2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)

Design Verification and Validation for Reliable Safety-critical Autonomous Control Systems

Rongjie Yan*†, Junjie Yang‡§, Di Zhu‡§ and Kai Huang✉‡§
* State Key Laboratory of Computer Science, Chinese Academic of Sciences, Beijing, China
† University of Chinese Academy of Sciences, Beijing, China
‡Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education, China
§School of Data and Computer Science,Sun Yat-sen University, Guangzhou, China
Email: yrj@ios.ac.cn, {{yangjj27, zhud5}@mail2,huangk36@mail}.sysu.edu.cn

***Abstract*—Providing guarantees on the system behavior is mandatory for safety-critical autonomous vehicles. Among these guarantees, proving the fulfillment of real-time constraints and reliability requirements on the system is a key issue, as their violation could result in unexpected and unsafe behaviors. The violation may come from the complicated interaction between software and hardware modules, or transient hardware faults. AUTOSAR, the most popular industrial standard in the automotive domain, provides an open standardized architecture for software development, where an application can be deployed on multiple electronic control units (ECUs). We present a verification and validation method for the design of such safety-critical autonomous control systems that could tolerate transient faults. The embedded implementation of an AUTOSAR model is transformed into a three-layer system model in timed automata, so that system behavior can be evaluated and checked with hard real-time constraints and the implementing architecture. We demonstrate the feasibility of the method with a simplified controller developed for the autonomous vehicles.**

***Index Terms*—Verification, validation, reliablility, safety-critical, real-time**

I. Introduction

Advanced control techniques for autonomous vehicles flourish. To have the techniques be used on actual vehicles in interaction with their environment, we must be able to guarantee their safe and reliable behavior. The complexity of implementing the controllers lies in dealing with the frequent environment changing and scenario switching, and the interaction between various subsystems and hardware modules under hard real-time constraints, which are always error-prone [1]. It is important to guarantee safety and reliability of such systems such that all the constraints can be met in every possible situation.

Best practices towards real-time analysis have relied on model-based approaches. Traditional model-based approaches allow modelling and simulation of applications, as well as generation of qualifiable code to be compiled and run on various real-time operating systems (ROTSs). Such ROTSs are based on event-triggered or time-triggered approach [2] for the execution of applications, and the deterministic behavior of an application may not be ensured before its execution. How to validate and guarantee safety of such systems becomes a great challenge for the designers [1]. Formal methods are recommended to design and analyze such systems [3]. Derived from OSEK/VDX [4], one of the most widely adopted automobile RTOS standard, AUTOSAR [5] can provide much benefits for automotive real-time development.

For these kinds of on-board control applications, any transient hardware fault may incur catastrophes, even if functionality for these control applications is well-established. And system-level mechanisms are mandatory to mitigate the impact of transient faults to ensure system reliability. We consider active redundancy [6] as the major technique to guarantee system reliability, which replicates software tasks into multiple copies. Redundancy, however, comes with costs, e.g., time and hardware facility for execution, and the mechanism to handle redundancy within the given architecture, which increases the design complexity of such systems.

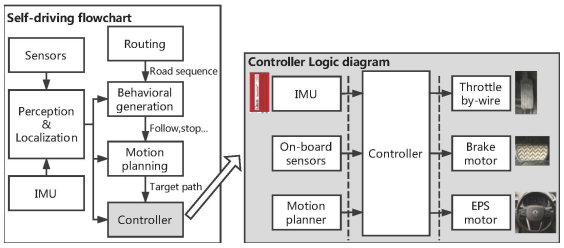


Fig. 1. Functionality of an autonomous controller

To motivate our work, let us consider an autonomous automobile controller shown in Fig. 1, whose role is to extract the target path derived from motion planning according to collected information from physical devices, and to control low level actuators to track the path. The frequency of path tracking is usually higher to follow environment updating. Meanwhile, the frequencies of every functionality in different scenarios, such as going straight and making U-turns, are also different. Since, for example, making a U-turn is generally harder than going straight, a higher frequency is required to minimize the tracking error. The tasks of the controller can be deployed on multiple Electronic Control Units (ECUs). We

This work has been partly funded by the National Key Basic Research (973) Program of China under Grant No. 2014CB340701, Key Research Program of Frontier Sciences, CAS, under Grant No. QYZDJ-SSW-JSC036, the CAS-INRIA major project under No.GJHZ1844, the National Science Foundation of China under Grant No. U1435220, No. U1711265, and the Fundamental Research Funds for the Central Universities under grant No.17lgjc40.

978-1-5386-9341-4/18/$31.00 ©2018 IEEE
DOI 10.1109/ICECCS2018.2018.00026

assume that the clocks of the ECUs are identical. And the ECUs are connected by Ethernet. The data transfers among tasks are via asynchronous buffers that: 1) a new arrival data will overwrite an existing one in the buffer, and 2) a data item will remain in the buffer till being overwritten. Its embedded implementation should meet safety and reliability guarantees and all timing constraints in all the scenarios.

To this purpose, apart from the functionality verification of the application, it is indeed necessary to have a model of the controller running on the system, to check that temporal relations are maintained, and real-time constraints will be ensured, depending on the scheduling policy of the system, and on the tasks properties (periods, deadlines, priorities). In practice, we observe that the violation of system requirements may come from unforeseen interactions between software and hardware modules. To emphasize our effort on investigating software and hardware interactions and providing system level guarantees, we assume that functionality of an application itself is correct.

The challenges of verification and validation for such systems are multi-fold. First, the software system itself is complex, e.g. event-based and preemptive behaviors are non-deterministic and hard to predict. Second, it is hard to evaluate the whole implementation with all the timing constraints. Third, introducing fault tolerant based redundancy incurs additional costs and accelerates state space explosion. We need a feasible and efficient solution to overcome the challenges.

Facing the challenges, we propose to analyze the AUTOSAR model in the embedded implementation, and transform it into a system model formalized with the network of timed automata (TA), with stopwatch semantics to support the modeling of preemptive event-based behavior. The redundancy can be instantiated by the tasks of the application model, and the synchronization between the replicas of same tasks can be implemented via broadcast or shared variables. To analyze the implementation, we adopt a three-layered system model which considers computation and communication, and the involved architecture information. State space explosion mentioned in the third challenge then can be relieved by using over-approximation or statistical model checking based techniques of TA and the associated tools.

