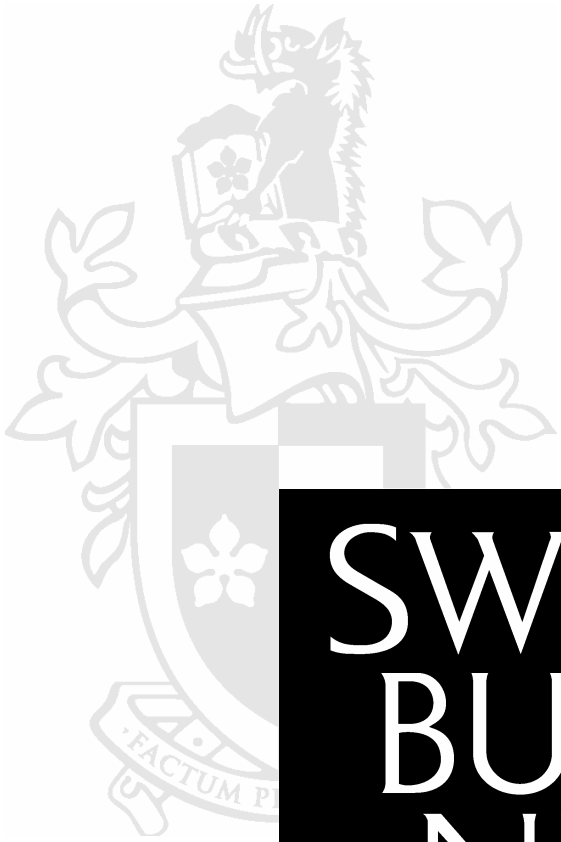
Faculty of Information and Communications Technologies

Architecture Description Languages for Automotive Systems – A Literature Review

Technical Report: C4-01 TR M49

Stefan Björnander, Mälardalen University, Sweden
Lars Grunske, Swinburne University of Technology, Australia
30th July 2008

.



SWINBURNE UNIVERSITY
OF TECHNOLOGY

Table of Contents

Swinburne University of Technology

Architecture Description Languages for Automotive Systems – A Literature Review

Abstract

An Architecture Description Language (ADL) can be described as a language designed to model a system at an architectural level with respect to its software, hardware, and communication links. Due to the increasing complexity of software systems in areas like embedded control and web-based information systems, modelling with ADLs have gained attention in the research community and in practical software development projects. The specific aim of this technical report is to provide a literature review on ADLs for automotive software systems. This literature review consequently focuses on aspects that are relevant for automotive systems like safety, reliability and modelling of Electronic Control Units (ECU).

1 Introduction and Background

Even though ADLs have been around for some time, there is no real consensus in the research community about what an ADL is and what distinguishes it from formal specifications, simulations or programming languages. Consequently, it is hard to determine when the first ADL was launched. However, the Specification and Description Language (SDL) represents an early language targeting system specification. SDL started in 1972 and has been developed by the International Telegraph and Telephone Consultative Committee (ITTCC). It was originally designed to specify and describe the functional behaviour of telecommunication systems. Later on, however, SDL has been used in various application domains (Holz, 2001).

In the eighties, the avionic industry took an interest in ADLs. The objective was to shift the focus of the application from lines-of-code to coarser-grained elements and their interconnection structure (Lewis et al., 2002). MetaH represents one of the first ADLs used in the avionic industry. It is a language and a toolset for describing, analysing, and implementing real-time mission critical computer systems for avionics control applications. In the beginning of the millennium, MetaH was extended into the Avionics Architecture Description Language (AADL). The name was later changed to the Architecture Analysis and Design Language in order to illustrate its intended all-purpose usage (Feiler et al., 2000), see section 3. At the same time, Orccad was jointly developed by the BIP project and the Robotics Department of INRIA Rhône-Alpes in order to model robotics control system (Simon et al. 1993).

