A Generation Method of ECU-Hardware-Dependent Description of Complex Device Drivers in AUTOSAR

Hideki Hirose[1] Hideki Takase[1] Kazuyoshi Takagi[1] Naofumi Takagi[1]

Abstract: In order to solve the increasing scale and complexity of automotive software development, AUTOSAR specification is provided as a standardized software development process. When developing automotive software based on AUTOSAR, it is required to develop the software by component-based way. ECU-Hardware-dependent components such as CDDs (Complex Device Drivers) and the OS have to be developed for each target ECU-Hardware. In this paper, we propose an automatic method for generating ECU-Hardware-dependent descriptions and corresponding CDD SW-Cs. In our method, a CDD is generated by combining Device Driver Form File and ECU-Hardware Configuration File. Device Driver Form File solves the difference between ECU-Hardware. Descriptions that depend on target ECU-Hardware are abstracted by DSL (Domain Specific Language). Such information is described in ECU-Hardware Configuration File. In addition, ARXML descriptions of SW-Cs can be generated by just describing their configuration in DSL. As a case study for the proposed method, we developed an automatic braking system using RC-car kit and RC-car control system. The contribution of our method is that it makes it possible to automatically generate a CDD SW-C compatible with AUTOSAR for each ECU-Hardware. When the ECU-Hardware is replaced by a different kind of ECU-Hardware, a CDD SW-C for the new ECU-Hardware can be easily obtained. Moreover, proper ARXML descriptions can be generated by the tool.Therefore, it is expected to bring the reduction of developers' load and the improvement of productivity in automotive software development.

Keywords: Automotive software, electronic control unit (ECU), design automation

1. Introduction

These days, the scale and complexity of automotive software have been increasing along with a growing demand for the functionality. In addition, the number of ECUs (Electronic Control Units) mounted on vehicles has increased. For example, luxury cars contain more than 140 ECUs[1].

To solve the above issue, a consortium of several companies in automotive domain provided AUTOSAR (AUTomotive Open System ARchitecture) [2] as an automotive software specification. AUTOSAR aims to standardize automotive software architecture and to realize component-oriented development of automotive software[3]. In AUTOSAR, automotive software is split into multiple layers. Each component is located in its corresponding layer. Since the application layer does not depend on hardware, application components can be reused in other automotive systems. In contrast, CDDs (Complex Device Drivers) and part of the OS are located in the layer which is dependent on hardware. Thus, these components must be developed for each target hardware.

When developing a CDD, it is required to describe the design of the CDD component. It must be described in ARXML, a specific XML format based on AUTOSAR schema. ARXML makes the development process complicated due to the lack of readability.

In this paper, we propose an automatic method for generating a CDD SW-C (SoftWare Component). A CDD SW-C can be generated just by writing its ECU-Hardware-dependent information in DSL (Domain Specific Language). In addition, ARXML description of the SW-C can be generated just by describing its configuration in DSL.

In the proposed method, CDD is generated by merging Device Driver Form File and ECU-Hardware Configuration File. Device Driver Form File solves the difference between various kinds of ECU-Hardware. ECU-Hardware-independent description among the original device driver is implemented in it. On the other hand, ECU-Hardware-dependent information such as operating frequency and pin location is described in ECU-Hardware Configuration File.

The information of SW-Cs which developers want to integrate into an existing automotive system is described in Additional SW-C Configuration File. Integrated ARXML files are generated by merging Additional SW-C Configuration File and ARXML files in the existing system.

As a case study for the proposed method, we developed an automatic braking system for RC-car kit and RC-car control system. We implemented our method as a CDD generator with python and developed Device Driver Form