The contributions of the paper are summarized as follows. First, we propose a framework for design verification and validation of safety-critical autonomous control systems. The framework is supported by transforming an AUTOSAR model into a system model in the network of timed automata, which encapsulates both temporal behaviors and timing constraints in its embedded implementation. By using this model, we can apply existing verification and validation techniques to analyze the AUTOSAR application. Second, the framework also supports reliability guarantee by tolerating transient hardware faults with minimized cost of active redundancy for software modules. Third, the experimental results from online measurement and model-based analysis of the case study demonstrate the feasibility of the framework.

The organization of paper is as follows. Sect. II discusses the related work. Sect. III concretizes the case study. We present necessary concepts in Sect. IV. Sect. V provides the description of the models. Sect. VI proposes the transformation from an AUTOSAR model to a network of timed automata. Sect. VII provides the experimental results on the case study, and Sect. VIII concludes.

II. Related work

As mentioned in [7], the development of AUTOSAR software components could follow model-based design methods. The tools such as Matlab/Simulink and Embedded Coder[1] can support for automotive software development. Dassault System AUTOSAR Builder[2] is a software platform for system and ECU (Electronic Control Unit) design, based on the ARTOP tool environment [8]. dSPACE[3] provides solutions for developing ECU software and mechatronic controls. AUTOSAR tool chain[4] of Vector provides support for model-based development, AUTOSAR basic software, and runtime environment configuration.

In the academic community, based on UML, Macher et al. propose a model-based software development and integrated toolchains for software architecture design [9]. The Discrete-Event System specification can be used for modeling and subsequent performance evaluation of AUTOSAR-based systems in [10]. And a simulation model is constructed for AUTOSAR-based electronic control units connected by a communication bus. The Timing Augmented Description Language (TADL) is applied for timing modeling and analysis in [11], with a case study of developing a speed-adaptive steer-by-wire system. The work in [12] presents basic features of an analyzable model of AUTOSAR required by scheduling analysis. Zhao et al. provide design optimization for AUTOSAR models with preemption thresholds and mixed-criticality scheduling [13]. The two works mainly focus on end-to-end timing analysis.

Applications in automotives are usually safety critical. To analyze safety-critical systems with strict timing constraints, the work in [14] adopts timed automata [15] in modelling applications and general system models with preemptive behaviors, such that key properties can be analyzed with UPPAAL [16]. A transformation from AUTOSAR software architecture and timing constraints to the network of timed automata is proposed in [17]. Both of them do no consider bus communication and the coordination with resource platform. Rajeev et al. propose to apply calendar automata for system modelling and end-to-end latency estimation of distributed ECU networks [18]. Different communication semantics are considered in end-to-end timing analysis for automotive embedded systems in [19].

To precisely capture system behavior in the embedded implementation of automotives, our system model covers computation and communication, and involves both temporal relations and timing constraints, as well as the sychronization

[1]https://www.mathworks.com/solutions/automotive/standards/autosar.html
[2]https://www.3ds.com/products-services/catia/products/autosarbuilder/
[3]http://dspace.de
[4]https://vector.com/vi_autosar_solutions_en.html

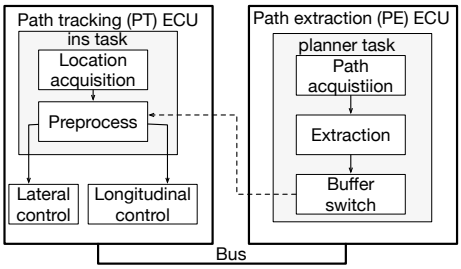


Fig. 2. An implementation of the controller

with redundancy to tolerate transient faults, which are the major distinctions from the aforementioned works.

On the other hand, the works considering fault tolerance mainly devote to drawing feasible scheduling strategies with certain optimization objectives. For the example in automotive systems, the work in [20] analyzes transient errors for automotive safety-critical applications. Though the framework proposed in [21] enables validation of timing and reliability, it concentrates on quantitative properties. We focus on the design of safe and reliable autonomous control systems by tolerating transient hardware faults.

III. Overview of the case study

We present the simplified structure of the controller in Fig. 2, whose role is to receive a target path and control low level actuators to track the path. It consists of two components: 1) Path tracking (PT), to track the path according to the input from IMU (Inertial Measurement Unit); 2) Path extraction (PE), to process the target path calculated from a motion planning module.

PT consists of location acquisition, preprocessing and control instruction output. Once data from IMU is available, tracking error will be calculated in the preprocessing step. This step also considers the output from the buffer switch task in PE when it is available. Next, lateral control and longitudinal control run in parallel. In the former, steering angle is calculated by a structure with both feedforward controller and feedback controller, and supplemented by yaw damping. The latter includes two PID controllers for throttle and brake, respectively.

PE involves path acquisition, extraction and buffer switch. Once a path is acquired in the first step, it will be delivered to the extraction step. The extraction step mainly targets for spline interpolation, radius calculation and other relevant computations. After the extraction, the buffer will be updated.

Difference in sampling rates makes the periods of PT and PE different. And the periods of PT (or PE) are different in various scenarios, such as making a U-turn, or going straight, though the worst case execution time (WCET) of every task in all scenarios is the same.

The two components are relatively independent, as PT does not empty the buffer shared with PE, and can still progress by reading the stale data from the buffer.

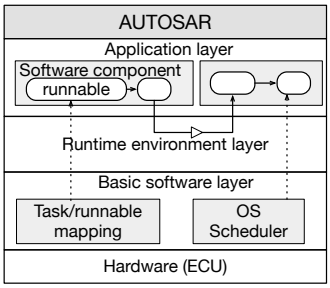


Fig. 3. The AUTOSAR layered architecture

IV. Preliminaries

In this section, we review the architecture of AUTOSAR, and shortly recap the concept of timed automata.

