Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2)

Report type: Deliverable D6.1.2
Report name: Case Study

Dissemination level: PU (Public)
Status: Final
Version number: 1.0
Date of preparation: 2010-04-29
### Authors

<table>
<thead>
<tr>
<th>Editor</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friedhelm Stappert</td>
<td><a href="mailto:friedhelm.stappert@continental-corporation.com">friedhelm.stappert@continental-corporation.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andreas Abele</td>
<td><a href="mailto:andreas.abele@continental-corporation.com">andreas.abele@continental-corporation.com</a></td>
</tr>
<tr>
<td>Frank Hagl</td>
<td><a href="mailto:frank.hagl@continental-corporation.com">frank.hagl@continental-corporation.com</a></td>
</tr>
<tr>
<td>Carl-Johan Sjöstedt</td>
<td><a href="mailto:sjostedt@kth.se">sjostedt@kth.se</a></td>
</tr>
<tr>
<td>Henrik Lönn</td>
<td><a href="mailto:Henrik.Lonn@volvo.com">Henrik.Lonn@volvo.com</a></td>
</tr>
<tr>
<td>Anders Sandberg</td>
<td><a href="mailto:Anders.sandberg@mecel.se">Anders.sandberg@mecel.se</a></td>
</tr>
</tbody>
</table>

### The Consortium

<table>
<thead>
<tr>
<th>Volvo Technology Corporation (S)</th>
<th>VW/Carmeq (D)</th>
<th>Centro Ricerche Fiat (I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continental Automotive (D)</td>
<td>Delphi/Mecel (S)</td>
<td></td>
</tr>
<tr>
<td>Mentor Graphics Hungary (H)</td>
<td>CEA LIST (F)</td>
<td></td>
</tr>
<tr>
<td>Kungliga Tekniska Högskolan (S)</td>
<td>Technische Universität Berlin (D)</td>
<td>University of Hull (GB)</td>
</tr>
</tbody>
</table>
## Revision chart and history log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>3403</td>
<td>07.04.2010</td>
<td>initial version</td>
</tr>
<tr>
<td>3414</td>
<td>08.04.2010</td>
<td>adapting structure of sections</td>
</tr>
<tr>
<td>3444</td>
<td>15.04.2010</td>
<td>Introduction extended, Sects 2 and 3 revised, Sect 8 added.</td>
</tr>
<tr>
<td>3513</td>
<td>21.04.2010</td>
<td>Added content to Sect 4</td>
</tr>
<tr>
<td>3559</td>
<td>23.04.2010</td>
<td>update List of abbreviations, update Sects. 2 and 8</td>
</tr>
<tr>
<td></td>
<td>23.04.2010</td>
<td>Security System Update (SAE-AADL remodelling)</td>
</tr>
<tr>
<td>26.04.2010</td>
<td>update table 1, ref. [9], Sect. 4; Increased size of figures in sect. 7</td>
<td></td>
</tr>
<tr>
<td>29.04.2010</td>
<td>Fig. 4 repaired, increased size of figures in Sect. 5, use case descriptions in Sect. 3 enhanced, document finalized.</td>
<td></td>
</tr>
</tbody>
</table>
# List of abbreviations

Table of terms and abbreviations used in this document

<table>
<thead>
<tr>
<th>Term / Abbreviation</th>
<th>Denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>Adaptive Cruise Control</td>
</tr>
<tr>
<td>BCU</td>
<td>Body Control unit</td>
</tr>
<tr>
<td>CC</td>
<td>Cruise Control</td>
</tr>
<tr>
<td>CDD</td>
<td>Complex Device Driver</td>
</tr>
<tr>
<td>EMS</td>
<td>Engine Management System</td>
</tr>
<tr>
<td>ESP</td>
<td>Electronic Stability Program</td>
</tr>
<tr>
<td>FAA</td>
<td>Functional Analysis Architecture</td>
</tr>
<tr>
<td>FC</td>
<td>Functional Constraint</td>
</tr>
<tr>
<td>FDA</td>
<td>Functional Design Architecture</td>
</tr>
<tr>
<td>FMEA</td>
<td>Failure Mode and Effects Analysis</td>
</tr>
<tr>
<td>HVAC</td>
<td>Heating, Ventilation, Air Condition</td>
</tr>
<tr>
<td>IIC</td>
<td>Instrument Cluster</td>
</tr>
<tr>
<td>SWC</td>
<td>Software Component</td>
</tr>
<tr>
<td>TADL</td>
<td>Timing Augmented Description Language</td>
</tr>
<tr>
<td>TBD</td>
<td>To be defined</td>
</tr>
<tr>
<td>UC</td>
<td>Use Case</td>
</tr>
<tr>
<td>VFB</td>
<td>Virtual Functional Bus</td>
</tr>
</tbody>
</table>
Table of contents

Authors ........................................................................................................................................................................... 2
Revision chart and history log ........................................................................................................................................... 3
List of abbreviations ......................................................................................................................................................... 4
Table of contents ............................................................................................................................................................. 5
List of Figures .................................................................................................................................................................. 7
1 Introduction .................................................................................................................................................................... 9
2 The HAVEit functional architecture .......................................................................................................................... 11
   2.1 Introduction ............................................................................................................................................................ 11
   2.2 HAVEit functional structure overview .................................................................................................................. 12
   2.3 Modeling the HAVEit Command Layer on Vehicle Level .................................................................................. 12
      2.3.1 Feature Model HAVEit Demonstrators ......................................................................................................... 12
      2.3.2 Vehicle Features of the Command Layer .................................................................................................. 13
      2.3.3 HAVEit WP4300 vehicle features .................................................................................................................. 15
   2.4 Modeling the HAVEit Command Layer on Analysis Level .................................................................................. 20
      2.4.1 EnvironmentDetection ................................................................................................................................... 22
      2.4.2 DriverInterface ............................................................................................................................................... 23
      2.4.3 CommandLayer ............................................................................................................................................. 24
      2.4.4 ExecutionLayer .............................................................................................................................................. 25
3 Cruise Control System .................................................................................................................................................... 26
   3.1 Hardware topology ................................................................................................................................................... 26
   3.2 Mapping of functionalities / blocks to hardware ................................................................................................... 26
   3.3 Main Functionalities ............................................................................................................................................... 27
   3.4 System Overview Cruise Control ......................................................................................................................... 27
   3.5 General System Requirements ................................................................................................................................ 28
      3.5.1 Basic Preconditions / boundary conditions .................................................................................................. 28
   3.6 Cruise Control Functionality HMI .......................................................................................................................... 28
   3.7 Cruise Control Use Cases ....................................................................................................................................... 28
   3.8 Main Functional Blocks and Signal Flow .............................................................................................................. 31
   3.9 Modelling Cruise Control using EAST-ADL ....................................................................................................... 31
4 Modelling of Cooperative Systems .................................................................................................................................... 32
5 Brake System .................................................................................................................................................................... 34
   5.1 Introduction ............................................................................................................................................................ 34
   5.2 Brake System Use Cases ....................................................................................................................................... 34
   5.3 Brake System Architecture .................................................................................................................................... 34
5.4  System Model in Papyrus ................................................................. 40
5.5  AUTOSAR model ........................................................................ 47
5.6  Safety Assessment ...................................................................... 48
6  Steering System ............................................................................ 50
  6.1  Use case diagram ..................................................................... 51
  6.2  Simulink model ........................................................................ 52
  6.3  EAST-ADL model ...................................................................... 52
7  Security System ............................................................................ 55
  7.1  Security System Scenario .......................................................... 55
  7.2  Responsibilities of Security System .......................................... 56
  7.3  Concepts for modelling on Design and Implementation level .......... 56
  7.4  UML profile extensions based on AUTOSAR semantic .......... 58
  7.5  SAE-AADL overview .................................................................. 59
  7.6  SAE-AADL in Automotive process model ........................................ 61
  7.7  SAE-AADL and logical Design/Requirements Engineering .......... 62
  7.8  SAE-AADL in System Design ..................................................... 63
  7.9  SAE-AADL in Software Architecture design ................................ 68
  7.10 SAE-AADL in Software Component Design ................................. 70
  7.11 SAE-AADL and Code development ............................................. 74
  7.12 SAE-AADL integration in UML/AUTOSAR modelling approaches .................. 74
  7.13 SAE-AADL and EAST-ADL ....................................................... 76
  7.14 Conclusions and further work .................................................... 77
8  Safety analysis ............................................................................. 78
  8.1  Requirements on Vehicle level ..................................................... 78
  8.2  Driving situations ..................................................................... 79
  8.3  Preliminary Hazard analysis and ASIL assignment .......................... 80
  8.4  Safety Analysis with HipHops and EPM ....................................... 85
9  Input from previous ATESST2 work ................................................... 86
10 Contribution to overall ATESST2 objectives ....................................... 87
11 Conclusions .................................................................................. 88
12 References .................................................................................... 89
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EAST-ADL topics covered by the case studies</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>EAST-ADL abstraction levels covered by the case studies</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Simplified view: Separation into Command layer and Execution layer</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>HAVEit Functional Structure</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Overview FAA with EnvironmentDetection, DriverInterface, CommandLayer, and ExecutionLayer</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Insight of EnvironmentDetection figured in Figure 5</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Insight of DriverInterface figured in Figure 5</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>Insight of CommandLayer figured in Figure 5</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>Insight of ExecutionLayer figured in Figure 5</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>Hardware Topology</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>Use Case Diagram</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>Cruise Control Block Diagram</td>
<td>31</td>
</tr>
<tr>
<td>13</td>
<td>Cooperative function modelling style</td>
<td>32</td>
</tr>
<tr>
<td>14</td>
<td>System overview of a Cooperative function where two vehicles are modelled</td>
<td>33</td>
</tr>
<tr>
<td>15</td>
<td>Hardware Architecture of the brake system including the transfer functions of the sensors and actuators</td>
<td>35</td>
</tr>
<tr>
<td>16</td>
<td>Hardware Setup</td>
<td>35</td>
</tr>
<tr>
<td>17</td>
<td>Design architecture of the validator, including middleware abstraction, hardware architecture and environmental model</td>
<td>36</td>
</tr>
<tr>
<td>18</td>
<td>Software implementation example</td>
<td>38</td>
</tr>
<tr>
<td>19</td>
<td>The mapping of functional entities to the HW platform. Note that the system is only realized for one wheel</td>
<td>39</td>
</tr>
<tr>
<td>20</td>
<td>A possible SW to HW mapping for the Validator</td>
<td>40</td>
</tr>
<tr>
<td>21</td>
<td>System Model with ports and connectors hidden</td>
<td>41</td>
</tr>
<tr>
<td>22</td>
<td>Analysis Architecture</td>
<td>42</td>
</tr>
<tr>
<td>23</td>
<td>Design Architecture</td>
<td>42</td>
</tr>
<tr>
<td>24</td>
<td>Hardware Design Architecture, physical aspects</td>
<td>43</td>
</tr>
<tr>
<td>25</td>
<td>Hardware Design Architecture, logical aspects (the transfer functions)</td>
<td>43</td>
</tr>
<tr>
<td>26</td>
<td>Functional Design Architecture</td>
<td>44</td>
</tr>
<tr>
<td>27</td>
<td>Allocation Constraints</td>
<td>45</td>
</tr>
<tr>
<td>28</td>
<td>Realizations defined in the context of the current system model</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>Realizations that are defined in the context of each type</td>
<td>46</td>
</tr>
<tr>
<td>30</td>
<td>A subset of the requirements</td>
<td>46</td>
</tr>
<tr>
<td>31</td>
<td>The implementation architecture</td>
<td>47</td>
</tr>
<tr>
<td>32</td>
<td>The AUTOSAR SW components represented in Vector DaVinci</td>
<td>47</td>
</tr>
<tr>
<td>33</td>
<td>Indication of representation of Hazard information and Safety Goal for Service Brake</td>
<td>48</td>
</tr>
</tbody>
</table>
Figure 34. Indication of representation of Functional Safety Concept for Item Service Brake
Figure 35. Indication of formalization of safety requirement for Item Service Brake
Figure 36: Example power-assisted steering
Figure 37: Use case diagram of the Steering System
Figure 38: The original Simulink model of the EPAS System to the left, on the right hand side the same models divided in subsystems, by component
Figure 39: WIP Papyrus model of FAA and Environment model
Figure 40: The EPAS modeled in Modelica (above) and EAST-ADL, using FunctionPowerPorts. In a real case, this model should perhaps consist of four subsystems: One of the steering column, one of the assist motor, and the pinion.
Figure 41: Security System Scenario
Figure 42: Security System Overview
Figure 43: UML model of internal structure of the security system
Figure 44: AADL usage within an Automotive process framework
Figure 45: state automata armed Control in UML
Figure 46: Subsystems of Security System test system
Figure 47: generic HW and SW of security system
Figure 48: Security System HW with one deployment target
Figure 49: Security System HW with two deployment targets
Figure 50: Security system and collaborating systems defined as SAD-AADL Thread Groups
Figure 51: Security System with mode managers
Figure 52: Transitions and modes of the security system
Figure 53: Components of Security System given as threads
Figure 54: The platform accesses of the components of the security system
Figure 55: Refinement closely coupled threads
Figure 56: Refinement loosely coupled threads
Figure 57: State Automata aggregated by a thread
Figure 58: Driving situations considered in the risk analysis
1 Introduction

