Developing Automotive Products Using the EAST-ADL2, an AUTOSAR Compliant Architecture Description Language

P. Cuenot[1], P. Frey[2], R. Johansson[3], H. Lönn[4], M.-O. Reiser[6], D. Servat[5], R. Tavakoli Kolagari[6], D.J. Chen[7]

1: Siemens VDO Automotive SAS, a Continental Corporation company, 1av. Paul Ourliac, BP 1149, 31036 Toulouse, France
2: ETAS GmbH, Borsigstr. 14, 70469 Stuttgart, Germany
3: Mentor Graphics Corporation, Automotive Networking Business Unit, Theres Svenssons Gata 15, SE-417 55 Gothenburg, Sweden
4: Volvo Technology Corporation, Mechatronics and Software, SE-405 08 Gothenburg, Sweden
5: CEA LIST, Gif sur Yvette F-91191, France
6: Technical University of Berlin, Software Engineering Group, D-10587 Berlin, Germany
7: Royal Institute of Technology, SE-100 44 Stockholm, Sweden

Abstract: Current development trends in automotive software feature increasing standardization of the embedded software structure. But it still remains the critical issue of the overall engineering information management to control the system definition and manage its complexity. System modeling based on an Architecture Description Language (ADL) is a way to keep these assets within one information structure. The original EAST-ADL was developed in the EAST-EEA project (www.east-eea.org) and basic concepts were reused in the AUTOSAR standardization initiative. The original EAST-ADL is currently refined in the ATESST project (www.atesst.org) to EAST-ADL2. This paper presents the results of the language extension provided by the EAST-ADL2 domain model and focuses on its possible extension of the AUTOSAR standard to support decomposition of E/E automotive systems.

Keywords: Modeling, Abstraction, ADL, AUTOSAR

1. Introduction

Current development trends in automotive software feature increasing standardization of the embedded software structure, in particular manifested by the AUTOSAR standardization initiative. The AUTOSAR consortium [1] defines a generic software architecture platform by standardization of its infrastructure and a communication layer suitable for distributed hardware architectures. The specification of application software components is standardized, such that these can be reused and integrated on the AUTOSAR platform by a third party. Software reuse is thus favored and implementation-specific dependencies between application software and hardware is avoided.

The AUTOSAR approach improves the OEM/supplier development relation and data interchange. Higher quality and dependability is foreseen and cost and complexity can be managed appropriately.

The AUTOSAR standard for E/E architectures is becoming more and more mature; its release 3.0 has been completed in December 2007. Automotive OEMs are planning to use AUTOSAR for series production and all new development of embedded automotive software will in future be compliant to it.

But there are still some numbers of issues outside the scope of this standardization initiative that are necessary for managing the engineering information and its assets attached to system definition.

System modeling based on an architecture description language (ADL) is a way to keep the engineering information in a well-defined information structure. In the context of the ATESST project we have reused and extended the original EAST-ADL language primarily developed in the EAST-EAA project. We consider that Model Based Development (MBD) as supported by the EAST-ADL is complementary to the AUTOSAR approach. Through this combination, it is possible to support system modeling down to the componentization level (AUTOSAR). It is thus a means for efficient development and management of the complexity of automotive embedded systems: Concepts from MBD and CBD reinforce one another [2]. We will demonstrate how EAST-ADL2 solves the challenge of the full integration of the two, by describing engineering information supported by the EAST-ADL2 language.

The important complements to AUTOSAR represented by the automotive domain specific language EAST-ADL2, are:

- requirements modeling and tracing including capability for specific adaptation,
- feature modeling including concepts to support product lines,

- structural and behavioral modeling of functions and hardware entities in the context of distributed systems, and
- other information, such as a definition of function timing and failure modes, support of system analysis.

A main result of the ATESST project is the EAST-ADL2 domain model, released as a public UML2 profile. The following sections describe the EAST-ADL2 with reference to the AUTOSAR standard and how this combination allows modeling of E/E automotive systems.

Section 2 presents the structural view of the language constructs including structural relation to AUTOSAR. Section 3 gives details on behavioral modeling including behavioral relation to AUTOSAR. Section 4 describes requirements modeling being orthogonal to the other models. Section 5 addresses applied product line techniques and variability modeling essential for the automotive domain. Section 6 describes timing modeling support of the language. Section 7 provides error modeling overview for analysis of failure propagation. Finally, conclusions are drawn with consideration of the designers' and end-users' perspectives.

2. Structural view of EAST-ADL2

The following section describes the organization of EAST-ADL2 with respect to structural view to describe how to capture system information and how models are related to each other.

EAST-ADL2 structural overview

EAST-ADL2 is an architecture description language defined as a domain-specific language for the development of automotive electronic systems. It includes modeling entities to describe features, requirements, variability, software and hardware components, and specific annotations associated to models to support the analysis of the system.

The core concept of the structural organization of EAST-ADL2 is the description of the models in different abstractions levels (see Figure 1). The electronic functions/features are described at different levels of abstraction, reflecting the details of the architecture and implicitly different stages in the engineering process. The different artifacts drive the functional decomposition of the functions from abstract models down to implementation in software components and hardware elements of the system architecture.

