Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1575

Rapid Prototyping in AUTOSAR Based Systems

Owes Khan [1], Norbert English [1], Wolfram Hardt [1]

Abstract: Application prototypes involving complex algorithms for purposes like ADAS are often developed on PC based systems. For validation, these prototypes must be executed on automotive embedded systems. Furthermore, with increasing industry demand of AUTOSAR based ECUs, integration with AUTOSAR environment also becomes a necessity. The process in itself is time consuming due to the fulfilment of various steps involved in the AUTOSAR methodology. On the contrary, for rapid prototyping, time and effort for non-functional requirements should be minimal. A concept for automated integration of non-AUTOSAR compliant prototype codes within an AUTOSAR system, along with application specific configuration and generation of the complete system is proposed by this paper. The main aim is focused on memory usage reduction of AUTOSAR BSW along with the integration of prototype codes to facilitate an AUTOSAR Processor-in-Loop test environment for the application.

Keywords: AUTOSAR; Rapid Prototyping; Integration

1 Introduction

1.1 AUTOSAR

AUTomotive Open System Architecture is abbreviated as AUTOSAR, and is a standardised architecture for the development of software in the automotive industry. It is also a collaboration of automotive OEMs (Original Equipment Manufacturers), suppliers, tool vendors and other electronics, semiconductors and software companies. The main objective of AUTOSAR lies in formulating and establishing open standards in automotive E/E (Electrical & Electronic) architectures to facilitate a basic infrastructure for development of automotive software. With such an industry standard cost saving, quality and reduction in complexity are just some of the advantages].[Ro11][Fü10] The AUTOSAR consortium gives specifications for the main topics of Application Interfaces (API), architecture, methodology.

The standard AUTOSAR consists of three main layers formed by Application layer, RTE(Run Time Environment) and BSW(Basic SoftWare). The application layer is formed of SoftWare Components(SWC) which communicate with the RTE. The main application in perspective is located here. This layer is independent of target hardware.[KF09] RTE handles SWC-SWC

[1] Technische Universität Chemnitz, Technische Informatik, Straße der Nationen 62, 09111 Chemnitz, Germany
owes.khan/norbert.englisch/wolfram.hardt@informatik.tu-chemnitz.de

doi:10.18420/in2017_157

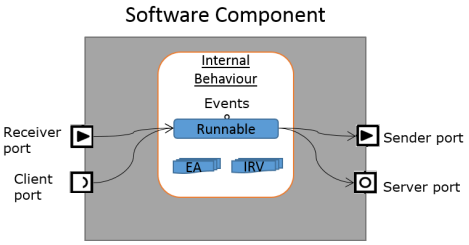

Fig. 1: AUTOSAR software component

and inter-ECU information exchange. It also serves as a medium to communicate with the lower layer of architecture. The main task of RTE is to make SWCs not dependent on a specific hardware allowing focus on application development[KF09] and is generated specifically for each configuration. BSW does not have any functionality in itself. This layer is the most hardware dependent part of all the three main layers. The main task of this layer is to provide hardware independent services to the upper layers. BSW includes many modules categorised under 5 main sections: Services, communication, ECU Abstraction, Complex Device Drivers (CDD) and the OS. Every module follows a fixed specification and provides APIs defined by AUTOSAR.[KF09] To work with AUTOSAR environment all the BSW modules need to be configured. A list of these modules can be found here [AU].

One important block from the AUTOSAR architecture for this paper is CDD. It is a mechanism or a part of AUTOSAR which comes into role when unsupported modules are to be implemented. Examples could be complex sensor and actuator control, non-standardised drivers. Software codes implemented in CDD obtain complete AUTOSAR conformance with an advantage of direct access to the hardware. However CDD must still obey RTE rules in order to communicate with other SWCs. One of the main aims of introducing CDD as a part of AUTOSAR is to provide solutions for migration from one AUTOSAR release to another. This avoids re-engineering of the complete existing application. Moreover, many hardware modules are not supported by AUTOSAR. For newer hardwares like MOST-bus or APIX2, no modules exists [DHM12]. Furthermore, the BSW modules can be accessed with very few limitations[KF09].