In the middle of the nineties, there was a virtual explosion of ADLs. Many languages were introduced for a number of different applications. ACME is an ADL developed by Carnegie Mellon University in 1995. It is intended to be a common interchange format for other architecture design languages and tools (Garlan et al., 1997). Unicon is a software ADL, also developed by Carnegie Mellon University in 1995. The focus of Unicon is on supporting the descriptions of architectural abstractions and styles found in various software systems and on constructing new systems from the architecture descriptions (Shaw et al., 1995). Rapide is a language and toolset developed by Stanford University in 1996, with the aim of supporting component-based development of distributed systems (Luckham and Vera, 1996). Wright is a software ADL developed by Carnegie Mellon University in 1997. It supports the formalization of architectural styles and model verification and validation (Allen, 1997). ACME, Unicon, Rapide, and Wright are only a subset of the ADLs created in the nineties; Medvidovic and Taylor (2000) and El-khoury et al. (2002) provide more complete surveys of ADLs. More recently, the xADL 2.0 (Dashofy et al., 2005) has been introduced; it is an approach to develop extendable ADLs based on XML 2.0. π-ADL (Oquendo, 2004) is an ADL based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures.

In the beginning of the new millennium, the automobile industry realized that ADLs could be very useful tools when modelling the ever increasing complexity of the hardware and software systems of a modern car. The automobile industry is facing new challenges during the first few decades of the new millennium. There are estimations that electronics make 90% of the innovations, 80% out of which occur in the area of software (Hardung et al., 2004). There are several ongoing projects whose purposes are to address the complexity of car electronic systems, as described in section 2.

1.1 Structure

Section 2 surveys technologies and architectures relevant to the automotive industry:

- 2.1: AUTOSAR, an architecture platform developed for car electronic systems.
- 2.2: UML, SysML, and MARTE. UML is a language designed to model object-oriented systems, SysML and MARTE are UML extensions designed to model real-time systems.
- 2.3: ATESST and EAST-ADL. EAST-ADL is an ADL developed by ATESST, a car manufacturer consortium.
- 2.4: ADL for modelling processors and electronic control units (EDUs). Some ADLs designed to model processors are introduced.

Section 3 covers AADL, an all-purpose ADL, originally developed for the avionic industry, which can be applied to the automotive domain. Section 3 also provides a demonstration of AADL code and describes the AADL Error Annex.

2 ADLs for Automotive Systems

During the first decade of the new millennium, several large projects whose purposes are to address the complexity of automobile systems have been launched. AUTOSAR is an open and standardized architecture platform focused on the electronic systems of automobiles. UML is a graphical language for modelling object-oriented systems. SysML and MARTE are extensions of UML designed to model real-time systems. EAST-ADL is an ADL developed by a car manufacturer consortium to model car electronic systems.

2.1 AUTOSAR

The discussions about AUTOSAR (Automotive Open System Architecture, www.autosar.org) started in 2002 when representatives of several car manufacturers and tool vendors formed a consortium. The consortium includes, as core members, the BMW Group, Continental, Daimler, Ford, Opel, PSA Peugeot Citroën, Toyota, and Volkswagen AG. The architecture consists of 54 Basic SoftWare Modules (BSWM), each one of them responsible for a specific service, such as head lights or wind sweeper. For each BSWM, a specification is defined, but the implementation decisions are not specified. The idea is that the manufacturers compete with each other when implementing the software specified by the BSWM (Freund et al., 2008). Consequently, AUTOSAR can be used to specify a component interfaces without risking disclosing implementation specific information.

Since the developers implement the software code for the BSWM and can, in theory, install it in any car model, AUTOSAR will hopefully result in a higher degree of component reuse and high quality software. For the software developer, the software library Runtime Environment (RTE) is available, which in turn uses the MicroController Abstraction Layer (MCAL) to communicate with the hardware. The key benefit of AUTOSAR is that it provides a standard language to specify component interfaces and their interactions. As AUTOSAR provides the software developer with RTE in the same way as Windows provides the developer with a standardized API, AUTOSAR has been described as a “Windows for Car Electronics” (Tångring 2008).

2.2 UML, SysML, and MARTE

The Unified Modelling Language (UML) is based on James Rumbaugh's OMT (Object-Modelling Technique) method, Grady Booch's method for object-oriented design, and Ivar Jacobson’s OOSE (Object-Oriented Software Engineering) method.

In 1996, the three researches decided to address the issue that the modelling languages of the day did not support the object-oriented theory. An international consortium called the UML Partners was organized under their leadership to complete the first version of the UML specification. In January 1997, the UML Partners' UML 1.0 specification draft was proposed to the OMG (Object Management Group, www.omg.org). After some debate between the group members, they decided that classes shall be represented by boxes as in Rumbaugh's OMT method rather than the clouds of the Booch method. In November 1997, UML 1.1 was adopted by OMG (Larman, 2004).