Modeling of the electronic systems of a vehicle starts with capturing the functions at the Vehicle level with product line organization and description.

These functions are realized at the Analysis level by abstract entities describing models of software functions (such as "ADLFunction") and devices (such as "FunctionalDevice") that interact with the vehicle environment. The Analysis level captures the principal interfaces and the behavior of the subsystems of the vehicle.

On the Design level, models are refined with more implementation-oriented aspects that allow a subsequent software decomposition of the functional architecture. Devices are split into elements of the hardware architecture such as sensors or actuators, and the software parts for signal transformation (such as "LocalDeviceManager"). Middleware is modeled to project the platform specific services and functionality to the functional level The hardware architecture is introduced in parallel to capture the hardware entities as abstract elements (e.g. I/O, sensor, actuator, power, ECU, electrical wiring including communication bus) to describe the topology of the electronic architecture of the systems. The overall structure is such that one or several entities can be later realized by AUTOSAR entities. Design level allows preliminary allocation of software entities and provides the basis for verification either by simulation or analysis techniques such as timing and dependability modeling.

The implementation of software components such as basic software and detailed software topology is not defined in EAST-ADL2, whereas it is in AUTOSAR; this is why we propose to use AUTOSAR for the implementation levels. Full traceability is supported from function definitions at the Vehicle level to AUTOSAR entities. De facto, Operational level is hidden by AUTOSAR concepts via deployment of the configured AUTOSAR Run Time Environment.

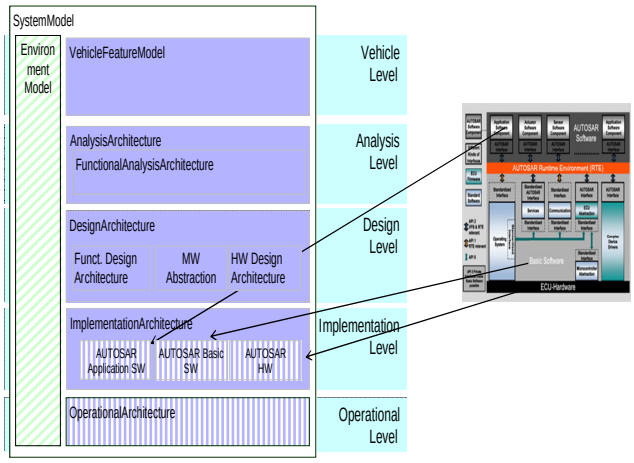

Figure 1: EAST-ADL2 abstraction layers and relation to AUTOSAR.

Figure 1 depicts the substitution of the implementation level of EAST-ADL2 by AUTOSAR, and how abstract EAST-ADL2 concepts match AUTOSAR: FunctionalDesignArchitecture defines application functionality of the AUTOSAR application software architecture, the Middleware Abstraction

represents the functionality of the, basic software architecture and the hardware design architecture corresponds to AUTOSAR topology and hardware entities.

Relation EAST-ADL2 – AUTOSAR

As described above, the structural relation between EAST-ADL2 and AUTOSAR is complementary. EAST-ADL2 provides modeling artifacts for functional modeling supporting a refinement process for decomposition of features to artifacts, such as software and hardware elements, while keeping track of the system architecture description. Software implementation details are not captured as AUTOSAR provides lower level implementation mechanisms to support full description of software components and related information and techniques for deployment in the AUTOSAR standardized platform.

The “ADLFunction” introduces a degree of freedom for the AUTOSAR software implementation architecture, via the dedicated association “ADLBehavioralMapping” to map “ADLFunction" to "Runnable Entities” of AUTOSAR. This mechanism allows packaging multiple “ADLFunction” entities to software components and to optimize various aspects during implementation.

“LocalDeviceManager” is the interface functionality for sensors and actuators and is realized by the “SensorActuatorSoftwareComponent” of AUTOSAR.

The port concept of EAST-ADL2 is inherited from SysML for data descriptions (such as “ADLFlowPort”) and service interaction (such as “ADLClientServerPort”). The entities match the AUTOSAR “SenderReceiver” and “ClientServer” ports. However, AUTOSAR port configuration and RTE services include mechanisms abstracted on the level of EAST-ADL2.

The hardware architecture of EAST-ADL2 is complementary to the AUTOSAR system topology and ECU resource definition and configuration. AUTOSAR focuses on the impact of software implementation based on the required configuration of the various software elements (bus topology including communication signal allocation, "Runnable Entities" allocation to task from operating system, driver configuration in relation to ECU pins ...). EAST-ADL2 entities abstract the overall hardware topology to capture physical elements of the vehicle electronic architecture and wiring harness. Some elements are present on both sides but with abstract representations: “ECU”, “IOPort” and “Communication” bus as elementary entities are not decomposed at this level. A more complex relation is abstracted with hardware ports of EAST-ADL2 (as “ADLHWPort”). This entity is realized by platform “HAL” service to map software IO driver and hardware pin configuration at AUTOSAR Level.

Finally, interfaces of complex device drivers and AUTOSAR platform services independent of hardware allocation are captured by “ADLFunction” entities in middleware.