This deliverable contains the description of the demonstrators and case studies developed in WP6. Having one single case study covering all aspects of the EAST-ADL would not be feasible in practice. Therefore, different case studies have been developed by different partners, covering different topics and different abstraction levels of the EAST-ADL, and serving as assessment and exemplary application of the results of the other WPs. The applications include: cruise control, a brake system, a steering system, and a car-access security system. There was a close cooperation with the HAVEit project [4], which deals with highly automated driving technologies. The vehicle architecture defined by HAVEit has been modelled by means of the EAST-ADL, and the cruise control model has been integrated in this architecture. A safety analysis was performed for the cruise control system, based on the ISO DIS 26262 [8], and applying the methods and tools developed in the project. An example on modelling style supporting Cooperative system is added. The topics addressed by the individual case studies are depicted in Figure 1.

<table>
<thead>
<tr>
<th></th>
<th>Cruise Control System</th>
<th>Brake System</th>
<th>Steering System</th>
<th>Security System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behaviour</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cooperative Systems</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment Modelling</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Feature Modelling</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Methodology</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Variability</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>System&amp;SW design</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

*Figure 1: EAST-ADL topics covered by the case studies*

Figure 2 shows how the EAST-ADL abstraction levels are covered by the different case studies.
This document is structured as follows. First, Sect. 2 describes the overall functional architecture as developed in the HAVEit project, and how this is modeled by means of the EAST-ADL. In Sect. 3 the cruise control case study is described, and Sect. 4 explains how to model a cooperative system, using as an example the cruise control extended to platoon-driving functionality. The case studies brake system, steering system, and security system are described in Sections 5, 6, and 7 respectively. The results of the safety analysis are laid down in Section 8. Finally, Sections 9 to 11 provide an overview on how the case studies are embedded in the overall ATESST2 project, and conclusions are given.
2 The HAVEit functional architecture

2.1 Introduction

This chapter describes the overall vehicle architecture applied in the demonstrator. From technical viewpoint it follows the architecture worked out in the HAVEit project [4].

Principals:

- The Command Layer coordinates the requested Vehicle Motion. The requested vehicle motion is based on Driver's input, detected environment and driving situation
- The Execution Layer realizes the requested vehicle motion within the possible boundary conditions by commands towards actuators

Note that, within both layers several dedicated coordination takes place.

Example for Command layer: Coordination between driver's input and co-driver's input based on detected environment and recognized traffic situation.

Example for Execution layer: Brake blending in case of regenerative braking or engine braking. In case that a sufficiently large deceleration is requested, the dedicated brake actuator is selected: (Friction) service brake and/or recuperative braking in case of a hybrid electrical vehicle.

![Diagram of vehicle architecture](image)

**Figure 3: Simplified view: Separation into Command layer and Execution layer**
2.2 HAVEit functional structure overview

The HAVEit functional structure focuses on vehicle autonomous driving, driver assist functionalities and manual driving. Other aspects, such as comfort functionalities like climate control, are not considered in HAVEit.

![HAVEit Functional Structure Diagram](image)

Figure 4: HAVEit Functional Structure

2.3 Modeling the HAVEit Command Layer on Vehicle Level

The Vehicle Feature Model is separated into two main parts:

- The HAVEit demonstrators
- The features of the command layer

2.3.1 Feature Model HAVEit Demonstrators

Each Demonstrator has its own set of command layer features. The structure of the feature model enables, that other HAVEit demonstrators can also be selected and modelled, too.
The selection is only valid in case that exactly one demonstrator is selected.

### 2.3.2 Vehicle Features of the Command Layer

Possible Vehicle Features of the command layer have been worked out in this feature model. Note, that not all modelled vehicle features of the command layer are currently used.

The goal was to set up a feature model which can be extended easily.

**Structure of the Command Layer Feature Model:**
- basic vehicle features
- available automation levels

Each requested automation level requires its dedicated basic vehicle features

1. **Available_Automation_levels_Driver_only:**
   The driver is fully responsible for the vehicle movement. The driver is not supported by any electronic means

2. **Available_Automation_levels_Driver_Assisted:**
   In this level the driver is assisted by electronic means on several levels:
   a. The driver only receives some helpful feedback in case of detected critical driving situations, but no actions directly influencing the vehicle movement are taken.
   b. The driver is supported by direct action of the electronics in case of a detected critical driving situation, but no feedback is generated.
   c. The driver is supported both, by feedback and direct action in case of a detected critical driving situation.
3. **Available Automation_levels Semi_automated:**
The vehicle movement in one direction, either lateral or longitudinal, is performed by electronic means. Note, the XOR relationship is important here.

4. **Available Automation_levels highly_automated:**
The vehicle movement in both directions, lateral and longitudinal, are performed by electronic means.

The basic vehicle features are structured as follows:

- **AutonomousVehicleMovementControl:** In this branch all the features related to vehicle movement control are collected, where the electronics is controlling the vehicle movement without further driver's input after activation.

- **InterventionOnVehicleMovement:** In this branch all the features related to vehicle movement control are collected, where the driver controls the vehicle movement within certain boundary conditions, but in case of reaching the boundary conditions the driver's commands are ignored or limited.

- **AssistanceOnVehicleMovement:** In this branch all the features are collected, where the driver receives a feedback in case of detected critical driving situations, but the electronics does not perform any further actions, like intervention of movement control.

- **DriverMonitoring:** In this branch all the features are collected, related to monitoring the driver with respect to his state and/or conditions to drive the vehicle. The driver monitoring results are used both, system internal, as well as for driver's feedback generation.

The relations between the selectable automation levels and basic vehicle features are:
That means for example, in case that `Available_Automation_levels_Semi_automated_long_only` is selected, `longitudinalControl` must be selected, too.

Otherwise the selection is invalid.

### 2.3.3 HAVEit WP4300 vehicle features

In the following sections only the HAVEit demonstrator WP4300 is considered in more detail, and a short description of the selected vehicle features is given.

Variant decision for HAVEit WP4300:
2.3.3.1 lane_keeping

- **VehicleFeaturesCommandLayer**
  - **Autonomous/vehicleMovementControl**
    - **lateralControl**
      - **lane_keeping**

The lateral vehicle movement control is performed. **lane_keeping** requests, that the vehicle shall be able to keep the once selected lane autonomously.

2.3.3.2 CC_Basic

- **VehicleFeaturesCommandLayer**
  - **Autonomous/vehicleMovementControl**
    - **longitudinalControl**
      - **CC_Basic**

The longitudinal vehicle movement control is performed. **CC_Basic** requests, that the vehicle shall be able to run with a given constant speed.
2.3.3.3  CC_SpeedSetPoint_CurrentVehicleSpeed

The feature CC_SpeedSetPoint_CurrentVehicleSpeed requests, that the current vehicle speed shall be used as target speed setpoint for the feature CC_Basic.

2.3.3.4  CC_LowDecelleration

The feature CC_Basic requests to run the vehicle with a constant speed. In most vehicle operation conditions this is possible by regulating the drive torque. In case of a loaded vehicle running downhill it might be necessary to provide negative drive torque.

The feature CC_LowDecelleration requests, that only a very limited negative drive torque can be requested by CC_Basic.

The other alternative (CC_SmartDecelleration) would be, that CC_Basic is able perform a service brake request.

2.3.3.5  CC_DistanceControlled

The longitudinal vehicle movement control is performed.

The feature CC_DistanceControlled requests, that the longitudinal vehicle movement is performed in such a way, that the distance to an object in front is controlled.

2.3.3.6  ACC_StopAndGo
The feature ACC_StopAndGo requests how CC_DistanceControlled is performed. In case, that an CC_DistanceControlled-relevant object forces an vehicle stop, then the feature ACC_StopAndGo requests that the vehicle is able to start to move/drive off autonomously.

2.3.3.7 ACC_whileDriving_FollowObjectInFront

The feature ACC_whileDriving_FollowObjectInFront requests, that the detected object ahead is regarded as leading vehicle, relevant for CC_DistanceControlled. Especially for several lanes in the same direction, the objects/vehicles on the right or left lane are ignored in case of a detected object/vehicle in the same lane ahead.

Note, that this (simple) behaviour might lead to an unwanted right overtaking. The feature ACC_whileDriving_AvoidRightOvertaking requests to avoid this behaviour.

2.3.3.8 ACC_FullDecelleration

The feature ACC_FullDecelleration requests, that CC_DistanceControlled is able to request a full range deceleration.

2.3.3.9 DriverInattentiveDetectionAndFeedback

The feature DriverInattentiveDetectionAndFeedback requests, that the driver is observed with respect to inattentiveness. In case of a detected inattentiveness a feedback to the driver is performed.

2.3.3.10 DriverDrowsinessDetectionAndFeedback

© 2010 The ATEST2 Consortium
The feature **DriverDrowsinessDetectionAndFeedback** requests, that the driver is observed with respect to drowsiness. In case of a detected drowsiness a feedback to the driver is performed.

### 2.3.3.11 DriverMisuseDetectionAndFeedback

- ✔ VehicleFeaturesCommandLayer
- ✔ DriverMonitoring
- ✗ DriverMisuseDetectionAndFeedback

The feature **DriverMisuseDetectionAndFeedback** requests, that the driver is observed with respect to any misuse of the HAVEit system. In case of a detected misuse a feedback to the driver is performed.

Example for misuse: The driver reads a newspaper while the vehicle is running in any high automation level.

### 2.3.3.12 Available_Automation_levels

- ✔ VehicleFeaturesCommandLayer
- ✔ Available_Automation_levels

The feature **Available_Automation_levels** requests, that the available automation level are selected either on drivers request or based on the detected driving and traffic situations. This includes the handling of related transition between the related automation levels.

