.
4.3.2.2 EB Assist ADTF key features

The key performance indicators of EB Assist ADTF can be summarized as follows:

- Easy exchange of data and components
- Flexible and extendable set of modules
- Live visualization of data and results
- Comfortable GUI for configuration and control
- Real-time data recording, streaming and playback
- Modular programming concept with straightforward interfaces (i.e. I/O pins with cascadable filter programs)

4.3.2.3 How EB Assist ADTF works

The infrastructure of EB Assist ADTF provides the basis for the software development cycle for driver assistance functions and supports the engineer in the software testing and verification process. The framework connects a development environment with an interactive work environment. Without writing a single line of code developers are able to create new configurations by using the graphical user interface (Figure 18) and existing modules. The signal data flow between software components is defined by drag and drop and can be executed immediately, so that the effects are instantly visible. The provided examples, libraries and tool boxes facilitate the development of new and complex driver assistance and active safety software modules which can easily be integrated into the framework.

![Figure 18 - EB Assist ADTF screenshot of GUI for a typical application with video camera](image)
EB Assist ADTF describes a binary standard. Functional interfaces and data formats are open for developers. EB Assist ADTF is available for Microsoft Windows and Linux operating systems.

### 4.3.3 **GOLD (University of Parma)**

GOLD is the main ADAS development framework used by University of Parma (Figure 19). GOLD is able to provide the programmer with all the tools for a fast prototyping of applications in the automotive field. It allows to acquire images from multiple cameras (analog, digital, many different video formats) and data from many different sources (radar, laser scanner, CAN data,...); all the acquired data are timestamped and stored on disk for an efficient playback in laboratory. Accelerated graphical boards allow to speed up the processing and the rendering of the results.

Applications are developed as plug-ins and may be detached from the GOLD framework, once the algorithm is finally tested and freezed.

![GOLD Console Screenshot](image)

**Figure 19 - Screenshots of the main GOLD console**

While GOLD was born as a development tool it can be also used as a real-time engine for the developed systems also enabling a remote control or data inspect if needed.

The GOLD software has been originally developed for the Linux Operating System but is now also available for other operating systems as well.

### 4.3.4 **RTMaps (Intempora)**

This section focuses on RTMaps from Intempora SA, ultimate technology for real time multisensor applications (Figure 20, Figure 21).
Whereas most development software are not targeted at the specific multi-sensor implementation obstacles, RTMaps technology has been originally designed to focus on daily development constraints of such challenging applications and their stakes. Its component-based development process releases the engineer or the researcher from tedious tasks such as data acquisition, synchronization, recording, playback, visualization and so on. RTMaps also leads to robust, optimized and evolutive applications of the best performance/cost ratio.

Figure 20 - Global schema: inputs, processing and outputs

Figure 21 - Local map construction with lane detection and vehicles detection, based on lidar and dense stereovision (courtesy of IFSTTAR)
The main **RTMaps Software Features** can be summarized as follows:

- Asynchronous data acquisition & processing
- Supports any kind and number of sensors and actuators (including many camera standards, CAN bus and DBC files parsing, GPS, IMUs, lidars, radars, etc.)
- Precise timestamping
- Graphical interface (RTMaps Studio)
- « Block of components » functions building (hierarchical designs)
- Modular and multithreaded architecture
- Performance monitoring and customizable alarm generation
- Component development in C/C++ with the RTMaps SDK
- Master/slave mode with shared reference clock for distributed sensors and actuators
- Functionalities for advanced versatile data loggers design
- Recording and real-time playback of sensors data
- Interfaces with simulators
- Easy and royalty-free deployment of applications with the RTMaps Runtime engine.
- Customizable runtime graphical interfaces using higher level languages (QML, C#, Java, XAML, Qt, C/C++...)

Widely used in industries, research labs and universities, RTMaps Technology acquires data asynchronously i.e. "on the fly", each data sample being captured at its own genuine pace. Precise time stamps are assigned to each data sample. RTMaps modular component-based architecture ensures easy application maintenance and upgradability and allows productive re-use of previous developments. The exchange of software components and recorded datasets also makes teams cooperate easily. RTMaps version 4 provides a lot of significant improvements such as a completely redesigned Studio (more intuitive to be more efficient), a bunch of new components and diagnostics tools and monitors, improved application deployment procedures, support of hierarchical representation of components, better internationalization with UTF-8 support, and an even more modular architecture.

### 4.4 Control algorithms development tool

#### 4.4.1 Main tool specifications

The tool used for control algorithms development has to be so much faster than traditional languages. The user can tune the implemented algorithm according to the requirements of the project and simulate it in the context of a larger system model. The tool allows exploring design alternatives to meet the requirements of limited memory and hardware footprint. Once the algorithm is functionally correct, the user can optimize it in terms of performance and maintainability. Integrated tools know how to identify potential problems and
recommend appropriate changes. Starting from the developed model, it is possible to automatically generate C code and download it on the hardware platform. The generated source code can be used for real-time and non real-time applications, including simulation acceleration, rapid prototyping, and hardware-in-the-loop testing. The user can tune and monitor the generated code or run and interact with the code outside the development environment.