3. Behavior models

The goal of EAST-ADL2 with respect to behavior is to describe how model components (from different tools, in different modeling languages, or just representing code) are related to each other in order to capture behavior and algorithms of the vehicle systems as well as the environment. A report publicly available summarizes the work conducted during the ATESST project (see [3]). Two main issues are accounted for here: 1) behavioral semantics of the “ADLFunction” entities and 2) mapping to AUTOSAR behavioral constructs.

EAST-ADL2 behavioral semantics

In EAST-ADL2, behavior modeling 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 behavioral composition and to make the function execution independent of behavioral notations: inside each function, the data transformation could be described according to various languages and paradigms, and various legacy tools including general UML tools and domain-specific tools (e.g. Simulink, ASCET).

The triggering of each function is defined by time or an event on one of the input ports. A precedence constraint construct allows definition of any constraint that needs to be honored when the scheduling and task allocation is made.

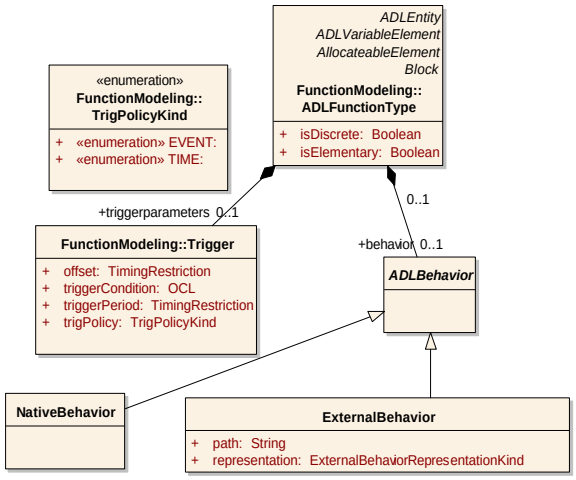

Figure 2: “ADLFunction” entities and triggering aspects

There are two types of “ADLFunction” entities, time discrete and time continuous (see Figure 2). Each time the function is invoked, the read-execute-write

pattern is performed, i.e. all inputs are read at the same time and outputs are written at the same time. For discrete functions, this is done after a computation delay, “ExecutionTime”. Time continuous functions (“ADLContFunction”) define the transfer functions from input to output, and the computation rate is infinite. In practice, invocations are implicitly defined by the solver tool used to simulate or analyze a set of connected functions. In order to allow different behavioral definitions, all inputs are read simultaneously, and all outputs are written simultaneously.

Time discrete “ADLFunction” entities, “ADLDiscFunction”, have an explicitly defined invocation pattern. Functions may be time-triggered, in which case time alone causes execution to start. Event triggered functions may be invoked due to data arrival or calls on the input ports. This is defined by the “TrigPolicy” attribute. “TriggerPeriod” defines the time between invocations for time trigged functions or the inter-arrival time between invocations for event-triggered functions. Offset is the offset of invocations relative to a nominal period for time trigged functions. It is not applicable to event triggered functions A textual/OCL “TriggerCondition” rule may be used to define the conditions for when the function is invoked. For example, a port value has to exceed X or data must arrive on more than one port. In order for offsets and periods to be meaningful, the assumption regarding synchrony has to be declared. This is done with the “TimeBase”, which lists all “ADLFunction” entities that share the same time base (not shown on figure). This may be realized by co-allocation to the same ECU, or co-allocation to a set of ECUs with a shared clock (e.g. based on FlexRay).

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 according to the behavioral semantics of EAST-ADL2.

These concepts are implemented as stereotypes applicable to UML classes and UML behavioral constructs such that information can be shown both in composite structure diagrams and behavioral diagrams, such as activity diagrams. “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.). “NativeBehaviors” are mapped directly to UML “Behaviors” such that “StateMachines”, “Activities”, or “Interactions” – depicted as sequence diagrams – can be used w.r.t to modeling needs. The application of EAST-ADL2 stereotypes on these UML concepts alters UML2 semantics such that among other things, triggering policies and run-to-completion assumption hold (see [3]).

Relation EAST-ADL2 – AUTOSAR

As said previously, AUTOSAR concepts are organized as the implementation level of EAST-ADL2. Elementary “ADLFunction” entities are to be mapped to AUTOSAR behavioral units, which are the “RunnableEntities” contained in “AtomicSoftwareComponents” (SWC). Different mappings can be made, depending on the implementation configuration. It is also possible to leave out the traceability to "RunnableEntities", if this level of detail is unnecessary, or if component internals are yet unknown or confidential.

The ports of EAST-ADL2 models are transformed into ports on the AUTOSAR SWC. One or several EAST-ADL2 ports may be realized by one AUTOSAR port, as these may have several “signals” or data elements per interface.

Figure 3 below shows an example on how elementary “ADLFunction” entities are mapped to three AUTOSAR SWCs. Each elementary “ADLFunction” has its own logical execution thread and no internal concurrency. It therefore maps well to a “Runnable”. Note that the ports follow the elementary “ADLFunction” entities to the AR SWCs, and that this is one out of several possible mappings.

