INVITED: Extensibility-Driven Automotive In-Vehicle Architecture Design

Qi Zhu
University of California, Riverside
Riverside, USA
qzhu@ece.ucr.edu

Hengyi Liang
University of California, Riverside
Riverside, USA
hlian010@ucr.edu

Licong Zhang
Technical University of Munich
Munich, Germany
licong.zhang@tum.de

Debayan Roy
Technical University of Munich
Munich, Germany
debayan.roy@tum.de

Wenchao Li
Boston University
Boston, USA
wenchao@bu.edu

Samarjit Chakraborty
Technical University of Munich
Munich, Germany
samarjit@tum.de

ABSTRACT

Increasingly more software-based applications are being developed and deployed in modern vehicles. As a result, the extensibility of a system design has become an important issue in order to accommodate more future applications and update of existing ones on one hand and reduce the effort and cost of re-design, test and validation on the other. In this paper, we discuss the extensibility-driven design in the automotive E/E architecture. We explain the motivation for such a design objective and discuss the definition of extensibility metric and extensibility-driven design methods under two different setting, namely the system based on CAN bus and FlexRay bus. Based on these two examples, we illustrate the importance and advantages of extensibility-driven design in the automotive E/E architecture.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**;

KEYWORDS

extensibility, automotive E/E architecture, CAN, FlexRay

ACM Reference format:
Qi Zhu, Hengyi Liang, Licong Zhang, Debayan Roy, Wenchao Li, and Samarjit Chakraborty. 2017. INVITED: Extensibility-Driven Automotive In-Vehicle Architecture Design. In *Proceedings of DAC '17, Austin, TX,USA, June 18-22, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3061639.3072956

1 INTRODUCTION

The automotive Electrical/Electronic (E/E) system is currently undergoing drastic changes. Current premium-class vehicles may consist of up to 100 Electronic Control Units (ECUs) and hundreds of million lines of code [1]. Increasingly more software-based functions are being developed and deployed rapidly, especially from the domain of Advanced Driver Assistance Systems (ADAS), autonomous driving and connectivity. As a result, the ability of the system to accommodate new functions or update existing ones has become one important requirement on the future automotive E/E architecture. To enable this ability, however, is not an easy task. Many of the functions in a vehicle are implemented in a distributed manner, i.e., the software implementations are partitioned into several tasks with data dependencies. The tasks and the data transmission are mapped on a physical architecture consisting of a number of ECUs and several communication buses. The tasks are mapped on the ECUs and the data transmission is realized through bus protocols like Control Area Network (CAN), FlexRay or Ethernet. The common scenario is that multiple functions must share embedded resources like the communication, computation and memory resources. Therefore, the ability of the system to accommodate new applications or applications with changed resource requirements depends strongly on the platform parameters like mapping and scheduling of other applications sharing the same resources.

On the other hand, the design and development of the E/E system in the automotive domain usually follows an incremental design process, where the design of existing components are kept unchanged as much as possible during the next design iteration. One important reason here is the cost of re-design. Since many of the functions in the automotive domain are safety-critical control functions, such applications need to be rigorously tested and validated for their functional and non-functional (e.g., timing) behaviours due to the requirements for safety and reliability. If the parameters (e.g., task mapping, message packing and scheduling) of existing functions need to be re-designed, they need to go through the testing and validation process again, thus resulting in much more cost. Due to the cost-sensitive nature of the automotive industry, such effort and cost need to be kept as low as possible. Therefore, the optimization of a system design towards *extensibility*, i.e., additional applications are more easily added to the current system without change or with minimal changes of existing applications, has become an important issue.

In the embedded and real-time systems community, the traditional design optimization objectives are timing and resource-efficiency. The problem of extensibility has not been sufficiently addressed. In the recent years, some research works have emerged towards the quantification of extensibility and the extensibility-driven design in different settings [3–8]. Such works usually provide certain metrics to quantify the extensibility of a system design and propose extensibility-driven design methods based on these metrics. Although extensibility is a general problem that holds true for the entire E/E architecture, for different system setting, the definitions and methods might be different. In this paper, we use two examples, one for CAN-based system and one for FlexRay-based system, to illustrate the importance and the advantages of extensibility-driven design in the automotive E/E architecture. In Section 2, we first explain a method for extensibility-driven mapping for a system based on the CAN bus. This is followed by another approach on

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DAC '17, Austin, TX,USA
© 2017 ACM. 978-1-4503-4927-7/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3061639.3072956

extensible scheduling for a FlexRay-based system in Section 3. We then conclude in Section 4.

2 EXTENSIBILITY-DRIVEN MAPPING FOR CAN-BASED SYSTEMS

CAN is the prevalent bus protocol in current automotive electronic systems. Many automotive functions (e.g., those for active safety) collect data from sensors, perform computation on a set of ECUs connected through CAN buses, and then send command to actuators. In this section, we discuss approaches to model, analyze and optimize extensibility for such CAN-based *distributed* systems.

