A Process Algebraic Approach to Resource-Parameterized Timing Analysis of Automotive Software Architectures

Jin Hyun Kim, Inhye Kang, Sungwon Kang *Member IEEE*, and Abdeldjalil Boudjadar

***Abstract*—Modern automotive software components are often first developed by different suppliers and then integrated under limited resources by a manufacturer. The integration of software components under various resource configurations is prone to timing errors because 1) the components are resource-independently designed by the supplier; 2) the components are viewed by the manufacturer as black boxes during the integration stage, so that imposing resource constraints/requirements on their behavior is a challenge. This paper introduces an engineering awareness environment for the analysis of automotive systems with respect to two perspectives: 1) Time-aware design models which correspond to the supplier perspective; 2) Resource-aware design models imposed by the manufacturer during integration. To this end, first, we propose two timed behavioral models: a time-constrained model (TcM) and a resource-constrained model (RcM) that are extended from a functional model (FM). A timing analysis of applications can hence be conducted incrementally by adopting the separation of concerns principle coming from the Model-Driven Architectures (MDA). Second, given a basic application component description of AUTOSAR with timing properties, we specify how to define the behavior of the basic components as process terms using a process algebra, ACSR-VP, in order to exploit the description capability of the language for both timing aspects and resource-constrained aspects of a system. As a result, a timed behavioral model of a system can be seamlessly refined by various resource configurations and both platform-independent and platform-dependent timing properties of real-time systems can be analyzed in a consistent and efficient manner.**

***Index Terms*—Formal specification, Timing analysis, Automotive software architectures, Timing extensions, AUTOSAR, Process algebra.**

I. Introduction

NOWADAYS, safety standards and regulations for automotive systems, such as ISO 26262 [9] and IEC 61508 [15], demand the development of software/hardware components of electrics/electronics systems according to specific and well defined safety integrity levels. For safety-critical or safety-related components, a rigorous development process and formal analysis methods are required and/or recommended so that the developed system can comply with the required safety integrity degree.

Traditionally, an automotive system is developed in such a way that many components in the system are independently designed and implemented by different suppliers and then integrated by a manufacturer. Hence, it is indispensable to ensure not only consistency of interfacing and interacting of software components but also timeliness of the components. For this reason, standardized software architecture (SA), such as AUTOSAR [1] and AADL [5], has been extended with timing attributes to capture and analyze the time-constrained behavior of software systems. In particular, real-time properties of integrated components are easily violated when the components are integrated due to limited resources and the non-consideration of the deployment platform when designing the components by a supplier. In fact, because of commercial purposes a manufacturer would not disclose the detailed description of the deployment platform to the supplier.

Timing properties of concern to supplier and manufacturer can be different. The supplier is more concerned about a function-related timing property while the manufacturer is more concerned about platform-oriented property including the function-related timing property. The supplier is less concerned about the limitation of resources of an operating platform, such as CPU, memory, and network. From the perspective of manufacturer, a configuration that requires less resource is a better one, and thus manufacturers direct more efforts toward integrating as many components as possible into limited resources to reduce the system development cost. That may lead the execution of applications to delay and deviate from the desired time bound due to a priority relation in using shared resources. Therefore, there is a need of a model that can capture both perspectives of the supplier and the manufacturer as well as an analysis technique that can analyze a system model with respect to both perspectives.

The separation of concerns in Model-Driven Architecture (MDA) [4] is an essential principle that enables applications to be designed independently from a specific platform. It requires each domain-specific application to individually devise its modeling and analysis techniques appropriate for its domain. However, the separation of concerns principle for the analysis of timing properties has not drawn much attention yet.

Many approaches to timing analysis for software architectures with timing extensions, such as [10], [19], have focused on a timed behavior of components that is not concerned about platform constraints. Moreover, they do not present a formal

J. H. Kim is with the Department of Computer and Information Science, University of Pennsylvania, Philadelphia, USA, E-mail: jinhyun@seas.upenn.edu

S. Kang is with the Department of Computer Science, KAIST, Daejeon, 305-701, Korea, E-mail: sungwon.kang@kaist.ac.kr

I. Kang is with the Department of Mechanical and Information Engineering, University of Seoul, Seoul, 130-743, Korea, E-mail:inhye@uos.ac.kr

A. Boudjadar is with the Department of Computer and Information Science, Linköping University Sweden, Email: abdeldjalil.boudjadar@liu.se

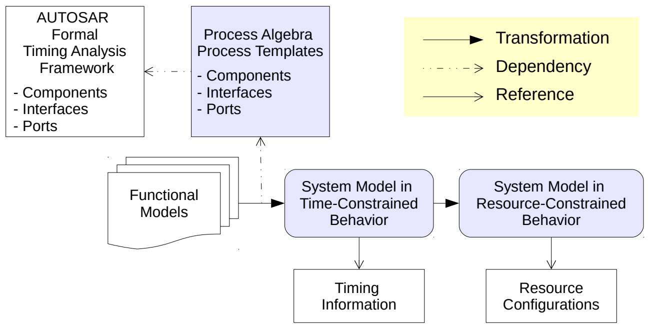

AUTOSAR Formal Timing Analysis Framework
- Components
- Interfaces
- Ports
Process Algebra Process Templates
- Components
- Interfaces
- Ports
Transformation
Dependency
Reference
Functional Models
System Model in Time-Constrained Behavior
System Model in Resource-Constrained Behavior
Timing Information
Resource Configurations

Fig. 1. Our approach to timing analysis for AUTOSAR

way to refine a platform-independent timed behavior model of applications with platform-imposed resource constraints for platform-concerned analysis. This leads to derive an efficient way of parameterizing a real-time application with resource constraints such that the timing design of software architectures can be evaluated under varying resource constraints.

This paper presents a process algebraic approach for a formal timing analysis of a software architecture (SA) design for automotive systems adopting the separation of concerns principle, according to the concerns of both supplier and manufacturer. Also, it provides an efficient way of parameterizing a timing design of AUTOSAR SA with resource constraints by exploiting the primitive of process algebra for resource constraints. In particular, we exploit the specification capability of a process algebra language, ACSR-VP [31], which encodes a resource configuration upon timing specifications of a system. Fig. 1 depicts our timing analysis approach that exploits a formal specification and verification techniques. As shown in Fig 1, a time-constrained behavior model is obtained by extending a functional model with real-time constraints, and a resource-constrained model is obtained by extending the time-constrained behavior model with resources constraints imposed by a given platform and these models are consistently and systematically transformed from one to another. The two behavioral models will be analyzed against the specified timing requirements using formal analysis tools. In our case, we use the VERSA tool which is a powerful reachability analysis tool for the specifications given in terms of ACSR-VP.

In this paper, first, we formally define a timing analysis framework for AUTOSAR software architecture (SA), where the structure and properties of individual models are presented together with a timed behavior model that can be refined by embedding the resource constraints. In this framework, a time-constrained model (TcM) and a resource-constrained model (RcM) are distinguished so that timing requirements to be implemented by suppliers can be designed and implemented independently from a specific platform. The formal definition of individual AUTOSAR SA description primitives for components, interfaces and ports adheres closely to the AUTOSAR requirements. Second, the individual description elements are defined by process terms of ACSR-VP and given in the form of macro templates such that a behavior model of an AUTOSAR SA is systemically modeled by instantiating the corresponding templates.

Mainly, the contribution of this paper includes:

- A timing analysis framework for AUTOSAR software architecture models,
- Formal specification of AUTOSAR SA description elements using process terms of process algebra,
- Two formal models for timing analysis oriented to suppliers and manufactures, respectively.

For our framework, we use ACSR-VP [17], a process algebra language for real-time systems, and VERSA [6], a tool of ACSR-VP. ACSR-VP is practical to describe both a timed behavior of systems and a resource-constrained behavior of timed systems. In particular, it supports the concept of a priority relation of current processes for shared resources, thus it is more effective to refine a timed behavior of a system with platform-given resource constraints than other formalisms. However, our methodology can be realized by any formalism that supports the concept of timed behavior and the means that can implement the concept of a priority relation.

This paper illustrates our approach by conducting a case study, where an air system of engine control units in the AUTOSAR software architecture with timing extensions is specified and analyzed with respect to both timing requirements and resource constraints.

Section II presents the necessary background of our work. In Section III, we formally define our timing analysis framework, where individual modeling elements are characterized by functionality, time and resources. In Section IV, we define the individual AUTOSAR SA descriptive elements by a process algebra and present them with macro process templates. Also, we present a way of developing two behavior models, TcM and RcM, for AUTOSAR SA models by composing the process templates. Section V is a case study where a software component of an engine control system designed according to the AUTOSAR standard, extended with time constraints, is checked against the well-known timing property, end-to-end delay. In Section VI, we discuss the related work. Finally, Section VII concludes this paper.

II. Background

This section presents the necessary background of our work: the AUTOSAR standard, the syntax and semantics of the ACSR-VP process algebra.

A. AUTOSAR: An Automotive Software Architecture

AUTOSAR [1] is a layered architectural description for automotive software systems developed jointly by manufacturers, suppliers, and tool developers. The motivation behind the introduction of such an architectural description is the decoupling of the functionality from the supporting hardware and software services. Besides, AUTOSAR provides a uniform way of integrating functional modules from multiple suppliers. It is an open standard for automotive Electrics/Electronics (E/E) architectures that will represent an infrastructure on which software application can be constructed with the assistance of a standardized SA, interfaces between components, development methodology, and software developing tools.

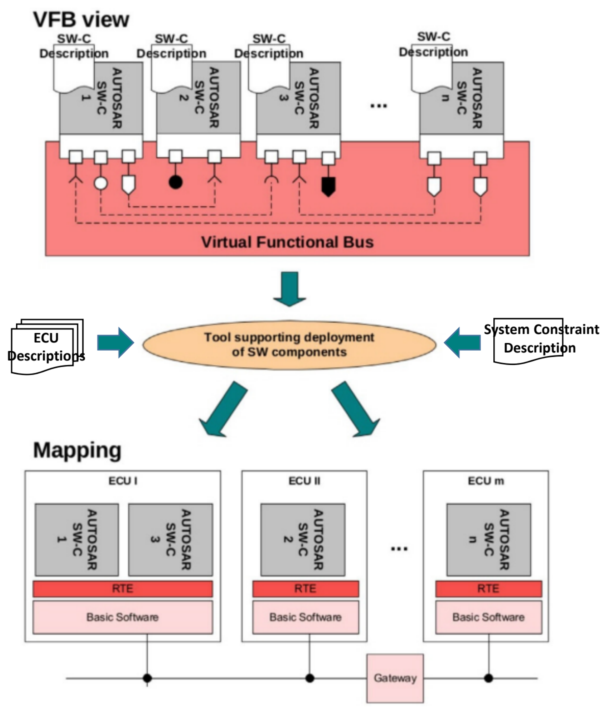


Fig. 2. AUTOSAR design methodology [8]

Fig. 2 depicts the development methodology adopted by AUTOSAR: VFB (Virtual Functional Bus) views the system as a composition of three primary description elements: **components**, various standardized **ports**, and **interfaces** [8]. It focuses on the application functionalities (**SW Component Descriptions**) to be implemented. A VFB design can be extended with the timing information in case of real-time features, while ignoring the resource constraints because the target platform is not considered at that stage.

