Towards Harmonizing Multiple Architecture Description Languages for Real-Time Embedded Systems

Tahir Naseer Qureshi, Martin Törngren, Magnus Persson, De-Jiu Chen, Carl-Johan Sjöstedt
Division of Mechatronics, Department of Machine Design
KTH-The Royal Institute of Technology
Stockholm, Sweden
{tnqu, martin, magnper, chen, carlj}@md.kth.se

***Abstract*— The increasing complexity of real-time embedded systems requires appropriate methods and techniques to support the development including the specification and analysis of different architectural aspects. A large number of architectural description languages (ADL) have been proposed with varying focus and application domains. There is a need for harmonization of these ADLs. This can be from develoloping and understanding of how they differ or could be synergistically combined for increasing the overall development efficiency and fulfilling the ever increasing functional and non-functional requirements on a system. This paper addresses this issue and focuses on four different ADLs: EAST-ADL, AUTOSAR, AADL and Rubus. In this work we compare these ADLs, identify possible usage scenarios involving more than one ADL and discuss some of the underlying challenges. A representative industrial case study of a brake-by-wire system is used to support the work.**

Keywords-Architecture description language (ADL); AADL, EAST-ADL, AUTOSAR; Rubus; Complexity management; embedded systems; model transformation.

I. Introduction

The complexity of embedded systems and their development process has increased considerably during the last few decades. A few of the contributing factors are increasing number of software components distributed over a network, their interaction, changing technology, increased non-functional requirements such as safety, comfort, reliability, changing and emerging roles of different stakeholders.

Different solutions and methods have been proposed to handle different aspects of the existing complexity. Model-based development (MBD) [1] in particular has gained the attention from industry as well as researchers in the field of embedded systems. Depending on context, there exist different interpretations of model-based development. For example, from control systems engineering point of view, the use of Matlab/Simulink [2] models for design and analysis followed by generation of an executable code for implementation is considered as model-based development. In another context, MBD can be the use of UML based graphical model for design and use of some other language for implementation.

An Architectural Description Language (ADL) is one of the solutions in model-based development. It exists in different forms depending on the applied industrial or application domain. According to IEEE 1471 standard [3], an architecture is defined as:

"The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment and the principles guiding its design and evolution"

While an architectural description is the specification of an architecture, an ADL provides the syntax, either textual or graphical, for an architecture description. The need for expressing and describing architectures of real-time embedded systems has been expressed for several decades. This need has been especially motivated for system integrators, which have to build a system out of a large number of functions and (software and hardware) components. An example of an early ADL is MetaH, which later evolved into the AADL (now standardized by the SAE), [9].

Modeling languages like UML [4] or SysML [5] provides some generic solutions for architectural descriptions. However, specific solutions e.g. MARTE [6] targeting specific aspects of a real-time system is emerging in the form of new UML profiles. In addition there also exist proprietary ADLs e.g. Rubus [7] [1], and domain specific ADLs, for example as part of AUTOSAR [11] and the AADL [9].

Associated with the ADLs are tools for performing different development activities including the specification of an architecture using these ADLs, analysis of functional and non-functional properties, synthesis etc. On one hand, some of the activities are possible using the same tools used for specifying architecture. For example, with OSATE [8], it is possible to specify an architecture using AADL, as well as perform some non-functional analysis. In contrast, additional analysis is carried out by other external tools. For example, Simulink for behavioral analysis of a system specified using EAST-ADL [12]. In this case the required information is transferred either manually or automatically from the architecture description to the analysis tool and vice versa.

[1] With Rubus, we refer to the architecture description language that is used for high-level design (not the corresponding real-time operating system).

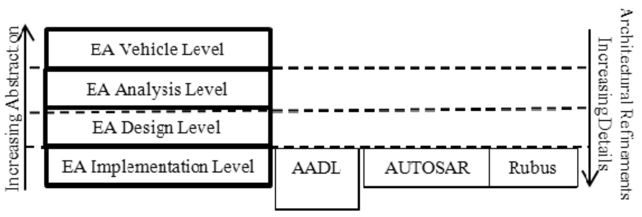

Figure 1. Hypothetical relation between EAST-ADL (EA), AADL, AUTOSAR and Rubus.

With the above-mentioned facts, the need for harmonization (e.g. language combination, tools sharing, model transformation for analysis etc.) is evident. For example, automotive systems are today increasingly being developed using ADLs, and those include at least AUTOSAR-ADL[2] [11], Rubus [7] and EAST-ADL [10]. For the same system, these ADLs can be used with different focuses aspects and levels of abstraction; it is therefore important to understand their relations and how they potentially could be combined. Moreover, since the ADLs often are developed in close cooperation with research, they also come along for example with advanced analysis capabilities. If ADL models could be transformed into other representations with the relevant properties maintained, it might be possible to reuse new analysis features.