A. AUTOSAR architecture

AUTOSAR [5] provides a layered architecture consisting of the application layer (AP), the runtime environment (RTE), the basic software layer (BSW) and the hardware layer (HW), as shown in Fig. 3. Software components (SWCs) at the top layer realizes the behavior of the overall developed application. The behavior of an SWC is realized by *RunnableEntities* (abbreviated: runnable) that are associated with implementations, which can exist in form of C functions. The RTE handles communication between different components of the application layer (e.g., various SWCs), and between the application layer and the BSW layer (e.g., for accessing HW). The BSW layer provides services for accessing the HW as well as the operating system (OS) functionalities, where task-runnable mapping (TRM) describes the execution order of runnables. An OS scheduler at runtime handles the execution of each task according to its configuration.

The AUTOSAR application model we consider in the paper consists of

- SWC: encapsulates the functionality of the application;
- Event chain: temporal relation between events;
- Period: time interval between the occurrence of two consecutive events;
- Runnable: belonging to an atomic software component.

Considering the case study shown in Fig. 2, the model contains two SWCs, seven runnables. Event chains are maintained by the temporal relation between the runnables. And periods come from the periods of path tracking and path extraction elements. The three runnables in PE are mapped to one task, the first two runnables in PT are grouped into one tasks, and the other runnables keep unchanged.

B. Timed automata with stopwatch

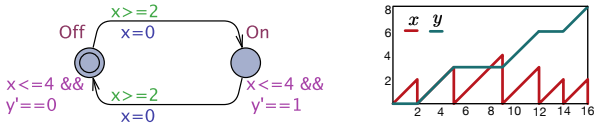
It is essential to encode and validate timing constraints when the application is integrated into the hardware platform. Consequently, we employ timed automata (TA) [15] and stopwatch automata (SWA) [22] for system modelling.

Timed automation is a classical formal model for the design of real-time systems. It consists of:

- a finite set of clocks to encode the elapse of time, and each of which is evaluated to a real number and can be reset.
- a finite set of locations to describe the local status, and each of which can be labeled with an invariant over clocks to restrict the time spent in the location.
- a finite set of transitions between pairs of locations to depict status switch between various locations, each of which can be labeled with a guard over clocks, a synchronization channel to coordinate with the corresponding transition of other automata, and an update of clocks.

In timed automata with stopwatch, clocks can be assigned to rate 0 or 1 in a location. When a clock is assigned to 0, it freezes and keeps the time it stops. When the clock is assigned to 1, it will resume from the last frozen point.

The model in Fig. 4 (a) is a timed automaton using a clock x to enforce the switch between two locations `Off` and `On`, where `Off` is the initial location. A stopwatch y which is running only in location `on` is introduced to measure the accumulated time in `On`. Therefore, the values of y keep unchanged in location `Off`, as shown in Fig. 4 (b).



(a) timed automaton with stopwatch (b) simulation

Fig. 4. A timed automaton with stopwatch

A system model can be encoded by a network of timed automata, coordinated by synchronization channels among the transitions.

V. System Description

A. Application model

As data transfer between tasks is implemented via asynchronous register buffers, the application model is denoted by a tuple $\mathcal{K} = \langle \Omega, N, P \rangle$, where $\Omega = \{\gamma_1, \ldots, \gamma_n\}$ is the set of runnable processes, $N \subseteq \Omega \times \Omega$ shows the connection between various runnables, and $P = \{p_1, \ldots, p_n\}$ is the set of periods of all the runnables. For a pair of runnables with $\gamma_i \rightarrow \gamma_j \in N$, γ_i is a predecessor of γ_j, i.e., $\gamma_i \in pre(\gamma_j)$, and γ_j is a successor of γ_i with $\gamma_j \in succ(\gamma_i)$.

The abstraction emphasizes on the temporal relation maintained by data transfer, and ignores the implementation details of the application model which can be verified before system integration.

B. Model unfolding with redundancy

In the paper, we consider that tolerating transient faults is implemented by the redundancy of tasks in an application model. We expect to calculate the minimal redundancy for reliability guarantee.

Given a system reliability requirement $\mathcal{R}$, and the application model $\mathcal{K} = \langle \Omega, N, P \rangle$ with the reliabilities of individual runnables $\{\rho_i \mid 1 \leq i \leq |\Omega|\}$ (each of which depends on the worst case execution time and the reliability of the allocated processor [23]), we could apply the greedy algorithm to calculate the degree of redundancy r for each runnable according to Equation 1, such that the system reliability can be satisfied with the minimal number of replicas.

$$\begin{aligned} &minimize(\textstyle\sum_{i=1}^{|\Omega|} r_i) \\ &\text{subject to:} \\ &\qquad \textstyle\prod_{i=1}^{|\Omega|}(1-(1-\rho_i)^{r_i}) \geq \mathcal{R} \end{aligned} \tag{1}$$

In the existence of redundancy of runnables, additional voting facilities and connections between various runnables will also be introduced. We should unfold the original with the assigned redundancy. Informally speaking, if the number of replicas of a runnable is greater than one, we need to set the connection between the replicas and the predecessors of the runnable. Moreover, it is necessary to introduce one voter, create new connections from the replicas to the voter, and add connections from the voter to the successors of the runnable. The procedure can be described by Algorithm 1, where r_k is the number of replicas for runnable γ_k, and the introduced v_k is the voter for γ_k.

Algorithm 1 unfolding($\mathcal{K}, R$)

Input: application model $\mathcal{K} = \langle \Omega, N, P \rangle$ with n runnables;
  the set of assigned number of replicas $R = \{r_1, \ldots, r_n\}$
Output: unfolded application model $\mathcal{K}$
$Q_0 = \{\gamma \in \Omega \mid pre(\gamma) = \emptyset\}$
$i = 0$
while $Q_i \neq \emptyset$ **do**
  $Q_{i+1} = \{succ(\gamma) \mid \gamma \in Q_i \wedge succ(\gamma) \notin \bigcup_{j=0}^{i} Q_j\}$
  $i++$
$depth = i$
for $(j = depth - 1; j \geq 0; j--)$ **do**
  while $Q_j \neq \emptyset$ **do**
    $\gamma_k = pop(Q_j)$
    if $r_k > 1$ **then**
      $\Omega = \Omega \cup \{v_k\}$
      $p_{v_k} = p_k$
      $P = P \cup \{p_{v_k}\}$
      for $(\forall \gamma' \in succ(\gamma_k))$ **do**
        $N = N \cup \{v_k \rightarrow \gamma'\}$
      for $(i = 0; i < r_k; i++)$ **do**
        $\Omega = \Omega \cup \{\gamma_{ki}\}$
        $N = N \cup \{\gamma_{ki} \rightarrow v_k\}$
        $p_{\gamma_{ki}} = p_k$
        $P = P \cup \{p_{\gamma_{ki}}\}$
        for $(\forall \gamma' \in pre(\gamma_k))$ **do**
          $N = N \cup \{\gamma' \rightarrow \gamma_{ki}\}$
      $\Omega = \Omega \setminus \gamma_k$
return $\mathcal{K}$;