We measure extensibility based on how much the execution time of tasks can be increased without changing system configuration or violating *timing constraints*, as defined in [7, 8]. Such definition not only reflects how easy it is to add future functionality (or updating existing ones) within minimum changes, but also shows how robust the system is with respect to variations in task execution time. The timing constraints include schedulability of tasks on ECUs and messages on CAN buses, as well as end-to-end latency deadlines along functional paths.

The automotive functions are often captured by functional blocks communicating through signals. During software implementation, these blocks are mapped into tasks and then tasks are allocated to ECUs. The signals are mapped into local communication or packed into messages that are exchanged over buses. The task execution and message transmission are scheduled based on the assignment of priorities. To ensure the timing constraints are met in the worst case and to further optimize the extensibility metric, we may formulate an *extensibility-driven mapping* problem to explore the allocation of tasks to ECUs, the packing of signals to messages, the allocation of messages to CAN buses, and the assignment of priorities to tasks and messages. This extensibility-driven mapping problem requires detailed timing modeling and analysis of task execution and CAN message transmission. Next, we will introduce the system model and extensibility metric definition, discuss the key timing constraints in addressing this extensibility metric, and briefly introduce the algorithm for extensibility-driven mapping as presented in [7].

2.1 System Model

We consider systems based on *static priority based scheduling* of periodic tasks and messages. Periodically activated tasks on ECUs read the input data from sensors, compute intermediate results, and write them to output buffers – from where they can be read by other tasks or used for assembling the data content of messages. Periodically activated messages transmit the data from output buffers to input buffers on remote ECUs. Local clocks on different ECUs are not synchronized. Tasks may have multiple fan-ins and messages can be multi-cast. Eventually, task outputs are sent to the system's output devices or actuators.

We assume that the functional blocks have already been mapped into software tasks, and the functionality is now represented as a directed task graph $\mathcal{G} = (\mathcal{T}, \mathcal{J})$. $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ is the set of tasks that perform the computation. $\mathcal{J} = \{s_1, s_2, ..., s_m\}$ is the set of signals that are exchanged between task pairs. src_{s_i} and $\{dst_{s_i,j}\}$ denote the source task and the set of destination tasks of signal s_i, respectively (note that communication can be multi-cast). During mapping, this task graph is mapped onto an architecture platform that consists of a set of ECUs $\mathcal{V} = \{v_1, v_2, ..., v_p\}$ connected through a set of CAN buses $\mathcal{B} = \{b_1, b_2, ..., b_q\}$.

τ_i is periodically activated with period t_{τ_i}, and executed with priority γ_{τ_i}. The periods of communicating tasks are assumed to be harmonic, which is usually true in practical designs. Tasks are scheduled with preemption according to their priorities, and a total order exists among the task priorities on each ECU. We use $e_{\tau_i,v}$ to denote the execution time of task τ_i on ECU v. In the following, the v subscript is dropped whenever the formula refers to tasks on one ECU. Finally, r_{τ_i} denotes the worst case response time of τ_i.

For a signal s_i, the ECUs to which the source task src_{s_i} and the destination task $dst_{s_i,j}$ are allocated are called source and destination ECUs, respectively. If the source ECU is the same as all the destination ECUs, the signal is local. Otherwise, it is global and must be packed into a message transmitted on the CAN buses between the source ECU and all its destination ECUs. Only signals with the same period, same source ECU and same communication bus can be packed into the same message. For message m_i, t_{m_i} denotes its period, γ_{m_i} denotes its priority, and e_{m_i} denotes its worst case transmission time on a bus with unit speed. The worst transmission time on bus b_j is $e_{m_i}/speed_{b_j}$, where $speed_{b_j}$ is the transmission speed of b_j. r_{m_i} is the worst case response time on a bus with unit speed. In addition, in complex systems the source and destination tasks may not reside on ECUs that share the same bus. In this case, a signal exchanged among them will have to go through a gateway ECU and be forwarded by a gateway task.

A path p on the task graph $\mathcal{G}$ is an ordered interleaving sequence of tasks and signals, defined as $p = [\tau_1, s_1, \tau_2, s_2, ..., s_{k-1}, \tau_k]$. $src(p) = \tau_1$ is the path's source and $snk(p) = \tau_k$ is its sink. Sources are activated by external events, while sinks activate actuators. Multiple paths may exist between each source-sink pair. The worst case end-to-end latency incurred when traveling a path p is denoted as l_p. The path deadline for p, denoted by d_p, is an application requirement that may be imposed on selected paths.