Furthermore this kind of tool allows to non-intrusively find operating points and compute exact linearization of the implemented models at various operating conditions. It also provides tools for computing simulation-based frequency responses without modifying your model. A graphical user interface (GUI) lets design and analyze arbitrary control structures modelled by the user, such as cascaded, prefilter, regulation, and multiloop architectures.

Summarizing, the tools for the design of control operations support every stage of the development process, from modeling to the distribution system through automatic code generation.

In the table below the **required main features** of control algorithms development tools are summarised.

**Table 4 – Specifications of control algorithms development tool**

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large library of filters</td>
<td>Availability of a great variety of filters to simulate the behaviour of the system in different operating conditions</td>
</tr>
<tr>
<td>Powerful realistic solvers and models libraries</td>
<td>Availability of different kind of mathematical solvers and models already developed</td>
</tr>
<tr>
<td>Linking between requirements, model, code</td>
<td>Requirements traceability. Should be possible to find code and model blocks that implement a requirement and vice versa.</td>
</tr>
<tr>
<td>Code generation</td>
<td>C and C++ code for use on embedded processors, on-target rapid prototyping boards, and microprocessors used in mass production</td>
</tr>
<tr>
<td>Integration of external code</td>
<td>Component development e.g. in C/C++</td>
</tr>
<tr>
<td>Verification and validation at model and code level</td>
<td>Possibility to detect statics and dynamics errors, perform coverage analysis and other check in order improve robustness and quality of model and code</td>
</tr>
<tr>
<td>External connectivity</td>
<td>Possibility to interface the development environment with other modules</td>
</tr>
<tr>
<td>Graphical user interface</td>
<td>Availability of a graphical interface to enter input, check the performance of the system and read the output</td>
</tr>
</tbody>
</table>

In the next paragraphs Matlab and Simulink/Stateflow are shortly described.
4.4.2 **Matlab/Simulink/Stateflow (MathWorks)**

MATLAB is a high-level programming language and interactive environment for technical computing, and includes functions for algorithm development, data analysis, numeric computation, and visualization.

Stateflow® is an environment for modeling and simulating combinatorial and sequential decision logic based on state machines and flow charts. Stateflow lets you combine graphical and tabular representations, including state transition diagrams, flow charts, state transition tables, and truth tables, to model how your system reacts to events, time-based conditions, and external input signals.

Simulink® is a block diagram environment for multi-domain simulation and Model-Based Design (Figure 22). It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems.

Simulink provides a graphical editor, customizable block libraries, and solvers for modelling and simulating dynamic systems. It is integrated with MATLAB®, enabling to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

Simulink blocks can be developed in C/C++ (inside what they call “S-Functions”), in Matlab or Fortran.

**Figure 22 - Modelling an automatic transmission controller by Simulink**

Simulink provides various blocks toolboxes for multiple domains. For control algorithms development, the Control System Toolbox™ provides industry-standard algorithms and apps for systematically analysing, designing, and tuning linear control systems. You can specify your system as a transfer function, state-space, pole-zero-gain, or frequency-response model. Apps and functions, such as step response plot and Bode plot, let you visualize system behaviour in time domain and frequency domain. You can tune compensator parameters using automatic PID controller tuning, Bode loop shaping, root locus method, LQR/LQG design, and other interactive and automated techniques. You can validate your design by verifying rise time, overshoot, settling time, gain and phase margins, and other requirements.
Simulink is particularly powerful when it comes to code generation of control algorithms towards execution targets. The Embedded Coder® generates readable, compact, and fast C and C++ code for use on embedded processors, on-target rapid prototyping boards, and microprocessors used in mass production. Embedded Coder enables additional MATLAB Coder™ and Simulink Coder™ configuration options and advanced optimizations for fine-grain control of the generated code’s functions, files, and data. These optimizations improve code efficiency and facilitate integration with legacy code, data types, and calibration parameters used in production. You can incorporate a third-party development environment into the build process to produce an executable for turnkey deployment on your embedded system.

In the frame of the DESERVE project, Simulink will be used for both the development of control algorithms and the simulation of the vehicle dynamics when working in the offline development toolchain. At porting applications to real vehicle prototypes, it will be possible to generate code from the control algorithm models in Simulink towards the dSpace µAutoBox targets for real-time execution. In offline mode, it will be particularly necessary to ensure virtual time coherence between the various tools interfaced and working together (like for instance, Pro-SIVIC – RTMaps – Simulink) as all these tools have their own way of handling the time flow.
5. COST MODELS ON HARDWARE IMPLEMENTATIONS

Identifying the best software/hardware realization of a system can be performed by an iterative process, see Figure 23. The application designer selects algorithms and defines their parameters. The evaluation of a hardware mapping of these algorithms indicates whether the hardware meets the requirements or not and may lead to new insights or hints to the application designer that describe how the application parameters might be modified in order to allow a better hardware realization.