[1] Kyoto University

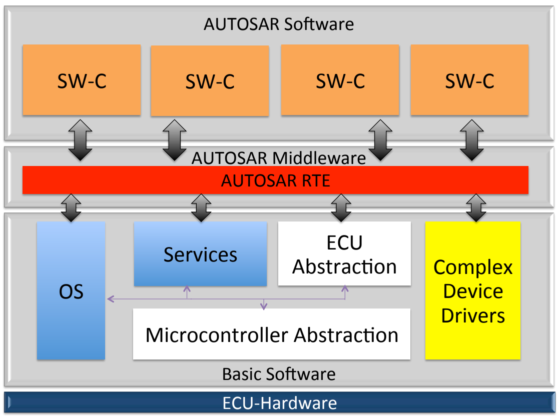


Fig. 1 ECU architecture in AUTOSAR

File for a range detection sensor with C. Next, we created ECU-Hardware Configuration Files for two kinds of ECU-Hardware. For each of them, we obtained a CDD SW-C for the range detection sensor by inputting those files into the generator. In addition, we developed an application SW-C which automatically brakes the vehicle according to the distance the sensor provides. We described the configuration of the CDD SW-C and the application SW-C in DSL, and combined the descriptions with the existing ARXML files by the generator. We executed the integrated system on the RC-car kit and confirmed that it worked correctly.

The contribution of our study is that CDD SW-Cs can be generated just by writing DSL descriptions. When the ECU-Hardware is replaced by a different one, a CDD SW-C for the new ECU-Hardware can be easily obtained. Moreover, a proper ARXML description can be generated from the DSL descriptions of SW-Cs. It means that integrating CDD SW-Cs into existing automotive systems can be completed with only our tool. Therefore, it is expected that the labor and the cost for CDD development will be reduced.

2. AUTOSAR

2.1 ECU Architecture in AUTOSAR

As shown in **Fig. 1**, in AUTOSAR, ECU architecture consists of four layers: AUTOSAR Software layer, AUTOSAR Middleware layer, Basic Software layer and ECU-Hardware.

AUTOSAR realizes component-based development. Therefore, development is carried out for each component. Moreover, when developing components in higher layers, developers don't have to take account of physical configuration. Thus, it is easy to improve components and to reuse them in other automotive systems, and it leads to the reduction of the cost of development[4]. However, some components in Basic Software layer, such as CDDs, depend on ECU-Hardware. Thus, they need to be developed for each target ECU-Hardware.

2.2 Software Development Flow

The flow of developing automotive software based on AUTOSAR is the following:

(1) SW-C Description
Determining what function to realize with each SW-C and designing connections between SW-Cs.

(2) System Description
Describing the definitions of the system, including ECUs and SW-Cs.

(3) ECU Configuration
Collecting information necessary for generating Basic Software and RTE for each ECU.

(4) ECU Integration
Implementing initialization processing which depends on ECUs, generating Basic Software and RTE, and building the entire software.

All the information, such as composition of the system, must be described in ARXML[5] ARXML is a specific XML format based on AUTOSAR schema. A description example of ARXML is shown in **Fig. 2**. Ports, runnables and interfaces are described with tags specified by AUTOSAR. Since ARXML lacks readability, it is difficult to edit ARXML files by hand.

3. AN AUTOMATIC GENERATION METHOD OF COMPLEX DEVICE DRIVERS

3.1 Overview

As a solution to the issues of CDD development, we propose a method to automatically generate CDD SW-Cs.

An overview of the CDD generator is shown in **Fig. 3**. Device Driver Form File, ECU-Hardware Configuration File, original ARXML files and Additional SW-C Configuration File are input to the generator. SW-C Generator generates a CDD SW-C by merging Device Driver Form File and ECU-Hardware Configuration File. ARXML Generator appends ARXML description of SW-Cs to original ARXML files.

3.2 SW-C Generator

Fig. 4 shows a description example of a Device Driver Form File and an ECU-Hardware Configuration File.

In Device Driver Form File, only ECU-hardware-independent descriptions among the original device driver is implemented. ECU-Hardware-dependent codes are described with "Abstract Expressions" like '`@<desc>:<name>`'. In `<desc>`, what kind of code is abstracted is given, such as function (func), value and code block (block). For example, '`@func:READ_PWM`' means that a function is abstracted and named '`READ_PWM`'. The ECU-Hardware-independent part is written in the language used in the existing automotive systems, such as C.