The presented work contributes towards the envisioned harmonization. The relation between the four ADLs; EAST-ADL, AADL, AUTOSAR-ADL and Rubus shown in Figure 1 is investigated. The different EAST-ADL (EA) levels correspond to different abstractions. In particular a comparison framework is used for a conceptual mapping of the four ADLs and with the help of a case study, additional tool and analysis relations are identified. The case study is a representative industrial brake-by-wire system. This case study has also been used in the ATESST [12], MAENAD [14] and TIMMO [13] projects. Due to space limitation, the details are not included in the paper. However, we would like to mention that this case study covers all the levels of abstractions shown in Figure 1, comprises of a large variety of constraints as well as a model for the environment. While different parts of the case study were specified, modeled and simulated using a few of the tools related to the considered ADLs, some of the results were obtained from other relevant information such as language specifications. SystemDesk and TargetLink from dSpace [15], Matlab/Simulink, OSATE and Rubus Designer are the major tools used in the work.

The paper is organized as follows: A brief overview of the comparison framework is presented in the next section followed by the presentation of selection rationale in section III. A description of the four ADLs is presented in sections IV - VII. The comparison results are presented in section VIII. A brief overview of the related work is presented in section IX. Finally, the paper is concluded in section X.

[2] Note that AUTOSAR is an automotive software architecture initiative and that we here refer to the architecture description means that are part of this initiative (and for the sake of brevity, will use the term AUTOSAR- ADL).

II. Comparison Framework

The comparison presented in this paper is a modification of the framework defined in [16] to align with our objectives. This modified version uses the same naming but different usage for many of the attributes. According to the framework, an ADL can be evaluated from two different aspects; the modeling features and the tool support. The former aspect is further categorized into components, connectors and architectural configuration.

A few important aspects compared in the framework are as follows:

- Industrial application for which an ADL was originally developed and applied.
- Characteristics of the constituting components and connectors such as independence from the platform on which the components will be deployed and the type of inter-component communications.
- Support for behavioral modeling and analysis, i.e. to know the extent on the reliance on external tools as well as built-in analysis within a specification tool.
- Similarly to the last point, the support for the specification and analysis of non-functional properties is also compared.
- From the overall architectural view, support for scalability (i.e. if it is possible to scale the architecture for larger or smaller platform or resources) and traceability between different artifacts and requirements are considered.
- The tools associated with ADLs are compared for their specification and analysis support and maturity.

Further clarification is provided in the table presented in the results section in the form of questions associated for each evaluated feature.

III. Motivation and Selection Rationale

ADLs provide multiple views/aspects of the system to enable advanced analysis. They have been lagging in industrial adoption compared to behavioral modeling approaches. The reason are the challenges in deploying ADLs, including maturing of tools and anchoring with respect to embedded systems platforms (i.e. corresponding abstractions should be possible to map to platform concepts in a sound way – and tools that support this are required).

For real-time systems, several ADLs can be considered. A brief motivation for the selection and exclusion of different ADLs for the presented work are as follows:

- EAST-ADL and AADL cover a large number of artifacts related to embedded systems, but remain at different levels of abstraction.
- AADL has a long development history, stemming from MetaH, and has further evolved including harmonization efforts with respect to UML/MARTE [6]. Both EAST-ADL and AADL are very interesting since the have been closely aligned to a number of analysis methods and theories, such as different types of safety and timing analysis. EAST-ADL has incorporated concepts from SysML and has also partly been aligned with MARTE.

- AUTOSAR-ADL and Rubus lie on the implementation level. The latter can be seen as a light-weight alternative to the former, with additional focus on specification and handling of real-time requirements [7].

- HDLs – Hardware description languages, such as VHDL, Verilog and SystemC, are evolving towards higher-level abstractions (i.e. transaction level), including both behavioral and structural abstractions. The main emphasis so far has been on simulation-based analysis. Research efforts mainly focus on the functional/behavioral evaluation and not primarily on architecture.

- MARTE [6] is a UML profile dedicated to embedded real-time systems. The MARTE profile is highly relevant but indeed already (at least partially) aligned to both AADL and EAST-ADL, and therefore not covered here.

- SysML – the OMG Systems Modeling Language [5] is a UML profile specification of embedded systems and its environment. It supports the specification of requirements, system dynamics in terms of differential equations etc. SysML is a generic approach, not dedicated to embedded systems and also similar to EAST-ADL.