![Figure 23 - Design space exploration framework in the design flow for heterogeneous hardware architectures](image)

The rapid evaluation of hardware platforms in order to compare different realizations and find the best possible implementation, i.e., a design space exploration, is needed. Evaluating of hardware platforms and derivation of hardware costs is a laborious task that can take a long time. To make the design space exploration practical, cost models can be employed [15].

The following subsections specify the design space exploration framework and the quantitative cost models on which the framework is built on.

5.1 Design Space Exploration Framework

There are usually many different possible mappings of application building blocks to hardware (architecture) blocks as the hardware can be implemented using different technologies. Considering the different characteristics of these technologies, the possibility to explore the design space is needed in order to find solutions that meet the requirements. On the other hand, the evaluation of a software/hardware mapping can deliver design support at different levels in the design process.
The design space usually is spanned by a huge number of objectives, e.g., implementation cost, power consumption, latency, silicon area, flexibility, etc. (Figure 24). It is not feasible to implement or simulate a variety of different possible hardware realizations for evaluation and comparison purposes as the effort would be much too high and the simulation of complex hardware would take too long. Therefore, a cost evaluation and design space exploration based on quantitative cost models will be implemented.

The model based cost evaluation supports the system designer by allowing an exploration of the prevailing design space without the need for deep knowledge of the hardware technologies. Thus, the system can be analysed in early design stages and the feature estimates provided by the cost models can be used to drive the design process.

The design space exploration is composed of the following steps:

1. To evaluate the costs of a system, it has to be partitioned into hardware building blocks that can be described independently by specific parameters. Examples for such building blocks include image filters with the number of filter coefficients as a parameter.
2. After specifying the building blocks, a mapping of these blocks to hardware technologies is performed. This determines the cost models to use for evaluating the characteristics of the single blocks.
3. A specification of the system, i.e., the required characteristics, has to be fixed. These characteristics could be, for example, maximal power consumption, maximal silicon area, minimal throughput, etc.
4. The cost models are evaluated to compute the estimated characteristics of the hardware building blocks.
5. A report describing the evaluated system is generated. It contains the estimated cost values and indicates, if the system can meet the requirements. From this report the designers can derive design support for different levels of the design process.

**Figure 24 - Objectives in the design space**
5.2 Cost Models

The design space exploration framework is built on quantitative cost models that describe different possible hardware implementations of application building blocks. The cost models are collected in a model library. For each application building block, the model library contains a family of cost functions that describe the realization of that building block on a specific hardware technology. The cost functions in a family describe different characteristics of the specific hardware realization of the building block. These characteristics include performance (latency), power consumption, and silicon area. However, the model library is flexible and allows the integration of cost functions for additional characteristics, e.g., implementation cost, flexibility, throughput rate, etc. The evaluation tool will use all the available cost functions for a basic block.

The cost models, or the cost functions, can be specified in different ways. The system will at least operate on the following types:

- A cost function can be specified as a set of data points, for example derived from data sheets or single measurements. These data sets are used as lookup tables and can provide precise results for a specific combination of input values. Potentially, the system might be able to interpolate the values in the data set to allow evaluation of parameter combinations that were not examined previously.
- A cost function can be given as an analytical function that approximates the hardware costs and may be derived from different implementations of a hardware block.
6. CONCLUSIONS

The present report describes the methodology on how to use the DESERVE platform in developing ADAS applications.

This report is strictly linked with D213 “Development method (first release)” deliverable, associated with work package 2.1. D132 report introduces the methodology from a general point of view defining the specifications of the development tools to be used in the DESERVE platform and summarising the main features of some tools, available on the market. D213 is focused on the specific tools associated with the DESERVE Development Platform and with the rapid prototyping phase, and defines more detailed guidelines on the use of the DESERVE platform.

The DESERVE development process has to adapt the actual V-model cycle in order to:

- provide a common environment for design, development and testing of ADAS functions;
- provide a common environment for coexistence of ADAS functions;
- allow reuse of pre-validated software components.

The report describes the main phases of the development process of DAS applications:

- Concept (use cases and requirements definition, coexistence between ADAS functionalities)
- Implementation (configuration of environment and sensors simulators, architectures definition, development of perception / fusion and control algorithms,)
- Verification & Validation (Model In the Loop, Software in the Loop and Hardware in the Loop)
- Integration on vehicle

The report defines the specifications of the development tools and provides a short description of specific tools to be used in the DESERVE platform:

- Environment and sensor simulation
- Vehicle dynamics simulation
- Perception and fusion development
- Control algorithms development

The last chapter describes cost models on hardware implementations based on standard components, FPGA, and System On Chip.
REFERENCES

[1] www.deserve-project.eu
[2] D121 Development Platform Requirements, DESERVE
[3] D213 Development method (first release), DESERVE
[14] M. Peperhowe - dSPACE, ADAS in Real-Time with dSPACE ASM, Stuttgart, Testing Expo, 05.06.2013