The AUTOSAR templates provide the means, such as **runnables**, to describe the timing properties of a system's dynamics, which are determined by the consumption of computation, communication, and other hardware resources [7]. Given resource descriptions, such as ECU (Electronic Control Unit), and various system constraints, each component in a VFB design is mapped to a specific ECU and connected by a network. Each component of the VFB includes one or more runnables with the assistance of the standardized and non-standardized basic software. At this stage, the system's functionalities are designed in accordance with resource and system constraints and will be implemented to satisfy functional and non-functional requirements including the timing requirements.

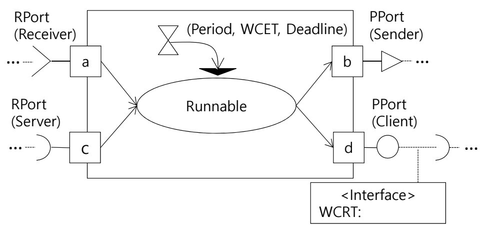


Fig. 3. Component model of AUTOSAR with timing extensions [21]

Fig. 3 shows a basic component of AUTOSAR SA and its connecting interface that are extended with timing information. Basically, the AUTOSAR SA standard models consist of components, interfaces, and ports. A component has well-defined ports, through which it communicates with other components. The component behavior is given in terms of “runnable entity (runnable)" describing its dynamics.

A port-interface (interface) associates a component with other components. It defines the contract that must be fulfilled by the port providing or requiring an interface. AUTOSAR provides six interface types, however in this paper we limit ourself to two types only: “Client-Server" and “Sender-Receiver." The Client-Server conveys the operations that can be invoked by components, whereas the Sender-Receiver interface supports the data communication.

A port of components is an interaction point that is used to read/write data and receive/send signals from/to components and the system environment. A port of a component is either a “PPort" or an “RPort", which is one of six different types, such as “Sender-Receiver, “Client-Server", “Parameter", etc. The ports must be compatible with the corresponding interfaces. A runnable entity (runnable) is defined by a sequence of instructions (operations) that can be triggered by the Run-Time Environment. For this reason, runnables are exploited for VFB timing extension [8].

The AUTOSAR requirements on timing extensions [7] require the means to describe timing properties and constraints by means of event chains. A timing property to be specified includes the time consumed for computation, communications, and use of the hardware resources.

We adopt a conventional timing extension for a component of AUTOSAR SA [21] where individual runnables of components are refined with real-time attributes, i.e. an execution time, a period, and a deadline, and the interfaces are also extended with a worst-case response time. In Fig. 3, the runnable (Runable1) is constrained to three timing properties (Period, WCET, Deadline). The interface (Interface) is constrained only by the worst-case response time (WCRT) because it is activable whenever the associated component performs an action on the underlying port and stays operational until the third part component accepts the delivered data/event. The runnable associated with a component can be constrained with a deadline.

B. ACSR-VP and VERSA

Algebra of Communicating Shared Resources with Value Passing (ACSR-VP) [31] is a formal specification language for real-time systems in which data values can be passed between communicating processes. ACSR-VP extends ACSR [22] with a dynamic priority and a value-passing mechanism as follows: 1) priorities can be evaluated through expressions associated to the individual processes, and 2) an instantaneous event is associated with a value expression to denote a communication time for the value. For example,$\{(cpu, y{+}1)\}$ is a timed action with resource cpu and priority $(y+1)$. For value passing, $(l, 3)?x$ is an event action requesting for the event l at priority 3. When the event l occurs, the process stores the received data in variable x. Similarly, $(l, y)!5$ is an output event l sending value 5 with priority y.

1) Syntax of ACSR-VP: The syntax of the ACSR-VP algebra P is as shown in Table I.

TABLE I
SYNTAX OF ACSR-VP

$$\begin{array}{ccl}
P & ::= & NIL \mid A : P \mid e.P \mid be \rightarrow P \mid P_1 + \ P_2 \mid P_1 || P_2 \\
 & \mid & \mid P \triangle^t_a (Q, R, S) \mid [P]_I \mid P \backslash F \\
 & \mid & P \backslash\backslash I \mid rexX.P \mid C(\overrightarrow{x}) \\
e & ::= & (\tau, ve) \mid (l?, ve) \mid (l!, ve) \mid (l?\overrightarrow{x}, ve) \\
 & \mid & (l!\overrightarrow{ve_1}, ve) \\
A & ::= & \{S\} \\
S & ::= & \epsilon \mid (r, ve), S
\end{array}$$

We use x for value variables, ve and ve_1 for value expressions, and be for boolean expressions. The vectors of variables expressions are denoted by $\overrightarrow{x}$ and $\overrightarrow{ve}$, respectively. Also, we use l to denote an event, and r for a resource. F and I are used for a set of events and a set of resources, respectively.

NIL represents a deadlock process, i.e. the process cannot make any progress. $A : P$ and $e.P$ are, respectively, action and event prefix terms of the process P. A conditional process term $be \rightarrow P$ is introduced to incorporate conditions on value variables. P can execute if the Boolean expression be is evaluated to be true. $P_1 + P_2$ represents a non-deterministic choice between P_1 and P_2. $P_1 \| P_2$ denotes the parallel composition of P_1 and P_2. The **Scope** operator $\triangle$ imposes certain constraints on a process. So that for term $P \triangle^t_a (Q, R, S)$, the process P is required to fire the event a before the time quantum t expires. If the event a is triggered by the process P successfully before the time t, it proceeds to the process Q. Otherwise, it is taken over by the process R. The process S in the above expression is an interrupting process so that the process P can be interrupted at any time by the execution of the process S. The **Close** operator $[P]_I$ denotes that the process P exclusively uses all the resources in I. The **Restriction** operator $P \backslash F$ deprives process P to execute events in F. However, the τ event caused by the event in F is still allowed, so that only the communication between two processes using complementing events in F is allowed. $P \ \backslash\backslash \ I$ describes the behavior of P whereby the resources having identities in I are concealed. The **Recursive** operator $rexX.P$ denotes a recursive process that repeats the behavior of P. A process constant $C(\overrightarrow{x})$ is associated with a process definition of the form $C(\overrightarrow{x}) \overset{def}{=} P$ where the process P executes exploiting the values passed by the vector of variables $\overrightarrow{x}$. We also use $(\emptyset^k : P)$ as a shorthand for the process $(\emptyset : ... : \emptyset : P)$ that repeats $\emptyset$ k times, representing an idling of a process for k time units.

2) The Operational Semantics of ACSR-VP: The semantics of ACSR-VP is defined in two steps. First, a unprioritized transition system $P \overset{\alpha}{\rightarrow} P'$ is defined. Second, the transition system is refined by $P \overset{\alpha}{\rightarrow}_\pi P'$ with priority information where preempted actions are eliminated [17]. The transition rules in Table II represent unprioritized operational semantics of ACSR-VP. The rules are identical to the transition rule of ACSR [17]. Rule **ParC2** states that two parallel processes P and Q communicate by passing a value through k. Rule **ActI4** accounts for the need to evaluate an expression for the priority and the value to be transmitted. Rule **ActI3** states that the input prefix process $(l?x, ve).P$ executes an input operation reading the values in the variable vector x. Rule **ParC2** describes synchronization between P and Q using the complementing events $l?$ and $l!$, resulting in the τ event with the priority $n + m$. Rule **Cond** states that if the condition be holds, then the process P takes action, and otherwise the process P behaves like NIL. According to Rule **Rec**, the process $C(\overrightarrow{ve})$ behaves in the same way as $P[\overrightarrow{ve}/\overrightarrow{x}]$ if there exists a process declaration $C(\overrightarrow{x}) \overset{def}{=} P$.

***Example** 2.1:* The following ACSR-VP description specifies a timer that alarms when a given time duration elapses.

$$\begin{array}{rcl}
P & \overset{def}{=} & (a?x, 1).P'(x, 0) + \emptyset : P \\
P'(y, t) & \overset{def}{=} & (t < y) \rightarrow \emptyset : P'(y, t + 1) \\
 & + & (t = y) \rightarrow (b!y, 3).P
\end{array}$$

The process P starts by receiving a time value x via synchronization on the channel a. The process $P'(x, 0)$ takes over the execution by comparing the current time t and the given time y (received as parameter in x). If t is less than y, $P'(y, t)$ is repeated with $t+1$. If t is equal to y, the event action $(b!(y), 3)$ executes by sending the value of y via channel b, and then P executes again. ■

A process term of ACSR-VP can be parameterized by a macro template and instantiated as an actual process to be executed. A macro template is defined by a starting process, a continuing process, input and output parameters. For example, a template ***PrcA*** is defined by the entering process P, the exiting process P', and two input parameters rid and pri as follows:

$$\begin{array}{ll}
\boldsymbol{PrcA} & \boldsymbol{(P, rid, pri, P')} \\
P & \overset{def}{=} \quad \{(rid, pri)\} : P' + \emptyset : P
\end{array}$$

The template ***PrcA*** is instantiable in the following way:

$$\begin{array}{ll}
PrcA & (Q, CPU, 3, Q') \\
PrcA & (R, CPU, 4, R')
\end{array}$$

which are expanded (instantiated) to be actual process terms. These two instances have identical behaviors as the following:

$$\begin{array}{rcl}
Q & \overset{def}{=} & \{(CPU, 3)\} : Q' + \emptyset : Q \\
R & \overset{def}{=} & \{(CPU, 4)\} : R' + \emptyset : Q
\end{array}$$

In the rest of this paper, the macro process template is referred to as a *process template*.

3) VERSA: VERSA (Verification Execution and Rewrite System for ACSR) is a formal verification for ACSR and ACSR-VP [6]. It checks whether or not the system reaches an undesired state that is not consistent with a verification property modeled in a process term. VERSA supports three types of analysis techniques:

1) Application of rewriting rules to ACSR specifications to deduce system properties.

TABLE II
UNPRIORITIZED TRANSITION RULES OF ACSR-VP

ActI1	$(l?, ve).P \xrightarrow{(l?,[ve])} P$	**ActI2**	$(l!, ve).P \xrightarrow{(l!,[ve])} P$
ActI3	$(l?x, ve).P \xrightarrow{(l?n,[ve])} P[n/x]$	**ActI4**	$(l!ve_1, ve_2).P \xrightarrow{(l![ve_1],[ve_2])} P$
ActI5	$(\tau, ve).P \xrightarrow{(\tau,[ve])} P$	**ActT**	$A : P \xrightarrow{A} P$
ChoiceL	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	**ChoiceR**	$\frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$
Cond	$\frac{P \xrightarrow{\alpha} P',\ [be]=true}{(be \rightarrow P) \xrightarrow{\alpha} P'}$	**Rec**	$\frac{P[\overrightarrow{k}/\overrightarrow{x}] \xrightarrow{\alpha} P'}{C(\overrightarrow{k}) \xrightarrow{\alpha} P}$, where $C(x) = P$
ParIL	$\frac{P \xrightarrow{e} P'}{P' \Vert Q \xrightarrow{e} P' \Vert Q}$	**ParIR**	$\frac{Q \xrightarrow{e} Q'}{P' \Vert Q \xrightarrow{e} P \Vert Q'}$
ParT	$\frac{P \xrightarrow{A_1} P', Q \xrightarrow{A_2} Q'}{P \Vert Q \xrightarrow{e} P' \Vert Q'}$, where $\rho(A_1) \cap \rho(A_2)$	**ParC2**	$\frac{P \xrightarrow{(l!x,m)} P',\ Q \xrightarrow{(l?k,n)} Q'}{P \Vert Q \xrightarrow{(\tau,m+n)} (P' \Vert Q'[k/x])]}$
CloseT	$\frac{P \xrightarrow{A_1} P'}{[P]_I \xrightarrow{A_1 \cup A_2} [P']_I}$, where $A_2 = \{(r, 0) \mid r \in I - \rho(A_1)\}$	**CloseI**	$\frac{P \xrightarrow{e} P'}{[P]_I \xrightarrow{e} [P]_I}$
ResT	$\frac{P \xrightarrow{A} P'}{P \backslash F \xrightarrow{A} P' \backslash F}$	**ResT**	$\frac{P \xrightarrow{e} P'}{P \backslash F \xrightarrow{e} P' \backslash F}$, where $\gamma(e) \notin F$
HideT	$\frac{P \xrightarrow{A} P'}{P \backslash\backslash I \xrightarrow{A'} P' \backslash\backslash I}$, where $A' = \{(r, p) \in A \mid r \notin I\}$	**HideI**	$\frac{P \xrightarrow{e} P'}{P \backslash\backslash I \xrightarrow{e} P' \backslash\backslash I}$

2) Construction of a state machine, automatic exploration and analysis of the state space to verify safety properties and test equivalence of alternative process formulations.
3) Interactive execution of the process specification to explore specific system behaviors and sample the execution traces of the system.

