ATESST

Grant Agreement 224442

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

Report type Deliverable D2.1

Report name State of practice and State of the art

Dissemination level	**PU**
Status	**Final**
Version number	**1.0**

Authors

Editor	E-mail
Anders Sandberg, Mecel AB	anders.sandberg@mecel.se

Authors	E-mail
Martin Törngren	Martin@md.kth.se
Martin Walker	Martin.Walker@hull.ac.uk
Yiannis Papadopoulos	y.i.papadopoulos@hull.ac.uk
Nidhal Mahmud	N.Mahmud@2006.hull.ac.uk
Huascar Espinoza	Huascar.Espinoza@cea.fr
Fulvio Tagliabò	fulvio.tagliabo@crf.it
Sandra Torchiaro	sandra.torchiaro@crf.it
Andreas Abele	Andreas.Abele@continental-corporation.com
Lars-Olof Berntsson	Lars-Olof.Berntsson@volvo.com
DeJiu Chen	Chen@md.kth.se
Henrik Lönn	Henrik.Lonn@volvo.com
David Servat	David.Servat@cea.fr
Friedhelm Stappert	Friedhelm.Stappert@continental-corporation.com
Matthias Biehl	biehl@md.kth.se

The Consortium

Volvo Technology Corporation (S)	VW/Carmeq (D)	Centro Ricerche Fiat (I)
Continental Automotive (D)	Delphi/Mecel (S)	
Mentor Graphics Hungary (H)	CEA LIST (F)	
Kungliga Tekniska Högskolan (S)	Technische Universität Berlin (D)	University of Hull (GB)

Revision chart and history log

Version	Date	Reason
0.1	2008-11-13	Initial draft with outlines.
0.2	2008-12-04	Outline with editor input requests
	2008-12-10	Added: List of Abbreviations Enhanced: Section 1.1 by electronic control systems References formatted as numbered items, to be insert in text via Insert->Reference->"Cross-references …"
	2008-12-11	Minor additions and updates. Text added for section 2.2, 3.1.1 and 3.1.2
	2008-12-18	Sections 3.1.3.1 (safety analysis) 3.1.6.2 (optimisation) incorporated in version …V0.1. by UOH (as requested by editor)
	2008-12-18	Section 3.1.6.1 incorporated in version …V0.1. by CEA
	2008-12-18	Sections 3.1.3 & 3.1.6 edited by UOH
	2008-12-19	Section 2.3 (CRF input: State of practice from the automotive industry)
0.3	2008-12-19	Section 3.1.3.2 (ISO 26262 adoption)
0.4	2008-12-20	Section 3.1.4: Text from Mark-Oliver Reiser added.
	2008-12-21	Updated complete reference scheme. All items in reference are numbered as “seq bibitem”. All are given a bookmark and are consequently used using “seq bibitem bookmarkTag” in the text.
	2009-01-19	Updated CON_AA distribution
0.5	2009-01-20	Integrated HL contributions
		Added tasks added at M2
	2009-01-26	Conclusions: UOH added part of conclusions regarding sections on safety analysis and optimisation
0.9	2009-01-26	Updated several sections, removed most revision marks.
		Added figure 2
		Added initial text in the Conclusions section
		Updated project, languages, tools section.
	2009-01-26	UoH: removed redundant text from concluding parts of sections 3.1.3 and 3.1.5. Most of this is now discussed in conclusions
	2009-01-29	UoH: Added introduction to 3.1.3 and two more references to Autosteve and AutoFMEA (Ricardo) tool
	2009-01-29	KTH: Added Section 4

List of abbreviations

ABS Anti-blocking System

ECU Electronic Control Unit

EDS Electronc emulated differential lock

EMS (Combustion) Engine Management System

ESC Electronic Stability Control

ETC Electronic Throttle Control

HIL Hardware in the loop

RAP Redundancy Allocation Problem

SIL Software in the loop

TCS Traction Control System

TCU Transmission control unit

Table of contents

List of Figures

1 Introduction

The introduction will look at challenges identified in the development of Automotive Embedded systems and how they address the following aspects of development.

- Managing system and design complexity
- Simplifying safety assessment and improving safety
- Achieving successful tradeoffs between quality requirements and cost

A background to how embedded systems have emerged and increased in complexity is presented as it provides a good background to what the challenges emerge from.

1.1 Characteristics of Automotive Embedded Systems

Automotive embedded systems have evolved enormously over the past decades. For example, the first commercial anti-brake locking system (ABS) of Bosch was introduced by Mercedes in 1978. The ABS system improves the braking performance and is today a standard feature in the automotive industry. The system was first presented in 1970 but at that time the available electronics could not cope with the ABS requirements delaying the commercial introduction by 8 years.

To further illustrate the dramatic introduction of computer based embedded control, consider the fact that a Mercedes car in 1986 contained six microprocessors; these were implemented as six stand-alone controllers (in the automotive industry these are referred to as ECUs, standing for Electronic Control Unit). In 1998 a corresponding Mercedes car contained some 60 microprocessor systems, together forming a distributed system including four networks (not to mention in addition some 113 electrical motors!).

In Engine Management Systems µCs has been used to control the engine purely by electronics. This was the first by-Wire system, which also triggered Safety topics. To clarify those topics the E-GAS Arbeitskreis has been founded in (~1997). Members of the E-GAS AK are German OEMs and some Suppliers.

Time table:

- 1986: Electronic Diesel Control BMW 524td
- 1994: SOP of the ETC70 system for BMW 850i 12 cylinder V engine
- 1996: electronic throttle control integrated in one ECU together with EMS at GM
- 1999: Engine management systems for VW and Audi with predecessor of torque structure
- 1999: Torque Structure at Daimler

The torque structure is also important from architectural view: It was one layer performing an overall coordination to do a consistent actuator control, here the throttle, injection and ignition.

The torque structure was mandatory to provide an (external) torque interface, which is used for TCS, EDS, and ESC.

The introduction of computer based embedded control has been driven both from the technical viewpoint, that of improving performance or introducing entirely new functions, and by market demands. At the same time the costs for development of electronics of which the vast part is software development increases dramatically, and today it is reaching about 40% of total costs with a tendency for further increase.

A typical example of an automotive embedded system is shown in Fig. 1, illustrating the electronic architecture of the Volvo XC90. The boxes in the figure represent Electronic Control Units (ECUs) and the lines represent communication networks. The intent of the figure is to show the complexity rather than the details. The maximum configuration contains about 40 ECUs. They are connected mainly by two CAN networks, one for power-train and one for body functionality. From some of the nodes, LIN sub networks are used to connect slave nodes into a subsystem. The other main structure is a MOST ring, connecting the infotainment nodes together, with a gateway to the CAN network for limited data exchange. The different networks illustrate the typical automotive domains, including vehicle dynamics control (left network, characterized by strict real-time requirements and safety related motion control), body electronics (including door control, climate control and instrument cluster), and infotainment and telematics. Through this separation, the more critical power-train functions on the CAN network are protected from possible disturbances from the infotainment system.

The diagnostics access to the entire car is via a single connection to one ECU. The figure shows approximately how the ECUs are placed in various locations in the car. The partitioning of functionality is decided by the location of the sensors and actuators used, but also by the combinations of optional variants that are possible. If a vehicle is sold with only a subset of the full functionality, the amount of physical hardware installed should be limited to the minimum necessary.

Many of the ECUs of a modern vehicle are provided by external suppliers, who work with many different vehicle manufacturers (or OEMs, original equipment manufacturers), providing similar parts. The role of the OEM is thus to provide specifications for the suppliers, so that the component will fit a particular vehicle, and to integrate the components into a product. Traditionally, suppliers have developed physical parts, but in modern vehicles they also provide software. As the computational power of the electronic control units (ECUs) increase, it will be more common to include software from several suppliers in the same nodes.

The current development trends in automotive software call for increasing standardization of the software structure in the nodes. The need to integrate software from different suppliers, supporting dependable real-time execution, and managing changes all call for a well-defined structure. The node architecture (see Figure 1) includes several important parts.

Diagnostic kernels provide an implementation of the diagnostic services that each node must implement to act as a client towards the off-board diagnostic tool. It relies on the communication software to access the networks and on the operating system to schedule diagnostic activities so that it does not interfere with the application functionality. Network communication software provides a layer between the hardware and the application software, so that communication can be described at a high level of abstraction in the application, regardless of the low-level mechanisms employed to send data between the nodes. Real-Time Operating Systems (RTOS) provide services for task scheduling and synchronization.

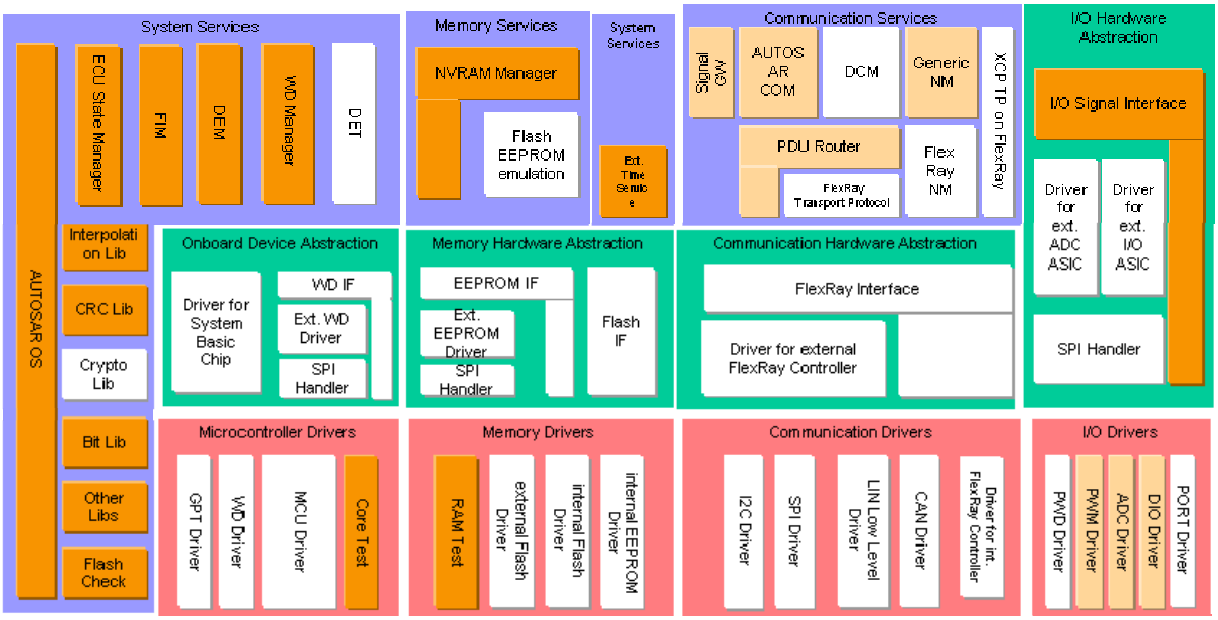

Figure 1 AUTOSAR basic software structure

All these components interact with each other and with the application, and must therefore have standardized interfaces, and at the same time provide the required flexibility. To minimize the use of hardware resources, the components are configurable to only include the parts that are really necessary in each particular instantiation.

For future system development, an important aspect is to create a more flexible software partitioning. The main use for this is probably not to find the optimal partitioning for each car on a given platform, since that would create too much work on the verification side, but to allow parts of the software to be reused from one platform to the next. This puts even higher demands on the node architecture, since the application must be totally independent from the hardware, through a standardized interface that is stable over time. Therefore, further standardization work is pursued within the AUTOSAR initiative [1].

Figure 1 shows the configurable layer of reusable software components making up the basic software in the AUTOSAR specification structure. The remaining parts of software in such a system is a 'signal database' layer called RTE that handles the main transfer of signals either between Application Software components (SW-C) that provide the functionality in such a system or from Basic Software (BSW) to SW-C. This structure enables the transfer of SW-C between ECU's as more strict definition of interfaces is required. It also enables the possibility to evaluate system effects when a SW-C is moved from one ECU to another as the dependencies on BSW can be estimated by the medium of the data transport.

Cars are typically manufactured in volumes in the order of millions per year. To achieve these volumes, and still offer the customer a wide range of choices, the products are built on platforms that contain common technology that has the flexibility to adapt to different kinds of vehicles. As an example, the Volvo XC90, which appeared in 2002, is based on the same platform as four previous Volvos launched since 1998.

Automotive embedded systems are further characterized by the following:

- **Users**
  In contrast to many other advanced machines, such as airplanes and medical devices, automotive products are utilized by all us. This has an important impact on the usability, service and dependability required of the products.
- **Dependability requirements**
  Automotive embedded systems have a fairly long life time and users expect the vehicles to

function over extended periods of time, leading to strict requirements on reliability, availability and maintenance. Automotive control systems are safety related. Not only is the control system required to operate reliably; the design of the system and its context must be carefully analyzed to consider what might go wrong, and what the system should do in such cases. In addition, security is becoming of increasing importance because of the possibilities and relative ease with which embedded control systems behaviour can be modified, e.g. by replacing memories/chips or by network intrusion.

- **Heterogeneity**
  As depicted by **Error! Reference source not found.**, automotive embedded systems are heterogeneous. They handle many different types of tasks with widely varying requirements. For example, the motion control related ECUs include functionality that can be characterized as hybrid systems, being composed of components that are best described by continuous- and discrete-time dynamic systems and finite state machines. Motion control is one central part of ECS. Although it's absolute size e.g. in terms of lines of code typically is relatively small compared to other functionality, the motion control functionality is coming along with real-time constraints, environment dependencies, and safety criticality. To handle this heterogeneity the embedded systems are normally structured into a system platform and applications, each with their own hierarchy in order to facilitate changes and reuse. Responsibilities of the system platform include for example logging, communication services and drivers for sensor readings. For the application there will be activities such as motion control, estimation of the environment state, and human machine communication.

- **Real-time constraints**
  These constraints arise due to interactions with the environment. From control system specifications, for example referring to required speeds of motion, the timing requirements on the embedded control system can be derived. The speed (or bandwidth) of the closed loop system will provide requirements on the timing of the controller, including the sampling periods and delays that can be allowed. These properties can also be taken into account in the control design, however, providing an additional dependency between the controller and its implementation.

- **Resource constrained implementations**
  Automotive embedded systems are often highly resource constrained because of the large series being produced. In such applications, tradeoffs between the system behaviour (quality of service) and the resources required (processing, memory and power) is essential.

- **Distributed systems**
  There has over the last decades been a strong trend to connect standalone controllers by networks, forming distributed systems. Another and closely related trend has been modularization, where for example, an electronic control unit is physically integrated into an engine, forming a sort of mechatronic module. Combining the concepts of networks and mechatronic modules makes it possible to reduce both the cabling and the number of connectors, the result of which is facilitated production and increased reliability. Distributed control systems first appeared in process control, and later in the 80s in aerospace, and in the 90s in the automotive industry. Distributed systems are characterized by the mapping problem, i.e. the need to assign functions to different nodes of a distributed system, to define the tasks of the system, and their implementation in software and/or hardware.

- **Complex development structure**
  Todays electronic systems are developed in a complex design environment. It is rare to find systems that are disconnected from other system in the vehicle. To some extent dynamic control has survived this change but for driver assistance systems the reality is that a system is defined at one organisational entity at the OEM but the actual development is done in parallel at several parts of an OEM organisation and likewise in parallel at several

Tier1 organisations delivering components to the vehicle builds. This puts tough requirements on how design decisions and requirements are organised, updated and distributed. To both give necessary details to all involved parties but also hide component related data to other component suppliers in the system.

- **Co-operative systems**
  An emerging set of systems are systems where the environment or individual vehicles affect the behaviour of other vehicles autonomously. These systems require new methods to model, test, simulate and validate in order to make them reliable in a automotive environment.

- **Tight coupling to the environment**
  The tight coupling between the control system and the controlled process is manifested in several ways. Apart from aspects related to physical integration and protection against a harsh environment, the control system is also fundamentally related to the controlled process. Typically, models of the environment are used in control design. In many cases, the control algorithms are synthesized from a validated model of the controlled system. In other cases, the controller parameters are tuned based on the overall system behaviour. For control systems this creates a dependency between the environment and the control system, creating a kind of contract between these two entities. Another type of environment coupling exists with humans interacting with the embedded control system. A driver “in the loop” is typical for vehicular systems. The situation arises where conflicts can occur – who is deciding the motion of the vehicle at any given point in time? Careful analysis is required and special care has to be given to the human/machine interface.

- **Parallel activities and triggering**
  Since the real world is truly parallel, there is typically a need to describe and handle several parallel activities. A typical control system normally includes both time- and event-triggered activities. In many cases, time-triggering follows naturally from the development of discrete time (sampled data) functions. However, in other cases the controlled process can be inherently event-triggered. This is the case for inherently sampled systems, one example being control of injection in a combustion engine; the point in time of injection depends on the speed and angular position of the engine parts. Event triggered functions thus include those who are inherently sampled and other functions who are not dictated or preferably implemented as periodic activities.

- **Field operational tests**
  To validate functions the norm is to use as many forms of testing as possible. For user functions one common form is to use field-test vehicles where development vehicles are given to external users for prolonged periods. One big problem with these trials is result feedback and data gathering during the trials.

All in all, the use of embedded control systems has paved the way for large improvements of machinery in terms of enhanced performance, flexible tailoring of product variants, and the introduction of completely new functionality such as for active safety control in vehicles. As a consequence product complexity is becoming a crucial issue in system development. Systems integration is today a serious problem in the automotive industry. This increased product complexity calls for more mature engineering approaches including the use of model and component based development.