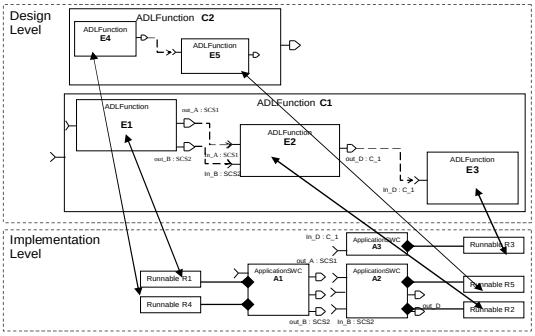

Figure 3 Mapping of EAST-ADL2 "ADLFunctions" to AR Runnables

One further aspect of the AUTOSAR behavior to consider is the execution condition of the runnable entities. These are executed in the context of a task policy but controlled by the Run-time environment (RTE) for execution condition. RTE manages all invocation of atomic software components, and in particular the scheduling of the runnables by triggering mechanism with RTE-Events. The triggering parameters associated to “ADLFunction” are considered as an abstraction of the AUTOSAR mechanism in a simplified context of synchronous execution (read-execute-write) for EAST-ADL2.

The execution semantics of AUTOSAR runnables can either be "asynchronous" (runnables triggered by event) or "synchronous" (runnables triggered by time or event period). Both concepts match the

execution semantics of the elementary "ADLFunction". Sleep mode and waiting point are not covered but are implementation issues. Behavior of functions in sleep mode via associated services (sending event and waiting point) are dedicated to real time operating systems, to gain CPU resource for application performance. From a design perspective there is no need to describe this behavior.

The matching between RTE events and EAST-ADL2 triggering mechanism can be summarized as follows:

AUTOSAR RTE events	ADLFunction triggers
Timing Event for periodical execution	TrigPolicy: 'periodic' TriggerPeriod:value Offset:none TriggerCondition:NA
DataReceivedEvent	TrigPolicy: 'event' TriggerPeriod:value Offset:none TriggerCondition:ADLFlowPort
OperationInvokedEvent	TrigPolicy: event TriggerPeriod:value Offset:none TriggerCondition:ADLClientServer
DataSendCompleteEvent	Not Applicable (see text justification)
WaitPoint	Not Applicable (see text justification)

The execution of "RunnableEntities" managed by AUTOSAR RTE configuration can thus be specified by the semantic properties of the "ADLFunction". AUTOSAR runnable may exhibit more complex behavior as well, but this is excluded from the EAST-ADL2 behavioral semantics in order to preserve analyzability.

4. Requirements models

Requirements are captured in EAST-ADL2 according to the principles of SysML[4]: Requirements are separate entities that are associated to its target elements with a specific association, "ADLSatisfy".

Requirements are related to each other to support traceability between requirements. Typically, requirements on the higher abstraction levels of EAST-ADL2 are refined to more detailed requirements on lower abstraction levels.

Verification and Validation is supported through the concept of Verification& Validation Cases. A "VVCase" is linked to requirements and target entities, in order to show how a certain requirement is verified in the context of a specific model entity. EAST-ADL2 distinguishes between the model entity that is verified to meet the requirement, from the target system.

An important aspect of traceability is the possibility to follow which requirements are the results of safety concerns. This is needed to comply with the upcoming automotive standard for safety, ISO 26262[5]. EAST-ADL2 also supports this standard by providing support for safety case, safety integrity levels and error propagation (see further section 7).

The Requirements Interchange Format (RIF)[6] has been considered in the ATESST project to represent Requirements from external tools. RIF is a general standard that supports the interchange of requirements, but also other engineering information. Due to this generality, an ADL such as EAST-ADL2 cannot support the full RIF without loosing stringency, and a subset is chosen instead.

5. Variability models

In order to give an overview of variability management in EAST-ADL2, we examine two questions:

- In what development situations and contexts is variability management needed? Or: For what parts of EAST-ADL2 is variability management support provided?
- What is the basic modeling means used for variability modeling and to which of these development situations/contexts are they applicable?

Needs

First, variability management starts on the Vehicle Feature Level, where model range features and variability is viewed. At this point, the purpose of variability management is to provide a highly abstract overview of the variability in the system such as the complete system together with dependencies between these variabilities. A "variability" in this sense is a certain aspect of the complete system that changes from one variant of the complete system to another. "Abstract" here means that for an individual variability it is not the idea to define how the system varies with respect to this variability but only that the system shows such variability. For example, the front wiper may or may not have a rain sensor. On vehicle level the impact of this variability on the design is not defined; only the fact that such variability exists is defined by introducing an optional feature named 'RainSensor'. This is later validated and refined during analysis and design.

While the details of how variability is actually realized in the system are largely suppressed on the vehicle level, they are just the focus of attention when managing variability on other areas of the development process. In fact, certain variability may lead to modifications in any development artifact, such as requirements specifications, and functional

models. With respect to EAST-ADL2, three areas are to be distinguished: (1) requirements, (2) the artifacts AnalysisArchitecture, DesignArchtiecture and ImplementationArchitecture and (3) test artifacts. Here, describing that certain variability occurs is not sufficient; it is necessary to describe how each variability concept affects and modifies the corresponding artifact.