UML2 is the major revision of UML, it was adopted by OMG in 2003. The UML2 standard constitutes of four parts (Arlow and Neustadt, 2005): the superstructure that defines the notation and semantics of the diagrams and their model elements, the infrastructure that defines the core meta model on which the superstructure is based, the Object Constraint Language (OCL) for defining rules for model elements, and the UML Diagram Interchange that defines how the UML2 diagram layout are exchanged.

UML is not an ADL and was not intended to be one. However, there are UML2 profiles that can describe system architectures (http://www.omg.org/technology/documents/profile_catalog.htm). A UML2 profile is an extension (well defined in OCL) of a subset of UML2 adapted to a specific area. SysML and MARTE are two UML2 profiles adapted to real-time applications.

SysML was originally suggested by the International Council on Systems Engineering (INCOSE) Model Driven Systems Design workgroup in 2001. They wanted to customize UML for system engineering applications. In 2003, SysML Partners (an informal association of industry leaders and tool vendors) distributed the first source specification of SysML. SysML was issued as a UML2 profile by OMG in 2007.

SysML uses seven of UML2's diagram types and adds two of its own: the requirement and parametric diagram types. The requirement diagrams can, when modelling an automobile system, be used to capture functional, performance and interface requirements. The parametric diagrams can be used to define performance, security and mechanical constraints (Grunske and Joyce, 2008).

MARTE (Modelling and Analysis of Real-Time and Embedded systems, OMG 2007) is a UML2 profile for real-time embedded systems. In 2005, the SPT (Schedulability, Performance and Time) profile was released. MARTE is an extension of that profile and was released in 2007 (Gérard et al., 2007).

As MARTE is designed to model real-time systems, it supports non-functional properties (NFP). The NFP Modelling Framework is a set of extensions to specify non-functional annotations. Its main part is the Value Specification Language (VSL), which is an application language supporting mathematical expressions, time expressions and variables. It is possible to declare variables of integer, real, time, date, tuple, and interval types (Espinoza, 2007).

The MARTE architecture consists of three main packages (Faugère, 2006):

- Marte Foundations. The Marte Foundations package defines all basics concepts required for model-based design and analysis of real-time systems. It includes concepts for modelling non-functional properties, time-related concepts, and a general resource model for platforms.
- Marte Design Model. The Marte Design Model allows the developer to design the system from requirement analysis to specification, design and implementation.
- Marte Analysis Model. The Marte Analysis Model defines specific model abstractions to be used by external tools. It is divided into three parts, focusing on quantitative analysis techniques, schedulability, and performance analysis.

2.3 ATESST and EAST-ADL

EAST-ADL (Electronic Architecture and Software Technology – Embedded Electronic Architecture) is an ADL that has been designed by a consortium driven by the European automotive industry. EAST-ADL is intended to support the development of automotive embedded software by capturing all the related engineering information. The scope is the hardware and software of the embedded automobile systems and its environment. The development and maintenance of EAST-ADL started in 2001 as part of the ATESST (Advancing Traffic Efficiency and Safety through Software Technology) project, which objective is "the Definition of a comprehensive, standardized ADL for the automotive domain based on the existing approach EAST-ADL, including Software and Hardware Components, Communication, Environment modelling, Requirements and V&V, Variant handling and product families." (Cuenot et al., 2007).

The second version, EAST-ADL2, is based on UML2, SysML, and MARTE. SysML concepts are being reused wherever applicable. Adaptation to MARTE is done by integrating MARTE concepts where real-time and embedded system properties are modeled.

EAST-ADL is designed to describe electronic functionalities on several abstraction levels from high level user-visible electronic features down to implementation details. EAST-ADL also supports the modelling of non-structural aspects, such as requirements behaviour, description and validation, and verification activities (De Bruyne, 2004).

The information model of the language is hierarchically composed of five levels of abstractions, each with corresponding system representation in parenthesis (Lönn, 2007):

- Vehicle. At the vehicle level, the electronic architecture is only seen from an external (end-user) perspective (Vehicle Feature Model).
- Analysis. The analysis level supports the analysis of the vehicle electronics (Analysis Architecture).
- Design. At the design level, platform-independent software is defined (Design Architecture).
- Implementation. At the implementation level, platform-independent code is defined. The RunTime Environment of AUTOSAR is integrated at the implementation level (Implementation Architecture).