### 2.3.3.13 Available_Automation_levels_Driver_only

- ✔ VehicleFeaturesCommandLayer
- ✔ Available_Automation_levels
- ✔ Available_Automation_levels_Driver_only

The feature **Available_Automation_levels_Driver_only** requests a level "Driver only", in which only the driver is able to control the vehicle movement. In case of an active automation level "Driver only" only the driver is able to control the vehicle movement.

### 2.4 Modeling the HAVEit Command Layer on Analysis Level

Based on the features given in section 2.3.2 a Functional Analysis Architecture (FAA) has been created.

The FAA model (see Figure 5) follows the functional layered HAVEit architecture according Figure 3. The Actuator layer is not modeled. Note, that with respect to safety this architecture is purely functional driven. Therefore it represents the bare, unsheltered functionality – no safety mechanisms are introduced.

Each Layer is realized as ADLFunctionType with its dedicated variability modeled as local feature model. This is expressed by the ⌧ on top right corner.
Figure 5: Overview FAA with EnvironmentDetection, DriverInterface, CommandLayer, and ExecutionLayer.
2.4.1 EnvironmentDetection

The EnvironmentDetection collects data from the environment. This data may include:

- C2X data, i.e. information received by means of car-to-car or car-to-infrastructure communication, e.g. information about road or traffic conditions ahead.
- Traffic signs, e.g. speed limits obtained by means of a traffic sign recognition system,
- Route information from the GPS navigation system
- Recognition of obstacles, pedestrians, or other traffic participants by means of an object detection system (e.g. radar or infrared sensors),
- Road data, especially lane information obtained from a road detection system.

All these data collection systems are optional. Depending on the chosen level of autonomous driving, more and more of the systems must be selected.
2.4.2 DriverInterface

The DriverInterface collects data from and delivers data to the driver. The Driver's input is mandatory. It reads the driver's requests by means of pedals, the steering wheel, and several switches. The feedback to the driver is optional and not modeled in detail. It may include e.g. force feedback pedals or steering wheel. Furthermore, the optional DriverMonitoring module observes the driver in order to detect e.g. drowsiness or inattentiveness.
2.4.3 CommandLayer

The Command Layer coordinates the requested vehicle motion based on Driver's input, detected environment and driving situation. It generates a requested MotionVector which is then given to the ExecutionLayer as input.

Figure 8: Insight of CommandLayer figured in Figure 5
The Execution Layer realizes the requested vehicle motion within the possible boundary conditions by commands towards actuators. It receives the MotionRequest from the CommandLayer and translates it into corresponding torque, steering, and brake requests to the PowertrainController, the BrakeSystem, and the SteeringSystem.
3 Cruise Control System

The cruise control system is based on an existing AUTOSAR demonstrator that had been developed at Continental prior to ATESST2. The existing hardware was re-used in ATESST2, and a system model according to the EAST-ADL and the HAVEit architecture was developed (see also Chapter 2). The hardware was also used for the security system case study described in Chapter 7.

3.1 Hardware topology

The system consists of 3 ECUs:

1. Body Control Unit (NEC V850) controlling the CC switches and the HVAC system
2. Instrument Cluster
3. Engine Control Unit containing the EMS and cruise control algorithm

The ECUs are connected via a high-speed CAN bus. Additional functionality (‘Rest of Vehicle’) is controlled by a PC which also connected to the CAN bus.

3.2 Mapping of functionalities / blocks to hardware

The following functionalities are independent of the hardware; in principle they can run on each available hardware.

- Switch Detection/Reading
- Algorithm and Actuator Control
- Display Speed and Cruise Control LEDs
• HVAC (air-conditioning compressor, no radio gateway), Switch Detection/Reading

3.3 Main Functionalities

This function consists of four main sub-functions:
• Cruise Control Input
• Cruise Control State machine
• Cruise Control Basic Functions
• Cruise Control Controller

Cruise Control Input:
This function acquires the raw signals of the Cruise Control input switches on driver's demand.

Cruise Control State Machine:
This function describes the determination of the transitions between the states and the calculation of the vehicle speed setpoint.

Cruise Control Basic Functions:
The function Cruise Control Basic Functions contains the basic functionality for a vehicle speed controller. It describes the calculation and the evaluation of the vehicle speed signal and the Cruise Control state diagram.

Cruise Control Controller:
This function describes the vehicle speed control algorithm. It uses the setpoint, which is calculated in Cruise Control State Machine and calculates a Torque Request.

3.4 System Overview Cruise Control

Figure 11: Use Case Diagram
3.5 General System Requirements

3.5.1 Basic Preconditions / boundary conditions

Maintain speed
Speed is maintained depending on the desired speed. Doing this the algorithm will request torque control to increase or decrease.

Transmission
The system will use automatic transmission. Only drive mode is supported. Thus no other mode can be set by the user. The prototype will have no gear shift within set up. Automatic gear shift will be done by algorithm.

Transmission should manage declutch or select neutral gear to avoid engine stalling.

3.6 Cruise Control Functionality HMI

<table>
<thead>
<tr>
<th>Cruise Control Switch mode</th>
<th>Cruise Control display light symbol</th>
<th>Cruise Control display speed value</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>off</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>set/resume</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>canceled</td>
<td>x</td>
<td>-</td>
</tr>
</tbody>
</table>

3.7 Cruise Control Use Cases

**UC: Set desired speed**
Precondition: Cruise control is turned on

The driver pushes the set speed button while the current speed is between 40 and 200 km/h. The cruise control algorithm memorizes the current speed based on the actual vehicle speed and starts to maintain this speed. The value of the maintained speed is displayed in the instrument cluster.

**UC: Cancel speed**
Precondition: Speed is being maintained.

Speed control is cancelled
- by the driver pressing the cancel button,
- by the driver pressing the brake pedal,
- when the actual speed goes above 200 km/h (below 40 km/h can't be shown in the demonstrator, thus not supported).

The system stops to maintain the speed. The desired speed is still memorized. Display of the speed value is turned off in the instrument cluster.
UC: Resume speed
Precondition: Speed control is turned on and speed was set and cancelled at least once before. Thus a desired speed is memorized.
Pressing the RESUME button after speed control has been cancelled causes the system to maintain the memorized speed, and the vehicle to accelerate or decelerate to this previously stored speed. The value of the maintained speed is displayed in the instrument cluster.

UC: Increase/decrease speed
Precondition: System is maintaining speed.
The desired vehicle speed can be increased or decreased by tapping the “+” or “-” button (limited between 40 and 200 km/h). Thus the algorithm will maintain the new desired speed. The value shown within the instrument cluster is updated.

UC: Override set speed
Precondition: System is maintaining speed.
The driver pushes the acceleration pedal in such a way that the speed increases above the desired/maintained speed. The driver is regulating the speed above the maintained desired speed (if the speed goes above 200 km/h, speed control is cancelled; see UC Cancel speed). when the driver releases the pedal, speed will be maintained by the algorithm again towards the desired speed previously set.

UC: Switch function on/off
If cruise control is turned off and driver pushes the 'on' button, the system switches to mode 'cruise control turned on'.
If cruise control is turned on and the driver pushes the 'off' button, the system switches to mode 'cruise control off'. The stored desired speed is deleted. The light symbol and displayed desired speed value in the instrument cluster is turned off.

UC: Going up hill
Precondition: System is maintaining speed.
Within the demonstrator a signal can simulate going uphill. Consequently, the algorithm will request more torque in order to maintain the desired speed. This will cause the engine to require more power causing the UC 'Disable/Release HVAC'.

UC: Disable/Release HVAC
Precondition: Engine is running
When the engine requires power, it will send a signal to the HVAC to turn off the compressor. In order to provide more power for the acceleration of the vehicle, the request of the <climate control> algorithm to the <climate compressor> is controlled by an 'enable' signal of the <torque control> algorithm. The <torque control > algorithm could deactivate the <climate compressor> by deactivating the 'enable' signal. The activation of the <climate compressor> is delayed by a re-triggerable time of 2 sec.
UC: Generate Engine torque
Precondition: Engine is running and gear engaged.
Depending on the request of the driver and the cruise control, the torque control will produce torque. This torque will be used to move the vehicle by means of the gearbox and the wheels.

UC: Engine running
If no torque is requested by the driver or the cruise control function, the torque control will keep the engine running in idle mode.

FC: driver request
Driver is not able to push the accelerator pedal and the brake pedal at the same time.
If both pedals are pushed then only the brake pedal is taken into account.

UC: Driver pushes on accelerator pedal
Precondition: Engine is running and gear engaged.
Driver pushes acceleration pedal.
The more the driver is pushing on the accelerator pedal the more acceleration is requested. Depending on the accelerator pedal position, the vehicle speed will increase (or decrease) until the torque at the wheels is equal to the needed torque to drive the car with constant speed.

UC: Driver pushes on brake pedal
Precondition: Engine is running and gear engaged.
When the driver pushes the brake pedal the vehicle speed will decrease until the brake pedal is released or the vehicle is stopped.

FC: Vehicle motion determination
Only longitudinal vehicle speed and acceleration will be calculated.

UC: Vehicle is running
Precondition: Engine is running and gear engaged.
If the driver pushes the accelerator pedal in such a way that the vehicle is running, vehicle speed will be calculated and provided to other functions.
3.8 Main Functional Blocks and Signal Flow

![Cruise Control Block Diagram]

**Figure 12: Cruise Control Block Diagram**

3.9 Modelling Cruise Control using EAST-ADL

The cruise control model on the upper abstraction levels (vehicle level and analysis level) of the EAST-ADL is integrated in the overall model of HAVEit functional architecture, and described in Chapter 2.

An AUTOSAR model, covering the implementation level of the EAST-ADL, already existed, as the cruise control case study was based on an existing AUTOSAR demonstrator. Furthermore, a model on design level was developed internally at Continental during the project. However, the design and implementation level of the cruise control model is not public.
4 Modelling of Cooperative Systems

Considerable time has been spent on defining how to model cooperative systems in EAST-ADL. The main issues have not been on language level apart but on modelling style definition. In a cooperative system, sensing and actuation is distributed over several vehicles and communicated between lead and following vehicles, as well as between vehicles and the environment (e.g. traffic signs, GPS). Modelling a cooperative system means describing multiple vehicles including sensors, actuators and vehicle mechanics, and their communication and interaction. As an example in WP6, the cruise control scenario was extended to 'platoon capable ACC'. In a platoon, a set of vehicles acts like one, enabling shorter headways in the platoon and higher throughput on the road. The platoon leader is the overall executor of platoon dynamics, i.e. braking, accelerating, lane switching. This behaviour requires a certain amount of communication between the involved vehicles, e.g. negotiation of platoon capabilities, vehicles joining/leaving a platoon, negotiation of leadership, etc.

The outcome of the work is to support the following modelling style:

![Figure 13: Cooperative function modelling style.](image)

The figure describes two system models, but could easily be extended to more. Each system has an instance of a LocalEnvironment that includes all HMI, Plant models and other vehicle local connections to each system model. Each local environment is connected to the Global environment where cooperative aspects like relative position is modelled. This Global environment needs to be adapted to the number of vehicle being modelled.

The FeatureModel on System of system level contains the platoon configuration or any configurations that are global to the function. All modelling principles on how a SystemModel is connected to the LocalEnvironment follows standard language rules and does not need documentation.
The language modifications that are necessary are concerning connectivity between System models. Normally electrical connections are modelled on logical level, meaning that the environment is circumvented with regards to modelling. But a normal Connector prototype cannot cross a abstraction level border, and even less be stretched across a system level. Hence changes are needed on Clamp connectors that are needed to connect architecture elements on the same abstraction level in different system models.