2.2 Extensibility Metrics

Different metrics can be defined to measure the extensibility of task execution time in a CAN-based system. The main definition used in [7, 8] is a weighted sum of each task's *execution time slack* over its period:

$$max. \quad E = \sum_{\tau_i \in \mathcal{T}} w_{\tau_i} \frac{\Delta e_{\tau_i}}{t_{\tau_i}} \tag{1}$$

where a task's execution time slack Δe_{τ_i} is defined as the maximum possible increase of its execution time e_{τ_i} without violating the design constraints, assuming the execution times of other tasks are not changed. The design constraints include resource utilization constraints for ECUs and CAN buses, schedulability constraints for tasks and messages, and end-to-end latency deadlines for paths.

The metric defined in (1) is a generic measurement of extensibility at task level, where w_{τ_i} is a preassigned weight that indicates how likely and how much the task's execution time will be increased in future functionality extensions. In practice, however, because of functional dependencies, execution time increases in tasks belonging to a set may need to be considered jointly. This can be done in several ways. One possible way is in the assignment of the w_{τ_i} weights as follows: 1) Identify a set of update scenarios $u_1, u_2, ...u_n$. Each scenario u_k includes a group of tasks $\mathcal{T}_k$ to be extended, and is assigned with a likelihood probability ρ_k. 2) For each update scenario u_k and $\tau_i \in \mathcal{T}_k$, assign a weight w_{i_k} to represent how much the task's execution time will be increased in this scenario. 3) Compute the final weight w_{τ_i} of a task as $w_{\tau_i} = \sum_{k:\tau_i \in \mathcal{T}_k} \rho_k * w_{i_k}$. In [7], a more explicit definition that identifies the groups of tasks that are functionally related is presented.

Finally, another formulation is to use execution time slack over original execution time, i.e. $\Delta e_{\tau_i}/e_{\tau_i}$, instead of using execution time slack over period in (1).

2.3 Timing Constraints

To measure and optimize the above extensibility metrics, it is essential to analyze the timing constraints that have to be satisfied in worst case.

End-to-end latency: Once tasks are allocated to ECUs, some signals are local and their transmission time is assumed to be zero. Others are global, and need to be transmitted on the CAN buses through messages. The time needed to transmit a global signal is equal to the transmission time of the corresponding message. Let r_{s_i} denote the worst case response time of a global signal s_i, and assume its corresponding message is m_j, then $r_{s_i} = r_{m_j}$.

The worst case end-to-end latency can be computed for each path by adding the worst case response times of all the tasks and global signals on the path, as well as the periods of all the global signals and their destination tasks on the path.

$$l_p = \sum_{\tau_i \in p} r_{\tau_i} + \sum_{s_i \in p \wedge s_i \in GS} (r_{s_i} + t_{s_i} + t_{dst_{s_i}}) \tag{2}$$

where GS is the set of all global signals. In the case that gateways are used across buses, the signals to and from possible gateway tasks, as well as the response time of the gateway task itself and the associated sampling delays must be included in the analysis. We need to include periods of global signals and their destination tasks because of the asynchronous sampling of communication data, as detailed in [2, 7]. For local signals, the destination task can be activated with a phase (offset) equal to the worst-case response time of the source task, under our assumption that their periods are harmonic. In this case, we only need to add the response time of the destination task.

For selected paths, there may be deadline constraints on end-to-end latencies, i.e., $l_p \leq d_p$.

Response time: As shown in (2), computing end-to-end latencies requires the computation of task and message response times (note that the response time of a global signal is equal to the response time of its corresponding message). In a system with preemptive priority-based scheduling, the worst case response time r_{τ_i} for a task τ_i depends on its computation time e_{τ_i}, as well as on the interference from higher priority tasks on the same ECU. In the case of $r_{\tau_i} \leq t_{\tau_i}$, r_{τ_i} can be calculated as follows.

$$r_{\tau_i} = e_{\tau_i} + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{r_{\tau_i}}{t_{\tau_j}} \right\rceil e_{\tau_j} \tag{3}$$

where $hp(\tau_i)$ is the set of higher priority tasks on the same ECU.

Worst case message response times are calculated similarly to task response times. The main difference is that message transmissions on the CAN bus are not preemptable. Therefore, a message m_i may have to wait for a blocking time BL_{max}, which is the longest transmission time of any frame in the system. Likewise, the message itself is not subject to preemption from higher priority messages during its own transmission time e_{m_i}. The response time can therefore be calculated with the following recurrence relation, in the case of $r_{m_i} \leq t_{m_i}$:

$$r_{m_i} = e_{m_i} + BL_{max} + \sum_{m_j \in hp(m_i)} \left\lceil \frac{r_{m_i} - e_{m_i}}{t_{m_j}} \right\rceil e_{m_j} \tag{4}$$

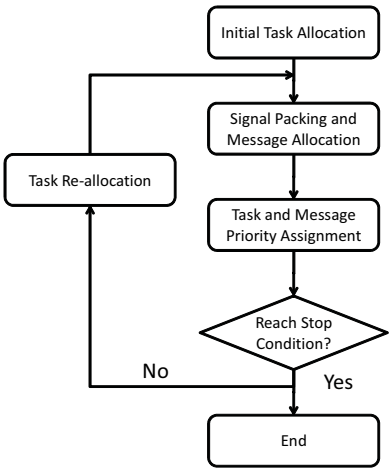

Figure 1: Algorithm flow for extensibility-driven mapping in CAN-based systems.