1.2 Model based development

It is well known that increased system complexity requires increasing abstraction levels for humans to deal with and develop such systems. The essence of model based development has the aim to provide:

- formalized descriptions providing abilities for automated analysis and synthesis

- abstraction capabilities and graphical representations, facilitating communication among system stake-holders

- possibility for reuse of designs, and of analysis and synthesis capabilities

- the basis for improved solution space exploration, verification and validation

Model based development is common practice in mature domains such as mechanical engineering and control engineering, and supported by computer aided engineering tools. Even though the practice is advanced there are still limitations. For example, in control engineering, the formalized descriptions deal primarily with control system behaviour in terms of steady-state and transient behaviour of sets of coupled differential equations. There is little, or much less concern, with issues such as implementation oriented structuring (e.g. how to partition algorithms into software modules) and how the hardware/software implementation will affect the control system behaviour, e.g. due to time-varying end-to-end delays and transient hardware faults.

The situation in software engineering is less mature. For example, there are companies and suppliers in the automotive industry that are certified for SW-CMM [2] level 3 but the maturity of software development in automotive industry has just about reached the lowest two levels of the SW-CMM. The use of software is not new in the automotive industry but the ability to consider networked systems with a proper process is still in its infancy.

Model based development of software is therefore an evolving area with many research efforts in place (see section 3 for an overview).

2 State of practice in E/E system development in the Automotive industry.

This section provides an introductory overview of model based development practices in the automotive industry. The overview is not complete but has the intention to provide representative snapshots to illustrate the widely varying state of practice and industrial needs. The interested reader may refer to for example [4] and [3] for more snapshots.

2.1 E/E system modelling and tool support

In general, development practices vary to a great extent over different companies, and across the automotive domains. This is not surprising because of the heterogeneity of the automotive embedded systems, as described in Section 1.1. While some domains are characterized by model based approaches supporting the development, other domains still mainly rely on written documents for specifications and hand-written code for software development. Although some analysis is possible on the code level, this level is not suitable for communication among developers and does not support early analysis and design. A particular complication and challenge is provided by the differences in traditions and also characteristics among the domains.

An essential facet of the characteristics is the differing models of computation, represented by closed-loop control in vehicle dynamics (continuous and discrete-time dynamics) and logic/state machines in e.g. body electronics (discrete event dynamic systems).

For control systems development in the automotive industry, model based development is in many areas already the standard design approach; however, the adoption and extent varies between different companies and subsystems. CAE tools supporting modelling, simulation and rapid control prototyping (RCP) largely facilitate development even without available mechanics and electronics hardware, and provide means for control system verification and validation, in the lab, as well as in-vehicle [5].

Companies with more mature processes utilize tool chains typically starting from functional design (e.g. using Matlab/Simulink/Stateflow), using rapid prototyping (through code generation and prototyping hardware), software in the loop simulation, production target code generation, and reuse of plant models in hardware in the loop simulation. Less advanced companies still tend to use e.g. Matlab/Simulink, but with no or little connection to the embedded systems implementation.

Model based verification is becoming increasingly used, where one example is the use of hardware in the loop simulation for both subsystem as well as system integration testing. In a hardware-in-the-loop simulator, the computer control system environment (i.e. the vehicle, road, driver actions as well as other relevant environment entities) is simulated in real-time, enabling system testing. With this approach automated testing can verify hundreds of test cases without user interaction and provide reports on the results. For active safety systems it is also possible to try dangerous or destructive scenarios over and over again without risking personnel. There are limitations though, as the sensing system irregularities and other effects where the real-world system would provide non-ideal data is not taken into account.

In order to perform software in the loop and hardware in the loop, for both integration and verification, it is mandatory to model the control path. This needs to be done in close relation to the dedicated functionalities. To do such system spanning tests in an effective way, an architecture definition is needed, e.g. vehicle partitioning.

However there are also control domains, such as automotive engine control, which rely heavily on look-up tables and calibration of systems for control purposes – i.e. with little tradition of model based control. However, look-up tables are also used to avoid high computation load at runtime. The values in the look-up tables are determined offline, for example by using Matlab / Simulink / Stateflow.

With respect to software modelling, the use of the UML has been slowly increasing. However, the collected experience of the consortium is that the use of the UML is not widely spread for

automotive embedded systems. Subsets of UML are used for certain dedicated purposes, mainly involving documentation (e.g. use-cases, MSCs). There are also cases where the use of UML tools is more elaborated including code generation and model level debugging.

The introduction of tool chains supporting model based control engineering is not unproblematic and is strongly related to and affected by organization, process and technology constraints.

Introducing tool chains causes a reliance and dependence on particular tool vendors and requires training of personnel. For OEMs, transitioning from specification of control systems to actually implementing them is a large step and has many implications including maintenance of in-vehicle software and possibly also a larger responsibility. On the other hand, it gives the OEMs better control of vital vehicle functionality.

Many OEMs only specify control systems that are in turn developed by subsystem suppliers (for example; e.g. BMW top models have some 60 ECUs where most are developed by some 30 subsystem suppliers). Increasingly, models such as Simulink diagrams are used in the communication between organizations.

Today, there is consequently a strong need for standardized ways of

- describing automotive embedded systems, to support communication
- managing the information involved in automotive embedded systems development and integrating the disparate sources of information, e.g. represented by UML, Simulink and safety analysis models, capturing different aspects of the system to be developed
- supporting control engineering, software issues and implementation in embedded distributed computer systems
- supporting various types of formal analysis techniques from different disciplines.

2.2 Model based software development.

Model based design (MBD) and automatic code generation are used for software development at vehicle manufacturers. For some years now, OEMS have used modelling to develop vehicle functions. Model-based approaches are systematically applied to the series currently in product development; the number of functions to be integrated is constantly increasing. Some of the functions developed comprise the entire ECU application software.

In each development cycle, the supplier is given the OEMs requirements and test information, that the supplier is responsible for producing software for the model, and implementing it on the ECU. At present, when functions are created via modelling, suppliers still have a high manual workload when integrating them into the ECU [9]. The amount of work involved greatly depends on the software architecture used by each supplier, even when the OEM has specified the communication part of the basic software. In some cases, the software architecture has to be adapted or specially extended. There is no completely standardized software architecture, so sometimes extensive coordination meetings have to be held with specific suppliers.

The need for coordination goes beyond just the software architecture. The OEM and the supplier also have to jointly define the description of metadata for integrating functions, such as the interface list for the functions, and the mapping of application signals to bus signals. Thus, the prerequisites for broader and process-safe use of model-based development are a uniform, supplier independent software architecture, and a standardized description of the metadata.

The AUTOSAR standard [1] defines software architecture for ECUs, an integration method, and the interchange formats that these require. In other words the AUTOSAR standard [1] largely addresses the requirements for the process-safe integration of model-based functions described above. AUTOSAR divides the application software of an ECU into several software components (SWCs), which communicate with one another via middleware (RTE). SWCs encapsulate the software and give it type definitions, allowing data exchange only via well-defined interfaces. Two

mapping steps are needed for integration on an ECU: first the SWC instances are mapped to ECUs, and then the data elements are mapped to network signals for communication across ECUs.

For the development of next generation software platforms, many OEMs are taking the first steps towards AUTOSAR architectures. The introduction process starts with AUTOSAR software architectures and works it way down. This will involve systematically dividing the software architecture into application parts and basic software parts, which will communicate via a defined interface. The base for defining this interface is the AUTOSAR standard. In this first step, the standard software is often based on the established OEMs core, to which selected AUTOSAR software services are added. In this early phase, the ECUs developed in this way will still be network-compatible with ECUs developed by a classic method. This means that AUTOSAR technology very often is introduced step by step.

With AUTOSAR, modelling is often performed at two levels [6]. Beside the behaviour level, where the behaviour of the functions is modelled, there is an architecture level where the interfaces of the SWCs and their connections have to be formally described. In a top-down strategy, when new vehicle functions are developed, it is useful to first subdivide their functionality into several SWCs, and then to define their interfaces at architecture level. The behaviour of the resulting SWCs is modelled. For previously existing function models, a bottom-up procedure can be used to generate the the SWC descriptions from the model interfaces. The resulting SWCs are then connected with one another at architecture level.

Stepwise introduction means that it is not possible to produce a complete top-down design of a whole vehicle. It also means that at the level of single ECUs, not all functions are initially available as models. An iterative strategy has therefore provided useful.

The resulting SWCs are collected together in a composition at architecture level and networked with one another. The remaining unconnected ports are led through to the outside, which turns them into ports in the composition. The ports now represent the communication interface of the ECU. Data elements referenced via the ports can be mapped to the signals specified for the ECU by the communication matrix. This makes it possible to create the SWC structure of an ECU at a reasonable cost.

AUTOSAR is answering a longstanding need to standardize the description formats, and interfaces for model-based function development. The type-safe AUTOSAR descriptions make it possible to ensure consistency between separately developed function models at the very early stage in the development process, when the OEM hand over the function models to the ECU supplier. The expectation is that this will make the function integration by the supplier much more efficient. Coordination meetings held between the OEM and the supplier to discuss software architecture are considerably more productive because both sides can use terms that are standardized by AUTOSAR. The current necessity of using two development tools (for modelling behaviour and for describing interfaces) in developing AUTOSAR compliant function networks poses new challenges, as each system has to be divided into manageable and logically useful software components. In the future, the transition between different modelling tools, often from different vendors, will have to be made more efficient to ensure a “round trip”. Moreover, the current division of AUTOSAR development environments, into tools for architecture modelling and system integration, and tools for behaviour modelling with their own auto coders, is to a large part due to the tool domains. This limits the potential for tool-independent system modelling and resource optimization. Architecture and design decisions therefore need to be thought through carefully at the outset. Because the AUTOSAR descriptions are so extensive, and because individual tools do not yet provide complete equivalents, developers have to begin by deciding on a subset of the standard via suitable application profiles [6].

The definition of the AUTOSAR standard is not yet complete. Close cooperation with the standardization groups, research community and tool manufactures is a necessary to ensure that investments made in converting to AUTOSAR are future-proof.

2.3 Example of state of practice from the Automotive industry.

The activities related to the Safety are directly integrated into the normal RAM(S) tasks, there isn't a specific safety design flow. Usually are applied the classical RAM(S) concepts, coming from functional analysis, for the system reliability.

These analysis are performed especially thanks to the data (feedback) acquired on field.

To perform these types of analysis, the techniques usually used are PHA (Preliminary Hazard Analysis) the FMEA (Failure mode and Effect Analysis) and the FTA (Fault Tree Analysis).Test benches, whether SIL (Software In the loop) or HIL (Hardware In the Loop), are developed to verify the specifications' coverage and to validate the system. Furthermore, fast prototyping instruments and tool chain as Matlab/Simulnk/StateFlow are used to support these kinds of analysis.

The introduction of new standards (ISO 26262, IEC61508,..) is forcing to introduce new formal or semi-formal methods to manage the requirements and specifications, allowing a more controlled approach for system verification and validation.

As mentioned in section 1.1 the state of practice development is as heterogeneous as the different systems or domains. In the different domains, the above mentioned tools are applied. In order to have an efficient internal work flow and in order to reach requested quality standard (like CMMi[2], AutomotiveSpice[6]), several processes are defined and consequently applied. The following statements are valid for the application development.

- **Powertrain:**
  Engine Management systems:
  Function development and architecture development is separated. A System is partitioned into several hundreds modules. Often used modules are grouped into bigger, configurable packages. An EMS consists of several 10s of those packages. By performing such a packaging, the number of interfaces is reduced significantly.
  Transmission control systems:
  Compared to EMS the functional content of TCUs differ quite strongly, therefore a customer specific clustering is done.

- **Chassis:**
  Basic Brake Systems:
  Main focus in the development is on constructive side, but not electronics
  Higher level Brake functions:
  Those functionalities are strongly realized by using electronics and dedicated software. At top level a separation into modules is done.

  Basic Brake and higher level brake functions are developed and verified using SIL and HIL.

  Further chassis systems, like suspension damping control, steering control are developed customer specific.

- **Active Safety Systems**
  These use actuators provided through the Powertrain and Chassis systems. In additions powerful sensor systems enabling functions like Collision Mitigation by Braking are added to the vehicle architecture. As for Chassis systems, Active Safety systems use SIL and HIL during the function development. At VDI's Electrinic in Fahrzeug fair 2007 Audi showcased examples of support architectures for function development. As the basis for active safety functions are control this method is well suited for the purpose. For the sensor system there is more reliance on implementation efficiency and application specific hardware. In some sense this work is generic but it is often necessary to adapt the sensing system functionality for the function flora that uses the sensor system output.

- **Body and Interior:**
  The functional content and also the modelling strongly varies, depending on the customer.

In some projects an autocode generation using TargetLink, based on Simulink/Statemate models is performed.

- **Infotainment systems**
  The functional content is more limited than the above and more standards are available, MOST[10] being one, Bluetooth profiles[11] being another that enables partitioning of systems. Attempts are made to support the verification process by using model based definition of behaviour using UML sequence charts and derive test cases from these. Model based development is limited as the target platforms are typically combinations of DSP and normal processors making low-level MBD difficult.

3 State of the art efforts related to the EAST-ADL2.0 work

3.1 Language concepts

This section describes the current state of the art in the themes guiding the work in the project.

3.1.1 Structure modelling concepts

Model based development for embedded systems, and in particular automotive systems can be supported in various ways. The AADL is a modelling language dedicated to embedded systems with its roots in the aerospace domain. Compared to the EAST-ADL2 and AUTOSAR combination it covers parts of this scope. However, because of its overlap with AUTOSAR on the software architecture level, and the lack of complementary abstraction levels it does not provide an appropriate structural framework for automotive systems development. Also, the support for feature modelling, requirements and variability is unique for EAST-ADL2.

The AUTOSAR approach to structure modelling is based on a component hierarchy with ports and connectors. Ports are typed by an interface which may have several data elements (for data flow) or operations (for client-server). The component hierarchy is clearly defined through a type and prototype pattern. AUTOSAR also solves the problem of referencing specific instances of reused components by means of the InstanceRef construct. For example, if a specific component somewhere in a hierarchy is allocated to a specific ECU, the InstanceRef construct includes the hierarchy path to the component, to avoid that all components of this type are allocated on the same ECU.

SysML and MARTE are UML profiles that augment plain UML with constructs for systems engineering and embedded real-time systems modelling, respectively. Both approaches, and even plain UML are useful tools in automotive development and EAST-ADL2 has integrated some of these concepts, for example requirement concepts from SysML and timing constructs from MARTE. The general structuring approach found in SysML based on blocks and ports is generally highly appropriate and applied in EAST-ADL2.

What is not present in any of these approaches is the concept of abstraction levels and a model structure tailored for automotive use through several lifecycle phases. The EAST-ADL2 structure is a framework that both supports the modelling needs and guides modelling in a way that improves model exchange and understanding between stakeholders.

3.1.2 Behaviour modelling concepts

In EAST-ADL2, behaviour modelling relies on the definition of a set of elementary functions that are executed based on the assumption of synchronous run-to-completion execution (read inputs from ports, compute, and write outputs on ports). This was chosen to enable analysis and behavioural composition and to make the function execution independent of behavioural notations: inside each function, the data transformation can be described according to various languages and paradigms, and various legacy tools including general UML tools and domain-specific tools (e.g. Simulink, ASCET).

Functions own an “ADLBehavior” that is refined in “ExternalBehavior”, when definition is made in external tools (e.g. Simulink, ASCET, etc.) and “NativeBehavior”, when definition is made with pure EAST-ADL2 constructs.

3.1.2.1 Native behaviour:

"NativeBehaviors" are mapped directly to UML "Behaviors" such that "StateMachines", "Activities", or "Interactions" – depicted as sequence diagrams – can be used w.r.t to modelling needs.

An important aspect of the definition of native behaviour is the relationship between behavioural models and composite structures. This was tackle in a pragmatic fashion during ATESST1: the application of EAST-ADL2 stereotypes on UML behavioural concepts alters UML2 semantics such that among other things, triggering policies and run-to-completion assumption hold (see deliverable [12]). A recent paper accounts for this issue in a more general and fundamental fashion [13]. The authors show that very few in-depth studies of UML2 composite structures have been published, resulting in misinterpretations and ambiguities on composition mechanisms and propagation semantics of ports. References of up-to-date research in this field are provided and an alternative approach to the use of stereotypes is advocated: rely on the definition of explicit behaviours on the ports of the composite structures, so that delegation schemes, unexpected requests among other things are properly identified.

3.1.2.2 External Behaviour:

Off-the-shelf tools for behavioural modelling like SCADE, ASCET, Simulink, etc. all support model based development with analysis and synthesis to various degrees. It is not probable that a single tool will be used for an entire vehicle development project, but model integration is necessary. EAST-ADL2 supports this aspect by allowing external representation of behaviour and concepts for integration with requirements management tools.

"ExternalBehaviors" are mapped to the UML "OpaqueBehavior", which features both a language and body attributes (holding references to the type of external tool and language used, e.g. Simulink, ASCET, etc.).

3.1.2.3 Continuous-time systems modelling at different realization levels

Continuous-time behaviour is foremost needed to describe the controlled system, i.e. the plant model. It is also possible that control functions, e.g. PID control, can be described using continuous-time functions at the functional analysis abstraction level.