The root element is a package of some sort and needs to be determined at language level. At the same point it needs to be possible to use connectors between the entities. A language implementation needs to support this concept.

A more to the point modelling example showing the contents of the local environment, global environment and the Analysis level that would be inside a system model for two vehicles.

![System overview of a Cooperative function where two vehicles are modelled.](image)

Some connectors are left out in the figure, especially the issues on how to connect FlowPorts on AnalysisLevel artefacts on different SystemModels. More details on how to model cooperative systems can also be found in the deliverable of WT 3.4, and in the according concept presentation no. 12.
5 Brake System

The brake system is a distributed brake-by-wire system with basic functionality. It is represented in the validator model, but also as a physical setup with brake pedal, electrical brake and a single wheel. The electrical architecture is a 2-ECU topology with FlexRay communication between.

5.1 Introduction

Brake-by-wire is a term used for a braking system where no mechanical connection exists between the brake pedal pressed by the driver and the brakes located at the wheels. The position of the brake pedal is measured by a sensor and this information is the basis for computing the applied brake force. The brake force command is sent to the brake actuators electronically.

Anti-lock Braking Systems (ABS) have been used in motor vehicles for a long time. ABS prevents the wheels from locking during braking and thus allows the driver to maintain control over the vehicle. When a wheel is locked, the driver loses the ability to control the vehicle due to skidding, which makes it very hard to avoid an accident. By releasing some of the brake force the wheel can start to slowly turn while still braking, thus enabling both braking and steering at the same time.

Even though mechanical systems have been built, modern ABSs are controlled by electronics, to detect locked wheels and release hydraulic pressure in case of wheel lock. A brake-by-wire ABS goes further by replacing hydraulic command by electrical wires. It is typically distributed, i.e., sensor measurement, actuator control and computations are spread across multiple ECUs. Signals for wheel lock detection and brake force are sent as messages over a generic communication medium, typically a bus, such as CAN or FlexRay. This enables a very flexible solution, using fewer cables, but also introduces new engineering challenges to ensure proper timing characteristics of the system.

5.2 Brake System Use Cases

UC: Engage brake to a degree that no wheel lock occurs

Precondition: Vehicle Speed is non-zero

Driver engages the brakes. Within 50 ms, there is a noticeable deceleration. The deceleration is proportional to the degree of engagement.

UC: Engage brake to a degree that wheel lock occurs

Precondition: Vehicle Speed is non-zero

Driver engages the brakes with sufficient intensity to cause the wheels to lock. Within 50 ms, there is a noticeable deceleration. The deceleration is proportional to the degree of engagement. When the wheel locks, it is immediately released and a pulsating deceleration is experienced.

5.3 Brake System Architecture

This section presents the architecture of the brake system. The diagrams shown in this section use EAST-ADL and AUTOSAR terminology.

The design is first shown in a full four-wheel setup. Out of this setup, only a single wheel system will be modelled, thus simplifying the setup considerable without loosing the conceptual aspects.

Hardware Architecture

The hardware architecture shown in Figure 15 shows the full four wheel system. Each wheel has a designated ECU (ABS_FL – ABS_RR) responsible for control of a wheel speed sensor and a
brake actuator. A central brake ECU has the brake pedal sensor attached. All five ECUs are directly interconnected over a bus.

![Figure 15: Hardware Architecture of the brake system including the transfer functions of the sensors and actuators](image)

From the hardware architecture shown in Figure 15 a single wheel with the attached wheel ECU and the central brake ECU will be modelled and subsequently implemented as a physical prototype. They are connected through a flexray bus as shown in Figure 16.

External sensor and actuator used for the validator are connected directly to ECUs using local I/O. For demonstration purposes a miniature wheel with an electrical braking system is used. A simple pedal is used to simulate the driver trying to brake the car. Sensors are attached to the wheel (measuring speed) and the pedal (for measuring the pedal angle). An actuator attached to the brake allows applying a braking force to the wheel.

![Figure 16: Hardware Setup](image)

The following hardware entities are envisioned:

- 2 ECUs for running control algorithms (ECU1) and interfacing with wheel sensors and actuators (ECU2). The ECUs are internal prototype ECUs based on the Freescale MPC5516G CPU.
- A dedicated bus connecting the two sensors, based either on CAN or FlexRay
- A physical wheel serving as demonstrator for the braking functionality
- A sensor attached to the wheel for measuring the wheel speed. The sensor wheel speed sensor is attached to ECU2.
- An electrical brake attached to the wheel, attached to ECU2.
- A simple braking pedal attached to ECU1 for providing the braking pedal angle.

Software Architecture

The SW platform used on each ECU is based on AUTOSAR, with the ABS application implemented on top as an AUTOSAR application consisting of multiple AUTOSAR SW components (SW-C). The AUTOSAR BSW is provided by the Volvo AUTOSAR Platform (VAP), which includes all major BSW components, including communication, OS, diagnostics etc. VAP is based on AUTOSAR release 3.0.

Functional Architecture

To get an overview of the software architecture we first illustrate the overall functional architecture of the brake system in Figure 17. It corresponds to the Design level in EAST-ADL (a manual drawing capturing relevant aspects only). It has the full four-wheel configuration, although the validator will realize one out of four wheel controllers.

![Figure 17. Design architecture of the validator, including middleware abstraction, hardware architecture and environmental model.](image-url)

In the functional architecture the brake coordinator requests brake force from each of the wheels based on the brake pedal request. In this example there are no competing functions like ESP or ACC that potentially also could provide brake requests. The list below explains the entities in Figure 17 in more detail.

- FunctionalDesignArchitecture
o BrakePedal<<LocalDeviceManager>>  
Translation from voltage to pedal angle

o BrakeController<<DesignFunction>>  
Control of brake force for each wheel based on pedal angle

o ABSx4<<DesignFunction>>  
Hierarchichal function containing 4 ABS Controllers, one for each wheel. Calculates ABS valve commands based on vehicle speed and wheel speeds

o WheelSensor<<LocalDeviceManager>>  
Translation from pulse count to wheel speed

o BrakeActuator<<LocalDeviceManager>>  
Translation from brake force to voltage

o MiddlewareAbstraction

o BrakeIO<<MWFunction>>  
Transfer function from voltage request to voltage output to account for drivers and electronics and I/O for the brake actuator.

o PedalIO<<MWFunction>>  
Transfer function from voltage at pedal sensor to voltage reading to account for drivers and electronics and I/O for the wheel speed sensors

o WSensIO<<MWFunction>>  
Transfer function from pulse train at pedal sensor to pulse reading in to account for drivers and electronics and I/O for the wheel speed sensors

- HardwareDesignArchitecture (only transfer functions shown here, see also Figure 15)
  o BrakeX_Y  
  Brake actuator at the X/Y location
  
  o BrakePedal  
  The brake pedal
  
  o WheelSensorX_Y  
  Wheel speed sensor at the X/Y location

A possible SW implementation of the validator system is illustrated in Figure 18. Here AUTOSAR SW components (SW-C) are traced to the functions shown in the functional design architecture.

**SW Implementation**

The system consists of a number of interacting SW-Cs that transform the inputs to the system (pedal angle, wheel and vehicle speeds) into a braking force sent to the brake actuator. The pedal angle and the wheel speed are collected using actual sensors. In a real application, the vehicle speed is derived from multiple sensor readings. However, in this validator, the value is assumed to be sent from a dedicated component in the system responsible for this calculation (the Velocity SW-C).
Figure 18: Software implementation example.

Each sensor and actuator is mapped to a Sensor/Actuator SW-C. The brake controller consists of three separate SW-Cs: BaseBrake which computes the desired brake force, an arbiter which handles input from multiple functionalities, such as ACC, brake assist systems etc., and the ABS component that handles the ABS functionality for the wheel. However in this case there is only one.

Depending on HW configuration, several mappings of SW-Cs to ECUs are possible. This facilitates design space exploration, also with respect to timing.
Figure 19: The mapping of functional entities to the HW platform. Note that the system is only realized for one wheel.

Figure 19 shows a possible mapping of functional entities to the chosen HW platform. This mapping has a direct impact on the timing characteristics of the system, as it defines which signals between functional entities give rise to communication on the bus and which do not.

Finally, the SW-components might be mapped to the HW architecture as shown in Figure 20, given the design architecture in Figure 19.
Figure 20: A possible SW to HW mapping for the Validator.

5.4 System Model in Papyrus

Below is a series of diagrams representing the EAST-ADL UML model of the Brake system. Note that the diagrams differ from the previous diagrams because the latest version of the language was not available at the time of modelling.
Figure 21. System Model with ports and connectors hidden
Figure 22. Analysis Architecture

Figure 23. Design Architecture
Figure 24. Hardware Design Architecture, physical aspects

Figure 25. Hardware Design Architecture, logical aspects (the transfer functions)
Figure 26. Functional Design Architecture
Figure 27. Allocation Constraints

In EAST-ADL entities on one abstraction level realizes entities on the next level up, and this can be traced with ADLRealization dependencies. The decision to realize one entity with another may be either unique for a specific project or universal. In case of a context specific decision, ADLRealization is owned by the current SystemModel (Figure 28), while in the universal case, the ADLRealization is part of the type definition of the realizing entity (Figure 29).

Figure 28. Realizations defined in the context of the current system model.
Figure 29. Realizations that are defined in the context of each type

Figure 30. A subset of the requirements
5.5 AUTOSAR model

The implementation architecture of the brake system is defined using AUTOSAR. Based on this definition there is also a real implementation of the brake system defined using two Flexray-equipped ECUs with AUTOSAR software platform.

Figure 31. The implementation architecture

Figure 32. The AUTOSAR SW components represented in Vector DaVinci
5.6 Safety Assessment

The ISO/DIS 26262 functional safety standard concerns automotive embedded systems and prescribes an extensive information basis for the proper argumentation that a system is safe. With EAST-ADL and AUTOSAR, this information can be organized appropriately, and the required documentation and analysis supported. We will explain how each of the standard's safety lifecycle phases relates to the available modelling concepts.

In the Concept Phase of ISO/DIS26262, risk assessment is carried out based on the pure functionality of each Item. The feature definition on Vehicle level is thus the appropriate basis for identification of Hazards and subsequent definition of Safety Goals with a certain ASIL, Automotive Safety Integrity Level. Safety Goals are met by a Functional Safety Concept, which is a set of functional safety requirements allocated to abstract architectural elements. The EAST-ADL Functional Analysis Architecture with safety requirements is thus the appropriate means to represent this information. With an error model describing the potential faults and fault propagations as well as system failures, these safety requirements can be formalized adequately as EAST-ADL Safety Constraints.

The System Phase of ISO/DIS26262 involves the definition of a Technical Safety Concept, which is a set of technical safety requirements and abstract software and hardware architecture elements that elaborate the functional safety concept. The EAST-ADL functional and hardware design architectures with technical safety requirements capture this information, and the error model with safety constraints may be used to formalize requirements.

In the Software/hardware implementation Phase of ISO/DIS26262, detailed software and hardware requirements are defined and allocated to system elements. These system elements are appropriately represented by the AUTOSAR software component template and system topology.