For tasks and messages, their response times should be within their deadlines to ensure schedulability. In most cases, the deadlines are set the same as periods, and we will have schedulability constraints as $r_{\tau_i} \leq t_{\tau_i}$ and $r_{m_i} \leq t_{m_i}$.

Extensibility computation: Based on the above formulas and constraints for end-to-end latencies and response times, as well as utilization constraints, we can compute the extensibility for each task, denoted as Δe_{τ_i}. If Δr_{ij} denotes the increase of task τ_j's response time r_{τ_j} when task τ_i's computation time e_{τ_i} is increased by Δe_{τ_i}, the end-to-end latency constraints and utilization constraints are expressed as follows:

$$\sum_{\tau_j \in p \wedge \tau_j \in (lp(\tau_i) \cup \{\tau_i\})} \Delta r_{ij} \leq d_p - l_p \qquad \forall p, \forall \tau_i \in \mathcal{T} \tag{5}$$

$$\frac{\Delta e_{\tau_i}}{t_{\tau_i}} + \sum_{\tau_j \in \mathcal{T}(v)} \left(\frac{e_{\tau_j}}{t_{\tau_j}}\right) \leq u_v \qquad \forall v, \forall \tau_i \in \mathcal{T}(v) \tag{6}$$

where $lp(\tau_i)$ is the set of lower priority tasks on the same ECU as τ_i, $\mathcal{T}(e)$ denotes the set of the tasks on ECU v, and u_v denotes the maximum utilization allowed on e.

The relation between Δr_{ij} and Δe_{τ_i} can be derived from (3) as follows.

$$\Delta r_{ij} = \sum_{\tau_k \in hp(\tau_j)} \left(\left\lceil \frac{r_{\tau_j} + \Delta r_{ij}}{t_{\tau_k}} \right\rceil - \left\lceil \frac{r_{\tau_j}}{t_{\tau_k}} \right\rceil \right) e_{\tau_k} + \left\lceil \frac{r_{\tau_j} + \Delta r_{ij}}{t_{\tau_i}} \right\rceil \Delta e_{\tau_i} \qquad \forall \tau_j \in lp(\tau_i) \tag{7}$$

$$\Delta r_{ii} = \Delta e_{\tau_i} + \sum_{\tau_k \in hp(\tau_i)} \left(\left\lceil \frac{r_{\tau_i} + \Delta r_{ii}}{t_{\tau_k}} \right\rceil - \left\lceil \frac{r_{\tau_i}}{t_{\tau_k}} \right\rceil \right) e_{\tau_k} \tag{8}$$

2.4 Extensibility-Driven Mapping

Based on Equations (2) to (8), we can compute task extensibility Δr_{τ_i} for any *given mapping*. For instance, one simple and possibly time-consuming approach is to use the bisection (i.e., binary search) method. It is much more challenging to optimize the extensibility metric as defined in (1) through the exploration of different mapping

options (i.e., different task and message allocation, signal packing, and task and message scheduling). We may introduce variables to represent these options and try to formulate an integrated formulation for extensibility-driven mapping optimization. However, such formulation is non-linear and cannot be linearized due to the second term in Equation (7). It could be solved by generic nonlinear solvers but the complexity is in general too high for industrial size applications.

Therefore, in [7], we propose an algorithm that includes two stages for reducing complexity: one initial stage that is based on mixed integer linear programming (MILP), and one refinement stage that consists of several steps based on heuristics.

The flow of this algorithm is shown in Figure 1. First, we decide the initial allocation of tasks by solving an MILP formulation. Utilization constraints are considered in place of the true extensibility metric to allow a linear formulation. In this stage, we also assume each CAN message only contains one signal, and assume the initial task and message priorities are either given or assigned based on rate monotonic policy. Then, a series of heuristics is used in the refinement stage: in the signal packing and message allocation step, a heuristic is used to decide signal-to-message packing and message-to-bus allocation. In the task and message priority assignment step, an iterative method is designed to assign the priorities of tasks and messages. After these steps are completed, if the design constraints cannot be satisfied or if further improvement of extensibility is needed, the tasks can be re-allocated and the process repeated. Because of the complexity of the MILP formulation, a heuristic is designed for task re-allocation, based on the extensibility and end-to-end latency values obtained in the previous steps.

