A Hierarchical Modeling Method for AUTOSAR Software Components

Ruyi Wu, Hong Li, Min Yao, Jinbo Wang, Yuhao Yang
College of Computer Science
Zhejiang University
Hangzhou, China
{wuruyi, lihong, myao, kimwang, hhbyyh}@zju.edu.cn

***Abstract*—Software component plays an important role in the AUTOSAR development process. In this paper, we propose a hierarchical modeling method for AUTOSAR software components, using which developers can conveniently create an AUTOSAR system. Taking into account the various software models of AUTOSAR application layer, we provide three diagram models to properly group the software components into four layers and a component library to improve reusability of component models. In addition, the hierarchical development of the model, along with auto-generation of XML description, is demonstrated based on the application case study of a door light control system.**

Keywords-AUTOSAR; software component; modeling; hierarchical

I. INTRODUCTION

In recent years, modern automotive E/E (Electrical and Electronics) systems have reached a high level of complexity, leading to a corresponding increase in the complexity of the deployed software [1]. Increasing complexity of automotive embedded software, increasing needs for software reusability and shorter development lifecycle all require new software architecture standards [2] [3]. The proposition of AUTOSAR attempts to provide solutions to these problems.

AUTOSAR (AUTomotive Open System ARchitecture) is an open standardized architecture for automotive electronics software, which completely separates an application from its infrastructure [4]. The application consists of software components which are discrete pieces of code offering one or more pieces of functionality. Each software component can only interact with other components and their related infrastructure via VFB (Virtual Functional Bus) [5]. A runnable (Runnable Entity) is a part of an atomic software component which is described by a sequence of instruction that can be started by the RTE (Runtime Environment). And the BSW (Basic Software) provides services to software components [4]. Fig. 1 shows the layered software architecture of AUTOSAR.

The AUTOSAR methodology [6] for guiding the development process can be divided into two categories: system configuration and ECU configuration. However, the methodology only provides some guidance but not a complete recipe, which fails to show how the workflows are carried out [7]. A major challenge facing AUTOSAR system development in general is to provide a useful modeling method to generate input for system configuration.

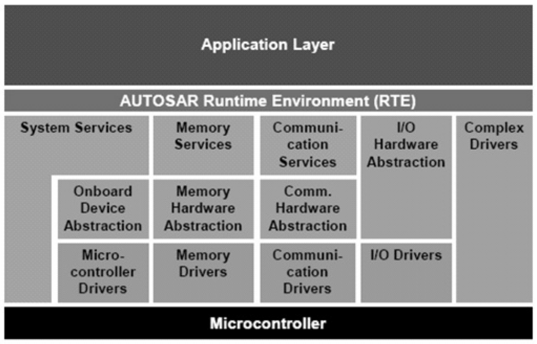


Figure 1. AUTOSAR Software Architecture [4]

Several companies have products that support AUTOSAR-based development, including Mathworks, Vector and EB, among others. Simulink from Mathworks is a widely used modeling tool that supports modeling, simulation and generation of AUTOSAR-compliant code [8]. However, the AUTOSAR library provided by Simulink cannot fully meet the AUTOSAR specification and is only suitable for low-level modeling. Vector and EB's products are mainly focused on the configuration, and lack related modeling tools [9] [10].

In this paper, we propose a hierarchical modeling method for AUTOSAR software components to meet the challenge. This is an interactive modeling method with four layers: top-level composition layer, component layer, internal behavior layer and implementation layer. The first two layers are supported by three diagram models, while the latter two layers are supported by Simulink. In addition, a component resource library is also provided along with this method, improving the reusability of software component. An application can be created by selecting components with the required functionality from this library. The output of the modeling method is an XML description generated in accordance to AUTOSAR XML Schema [11], which is used as an input of system configuration.

The remainder of this paper is organized as follows. In Section 2, we present the formal description of AUTOSAR software component models. We describe the process and implementation of hierarchical modeling method in section 3. In section 4, we offer a case study to describe an application example of DLC (Door Light Control) using this method. Finally, the concluding section summarizes this paper and discusses future work.