![Diagram of Safety Assessment](image)

Figure 33. Indication of representation of Hazard information and Safety Goal for Service Brake
Figure 34. Indication of representation of Functional Safety Concept for Item Service Brake

Figure 35. Indication of formalization of safety requirement for Item Service Brake
6 Steering System

The steering system consists of an electric column lock (ECL) and electric power-assisted steering (EPAS), which replaces hydraulic power-assisted steering (HYPAS). Typical advantages of using EPAS compared with HYPAS include:

- Better fuel economy, since power is taken from the engine on demand, and not continuously from an engine-driven pump.
- Savings in development time, since steering characteristics can be tuned in-vehicle, through software.
- Reduce part numbers for the manufacturer, since the EPAS system can be made to automatically select its software configuration and calibration to match different vehicle variants.
- Reduced system weight and volume.
- Improved functionality, e.g. speed sensitivity, yaw damping, active return and optional steering “feel” settings.
- Reduced environmental impact, since no hydraulic fluid is used, fuel consumption is reduced, and recyclability is increased.

In addition, EPAS can be seen as an enabling technology for new functionality such as automated parking or collision avoidance.

There are different configurations of the EPAS, this example uses a double-pinion configuration, where the assist motor is placed beside the system. This configuration is used for heavy vehicles; in lighter vehicles, a single pinion is used, and the assist motor is packaged on the steering column, steering rack or in the pinion.

Figure 36: Example power-assisted steering
6.1 Use case diagram

A use-case diagram of the steering system is shown in Figure 37. The major use case is of course steering. Another functionality which could use the EPAS is automated parking. The “pure" Mechanical Steering is used as a back-up, for steering without any support from the motor or battery. The Electric Column Lock (ECL) is also an important part of the steering system, if it is locked, neither Mechanical nor Power-assisted steering is possible.

Figure 37: Use case diagram of the Steering System
6.2 Simulink model

A MATLAB/Simulink model in Figure 38 was adapted from the paper “A Sensorless Optimal Control System for an Automotive Electric Power Assist Steering System” (Parmar/Yung). It was corrected (the pinion ratio \(1/r\) is left out at one point of the diagram in that paper), and then modularized. The objective was to modularize the simulation blocks into different components, each represented by a Simulink subsystem.

![Simulink model](image)

**Figure 38:** The original Simulink model of the EPAS System to the left, on the right hand side the same models divided in subsystems, by component

6.3 EAST-ADL model

One of the objectives of this case study is to show how a more complex environment/plant model could be represented in EAST-ADL. To show the differences between the different modelling styles of EAST-ADL, and how they relate to different tools, the EPAS system is modelled in two different ways. In the first case in Figure 39, the demonstrator is modelled using FunctionFlowPorts, and in the other case in Figure 40, a combination of FunctionFlowPorts and the newly introduced FunctionPowerPort is created. Note that due to unavailability of the new profile having the FunctionPowerPorts, this is a mock-up diagram made in Visio.
Figure 39: WIP Papyrus model of FAA and Environment model
Figure 40: The EPAS modeled in Modelica (above) and EAST-ADL, using FunctionPowerPorts. In a real case, this model should perhaps consist of four subsystems: One of the steering column, one of the assist motor, and the pinion.
7 Security System

7.1 Security System Scenario

The goal of this demonstrator was to evaluate methods to support modelling on abstraction levels with a special focus on modelling on design and implementation level and elaborating ways to get as much benefits as possible out of model. Important design goals here are a 1.1 mapping to the code in order to avoid redundancies, simulation and analysis capabilities for purposes like timing and scheduling analysis and generation capabilities into runtimes as AUTOSAR.

Input to the demonstrator was an already existing UML model of a security system, as well as a simulation and test environment for the security system. The security system was remodelled on all abstraction levels of EAST-ADL. A component structure was added, and the security system was ported to the AUTOSAR demonstrator.

The AUTOSAR demonstrator in this case are the suitcases as described in section 3.1.

The challenge was to close the gap between an already existing UML model of class and state automata and the AUTOSAR component structure. The benefits of the UML world, like inheritance and variability, class design, state automata and AUTOSAR benefits as the communication patterns Sender/Receiver and Client/Server or the AUTOSAR mode management should be available for system and software design. Also timing information and timing constraints should be available and the system should be ready for timing and scheduling analysis.
7.2 Responsibilities of Security System

Security System

The Security System controls the armed state and the alarms given in case of an unwanted intrusion (security alarm) or a dangerous situation, where a panic alarm is raised. The security system has to collaborate with modules providing relevant data and events derived from sensors, and modules providing access to required outputs like the InfoAndWarn and the Exterior Light. The module vehicle state controls all input panel and switches. The module locking module provides the important input locking state.

An alarm is possible in case of an intrusion during an armed state or a panic alarm in the disarmed state.

The vehicle is split in into three subzones (Front, Middle, Rear), which can be armed individually. The car can be in the partially armed mode (only some subzones are armed) or in the fully armed mode (all zones are fully armed).

The life cycle of the module is controlled in a separate component, which interacts with the system mode manager, and performs all necessary operations to start or shut down the system.

All components (Armed, Panic, Alarm, Zone, Life Cycle) of the module exist in various variants, which can be selected, when security system is configured. A super class component security system is defined, which aggregates super classes of used components, and subclasses of the security system aggregate subclasses or variants of used components.

7.3 Concepts for modelling on Design and Implementation level

Several approaches are possible for modelling on design and implementation level two of them were elaborated during the ATESS2 project.

- **UML class models**: UML class models are the preferred way of software modelling today, at least in the interior division of Continental and outside the automotive and embedded industry. Pure class models have a weak semantic compared to the AUTOSAR approach having dedicated patterns on the component level. The strength is the possible 1:1 mapping to code (Together) and the capability so simulate and generate state automata (Rhapsody). The
approach was not evaluated, since the introduction of components and communication are a step forward and not available in this approach. The demonstrator input was a UML class model and state automata of the security system.

- **UML Profile Extension for composite structure diagram elements** (based on AUTOSAR elements within EAST-ADL): This approach was elaborated by defining a UML profile for SW design based on AUTOSAR elements. The security system was remodelled on base of this profile for SW design. The benefit of this approach would have been a natural fit with EAST-ADL, which is also given as a UML profile. However it turned out that there are some shortcomings using this approach. The meta model turned out to be much more complex than expected in the beginning. With the presence of concepts as components, modes, states, client/server and sender/receiver patterns, inheritance, timings based on TIMMO, classes, ... the profile requires a very elaborated set of constraints in order to be applicable and get correct models.

- **AUTOSAR:** The AUTOSAR is a runtime model for components. Many issues important for a SW design are outside the scope of AUTOSAR. (inheritance, classes, state automata, variability, closely coupled components...). AUTOSAR modelling is of course closely connected to the AUTOSAR runtime, which is not always used as a runtime. Proprietary platforms, proprietary platforms with AUTOSAR elements, or emerging Linux platforms are also used in automotive systems.

- **EAST-ADL:** When it comes to function or structural modelling EAST-ADL can be seen as a simulink model with some AUTOSAR semantic. There are also some extensions towards AADL. When it comes to design and implementation level, EAST-ADL has not worked out execution semantic like AADL. EAST-ADL and SAE-AADL can be seen as complementary.

- **Simulink:** Simulink is used for special purposes as control loops in the automotive industry. Simulink is rather a graphical programming language than a tool for describing System and Software architectures. This approach was therefore not further evaluated.

- **SAE-AADL:** SAE-AADL offers a semantic rich set of modelling elements, which almost fulfill all modelling capabilities for modelling on design and implementation level. It also comprises system design and offers a lot of analysis capabilities. It is mature since it developed over decades, and provides good tooling support. The most crucial point was the applicability for automotive purposes, since it is mainly used in the avionics industry so far. The given sample shows, that SAE-AADL is also very well suited for the automotive domain. The SAE-AADL meta model is given in EMF.

- **MARTE:** MARTE and SAE-AADL are very close in its goals. The set of people working on the standard at least overlaps, sometimes it is stated that MARTE is a UML profile for SAE-AADL. Since MARTE is much younger than AADL, it seems that there is a lot of work ahead of MARTE in order to be as mature as SAE-AADL. For this reason MARTE was not further evaluated.
7.4 UML profile extensions based on AUTOSAR semantic

The goal in this approach was to add modelling capabilities for SW design to EAST-ADL. This should include AUTOSAR elements as components, modes, communication patterns, events, as well as UML elements as class modelling, state automata and inheritance. Also TIMMO constructs for timing analysis and constraints for verification and validation should be supported. These constructs should be integrated into EAST-ADL for modelling on design and implementation level. A quite complex UML profile for this purpose was worked out and implemented in Papyrus. The profile was applied on a UML model of the security system. Also some use cases and a functional model on analysis level were modelled and linked to the design and implementation model of the security system.

The profile turned out to be more complex than expected in the beginning. Components, Runnables, AUTOSAR patterns as Sender/Receiver, or Access Points were available also UML constructs as inheritance and classes. This combination made the application of the profile quite complex. In this case a good documentation, samples, constraints which indicate the wrong usage of a language construct, code generations, good error messages and if possible a graphical notation are necessary in order to enable a user to apply the profile. This is of course possible, but outside the scope of a research project. Therefore a product or an active open source community would be required.

Figure 43: UML model of internal structure of the security system
The UML model can be either annotated with the functional EAST-ADL view as well as with the profile extensions for SW design.

Instead of following this path, which wouldn't have resulted in a really satisfactory and applicable solution, AADL was further evaluated. AADL at least addresses many of the points, which were required for SW design, and even includes design on system level. An open question was the applicability of SAE-AADL for AUTOSAR like systems.

Also some of these modelling aspects will be covered by MARTE on a UML profile base in the future. An additional third path going in this direction wouldn't make sense. It makes more sense to try to apply SAE-AADL or MARTE and extend and bring in missing elements into MARTE or the SAE-AADL standard.

7.5 SAE-AADL overview

SAE-AADL is amazing in many aspects. It brings model based development into the domain of embedded system. It has a very long history. It evolved from ADA, and was developed in research programs in the avionics industry. It can be seen as a programming language with extensions for architectural matters. Unlike non-embedded modelling languages, the core modelling construct is the thread, and not a class. The thread is seen as the component of embedded systems. Class support and inheritance however is available to some extend. AADL has to be co-used with UML/C++/Java classes, when the full semantic of object oriented modelling is required. In this case SAE-AADL can be seen as the link between objects orientation and component modelling. Furthermore static instantiation and instance models are fully supported by the property value construct. AADL has a quite obvious mapping into the C-language.

A very strong point is the support of system and software design on architectural level. Components (here threads), communication patterns, system design capabilities, modes, data flows (equivalent to TIMMO event chains) are available. The model can be analyzed in many aspects. The extension mechanism allows to introduce concepts as behaviour extensions or error modelling extensions.

SAE-AADL complements the logical view within the UML 4+1 architectural views, which is covered by UML class and dynamic diagrams (state, activity) in the following way:

- logical view: data and subprograms can be composed to a class, which partly replace UML classes or link to an OO design
- process view: threads are element of concurrency, which define the communication patterns. They can also be seen as low level components.
• implementation view: Thread groups in AADL are aggregations of threads or components. Also Modules or subsystems can be seen as a synonym for thread groups. Normally units of this granularity are allocated to processors,

• physical view: In this view (also called deployment view) the HW architecture is represented and also the operational mapping: thread/processor, data/memory, connector/bus.