Figure 1 shows a typical structure of an SWC specified in AUTOSAR architecture. They contain application functionalities. The code here should follow specifications of AUTOSAR. As seen from the figure they should have a fixed structure. Ports allow external communication and the type of which must be specified using standardised interfaces. The communication defined can be achieved by using AUTOSAR specified formats in the main code. Runnables are the main executable entities which are either triggered by defined events or OS and can also be regarded as C functions. Exclusive areas (EA) are critical sections of the code. Further information about AUTOSAR SWCs can be found in the specifications [Sc15]. The structural and behavioural descriptions of SWCs are described through standard SWC descriptions, together with component code and its description, an SWC is complete.

Adding network architecture, ECU resource descriptions(abstract) and system constraints helps completing an application system architecture in AUTOSAR[Bu11] [Vo10]. This architecture has fixed specification so that it can be used to further generate RTE as well as BSW upon their configuration. Many tools exist to configure and create these layers individually.

Further information on AUTOSAR can be found here [AU].

1.2 Prototyping

Advanced driver assistance systems (ADAS) involve complex algorithms which need extensive prototyping before a final version could be released. The functionalities of such applications are mostly hardware independent and are developed on a PC platform using high level languages only to then port them to embedded systems. An example of such a system is Baselabs ADAS prototyping solution[Ba], where the algorithm is developed in C# and then converted to C. However. it still needs to be verified on a real world ECU environment for Processor-In-Loop / Hardware-In-Loop validations. The prototyping solutions also provide a generated C code after successful verification code on the PC based environment. The generated code C could then be easily integrated with embedded prototyping solutions. The code however still needs to be adapted for a specific target.

For rapid prototyping where time is an essential factor, solutions like the one provided by Baselabs prove to be very useful. With increasing popularity of AUTOSAR, rapid prototyping of ADAS applications must be done on AUTOSAR systems as well to make the ECUs AUTOSAR ready. However, due to multiple configurations and methodology[Bu11] involved in AUTOSAR, time and expert skills should be invested to make these prototypes ready since AUTOSAR adds complexity to the development process. Moreover, the C code generated by the prototyping solutions is non-AUTOSAR compliant and it needs to be adapted for integration. The focus for rapid prototyping should be on the functionality of the algorithm in itself and not on the underlying technicalities of the prototyping platform. For this purpose an automated support is needed for the complete process, from integration of non-compliant C code to the configuration and generation of AUTOSAR system specific to the provided application.

The goal of this paper is to define a method that can be used with AUTOSAR tool for automated rapid prototyping on AUTOSAR based systems. Figure 2 shows the involved steps.

Block 1 deals with taking C code and signal mapping information/network descriptions files as input to generate application system architecture along with extraction of information from provided C code for further steps. Details of the steps can be found in section 3.

Block 2 uses the system architecture and extracted information created by block 1 to generate configurations for AUTOSAR BSW specific to application provided in block 1. Block 2

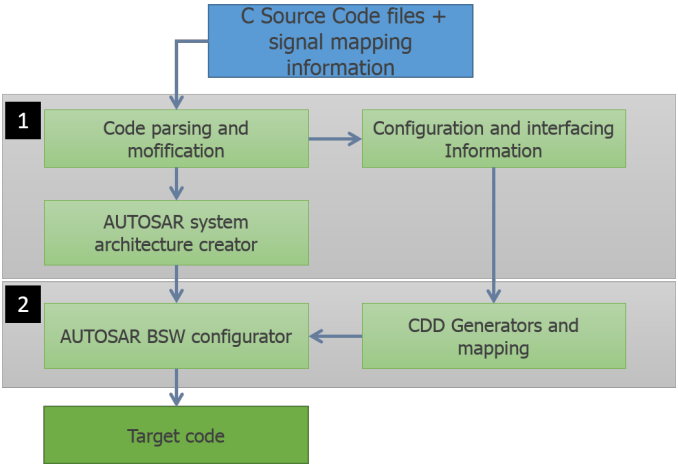

Fig. 2: Concept

also generates some BSW modules as a part of CDD. Further description can be found in section 4.

The aforementioned blocks provide support for generation of AUTOSAR system architecture, RTE and BSW which in turn are generated by tools. Our intention is not implement any AUTOSAR tools but to provide a method that can connect and automate the whole AUTOSAR methodology for a specific application. Also several templates for various descriptions of different elements in AUTOSAR already exist [AU].