II. Formal Description for Software Component Models

In this section, we introduce the definitions of major elements in AUTOSAR software component, including various components and interfaces.

A. *Definition of Component*

Definition 1. ***Component***: A Component is an encapsulation of AUTOSAR applications. Each component encapsulates part of functionality of the application. $C = C_c \cup A_{swc} \cup C_{swc}$, where C_c is a set of compositions; A_{swc} is a set of atomic software components; C_{swc} is a set of calibration software components.

Definition 2. ***Composition***: As a special component, composition includes a group of function-related components.

Definition 3. ***Atomic Software Component***: Atomic SWC (Atomic Software Component) is the smallest indivisible component entity, which cannot be distributed to multiple ECUs.

Definition 4. ***Calibration Software Component***: Calprm SWC (Calibration Software Component) is a special type of components, providing calibration function to automotive electronic systems.

Definition 5. ***Port***: Port is the interaction point of components. Port $P = P_p \cup P_r$, where P_p is a set of PPorts (Provided Port); P_r is a set of RPorts (Required port).

Definition 6. ***Connector***: Connector connects ports of components. Connector $R = R_a \cup R_d$, where R_a is a set of assembly connectors that represent the communication of components within a composition and R_d is a set of delegation connectors which interconnect components that are part of a composition and the composition. Remark:

- $R \subseteq P \times P$
- $R_a \subseteq P_p \times P_r$
- $R_d \subseteq P_p \times P_p \cup P_r \times P_r$

B. *Definition of and Interface*

Definition 7. ***Interface***: An interface consists of several ports, which defines the communication information between ports of components. $I = I_{cs} \cup I_{sr} \cup I_c$, where I_{cs} is a set of Client-Server interfaces; I_{sr} is a set of Sender-Receiver interfaces; I_c is a set of calibration interfaces.

Definition 8. ***Sender-Receiver Interface***: A S-R (Sender-Receiver) interface is used for the case of sender-receiver communication. $I_{sr} = (D_e, M_g)$, where D_e is a set of data elements representing the data units exchanged between sender and receiver, M_g is a set of mode groups which describe a collection of mode switches.

Definition 9. ***Client-Server Interface***: A C-S (Client-Server) interface defines two sets: operations and possible errors. $I_{cs} = (O_p, P_e)$, where O_p is a set of operations that can be invoked by a client and implemented by a server, P_e is a set of possible errors that may occur during this communication.

Definition 10. ***Calibration Interface***: Calibration Interface is a special kind of interface that allows modules to access static calibration parameters. I_c is composed of calprm elements.

III. Architecture And Implementation

The hierarchical modeling method for AUTOSAR software component is a visual modeling environment based on Eclipse, which is used to build AUTOSAR software component model and generate the corresponding description files. The process of software component modeling is shown in Fig. 2. This modeling method is divided into four layers, plus a component resource library to support the reusability for model entities. Furthermore, it is compatible with Matlab/Simulink, and it can generate detailed descriptions and C code. In addition, the modeling process is interactive and the generated description can be imported into the modeling environment for incremental development.