An important issue here was to show that SAE-AADL is usable in the context of the automotive domain and AUTOSAR like systems. Even being an automotive standard, SAE-AADL needs samples and affirmation for the use in AUTOSAR like systems. This was shown within the demonstrator by remodelling the security system in SAE-AADL.

Further readings on SAE-AADL:
SAE-AADL flyer:
WP3\3.4\AADLdocs\ EmbeddedSystems.pdf
Syntax Overview:
WP3\3.4\AADLdocs\ AADLsyntaxPresentation.pdf
Language summary:
WP3\3.4\AADLdocs\ AADLLanguageSummary.pdf
Comprehensive documentation:
WP3\3.4\AADLdocs\ AADLIntroduction06tn011.pdf
Intro Chen:
WP3\3.4\AADLCase\ AADL_CaseStudy_I_KTH_DC.pdf
Article Usage AADL:
http://www.embedded.com/design/opensource/201201811?_requestid=1246710
UML, AADL, SysML comparison:
WP3\3.4\AADLdocs\ UML_AADL_SysML_Comparison.pdf
UML 4+1 view for Software Intensive Systems:
WP3\3.4\AADLCase\ S09-58.pdf
7.6 SAE-AADL in Automotive process model

AADL fits well into the Automotive process models. The CMMI process model and AADL are both developed at the Software Engineering Institute of the Carnegie Mellon University/Pittsburgh. It supports the following process steps:

- Requirements Engineering (unsupported)
- System Design (supported)
- Software Architecture Design (supported)
- Component Software Design (supported)
- Code Development, Class Design (supported)
- Unit Test (unsupported)
- Integration Test (unsupported)
- System Test (unsupported)
- Acceptance test (unsupported)

Figure 44: AADL usage within an Automotive process framework

Process steps in this form can also be found and are followed at Continental. In the demonstrator sample, the goal was to work out an AADL model, supporting the process steps from above:

- AADL in Class/Logical Design: A class design can either be the logical design of an application, or the support of the realization of (platform) services. In the security system sample four state automata were modelled as classes with access methods. A class design can easily be mapped into C/C++ source code.

- AADL in System Design: In the system design the hardware and system architecture is defined. A system can consist either of hardware software or hardware and software elements. For system instances the operational mapping, e.g. Thread/Processor mapping can also be defined. Also the system boundaries between Hardware and Software are
defined. Additional information like timing requirements can be inserted on this level. The SW is represented by the AADL element process, the HW with the AADL elements processor, memory and bus, the system by the AADL element system.

- **AADL in Software Architecture Design:** In the architecture design the partitioning of the SW is defined. The architecture description includes the platform services, the collaboration of various subsystem and concepts for global mode management. GUI interfaces and collaboration with the GUI could also play an important role. The software (sub) systems are represented as AADL thread groups or threads for simple systems.

- **AADL in Component Software Design:** The defined subsystems (thread groups) are here broken down to threads. Threads can be seen as components in SAE-AADL. Threads may only be active in certain system modes, or define itself modes where different subprograms are called. Threads could be loosely coupled as in AUTOSAR but also closely coupled by the data access pattern. The runtime semantic allows scheduling analysis on base of the information given in an AADL model.

- **AADL and code development:** AADL provides the software elements data and subprogram. They can be co-used realizing a class like semantic. These elements are the entry points into code like the entry point of an AUTOSAR runnable. Data elements can be nested, subprograms can call other subprograms. For all elements properties can be defined and set. A reduced form of inheritance is available. If the full power of UML/C++ classes is required, subprograms and data elements can link these designs with the thread model.

### 7.7 SAE-AADL and logical Design/Requirements Engineering

SAE-AADL is not really a method to do logical designs with. The first source for logical designs is still UML. The behaviour is captured in dynamic diagrams (activity, state diagrams). Class diagrams and optionally sequence diagrams are used to structure the dynamic behaviour. Complex class designs cannot be done in SAE-AADL. In SAE-AADL only simple class designs with static classes only can be done. These simple classes can link into source code, developed or generated on base of logical designs. This allows the connection of behavioural diagrams into SAE-AADL component designs.

As an alternative to graphical design a language in order to define behaviour is proposed in a behavioural annex of SAE-AADL.

In security system itself consists of four state automata, which are encapsulated by simple SAE-AADL classes. The state automata were modelled in UML.
The following model samples show the deployment view of the security system. It is here now to be seen as a sample how to use SAE-AADL for system design. It shows the security system as a software and hardware system as well as it deployment onto a one or two processor scenario, as well as its integration into a test scenario. At the end it is shown, how to instantiate a system and how to add a thread to processor mapping (operational mapping).
Figure 46: Subsystems of Security System test system

The model shows the integration of the security system into a system consisting of the security system, a graphic PC and a PC providing a restbus simulation. Here no information on the internal structure of the software or hardware is given.
Figure 47: generic HW and SW of security system

The model shows the internal structure of the security system. A generic software and hardware system is defined in an AUTOSAR style. The SW structure remains independent of the HW structure. SW and HW are only connected by the operational mappings (thread to processor; data to memory; connector to bus). The SW system and HW system can be further broken down. Here in the system design, only the HW is further broken down. The SW is further broken down in the defined process steps SW architecture and component design.
Here the generic HW of the security system broken down to one ECU node. Processor node consists of AADL devices, busses, processors and a memory. The interface is inherited from generic HW system.

Figure 48: Security System HW with one deployment target
The HW system can also be broken down onto two HW System. This aggregate also inherits from the generic hardware system. The two HW systems can be further broken down into ECU nodes as for the one processor node.

Operational Mapping

The following model fragment shows the definition of the generic HW and SW system, but also a possible instantiation of the system with a Thread/Processor mapping.

```
SecuritySystemECU.implEvHw2 here extends and the generic SecuritySystemECU.impl by refining the HW system.

The properties Allowed_Processor_Binding and Actual_Processor_Binding show how to define a thread/processor mapping.

system implementation SecuritySystemECU.impl
  subcomponents
    hwSystem: system SecuritySystemECUHW;
    swSystem: process PISecuritySystem::SecSysProcess;
  connections
    PortGroupConnection1: port group swSystem.hmi -> mygraficOut;
    PortGroupConnection2: port group myPosOut -> swSystem.position;
    PortGroupConnection3: port group swSystem.exteriorLight -> hwSystem.exteriorLight;
    PortGroupConnection4: port group swSystem.buzzer -> hwSystem.buzzer;
    PortGroupConnection5: port group swSystem.plips -> hwSystem.plips;
```
PortGroupConnection6: port group swSystem.keyStatus $\rightarrow$ hwSystem.keyStatus;
PortGroupConnection7: port group hwSystem.panels $\rightarrow$ swSystem.panels;
PortGroupConnection8: port group hwSystem.switches $\rightarrow$ swSystem.switches;
BusAccessConnection1: bus access can $\rightarrow$ hwSystem.can;
end SecuritySystemECU.impl;

system implementation SecuritySystemECU.implEvHw2
extends SecuritySystemECU.impl
subcomponents
hwSystem: refined to system SecuritySystemECUHWH.impl2;
swSystem: refined to process PISSecuritySystem::SecSysProcess.implEvEvents;
properties
Allowed_Processor_Binding => reference hwsystem.ecu1.p1 applies to swSystem.infoWarn;
Allowed_Processor_Binding => reference hwsystem.ecu2.p2 applies to swSystem.immobilizer;
Actual_Processor_Binding => reference hwsystem.ecu1.p1 applies to swSystem.infoWarn;
Actual_Processor_Binding => reference hwsystem.ecu2.p2 applies to swSystem.immobilizer;
end SecuritySystemECU.implEvHw2;

7.9 SAE-AADL in Software Architecture design

In the architectural design the brake down or partitioning of a SW system takes place. The collaboration of the subsystem with other subsystems is shown (locking system, immobilizer, remote keyless entry, vehicle state). The model here also shows the integration into higher level software services (InfoWarn). Lower level platform services (AUTOSAR-BSW) need no presentation here. In most cases the access to the platform is done by collaborating systems of the security system and not directly by the security system.

Figure 50: Security system and collaborating systems defined as SAD-AADL Thread Groups
The system partitioning shows how to integrate the system in an overall system or how to build up a test environment, where module or unit tests of the security system can take place.

![Process Instance Diagram: null / modeManagers](image)

Figure 51: Security System with mode managers

The following model shows the collaboration of the security system with two mode managers. On one side the system mode manager and on the other side a mode manager based on modes provided by the security system itself. In this small sample only the security system is a consumer of the security system modes. In an overall vehicle system, other consumer of security system modes may exist. The security system mode manager is closely coupled to the security system.
Figure 52: Transitions and modes of the security system.

The diagram shows the mode transition diagram of the security system. Mode transitions are triggered by data events raised by the security system model manager or other mode manager. All elements of the security system can be controlled by the security system modes. This means threads can be started or stopped when entering or leaving a mode.

7.10 SAE-AADL in Software Component Design

In the following diagrams the component design of the security system is given. A component in SAE-AADL is represented by a thread. The security systems can be broken down into 6 threads, four of them are mode independent state machines (armedControl, panicAlarmControl, armedAlarmControl and zoneControl) and two mode dependent worker threads alarmRunner and positionRunner.
Figure 53: Components of Security System given as threads
Figure 54: The platform accesses of the components of the security system

The models above show the thread group security system with the containing threads, and the interfaces to other systems. The collaboration among the threads is not part of this (generic) model. In refinements of the thread and thread groups the threads collaborate using the event pattern or the data access pattern. (The mapping of AUTOSAR pattern and SAE-AADL patterns is given below). Also direct accesses to the platform (e.g. AUTOSAR BSW) are given in a separate diagram. The BSW itself is represented by a thread.
Figure 55: Refinement closely coupled threads

Refinement of the security system, collaboration between threads takes place using the data access pattern. This leads to closely coupled threads.

Figure 56: Refinement loosely coupled threads
Refinement of the security system, collaboration between threads takes place using the event pattern. This leads to loosely coupled threads (except one remaining data access).

### 7.11 SAE-AADL and Code development

The state automata, which are used by the control threads, are modelled in a logical design. Logical designs are broken down into classes, as the security system is given by four state automata and its encapsulating classes. Classes are aggregated by threads. The connection between threads and class designs is given in a thread implementation. The logical class can be transformed 1:1 into code on base of a 1.1 Mapping. The coding of the class method still has to be done here. If e.g. a state automata is part of the logical design, also methods can be partly or fully generated out of a state automata. If non static OÖ classes (UML/C++) are used the class design is given outside of SAE-AADL, and a static code entry point is provided, in order to connect the component model and the implementation classes.

![State Automata aggregated by a thread](image)

**Figure 57: State Automata aggregated by a thread**

### 7.12 SAE-AADL integration in UML/AUTOSAR modelling approaches

**Model to Configuration**

The mapping of SAE-AADL SW elements in to source code is quite obvious, which does not necessarily mean very simple. SAE-AADL semantic is much richer than AUTOSAR or UML modelling approaches. Threads and collaboration are normally configured outside of the model. (osek, can, proprietary XML files, EMF models). Depending on the semantic match of SAE-AADL
and these XML/EMF files a mapping or transformation must be defined. The modelling concepts of AUTOSAR are a subset of the modelling concepts of SAE-AADL.

Model to Code

In addition SW elements given in C-Source code can be represented in AADL quite easily. The subprogram maps on a C function or static C++ function, the data element maps on a C record. Classes in SAE-AADL are the logical encapsulation of C records by C-functions. The advantage of this mapping is that SAE-AADL is an excellent way of introducing objects and even inheritance into the C-language in addition giving the clue code to the thread or component model. The disadvantage of this mapping is that the full power of object orientation of UML/Java/C++ is not available. In this case only the static parts of the code can be represented in SAE-AADL, the non static classes must still be represented in UML or the programming language.