A typical continuous model of a plant system is described using one or many systems of constraints. Typically the plant has dynamics, which means it can be described using differential equations (DE:s), or differential algebraic equations (DAE:s).

Models of continuous-time systems can, like embedded systems, have different levels of abstraction. Since the expression abstraction level is reserved in EAST-ADL2, they are called realization levels. These realization levels were created to provide answers to such questions as:

- What is the behaviour of Simulink models?
- What is the behaviour of Modelica models?
- How is it possible to connect acausal and causal systems?

3.1.2.3.1 Acausal models

An acausal model is a model where the behaviour is represented as equations, or constraints. The word acausal could be misleading, since it is a negative definition (i.e. models not being causal). An acausal model should rather be seen as a set of equations, or constraints. Constraints can have a specified cause and effect, for example:

```
y = if v > limit then limit else v;
```

Although labeled as an acausal model, there is a causality where a value of v gives a value of y. The following model is of a bouncing ball:

```
der(height) = velocity;
der(velocity) =-g;
when height <= radius then
   reinit(velocity, -c*pre(velocity));
end when
```

The model is acausal in the sense that either of the variables `height`, `velocity` or `g` could be used as input to the model. Another way of expressing this is that the number of equations vs the number of unknowns have to be matched.

Acausal models are often associated with the Modelica language. Modelica combines acausal modelling with object-oriented thinking. For interfaces between physical components, it is favorable to use acausal modelling, together with power port connectors where effort and flow information is interchanged. This way, physical components can be connected to each other, similar to how they are connected physically. Another advantage of using this realization level is that equations can be simplified and even solved analytically.

Two new tools were released this year: Simscape 3.0 from the MathWorks, and MapleSim from Maplesoft. Both these tools are similar to Modelica: combining equation based effort/flow modelling, which enables physical modelling. Also SystemC-AMS and VHDL-AMS uses acausal models, but limited to electric systems.

3.1.2.3.2 Continuous causal models

Continuous causal models are typically used by control engineers, to describe a system and its controller. This abstraction level is also how continuous models are modeled using MATLAB/Simulink block diagrams. There is no one-to-one mapping of an equation to a continuous causal representation, the mapping depends on which variables that are used as input and output, but also if integral or differential causality is chosen. Another continuous causal representation is bond-graphs, which can be transformed into block diagrams [Karnopp2000].

3.1.2.3.3 Discretized models with solver

Discretized models typically have update and output functions, as a function of a time step. The selection of solver is crucial to get a valid simulation result, including consideration of stiff systems, selection of time-step etc. If hybrid models are simulated, the solver also needs to take into account zero-crossing effects. This level of abstraction can be described by the same means as a computer program, e.g. UML activity diagrams, state machines, c-code.

3.1.2.3.4 Discretized models with solver and platform implementation

Especially for real-time Hardware-in-the-loop simulations, it is crucial that the simulation can be run in real-time. The platform can have a limited numerical resolution, have memory constraints, etc. In a real-time system, the calculation time needs to be taken into account, and possibly scheduled.

3.1.2.4 Current status of EAST-ADL2 continuous-time behaviour

It is possible to define continuous causal ADL_funtions, by setting the attribute `is_Discrete` to false. A means to describe Acausal models is not available, and could be seen as a target for ATESST2. Discretized models should be possible to describe using EAST-ADL2 behaviour notation, this is however yet to be verified.

3.1.3 Safety modelling and analysis

The first part of this section contains a comparative, critical review of the SOTA in model-based safety analysis, a relatively new field of research that has dynamic grown and yielded a plethora of new results over that last fifteen years. The review discusses important results and identifies techniques that could provide a basis for the definition of appropriate safety/error modelling & analysis concepts in EAST-ADL2. The second part of this section focuses on ISO 26262, the emerging automotive safety standard. The standard is likely to influence developments in the sector and should, therefore, be also considered towards the definition of relevant concepts in EAST-ADL.

3.1.3.1 Model-Based Safety Analysis

Model-based safety analysis is an increasingly important technique in the design of safety-critical systems. This is particularly true of Real-Time Embedded Systems (RTES), which are growing ever more complex and which are increasingly distributed across a networked architecture or incorporated into cooperative systems. Often RTES are used in order to improve the safety of a system; for example, Electronic Stability Control in vehicles have been shown to be effective in maintaining control and saving lives by significantly reducing the number and severity of crashes [14]. Because RTES are widely used in safety-critical industries such as the automotive and aerospace industries, it is vital to be able to perform a thorough and accurate safety analysis of those systems to ensure they meet their dependability requirements. By identifying areas in a system where reliability or safety is deficient, actions can be taken to remedy the weaknesses and thereby improve the design of the system.

Motivation for Model-Based Safety Analysis. Traditional safety analysis has typically operated on an informal understanding of the system design. Such techniques include fault tree analysis (FTA) [15][16], in which the combinations of possible causes are deduced from the system failure, and Failure Modes & Effects Analysis (FMEA) [17], which analyses the possible effects each failure can have on the system. Techniques such as these are often carried out manually, either by a single person or a team of engineers, in order to produce comprehensive documents to fulfil safety requirements and to devise strategies to mitigate the effects of failure [18]. Although a great deal of valuable knowledge about the safety and reliability of the system is gained in the process, this type of informal, ad-hoc approach has a number of drawbacks.

Firstly, because the analysis takes place using informal knowledge of the failure behaviour of the system, the safety analysis is stored separately from the knowledge of the structure of the system, which is typically modelled more formally, and this can result in discrepancies or inconsistencies. Secondly, the primarily manual nature of the analysis process increases the risk of introducing errors or producing an incomplete analysis, particularly as the systems in question grow more intricate. Furthermore, a manual analysis is usually much more difficult and expensive, meaning that it is rarely carried out more than once and often only at the end of the design process to ensure that the design meets safety requirements, despite the potential benefits that multiple safety analyses can yield when used as part of an iterative design process. Finally, the informal nature of the results of such ad-hoc analysis makes it difficult to reuse that information, whether in a future iteration of the same system or in the design of a new system, particularly because the safety information is stored mainly within the results of separate analyses and is therefore separated from the system design itself.

In model-based safety analysis (MBSA), by contrast, the safety analysts and the system designers use the same model of the system, or at least models which are closely linked in some way, and this has a number of important benefits. Firstly, the resulting model is often more formal than a separate safety analysis and this can introduce the possibility of automating part of the process of safety analysis, e.g. by automatically generating fault trees from the system model or by simulating the failure behaviour of the system by injecting faults into the model. This not only simplifies the process, it also saves time and more importantly enables the safety analysis to be used as part of an iterative design process because new results can more easily be generated once the model has been changed. The more structured nature of the modelling also reduces the probability of

introducing errors or of omitting important detail since the safety information is linked to the structural/nominal model of the system. Finally, a model-based safety analysis is often much more suitable for reusing in other projects, because the safety information is packaged with the system model, usually at a component-level, making it possible to reuse parts of the system without necessarily needing to perform another separate safety analysis.

A considerable number of different MBSA approaches have been developed, and some of the more prominent examples will be discussed below. Model-based safety analysis and verification has also been investigated in a number of other recent projects, including ISSAC and its predecessor ESACS in the aerospace industries (where the goal was to develop a formal methodology and tools for the safety analysis of complex aeronautical systems), the ASSERT project with similar goals but more focused on software intensive systems specified in AADL, the SETTA project (focusing on the use of time-triggered architectures in automotive systems), and the SAFEDOR project (which aimed to develop new practices for the safety assessment of maritime systems).

Modelling in MBSA. As previously mentioned, one way of simplifying the process and of enabling useful iterations of safety assessment is to automate (or partly automate) the analysis process. In industry, there are established software tools that automate calculations on manually constructed fault trees or assist clerical tasks in essentially manual FMEA processes (e.g. Isograph's *FaultTree*+ tool). However, the synthesis of predictive system failure models such as fault trees and FMEAs remains manual. Over the last 15 years, research has focused on further simplifying safety assessment by automating the synthesis process. This work has followed two different paradigms, each defining a distinct way of synthesising system failure models from other system information. The first paradigm can be called *compositional failure analysis* while the second *behavioural fault simulation* [19][20]. Compositional techniques are usually *deductive* in nature, i.e. safety analysis proceeds from system failures to determine the causes of such failures), while behavioural simulation techniques are typically *inductive* in nature, i.e. the analysis moves forwards from known causes to determine the effect of such causes in the system). There is also a separate paradigm for the diagnosis of faults in active systems based on models of those systems (rather than synthesis of failure analyses from system design models) known as *model-based diagnostics*. A more thorough comparison of the various techniques can be found in "Towards a practicable process for automated safety analysis" [21].

Compositional Safety Analysis approaches

In compositional failure analysis, system failure models are constructed from component failure models using a process of composition. System failure models are, or can be automatically translated to, well known dependability evaluation models including fault trees, stochastic Petri-nets and Markov chains. Techniques that follow the compositional approach include: Failure Propagation and Transformation Notation (FPTN) [22], Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [23], Component Fault Trees (CFT) [24], State-Event Fault Trees (SEFT) [25] and Failure Propagation and Transformation Calculus (FPTC) [26]. The Error Model Annex of the AADL also falls into this category as components are hierarchically annotated with state-based failure information, which can then be subsequently analysed by tools [27].

It is important to note that in these approaches, the system failure model is not always merged with the nominal model of the system. In AADL, for example, the failure (or error model) model extends the nominal model of the system and has references to the hierarchy and components in that model; similarly, in HiP-HOPS, the failure model and nominal model are closely connected, as failure data is embedded in annotations to the system components and can make semantic references to system properties. In other techniques such as FPTN, CFT, SEFT and EAST-ADL2, the error model and the nominal model are separate models which are more loosely connected.

Behavioural Fault Simulation approaches

In behavioural fault simulation, system failure models equivalent to an FMEA are produced by injecting faults into executable formal or semi-formal specifications of a system, thereby establishing the system-level effects of faults. A modelling tool is typically expected to do the

analysis automatically. The modelling tool can be a simulator containing libraries of components and their faults, in which case it is typically useful in a particular domain (e.g. simulation of electrical diagrams, piping and instrumentation diagrams etc). However, the modelling tool can also be more generic, e.g. enabling representation and formal analysis of systems represented as state-automata. In this case, the tool can be applied in many domains. Techniques that follow this later approach include safety analysis using formal techniques such as Altarica [28], FSAP-NuSMV [29], Software Deviation Analysis [30], DCCA [31], as well as fault simulators such as AutoSteve [UOH_01] and AutoFMEA by Ricardo [UOH 02].

Model-based Diagnostics

Fault diagnosis is the process of inferring the causes of system failure from their observable symptoms. In model-based diagnosis this is partly achieved by reasoning on an executable model of the system [44]. Typically, in a real-time diagnosis scenario, the state predicted by the executable model is compared to observations of state variables of the system that can be monitored using sensors. Any discrepancy between the actual and predicted system state indicates a symptom of failure which is further investigated to identify the causes of that failure. The process involves identifying and progressively eliminating sets of candidate causes by performing more comparisons between predicted and actual states. Fault diagnosis is a process that can be carried out either in real-time or off-line following an indication that a failure is present. Running an executable model and performing the required inferences is often infeasible in real time, so simpler diagnostic models are often used including fault trees or decision trees. In several approaches, such diagnostic trees are automatically or semi-automatically constructed from design models of a system [43][35].

There are a number of model-based diagnostics tools that have been developed in recent years, such as RAZ'R [32], MDS [33], and RODON [34]. Although many such tools employ existing programming or modelling languages to represent a predictive model of behaviour or a simplified logical model of fault propagation, others use custom-designed languages such as Rodelica [35] which is a declarative equation-based language derived from Modelica.

Modelling Primitives for Error Modelling. Although the model-based safety analysis and diagnostic techniques mentioned above share common goals, they do not necessarily share a common approach. One of the primary distinctions between these techniques is in the way they choose to model failures. These broadly fall into one of three categories: fault propagation-based, state-based, and equation-based.

Fault Propagation based approaches

In these approaches, the failure behaviour of the system is modelled as a propagation of failure from one component to the next until it becomes a system-level failure. The propagation typically follows lines of communication or other connections between components, e.g. dataflow, fluidic, energy connections etc. Failures can also propagate by other means, e.g. by the physical proximity of components. The failure propagation itself is typically described using some form of logic, e.g. in the case of FPTN, FPTC, CFT and HiP-HOPS, this is done with Boolean logic, relating output failures to a combination of input and internal failures. More complex forms of propagation can be described with more complex logic, e.g. a propagation of failure that is dependent on the sequence of failure can be described in HiP-HOPS using Pandora [36], a form of temporal logic.

State-based approaches

Another way of describing the failure behaviour of a system is to use states. In these approaches, the occurrence of a failure is modelled as the transition of the system from a nominal state into a failed state. This is the approach employed by AADL and Altarica, for example. In AADL, the error model is implemented as a form of stochastic automaton; error and repair events are defined, along with possible error states and input / output events (to support propagation to and from other components), and then transitions are defined to show how events can cause the system or component to transit from one state to another. Properties can also be defined for error states and this mechanism allows the addition of quantitative failure data such as failure and repair rates.

Equation-based approaches

A third possibility is to represent the occurrence of failures using equations. This is the approach used in Rodelica, the model-based diagnostics language based on Modelica (also equation-based). Rodelica provides semantics for failure modes and intervals, allowing for the specification of component-level failure data. Other equation-based languages include gProms [37], VHDL-AMS [38], and Ascend [39].

Critique of MBSA approaches. The number of different types of approaches (e.g. compositional vs behavioural, state-based vs propagation-based) is due in large part to the different *goals* of these approaches. Some techniques are geared towards off-line safety analysis and diagnosis, like Rodelica and the other model-based diagnostics approaches; others, like AADL's error model annex and HiP-HOPS, are intended primarily as design aids to help achieve safety requirements or, like Altarica, DCCA, and FSAP/NuSMV, as formal methods for verification of system safety using model checking. Some techniques also offer additional capabilities, e.g. HiP-HOPS provides capabilities for multi-objective design optimisation enabling evaluation of candidate designs which are automatically generated by genetic algorithms [40]. Therefore, any commentary on these techniques has to take into account the goal the technique was designed to achieve.

Different approaches also offer different levels of automation. Although most techniques offer *some* form of automated safety analysis, the scope and performance of this automation varies considerably. For example, behavioural fault simulation approaches like Altarica, DCCA, and FSAP-NuSMV in theory offer more complete automation than compositional safety analysis techniques because they base the simulation on a full behavioural model of the system; because a simulator of model-checker is expected to perform most of the assessment, they require less additional modelling by the designer to handle errors. They also tend to require a simpler form of component failure modelling (typically only internal failure modes are considered). However, the effort required for behavioural system modelling – especially formal modelling – should not be underestimated.

This higher degree of automation does come with a price, however: a higher computational cost than compositional safety analysis techniques, which typically employ algorithms of lower complexity. Most behavioural simulation techniques are also inductive, i.e. the assessment proceeds from known causes to unknown effects, and in this type of analysis, the effective assessment of combinations of causes is at best very difficult and at worst impossible to achieve due to combinatorial explosion. Assuming that there are n possible component failures in a system, assessment of combinations of m of those failures requires that the analysis is repeated $n!/((n-m)!*m!)$ times. For a system that has 1000 failure modes, assessment of the effects of combinations of 2 failure modes requires that the analysis is repeated approximately half a million times. Although this number can be reduced by carefully exploiting assumptions of independence and by taking advantage of the monotonic properties of failure in coherent failure scenarios, the problem of combinatorial explosion still persists. In deductive approaches, e.g. fault tree approaches like Component Fault Trees and HiP-HOPS, the analysis of propagation of failures is deductive (from effects to causes) and therefore not as prone to combinatorial explosion. Fault trees are synthesised in linear time and this time is not determined by the highest order cutset (i.e. the maximum number of failure modes considered in combination, which is defined only by the positioning and nesting of AND gates in the error propagation model). In the case of HiP-HOPS, this has enabled not only application of the technique to large systems but also its combination with computationally greedy heuristics such as genetic algorithms for the purpose of architectural optimisation with respect to dependability and cost [40].

Another problem with formal techniques is that they typically define their own language for nominal *and* failure modelling, meaning it is not always fully compatible with other widely used design languages and tools. On the other hand, some approaches, e.g. HiP-HOPS, focus only on failure modelling and can easily complement design languages that focus on descriptions of nominal behaviour. As a result, HiP-HOPS has been used with a number of different tools and modelling languages in the past, including EAST-ADL, Matlab Simulink, and SimulationX (which is based on Modelica).

Many formal techniques also tend to focus on functional safety analysis only, whereas other approaches such as HiP-HOPS and AADL offer capabilities for probabilistic analysis (e.g. Poisson, Binomial and Weibull calculation models) and in the case of HiP-HOPS, capabilities for common cause and zonal analyses as well. Probabilistic analysis not only enables long term system reliability and availability prediction, it is also important for software safety. Clearly, there is often a need in software design to consider the probability of failure of components, otherwise it is practically impossible to decide on appropriate techniques to ensure data integrity, fault detection and fault tolerance. If controls rely on certain inputs, for example, we need to know the level of integrity with which these inputs are provided by sensors and therefore the failure modes of those sensors and their probability of occurrence.