2 Existing approaches

Several Model-in-Loop(PiL), Hardaware-in-Loop(HiL) and Processor-in-Loop(PiL) approaches already exist in the automotive domain. In the AUTOSAR domain methods such as Virtual Verification Platforms already exist that use a linux based HiL environment on x86 processors [DHM12]. CDDs is also extensively usage here for integration of the application layer and RTE. The prototypical code is then run on the platform under soft-real time conditions. Another very similar approach for rapid prototyping solution is shown in [MCW12].

Successful attempts have also been made to enable custom functionality in BSW while being AUTOSAR conformed [Ki13].

3 Integration with AUTOSAR

The generated non-AUTOSAR compliant C code explained in section 1 should be adopted as per AUTOSAR specification. For this the code needs to be parsed to obtain structural and behavioural information and then modified to fit AUTOSAR specifications as SWCs.

Since SWCs are very dependent, they cannot be separated just by parsing code. A thorough understanding of the logic implemented is required for its determination. Hence following the informal standard methods of programming, each C file is assumed to be as one component. Also since the functionalities of these C files are intended for automotive domain, we assume that a MISRA-C check has already been performed on them.

For parsing a source code C file, integrating it as an AUTOSAR SWC and generating its SWC description, two aspects need to be determined at first. These are the structural and behavioural aspects. Structural architecture is nothing but how the code skeleton looks like(SWC description) whereas the later is how a source code behaves during runtime(Internal behaviour).

3.1 Determining structural architecture

This consists of various static elements of the code which can be determined by parsing either header files or the C codes themselves:

- Function prototypes: These are represented as function declarations in source code. They need to be declared as Runnables in an AUTOSAR SWC.
- Data types: Declarations like structures and data types that are standard to a application, can be parsed to include them in the application system architecture. Also AUTOSAR already specifies a standard data type header file [Sc15], so duplication should be avoided. Global variables should be defined as Inter Runnable Variables(IRV) in a component and also their datatypes need to be declared in the main AUTOSAR system. Other inclusions like static or constant variables can be identified from C syntaxes to be included in the SWC description.
- Ports and interfaces: Ports and interface in AUTOSAR represent SWC-SWC, network or BSW communication. When a call to an external function or an addressing to a NW/HW communication (e.g. CAN) occurs a port can be created. The interface description is marked by the variable(s) taking part in the this communication. As one of the methods, the type of interface can be determined by checking for nested function calls. If it is the case then a sender-receiver interface can be mapped to these ports. For a different case where client-server interface is essential even if nested call is detected then it should be marked by a comment for the source code parser to determine. If a return value is expected, then this marks a need for a client-server

interface. Network communication statements like access to CAN messages require a sender-receiver interface.

- Mapping: Mappings need to be realised to standardise communication of SWCs through AUTOSAR RTE. This also configures RTE for its generation. Connection between SWC ports can be determined by detecting function calls. Signal mapping in case of network communications or IO handling in our concept is not needed since we are using application specific CDDs generated as a part of BSW. Details can be found in section 4.

3.2 Determining internal behaviour

- Data access: For each component data accesses can be determined by the variables in its function scope, the data elements of interfaces and global variables. To use features of measurement and calibration parameters, they can be mentioned through explicit comments.
- Runnable triggering: If the runnables are time triggered, then such conditions can be mentioned as part of comments in function prototypes for the parser to determine. Sender or server interfaces mapped to ports determine if triggering is needed. If there are calls internal to SWC runnable, this can be indicated as inter-runnable triggering.
- Exclusive areas: The critical sections of any program should be aware to the programmer. Hence comments can be used to mark them or directives can also be used to recognise them from a source C code.