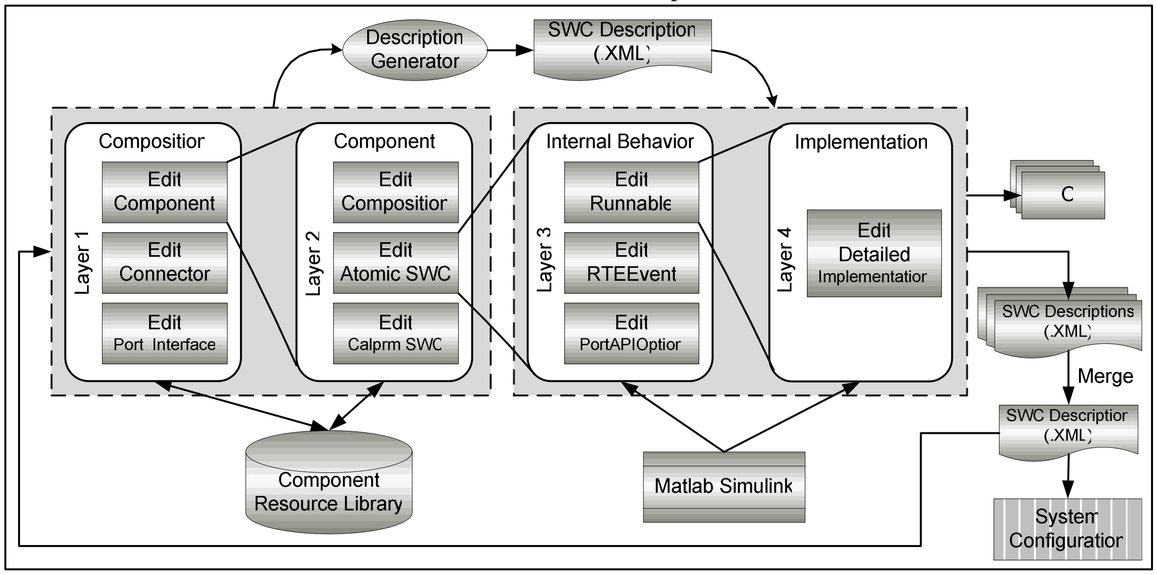


Figure 2. Process of software component modeling

A. *Software Component Modeling*

As shown in Fig. 2, software component modeling is divided into four layers. The first layer is top-level composition modeling, which presents the entire system architecture; the second layer is used to design various components; the third layer provides internal behavior model for atomic SWC and the last layer implements runnables.

We design three diagram models to support the hierarchical modeling, named composition diagram model, atomic SWC diagram model and Calprm SWC diagram model. The definitions of these three models are as follows:

1) Composition diagram model

Composition Diagram model is a six-tuple $M_{cpz} = (C, P, I, R, R_{pi}, R_{cp})$, where R_{pi} is a set of relationship between port and interface; R_{cp} is a set a relationship between component and port. Define $c_t \in C_c$ as the only composition for every composition diagram. Remark:

- $R_{pi} \subseteq P \times I$
- $R_{cp} \subseteq C \times P$, $\exists r_{cp}(c,p) \in R_{cp}$ means that the port p belongs to the component c.
- $\forall i \in I \wedge \exists r_{pi}(p,i) \in R_{pi} \rightarrow \exists r_{cp}(c_t,p) \in R_{cp}$, for each interface, if there is a port related to the interface, then this port must belong to the top-level composition.
- $\forall r_a(p_1,p_2) \in R_a \rightarrow \nexists r_{cp}(c_t,p_1) \in R_{cp} \wedge$ $\nexists r_{cp}(c_t,p_2) \in R_{cp} \wedge p_1 \in R_p \wedge p_2 \in R_r$, for each assembly connector, it cannot connect with the top-level composition.
- $\forall r_d(p_1,p_2) \in R_d \rightarrow \exists r_{cp}(c_t,p_1) \in R_{cp} \wedge$ $\nexists r_{cp}(c_t,p_2) \in R_{cp} \wedge (p_1,p_2 \in P_p \vee p_1,p_2 \in P_r)$, for each delegation connector, it represents the relationship between top level composition and components within this composition. The connector is from PPort to PPort or from RPort to RPort.
- $\forall r(p_1,p_2) \in R \wedge \exists r_1(p_1,i_1) \in R_{pi} \wedge \exists r_2(p_2,i_2) \in$ $R_{pi} \rightarrow i_1,i_2 \in I_{cs} \vee i_1,i_2 \in I_{sr} \vee i_1,i_2 \in I_c$, for each connector, if ports are related to their interfaces, then these interfaces must in the same type.

The meta-model of composition diagram is shown in Fig. 3. When building an AUTOSAR application, developer first needs to create a top-level composition container, where components can be created or imported from the existing library, and then interconnected with connectors.

