Automotive System Development Based on Collaborative Modeling Using Multiple ADLs

Shin'ichi SHIRAISHI
Toyota InfoTechnology Center Co., Ltd.
Akasaka 6-6-20, Minato-ku
Tokyo, Japan 107-0052
shiraishi@jp.toyota-itc.com

Mutsumi ABE
Toyota InfoTechnology Center Co., Ltd.
Akasaka 6-6-20, Minato-ku
Tokyo, Japan 107-0052
m-abe@jp.toyota-itc.com

ABSTRACT

This paper presents two different model-based approaches that use multiple architecture description languages (ADLs) for automotive system development. One approach is based on AADL (Architecture Analysis & Design Language), and the other is based on collaborative modeling by SysML (Systems Modeling Language) and MARTE (Modeling and Analysis of Real-Time and Embedded systems). In this paper, the detailed modeling steps for both approaches are explained through a real-world automotive development example: a cruise control system. Moreover, discussion of the modeling steps offers a qualitative comparison of the two approaches, and then clarifies the characteristics of the different types of ADLs.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Design Tools and Techniques—*modules and interfaces*

General Terms

Experimentation

Keywords

model-based development, automotive systems, architecture description languages, AADL, SysML, MARTE

1. INTRODUCTION

The electrical and electronic (E/E) component of automotive systems is continuously growing. For example, the ratio of the production cost of E/E components to that of the other types of components is expected to exceed 40% in 2015. In particular, the size of software of E/E components is expanding exponentially, such that the complexity of software has become a critical concern in effective automotive system development. In addition to development efficiency, quality assurance for the software is another serious concern, as was seen in several recalls occurred in the past.

For these reasons, several model-based approaches have attracted considerable attention. As these approaches formalize system designs, they might provide formal design verification of automotive systems. These formal kinds of evidence have a potential to efficiently constitute quality assurance cases.

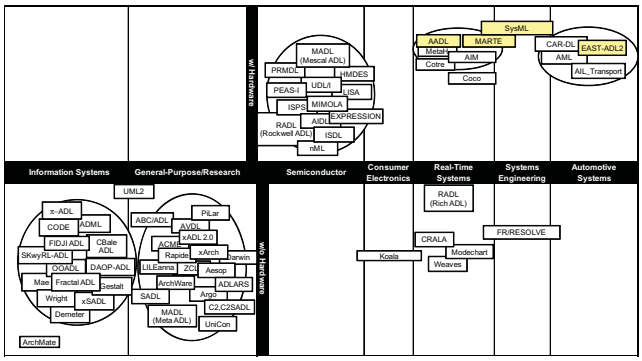

Figure 1: Classification of Architecture Description Languages.

This paper chooses architecture description languages (ADLs) from among several modeling languages, and focuses on their practical application to the automotive domain. Specifically, two types of approaches based on different ADLs are introduced. We will explain how these approaches realize the development phases of automotive system, and thereafter clarify their characteristics using several example models.

2. ARCHITECTURE DESCRIPTION LANGUAGES AND THEIR CLASSIFICATION

Architecture description languages (ADLs) have been widely studied across several research fields and industries. Medvidovic et al. have already conducted an exhaustive survey of ADLs from a purely technical point of view [1]. In contrast to previous research [1], we performed our survey from a practical point of view. The purpose of our survey is to find good candidates for application to the automotive industry. In our survey, we collected more than sixty ADLs from published research papers and project reports, and briefly investigated each ADL. One result of our survey is a classification of the collected ADLs, as shown in Fig. 1.

When considering the following features of automotive systems: (1) large-scale systems, (2) real-time systems, (3) embedded systems; we found four promising candidates, shown as filled boxes in Fig. 1. These candidates are *AADL* (Architecture Analysis and Design Language) [2], *SyML* (Systems Modeling Language) [3], *MARTE* (Modeling and Analysis of Real-Time and Embedded systems) [4], and *EAST-ADL2* [5]. AADL is standardized by SAE International (Society of Automotive Engineers). SysML and MARTE are published by OMG (Object Management Group). EAST-ADL2, which is not yet standardized, is an outcome of the ATESST (Advancing Traffic Efficiency and Safety through Software Technology) project funded by an EU committee.