In ECU-Hardware Configuration File, information which is dependent on ECU-Hardware, such as operating frequency and pin location, is described in DSL. The codes corresponding to the Abstract Expressions in Device Driver Form File are described with separating symbol '`:`' and closing symbol '`$$`' like following:

```
<name> : <codes>$$
```

Abstract Expressions can represent various kinds of codes,

```
<COMPLEX-DEVICE-DRIVER-SW-COMPONENT-TYPE>
  <SHORT-NAME>CddRadarControl</SHORT-NAME>
  <PORTS>
    <P-PORT-PROTOTYPE>
      <SHORT-NAME>DistanceSrv</SHORT-NAME>
      <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">
        /RcCar/IfDistance
      </PROVIDED-INTERFACE-TREF>
    </P-PORT-PROTOTYPE>
  </PORTS>
  <INTERNAL-BEHAVIORS>
    <SWC-INTERNAL-BEHAVIOR>
      ......
      <RUNNABLES>
        <RUNNABLE-ENTITY>
          <SHORT-NAME>MeasureDistance</SHORT-NAME>
          <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
          <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
          <SYMBOL>MeasureDistance</SYMBOL>
        </RUNNABLE-ENTITY>
      </RUNNABLES>
    </SWC-INTERNAL-BEHAVIOR>
  </INTERNAL-BEHAVIORS>
</COMPLEX-DEVICE-DRIVER-SW-COMPONENT-TYPE>
```

Fig. 2 A description example of ARXML

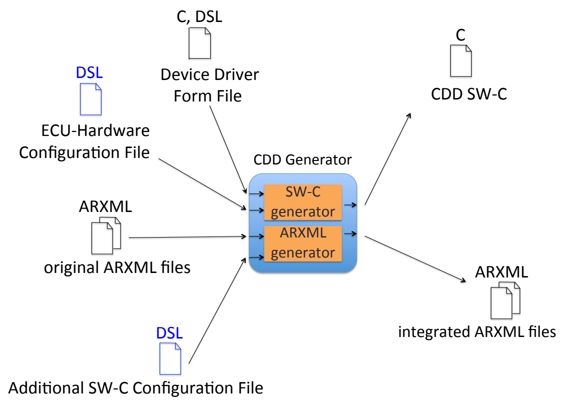
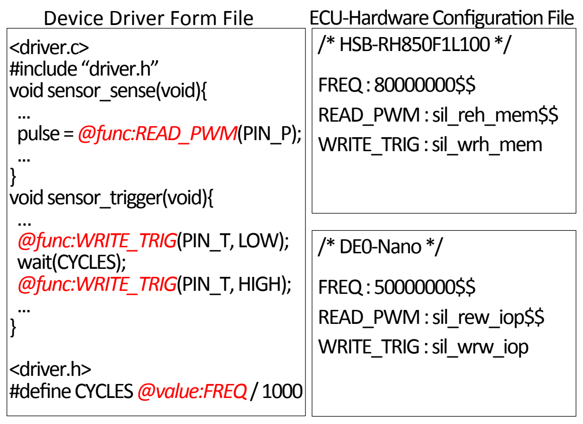


Fig. 3 An overview of CDD generator



Device Driver Form File

```
<driver.c>
#include "driver.h"
void sensor_sense(void){
 ...
 pulse = @func:READ_PWM(PIN_P);
 ...
}
void sensor_trigger(void){
 ...
 @func:WRITE_TRIG(PIN_T, LOW);
 wait(CYCLES);
 @func:WRITE_TRIG(PIN_T, HIGH);
 ...
}

<driver.h>
#define CYCLES @value:FREQ / 1000
```

ECU-Hardware Configuration File

```
/* HSB-RH850F1L100 */
FREQ : 80000000$$
READ_PWM : sil_reh_mem$$
WRITE_TRIG : sil_wrh_mem
```

```
/* DE0-Nano */
FREQ : 50000000$$
READ_PWM : sil_rew_iop$$
WRITE_TRIG : sil_wrw_iop
```

Fig. 4 A description example of files input to SW-C Generator

such as parameters and functions. In Definition File, what kind of code should be described in ECU-Hardware Configuration File for each Abstract Expression is described. As shown in **Fig. 5**, Definition File is described in XML. A brief description, name and type are provided for each Abstract Expression in <exp>. If an Abstract Expression represents a function, the type of the value the function returns and the arguments of the function are also designated. <args> contains a certain number of <arg>. A brief description, number and type are contained in each <arg>. The number stands for the order of the argument. It is supposed that Definition File is provided for each device along with Device Driver Form File.