2) Atomic SWC diagram model

Atomic SWC Diagram model is a five-tuple $M_a = (a_{swc}, P, I, R_{pi}, R_{cp})$, where $a_{swc} \in A_{swc}$ is the only atomic SWC of each atomic SWC diagram; R_{pi} is a set of relationships between ports and interfaces; R_{cp} is a set of relationships between components and ports:

- $R_{pi} \subseteq P \times I$
- $R_{cp} \subseteq \{a_{swc}\} \times P$

$\forall i \in I_c \wedge \exists r(p,i) \in R_{pi} \rightarrow p \in P_r$, a calibration component can only contain RPorts.

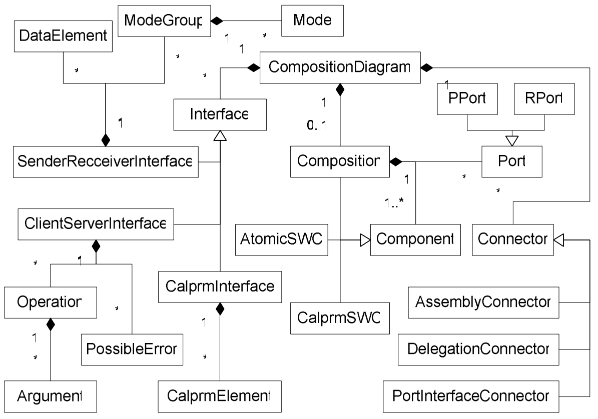


Figure 3. Meta-model of composition diagram

Fig. 4 illustrates the meta-model of atomic SWC diagram. As atomic SWC is the smallest software unit, atomic SWC diagram is the bottom-level diagram that cannot contain other diagrams. In addition, for calibration interfaces in designing atomic SWCs, only RPort is supported.

3) Calprm SWC diagram model

Calprm SWC Diagram model is a five-tuple $M_c = (c_{swc}, P, I, R_{pi}, R_{cp})$, where $c_{swc} \in C_{swc}$ is the only calibration component of each Calprm diagram; R_{pi} is a set of relationship between PPort and calibration interface; R_{cp} is a set of relationship between calibration software component and PPort. Remark:

- $R_{pi} \subseteq P \times I$
- $R_{cp} \subseteq \{c_{swc}\} \times P$
- $\forall p \in P \rightarrow p \in P_p$
- $\forall i \in I \rightarrow i \in I_c$

The calprm SWC diagram is simpler than atomic SWC diagram, which can only contain PPort. The corresponding interface must be calibration interface. The meta-model of calprm SWC diagram is shown in Fig. 5.

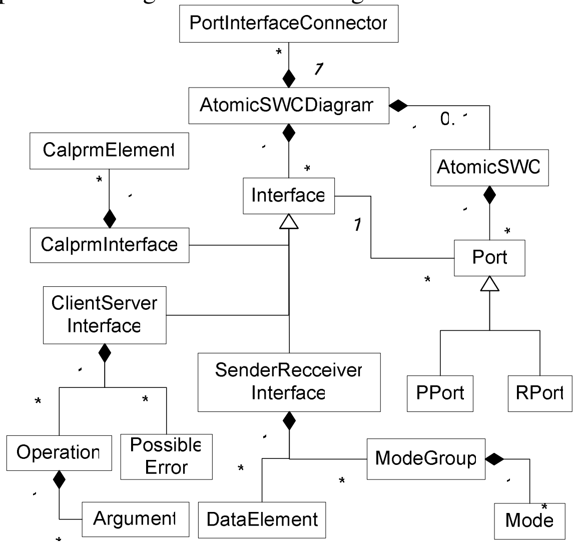


Figure 4. Meta-model of atomic SWC diagram

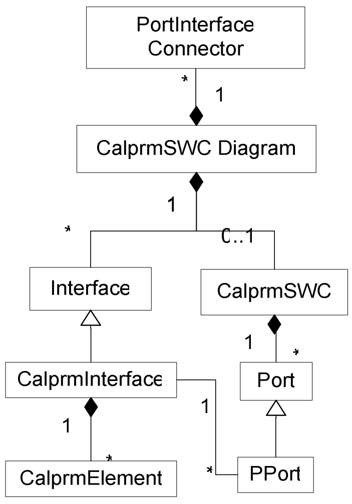


Figure 5. Meta-model of calprm SWC diagram

In the process of software component modeling, different type of model entities can be imported into component resource library (as long as the type of model exists in the library), and can be exported from the library when needed.