3 EXTENSIBILITY-DRIVEN DESIGN WITH FLEXRAY

Over the last few years, FlexRay has become an important automotive in-vehicle communication network for the implementation of safety-critical and fault-tolerant systems. Typically, the design and implementation of such systems involves intensive testing and verification. Correspondingly, once the system properties are verified, it is not recommended to modify the corresponding design parameters. Therefore, given the iterative design paradigm followed in automotive domain, it is important to design such systems in an extensible manner. Towards this, in this section, we will first discuss the FlexRay protocol, and subsequently, we will state and explain the extensibility metrics for FlexRay network design and describe how schedules for such a network can be computed considering the metrics.

3.1 The FlexRay Protocol

FlexRay supports hybrid communication protocol, and correspondingly, each FlexRay time cycle is mainly partitioned into *static* (ST) and *dynamic* (DYN) segments as shown in Figure 2. The static segment exhibits time-division multiple access (TDMA) communication and comprises number of *slots* of equal length (Δ), which can be represented as $\mathcal{S}_{ST} = \{1, 2, ..., N\}$. Here, a message assigned to a static slot is transmitted within the corresponding time window. Thus, the start and end of a message transmission is exactly known. On the other hand, dynamic segment is partitioned into number of *minislots* of equal length (δ), where typically $\delta << \Delta$. A message assigned to the dynamic segment may consume more than one minislot as shown in Figure 2. Correspondingly, a dynamic slot is a logical entity with one or more minislots allowing flexible TDMA communication. Here, dynamic slots are denoted by $\mathcal{S}_{DYN} = \{N + 1, ..., M\}$.

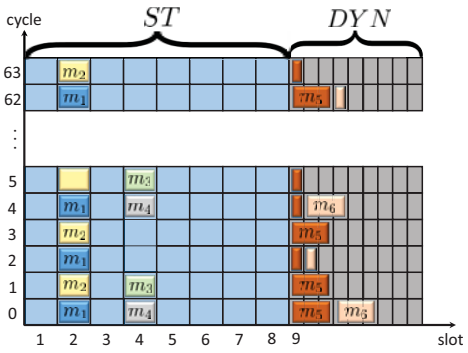

Figure 2: Example of FlexRay communication.

This hybrid communication can be realized using a slot counter, C, where the counter starts from 1 at the beginning of each cycle. Here, when $C = j$, then the message m_i assigned to the j-th slot is sent over the bus (if ready). In the static segment, the counter is incremented after every Δ time units, i.e., at the end of each slot. Correspondingly, if no new data has arrived at the beginning of the slot then the whole slot length, i.e., Δ time units, are wasted as shown in Figure 2 for m_2 in the cycle 5. On the other hand, in the dynamic segment, when a message m_i has some data to be sent then the counter is updated at the end of the last minislot where the message is transmitted. However, if there is no data then the counter is updated at the end of the current minislot, and correspondingly, only δ time units are wasted (refer to Figure 2 for messages m_4 and m_5). Correspondingly, the start and end of a message transmission may vary. Thus, it may be noted that FlexRay is time-deterministic in the static segment and resource-efficient in the dynamic segment.

FlexRay schedules: FlexRay communication is organized as an infinite repetition of 64 bus cycles, i.e., the cycle counter counts from 0 to 63 and then resets. Therefore, the resource allocation in any 64 consecutive cycles will be repeated in the next 64 cycles and so on. For this definition, a message m_i is transmitted with a schedule Θ_i which can be represented as a tuple $\Theta_i = \{S_i, B_i, R_i\}$. Here, (i) S_i is the assigned *slot number*, (ii) the *base cycle*, B_i, represents the first cycle where the message is allowed to be sent, and, (iii) the *cycle repetition rate*, R_i, denotes the number of cycles between two consecutive time slot allocations. In Figure 2, schedules of the messages are given by $\Theta_1 = \{2, 0, 2\}$, $\Theta_2 = \{2, 1, 2\}$, $\Theta_3 = \{4, 1, 4\}$, $\Theta_4 = \{4, 0, 4\}$, $\Theta_5 = \{9, 0, 1\}$ and $\Theta_6 = \{10, 0, 2\}$. Furthermore, it must be noted that FlexRay allows *slot-multiplexing*, i.e., same slot number can be assigned to more than one message, however, they must not overlap. This is illustrated by $\{m_1, m_2\}$ and $\{m_3, m_4\}$ in Figure 2. In addition, other FlexRay scheduling constraints include (i) $R_i \in \{2^n | 0 \leq n \leq 6\}$ and (ii) $B_i < R_i$.