IV. EAST-ADL

EAST-ADL [10] evolved from the EAST-EEA and ATESST projects and been extended and refined in the ATESST2 [12] project. The EAST-ADL language has also been either adopted or being considered in several other research projects including TIMMO [13], EDONA [17] and CESAR [18]. It is partly included as an annex in the recent MARTE standard [6]. The purpose of EAST-ADL is to specify automotive embedded systems at different levels of abstractions from different views, providing a domain specific approach for separation of concerns and information management between different engineering disciplines. The four abstraction levels in EAST-ADL are *Vehicle level* related to features and product variations, *Analysis level* specifying higher level functionalities, *Design level* related to the detailed design, component allocation. This is further divided into Functional Design Architecture and Hardware Design Architecture specifying application and hardware architectures respectively, and *Implementation level* which is the lowest level of the architecture related to the final development phases and standards such as AUTOSAR [11].

The language has separate packages enabling modular system modeling. Extensions for modeling environment, behavior, dependability, and timing etc. are also provided. These extensions are allowed at different levels of abstraction and enables the separation of the description of design function behaviors and the specification of related constraints for such behaviors like timing and safety.

EAST-ADL can be used with several tools. However, the most commonly used tool for modeling using EAST-ADL is PapyrusUML [19]. External tools are used for different kinds of analysis such as simulation and safety analysis. Examples of external tool usage scenarios include model checking using SPIN [21] and error analysis using HIP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) [20]. The latter is illustrated in [21].

V. AADL

AADL (Architecture Analysis and Design Language) [9] was originally developed for avionics systems development, and based on its predecessor MetaH. The focus of AADL lies on the detailed architecture specification and verification. The language can either be adopted as part of a top-down development approach, supporting detailed architecture design and integration analysis, or for reverse engineering efforts in modeling and analyzing (changes to) legacy systems. The language specifications [9] do not explicitly specify more than a single level of abstraction (the entire system) or an implied development approach. Similarly to EAST-ADL, it also has core language constructs and the extensions namely *error modeling* for dependability aspects of reliability, availability etc., *meta-model for XMI/XML annex* for supporting additional tools, *graphical notation* for graphical modeling, *behavior modeling*, *ARINC 653* to support the standard for aircraft industry, *UML2.0 profile* to support the use of UML

The most common tool used for modeling with AADL is OSATE (An Extensible Open Source AADL Tool Environment) [8]. It is possible to perform resource and end-to-end latency flow analyses, semantic analysis of modal system as well as security level checking directly within the tool environment. This is exemplified by the AADL consortium with different case studies. In addition, tools such as Simulink can be used for analyzing AADL models. One such example is [23]. AADL has also been incorporated in the latest MARTE standard [6]. The current evolution includes update of the behavior, data modeling and ARINC annexures. AADL has its own drawbacks, such as complex component compositions and property ambiguity [24].

VI. AUTOSAR-ADL

AUTOSAR (AUTomotive Open System ARchitecture) [11] is a standard for dealing with the ever-growing complexity of automotive embedded systems. It provides a basis for modular software architecture with standardized interfaces and specifications of a run-time environment. Configuration management, e.g. relocation of functionality from one computation node to another at development time is also supported by the AUTOSAR standard. The standard also allows the use of commercial-off-the-shelf SW-components.

The result of the AUTOSAR effort is a framework and methodology [25] for standardized automotive software and hardware. A six-layered software architecture is a part of the AUTOSAR framework. The layers are Application, Run Time Environment (RTE) providing middleware functionality, Service, ECU (Electronic Control Unit) Abstraction, Complex Drivers, and Micro-controller Abstraction [26]. While the ECU abstraction is the lowest layer, the Application is the highest one. The AUTOSAR *atomic software components* are categorized into application and infrastructure. While the former is related to providing software functionalities such as control algorithm etc. the latter is used for different services related to an ECU. In addition a concept of virtual functional bus (VFB) is also introduced to resolve control-related integration problems. Furthermore, AUTOSAR supports two possible communication types between its components: client-server and sender-receiver.

For the description of architecture and its components standard templates are provided such as [37] for software components. These *templates* correspond to the *ADL part of AUTOSAR* providing a means to specify and relate different architectural entities.

Several types of tools are required and being adapted to configure AUTOSAR-based embedded systems. A few examples of the tools include, TargetLink and SystemDesk by dSpace[15] for application modeling, code generation, and system design. Similar tools are also provided by Vector [27]. Configuration of ECUs is carried out using tools such as PICEA SUITE from Mecel AB [28].

AUTOSAR and its ADL are evolving with time. The latest release, i.e. version 4.0, provides additional support for dynamic scheduling and introduction of multi-core architectural concepts. The current evolution includes extensions to address non-functional aspects such as safety (partial alignment to the forthcoming ISO/DIS 26262 safety standard) and timing (findings from the TIMMO project – TADL).