III. TIME AND RESOURCE-ORIENTED ANALYSIS

This section discusses our modeling and analysis methodology according to the perspectives of both suppliers and manufacturers. This paper aims at analyzing a timed behavior of an AUTOSAR SA description modeled from a VFB perspective [8] with timing extensions [7]. To that end, we provide a behavior semantics of AUTOSAR SA by defining its basic descriptive elements with ACSR-VP process terms. Defining the semantics relies on the informal descriptions about the four primary SA descriptive elements: a component, a port, an interface, and a runnable. The timing properties regarding the dynamic of the system are characterized by instantaneous or time resource-consuming actions made by runnables and interfaces. An AUTOSAR architecture design can be formally constructed by composing the descriptive elements in ACSR-VP process terms, and then performing the appropriate analysis for an ACSR-VP model of the AUTOSAR architecture design.

A. Modeling Methodology

In practice, many E/E components of automotive systems with the same functionality are reusable by different automotive systems. The different execution deployment operating platforms can operate the same software components with different resource configurations. As a result, the behavior of the software component can deviate from the desired one. For this reason, the functionality of a system should pass through at least two analysis phrases: at designing a functionality and porting the implemented functionality on a platform.

From the same perspective, timing properties of real-time systems can be divided into two classes: resource-independent and resource-dependent timing properties. A resource-independent timing property is a property that is to be implemented as a part of functionality. For instance, the following requirement

"The turn indicators should flash by a ON/OFF duty cycle"

includes a timing property, ON/OFF duty cycle [1], as a part of the functionality. The timing property is implemented to have a delay.

Meanwhile, a resource-dependent timing property is a timing property that should be satisfied by the implemented systems. For instance, the following requirement

"The door should be locked within 10 ms when the key commands to lock"

is an end-to-end delay timing requirement. In practice, a resource-dependent timing property cannot be analyzed prior to the integration of software components exploiting limited resources under various system constraints. Logically, execution of the current components sharing resources is subject to the delay due to the prioritized acquisition of shared resources. The analysis of a timing design ignoring resource constraints, assumes that the execution of a system is given unlimited resources.

In this paper, a resource-independent property is modeled with functional behaviors by a Time-constrained Model (TcM) while a resource-dependent property is captured by a Resource-constrained Model (RcM) that refines TcM with resource and system constraints. Compared to the Platform-Independent Model (PIM) and the Platform-Specific Model(PSM) in MDA, the focus of TcM and RcM is more on timing properties.

[1]A duty cycle is the percentage of one period that elapses for a signal to complete an on-and-off cycle.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2527624, IEEE Transactions on Industrial Informatics

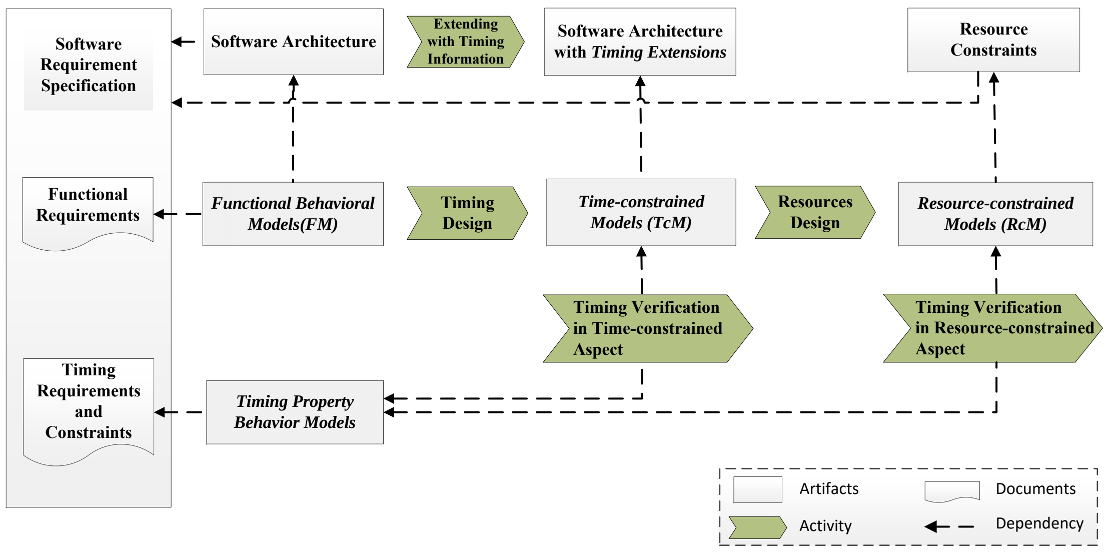

Software Requirement Specification
Software Architecture
Extending with Timing Information
Software Architecture with Timing Extensions
Resource Constraints
Functional Requirements
Functional Behavioral Models(FM)
Timing Design
Time-constrained Models (TcM)
Resources Design
Resource-constrained Models (RcM)
Timing Verification in Time-constrained Aspect
Timing Verification in Resource-constrained Aspect
Timing Requirements and Constraints
Timing Property Behavior Models
Artifacts
Documents
Activity
Dependency

Fig. 4. Our Approach to Formal Timing Verification of Automotive Software Architecture

The ultimate goal of our paper is to analyze the timing properties of a system represented by a behavioral model. Basically, our model is based on an architectural design that describes the structure of a system, in particular, the component model of AUTOSAR SA. However, it is not originally defined in a formal semantics, hence the behavior semantics of an AUTOSAR SA is formally defined by ACSR-VP process terms in a way that 1) the AUTOSAR SA descriptive elements, i.e. components, ports, interfaces, and runnables are individually defined by behavioral elements using ACSR-VP process terms, 2) the behavior of a component is composed of relevant basic elements, 3) and individual component behaviors are composed as a system behavior model. Each basic element is parameterized by a macro template with a formal definition of the element's behavior using the ACSR-VP process terms. The macro templates are instantiated with actual parameters.

B. Analysis Methodology

To develop timely correct automotive applications, we propose the following development process: first, a manufacturer requires an automotive component by providing a functional requirement specification including resource-independent timing requirements. Using AUTOSAR, such a functional requirement is given in the form of a VFB specification, which might be extended with timing requirements for resource-independent timing requirements. Given a functional requirement of a component, a supplier designs functionality and characterizes timing properties of the component. Then, their timing properties can be specified by refining runnables and interfaces of a given VFB (Virtual Functional Bus) specification with specific timing information, such as a WCET, a period, and a deadline. Third, a manufacturer checks supplier-designed behavior of applications against platform constraints before integration of the applications and a specific platform.

Our approach supports the models shown in Fig. 4 to support the development process mentioned above. This framework introduces three behavioral models: the Functional Behavioral Model (FM), the Timed-constrained Model (TcM), and the Resource-constrained Model (RcM). First, FM captures a behavior of individual components complying with functional requirements according to a given software architecture description, but it ignores timing information. FM can be used together with a VFB specification to describe a detailed functionality of components. Second, TcM extends FM w.r.t. timing information regarding functional behaviors, based on timing extensions annotated to a SA description. This timing information is independent from a platform and must be implemented as a function. TcM can be used for supplier's design that specifies a timed functional behavior of components independent from a platform. Third, RcM extends TcM w.r.t. resource constraints regarding the resource and system constraints including resource mapping tables, scheduling policies for shared resources, and communication mechanisms. RcM can be used for a manufacturer to check a designed timed behavior of suppliers against various platform constraints.

A FM and TcM can be used for PIM of applications. For example, a FM is useful for a manufacturer of the automotive system to provide suppliers with functional requirements without timing features. A TcM can be used to describe functional requirements together with timing requirements that are independent from a platform. Meanwhile, a RcM is a PSM in terms of behavior including both application and platform features. It is useful for a manufacturer to specify and analyze a configuration of platform resources.

Our timing analysis investigates TcM and RcM with respect to timing requirements and constraints represented by a timing property behavior model, which monitors if the system satisfies timing requirements and constraints.

In the following section, we formally describe our analysis methodology. First, we define the relevant modeling elements. Second, we define the composition of the modeling elements for a component and a composition of components for a system.

C. Formal Definition of SA Analysis Framework

Basically, a behavior of a system is modeled with two actions, timed and untimed (instantaneous) actions. Let A denote a processing action that consumes time and resources. Let E denote an untimed and instantaneous action which could be a pure signal or data carrying. The AUTOSAR SA is designed using the modeling elements of AUTOSAR and denoted by $< \mathcal{C}, \mathcal{P}, \mathcal{I}, \mathcal{R} >$, where:

- $\mathcal{C}$ is a set of components,
- $\mathcal{P}$ is a set of ports,
- $\mathcal{I}$ is a set of interfaces,
- $\mathcal{R}$ is a set of runnables.

Individual modeling elements are specified by a specific identity i.e. a process and actions.

A component is characterized by a set of runnables and ports. A port is given by its interfacing data and events whereas the interface is given by the associated component's ports and communicating actions.

In the case of ports, we consider only two pairs of compatible ports: RPort-Server and PPort-Client, and PPort-Sender and RPort-Receiver ports. A *port* has one of the following types:

$$\begin{aligned}\mathcal{P}_{type} =\{&server_{rport}, client_{pport}, \\ &sender_{pport}, receiver_{rport}\}\end{aligned} \tag{1}$$

A port $p \in \mathcal{P}$ is characterized by a port identity, an event carrying in/out signal and data, and its type. The set $\mathcal{P}$ of ports is defined by:

$$\mathcal{P} \subseteq \mathcal{P}_{id} \times E_{\mathcal{P}} \times \mathcal{P}_{type} \tag{2}$$

where $\mathcal{P}_{id}$ is a set of port identities, $E_{\mathcal{P}} \subseteq E$ denotes the set of events coming to the port or going out from the port, and $\mathcal{P}_{type}$ is a port type.