Finally, some techniques are more supportive of reuse than others. Although most allow reuse in some form, some techniques provide dedicated support for reuse of failure data, e.g. the introduction of inheritable composable specifications of failure patterns in HiP-HOPS [41] or the AADL's support for the storage and reuse of error information in a library (including capabilities for adapting and overwriting information in specific situations) [42].

In summary, the various model-based analysis approaches discussed above have different strengths and benefits, e.g. the high degree of automation possible with formal behavioural simulation techniques versus the higher performance, more scalable algorithms available with deductive compositional techniques. These two paradigms must therefore be both supported by EAST-ADL2. HiP-HOPS provides a good example of a compositional technique and offers good capabilities for safety analysis, however it lacks capabilities for formal verification. For this reason, the state-based error model annex of AADL could also prove useful as an input to the definition of the error model of EAST-ADL2. Finally, the capabilities of Rodelica as a means of facilitating modelling for fault diagnosis in EAST-ADL2 will also need to be further explored.

3.1.3.2 ISO°26262 adoption

The introduction of new functionalities, with an impact on the vehicle stability or handling, could cause a higher hazard level in case of malfunction or failure.

The new functionalities have to be considered “safety relevant” and potentially “safety critical”; it means that, in case of faults, these functionalities could have a significant impact on the system behaviour.

As a consequence, new functionality in the area of driver assistance, vehicle dynamics control, active and passive safety systems, increasingly touches the domain of safety engineering. Future development and integration of these functionalities will further increase the need of:

- New technologies to enable such systems to function more effectively with increased design complexity and managing the safety;

- Safe system development processes within the possibility to show evidence that all reasonable safety objectives are met (acceptable risk reached). This highlights the role of staff engaged in the design, development and maintenance of these safety-related systems. The achievement of sufficiently low levels of risk is critically dependent on individual and team competence

With the trend of increasing complexity, software content and mechatronic implementation, there are increasing risks from systematic failures and E/E random hardware failures, often rooted in management and engineering processes.

The new International Standard ISO°26262 includes guidance to avoid these risks by providing appropriate requirements and processes.

This International Standard is the adaptation of IEC°61508 to comply with needs specific to the application sector of E/E systems within road vehicles. This adaptation applies to all activities

during the safety lifecycle of safety-related systems comprised of electrical, electronic, and software elements that provide safety-related functions.

System safety is achieved through a number of safety measures, which are implemented in a variety of technologies (for example: mechanical, hydraulic, pneumatic, electrical, electronic, programmable electronic etc). While this International Standard is concerned with E/E systems, it may also provide a framework within which safety-related systems based on other technologies may be considered.

The ISO°26262:

- Provides an automotive safety lifecycle (management, development, production, operation, service, decommissioning) and supports tailoring the necessary activities during these lifecycle phases;

- Provides an automotive specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs);

- Uses ASILs for specifying the item's necessary safety requirements for achieving an acceptable residual risk; and

- Provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety being achieved.

Functional safety is influenced by the development process (including such activities as requirements specification, design, implementation, integration, verification, validation and configuration), the production and service processes and by the management processes.

Safety issues are intertwined with common function-oriented and quality-oriented development activities and work products. The ISO°26262 addresses the safety-related aspects of the development activities and work products and it’s applicable to:

- E/E systems installed in road vehicles
- Interaction among these systems and vehicle systems
- Malfunction related to safety critical systems
- Foreseeable operational errors
- Foreseeable misuse
- Foreseeable maintenance malfunctions
- All lifecycle
- New systems only: on the market after the publication of the International Standard
- Series production road vehicles

This International Standard is based upon a V-Model as a reference process model for the different phases of product development; It consists of the following parts, under the general title Road vehicles — Functional safety:

Part 1: Glossary

Part 2: Management of functional safety

Part 3: Concept phase

Part 4: Product development: system level

Part 5: Product development: hardware level

Part 6: Product development: software level

Part 7: Production and operation

Part 8: Supporting processes

Part 9: ASIL-oriented and safety-oriented analyses

The shaded "V"s represent the relations between Parts 3, 4, 5, 6 and 7 of this International Standard.

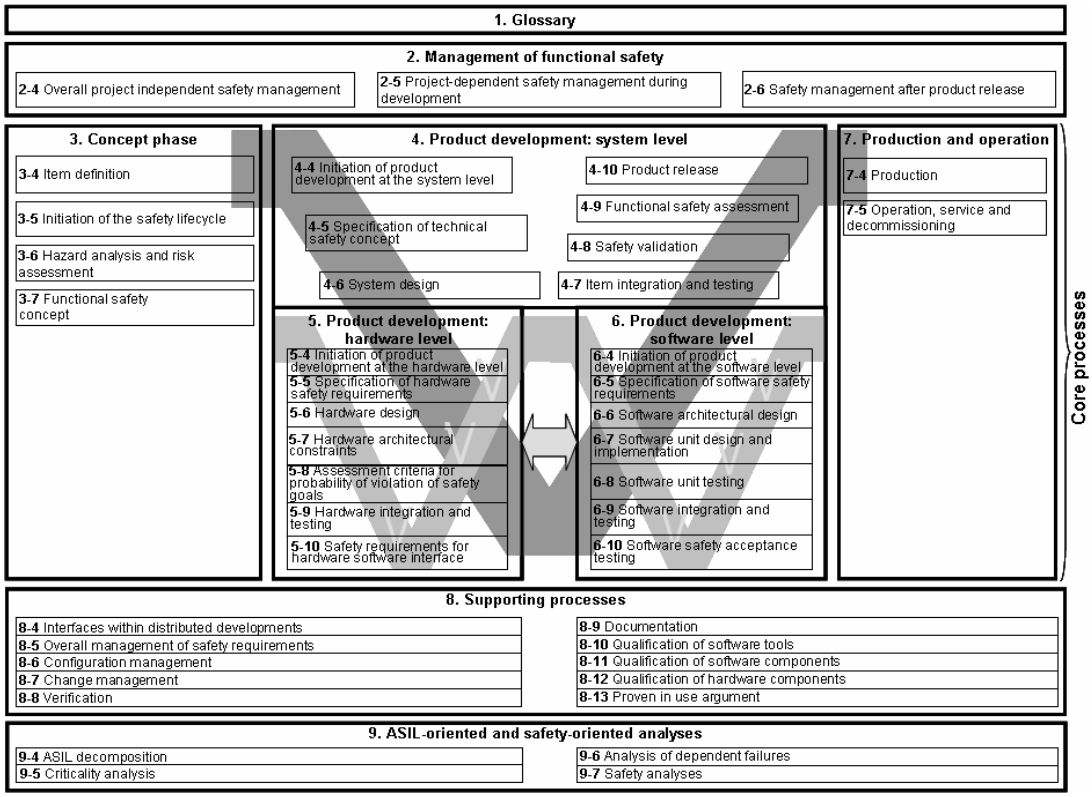

Figure 2 ISO26262 standard structure.

Part 3 of the International Standard specifies the requirements on the concept phase for automotive application. This phase, under road vehicles manufacturers' responsibilities, includes the item definition, the initiation of safety life cycle, the hazard analysis and risk assessment (ASIL determination, Safety goals and Safe states definition) and safety requirements definition.

Part 4 specifies the requirements on product development at the system level. This phase, under road vehicle manufacturers' responsibilities, includes the safety technical specification, the system design integration and testing, the safety validation and assessment, the product release.

Part 5 specifies the requirements on product development at the hardware level (safety technical design). This phase is primarily the responsibility of component suppliers. Because the manufacturer has the overall responsibility for meeting safety requirements, the evidence for safety has to be provided to the vehicle manufacturer with an appropriate level of detail.

Part 6 specifies the requirements on product development at the software level for automotive applications (safety technical design). As with part 5, this phase is primarily the responsibility of component suppliers. Because the manufacturer has the overall responsibility for meeting safety requirements, the evidence for safety has to be provided to the vehicle manufacturer with an appropriate level of detail.

Part 7 specifies the requirements on production as well as operation, service and decommissioning. This phase is under the responsibility of the vehicle manufacturer.

3.1.4 Product family and variability modelling

Over the past 10 to 15 years, software product families or software product lines have drawn increasing attention from software science and practice. The fundamental principle of product line

oriented development is to no longer develop the individual products of a software manufacturer independently from one another and in parallel, but as instances of an embracing product line infrastructure, which are then derived from this infrastructure by way of configuration and, if required, manual adaptation. Thereby, the focus of development shifts from the individual products to the set of products the manufacturer has on offer, i.e. his product line, thus turning this product line into a tangible entity of development.

In addition to the state-of-the-art investigated during ATESST1 and documented in the related ATESST1 deliverables, an important new related research activity came up recently: the addition of variability concepts to AUTOSAR was considered by the AUTOSAR consortium. At the time of writing, however, no details on these efforts are published yet and they can therefore not be documented in detail here.

Furthermore, many other contributions were published by the research community, esp. in the form of conference or journal papers, which may be of interest to ATESST2. A detailed account of these smaller publications is beyond the scope of this overview deliverable; also it is not possible to evaluate their relevance for EAST-ADL2 without a concrete research question in mind. Therefore, they will be investigated in more detail within WT3.3 in relation to individual EAST-ADL2 extensions developed there.

3.1.5 Analysis-Driven Architecture Evaluation and Optimization

This section is divided into two parts, the first part discusses modelling requirements to support analysis and the second focusing more on modelling support required to perform optimization on the described architecture.

3.1.5.1 Architecture Design and Analysis Integration

While the maturity of analysis techniques[1] has led to a set of well established mathematical formalisms in software engineering, such as for example Fault Tree Analysis (FTA), Rate Monotonic Analysis (RMA) and further extensions, Petri nets, queuing theory, and timed automata, their widespread use with complex industrial systems and into integrated tool environments still remains largely open. Analysis is a difficult and time-consuming task, and to save time, many industries either forgo it until absolutely necessary or train their designers to perform preliminary analysis. However, most designers are under-trained in analysis and too busy to perform useful analysis.

In order to perform these analyses, the design representation must be first transformed into a formalism that admits a form of mathematical evaluation. This formalism is referred here as “analysis model”. Analysis tools accept as inputs these analysis models and evaluate them mathematically to produce results which are then used to successively refine the design models. Although both the design and the analysis models are views of the same system, they describe it at different levels of semantic abstraction. Obtaining the latter from the first is generally a difficult task. Hence, design-analysis integration often turns out to be a difficult proposition. For instance, current industrial tools for timing analysis, such as for example SymTA/S from Symtavision, TriPacific’s RapidRMA, Livedevices’ Real-Time Architect, CoMET from VaST and Vector’s CANAlyzer, and academic ones such as MAST and TIMES, provide very strong solutions for analyzing real-time properties, but each does so with different system representation formalisms.

[1] In this document, analysis refers to some kind of engineering/quantitative analysis that uses mathematical techniques to study certain quality attributes of the system. They include stress, thermal or fluid analysis in mechanical engineering, and performance, safety or reliability analysis in software engineering.

In addition, due to the large syntactic and semantic gap between design and analysis representations, some design information must undergo significant simplification (e.g., behaviour model transformation) or refinement (e.g., addition of the underlying OS services model of a system) before being fed into the analysis models understood by analysis applications. This is usually a tedious, slow, and error-prone process that characterizes the infamous “islands of automation”.

The software engineering community has invested special efforts in incorporating the abilities to specify analytical constructs and quality properties with enough expressive power, while still preserving the modelling abstraction level used by system architects. Important research work has been carried out in order to provide modelling languages (e.g., UML, SDL, AADL) with clear and well-formed semantic to support analysis (a summary is proposed in [54]). The ultimate goal is to enable designers to perform analysis directly from their architecture design tools (by calling analysis tools via specific user interfaces) thus reducing the time required to prepare a model for analysis.

However, most of the current work is characterized by providing monolithic and particular solutions for specific analysis techniques, limiting the capacity of reuse, composition, and interoperability between heterogeneous approaches. These aspects reduce enormously their use in real complex embedded systems, which are intrinsically heterogeneous. In most of cases, no single methods can support the analysis of entire systems. In addition, current trends in the model-based engineering community focus more on how to represent analyzable systems, than on how to build these representations in a global design flow, and to merge analysis results in the global system solution. A fundamental shortcoming in current model-based analysis research is the inability to capture decision-related knowledge and the context in which the analysis results are applied [50].

This raises the need of more powerful model-based analysis support and flexible modelling mechanisms to drive design from techniques such as design space exploration and sensitivity analysis.

Design Space Exploration. In order to be able to make right system platform selections, the feasibility of candidate application-platform bindings need to be predicted w.r.t. different non-functional requirements and constraints. Design space exploration assists designers to efficiently decide among candidate implementation alternatives especially when the space of possible solutions is large. The decisions are usually located according to several design goals, and the alternatives therefore represent a multi-criteria decision problem [51]. In an integrated model-based process of real-time systems, these criteria should be typically distributed in different model views which provide specific quality constraints and predictions –typically, deadlines and resource capacity vs. response times and slacks- obtained from various information sources, such as safety analysis, performance simulation, or scheduling analysis.

Furthermore, in a scenario for efficiently analyzing systems, there is the variety of aspects that need to be analyzed and also the number of solution methods that can be performed (and their information requirements). Regarding the problem of dependability versus cost optimization, Papadopoulos and Grante [62] propose a multi-objective optimization approach that uses Pareto frontiers and genetic algorithms to explore optimal tradeoffs between dependability attributes and cost [55] (see also section 3.1.6.2). In the timing analysis field, Racu et al [CEA 9] proposed an approach based on genetic algorithms to optimize power consumption parameters while meeting timing requirements. This is implemented in the SymTA/S tool.

Sensitivity Analysis. It cannot be expected that all non-functional information required for analysis is fully available up front. Instead, designers must work with incomplete specifications, early property estimates, or even flexible constraints that must be balanced with other parameters. Sensitivity analysis is an approach to deal with those design uncertainties, i.e., how far a particular parameter can be changed without affecting the feasibility of the system. In general, sensitivity analysis shows how “sensitive” are the overall results to a given parameter. It allows the system designer to assess the system-level impact of changes in quality properties of individual hardware and software components. For example, variations in the implementation of different application

parts, functional extensions, or changes of timing at subsystem or system interfaces are issues that can turn a previously conforming system into one that violates non-functional constraints.

Particularly for scheduling-aware analysis techniques, there is a number of works providing search algorithms to one-dimensional and multidimensional (simultaneous variations of a set of parameters) sensitivity analysis [53]. Although the sensitivity analysis process may be fully delegated to analysis tools, there is an important gap in modelling languages for addressing the correct parameterization (e.g., determination of dependent, independent, and critical parameters) and solution selection from a modelling viewpoint.

Preliminary Conclusions, Open Issues & Way Forward Towards Analysis Integration in EAST-ADL2. From a modelling language viewpoint, some aspects need more investigation. For instance, current modelling solutions for describing single-point "allocations" as proposed by SysML or MARTE lack important features required for analysis. Suppose that we link an application model with a platform model (application-platform allocation) in the context of a given analysis scenario. The non-functional annotations involved in the application model (e.g., execution times, memory usage) and the platform model (e.g., resource utilization) will get a specific set of values. The SysML and MARTE allocations are specified by a unique relationship between the source and the target models. The first question that arises is, "what happens if we specify multiple allocation cases?", for example, in order to allocate the same application in different available platforms. We actually do not have means to declare multiple versions of non-functional values on the same models. We can formalize this limitation as follows:

> *"Given various allocation alternatives of application/platform models, how to specify the different property value versions (resulting from the allocation) annotated in the internal modelling parts of the allocated application/platform models?"*

Furthermore, different model analysis methods focus on different aspects of the model. Building heterogeneous models of analysis contexts therefore implies to integrate analysis results from different views and/or subsystems and to calculate global predictions.

> *"How can we express the relationships between different analysis methods to calculate global quality parameters (derived predictions, optimization objective functions, measures of effectiveness)?"*

On the other hand, although the sensitivity analysis process may be fully delegated to analysis tools, there is an important gap in modelling languages for addressing the correct parameterization (e.g., determination of dependent, independent, and critical parameters) and solution selection from a modelling viewpoint:

> *"How can we drive sensitivity analysis tools by properly qualifying parameters in design models?"*

In [50], the authors have used the basic MARTE mechanisms to specify non-functional variables to compose multiple quantitative scenarios that are further evaluated to make efficient design decisions. An extension of this approach in ATESST-2 could provide a basis for analysis integration in the context of the various analyses that are possible using the framework of EAST-ADL2 models.

3.1.5.2 Optimisation

Motivation for Optimisation. A particularly challenging aspect of a multi-objective analysis of a complex system under design that was briefly mentioned in the preceding section is that of design exploration. Embedded systems are characterised by sharing of information and hardware resources which means that large numbers of configuration and reconfiguration options are available not only at design time but also at run time, due to the use of shared processors and communication channels. When the functions of a system can be delivered by multiple different configurations, designers are faced with hard optimisation problems, especially when the design space is very large (as is typical for complex systems).

The dependability of individual configurations can of course be determined using safety analysis and verification techniques such as fault tree analysis and model checking. However, satisfying dependability requirements with optimal use of resources and minimal costs requires additional technological support in the form of a global optimisation process. Where it is possible to fulfil all dependability requirements within economical and technical bounds, the architecture that has minimal costs is the optimisation goal. If all dependability requirements cannot be met with acceptable costs, then the problem becomes one of finding architectures that achieve optimal trade-offs between dependability and cost.

It is widely accepted that the various formulations of the above represent hard, combinatorial multi-objective optimisation problems that can only be approached systematically with the aid of optimisation techniques and computerised algorithms that can effectively search for optimal solutions in large potential design spaces.