VII. Rubus

Rubus [7] is an ADL for the specification of safety-critical embedded system architectures. It originated from BASEMENT [29] and has been used in many industries related to both the automotive and avionics domain. Rubus has been used as a real-time operating system for many years but now it has an ADL layer on top of it. The main artifact of Rubus ADL is its component model (CM). While the latest version i.e RubusCMv3 is presented in [36], RubusCMv4 is under development. A few of the objectives of Rubus are the support for an overall system view and reasoning about different aspects at a higher level of abstraction, separation of views such as functional and temporal enabling the specification of both real-time requirements and properties, early analysis due to a formal syntax providing estimates of implementation aspects such as memory consumption.

Similar to an atomic software component in AUTOSAR-ADL, the basic artifact is a software circuit (SWC)[3]. A SWC can have one or more associated behaviors, interfaces and internal state data. The execution method followed by a SWC is run-to-completion. It also uses the concept of composite (as in AUTOSAR-ADL), which is a collection of SWCs where the SWCs can be distributed into different computing nodes. In contrast, a collection of SWCs called an assembly can only be allocated to one single node. The concept of an overall mode of a system can also be specified. Additional artifacts of RubusCMv3 include items such as clock, interrupt or events, down-sampling and precedence. In terms of timing requirements deadline, offset and periodic jitter can be specified. In contrast to AUTOSAR-ADL, Rubus provides a relatively simple methodology, starting with functionality design followed by the separation of data and control flows, choice of an execution model i.e. time or event-triggering.

[3] Although the short form of an AUTOSAR software component and a Rubus Software circuit is same, the concept is not the same.

From analysis and tooling view, Rubus is supported by dedicated tools namely Rubus Designer, Rubus Analyzer, and Rubus Inspector for simulation and analysis provided by Arcticus System AB [7]. A few of the examples of analysis are end-to-end response time, overall stack size, effect of different priority assignments etc. Furthermore, the tools are also integrated with other higher-level analysis tool such as Simulink and LabView. Due to this integration, it is possible to perform co-simulation as well as utilization of automated code generation features of the tools for a final platform-specific implementation.

VIII. Comparison Results

A comparison based on the framework discussed in section II is shown in Table 1. A central commonality is that all ADLs are based on the concepts of components with ports that interact over connections. Components, ports and connectors are provided with properties that together allow a structural system description based on which certain analysis is possible. For all ADLs, a basic emphasis is on the system structure with essentially black boxes (components), where the contained behavior would be expected to be expressed using different formalisms (e.g. C code, a Simulink model, etc.). The ADLs mainly capture so called execution behavior, which is related to components execution timing and the method for sharing resources. However, different types of components are supported and with somewhat different assumptions on the underlying models of computation and properties of interest.

It is only EAST-ADL which explicitly supports the specifications of requirements, product features and variability. Although AUTOSAR-ADL only covers a single level of abstraction, it is the most complex and detailed. This is due to its large variety of components including platform specific ones. Tool support in general is not so mature for most of the considered ADLs. From the architecture specifications perspective, only OSATE is found to be mature. The specification tools for other ADLs need further refinements especially for stability. Because of Simulink's dominating position, most of the ADLs try to take a position with respect to it, for example by developing mappings between ADLs and Simulink. Therefore, it can serve to be a part of envisioned harmonization. For example, using is as the common tool for simulating behavior and code.

In addition to the above, it is also observed that the combined coverage of the four ADLs considered is enormous including multiple engineering disciplines and concerns related to the development of embedded systems. Therefore, it is non-trivial to cover all the harmonization possibilities in this paper. Based on the comparison table and the experience from the case study, a subset of different scenarios for language and tool usage illustrated in Figure 2 are identified. One major assumption for this scenario is that the development follows a top-down EAST-ADL development methodology [30]. The methodological choice is due to the higher abstraction levels supported by EAST-ADL lacking in other ADLs. The