An *interface* $\iota \in \mathcal{I}$ is modeled by a processing action that handles communication between components. It connects an input port of a component and an output port of another component. It is regulated by a communication protocol, which is distinguished by the interface type. According to the AUTOSAR interface types, the type of an interface $\iota \in \mathcal{I}$ is:

$$\mathcal{I}_{type} = \{server\_client, sender\_receiver\} \tag{3}$$

where $server\_client$ is the type of the interface connecting $server$ and $client$ ports, and $sender\_receiver$ is the type of the interface connecting $sender$ and $receiver$ ports. The type of an interface implies the protocol that is followed by the activated interface.

An interface $\iota \in \mathcal{I}$ is characterized by an input event associated with an output port of a component, an output event associated with an input port of another component, and a timed action to process a communication. The set $\mathcal{I}$ of interfaces is defined by:

$$\mathcal{I} \subseteq \mathcal{I}_{id} \times E_{in} \times A \times E_{out} \times \mathcal{I}_{type} \tag{4}$$

where $\mathcal{I}_{id}$ is a set of interface identities, and $E_{in} \cup E_{out} \subseteq E$.

A runnable of AUTOSAR SA is an action process that executes the user-defined actions, exploiting various I/O ports to communicate with others. The behavior of a runnable run is defined by a list of processing actions with port actions, which is in the following form of:

$$run = p_1, a_1, ..., a_n, p_2, ... \tag{5}$$

where $a_1, a_n \in A$ and $p_1, p_2 \in \mathcal{P}$.

A *component* $c \in \mathcal{C}$ consists of one or more runnables and ports. The set of components $\mathcal{C}$ is defined by:

$$\mathcal{C} \subseteq 2^{\mathcal{R}} \times 2^{\mathcal{P}} \tag{6}$$

In our framework, a system model of AUTOSAR consists of a set of components, ports interfacing a component, and interfaces associating components. For timing analysis of a system model, we introduce the following three models. First, a functional behavioral model (FM) is defined by the union of a subset of components and a subset of interface instances:

$$FM = (\mathcal{C}, \mathcal{I}) \tag{7}$$

FM is a functional model that does not account for timing information.

A TcM extends a FM with only timing information ignoring resource-related information. For timing information, a timing attribute $t \in T$ is defined by a composition of a period, a WCET, and a deadline:

$$T = prd \times WCET \times dline \tag{8}$$

where prd, $WCET$, and $dline$ denote a period, the WCET, and a deadline of a runnable, respectively, such that $prd, WCET, dline \in \mathbb{N}$.

Runnable and interfaces can execute processing actions that consume time and use resources. Two constraints functions are introduced to restrict them to timing constraints. First, a constraint function to associate a runnable and timing constraints is defined by:

$$\Gamma^{\mathcal{R}} : \mathcal{R} \rightarrow T \tag{9}$$

which associates a runnable with a time attribute.

Second, a constraint function to associate an interface and timing constraints is defined by:

$$\Gamma^{\mathcal{I}} : \mathcal{I} \rightarrow \mathbb{N} \tag{10}$$

where $\mathbb{N}$ denotes the WCRT (worst-cast response time) of an interface communicating action.

A TcM is obtained by restricting process actions of component's runnables and interfaces to timing constraints. Given component's runnables and interfaces restricted by timing constraints, a TcM extends a FM such that:

$$\mathsf{TcM} = (\mathsf{FM}, \Gamma^{T}), \text{ where } \Gamma^{T} = (\Gamma^{\mathcal{R}}, \Gamma^{\mathcal{I}}) \tag{11}$$

Let R be a set of resources and $\mathcal{A}$ a set of scheduling algorithms. A RcM extends a TcM with resource constraints, such as sharing policy. A resource is used by actions of runnables and interfaces according to a scheduling (sharing) algorithm. Thus, a resource constraint is given as a relation, which is defined by:

$$\Gamma^{R} \subseteq R \times 2^{A} \times \mathcal{A} \tag{12}$$

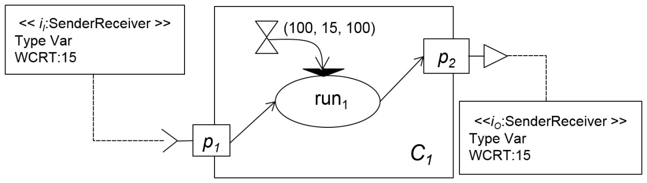


Fig. 5. Component and Interface Model with Timing Extensions

where A is a set of processing actions executed by runnables and interfaces.

Given resource constraints Γ^R, a RcM is constructed such that:

$$\mathsf{RcM} = (\mathsf{TcM}, \Gamma^R) \tag{13}$$

***Example** 3.1:* Fig. 5 shows a basic AUTOSAR component connected with two interfaces. It is extended with timed information on the runnable and interfaces. A FM for this model can be as :

$$\begin{aligned}
&\mathcal{C} = \{(\mathcal{R}, \mathcal{P})\}\\
&\mathcal{R} = \{run_1 = (p_1, exe_1, exe_2, p_2, p_3)\}\\
&\mathcal{I} = \{(\iota_i, e_{in}, e_{pin}, iexe_1, sender\_receiver),\\
&\qquad (\iota_o, e_{out}, e_{pout}, iexe_2, sender\_receiver)\}\\
&\mathcal{P} = \{(p_1, e_{pin}, reciver), (p_2, e_{pout}, sender)\}\\
&\mathsf{FM} = (\mathcal{C}, \mathcal{I})
\end{aligned}$$

For TcM obtained from the FM, the runnables and interfaces are extended with timing information as, for example:

$$\begin{aligned}
&\Gamma^{\mathcal{R}}(run_1) = (100, 15, 100),\ \Gamma^{\mathcal{I}}(\iota_i) = 15\ , \Gamma^{\mathcal{I}}(\iota_o) = 15\\
&\mathsf{TcM} = (\mathsf{FM}, \Gamma^{\mathcal{R}} \cup \Gamma^{\mathcal{I}})
\end{aligned}$$

For RcM from the TcM, a constraints on resources can be given as, for example:

$$\begin{aligned}
&\Gamma^R = \{(CPU_1, \{exe_1, exe_2\}, FP), (CAN_1, \{iexe_1, iexe_2\}, FP)\}\\
&\mathsf{RcM} = (\mathsf{TcM}, \Gamma^R)
\end{aligned}$$

In some cases, it might be necessary to give more information about resource constraints, such as priority relations. ■

TcM is analyzed against timing requirements, focusing on a resource-independent property. Meanwhile, RcM is analyzed to check if the system in RcM satisfies timing requirements of both resource-independent properties and resource-dependent properties.

IV. Formal Timed Models for Automotive Software Architecture

A formal behavioral model of AUTOSAR SA description is constructed in accordance with the formal SA given in the previous section.

A system is a parallel composition of component and interface processes in process terms. A component process is captured by a list of port and runnable action processes. In particular, a runnable process describes a sequence of processing actions, and a port process describes input and output event actions. An interface process describes communication actions, such as reading/writing, receiving/sending, and data processing actions for communicating.

TABLE III
The concept process model of AUTOSAR SA specification in ACSR-VP

$$\begin{aligned}
S &\overset{def}{=} \mathcal{C} \mid\mid \mathcal{I}\\
\mathcal{C} &\overset{def}{=} c_1 \mid\mid ... \mid\mid c_n\\
\mathcal{I} &\overset{def}{=} \iota_1 \mid\mid ... \mid\mid \iota_n\\
c_1 &\overset{def}{=} run_1 \mid\mid ... \mid\mid run_n\\
run_1 &\overset{def}{=} start\_run_1.p_{in}.a_1...a_n.p_{out}.run\_end_1\\
&...\\
\iota_1 &\overset{def}{=} intf\_read\_port_1.intf\_a_1.intf\_write\_port_1\\
&...
\end{aligned}$$

Table III shows the conceptual behavioral model of an AUTOSAR SA specification. S is the top process of the system, which is composed of a set of parallel components $\mathcal{C}$ and interfaces $\mathcal{I}$. Note that the structure of the process S is consistent with the definition of FM in Eq. 7. In the same way, the component c_1 is captured by one or more sequences of port actions (p_{in} and p_{out}) and processing actions ($a_1, ..., a_2$) according to the definition of the component in Eq. 6, the behavior of runnable run_1 according to Eq. 5, and the interface ι_1 according to Eq. 4.

In the following, we define the behavior semantics of individual descriptive elements of AUTOSAR SA using ACSR-VP process terms. The behaviors of individual descriptive elements of AUTOSAR SA are based on their informal descriptions in [8] and timing extensions of AUTOSAR SA are based on [7].

A. Processing Action Process

In designing AUTOSAR SAs for operating platforms, runnables and interfaces that execute processing actions that consume time and use resources are restricted to a limited set of resources, which might be shared according to a sharing policy.

The sharing of resources might lead to delay of their behaviors for an unexpected time. It is thus necessary to take into account sharing mechanisms for shared resources, i.e. scheduling mechanisms, when analyzing the timed behaviors of runnables and interfaces.

Using ACSR-VP process terms, we design a processing action taking into account three scheduling mechanisms: Preemptive Fixed-Priority (PFP), Non-preemptive Fixed-Priority (NFP), and Preemptive Dynamic-Priority, i.e. Earliest Deadline First (EDF).

1) Scheduled Processing Actions: Basically, a processing action is defined as a scheduled timed action of ACSR-VP controlled by a scheduling mechanism. A processing action can use and consume a resource for a certain time. It can preempt a resource or be preempted according to a scheduling policy. In ACSR-VP, a timed action is scheduled according to its priority, which is determined statically or dynamically. According to this principle, a processing action is characterized

TABLE IV
PROCESSING ACTION TEMPLATES FOR RUNNABLES

PREEMPTIVE SCHEDULED ACTION Temp1. PExRnb (P, rid, pri, tm, P')
$P \overset{def}{=} P_1(0)$ $P_1(ct) \overset{def}{=} (ct = tm) \rightarrow P'$ $\quad + \ (ct < tm) \rightarrow \{(rid, pri)\} : P_1(ct+1) + \emptyset : P_1(ct)$
NON-PREEMPTIVE SCHEDULED ACTION Temp2. NPExRnb(P, rid, pri, tm, P')
$P \overset{def}{=} P_1(0)$ $P_1(ct) \overset{def}{=} (ct = tm) \rightarrow P'$ $\quad + \ (ct < tm) \rightarrow \{(rid, pri)\} : P_1(ct+1)$
PREEMPTIVE EXECUTION by EDF Temp3. PExRnbEDF (P, rid, tm, dl, P')
$P \overset{def}{=} P_1(0,0)$ $P_1(ct,t) \overset{def}{=} (t = dl) \rightarrow NIL$ $\quad + \ (ct = tm) \rightarrow P'$ $\quad + \ (ct < tm) \rightarrow \emptyset : P_1(ct, t+1)$ $\quad + \ \{(rid, (dl_{max} - dl) + t)\} : P_1(ct+1, t+1)$

TABLE V
TEMPLATES FOR PORTS

SERVER PORT Temp4. PortSrv(P, requestchan, pri, P', P")
$P \overset{def}{=} (requestchan?, pri).P' \ + \ P''$
CLIENT PORT Temp5. PortClnt(P, listenchan, pri, P', P")
$P \overset{def}{=} (listenchan!, pri).P' + P''$
RECEIVER PORT Temp6. PortRcv(P, writechan, pri, P'(data), P")
$P \overset{def}{=} (writechan?data, pri).P'(data) \ + P''$
SENDER PORT Temp7. PortSnd(P, readchan, pri, P', P")
$P \overset{def}{=} (readchan!data, pri).P' + P''$

by three attributes: a resource identity (rid), a priority (pri), and a resource-using time (ct).