3.3 Code adaptation

After determination of structural and behavioural properties of each source C file SWC descriptions of each component can be generated along with their mappings. However the original code still needs to modified to adapt as per the AUTOSAR specification. The adaptation involve the following:

- Inclusion of RTE headers:
  #include RTE_SWCname.h
- Replacement of external functional calls, Network/Hardware accesses by RTE statements
  RTE_Read_SWCname_portname_interfacename (&variable)
- Renaming of function names
  void SWCname_func1()
- Declaration of extra variables needed for RTE access.

```
void func1();
void func2();

uint8 globalVar;

void func1()
{
uint8 datatype1;
uint32 datatype2;
can_receive(datatype1); /* dbc signal1 */
component_call(datatype1,datatype2);
/* program code */
can_send(datatype2); /* dbc signal2 */
}

void func2(uint8 param1)
{
/*program code*/
func1();
/*exclusive area*/
/*program code*/
/*exclusive area end*/
}
```

Fig. 3: Sample non-AUTOSAR compliant C code

- Inclusion of RTE acknowledgements.
  e.g. *status = E_OK*
- Marking Exclusive areas in code.
  void Rte_Enter_uniquename ();
  void Rte_Exit_uniquename ();

The specification of RTE can be found here [Sc15]. However the formatting rules of these statements in our concept are incorporated in the parsing tool which can be easily adapted in case of a version change.

Figure 3 shows a typical implementation of functional codes in one C file. For better explanation, information that can be extracted from just this code are :

- Runnables: func1() and func2()
- Gobal variable(IRV): globalVar
- Ports: CAN message send and receive, external function call(component_call)
- interfaces: 2 sender-receiver interfaces from CAN communication, one client-server

or sender-receiver (can be determined through checking the other component) for component_call function.

- Exclusive area marked in func2()
- inter-runnable trigger to func1() from func2()
- a receive-event(data available) trigger for func1()

Once a successful parsing of all the component source codes has been carried out, SWCs and their mappings can be generated in any application layer AUTOSAR tool through its API invocations. Further import of network configuration files enables determining the topology to generate the system. The extracted information is also helpful for configuration of BSW which is explained in section 4.

4 Configuration of BSW and generation of application specific CDDs

On a successful set up of application layer tools, system description including ECU descriptions (using AUTOSAR standard template or if provided by tool) can be generated, which can be used in a BSW/RTE generator tool after system import. The descriptions are again generated in AUTOSAR standard formats that enable transferability. RTE at this point should be already configured by any tool that is able to import standard AUTOSAR system description. For configuration of AUTOSAR BSW, only the necessary modules are considered and the configurations of which are portable and directly affected by application specifics. The extracted information from integration process helps in creation of OS tasks where runnables can be mapped. In case of time triggering, OS alarm also needs should be set up. Initialisation functions can either be included as OS init tasks or added to SCHM (BSW module : Schedule Manager). Detection of such functions must be explicitly mentioned in comments for the parser to detect and extract this information.

Rapid prototyping platforms often are different from actual target hardware. Hence not all BSW configurations can actually be carried forward to the final realisation of application. Hence dynamic configurations like communication drivers, IO access etc., where changes in application creates an impact on the whole BSW stacks, we have used a concept that generates specific BSW for the application. in CDD. By using this approach, minimalistic and optimised drivers for HW abstraction can be used. The application layer still has the advantage of running in an AUTOSAR environment on the target hardware. No changes are required in any of the application architecture configurations.

Figure 4 shows the system overview for an application with generated CDDs. Mapping information acquired through integration process in section 3 is used in generating standardised AUTOSAR interfaces to communicate with SWCs. No communication signals were mapped during the integration process, this task is done by generated CDD stack directly. Integration process also includes signal information in source code which is adapted with

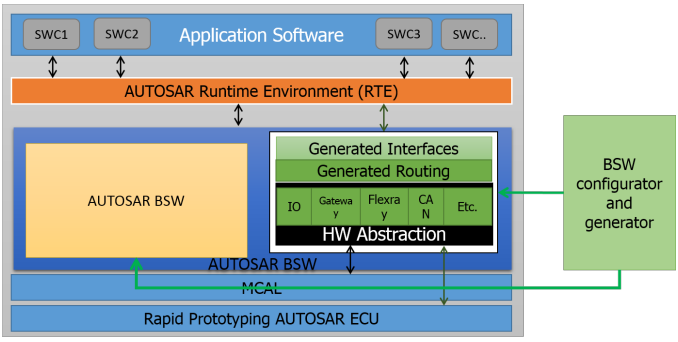

Fig. 4: AUTOSAR system overview using generated CDDs

RTE statements and thereby creating ports and interfaces in the main architecture for it. The generated CDDs use the same interfaces to read and write signal data. Information regarding the routing that is generated is acquired through standard network description files (e.g. dbc, fibex, or ldf).