If we apply ADLs to real-world automotive system development, we need to pay attention to the *development* aspect, in addition to

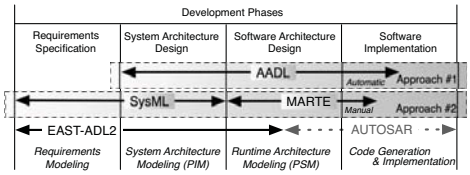

Figure 2: Comparison of Design Phase Coverage of ADLs.

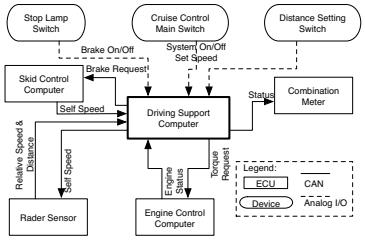

Figure 3: Adaptive Cruise Control System.

the above *system* characteristics, such as (4) handling a great deal of legacy code, and (5) distributed development including several suppliers. On the contrary, EAST-ADL2 does not provide a bridge between models and source code, and instead delegates this task to AUTOSAR (AUTomotive Open System ARchitecture) [6]. This fact can be illustrated by Fig. 2, which shows the ADLs corresponding to each development phase[1]. Moreover, EAST-ADL2 is not yet standardized; therefore, we cannot expect it to be widely adopted among supplier companies. These facts imply that EAST-ADL2 is difficult to use for the purpose of real-world automotive system development. Therefore, in considering the design phase coverage of ADLs shown in Fig. 2, we can focus on the following two approaches: an AADL-based method (hereafter *approach #1*) and another method based on the combination of the SysML and MARTE (*approach #2*).

3. MODEL-BASED DEVELOPMENT USING MULTIPLE ADLS

This section provides a detailed discussion of ADL-based development steps via an automotive system example, specifically, an adaptive cruise control (ACC) system. The ACC system provides the following two major functions: (i) constant-speed cruise (CSC) control, and (ii) constant-distance cruise (CDC) control. Figure 3 shows an outline of the ACC system. The Driving Support Computer (DSC), which is an ECU (Electrical Control Unit), is the central component of the ACC system. Thus, the software running on the DSC is the main topic of this section.

In the following subsections, we will explain the software development of the ACC system according to the two approaches, in three steps: (i) System Modeling, (ii) Runtime Architecture Modeling, and (iii) Code Generation (see Fig. 2).

3.1 Approach #1: AADL-Based System Development

3.1.1 System Architecture Modeling

The system architecture indicates system functions by combining logical components. Therefore, we can use the **system** component of the AADL as a symbol for logical components. Figure 4 and Listing 1 show the same system architecture model using different notations, that is, textual and graphical. In this model, the

[1]The behavior aspect is intentionally ignored due to space limitations.

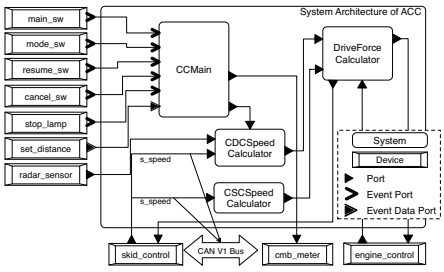

Figure 4: System Architecture Model in AADL.

system architecture *ACCSystem.SysArch* is described as the combination of components such as the logical software component (*ACCSoftware.log*) and the execution platform (*DSCHardware*).

The same model also defines the interfaces among the software component *ACCSoftware.log*, the CAN bus (*hs_can_bus_v1*), and the peripheral devices such as the skid control computer (*SKID_CONT*). The units of data exchanged among these components, e.g., *kph_type* (km/h), are also specified there.

Listing 1: System Architecture Model.

```
system ACCSystem
end ACCSystem;
-- System Architecture Model
system implementation ACCSystem.SysArch
  subcomponents
  -- Logical Software & Hardware Components
  ACCSoftware: system ACCSoftware.log;
  DSCHardware: system DSCHardware;
  -- Peripherals
  SKID_CONT: device devices::skid_control;
  (snip)
  connections -- CAN bus accesses
  GC0000: data port SKID_CONT.s_speed->ACCSoftware.s_speed;
  can_msg_v1_0000: bus access DSCHardware.hs_can_bus_v1
    -> ACCSoftware.hs_can_bus_s_speed;
    (snip)
end ACCSystem.SysArch;
system ACCSoftware -- Top-Level Software Component
  features -- Interfaces with units
  s_speed: in data port types::kph_type;
  (snip)
end ACCSoftware;
-- Logical Software Component
system implementation ACCSoftware.log
  subcomponents -- Logical Software Subcomponents
  CSCSpeedCalculator: system csc_speed_calculator;
  CDCSpeedCalculator: system csc_speed_calculator;
  (snip)
end ACCSoftware.log;
```