Table IV shows three scheduled processing actions defined by ACSR-VP process terms and parameterized by process templates with formal parameters regarding timing attributes. In addition to the basic parameters, the process templates add two more parameters: the ID of the entering process P of the template, and the ID of the exiting process P'. The process P is a process that begins a scheduled processing action, and the exiting process P' is a process that should be executed after the scheduled processing action finishes.

The template PExRnb defines a preemptive fixed-priority action: the process P initiates the process $P_1(0)$, and then it resets the execution time to 0. The process $P_1(ct)$ deviates according to the condition of ct. If $ct = tm$, $P_1(ct)$ executes the process P' to finish the processing. Otherwise, it selects either the timed action or the idling action according to the availability of a resource: If the resource r is available, $P_1(ct)$ performs the timed action $\{(rid, pri)\}$ for 1 time unit. Otherwise, it performs an idling action $\emptyset$. The parameter ct of P_1 is the execution time spent on consuming resources, and hence increases only when $P_1(ct)$ executes the timed action $\{(rid, pri)\}$, and then proceeds to $P_1(ct+1)$. $P_1(ct)$ proceeds to the exiting process P' when ct becomes equal to tm, i.e. $P_1(ct)$ uses the resource for the required execution time tm.

In a similar way, a non-preemptive action NPExRnb is defined so that a processing action is enforced to continue to the end as soon as it begins. The difference of NPExRnb from PExRnb is that if $ct < tm$, it executes the timed action $\{(rid, pri)\}$ without being preempted until the condition $ct = tm$ holds.

A preemptive action according to EDF can be modeled as the template PExRubEDF. In contrast with the former scheduled processing actions, it is not given a priority as its parameter because the priority is computed according to the absolute deadline (dl) and the current time (t) since the process has begun by the EDF scheduling policy.

B. AUTOSAR SA Element Processes

1) Port Processes: A process for a port is modeled in a straightforward way. In this paper, we present four types of ports: Server, Client, Sender, and Receiver. Basically, a port process interfaces between a component and an interface by transmitting signals and data.

In Table V, the port templates PortSrv and PortClnt represent Server port and Client port, respectively. The templates PortRcv and PortSnd correspond to Receiver port and Sender port,respectively. PortSrv and PortClnt use the signal channel primitives of ACSR-VP, i.e. $requestchan$ and $listenchan$, Meanwhile, PortRcv and PortSnd use the data channel primitives, i.e. $data$. The process P of all the port processes performs an input or output action.

The process P in PortSrv listens to a service request from a component via the signal channel $requestchan$, and PortClnt requests a service to other component via the signal channel $listenchan$. The port templates PortRcv and PortSnd pass $data$ through the channels $writechan$ and $readchan$ with priority pri. PortRcv reads data from other components while PortSnd transmits data to other components to an interface. In the case of PortRcv, P invokes the process $P'(data)$ to process the data after reading it from the channel $readchan$.

In case of that an input or output action of the port processes are unsuccessful, P'' can perform a special action such that the port become synchronous or asynchronous. For instance, if P'' is defined as $\emptyset : P$, the process P repeats an input or output action again after 1 time unit.

2) Interface Process: AUTOSAR requires the port interfaces to function in accordance with communication contracts [8]. In our framework, we are concerned not only with communication contracts but also with time and resource-constraints. We present two different communication interfaces

TABLE VI
TEMPLATES FOR INTERFACES

SERVER/CLIENT INTERFACE
Temp8. InfSrvClnt(P, listenchan, requestchan, rid, pri, tm, init)

$$\begin{aligned} P &\overset{def}{=} \emptyset : P + (listenchan?, pri).P_1(0) \\ P_1(ct) &\overset{def}{=} (ct = tm) \rightarrow P_2 \\ &\quad + \ (ct < tm) \rightarrow \{(rid, pri)\} : P_1(ct+1) + (init?, pri).P \\ &\quad + \ \emptyset : P(ct) \\ P_2 &\overset{def}{=} \emptyset : P_2 + (requestchan!, pri).P + (init?, pri).P \end{aligned}$$

SENDER/RECEIVER INTERFACE
Temp9. InfSndRcv(P, readchan, writechan, rid, pri, tm, init)

$$\begin{aligned} P &\overset{def}{=} \emptyset : P + (readchan?val, pri).P_1(0) \\ P_1(ct) &\overset{def}{=} (ct = tm) \rightarrow P_2 \\ &\quad + \ (ct < tm) \rightarrow \{(rid, pri)\} : P_1(ct+1) + (init?, pri).P \\ &\quad + \ \emptyset : P(ct) \\ P_2 &\overset{def}{=} \emptyset : P_2 + (writechan!val, pri).P + (init?, pri).P \end{aligned}$$

TABLE VII
TEMPLATES TO COMPOSE COMPONENT ELEMENTS

START
Temp10. StartRnb(Pstart, st, pri, P)

$$Pstart \overset{def}{=} (st!, pri).P + \emptyset : Pstart$$

STOP
Temp11. StopRnb(Pstop, done, pri)

$$Pstop \overset{def}{=} (done!, pri).NIL$$

DISPATCHER
Temp12. DPRnb(Disp, PStart, prd, dl, pri, st, done)

$$\begin{aligned} Disp &\overset{def}{=} (Start \parallel Mon)\backslash\{st, done\} \\ Start &\overset{def}{=} Pstart\triangle^{prd}_{dummy}(NIL, Start, NIL) \\ Mon &\overset{def}{=} (st?, pri).Fin\triangle^{end}_{dl}(Mon, NIL, NIL) + \emptyset.Mon \\ Fin &\overset{def}{=} \emptyset : Fin + (done?, pri).(end!, pri).NIL \end{aligned}$$

observing AUTOSAR: Server/Client and Sender/Receiver. The interface Server/Client is used for event communication and the interface Sender/Receiver for data communication.

Basically, the behavior of the interfaces is similar to that of the port processes, except that it is constrained by time and resources. An interface process begins by reading an input data/signal from a port of a source component, and then it executes a processing action to manipulate the input data/signal. It then finalizes the communication by transmitting the processed data/signal to a port of a destination component. For instance, the process P in Server/Client interface template of Table V is activated when receiving the event $listenchan$ from a Client port. After processing the event, the process P_2 sends the event $requestchan$ to a Server port which is receptive to the event.

The templates InfSrvClnt and InfSndRcv in Table VI are the process templates that define the behavior of the Server/Client and Sender/Receiver interfaces, respectively. The interface process in InfSrvClnt interacts with the Server and Client ports. It listens to a request from a Client port via the channel $listenchan$, and serves a Server port via $requestchan$. The template InfSndRcv is the same as InfSrvClnt, except that it uses valued (data-delivering) channels, $readchan$ and $writechan$, to transmit data. The interface process in InfSndRcv interacts with the Receive and Sender ports. It receives a datum from a Sender port via the channel $readchan$, and sends a datum to a Receive port via $writechan$.

For the description of the use of resource and time, the process $P_1(ct)$ in templates InfSrvClnt and InfSndRcv is designed to exploit the resource rid for ct time units.

3) Component Process: A basic component of AUTOSAR SA consists of one or more runnables and ports. In practice, a runnable is initiated by a regularly periodic event or by an irregular event. We introduce a dispatcher that dispatches a runnable periodically. In our model, the dispatcher has two functionalities: initiating the execution of a runnable and monitoring if a runnable misses the deadline.

The dispatcher cooperates with a starter and a reporter in the following way: First, the starter begins execution of a runnable process, synchronizing with a dispatcher and initiates monitoring of the runnable process at the same time; Second, the reporter notifies a dispatcher of the termination of a runnable process in order for the dispatcher to check if the execution of the runnable process has missed the deadline or not.

Table VII lists the starter (StartRnb), reporter (StopRnb), and dispatcher DPRnbl: The process P in the template StartRnb is a runnable process that is to be initiated by a dispatcher. The $Pstart$ synchronizes with a dispatcher by event st so that the dispatcher begins to monitor if the runnable process has missed the deadline. Whenever a runnable process finishes, it activates the process $Pstop$ in the template StopRnb, which notifies a dispatcher by event $done!$ to check if the execution of the runnable process misses the deadline or not.

The process template DPRnbl in Table VII is a dispatcher. It has two functionalities: initiating and monitoring. The process $Start$ in DPRnbl is used to initiate a runnable process in a way that the process $Start$ activates the process $Pstart$, which is the associated starter that waits until the period prd expires. Simultaneously, the process Mon begins by synchronization with the starter by event st and executes the process Fin, which waits for $done$. If the synchronization between a reporter and Fin is made by $done$, the process Fin fires the event end to lead Mon to repeat the waiting of the event st. Unless the process Mon receives the event end before the deadline dl elapses, a deadlock occurs with executing the NIL process.

C. System Process

The runnables, ports, interfaces, and dispatchers in ACSR-VP process terms constitute a system component model, which is obtained by binding those processes with associating events.

TABLE VIII
TEMPLATE TO CONSTITUTE A SYSTEM MODEL

SYSTEM COMPOSITION
Temp13. SysComp(SYS, Components, Interfaces, SyncEvents)
$SYS \overset{def}{=} (Components \mid\mid Interfaces) \backslash SyncEvents$

The system process SYS in Table VIII composes components and interfaces into a system process by using a set of synchronization events. That is, all the interfaces are combined with ports used by runnables in components using complementing events, e.g. $e!$ and $e?$. $SyncEvents$ is a set of complementing events that enable the synchronization between interfaces and ports.

D. Examples of TcM and RcM in Process Templates

1) TcM Construction: Listing 1 shows the TcM of Fig. 5 that is modeled using process templates. It considers only timed behaviors of runnables and interfaces without resource concerns. Thus, the time-concerning process templates, i.e. IntSndRcv, PExRnb, and DPRnb, are given execution times and exclusive resources without specific priorities.

The process template IntSndRcv in Line 2 represents the interface I of Fig. 5. The first parameter I1 denotes the process ID of the interface process, and the second parameter I is its input event from a port. The third parameter p1in is the event that goes to a port of its communicating component. In this case, the port process PortRcv(Port1, p1in, 0, RunExe1) in Line 7 will respond to the event from this interface. The fourth and fifth parameters are, respectively, a resource and time that are exploited to execute this interface process. The last parameter is the event that resets the interface process. In Line 6-7, the runnable run_1 of Fig. 5 is represented by process templates. The runnable is initiated by template StartRnb when the dispatcher template DPRnb in Line 13 sends the event startrun1 to StartRnb. The runnable expects to receive data via the template PortRcv which is connected with the interface in Line 2 via event p1in. When the process in template PortRcv receives p1in, it activates the process RunExe1 that is the first parameter of template PExRnb. Note that PExRnb specifies the execution of the runnable that consumes 15 time units using the resource RunExe1CPU. Afterwards, the process PExRnb invokes the process Port2 that is a port process of template PortSnd in Line 9, which is connected with the interface of I2 in Line 3 via the event p2out. In the end, the runnable finalizes its action using the process template StopRnb by sending the event donerun1 to the dispatcher template DPRnb so that DPRnb notices the end of the runnable's execution.