Often for complex algorithms like in ADAS, many sensor data parameters need to be collected. These could also be large objects of struct type too. Generated routing eliminates the need of extra effort of SWCs/LOCs for data collection is also eliminated.

As it can be seen from the block diagram in figure 4, multiple communication protocols can be included in the CDD stack. These protocol units are generated using configuration information and they contain standardised interfaces for HW abstraction and routing. The layer consists of drivers that encapsulate micro-controller specific code and addresses which are also generated using device pertinent configuration. This layer can either be used with self implemented code or the MCAL layer provided by manufacturers can be used.

The dependencies to the AUTOSAR BSW when including such a CDD stack. Hence for which, configuration of the following BSW modules need to be generated:

- OS: configuration of AUTOSAR Interrupt service routines(ISR [Zh13]) which are used in CDD and creation of tasks that are not interrupt triggered.
- ECUM (ECU state manager): This module contains a list of driver initialisation functions. CDD stack based functions also need to be configured here.
- SCHM (Schedule manager): Main functions should be configured here.

The BSW configurator shown in figure 4 will be generating these configurations for use with any BSW tool. Ready templates which can be reused for BSW module descriptions are also provided by AUTOSAR.[KF09] For other module configurations like port that do not change based on application, templates can be used to avoid re-configuration.

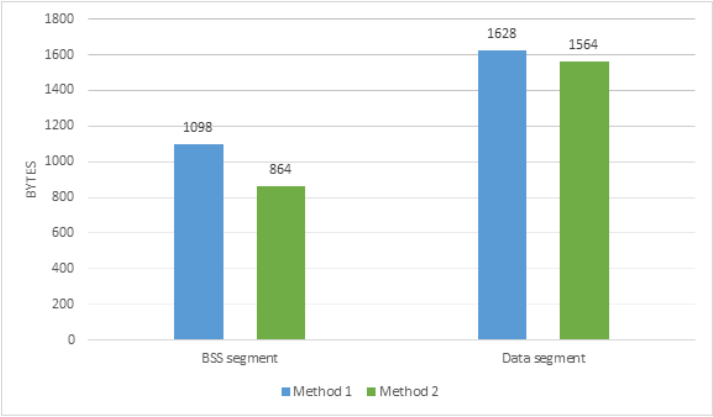

Fig. 5: Memory usage comparison chart 1

5 Implementation and results

To prove our theorised method, we have implemented a tool that performs our methods for generation of an AUTOSAR system.

For implementation, we have used a SPC560P[ST] board with compliant AUTOSAR version 3.0 application layer and BSW tools. A sensor fusion algorithm for processing raw images from a camera and ultrasonic sensor served as an application prototype. The code consisted of 1K LOC (Lines Of Code) that collects sensor data over CAN bus. The communication of CAN bus was specified using a dbc file consisting of 10 CAN signals. Verification of the processed data was done through passing over the output to a computer and thereby diagnosing it.

The generated code was checked for AUTOSAR linked errors using static test tool called ASTAS [NE16]. Upon a successful static test, the generated code was then compiled and executed on the mentioned platform.

Figure 5 and 6 show a comparison chart of .bss, code and data segment memory usages between two methods used for evaluation. As seen from the charts Method 2 was able to save much more space in all three segments as compared to Method 1.

Both the methods use the same application code. Method 1 was implemented by manually adapting and modifying code to integrate it in AUTOSAR environment, thereby also making complaint and using tools manually create a complete system for the respective application. Necessary AUTOSAR BSW modules including CAN stack were used. Method 2 was implemented using theorised concept. For this, AUTOSAR CAN stack was not included thereby replacing it with CDD concept mentioned in section 4.