As a consequence it is always an option to co-use proprietary source code configuration tools or the AUTOSAR BSW configuration tool with SAE-AADL. There is always the option to represent these configurations in AADL on base of the C<->AADL mapping. Furthermore SAE-AADL itself defines language constructs, which allow the definition of properties and their static configuration for all model elements. In the following tables mappings between SAE-AADL and AUTOSAR and C/C+++ are given.

<table>
<thead>
<tr>
<th>SAE-AADL element</th>
<th>AUTOSAR element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>Component</td>
</tr>
<tr>
<td>Event Port, Event Data Port</td>
<td>Port with SR event semantic</td>
</tr>
<tr>
<td>Event Port, Event Data Port</td>
<td>Data Send/Receive Point</td>
</tr>
<tr>
<td>Data Port</td>
<td>Port with SR data semantic</td>
</tr>
<tr>
<td>Data Port</td>
<td>Data Access Point</td>
</tr>
<tr>
<td>Data Access subprogram</td>
<td>Port with Client Server Interface</td>
</tr>
<tr>
<td>Data Access server subprogram</td>
<td>Port with Client Server Interface</td>
</tr>
<tr>
<td>Subprogram</td>
<td>Runnable</td>
</tr>
<tr>
<td>Thread Group</td>
<td>Composite</td>
</tr>
<tr>
<td>Process</td>
<td>Not available</td>
</tr>
<tr>
<td>System</td>
<td>Not available</td>
</tr>
<tr>
<td>Data</td>
<td>Not available</td>
</tr>
<tr>
<td>Mode</td>
<td>Mode</td>
</tr>
</tbody>
</table>
EAST-ADL and SAE-AADL are both architectural languages, which have a lot in common and cover a lot of the same issues. They cover architectural issues as variability, timing, behaviour, error modelling, modes, structuring based on composite structure diagrams and other issues, although the languages are different in the scope and in its roots.

The views taken in the languages are different. SAE-AADL is more or less an extension for programming languages for architectural modelling. This excludes the view on more abstract levels as analysis level and feature level. Like AUTOSAR SAE-AADL covers the system and software design and implementation level. EAST-ADL on the other hand does not really cover the SW design and implementation level and refers and links to AUTOSAR elements. EAST-ADL covers the more abstract analysis and feature level.

Another important difference is the focus on functional models in EAST-ADL which roots in its strong relation to Simulink and control loop engineering. Components based or derived from EAST-ADL models are therefore based on this functional view. SAE-AADL has its roots in the programming language ADA. It tries to close the gap between component oriented architectural modelling and object oriented modelling used modelling domains with more complex architectures. SAE-AADL itself is not fully object oriented but can at least be co-used with OO models and add component models to these approaches.

The non overlapping views are the feature modelling view of EAST-ADL and the implementation view of SAE-AADL. When it comes to modelling on analysis and design level it is to my opinion domain dependant, if a more functional view as in control loop engineering or a more object oriented view as in event based SW domain models, is taken. In the following table a mapping between SAE-AADL and EAST-ADL elements are given.

### SAE-AADL element | EAST-ADL element
---|---
Thread | Function
Event Port, Event Data Port | Flow Port
Data Port | Flow Port with trigger
Data Access subprogram | Port with Client Server Interface
Data Access server subprogram | Port with Client Server Interface
Subprogram | Not available
Thread Group | Function
7.14 Conclusions and further work

The answer to the appropriate modelling concepts and tooling is always very specific to the modelling domain. Simulink could be the right choice for developing control loops. EAST-ADL may be the method of choice in a Simulink like environment with special focus on architectural issues. The view taken in this demonstrator approach is appropriate in an environment, where more complex embedded SW architectures are required (e.g. periodic & aperiodic events, modes). In this case SAE-AADL has been proven as appropriate and very mature. Further more a good open source tooling (OSATE) based on eclipse and EMF is available. The model is even given in an EMF model and a programming language at the same time.

In the last two month of the project the analysis capabilities, code generations and simulation capabilities will be examined. This will only be a first look at those capabilities, not a comprehensive evaluation, since only two months are left in the project.

AADL covers a lot of issues in system and software design. However a language extension to define and evaluate test cases is outside the scope of AADL. Such an extension would have to include an appropriate constraint language. Also the use of already existing timing information of the model could be in the scope of such a constraint language. This can be a subject to further extensions of SAE-AADL and MARTE and also EAST-ADL in order to support verification and validation.
8 Safety analysis

A safety analysis has been performed for HAVEit demonstrator system, with focus on the cruise control system. It applies the rules and concepts of the ISO DIS 26262 [8], and the methods and tools developed in the project.

8.1 Requirements on Vehicle level

The (functional) requirements refine the features given in Sect. 2.3.2. A non-fulfillment of a requirement is a malfunction, entailing a (potential) Hazard. Table 1 shows an extract of the features, requirements, and potential hazards. The entire list can be found in a separate file [9], which is part of the annex for this deliverable.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement</th>
<th>(potential) Hazard</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC_Basic</td>
<td>1: CC shall be activatable by the HMI</td>
<td>1: CC cannot be activated</td>
</tr>
<tr>
<td></td>
<td>2: CC shall be deactivatable by HMI</td>
<td>1: Unintended vehicle acceleration/propulsion (due to too high vehicle speed setpoint when activating)</td>
</tr>
<tr>
<td></td>
<td>3: In case CC_SuspendableAndResumeable=false:CC shall be deactivatable, if the service brake is activated</td>
<td>2: Unintended vehicle braking (due to too low vehicle speed setpoint when activating)</td>
</tr>
<tr>
<td></td>
<td>4: CC shall be deactivatable, if the parking brake is activated</td>
<td>3: Unintended vehicle acceleration/propulsion (because CC cannot be deactivated)</td>
</tr>
<tr>
<td></td>
<td>5: An active CC shall control the longitudinal vehicle speed in forward direction according to the target speed setpoint. Valid speed range is 40km/h to 200km/h.</td>
<td>4: Unintended vehicle acceleration/propulsion (because CC cannot be deactivated)</td>
</tr>
<tr>
<td></td>
<td>6: An active CC shall accelerate the vehicle by dedicated acceleration request via PTC</td>
<td>2: Unintended vehicle braking (because CC cannot be deactivated)</td>
</tr>
<tr>
<td></td>
<td>7: A passive CC shall not affect the longitudinal vehicle speed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8: CC shall stays active, if CC is active and the driver accelerates the vehicle above the current target speed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9: If CC is active and the driver accelerates the vehicle above the current target speed, the vehicle is decelerated according CC_DecelerationPerformance</td>
<td></td>
</tr>
<tr>
<td>CC_HoldDistance</td>
<td>1: ACC shall be activatable via HMI</td>
<td>1: ACC cannot be activated</td>
</tr>
<tr>
<td></td>
<td>2: In case CC is not active and ACC is activated, the CC target speed is set to the current vehicle speed.</td>
<td>1: Unintended vehicle acceleration/propulsion (with respect to distance to object ahead)</td>
</tr>
<tr>
<td></td>
<td>3: ACC controls the distance to the vehicle ahead for speeds below 200km/h</td>
<td>3: Distance to vehicle ahead too big (because vehicle speed is above limit)</td>
</tr>
<tr>
<td></td>
<td>4: ACC and nested CC shall be deactivated by pressing the service brake pedal</td>
<td>3: Distance to vehicle ahead too small (because vehicle ahead brakes)</td>
</tr>
<tr>
<td></td>
<td>5: ACC and nested CC shall be deactivated</td>
<td>4: Unintended vehicle acceleration/propulsion (due to too high PTC request)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7: Unintended vehicle acceleration/propulsion (due to too high PTC request)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8: too low vehicle acceleration/propulsion (driver cannot accelerate vehicle above target speed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6: Unintended vehicle braking (due to unintended CC deactivation)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9: Unintended vehicle braking (after driver's acceleration with active CC, depending on CC_DecelerationPerformance)</td>
</tr>
</tbody>
</table>
by activating the parking brake
6: ACC shall be deactivatable by HMI. CC gets re-activated
7: ACC and nested CC shall be deactivatable by HMI
8. The distance depends on the current vehicle speed

(ACC/CC cannot be deactivated)
5: Unintended vehicle acceleration/propulsion (ACC/CC cannot be deactivated)
7: Unintended vehicle acceleration/propulsion (ACC/CC cannot be deactivated)
6: Unintended vehicle braking (return to CC)
6: Unintended vehicle acceleration/propulsion (return to CC)
7: Unintended vehicle braking (ACC cannot be deactivated)
8: Distance to vehicle ahead too big (wrong computation of distance)
8: Distance to vehicle ahead too small (wrong computation of distance)

Table 1: Refinement of the features given in section 2.3.3 (extract)

8.2 Driving situations

The main relevant driving situations have been worked out. Other situations, like driving off-road, or driving at night, have not been considered.

The main parameters are:

- **Maximum speed:**
  - Standing
  - low speeds, up to maximum 60kph
  - mid speeds, up to 100kph
  - high speed, up to top speed

- **Bending situation**
  - straight road
  - bending classified
    - according bending radius: narrow or non-narrow
    - blind or non-blind
    - high bank or non-high bank

- **Lane situation**
  - one lane or more than one lane
  - one way or two ways

- **Road slope situation**
  - uphill / on flat road / downhill
  - no slope / moderate slope / high slope

- **Friction situation (tire – road)**
  - good / poor

Figure 58 gives an overview and shall be read row-wise in the following way:

Row 1: The following situations are created
- Situations, where the vehicle is standing
- bending is not relevant
- lanes are not relevant
- 5 different road slopes
- friction conditions poor and good

=>$ 1 \times 1 \times 1 \times 5 \times 2 = 10$ different situations, related to drive away situations

Row 2: $1 \times 3 \times 3 \times 5 \times 2 = 90$ different situations, related to driving in a city

Row 3: $1 \times 4 \times 3 \times 3 \times 2 = 72$ different situations, related to driving on country road

Row 4: $1 \times 3 \times 2 \times 3 \times 2 = 36$ different situations, related to driving on highway

So, $10 + 90 + 72 + 36 = 208$ different main driving situations are considered for the risk and hazard analysis.