Listing 1. TcM of Fig. 5 in process templates

```
1  //  Interface Defintion
2  IntSndRcv(I1, I, p1in, 0, INFCAN1, 15, init)
3  IntSndRcv(I2, p2out, O, 0, INFCAN2, 15, init)
4
5  // Runnable Defintion
6  StartRnb(Startrun1, startrun1, 0, Port1)
7  PortRcv(Port1, p1in, 0, RunExe1)
8  PExRnb(RunExe1, RunExe1CPU, 0, 15, Port2)
9  PortSnd(Port2, p2out, 0, EndR1)
10 StopRnb(Reportrun1, donerun1, 0)
11
12 // Dispatcher for Runnable
13 DPRnb (DispRun1, run1, 100, 100, 0, startrun1, donerun1)
14
15 // System Declaration
16 #define Interfaces = I1 || I2
17 #define EventSet = {p1in, p2out}
18 #define Componets = DispRun1
19
20 // System Defintion
21 SYSTEM(SYS, SubSYS, Interfaces, EventSet)
```

Note that the parameter of the time-concerning templates for resource is filled with an exclusive resource ID, such as INFCAN1 and RunExe1CPU, which is not shared by any processes. That is, the resource-concerning process execute without resource constraints.

2) RcM Construction: A RcM is obtained only by replacing the exclusive resources of runnable and communication interface processes of TcM with actual resources. For given resources, scheduling mechanisms are also implemented. Some scheduling policies, such as the fixed-priority scheduling algorithm, might require specifying a priority relation between processes. For such a case, a priority relation can be specified by the following grammar:

$$\begin{aligned} rid \quad &:= \quad pid \mid S \mid \epsilon \\ S \quad &:= \quad pid > S \mid pid : s \mid \ :: schp \end{aligned}$$

where rid denotes a resource ID and pid denotes a process ID that shares the resource RId. “>" denotes the priority precedence between processes. ":" means that priorities of the two processes are the same. Finally, $schp$ is a scheduling policy that determines priorities of processes.

For instance, a resource constraint for the runnables $P1$, $P2$, and $P3$ regarding a resource $ECU1$ can be given as follows:

$$ECU1 = P1 > P2 > P3 :: RM$$

For the execution RunExe1 at line 8 in Listing 1, RunExe1CPU is replaced with a concrete resource ECU1, and the priority 0 is replaced with 4. The final form of the template to be RcM is as follows:

```
8 PExRnb(RunExe1, ECU1, 4, 15, P2)
```

For interfaces I1 and I2 in the TcM above, resource constraints for them can be specified as follows:

$$CAN1 = I1 > I2 :: FIFO$$

Lines 2, 3 in Listing 1 are refined to RcM by the above resource constraints, resulting in :

```
2 InfSndRcv( I1, I, p1in, 15, CAN1, 25, init)
3 InfSndRcv( I2, plour, O, 15, CAN1, 30, init)
```

E. Specification of Timing Properties

A verification property is also specified as a process model, called the timing constraint process, independently from the system behavior model of AUTOSAR SA. The process model is combined with a system behavior model and used to check if the system behavior model satisfies timing properties required by the constrained process.

1) Timing Constraint Process: An end-to-end delay d_{etoe} is a delay that elapses between two causal events In and Out and can be defined by:

$$In \rightarrow Out \leq d_{etoe}$$

which requires that for the given input In, the system generates the output Out not before d_{etoe} time units elapses. This property can be formulated as a process model.

TABLE IX
TEMPLATE FOR THE COMPOSITION OF A SYSTEM MODEL AND END-TO-END PROPERTY

END-TO-END DELAY Temp14. EndToEndDly(P, d_{etoe}, In, Out)
$P \overset{def}{=} \emptyset : P + (In?, MaxPrio).P_1 \triangle^{end}_{d_{etoe}} (P, NIL, NIL)$ $P_1 \overset{def}{=} recX.(\{\} : X + (In?, MaxPrio).X$ $+ (Out?, 1).(end!, 1).NIL)$

Table IX shows a timing property model in ACSR-VP process terms. This process checks if the output Out is generated by the time d_etoe whenever it reads the input In. Unless the output is generated on time, the process results in deadlock.

Listing 2. Composition of system model and timing property model

```
// System model
SYSTEM(SYS, SubSYS, Interfaces, EventSet)

// Timing property model
EndtoEndDly(Property, 250, ChkIN, ChkOUT)

// Environment Model
Env=IEnv || OEnv
IEnv={}:IEnv + ('I,1).('ChkIN,1).TimeDly(100,IEnv)
OEnv={}:OEnv + (O,1).('ChkOUT,1).OEnv

// Composition of Models.
ES=(SYS||Env||Property)\{ChkIN, ChOUT}
```

A timing property to be investigated is specified as a model and composed with a system model for timing analysis. An environment model of the system is necessary to be composed with the system as well. Listing 2 shows an environment model, ENV, which consists of IEnv and OEnv. The process IEnv triggers the system every 100 time units by the event I. Whenever stimulating the system, IEnv triggers the timing property model, Property, by the event ChkIN to initiate analysis by the property model. The process OEnv is waiting for outputs from the system with expecting the event O for the end-to-end analysis. If it receives the event O, then it informs the timing property model of the end of the system's reaction by triggering the event ChkOUT. Finally, the composition of a system under analysis, the timing property model, and an environment model can be like the process ES in the last line of Listing 2.

V. CASE STUDY: THE AIR SYSTEM IN ENGINE CONTROL SYSTEM

In the case study conducted in this section, we illustrate how to design and verify an AUTOSAR SA component of an engine control system [21]. Through this case study, we show the applicability of our approach to practical development.

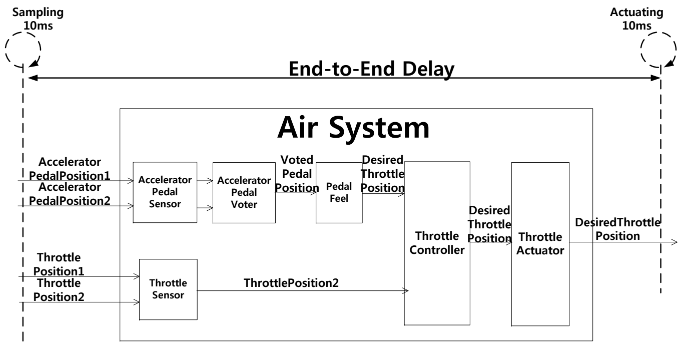


Fig. 6. Overview of air system [21]

The system we analyze is the air system, a component of an engine control system conforming to the AUTOSAR architecture. Fig. 6 shows the data flow description of the air system to compute a desired throttle position according to the current throttle position and the accelerator pedal position. The timing requirements of the system include the following:

- The sampling rate of inputs AcceleratorPedalPosition1 and AcceleratorPedalPosition2 is 10 ms.
- The sampling rate of inputs ThrottlePosition1 and ThrottlePosition2 is 10 ms.
- The actuating rate of the outputs DesiredThrottlePosition for the 4 inputs is 10 ms.

Among those timing requirements, we focus on the property that "the output of the air system should be released within 10 ms since an input is fed to the system."

A. *Air System in AUTOSAR SA with Timing Extensions*

Fig 7 shows the air system that is designed in an AUTOSAR component model with timing extensions. The runnables are extended with the period, the worst-case execution time, and the deadline. For instance, the runnable APedSensorRunnable is extended with 5 ms (millisecond), 0.2 ms, and 0,5 ms, which correspond, respectively, to the period, the worst-case execution time, and the deadline of the component. The interfaces are also extended with the worst-case response times. Based on the annotations on the AUTOSAR SA designs, we obtain a timed model of the air system for the timing analysis.

B. *TcM of Air System*

To obtain a TcM, a FM of the AUTOSAR is first constructed for the air system. Then a TcM is constructed as shown in Listing 3. In the TcM, it is shown that the runnable APedSensor is designed with a series of runnable processes together with port processes and interface processes. Notice that each process of runnables in the part of *System Composition* is assigned to an exclusive resource that is not shared with others, and that makes the model designed independently from resource constraints.