The unfolding in Algorithm 1 is backward. Initially, it divides runnables into various sets according to the levels of dependency (Lines 4-8). Then the exploration and unfolding start from the set with the maximal depth (Line 10). In the second iteration, a runnable is first popped from Q_j (Line 12). If the runnable does not contain any replicas, its dependency keeps unchanged. Otherwise, for a runnable with replicas (Line 13), the algorithm first creates a voter for the replicas (Lines 14-16), and sets up the connection from the voter to its successors (Line 18). Then the algorithm updates the runnable network by adding the replicas and their connections to the voter (Lines 20-23). Third, the algorithm creates the connections from the predecessors to the replicas of the process (Line 25). Finally, the runnable together with outdated connections are removed from the network (Line 26). The algorithm terminates when all the runnables have been explored and the corresponding replicas and voters are added into the application model. It is correct, for the preservation of data dependency relation between various runnables.

C. System model

To allow verification and validation for automotive applications, the model should include all temporal aspects having an impact on system safety constraints and all architectural aspects with an impact on system timing constraints. With these consideration, application load, application timing behavior, resource platform and allocation, communication behaviors between interacting components (tasks or messages) are essential ingredients of the system model, where communication modeling are necessary for inappropriate communication may violate temporal constraints, incur deadlock that the system cannot progress, or result in long latency.

Consequently, the application model will be extended and integrated to the system model. First, the set of runnables in the application model is assigned to a set of tasks T, where a task can contain several runnables with temporal relations. A task t is represented as $\langle w, \delta, p, bc, wc, d\rangle$, where w is the priority of the task, δ is the initial offset, p is the period, bc and wc are the best-case and worst-case execution times, respectively, and $d \leq p$ is the deadline. Second, the connection between various tasks is instantiated by the writing and reading messages for the communication between tasks and buffers. Therefore, a message is encoded as $\langle t^s, t^d, \sigma, \mu\rangle$, where $t^s, t^d \in T$ are respectively the source and the sink of the communication, σ marks the type of communication, i.e., either read or write, and μ records the volume of data transfer.

We present the structure of the system model in Fig. 5. It consists of tasks, schedulers, and buses/processors, corresponding to application, hardware dependent software (HdS), and hardware layers of an embedded implementation, respectively. The first and the third layers are always fixed for given applications and hardware platforms. And the second layer can be customized according to different mapping and scheduling strategies, to evaluate system property with certain constraints. The motivation of this classification is to separate computation, communication, mapping and scheduling, and

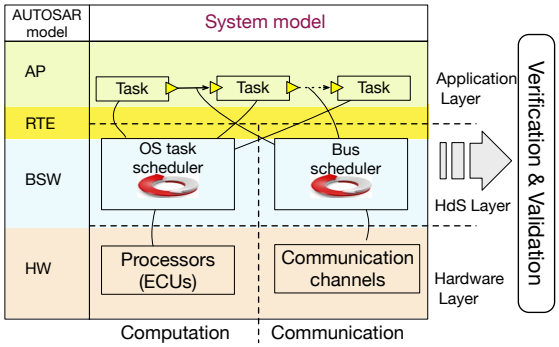


Fig. 5. The three-layer system model

explicitly describe how they are coordinated, following the motivation of AUTOSAR architecture.

Together with these notations, a system model is formally given as a tuple $\mathcal{S} = \langle \mathcal{O}, R, alloc, \dagger_t, \dagger_m\rangle$, where

- $\mathcal{O} = T \cup M$ can be tasks or messages.
- $R = E \cup B$ is the set of resources, which can be either ECUs in E or buses in B.
- $alloc : \{T \rightarrow E\} \cup \{M \rightarrow B\}$ shows the mapping relation between tasks (messages) to ECUs (buses).
- $\dagger_t$ is the scheduling policy of tasks.
- $\dagger_m$ is the scheduling policy of messages.

VI. System Model Building

Following the structure and the definition of the system model, we present the behavior and cooperation of the ingredients in timed automata. Without loss of generality, we consider a preemptive implementation of the system model with periodical tasks. In the system model, the runnables and the subchains of the AUTOSAR are considered as the subtasks in the system model, ECUs are regarded as processors.

A. Constructing system model

In the construction of the system model, the timed automata for tasks and messages are constructed separately, for the distinction between computation and communication. The connection between tasks and buffers are encoded by the synchronization between read/write requests. Architecture modelling is translated into timed automata and shared variables. And mapping is encoded in the attributes of tasks and messages.

1) Task transformation: As the connection of components via ports in application layer does not alter the temporal and timing constraints, we do not distinguish them in the transformation. The models of runnables, their connections and runnable-to-task mappings are encapsulated into task model. A basic task in AUTOSAR architecture consists of three locations, i.e., `ready`, `running` and `suspended`, where `suspended` marks the termination of the task, `ready` shows the activation of the task, and `running` shows that the task is running. With this representation, the status of being preempted is in location `ready`, which cannot be distinguished from the history of `suspended` or `running`. Meanwhile, the AUTOSAR software architecture makes the

hardware transparent, and data exchange is not visible in the application layer. However, to verify the correctness of the system model and validate its performance, we have to encode the counterparts in the task component.