Figure 58: Driving situations considered in the risk analysis

### 8.3 Preliminary Hazard analysis and ASIL assignment

<table>
<thead>
<tr>
<th>Situation</th>
<th>Speed</th>
<th>Bending</th>
<th>lane</th>
<th>slope</th>
<th>friction condition</th>
<th>Relevant for</th>
</tr>
</thead>
<tbody>
<tr>
<td>S20</td>
<td>low</td>
<td>straight road</td>
<td>one lane, one way</td>
<td>high slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S40</td>
<td>low</td>
<td>straight road</td>
<td>more than one lane, two ways</td>
<td>high slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S45</td>
<td>low</td>
<td>narrow, blind, non-high bank</td>
<td>one lane, one way</td>
<td>on flat road</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S46</td>
<td>low</td>
<td>narrow, blind, non-high bank</td>
<td>one lane, one way</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S47</td>
<td>low</td>
<td>narrow, blind, non-high bank</td>
<td>one lane, one way</td>
<td>moderate slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S48</td>
<td>low</td>
<td>narrow, blind, non-high bank</td>
<td>one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S49</td>
<td>low</td>
<td>narrow, blind, non-high bank</td>
<td>one lane, one way</td>
<td>high slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S50</td>
<td>low</td>
<td>narrow, blind, non-high bank</td>
<td>one lane, one way</td>
<td>high slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td></td>
<td>high bank</td>
<td>downhill</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----------</td>
<td>----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S55</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way on flat road good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S56</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way on flat road poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S57</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way moderate slope downhill good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S58</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way moderate slope downhill poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S59</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way high slope downhill good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S60</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way high slope downhill poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S66</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways on flat road poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S68</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways moderate slope downhill poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S70</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways high slope downhill poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S75</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>one lane, one way on flat road good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S76</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>one lane, one way on flat road poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S77</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>one lane, one way moderate slope downhill good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S78</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>one lane, one way moderate slope downhill poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S79</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>one lane, one way high slope downhill good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S80</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>one lane, one way high slope downhill poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S85</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>narrow, non-blind, non-high bank on flat road good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S86</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way on flat road poor CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S87</strong></td>
<td>low narrow, non-blind, non-high bank</td>
<td>more than one lane, one way moderate slope downhill good CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S88</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S89</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>high slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S90</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>high slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S95</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S96</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S97</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>moderate slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S98</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S99</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>high slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S100</td>
<td>low</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>high slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S115</td>
<td>mid</td>
<td>straight road</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S121</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>one lane, one way</td>
<td>on flat road</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S122</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>one lane, one way</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S123</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>one lane, one way</td>
<td>moderate slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S124</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S127</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>on flat road</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S128</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S129</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>moderate slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S130</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S133</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S134</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S135</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>moderate slope downhill</td>
<td>good</td>
<td>CC</td>
</tr>
<tr>
<td>S136</td>
<td>mid</td>
<td>narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S140</td>
<td>mid</td>
<td>non-narrow, non-blind, non-high bank</td>
<td>one lane, one way</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S142</td>
<td>mid</td>
<td>non-narrow, non-blind, non-high bank</td>
<td>one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S146</td>
<td>mid</td>
<td>non-narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S148</td>
<td>mid</td>
<td>non-narrow, non-blind, non-high bank</td>
<td>more than one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S152</td>
<td>mid</td>
<td>non-narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S154</td>
<td>mid</td>
<td>non-narrow, non-blind, non-high bank</td>
<td>more than one lane, two ways</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S158</td>
<td>mid</td>
<td>non-narrow, non-blind, high bank</td>
<td>one lane, one way</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S160</td>
<td>mid</td>
<td>non-narrow, non-blind, high bank</td>
<td>one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S164</td>
<td>mid</td>
<td>non-narrow, non-blind, high bank</td>
<td>more than one lane, one way</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S166</td>
<td>mid</td>
<td>non-narrow, non-blind, high bank</td>
<td>more than one lane, one way</td>
<td>moderate slope downhill</td>
<td>poor</td>
<td>CC</td>
</tr>
<tr>
<td>S170</td>
<td>mid</td>
<td>non-narrow, non-blind, high bank</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>poor</td>
<td>CC</td>
</tr>
</tbody>
</table>
This ASIL is a requested ASIL to be considered at the implementation of the feature.

Table 2: Summary of some relevant situations

<table>
<thead>
<tr>
<th>Situation</th>
<th>Hazardous Event</th>
<th>Safety Goal</th>
<th>Safe State</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>mid</td>
<td>non-narrow, non-blind, high bank</td>
<td>more than one lane, two ways</td>
<td>moderate slope downhill</td>
<td>poor</td>
</tr>
<tr>
<td>high</td>
<td>straight road</td>
<td>more than one lane, one way</td>
<td>on flat road</td>
<td>good</td>
</tr>
<tr>
<td>high</td>
<td>straight road</td>
<td>more than one lane, two ways</td>
<td>on flat road</td>
<td>good</td>
</tr>
</tbody>
</table>

Table 3: Hazardous Events for Cruise Control

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HE_CC_01</td>
<td>Unexpected vehicle deceleration in overtaking manoeuvre (when driving at medium-high speed)</td>
</tr>
<tr>
<td>HE_CC_02</td>
<td>Unexpected vehicle deceleration when driving in straight road at low speed (in any adherence condition)</td>
</tr>
<tr>
<td>HE_CC_03</td>
<td>Unwanted vehicle acceleration/propulsion when driving on straight/flat road, at high speed (in any adherence condition)</td>
</tr>
<tr>
<td>HE_CC_04</td>
<td>Unwanted vehicle acceleration/propulsion when driving on straight road, on high slope downhill, at low speed, in low adherence conditions</td>
</tr>
<tr>
<td>HE_CC_05</td>
<td>Unwanted vehicle acceleration/propulsion when going round a bend at low-medium speed, with medium-high lateral acceleration (good adherence)</td>
</tr>
<tr>
<td>HE_CC_06</td>
<td>Unwanted vehicle acceleration/propulsion when going round a bend at low-medium speed, in low adherence conditions</td>
</tr>
</tbody>
</table>

Table 4: Safety Goals for Cruise Control

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG_CC_01</td>
<td>Do not apply any improper engine torque overlay request (improper: - not enable by the driver; - not enable for resume; - to high engine torque overlay request)</td>
</tr>
<tr>
<td>SG_CC_02</td>
<td>notify the driver of function unavailability</td>
</tr>
</tbody>
</table>

Table 5: Safe States Goals for Cruise Control

<table>
<thead>
<tr>
<th>Feature</th>
<th>Hazard</th>
<th>Situation</th>
<th>Hazardous Event</th>
<th>Safety Goal</th>
<th>Safe State</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC_BASIC</td>
<td>Unintended vehicle acceleration/propulsion</td>
<td>S175, S181</td>
<td>HE_CC_03</td>
<td>SG_CC_01</td>
<td>SS_CC_01</td>
<td>A</td>
</tr>
<tr>
<td>CC_BASIC</td>
<td>Unintended vehicle acceleration/propulsion</td>
<td>S20, S40</td>
<td>HE_CC_04</td>
<td>SG_CC_01</td>
<td>SS_CC_01</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>CC_Basic</td>
<td>Unintended vehicle acceleration/propulsion</td>
<td>S45, S46, S47, S48, S49, S50, S55, S56, S57, S58, S59, S60, S66, S68, S70, S75, S76, S77, S78, S79, S80, S85, S86, S87, S88, S89, S90, S95, S96, S97, S98, S99, S100, S121, S122, S123, S124, S127, S128, S129, S130, S133, S134, S135, S136, S140, S146, S148, S152, S154, S158, S160, S164, S166, S170, S172, S175, S181</td>
<td>HE_CC_05</td>
<td>SG_CC_01</td>
<td>SS_CC_01</td>
<td>A</td>
</tr>
<tr>
<td>CC_Basic</td>
<td>unwanted freewheeling/missing propulsion</td>
<td>S115, S181</td>
<td>HE_CC_01</td>
<td>SG_CC_02</td>
<td>disable CC function</td>
<td>A</td>
</tr>
<tr>
<td>CC_Basic</td>
<td>too low vehicle acceleration/propulsion</td>
<td>S115, S181</td>
<td>HE_CC_01</td>
<td>SG_CC_02</td>
<td>disable CC function</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 6: Feature – Hazard – Situation – Hazardous Event – Safety Goal – Safe State – ASIL (extract from [9])

8.4 Safety Analysis with HipHops and EPM

The tools HipHops and EPM have been applied in order to perform a fault tree analysis and FMEA table generation for the HAVEit demonstrator WP4300.

The result is a html based representation of the fault trees and FMEA tables, which can be viewed and navigated with any web browser. It is delivered in a separate file [10].
Input from previous ATESST2 work

WP2 identifies the stakeholders’ needs that should drive the extension of EAST-ADL. The WP collects a list of stakeholders and provides a list of needs and requirements for the results of ATESST2. These requirements are taken into account for the definition and development of the WP6 case study.

WP3 defines new language constructs of the EAST-ADL, especially for behavior modeling, safety analysis, requirement management, and variant handling. These language constructs are employed and evaluated by the models of the case study.

WP4 develops the EAST-ADL metamodel and accompanying tool support, both of which will be employed for the case study development.

WP5 develops a design methodology and provisioning of guidelines for system modeling. These are taken into account and exemplary applied in the case study, thereby providing feedback to WP5 regarding their applicability.
10 Contribution to overall ATESTST2 objectives

The main contribution of WP6 to the ATESTST2 objectives is the assessment and exemplary application of the results of the other WPs as described in Chapter 9. That way, WP6 will throughout the project provide valuable feedback and improve the overall results of the project. The following list is an extract from the ATESTST2 Description of Work, describing the overall ATESTST2 objectives that are relevant for WP6:

- Harmonize the structural descriptions of EAST-ADL with the latest evolutions of existing approaches, i.e. the AUTOSAR initiative, the OMG (UML2, UML profile for MARTE and SysML) and with the SAE AADL. This will be the back bone onto which further language constructs can be attached. ATESTST2 results will be concretized in a new major release of the EAST-ADL.

- Develop requirements and V&V capabilities to deal with cooperative active safety systems. In particular, the V&V aspects of the interaction between the embedded real-time system, its environment and the application will be further investigated. Safety related requirements for such systems will be addressed and a way of supporting a safety case will be fully incorporated into the language.

- Develop adequate behavioral modeling for EAST-ADL. The purpose is to capture behavior (including non-desired behaviors) and algorithms of the vehicle systems and its environment, forming the cooperative active safety system. This includes developing a native behavioral notation that allows simulation and verification within the defined system model, providing the ability to assess desired as well as emerging systems behaviors.

- Develop and adapt analysis techniques suitable for assessing safety, reliability, performance and cost, including their trade-offs for cooperative active safety systems.

- Improve the support for field operational tests by providing explicit descriptions of desired behaviors, test-cases and test results. This has the potential to enhance the planning, execution and evaluation of field-operational tests. Further, it would reduce the need for a large amount of testing that instead can be covered by model-based simulation and analysis.

- Investigate, develop, and validate language mappings to Mathworks’ Simulink, Safety analysis and other relevant domain-specific modeling tools. This will enable realization of interfaces between external tools and an EAST-ADL tool environment. which in turn will enable V&V of EAST-ADL models using those tools.

- Develop support for reuse and variability management, especially user support for single product configuration with regard to safety aspects together with the possibility to describe configurations of cooperative active safety systems. Identified language constructs will be added to EAST-ADL.

- Develop methodology and guidelines supporting end user application of the ATESTST2 concepts. This is necessary to ensure that end users apply the concepts according to the same principles.

- Develop tools realizations, including an experimental tool, based on the Eclipse framework, as well as the investigation of the support possible through standard UML tools by use of the UML2 profile implementation. Interfaces to external domain tools will be developed for the experimental tool.

- Develop automotive system application examples for cooperative active safety systems that can be used to validate and demonstrate the architecture description language, tool and methodology developed in the ATESTST2 project.
11 Conclusions

This report has described the ATESST2 case study developed in WP6. The case study consists of several application examples provided by different partners. The goal was to provide an example system, including several scenarios, that aids communication among the project stakeholders and can be used in assessments and evaluations during the project. Throughout the project, WP6 provided feedback to other WPs regarding the applicability of the concepts developed, thereby improving the overall results of the project.
12 References


[7] ATESTST2_WP6_SecuritySystem_Overview_V0.1.doc in the WP6 deliverables folder