As an example problem, one mechanism that designers have for altering the safety and reliability characteristics of systems is the substitution of single components with fault tolerant schemes that incorporate redundant components and subsystems. The technique has been shown to work well in combination with measures that ensure the diversity of replicated (hardware or software) components and minimise the possibility of common cause failure. However, the decision on the optimal location and level of replication of components is a non-trivial task and this is especially true as systems increase in size and complexity. This problem of maximising reliability via such replication within given cost and other (e.g. weight) constraints has come to be known as the Redundancy Allocation Problem (RAP).

One of the objectives of the ATESST2 project is therefore to develop a novel approach to multi-objective evaluation and optimisation of systems, particularly with regard to safety and reliability. Such an approach would be generic and could include cost and any quality attribute (such as safety, reliability, availability, performance etc), albeit with two constraints: firstly, there has to be a technique capable of assessing the quality of the model in terms of the chosen attribute (e.g. in the case of safety, it must be possible to evaluate the safety of the system); secondly, there must be automatic transformations that can be applied to the model to alter the chosen attribute (e.g. in the case of reliability, one option would be the possibility of replicating a component). The goal of the approach would be to find designs that represent optimal or near-optimal tradeoffs amongst the chosen attributes and cost for a complex system. The potential benefits from such an optimisation technique are substantial and include the automation of complex evaluations and the establishment of a transparent, mathematical basis for achieving successful tradeoffs among quality and cost in the design of complex systems.

Approaches to Optimisation. Optimisation problems like the RAP have been addressed by numerous researchers. A review of early work on the RAP can be found in Frank et al (1977) [56] and Chen (1992) [57] has shown that the problem is NP-hard. Fyffe et al (1968) [58] used a dynamic programming algorithm to solve RAP in a series of 14 k-out-of-n subsystems and it was also solved using integer programming by Ghare and Taylor (1969) [59]. Nakagawa and Miyazaki (1981) [60] used a surrogate constraints approach to solve 33 variations of Fyffe's problem which have since become a benchmark for optimisation methods seeking to solve the RAP.

There are a number of heuristics that can be employed to perform this sort of optimisation, including simulated annealing, Tabu search, genetic and ant system algorithms; all of these have, for example, been applied to the RAP. Coit and Smith (1996) [61] have used a genetic algorithm (GA) to solve the 33 variations of RAP and have demonstrated results that improve those reported by application of exact mathematical methods. Kulturel-Konak et al (2003) [62] have used Tabu search which performs a guided neighbourhood search and is individual based rather than population based as with a GA. A limitation of all of these methods is the use of constraints in place of true multi-objective search. A single optimal solution is typically sought which maximises reliability within certain cost and weight constraints. However, in practice, designers are often interested in examining several optimal or near-optimal solutions that provide different tradeoffs among the parameters of the optimisation. To enable this type of optimisation, Kulturel-Konak et al (2005) [63], Papadopoulos and Grante (2005) [64], and Grunske (2006) [65] have all proposed

multi-objective heuristic approaches that generate Pareto frontiers of non-dominated solutions providing such tradeoffs.

Though there have been advances in solving RAP, one important limitation and a constant feature of all earlier approaches is the use of reliability block diagrams (RBDs) for reliability evaluation. RBDs have been extensively used because they fit the simplified modelling assumptions of series-parallel systems that have dominated RAP literature. Indeed, representation of failure behaviour in RBDs assumes that the system is formed in a series-parallel configuration of components, and either works or fails in a single failure mode which typically suggests complete loss of system function. A parallel configuration typically fails if all (or the majority of) constituent components fail and a series configuration fails if any of the subsystems in the series fails. These assumptions are hardly ever met in the design of complex systems and networks.

Another issue that has not been addressed in earlier work is limitations in the performance of optimisation algorithms. Most realistic systems by far exceed the computational requirements of the benchmark problems discussed in the literature. In the past, this has confined application of techniques to small examples. However, recent advances in parallel algorithms and progress in GRID computing create possibilities for major advances in the optimisation of dependable systems in the near future.

HiP-HOPS is a recent model-based analysis technique that offers a departure from RBDs [66]. HiP-HOPS overcomes the limitations of RBDs by providing automatic analysis of complex systems (i.e. not necessary series-parallel as in the RAP) that exhibit multiple failure modes. HiP-HOPS makes use of genetic algorithms to determine the optimal allocation of redundant components and has been demonstrated on the RAP, where it produces optimal or near-optimal solutions along the Pareto frontier. The safety analysis capabilities of HiP-HOPS are used as the evaluation function to determine the quality of the system variation being considered at each stage of the optimisation. This offers a significant performance advantage – because HiP-HOPS analyses usually take only a few seconds to complete, evolution over hundreds or thousands of generations of possible designs is relatively quick. Another advantage is the flexibility of HiP-HOPS, which also allows for other objectives to be considered in optimisation. Additionally, HiP-HOPS does not rely on fixed redundancy allocation schemes and so it is possible to experiment with other methods of improving safety, e.g. active and passive standby redundancies, recovery blocks, execution platform reassignment, safety monitors, and in the general case substitution of any hardware of software sub-architecture with user specified alternatives.

In summary, enabling model-based system optimisation could assist the systematic exploration of large and complex design spaces. However, there has been little practical progress in the field. To our knowledge there has been no work focused on dependability versus cost optimisation in the context of emerging and ADLs, and there is, therefore, a clear opportunity for a contribution to the SOTA in this area, e.g. by exploring how EAST-ADL2 can deliver such optimisation in conjunction with tools such as HiP-HOPS.

3.2 Meta-modelling languages

Two relationships form the core of models used in computer systems[2]. The first relationship, called “represented by”, identifies a representation role of a given modeled object over a model. The second relationship, called “conforms to”, identifies a dependency of a given model on a modelling language. In model-driven engineering (MDE), the latter relationship receives special attention since domain modelling languages are described and prescribed by models. These models are

[2] This section is adapted from H. Espinoza’s PhD thesis An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real-Time Systems, CEA LIST, september 2007.

called *metamodels*. A metamodel is yet another abstraction highlighting properties of the model itself. This model is said to conform to its metamodel like a program conforms to the grammar of the programming language in which it is written. This means that a metamodel describes the various kinds of contained model elements and the way they are arranged, related and constrained.

There are different taxonomies for defining the characteristics required to fully specify a domain-specific language (see for example [67], [68]). In all these classifications, we have identified three important specification criteria:

1. ***Syntax***. This includes a set of constructs that can be exchanged and specifies how they can be linked together to form valid expressions in a language (syntactical rules). *Abstract syntax* defines the rules for creation of well-formed sentences in a given language, whereas *concrete syntax* provides a concrete representational system for expressing the elements of a given language. In visual modelling languages, concrete syntax may be graphical, textual, or mixed.
2. ***Semantics***. To provide semantics for syntactical constructs, we need to describe their meanings in terms of some well-known semantic domain. This implies describing syntactical elements in terms of a formal, mathematical framework (denotational approach), a set of logical rules (axiomatic approach), or a set of rules for execution on an abstract machine (operational approach).
3. ***Pragmatics***. For modelling languages in particular, the relationship between the syntactical constructs and their understandability (to language users) is key to the effectiveness of model interpretation. Pragmatics take into account the visual features of concrete syntactical constructs (e.g. morphological, geometric, spatial and topological relationships) used by visual modelling languages to represent real objects and relationships.

The predominance of metamodels in visual modelling languages specifications raises the issue of their potential reuse. Indeed, the ability to reuse and/or specialize (parts of) existing metamodels is crucial to avoiding development of new modelling languages from scratch. There are two major ways to extend modelling languages [69]:

1. ***Heavyweight extension***. This approach implies the inclusion and refinement of new language constructs from an existing modelling language. It allows designers to extend, refine and modify the source language as required to create a new modelling language.
2. ***Lightweight extension***. This approach is restricted to the use and extension of an existing modelling language metamodel without modifying the abstract syntax or semantics of the source modelling language.

The choice of approach certainly depends on the particular language application and on what kind of (and how many) design resources are available (tools, know-how, etc.). In a domain-specific modelling language (DSML) that derives from a *lightweight extension* of a general-purpose language, such as UML, we must make tradeoffs between a number of advantages and disadvantages. This is the approach carried on with the EAST-ADL2 language implementation.

The main advantages of the lightweight extension mechanism are the low cost of support tools and a more effective language learning curve for multiple and long-term projects. This approach does not generally require very specialized training. Its disadvantages relate fundamentally to the base language, which must cover a broad semantic domain to be reused by multiple DSMLs. In UML, for instance, this implies specification of multiple semantic variation points that increases the complexity of the DSML design.

Profiles [70] are the built-in lightweight mechanism that serves to extend MOF-based languages. More specifically, profiles are used to customize UML for a specific domain or purpose via extension mechanisms that enrich the semantics and syntax of the language. A *stereotype* is the

basic feature for UML extension. It can be viewed as the specialization of an existing UML concept, which provides capability for modelling domain-specific concepts or patterns. Stereotypes may have attributes (also called *tags*) and be associated with other stereotypes or existing UML concepts. From a notational viewpoint, stereotypes can give a different graphical symbol for UML model elements. For instance, a class stereotyped as «clock» might use a picture of a clock symbol instead of the ordinary class symbol. Additionally, stereotypes can also be influenced by restrictions expressed in constraints. The standard machine-readable textual language for defining constraints in MOF-based languages is Object Constraint Language (OCL).

Since the recent UML2 versions, profiles have significantly enhanced both the syntax and the semantics of the extension mechanisms:

- Profiles integrate a special graphical notation that facilitates understanding of the defined extensions. Stereotypes are represented as classes linked to their base metaclasses by the “extension” association. Constraints and stereotype attributes are then represented as part of the stereotype definition.
- Profiles can use an excerpt of the whole UML metamodel. A typical profile that imports only a subset of UML is SysML. This makes SysML a compact DSML in terms of the number of UML concepts used.
- The stereotype definition concept is better positioned in the metamodelling pyramid promoted by OMG. Unlike UML 1.x, stereotypes are now considered M2 constructs.

However, efforts made to improve the profile mechanism have contrasted with the lack of guidance for its correct use. One common approach to designing UML profiles involves the so-called conceptual domain models. The language specification is defined at a first stage by means of a domain model (expressed in MOF or alternatively in UML itself), which abstracts away the UML mapping issues. The main intent of conceptual domain models is to separate concerns and enhance the appropriateness of the language construct with regard to the domain concepts. The domain model, which is created with pure domain considerations, is then translated into the target profile.

A recent paper [71] proposes a systematic methodology to design UML profiles by adopting a set of minimum framing rules. Since these rules are defined on the basis of regularly occurring design patterns, domain models can be subsequently checked for self-consistency and interactively transformed into stereotypes, tags and constraints. Each decision of the designers is interactively evaluated to alert them to potential UML language conflicts and propose more appropriate mapping decisions. Designer choices are stored in a decision file that allows profile regeneration, which is particularly useful when modifications are applied to the domain model.

Such a proposal is of particular importance in the work conducted in ATESST2, however some of the suggested ideas need to be adapted to the fact that EAST-ADL2 is a language for which implementations – in the form of profiles – already exist and have been continuously refined. Another aspect is that the conceptual domain language is as important as its implementation as profile and both need not be too divergent, which somewhat constrains the scope of design patterns applicable to enhance the sole profile.

AUTOSAR Metamodelling

AUTOSAR defines templates that are the format for exchange of software information within the automotive domain. Formally this is performed by the definition a metamodel that is transformed into an XML Schema. The metamodel is defined in the UML tool Enterprise Architect and AUTOSAR has defined rules for the design of this metamodel. Basically AUTOSAR uses a subset of UML, with additional restrictions and a UML profile to tag information in the metamodel [72]. These rules do not only facilitate the generation of an XML Schema of a particular structure but also the generation of editors directly from the AUTOSAR metamodel.

EAST-ADL2 is related to AUTOSAR such that EAST-ADL2 refers to AUTOSAR, i.e. a lightweight extension - even though AUTOSAR can also be seen as a subset of EAST-ADL2 where the AUTOSAR System is in the ImplementationArchitecture abstraction level of EAST-ADL2.

EAST-ADL2 has defined the domain model in the same tool as AUTOSAR uses. By applying the same rules as AUTOSAR poses on its metamodel some additional benefits can be reached:

1. The format of the domain model complies with the automotive domain de facto standard.
2. Distinction of different concepts and their relations as defined by AUTOSAR are formalized in the domain model. E.g. types, prototypes, and references to occurrences (instanceRef)
3. The formalization of the domain model would also serve as additional design information when the UML Profile and Tool platform is defined within the ATESST2 project.
4. The EAST-ADL2 domain model would be prepared to be used to define AUTOSAR compliant exchange formats and editors.
5. The experiences in designing a UML Profile for EAST-ADL2 would give experiences on how a profile for AUTOSAR should be designed.

4 EAST-ADL2 in the context of various evaluations and classifications of ADL

4.1 Introduction

For a long time the term Architecture Description Language has not been clearly defined in the research community. Instead frameworks for classification and comparison of languages have been devised. The languages claiming to be ADLs span a wide range: from programming languages to formal specification and simulation languages, that differ in the level of tool support, the extent to which the syntax and semantics of the language is captured formally, the support for model checkers, code generation and runtime support. We will evaluate the current status of EAST-ADL according to a widely recognized classification framework.

The benefits of ADLs include a shift in focus to larger scale thinking about software, also called "programming in the large", raising the importance of a higher level view of the software. Thus allowing better planning, aid understanding, aid communication through simple and often graphical syntax.

4.2 A Classification Framework for ADLs

A list of the minimum requirements for an ADL is provided by Medvidovic and Taylor [8]. An ADL must explicitly model components, connectors, and their configurations; furthermore, to be truly usable and useful, it must provide tool support for architecture-based development and evolution. An ADL is thus a language that provides features for modelling a software system's conceptual architecture, distinguished from the system's implementation. ADLs provide both a concrete syntax and a conceptual framework for characterizing architectures.

The classification framework proposed Medvidovic and Taylor [8] consists of several criteria that can be packaged into four groups: components, connectors, architectural configurations and tool support. In the following we will look at the criteria for each of these groups separately.

4.2.1 Components

ADLs allow a component based description of the structure of the system, where the structure can be described in terms of components and connectors. Components are units of computation or data structures.

- Interface: An interface specifies the services offered by the component and thus specifies the interaction points between components and the external world. An Interace also specifies the computational commitments and constraints.
- Types: A type provides an abstraction that encapsulates functionality for reuse. It can be instantiated multiple times and can even be parameterized.
- Semantics: The semantics provides a high level model of a components behavior.
- Constraints: A constraint specifies an assertion of the system.
- Evolution: Evolution support allows the modification of the properties of the component, in a controlled manner, with techniques such as subtyping and refinement.
- Non-functional properties: All the properties that cannot be derived from behavior need to be specified separately such as the properties needed for simulation, performance analysis.

4.2.2 Connectors

Connectors model the interactions among components and specify additional rules. Just as the components, the connectors are classified along the lines of interfaces, types, semantics, constraints, evolution and non-functional properties.

- Interface: The interface of a connector enables the proper connectivity between components by specifying the interaction between the connector and its attached components or other attached connectors.
- Types: A connector type is an abstraction that encapsulates component communication, coordination, and mediation decisions. This can be accomplished by an extensible type system, or a built-in, enumerated type.
- Semantics: The semantics is a high-level model of the connector's behavior. It entails the specification of an interaction protocol.
- Constraints: A constraint ensures the connector's adherence to an interaction protocol, e.g. by specifying multiplicity.
- Evolution: Evolution supports the modification of the connector's properties, e.g. by incremental information filtering, subtyping or refinement.
- Non-functional properties: A Non-functional property states the requirements for correct implementation and is used for e.g. simulation, performance analysis.

4.2.3 Configurations

While components and connectors specify the parts of an ADL model, the architectural configuration describes criteria for the model as a whole. The configuration can be thought of as a graph that describes the architectural structure.

- Understandability: The software architecture described by the ADL is an early communication conduit for different stakeholders. Therefore the ADL should present structural information with a simple and understandable syntax
- Compositionality: The software architecture is described at different levels of detail, and provide the ability to abstract parts away.
- Refinement and traceability: The ADL can be used as a bridge between informal specification and implementation. The description allows correct and consistent refinement into executable code, including traceability of changes.
- Heterogeneity: The ADL allows integration of preexisting components, components of different granularity, modelling language and implementation language. Thus an ADL needs to be open and provide facilities for different types of components and connectors.
- Scalability: An ADL provides abstractions to cope with software complexity and size.

- Evolvability: The ability for addition, removal, replacement, and reconnection in a configuration.
- Dynamism: While evolution captures offline changes, dynamism deals with structural and behavioral modifications during execution.
- Constraints: Global constraints that oft depend on constraints for components and connectors.
- Non-functional properties: Non-functional properties on a global level.

4.2.4 Tool support

ADLs provide a formal description of the system, thus they can be manipulated and reasoned about in an automated fashion by software tools. This tool support determines the usefulness of an ADL to a large extent.