In the example code shown in Fig. 4, three Abstract Expressions are defined. As described in Definition File in Fig. 5, '`READ_PWM`' is a function for reading the PWM pin connected to the sensor. It returns a *uint16* value and has one *void ** argument as the target pin. '`WRITE_TRIG`' is a function for writing data into the TRIGGER pin connected to the sensor. Its return-value is *void* and it has two arguments. One is for the pin and its type is *void **, and the other is for data, with its type *uint16*. '`FREQ`' stands for the *int* value of the frequency of ECU-Hardware in use.

The generator refers to Definition File and checks if the code described in ECU-Hardware Configuration File is correct. It confirms the type of code, the number of arguments and the type of arguments.

When using this tool, developers have to create ECU-Hardware Configuration File for the target ECU-Hardware according to Definition File. Inputting ECU-Hardware Configuration File and Device Driver Form File into the SW-C generator, they can obtain a CDD SW-C for the target ECU-Hardware. As a use case for SW-C Generator, we suppose developments to replace ECU-Hardware in the existing automotive systems. In such cases, developers don't have to re-create CDD SW-Cs for the new target ECU-Hardware. Instead, they write some description in DSL to obtain those CDD SW-Cs.

3.3 ARXML Generator

Fig. 6 shows a description example of Additional SW-C Configuration File. The part written in red corresponds to the ARXML description in Fig. 2. This file contains the information of SW-Cs which developers want to integrate into the existing automotive systems, interface for those SW-

```
<abstract-expressions>
    <exp>
        <desc>read the PWM pin</desc>
        <name>READ_PWM</name>
        <exp-type>function<exp-type>
        <return-value-type>uint16</return-value-type>
        <args>
            <arg>
                <desc>target pin</desc>
                <arg-number>1</arg-number>
                <type>void *</type>
            </arg>
        </args>
    </exp>
    <exp>
        <desc>write the data into the TRIGGER pin</desc>
        <name>WRITE_TRIG</name>
        <exp-type>function</exp-type>
        <return-value-type>void</return-value-type>
        <args>
            <arg>
                <desc>target pin</desc>
                <arg-number>1</arg-number>
                <type>void *</type>
            </arg>
            <arg>
                <desc>data to write</desc>
                <arg-number>2</arg-number>
                <type>uint16</type>
            </arg>
        </args>
    </exp>
    <exp>
        <desc>clock frequency of the target ECU-Hardware</desc>
        <name>FREQ</name>
        <exp-type>int</exp-type>
    </exp>
</abstract-expressions>
```

Fig. 5 A description example of Definition File

```
#DESCRIPTION
##INTERFACE
IfDistance:
  Protocol: ClientServer
  Operations:
    OpGetIfDistance:
      Args:
        command:
          Type: IDT_Distance
          Direction: OUT$$
##COMPONENTS
###Applications
###ComplexDeviceDrivers
CddRadarControl:
  P-Ports:
    DistanceSrv:
      Interface: IfDistance
  Internal:
    …
    MeasureDistance:
      MinimumStartInterval: 0.0
      CanBeInvokedConcurrently: false$$
##CONNECTORS
CddRadarControl_DistanceSrv_to_AutoBrakeManager_DistanceClt:
  Provider: CddRadarControl/DistanceSrv
  Requester: AutoBrakeManager/DistanceClt$$
#CONFIGURATION
CddRadarControl:
  MeasureDistance:
    Event: DistanceSrv_OpGetIfDistance$$
```

Fig. 6 A description example of Additional SW-C Configuration File

Cs, and connectors which connects the existing SW-Cs with them. The information to be added to System Description File and Configuration File is described in Description section and Configuration section, respectively.

Description section starts with '`#DESCRIPTION`', followed by blocks for communication interfaces, SW-Cs and connectors, which start with '`##INTERFACE`', '`##SWCS`', '`##CONNECTORS`', respectively. The block for SW-Cs consists of two sub-blocks for Application SW-Cs and CDD SW-Cs, starting with '`###Applications`' and '`###ComplexDeviceDrivers`', respectively. In each block, the component name with separating symbol '`:`' is written in the first line. In the following lines, the information of the component is described at one indention level (two spaces) deeper. The blocks end with closing symbol '`$$`'. Items to describe in Description section are listed in **Table 1**. An item and its corresponding description are separated by a separating symbol '`:`'. When enumerating parameters for operations and data, the item name is declared in the first line and the name of the operations or the data is described in the next line with one indention level deeper, followed by an enumeration of the parameters with another indention level deeper.