- Operational. At the operational level, the final binary software is defined. The MicroController Abstraction Layer of AUTOSAR is integrated at the operational level (Operational Architecture).

In order to support the development of complex automotive systems, EAST-ADL includes language features to specify required properties of the system at varying degrees of abstraction, to trace requirements between system refinements, and to refine the specification of requirements.

EAST-ADL differentiates between functional, quality, and safety requirements. A requirement can be traced from the abstract vehicle model all the way to the final hardware and software components. Moreover, EAST-ADL offers detailed means to explicitly model central artifacts of verification and validation activities (Sjöstedt et al., 2007).

As variability is becoming increasingly important to the automotive domain, an important focus of the development of EAST-ADL is to support variability management and product-line oriented development. EAST-ADL's variability management concept is able to analyze, define and evolve the variability of the automotive system (Gérard et al., 2007).

In a model driven approach, models at different levels of abstraction are used as primary artefacts throughout the software development process. When modelling a system, three levels of abstraction are normally used. At the highest level, a platform independent model of the application is built. The middle level is a platform specific model. The lowest level is code in a platform specific implementation language. EAST-ADL includes seven artefacts; four of them are intended for the application software: Vehicle View, Functional Analysis Architecture, Functional Design Architecture, and Logical Architecture. The remaining three artifacts are intended for the implementation: Hardware Architecture, Technical Architecture, and Operational Architecture.

An important objective of the ATESST project is the development of a modelling tool. There is an Eclipse-based tool available; it has support for profile editing and application.

2.4 ADLs for Processor and ECU modelling

An important field of application for ADLs is to model processors. In Qin and Malik (2005), such ADLs are partitioned into three categories: structural, behavioral, and mixed. The structural ADLs focus on the structural details on the microarchitectural level, which makes them suitable for modelling cycle accurate processors. The behavioral ADLs, on the other hand, describe the instruction sets of the processor, which makes them suitable to model compilers and instruction-set simulators.

However, due to the low resolution of the partition above, most ADLs fall into the mixed category. To address that issue, Qin and Malik (2005) present a new categorization based on the microarchitecture and instruction set of the processor. The categories are: the Discrete Event Models, the Synchronous Structural Model, the Synchronous Behavioral Model, and Petri-net-based models.

When it comes to modelling digital circuits, the Discrete Event Model is the model of choice. It was original used when designing the ADL MIMOLA (Zimmerman, 1979), which further development eventually lead to the development of the MSSH hardware synthesizer, the MSSQ code generator, the MSST self-test program compiler, the MSSB functional simulator, and the MSSU RTL simulator (Leupers and Marwedel, 1997).

As most processors are implemented as synchronous logic circuits, the Synchronous Structural Model is a popular choice when modelling processors. As they do not need an event calendar, they are suitable for simulation. Asim (Emer et al., 2002) is a processor-modelling environment developed by Compaq for modelling high performance processors. In the model, the module communication is driven by a clock.

The next category is the Synchronous Behavioral Model. UPFAST (Önder and Gupta, 1998) is an example of an ADL in that category. In order to gain simulation speed, the modules communicate by global states. The drawback of the method is that the model becomes less module. LISA (Pees et al., 1999) belongs to the category of domain-specific models. The characteristic feature of ADLs belonging to this category is that abstract pipeline stages are modeled. The last category is the Petri-net-based models (Murata, 1989). They are based on the mathematical foundation of Petri nets.

3 AADL

AADL (Architectural Analysis and Design Language, aadl.info) is a large and complete language intended for the design of both the hardware and the software of a system. It is an SAE (Society of Automotive Engineers, www.sae.org) standard and is based on MetaH and UML (Feiler et al., 2006). Compared to

MARTE, AADL is constrained in one respect: a specific phase the development life cycle is addressed, and other stages can't be addressed by AADL (Faugere, 2007).

The component abstractions of the AADL are separated into three categories. The first category is the application software:

- Thread. Can execute concurrently and be organized into thread groups
- Thread Group. Component abstraction for logically organizing threads or thread groups components within a process.
- Process. Protected address space whose boundaries are enforced at runtime.
- Data. Data types and static data.
- Subprogram. Model of a subprogram component that represents a callable piece of source code.

The second category is the execution platform (the hardware):

- Processor. Schedules and executes threads.
- Memory. Stores code and data.
- Device. Represents sensors and actuators that interface with the external environment.
- Bus. Interconnects processors, memory, and devices.