Component resource library is a triple $L = (C, I, D)$, for $C = C_c \cup A_{swc} \cup C_{swc}$, $I = I_{cs} \cup I_{sr} \cup I_c$ and D is a set of data types which involves two types: primitive type and composite type.

The organization of component library is shown in table I. The first column shows three types of resource in component resource library; the second column represents the resource type of each resource in column 1and the last column gives sub-resources of each type of resource. Take C-S interface as an example, it may contain operations and possible errors. The definition of resource is in nested form; therefore composition is the most complex entity in component resource library.

TABLE I. COMPONENT RESOURCE LIBRARY

Resource	Resource Type	Sub-Resource
Component	Composition	Component
		Port
		Interface
		Connector
	Atomic SWC	Port
		Interface
	Calprm SWC	PPort
		Calprm Interface
Interface	S-R Interface	Data Element
		Mode Group
	C-S Interface	Operation
		Possible Error
	Calprm Interface	Calprm Element
Data Type	Primitive Type	Char
		Boolean
		……
	Composite Type	Record
		Array

B. Description Generation

Description generator is based on the software component models designed in the former two layers and outputs XML descriptions, improving the interoperability of tools (provided by different suppliers).

In the generating process, description generator parses model entities layer by layer, and encapsulates the model information into intermediate data architecture, through which the description generator generates the final XML description. The XML description is generated according to AUTOSAR XML schema, and Fig. 6 shows part of composition schema. Corresponding to AUTOSAR meta-model, the intermediate data architecture is a directed acyclic graph (DAG). The DAG stores the complete information of application and the nodes represent the model entities while the directed edges represent aggregation relationships between related nodes.

Since Matlab/Simulink is widely used for control systems design, the description generated from the former two layers can be imported into Simulink for further development. The output of Simulink is several detailed XML descriptions and C files. These XML descriptions can be merged into one file by our description generator, and then used as the input of system configuration.

IV. CASE STUDY

In this section, we use our modeling method of AUTOSAR software component to develop a DLC (Door Light Control) system, in order to demonstrate the applicability and efficiency of our modeling approach. The DLC system controls the interior lighting in a vehicle. The DLC application is divided into six SWCs:

SWC A—Door request component
SWC B—Dimmer component
SWC C—Left door sensor component
SWC D—Right door sensor component
SWC E—Switch sensor component
SWC F—Dimmer actuator component

SWC A and B are combined as a door light control composition, which decides whether light should be lit or not according to the signals input from SWC C, D and E. Once SWC A gets the signal, it immediately notifies SWC

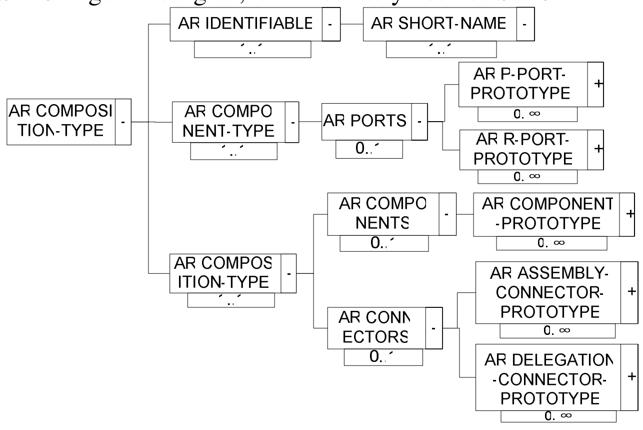


Figure 6. Composition schema

B, which then makes a decision as to whether the light should be lit.