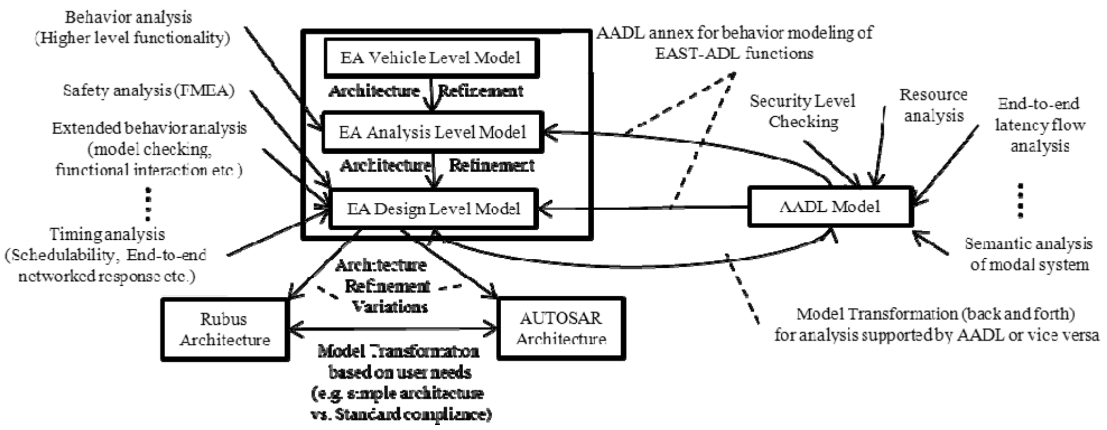

Figure 2. Language and tool usage scenarios.

Table 1: Comparison of Architecture Description Languages for Automotive Embedded Systems

Category	Features	Related Questions	AADL	EAST-ADL	AUTOSAR-ADL	Rubus
Scope and applicability		Is it generic or specific for an application domain?	Generic. Initially developed for avionics system but now applicable to automotive as well.	Developed specifically for automotive systems but applicable in other domains	Automotive specific software architecture	Generic and applicable to different application domains.
Support for modeling components	**Characteristics**	Are components platform independent?	The components are platform independent but it is also possible to specify platform related components.	The components are platform independent but it is also possible to specify platform related components.	Depending on the usage the components can be both platform independent and platform specific.	The components are platform independent but require platform specific attributes for analysis.
	Interface	What is the point of interface for a component?	Ports and port groups.	Ports and port groups.	Ports and port groups.	Port
	Types	What are the different kinds of instanstiable components?	Software (data. Subprograms, subprogram calls, threads, thread groups, processes, pre-declared runtime services) and Execution (Processors, Memory, buses, devices).	Analysis function type and prototype, design function type (Basic software function type, local device manager, hardware function type). The hardware function has sub-types of node, power supply and logical data bus)	Application Software, Service, ECU abstraction, Complex device driver, Sensor Actuator.	Software circuit
	Behavior	How is the behavior specified? Is it possible to use external tools / languages to define behavior?	Basic behavior such as modes is part of the core language. For additional behavior description, AADL behavior annex can be used. External tools can be used for generating code an AADL component.	Basic behavior such as modes is part of the core language. For additional behavior description the EAST-ADL behavior extension can be used. It is also possible to associate external models e.g. Simulink file to EAST-ADL components.	The fundamental behavior entity of AUTOSAR is a runnable (smallest code fragment provided by a component). The code is obtained by external tools. In addition it is also possible to define modes of a system which in turn can be used for different configurations.	The behavior of a SWC is specified by its source code. The code is obtained by external tools. Similar to AUTOSAR it is also possible to define modes of a system which in turn can be used for different configurations.
	Constraints	How are different kinds of constraints applied to the components and behavior? Is it possible via ports? Do additional constructs exist?	Constraints are defined as the properties of different components. The examples period for execution, priority or security level etc. Additional constructs can be provided by developing additional libraries.	Constraints are specified either by requirements or specific constructs provided by the extensions (e.g. timing or safety). Additional constructs can be provided by developing extensions.	It is possible to specify execution constraints as a part of the behavior definition. For example a constraint can correspond to an RTE event for triggering a runnable or an inter-runnable variable for data sharing.	Currently Rubus supports timing constraints i.e. deadline, offset and periodic jitter.
	Evolution	Is it possible to use sub-typing mechanisms for components and its	Sub-typing and instances are possible	EAST-ADL explicitly supports component prototyping mechanism.	Sub-typing and instances are possible	The library of the components allows sub-typing and instantiation