3.2 A motivational example

In the iterative design paradigm of automotive domain, new applications are added onto existing systems incrementally. Let us consider such a design iteration, where the existing FlexRay network looks like as shown in Figure 2 and we want to add a message m_7 for which $R_7 \leq 2$ is given. Here, we will consider two options, i.e., mapping m_7 onto slot 4 and dynamic segment respectively.

(i) In the first case, it is not possible to map m_7 on slot 4 with the existing schedules. However, if m_3 was schedule as $\Theta_3 = \{4, 2, 4\}$, then it would have been possible to map m_7 as $\Theta_7 = \{4, 1, 2\}$. Thus, we may say that existing schedule is less extensible with

regard to slot 4 as compared to a schedule where $\Theta_3 = \{4, 2, 4\}$ and $\Theta_4 = \{4, 0, 4\}$.

(ii) In the *dynamic segment*, let us consider that m_7 consumes 3 minislots if it has data required to be sent. Now, to illustrate extensibility of dynamic segment, it is also important to consider deadline for each real-time message mapped onto dynamic segment. Here, we assume absolute deadlines of the messages m_5, m_6 and m_7 as the ends of minislots 9, 6 and 5 respectively of the corresponding cycle where slots are allocated for them. Correspondingly, with the existing schedule, even the best possible schedule for m_7, i.e., $\Theta_7 = \{10, 1, 2\}$ violates the deadline for m_7. However, if m_5 and m_6 were mapped as $\Theta_5 = \{10, 0, 1\}$ and $\Theta_6 = \{9, 0, 2\}$ respectively then $\Theta_7 = \{9, 1, 2\}$ can satisfy the deadlines for all the messages. Thus, it may be said that existing schedule is not extensible enough for a future message like m_7.

This example illustrates the need for extensible design for a very small system. However, with real industrial networks with hundreds of slots and messages, it is challenging to derive metrics which consider the above-mentioned cases.

3.3 Extensibility Metrics

Towards quantifying the extensibility of FlexRay schedules, three metrics can be used [4, 5], namely (i) the *quality rating*, (ii) *grade of extensibility* and (iii) *extensibility index*.

Quality rating: The quality rating of a slot S_j quantifies the real-time capabilities of the slot while accommodating future messages. For a particular slot S_j, this metric can be defined as

$$P_1(S_j) = \begin{cases} 0, & \forall S_j \in \mathcal{R} \\ 1, & \forall S_j \in \mathcal{S}_{ST} \setminus \mathcal{R} \\ 1 - e^{-k(\frac{|S_j - (N+M)|}{S_j - (N+1)})}, & \forall S_j \in \mathcal{S}_{ST} \setminus \mathcal{R}, \end{cases} \tag{9}$$

where $\mathcal{R}$ represents the set of *reserved slots* and k is a coefficient. The reserved slots, i.e., $S_j \in \mathcal{R}$, are not available for the current design iteration, and thus, the corresponding extensibility is 0. For an available static slot, the timing of the slot is fixed and not influenced by other messages, and thus, their quality rating evaluates to 1. On the other hand, in the dynamic segment, the timing of an unreserved slot is influenced by the messages allocated to the preceding slots, and thus, their quality rating can be quantified according to the slot number. In general, the higher the slot number, the smaller is the value of the quality rating. It may be noted that the idea of extensibility here is to spare the slots with higher quality rating for future design as much as possible while satisfying the real-time constraints for current messages considering the worst-case in the future. This addresses the second problem discussed in Sec. 3.2.

Grade of extensibility: This metric indicates the number of available schedules that may be assigned to a particular slot S_j. The cycle repetition rates and the base cycles of the schedules allocated to a slot determines the extensibility of the slot in the case of *slot-multiplexing*. Here we characterize the number of choices for schedules with slot number S_j as

$$C(S_j) = \sum_{R_l} \alpha(R_l), \forall R_l \in \{1, 2, 4, 8, 16, 32, 64\}, \tag{10}$$

where $\alpha(R_l)$ denotes the number of choices available for the cycle repetition rate R_l. In the case of an empty slot, $\alpha(R_l) = R_l$. However, if some messages are already mapped on the slot S_j, $\alpha(R_l)$ must be smaller than R_l. The grade of extensibility of a slot S_j is thus defined as

$$P_2(S_j) = \frac{C(S_j)}{C(\hat{S}_j)}, \tag{11}$$

where $C(\hat{S}_j)$ denotes the number of schedule choices if S_j is empty. Here, the idea is to keep as many schedule options as possible for future. Thus, this addresses the first problem discussed in Sec. 3.2.