- Active Specification: An active specification reduces possible design options.
- Multiple Views: Multiple views of the same architecture are a way of reducing the perceived complexity of the architecture. It is also useful to provide different views for different stakeholders. At the same time one has to assure the consistency between the views.
- Analysis: Through analysis of the architecture errors can be detected early on, reducing their impact and cost.
- Refinement: The correctness and consistency between architectures cannot always be guaranteed, but through tool support confidence in the correctness and consistency can be gained.
- Implementation Generation: The end goal of the development process is the executable system and its implementation. Automated generation can ensure consistency and traceability between architecture and implementation.
- Dynamism: When changes are made to the architecture they are evaluated to determine if they are desirable and property-preserving.

4.3 Evaluation of EAST-ADL

We use the framework of Medvidovic and Taylor [8] introduced in the previous section to evaluate the current status of EAST-ADL.

4.3.1 Components

- Interface: EAST-ADL uses ports for defining the interface of a component. A variety of port constructs is available, such as the ADLFlowPort, ADLPortGroup and ADLClientServerPort.
- Types: EAST-ADL uses a Type/Prototype concept, where types are defined and later reused as prototypes. This concept is realized in the constructs ADLFunctionType and ADLFunctionPrototype. It is possible to parameterize these constructs using variability concepts.
- Semantics: The semantics of EAST-ADL is described textually in a language specification document. However, EAST-ADL is not specified with formal semantics.
- Constraints: To put constraints on EAST-ADL components, the Object Constraint Language (OCL) can be used. EAST-ADL also offers the construct DesignConstraint used for feature modeling.
- Evolution: Extensive support for the evolution of requirements is available. The evolution can be tracked explicitly by constructs such as ADLRefine, ADLSatisfy, ADLVerify, ADLRealization.
- Non-functional properties: EAST-ADL offers constructs for safety analysis and timing analysis. The construct TimingRestriction allows to give bounds on system timing attributes, i.e. end-to-end delays, periods, etc. The SafetyRequirements construct allow to augment components with information for safety analyses.

4.3.2 Connectors

- Interface: The interface of connectors is specified by its ports. EAST-ADL offers the same constructs for connectors as for components, namely ADLFlowPort, ADLPortGroup and ADLClientServerPort.
- Types: EAST-ADL uses a Type/Prototype concept.
- Semantics: The semantics of EAST-ADL is described textually in a language specification document. However, EAST-ADL is not specified with formal semantics.
- Constraints: OCL can be used for defining constraints on connectors.
- Evolution: There are no concepts for the evolution of connectors.

4.3.3 Configurations

- Understandability: The EAST-ADL implementation as a UML profile offers a graphical syntax, which improves understandability of complex configurations.

- Compositionality: EAST-ADL provides five levels of abstraction: Vehicle Layer, Analysis Level, Design Level, Implementation Level and Operational Level. These levels are useful for composition, but also for refinement and traceability.
- Refinement and traceability: EAST-ADL provides five levels of abstraction: Vehicle Layer, Analysis Level, Design Level, Implementation Level and Operational Level. The elements of different layers can be linked through typed traceability links such as ADLRefine, ADLSatisfy, ADLVerify and ADLRealization.
- Heterogeneity: EAST-ADL allows the integration of preexisting components at different levels of abstraction. It even allows the integration of "external" definitions, mainly for behavior modeling.
- Scalability: EAST-ADL has several approaches for dealing with complexity and size. Feature modeling offers one such approach, but also the five predefined levels of abstraction help to limit the perceived complexity.
- Evolvability: The EAST-ADL tools allow addition, removal, replacement and reconnection.
- Dynamism: At the moment there is no explicit language support for structural and behavioral modifications during execution.
- Constraints: Configurations and its components can be constrained using OCL.
- Non-functional properties: Just as with components, timing information and safety information can also be included on a global level.

4.3.4 Tool support

- Active Specification: The EAST-ADL tooling actively supports the user during specification, e.g. the tool prevents syntactically incorrect connections between components.
- Multiple Views: EAST-ADL provides possibilities for multiple views on different levels of abstraction. However, it is left up to the user to ensure the consistency between the different views.
- Analysis: EAST-ADL supports validation techniques such as simulation, rapid control prototyping (RCP) and hardware/software in the loop and safety analysis. Simulation requires execution of the plant model and the system model, which are supported.
- Refinement: EAST-ADL provides five levels of abstraction: Vehicle Layer, Analysis Level, Design Level, Implementation Level and Operational Level. The refinement between the first three levels is explicitly supported by the current EAST-ADL tools.

- Implementation Generation: Currently there is no automated transformation from EAST-ADL to an implementation language such as AUTOSAR.
- Dynamism: At the moment there is no support for property-preserving refactorings of models.

4.4 Conclusion

In this section we have introduced the framework of Medvidovic and Taylor for the classification and comparison of ADLs. We then evaluated EAST-ADL according to this framework. According to this framework, EAST-ADL fulfills all the criteria of an Architecture Description Language. We found that according to this framework, EAST-ADL can be improved regarding dynamism.

5 Conclusions

As can be seen in this document it is hard to define what should be in a architecture description language like EAST-ADL2. The project limits itself by setting a scope on language to provide a 'layered approach' to how electronic systems in vehicles can be defined. This is not enough as there are many ways to do this.

Choosing the right ones in the multitude of possible abstraction layers and viewpoints needing modelling support is one task. The important tasks in this project is the ability to define structure, behaviour, variability and requirements. To make matters even more complicated there are often several possible non-compatible ways to solve these problems. This document tries to show the different ways and their pros and cons. In this conclusion a justification based on project expertise and the history of the language itself.

Some issues stem from the project itself, others are consequences based on the introduction of new state-of-practice approaches that the language need to support or conform to. Projects like Autosar, TIMMO, ISO-26262 are sources for more input to the language. Autosar provides a full platform that the implementation EAST-ADL tries to model logically is executed physically on. TIMMO builds on EAST-ADL1 and adds ways to model timing. ISO-26262 has a need to model functions form a safety point of view. As state-of the art is updated the language needs to flex and adapt, not necessarily by changing the language but by giving an interpretation of how a specific task should be modelled.

Importance of analysis driven design

The importance of model based design comes from the aspects of seeing the model as a information storage where one can perform tasks automatically. Using a document approach you are limited to extraction of data for different purposes using manual labor. With a logical model of the system it is possible to do semi-automatic analysis by adding data to the model and extract data to external tools connected to the information model. But with more and more complex models the need to understand how systems affect systems and not only how components in a system interacts is needed and this is difficult to do without a larger scope model and automatic tools.

As the external tools use the data in the model data it is quite possible to make design/architecture decisions that are fed back to the model automatically. This makes tracking of change easier as the number of points where data for the system is stored is lowered. But it is possible that analysis driven design can be made in such a way that permutations of the model under design are created as the analysis is performed and one can trace the way the model has been updated by the tool chain. This is not really a language issue but a tool and model presentation issue.

Safety Modelling and Safety Analysis.

In section 3.1.3, we identified two paradigms for model-based safety analysis that currently define the SOTA in this area (compositional safety analysis & behaviour fault simulation) and reviewed a number of techniques that fall in this classification. We saw that the two paradigms have different strengths and benefits, e.g. higher degree of automation is possible with formal techniques or behavioural simulation while higher performance and more scalable algorithms are available with deductive compositional safety analysis techniques. Fault simulation performed on semi-formal or formal models tends to provide more accurate results when specific instances and scenarios of failure are examined. However, compositional safety analysis techniques can achieve wider coverage of faults and fault combinations, they are more flexible and can be applied to a greater range of systems. This makes both paradigms useful in their application to model-based design, and therefore a modelling language should be designed to support both paradigms.

HiP-HOPS and the AADL's Error Model Annex can together provide a good basis for the definition of the Error Model of EAST-ADL2. They both enable the modelling of system failure behaviour and allow analysis of that behaviour using tools. Because they do not rely on simulation or model checking, they have greater performance and allow for more expressive descriptions of the failure

behaviour. In the case of HiP-HOPS, this latitude makes it possible to develop extensions into reusable patterns of failure, temporal failure logic, and automatic design optimisation of dependability versus cost. HiP-HOPS also supports both FTA and FMEA but produces them via a deductive process, avoiding the combinatorial explosion issues inherent in purely inductive techniques like DCCA or Altarica.

Overall our analysis shows that HiP-HOPS offers a good range of capabilities and, since its failure semantics are independent of the modelling language used, it is flexible enough to be incorporated into other modelling languages. It could therefore provide a basis for defining error modelling and safety analysis in EAST-ADL2, and its multi-objective optimisation capabilities could be used to support optimisation in EAST-ADL2 as well. However, HiP-HOPS is based on fault propagation and lacks the concept of states. Although this is not necessarily a problem in terms of safety analysis, as Pandora makes it possible to extend HiP-HOPS with temporal logic to implicitly represent transitions to failed states, explicit state modelling could still be useful for formal verification & behavioural fault simulation e.g. using model-checkers. For this reason, the state-based error model annex of AADL could also prove useful as an input to the error modelling in EAST-ADL2 and we believe that it would still be possible to harmonise a state-based error model with HiP-HOPS and thereby continue to derive the benefit of its analysis & optimisation capabilities. Finally, the capabilities of Rodelica as a means of facilitating modelling for fault diagnosis in EAST-ADL2 will also need to be further explored.

There is very little work on integrating model-based safety analyses techniques with emerging ADLS and, therefore, there is a large potential for contribution by ATESST2. Any such contribution would need to take into account developments in the emerging ISO 26262 safety standard and enable application of the standard, e.g. by facilitating the allocation, decomposition and demonstration of ASILs as this was discussed in **Error! Reference source not found.**.

A more detailed analysis of SOTA in safety analysis and recommendations for error modelling in EAST-ADL2 will be given in deliverables I3.2.1 and I3.2.2 which are dedicated to these two topics.

Multi-objective Analysis and Optimisation

In section 3.1.5 we reviewed work on architecture-driven multi-objective analysis and optimisation. Current trends in architecture analysis are characterized by providing monolithic solutions for specific non-functional parameters (e.g., either performance or variability only). However, embedded system architectures need more flexible mechanisms and tools to specify and to evaluate many different non-functional design alternatives at all design levels. We have seen that, from a modelling language viewpoint, some aspects need additional investigation.

In general, we need sound modelling means to integrate non-functional information from different validation and verification viewpoints and provide an optimized global solution to a given design decision problem. In ATESST2, the thrust of the work has to be put in the specification mechanisms for integrating multidimensional information, its treatment (transformation/refinement) to enable mathematical calculation, and the methodological basis to allow developers for a systematic use of the proposed approach. The use of mathematical algorithms (search strategies, genetic algorithms, etc.) to explore design alternatives by optimizing objective functions is a core subject in this topic. Numerous and useful results can be found in the literature. The suitability of existing algorithms for automotive embedded systems has to be evaluated (as discussed later in this section).

From the architecture modelling viewpoint, some of the core issues have been defined as follows:

> *"Given various allocation alternatives of application/platform models, how to specify the different property value versions (resulting from the allocation) annotated in the internal modelling parts of the allocated application/platform models?"*

Furthermore, building heterogeneous models for analysis implies to integrate analysis results from different views and/or subsystems and to calculate global predictions.

> *"How can we express the relationships between different analysis methods to calculate global quality parameters (derived predictions, optimization objective functions, measures of effectiveness)?"*

Last but not least,:

> *"How can we drive sensitivity analysis tools by properly qualifying parameters in design models?"*

Some early contributions were identified on these topics, which will serve as a basis for the work in ATESST2, such as SysML parametric diagrams, and further proposals to model complex analysis contexts based on SysML and MARTE.

Focusing on the difficult issue of increasingly larger and complex potential design spaces, in section 3.1.5.2 we saw that the problem of model-based system optimisation, particularly in terms of dependability (i.e. safety, reliability, availability) and cost, is an important problem that could have significant implications for system design processes if solved; the ability to rapidly search through large design spaces to determine optimal or near-optimal design variants, particularly for complex systems such as real-time embedded systems or active cooperative systems, could result in designs of superior safety and reduced cost, quite aside from the time-saving benefits.

However, as we have shown, there has been little practical progress in the field, as most research has focused on solutions to relatively small and restricted theoretical problems; in particular, to our knowledge there has been no work focused on optimisation in the context of emerging modelling languages and ADLs. There is, therefore, a clear opportunity for a contribution to the state of the art in this area.

It would therefore be worthwhile to investigate further how appropriate dependability and cost modelling concepts can be developed to support multi-objective optimisation of EAST-ADL2 models in conjunction with tools such as HiP-HOPS that provide such advanced capabilities. The aim of optimization would be to automatically evolve models that do not necessarily meet dependability requirements to designs that fulfil such requirements with minimal costs. Optimisation could be achieved via exploration of potential design spaces using meta-heuristics such as genetic algorithms. The specification of design alternatives and variant sub-architectures the combinations of which define the potential design space will be described in EAST-ADL2 using an extension of the present variability concepts that will be achieved in ATESST-2.

Recommendations for dependability and cost modelling concepts to support optimisation in EAST-ADL2 will be given in deliverable I3.5.1 which is dedicated to this topic.

6 References

[1] “AUTOSAR Technical Overview” , v2.2.2, Release 3.1, www.autosar.org/download/specs_aktuell/AUTOSAR_TechnicalOverview.pdf

[2] CMMi

[3] ARTIST2 - Network of Excellence on Embedded Systems Design, Roadmap for Embedded Software and Systems. March 30, 2006 [Online]. Available < http://www.artistembedded.org/FP6/ARTIST2Events/Publications/Roadmap/

[4] M. Törngren and O. Larses, “Characterization of model based development of embedded control systems from a mechatronic perspective - drivers, processes, technology and their maturity.”, Technical Report, TRITA-MMK 2004:23, ISSN 1400-1179, ISRN/KTH/MMK/R-04/23-SE

[5] R. Jeutter, and B. Heppner, “Model-Based System Development – Is it the Solution to Control the Expanding System Complexity in the Vehicle?” SAE World Congress 2004. Detroit, MI. March 8-11. 2004. SAE 2004-01-0300.

[6] “Automotive Spice”, www.automotivespice.com

[7] IEEE Standard, Architecture Description Languages

[8] Nenad Medvidovic and Richard N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, 26(1), pp. 70-93, January 2000.

[9] Christian Dziobek, Florian Wohlgemuth, Thomas Ringler, “AUTOSAR in the Development Process”, dSPACE MAGAZINE, 1/2008, pp 6-13

[10] “MOST MSC Cookbook.pdf”, www.mostcooperation.com/publications/Specifications_Organizational_Procedures/index.html?dir=353

[11] “Bluetooth SIG”, www.bluetooth.org

[12] D3.2 Report on behavioural modelling with the EAST-ADL2, ATESST1 project, 2008.

[13] Cuccuru A., Gérard S. and Radermacher A., Meaningful Composite Structures - On the Semantics of Ports in UML2.. MoDELS, LNCS 5301:828-842, 2008

[14] S. A. Ferguson. (2007) "The Effectiveness of Electronic Stability Control in Reducing Real-World Crashes: A Literature Review", *Traffic Injury Prevention*, 8(4): 329 – 338, Dec. 2007.

[15] IEC 61025 (1990) "IEC (International Electrical Commission) Fault-Tree-Analysis (FTA)," Geneva.

[16] W.E. Vesely, M. Stamatelatos, J.B. Dugan, J. Fragola, J. Minarick, J. Railsback. (2002) *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance, USA.

[17] IEC 60812 (1991) “IEC (International Electrical Commission) Functional safety of electrical/electronical/programmable electronic safety/related systems, Analysis Techniques for System Reliability - Procedure for Failure Mode and Effect Analysis (FMEA),” Geneva.

[18] N. G. Leveson. (1995) *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[19] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj (1998). "Using abstraction and model checking to detect safety violations in requirements specifications,” *IEEE Transactions on Software Engineering*, vol. 24, no. 11, pp. 927–947, 1998. [25] IEC 60812, “IEC (Intern. Elect. Commission),

[20] J. D. Reese and N. G. Leveson. (1997) "Software deviation analysis," in *Proceedings of the 19th International Conference on Software Engineering*. ACM Press, 1997, pp. 250–261.

[21] O. Lisagor, J.A. McDermid, D.J. Pumfrey. (2006) "Towards a practicable process for automated safety analysis," in *Proceedings of ISSC* 2006.

[22] P. Fenelon and J.A. McDermid (1993). "An integrated toolset for software safety analysis." Journal of Systems and Software 21(3), pp 279-290.

[23] Y.I. Papadopoulos and J.A. McDermid. (1999) "Hierarchically performed hazard origin and propagation studies," in Computer Safety, Reliability and Security, 18th International Conference, SAFECOMP'99, Toulouse, France, September, Proceedings, ser. *LNCS*, M. Felici, K. Kanoun, and A. Pasquini, Eds., vol. 1698. Springer, 1999, pp.139–152

[24] B. Kaiser, P. Liggesmeyer, O. Mäckel. (2003). "A New Component Concept for Fault Trees", Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS '03)

[25] L. Grunske, B. Kaiser, Y. Papadopoulos. (2005) "Model-driven Safety Evaluation with State-Event-Based Component Failure Annotations." Component-Based Software Engineering, 8th International Symposium, CBSE 2005, Proceedings, pp 33-48.

[26] M. Wallace (2005) "Modular architectural representation and analysis of fault propagation and transformation." Electronic Notes in Theoretical Computer Science. 141(3):53-71.

[27] P. Feiler, A. Rugina. (2007) *Dependability Modelling with the Architecture Analysis and Design Language*. Technical report, CMU/SEI-2007-TN-043. Software Engineering Institute, Carnegie Mellon University, USA, Jul 2007.