Category	Features	Related Questions	AADL	EAST-ADL	AUTOSAR-ADL	Rubus
		features for systematic evolution of the architecture?				to some extent.
	Non-functional properties	Which additional non-functional properties can be specified?	Security levels, priority, timing and computational resources e.g. memory and bandwidth.	Functional safety and timing.	Timing.	Timing
Connectors	Characteristics	Does the language provide any difference between logical, functional and physical connectors?	AADL differentiates between functional, logical and parametric connections	EAST-ADL have different connectors for logical, functional, physical connections.	The connectors in an AUTOSAR architecture can be considered as virtual as dependencies between different ports is handled by the RTE.	Rubus differentiates between data and triggering.
	Interface	Which kind of interfaces is supported?	Event and data.	Sender-receiver, client server and communication channel.	Sender-receiver, client server and mode-switch.	Data and trigger.
	Semantics	Do the connectors follow any specific semantics?	Yes. The AADL specifications specify semantics for its connections e.g. immediate or delayed.	The EAST-ADL specifications specify semantics for its connections e.g. one size overwrite buffer.	The semantics are inherited form the associated ports.	The semantics are inherited form the associated ports.
	Types	What are the different kinds of connectors?	Port connections, parameter connections, access connection	Functional, communication, hardware and clamp connectors.	There exists no categorization of connectors.	There exists no categorization of connectors.
	Non-functional properties	Is it possible to specify non-functional properties such as end-to-end time etc.?	Yes by defining the end-to-end flows.	It is possible to use the extensions to specify non-functional properties.	Not explicitly.	Not explicitly.
Architectural Configurations	Understandable specifications	Are the specifications easily understandable?	Yes. Specifically with the graphical support.	Yes. Specifically with the graphical support.	Depending on the tool used, the specifications are understandable for overall system configuration.	With the graphical support of the available tool, the architectural specifications are easily understandable.
	Levels of abstractions	How many and which abstraction levels are supported by the language?	The design level and to some extent the implementation level.	All levels of abstraction but with weak support for the implementation level.	Implementation level and partially design level.	Implementation level and partially design level.
	Compositionality	Is it possible to specify hierarchical composition?	It is possible. This is also described in [24]	Component typing and prototyping is used for this purpose.	The concept of composition is provided for this purpose.	A composite SWC can be defined.
	Refinements and Traceability	Does the language provide any explicit traceability support between requirements and different artifacts? Does the language provide any constructs relating architectures at different levels of abstractions?	No support for requirements traceability and architectural refinements.	This is one of the main features of EAST-ADL.	No explicit support for requirements traceability and architectural refinements.	No explicit support for requirements traceability and architectural refinements.
	Scalability	Are the architectural configurations scalable?	Using the biding mechanism, scalability is obtained.	Scalability is dependent on the selected implementation platform e.g. AUTOSAR	Scalability is one of the main features of AUTOSAR.	Similar to AUTOSAR , Rubus components can also be scaled for different hardware platforms
	Variability	Does the language support handling of variations and versions of architecture configurations?	To limited extent. It is possible to specify different configurations and later use the binding mechanism.	EAST-ADL supports product line variations. Although not explicit, but the same support can be used for function variations.	No explicit support for variations.	No explicit support for variations.
	Non-functional properties	Is it possible to explicitly specify non-functional properties for overall architectural configurations?	Depending on how a system is perceived. For example a component can correspond to a system	It is possible by utilizing the same extensions used for specifying non-functional properties of components and	Non-functional property specifications are not explicit. However, some timing aspects are part of a component behavior.	Rubus does not support specification of non-functional properties.

Category	Features	Related Questions	AADL	EAST-ADL	AUTOSAR-ADL	Rubus
			of its own.	connectors.		
Tool support	Tool Examples	What are the examples of the tools for system specification?	A few examples include, OSATE, STOOD and ASSERT.	Prototype tools like PapyrusUML and one by MentorGraphics.	DaVinci developer, SystemDesk.	Rubus Designer is the only tool available.
	Multiple views	Do the tools support multiple views of an architecture i.e. text, graphical etc.?	It is possible to specify an architecture using graphical and textual format.	In addition to graphical and textual description support, EAST-ADL also supports different views such as safety, hardware and software.	It is possible to use both graphical and textual description. AUTOSAR also provides separation of hardware and software architecture.	Only graphical representation is supported. No special view is supported.
	Analysis support	Is it possible to perform analysis using the specification tools or additional tools are required?	With OSATE, it is possible to perform analysis such as security, flow etc. Additional tools can also be used.	EAST-ADL relies entirely on external tools i.e. no support provided by the architecture specification tool.	Some simulation is possible using the tools. However, due to the inherent AUTOSAR complexity, analysis is also complex requiring external tools.	It is possible to perform some timing analysis. For behavior, Simulink models can be linked to Rubus SWC.
	Maturity	How mature is the tool support?	The tool support for AADL is quite mature for analysis, specification as well as implementation.	EAST-ADL relies on external tools for analysis. However, the transformation of information between specification and analysis tools is at prototype level and therefore, not so mature.	The tool support is quite mature for executing AUTOSAR methodology.	The tool support is limited but mature enough to support the objectives of Rubus based architectural specification, analysis and implementation.

identified scenarios are as follows:

- AADL support for EAST-ADL methodology in two ways. Firstly, the AADL behavior annex can be used to specify behavior of EAST-ADL components. On the other hand, EAST-ADL models at design level can be transformed into AADL for the analysis already supported by AADL tools. Although, the two ADLs do not share the same abstraction levels, the commonality mentioned at the start of this section can be exploited for the transformation and analysis.
- For additional functional and non-functional analysis, tools such as Matlab/Simulink or HIP-HOPS can be used with the help of model transformations between the tools and ADLs similar to [21].
- AUTOSAR-ADL, Rubus and AADL an be considered as three alternate implementation platforms It may be required to transform Rubus models to AUTOSAR-ADL if compliance to standards is required. On the other hand AUTOSAR models can be transformed to Rubus models if a simple or highly safety critical implementation is required. The transformation will only be possible with a restricted subset of AUTOSAR-ADL. The transformation to Rubus can also lead to the use of analysis support by dedicated Rubus tools for safety critical systems.