Basic modeling means

Having answered question no. 1 above, we can now turn our attention to the second question: the basic modeling means provided as support for variability management in these different situations. They are: feature modeling, product decision modeling and multi-level feature trees.

The purpose of feature modeling is to define the commonalities and variabilities of the product variants within the scope of a product line. Usually feature models are used on a high level of abstraction, as described above for vehicle level variability.

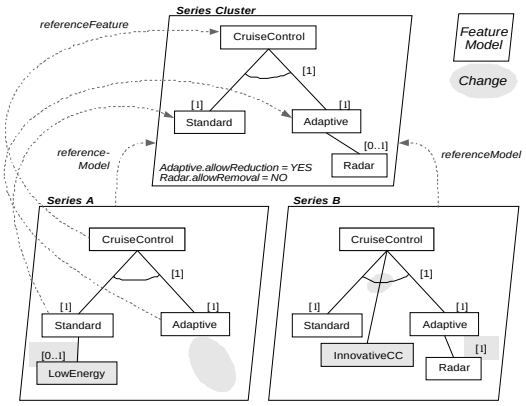

Figure 4. Reference model and referring models in the multi-level feature modeling approach.

However, in EAST-ADL2, they are also used on design level and get a much more concrete meaning there. Product decision modeling on the other hand is aimed at defining configuration: The configuration of a feature model *fa* – i.e. the selection and de-selection of its features – is defined in terms of the configuration of another feature model *fb*. A product decision model can thus be seen as a link from *fb* to *fa* that allows deriving a configuration of *fa* from a given configuration of *fb*. Finally, multi-level feature trees (see Figure 4) are a means to strategically manage two or more separate, independent product lines. With this instrument at hand, not all variants of the complete system need to be managed within a single, extremely complex global product line. It is instead possible to subdivide the product line into smaller, subordinate product lines (called product sublines) without loosing the possibility to manage them from a global perspective.

Variability management on the artifact level is driven by the variability captured on the VFM. This means that the main driver for variability and also variability instantiation is the Vehicle Feature Model. Variability on the artifact level essentially consists of the use of variation points and simple feature models (i.e. non-multi-level feature models) at the public interface of functions.

The basic idea of the artifact level variability modeling is that whenever variability on the respective abstraction level occurs, a variation point is introduced (see Figure 5). The variation point is introduced and is linked via its variation point configuration to the possible variants, which may replace the variation point. The variation point can be replaced by either of the specified variants. If the variation point has the cardinality [0..1] the variation point as a whole can be deselected, meaning that the variation point is optional. The variation point in Figure 5 is not optional (marked by the cardinality [1]) and can only be replaced by one of its two variants. The variation point has one input port and two output ports. Variant 1 only needs one output port, whereas variant 2 needs two output ports. So the variation point has the superset of ports of all possibly replacing variants. But those ports that are not used in all circumstances are marked by the cardinality [0..1], meaning that they are optional. Also, the port expecting input from an optional output port is marked by the cardinality [0..1] (i.e. implicit optional) in order to reflect that this port cannot in all instantiations expect input from this port, which must be coped with in the behavior of the respective function.

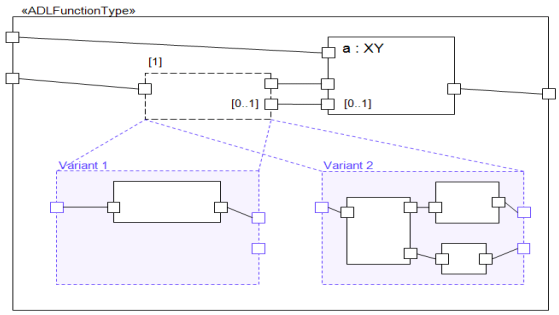

Figure 5. An "ADLFunction" as a variation point with two variants.

To summarize the discussion on ports we can say:

- An optional port is a port in a variation point that is not used by all variants.
- An implicit optional port is the port of a function that will always appear in the implementation, but it will not always be used:
  - an implicit optional output port means:
    à data will always be sent; but there are cases when there is no other function that *consumes* this data.

- an implicit optional input port means:
  à data is always expected; but there are cases when there is no other function that actually *sends* such data.

By introducing implicit optional ports it is possible to reduce explicit variability modeling, because the function with an implicit optional port does not need to be variable as a whole.

As depicted in Figure 6, the starting point of variability modeling is on the VFM level. The core feature model of the VFM influences the instantiation of artifact variability. Each single artifact entity (especially ”ADLFunction“) has at its public interface a parameterized feature model that is a function feature model (see Figure 6), without being a multi-level feature model. These public interfaces describe internal variability of the functions. This is needed to have an adequate overview of the respective variability of the functions.