[28] P. Bieber, C. Castel, C. Seguin. (2002) "Combination of fault tree analysis and model checking for safety assessment of complex system," *in Proceedings of the 4th European Depting Conference on Dependable Computing* (EDCC), ser. LNCS, vol. 2485. Springer, pp. 19–31.

[29] M. Bozzano, A. Cavallo, M. Cifaldi, L. Valacca, and A. Villafiorita. (2003) "Improving safety assessment of complex systems: An industrial case study." in FME 2003: Formal Methods, International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003, Proceedings, ser. Lecture Notes in Computer Science, K. Araki, S. Gnesi,and D. Mandrioli, Eds., vol. 2805. Springer, pp. 208–222.

[30] M.P.E. Heimdahl, Y. Choi, and M.W. Whalen. (2005) "Deviation analysis: A new use of model checking," Automated Software Engineering, vol. 12, no. 3, pp. 321–347.

[31] M. Güdemann, F. Ortmeier, and W. Reif. (2007) "Using deductive cause-consequence analysis (DCCA) with SCADE," in Computer Safety, Reliability, and Security, 26th International Conference, SAFECOMP 2007, ser. *LNCS*, F. Saglietti and N. Oster, Eds., vol. 4680. Springer, pp. 465–478.

[32] K. Lunde, R. Lunde, B. Münker (2006). "Model-Based Failure Analysis with Rodon" In Proceedings of ECAI 2006 - 17th European Conference on Artificial Intelligence Riva del Garda, Italy, August 29 -- September 1 2006

[33] J. Mauss, V. May and M. Tatar (2000). "Towards Model-Based Engineering: Failure Analysis with Mds." In Proceedings of ECAI-2000 Workshop on Knowledge-Based Systems for Model-Based Engineering, Berlin, Germany, 2000

[34] M. Sachenbacher, P. Struss and C. M. Carlén (2000). "A Prototype for Model-Based on Board Diagnosis of Automotive Systems." AI Communications, vol: 13, issue: 2. pg 83 - 97, 2000

[35] P. Bunus and K. Lunde. (2008) "Supporting Model-Based Diagnostics with Equation-Based Object Oriented Languages." 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT). Jul 8 2008, Paphos, Cyprus.

[36] M. D. Walker, L. Bottaci, Y.I. Papadopoulos. (2007). "Compositional Temporal Safety Analysis", SAFECOMP 2007, LNCS 4680:105-119, Springer, 2007.

[37] P.I. Barton and C.C. Pantelides. (1993) "Gproms – a Combined Discrete/Continuous Modelling Environment for Chemical Processing Systems." Society for Computer Simulation, Simulation Series 25(3):25-34, 1993.

[38] E. Christen and K. Bakalar. (1999). "Vhdl-Ams - a Hardware Description Language for Analog and Mixed-Signal Applications." IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol: 46, issue: 10. pg 1263-1272, 1999

[39] P.C. Piela, T.G. Epperly, K.M. Westerberg and A.W. Westerberg (1991). "Ascend: An Object-Oriented Computer Environment for Modeling and Analysis: The Modeling Language." Computers and Chemical Engineering, vol: 15, issue: 1. pg 53-72, 1991

[40] W. Zeng, Y. Papadopoulos, D. Parker. (2007), Reliability Optimization of Series-Parallel Systems Using Asynchronous Heterogeneous Hierarchical Parallel Genetic Algorithm, Journal of Mind and Computation, 1(4): 403-412, China Academic Electronic Publishing House.

[41] I.P. Wolforth, M.D. Walker, Y.I. Papadopoulos. (2008) "A language for failure patterns and application in safety analysis." IEEE Conference on Dependable Computing Systems (DEPCOS'08), June 26-28 2008, Szklarska Poreba, Poland, June 2008.

[42] P. H. Feiler, D. P. Gluch and J. J. Hudak. (2006) "The Architecture Analysis and Design Language (AADL): An Introduction", Technical report, CMU/SEI-2006-TN-011, 2006. Software Engineering Institute, Carnegie Mellon University, USA

[43] Papadopoulos Y. (2003) Model-based system monitoring and diagnosis of failures using State-charts and Fault Trees, Reliability Engineering and System Safety, 81:325-341, Elsevier.

[44] Davis R., Hamscher W. (1992) Model Based Reasoning: Troubleshooting, in Hamscher W., Console L., de Kleer J. (eds.), Readings in Model-based Diagnosis, pages 3-28, Morgan Kaufman, ISBN: 1-55860-249-6.

[45] ISAAC: http://www.cert.fr/isaac

[46] ESACS: http://www.cert.fr/esacs

[47] SAFEDOR: http://www.safedor.org

[48] SETTA: http://www.vmars.tuwien.ac.at/projects/setta

[49] ASSERT:http://www.assert-project.net

[50] H. Espinoza, D. Servat, and S. Gérard, Leveraging Analysis-Aided Design Decision Knowledge in UML-Based Development of Embedded Systems, SHARK-ICSE 2008. Leipzig, Germany. May 2008.

[51] S. Künzli, "Efficient design space exploration for embedded systems", PhD Thesis, Eidgenössische Technische Hochschule ETH Zürich, April 2006.

[52] R. Racu, A. Hamann, R. Ernst, B. Mochocki, X. Sharon Hu: "Methods for power optimization in distributed embedded systems with real-time requirements". CASES 2006: p. 379-388.

[53] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst. System Level Performance Analysis - the SymTA/S Approach. In IEE Proceedings Computers and Digital Techniques, Vol. 152, Is. 2, March 2005.

[54] H. Espinoza, "An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real-Time Systems", PhD Thesis, University of Evry, FRANCE. September 2007.

[55] Papadopoulos Y., Grante C. (2005) Evolving car designs using model-based automated safety analysis and optimisation techniques, Journal of Systems and Software, Elsevier Science, 76(1):77-89, Elsevier, 2005

[56] Frank, A.T., C. Hwang, and W. Kuo (1977). Optimization techniques for system reliability with redundancy - a review. *IEEE Transactions on Reliability*, **R-26(3)**, 148-155.

[57] Chen, M.S. (1992) On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*, **11**, 309-315.

[58] Fyffe, D.E., W.W. Hines, and N.K. Lee (1968). System reliability allocation and a computational algorithm. *IEEE Transactions on Reliability*, **17(2)**, 64-69.

[59] Ghare, P.M. and R.E. Taylor (1969). Optimal Redundancy for Reliability in Series Systems. *Operations Research*, **17(5)**, pp. 838-847.

[60] Nakagawa, Y. and S. Miyazaki (1981). Surrogate constraints algorithm for reliability optimization problems with two constraints*. IEEE Transactions on Reliability*, **R-30**, 175-180.

[61] Coit, D.W. and A.E. Smith (1996). Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability*, **45(2)**.

[62] Kulturel-Konak, S., A.E. Smith, and D.W. Coit (2003). Efficiently Solving the Redundancy Allocation Problem Using Tabu Search. *IIE Transactions*, **35(6)**, 516-526.

[63] Kulturel-Konak, S., F. Baheranwala, and D.W. Coit (2005). Pruned Pareto-Optimal Sets for the System Red undancy Allocation Problem Based on Multiple Prioritized Objectives. *under review at the European Journal for Operations Research*.

[64] Papadopoulos Y. and C. Grante (2005). Evolving car designs using model-based automated safety analysis and optimisation techniques. *Journal of Systems and Software,* **76(1)**, 77-89.

[65] Grunske, L. (2006). Identifying “good” architectural design alternatives with multi-objective optimization strategies. In: *28th International Conference on Software Engineering* (Osterweil, L.J., Rombach, H.D., Soffa, M.L. (Ed)), 849-852, ACM, Shanghai, China.

[66] Zeng W., Papadopoulos Y., Parker D. (2007), "Reliability Optimization of Series-Parallel Systems using an Asynchronous Heterogeneous Hierarchical Parallel Genetic Algorithm", Journal of Mind and Computation, 1(4): 403-412, China Academic Electronic Publishing House.

[67] G. Guizzardi, L. Ferreira Pires, M. van Sinderen: “Ontology-Based Evaluation and Design of Domain-Specific Visual Modeling Languages,” Proceedings of the 14th International Conference on Information Systems Development, Karlstad, Sweden (2005)

[68] A. Cuccuru, C. Mraidha, F. Terrier, S. Gerard: “Templatable Metamodels for Semantic Variation Points,” ECMDA 2007, Haifa, Israel (June 2007)

[69] B. Selic, A Systematic Approach to Domain-Specific Language Design Using UML, Proc. of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2007, ISORC'07, pp. 2-9 ISBN: 0-7695-2765-5, May 2007.7, pp. 2-9 ISBN: 0-7695-2765-5, May 2007.

[70] L. Fuentes, A. Vallecillo: "An Introduction to UML Profiles," UPGRADE, The European Journal for the Informatics Professional, 5(2):5-13, April 2004, ISSN: 1684-5285.

[71] F. Lagarde, H. Espinoza, F. Terrier, Ch. André and S. Gérard, Leveraging Patterns on Domain Models to Improve UML Profile Definition. Fundamental Approaches to Software Engineering (FASE 08), 4961:116-130, March 2008

[72] AUTOSAR Template Modeling Guideline, www.autosar.org

[UOH_01] N Snooke, C. Price, Automated Failure Effect Analysis for PHM of UAV, Proceedings of the International Safety and Reliability Conference (ISSRC 2008); Singapore; April 2008; ISBN: 978-981-08-0446-6

[UOH_02] Ricardo Company, AutoFMEA Tool Presentation, www.ricardo.com/download/pdf/Ricardo_20_Slides_AutoFMEA.pdf

Appendix A Research and standardization activities

This list describes research and standardization activities in areas related to EAST-ADL2 scope.

ArtistDesign

http://www.artist-embedded.org/artist

Follow up of the ARTIST2 network of excellence on Embedded Systems Design.

ASSERT

http://www.assert-online.net

ASSERT is a (finished) project with the goal to improve the system-and-software development process for critical, embedded real-time systems in the Aerospace and Transportation domains. ASSERT improved the Systems Engineering practice in this area by taking a proof-based approach. In addition, a reference architecture that can be reused and instantiated in critical applications was developed. ASSERT was based on the AADL.

AIDE

www.aide-eu.org

AIDE is a (finished) integrated project in the eSafety area. The focus in AIDE is on system support to handle human behavioural aspects of new safety functions. AIDE applications represent the kind of complex, safety-related systems that require rigorous development support to manage their complexity and achieve correctness and safety.

CESAR

CESAR is an ARTEMIS project starting in 2009 with the purpose of identifying means to meet safety requirements and standards. The project covers several domains, and the idea is to identify a common core and define specializations for automotive, aerospace, automation and rail domains.

COMBEST

www.combest.eu

COMBEST will, during 2008-2010, provide a formal framework for component based design of complex embedded systems. This framework will, by building on substantial highly recognized background results of the academic partners, partly carried out within the integrated project SPEEDS, enable and provide:

Enable formal integration of heterogeneous components, such as with different models of communication or execution;

Provide complete encapsulation of components both for functional and extrafunctional properties and develop foundations and methods ensuring composability of components;

Enable prediction of emergent key system characteristics such as performance and robustness (timing, safety) from such characterizations of its subcomponents;

Provide certificates for guarantees of such key system characteristics when deployed on distributed HW-architectures

CVIS

www.cvisproject.org

The CVIS project aims to design, develop and test the technologies needed to allow cars to communicate and network directly with the roadside infrastructure.

DECOS

http://www.decos.at

DECOS -Dependable Embedded Components and Systems- is an integrative project that will develop the basic enabling technology to move from a federated distributed architecture (Architecture based on functional cluster in which the subsystems are implemented on “dedicated networks and discrete hardware resources”) to an integrated distributed architecture (the next generation dependable integrated architectures where “Safety Critical” and “Non Safety Critical” functionalities are combined in an integrated architecture by means of a dependable infrastructure and where the different networks of the sub-systems are simplified) in order to reduce development, production and maintenance cost and increase the dependability of embedded applications in many application domains. DECOS plans to develop technology invariant software interfaces and encapsulated virtual networks with predictable temporal properties such that application software can be transferred to a new hardware and communication base with minimal effort (legacy re-use). The DECOS methodology and the tools has been evaluated by building three applications in the automotive, aerospace and control domain, respectively. DECOS builds on the substantial results of previous European research projects (NextTTA, FIT, TTA, SETTA, RISE, X-By-Wire, PDCS, DEVA, DSOS). The components and tools developed within DECOS covers: cluster design, middleware and code generators, validation and certification as well as systems-on-a-chip (SoCs) for high dependability applications.

Decos is a (finished) FP6 project proposing a distributed execution platform and tools for the design of dependable embedded systems. The goal is to improve diagnosis, maintenance and dependability, reduce development and component cost and address intellectual property issues

EASIS

www.easis.org

EASIS is a (finished) project with the goal to define and develop a powerful and highly dependable in-vehicle electronic architecture. In addition, the project addressed methodology and tools supporting the development of these systems.

EDONA

http://www.edona.fr/

The EDONA (Environnements de Développement Ouverts aux Normes de l'Automobile) French project of System@tic Paris-Région cluster aims at the building of an open platform that facilitates the realization of business modular development chains, interoperable and adaptable to the different needs of actors and business of the automotive industry. It aims at developing an Eclipse based tool chain for development of AUTOSAR systems including modelling, simulation, test and HMI.

ESACS

http://www.cert.fr/esacs/

ESACS (finished) is a RTD project that responds to the Growth 2000 call, Key Action "New perspectives in Aeronautics".

The technical and scientific objectives of ESACS are to define a methodology to improve the safety analysis practice for complex systems development, to set up a shared environment based on tools supporting the methodology, to validate the methodology through its application to case studies. The environment between design and safety will consist of tools to generate parts of the safety analysis using information extracted directly from the system model and of a repository including all the safety information related to the complex system under development.

Families

www.esi.es/en/Projects/Families/

FAMILIES (finished) is a next project in a sequence of following projects: ARES and PRAISE, then ESAPS, and CAFE.

ITEA projects ESAPS, and CAFÉ has lead to a recognized European community on the subject of System Family Engineering. The FAMILIES project aims at growing the community, consolidating results into fact-based management for the practices of FAMILIES and its preceding projects, and to explore fields that were not covered in the previous projects, in order to complete the Framework.

GST

www.gstforum.org/en/home.htm

GST is a Framework Programme 6 project addressing the standardization of telematics services.

HARTES

www.hartes.org

hArtes is an European project aiming at laying the foundations of a new holistic approach for the design of complex and heterogeneous embedded solutions (hardware and software), from the concept to the silicon (or B2B, from the brain to bits). From the application point of view, the complexity of future multimedia devices is becoming too big to design monolithic processing platforms. This is where the hArtes approach with reconfigurable heterogeneous systems becomes vital.

IEC 61508

www.iec.ch

The international standard to be considered when electrical/electronic/programmable electronic systems are used to carry out safety functions. The standard serves a dual purpose, first to enable application sector standards using IEC61508 as the basis, and to provide a standard for functional safety system for application sectors not yet adopting a safety standard.

INTERESTED

www.interested-ip.eu/index.html

INTERESTED as an EU programme with aims regarding the tool-ennvironments that increase productivity when developing complex embedded systems.

Among the project aims are to integrate the requirements of Major Tool Users of embedded systems tools to realize a reference and open interoperable embedded systems tool-chain, having in mind a broad socio-economic benefit for the European citizens, the performance of Embedded Systems generating long term societal benefits such as increased aircraft and transportation safety, reduced fuel and energy consumption and competitiveness of key European industries.

ISAAC

www.cert.fr/isaac

Project ISAAC (*FP6-2002-Aero-1-501848*) builds upon and extends the results of ESACS that has shown the benefit of using formal techniques to assess aircraft safety. Our goal is to go a step further into the improvement and integration of safety activities of aeronautical complex systems. Potential benefits range from higher confidence in the safety of systems to increased competitiveness of European industries.

ISO 26262

www.iso.org

The adapation of the functional safety standard IEC61508 for the automotive industry.

MARTES

www.martes-itea.org

MARTES (finished) (Model-based Approach to Real-Time Embedded Systems development)

The aim of the MARTES project is the following: The definition, construction, experimentation, validation and deployment of a new model-based methodology and an interoperable toolset for Real-Time Embedded Systems development, and the application of these concepts to create a development and validation platform for the domain of embedded applications on heterogeneous platforms architectures.

MISSA

http://www.offis.de/projekte/v/225/missa_e.php

Methodology and tools for formal linking between safety requirements on different design levels; MISSA project already planned to integrate their results in existing platforms like RTP

Modelisar

www.itea2.org/public/project_leaflets/MODELISAR_profile_oct-08.pdf

Modelisar is a project that intends to support rapid control prototyping in the automotive domain by defining an interface between models of the automotive embedded system (includes AUTOSAR models) and the controlled system.

ModelPlex

www.modelplex.org

MODELPLEX (*MODELling solution for comPLEX software systems*) is an IST project funded from call 2.5.5 and will last for 36 months.

MODELPLEX will be driven by Industrial Use Cases ensuring the applicability and the integration of the different technologies produced by the academics and industrial partners. This approach will allow an iterative process where the technology providers will receive continuous feedback from the Industrial Use Cases implementer and benefit from a richer and immediate return on experience.