The third category is the system component. System components are composites that can consist of other systems as well as software or hardware components.

The components types are defined using a parameterized set of properties. Furthermore, components communicate with each other through ports. It is possible to define physical port-to-port connections as well as logical flows through chains of ports. Component definitions are divided into component types holding the public (visible to other components) features, and component implementations that define the private parts of the component.

The AADL standard includes runtime semantics for mechanisms of exchange and control of data, including message passing, event passing, synchronized access to shared components, thread scheduling protocols, and timing requirements.

AADL can be used to model and analyze systems already in use as well as to design new systems. AADL can also be used in the analysis of partially defined architectural patterns. Moreover, AADL supports the early prediction and analysis of critical system qualities, such as performance, schedulability, and reliability.

Within the core language, property sets can be declared that add new properties for components. Additional models and properties can also be included by utilizing the extension capabilities of the language. The properties and extensions can be used to incorporate analyses at the architectural design level.

AADL components interact through defined interfaces. A component interface consists of directional flow through data ports for state data, event data ports for message data, event ports for asynchronous events, subprogram calls, and explicit access to data components. Application components have properties that specify timing requirements such as period, worst-case execution time, deadlines, space requirements, and arrival rates (Feiler et al. 2007).

There is a number of tools developed for AADL. One of them is OSATE (Open Source AADL Tool Environment, downloadable at aadl.info), which is a plug-in for the Eclipse environment (www.eclipse.org). It supports analysis and simulation of AADL models. Another analysis extension is AADL's Error Annex, which provides the capability to annotate AADL components with dependability related information called AADL Error Models.

In this section, first the general concepts of AADL will be introduced and the case study of a fire alarm system will be partially specified in AADL to provide an example. Afterwards, the specific concepts to define an annotation in terms of the AADL's Error Annex will be described.

3.1 Introduction to AADL

AADL components are specified through component types and component implementations (Feiler et al. 2006). A component type defines the component's interface in terms of interaction points (data, event and event data ports) with other components and externally observable properties. A component type can be defined as one of three component categories: application software, execution hardware and composite system (system, for short). AADL application software component types include thread, thread group, process, data or subprogram. The hardware component type includes processor, memory, device (e.g., a sensor or an actuator), and bus components.

```
system implementation FireAlarmSystem.impl
   subcomponents
      HW: processor PIC;
      AU: process AlarmUnit;
      WD: system WatchDog;
      SS: device SmokeSensor;
      SP: device Sprinkler;
   connections
      C0: event port fire breaks out -> SS.fire breaks out;
      C1: event port SS.smoke detected -> AU.smoke detected;
      C2: event port AU.sprinkle -> SP.sprinkle;
      C3: event port AU.i am alive -> WD.i am alive;
      C4: event port WD.are you alive -> AU.are you alive;
      C5: event port WD.reset -> HW.reset;
   properties
      Actual Processor Binding => reference HW applies to AU;
end FireAlarmSystem.impl;
```

A component implementation defines the internal structure of a component. It may declare a set of sub-components and defines how the ports of the sub-components are connected as well as what application software is deployed on which execution hardware. Furthermore, a component implementation may define a set of operational modes of the components and transitions between these modes. Additionally, simple property sets such as security-levels, scheduling parameters, etc. and complex annotations defined in annex libraries can be attached to components types and component implementations. As an example, the component implementation of the fire alarm system is specified above. In this specification, the AADL section on subcomponents defines the used components, and the section on connections specifies which of their input and output ports are connected. Finally, the section properties defines that the process AU is executed on the processor HW.

3.2 AADL's Error Annex

AADL's Error Annex provides the capability to annotate AADL components with dependability related information called AADL error models. Similar to architectural AADL models, error models have two levels of description: an error type level and an error instance/implementation level. An error model at the type level, defines a set of error states. These error states can also describe error free states. An error model type further defines a set of error events and its occurrence probabilities, if known. These error events can be normal error events as well as other events such as repair events. The occurrence probability of an error event can be static (defined by the language keyword `fixed`), exponentially distributed (defined by the language keyword `poisson`) or can have a user-defined non-standard distribution (`non-standard`).

```
package FireAlarmErrorLib
public
annex error model {**
    error model SensorErrorModel
    features
        error free: initial error state;
        unavailable, babbling: error state;
        smoke detected omission: out error propagation {Occurrence => fixed 1};
        smoke detected commission: out error propagation
                                   {Occurrence => poisson 1E-3};
        fail stop, fail babble: error event;
    end SensorErrorModel;

    error model implementation SensorErrorModel.Standard
    transitions
        error free ->[fail stop]-> unavailable;
        error free ->[fail babble]-> babbling;
```

```
        unavailable ->[out smoke detected omission]-> unavailable;
        babbling ->[out smoke detected commission]-> babbling;
    end SensorErrorModel.Standard;
…
**};
end FireAlarmErrorLib;
```