Given the mapping information and the resource platform, the best/worst case execution time, the frequency of a task can be decided (We assume that the deadline and the period of a task are the same). We add additional locations to encapsulate data exchange process. When a task is ready, the access to hardware resources follows the specified scheduling policy. Consequently, a task first sends its request to a scheduler and waits for the permission. Once it acquires the permission, it could fetch the required data and start execution. When the execution terminates, it could write the computed data to the buffer and release the occupation of the processor, and repeat the same procedure in the next iteration. During the reading/writing or executing step, the allocated processor may be preempted by other tasks with a higher priority.

The timed automaton for task modelling in the system model is depicted in Fig. 6. We introduce a clock t_p to record periodical time elapse, and a stopwatch t_e to calculate the time of execution. Function *isReady(tid)* is to check whether its predecessors have finished execution and data transfer. Function *isSched(tid)* is to check whether the process acquires the processor. Before and after execution, it updates the status of the task. During the execution, the occupied processor may be preempted by other tasks, and the task has to wait until it is resumed.

2) Message transformation: The dependency of two tasks in an application model is implemented via writing and reading messages, which are respectively responsible for reading/writing data from/to buffer. However, the operation in hardware architecture is not atomic. The communication between two tasks is concretized into three elements and two steps in the system model: a read channel, a buffer, and a write channel, and two kinds of communication with the buffer.

Informally speaking, a communication consists of two stages: setup and data transfer. For the first stage, it requires the participation of a processor. However, the second stage may not require computation resources, with the help of hardware facilitaties, e.g., DMA (Direct Memory Access). We assume that the processor that accommodates the task takes charge of the setup and data transfer of a read/write action. The cost of data transfer depends on the hardware architecture and the mapping. The model in Fig. 7 shows the behavior of a `read` channel, where tid and tjd are the identities of the source and the target, respectively, parameter 0 in $isSched\_comm$ marks the read message, and $comm$ records the reading cost. It is invoked by $read$ action of some task tid, and sends the request to the bus scheduler for communication. During data transfer, the operation may be interrupted by the preemption of the processor from other tasks. Once the operation is over, the channel sends the acknowledgement back to the owner task.

The model of a `write` channel is similar except for the names of synchronization and the parameters.

3) Scheduler modelling: We present two schedulers, i.e., for tasks and messages, respectively.

Task scheduler. We apply a global scheduler for the scheduling of tasks allocated to various ECUs. As the mapping is static, task migration is not allowed even for ECUs with dual-cores.

The task scheduler can support various event triggered (ET) scheduling policies. It employs a queue for every processor in an ECU to receive requests and manage the scheduling sequence of the allocated tasks. The head of the queue can be granted to the usage of the processor. The scheduler shown in Fig. 8 accepts the request from some task, checks the status of the allocated processor, and inserts the task in the queue according to certain scheduling policies. When the status of the queue changes, the scheduler may suspend or resume some task. Time cannot elapse during the scheduling step. Currently, the scheduler provides the following strategies for selection:

- the first-in-first-out (FIFO) policy of ET, the queue is ordered according to the sequence of requests. The new request will be appended to the end of the queue, and preemption is not allowed.
- the fixed priority (FP) policy of ET, the queue is ordered according to the descending order of priorities. The new request will be inserted into the queue according to its priority.
- the earliest deadline first (EDF) policy of ET, the queue is ordered in the ascending order of the difference between the deadline of tasks and the moment when the new request comes. The new request will be inserted into the position according to its difference between the deadline and the elapsed time in the current period.

Message scheduler. Similar to the task scheduler, the message scheduler accepts the requests from messages and checks the availability of the corresponding hardware resources. It also employs a queue per hardware resource to record the requests. To ease the modelling, the distinction between inter and intra ECU communication is encoded by the cost of communication.

4) Hardware resource modelling: Since every processor in an ECU contains a queue to record the tasks being executed or ready (preempted), we could employ the size and the content of the queue to represent the status of a processor. That is, if the size of the queue is zero, the processor is idle. If it contains some tasks, the first in the queue acquires the processor, and others are ready(preempted) and waiting to be scheduled.

Similarly, a buffer could also be modelled with two shared variables, i.e., one for the status of the buffer, and the other for the size of the buffer. The similar idea can be applied for the bus model.

B. Model instantiation

To guarantee that all replicas of a task are executed in the same period, we need extra facilities (a timed automaton for a set of replicas from the same task to record the duration from the first trigger of a replica to the end execution of all its replicas) to encode the constraint.

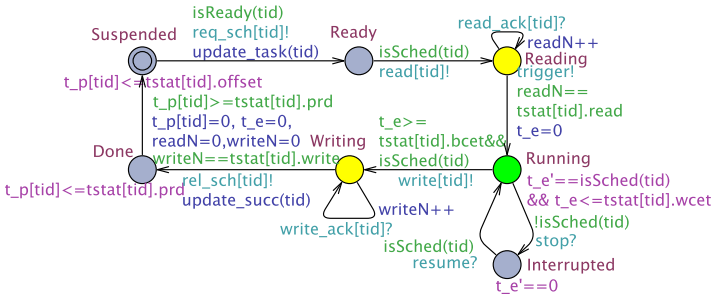


Fig. 6. The process model with stopwatch

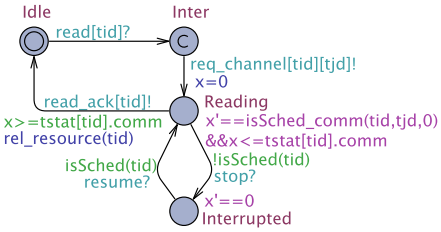


Fig. 7. Read channel with stopwatch

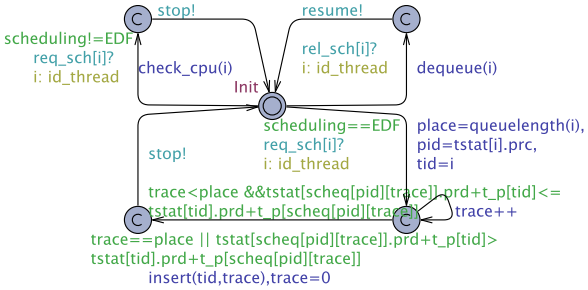


Fig. 8. Task scheduler

AUTOSAR timing extensions can be encoded in the proposed TAs or requirement specific automaton and the corresponding properties, as discussed in [17].