3.1.2 Runtime Architecture Modeling

The runtime architecture of the software consists of physical software entities such as processes and threads. Therefore, we can straightforwardly use the **process** and **thread** components for runtime architecture modeling. By combining these physical entities and substantiating the interfaces in the software component *ACCSoftware*, we can obtain the runtime architecture model (*ACCSystem.RunArch*), as shown in Listing 2 and Fig. 5. In the listing, the AADL elements, **extends** and **refined to**, give an inheritance mechanism and replace the logical software component *ACCSoftware.log* by the physical one *ACCSoftware.phy*.

Figure 5 shows that there exists a single process that contains six internal threads. Moreover, the runtime architecture model describes communication between threads (internal communication) and between threads and CAN buses (external communication). If we can determine real-time properties, e.g., the execution period and execution time of threads, OSATE (Open Source AADL Tool Environment) yields several kinds of analysis such as schedulability analysis and end-to-end latency analysis.

Listing 2: Runtime Architecture Model.

```
-- Runtime Architecture Model
system implementation ACCSystem.RunArch
  extends ACCSystem.SysArch
```

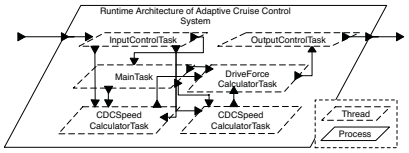

Figure 5: Runtime Architecture Model in AADL (cf. Fig. 4).

```
subcomponents -- Logical Component -> Physical Component
  ACCSoftware: refined to system ACCSoftware.phy;
end ACCSystem.RunArch;
 -- Physical Software Component
system implementation ACCSoftware.phy
  subcomponents
  SoftwareProcess: process ACCSoftwareProcess;
end ACCSoftware.phy;
process ACCSoftwareProcess -- Single Process
  features -- Interfaces
  s_speed_in: in data port types::kph_type;
end ACCSoftwareProcess;
process implementation ACCSoftwareProcess.impl
  subcomponents -- Six Threads
  InputTask  : thread input_controller_thread.impl;
  CSCSpeedCalculatorTask: thread csc_sc_thread.impl;
  (snip)
  connections -- CAN Bus & Inter-Task Communication
  GC0000: data port s_speed_in->InputTask.s_speed_in;
  IC0000: data port InputTask.f_s_speed_out
    ->CSCSpeedCalculatorTask.f_s_speed_in;
    (snip)
end ACCSoftwareProcess.impl;
```

3.1.3 Code Generation

We can derive source code templates from the runtime architecture model obtained in Sect. 3.1.2. For example, if we use the OSEK/VDX-C platform [7], we can produce a configuration file in OIL (OSEK Implementation Language) shown in Listing 3 and the skeleton code shown in Listing 4. The AADL specification [2] itself contains the guidelines for translation from AADL descriptions into source code. Moreover, a source code generator[2] has been developed for some platforms. However, it should be noted that these translations depend heavily on the platform used; that is, different target platforms require different translation strategies.

Listing 3: Generated Configuration in OIL.

```
/* Tasks */
TASK InputControllerTask {
  TYPE = BASIC;
  /* Snip */
  }
TASK CSCSpeedCalculatorTask{
  TYPE = BASIC;
  /* Snip */
  }
/* Messages */
Message GC0000 { /* CAN Message (External Communication) */
  TYPE = EXTERNAL;
  ACCESSNAME = {s_speed, s_speed_in};
  CAN_ADDRESS = {can_msg_v1_0000};
  /* Snip */
  };
Message IC0000 {/* Inter-Task Message (Internal Communication) */
  TYPE = INTERNAL;
  ACCESSNAME = {f_s_speed_out, f_s_speed_in};
  /* Snip */
  };
```