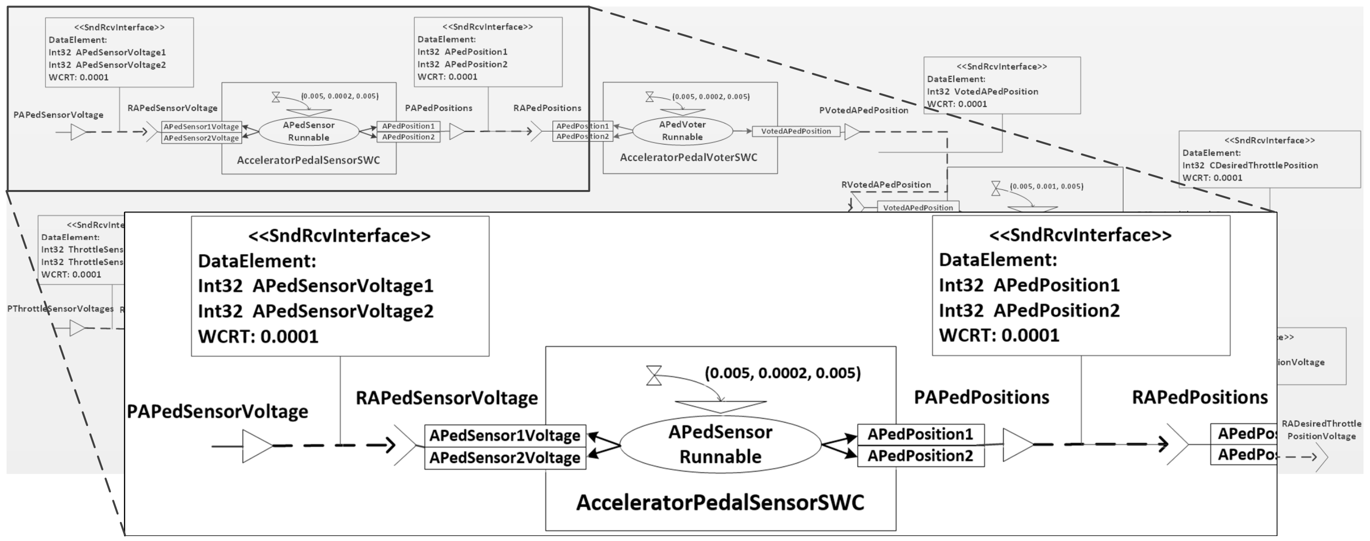


Fig. 7. Air System in AUTOSAR with Timing Extensions

Listing 3. TcM of the air system
```
#define PESet{RAPSenrVolt1_2, PAPedPos1_2, RAPedPos1_2
      ,...}
#define IEvtSet{PAPSenrVolt1_2, PThrSenrVolt1_2, RADThrPo
      ,...}
#define PREvtSet{ChkIN, ChkOUT}

//Interface  : APSenrVolt
InfSndRcv2(APSenrVolt, PAPSenrVolt1_2, RAPSenrVolt1_2,
              IRscAPSenrVolt, IPriAPSenrVolt, 1,
       InitAPSenrVolt)
..
//Interface  : APedPos9
InfSndRcv2(APedPos, PAPedPos1_2, RAPedPos1_2, IRscAPedPos,
        IPriAPedPos, 1, InitAPedPos)
...
StartRnb(RnbAPSenr, stRnbAPSenr, RPriAPSenr,
       RPortRAPSenrVolt1_2)
PortRcv2(RPortRAPSenrVolt1_2, RAPSenrVolt1_2, }
RnbAPSenrExe1, EndRnbAPSenr)
PERnb(RPriAPSenr, RnbAPSenrExe1, RRscAPSenr, 2, 50,
       PPortPAPedPos)
PortSnd2(PPortPAPedPos, PAPedPos1\_2, RPriAPSenr,
       EndRnbAPSenr)
StopRnb(EndRnbAPSenr, spRnbAPSenr, RPriAPSenr)
DPRnb(DPRnbAPSenr, RnbAPSenr, 50, 50, RPriAPSenr,
       stRnbAPSenr, spRnbAPSenr)
...
//  System Composition
InfSys1 =  [APSenrVolt]\{IRscAPSenrVolt}
        ||  [APedPos]\{IRscAPedPos}
        ||  [VtdAPedPos]\{IRscVtdAPedPos}
        ||  [CDThrPos]\{IRscCDThrPos}
...
SubSys1 =  [DPRnbAPSenr]\{RRscAPSenr}
    ||  [DPRnbAPVot]\{RRscAPVot}
        ||  [DPRnbThrCnt]\{RRscThrCnt}
        ||  [DPRnbThrAct]\{RRscThrAct}
        ||  [DPRnbThrSenr]\{RRscThrSenr}
InfSys = InfSys1
SubSys = SubSys1
Sys = InfSys || SubSys
SysComp(S, SubSys, InfSys, PESet)
```

C. RcM of Air System

An RcM is constructed by assigning shared resources and priorities to runnables and interfaces in TcM. In Listing 4, the RcM has two more descriptions, *Priority Assignments for Runnables*, and *System Composition*. In *Priority Assignments for Runnables*, different priorities are given to runnables and interfaces. In *System Composition*, different resources are also given to them. For instance, ECU1 is assigned to APedSensor (DPRnbAPSenr) and ThrottleSensor (DPRnbThrSenr). ECU2 to APVoter (DPRnbAPVot), ThrottleController (DPRnbThrCnt), and ThrottleActuator (DPRnbThrAct). Individual runnables and interfaces are given different priorities for shared resources.

Listing 4. RcM of the air system
```
//  Priority Assignments for Runnables
#define      RPriAPSenr     10
#define      RPriThrSenr     9
#define      RPriAPVot       7
#define      RPriThrCnt      5
#define      RPriThrAct      3
//  Priority Assignments for Interfaces
#define      IPriAPSenrVolt      20
#define      IPriThrSenrVolt     15
#define      IPriThrPos          13
#define      IPriAPedPos         11
#define      IPriVtdAPedPos       9
#define      IPriCDThrPos         7
#define      IPriADThrPos         5
...
//     System Composition
InfSys1 = [APSenrVolt || ThrSenrVolt || ThrPos || APedPos
      ]\{CAN1}
InfSys2 = [VtdAPedPos || CDThrPos || ADThrPos ]\{CAN2}
SubSys1 = [DPRnbAPSenr || DPRnbThrSenr  ]\{ECU1}
SubSys2 = [DPRnbAPVot || DPRnbThrCnt || DPRnbThrAct]\{ECU2
      }
InfSys = InfSys1 || InfSys2
SubSys = SubSys1 || SubSys2
Sys = InfSys || SubSys
SysComp(S, SubSys, InfSys, PESet)
```

D. Timing Verification

A timing verification of the air system was performed to check if the system satisfies the *end-to-end* property. For variation of the analysis, with the same execution time, the settings of the system were varied with in terms of the following:

- End-to-end delay,
- Periods and deadlines of runnables,
- Resource configuration, such as assignment of resource to runnables and interfaces, and scheduling policy for resources.

The TcM is checked to see if timed behaviors of runnables with periods and deadlines satisfy the end-to-end delay, which depends on the runnable's real-time attributes, e.g. execution period, deadline and worst-execution time. That is, the runnable attributes, i.e. periods and deadlines, should be feasible w.r.t. timing requirements.

1) Verification Result 1: The results of checking the TcM of the air system against timing requirements varying periods and deadlines are shown in Table X, in which the setting of runnable real-time attributes is satisfiable for the system to return the output within 25 ms, but cannot generate the output within 20 ms.

TABLE X
TIMING VERIFICATION RESULTS OF TCM (TIME UNIT: MS)

Runnable	Period	WCET	Deadline
APedSensor	5	0.2	5
APedVoter	5	0.3	5
ThrottleSensor	5	0.5	5
ThrottleController	5	1	5
ThrottleActuator	10	1	10

Timing Constraints		Verification Results	
End-to-End	PAPSenrVolt1_1 → RADThrPos	$\leq$ 25ms	OK
		$\leq$ 20 ms	Not OK

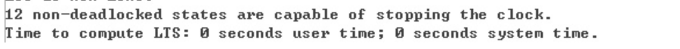
2) Verification Result 2: The RcM is checked to see if resource configurations for priority relations and scheduling policies are feasible for the system to satisfy timing requirements. After the setting of TcM is proved feasible satisfying timing requirements, the corresponding RcM is checked to see if the system on such resource constraints satisfies the same timing requirements. With the same runnable's attributes in Table X, we varied resource configurations as given in Table XI. As a result, we could verify that Resource Configuration I has a shorter delay than Resource Configuration II.

3) Verification Result 3 (Scheduling Policy): The same system was checked with the same resource setting of runnables except scheduling policies, i.e. EDF, FIFO, or Round-Robin. However, variation of the scheduling policy do not enhance the end-to-end delay because all the runnables are in a close causal relation.

4) Experiment Environment and Verification Time: The experiment environment was set up as follows:

- Intel CPU Core i7-3520 2.90 GHz,
- Memory 8 GB,
- Windows 64 bit.

The average verification time for each model was less than 2 minutes. The following screen shot shows the size of the system in verification and the time to compute a property given for the verification.



```
State machine contains 3503 reachable states (0 deadlocked), 5061 edges.
Edges represent 3232 timed transitions,
                1817 internal actions, and
                12 external untimed actions.
LTS is non-Zeno.
12 non-deadlocked states are capable of stopping the clock.
Time to compute LTS: 0 seconds user time; 0 seconds system time.
```

VI. RELATED WORK

Related work is discussed in three perspectives: A) general timing analysis methods, B) timing analysis of automotive systems, and C) timing design and analysis for software architecture of automotive systems.

TABLE XI
TIMING VERIFICATION RESULT OF RCM ($\times 10^3$ SEC)

Runnable	Period	WCET	Deadline
APedSensor	5	0.2	5
APedVoter	5	0.2	5
ThrottleSensor	5	0.5	5
ThrottleController	5	1	5
ThrottleActuator	10	1	10

Resource Constraints I			
ECU1 ::= RRscAPSenr > RRscThrSenr:: RM			
ECU2 ::= RRscAPVot > RRscThrCnt > RRscThrAct::RM			
CAN1::= IPriAPSenrVolt > IPriThrSenrVolt > IPriThrPos > IPriAPedPos:: FP			
CAN2::= IPriVtdPedPos > IPriCDThrPos> IPriADThrPos :: FP			
Timing Constraints		**Verification Results**	
End-to-End	PAPSenrVolt1_1 → RADThrPos	$\leq$ 21.4 ms	OK
		$\leq$ 20 ms	Not OK

Resource Constraints II			
ECU1 ::= RRscAPSenr > RRscThrSenr > RRscAPVot > RRscThrCnt > RRscThrAct :: **EDF/Round-Robin/FIFO**			
CAN1::= IPriAPSenrVolt > IPriThrSenrVolt > IPriThrPos > IPriAPedPos > IPriVtdPedPos > IPriCDThrPos> IPriADThrPos :: **FP**			
Timing Constraints		**Verification Results**	
End-to-End	PAPSenrVolt1_1 → RADThrPos	$\leq$25 ms	OK
		$\leq$ 20 ms	Not OK

A. Formalism for Timing Analysis

For last several decades, numerous analytical methods, such as [13], [29], [30], [32], have been introduced to investigate real-time properties of the systems, such as schedulability and response-time of the system. However, as real-time systems become more complicated and heterogeneous, model-based approaches are drawing much attention. Timed Automata (TA) [11] is one of the most well-known formalisms, which is supported by the UPPAAL[14], a model checker for TA. Recently, the authors in [16] have applied TA and its variations to a complicated scheduling system to verify various timing properties by using UPPAAL symbolic and statistical model checking techniques. The literature [18] presents a formal framework where feature requirements of each component are parameterized by timing requirements so as to facilitate the separation of timing constraints from functional decompositions.

The salient difference between our work and the previous works is in the way the notion of resource constraints is related with timing requirements. In the previous studies, a resource is not the primary concern in designing a system, hence it is required to implement resource-sharing mechanisms to account for resource constraints that may cause the state-explosion problem. ACSR-VP views a resource as the primary reason that causes the delay of job's execution and provides a primitive for resource so that there is no need to implement any scheduling function to schedule jobs for shared resources. Using ACSR-VP, we parameterize a timed system with resource constraints in the primitives of ACSR-VP for resources.

B. Timing Analysis of Automotive Systems

For timing analysis of automotive E/E systems, simulation is a widely-used method. In [27], Krause et al. provided a simulation method to analyze timing properties of an AUTOSAR model by transforming the model to the corresponding SystemC model. In [33], Monot et al. dealt with the problem of sequencing and grouping runnables on limited ECUs for a multi-core setting of the system. It views a resource constraint in the same angle as ours but focuses more on finding a feasible group of runnables for a limited set of ECUs rather that timing analysis. Anssi et al. [12] studied schedulability analysis and task scheduling in automotive ECUs in the context of AUTOSAR. They modeled a cruise control system in the same point of view, and checked the schedulability of the system using MAST [3].

Scheickl [34] presented a comprehensive overview of automotive software design in terms of timing design. Feiertag et al. [20] presented a timing analysis framework for end-to-end delay. They highlighted different path semantics of end-to-end delays that must be carefully dealt with during timing modeling and analysis. Lakshmanan et al. [28] also studied an analytical way of calculating an end-to-end delay for networked AUTOSAR-compliant systems from the viewpoint of networking.

Compared to the previous work, our method is more flexible so that any scheduling mechanism for shared resources can be formulated. It is more rigorous so that our analysis techniques guarantees the verified properties of a system with 100% certainty such that the result obtained by our analysis methods can satisfy a high software integrity level.

C. Model-based Timing Analysis of Software Architecture

In principle, our method is one of realizations of MDA [4], in which software systems is driven by a platform-dependent architecture and a platform-independent architecture to improve application's re-usability and applicability to various platforms. However, our method focuses more on timing properties such that they can be analyzed from both PIM and PSM perspectives.

Sokolsky et al. in [35] presented a formal specification and verification framework that is similar to our approach: AADL, an architecture description language for embedded systems, is used to describe the SA [5], and ACSR and VERSA are used for formal specification and formal verification, respectively.