VII. Experimentations

We conduct experiments on the autonomous vehicle controller depicted in Fig. 2. The autonomous vehicle is modified from Dongfeng Fengshen AX7 SUV (Fig. 9(a)) with drive-by-wire ability. The vehicle is equipped with a variety of sensors and low-level actuators. The actuators, e.g., electronic power steering motor, brake motor, and throttle by-wire, receive and actuate commands from the controller. Vehicle data such as current steering angle is sensed by on-board sensors and obtained through a CAN bus. The IMU used for localization is an Inertial Navigation System (INS) aided by external Differential Global Positioning System (DGPS). Other sensors, e.g., LiDAR, Radar and camera, provide data for the centre computer. Then the centre computer delivers environment perception and generates a target path that is a sequence of position points with maximum speed information. Next, the controller keeps the vehicle tracking this path.

In the autonomous control system (Fig. 9(b)), the controller is a real-time application running under a Linux kernel with PREEMPT_RT patch. The type of processors for controller execution is Raspberry Pi 3, namely a 1.2GHz quad-core ARM Cortex A53 cluster. The communication between various processors is via Ethernet. We conduct real urban road tests and simulation tests with scenarios covering straight, curve, lane change, and u-turns.

The three functional blocks in PE are organized into *planner* task. The first two functional blocks in PT are grouped into

(a) Dongfeng Fengshen AX7

(b) Autonomous control systems

Fig. 9. The autonomous vehicle

ins task, and the other two tasks are *lateral* and *longitudinal*, respectively. The priorities of tasks in PT are higher than that of *planner*. The frequencies of PE and PT among various scenarios of the vehicle are listed in Table I, where "N/A" means not available. The execution times of the tasks are recorded when the controller runs on one processor. Their best and worst case execution times are analyzed based on the collected data, as presented in Table II.

TABLE I
Various frequencies (Hz) of PE and PT in different scenarios

Speed	Path tracking (PT)			Path extraction (PE)		
	Straight	Turn	U-turn	Straight	Turn	U-turn
10km/h	100	100	120	10	10	12
20km/h	100	100	181	10	10	19
30km/h	100	107	N/A	10	11	N/A
40km/h	120	N/A	N/A	12	N/A	N/A
60km/h	197	N/A	N/A	20	N/A	N/A

We mainly investigate system behavior under event triggered architecture, where model checker UPPAAL [16] is

TABLE II
WORST CASE EXECUTION TIME OF THE TASKS

tasks	planner	ins	lateral	longtitudinal
WCET(ms)	6.9216	0.3662	0.6674	0.5932
BCET(ms)	1.862	0.123	0.079	0.130

employed to verify whether the system is deadlock-free, and UPPAAL SMC is applied for validating timing property on worst case response time. The valuation of time in the controller is amplified 100 times to encode those constraints in UPPAAL. All the verification and validation experiments are performed on a MacBook with intel 2.5 GHz processor and 16.0 GB memory.

A. Deadlock checking

We consider two configurations in two scenarios presented in Table I for the deadlock-freedom checking of the case study: 1) all the tasks are allocated to the same processor; 2) *planner* is allocated to one processor, and the other three of the controller are allocated to the second processor, as shown in Fig. 2. As the WCET of *planner* is longest, and the reliability is lower, we assign three replicas for it to tolerate transient fault and meet system reliability requirement of $1 - 10^{-9}$. The analysis on deadlock-freedom of these configurations with various scheduling strategies under event triggered architecture is presented in Table III. In the table, "Y" marks the existence of deadlock, and "N" shows the system is deadlock-free. The cost is in seconds.

1) (100Hz, 10Hz): With one processor, a deadlock is detected under FP and EDF policies when no redundancy exists. The deadlock comes from the preemption between *planner* and *ins*. As illustrated in Fig. 10, when the former is writing data to the buffer, process *ins* is ready and applies for the processor. Because the priority of *ins* is greater, *planner* is interrupted. However, while *ins* is attempting to read data from the same buffer, the buffer is locked by *planner* and unaccessible. Consequently, none of the tasks can progress. Though we assume that writing is non-blocking in the application model, the implementation involves more low level details and the deadlock appears in writing. Therefore, it is still necessary to check the system model even if the functionality of the application model is correct. We can introduce additional policies, e.g. the process being busy in reading/writing cannot be preempted, to eliminate this deadlock. The deadlock does not exist under FIFO policy, for the non-existence of preemption.

When redundancy for *planner* exists, a deadlock is detected even under FIFO policy. The reasoning being that continuous execution of replicas from *planner* makes the deadline constraints of the other three be violated, and PT cannot progress any more. When the system is equipped with two processors, *planner* and *ins* do not interfere with each other. And the period of *planner* is enough to allow the temporal execution of the replicas without violating any constraints. Hence, no deadlock exists.

2) (197Hz, 20Hz): With one processor, we also detect a deadlock under FIFO policy, without any redundancy. In this setting, the period of PT is smaller than the WCET of *planner*. And executing the task makes the deadline violation of PT. Similar to the first scenario, when equipped with two processors, the system is deadlock-free.

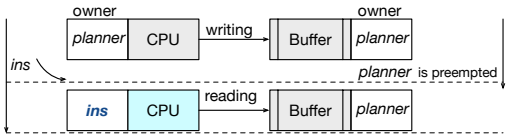


Fig. 10. Deadlock scenario under FP/EDF with one processor

Discussion. With stopwatches, UPPAAL yields an over-approximation of the reachable states, due to the encoding limitation of reachable states in difference bounded matrices (DBMs) [22], [24]. Without modifying any reachable states in the model, the over-approximation may relax the constraints with stopwatches, and introduce unreachable states. In this case, if the tool reports the existence of deadlock, it may be caused by the introduced unreachable states, and we need to check whether the deadlock is reachable. However, if the tool reports the non-existence of deadlock, it is conjectured that the original system contains no deadlock and the result is sound (which remains to be proved formally though). We can observe that with the current allocation of two processors, no deadlock occurs, for there exists no interference between PT and PE. However, if one of replicas of *planner* and the tasks of PT are allocated to the same processor, the deadlock may appear, as the cases with one processor.