Configuration section starts with '`#CONFIGURATION`'. In the following lines, startup information for runnables of a SW-C is described in a block. The symbols and the indention rules are the same as those in Description section. The items to describe in this section are listed in **Table 2**.

The System Description File and the Configuration File of the existing system are input to the tool. The generator refers to Additional SW-C Configuration File and integrates the information into those ARXML files.

With ARXML generator, developers can complete CDD integration into existing systems without using expensive commercial tools. Therefore, it reduces the cost of the development of CDDs.

4. CASE STUDY

We implemented the proposed method as CDD generator with Python. With the use of it, we developed a CDD of a range detection sensor for two kinds of ECU-Hardware: HSBRH850F1L100[6] and DE0-Nano[7]. In addition, we developed an automatic braking application which works according to the distance provided by the sensor. We integrated the application and the CDD SW-C into the existing system and confirmed that the integrated system worked

Table 1 Items to describe in Description section

Item	Description
Protocol	The protocol of an interface between SW-Cs
Operations	Enumerate operations called in communication with Client-Server protocol
Args	Enumerate data exchanged in the operation
Type	The type of data
Direction	The direction of data
P-Ports	Enumerate sending ports of a SW-C
R-Ports	Enumerate receiving ports of a SW-C
Interface	The interface the port connects to
Internal	Enumerate internal behaviors of a SW-C
Provider	Sender SW-C and its port
Requester	Receiver SW-C and its port

Table 2 Items to describe in Configuration section

Item	Description
Event	Events which wake the runnable

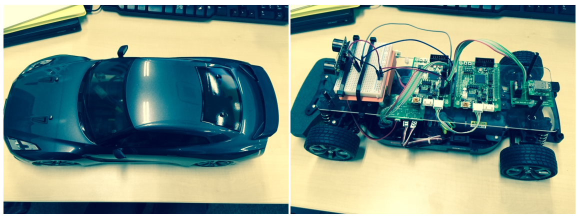


Fig. 7 The appearance of RC-car kit

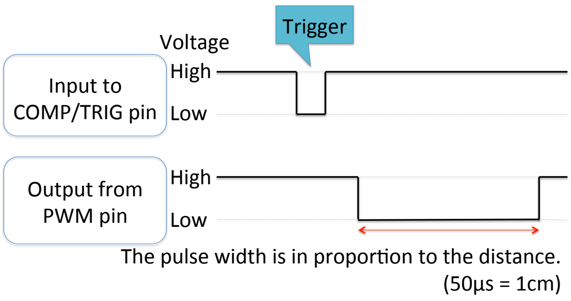


Fig. 8 Behavior of URM37

correctly on the RC-car kit.

4.1 Target

4.1.1 RC-car Kit

RC-car kit[8], which realizes to control RC-cars by the use of microcomputer boards and FPGA boards as ECU-Hardware, is available on the market. HOKUTO DENSHI's HSBRH850F1L100 and Altera's DE0-Nano can be used as ECU-Hardware. This kit makes it possible to simulate development of automotive software for real cars on a small scale. The appearance of the kit is shown in **Fig. 7**.

4.1.2 RC-car Control System

Center for Embedded Computing Systems, Nagoya University and its cooperators developed RC-car control system according to AUTOSAR specification. TOPPERS/ATK2[9], provided by TOPPERS Project[10], is employed for the OS of the system. The target hardware of this program is HSBRH850F1L100 and DE0-Nano. In addition, the program for HSBRH850F1L100 has two versions: the OS task version and the SW-C version. In the OS task version, all the applications are implemented as OS tasks. On the other hand, they are implemented as SW-Cs in the SW-C version of the program. Only the OS task version of the program is provided for DE0-Nano.

As specified in AUTOSAR, RTE is required for running SW-Cs. RTE can be generated with the use of RTE generator[11] provided by TOPPERS Project.

4.1.3 Range Detection Sensor

We employed URM37[12] as a range detection sensor. It measures the distance to obstacles ahead by emitting ultrasonic waves. Behavior of URM37 is shown in **Fig. 8**. When a trigger pulse is sent to COMP/TRIG pin of URM37, it measures the distance and returns a pulse via PWM pin. The width of the returned pulse is in proportion to the measured distance, 50 μm corresponding to 1 cm.