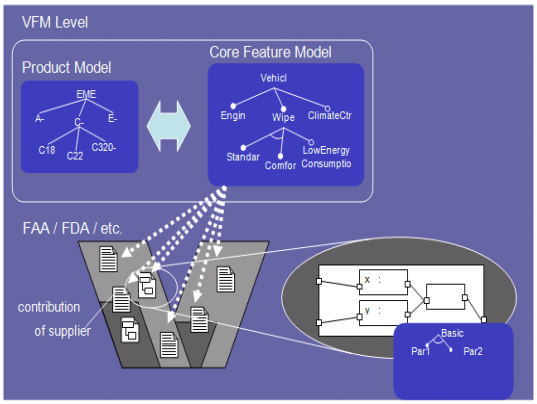

Figure 6. Function feature model at the public interface of a function.

Besides using feature models at the public interface of single modeling entities, feature models are used to describe the variability of a complete product sub-line at an artifact (e.g. FDA).

In order to make the development of individual subsystems independent from the whole system, artifact lines are introduced. This is of specific interest for suppliers because they use their own product line approach for the subsystems they develop. So the artifact line as a whole has an own feature model, whose instantiation is driven once again by the core feature model of the VFM level using product decisions.

In order to extend language elements of other languages or standards connected to EAST-ADL2 (like AUTOSAR, SysML) with the variability approach of EAST-ADL2, the respective element is enhanced by a generalized link to the ”ADLVariableElement“ (see Figure 7). Hence, the variability technique of EAST-ADL2 can also be used for these elements.

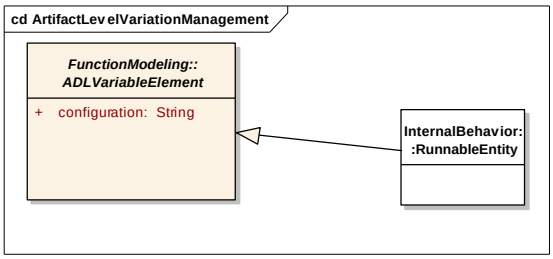

Figure 7. Extend AUTOSAR ”RunnableEntity“ to a variable element.

6. EAST-ADL2 Timing Modeling Support

Why automotive timing is important

Many automotive functions are control applications which often impose hard real-time requirements on their form of realization in the E/E system. Such control applications can mainly be found in the power-train domain (e.g. engine management, transmission control) and chassis domain (electronic stability program, antilock braking system), but also in the body domain where even non-control applications have timing requirements (indicator, window lifter). For control applications, it is important that the sensing of input data, actual control algorithm computations and output data actuation are in synchrony with the speed of the controlled plant and with its proceeding dynamics. In distributed embedded systems, the task of implementing potentially distributed control applications becomes truly challenging, especially when multiple sensors and multiple actuators are involved where determinism and consistency in both the time and value domain for the involved sensors and actuators are required.

Timing as integrated engineering information and relation to AUTOSAR

The EAST-ADL2 provides support to capture specific engineering information which is relevant for the timing of automotive functions. Conceptually, timing information can be distinguished into timing requirements (what is demanded), timing properties. (what is offered), and timing contracts (what is negotiated between stakeholders). This is in-line with how OMG in the UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE)[7], represents non-functional properties as of either required, offered or contract nature. The general notion is that the actual timing properties of a solution (=implementation, realization) must satisfy the specified timing requirements stemming from the automotive function specification.

Figure 8 presents an overview sketch how the considered timing information is perceived in general in the EAST-ADL2 system model. Note that the start of the arrows describes the origin of timing related engineering information, and the direction of the

arrow (top-down or bottom-up) describes their inter-abstraction level relation. The concept of contract based timing is a way of decomposing the complex problem of end-to-end timing, which may depend on detailed implementation details from a number of different companies not willing to share too much data among each other. The timing contracts thus also enables complete verification on a higher level of abstraction, without having all implementation details on lower level of abstraction present. As EAST-ADL2 is still being developed, the language currently covers aspects of the timing requirements part on the two functional abstraction levels only (Analysis Level and Design Level in Figure 8).

Figure 8. Timing information as perceived in the EAST-ADL2 system model

Ensuring operational correctness

To ensure operational correctness with respect to timing, validation and verification activities (V&V) can be performed on every abstraction level. However, different techniques provided through different tools or a combination of those might have to be applied. Note that these V&V techniques may require specific input in the form of execution time bounds which can be derived in different quality and by different techniques themselves (e.g. measurements, mathematical code analysis). The so derived timing properties of the system or function are checked against timing requirements which have been derived either directly from a contract or through refinement from the original function specification on higher levels of abstraction (Analysis and Design levels).

In the EAST-ADL2, timing requirements are distinguished into various kinds of delays (= latencies) as well as specific timing requirements for the temporal synchronization of input data or output data respectively. Delays can either be end-to-end delays from sensors to actuators or atomic delays or composed delays for the single time consuming modeling entities which are part of a timing chain. The latter type of delays are referred to as timing-chain-segments of an end-to-end timing chain (=end-to-end delay). End-to-end delays are subject to segmentation along the functional decomposition track, i.e. when progressing from a functional model on Analysis Level to a refined functional model on Design Level to an implementation model in terms of AUTOSAR software component architecture on Implementation Level. Note that the level of end-to-end timing delays is not automatically following the level of abstraction. For example can the number of elementary "ADLFunction" timing segments present on the Design level, be much higher than the number of (composed) software component timing segments on the (AUTOSAR) implementation level, when representing the same end-to-end timing.