B. Worst case response time analysis

As the priority of task *planner* is lower, it could be always preempted by the other three tasks when implemented in one processor (ECU). To measure the maximal time of the task from activation to being finished execution (worst case response time, WCRT) with EDF scheduling in scenario with frequency (100Hz, 10Hz), we add a timed automaton and a clock *plc* to the deadlock-free system model. The clock begins to progress when *planner* is ready and stops when the release of processor occurs. We use the following SMC query of UPPAAL SMC to check the WCRT:

$$E[<= 100000; 1000](max : plc)$$

The query requires UPPAAL SMC to return a probability distribution on the average of the maximum value of the clock *plc* from 1,000 individual simulation traces, each of which runs for 100,000 time units.

1) WCRT without redundancy: In Fig. 11, we present the probability distributions with EDF scheduling when PT and PE are mapped to the same processor, where the column in blue shows the probability of certain valuations, and the green line presents the average value. In the preemptive scheduling with EDF, *planner* is always executed after the other three tasks of PT. Fig. 11 shows that the WCRT is 852, and the average of WCRT over all produced traces is 813.

TABLE III
DEADLOCK-FREEDOM ANALYSIS UNDER VARIOUS CONFIGURATIONS

Frequency	Processors	Without redundancy						With redundancy					
		FIFO		FP		EDF		FIFO		FP		EDF	
		deadlock	cost	deadlock	cost	deadlock	cost	deadlock	cost	deadlock	cost	deadlock	cost
(100Hz,10Hz)	1	N	0.013	Y	0.014	Y	81.495	Y	19.08	Y	18.973	Y	73.195
	2	N	0.014	N	0.014	N	114.724	N	499.452	N	499.599	N	501.434
(197Hz,20Hz)	1	Y	0.814	Y	0.763	Y	0.765	Y	6.655	Y	6.719	Y	6.732
	2	N	1.358	N	1.361	N	37.497	N	257.753	N	254.956	N	256.6

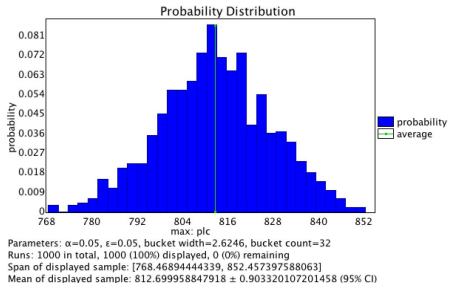


Fig. 11. Worst case response time for one *planner*

2) WCRT with redundancy: We further conduct the WCRT of planner for the controller implemented with three replicas of each task, under EDF policy, in scenario with frequencies 100Hz and 10Hz for PT and PE. Fig. 12 provides the WCRT probability distribution for planner equipped with three replicas. The WCRT with one processor is around 35ms where the vehicle can move ahead for 0.3 meters in the speed of 30km/h, which is acceptable in the navigation.

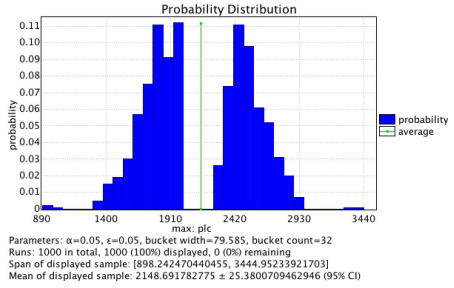


Fig. 12. Worst case response time for *planner* with redundancy

C. End-to-end analysis with on-line simulation

To evaluate the cooperation of the controller with other functional components under timing constraints, we conduct a road testing with the autonomous vehicle to check the end-to-end delay of PT and PE, respectively.

The length of path for the on-line simulation is 650 meters, as shown from point A to D in Fig. 13. The scenarios include going straight, making U-turns and changing lines. That is, the vehicle starts from point A, makes a U-turn at point B, then turns to left at point C, finally stops at point D. We take the selected path as the navigation output, and the vehicle still needs to plan the concrete path to keep tracing and avoid obstacles. The vehicle can accelerate or decelerate autonomously, within speed limitation of 40km/h.

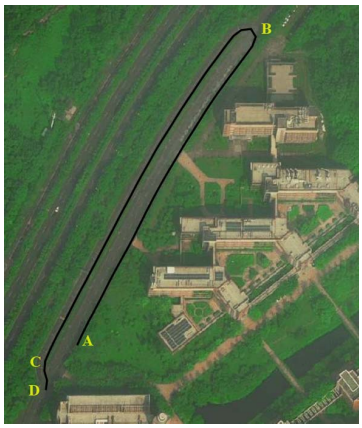


Fig. 13. The path for on-line simulation.

Each task in the controller is equipped with three replicas. A dedicated Ethernet is used in the experiment. There is no other data traffic like raw sensor data flows in the network.

We deploy two processors for fault-tolerance with the redundancy of three replicas for each individual task. The road testing is conducted within 120 seconds.

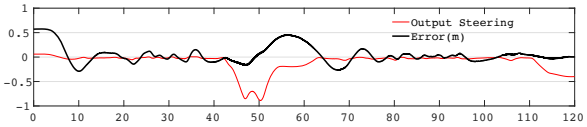
The end-to-end latency for PT records the duration from generating pose information in localization module till the trigger of the actuators, which includes the transmission of pose information, the execution of tasks and replicas in PT, communications between tasks and voters, and majority voting. The latency for PE records the duration from motion planning module generating reference path till the ending of buffer switch, which includes the transmission of reference path and the execution of tasks and replicas in PE, and additional latency caused from redundancy.