Extensibility index: The above two metrics quantify the extensibility of a slot from different perspectives. However, to understand the trade-off between the two extensibility measures for a particular slot, a combined metric can be derived as

$$E(S_j) = P_1(S_j)P_2(S_j), \quad \forall S_j \in \mathcal{S}_{ST} \cup \mathcal{S}_{DYN}. \tag{12}$$

The extensibility index helps the system designer to observe the extensibility of a slot by considering both the number of available schedules for future messages and the corresponding real-time capabilities, and therefore, the resource bottlenecks for communication can be identified and suitable scheduling strategies can be employed.

Effective network extensibility: The extensibility index is only defined for a specific slot S_j. To represent the extensibility of a FlexRay cluster, we sum them up for all the slots,

$$E_{FR} = \frac{1}{N+M} \sum_{S_j=1}^{N+M} E(S_j). \tag{13}$$

Consider further that $\mathcal{M}$ denotes the set of messages to be scheduled and $\mathcal{L} \subseteq \mathcal{M}$ denotes the set of messages that that have already been assigned feasible schedules and κ denotes the penalty coefficient. We can then represent the effective network extensibility as

$$E_{eff} = max(E_{FR} - \frac{\kappa(|\mathcal{M}| - |\mathcal{L}|)}{N+M}, 0) \tag{14}$$

3.4 Extensibility-driven Schedule Synthesis

Schedule synthesis is an important problem in the design of FlexRay-based ECU networks. The extensible schedule synthesis addresses the problem of assigning messages to FlexRay schedules so that the real-time requirement of the messages are met and the network extensibility is optimized, i.e., the network is more likely to accommodate future messages. Towards this, we first need to derive the conditions for meeting the real-time requirements, i.e., the message will be successfully transmitted and that the worst-case network delay meets the corresponding deadline.

Compatibility Test: To compute the worst-case delay for a message mapped on the static segment is straight forward due to its deterministic nature. The delay for a schedule on a dynamic slot, on the other hand, depends on the behaviours of the messages mapped on preceding dynamic slots. Taking the future messages into consideration, the worst-case delay for a message assigned to a schedule $\Theta_i = \{S_i, B_i, R_i\}$ on the dynamic segment can be represented as

$$\tilde{D}_i = R_i T_{bus} + \max_{k \in \mathcal{K}_i} (\sum_{j=N+1}^{S_i - 1} \tilde{e}_j) + e_i. \tag{15}$$

Here, the first term, $R_i T_{bus}$, denotes the blocking time of the schedule, i.e., when a message just missed its schedule, it has to wait until the next instance of the schedule. The third term, e_i, denotes the transmission time of the message can can be computed as $e_i = c_i \delta$, where c_i is the number of minislots required. The second term calculates the delay component corresponding to the number of minislots by which the slot S_i may shift in the worst-case among

all possible instances of message transmission. Here, $\mathcal{K}_i$ represents the set of cycle numbers in which the slot S_i is allocated to m_i and is given by $\mathcal{K}_i = \{B_i + nR_i | n \in \mathbb{Z} \wedge 0 \leq n < \frac{64}{R_i}\}$. In addition, $\bar{e}_j$ is given by

$$\bar{e}_j = \begin{cases} (c_l - 1)\delta, & \exists_{m_l \in \mathcal{L}} (S_l = j) \wedge k \in \mathcal{K}_l \\ (c_{wc} - 1)\delta, & \text{otherwise,} \end{cases} \tag{16}$$

where c_{wc} represents the largest possible size of future messages in terms of number of minislots consumed and $\mathcal{L}$ is the current set of messages. Here, the first case implies that the lower numbered dynamic slot is already assigned to a message while the second case considers an empty slot where the largest sized message will be assigned in future. The worst-case delay $\bar{D}_i$ needs to meet the deadline d_i specified for the message. Besides the deadline constraint, we also need to guarantee that the transmission of the message is ensured for each instance of the schedule Θ_i. Here the FlexRay protocol defines a parameter $pLatestTx$ denoting the highest mini-slot counter that a message transmission is allowed in a communication cycle. Considering also the future messages, this constraint can be represented as

$$\bar{\mu}_i = \frac{1}{\delta} \max_{k \in \mathcal{K}_i} \Big(\sum_{j=N+1}^{S_i - 1} (\bar{e}_j + \delta) \Big). \tag{17}$$

Therefore, the compatibility test consists of checking simultaneously the two conditions

$$(\bar{D}_i \leq d_i) \wedge (\bar{\mu}_i < pLatestTx). \tag{18}$$

Schedule Synthesis: Having defined the compatibility test, the FlexRay schedules can be synthesized towards optimizing the extensibility using the following algorithm *Max-E Heuristic*.