User challenges

Clearly, the EAST-ADL2 users' challenge will be to perform a functional decomposition and refinement of top-level functions from Analysis Level to Design Level while considering the segmentation of the end-to-end delays into single timing chain segments at the same time. By introducing well defined segments EAST-ADL supports the concept of timing contracts, thus fitting to the automotive world where the responsibility for fulfilling end-to-end timing requirements will continue to be split between a Car OEM and a number of TIER1s. The proposed EAST-ADL2 will have to prove its soundness and applicability in future evaluations. However, only recently the demand for additional levels of abstractions and separation of concerns has been raised and should not be overheard [8].

The general concept of traceability of engineering information between multiple levels of abstraction, which is also applicable to timing information, enables sound and comprehensible design decisions which are documented in the overall system model.

7. EAST-ADL2 Error Modeling Support

Why error modeling is necessary

Automotive embedded systems are inherently safety critical due to the devices and dynamics under control. In other words, an error of such systems, if not detected and properly handled, could endanger human life and result in damages to environment and property. Currently, embedded software is increasingly introduced and integrated in modern automotive systems, accounting to a large portion of innovations and also to growing product complexity. Meanwhile, the competitive forces are also driving for short time-to-market, cost efficiency, configuration flexibility and accommodation of new technologies, etc. For these reasons, the design of safety in embedded automotive systems is posing an increasing challenge for the developers. While state-of-the-art safety analysis techniques providing analysis support for deriving the causes and consequences of errors, the difficulties remain in capturing and maintaining plausible errors, safety

requirements, and other related information while performing design refinement, changes and evolution, and in providing arguments that a system is safe enough.

The overall objective of the EAST-ADL2 error modeling is to allow an explicit reasoning of functional safety and thereby to facilitate safety engineering along with an architecture design or maintenance process. As an overall system property, safety is concerned with the abnormalities (in terms faults, errors, and failures) and their consequences under given certain environmental conditions. Functional safety represents the part of safety that depends on the correctness of a system operating in its context [9]. In other words, it addresses the hazardous abnormalities of a system in its operation (e.g., component errors and their propagations).

Error modeling as an analytical view extension to nominal architecture

EAST-ADL2 facilitates safety engineering in regards to the system modeling and information management. While supporting the safety design through its intrinsic architecture modeling and traceability support, the language allows the developers to explicitly capture the error logics in terms of component errors and the error propagations in an architecture error model through its error modeling support (see also Figure 9). The error modeling is treated as a separated analytical view orthogonal to the nominal architecture model. This separation of concern in modeling is considered necessary in order to avoid some undesired effects of error modeling, such as the risk of mixing nominal and erroneous behavior in regards to the comprehension, reuse, and system synthesis (e.g., code generation).

The EAST-ADL2 error modeling package extends a nominal architecture model, typically at the functional and design levels, with the information of failure semantics and error propagations. The failure semantics can be provided in terms of logical or temporal expressions, depending on the analysis techniques and tools of interest. Such analytical information, together with environmental conditions, forms the basis for identifying the likely hazards, reasoning about the causes and consequences, and thereby deriving safety requirements. The relationships of local error behaviors are captured by means of explicit error propagation ports and connections. Due to these artifacts, EAST-ADL2 allows advanced properties of error propagations, such as the logical and temporal relationships of source and target errors, the conditions of propagations, and the synchronizations of propagation paths. Hazards or hazardous events are characterized by attributes for severity, exposure and controllability. A hazardous event may be further detailed by e.g. use cases, sequence or activity diagrams. In an architecture specification, an error is allowed to propagate via design specific architectural relationships when such relationships also imply behavioral or operational dependencies (e.g., between software and hardware). In EAST-ADL2, a safety requirement derived from the safety analysis has attributes specifying the hazard to be mitigated, the safety integrity level (ASIL/SIL), operation state, fault time span, emergency operation times, safety state, etc [5],[9]. The safety requirement is then traced to or used to derive other nominal requirements such as in regards to safety functions and performance.

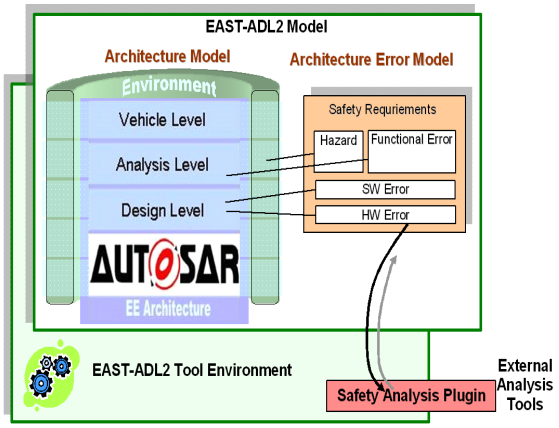

Figure 9. EAST-ADL2 Architecture Error Model as a separate architecture view extending the nominal architecture model and providing analysis leverage through external tools**.**

Analysis leverage through external tools and other engineering support

Given an error model, the analysis of the causes and consequences of failure behaviors can be automated through external tools. There is currently a (prototype) analysis plug-in in the Eclipse environment, allowing the integration of the HiP-HOPS tool (Hierarchically Performed Hazard Origin and Propagation Studies) [10] for static safety analysis in terms of FFA, FTA, and FMEA. The analysis leverage includes fault trees from functional failures to software and hardware failures, minimal cut-sets, FMEA tables for component errors and their effects on the behaviors and reliability of entire system.