A few of the challenges to achieve the above mentioned usage scenarios are the lack of concrete mapping between different artifacts of ADLs, closed source code of the tools and their meta-models, which restrict the model transformation for analysis, lack of standard ways to represent results of different analysis obtained from different tools.

IX. Related Work

Architectural description languages have been surveyed and compared by a number of researchers each with different perspective and focus. In [16] a framework for classification and comparison of different ADLs is presented. The framework is divided into four main categories of components, connectors, architecture configurations and the available tool support. Each category has its own attributes such as “Refinement and Traceability” for the category of architecture configuration. A taxonomic survey of ADLs is also presented in [31] where the classification is made on the classes of the supported systems, inherent properties of the language and the technology support provided. These two surveys provide a good overview of different characteristic and comparison of then existing ADLs.

In another effort [32], the authors surveyed different ADLs developed for the design and implementation of micro-architectures of processors from a model-based development point of view.

A survey on different methods for modeling embedded computer systems is presented in [33]. The main focus is on the languages for embedded system design. The authors also followed a model-based development approach for designing embedded systems. The framework used in this work had five categories of Contents, Design Context, Analysis Context, Language, and Tools. Just like [16], the categories have their own attributes covering aspects such as abstractions, details, and the relations between these attributes. The survey resulted in a comparison of 12 different ADLs.

With the passage of time, the different ADLs surveyed in the above-mentioned work have either further evolved or became obsolete. Moreover, the introduction of graphical languages such as SysML [5], UML [4], tools such as

Matlab/Simulink [2] and the increasing number of disciplines and views of embedded systems, different domain-specific solutions have been proposed.

X. Discussion and Conclusion

In this paper, we have provided a comparison of four architecture description languages related to real-time embedded systems. In addition, we have also identified usage scenarios and associated challenges possible while using the considered ADLs. This work can be considered as a step towards increasing the overall efficiency of the embedded systems development process by utilizing the existing formalisms and tools. A more fine grained mapping between different ADLs, including aspects such extent of modularity and scalability, support for different models of computations, variability support, safety and security analysis can be considered as the possible future extensions of the presented work. These results are also the basis for our work in the MAENAD [14] project targeting future solutions of full electric vehicles. We are currently refining the EAST-ADL behavior annex, developing tool specific extensions of the behavior annex for a native representation of different tools, mapping scheme between different ADLsand improvement of current plug-ins for Simulink and HiP-HOPS [21]. The additional analysis tools which are currently being addressed in MAENAD are Modelica, UPPAAL, SPIN model checker, SCADE [34], and MAST (Modeling and Analysis Suite for Real-Time Applications) [35].

References

[1] M. Törngren, D. Chen, D. Malvius, and J. Axelsson, "Model-Based Development of Automotive Embedded Systems," in Automotive Embedded Systems Handbook, N. Navet and F. Simonot-Lion, Eds. CRC Press, 2009, ch. 10, pp. 1-52.

[2] Mathworks. (2010, May) Matlab Website. [Online]. http://www.mathworks.com/

[3] IEEE, "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems -Description," IEEE Std 1471-2000.

[4] Object Management Group. (2010, May) Unified Modeling Language. [Online]. http://www.uml.org/

[5] SySML. Systems Modeling Language. [Online]. http://www.sysml.org/

[6] Object Management Group, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems: Version 1.0," formal/2009-11-02, 2009.

[7] Articus Systems AB. (2010, Jul.) Articus Systems - The Choice for Dependable Real-Time Systems. [Online]. http://www.arcticus-systems.com/

[8] The SEI AADL Team, "An Extensible Open Source AADL Tool Environment (OSATE)," The Software Engineering Institute, Carnegie Mellon University, 2006.

[9] SAE, "SAE Architecture Analysis and Design Language (AADL)," SAE International Standard AS5506, 2004.

[10] ATESST Consortium, "EAST-ADL2 Domain Model Specifications," Project Deliverable D4.1.1 http://www.atesst.fr/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-05-03.pdf, 2010

[11] AUTOSAR. (2010) AUTomotive Open System ARchitecture. [Online]. http://www.autosar.org.