MODELPLEX will define and develop a coherent infrastructure specifically for the application of MDE to the development and subsequent management of complex systems within a variety of industrial domains, where “complexity” is characterized by a combination of size, heterogeneity, legacy system management, dynamicity, distribution and autonomy of systems.

Mogentes

www.mogentes.eu

The MOGENTES (1.1.2008 - 31.12.2010) project, EU FP7, has the goal to enhance testing and verification of dependable embedded systems by means of automated generation of test cases. The aims are to cover both functional safety and reliability aspects of embedded systems verification.

OMG

www.omg.org

International organisation that amongst other activities develops standards for information interchange like UML, SysML, CORBA.

OpenEmbeDD

openembedd.inria.fr

OpenEmbeDD is an Eclipse-based "Model Driven Engineering" platform dedicated to Embedded and Real-Time systems (E/RT).

Its aim is to offer engineers who design and develop E/RT software the means to express, simulate, validate and test the targeted system before any component has solded on a circuit board.

PREVENT

www.prevent-ip.org

PReVENT is a (finished) FP6 IP that focused on developing Advanced Driver Assistance Systems that help drivers avoid accidents by informing them about potential dangers, warning them if there is no reaction to the information and actively assisting or ultimately intervening if necessary. The project primarily addressed concept development but also some methodology.

SAFEAIR II

http://www.ist-world.org/ProjectDetails.aspx?ProjectId=00c8e416d4af4a38ba774a4bec443b4a&SourceDatabaseId=9cd97ac2e51045e39c2ad6b86dce1ac2

SafeAir II will secure the leading edge ASDE (Avionics System Development Environment) tool set and its associated methodology developed in the IST SafeAir 1999-10913 project, while including relevant improved functionalities for end users and demonstrating dramatic cost effectiveness. Beyond SafeAir results, SafeAir II (finished) will result in a complete and coherent methodology and development framework to be customised in each industrial company involved in the embedded systems development, to be able to demonstrate the Y life-cycle in secure conditions.

SAFEDOR

www.safedor.org

Safedor is a large FP6 IP (integrated project) ending in 2009, developing techniques for risk-based design of ships. A number of sub-projects are developing methods for model-based safety analysis and optimisation of engineering systems on-board ship including programmable embedded systems. Techniques and tools that underpin this work include HiP-HOPS, a dependabiliy analysis tool further developed in ATESST-2, and Simulation X, a simulation tool that implements Modellica.

SAFESPOT

www.safespot-eu.org

Safespot is a FP6 IP with the objective to understand how intelligent vehicles and intelligent roads can cooperate to improve road safety. This is done by extending the time horizon for acquiring safety relevant information for driving, as well as to improve the precision, the reliability and the quality of driver information, and to introduce new information sources.

SETTA

www.vmars.tuwien.ac.at/projects/setta/index1.htm

SETTA (finished) was a FP5 project on systems engineering of time-triggered-architectures. The application domain was safety-critical, distributed, real-time applications such as fly-by-wire or drive-by-wire.

SPEEDS

www.speeds-eu.com

The IP SPEEDS (Speculative and Exploratory Design in Systems Engineering), an Eu FP6 project ending in Oct 2009, aims at providing a rich component model (HRC) for component based engineering, in particular providing multiple views for different engineering disciplines. SPEEDS builds on existing standards like SysML or AUTOSAR. It defines the modelling concepts, methodologies and analysis mathematics while incorporating them in an environment of commercial development tools.

SQUALE

www.squale.org

SQUALE (Security, Safety and Quality Evaluation for Dependable Systems), an open source project started in june 2008, sponsored by System@tic's 5th Call for projects SQUALE defines a set of dependability assessment criteria covering all dependability attributes independent from a specific application domain.

TIMMO

https://www.timmo.org/

TIMMO is an ITEA2 project (ITEA 2 project 06005) that started in April 2007 and extends to September 2009.

TIMMO is developing a common, standardized infrastructure for the handling of timing information during the design of embedded real-time systems in the automotive industry. This will shorten the development cycle and also increase its predictability.

Appendix B Industry activities

This list describes industry activities in areas related to EAST-ADL2 scope.

AUTOSAR – AUTOmotive Open System Architecture

www.autosar.org

The AUTOSAR standard will serve as a platform upon which future vehicle applications will be implemented and will also serve to minimize the current barriers between functional domains. It will, therefore, be possible to map functions and functional networks to different control nodes in the system, almost independently from the associated hardware.

MISRA

www.misra.org.uk

Develops amongst other things standards for how languages are used in a safe way, MISRA C, MISRA C++ and MISRA SL/SF, which are de-facto standards for how software in the automotive industry should be written.

SWAP

www.vinnova.se/misc/VINNOVA-projekt/Projekt---Listhuvud/15534/

Project that aims at developing an application platform for Autosar compliant development. This development includes dedicated tools, hardware, Autosar compliant basic software and a test-bench where an application of software components can be verified.

Appendix C Languages

Open or standardized modelling languages that are currently used for designing real-time safety critical embedded systems:

AADL (Architecture Analysis & Design Language)

http://aadl.info
AADL is developed by a Society of Automotive Engineers (SAE) sponsored committee of experts and was approved and published as SAE Standard AS-5506 in November 2004. AADL is designed for the specification, analysis, and automated integration of real-time performance-critical (timing, safety, schedulability, fault tolerant, security, etc.) distributed computer systems.

ALTARICA --> Chen

altarica.labri.fr

ALTARICA is a Data-Flow modelling language that can be seen as a generalization of Block diagrams and Petri nets. It is based on the notion of mode automata that are finite states automata with inputs and outputs flows. It is a hierarchical where components exchange information by means of two mechanisms: flows that propagate values through the model and synchronization of events that forces two or more events to be simultaneous.

BIP

www-verimag.imag.fr/~async/bip.php

BIP is a modelling language for concurrent systems that considers that components are the superposition of three distinct layers describing, respectively “Behaviour, Interaction and Priority”. Interaction involves synchronization between components behaviour with possible transfer of data.

CCM

www.omg.org

The Corba Component Model, CCM, is an OMG standard designed for expressing distributed component based applications.

ESTEREL

www-sop.inria.fr/meije/esterel/esterel-eng.html

ESTEREL is both a programming language, dedicated to programming reactive systems, and a compiler, which translates Esterel programs into finite-state machines. It is one of a family of synchronous languages, like Lustre or Signal/Polychrony, which are particularly well-suited to programming reactive systems, including real-time systems and control automata. Esterel is the kernel language for the Esterel Studio toolset developed by Esterel-Technologies.

FIACRE

www-sop.inria.fr/oasis/fiacre

FIACRE is a French acronym for “Format Intermédiaire pour les Architectures de Composants Répartis Embarqués” (Intermediate Format for Architectures of Embedded Distributed Components). FIACRE is a formal intermediate modelling language to represent both the behavioural and timing aspects of systems –in particular embedded and distributed systems- for formal verification and simulation purposes.

LUSTRE

www-verimag.imag.fr/~synchron

LUSTRE is a synchronous declarative language for programming reactive systems. It is declarative because a description is a set of equations that must always be verified by the program variables. A program variable in Lustre is considered to be a function of multi-form time: it has an associated

clock, which defines the sequence of instants where the variable takes its values. In that sense, Lustre belongs to the family of synchronous languages (like Esterel and Signal/Polychrony).

MODELICA

www.modelica.org

Modelica is free available language specification for an object-oriented modelling language for large, complex, and heterogeneous physical systems.

It is suited for multi-domain modelling, for example mechatronic models in automotive, aerospace and robotics applications involving mechanical, electrical, hydraulic and control subsystems.

SPEEDS HRC

SPEEDS HRC meta-model – called Heterogeneous Rich Components (HRC) – supports a design representation of electronic components based on several layers of abstraction. HRC components are conform to components of SysML and follow an assume/promise approach (so-called contract-based approach) where each component has a black-box model, which explicates assumptions about its environment and state corresponding promises on the service offered by the component to the environment. The HRC meta-model definition provides a common meta-model, including different viewpoints (functional as well as extra-functional ones) with a system-wide rigorous formal semantics.

SystemC

www.systemc.org

SystemC is a IEEE Standard 1666™-2005, it is a language built in standard C++ by extending the language with the use of class libraries. SystemC addresses the need for a system design and verification language that spans hardware and software. The language is particularly suited to model system's partitioning, to evaluate and verify the assignment of blocks to either hardware or software implementations, and to architect and measure the interactions between and among functional blocks.

VHDL and VHDL-AMS

www.eda-stds.org

VHDL, VHSIC Hardware Description Language,, IEEE standard 1076 and derivative is commonly used as a design-entry language for field-programmable gate arrays and application-specific integrated circuits in electronic design automation of digital circuits. VHDL was originally developed at the behest of the US Department of Defense in order to document the behaviour of the ASICs that supplier companies were including in equipment.

VHDL-AMS is a derivative of VHDL (IEEE standard 1076-1993). It includes analog and mixed-signal extensions (AMS) in order to define the behaviour of analog and mixed-signal systems (IEEE 1076.1-1999).

UML (Unified Modelling Language) and derivatives SysML, MARTE

www.uml.org

UML is a graphic modelling language structured on a meta-model defining the modelling elements (concept handled by the language) and the semantics of these elements (definitions and meaning of their uses). It is a formal language structured around three categories of diagrams: Structure diagrams, Behaviour diagrams, and Interaction diagrams (that can be considered as a sub-category of behaviour diagrams).

SysML is a modelling language based on version 2.0 of UML (Unified Modelling Language) developed to meet systems engineering requirements. SysML is not simply an UML profile. It comprises a subassembly of the UML 2.0 language (limited, to simplify its learning and implementation in the tools) and an extension of the UML 2.0 language, containing new structures and diagrams required for systems engineering.

MARTE is the UML profile for Modelling and Analysis of Real- Time Embedded systems. It provides support for specification, design, and verification/ validation stages. This new profile is intended to replace the existing UML Profile for Schedulability, Performance and Time (SPT). MARTE consists in defining foundations for model-based description of real time and embedded system for hardware and software.

MARTE also provides features for performance and schedulability analysis.

Appendix D Tools

A list of commercial, non-commercial and research tools that related to safety, architecture, AUTOSAR modelling of electrical systems.

ARALIA

www.arboost.com

ARALIA which has been developed by Arboost based on technologies designed for ALTARICA, is a computation engine for Boolean risk assessment models (Fault Trees, Bloc Diagrams, Event Trees...) relying on the Binary Decision Diagrams technology.

ARALIA is integrated in the ARALIA workshop and in the Cecilia OCAS toolset, which are both distributed by Dassault Data Service (see http://www.dassault-data-services.fr).

ASCET (Advanced Simulation and Control Engineering Tool)

ASCET, developed by ETAS is a product family for model-based design of embedded automotive software. As an authoring tool for ECU Software it is mainly used by function- and software-developers in the automotive industry (OEM and supplier) to develop software for control-functions and -algorithms. The tool has a block-diagram style similar to Simulink, but is (as mentioned) tailored for the use in the automotive industry. From a safety point of view, the tool is quite unique, because ASCET's code generator is the first one, which is certified for the use in IEC 61508 SIL 3 rated projects.

AUTOSAR Tools

www.vector.com

The Tool Environment DaVinci supports the complete workflow for design, configuration, simulation, test and deployment of AUTOSAR compliant software for electronic control units. The Environment consists of the following modules:

- DaVinci System Architect for the design of the distributed system.
- DaVinci Network Designer for the network communication design and schedule definition.
- DaVinci Developer for the application design and RTE (Run-time environment) configuration.
- Microsar Configuration Suite for the configuration of the AUTOSAR basic software.

www.dspace.com

The following tools are provided from dSPACE for design, configuration and deployment of

AUTOSAR compatible software:

- SystemDesk for definition of the vehicle and software system architecture,
- TargetLink with an AUTOSAR blockset for application design,
- Tresos for the basic software configuration and generation of the RTE.

http://www.artop.org/

The AUTOSAR Tool Platform (Artop) is an implementation of common base functionality for AUTOSAR development tools. Artop, including its source code, is available free of charge to all AUTOSAR members and partners. The Artop development process is transparent and based on a community approach driven by AUTOSAR members and partners. The community that develops Artop is organized as the Artop User Group.

Doors

Doors is a commercial requirement management tool.

Eclipse based tools

www.openembedd.org

OPENEMBEDD is an Eclipse-based "Model Driven Engineering" platform dedicated to Embedded and Real-Time systems (E/RT). Its aim is to offer engineers, who design and develop E/RT software the means to express, simulate, validate and test the targeted system before any component has solded on a circuit board.

EPF Composer

Used to document process-flows in TIMMO and Autosar.

pure::variants

www.pure-systems.com

pure::variants is the tool to outline and manage efficiently all parts of software products with their components, restrictions and terms of usage. With this set of information and with the continuous tool support throughout the entire software configuration process valid solutions are created automatically from the choosen features.

EXITE ACE

http://www.extessy.com/?id=9fec65c63b007e50b906bff21c854729

EXITE ACE is a framework for

• Virtual integration and testing of systems of components, providing

• Real-time and non-RT simulation supported

• Distributed / cluster simulation

• AUTOSAR component testing

• MIL/SIL/HIL execution capabilites for Simulink, Targetlink, Dymola, Rhapsody, Artisan Studio, ASCET.

GeneAuto

www.geneauto.org

GeneAuto is a toolbox for automatic code generation from a subset of Simulink/Stateflow ® and Scicos modelling languages to imperative programming languages (currently C easily extensible to Ada, C++, Java, C#...) compliant with safety critical system certification rules (DO178, ECSS…). The toolbox provides several elementary tools, which are used to build tool chains satisfying different kinds of constraints (traceability, resource use minimisation…). Some of the tools are developed using formal technologies in order to reduce the qualification costs for the tool chains.

MODELICA based tools

Several tools based on Modelica are available, for example, Dymola has been developed by Dynasim (see http://www.dynasim.se), and Mathmodelica by Mathcore (see www.mathcore.com)

Ptolemy, Ptolemy II

http://ptolemy.eecs.berkeley.edu/index.htm

The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined *models of computation* that govern the interactions between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation.

Scade

http://www.esterel-technologies.com/products/scade-suite/

SCADE is a graphical modelling language that has been developed by Verilog (based on Airbus and Schneider Electric requirements), which has been bought by Telelogic and then by Esterel-Technologies (see http://www.esterel-technologies.com). It is based on the Lustre synchronous model of computation.

SIGNAL/POLYCHRONY

www.irisa.fr/espresso/Polychrony

SIGNAL/POLOCHRONY is an integrated development environment and technology demonstrator for computer-aided embedded software design. It is based on a synchronous multi-clocked model of computation implemented in the data-flow language Signal. The toolset consists of a compiler, a model checker and control synthesiser.

Matlab/Simulink/Targetlink tools

Matlab/Simulink, www.mathworks.com

Commonly used language for behaviour definition in automotive systems development.

Targetlink, www.dspace.com

Tool chain with execution platforms with customizable hardware, target code generator using a subset of the Simulink blocks available. Often combined with simulation hardware for experimental vehicles enabling the verification of production models in real vehicles.

EXACT, http://www.extessy.com/?id=3b7efa09444a31c5d58596e5bbf87d47

EXACT is a test environment closely integrated with Simulink and Targetlink suitable for functional validation of components running model-, software- and processor- in-the-loop tests and code generator qualification.

Scilab/Scicos

www.scilab.org

Scilab/Scicos is a graphical dynamical system modeller and simulator toolbox included in the Scilab ® engineering and scientific computation software. With Scicos you can create block diagrams to model and simulate the dynamics of hybrid dynamical systems and compile your models into executable code. Scicos is used for signal processing, systems control, queuing systems, and to study physical and biological systems. New extensions allow generation of component based modelling of electrical and hydraulic circuits using the Modelica language.

SYNDEX

www-rocq.inria.fr/syndex

SYNDEX is a CAD software based on the AAA methodology (Algorithm Architecture Adequation). It allows specifying application algorithms and distributed architectures, as well as to perform their adequation by exploring manually and/or automatically the possible implementations while satisfying real-time constraints. It automatically generates the code corresponding to the chosen implementation. Suited for rapid prototyping it allows hardware/software codesign.

SystemWeaver

www.systemite.se

SystemWeaver is a MBD software that allows the storage of design decisions, requirements and other system relevant data in a database. The information structure follows the system structure

used by the modeler rather. It uses a customer defined meta-model for the data, and makes versioning of all entities from systems to individual requirements possible. It allows for generation of specifications out of the information in the database according to user defined rules.

Topcased

www.topcased.org

Topcased is a software development environment primarily dedicated to the realization of critical embedded systems including hardware and/or software.

Topcased promotes model-driven engineering and formal methods as key technologies. Topcased is released as free/libre/open-source software by a group of partners from various organisations.

Parts of Topcased are also included in OpenEmbedd previously described. Here is a list of components from Topcased Ganymede version (Topcased version 2, July 2008) useable in the context of ATESST2 project:

- Model editors (UML2 editor, SysML editor, AADL editor, SAM editor, EMF editor, TOPCASED-MF),

- Model transformation features (FIACRE, SMUC, UML to C/Java/Python code generation, Model To Doc), model simulation and verification tools (TOPCASED Model Simulation tools, TOPCASED-VF,

- Integrated development environment based on ECLIPSE (Cchecker, GNATBench)

- Interoperability and transversal services (TOPCASED-Bus, plug-ins for remote tools connections to the TOPCASED environment, change management facilities, requirement traceability mean (TRAMWAY), configuration management (TVM))