Further supports of EAST-ADL2 for safety engineering include the integration of safety case, referring to a technique that provides a structure for the qualitative argumentation about why a system is safe [11]. The integration, which is currently under development, aims to enable the development of safety case based on the architecture and error information captured in EAST-ADL2. This complements other standardization efforts such as in

the ISO WD 26262 [5] and in the MISRA safety guidelines [12], which emphasize the importance of safety case but provide no coverage on the content and implementation.

8. Conclusion

The ATESST project defining the EAST-ADL2 language has been running for the last two years (since beginning of 2006). During this period there have also been running two initiatives having a major impact on the development of automotive E/E systems: AUTOSAR [1] and the ISO working group on functional safety for road vehicles (ISO TC 22/SC 3/WG 16) [5]. Both these are aiming at standardization and at enabling the development of safety-critical automotive E/E systems while the complexity of these systems are growing rapidly. The focus of safety in AUTOSAR has especially grown in its second phase starting 2007.

The implications of these two initiatives on the definition of EAST-ADL2 have been obvious. The applicability of an automotive ADL will be heavily dependent on its conformance to AUTOSAR and to the upcoming safety standard ISO/WD 26262.

Thus, the evolvement of EAST-ADL2 has had a focus of being complementary to AUTOSAR, supporting modeling the more abstract levels, thus extending AUTOSAR in a consistent way.

In ISO/WD26262 there are requirements on the existence of a so called safety case. A safety case is a tool that provides structure to the qualitative argumentation about why a system is safe. This is done by separating the argumentation from the facts or justifications and providing an explanation of the relationships and dependencies between them.

EAST-ADL2 includes a meta-model for a safety case, thus enabling safety case development in close connection to the system model. Furthermore, modeling systems in EAST-ADL2 makes it possible to provide explicit descriptions of faults in functions, software and hardware, and the mechanisms by which they can propagate. Such descriptions in turn facilitate safety analysis techniques like FTA and FMEA, generating evidences for the safety cases.

In addition, the traceability, consistency and rigor required when developing safety-related systems are well supported by an ADL.

To conclude, EAST-ADL2 both supports the generation of a safety case as prescribed in ISO WD 26262, as the generation of its supporting evidence.

Having had the opportunity to define this architecture description language in parallel with the dynamic phases of the definitions of AUTOSAR and ISO WD 26262, EAST-ADL2 has a good potential to become a de facto standard as it fits well with the major critical needs of the automotive industry of today. To enable a wide spread use of EAST-ADL2, the domain model is now released as a public UML2 profile.

9. Acknowledgement

This work was supported by contribution of all the partners of the ATESST project consortium funded by the European Commission. We wish to acknowledge useful participation of all partners and feedback from the anonymous reviewers.

10. References

[1] AUTOSAR Development Partnership, http://www.autosar.org

[2] M. Törngren, D.J. Chen, I. Crnkovic, “Component-based Development vs. Model-based Development: A Comparison in the Context of Vehicular embedded Systems”, Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications.

[3] ATESST consortium, Report on behavioral modeling within EAST-ADL2, D3.2 deliverable, http://www.atesst.org/, December 2007.

[4] SysML Partners. Systems Modeling Language (SysML) open source specification project, http://www.sysml.org

[5] International Organization for Standardization: ISO Working Draft 26262 Baseline 10, 2007.

[6] HIS, Specification Requirements Interchange Format (RIF), version 1.1a, 2007.

[7] OMG, UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE), Beta1, OMG Document Number: ptc/07-08-04, August 2007.

[8] Alberto Sangiovanni-Vincentelli, Marco Di Natale: “Embedded System Design for Automotive Applications”, IEEE Computer, Volume 40, Issue 10, IEEE Computer Society Press, October 2007.

[9] International Electrotechnical Commission Functional safety of electrical/electronic/ programmable electronic safety-related systems – Part 0: Functional safety and IEC 61508, 2005.

[10] Y.Papadopoulos, J.A. McDermid: “Hierarchically Performed Hazard Origin and Propagation Studies”. SAFECOMP, 1999.

[11] T.P. Kelley, PhD thesis: "Arguing Safety - A Systematic Approach to Managing Safety Cases", University of York, 1998.

[12] The Motor Industry Software Reliability Association: "Development Guidelines for Vehicle Based Software", MISRA, 1994.

11. Glossary

ATESST: Advancing Traffic Efficiency and Safety through Software Technology
CBD: Component Based Development
E/E: Electronic Environment
HAL: Hardware Abstraction Layer
IO: Input Output
MBD: Model Based Development