[12] Advancing Traffic Efficiency and Safety through Software Technology. (2009) [Online]. http://www.atesst.org.

[13] TIMMO Consortium. (2010, May) TIMMO Websiite. [Online]. http://www.timmo.org/

[14] MAENAD Consortium (2011, May), MAENAD Website, [Online], http://www.maenad.edu/

[15] dSPACE. (2009) dSPACE website. [Online]. http://www.dspaceinc.com/ww/en/inc/home.cfm

[16] N. Medvidovic and R. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," IEEE Transactions On Software Engineering, vol. 26, no. 1, pp. 70-93, Jan. 2000.

[17] EDONA Consortium. (2010, Jul.) EDONA:Environnements de Développement Ouverts aux Normes de l'Automobile. [Online]. http://www.edona.fr/home/index.htm.

[18] CESAR Consortium. (2010, Jul.) CESAR Project Website. [Online]. http://www.cesarproject.eu/

[19] CEA LIST. (2010) PapyrusUML. [Online]. http://www.papyrusuml.org/

[20] Y. Papadopoulos, J. A. McDermid, R. Sasse, and G. Heiner, "Analysis and Synthesis of The behaviour Of Complex Programmable Electronic Systems In Conditions of Failure," Reliability Engineering and System Safety, vol. 71, no. 3, pp. 229-247, Mar. 2001

[21] M. Biehl, C. DeJiu, and M. Törngren, "Integrating Safety Analysis into the Model-based Development Toolchain of Automotive Embedded Systems," in In Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems LCTES'10, Stockholm, 2010, pp. 125-131

[22] L. Feng, DJ. Chen, H. Lönn and M. Törngren, "Verfiying System Behaviors in EAST-ADL2 with the SPIN Model Checker", IEEE International Conference on Mechatronics and Automation. Xi'an, China, August 4-7, 2010.

[23] P. Feiler, "AADL and Simulink," Software Engineering Institute, Carnegie Mellon, 2006.

[24] S. Gérard, et al., "UML&AADL '2007 Grand Challenges," ACM SIGBED Review, vol. 4, no. 4, pp. 1-17, Oct. 2007.

[25] The AUTOSAR Consortium, \AUTOSAR Methodology," Tech. Rep. V1.2.1, R3.0, Rev 001, http://www.autosar.org/download/AUTOSAR_Methodology.pdf, 2008

[26] The AUTOSAR Consortium, Layred Software Architecture," Tech. Rep. V2.2.1, R3.0, Rev 001, 2008.

[27] Vector. (2010, Jul.) Vector [AUTOSAR-A Decision for the Future]. [Online]. http://www.vector.com/vi_autosar_solutions_en.html

[28] Mecel AB. (2010, Jul.) Mecel Website. [Online]. http://www.mecel.se/

[29] H. Hansson, H. Lawson, M. Strömberg and S. Larsson, "BASEMENT a Distributed Real-Time Architecture for Vehicle Applications," Real-Time Systems, vol. 11, no. 3, pp. 223-244, Nov. 1996.

[30] The ATESST2 Consortium, \Methodology Guidelines When Using EASTADL2," Project Deliverable 5.1.1, http://www.atesst.org/home/liblocal/docs/ATESST2_Deliverable_D5.1.1_V1.1 .pdf, June 2010.

[31] P. C. Clements, "A Survey of Architecture Description Languages," in Eighth International Workshop on Software Specification and Design, 1996.

[32] W. Qin and S. Malik, "A Study of Architecture Description Languages from a Model-based Perspective," in Proceedings of the Sixth International Workshop on Microprocessor Test and Verification, 2005, pp. 3-11.

[33] J. El-Khoury, D. Chen, and M. Törngren, "A Survey of Modelling Approaches for Embedded Computer Control Systems," Mechatronics Lab, Department of Machine Design. Royal Institute of Technology., Stockholm, Technical Report TRITA - MMK 2003:36, ISSN 1400 -1179, ISRN KTH/MMK/R-03/11-SE, 2003.

[34] Esterel Technologies, SCADE website, [Online], http://www.esterel-technologies.com/products/scade-suite/ , May 2011.

[35] MAST website, [Online] http://mast.unican.es/, May 2011.

[36] K. Hänningen, J. Maäki Turja, M. Nolin, M. Lindberg, J. Lundbäck and KL, Lundbäck, "Supporting Engineering Requirements in the Rubus Component Model", MRTC report ISSN 1404-3041 ISRN MDH-MRTC-223/2008-1-SE, Mälardalen Real-Time Research Centre, Mälardalen University, February, 2008.

[37] The AUTOSAR Consortium, "Software Component Templatee," Tech. Rep. V3.3.0, R3.0, Rev 001.