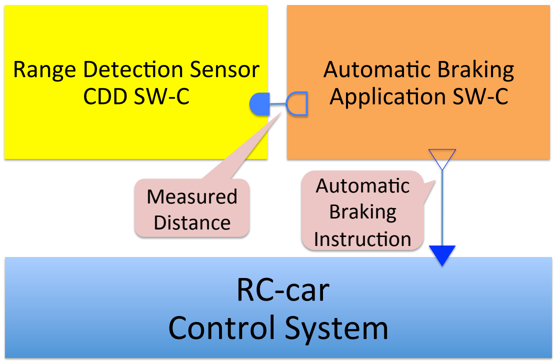

Range Detection Sensor CDD SW-C
Automatic Braking Application SW-C
Measured Distance
Automatic Braking Instruction
RC-car Control System

Fig. 9 The configuration of the system we developed

4.2 Applying Our Method to CDD Generation

A CDD of URM37 is required to have roughly two functions: pulse sending/receiving function and pulse width measuring function. Thus, we implemented Device Driver Form File of URM37 which has an initialization module and modules for the two functions. Codes for the pin location, the operating frequency and other processings depending on ECU-Hardware were abstracted in Device Driver Form File based on the rule set out in 3.2. The number of lines of Device Driver Form File was 112. 15 places were abstracted by 14 Abstract Expressions.

Next, we implemented ECU-Hardware Configuration Files for HSBRH850F1L100 and DE0-Nano. Concrete codes for Abstract Expressions in Device Driver Form File were described complying with the rule in 3.2. The number of lines was 46 for HSBRH850F1L100 and 14 for DE0-Nano.

We obtained a CDD SW-C for each ECU-Hardware by inputting those files into the SW-C Generator of the tool. The number of lines of the generated CDD SW-C was 144 for HSBRH850F1L100 and 112 for DE0-Nano.

4.3 Implementation of Automatic Braking System

We developed an automatic braking application SW-C, which automatically stops the vehicle when the distance to obstacles ahead is shorter than the threshold. The configuration of the system with the automatic braking application is shown in **Fig. 9**. The automatic braking application SW-C calls the range detection sensor CDD SW-C to get the distance to obstacles ahead. The automatic braking application SW-C determines whether automatic braking should be executed or not, and sends an instruction to the RC-car control system.

We utilized the ARXML Generator when integrating the application with the SW-C version of the RC-car control system for HSBRH850F1L100. We described the information of the range detection sensor CDD SW-C and the automatic braking application SW-C on Additional SW-C Configura-

Table 3 Lines of codes required to write and generated by the tool

ECU-Hardware	required to write	generated by the tool
HSBRH850F1L100	46 lines	144 lines
DE0-Nano	14 lines	112 lines

tion File. We input the ARXML files of the original RC-car control system and Additional SW-C Configuration File into the ARXML Generator, and obtained integrated ARXML files. Since the SW-C version of the RC-car control system is not provided for DE0-Nano, we developed necessary ARXML files without the tool.

We built the software and downloaded it onto each of HSBRH850F1L100 and DE0-Nano. We confirmed that the instruction of automatic braking was normally transmitted on RC-car kit.

5. RELATED WORK

There are some related approaches that help developing automotive software.

In [13], the authors propose use of DSL in modeling situation awareness for Car-to-X applications. In their approach, application codes are generated from DSL description that defines the way the application runs. It helps porting applications into other automotive systems that use different languages.

[14] proposes a protocol analyzer with application level communication abstractions and complex scenarios. The proposed protocol analyzer can specify, monitor and test complex scenarios. The authors also presented a DSL based on their approach so that automotive software developers can use their approach without special knowledge.

In [15], the authors proposes a basic software (BSW) configuration generator. Their generator refers to a spreadsheet which defines hardware-software interface (HSI) information, and generates BSW drivers and software interfaces. Their approach closes the gap between model-driven system engineering tools and software engineering tools, and enhances model-based software engineering framework for automotive software.

[16] proposes an approach to relocate AUTOSAR components between ECUs without changing the overall functionalities. It helps developers compare possible architectures during the design or prototyping phases.