SWC C, D and E are all sensor components, which obtain signals from the interface of the top-level composition, then transmit them to the door light control composition. However, SWC E has a higher priority than SWC C and D, which should be executed non-preemptively. SWC C and D have normal priorities.

SWC F monitors the dimmer request, which will judge the request to be aware of the light's state and change it.

Fig. 7 shows the top two layers of the DLC application. The developer can use our hierarchical modeling method to create applications layer by layer and the consistency between different layers will be guaranteed. All the signals are stored in the interfaces and transmitted through the ports of components.

The XML description is shown in Fig. 8, which is generated automatically from the model.

V. CONCLUSION AND FUTURE WORK

We have presented development and implementation of a hierarchical modeling method process for AUTOSAR software component. The objective of this modeling method is the creation of reusable and reliable software components. The proposed approach implements the AUTOSAR application layer as a four-layer development process, supporting resource reuse. The generated XML description conforms to the AUTOSAR specification and can be the input of system configuration. The example of door light control system is used to demonstrate shown the advantage of this method.

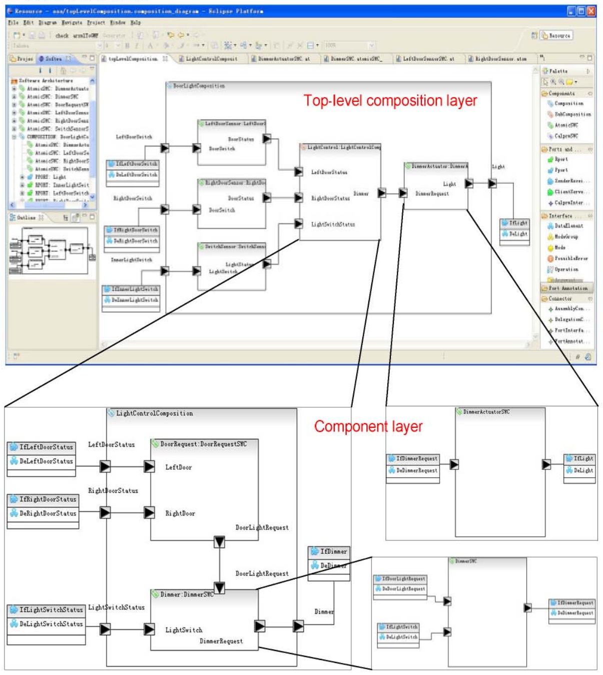
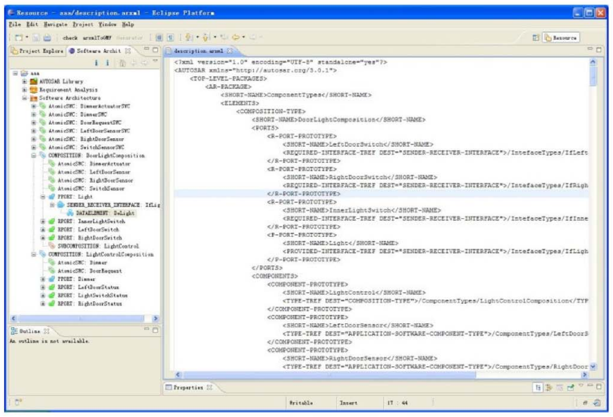