An error model implementation declares the error transitions between error states. These transitions can be triggered by internal error events or by error propagations from external components. (Note that most existing safety evaluation methods use the term failure propagations instead of error propagations.)

Each AADL error model can be stored in a library and can be (re)used for different AADL components. Consequently, an AADL error model can describe the error behaviour of a set of similar components. As examples, several different error models for simple hardware and software components have been defined in Feiler et al. (2007). Above an extract of the error model annex library `FireAlarmErrorLib` for the fire alarm system is shown. Specifically, the error model for a sensor is defined. This error model contains three states, one for the correct behaviour of the sensor (`error free`) and two error states (`unavailable` and `babbling`). In the error state `unavailable` the sensor omits to send the message `smoke detected` in case a fire occurs and in the error state `babbling` the sensor falsely sends the message `smoke detected`. In the first case the error is propagated to the environment with a fixed probability of one, meaning that the senor will always omit to send the message. In the second case the wrong message is sent with an exponentially distributed probability with the rate 10^{-3} per second. The transitions between the error states are defined in the implementation of the error model `SensorErrorModel.Standard`.

```
device implementation SmokeSensor.simple
…
annex error model {**
    Model => FireAlarmErrorLib::SensorErrorModel.Standard;
    Occurrence => poisson 1E-7 applies to error fail stop;
    Occurrence => poisson 1E-7 applies to error fail babble;
**};
end SmokeSensor.simple;
```

As shown above, error models from the error annex library can be annotated or associated to an architectural component `Model => FireAlarmErrorLib:: SensorErrorModel.Standard;`. When an architectural component is annotated with an error model it is further possible to tailor it with specific information such as occurrence probabilities for error events. In this example, the occurrence rates for internal error events `fail stop` and `fail babble` have been defined. By using the same statement also already defined properties can be overwritten.

To analysis an AADL error model there are currently two approaches available. The first approach automatically translates an error model into a standard fault tree (Joshi et al. 2007). The second approach generates Generalized Stochastic Petri Nets (GSPNs) from error model specifications and uses existing tool for quantitative analysis (Rugina et al. 2006).

4 Conclusions

During the last three decades, the architecture description languages have evolved and as we approach the end of the first decade of the new millennium, there are several powerful modelling tools available for the automobile industry. The introduction of information technology in automotive systems has led to functions unimaginable a decade ago. However, with such highly advanced and complex functions, appropriate design methods and tools have become crucial.

The dominating standards on the automobile market of today are AUTOSAR and OMG. In ATESST, the entities corresponding to software components and hardware components are taken from the AUTOSAR standard, put in a context where the EAST-ADL system modelling concepts can be used. Similar to SysML and MARTE, AADL is likely to become an UML2 profile in the near future since it is partly based on UML2.

5 References

Allen, R.-J. A Formal Approach to Software Architecture, Ph.D. Thesis, Carnegie Mellon University, Technical Report Number: CMU-CS-97-144, May, 1997.

Arlow J. and Neustadt, I.UML2 and the Unified Process: Practical Object-Oriented Analysis and Design, 2nd Edition, Addison-Wesley Professional, 2005.

Chen, D. (ed). EAST-ADL assessment and update suggestions. Advancing Traffic Efficiency and Safety through Software Technology (ATESST). www.atesst.org/home/liblocal/docs/ATESST_Deliverable_D2.2.2_V1.1.pdf.

Cuenot P., Chen D.-J., Gerard S., Loenn H., Reiser M-O., Servat D., Kolagari R.T., Toerngren M., and Weber M. Towards Improving Dependability of Automotive Systems by Using the EAST-ADL Architecture Description Language. In Architecting Dependable Systems IV, volume 4615 ofLNCS, Springer, 2006, pp. 39–65.