6. CONCLUSION

We have proposed a method to generate CDD SW-Cs for the purpose of reducing the labor and the cost for CDD development and improving the efficiency of development of automotive software. When developing CDD SW-Cs with the proposed method, developers describe the necessary information which is dependent on the target ECU-Hardware on ECU-Hardware Configuration File with DSL. A CDD SW-C can be obtained by inputting Device Driver Form File and ECU-Hardware Configuration File to the proposed CDD generator. Moreover, by describing the configuration of SW-Cs on Additional SW-C Configuration File and inputting it with the ARXML files of the existing system into the tool, developers can obtain merged ARXML files.

The proposed method saves time and effort for developing CDDs. CDD SW-Cs can easily be generated just by writing DSL descriptions. When the ECU-Hardware is replaced by a different kind of ECU-Hardware, a CDD SW-C for the new ECU-Hardware can be easily obtained. In addition, when integrating developed SW-Cs into an existing automotive system, appropriately integrated ARXML files can be obtained without other ARXML editing tools.

In future work, we would evaluate our CDD generator by comparing a SW-C developed with the tool and one without it in the amount of code and execution time. We would also simplify the rule of descriptions in Additional SW-C Configuration File. In the current rule, information about one SW-C is divided into information for the Configuration section and information for the Description section. For convenience, we would like to put it into one place.

References

[1] Sakurai, T.: Current Status of Automotive Embedded Software : AUTOSAR and ISO 26262 (Reliability and Safety of Embedded Systems) (in Japanese), *Reliability Engineering Association of Japan*, Vol. 36, No. 4, pp. 197–205 (2014).

[2] AUTOSAR: AUTomotive Open System ARchitecture (online), `http://www.autosar.org/` (2016.07.04).

[3] Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K. and Lange, K.: AUTOSAR–A Worldwide Standard is on the Road, *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, Vol. 62 (2009).

[4] Suzumura, N. and Katsuki, S.: Overview and Trends of European Automotive Embedded Technology Approach (in Japanese), *EMB*, Vol. 9, pp. 1–12 (2009).

[5] Fennel, H., Bunzel, S., Heinecke, H., Bielefeld, J., Fürst, S., Schnelle, K.-P., Grote, W., Maldener, N., Weber, T., Wohlgemuth, F. et al.: Achievements and exploitation of the AUTOSAR development partnership, *Convergence*, Vol. 2006, p. 10 (2006).

[6] HOKUTO DENSHI: HSBRH850F1L100 (online), `http://www.hokutodenshi.co.jp/7/HSBRH850F1L100.htm` (2016.07.04).

[7] Terasic: DE0-Nano (online), `http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=593&PartNo=2` (2016.07.04).

[8] DENSHI, H.: RC-car kit (online, in Japanese), `http://www.hokutodenshi.co.jp/7/HSBRH850F1L100.htm#rccar` (2016.07.04).

[9] TOPPERS Project: TOPPERS/ATK2 (online), `https://www.toppers.jp/en/atk2.html` (2016.07.04).

[10] TOPPERS Project: `https://www.toppers.jp/en/index.html` (2016.07.04).

[11] TOPPERS Project: TOPPERS/A-RTEGEN (online, in Japanese), `https://www.toppers.jp/a-rtegen.html` (2016.07.04).

[12] DFRobot: URM37 V3.2 Ultrasonic Sensor (online), `http://www.dfrobot.com/wiki/index.php/URM37_V3.2_Ultrasonic_Sensor_(SKU:SEN0001)#Specification` (2016.07.04).

[13] Schafer, J. and Klein, D.: Implementing Situation Awareness for Car-to-X Applications Using Domain Specific Languages, *Vehicular Technology Conference (VTC Spring), 2013 IEEE 77th*, IEEE, pp. 1–5 (2013).

[14] Reichert, T., Klaus, E., Schoch, W., Meroth, A. and Herzberg, D.: A language for advanced protocol analysis in automotive networks, *2008 ACM/IEEE 30th International Conference on Software Engineering*, IEEE, pp. 593–602 (2008).

[15] Macher, G., Obendrauf, R., Armengaud, E. and Kreiner, C.: Automated generation of basic software configuration of embedded systems, *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, ACM, pp. 461–464 (2015).

[16] Saudrais, S. and Chaaban, K.: Automatic relocation of autosar components among several ecus, *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, ACM, pp. 199–204 (2011).