Algorithm 1 Max-E Heuristic

Input: FlexRay Configuration, set of messages $\mathcal{M}$
Output: $\Theta_{\mathcal{M}} = \{\Theta_i\}$

```
1:  R_max,i = ComputeRmax();
2:  I = SortByRmax(); // in ascending order
3:  for all i ∈ I do
4:      for S_i = N + 1; S_i ≤ (N + M); S_i + + do
5:          success = False;
6:          E_i(S_i) = ComputeExtensibility(S_i);
7:          for all R_i ≤ R_max,i do
8:              for all B_i < R_i do
9:                  if ((CompatibilityTest() ∧ ProtocolCheck()) == True) then
10:                     success = True;
11:                     Θ_i = (S_i, B_i, R_i);
12:                     E_i,new(S_i) = ComputeExtensibility(S_i, Θ_i);
13:                     E_i,Δ(S_i) = E_i(S_i) − E_i,new(S_i);
14:                     StoreSchedule(Θ_i, E_i,Δ(S_i));
15:                 end if
16:             end for
17:         end for
18:         if ((success == False) ∧ (SlotIsEmpty(S_i) == True)) then
19:             break;
20:         end if
21:     end for
22:     Θ_i = MaxExtensibility(E_i,Δ);
23:     return Θ_i;
24: end for
```

where $R_{max,i}$ is defined as

$$R_{max,i} = min(2^{\lfloor \log_2(\frac{min(p_i, d_i)}{2T_{bus}}) \rfloor}, 64) \tag{19}$$

and represents the maximal allowed value of the cycle repetition rate [5], where p_i denotes the period of the message. Here the maximal repetition rates of all the messages in $\mathcal{M}$ are computed and the messages are sorted according to this value in ascending order (Line 1 - 2). Then the schedule for each messages is computed (Line 3 - 23). For each message, we iterate through all possible slot numbers on the dynamic segment (Line 4) and compute the extensibility of the slot before assigning a schedule (Line 6). Then, we iterate through possible values of repetition rate R_i and the base cycle B_i (Line 7 - 8). For each combination of $\{S_i, B_i, R_i\}$, we first check whether the compatibility test is passed and the schedule is compatible with the protocol (Line 9). If this is true, a new extensibility $E_{i,new}$ and the remaining extensibility $E_{i,\delta}(S_i)$ after assigning the schedule are computed. These results are then stored for further selection. If no valid schedule can be found until an empty slot is reached, we do not assign any schedule for the current message (Line 18 - 20), since if compatibility test fails for the current empty slot, there will not be any slot after this that can meet this conditions. Once all possible schedules for the message is stored, we choose to assign the message to a schedule that maximize the remaining extensibility (Line 22).

4 CONCLUDING REMARKS

Extensibility is an important metric in the design of automotive E/E architectures that represents the ability of the system to accommodate changes and additional applications without or with minimal changes to the current design. Extensibility-driven design optimizes the design towards maximizing this ability and thus making it easier to add additional applications and reduce the cost for re-design, test and validation of existing applications. In this paper, we have discussed the extensibility metric and extensibility-driven design for two setting based respectively on CAN and FlexRay. Although extensibility is a metric that has become increasingly more important in the automotive domain, there have not been sufficient related works addressing this problem. Therefore, more research efforts are necessary towards accurate yet light-weight characterization of this metric under different settings and efficient extensibility-driven design methods.

REFERENCES

[1] Robert N. Charette. 2009. This Car Runs on Code. *IEEE Spectrum* (Feb. 2009).

[2] Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. 2007. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th annual Design Automation Conference*. ACM, 278–283.

[3] Paul Pop, Petru Eles, Zebo Peng, and Traian Pop. 2004. Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 8 (2004), 793–811.

[4] Reinhard Schneider, Dip Goswami, Samarjit Chakraborty, Unmesh Bordoloi, Petru Eles, and Zebo Peng. 2011. On the quantification of sustainability and extensibility of FlexRay schedules. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*. IEEE, 375–380.

[5] Reinhard Schneider, Dip Goswami, Samarjit Chakraborty, Unmesh Bordoloi, Petru Eles, and Zebo Peng. 2014. Quantifying notions of extensibility in flexray schedule synthesis. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 19, 4 (2014), 32.

[6] Wei Zheng, Jike Chong, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. 2005. Extensible and scalable time triggered scheduling. In *Application of Concurrency to System Design, 2005. ACSD 2005. Fifth International Conference on*. IEEE, 132–141.

[7] Qi Zhu, Yang Yang, Marco Natale, Eelco Scholte, and Alberto Sangiovanni-Vincentelli. 2010. Optimizing the software architecture for extensibility in hard real-time distributed systems. *IEEE Transactions on Industrial Informatics* 6, 4 (2010), 621–636.

[8] Qi Zhu, Yang Yang, Eelco Scholte, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. 2009. Optimizing extensibility in hard real-time distributed systems. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*. IEEE, 275–284.