Listing 4: Generated Skelton Code of Tasks.

```
TASK (InputControllerTask) {
  float l_s_speed_in = 0;
  while(ReceiveMessage(GC0000, s_speed_in) == E_OK){
  /*CAN Message (External Communication) */
    l_s_speed_in = s_speed_in;}
    /* Snip */
  SendMessage(IC0000, f_s_speed_out) ;
  /*Inter-Task Message (Internal Communication)*/
  }
TASK (CSCSpeedCalculatorTask) {
  float l_f_s_speed_in = 0;
  while(ReceiveMessage(IC0000, f_s_speed_in) == E_OK){
    l_f_s_speed_in = f_s_speed_in; }
    /* Snip */
}
```

[2]http://penelope.enst.fr/aadl/

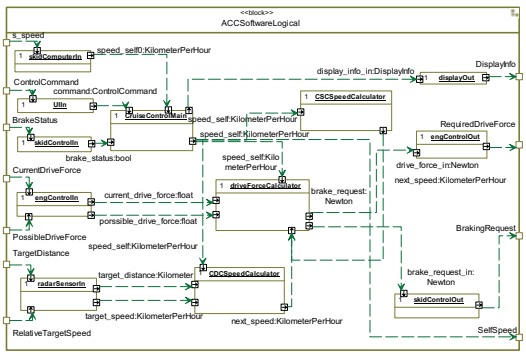

Figure 6: System Architecture Model in SysML.

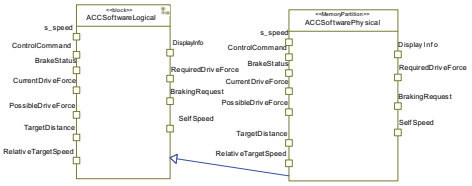

Figure 7: Seamless Refinement from System Architecture to Runtime Architecture.

```
}
```

In summary, AADL allows us to perform the following modeling steps seamlessly: (i) system architecture modeling, (ii) refinement of runtime architecture models, and (iii) translation into source code templates.

3.2 Approach #2: SysML / MARTE-Based System Development

3.2.1 System Architecture Modeling

The system architecture can also be described using the SysML internal block diagram (IBD) whose stereotype is «block»[3]. Figure 6 shows the system architecture model in SysML. In the figure, we can find similar components to those of the AADL model in Fig. 4. Due to space limitations, Fig. 6 only shows the logical software component ACCSoftwareLogical; however, the interfaces between the software component and its peripheral devices are also described in the system architecture model. In the SysML model, interfaces are defined with units by «flowSpecification», «flowProperty», and «ValueType» similarly to Listing 1.

3.2.2 Runtime Architecture Modeling

As shown in Fig. 2, SysML cannot describe the runtime architecture. On the other hand, MARTE is equipped with a sufficiently wide vocabulary of elements to allow runtime architecture modeling, e.g., «MemoryPartition» and «swSchedulableResource» represent processes and threads, respectively. Thus, here we use MARTE in a complementary way.

First, the top-level physical software component, which is a process, is described using «MemoryPartition». The process model inherits the interface definitions from the logical software component in the system architecture. This can be depicted as shown in Fig. 7 where the inheritance mechanism allows the process (ACCSoftwarePhysical) to inherit the interfaces from the logical software component (ACCSoftwareLogical). Next, the top-level software component (ACCSoftwarePhysical) is decomposed into subcomponents (threads), which can be represented by «swSchedulableResource». Figure 8 shows that ACCSoftwarePhysical consists of six threads, in the same way as shown in Fig. 5.

[3]«» indicates a stereotype.

Table 1: Comparison of Approaches #1 and #2.

Viewpoints	Requirements of ADLs	Approach #1: AADL	Approach #2: SysML / MARTE	Relevant Sections
(1) Large-Scale Systems	Hierarchical Architecture	**`System`** components can have a hierarchical architecture.	«Block» can have a hierarchical architecture.	Sects. 3.1.1 and 3.2.1
	Compositional Architecture	Component type definition provides formal interface specifications.	«flowSpecification», «flowProperty», and «ValueType» formally define interfaces.	
(2) Real-Time Systems	Real-Time Properties	Several properties such as deadline and execution time can be specified.		Sects. 3.1.2 and 3.2.2
(3) Embedded Systems	Software Architecture	Both the logical and physical software architecture can be described.		Sects. 3.1 and 3.2
	Hardware Architecture (Execution Platform)	Several elements such as **`device`**, **`bus`**, and **`processor`** are available.	HRM (Hardware Resource Modeling) of MARTE is available.	N/A
(4) Legacy Code Handling	*Code Generation*	Automatic / Manual	Manual	Sects. 3.1.3 and 3.2.3
(5) Distributed Development	Open Standard	SAE Standard	OMG Standard	Sect. 2
	Seamless Refinement	By **`extends`** and **`refined to`**.	By the inheritance mechanism.	Sects. 3.1.2 and 3.2.2
(6) High-Level Quality Assurance	Formal Notation	Specified by BNF (Backus-Naur-Form).	Specified by UML Metafile.	N/A
	Formal Verification	OSATE provides formal analyses, e.g., schedulability analysis,	Not supported by a standard tool.	Sects. 3.1.2 and 3.2.2
(7) Large-Scale Product Line	*Variability Modeling*	External specifications can be varied using **`extends`** and internal specifications using **`refined to`**.	Only external specifications can be varied using the inheritance of «block».	N/A, but discussed in [8].