For timing extension of automotive SAs, Timing models (TIMMO) developed Timing Augmented Descriptive Language(TADL), named a timing modeling language. TADL extends EAST-ADL2 [2] and AUTOSAR [10] with timing information. This method augments the structural concepts with information related to timing and events referring to structural elements. Our AUTOSAR description adopts the timing extension of AUTOSAR SA description by augmenting of timing information. However, we have created a way of formally specifying the behavior of individual descriptive elements of AUTOSAR components and analyzing the behavior of applications in terms of platform-constraints. So that our method is more useful for a high integrity software system.

The most recent relevant work is the work of Kim el al. in [23], [24]. In this work, a cyber-physical system is modeled adopting MDA. In particular, a PSM in their work is captured by two layers, an application layer and a platform layer, which are distinguished by Input/Output and Monitor/Control variables individually. In this work, the computation and communication time of applications depending on a platform are abstracted by a delay which is physically measured. Distinguished from [23], [24], a platform constraint in our work is imposed upon application's behavior in the form of a specification, instead of an independent model. So that application's behavior can be updated to account for specific platform constraints just by specifying resource constraints upon time and resource actions of our model.

The work in [36] by Voss et.al is also similar with this work. The authors provided a framework where a suitable software architecture is estimated by computing task and message schedules that are optimized with respect to a global discrete time base. However, our work is more specific for AUTOSAR by providing process templates of descriptive elements of the AUTOSAR SA.

In [21], Frey also studied timing extensions of AUTOSAR and timing analysis. His approach covers the overall process of design, timing extensions, and analysis of AUTOSAR SA. In [25], Klobedanz et al. presented a case study on timing modeling and analysis for software systems in AUTOSAR SA description. In [19], Paul et al. introduced a timing analysis tool, called ViTal (A Verification Tool for EAST-ADL), using UPPAAL Port for EAST-ADL. They provided a model checking technique for EAST-ADL descriptions w.r.t. timing and functional behavioral requirements.

Compared with Frey's work [21], basically, we adopt his timing extension method for AUTOSAR SA. However, our work covers a wider spectrum of artifacts from functional requirement to an architecture description extended with timing and resource information. Moreover, we pursue a formal framework where a proof for safety of a given system can be created by formal verification techniques. Klobedanz's work [25] also focused on timing analysis in terms of schedulability, similarly with this work. His work analyzes the usage of resources and the latency of processing. The main difference between our work and Klobedanz's is that our approach sticks to the principle of MDA. In Klobedanz's work, resources, such as ECU and FlexRay, are assigned to tasks in advance and are checked in terms of schedulablity. Afterwards, tasks divided into resources are modeled to AUTOSAR. On the contrary, in our approach, a SA is first built, and then it is extended with timing attributes. After going through timing verification for the timing design on timing-extended SA, the SA is restricted by resource constraints that are determined by the platform. Thus, our platform-independent SA model is more applicable. Analogous to Paul's work [19], our approach deals with timing properties of functionality of real-time systems. However, our approach focuses on a timed behavior of the system varying resource constraints to evaluate the composability of application with a given platform [26].

VII. Conclusions

Recently, numerous automotive software components have been partially developed by independent suppliers, and then integrated into a single system by a manufacturer. Such software components are highly vulnerable to timing errors. For this reason, the standardized automotive SAs, such as AUTOSAR and AADL, have been extended with timing aspects. As a consequence, a timing analysis for the automotive SA should be performed from the viewpoints of both the supplier and the manufacturer, i.e. resource-independent and dependent viewpoints. During the timing analysis stage, software architects need to be able to identify appropriate values for the timing attributes such as periods, deadline, priorities of execution units, and optimal resource configurations for various settings of automotive SAs. To deal with these issues, this paper introduces the following contributions:

1) Based on the separation of concerns principle in MDA, we proposed two timed models, (time-constrained) TcM and (resource-constrained) RcM, that represent the supplier and manufacturer's concerns, respectively.
2) We develop an efficient way to obtain TcM and RcM for each modeling element of the AUTOSAR components using our formal description templates. Moreover, we propose a systematic way to compose the description elements to form one system.
3) TcM helps to find feasible timing attributes of automotive SA, such as periods, deadlines, and priorities of processing units.
4) RcM help in identifying feasible resource configurations including scheduling policies that satisfy the timing requirements.

Furthermore, our formal verification framework can be used to provide formal safety proofs of automotive software systems that are required by the international safety standards, such as ISO 26262.

Acknowledgment

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2015-H8501-15-1015) supervised by the IITP(Institute for Information & communications Technology Promotion) and by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2013R1A1A2060586).

References

[1] AUTOSAR: Technical Overview. http://www.autosar.org.
[2] EAST-ADL. http://www.east-adl.info/index.html.
[3] MAST: Modeling and Analysis Suite for Real-Time Applications. http://mast.unican.es/.
[4] OMG Model Driven Architecture. http://www.omg.org/mda.
[5] SAE International Architecture Analysis & Design Language (AADL). http://www.aadl.info/aadl/currentsite/.
[6] VERSA: Verification Execution and Rewrite System for ACSR. http://www.cis.upenn.edu/~lee/duncan/versa.html.
[7] AUTOSAR: Requirements on Timing Extensions. http://www.autosar.org, 2009.
[8] AUTOSAR: Virtual Functional Bus. http://www.autosar.org, 2009.
[9] ISO/DIS 26262-1-Road vehicles-Functional safety-Part 1 Glossary. Technical report, Geneva, Switzerland, July 2009.
[10] TIMMO(TIMing MOdel). http://www.timmo-2-use.org, 2009.
[11] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
[12] S. Anssi, S. Tucci-Piergiovanni, S. Kuntz, S. Gerard, and F. Terrier. Enabling scheduling analysis for AUTOSAR systems. In *Proceedings of ISORC 2011*, pages 152 –159, March 2011.
[13] N. Audsley. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8:284–292(8), September 1993.
[14] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.
[15] R. Bell. Introduction and Revision of IEC 61508. In C. Dale and T. Anderson, editors, *SSS*, pages 273–291. Springer, 2011.
[16] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikucionis, U. Nyman, and A. Skou. Widening the schedulability of hierarchical scheduling systems. In *FACS 2014, Bertinoro, Italy, September 10-12, 2014, Revised Selected Papers*, pages 209–227, 2014.
[17] J.-Y. Choi, I. Lee, and H.-L. Xie. The specification and schedulability analysis of real-time systems using acsr. In *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*, pages 266–275, Dec 1995.
[18] M. Dixit, S. Ramesh, and P. Dasgupta. Time-budgeting: a component based development methodology for real-time embedded systems. *Formal Aspects of Computing*, 26(3):591–621, 2014.
[19] E. P. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson. ViTAL : A verification tool for EAST-ADL models using UPPAAL PORT. In *Proceedings of the 17th IEEE International Conference on Engineering of Complex Computer Systems*, July 2012.
[20] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Proceedings of the 1st Workshop on CRTS*, November 2008.
[21] P. C. Frey. *A Timing Model for Real-time Control-systems and Its Application on Simulation and Monitoring of AUTOSAR Systems*. PhD thesis, University ULM, 2010.
[22] R. Gerber and I. Lee. CCSR: A calculus for communicating shared resources. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1990.
[23] B. Kim, L. Feng, L. T. X. Phan, O. Sokolsky, and I. Lee. Platform-specific timing verification framework in model-based implementation. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 235–240, San Jose, CA, USA, 2015. EDA Consortium.
[24] B. Kim, H. Hwang, T. Park, S. Son, and I. Lee. A layered approach for testing timing in the model-based implementation. In *DATE '14*, pages 1–4, March 2014.
[25] K. Klobedanz, C. Kuznik, A. Thuy, and W. Müller. Timing modeling and analysis for AUTOSAR-based software development: A case study. In *Proceedings of DATE '10*, pages 642–645. European Design and Automation Association, 2010.
[26] H. Kopetz and R. Obermaisser. Temporal composability [real-time embedded systems]. *Computing Control Engineering Journal*, 13(4):156–162, Aug 2002.
[27] M. Krause, O. Bringmann, A. Hergenhan, G. Tabanoglu, and W. Rosentiel. Timing simulation of interconnected autosar software-components. In *DATE '07*, pages 1 –6, April 2007.
[28] K. Lakshmanan, G. Bhatia, and R. Rajkumar. Integrated end-to-end timing analysis of networked autosar-compliant systems. In *DATE '10*, pages 331 –334, March 2010.
[29] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171, Dec 1989.
[30] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237 – 250, 1982.
[31] S. Lim and J. Choi. Specification and verification of real-time systems using ACSR-VP. In *4th International Workshop on RTCSA '97*, pages 135–142. IEEE Computer Society, 1997.
[32] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.
[33] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. Multisource software on multicore automotive ecus-combining runnable sequencing with task scheduling. *Industrial Electronics, IEEE Transactions on*, 59(10):3934–3942, Oct 2012.
[34] O. Scheickl and M. Rudorfer. Automotive real time development using a timing-augmented AUTOSAR specification. In *Proceedings of ERTS2008*, 2008.
[35] O. Sokolsky, I. Lee, and D. Clarke. Schedulability Analysis of AADL models. In *Proceedings of the 20th international conference on Parallel and Distributed Processing*, pages 179–179, Washington, DC, USA, 2006. IEEE Computer Society.

[36] S. Voss and B. Schatz. Deployment and scheduling synthesis for mixed-critical shared-memory applications. In *Engineering of Computer Based Systems (ECBS), 2013 20th IEEE International Conference and Workshops on the*, pages 100–109, April 2013.

Jin Hyun Kim is a post- doctoral researcher in Department of Computer and Information System in University of Pennsylvania, US. He received MS and Ph.D from Korea University in 2011. From 2012 to 2013, he was a post-doctoral researcher in Korea Advanced Institute of Science and Technology (KAIST). From 2012 to 2015, he was a post- doctoral researcher in Aalborg University, Denmark, and INIRA/IRISA, France. His research interests include design and analysis of Cyber-Physical Systems, real-time system model checking, process algebra, software product line engineering, and security-aware scheduling systems.

Inhye Kang is a professor at University of Seoul. She received her Ph.D. from University of Pennsylvania in 1997. Prior to joining University of Seoul in 2002, she was a senior researcher at Samsung SECUI. Her research interests include formal methods, software engineering, and security.

Sungwon Kang received his BA from Seoul National University, Korea in 1982, and received his MS and PhD in computer science from the University of Iowa, USA in 1989 and 1992, respectively. From 1993, he was a principal researcher of Korea Telecom R & D Group until October 2001, when he joined Korea Advanced Institute of Science and Technology (KAIST). During 2003-2014, he was an adjunct faculty member of Carnegie-Mellon University for the Master of Software Engineering Program. He served as a co-chair of 1997 IWTCS and 2001 Formal Techniques for Networked and Distributed Systems, a program co-chair of 2011 SERA, and a program track chair of the 2014-2016 ACM Symposia on Applied Computing/Software Architecture: Theory, Technology, and Applications. He served as the editor of Korean Journal of Software Engineering Society during 2012-2014. His current research areas are software architecture, software product line, software testing and data-based software engineering.

Abdeldjalil Boudjadar (Jalil) is a postdoctoral researcher in Computer Science at Linköping University, Sweden. His research interests include model-based design, formal specification and analysis of embedded real-time systems. Jalil has obtained his PhD degree from Toulouse University, France.