Cuenot, P.et al. Managing Complexity of Automotive Electronics Using the EAST-ADL. Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), IEEE Proceedings, 2007,pp. 353-358.

Dashofy, E. M. et al. A comprehensive approach for the development of modular software architecture description languages ACM. Transactions on Software Engineering and Methodology (TOSEM). Volume 14 , Issue 2, (April 2005).

Cuenot, P., Chen, D., J., Gérard, S., Lönn, H., Reiser, M.-O., Servat, D., Sjöstedt, C.-J., R. Tavakoli, R., Törngren, M., and Weber, M. Managing Complexity of Automotive Electronics Using the EAST-ADL. UML & AADL Workshop, July 14, 2007. aadl.enst.fr/WS_UML_AADL_2007/ATESST_UML-AADL-Workshop_20070707.pdf.

De Bruyne, V., Simonot-Lion, F., Trinquet Y. EAST-ADL – An Architecture Description Language. IFIP World Computer Congress, Toulouse – France, 27 August 2004, Workshop on Architecture Description Languages WADL' 2004.

Demathieu, S. et al. First Experiments Using the UML Profile for MARTE. Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium. 5-7 May 2008, pp. 50 – 57.

El-khoury, J., Chen, D., Törngren, M.. A Survey of Modelling Approaches for Embedded Computer Control Systems, Technical Report, TRITA-MMK 2003:36, ISSN 1400 –1179, ISRN KTH/MMK/R-03/11-SE, 2003.

Faugére, M. et al. MARTE: Also an UML Profile for Modelling AADL Applications Engineering Complex Computer Systems, 2007. 12th IEEE International Conference. 11-14 July 2007, pp. 359 – 364.

Emer, J. et al. Asim: A performance model framework. IEEE Computer, February 2002, pp. 68–76.

Espinoza, H., Medina, J., Dubois, H., Gerard, S., Terrier, F. Towards a UML-Based Modelling Standard for Schedulability Analysis of Real-Time Systems. 2006. www.ctr.unican.es/publications/he-jlm-hd-ft-sg-2006a.pdf.

Feiler P. H., Gluch D. P., and Hudak J. J., The Architecture Analysis and Design Language (AADL): An Introduction. Technical report, CMU/SEI-2006-TN-011, 2007.

Feiler P. H., Gluch D. P., Hudak J. J., and Lewis B. A., "Embedded Systems Architecture Analysis Using SAE AADL." Technical report, CMU/SEI-2007-TN-043, 2007.

Fowler, M., UML Distilled. Pearson Professional Education. 2003.

Frana, R. B. et al. The AADL behaviour annex - experiments and roadmap. Engineering Complex Computer Systems, 2007. 12th IEEE International Conference. 11-14 July 2007, pp. 377 – 382.

Freund, U. Mulit-level system integration based on AUTOSAR. ICSE '08: Proceedings of the 30th international conference on Software engineering. May 2008.

Friedenthal, S., Moore, A., Steiner, F. OMG Systems Modelling Language (OMG SysML™) Tutorial. INCOSE Intl. Symp, 2006. www.omgsysml.org/SysML-Tutorial-Baseline-to-INCOSE-060524-low_res.pdf.

Gérard, S., Espinoza, H., Tahaand, S., Thomas, F. MARTE: the future OMG standard for MDE of RTES. 1st workshop on UML and AADL, ENST, Paris, October 9, 2006.

Gérard, S. et al. UML&AADL '2007 Grand Challenges. ACM SIGBED Review, Volume 4 Issue 4, October 2007.

Grunske L., Joyce D., Quantitative Risk-based Security Prediction for Component-Based Systems with Explicitly Modelled Attack Profiles. Journal of Systems and Software (JSS), Elsevier, Volume 81, Issue 8, August 2008, pp. 1327-1345.

Grunske L., Early Quality Prediction of Component-Based Systems - A Generic Framework, Journal of Systems and Software, Elsevier, Volume 80, Issue 5, May 2007, pp. 678-686

Hardung, B., Kölzow, T., Krüger, A. Reuse of software in distributed embedded automotive systems. EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software . September 2004.

Holz, E., SDL-2000 Tutorial, Formal Methods Europa, FME 2001, Berlin, Germany, March 2001.

Joshi A., Vestal S., and Binns P., Automatic Generation of Static Fault Trees from AADL Models. In DSN Workshop on Architecting Dependable Systems, Lecture Notes in Computer Science, to appear. Springer, 2007.