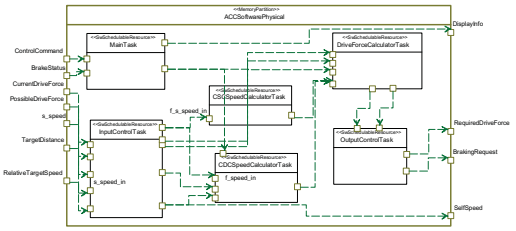

Figure 8: Runtime Architecture Model in MARTE (cf. Fig. 6).

Similar to the case of AADL in Sect. 3.1.2, we can add certain real-time properties. However, unlike the AADL model, there does not exist any standard tool environments that can bring formal analysis based on these real-time properties. Therefore, if we need these kinds of analysis, we must translate the MARTE models into AADL models that are analyzable by OSATE.

3.2.3 Code Generation

Similar to the method of Sect. 3.1.3, we can generate source code templates from the runtime architecture model shown in Fig. 8. Unfortunately, unlike in AADL, no code generation guidelines or automatic code generators are available. Therefore, hand coding that considers the target platform is necessary.

As is shown in Fig. 2, neither SysML not MARTE can cover all the development phases of automotive systems by itself. However, the above discussion shows that the combined use of SysML and MARTE allows us to perform each step of model-based automotive system development.

3.3 Comparison of Model-Based Approaches

Table 1 summarizes the qualitative characteristics of the two approaches. In addition to the five viewpoints mentioned in Sect. 2, this table contains two additional viewpoints: (6) high-level quality assurance (accountability to customers) and (7) large-scale product line (variation handling).

The table shows that both approaches bring similar effects from each viewpoint, except for the following three items: code generation, formal verification, and variability modeling. Based on these criteria, approach #1 (AADL) is superior to approach #2 (SysML and MARTE). Therefore, so far as these items are important to a given development project, AADL is a better candidate than SysML and MARTE. Otherwise, we can choose a suitable approach while considering the information literacy of engineers who are involved in a project.

4. CONCLUSION AND FUTURE WORK

In this paper we discussed ADL-based approaches to automotive system development. We derived two different kinds of approaches: (i) an AADL-based method, and (ii) a collaborative modeling method using SysML and MARTE. Detailed modeling steps of each approach were explained through modeling trials based on an ACC system. These trials also proved that both approaches can cover similar phases of automotive system development. Moreover, by comparing the two approaches from multiple viewpoints, we clarified the differences between the approaches, namely among the three ADLs. The comparison also showed that both approaches offer several different advantages to automotive system development.

This paper focused on only the architectural aspects of automotive systems due to space limitations. It is also necessary to perform a wide-scope comparison of ADL-based approaches considering their behavioral aspects.

5. REFERENCES

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

[2] *Architecture Analysis & Design Language (AADL)*. AS5506A, SAE International, 2009.

[3] *OMG Systems Modeling Language (OMG SysML)*. Object Management Group, 2007.

[4] *A UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)*. Object Management Group. http://www.omgmarte.org.

[5] *EAST-ADL2*. ATESST (Advancing Traffic Efficiency and Safety through Software Technology). http://www.atesst.org.

[6] *Automotive Open System Architecture (AUTOSAR)*. http://www.autosar.org.

[7] J. Lemieux, *Programming in the OSEK/VDX Environment*. CMP, 2001.

[8] S. Shiraishi, "An AADL-based approach to variability modeling of automotive control systems," in *Model Driven Engineering Languages and Systems (MODELS2010)*, vol. I, pp. 346 – 360, 2010.