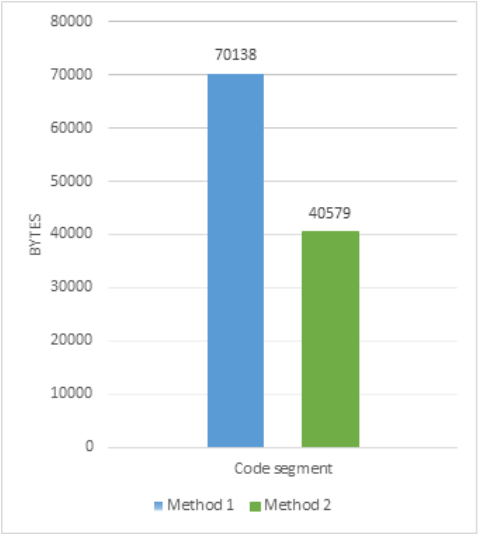

Fig. 6: Memory usage comparison chart 2

6 Conclusion

A concept for automating AUTOSAR based rapid prototyping is successfully theorised and implemented. Automation steps for integration of non-AUTOSAR compliant C code, configuration of related BSW and generation of application specific CDDs have been defined and implemented. The only required inputs included C source code files with some additional comments and network description files.The concept integrates provided C source code files within AUTOSAR system by modifying them and generating an application system architecture. Configurations for BSW modules that have direct impacts from application were done automatically. The generated CDDs are application specific, version independent and exist as a part of BSW directly interfaced with the main application SWCs. For implementation of Method 1, AUTOSAR skills and knowledge is required. However for Method 2, just basic knowledge is required and the complete implementation process takes about just a few minutes thereby reducing time and effort considerably. The theorised concept helps running a prototypical code on a real world ECU with an AUTOSAR conformed application, and hence enabling PIL validations. Furthermore, memory consumption is greatly reduced to accommodate more application components and still providing the needed environment. Since generated BSW configurations, application architecture and the modified code are according to AUTOSAR standards, they can be further reused in next development stages. The CDD stack is also usable apart from rapid prototyping in cases where driver optimisation or special drivers are needed.

References

[AU] AUTOSAR, "Platforms ". www.autosar.org/standards/classic-platform, Web. 11 June 2017.

[Ba] Baselabs, "Baselabs GmbH". www.baselabs.de, Web. 25 June 2017.

[Bu11] Bunzel, Stefan: Autosar–the standardized software architecture. Informatik-Spektrum, 34(1):79–83, 2011.

[DHM12] Deicke, Markus; Hardt, Wolfram; Martinus, Marcus: Virtual Validation of ECU Software with Hardware Dependent Components Using an Abstraction Layer. In: Simulation und Test für die Automobilelektronik. Expert Verlag, pp. 181–188, 2012.

[Fü10] Fürst, Simon: Challenges in the design of automotive software. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010. IEEE, pp. 256–258, 2010.

[KF09] Kindel, Olaf; Friedrich, Mario: Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis. dpunkt Verlag, 2009.

[Ki13] Kim, Taeho; Maeng, Ji Chan; Yoon, Hyunmin; Ryu, Minsoo: Understanding and Extending AUTOSAR BSW for Custom Functionality Implementation. In: Multimedia and Ubiquitous Engineering, pp. 1057–1063. Springer, 2013.

[MCW12] Martinus, Marcus; Cutura, Zoran; Würz, Thomas: A Solution for Virtual Integration and Reusability of Software in Infotainment Systems. ATZelektronik worldwide, 7(1):42–45, 2012.

[NE16] Norbert Englisch, Roland Mittag, Felix Hänchen Owes Khan Alejandro Masrur Wolfram Hardt: Efficiently Testing AUTOSAR Software Based on an Automatically Generated Knowledge Base. In: 7th Conference on Simulation and Testing for Vehicle Technology. pp. 87–97, May 2016.

[Ro11] Roebuck, Kevin: Encryption: High-impact Strategies-What You Need to Know Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. 2011.

[Sc15] Scheid, Oliver: AUTOSAR Compendium: Part 1 - Application & RTE. dpunkt Verlag, 2015.

[ST] ST Electronics, "RM0022 Reference Manual". `http://www.st.com/content/ccc/resource/technical/document/reference_manual/11/3c/05/79/a1/6a/4a/4a/CD00204027.pdf/files/CD00204027.pdf/jcr:content/translations/en.CD00204027.pdf`, Web. 11 June 2017

[Vo10] Voget, Stefan: AUTOSAR and the automotive tool chain. In: Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, pp. 259–262, 2010.

[Zh13] Zhu, Longfei; Liu, Peng; Shi, Jianqi; Wang, Zheng; Zhu, Huibiao: A timing verification framework for AUTOSAR OS component development based on Real-Time Maude. In: Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on. IEEE, pp. 29–36, 2013.