Krause, M. et al. Timing simulation of interconnected AUTOSAR software-components. DATE '07: Proceedings of the conference on Design, automation and test in Europe. April 2007

Larman, C. Applying UML and Patterns. Prentice Hall. 2004.

Lewis, B., Vestal, S., Feiler, P. Avionics Architecture Description Language (AADL) Seminar. Toulouse (France), October 1st and 2nd, 2002. www.axlog.fr/aadl/aadlseminar.pdf.

Lönn, H. Advancing Traffic Efficiency and Safety through Software Technology (ATESST). The Modelling Approach in ATESST. 2007. www.atesst.org/home/liblocal/docs/ATESST_ModelingOverview.pdf

Leupers, R. and Marwedel, P. Retargetable generation of code selectors from HDL processor models. In Proceedings of Conference on Design Automation and Test in Europe, 1997, pp. 140–144.

Medvidovic, N. and Taylor, R.N. "A classification and comparison framework for software architecture description languages", Software Engineering, IEEE Transactions on. Volume 26, Issue 1, Jan. 2000, pp. 70 – 93.

Mraidha, C. et al. An Execution Framework for MARTE-Based Models Engineering of Complex Computer Systems. ICECCS 2008. 13th IEEE International Conference. March 31 2008-April 3 2008, pp. 222 – 227.

Murata, T. Petri Nets: Properties, analysis and applications. Proceedings of the IEEE, 77(4):541–580, 4, 1989.
OMG. Uml profile for marte, beta 1, ptc/07-08-04. http://www.omg.org/cgi-bin/doc?ptc/2007-08-04. 2007.
Önder, S. and Gupta, R. Automatic generation of microarchitecture simulators. In Proceedings of the IEEE International Conference on Computer Languages, pp. 80–89, May 1998, pp. 80–89.
Pees, S. et al. LISA – machine description language for cycle-accurate models of programmable DSP architectures. In Proceedings of Design Automation Conference, 1999, pp. 933–938.
Papadopoulos Y., McDermid J. A., Sasse R., and Heiner G.. "Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure". Int. Journal of Reliability Engineering and System Safety, 71(3):229–247, 2001.
Pimentel, J, R. An Incremental Approach to Task and Message Scheduling for AUTOSAR Based Distributed Automotive Applications. SEAS '07: Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems. May 2007.
Qin, W. and Malik, S. 2005. A Study of Architecture Description Languages from a Model-based Perspective. Proceedings of the Sixth International Workshop on Microprocessor Test and Verification (MTV'05).
Singhoff, F. et al. Scheduling and memory requirements analysis with AADL. Annual International Conference on Ada Proceedings of the 2005 annual ACM SIGAda international conference on Ada: The Engineering of Correct and Reliable
Software for Real-Time & Distributed Systems using Ada and Related Technologies, pp. 1 – 10.
Sjöstedt, C.-J. et al. Developing Dependable Automotive Embedded Systems using the EAST-ADL - Representing continuous time systems in SysML. Presentation of EAST-ADL; An. architecture description language for. automotive embedded systems. 2007. www.ida.liu.se/~pelab/conf/eoolt07/pres/ presentation-paper3.pdf.
Sokolsky, O. et al. Schedulability analysis of AADL models. Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. 25-29 April 2006.
Tångring, J. "Autosar – a Windows for Car Electronics", The Electronics Magazine, No. 29, 2008.
Törngren. M. (ed). Exploitation strategies. Advancing Traffic Efficiency and Safety through Software Technology (ATESST). 2007. www.atesst.org/home/liblocal/docs/ATESST_Deliverable_D2.2.2_V1.1.pdf.
Weilkiens, T. Systems Engineering With Sysml/Uml. Elsevier Science & Technology, 2008.
Vestal, S. "A Cursory Overview and Comparison of Four Architecture Description Languages", technical report, Honeywell Technology Center, Feb. 1993.
Wild, D., Fleischmann, A., Hartmann, J., Pfaller, C., Rappl, M., and Rittmann, S. An Architecture-Centric Approach towards the Construction of Dependable Automotive Software. SAE International. April 2006. wwwbroy.in.tum.de/publ/papers/2006-01-1222.pdf.
Zimmerman. G. The MIMOLA design system: A computer-aided processor design method. In Proceedings of Design Automation Conference, June 1979, pp. 53–58.