Figure 7. Door light control system



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AUTOSAR xmlns="http://autosar.org/3.0.1">
    <TOP-LEVEL-PACKAGES>
        <AR-PACKAGE>
            <SHORT-NAME>ComponentTypes</SHORT-NAME>
            <ELEMENTS>
                <COMPOSITION-TYPE>
                    <SHORT-NAME>DoorLightComposition</SHORT-NAME>
                    <PORTS>
                        <R-PORT-PROTOTYPE>
                            <SHORT-NAME>LeftDoorSwitch</SHORT-NAME>
                            <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/IntefaceTypes/IfLeft
                        </R-PORT-PROTOTYPE>
                        <R-PORT-PROTOTYPE>
                            <SHORT-NAME>RightDoorSwitch</SHORT-NAME>
                            <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/IntefaceTypes/IfRigh
                        </R-PORT-PROTOTYPE>
                        <R-PORT-PROTOTYPE>
                            <SHORT-NAME>InnerLightSwitch</SHORT-NAME>
                            <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/IntefaceTypes/IfInne
                        </R-PORT-PROTOTYPE>
                        <P-PORT-PROTOTYPE>
                            <SHORT-NAME>Light</SHORT-NAME>
                            <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/IntefaceTypes/IfLigh
                        </P-PORT-PROTOTYPE>
                    </PORTS>
                    <COMPONENTS>
                        <COMPONENT-PROTOTYPE>
                            <SHORT-NAME>LightControl</SHORT-NAME>
                            <TYPE-TREF DEST="COMPOSITION-TYPE">/ComponentTypes/LightControlComposition</TYP
                        </COMPONENT-PROTOTYPE>
                        <COMPONENT-PROTOTYPE>
                            <SHORT-NAME>LeftDoorSensor</SHORT-NAME>
                            <TYPE-TREF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE">/ComponentTypes/LeftDoorS
                        </COMPONENT-PROTOTYPE>
                        <COMPONENT-PROTOTYPE>
                            <SHORT-NAME>RightDoorSensor</SHORT-NAME>
                            <TYPE-TREF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE">/ComponentTypes/RightDoor
```

Figure 8. XML description

There are several directions for future work: (1) Set up useful software component modeling rules and constraints to guide developers to create a more complete system; (2) Keep dynamic consistency between software component models and the generated software component description; (3) Improve the method to have better compatibility with third-party developing tools.

ACKNOWLEDGMENT

This research was supported by National Important Science & Technology Specific Projects under Grant No.2009ZX01038-001 and 2009ZX01038-002 and 863 National High Technology Program under Grant No. 2007AA01Z129 and 2007AA010304 and Key Program of Science and Technology Department of Zhejiang Province under Grant No. 2008C01036-3.

REFERENCES

[1] D. Kum, G. M. Park, S. Lee and W. Jung, "AUTOSAR Migration from Existing Automotive Software," International Conference on Control, Automation and Systems (ICCAS 08), pp. 558-562, 2008.

[2] H. Heinecke, W. Damm et al, "Software Components for Reliable Automotive Systems," Proceedings of the conference on Design, Automation and Test in Europe, pp. 549-554, 2008.

[3] A. Pretschner, M. Broy, I. H. Kruger and T. Stauner, "Software Engineering for Automotive Systems: A Roadmap," Future of Software Engineering (FOSE 07), pp. 55-71, 2007.

[4] AUTOSAR Partnership, "Technical Overview V2.2.2 R3.1 0001," http://www.autosar.org/, 2009.

[5] AUTOSAR Partnership, "Specification of the Virtual Functional Bus V1.0.2 R3.1 Rev 0001," http://www.autosar.org/, 2009.

[6] AUTOSAR Partnership, "AUTOSAR Methodology V1.2.2 R3.1 Rev 0001," http://www.autosar.org/, 2009.

[7] M. Kumar, J. Yoo and S. Hong, "Enhancing AUTOSAR Methodology to a COTS-based Development Process via Mapping to V-Model," IEEE Symposium on Industrial Embedded Systems, 2009.

[8] MathWorks, "Key products support for AUTOSAR," http://www.mathworks.com/automotive/standards/autosar.html, 2008.

[9] Vector, "AUTOSAR Solution," http://www.vector.com, 2009.

[10] EB, "EB suite for AUTOSAR," http://www.elektrobit.com/, 2008.

[11] AUTOSAR Partnership, "AUTOSAR Schema Definition V3.2.0 R3.1 0002," http://www.autosar.org/, 2009.