TABLE IV
END-TO-END DELAY OF THE ON-LINE SIMULATION (MS)

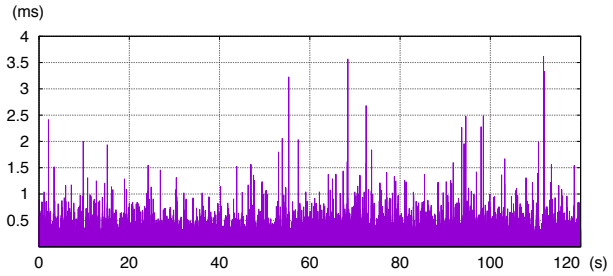
Component	Minimal	Maximal	Average	Relative deadline
PT	0.5234	3.7870	0.6188	5.0
PE	20.3281	25.3125	23.9560	50.0

In Table IV, all the latency are smaller than their deadlines. As the period of PT is shorter, its maximal latency is closer to the deadline. However, for PE, it can be finished quite early. The minimal end-to-end latency is even smaller than the WCET of tasks in PT. The reason being that we take the worst case execution time in static analysis, to ensure that the system can always meet timing constraints. We provide

characteristics of lateral vehicle control and the distribution of end-to-end latency during the road testing in Fig. 14, where the horizontal shows the time in seconds. In Fig. 14(a), `error` means the offset between target path and real trajectory, and `Output Steering` means the normalized steering angle where 1 stands for steering to the left end and -1 stands for steering to the right end. According to the change of `Output Steering`, we can infer that the vehicle makes the U-turn around 50 seconds, and approaches point C around 110 seconds. During the road testing, the absolute errors between two trajectories in straight paths and normal turns are less than 0.3 meters, while the absolute error of U-turns is less than 0.5 meters. The bigger errors from point B come from unstable path reference generated from the planning module. The error at point A is resulted from the difference between the locations of the vehicle before testing and the reference. The distribution in Fig. 14(b) shows that changing directions in points A, B and C could incur long delays (in milliseconds). However, as shown in Fig. 14(a), the caused errors are acceptable for the scenario.



(a) Lateral vehicle control



(b) Distribution of end-to-end delay

Fig. 14. The results in road testing.

VIII. Conclusion

One of significant issues in developing safety-critical autonomous control system is how to correctly integrate applications with a given platform, such that the application behavior does not violate any requirements. We present a design and analysis framework for such systems with reliability and safety guarantees, which is obtained by unfolding the redundancy in the application model for transient fault tolerance and integrating it into the system model transformed from an AUTOSAR model. The system model encapsulates not only temporal relation between computation and communication ingredients, but also the necessary timing constraints, which can be applied with existing techniques for safety property verification and performance evaluation.

References

[1] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE*, vol. 4, no. 1, pp. 15–24, 2016.

[2] R. Obermaisser, *Event-triggered and time-triggered control paradigms*. Springer Science & Business Media, 2004, vol. 22.

[3] S. Bliudze, S. Furic, J. Sifakis, and A. Viel, "Rigorous design of cyber-physical systems," *Software & Systems Modeling*, pp. 1–24, 2017.

[4] O. Group *et al.*, "Osek/vdx operating system specification," *site web: http://www. osek-vdx. org*, 2005.

[5] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange, "Autosar–a worldwide standard is on the road," in *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, vol. 62, 2009, p. 5.

[6] J. Jiang and X. Yu, "Fault-tolerant control systems: A comparative study between active and passive approaches," *Annual Reviews in control*, vol. 36, no. 1, pp. 60–72, 2012.

[7] G. Sandmann and R. Thompson, "Development of autosar software components within model-based design," SAE Technical Paper, Tech. Rep., 2008.

[8] C. Knüchel, M. Rudorfer, S. Voget, S. Eberle, R. Sezestre, and A. Loyer, "Artop–an ecosystem approach for collaborative autosar tool development," in *International Congress on Embedded Real Time Software and Systems*, 2010.

[9] G. Macher, E. Armengaud, and C. Kreiner, "Automated generation of AUTOSAR description file for safety-critical software architectures," *Informatik 2014*, 2014.

[10] J. Denil, P. De Meulenaere, S. Demeyer, and H. Vangheluwe, "DEVS for AUTOSAR-based system deployment modeling and simulation," *Simulation*, vol. 93, no. 6, pp. 489–513, 2017.

[11] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller, "Timing modeling and analysis for autosar-based software development-a case study," in *DATE*. IEEE, 2010, pp. 642–645.

[12] S. Anssi, S. Tucci-Piergiovanni, S. Kuntz, S. Gérard, and F. Terrier, "Enabling scheduling analysis for autosar systems," in *ISORC*. IEEE, 2011, pp. 152–159.

[13] Q. Zhao, Z. Gu, and H. Zeng, "Design optimization for autosar models with preemption thresholds and mixed-criticality scheduling," *Journal of Systems Architecture*, vol. 72, pp. 61–68, 2017.

[14] J. H. Kim, A. Legay, K. G. Larsen, M. Mikucionis, and B. Nielsen, "Resource-parameterized timing analysis of real-time systems," in *HVC*, 2015, pp. 190–205.

[15] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[16] G. Behrmann, A. David, and K. Larsen, "A tutorial on uppaal," *Formal methods for the design of real-time systems*, pp. 33–35, 2004.

[17] S. Beringer and H. Wehrheim, "Verification of autosar software architectures with timed automata," in *Critical Systems: Formal Methods and Automated Verification*. Springer, 2016, pp. 189–204.

[18] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh, "Schedulability and end-to-end latency in distributed ECU networks: Formal modeling and precise estimation," in *EMSOFT*. ACM, 2010, pp. 129–138.

[19] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[20] S. Pandey and B. Vermeulen, "Transient errors resiliency analysis technique for automotive safety critical applications," in *DATE*, 2014, p. 9.

[21] B. Zheng, H. Liang, Q. Zhu, H. Yu, and C.-W. Lin, "Next generation automotive architecture modeling and exploration for autonomous driving," in *VLSI (ISVLSI)*. IEEE, 2016, pp. 53–58.

[22] F. Cassez and K. Larsen, "The impressive power of stopwatches," in *International Conference on Concurrency Theory*. Springer, 2000, pp. 138–152.

[23] J. Huang, S. Barner, A. Raabe, C. Buckl, and A. Knoll, "A framework for reliability-aware embedded system design on multiprocessor platforms," *Microprocessors and Microsystems*, vol. 38, no. 6, pp. 539–551, 2014.

[24] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou, "Statistical and exact schedulability analysis of hierarchical scheduling systems," *Science of Computer Programming*, vol. 127, pp. 103–130, 2016.